

Lesson 02 Demo 03

Filters

Objective: To explore filters by creating a new filter and working on the login filter's response, pre-processing, and post-processing

Tools Required: Eclipse IDE

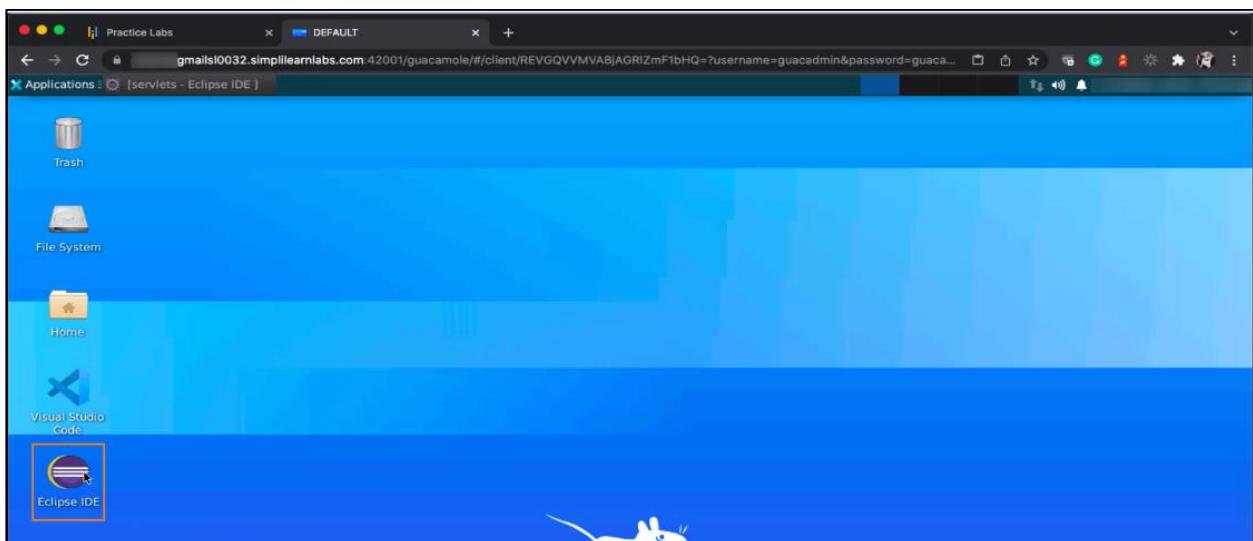
Prerequisites: None

Steps to be followed:

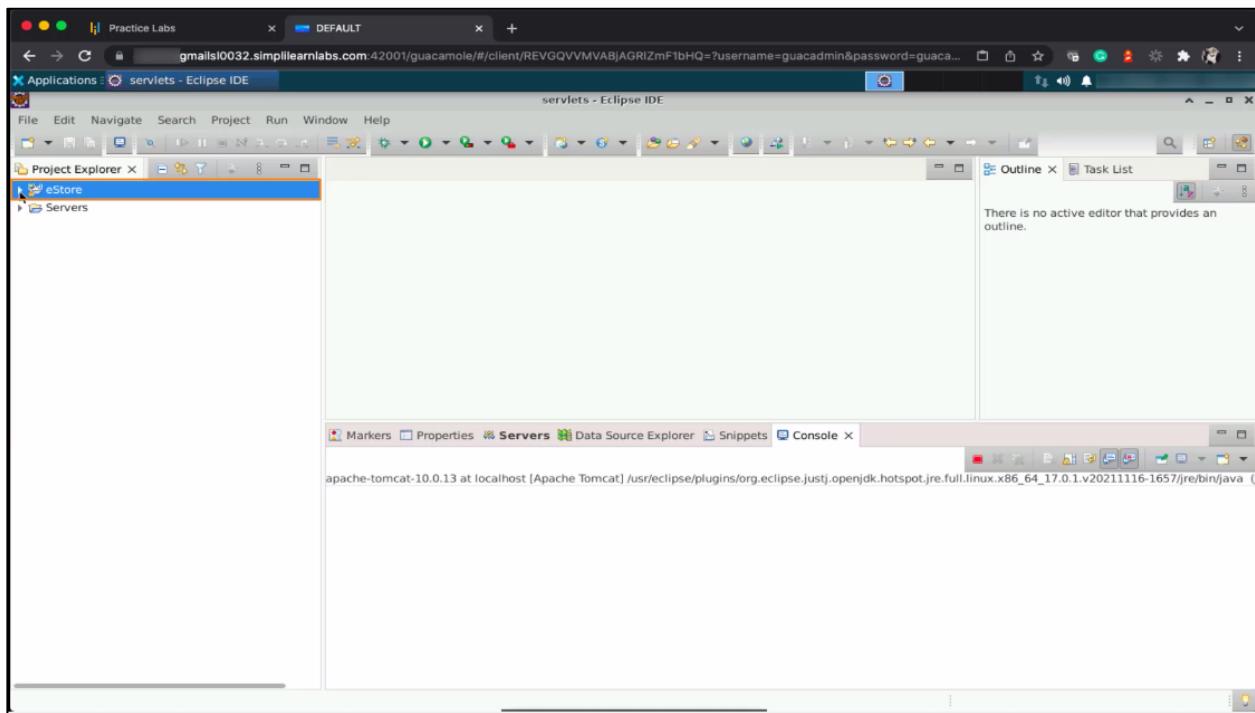
1. Create a new filter
2. Work with mandatory methods
3. Work with a filter mechanism
4. Work on the login filter's response
5. Make a filter for login parameters

Step 1: Create a new filter

1.1 Open Eclipse IDE

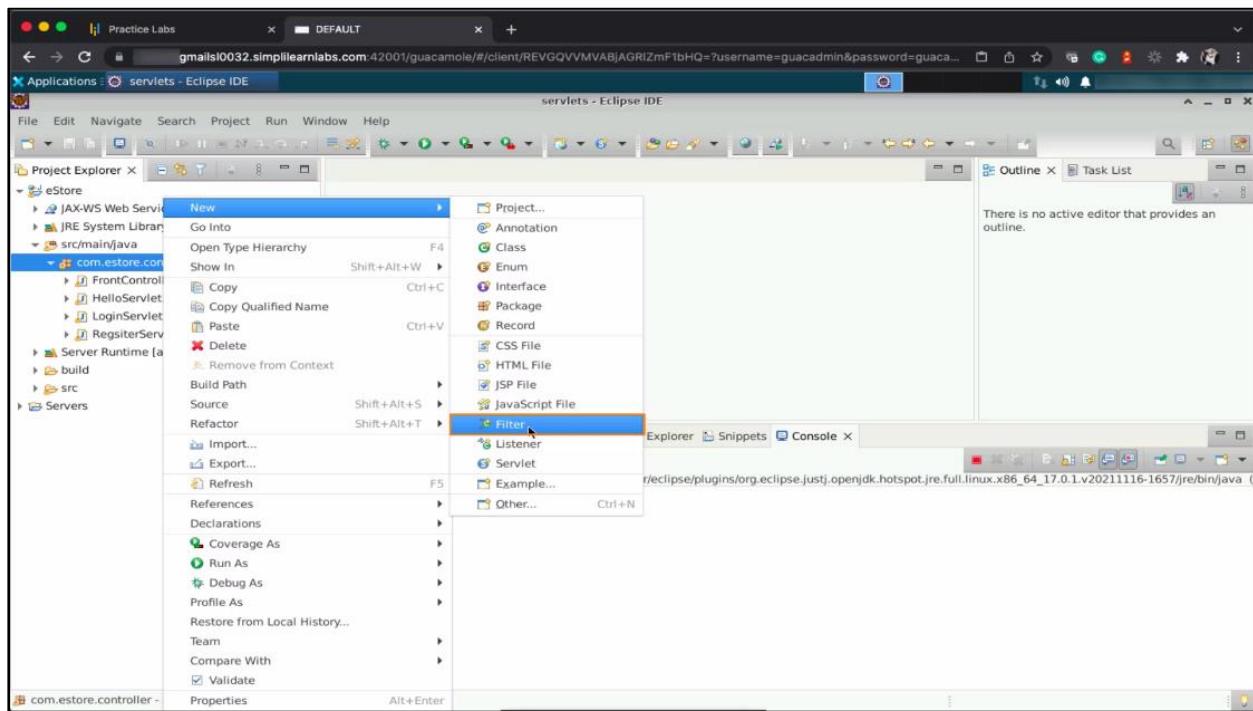


1.2 Navigate to the **eStore** project



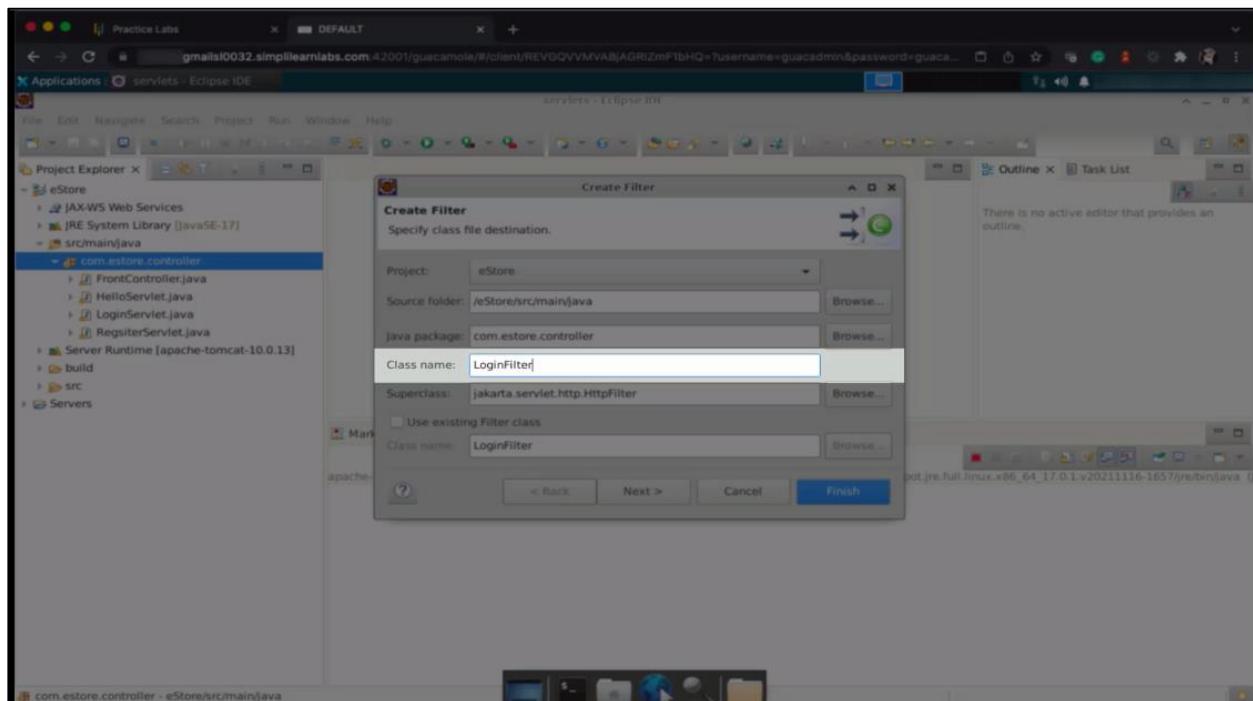
Note: Please refer to the previous demo on how to create the **eStore** project

