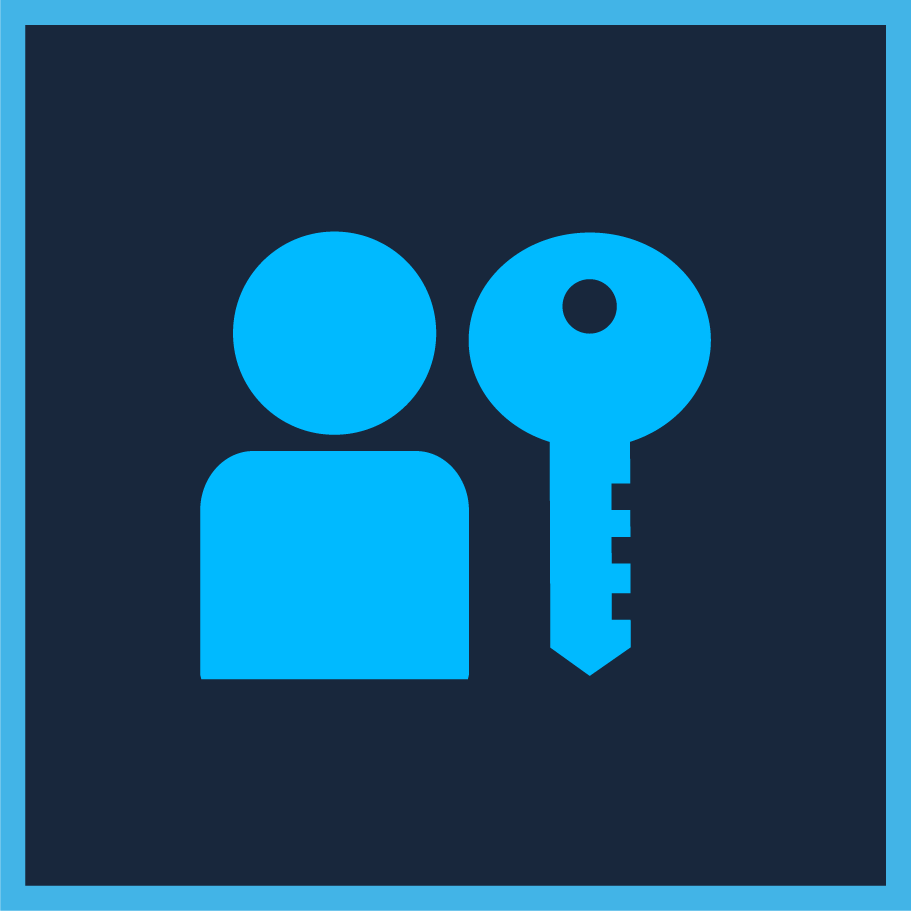
Authenticatie onderzoek



|  |
| --- |
| **Datum: 22-04-2021** |
| **Versie: 0.4** |
| **Bestandsnaam:** Authenticatie onderzoek |
| **Auteurs:**  Roel Lucassen, Luuk Ebenau, Youri Saman |

# Versiebeheer

|  |  |  |
| --- | --- | --- |
| Versie | Wat | Datum |
| 0.1 | Inhoudsopgave, vooronderzoek en begonnen met methodes | 22/04/2021 |
| 0.2 | Methodes afgemaakt | 24/04/2021 |
| 0.3 | Document design aanpassen. | 31/05/2021 |
| 0.4 | Grammatica aanpassen | 02/06/2021 |

# Inhoudsopgave

[Versiebeheer 2](#_Toc73662290)

[Inhoudsopgave 3](#_Toc73662291)

[Inleiding 4](#_Toc73662292)

[Vooronderzoek 5](#_Toc73662293)

[Probleemanalyse 5](#_Toc73662294)

[Hoofdvraag 5](#_Toc73662295)

[Deelvragen 5](#_Toc73662296)

[Onderzoek 6](#_Toc73662297)

[Methodes 6](#_Toc73662298)

[Wat zijn de eisen vanuit de stakeholder(s) waar de authenticatie aan moet voldoen? 6](#_Toc73662299)

[Welke authenticatie methodes en technologieën zijn er beschikbaar? 7](#_Toc73662300)

[HTTP Basic Auth 7](#_Toc73662301)

[HTTP Digest Authentication 7](#_Toc73662302)

[Sessie authenticatie 7](#_Toc73662303)

[Token authenticatie 9](#_Toc73662304)

[One time passwords (OTP) 11](#_Toc73662305)

[Welke authenticatie methodes zijn het meest geschikt voor onze applicatie? 11](#_Toc73662306)

