SIT707 Software Quality and Testing

Pass Task: Integrate web front-end with Java API back end as part of integration testing

Overview

Web applications are often developed as separate front-end and back-end modules where HTML based front ends (or web pages or client interface) interact with the back end (or cloud) through well-defined APIs to receive services including authentication and access cloud resources. Development of front and back ends may progress separately until they are merged to test their integration. In this task you will need to develop HTML pages which will then communicate to the HTTP endpoints (such as Java Servlets). These back-end HTTP request handlers invoke business logic to provide services such as a login service that can access user directory or database to authenticate a login request. You will need to create business logic functions as needed and unit test them as you develop – wearing a developer hat to test all your function features. Next you will need to use login HTML pages and use selenium to do functional testing of this page – wearing a software quality assurance team member's hat. The main goal wearing both hats would be to test the software to break it from dev team and testing team's point of view. This also facilitates to experience an integration testing – connecting front-end to back-end. **In addition**, you would need to justify if the tests generated in this task covered aspects of integration testing and at what level and recommend, if any.

Submission details

Use the instructions on the following page to carry out this task's steps.

Submit the following files to OnTrack.

- A PDF file listing unit test cases from developer point-of-view and functional test cases from test-team point-of-view. You can use a table to describe each test case name, description, intended result and test results.
- A self-reflection on the difference between writing test cases for developers themselves (unit tests) and testing team-members (functional tests). Justify your findings w.r.t. a Venn diagram specially, any untested regions of functional requirements.
- Your justification regarding aspects of integration testing this project covers.
- Your program's source code.
- A screenshot of your GitHub page where your latest project folder is pushed.

You want to focus on the following key ideas, and make sure you can explain them in relation to your program.

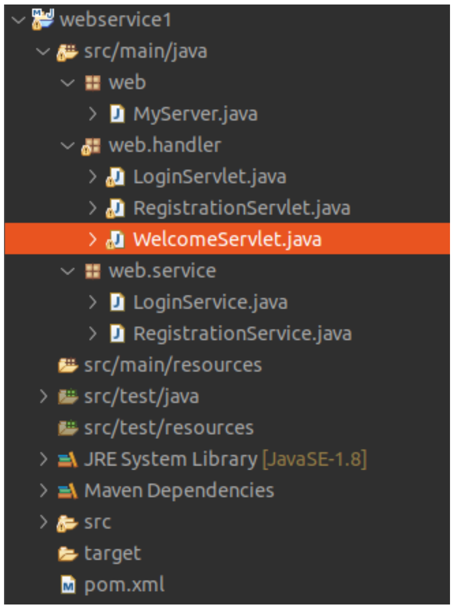
- Unit testing and functional testing.
- Satisfy functional specifications.
- Integration tests in terms of functional testing (integrating front-end with the back end).

Instructions

For this task you will need to

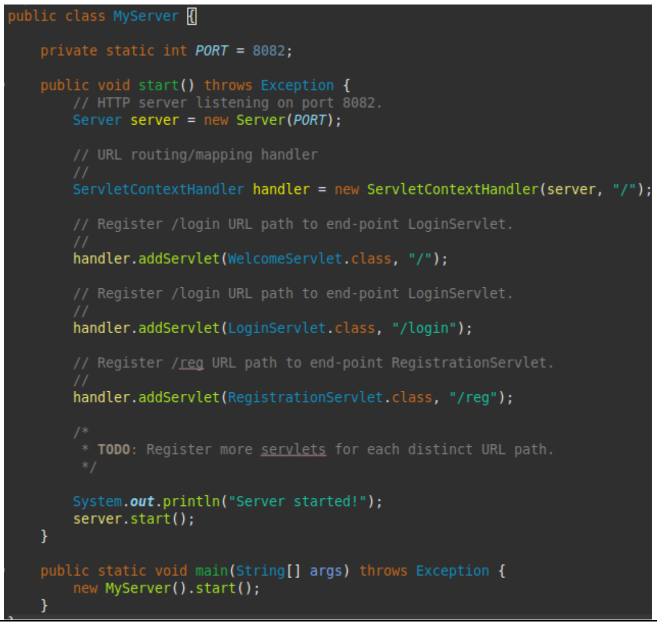
1. Download task8_1P.zip Java project and unzip it in a common folder (say, java_projects) which you will be using to store all the weekly projects.
2. Import the project (as a *maven* project) in Eclipse IDE.
3. Observe the project folder structure as below -

