Research Rapport Security

Contents

[Inleiding 1](#_Toc1453386256)

[Onderzoeksvraag en opzet 2](#_Toc876748252)

[Hoofdvraag 3](#_Toc569135978)

[Deelvragen 3](#_Toc1064808691)

[Welke veelvoorkomende beveiligingsproblemen bestaan er voor webapplicaties, zoals OWASP Top 10? 3](#_Toc803189923)

[AO1-Broken Access Control: 4](#_Toc1571749706)

[AO3-Injection 4](#_Toc1482397402)

[AO9-Security Logging and Monitoring Failures 4](#_Toc1812591575)

[Welke maatregelen kan ik nemen om mijn applicatie (Vue en springboot) te beschermen tegen ongeautoriseerde toegang en datalekken? 5](#_Toc289307739)

[Maatregelen: 6](#_Toc1532596829)

[Wat moet er veranderen 6](#_Toc596087042)

[Prototyping 9](#_Toc1438371912)

[Heeft deze prototype geholpen? 16](#_Toc796432620)

[Welke tools kan ik gebruiken om de beveiliging van mijn applicatie te blijven monitoren en verbeteren? 18](#_Toc1575813586)

[Tools 19](#_Toc1882734074)

[Comparison 20](#_Toc1916992734)

[Conclusie 22](#_Toc1412004354)

[References 23](#_Toc2138985420)

# Inleiding

Bij het maken van mijn applicatie wil ik ervoor zorgen dat deze goed beveiligd is tegen veelvoorkomende veiligheidsproblemen. Slechte beveiliging kan leiden tot problemen zoals datalekken, hacking en verlies van vertrouwen bij gebruikers.

In dit onderzoek kijk ik naar welke beveiligingsrisico’s mijn applicatie kan tegenkomen, hoe ernstig deze zijn en wat ik kan doen om deze te voorkomen. Het doel is om praktische maatregelen te vinden waarmee ik mijn applicatie veiliger kan maken.

# Onderzoeksvraag en opzet

## Hoofdvraag

Hoe kan ik mijn applicatie effectief beschermen tegen veelvoorkomende beveiligingsproblemen en ervoor zorgen dat deze voldoet aan moderne beveiligingsstandaarden?

## Deelvragen

* Welke veelvoorkomende beveiligingsproblemen bestaan er voor webapplicaties, zoals OWASP Top 10?
  + Library: Literature Study
    - Met deze methode wil ik voornamelijk kijken wat voor soort bekende beveilings gevaren er zijn. Met focus op wat mij het meeste aanspreekt in relatie met mijn project
* Welke maatregelen kan ik nemen om mijn applicatie (Vue en springboot) te beschermen tegen ongeautoriseerde toegang en datalekken?
  + Library: Literature Study
    - Met deze methode wil ik voornamelijk kijken wat er gedaan kan worden om problemen van de vorige deelvraag te voorkomen
  + Field: Problem Analysis
    - Met deze methode wil ik achter komen wat er aan mijn applicatie voor deze maatregelen veranderd moet worden en hoe hevig mijn huidige applicatie aangepast moet worden
  + Workshop: Prototyping
    - Met deze methode wil het toepassen van de maatregelen die ik ben tegengekomen om erachter te komen of het iets is voor mijn applicatie
  + Lab: A/B Testing
    - Met deze methode wil ik kijken of de toepassing bij het prototype daad werkelijk een verandering heeft gemaakt in veilig dor het te vergelijken met hoe het ervoor was.
* Welke tools kan ik gebruiken om de beveiliging van mijn applicatie te blijven monitoren en verbeteren?
  + Library: Literature Study
    - Met deze methode wil ik kijken naar wat voor tools er zijn die ik kan gebruiken om de veiligheid te testen.
  + Workshop: Multi-Criteria decision making
    - Met deze methode wil ik de tools vergelijken die ik had gevonden zodat ik makkelijker een keuze kan maken welke ik wil gebruiken.

# Welke veelvoorkomende beveiligingsproblemen bestaan er voor webapplicaties, zoals OWASP Top 10?

Ik heb hier voornamelijk gekeken naar beveiligingsproblemen die te zien waren in OWASP Top 10. Hieruit spraken er een paar mij aan als belangrijk, voornamelijk in de context dat het voor mijn individueel project bedoelt is

## AO1-Broken Access Control:

**Wat is het?**

Broken Access Control treedt op wanneer een applicatie niet goed controleert welke acties of gegevens een gebruiker mag zien of uitvoeren. Hierdoor kunnen ongeautoriseerde gebruikers toegang krijgen tot gevoelige gegevens of functies.

**Voorbeelden:**

* Directe toegang tot admin-pagina's door een URL (bijvoorbeeld /admin) zonder correcte autorisatiechecks.
* Manipulatie van objectreferenties, zoals een ID in de URL (/profile?id=123), om toegang te krijgen tot andere gebruikersgegevens.