1.3 Create a new filter by right-clicking on the project, selecting **New**, and clicking on **Filter**

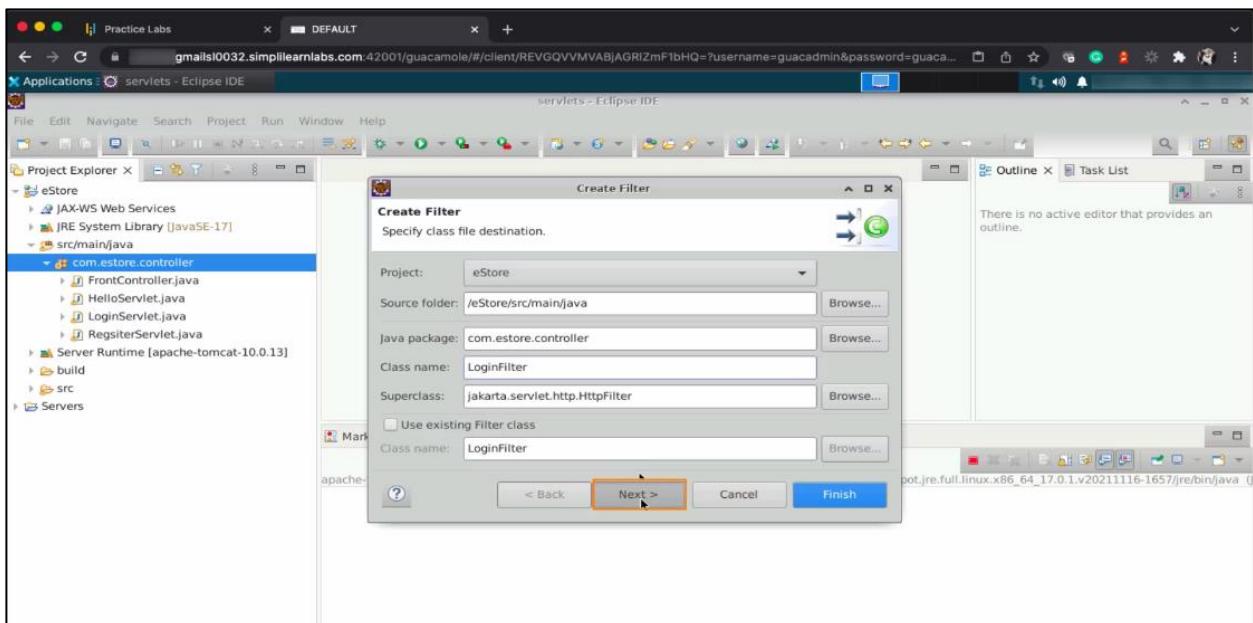


Observation: Rather than requests going directly to the servlet, the request will first navigate to the filter which will then send the request to the servlet.

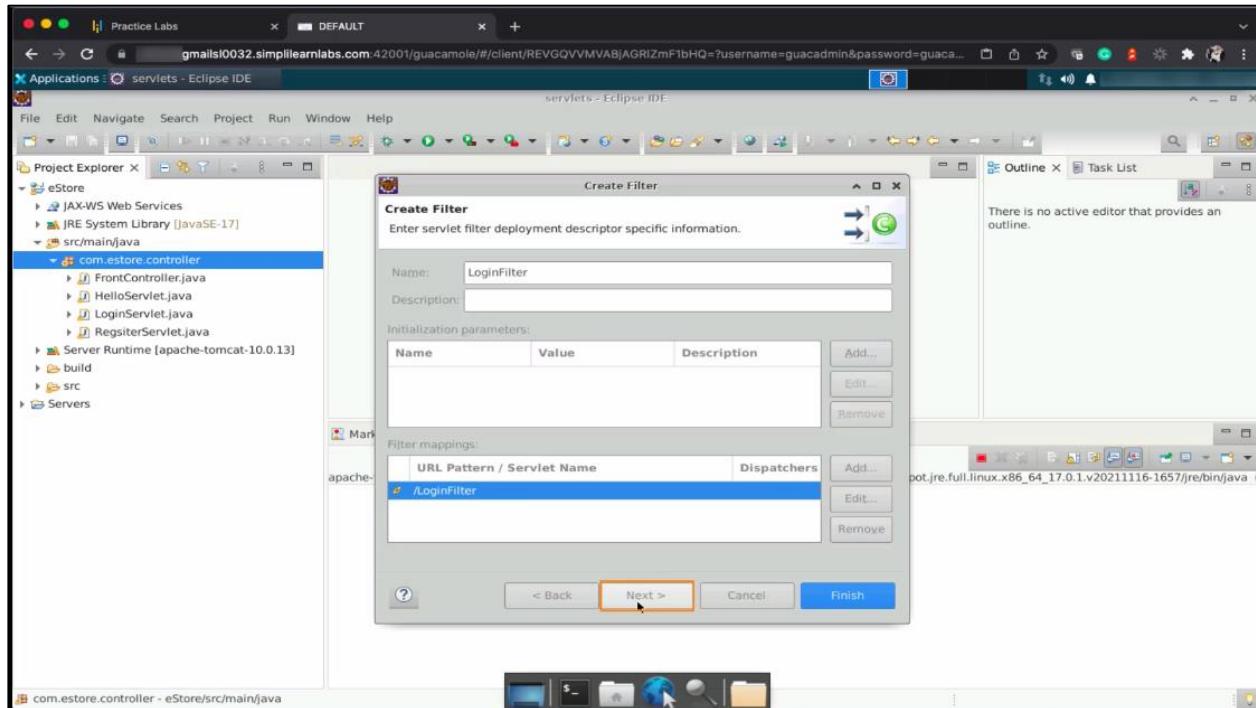
1.4 Name the filter as **LoginFilter**



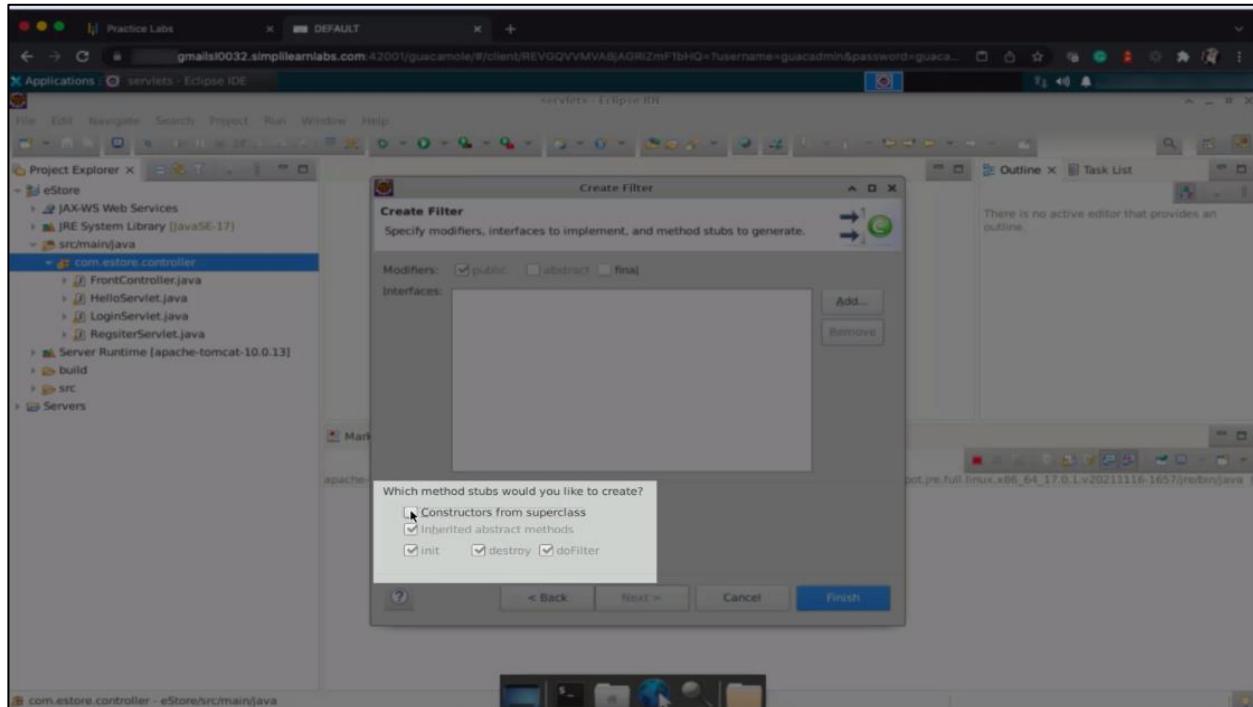
1.5 Click on **Next**



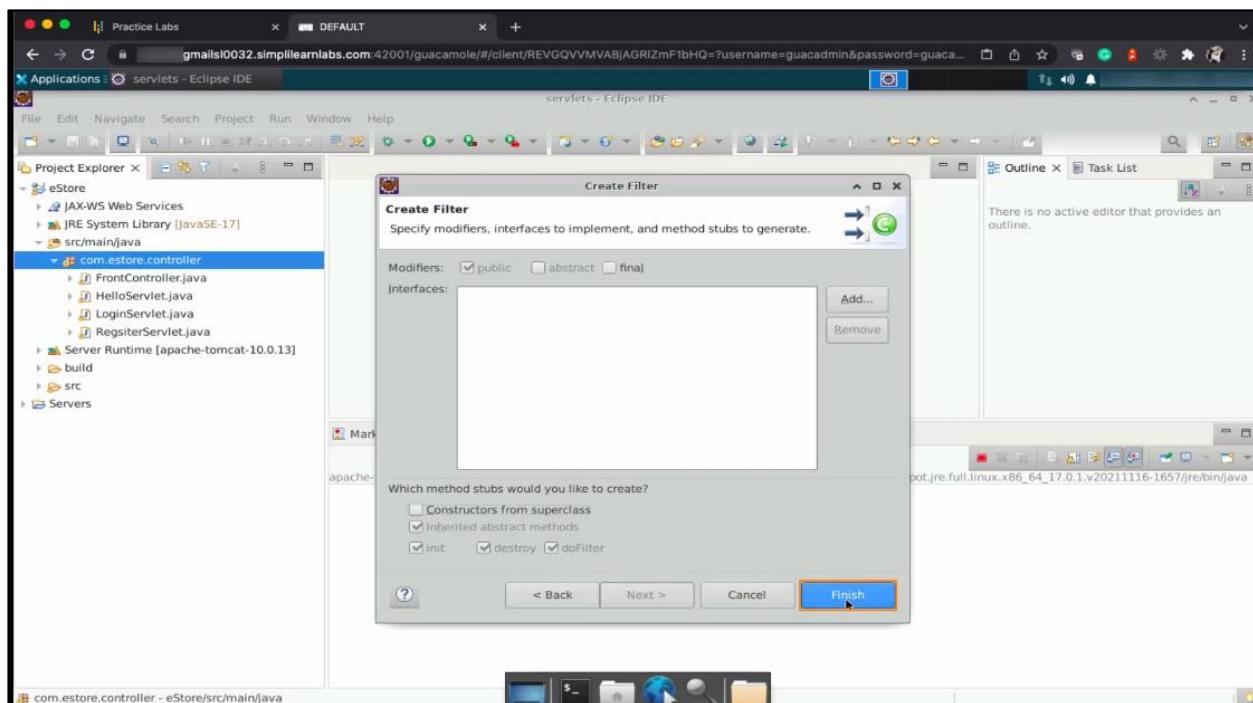
1.6 Specify the URL patterns by selecting the **/LoginFilter** as the URL-pattern and clicking on **Next**



