Servlet Example in Java with Database Connection and Log4j integration

We have provided a lot of tutorials on servlets in java, this tutorial is aimed to use all of those information to create a full-fledged web application with database connectivity and log4j integration for logging.

I strongly recommend to check out following tutorials if you are not familiar with any of these. All of them contains sample project that you can download and run for understanding the core concepts of Servlet API.

1. J2EE Web Application Overview
2. Servlet Tutorial for Beginners
3. Session Management in Servlets
4. Servlet Filter
5. Servlet Listener
6. Servlet Exception Handling
7. Servlet Cookie Example

Use Case

Develop a web application that should have following features.

1. User can register and then login to the application.
2. The users information should be maintained in database.
3. Use standard logging framework log4j.
4. The application should support session management, no JSPs should be visible without session. Users can logout anytime from the application.
5. We should not show application and server details to user incase of any exception in application or other common errors like 404.

Once we have our basic application ready, we can move on to add other features.

Design Decisions

1. Since login page is the entry point of application, we will have a simple login.html where user can enter their credentials; email and password. We can't rely on javascript validations, so we will do server side validation and incase of missing information we will redirect user to login page with error details.

2. We will have a register.html from where user can register to our application, we will provide it's link in the login page for new user. User should provide email, password, name and country details for registration. If any information is missing, user will remain on same page with error message. If registration is successful, user will be forwarded to the login page with registration success information and they can use email and password to login.
3. We will use MySql database for persisting user information. We will create a new database, user and Users table for our application. Since our application totally depends on Database Connection, we will create a servlet context listener to initialize the database connection and set it as context attribute for other servlets. We will keep DB configuration details configurable through deployment descriptor. We will also add MySql Java Connector jar to the application libraries.
4. Since we want to use log4j and configure it properly before usage, we will utilize servlet context listener to configure log4j and keep the log4j configuration XML file location in web.xml init parameters. We will write our application logs in a separate log file dbexample.log for easier debugging.
5. Incase of any exceptions like "Database Connection Error" or 404 errors, we want to present a useful page to user. We will utilize servlet exception handling and write our own Exception Handler servlet and configure it in deployment descriptor.
6. Once the user logins successfully, we will create a session for the user and forward them to home.jsp where we will show basic information of the user. We will have a model class User that will store the user data into session. User home page also provide logout button that will invalidate the session and forward them to login page.
7. We need to make sure all the JSPs and other resources are accessible only when user has a valid session, rather than keeping session validation login in all the resources, we will create a Servlet Filter for session validation and configure it in deployment descriptor.
8. We will use Servlet 3.0 features for servlet configuration, listeners and filters rather than keeping all of these in deployment descriptor. We will use Eclipse for development and Tomcat 7 for deployment.

Based on above requirements and design decisions, we will create our dynamic web project whose project structure will look like below image.

servlet-example-project-database-log4j

Let's understand each one of the components and understand their implementation.

Database Setup

We will use below MySql script for new Database, User and Table setup to be used in our application.

MySql_Setup.sql

```
-- login with root to create user, DB and table and provide grants

create user 'pankaj'@'localhost' identified by 'pankaj123';

grant all on *.* to 'pankaj'@'localhost' identified by 'pankaj123';

create database UserDB;

use UserDB;

CREATE TABLE `Users` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL DEFAULT '',
  `email` varchar(20) NOT NULL DEFAULT '',
  `country` varchar(20) DEFAULT 'USA',
  `password` varchar(20) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

Login and Registration HTML Pages

Login and Registration HTML pages are simple pages with form to submit the user information, their code looks like below.

login.html

login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="US-ASCII">
<title>Login Page</title>
</head>
<body>
<h3>Login with email and password</h3>
<form action="Login" method="post">
<strong>User Email</strong>:<input type="text" name="email"><br>
<strong>Password</strong>:<input type="password" name="password"><br>
<input type="submit" value="Login">
</form>
<br>
If you are new user, please <a href="register.html">register</a>.
</body>
</html>
```