Directory structure



```
webservice1
  src/main/java
    web
      MyServer.java
    web.handler
      LoginServlet.java
      RegistrationServlet.java
      WelcomeServlet.java
    web.service
      LoginService.java
      RegistrationService.java
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
```

MyServer.java

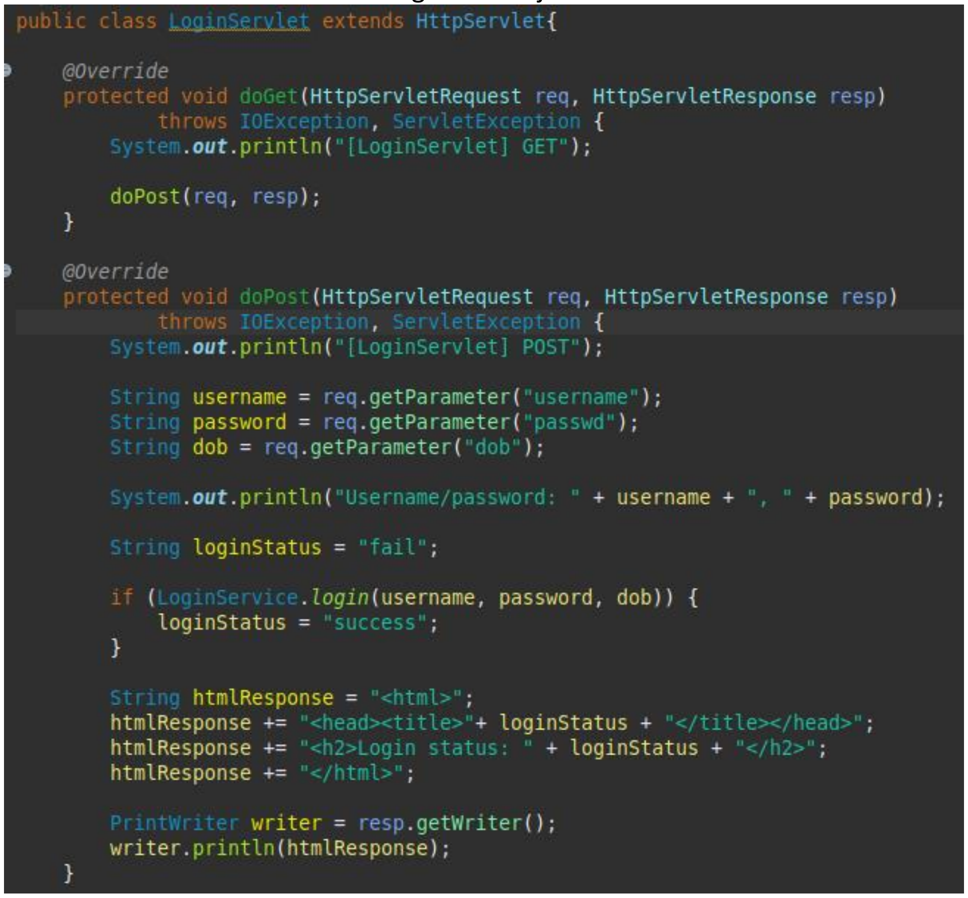


```java
public class MyServer {

    private static int PORT = 8082;

    public void start() throws Exception {
        // HTTP server listening on port 8082.
        Server server = new Server(PORT);

        // URL routing/mapping handler
        //
        ServletContextHandler handler = new ServletContextHandler(server, "/");

        // Register /login URL path to end-point LoginServlet.
        //
        handler.addServlet(WelcomeServlet.class, "/");

        // Register /login URL path to end-point LoginServlet.
        //
        handler.addServlet(LoginServlet.class, "/login");

        // Register /reg URL path to end-point RegistrationServlet.
        //
        handler.addServlet(RegistrationServlet.class, "/reg");

        /*
         * TODO: Register more servlets for each distinct URL path.
         */

        System.out.println("Server started!");
        server.start();
    }

    public static void main(String[] args) throws Exception {
        new MyServer().start();
    }
```