1.7 Unselect the checkbox for Constructor from superclass



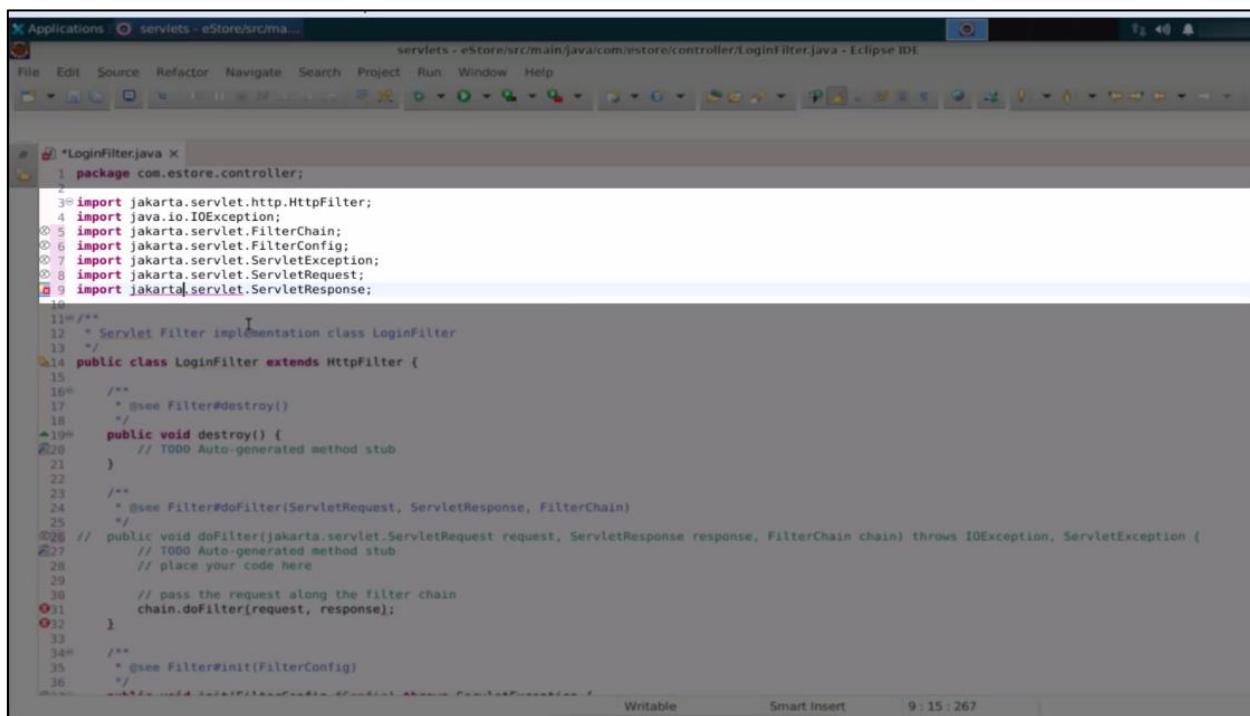
1.8 Click on Finish



Step 2: Work with mandatory methods

2.1 Navigate to the **LoginFilter.java** file and import the following packages:

```
import.jakarta.servlet.FilterChain;
import.jakarta.servlet.FilterConfig;
import.jakarta.servlet.ServletException;
import.jakarta.servlet.ServletRequest;
import.jakarta.servlet.ServletResponse;
```



The screenshot shows the Eclipse IDE interface with the title bar "servlets - eStore/src/main/java/com/estore/controller/LoginFilter.java - Eclipse IDE". The main editor window displays the Java code for the LoginFilter class. The code includes imports for jakarta.servlet.FilterChain, FilterConfig, ServletException, ServletRequest, and ServletResponse. It defines a public class LoginFilter that extends HttpFilter. The class contains a destroy() method and a doFilter() method. The doFilter() method takes HttpServletRequest, HttpServletResponse, and FilterChain parameters and calls chain.doFilter(request, response). The code is annotated with Javadoc comments and TODO notes.

```
1 package com.estore.controller;
2
3 import jakarta.servlet.http.HttpServlet;
4 import java.io.IOException;
5 import jakarta.servlet.FilterChain;
6 import jakarta.servlet.FilterConfig;
7 import jakarta.servlet.ServletException;
8 import jakarta.servlet.ServletRequest;
9 import jakarta.servlet.ServletResponse;
10
11 /**
12  * Servlet Filter implementation class LoginFilter
13  */
14 public class LoginFilter extends HttpServlet {
15
16     /**
17      * @see Filter#destroy()
18     */
19     public void destroy() {
20         // TODO Auto-generated method stub
21     }
22
23     /**
24      * @see Filter#doFilter(HttpServletRequest, HttpServletResponse, FilterChain)
25     */
26     public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws IOException, ServletException {
27         // TODO Auto-generated method stub
28         // place your code here
29
30         // pass the request along the filter chain
31         chain.doFilter(request, response);
32     }
33
34     /**
35      * @see Filter#init(FilterConfig)
36     */
37 }
```

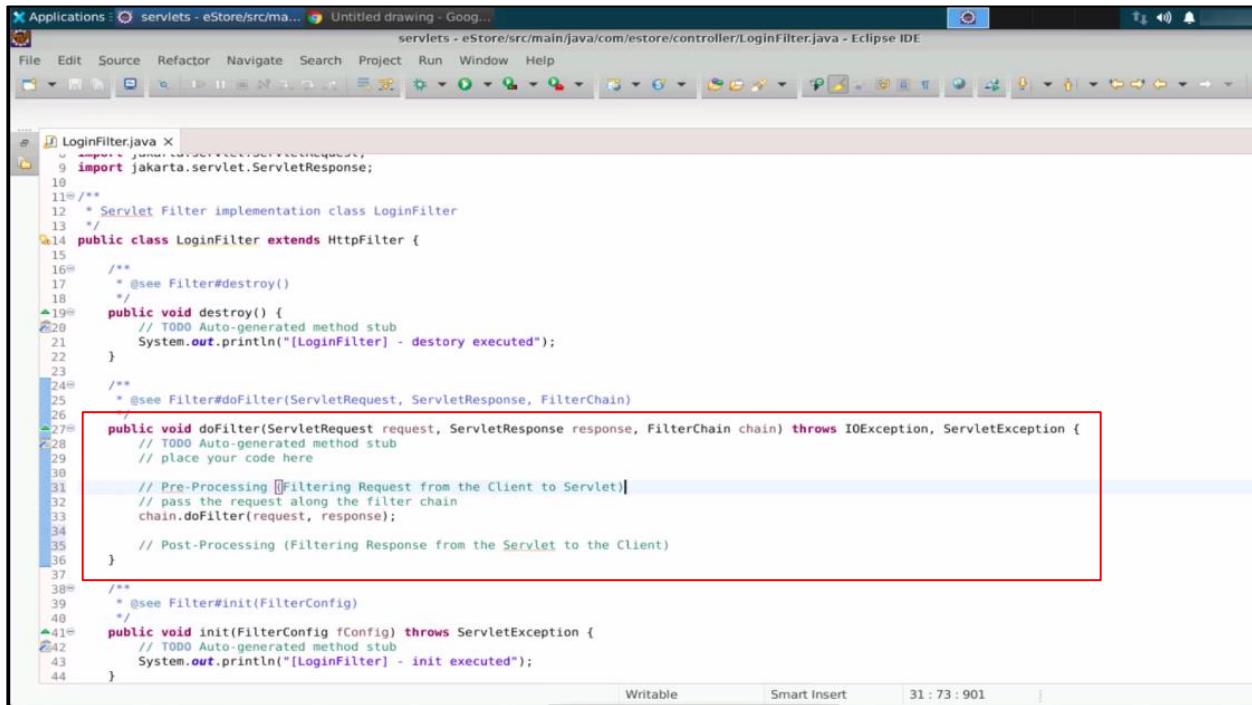
2.2 Specify the **LoginFilter** and include the execution of the destroy and init methods, you can modify the code as follows:

```
System.out.println ("[LoginFilter] – destroy executed");
System.out.println ("[LoginFilter] – init executed");
```

```
1 LoginFilter.java X
  ...
  9 import jakarta.servlet.ServletRequest;
10
11 /**
12  * Servlet Filter implementation class LoginFilter
13 */
14 public class LoginFilter extends HttpFilter {
15
16     /**
17      * @see Filter#destroy()
18     */
19     public void destroy() {
20         // TODO Auto-generated method stub
21         System.out.println("[LoginFilter] - destroy executed");
22     }
23
24     /**
25      * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
26     */
27     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
28         // TODO Auto-generated method stub
29         // place your code here
30
31         // pass the request along the filter chain
32         chain.doFilter(request, response);
33     }
34
35     /**
36      * @see Filter#init(FilterConfig)
37     */
38     public void init(FilterConfig fConfig) throws ServletException {
39         // TODO Auto-generated method stub
40         System.out.println("[LoginFilter] - init executed");
41     }
42 }
43 }
```

Step 3: Work with a filter mechanism

3.1 Return to the **LoginServlet.java** file and specify the pre-processing and post-processing of data

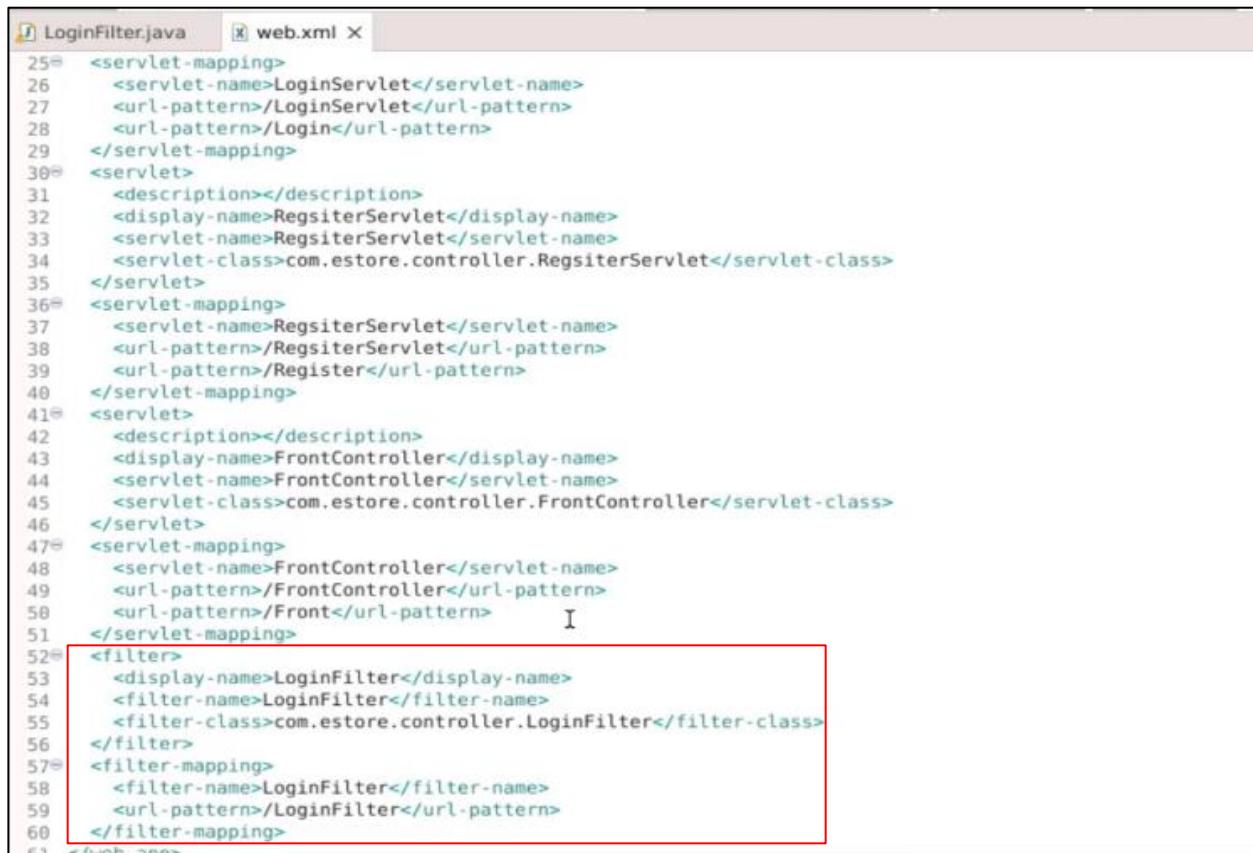


```
File Edit Source Refactor Navigate Search Project Run Window Help

LoginFilter.java X
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class LoginFilter extends HttpFilter {
    public void destroy() {
        // TODO Auto-generated method stub
        System.out.println("[LoginFilter] - destroy executed");
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        // Pre-Processing [Filtering Request from the Client to Servlet]
        // pass the request along the filter chain
        chain.doFilter(request, response);
        // Post-Processing (Filtering Response from the Servlet to the Client)
    }
    public void init(FilterConfig fConfig) throws ServletException {
        // TODO Auto-generated method stub
        System.out.println("[LoginFilter] - init executed");
    }
}
```

The screenshot shows the Eclipse IDE interface with the `LoginFilter.java` file open in the editor. The code defines a `HttpFilter` implementation. The `doFilter` method is highlighted with a red box, indicating the area where the student should place their code. The code within the `doFilter` method is a placeholder: it prints a message to the console and then passes the request and response through the filter chain using the `chain.doFilter` method.

