2019

Turvallinen ohjelmointi - kurssin harjoitustyö

HIRSIPUU-PELI

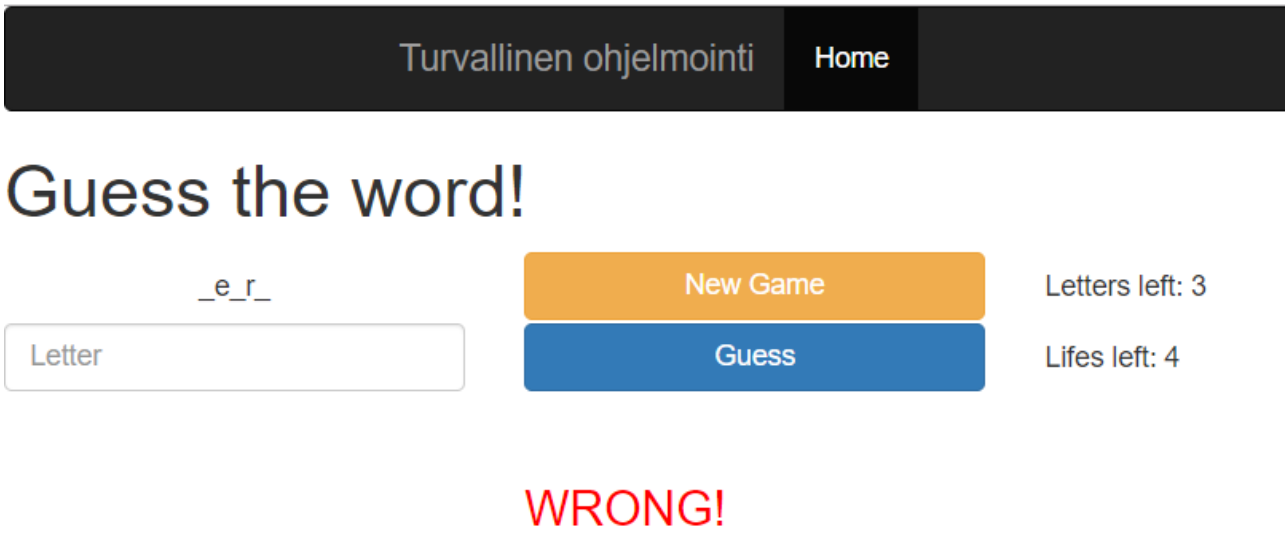
NICO AHOLA

1. Johdanto

Tässä dokumentissa kuvataan kurssin Turvallinen ohjelmointi harjoitustyön toteutus. Harjoitustyönä toteutettiin hirsipuupeli tyyppinen web-sovellus. Ohjelmointikielenä käytettiin Javaa (back-end) ja JavaScriptiä (front-end). Frameworkina toimii Spring Boot, joka on käytännössä kattavampi versio normaaliin Springiin verrattuna. Spring ja erityisesti sen moduuli Spring Security toteuttavat suuren osan sovelluksen tietoturvasta ilman käyttäjän erikseen tekemiä asetuksia, minkä takia nämä sisältävä Spring Boot ja tavukoodiksi kääntyvä Java ovat hyviä vaihtoehtoja tietoturvallisien sovelluksien kehitykseen.

2. Sovelluksen kuvaus

Sovellus on peli, jossa käyttäjälle näytetään alaviivoin jokin sana. Käyttäjän tulee arvata kirjain kerrallaan mikä sana on kyseessä. Väärästä vastauksesta pelaaja menettää elämän, mutta oikeasta vastauksesta kirjain(/kirjaimet) asetetaan ja näytetään oikealla paikallaan sanassa. Peli päättyy, kun käyttäjä on arvannut kaikki sanan kirjaimet ennen elämien loppumista tai kun elämät loppuvat. Kuvassa 1 on pelin peliruutu.