[Hoe kan authenticatie op een veilige manier uitgevoerd worden? 13](#_Toc73662307)

[Gebruik van zwakke tokens 13](#_Toc73662308)

[Een open connectie 13](#_Toc73662309)

[Niet managen van de vervaldatum/tijd van een token 13](#_Toc73662310)

[Meerdere logins op hetzelfde moment 14](#_Toc73662311)

[Eigen tokens maken 14](#_Toc73662312)

[Hoe laten we alle interne systemen gebruik maken van hetzelfde authenticatie systeem? 15](#_Toc73662313)

[Mogelijkheden 16](#_Toc73662314)

[Conclusie 16](#_Toc73662315)

# Inleiding

In dit onderzoek hebben we onderzocht wat de beste authenticatiemethode is binnen de context van het Mindef project.

Vooronderzoek

In dit hoofdstuk verkennen we het concrete probleem waar dit onderzoek uit voort komt en kijken we op welke manier die sub-vragen hierbij bij kunnen dragen.

## Probleemanalyse

De probleemanalyse is gedaan door middel van de vijf W’s: **w**ie, **w**at, **w**aar, **w**anneer, en **w**aarom.

**Wie ondervindt het probleem?** De gebruikers van het systeem.

**Wat is het probleem?**  Authenticatie moet op een veilige manier gebeuren, zodat een tegenstander het systeem niet in kan komen.

**Waar bevindt het probleem zich?** Het probleem bevindt zich in de applicatie.

**Wanneer komt dit probleem op?** Wanneer de applicatie in het veld gebruikt gaat worden.

**Waarom bestaat dit probleem?** Het probleem ontstaat wanneer er nog geen veilige authenticatie is gemaakt voor de applicatie.

## Hoofdvraag

*Hoe kan authenticatie zo opgesteld worden dat alle services, vanuit één service geautoriseerd, op een veilige manier met elkaar communiceren?*

## Deelvragen

* Welke authenticatie methodes en technologieën zijn er beschikbaar?
* Wat zijn de eisen vanuit de stakeholder(s) waar de authenticatie aan moet voldoen?
* Hoe kan authenticatie op een veilige manier uitgevoerd worden?
* Hoe laten we alle interne systemen gebruik maken van hetzelfde authenticatie systeem?

# Onderzoek

## Methodes

|  |  |
| --- | --- |
| Methode | Motivering |
| [Design pattern research](http://ictresearchmethods.nl/Design_pattern_research) | Deze methode helpt met het opzoeken van goede design patterns voor de authenticatie |
| [Community research](http://ictresearchmethods.nl/Community_research) | Mensen hebben al eerder dit soort vragen onderzocht en het kan handig zijn om dit door te lezen. |
| [Best good and bad practices](http://ictresearchmethods.nl/Best_good_and_bad_practices) | Kijken wat bij andere mensen wel of niet heeft gewerkt om het probleem op te lossen |
| [Stakeholder analysis](http://ictresearchmethods.nl/Stakeholder_analysis) | Om deze vraag goed te beantwoorden willen we de wensen van de stakeholders rondom authenticatie onderzoeken |
| [Security test](http://ictresearchmethods.nl/Security_test) | Door middel van een security test kunnen we in kaart brengen naar welke kwetsbaarheden we moeten kijken bij het valideren van de security van de applicatie. |
| [IT architecture sketching](http://ictresearchmethods.nl/IT_architecture_sketching) | Een Architecture sketch helpt met het inzichtelijk maken van de componenten van het systeem, wat inzicht geeft in eventuele beperkingen om rekening mee te houden. |

## 

## Wat zijn de eisen vanuit de stakeholder(s) waar de authenticatie aan moet voldoen?

Uit eerdere gespreken zijn een al een aantal eisen opgesteld voor de authenticatie binnen de applicatie:

* Het device van een gebruiker moet automatisch herkend worden waarna een gebruiker in kan loggen in het systeem.
* De gebruiker moet kunnen inloggen met een voor hem gegenereerde pincode.

Om de eisen van de stakeholders verder in kaart te brengen is nog een vraag gesteld:

* Om een veilige authenticatie te behouden denken wij aan het instellen van een bepaalde tijd waarin een interactie gedaan moet worden met het dashboard, indien dit niet wordt gedaan moet de gebruiker opnieuw inloggen met zijn pincode. Hoe denk jij hierover?
  + **Antwoord**: Authenticatie moet plaatsvinden onder operationele omstandigheden, men kan bijvoorbeeld verwikkeld zijn in een vuurgevecht. Dan is het niet handig om met pincodes te werken. Kan het bijvoorbeeld met een vingerafdruk of iets anders wat heel makkelijk is om te doen onder stressvolle omstandigheden?