register.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="US-ASCII">
<title>Register Page</title>
</head>
<body>
<h3>Provide all the fields for registration.</h3>
<form action="Register" method="post">
<strong>Email ID</strong>:<input type="text" name="email"><br>
<strong>Password</strong>:<input type="password" name="password"><br>
<strong>Name</strong>:<input type="text" name="name"><br>
<strong>Country</strong>:<input type="text" name="country"><br>
<input type="submit" value="Register">
</form>
<br>
If you are registered user, please <a href="login.html">login</a>.
</body>
</html>
```

Deployment Descriptor and log4j configuration

We will have log4j configuration file inside WEB-INF folder and it will be packaged with the application WAR file.

log4j.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"
    debug="false">
    <appender name="dbexample" class="org.apache.log4j.RollingFileAppender">
        <param name="File" value="${catalina.home}/logs/dbexample.log"/>
        <param name="Append"            value="true" />
        <param name="ImmediateFlush"    value="true" />
        <param name="MaxFileSize"       value="20MB" />
        <param name="MaxBackupIndex"    value="10" />
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-4r [%t] %-5p %c %x - %m%
        </layout>
    </appender>
```

```
    <logger name="com.journaldev" additivity="false">
        <level value="DEBUG" />
        <appender-ref ref="dbexample"/>
    </logger>

    <root>
        <level value="debug" />
        <appender-ref ref="dbexample" />
    </root>

</log4j:configuration>
```

Our deployment descriptor (web.xml) looks like below.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
  <display-name>ServletDBLog4jExample</display-name>
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>
  <context-param>
    <param-name>dbUser</param-name>
    <param-value>pankaj</param-value>
  </context-param>
  <context-param>
    <param-name>dbPassword</param-name>
    <param-value>pankaj123</param-value>
  </context-param>
  <context-param>
    <param-name>dbURL</param-name>
    <param-value>jdbc:mysql://localhost:3306/UserDB</param-value>
  </context-param>
  <context-param>
    <param-name>log4j-config</param-name>
    <param-value>WEB-INF/log4j.xml</param-value>
  </context-param>

  <error-page>
    <error-code>404</error-code>
    <location>/AppErrorHandler</location>
  </error-page>
  <error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/AppErrorHandler</location>
  </error-page>

  <filter>
    <filter-name>AuthenticationFilter</filter-name>
    <filter-class>com.journaldev.servlet.filters.AuthenticationFilter</filter-
  </filter>
  <filter-mapping>
    <filter-name>AuthenticationFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>
```

Notice following points in the web.xml configuration.

1. login.html is provided welcome file in the welcome files list.
2. Database connection parameters are made configurable and kept as servlet context init params.
3. log4j configuration file location is also configurable and relative location is provided as context init param.
4. Our custom exception handler servlet AppErrorHandler is configured to handle all the exceptions thrown by our application code and 404 errors.
5. AuthenticationFilter is configured to filter all the incoming requests to the application, this is the place where we will have session validation logic.

Model Classes and Database Connection Manager Class

User.java is a simple java bean that will hold the user information as session attribute.

User.java

```
package com.journaldev.util;

import java.io.Serializable;

public class User implements Serializable{

    private static final long serialVersionUID = 6297385302078200511L;

    private String name;
    private String email;
    private int id;
    private String country;

    public User(String nm, String em, String country, int i){
        this.name=nm;
        this.id=i;
        this.country=country;
        this.email=em;
    }

    public void setName(String name) {
        this.name = name;
    }


    public void setEmail(String email) {
        this.email = email;
    }


    public void setId(int id) {
        this.id = id;
    }


    public void setCountry(String country) {
        this.country = country;
    }


    public String getName() {
```

```
        return name;
    }

    public String getEmail() {
        return email;
    }

    public int getId() {
        return id;
    }

    public String getCountry() {
        return country;
    }

    @Override
    public String toString(){
        return "Name="+this.name+", Email="+this.email+", Country="+this.count
    }
}
```

DBConnectionManager.java is the utility class for MySql database connection and it has a method that returns the connection object. We will use this class for database connection and then set the connection object to servlet context attribute that other servlets can use.

DBConnectionManager.java

