Software Testing (S731)

Client-side web applications testing (bypass testing)

PRAKHAR GARG MT2021095

NAVNEET SINGH MT2021081

Objective:

Projects that test a web application's client-side code by creating test cases that bypass client-side validation and send modified or corrupt input to the server.

Code Description:

We have made a Users portal login and signup and used the code for testing. The website employs a SQL database and SpringBoot as backend, respectively. We have mostly tested the Login and Signup components of the website. In order to test our backend for responses to unexpected inputs, we therefore bypass the client side of our program. An individual who has valid user credentials must "log in" to any application in order to access it. 'Logging in' is typically needed to access a particular page that is hidden from trespassers.

We'll see the "Test Scenarios for Login Page" in this project. For the security component of any application, testing the login page is crucial. Here, we'll attempt to cover the most common Login Page scenarios, including functional test cases.

Strategy Used for Testing:

The World Wide Web offers programmers a fresh method for distributing sophisticated, interactive applications with intricate user interfaces and numerous back-end software components combined in creative and intriguing methods. Web applications are made up of various software elements that interact with one another and with users in fresh ways. Web software components must adhere to strict reliability, availability, and usability standards while being spread across numerous computers and organizations, frequently produced and integrated dynamically, written in various languages, and running on various hardware platforms. Software engineers face both new challenges and gain powerful new skills due to these traits.

Therefore, it's crucial to conduct thorough testing for any website on the internet. One such test is bypass testing, which essentially examines the server's response when client-side code is forcibly bypassed. Because Web applications use a strange combination of client-server, HTML GUI, and JavaScript technologies, bypass testing provides a distinct and innovative technique to develop test cases.

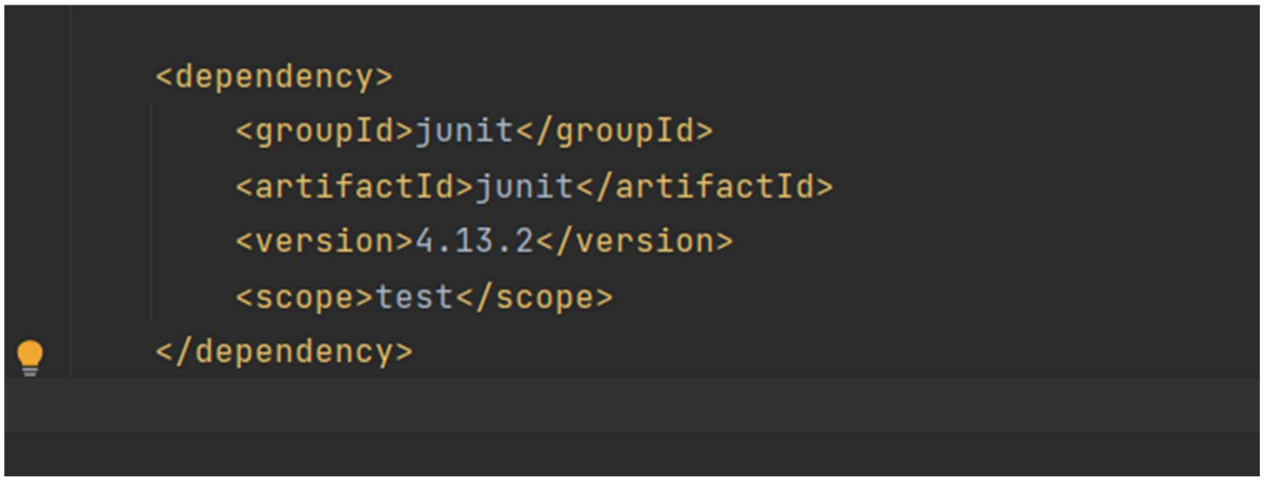
The HTML form elements are examined in order to implement bypass testing, and each input element is treated as a parameter. Values are selected to go against each constraint set by HTML and JavaScript. The values are currently selected using straightforward criteria, however we intend using regular expressions to specify the limits and then alter the expressions in a manner similar to mutation to produce invalid inputs..

Tools used for Testing:

To run and test our code, we utilised the IntelliJ IDE, while JUnit served as the testing framework for designing and building simple test cases. In our project, we used Maven as a dependency resolver. The dependence on JUnit is seen in Figure 1.