Kuva 1: Pelinäkymä

Varsinaisen pelin lisäksi sovelluksessa on kirjautumis- ja rekisteröitymistoiminnallisuus. Käyttäjä syöttää rekisteröitymislomakkeeseen haluamansa käyttäjänimen ja salasanan, sekä salasanan uudestaan varmistuksena. Kun käyttäjä on syöttänyt sääntöjen mukaisen tunnuksen ja salasanan, käyttäjä tallennetaan tietokantaan. Tämän jälkeen käyttäjä voi kirjautua sisään luomillaan tunnuksilla. Vain kirjautuneet käyttäjät voivat pelata peliä. Kirjautumattomille käyttäjille näytetään vain kirjautumis- ja rekisteröitymissivut. Kirjauduttuaan käyttäjä voi kirjautua ulos, minkä jälkeen hänet ohjataan takaisin kirjautumissivulle.

3. Tietoturvallisuus

3.1 Konfigurointi

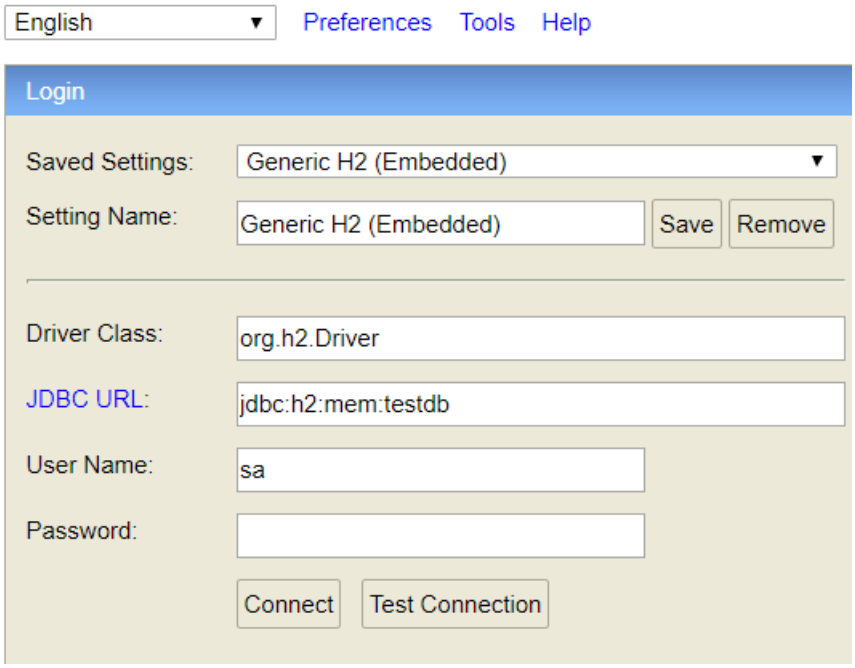
Sovelluksessa käytetty Spring Boot -framework tarjoaa useita riippuvuuksia, jotka helpottavat web-sovelluksien kehitystä [https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle]. Yksi näistä riippuvuuksista on Spring Security, jonka sisällyttäminen suojaa sovelluksen automaattisesti. Tässä työssä Spring Securityn automaattisesti käytössä olevasta UserDetailsService-luokasta on tehty oma toteutus UserDetailsServiceImpl, jolla korvataan oletuskirjautumisnäkymä. Myös Spring Securityn WebSecurityConfigurerAdapter-luokkaa on laajennettu vastaavaan paremmin tämän sovelluksen tarpeita. Configuration- ja EnableWebSecurity-annotaatiot varmistavat, että koko oletussuojaus ei katoa [https://www.baeldung.com/spring-boot-security-autoconfiguration]. Annotaatiot ovat @-merkillä varustettuja toiminnallisuuksia, jotka piilottavat paljon ohjelmoijalle ”turhaa koodia”.

Oleellisin konfigurointiluokan metodi on configure, jossa määritetään muun muassa sallitut URL-osoitteet ja kirjautumisnäkymä. Tässä työssä kaikille sallituiksi osoitteiksi on asetettu ”/login” ja ”/registration”, jotka ovat siis kirjautumis- ja rekisteröitymisnäkymät. Muihin osoitteisiin vaaditaan autentikoituminen, eli käytännössä kirjautuminen. Lisäksi ”/resources/**” ja ”/webjars/**” on asetettu sovellukselle sallituiksi tarpeellisiin resursseihin (esimerkiksi Bootstrap) pääsemiseksi.