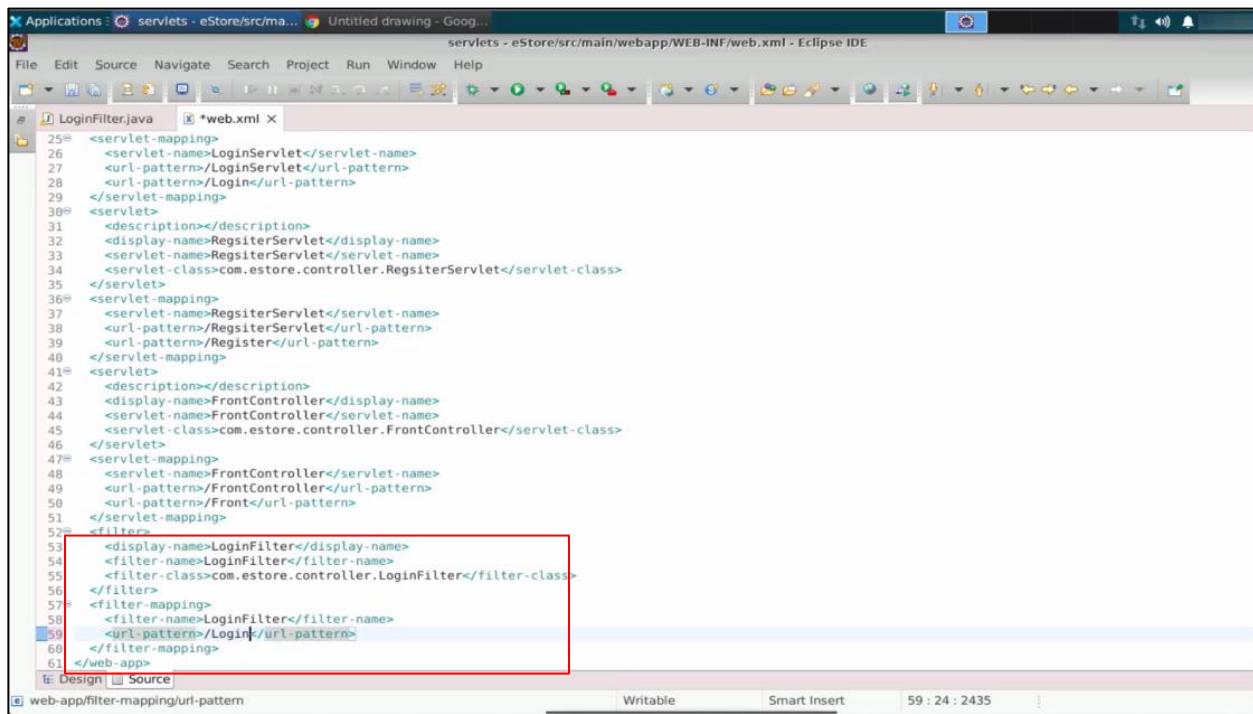
3.2 Open the **web.xml** file where you will find the login filter and filter mapping that has been created with the name **LoginFilter**



```
1 LoginFilter.java  X web.xml X
2
3 25<servlet-mapping>
4 26  <servlet-name>LoginServlet</servlet-name>
5 27  <url-pattern>/LoginServlet</url-pattern>
6 28  <url-pattern>/Login</url-pattern>
7 29</servlet-mapping>
8 30<servlet>
9 31  <description></description>
10 32  <display-name>RegsiterServlet</display-name>
11 33  <servlet-name>RegsiterServlet</servlet-name>
12 34  <servlet-class>com.estore.controller.RegsterServlet</servlet-class>
13 35</servlet>
14 36<servlet-mapping>
15 37  <servlet-name>RegsiterServlet</servlet-name>
16 38  <url-pattern>/RegsterServlet</url-pattern>
17 39  <url-pattern>/Register</url-pattern>
18 40</servlet-mapping>
19 41<servlet>
20 42  <description></description>
21 43  <display-name>FrontController</display-name>
22 44  <servlet-name>FrontController</servlet-name>
23 45  <servlet-class>com.estore.controller.FrontController</servlet-class>
24 46</servlet>
25 47<servlet-mapping>
26 48  <servlet-name>FrontController</servlet-name>
27 49  <url-pattern>/FrontController</url-pattern>
28 50  <url-pattern>/Front</url-pattern> I
29 51</servlet-mapping>
30 52<filter>
31 53  <display-name>LoginFilter</display-name>
32 54  <filter-name>LoginFilter</filter-name>
33 55  <filter-class>com.estore.controller.LoginFilter</filter-class>
34 56</filter>
35 57<filter-mapping>
36 58  <filter-name>LoginFilter</filter-name>
37 59  <url-pattern>/LoginFilter</url-pattern>
38 60</filter-mapping>
39 61</web-app>
```

Step 4: Work on the login filter's response

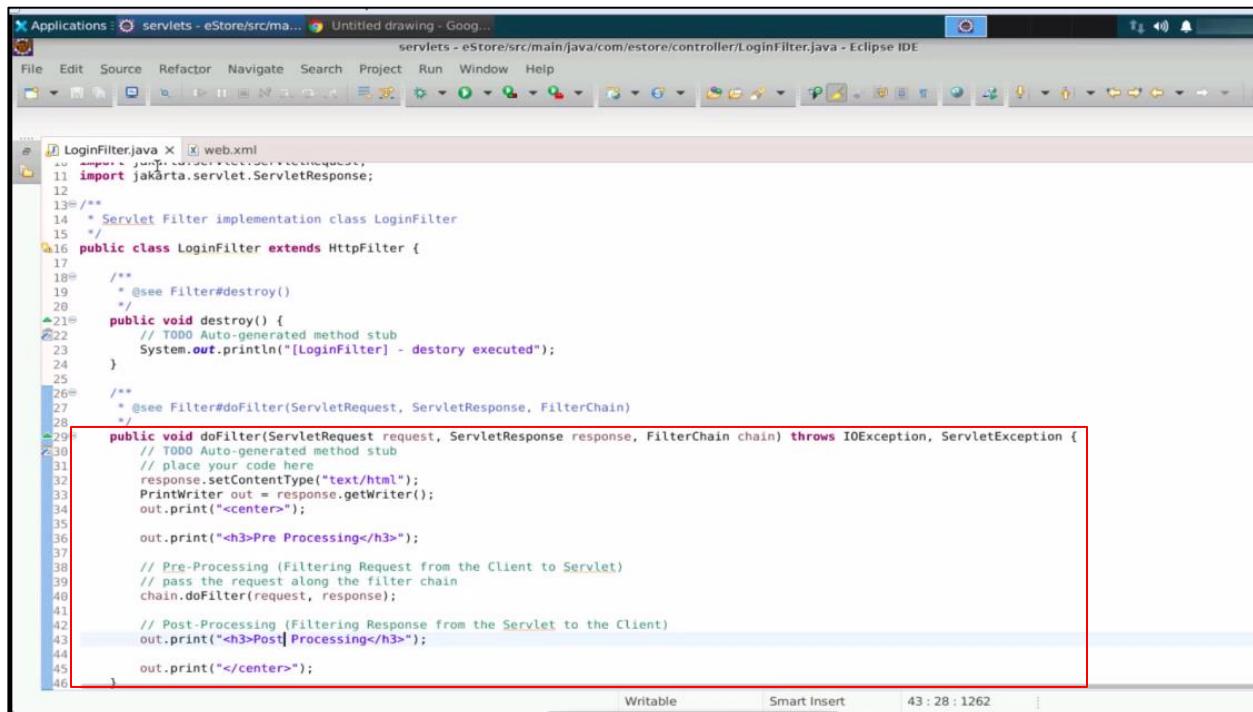
4.1 Change the value of the url-pattern from /LoginFilter to /Login



```
File Edit Source Navigate Search Project Run Window Help
File Edit Source Navigate Search Project Run Window Help
LoginFilter.java *web.xml
25< servlet-mappings>
26  < servlet-name>LoginServlet</servlet-name>
27  < url-pattern>/LoginServlet</url-pattern>
28  < url-pattern>/Login</url-pattern>
29 </servlet-mapping>
30< servlet>
31  < description></description>
32  < display-name>RegisterServlet</display-name>
33  < servlet-name>RegisterServlet</servlet-name>
34  < servlet-class>com.estore.controller.RegisterServlet</servlet-class>
35 </servlet>
36< servlet-mappings>
37  < servlet-name>RegisterServlet</servlet-name>
38  < url-pattern>/RegisterServlet</url-pattern>
39  < url-pattern>/Register</url-pattern>
40 </servlet-mapping>
41< servlet>
42  < description></description>
43  < display-name>FrontController</display-name>
44  < servlet-name>FrontController</servlet-name>
45  < servlet-class>com.estore.controller.FrontController</servlet-class>
46 </servlet>
47< servlet-mappings>
48  < servlet-name>FrontController</servlet-name>
49  < url-pattern>/FrontController</url-pattern>
50  < url-pattern>/Front</url-pattern>
51 </servlet-mapping>
52< filters>
53  < display-name>LoginFilter</display-name>
54  < filter-name>LoginFilter</filter-name>
55  < filter-class>com.estore.controller.LoginFilter</filter-class>
56 </filter>
57< filter-mapping>
58  < filter-name>LoginFilter</filter-name>
59  < url-pattern>/Login</url-pattern>
60 </filter-mapping>
61</web-app>
Design Source
web-app/filter-mapping/url-pattern Writable Smart Insert 59 : 24 : 2435
```

This is because the URL pattern specified for the filter will match that of the servlet. As a result, the request will be intercepted by the filter before it reaches the servlet directly.

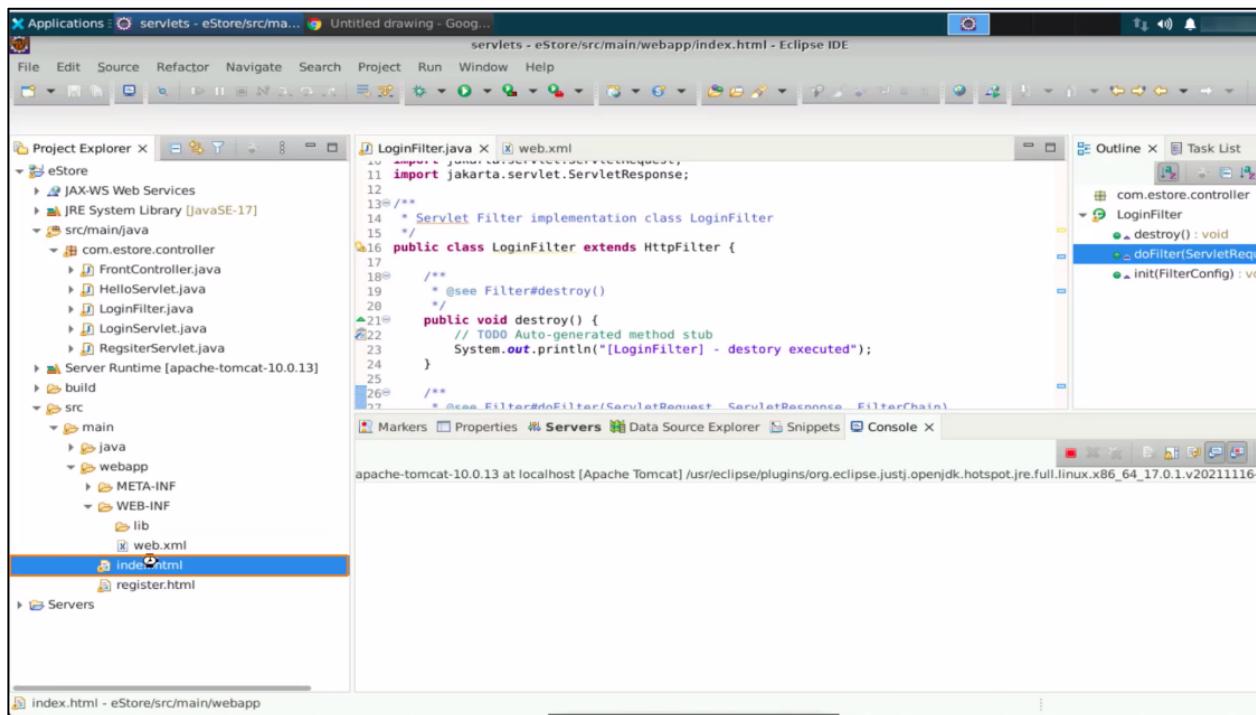
4.2 In **LoginFilter.java**, modify the response to include pre-processing and post-processing messages using the **setContentType()** method and **<h3>** tags