## AO3-Injection

**Wat is het?**

Injection-aanvallen, zoals SQL-injection, gebeuren wanneer onbeveiligde invoer direct wordt verwerkt door een interpreter (bijvoorbeeld een database of shell), waardoor aanvallers onbedoelde opdrachten kunnen uitvoeren.

**Voorbeelden:**

* SQL-injection: SELECT \* FROM users WHERE username = 'x' OR '1'='1'
* Cross-site Scripting (XSS): Kwaadaardige scripts injecteren in webpagina’s om andere gebruikers te misleiden.

## AO9-Security Logging and Monitoring Failures

**Wat is het?**

Gebrek aan logging of monitoring kan leiden tot vertraagde of gemiste reacties op beveiligingsincidenten. Zonder goede logging blijven aanvallen vaak onopgemerkt.

**Voorbeelden:**

* Niet loggen van mislukte inlogpogingen of verdachte activiteiten.
* Geen alarmen of meldingen bij verdachte activiteiten, zoals meerdere mislukte aanmeldingen van een IP.

# Welke maatregelen kan ik nemen om mijn applicatie (Vue en springboot) te beschermen tegen ongeautoriseerde toegang en datalekken?

Op basis van de OWASP Top 10 kunnen de volgende maatregelen worden genomen om je applicatie te beschermen tegen ongeautoriseerde toegang en datalekken:

## Maatregelen:

**AO1-Broken Access Control:**

* **Rollen en Rechten:** Implementeer rolgebaseerde toegangscontrole (RBAC) om gebruikers alleen toegang te geven tot gegevens en functionaliteiten die relevant zijn voor hun rol.
* **Server-side Controle:** Zorg ervoor dat toegangscontrole altijd op de server wordt uitgevoerd, met veilige authenticatie en autorisatieprocessen.
* **Beveiligde URL’s:** Voorkom ongeautoriseerde toegang tot gevoelige pagina’s door server-side controles voor URL-manipulatie.

**AO3-Injection:**

* **Prepared Statements:** Gebruik voorbereid statements om SQL-injectie te voorkomen, en implementeer beveiligingsmaatregelen tegen Cross-Site Scripting (XSS)-aanvallen.
* **Validatie en Sanitisatie:** Valideer en saniteer gebruikersinvoer zowel aan de client-side als server-side om risico’s van injecties te minimaliseren.

**AO9-Security Logging and Monitoring Failures:**

* **Logging van Beveiligingsgebeurtenissen:** Log mislukte inlogpogingen en verdachte activiteiten om beveiligingsincidenten te kunnen volgen.
* **Real-time Monitoring:** Gebruik monitoringtools om verdachte activiteiten in real-time te detecteren en waarschuwingen in te stellen voor verdachte gebeurtenissen.

## Wat moet er veranderen

Nu heb ik wel neergelegd wat de maatregeling kunnen zijn voor de 3 delen van de OWASP Top 10 waar ik op focus, maar het is van belang om te weten wat ik hiervoor zou moeten doen en hoe groot de aanpassing zou zijn aan mijn huidige applicatie.  
  
**AO1 - Broken Access Control**

**Probleem:** Ongeautoriseerde gebruikers kunnen toegang krijgen tot gevoelige gegevens of functionaliteiten in mijn applicatie.

* **Huidige situatie:** Mijn Spring Boot applicatie maakt gebruik van DGS GraphQL voor data-uitwisseling, maar er wordt op het moment nog niet gekeken of gebruikers toegang hebben om sommige acties te doen. Dit betekent dat gebruikers toegang kunnen krijgen tot belangrijke functies door het manipuleren van URL’s of verzoeken.
* **Aanpassingen:**
  + **Rolgebaseerde Toegangscontrole (RBAC):** Ik zou bijvoorbeeld gebruik kunnen maken van Spring Security om rolgebaseerde toegangscontrole (RBAC) er mee toe te voegen. Dit betekent dat ik gebruikersrollen kan maken voor in mijn applicatie die ervoor moet zorgen dat alleen geautoriseerde gebruikers toegang hebben tot bepaalde GraphQL-queries en mutaties.
  + **Server-side Controle:** Toegangscontrole zal volledig server-side moeten gebeuren. Ik zal ervoor moeten zorgen dat gevoelige data niet toegankelijk is via manipulatie van URL’s of via niet-geautoriseerde API-aanroepen. Dit houdt in dat ik server-side beveiligingsregels moet implementeren in Spring Boot en ervoor moet zorgen dat de beveiliging van GraphQL-endpoints goed is ingesteld.
  + **Frontend Toegang:** In Vue zal ik route guards moeten toevoegen die controleren of een gebruiker de juiste rol heeft om bepaalde pagina’s te bezoeken. Alleen geautoriseerde gebruikers mogen toegang krijgen tot beschermde onderdelen van de applicatie.
* **Grootte van de aanpassing:** Dit zou een matige aanpassing zijn, aangezien ik Spring Security moet integreren en de toegangscontrole zowel aan de server- als client-kant moet implementeren. Daarnaast zou ik dan ook mogelijk mij Integration en E2E test moeten verbouwen zodat die weer werken.