Hieruit kan geconcludeerd worden dat gebruikers moeten kunnen inloggen met een pincode, maar uiteindelijk ook met een extra factor. Denk hierbij aan een biometrische factor, zoals een vingerafdruk of een iris, of een mechanische factor, zoals een NFC-chip.

## Welke authenticatie methodes en technologieën zijn er beschikbaar?

### HTTP Basic Auth

Basic Auth, die is ingebouwd in het HTTP-protocol, is de meest eenvoudige vorm van verificatie. Hiermee worden inloggegevens verzonden in de verzoekheaders bij elk verzoek.

Gebruikersnamen en wachtwoorden zijn niet versleuteld. In plaats daarvan worden de gebruikersnaam en het wachtwoord samengevoegd met behulp van een: symbool om een enkele string te vormen: “gebruikersnaam: wachtwoord”. Deze tekst is gecodeerd met base64.

Voordelen

* Omdat er niet veel bewerkingen plaatsvinden, kan authenticatie met deze methode sneller zijn.
* Makkelijk te implementeren.
* Ondersteund door alle grote browsers.

Nadelen

* Base64 is niet hetzelfde als versleuteling. Het is gewoon een andere manier om gegevens weer te geven. De met base64 gecodeerde tekenreeks kan eenvoudig worden gedecodeerd omdat deze in platte tekst wordt verzonden. Daarom is HTTPS/SSL absoluut essentieel.
* Bij elk verzoek moeten legitimatiegegevens worden verzonden.
* De gebruiker kan alleen worden uitgelogd door de inloggegevens te herschrijven met een ongeldige.

### HTTP Digest Authentication

HTTP Digest Authentication (of Digest Access Authenticatie) is een veiligere vorm van HTTP Basic Auth. Het belangrijkste verschil is dat het wachtwoord wordt verzonden in MD5-gehashte vorm in plaats van in ‘plain tekst', dus het is veiliger dan Basic Auth.

Voordelen

* Veiliger dan basisverificatie omdat het wachtwoord niet in platte tekst wordt verzonden.
* Makkelijk te implementeren.
* Ondersteund door alle grote browsers.

Nadelen

* Bij elk verzoek moeten legitimatiegegevens worden meegestuurd.
* De gebruiker kan alleen worden uitgelogd door de inloggegevens te herschrijven met een ongeldige.
* In vergelijking met basisverificatie zijn wachtwoorden minder veilig op de server omdat bcrypt niet kan worden gebruikt.
* Kwetsbaar voor man-in-the-middle-aanvallen.

### Sessie authenticatie

Met sessie-gebaseerde authenticatie (of sessiecookie-authenticatie of cookie-gebaseerde authenticatie) wordt de status van de gebruiker opgeslagen op de server.

De gebruiker hoeft niet bij elk verzoek een gebruikersnaam of wachtwoord op te geven. In plaats daarvan valideert de server na het inloggen de inloggegevens. Indien geldig, genereert het een sessie, slaat het op in een sessieopslag en stuurt vervolgens de sessie-ID terug naar de browser. De browser slaat de sessie-ID op als een cookie, die elke keer dat er een verzoek naar de server wordt verzonden, wordt mee verzonden.

Op sessie gebaseerde verificatie is stateful. Elke keer dat een cliënt de server opvraagt, moet de server de sessie in het geheugen lokaliseren om de sessie-ID terug te koppelen aan de geassocieerde gebruiker.

**Voordelen[[1]](#footnote-2)**

* **Performance:** Snellere volgende aanmeldingen, omdat de inloggegevens niet vereist zijn.
* **Usability**: verbeterde gebruikerservaring.
* **Easily implemented**: Vrij eenvoudig te implementeren. Veel frameworks (zoals Django) bieden deze functie out-of-the-box.

**Nadelen**

* **Stateful**: De server houdt elke sessie aan de serverzijde bij. De sessieopslag, die wordt gebruikt voor het opslaan van gebruikerssessie-informatie, moet worden gedeeld door meerdere services om authenticatie mogelijk te maken. Daarom werkt het niet goed voor RESTful-services, aangezien REST een stateless protocol is.
* **Unnecessary cookies:** Bij elk verzoek worden cookies meegestuurd, ook als er geen authenticatie voor nodig is
* **CSRF**: Kwetsbaar voor CSRF-aanvallen
* **Bad Scaling**: werkt niet goed samen met microservices

Het voordeel van sessie authenticatie is dat de gebruikers kan bijhouden. Dit is erg aantrekkelijk voor het onderzoek om dat wij veel macht over de gebruiker wil hebben mocht het verkeerd gebruikt worden.