```
package com.journaldev.util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnectionManager {

    private Connection connection;

    public DBConnectionManager(String dbURL, String user, String pwd) throws (
        Class.forName("com.mysql.jdbc.Driver");
        this.connection = DriverManager.getConnection(dbURL, user, pwd);
    }

    public Connection getConnection(){
        return this.connection;
    }
}
```

Servlet Context Listener

AppContextListener.java is the servlet context listener implementation that will initialize the Database connection when application context is initialized and it also configures the log4j using it's configuration xml file. Notice the use of context init params for DB connection and log4j configuration.

When context will be destroyed, we are closing database connection in contextDestroyed() method.

Since we are using Servlet 3, we don't need to configure it in web.xml and we just need to annotate it with @WebListener annotation.

AppContextListener.java

```java
package com.journaldev.servlet.listeners;

import java.io.File;
import java.sql.Connection;
import java.sql.SQLException;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.xml.DOMConfigurator;

import com.journaldev.util.DBConnectionManager;

@WebListener
public class AppContextListener implements ServletContextListener {

    public void contextInitialized(ServletContextEvent servletContextEvent) {
        ServletContext ctx = servletContextEvent.getServletContext();

        //initialize DB Connection
        String dbURL = ctx.getInitParameter("dbURL");
        String user = ctx.getInitParameter("dbUser");
        String pwd = ctx.getInitParameter("dbPassword");

        try {
            DBConnectionManager connectionManager = new DBConnectionManager(dbURL, user, pwd);
            ctx.setAttribute("DBConnection", connectionManager.getConnection());
            System.out.println("DB Connection initialized successfully.");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        //initialize log4j
        String log4jConfig = ctx.getInitParameter("log4j-config");
        if(log4jConfig == null){
            System.err.println("No log4j-config init param, initializing log4j with BasicConfigurator");
            BasicConfigurator.configure();
        }else {
            String webAppPath = ctx.getRealPath("/");
            String log4jProp = webAppPath + log4jConfig;
            File log4jConfigFile = new File(log4jProp);
            if (log4jConfigFile.exists()) {
                System.out.println("Initializing log4j with: " + log4jProp);
                DOMConfigurator.configure(log4jProp);
            } else {
                System.err.println(log4jProp + " file not found, initializing with BasicConfigurator");
                BasicConfigurator.configure();
            }
        }
        System.out.println("log4j configured properly");
    }

    public void contextDestroyed(ServletContextEvent servletContextEvent) {
```

```
        Connection con = (Connection) servletContextEvent.getServletContext()
        try {
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Exception and Error Handler

AppErrorHandler.java is our application exception handler servlet configured in deployment descriptor, it provides useful information to the user incase of 404 errors or application level exceptions and provide them hyperlink to go to login page of application.