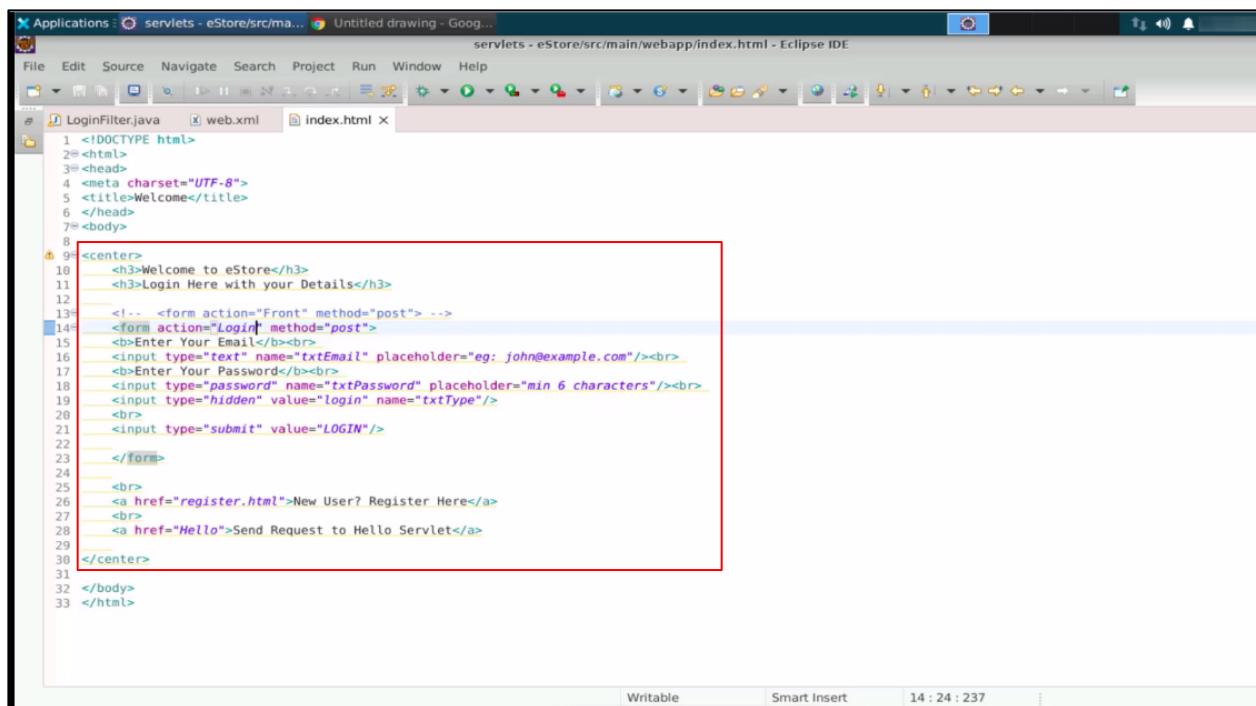
```
File Edit Source Refactor Navigate Search Project Run Window Help

LoginFilter.java x web.xml
1 import java.io.IOException;
2 import jakarta.servlet.FilterChain;
3 import jakarta.servlet.ServletException;
4 import jakarta.servlet.http.HttpServletRequest;
5 import jakarta.servlet.http.HttpServletResponse;
6
7 /**
8  * Servlet Filter implementation class LoginFilter
9  */
10 public class LoginFilter extends HttpFilter {
11
12     /**
13      * @see Filter#destroy()
14      */
15     public void destroy() {
16         // TODO Auto-generated method stub
17         System.out.println("[LoginFilter] - destroy executed");
18     }
19
20     /**
21      * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
22      */
23     public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws IOException, ServletException {
24         // TODO Auto-generated method stub
25         // place your code here
26         response.setContentType("text/html");
27         PrintWriter out = response.getWriter();
28         out.print("<center>");
29
30         out.print("<h3>Pre Processing</h3>");
31
32         // Pre-Processing (Filtering Request from the Client to Servlet)
33         // pass the request along the filter chain
34         chain.doFilter(request, response);
35
36         out.print("<h3>Post Processing</h3>");
37
38         // Post-Processing (Filtering Response from the Servlet to the Client)
39         out.print("<h3>Post Processing</h3>");
40
41         out.print("</center>");
42     }
43 }
44
45 }
```

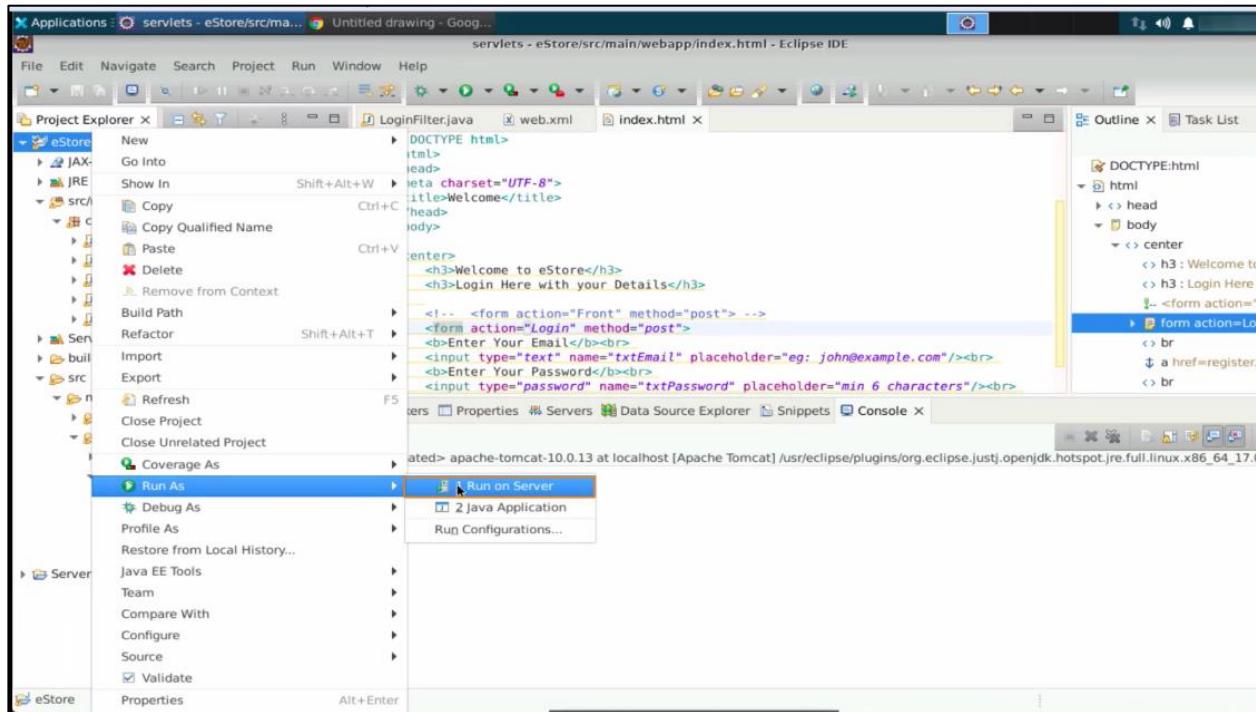
4.3 Open the index.html file



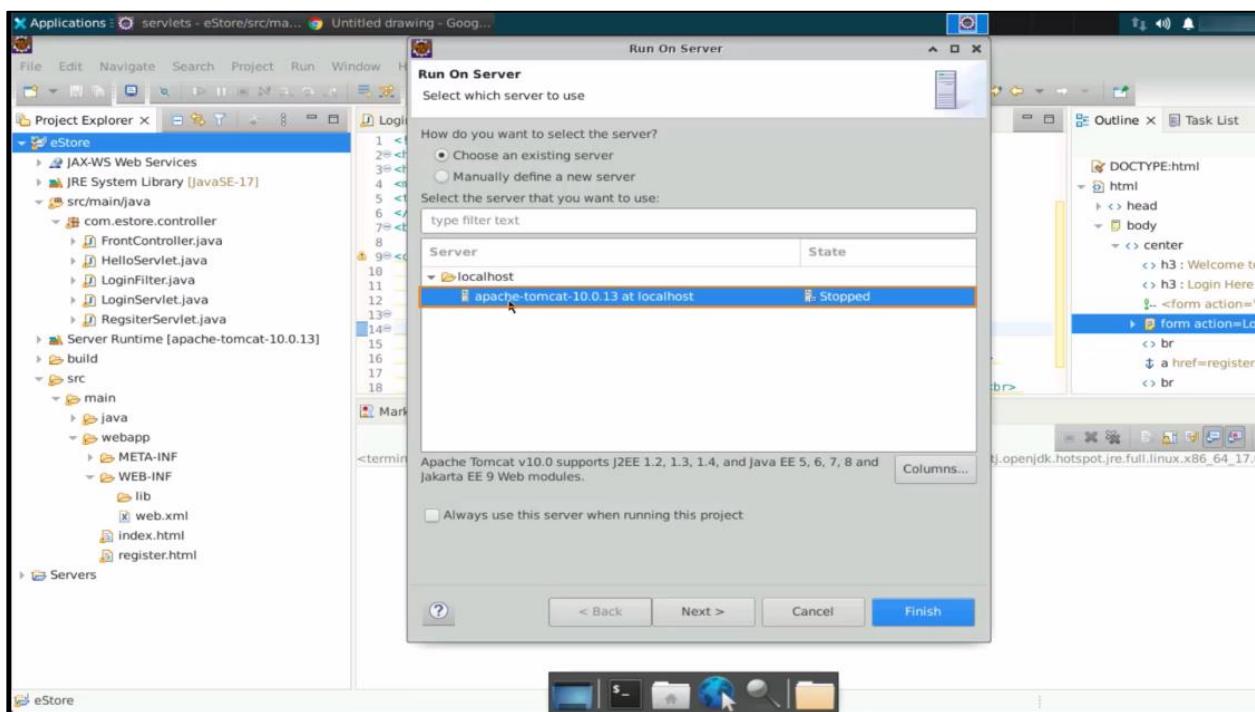
4.4 In the index.html file, change the action to **Login** and the method to **post** (13 and 14)



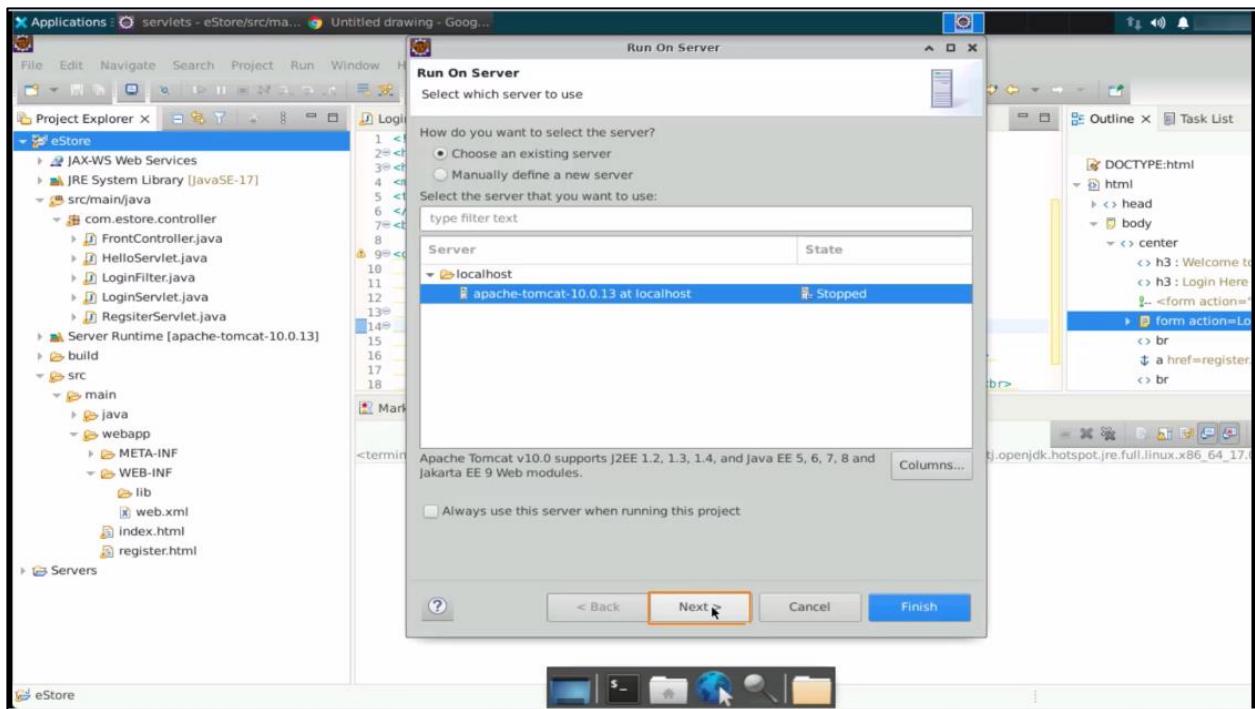
4.5 Navigate to the **eStore** project, right-click on it, select **Run As**, and click on **Run on Server**