### Token authenticatie

Deze methode gebruikt tokens om gebruikers te authentiseren in plaats van cookies. De gebruiker verifieert met geldige inloggegevens en de server retourneert een ondertekend token. Dit token kan worden gebruikt voor volgende verzoeken. Een voorbeeld van token gebaseerde authenticatie is JSON Web Token**[[2]](#footnote-3).**

#### JSON Web Token

JSON Web Token (JWT) is een compacte en veilige manier om informatie over het netwerk uit te wisselen. JWT helpt de integriteit en authenticiteit van de informatie te behouden omdat deze digitaal is ondertekend met een geheim of openbaar sleutelpaar met RSA of ECDSA.

Een belangrijk ding om in gedachten te houden over JWT is dat het een ondertekend token is en niet een geencrypte token. Dat houdt in dat JWT de informatie niet kan verbergen, ook al kan het wel de integriteit van de JW Token verifiëren.

HTTP is een stateless protocol, wat betekent dat een nieuw request niets onthoudt van het vorige request. Dus voor elk verzoek zou moeten worden ingelogd om geuthoriseerd te worden. Dit kost een hoop werk. Normaal zouden hiervoor sessions gebruikt kunnen worden, om de authenticatie in de server op te slaan. Dit geeft echter schaalbaarheid en prestatie issues als er veel gebruikers binnen de applicatie aanwezig zijn.

De (JWT) maakt geen gebruik van sessions en voorkomt daarmee de bovenstaande problemen. Wanneer inloggegevens naar de server worden gestuurd zal de server een JSON Web Token terugsturen.

**Voordelen[[3]](#footnote-4)**

* **Compactheid:** JSON is minder uitgebreid dan XML en daarom neemt het, wanneer het is gecodeerd, minder ruimte in beslag, waardoor JWT compacter wordt dan SAML.
* **Geen sessie nodig:** de JWT kan alle benodigde informatie over de gebruiker bevatten en daarom is het niet nodig om een sessieobject op de server te onderhouden, waardoor servergeheugen wordt bespaard.
* **Ingebouwde vervaldatum**: het JWT heeft claims die kunnen worden gebruikt om het een vervaldatum/-tijd toe te wijzen. Daarom kan het token vanzelf ongeldig worden na de vervalperiode.
* **Geen cookies nodig**: het token kan worden opgeslagen in de localStorage, indexDB of een native store. Dit biedt bescherming tegen CORS- en CSRF-aanvallen.

**Nadelen**

* **Compromised Secret Key:** het beste en het ergste van JWT is dat het op slechts één sleutel vertrouwt. Bedenk dat de sleutel is gelekt door een onzorgvuldig ontwikkelaar/beheerder, het hele systeem is gecompromitteerd wordt.
* **Cannot manage client from the server:** Doordat de jwt token lokaal staat is het niet mogelijk om de lokale jwt token te verwijderen vanaf de server.
* **Crypto-algo can be deprecated** : JWT vertrouwt volledig op het ondertekeningsalgoritme. Nu, hoewel het niet vaak voorkomt, zijn in het verleden veel coderings-/ondertekeningsalgoritmen verouderd.
* **Data Overhead:** de grootte van het JWT-token zal groter zijn dan die van een normale sessietoken. Hoe meer gegevens er toevoegt aan het JWT-token, hoe meer data het meestuurt. Onthoud dat voor elk verzoek het token erin nodig is voor verzoekverificatie. Stel dat een 1 KB JWT-token betekent dat elk verzoek 1 KB upload over het hoofd heeft, wat erg slecht is in gevallen van een lage snelheid van het internet.

Het voordeel van deze tokens is dat dit zeer simpel te implementeren is, en toch redelijk veilig is. Het nadeel van een JW Token is dat een server geen beheersing heeft over deze token.

Naast JW Tokens zijn er ook nog andere standaarden[[4]](#footnote-5), die verschillende voor- en nadelen bieden. Een hiervan is de API KEY authenticatie.

#### API KEY

API Key is een van de simpelste token standaarden voor autorisatie binnen een service. Dit is een unieke code waarmee toegang verkregen wordt tot een bepaalde applicatie. Eventueel kunnen deze tokens zo gemaakt worden dat ze maar eenmalig te gebruiken zijn.