AppErrorHandler.java

```
package com.journaldev.servlet.errorhandler;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/AppErrorHandler")
public class AppErrorHandler extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse resp
        processError(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse res
        processError(request, response);
    }

    private void processError(HttpServletRequest request,
            HttpServletResponse response) throws IOException {
        // Analyze the servlet exception
        Throwable throwable = (Throwable) request
                .getAttribute("javax.servlet.error.exception");
        Integer statusCode = (Integer) request
                .getAttribute("javax.servlet.error.status_code");
        String servletName = (String) request
                .getAttribute("javax.servlet.error.servlet_name");
        if (servletName == null) {
            servletName = "Unknown";
        }
        String requestUri = (String) request
                .getAttribute("javax.servlet.error.request_uri");
        if (requestUri == null) {
            requestUri = "Unknown";
        }

        // Set response content type
          response.setContentType("text/html");
```

```
            PrintWriter out = response.getWriter();
            out.write("<html><head><title>Exception/Error Details</title></head
            if(statusCode != 500){
                out.write("<h3>Error Details</h3>");
                out.write("<strong>Status Code</strong>:"+statusCode+"<br>");
                out.write("<strong>Requested URI</strong>:"+requestUri);
            }else{
                out.write("<h3>Exception Details</h3>");
                out.write("<ul><li>Servlet Name:"+servletName+"</li>");
                out.write("<li>Exception Name:"+throwable.getClass().getName()+'
                out.write("<li>Requested URI:"+requestUri+"</li>");
                out.write("<li>Exception Message:"+throwable.getMessage()+"</li
                out.write("</ul>");
            }

            out.write("<br><br>");
            out.write("<a href=\"login.html\">Login Page</a>");
            out.write("</body></html>");
        }
    }
```

Servlet Filter

AuthenticationFilter.java is our Filter implementation and we are validating user session here.

AuthenticationFilter.java

```
package com.journaldev.servlet.filters;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.log4j.Logger;

@WebFilter("/AuthenticationFilter")
public class AuthenticationFilter implements Filter {

    private Logger logger = Logger.getLogger(AuthenticationFilter.class);

    public void init(FilterConfig fConfig) throws ServletException {
        logger.info("AuthenticationFilter initialized");
    }

    public void doFilter(ServletRequest request, ServletResponse response, Fi

        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;

        String uri = req.getRequestURI();
        logger.info("Requested Resource::"+uri);

```

```
        HttpSession session = req.getSession(false);

        if(session == null && !(uri.endsWith("html") || uri.endsWith("Login")
            logger.error("Unauthorized access request");
            res.sendRedirect("login.html");
        }else{
            // pass the request along the filter chain
            chain.doFilter(request, response);
        }


    }

    public void destroy() {
        //close any resources here
    }

}
```

Notice the use of @WebFilter annotation, we can also provide URL Patterns for filter here but sometimes it's good to have in web.xml to easily disable the filters.

Servlet Classes

LoginServlet resource is used to validate the user input for login and forward them to home page or incase of missing data, provide useful information to user.

LoginServlet.java

```
package com.journaldev.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.log4j.Logger;

import com.journaldev.util.User;

@WebServlet(name = "Login", urlPatterns = { "/Login" })
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    static Logger logger = Logger.getLogger(LoginServlet.class);

    protected void doPost(HttpServletRequest request, HttpServletResponse resp
        String email = request.getParameter("email");
        String password = request.getParameter("password");
        String errorMsg = null;
```

```
        if(email == null || email.equals("")){
            errorMsg ="User Email can't be null or empty";
        }
        if(password == null || password.equals("")){
            errorMsg = "Password can't be null or empty";
        }

        if(errorMsg != null){
            RequestDispatcher rd = getServletContext().getRequestDispatcher("/
            PrintWriter out= response.getWriter();
            out.println("<font color=red>"+errorMsg+"</font>");
            rd.include(request, response);
        }else{

        Connection con = (Connection) getServletContext().getAttribute("DBConn
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
            ps = con.prepareStatement("select id, name, email,country from Use
            ps.setString(1, email);
            ps.setString(2, password);
            rs = ps.executeQuery();

            if(rs != null && rs.next()){

                User user = new User(rs.getString("name"), rs.getString("emai
                logger.info("User found with details="+user);
                HttpSession session = request.getSession();
                session.setAttribute("User", user);
                response.sendRedirect("home.jsp");;
            }else{
                RequestDispatcher rd = getServletContext().getRequestDispatche
                PrintWriter out= response.getWriter();
                logger.error("User not found with email="+email);
                out.println("<font color=red>No user found with given email id
                rd.include(request, response);
            }
        } catch (SQLException e) {
            e.printStackTrace();
            logger.error("Database connection problem");
            throw new ServletException("DB Connection problem.");
        }finally{
            try {
                rs.close();
                ps.close();
            } catch (SQLException e) {
                logger.error("SQLException in closing PreparedStatement or Res
            }

        }
        }
    }

}
```

LogoutServlet is simple and invalidates the user session and forward them to login page.