Konfiguroinnissa käytetään myös passwordEncoder-metodia, joka palauttaa BCryptPasswordEncoder-olion. BCryptPasswordEncoder on tässä työssä, ja Springissä yleisestikin käytetty salasanan kryptaamisessa käytetty luokka. Se käyttää BCrypt nimistä hash-funktiota, jonka oletusarvoinen kryptauskierrosten iteraatioiden määrä on 1024 (2^10) [https://docs.spring.io/spring-security/site/docs/4.2.12.RELEASE/apidocs/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html]. Potenssin arvon voi valita väliltä 4-31. BCrypt perustuu symmetriseen Blowfish-lohkosalaajaan, ja se sisältää suolauksen, minkä ansiosta samoilla selkokielisillä salasanoilla on eri tiivistearvo [https://www.stubbornjava.com/posts/hashing-passwords-in-java-with-bcrypt]. Blowfish on avainten vaihdossa käytettynä muita lohkosalaajia hitaampi, mikä yhdistettynä useisiin iteraatioihin tiivistettä laskiessa, tekee BCryptistä hitaan [https://auth0.com/blog/hashing-in-action-understanding-bcrypt/]. Hitaus on salasanoja salattaessa etu, koska tällöin brute-force hyökkäykset käyvät helposti liian tehottomiksi toteuttaa käytännössä. Koska iteraatioiden määrää voidaan kasvattaa yhdellä parametrin arvolla, BCrypt pystyy sopeutumaan kasvavaan laskentatehoon ajan kuluessa.

Työn tietokantana käytettiin H2:ta, joten menemällä URL-osoitteiseen ”localhost:8080/h2-console” päästään tekemään tietokantaan kyselyjä. Osoitteesta avautumaan tulee syöttää kuvan 2 mukaiset tiedot päästäkseen tekemään kyselyjä. Tässä työssä ei ole kuin USER-taulu, jonne tekemällä kyselyn ”SELECT * FROM USER” saadaan kaikki käyttäjät salasanoineen ja ID:neen näkyviin kuvan 3 näyttämällä tavalla. Kuvasta huomataan, että rekisteröityneen ”Usser” käyttäjän salasana todellakin

on salattu BCrypt:llä. Toinen käyttäjä on testikäyttäjä, jonka salattu salasana syötettiin manuaalisesti ohjelmakoodin puolella.