**Voordelen**

* Simpel te implementeren

**Nadelen**

* Niet user specifiek
* Tokens kunnen niet worden ingenomen
* Alleen geschikt voor interne backend authenticatie, niet voor front to backend authenticatie.

Verder is er ook nog het OAuth2 protocol, dit is een gestandaardiseerd protocol welke op de JW Tokens is gebaseerd. Het gestandaardiseerde bied zowel voor- als nadelen die overwegen moet worden.

#### OAUTH2 token

Auth2 is een veel gebruikte standaard die sinds een aantal jaar erg populair is. Dit is op dit moment de meest gedefinieerde standaard en wordt door vrijwel alle grote webapplicaties gebruikt.

Een van de hoofdpunten die oauth2 anders maakt dan de rest is dat de aangeraden oauth2 flow de gebruiker redirect naar een externe login pagina, wat als voordeel biedt dat de authenticatie van elke gebruik makende website goed geïmplementeerd is. Het nadeel is dat dit soms omslachtig kan voelen voor non-web applicaties.

**Voordelen**

* User specifieke tokens
* De autorisatie server kan tokens innemen omdat ze bij elke aanvraag gecontroleerd worden
* Gecentraliseerde authenticatie
* Veilig
* De loginpagina's zijn extern gehost en kunnen dus door meerdere applicaties gebruikt worden
* Login sessies worden bijgehouden

**Nadelen**

* Recommended flows require a redirect to another domain for logging in
* Requires a dependency on a seperate authorization server
* More complex te set up
* Minder geschikt voor non- web applicaties

### 

### One time passwords (OTP)

Eenmalige wachtwoorden (OTP's) worden vaak gebruikt als bevestiging voor authenticatie. OTP's zijn willekeurig gegenereerde codes die kunnen worden gebruikt om te verifiëren of de gebruiker is wie hij beweert te zijn. Het wordt vaak gebruikt nadat een gebruiker geverifieerd is door twee-factor-authenticatie.

Om OTP te gebruiken, moet een vertrouwd systeem aanwezig zijn. Dit vertrouwde systeem kan een geverifieerd e-mailadres of mobiel nummer zijn.

Moderne OTP's zijn stateless. Ze kunnen op meerdere manieren worden geverifieerd. Hoewel er een paar verschillende soorten OTP's zijn, is Time-based OTP's (TOTP's) misschien wel het meest voorkomende type. Eenmaal gegenereerd, vervallen ze na een bepaalde tijd.

Omdat een extra beveiligingslaag wordt gecreëerd, worden OTP's aanbevolen voor apps met zeer gevoelige gegevens, zoals internetbankieren en andere financiële diensten.

**Voordelen**

* Voegt een extra beschermingslaag toe.
* Geen gevaar dat een gestolen wachtwoord kan worden gebruikt voor meerdere sites of services die ook OTP's implementeren.

**Nadelen**

* U moet de seed opslaan dat wordt gebruikt voor het genereren van OTP's.
* OTP-agents zoals Google Authenticator zijn moeilijk opnieuw in te stellen indien de herstelde code is kwijtgeraakt.
* Er doen zich problemen voor wanneer het vertrouwde apparaat niet beschikbaar is (lege batterij, netwerkfout, enz.). Daarom is meestal een back-upapparaat vereist dat een extra aanvalsvector toevoegt.

### Welke authenticatie methodes zijn het meest geschikt voor onze applicatie?

Aangezien tussen de applicaties zo min mogelijk onderlinge afhankelijkheden moeten zijn, is een gedecentraliseerde authenticatie methode zoals JSON Web Tokens geschikter dan OAuth2; waar alles gecontroleerd wordt door een centrale service. Verder is er vanuit de opdrachtgever behoefte aan embedded authenticatie methodes zoals bijvoorbeeld NFC, vingerafdruk scanner of irisscan. Aangezien de veiligste flow van OAuth2 gebruikt maakt naar een externe login pagina die geen toegang heeft tot embedded hardware, is dit geen geschikte optie.

Verder is er behoefte om de ingelogde gebruiker te kunnen authentiseren, en daarom is API KEY authenticatie geen geschikte methode.

Verder is er nog de OTP (One Time Passwords). Deze methode kan gebruikt worden wanneer een persoon eenmalig toegang moet geven worden tot het systeem, en hierdoor kan deze gebruiker d.m.v. deze token een JW Token genereren waarmee hij toegang krijgt tot (een gedeelte van) de applicatie.

Concluderend hieruit is voor onze applicatie JW Token Authenticatie het meer geschikt, met een optie om een JW Token te genereren d.m.v. een OTP Code.