File name	Description
MyServer.java *Package: web*	HTTP server module, you need to run this file to start web server. It listens to port 8082. Open URL http://127.0.0.1:8082 in a browser and you should see a greeting message from the running server.
WelcomeServlet.java *package: web.handler*	Handles URL / (http://127.0.0.1:8082/) and displays a greeting message.
LoginServlet.java *Package: web.handler*	Handles URL /login (http://127.0.0.1:8082/login) and calls LoginService.login() function to authenticate user.
RegistrationServlet.java *Package: web.handler*	Handles URL /reg and calls RegistrationService.register() function. PASS tasks can leave this function as it is.

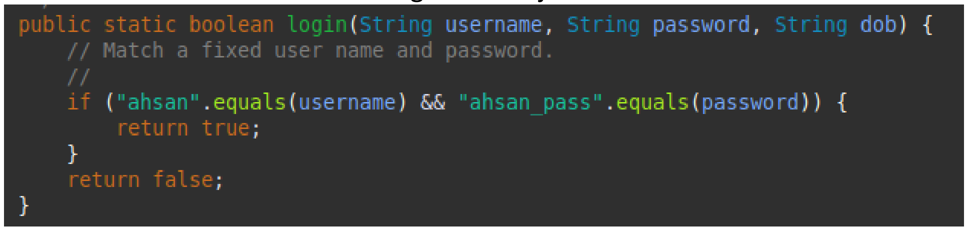
4. The /login URL is routed to LoginServlet which extracts HTML form parameters such as username, password and DoB and calls **LoginService.login()** to authenticate user. **The login function does not make use of DoB field in the example which you need to use for authentication.** Finally, renders a HTML string to display on browser as a response. Note that the generated HTML string creates page title to reflect the login status, this is to help to test the response by checking the page title from Selenium. You can come up with your own response mechanism so that a form submission response can be validated from Selenium side.

LoginServlet.java



```java
public class LoginServlet extends HttpServlet{

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws IOException, ServletException {
        System.out.println("[LoginServlet] GET");

        doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws IOException, ServletException {
        System.out.println("[LoginServlet] POST");

        String username = req.getParameter("username");
        String password = req.getParameter("passwd");
        String dob = req.getParameter("dob");

        System.out.println("Username/password: " + username + ", " + password);

        String loginStatus = "fail";

        if (LoginService.login(username, password, dob)) {
            loginStatus = "success";
        }

        String htmlResponse = "<html>";
        htmlResponse += "<head><title>"+ loginStatus + "</title></head>";
        htmlResponse += "<h2>Login status: " + loginStatus + "</h2>";
        htmlResponse += "</html>";

        PrintWriter writer = resp.getWriter();
        writer.println(htmlResponse);
    }
```

