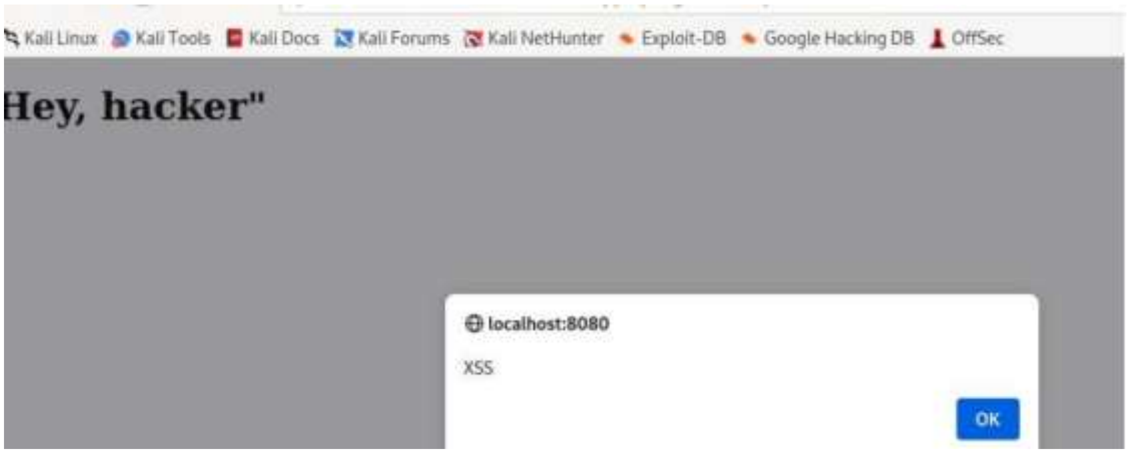
4.2C: Injection attack preventions using hibernate validator

- In this assignment, I also elaborated more on shields of the 05-coachwebapp-spring application against injections one type in particular – Cross-Site Scripting (XSS). To avoid users uniquely inserting gainful scripts or labelling input data as <> using the format, I improved the efficiency of the regular expression pattern in Task 4.1P.

Testing Regex

- To rule out these inputs I changed the regex after confirming with the server-sided 05-coachwebapp-spring that it was still recognizing the inputs. I used the following data to test the initial implementation:
  First Test - hacker "<script>alert('XSS');</script>"
- As a consequence of this input getting past the regex, the script ran while the application itself remained vulnerable to simple XSS injection.



Hey, hacker"

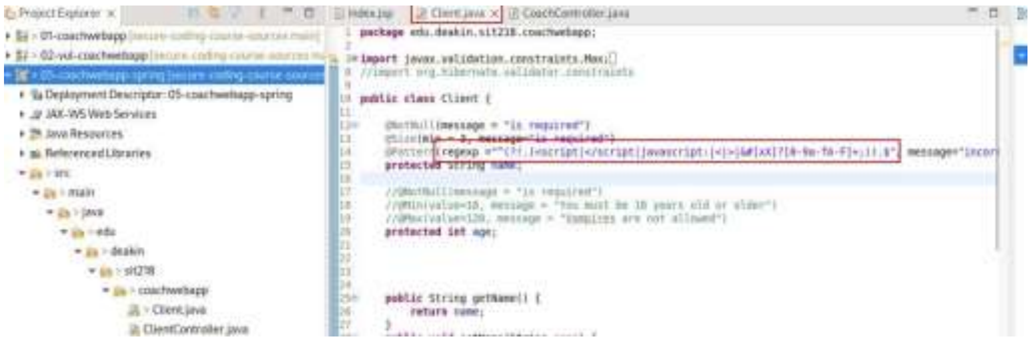
localhost:8080

XSS

Corrected Regex Pattern

- I modified the regex pattern to accept these patterns to avoid character encoding. Extensions like &#97; for a, deal with obfuscated or altered script tags with attributes and block basic HTML tags such as <Script>, <a>, <img>, and other HTML Basic tags. –

@Pattern(regexp = "^(?!.*(]+>|<script|</script|javascript:|vbscript:|on\\w+|alert\\(|&#[xX]?[0-9a-fA   F]+;)).*$", message = "incorrect format")

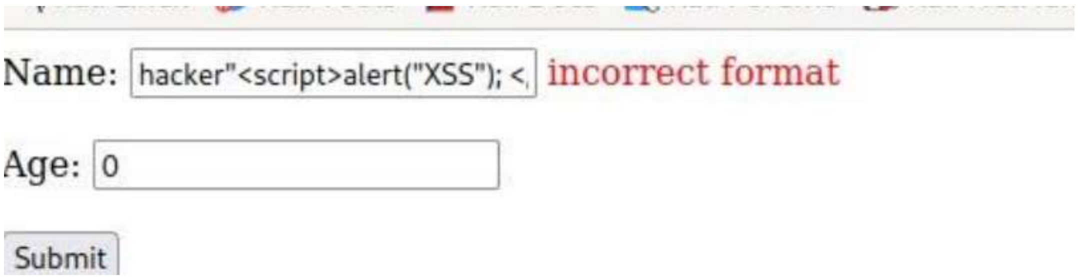


```
package edu.deakin.sit218.coachwebapp;

import javax.validation.constraints.Max;
//import org.hibernate.validator.constraints

public class Client {

    @NotNull(message = "is required")
    @Size(min = 3, message="is required")
    @Pattern(regexp ="^(?!.*(<script|</script|javascript:|<|>|&#[xX]?[0-9a-fA-F]+;)).*$", message="incor
    protected String name;

    //@NotNull(message = "is required")
    //@Min(value=18, message = "You must be 18 years old or older")
    //@Max(value=120, message = "Vampires are not allowed")
    protected int age;



    public String getName() {
        return name;
    }
```

- It can be noted that the pattern that is designed by using negative look ahead prevents common XSS vectors allowed the input which contains the unsafe tags, event handlers, encoded characters, and script injection techniques. It ensures the input is clean from the XSS inoculating both the plain and obscured patterns.

Tested the corrected one

Test - “<script>alert(‘XSS’);</script>”
Output - : The script was successfully blocked, and no execution occurred



Name: hacker"<script>alert("XSS"); < incorrect format

Age: 0

- As long as I tuned the regular expression a little, I managed to block all the hazardous inputs that bypassed the first check. With this regex adjusted, simple and encoded XSS injections, as well as other complex injections will also be blocked by the system. This enhances the security guard against XSS attacks for the 05-coachwebapp-spring application.

References

'5 methods for Bypassing XSS Detection in WAFs' (2022) Volkis [Preprint]. https://www.volkis.com.au/blog/bypass-xss-in-wafs/.