## Hoe kan authenticatie op een veilige manier uitgevoerd worden?

Bij de vorig vraag is eruit gekomen dat wij tokens gaan gebruiken voor het regelen van de authenticatie en autorisatie. Voor deze methode is onderzoek gedaan naar de meest voorkomende fouten die gemaakt worden en hoe deze te voorkomen.

### Gebruik van zwakke tokens

In een artikel van Artūras Aleksandrovas[[5]](#footnote-6), werd beschreven dat het belangrijk is om tokens zo sterk mogelijk en niet te raden te maken. Ze moeten zo gegenereerd worden dat een aanvaller niet een token kan gokken of extrapoleren.

**Een paar standaardregels voor het genereren voor sterke tokens:**

* Gebruik een extreem grote set van mogelijke waardes.
* Werk met een sterke bron van pseudoniemen, om ervoor te zorgen dat er een onmogelijk raadbare waardes zijn.
* Maak de tokens lang genoeg (tenminste 16 bytes).

**Voordelen**

* Brute force duurt erg lang om een authenticatie token te raden.
* De meeste populaire frameworks hebben al een methode die sterke tokens genereren, het wiel hoeft dus niet opnieuw uitgevonden te worden.

**Nadelen**

* Soms kan het genereren van sterke tokens een paar seconden duren. Hierdoor zal een service wel trager worden wat kan leiden tot frustratie van de gebruikers.
* De frameworks die gebruikt worden om de tokens te genereren moeten wel up to date blijven.
* Token namen worden vaak opengesteld, hierdoor kunnen aanvallers weten welke technologie er gebruikt wordt en kan de aanvaller hier gebruik van maken.

### Een open connectie

In het artikel schreef Artūras ook dat het slim is om tokens binnen een gesloten connectie (HTTPS) te gebruiken. Via deze veilige verbinding kan een buitenstaander de tokens niet inzien. Een veelgemaakte fout is dat sommige pagina's niet via de veilige https-verbinding gaan. Op deze pagina's kan een misbruiker de tokens zonder enig probleem inzien, dit moet voorkomen worden. Daarom moet op elke pagina een veilige gesloten connectie zijn.

### Niet managen van de vervaldatum/tijd van een token

Een ander probleem dat Artūras benoemd in zijn artikel is dat sommige developers niet de vervaldatum instellen van de tokens.

Er zijn twee categorieën van technieken die de vervaltijd regelen:

* Automatisch
  + De applicatie zou de gebruikersactiviteit bij moeten houden. Als er binnen een bepaalde tijd (bv. 15 minuten) geen activiteit is zou de token moeten vervallen. De gebruiker zou dan een melding, als een soort waarschuwing, moeten krijgen zodat hij de keuze krijgt om de website verder te gebruiken.
  + Als de gebruiker constant actief is zou de token automatisch moeten hernieuwd worden. Hierdoor zal een aanvaller maar een korte tijd schade kunnen doen binnen de applicatie.
* Manueel
  + Als een gebruiker uitlogt moet de token verwijderd worden. Hierdoor wordt voorkomen dat dezelfde token later nog gebruikt kan worden.
  + Het zou mogelijk zijn om te traceren dat een browser wordt afgesloten. Tijdens dit “event” zou de token verwijderd kunnen worden. Dit in combinatie met een automatische vervaldatum zou een goede extra beveiliging zijn.

### Meerdere logins op hetzelfde moment

In sommige gevallen zal het goed beveiligen van een applicatie gepaard komen met een minder comfortabele ervaring voor gebruikers. Artūras schrijft dat het slim is om na te denken over het parallel draaien van authenticatie voor gebruikers. In sommige gevallen is het namelijk niet nodig en zou het dus extra veiligheid met zich meebrengen dat als een gebruiker is ingelogd op een bepaalde plek, het niet mogelijk is om op een andere plek. Eén van de gebruikers moet dus gekickt worden.