**AO3 - Injection**

**Probleem:** Ongecontroleerde invoer kan leiden tot SQL-injectie, XSS-aanvallen of andere injectie-aanvallen in mijn applicatie.

* **Huidige situatie:** Mijn applicatie maakt gebruik van Spring JPA voor database-interacties en Spring Data MongoDB voor documentopslag, maar de invoervalidatie en sanitatie zijn momenteel niet strikt afgedwongen. Dit zou kunnen leiden tot kwetsbaarheden zoals SQL-injecties of XSS-aanvallen.
* **Aanpassingen:**
  + **Prepared Statements en Parameter Binding:** Aangezien ik Spring JPA gebruik, worden SQL-injecties grotendeels beschermd door het gebruik van JPA-repositories, maar dit helpt maar tot zover en ik moet er zeker van zijn dat alle query’s goed worden geparameteriseerd. In MongoDB moet ik ook zorgdragen voor het veilig verwerken van query’s zonder dat gebruikersinvoer direct in de database-query’s wordt ingevoegd.
  + **Validatie en Sanitisatie:** Ik moet zowel client-side (Vue) als server-side (Spring Boot) invoer valideren en saniteren. Voor Vue kan ik formuliervalidatie implementeren voordat gegevens naar de server worden gestuurd. Aan de serverzijde moet ik gebruik maken van validatie-annotaties zoals **@NotNull** en **@Size** om invoer te controleren. Ook moet ik ervoor zorgen dat alle gebruikersinvoer wordt gesanitiseerd om XSS-aanvallen te voorkomen.
  + **Sanitatie van Bestanden:** Voor de bestandshandling moet ik ervoor zorgen dat bestandstypen goed worden gevalideerd en dat bestanden op een veilige manier worden opgeslagen en gepresenteerd, zonder risico op schadelijke inhoud.
* **Grootte van de aanpassing:** Deze wijziging zou een matige aanpassing zijn. Het vereist het uitbreiden van validatie- en sanitisatielogica zowel aan de client- als serverzijde. De impact is vooral aan de backend, maar het heeft ook invloed op de frontend, vooral in het geval van formuliervalidatie.

**AO9 - Security Logging and Monitoring Failures**

**Probleem:** Gebrek aan logging en monitoring maakt het moeilijk om beveiligingsincidenten tijdig op te merken in mijn applicatie.

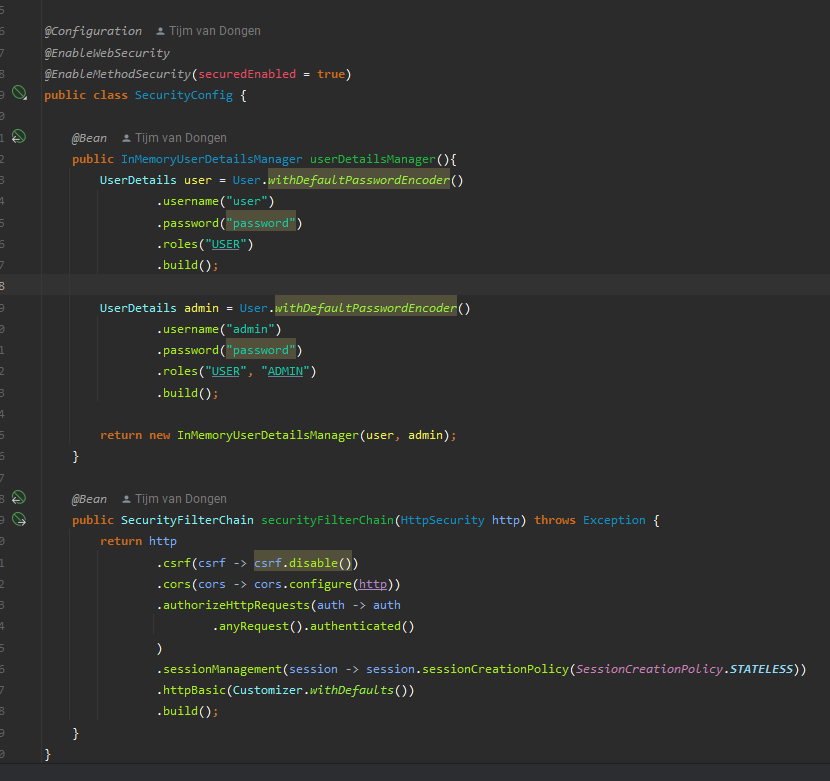
* **Huidige situatie:** Mijn applicatie heeft momenteel geen uitgebreide beveiligingslogging en monitoring. Het is daardoor moeilijk om verdachte activiteiten zoals mislukte inlogpogingen of ongeautoriseerde toegang op te merken.
* **Aanpassingen:**
  + **Logging van Beveiligingsgebeurtenissen:** Ik moet logging toevoegen voor beveiligingsgebeurtenissen, zoals mislukte inlogpogingen en verdachte toegangspatronen. Dit kan ik doen door SLF4J en Logback te integreren in mijn Spring Boot-applicatie en logberichten te genereren voor beveiligingsrelevante gebeurtenissen.
  + **Realtijd Monitoring:** Door gebruik te maken van tools zoals Spring Boot Actuator, Prometheus en Grafana kan ik real-time monitoring van de applicatie instellen. Ik kan waarschuwingen instellen voor verdachte activiteiten, zoals meerdere mislukte inlogpogingen, en kritieke fouten in het systeem die wijzen op beveiligingsproblemen.
  + **Geavanceerde Waarschuwingen:** In aanvulling op de monitoring, kan ik ook een systeem voor waarschuwingen implementeren om snel te reageren op potentiële aanvallen of beveiligingsincidenten.
* **Grootte van de aanpassing:** Deze wijziging zou een grote aanpassing zijn. Het vereist de integratie van loggingsystemen en monitoringtools, wat extra configuratie en werk aan de serverkant met zich meebrengt.