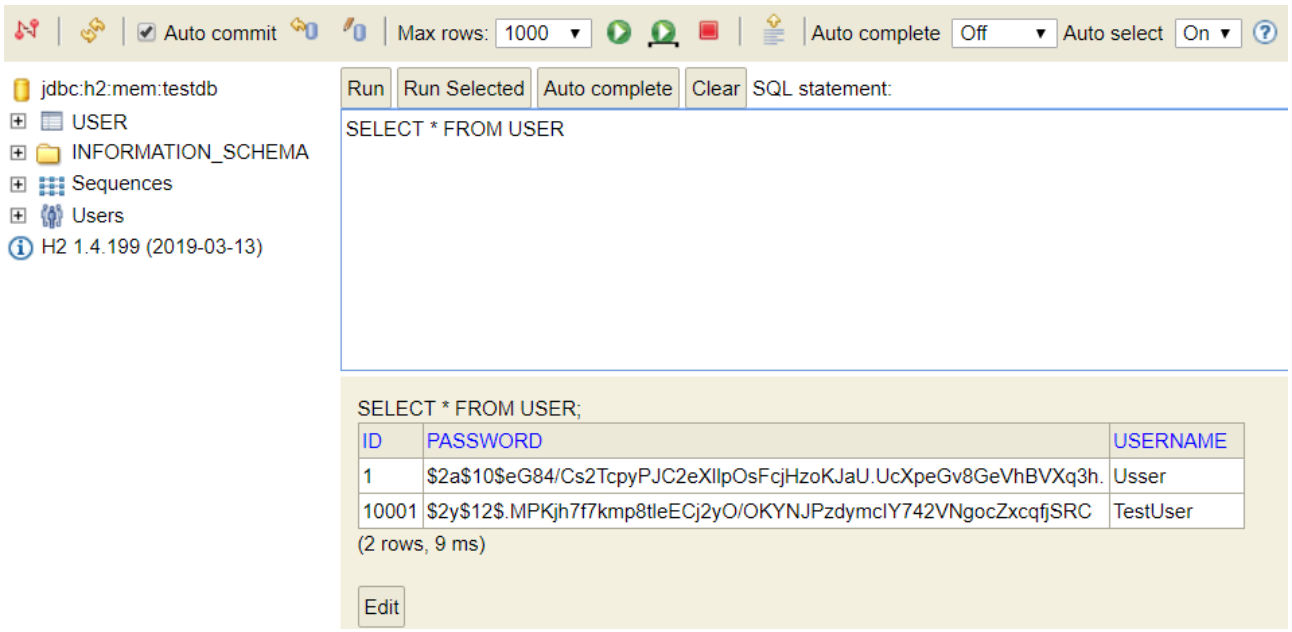


Kuva 2: H2-konsolin avausnäkymä

Jotta Spring Security ei estä H2-konsolia, konfigurointiluokan ensimmäiseen configure-metodiin tulee lisätä

```
http.csrf().disable();
http.headers().frameOptions().disable();
```

ja lisäksi antMatchers():n sisään pitää lisätä ” "/h2_console/**" ”.



Kuva 3: USER-taulu

3.2 Rekisteröityminen ja kirjautuminen

Käyttäjän rekisteröinnissä käyttäjänimen täytyy olla 2-20 merkkiä pitkä ja koostua vain kirjaimista tai numeroista. Käyttäjänimi on uniikki, joten käyttäjä ei voi luoda käyttäjää jo olemassa olevalla nimellä. Salasanan taas täytyy olla 8-60 merkkiä pitkä ja sen tulee sisältää pieniä ja isoja kirjaimia sekä numeroita. Se voi myös sisältää erikoismerkkejä. Salasana syötetään kahteen kertaan, jotta käyttäjä välttyisi kirjoitusvirheiltä. Kaikki edellä mainitut tarkastukset tehdään palvelimen eli back-endin puolella. Tarkastukset ovat Springissä helppo toteuttaa eri annotaatioilla. Tässä työssä käytettiin lähinnä @NotBlank ja @Pattern annotaatioita. Ensimmäinen estää kenttää olemasta tyhjä (tai null) ja toiseen annetaan parametrina regex-ilmaisu. UserController-luokassa rekisteröintiosoitteen POST-pyynnössä otetaan User-olio kiinni ja sen oikeellisuus tarkistetaan @Valid annotaatiolla. Mikäli jokin User-luokan ehto ei täyty, POST-pyyntö ei mene läpi ja käyttäjälle näytetään virheilmoitus. Validointiin liittyvät ovat osa Hibernate Validatoria [https://hibernate.org/validator/]. Hibernate on työssä käytetty (Spring Bootin mukana tuleva) framework, jolla Javan oliot muutetaan tietokantaolioiksi.

Kirjautumisessa rekisteröitymisen kaltaista POST-käsittelijää ei ole tarvinnut tehdä, koska Spring hoitaa sen automaattisesti. Työssä on siis vain luotu kirjautumisnäkymä ja kontrolleriluokassa GET-pyynnön käsittelijä, joka palauttaa kyseisen kirjautumissivun.