LogoutServlet.java

```
package com.journaldev.servlet;

import java.io.IOException;

```

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.log4j.Logger;

@WebServlet(name = "Logout", urlPatterns = { "/Logout" })
public class LogoutServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    static Logger logger = Logger.getLogger(LogoutServlet.class);

    protected void doPost(HttpServletRequest request, HttpServletResponse resp
        response.setContentType("text/html");
        Cookie[] cookies = request.getCookies();
        if(cookies != null){
        for(Cookie cookie : cookies){
            if(cookie.getName().equals("JSESSIONID")){
                logger.info("JSESSIONID="+cookie.getValue());
                break;
            }
        }
        }
        //invalidate the session if exists
        HttpSession session = request.getSession(false);
        logger.info("User="+session.getAttribute("User"));
        if(session != null){
            session.invalidate();
        }
        response.sendRedirect("login.html");
    }

}
```

RegisterServlet is used by users to register to the application and then forward them to login page with registration success message.

RegisterServlet.java

```
package com.journaldev.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.log4j.Logger;

@WebServlet(name = "Register", urlPatterns = { "/Register" })
public class RegisterServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

```

```java
    static Logger logger = Logger.getLogger(RegisterServlet.class);

    protected void doPost(HttpServletRequest request, HttpServletResponse resp
        String email = request.getParameter("email");
        String password = request.getParameter("password");
        String name = request.getParameter("name");
        String country = request.getParameter("country");
        String errorMsg = null;
        if(email == null || email.equals("")){
            errorMsg = "Email ID can't be null or empty.";
        }
        if(password == null || password.equals("")){
            errorMsg = "Password can't be null or empty.";
        }
        if(name == null || name.equals("")){
            errorMsg = "Name can't be null or empty.";
        }
        if(country == null || country.equals("")){
            errorMsg = "Country can't be null or empty.";
        }

        if(errorMsg != null){
            RequestDispatcher rd = getServletContext().getRequestDispatcher("/
            PrintWriter out= response.getWriter();
            out.println("<font color=red>"+errorMsg+"</font>");
            rd.include(request, response);
        }else{

        Connection con = (Connection) getServletContext().getAttribute("DBConn
        PreparedStatement ps = null;
        try {
            ps = con.prepareStatement("insert into Users(name,email,country, p
            ps.setString(1, name);
            ps.setString(2, email);
            ps.setString(3, country);
            ps.setString(4, password);

            ps.execute();

            logger.info("User registered with email="+email);

            //forward to login page to login
            RequestDispatcher rd = getServletContext().getRequestDispatcher("/
            PrintWriter out= response.getWriter();
            out.println("<font color=green>Registration successful, please log
            rd.include(request, response);
        } catch (SQLException e) {
            e.printStackTrace();
            logger.error("Database connection problem");
            throw new ServletException("DB Connection problem.");
        }finally{
            try {
                ps.close();
            } catch (SQLException e) {
                logger.error("SQLException in closing PreparedStatement");
            }
        }
        }

    }

}
```

Home JSP Page

home.jsp is the home page for user after successful login, we are just showing some user information here and providing them option to logout.

home.jsp