Er zijn een paar punten om over na te denken:

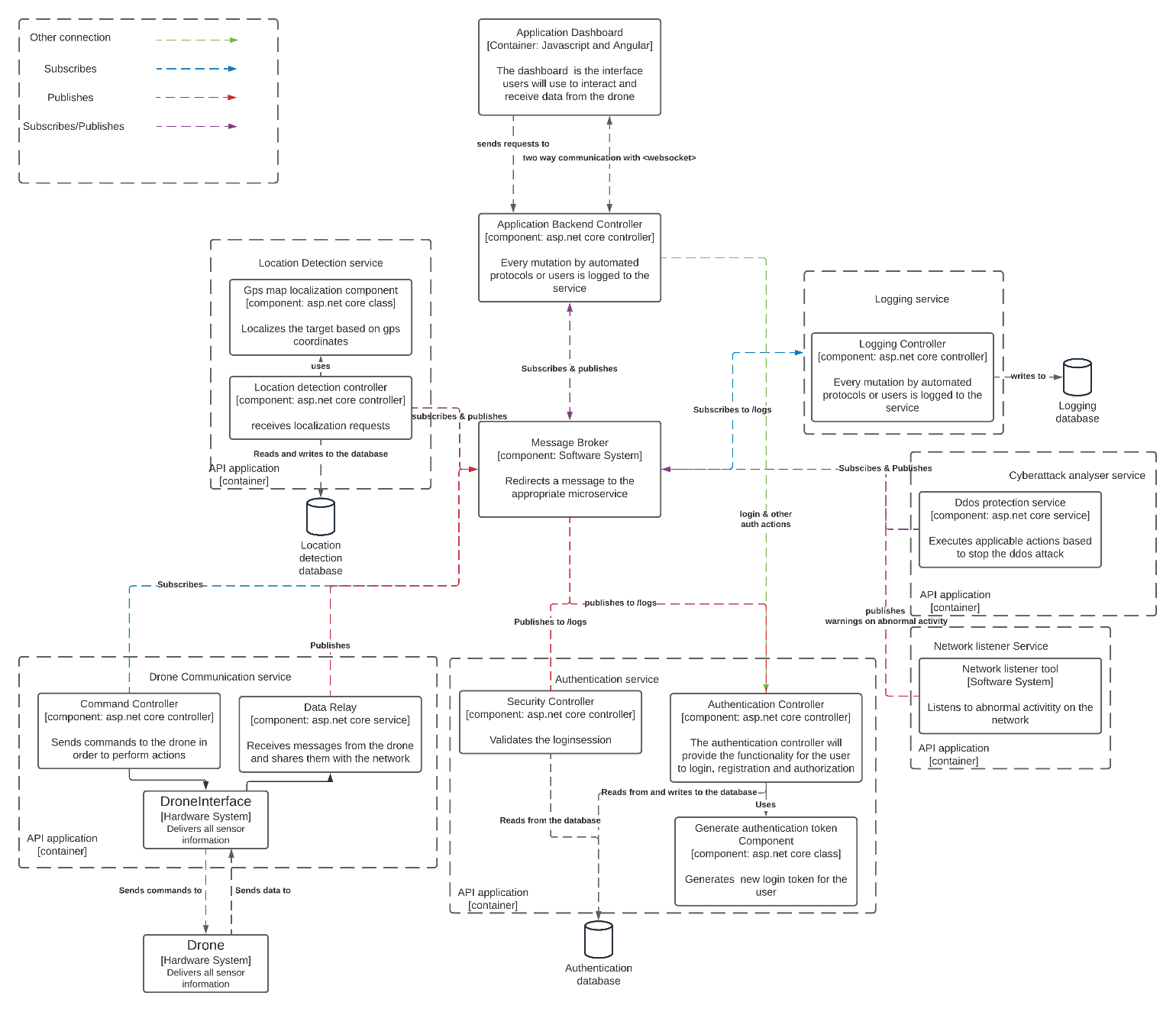
* Als een token wordt gestolen en de aanvaller is ingelogd op zijn pc, kan een admin (en ook de originele gebruiker) ten alle tijden verwittigd worden en hierop snel handelen.
* Het wordt bijna onmogelijk om een actieve sessie over te nemen (zonder dit op te merken).

### Eigen tokens maken

Voor sommige programmeurs is het een uitdaging om tokens te implementeren in hun applicatie. Ze willen alles zelf doen en schrijven hun eigen algoritmes voor het genereren en hashen van de tokens. Hierin kunnen fouten voorvallen of zijn de algoritmes te simpel waardoor ze makkelijk te kraken zijn. Het is dus slim om een erkende library te gebruiken die tokens genereerd, zoals ook in dit forum wordt aangeraden door Daniel Szpisjak[[6]](#footnote-7), als reactie op de vragen die een gebruiker stelt. Deze zijn doorgaans veel moeilijker te kraken.

## Hoe laten we alle interne systemen gebruik maken van hetzelfde authenticatie systeem?

De authenticatie tokens worden allemaal gecentraliseerd, met een secret, in de authenticatieservice gegenereerd. Bij het gebruik van een microservice architectuur is het de bedoeling dat elke service onafhankelijk van de autheticatieservice functioneert. Hiervoor wordt een package gedistribueerd waarmee dezelfde secret wordt gebruikt om de geldigheid van de token te controleren. Zo is iedere service zelf verantwoordelijk voor het verifiëren van de loginsessie.