3.3 Peli

Varsinaisessa pelinäkymässä on yksi käyttäjän syötettä vaativa input-laatikko. Laatikkoon tulee syöttää yksi kirjain, joka on siis pelaajan arvaus. HTML:ssä estetään ASCII-koodien avulla käyttäjää kirjoittamasta mitään muuta kuin kirjaimia, ja lisäksi syötteen pituus on rajattu yhteen. Näiden rajoitusten lisäksi JavaScriptissä on erillinen funktio tarkastamaan, onko käyttäjän syöte todellakin yksi kirjain. Mikäli pelaaja syöttää jotain muuta kuin yhden kirjaimen, mitään ei tapahdu. Peli toimii siten, että joka arvauksen jälkeen (onnistuneen tai epäonnistuneen) peli pyytää käyttäjää täyttämään arvauskentän. Tämä johtuu siitä, että syöte tyhjennetään aina ennen POST-pyynnön lähetystä ja ”required” kenttä ei päästä pyyntöä lähtemään. Pahimmassa tapauksessa haitallinen käyttäjä voisi hankkiutua eroon näistä front-end tarkistuksista, jolloin POST-pyyntö menisi back-endin puolelle. Back-endin puolella on kuitenkin erillinen luokka käyttäjän syötteelle sekä rekisteröitymisen lailla @Valid annotaatio, joka suorittaa saman yhden kirjaimen tarkistuksen kuin front-end. Tällöin kuitenkin peli alkaa alusta. Toisin sanoen haitallista käyttäjää rangaistaan aloittamalla peli alusta, ja peli toimii kunnolla vain normaalille käyttäjälle.

3.4 OWASP

OWASP (Open Web Application Security Project) on verkon tietoturvallisuuteen keskittynyt organisaatio. Se laatii muun muassa Top 10 -listoja verkon pahimmista haavoittuvuuksista. Tässä luvussa esitellään lyhyesti tämän työn suojautumista osalta vuoden 2017 listan [https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project] pahimmista haavoittuvuuksista.

- **Injection**

Käyttäjän syötteet validoidaan joka kerta Springin annotaatioilla. Ohjelmassa ei suoriteta ohjelmoijan toimesta SQL-kyselyitä, eikä niitä tällöin myöskään yhdistetä suoraan käyttäjän syötteeseen.

- **Broken Authentication**
  Salasanat pakotetaan aikaisemmin mainittujen salasanasääntöjen perusteella olemaan vähintäänkin melko turvallisia. Toisaalta salasanoja ei tarkisteta yleisempien salasanojen listalta, joten yksinkertaisimmillaan salasana ”Pass123!” on mahdollinen. Sovelluksessa ei ole mahdollista salasanan palautukselle, mikä ei ole kovin käyttäjäystävällistä, mutta on toisaalta tietoturvan kannalta parempi. Tämä siksi, että jos varmistus lähetetään esimerkiksi sähköpostiin ja haitallinen käyttäjä on saanut sähköpostin hallintaansa, saa hän varmistuksen myötä myös sovelluksen käyttäjätunnuksen hallintaansa. Sen sijaan sovellukseen voisi lisätä salasanan yleisyyden tarkastelun lisäksi kirjautumiskertojen rajoituksen, mikä estäisi brute-force kirjautumisia. Sessionhallinta Springissä on valmiina siten, että käyttäjä kirjataan ulos tietyn väliajan jälkeen. Tämä perustui kokeelliseen havaintoon, eikä kirjoittanut valitettavasti onnistunut löytämään Springin dokumentaatiosta tästä mainintaa.
- **Sensitive Data Exposure**
  Sovelluksessa arkaluonteista tietoa ovat käytännössä vain käyttäjien salasanat. Niitä käsitellään vain back-endin puolella, jonne käyttäjälle ei ole pääsyä, ja ne tallennettaan salattuina tietokantaan. Salauksessa käytetään vahvaa ja luotettua BCrypt:iä salasanojen tiivisteiden muodostamiseen.
- **XML External Entities (XXE)**
  Ohjelmassa ei käytännössä käsitellä ollenkaan XML:ää, joten XXE-hyökkäystä ei ole mahdollista toteuttaa. Vain ohjelman riippuvuuksista vastaava tiedosto pom.xml on XML-muotoinen, mutta siihen ei pysty vaikuttamaan kuin lähdekoodin hallussapitäjä.
- **Broken Access Control**
  Kirjautumattoman käyttäjän pääsy muille kuin kirjautumis- ja rekisteröitymissivulle on estetty konfigurointitiedostossa. Konfiguroinnissa on käytetty nimenomaan whitelist-tyyppistä ratkaisua, eli mikään osoite ei ole voinut jäädä puuttumaan esimerkiksi inhimillisen virheen takia. Front-endin puolella ei suoriteta mitään tarkastuksia autentikoinnin suhteen, vaan kaikki tarkistukset ovat back-endin puolella, jonne käyttäjä ei pääse käsiksi.
- **Security Misconfiguration**
  Sovellus käyttää uusinta versiota Spring Bootista, joka taas käyttää tuoreimpia versioita komponenteistaan. Spring ja Spring Boot ovat laajasti käytettyjä, joten niiden ajantasaisuudesta pidetään hyvää huolta. Spring Security on käytössä melkein kokonaan oletusasetuksillaan, vain sen configure-metodi on korvattu tätä sovellusta paremmin vastaavaksi. Sen asetukset ovat helppolukuisia ja sallitut osoitteet täytyy erikseen kirjoittaa, joten on edes teoriassa vaikea sallia mitään sellaista, joka aiheuttaisi haavoittuvuuden. Spring Securityn ansiosta virheilmoitukset eivät paljasta liikaa sovelluksesta, ellei applications.properties tiedostoon mene erikseen lisäämään esimerkiksi ”debug=true”.
- **Cross-Site Scripting (XSS)**
  Spring Bootin ansiosta XSS-hyökkäyksille löytyy valmiiksi suojauksia. Http-otsikoista löytyy ”X-Content-Type-Options: nosniff” ja ”X-XSS-Protection: 1; mode=block”. Ensimmäinen estää selaimia nuuskimasta sisältötyyppejä (engl. content sniffing), jota hyväksikäyttäen