## Prototyping

**AO1 - Broken Access Control**

Ik heb voor Broken Access Control voornamelijk gebruik gemaakt van Spring Security en hiermee een Rolebased authentication toegevoegd. Dit is op het moment nog relatief simpel maar dit is genoeg om alvast een goed idee te krijgen voor mijn applicatie en of het werkt.

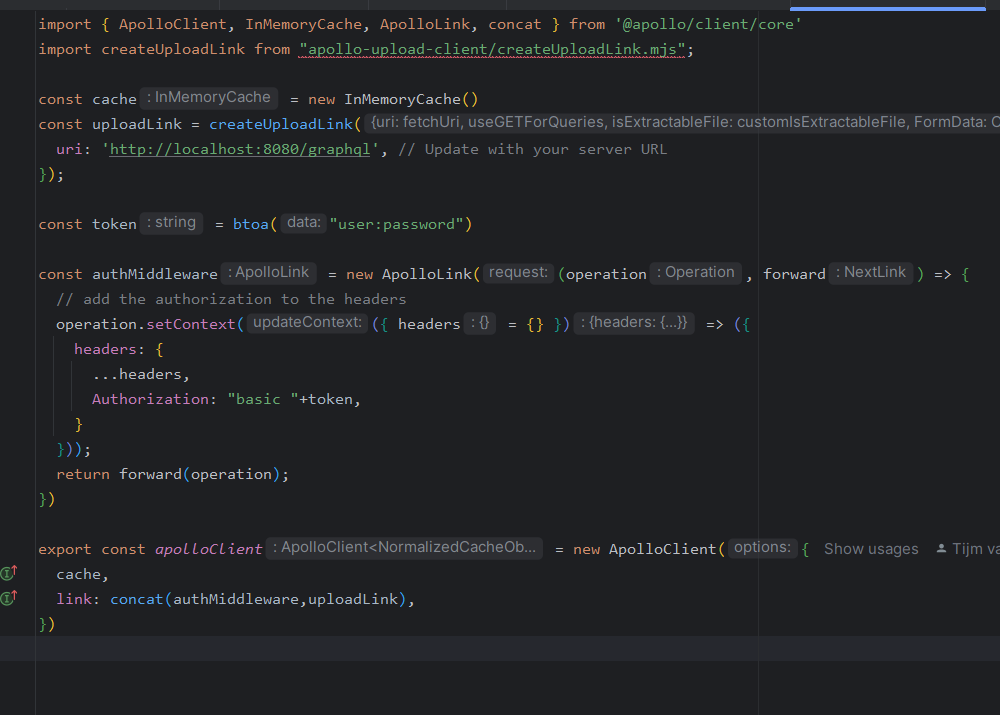
Hier onder zie je hoe ik mijn Config van spring security had op gezet, hierbij kun je ook zien dat ik 2 vaste rollen heb toegevoeg die een gebruiker gebruik van kan maken en welke rechten die hebben. In de SecurityFilterChain wordt er aan gegeven hoe de applicatie omgaat met het handelen van Security en Outside API calls.



Hier onder word aan gegeven hoe ik kan welke rol een gebruiker moet hebben om er gebruik van te maken. Hier bij gebruik ik @Secured deze hard check of de Authorization die word meegegeven de Rol van User heeft.



Hier onder is de configuratie van apollo om een de Authorization token mee te geven aan de GraphQL calls.



**AO3 - Injection**

Ik heb hiervoor in mijn applicatie voornamelijk gericht op sanitisatie, zo heb ik gebruik gemaakt van jakarta.validations.constraints om ervoor te zorgen dat fields in entiteiten zich houden niet iets anders kunnen zijn dan verwacht.



Ook maak ik gebruik van Tika (een dependancy van Apache) om een bestand die worden geupload te check of die daad werkelijk wel het verwachte bestand type (en niet een verborgen exe/bat) is en of ze voldoen aan de eisen van het project (bestaands grote voornamelijk).



Daarnaast heb ik ook nog voor de front-end nog extra checks toegevoegd die er voor zouden zorgen dat er in de front het niet snel fout zou gaan met het toevoegen van SQL injections en verkeerde bestanden.  