LoginService.java



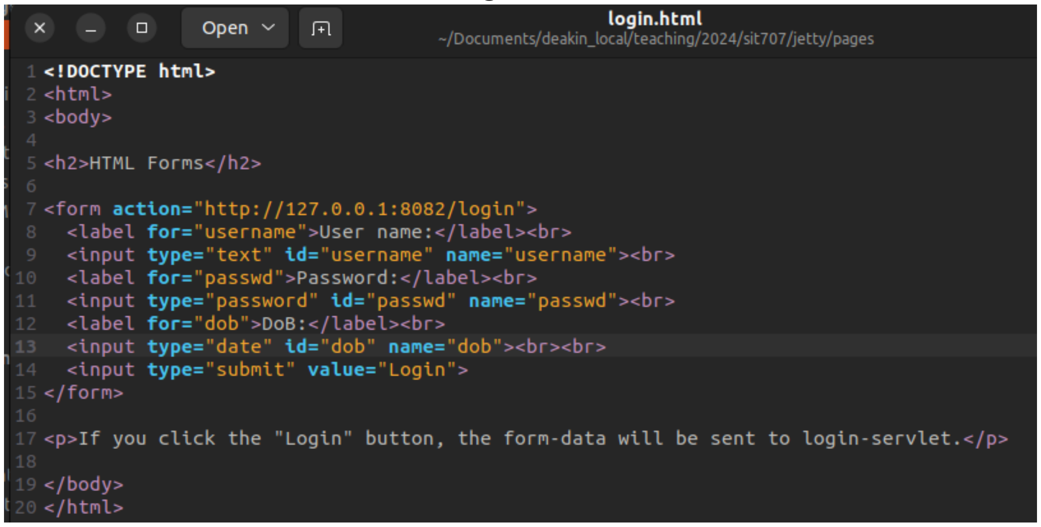
```java
public static boolean login(String username, String password, String dob) {
    // Match a fixed user name and password.
    //
    if ("ahsan".equals(username) && "ahsan_pass".equals(password)) {
        return true;
    }
    return false;
}
```

5. **As a developer**, you must update the login function to make use of all 3 parameters such as username, password and dob (date string format yyyy-mm-dd) and **generate**

unit test cases to test to cover any anomaly using one or more of boundary value, equivalence class or decision-table based tests as appropriate.

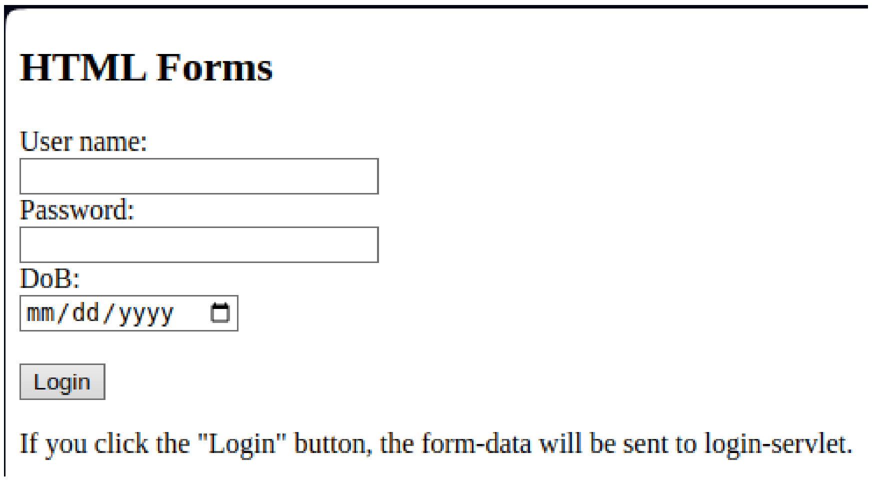
6. **As a test-team member**, you must use Selenium to **write functional test cases** by loading login.html and testing the login function and aim to find function flaws by using all ranges of values including valid and invalid values. You may need to update Java files in **web.service** package (LoginService.java) to enhance the authentication logic and unit test them and update Java files in **web.handler** package (LoginServlet.java) to generate authentication response which can be easily tested in Selenium. Currently response HTML page's title is set to string 'success' or 'fail' based on authentication logic. You may not require updating login.html file, but you should keep the fields same (username, password, and dob).