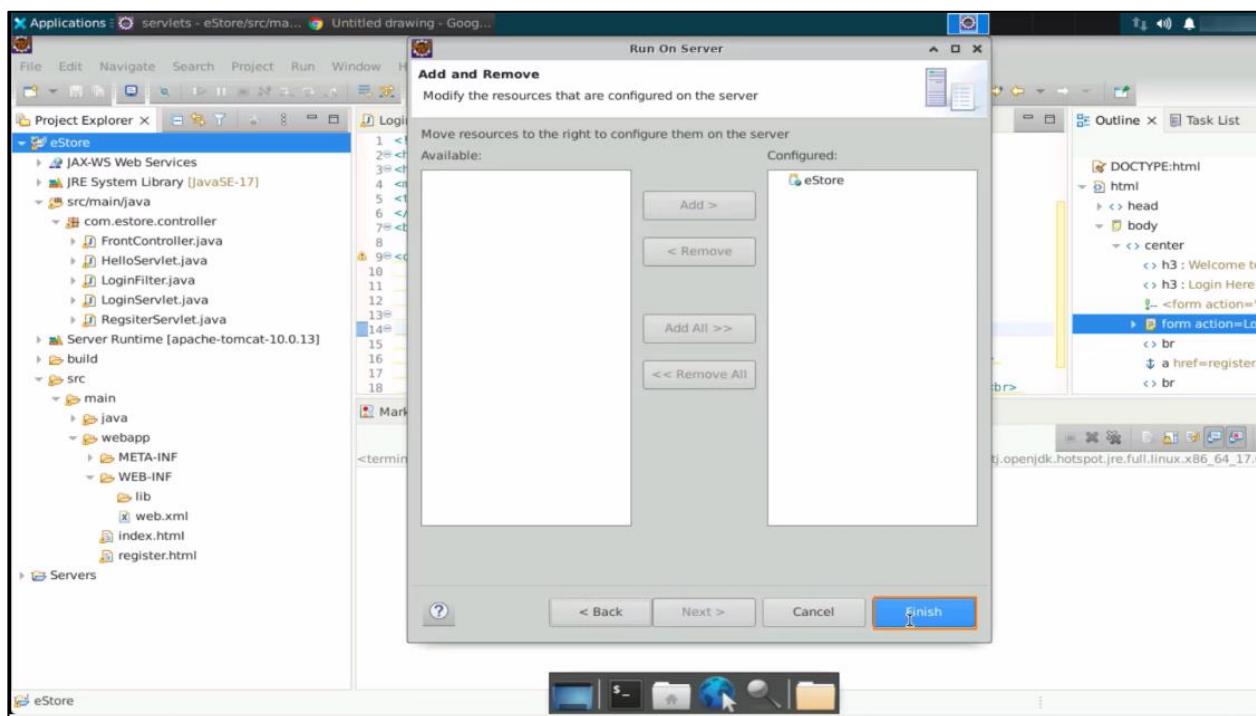
4.6 Select the apache-tomcat-10.0.13 as the server



4.7 Click on Next



4.8 Click on Finish



4.9 Navigate to the Eclipse Console tab where the message [LoginFilter] – init executed indicates the successful initialization of the **LoginFilter**

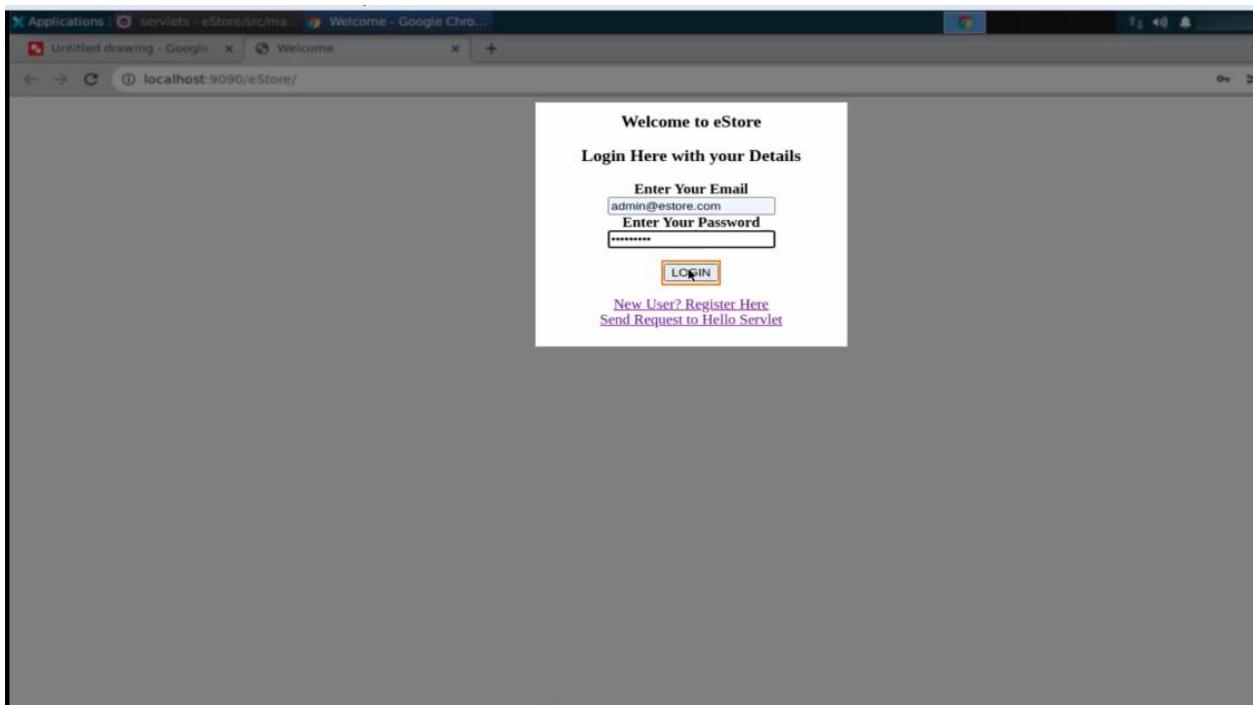
The screenshot shows the Eclipse IDE interface. On the left is the Java code for a JSP file named 'index.jsp'. The code includes HTML for a welcome message and a login form. On the right is the 'Console' tab, which displays the Tomcat server logs. The logs show the startup of the Apache Tomcat Native library, the initialization of the ProtocolHandler for 'http-nio-9090', and the startup of Catalina. A specific log entry '[LoginFilter] - init executed' is highlighted in yellow, indicating the successful initialization of the custom filter.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Welcome</title>
6 </head>
7 <body>
8
9 <center>
10 <h3>Welcome to eStore</h3>
11 <h3>Login Here with your Details</h3>
12
13<!-- <form action="Front" method="post"> -->
14<form action="Login" method="post">
15 <b>Enter Your Email</b><br>
16 <input type="text" name="txtEmail" placeholder="eg: john@example.com"/><br>
17 <b>Enter Your Password</b><br>
18 <input type="password" name="txtPassword" placeholder="min 6 characters"/><br>
```

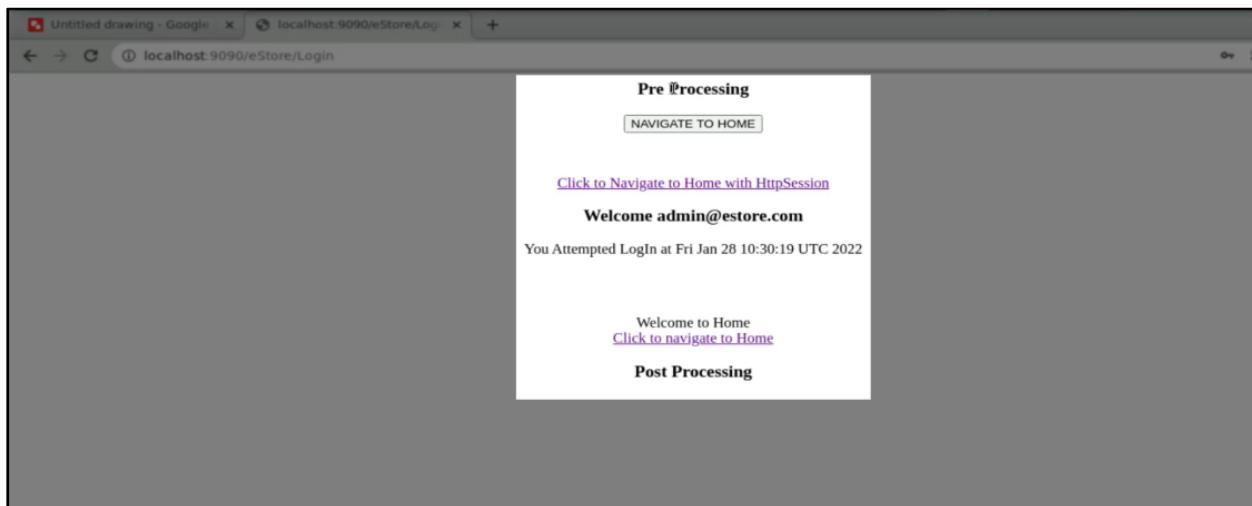
Markers Properties Servers Data Source Explorer Snippets Console

```
apache-tomcat-10.0.13 at localhost [Apache Tomcat] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full INFO: Command line argument: -XX:+ShowCodeDetailsInExceptionMessages
Jan 28, 2022 10:29:50 AM org.apache.catalina.core.AprLifecycleListener lifecycleEvent
INFO: The Apache Tomcat Native library which allows using OpenSSL was not found on the java.library.path.
Jan 28, 2022 10:29:50 AM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["http-nio-9090"]
Jan 28, 2022 10:29:50 AM org.apache.catalina.startup.Catalina load
INFO: Server initialization in [686] milliseconds
Jan 28, 2022 10:29:50 AM org.apache.catalina.core.StandardService startInternal
INFO: Starting service [Catalina]
Jan 28, 2022 10:29:50 AM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet engine: [Apache Tomcat/10.0.13]
[LoginFilter] - init executed
Jan 28, 2022 10:29:50 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-9090"]
Jan 28, 2022 10:29:50 AM org.apache.catalina.startup.Catalina start
```

4.10 Enter the login details and click on **LOGIN**



You can see the output as **Pre-processing** and **Post-processing**.



This means that the request is sent from the filter to the Servlet, and the response is sent from the Servlet to the filter.

Step 5: Make the filter work for login parameters

5.1 Navigate to **LoginFilter.java** and extract the email and the password from the request using the **request.getParameter()** method (lines 38 and 39)

```

    ...
    // place your code here
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.print("<center>");
    out.print("<h3>Pre Processing</h3>");
    String email = request.getParameter("txtEmail");
    String password = request.getParameter("txtPassword");
    ...
  
```

5.2 Navigate to the console logs where you will see that the filter has been destroyed

```

INFO: Starting service [Catalina]
Jan 28, 2022 10:29:50 AM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet engine: [Apache Tomcat/10.0.13]
[LoginFilter] - init executed
Jan 28, 2022 10:29:50 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-9090"]
Jan 28, 2022 10:29:50 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in [374] milliseconds
Jan 28, 2022 10:31:30 AM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/eStore] has started
[LoginFilter] - destroy executed
[LoginFilter] - init executed
Jan 28, 2022 10:31:30 AM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/eStore] is completed
  
```

5.3 For general pre-processing, check if the email or password is empty by using the **isEmpty()** function and use the **doFilter()** method to perform the filtering in pre-processing