Zoals in bovenstaande figuur te zien is, hebben de microservices door deze architectuur geen directe afhankelijkheid van de authenticatieservice. In het andere scenario waarin elke service de token door de authenticatieservice laat controleren, heeft daardoor elke service een afhankelijkheid (en dus een lijntje) naar de authenticatieservice.

### Mogelijkheden

OAuth[[7]](#footnote-8), is een protocol dat een token zoals JWT kan gebruiken om authenticatie te regelen binnen een applicatie.

OAuth2 biedt de mogelijkheid tot 2 flows[[8]](#footnote-9), die eventueel in aanmerking zouden komen binnen deze architectuur: de authorization code flow en de implicit flow.

* In de authorization code flow beschikt de client over een client secret, en kan hiermee vanuit de applicatie zelf autoriseren. Aangezien de secret geheimgehouden moet worden kan deze niet rechtstreeks gebruikt worden in een website. Deze flow kan wel gebruikt worden wanneer een backend als een tussenmedium gebruikt wordt. Er zal niet direct gebruik worden gemaakt van het OAuth protocol maar de methode zal nagebootst worden in de microservice architectuur.
* In de implicit flow wordt de user geredirect naar een externe loginpagina, en is daarom ideaal voor websites en ook de veiligste methode. In het dashboard wordt gebruik gemaakt van een interne login pagina, alles binnen dit project moet binnen het interne netwerk blijven voor extra veiligheid. Deze interne login kan gebeuren met een inlogcode, maar vanuit de wensen van de stakeholders zal er later ook een andere factor worden gebruikt voor de authenticatie. Denk hierbij aan een biometrische factor (vingerafdruk, iris) of een mechanische factor (NFC). Hierdoor is de impliciet flow niet bruikbaar volgens de gewenste requirements.

# Conclusie

De conclusie die we uit dit onderzoek kunnen trekken is dat JW Tokens gebruikt worden om de authenticatie te regelen; deze tokens dienen geconfigureerd te worden volgens de benoemde veiligheidsmaatregelen binnen dit onderzoek. Dit houdt in dat er sterke tokens gegenereerd moeten worden door een veilige library waardoor de token veilig is. Deze tokens zullen daarna in elke service worden gecheckt door het gebruik van dezelfde library om de token te valideren op geldigheid.

1. Shaji, A. (2020, 23 december). Web Authentication Methods Compared. TestDriven.io. https://testdriven.io/blog/web-authentication-methods/ [↑](#footnote-ref-2)
2. Bhuwad, A. (2020, 8 december). Best Guide to JSON Web Token (JWT) | Latest Guide | The Startup https://medium.com/swlh/all-you-need-to-know-about-json-web-token-jwt-8a5d6131157f [↑](#footnote-ref-3)
3. Golwalkar, R. (2020, 4 maart). Pros and cons in using JWT (JSON Web Tokens) - Rahul Golwalkar https://medium.com/@rahulgolwalkar/pros-and-cons-in-using-jwt-json-web-tokens-196ac6d41fb4 [↑](#footnote-ref-4)
4. Zapier. (2019, March 16) API Keys vs OAuth Tokens vs JSON Web Tokens - The Zapier Engineering Blog | Zapier. Retrieved March 18, 2021, from <https://zapier.com/engineering/apikey-oauth-jwt/> [↑](#footnote-ref-5)
5. Artūras Aleksandrovas. (2019, May 17) Five risks and tips when securing user authentication tokens · Devbridge. Retrieved March 19, 2021, from <https://www.devbridge.com/articles/five-risks-and-tips-when-securing-user-authentication-tokens/>. [↑](#footnote-ref-6)
6. Daniel Szpisjak. (2021) rest - Vulnerabilities of using a token based authentication such as this - Information Security Stack Exchange. Retrieved March 24, 2021, from https://security.stackexchange.com/questions/122410/vulnerabilities-of-using-a-token-based-authentication-such-as-this [↑](#footnote-ref-7)
7. Aaron Parecki. (2021) OAuth 2.0 — OAuth. Retrieved March 25, 2021, from <https://oauth.net/2/jwt/> [↑](#footnote-ref-8)
8. Auth0. (2021) OAuth 2.0 Authorization Framework. Retrieved March 25, 2021, from https://auth0.com/docs/protocols/protocol-oauth2 [↑](#footnote-ref-9)