Bypass Testing Steps:

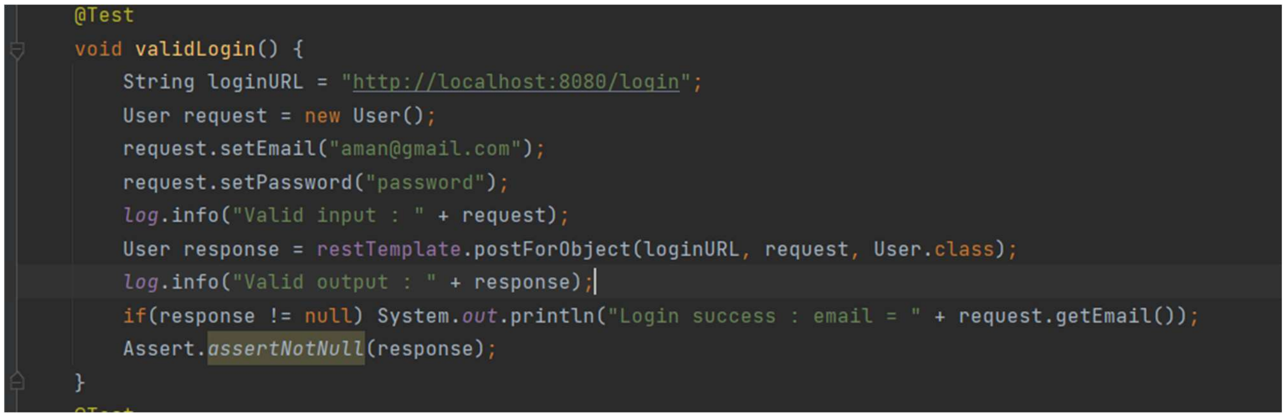
1. 1. Use Maven Dependency Resolver to download the JUnit dependency.