käyttäjä pystyisi suorittamaan haitallista koodia [https://docs.spring.io/spring-security/site/docs/5.0.x/reference/html/headers.html]. Jälkimmäinen otsikko taas auttaa reflektoitujen XSS-hyökkäyksien torjumisessa estämällä XSS:n suorituksen sellaisen havaitessa. Dokumentissa aikaisemmin esiteltyjen keinojen avulla kaikki käyttäjän syötteet myös sanitoidaan, mikä estää XSS-hyökkäyksien toteutumista.

- **Using Components with Known Vulnerabilities**
  Spring Bootissa ei käytännössä ole haavoittuvuuksia [https://www.cvedetails.com/vulnerability-list/vendor_id-15183/product_id-43815/Pivotal-Software-Spring-Boot.html]. Tiedetyt haavoittuvuudet ovat olleet vanhemmissa versioissa, joita ei tässä työssä ole käytetty. Muita komponentteja ovat Bootstrap, Jquery ja Thymeleaf, jotka kaikki ovat laajasti käytössä ja eivät sisällä (tunnettuja) haavoittuvuuksia, joita voisi tässä sovelluksessa hyväksikäyttää [https://snyk.io/vuln/npm:bootstrap][https://www.cvedetails.com/vulnerability-list/vendor_id-6538/Jquery.html].