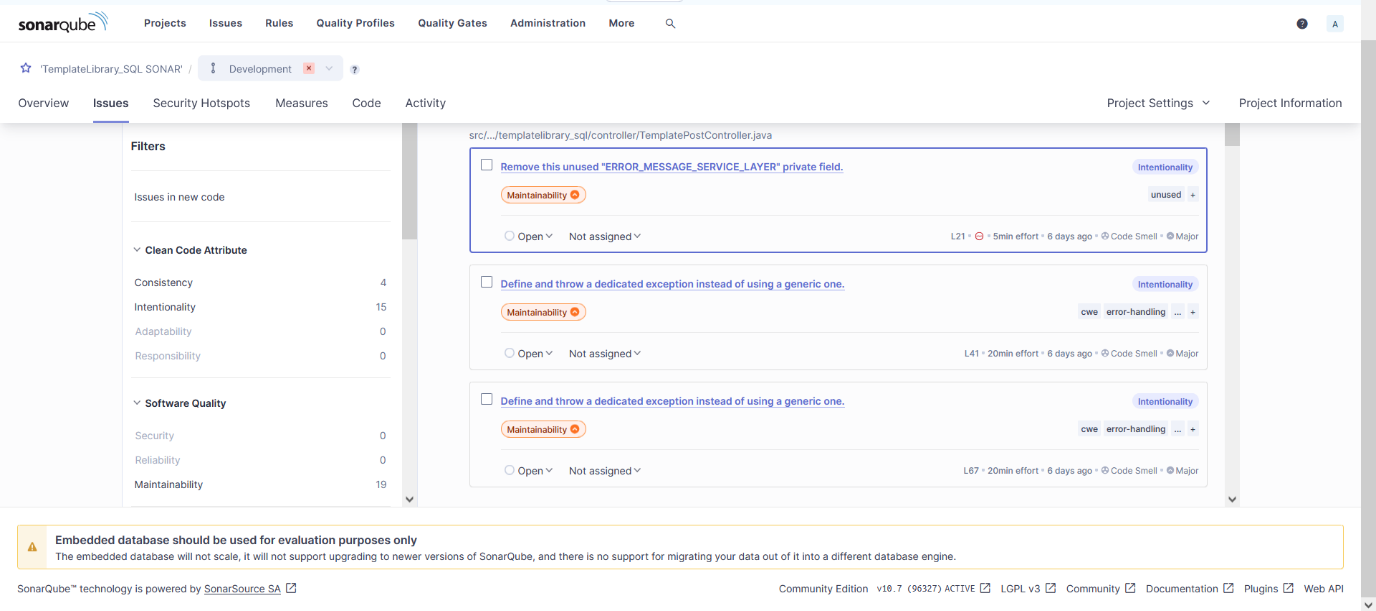

**AO9 - Security Logging and Monitoring Failures**

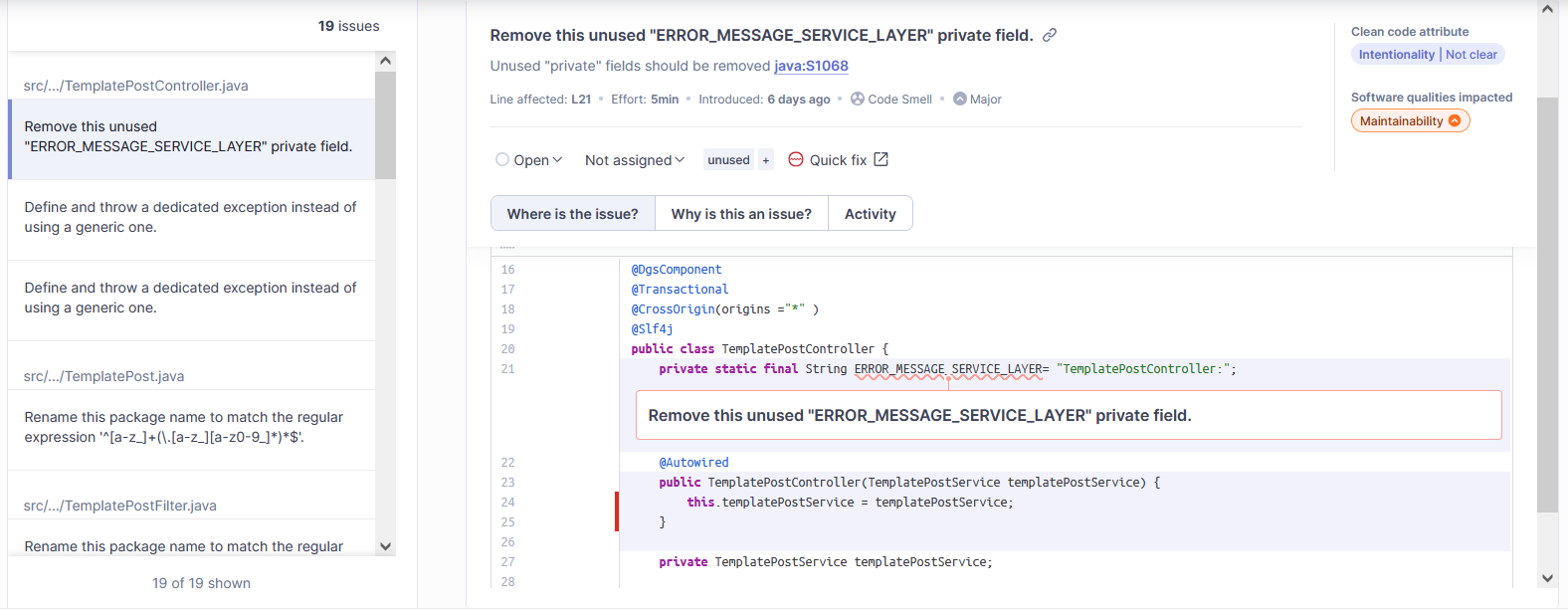
Ik heb hiervoor in mijn applicatie gericht op het invoegen van monitor tools en logging, hiervan heb ik voor logging gebruik gemaakt van SLF4j om mijn logs toe te voegen aan mijn applicatie. Daarbij heb ik ze voornamelijk ingevoegd om de try catch error handeling. Hier onder zie je een voorbeeld van een method in my Service layer die gebruik maakt van Logging, hierin word het gebruikt om de error te weer geven in de applicatie en waar het zich plaat vond.