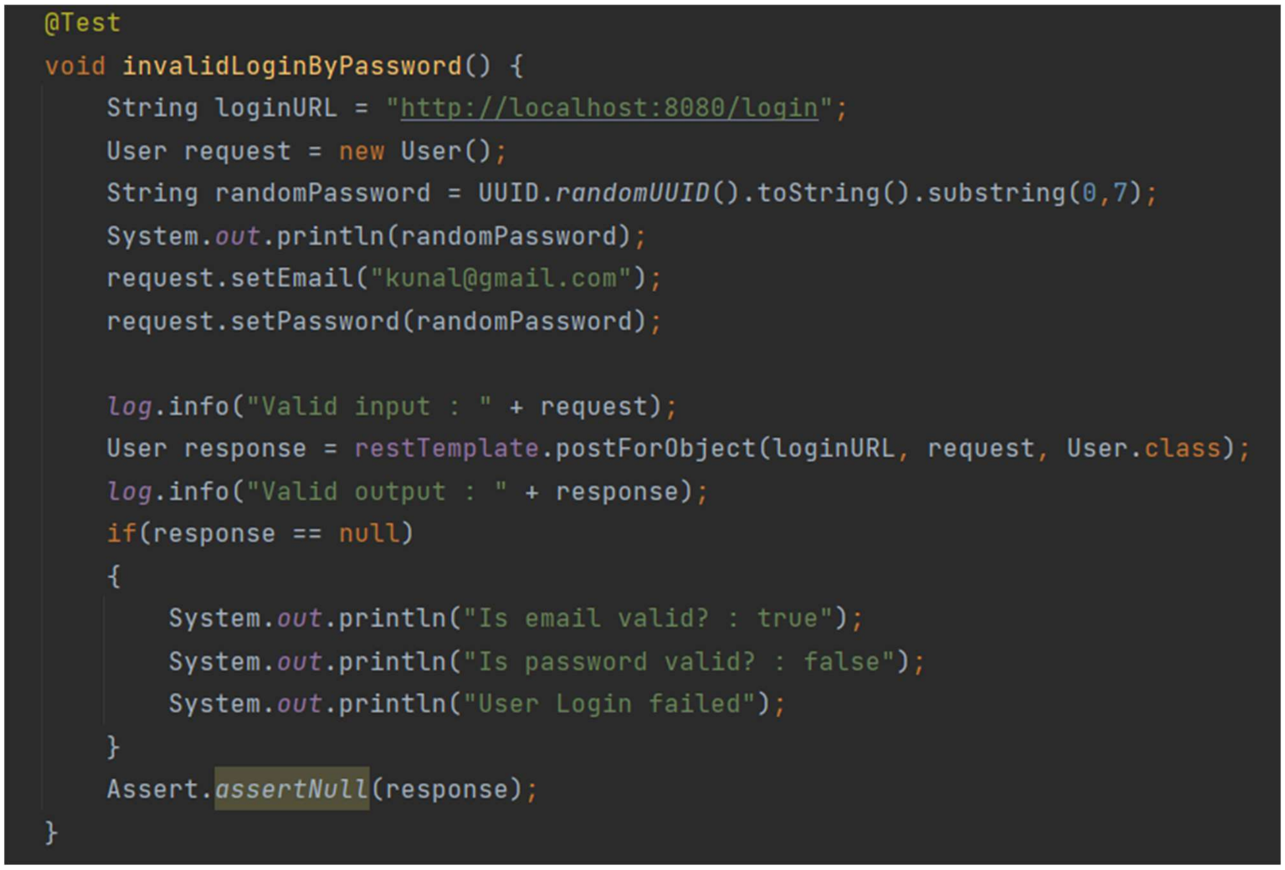
```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

Figure 1: JUnit dependency

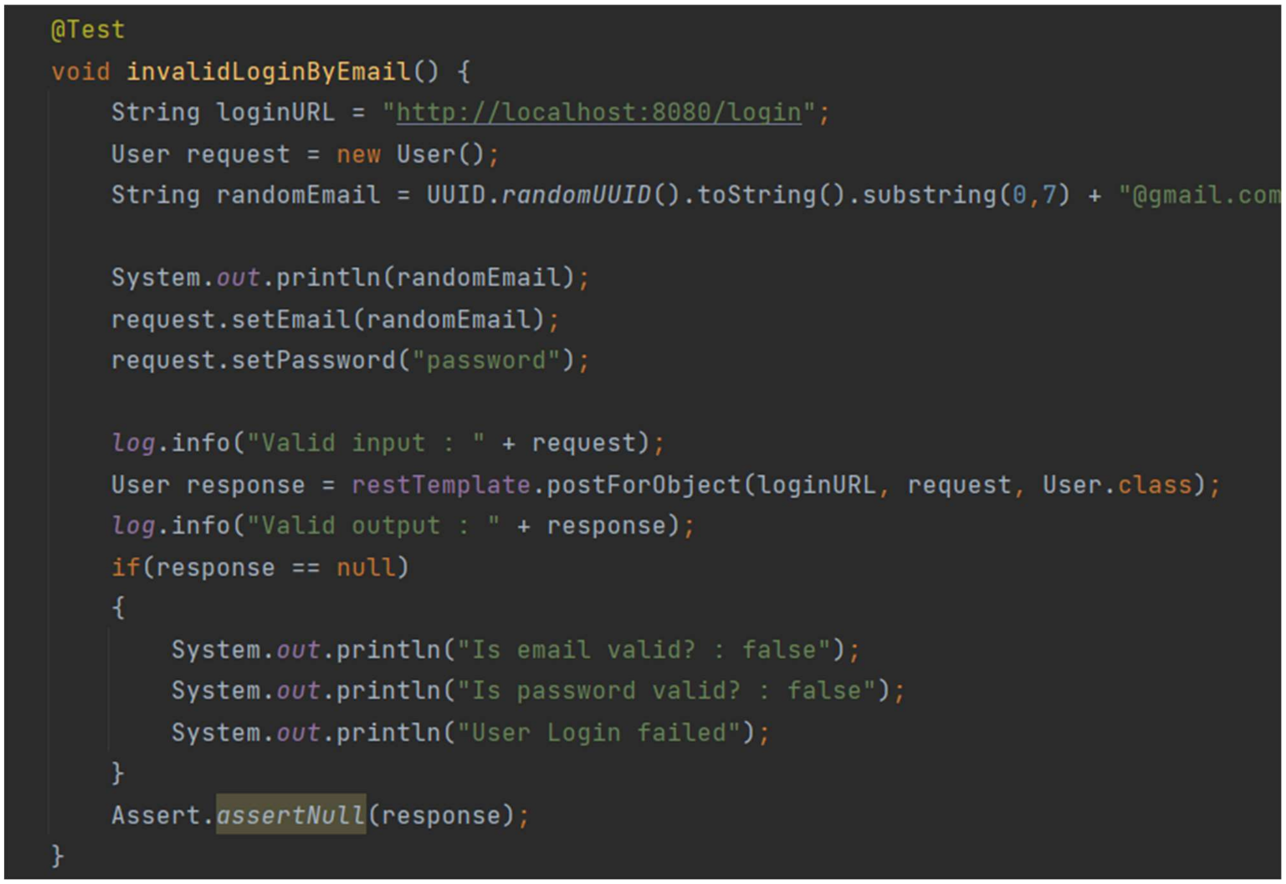
2. In the bypass testing technique, we transmit the inputs to the server and bypass the client-side scripts and capture the server's response. We have added backend validations for the signup script. The validation for each input is displayed in Figure 3.



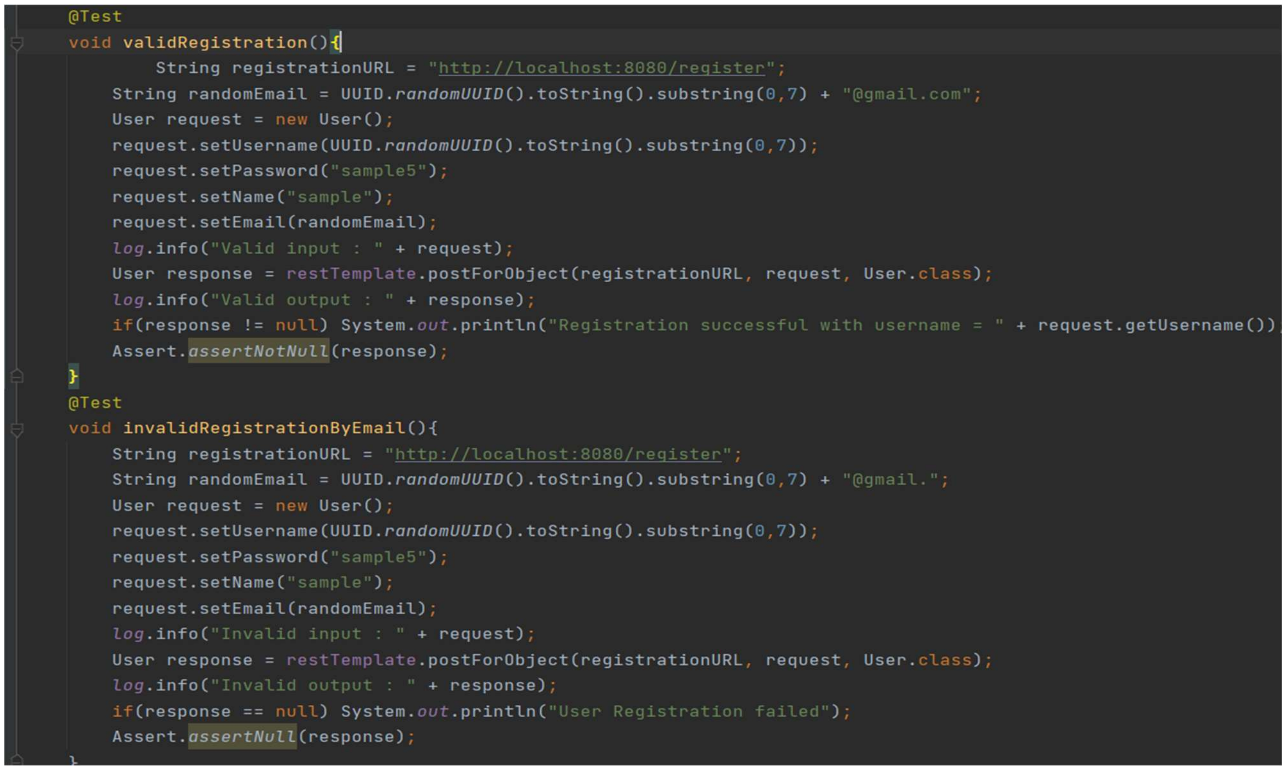
```java
@Test
void validLogin() {
    String loginURL = "http://localhost:8080/login";
    User request = new User();
    request.setEmail("aman@gmail.com");
    request.setPassword("password");
    log.info("Valid input : " + request);
    User response = restTemplate.postForObject(loginURL, request, User.class);
    log.info("Valid output : " + response);
    if(response != null) System.out.println("Login success : email = " + request.getEmail());
    Assert.assertNotNull(response);
}
```