```
<%@page import="com.journaldev.util.User"%>
<%@ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Home Page</title>
</head>
<body>
<%User user = (User) session.getAttribute("User"); %>
<h3>Hi <%=user.getName() %></h3>
<strong>Your Email</strong>: <%=user.getEmail() %><br>
<strong>Your Country</strong>: <%=user.getCountry() %><br>
<br>
<form action="Logout" method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
```

The JSP page still contains a lot of java code because we are not using JSP tags, we will look into this in JSP tutorials. As of now please bear with this.

Run the Application

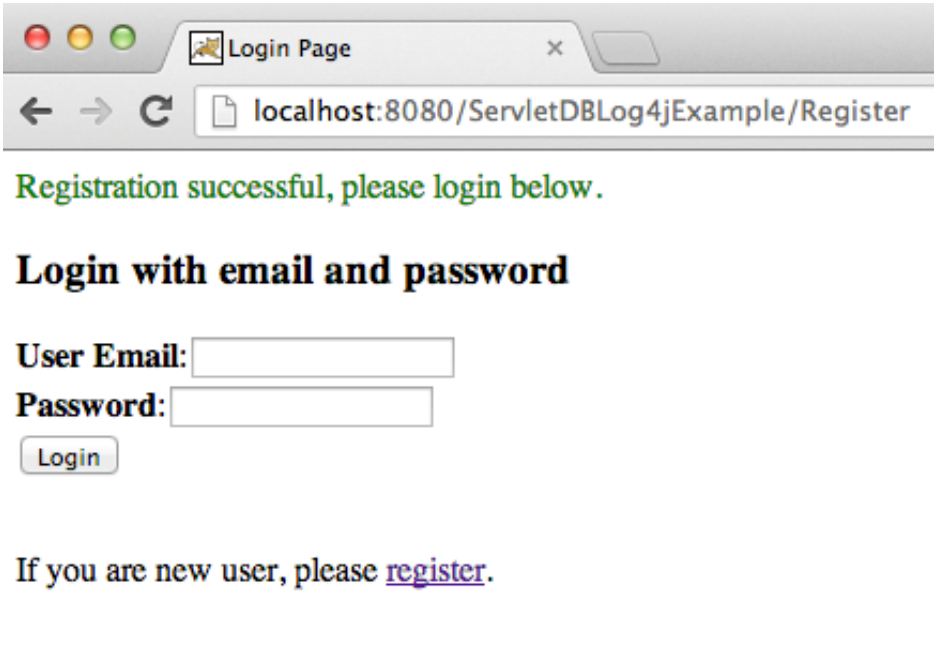
Our application is ready for execution, I would suggest to export it as WAR file and then deploy to tomcat rather than deploying it directly to Tomcat server from Eclipse so that you can easily look into the log4j log file for debugging.

Some sample execution pages are shown in below images.

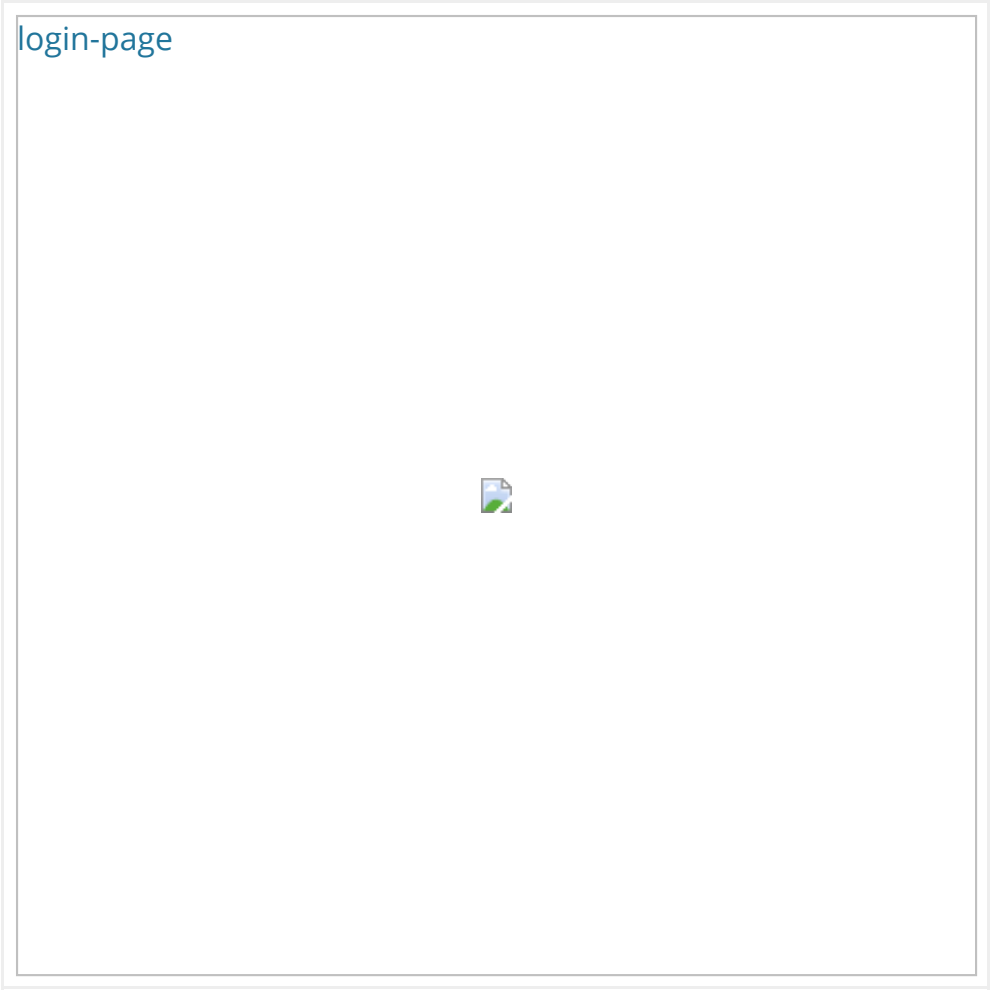
User Registration Page:

registration-page

Registration Success Page:

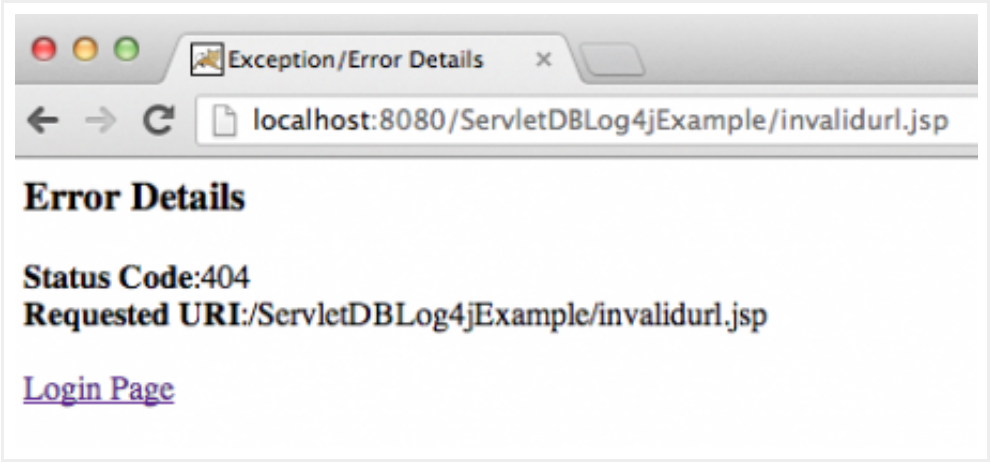


Login Page:

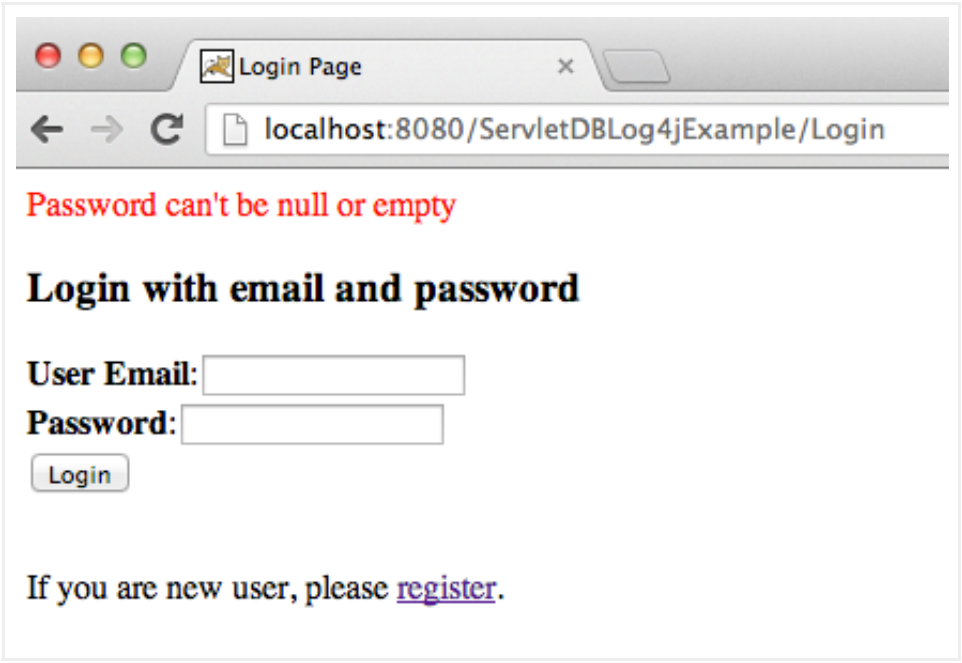


User Home Page:

404 Error Page:



Input Validation Error Page:



log4j Log File:

dbexample.log