Voor het monitoren heb ik voornamelijk gebruik gemaakt van SonarQube om static code analysis te doen (deze keuze was gemaakt op basis van de Rubix hier onder en het document Devlog Sonarqube).

Hier onder zie een voorbeeld van de sonarqube omgeving die ik heb aangemaakt voor mijn SQL backend systeem, waarvan je kan zie dat die aangeeft welke problemen die heeft gedetecteerd.



Hierbij geeft SonarQube aan waar precies de fout zit (Dus welk bestand en welke lijn) en hoe belangrijk het probleem is.

Zo had het eerder nog een probleem gedetecteerd waarbij het niet veilig vond hoe ik gebruik maakte van @Autowired. Hierbij legt het ook uit waarom het een fout is en hoe je zou kunnen fixen om het compliant te maken.

Daarnaast is het ook nog te kopelen met de SonarLint Plugin, het voordeel hieraan is dat je dan de error’s die sonarqube had gezien ook kunt zien in je IDE. Hierdoor kun je veel sneller problemen oplossen aangezien je kan programmeren on the go zonder de SonarQube applicatie zelf te bekijken wat tijd kan kosten.

## Heeft deze prototype geholpen?

**AO1 - Broken Access Control**

**Zonder Spring Security:**

Voordat ik Spring Security implementeerde, was het relatief eenvoudig om toegang te krijgen tot de API van mijn applicatie zodra deze werd blootgesteld. Er was geen beveiliging aanwezig, wat betekent dat onbevoegden zonder toestemming gegevens konden aanpassen. Dit bood grote risico’s voor de integriteit van de applicatie. De toegang tot de data was volledig onbeveiligd, wat het makkelijk maakte om misbruik te maken van de applicatie.

**Met Spring Security:**

Na de implementatie van Spring Security realiseerde ik me meteen hoe de beveiliging van de applicatie aanzienlijk kon worden verbeterd. Zelfs met slechts een eenvoudig rollensysteem en basis encryptie voor authenticatie was de beveiliging al veel robuuster. Het gebruik van JWT voor het toekennen van tokens aan gebruikers zou de veiligheid verder verhogen, waardoor onbevoegden geen toegang meer hadden tot gevoelige gegevens en de applicatie beter beschermd was tegen misbruik.

**AO3 - Injection**

**Zonder Sanitization:**

Voordat ik sanitization-methoden implementeerde, was het relatief eenvoudig om malware of SQL-injectie toe te voegen aan mijn applicatie. Hoewel Spring JPA zelf enige bescherming biedt tegen SQL-injectie, is deze bescherming soms niet voldoende voor complexere queries. Dit betekende dat er nog steeds kwetsbaarheden aanwezig waren die door aanvallers misbruikt konden worden om schadelijke code in de applicatie te injecteren.

**Met Sanitization:**

Na het toevoegen van sanitization-methoden heb ik ervoor kunnen zorgen dat het veel moeilijker werd voor hackers om SQL-injectie uit te voeren. Door te controleren op verdachte input en ervoor te zorgen dat alleen entiteiten die voldoen aan de vereisten worden aangemaakt, werd de applicatie beter beschermd. Daarnaast wordt door de integratie van Tika gecontroleerd of het geüploade bestand daadwerkelijk het type bestand is dat we willen toestaan. Op de website zelf wordt ook geforceerd om te voldoen aan de eisen van entiteiten, zoals het juiste bestandstype en het verbod op bepaalde karakters in tekstvelden.

**AO9 - Security Logging and Monitoring Failures**

**Zonder Logging en Monitoring:**

Voordat ik gebruik maakte van logging en monitoring, had ik geen tools om te zien of mijn applicatie mogelijk problemen had die ik nog niet had opgemerkt, of wanneer er een probleem was waar ik niet wist wat de oorzaak was. Dit maakte het vaak moeilijk om snel de oorzaak van een probleem te achterhalen, waardoor ik veel tijd kwijt was aan het oplossen van issues die veel sneller opgelost hadden kunnen worden.