```java
@Test
void invalidLoginByPassword() {
    String loginURL = "http://localhost:8080/login";
    User request = new User();
    String randomPassword = UUID.randomUUID().toString().substring(0,7);
    System.out.println(randomPassword);
    request.setEmail("kunal@gmail.com");
    request.setPassword(randomPassword);

    log.info("Valid input : " + request);
    User response = restTemplate.postForObject(loginURL, request, User.class);
    log.info("Valid output : " + response);
    if(response == null)
    {
        System.out.println("Is email valid? : true");
        System.out.println("Is password valid? : false");
        System.out.println("User Login failed");
    }
    Assert.assertNull(response);
}
```



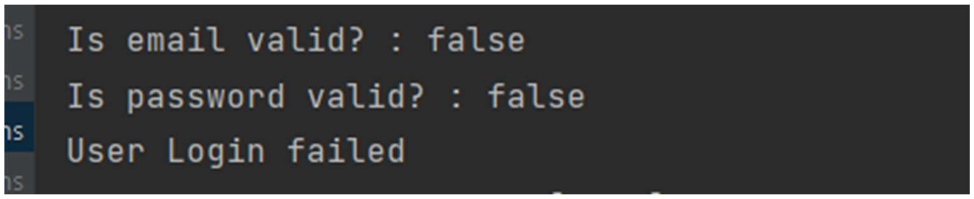
```java
@Test
void invalidLoginByEmail() {
    String loginURL = "http://localhost:8080/login";
    User request = new User();
    String randomEmail = UUID.randomUUID().toString().substring(0,7) + "@gmail.com

    System.out.println(randomEmail);
    request.setEmail(randomEmail);
    request.setPassword("password");

    log.info("Valid input : " + request);
    User response = restTemplate.postForObject(loginURL, request, User.class);
    log.info("Valid output : " + response);
    if(response == null)
    {
        System.out.println("Is email valid? : false");
        System.out.println("Is password valid? : false");
        System.out.println("User Login failed");
    }
    Assert.assertNull(response);
}
```



```java
@Test
void validRegistration(){
        String registrationURL = "http://localhost:8080/register";
    String randomEmail = UUID.randomUUID().toString().substring(0,7) + "@gmail.com";
    User request = new User();
    request.setUsername(UUID.randomUUID().toString().substring(0,7));
    request.setPassword("sample5");
    request.setName("sample");
    request.setEmail(randomEmail);
    log.info("Valid input : " + request);
    User response = restTemplate.postForObject(registrationURL, request, User.class);
    log.info("Valid output : " + response);
    if(response != null) System.out.println("Registration successful with username = " + request.getUsername())
    Assert.assertNotNull(response);
}
@Test
void invalidRegistrationByEmail(){
    String registrationURL = "http://localhost:8080/register";
    String randomEmail = UUID.randomUUID().toString().substring(0,7) + "@gmail.";
    User request = new User();
    request.setUsername(UUID.randomUUID().toString().substring(0,7));
    request.setPassword("sample5");
    request.setName("sample");
    request.setEmail(randomEmail);
    log.info("Invalid input : " + request);
    User response = restTemplate.postForObject(registrationURL, request, User.class);
    log.info("Invalid output : " + response);
    if(response == null) System.out.println("User Registration failed");
    Assert.assertNull(response);
}
```

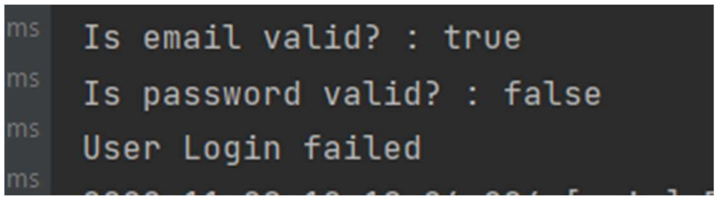
Figure3. server-side validation scripts

3. The testing procedure is automated via JUnit. We created two different test cases: those with invalid input and those with valid input. And in both instances, we captured the server's answer.



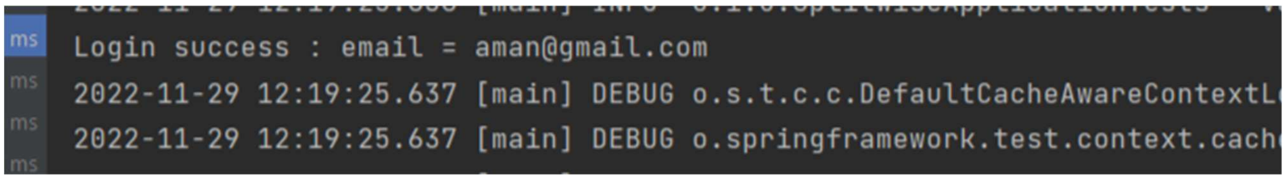
```
Is email valid? : false
Is password valid? : false
User Login failed
```

Figure4. Invalid test case result



```
Is email valid? : true
Is password valid? : false
User Login failed
```

Figure 5. Invalid test case result



```
Login success : email = aman@gmail.com
2022-11-29 12:19:25.637 [main] DEBUG o.s.t.c.c.DefaultCacheAwareContextL
2022-11-29 12:19:25.637 [main] DEBUG o.springframework.test.context.cach
```

Figure 6. Valid test case result

References:

1. Jeff Offutt, Ye Wu, Xiaochen Du and Hong Huang. 2014. Bypass testing of web applications. Information and Software Engineering.