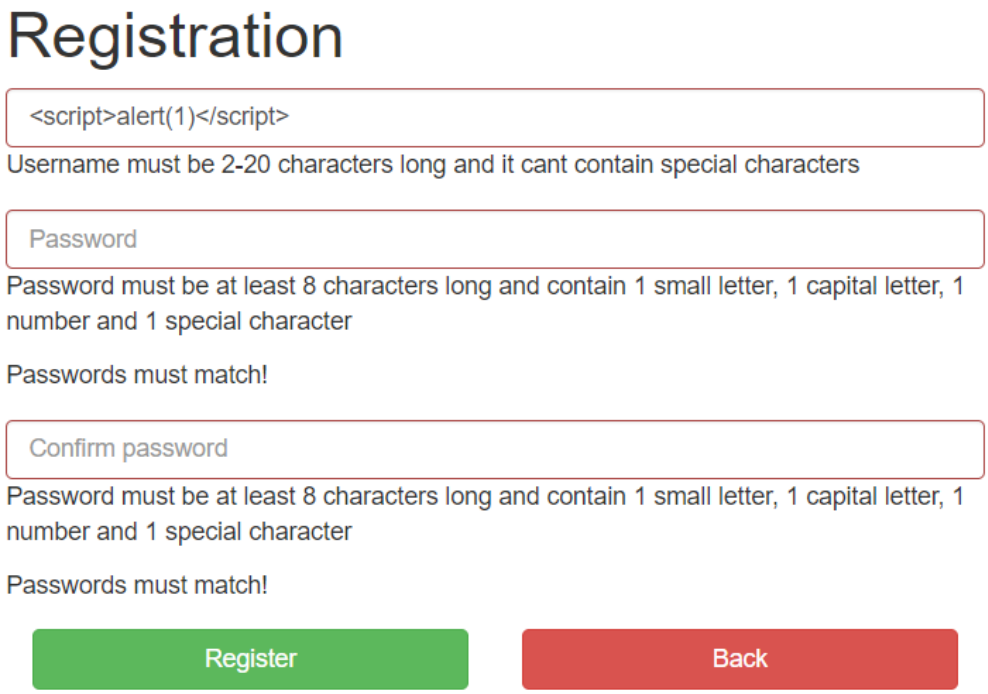
Lisäksi sovelluksessa on suojauduttu CSRF-hyökkäyksiltä. Springin dokumentaation [https://docs.spring.io/spring-security/site/docs/5.0.7.RELEASE/reference/htmlsingle/#csrf] mukaan CSRF-hyökkäyksien torjuntaan vaaditaan kolme vaihetta:

1. Asianmukaisten http-verbien käyttö
2. CSRF-suojauksen konfigurointi
3. CSRF-tokenin sisällyttäminen

Ohjelmassa kaikki tietoja lähettävät pyynnöt on asetettu POST-tyyppisiksi ja vain tietoja hakevat pyynnöt GET-tyyppisiksi, joten ensimmäinen kohta täyttyy. Springin dokumentaation mukaan CSRF-konfigurointi on suoritettu automaattisesti uusimmissa Spring Securityn versioissa, joten toinenkin kohta on hoidettu. Springin dokumentaatio kertoo myös, että jos on käytetty @EnableWebSecurity-annotaatiota, ja käytössä on tuore versio Thymeleafista, CSRF-token sisällytetään automaattisesti. Nämä molemmat kohdat täyttyvät. Konfigurointi-luokassa on @EnableWebSecurity ja jokaisessa html-lomakkeessa <form> tagien sisällä on th:action="@{/url}", joka on Thymeleafin syntaksia, mikä pakottaa CSRF-tokenin sisällyttämisen piilotettuun kenttään.