**Met Logging en Monitoring:**

Na het implementeren van logging en monitoring-tools heb ik ervoor gezorgd dat ik precies kan zien waar iets fout gaat. Met behulp van logging kan ik direct achterhalen waar een specifieke fout zich voordoet, wat me helpt om gerichter te zoeken naar de oorzaak. Ook kan ik hiermee bepaalde fouttypes uitsluiten, waardoor ik sneller de juiste fout kan vinden te midden van andere errors. Daarnaast zorgen monitoring-tools zoals SonarQube ervoor dat ik fouten of problemen detecteer die ik anders misschien niet had opgemerkt. SonarQube controleert of mijn code voldoet aan standaarden en of deze op een veilige en efficiënte manier is geschreven. Het biedt ook gedetailleerde uitleg over waarom iets fout is en wat ik eraan kan doen.

# Welke tools kan ik gebruiken om de beveiliging van mijn applicatie te blijven monitoren en verbeteren?

## Tools

Hoewel we deze verbeteringen hebben toegevoegd is het wel belangrijk om erachter te komen of ze daad werkelijk werken en of de aanpassingen niet een te grootte impact hebben gehad op de applicatie.

**AO1 - Broken Access Control**

* **OWASP ZAP**: Een open-source dynamische applicatiebeveiligingstesttool (DAST) voor het detecteren van kwetsbaarheden zoals ongeautoriseerde toegang via URL-manipulatie en andere toegangsmethoden.
* **Burp Suite**: Een populaire tool voor webbeveiligingstests die dynamisch controleert op beveiligingslekken, waaronder toegangscorrectheid.
* **Fortify**: Voert zowel statische als dynamische tests uit om toegangszaken zoals ongeautoriseerde toegang of privilege escalation te detecteren.

**AO3 - Injection**

* **OWASP ZAP**: Detecteert injectie-aanvallen zoals SQL-injectie, XSS en andere vormen van injecties die de applicatie kunnen compromitteren.
* **Burp Suite**: Detecteert en exploiteert kwetsbaarheden zoals SQL-injectie en XSS-aanvallen door middel van de Intruder- en Scanner-functies.
* **SQLMap**: Een krachtige open-source tool voor het automatiseren van het testen op SQL-injecties en het uitvoeren van geavanceerde SQL-injectie-aanvallen.
* **Fortify**: Voert statische en dynamische beveiligingstests uit die injecties zoals SQL-injectie of andere kwetsbaarheden kunnen opsporen.

**AO9 - Security Logging and Monitoring Failures**

* **SonarQube**: Voert statische code-analyse uit om te helpen bij het verbeteren van de beveiligingslogica in je code, zoals logging, foutafhandeling en beveiligingsmonitoring.
* **Prometheus en Grafana**: Voor het real-time monitoren van je applicatie. Prometheus verzamelt tijdreeksdata en Grafana visualiseert deze data in dashboards om verdachte activiteiten en beveiligingsincidenten snel te identificeren.
* **ELK Stack (Elasticsearch, Logstash, Kibana)**: Voor het verzamelen, verwerken en visualiseren van loggegevens in real-time, waardoor je verdachte activiteiten kunt detecteren.
* **Splunk**: Een krachtige tool voor logbeheer en monitoring die gegevens verzamelt uit verschillende bronnen, analyseert en waarschuwingen genereert bij verdachte activiteit.
* **Graylog**: Een andere logmanagementtool die helpt bij het verzamelen, visualiseren en analyseren van logdata in real-time om beveiligingsincidenten op te sporen.

## Comparison

Hier voor heb ik nog een Rubix op gesteld om voor alle 3 de AO’s om de verschillen tussen de tools duidelijk te maken, ik heb ze hier bij voornamelijk vegeleken op de volgende aspecten: Soort Testen, of die gratis is, Of het self hosted is en nog extra opmerking over ze.

**AO1 - Broken Access Control**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tool** | **Type Testen** | **Gratis** | **Self-Hosted** | **Opmerkingen** |
| **OWASP ZAP** | Dynamische applicatiebeveiligingstests (DAST), detectie van toegangscorrectheidsproblemen | Ja | Ja | Testen op misconfiguratie van toegangspunten en URL-manipulatie |
| **Burp Suite** | Dynamische beveiligingstests, toegangscorrectheid en privileges escalatie | Beperkt (gratis versie) | Nee | Heeft een betaalde versie voor uitgebreidere functionaliteit |
| **Fortify** | Statische en dynamische beveiligingstests, detectie van toegangscorrectheidsproblemen | Nee | Ja | Geavanceerde enterprise-level oplossing |