```

LoginFilter.java
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
    // TODO Auto-generated method stub
    System.out.println("[LoginFilter] - destroy executed");

    /**
     * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
     */
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        // TODO Auto-generated method stub
        // place your code here
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.print("<center>");

        // Pre-Processing (Filtering Request from the Client to Servlet)
        out.print("<h3>Pre Processing</h3>");

        String email = request.getParameter("txtEmail");
        String password = request.getParameter("txtPassword");

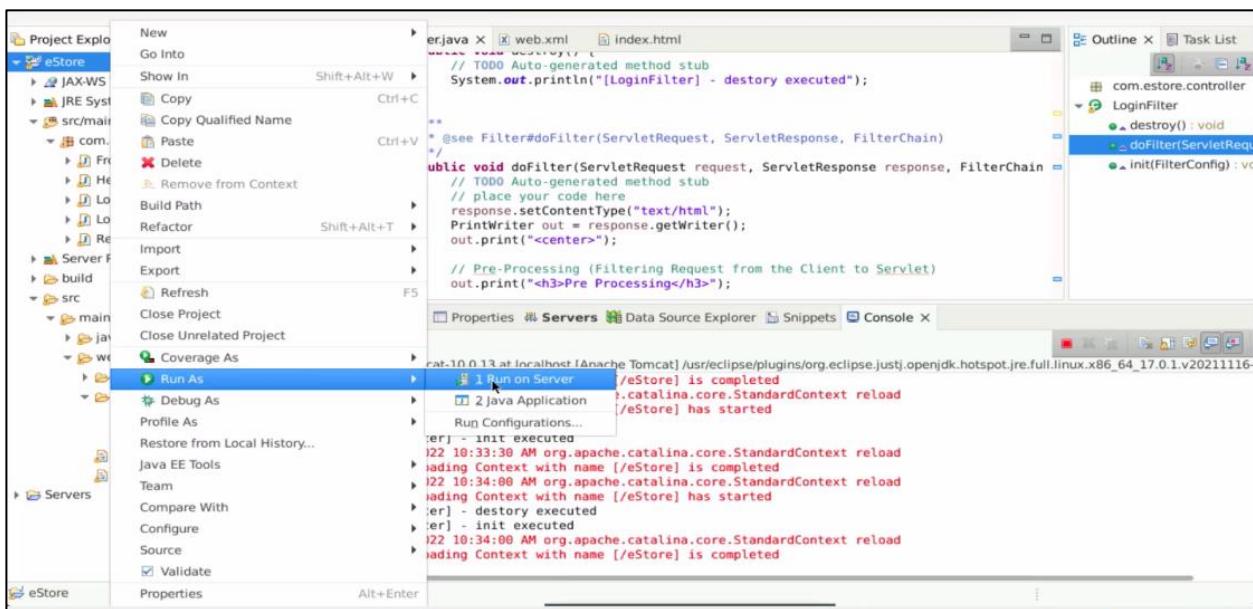
        if(email.isEmpty() || password.isEmpty()) {
            // pass the data to LoginServlet
            chain.doFilter(request, response);
        } else {
            out.print("<h3>Sorry!! Email and Password Cannot be Blank</h3>");
        }

        // Post-Processing (Filtering Response from the Servlet to the Client)
        out.print("<h3>Post Processing</h3>");

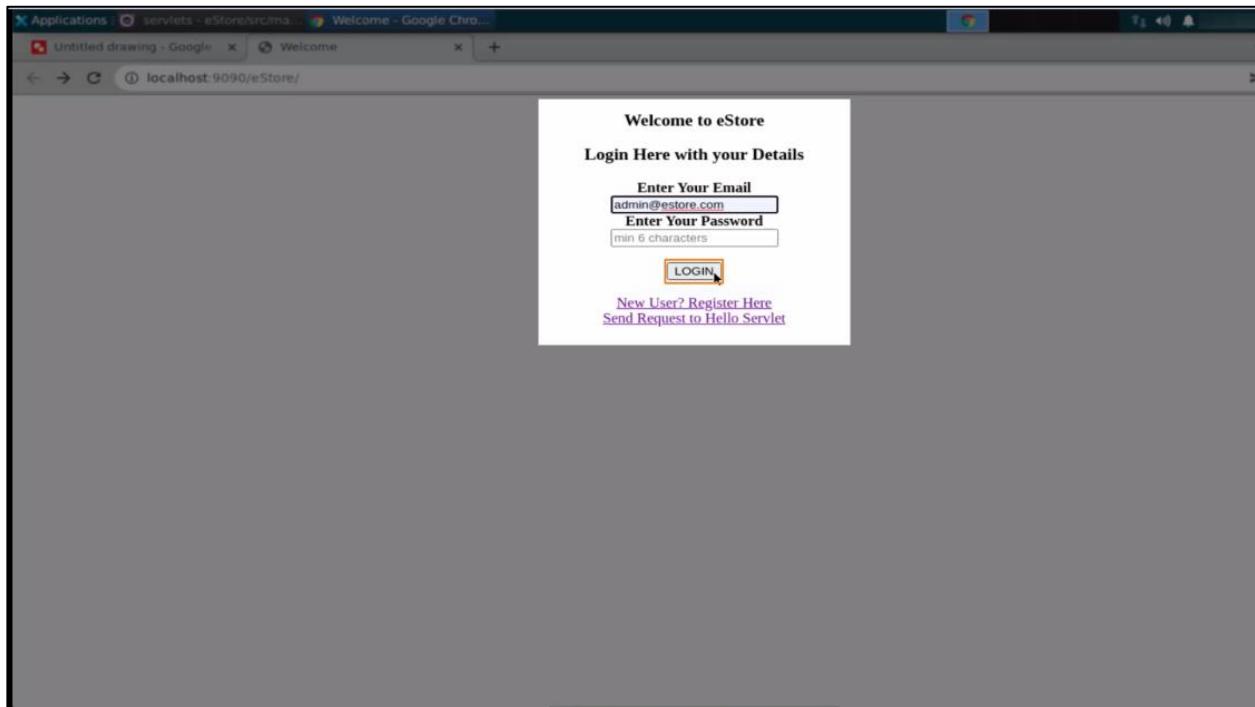
        out.print("</center>");
    }
}

```

5.4 Next, rerun the code by right-clicking on the **eStore** project, selecting **Run As**, and clicking on **Run on Server**

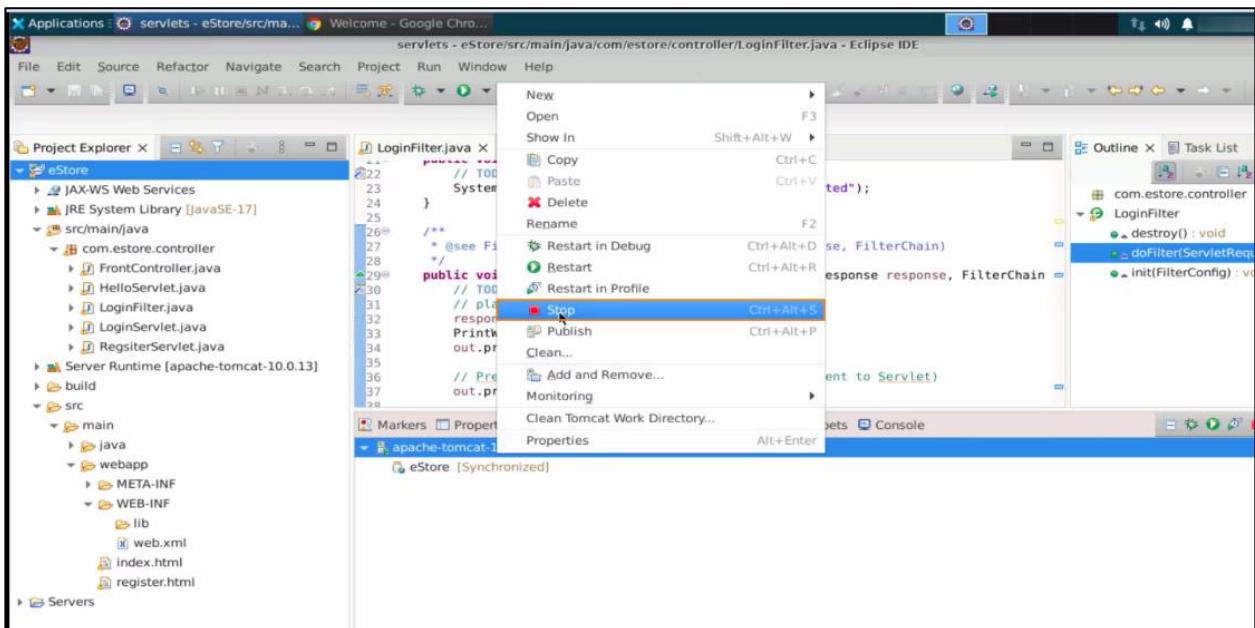


5.5 Refresh the page and enter only the email, and click on **LOGIN**



You can see that the server is not yet initialized.

5.6 Go back and stop the server by clicking on **Stop**



5.7 If the email or password is blank, filter the request and the response using the `chain.doFilter()` method

```

response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.print("<center>");

// Pre-Processing (Filtering Request from the Client to Servlet)
out.print("<h3>Pre Processing</h3>");

String email = request.getParameter("txtEmail");
String password = request.getParameter("txtPassword");

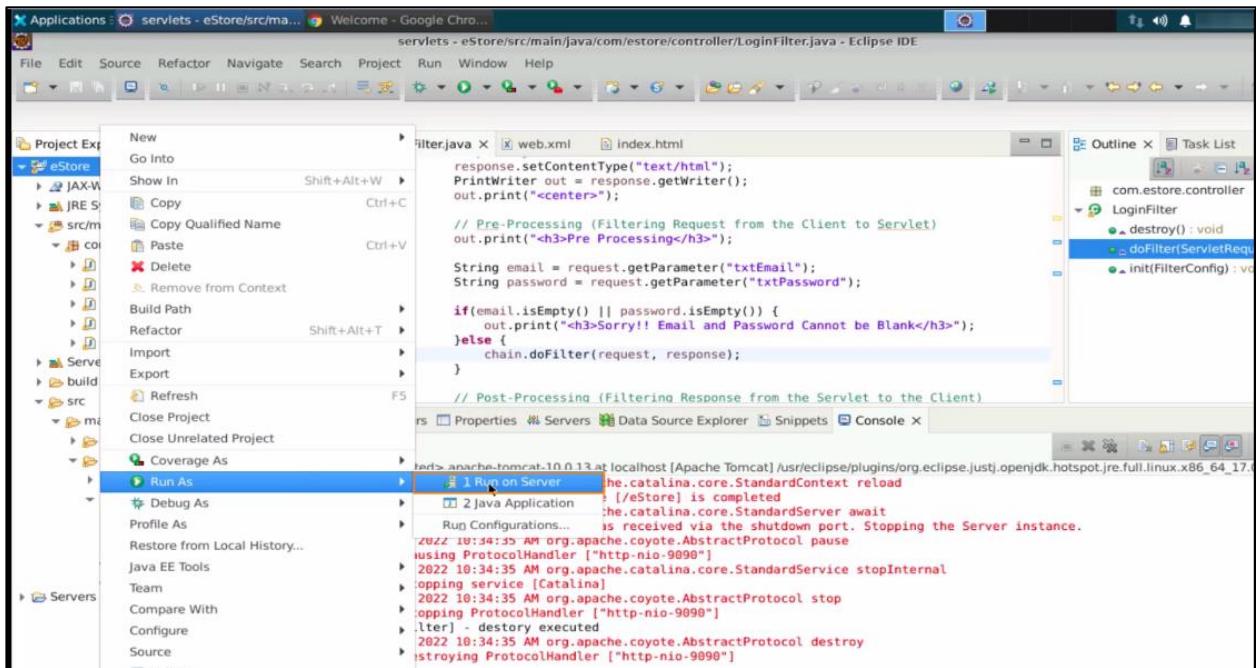
if(email.isEmpty() || password.isEmpty()) {
    out.print("<h3>Sorry!! Email and Password Cannot be Blank</h3>");
} else {
    chain.doFilter(request, response);
}

// Post-Processing (Filtering Response from the Servlet to the Client)