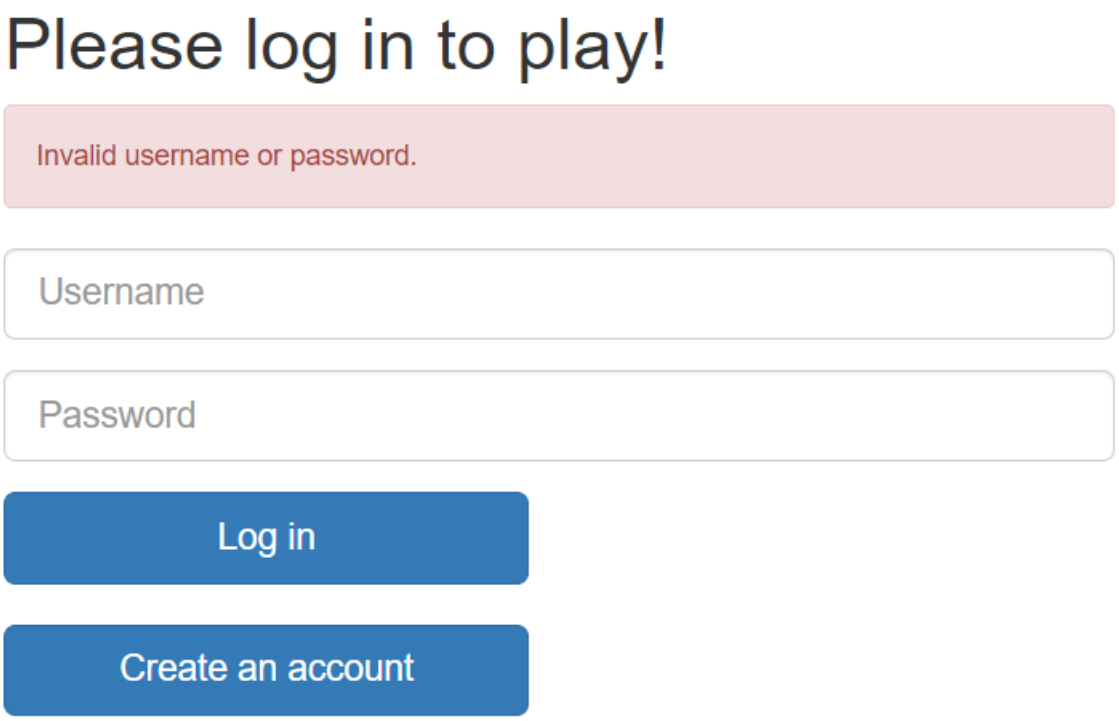
4. Testaus

Sovelluksen tietoturvan testaamiseksi suoritettiin manuaalista testausta. Ensimmäiseksi rekisteröintilomakkeeseen annettiin käyttäjänimeksi ”<script>alert(1)</script>” ja salasanaksi vain ”pass” sekä varmistussalasanaksi ”12345”. Kuvasta 4 ilmenee, että sovellus huomasi virheet ja reagoi asianmukaisesti.



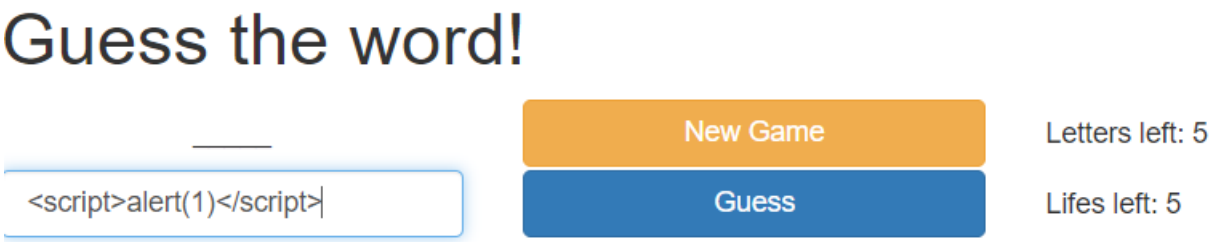
Kuva 4: Rekisteröintilomakkeen virheilmoitukset

Yrittäessä kirjautua väärällä käyttäjänimellä ja/tai salasanalla, sovellus antaa aina kuvan 5 mukaisen virheilmoituksen.



Kuva 5: Epäonnistunut kirjautuminen

Ohjelma ei siis kerro menikö kirjautumisessa käyttäjätunnus, salasana vai molemmat väärin. Tämä vaikeuttaa brute-force tyyppisiä hyökkäyksiä, koska haitallinen käyttäjä ei tiedä onko hän onnistunut arvaamaan esimerkiksi käyttäjätunnuksen oikein. Pelinäkymässä olevat front-end estot estävät kirjoittamasta muuta kuin kirjaimia (isoja tai pieniä) tekstikenttään. Kuitenkin kehittäjän työkaluilla maksimipituuden voi asettaa arvosta yksi (1) arvoon sata (100) ja kiellettyjä merkkejä sisältävän tekstin voi kopioida ja liittää muualta. Tällä tavoin kenttään voidaan liittää skripti ”<script>alert(1)</script>” kuvan 6 osoittamalla tavalla. Lisäksi lomakkeesta on poistettu ”required=true” kenttä.



Kuva 6: Front-end estojen kierto pelinäkymässä

Suorittamalla arvauksen Guess-napista skriptiä ei kuitenkaan suoriteta, vaan pelinäkymä vain päivittyy kuten luvussa 3.3 kerrottiin. Tämä johtuu siis siitä, että pyyntö meni back-endille, joka palauttaa syötteen validoinnin yhteydessä pelinäkymän.