**AO3 – Injection**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tool** | **Type Testen** | **Gratis** | **Self-Hosted** | **Opmerkingen** |
| **OWASP ZAP** | DAST, detectie van SQL-injecties, XSS en andere injecties | Ja | Ja | Detecteert veel soorten injectie-aanvallen |
| **Burp Suite** | DAST, detectie van SQL-injectie en XSS-aanvallen | Beperkt (gratis versie) | Nee | Uitgebreide functionaliteit in betaalde versie |
| **SQLMap** | Automatische SQL-injectie tests, exploitatie | Ja | Ja | Specifiek voor SQL-injectie tests |
| **Fortify** | Statische en dynamische tests, detectie van injecties | Nee | Ja | Detecteert injecties zoals SQL-injecties |

**AO9 - Security Logging and Monitoring Failures**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Type Testen** | **Gratis** | **Self-Hosted** |
| **SonarQube** | Statische code-analyse, detectie van ontbrekende logging of slechte beveiliging | Ja | Ja |
| **Prometheus** | Real-time metrics verzamelen, monitoring van activiteiten | Ja | Ja |
| **Grafana** | Visualisatie van logs en metrics, real-time dashboards | Ja | Ja |
| **ELK Stack (Elasticsearch, Logstash, Kibana)** | Logging en loganalyse, real-time eventmonitoring | Ja | Ja |
| **Splunk** | Logbeheer en monitoring, real-time detectie van incidenten | Nee | Ja |
| **Graylog** | Logbeheer en analyse, real-time monitoring | Ja | Ja |

# Conclusie

Na het onderzoeken en prototypen van verschillende methoden om mijn applicatie veiliger te maken, heb ik belangrijke stappen gezet om deze te beschermen tegen veelvoorkomende beveiligingsproblemen. Door het implementeren van Logging, Monitoring, Sanitization, en Spring Security is de beveiliging van mijn applicatie aanzienlijk verbeterd.

Logging en Monitoring stellen mij in staat om problemen sneller te detecteren en de oorzaken ervan gericht te analyseren. Logging zorgt ervoor dat fouten specifiek gelokaliseerd kunnen worden en biedt inzicht in de werking van de applicatie. Monitoring-tools zoals SonarQube helpen mij daarnaast om verborgen fouten en kwetsbaarheden op te sporen, en ze controleren of mijn code voldoet aan moderne standaarden voor veiligheid en kwaliteit. Dit draagt bij aan een robuustere en toekomstbestendige applicatie.

Met Sanitization heb ik ervoor gezorgd dat invoer van gebruikers veiliger wordt verwerkt, waardoor het risico op injection-aanvallen, zoals SQL-injectie of malware, sterk is verminderd. Dit wordt bereikt door invoervalidatie en door te controleren of geüploade bestanden voldoen aan de gestelde eisen. Daarnaast zorgt dit ervoor dat alleen correcte en betrouwbare data wordt verwerkt binnen mijn applicatie.

Door het toepassen van Spring Security zijn belangrijke toegangscontroles toegevoegd, zoals een eenvoudig rollensysteem en basis authenticatie. Dit maakt het moeilijker voor onbevoegde gebruikers om toegang te krijgen tot gevoelige gegevens of functionaliteiten. Het gebruik van JWT (JSON Web Tokens) zou de authenticatie nog verder versterken en de algehele beveiliging naar een hoger niveau tillen.

Ik denk dat mijn applicatie door deze implementaties al een stuk meer beveiligd is geworden dan dat die eerst was. Met deze maatregelen is mijn applicatie beter beschermd tegen veelvoorkomende beveiligingsproblemen, zoals beschreven in de OWASP Top 10, en voldoet het aan moderne beveiligingsstandaarden. Hoewel dit nog niet helemaal voldoende is om echt veilig te concluderen denk ik wel dat het een goede basis legt om op verder uit te breiden, maar voor nu is het op een goede plek.

# References

Fortify. (n.d.). From Fortify: https://www.microfocus.com/en-us/products/fortify-static-code-analysis-sast/

Grafana. (n.d.). From Grafana: https://grafana.com/

owasp. (n.d.). From owasp top 10: https://owasp.org/Top10/

owasp. (n.d.). From owasp AO1 Broken Acces Control: https://owasp.org/Top10/A01\_2021-Broken\_Access\_Control/

owasp. (n.d.). From owasp AO3 Injection: https://owasp.org/Top10/A03\_2021-Injection/

owasp. (n.d.). From owasp AO9 Security Logging and Monitoring Failures: https://owasp.org/Top10/A09\_2021-Security\_Logging\_and\_Monitoring\_Failures/

Prometheus. (n.d.). From Prometheus: https://prometheus.io/

qos. (n.d.). From Logback: https://logback.qos.ch/

qos. (n.d.). From SLF4J: https://www.slf4j.org/

Sonarqube. (n.d.). From Sonarqube: https://www.sonarqube.org/

SQLMap. (n.d.). From SQLMap: http://sqlmap.org/

Stack, E. (n.d.). From ELK Stack: https://www.elastic.co/elk-stack

Suite, B. (n.d.). From Burp Suite: https://portswigger.net/burp

ZAP, O. (n.d.). From OWASP ZAP: https://www.zaproxy.org/