```
0    [localhost-startStop-1] INFO  com.journaldev.servlet.filters.Authenticat
1    [localhost-startStop-1] INFO  com.journaldev.servlet.filters.Authenticat
37689 [http-bio-8080-exec-3] INFO  com.journaldev.servlet.filters.Authenticat
37689 [http-bio-8080-exec-3] ERROR com.journaldev.servlet.filters.Authenticat
37693 [http-bio-8080-exec-4] INFO  com.journaldev.servlet.filters.Authenticat
51844 [http-bio-8080-exec-5] INFO  com.journaldev.servlet.filters.Authenticat
77818 [http-bio-8080-exec-7] INFO  com.journaldev.servlet.filters.Authenticat
77835 [http-bio-8080-exec-7] INFO  com.journaldev.servlet.LoginServlet  - User
77840 [http-bio-8080-exec-8] INFO  com.journaldev.servlet.filters.Authenticat
98251 [http-bio-8080-exec-9] INFO  com.journaldev.servlet.filters.Authenticat
98251 [http-bio-8080-exec-9] INFO  com.journaldev.servlet.LogoutServlet  - JSE
98251 [http-bio-8080-exec-9] INFO  com.journaldev.servlet.LogoutServlet  - Use
98254 [http-bio-8080-exec-10] INFO  com.journaldev.servlet.filters.Authentica
109516 [http-bio-8080-exec-10] INFO  com.journaldev.servlet.filters.Authentica
109517 [http-bio-8080-exec-10] INFO  com.journaldev.servlet.RegisterServlet
127848 [http-bio-8080-exec-10] INFO  com.journaldev.servlet.filters.Authentica
223055 [http-bio-8080-exec-2] INFO  com.journaldev.servlet.filters.Authentica
223056 [http-bio-8080-exec-2] INFO  com.journaldev.servlet.LoginServlet  - Use
223059 [http-bio-8080-exec-2] INFO  com.journaldev.servlet.filters.Authentica
231931 [http-bio-8080-exec-2] INFO  com.journaldev.servlet.filters.Authentica
```

Download Resources

1. MySql Java Connector Jar
2. Log4j Jar

I hope you liked the article and understand the core concepts of Servlet API, please share and let me know your opinions through comments.