Login.html



```
<!DOCTYPE html>
<html>
<body>

<h2>HTML Forms</h2>

<form action="http://127.0.0.1:8082/login">
  <label for="username">User name:</label><br>
  <input type="text" id="username" name="username"><br>
  <label for="passwd">Password:</label><br>
  <input type="password" id="passwd" name="passwd"><br>
  <label for="dob">DoB:</label><br>
  <input type="date" id="dob" name="dob"><br><br>
  <input type="submit" value="Login">
</form>

<p>If you click the "Login" button, the form-data will be sent to login-servlet.</p>

</body>
</html>
```

Web view



HTML Forms

User name:

Password:

DoB:

mm/dd/yyyy

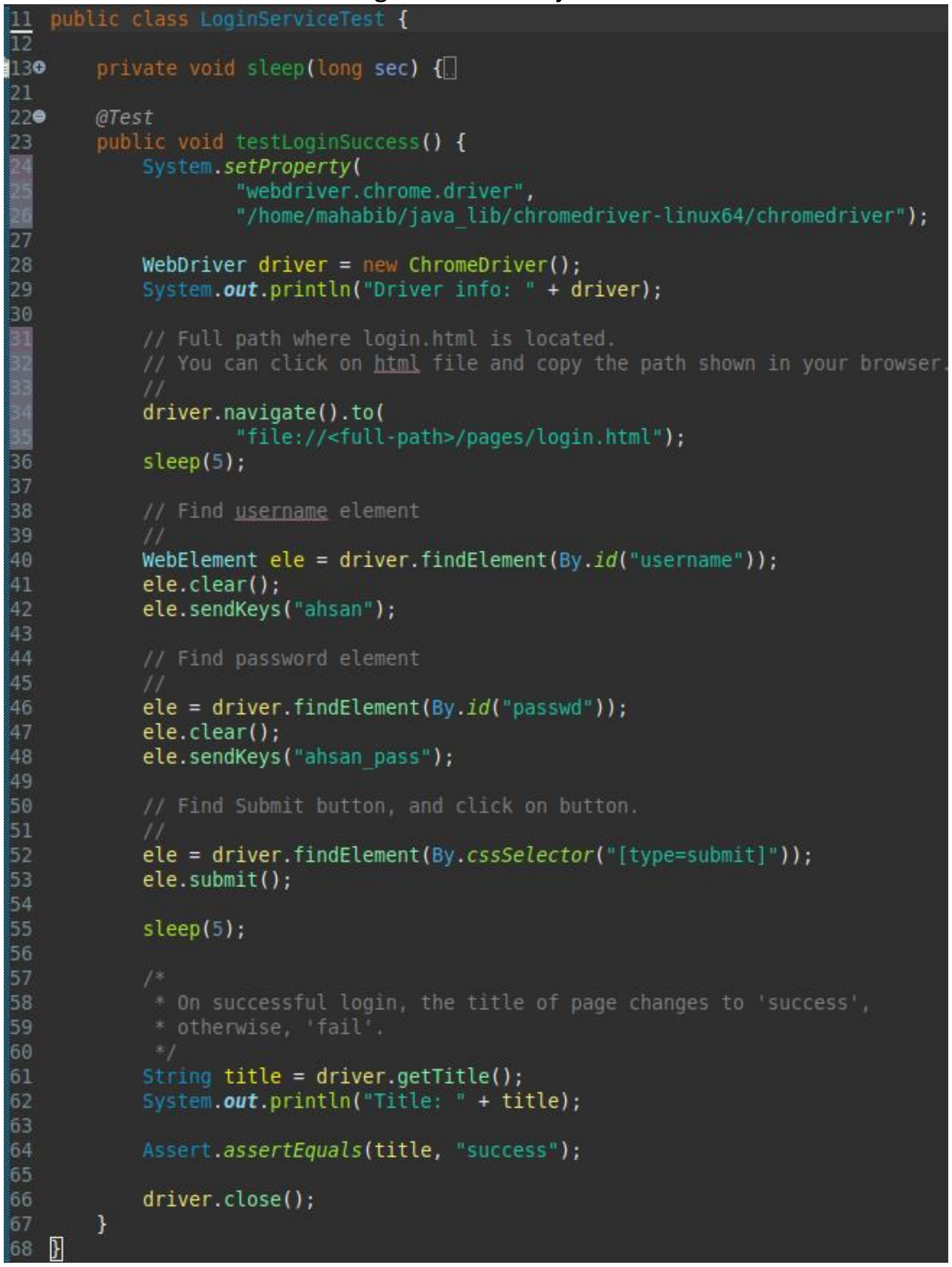
If you click the "Login" button, the form-data will be sent to login-servlet.