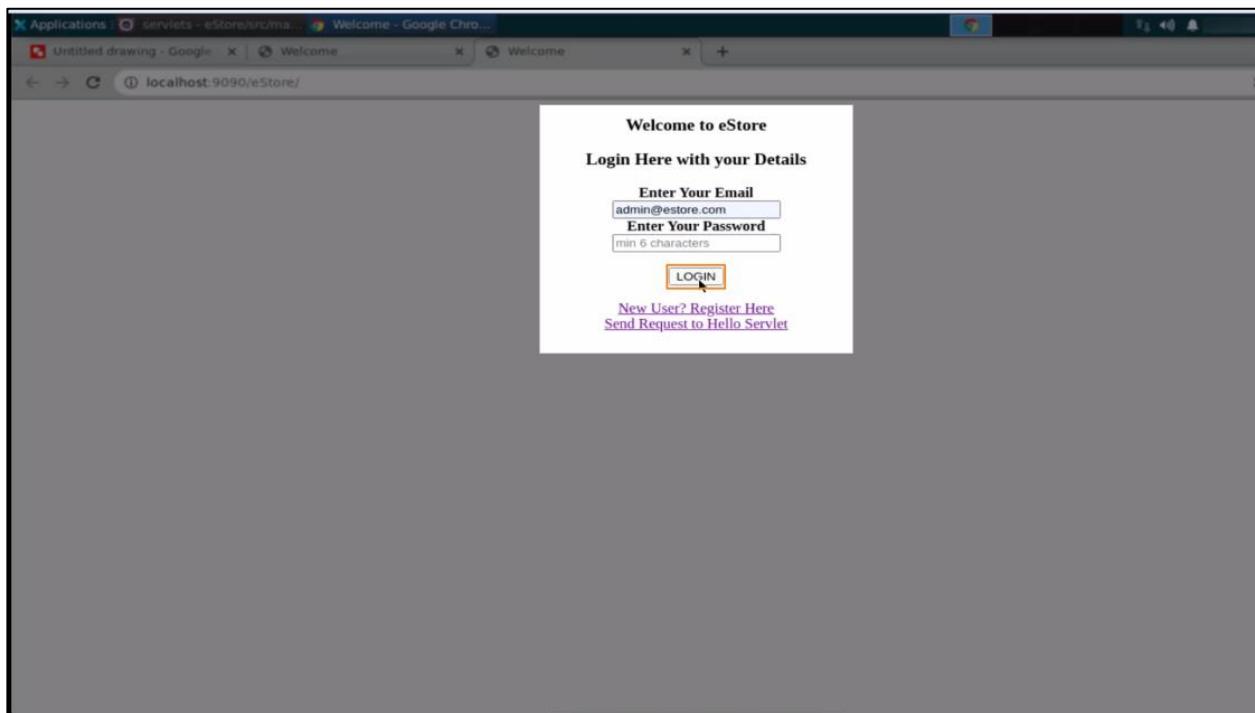
```

The screenshot shows the Eclipse IDE interface with the Project Explorer, Outline, and Console views. The code editor displays LoginFilter.java with the logic for filtering empty email and password fields. A red box highlights the section where the `chain.doFilter()` method is called. The Console view shows the server logs for Apache Tomcat 10.0.13.

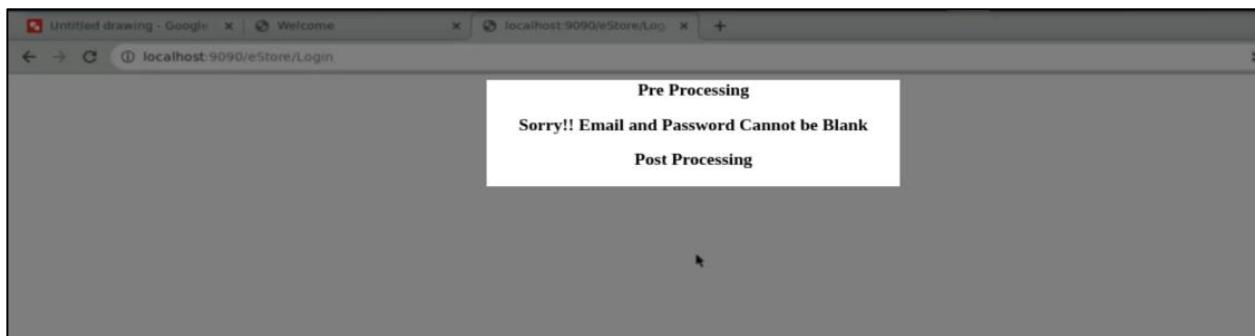
5.8 Right-click on eStore, select Run As, and Run on Server



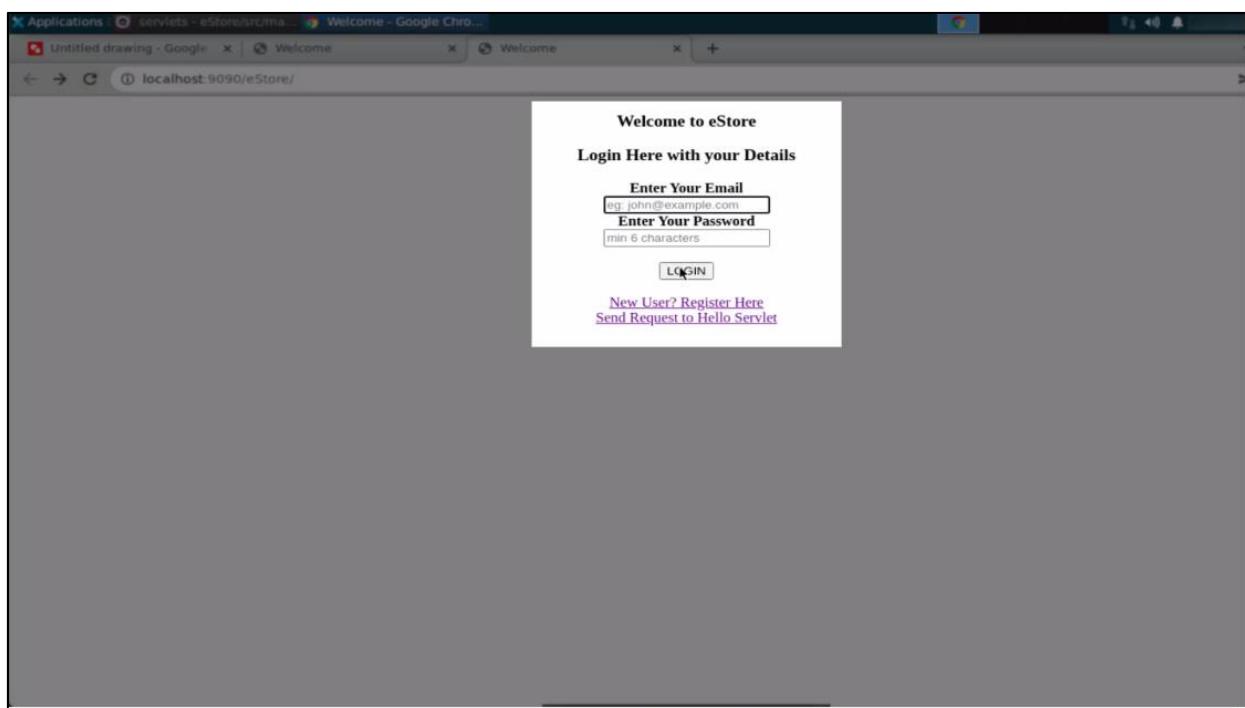
5.9 Enter the email and click on **LOGIN**



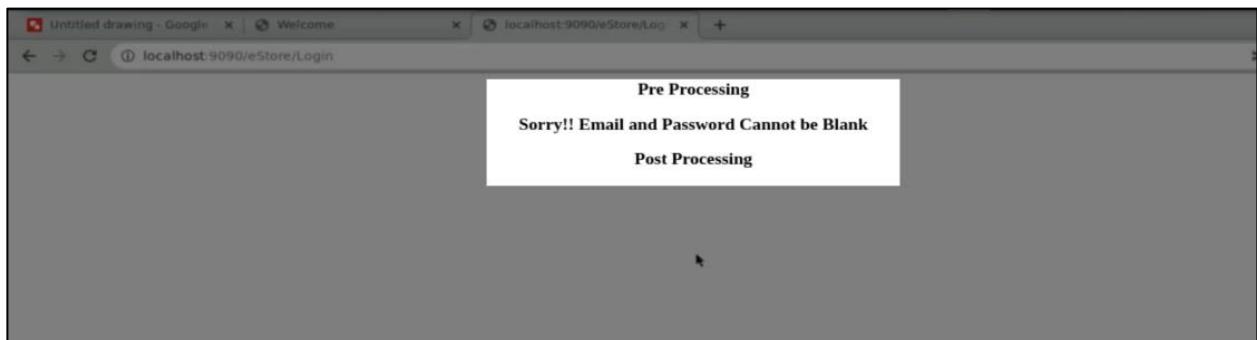
You will see the output: **Sorry!! Email and Password Cannot be Blank**. This means that the filter is working.



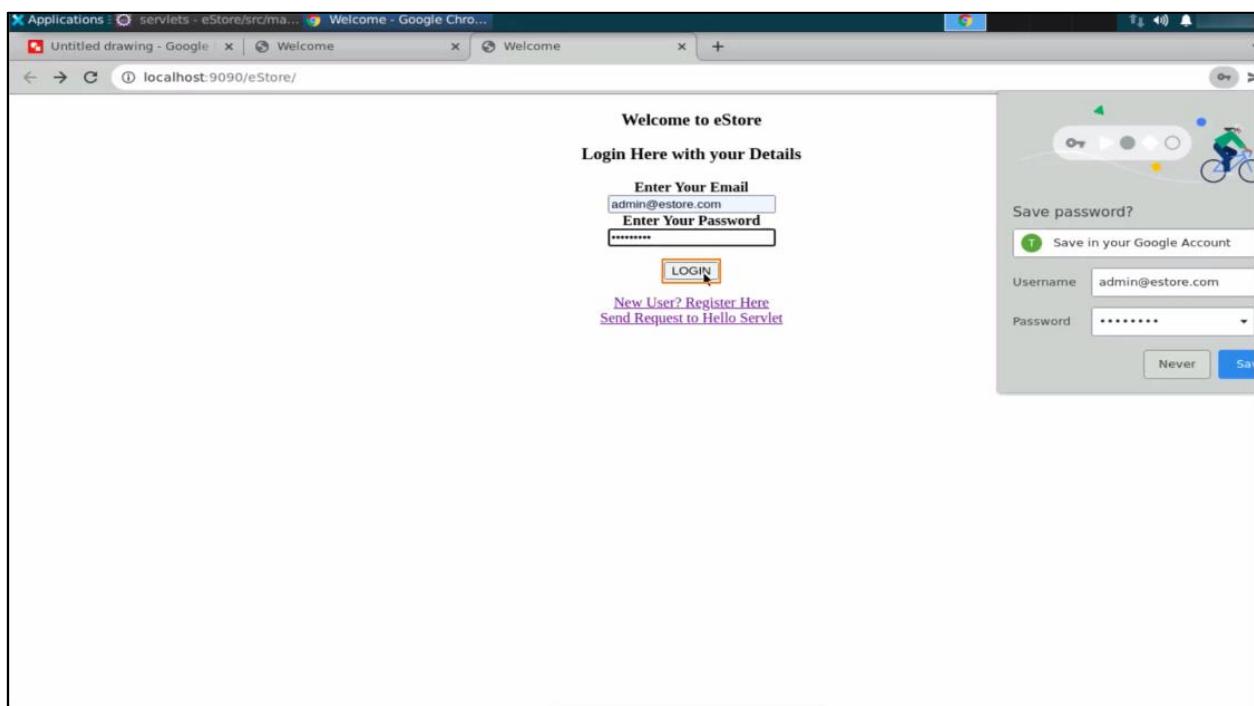
5.10 Keep both the text boxes empty, and click on **LOGIN**



The filter will check for both the username and password, and if either of them is empty, the same output will be displayed.



5.11 Enter the username and password, and click on **LOGIN**



Observation: You can observe the output as **Welcome admin@estore.com**, indicating that the filter has successfully forwarded the request to the Servlet and returned the response to the client.



Following these steps, you have successfully created a new filter and observed the login filter's response, pre-processing, and post-processing.