7. Selenium test example is shown in below figure and described in a table.

Line 34	Selenium web-driver opens login.html page.

Line 40-48	Locates fields including username and passwd and fills in new values.
Line 52-53	Locates submit button and clicks on it to submit the form.
Line 64	Verifies authentication response if it equals to a page title “success”.

LoginServiceTest.java



```
11 public class LoginServiceTest {
12
13     private void sleep(long sec) {□
21
22     @Test
23     public void testLoginSuccess() {
24         System.setProperty(
25                 "webdriver.chrome.driver",
26                 "/home/mahabib/java_lib/chromedriver-linux64/chromedriver");
27
28         WebDriver driver = new ChromeDriver();
29         System.out.println("Driver info: " + driver);
30
31         // Full path where login.html is located.
32         // You can click on html file and copy the path shown in your browser
33         //
34         driver.navigate().to(
35                 "file://<full-path>/pages/login.html");
36         sleep(5);
37
38         // Find username element
39         //
40         WebElement ele = driver.findElement(By.id("username"));
41         ele.clear();
42         ele.sendKeys("ahsan");
43
44         // Find password element
45         //
46         ele = driver.findElement(By.id("passwd"));
47         ele.clear();
48         ele.sendKeys("ahsan_pass");
49
50         // Find Submit button, and click on button.
51         //
52         ele = driver.findElement(By.cssSelector("[type=submit]"));
53         ele.submit();
54
55         sleep(5);
56
57         /*
58          * On successful login, the title of page changes to 'success',
59          * otherwise, 'fail'.
60          */
61         String title = driver.getTitle();
62         System.out.println("Title: " + title);
63
64         Assert.assertEquals(title, "success");
65
66         driver.close();
67     }
68 }
```

8. You will need to update the LoginServiceTest.java file to add additional functional test cases using Selenium.
9. You will need to create a new test file to include your unit test cases.
10. Upload your project folder to your GitHub account and take a screenshot.

Your Task

For this task, you will need to -

1. Study login.html page and webserivce1 Eclipse Java project's source and test files to understand how the HTML form in login.html is submitted to interact with Java web handler function located in LoginServlet.java. For this, you will need to run the MyServer.java file to start HTTP web server.
2. Improve authentication logic in LoginService.java and write unit test cases (create a new test file) to cover above 90% code coverage.
3. Update LoginServlet.java to generate HTTP response suitable for Selenium to receive it so functional tests can be done easily. Currently, the response web page's title reflects the authentication response.
4. Update LoginServiceTest.java to include more functional test cases based on Selenium.
5. Your goal should be to critically write test cases from both developer and test-team member's point of view to find flaws in the system.
6. Follow submission instructions on the first page and submit your work.

Submit your work

When you are ready, login to OnTrack and submit your pdf which consolidates all the items mentioned in the submission detail section above. Remember to save and backup your work.

Complete your work

After your submission, your OnTrack reviewer (tutor) will review your submission and give you feedback in about 5 business days. Your reviewer may further ask you some questions on the weekly topics and/or about your submissions. You are required to address your OnTrack reviewer's questions as a form of task discussions. Please frequently login to OnTrack for the task ***Discuss/Demonstrate*** or ***Resubmit*** equivalent to fix your work (if needed) based on the feedback to get your task signed as ***Complete***.