

Het onderzoeksonderwerp van deze graduaatsproef richt zich op de ontwikkeling van een authenticatiesysteem voor webapplicaties waarbij gebruikers de optie hebben om in te loggen via hun Google-account of een persoonlijk e-mailadres. Speciale aandacht wordt besteed aan het integreren van een e-mailverificatieproces om de veiligheidsmaatregelen te versterken. Dit authenticatieproces wordt onderzocht binnen de context van gebruikersvriendelijkheid en beveiliging.

Ferah Talha
Janssen Patrick
Kaya Talha
Brisk Studio, Jan Habexlaan 17/17, 3600 Genk

Woord vooraf

“Deze graduaatsproef is tot stand gekomen door de waardevolle input en de aanmoediging van mijn werkplekcoach Talha Kaya. Hij suggereerde niet alleen het onderwerp voor dit project maar voorzag mij ook van de essentiële designs die de basis vormden voor de interface van deze proef. Zijn steun was onmisbaar en daarvoor ben ik hem zeer dankbaar.

Tijdens mijn stage bij Brisk studio heb ik bovendien praktische ervaring opgedaan wat heel belangrijk was voor het voltooien van deze proef. De kennis en vaardigheden die ik daar heb opgebouwd hebben direct bijgedragen aan zowel de theoretische als de praktische aspecten van mijn werk.

Ik wil ook mijn PXL-coach Patrick Janssen bedanken voor zijn begeleiding en de adviezen die hij me gaf tijdens dit traject. Zijn begeleiding was heel waardevol bij het behalen van mijn leerdoelen.

Deze proef weerspiegelt mijn inzet om een effectieve bijdrage te leveren aan de verbetering van gebruikersvriendelijke en veilige authenticatiemethoden voor digitale applicaties. Dit alles zou niet mogelijk zijn geweest zonder de ondersteuning en inspiratie van de mensen om mij heen.”

Talha Ferah

Inhoudsopgave

Inhoud

Woord vooraf	2
Inhoudsopgave	3
1 Bedrijfsvoorstelling	4
2 Projectvraag, onderzoeksacties en resultaten.....	5
2.1 Situering probleemstelling.....	5
2.2 Projectvraag en deelvragen	7
2.2.1 Projectvraag	7
2.2.2 Deelvragen	7
2.3 Onderzoekacties.....	14
2.4 Verzamelde Resultaten	15
2.4.1 Waarom is SSO belangrijk voor gebruikers?	15
2.4.2 Waarom kiezen voor Google SSO?.....	16
2.4.2.1 Hoe werkt Google SSO?	17
2.4.3 Hashing Algorithmes	18
2.4.4 Email verificatie methodes	19
2.4.4.1 Voorkomen dat emails in de spamfolder terechtkomen	23
2.4.5 Server-side gegevensbeheer met Node.js	21
3 Conclusies en aanbevelingen	22
3.1 Conclusies	22
3.2 Aanbevelingen	23
4 Persoonlijke reflecties en kritische kanttekeningen	23
5 Referentielijst.....	24
6 Bijlagen	25

1 Bedrijfsvoorstelling

Brisk Studio gelegen in de Benelux staat bekend als een vooruitstrevende, onafhankelijke dienstverlener op het gebied van UI/UX-design met een solide trackrecord van meer dan tien jaar in het versterken van de digitale aanwezigheid van bedrijven. Met een focus op MKB's, startups en scale-ups biedt Brisk Studio een brede keuze aan diensten waaronder het ontwikkelen van websites en webshops, grafisch ontwerp en uitgebreide UI/UX-oplossingen die niet alleen visueel aantrekkelijk zijn maar ook optimaal gebruiksgemak bieden.

Bij Brisk geloven ze in het bouwen van langdurige klantrelaties door diensten te leveren die aansluiten bij de unieke behoeften van elk bedrijf. Dit doen ze door nauw samen te werken met hun klanten tijdens elk project van conceptontwikkeling tot en met de uitvoering om ervoor te zorgen dat elk ontwerp de bedrijfsidentiteit versterkt en de gebruikerservaring optimaliseert.

Deze aanpak heeft Brisk Studio geholpen om een indrukwekkend portfolio en een sterke reputatie voor betrouwbaarheid en kwaliteit in de digitale designwereld op te bouwen. Ze blijven zich inzetten voor de nieuwste technologieën en beste praktijken wat hen helpt om consistent aan de verwachtingen van hun klanten te voldoen en deze vaak te overtreffen.



2 Projectvraag, onderzoeksacties en resultaten

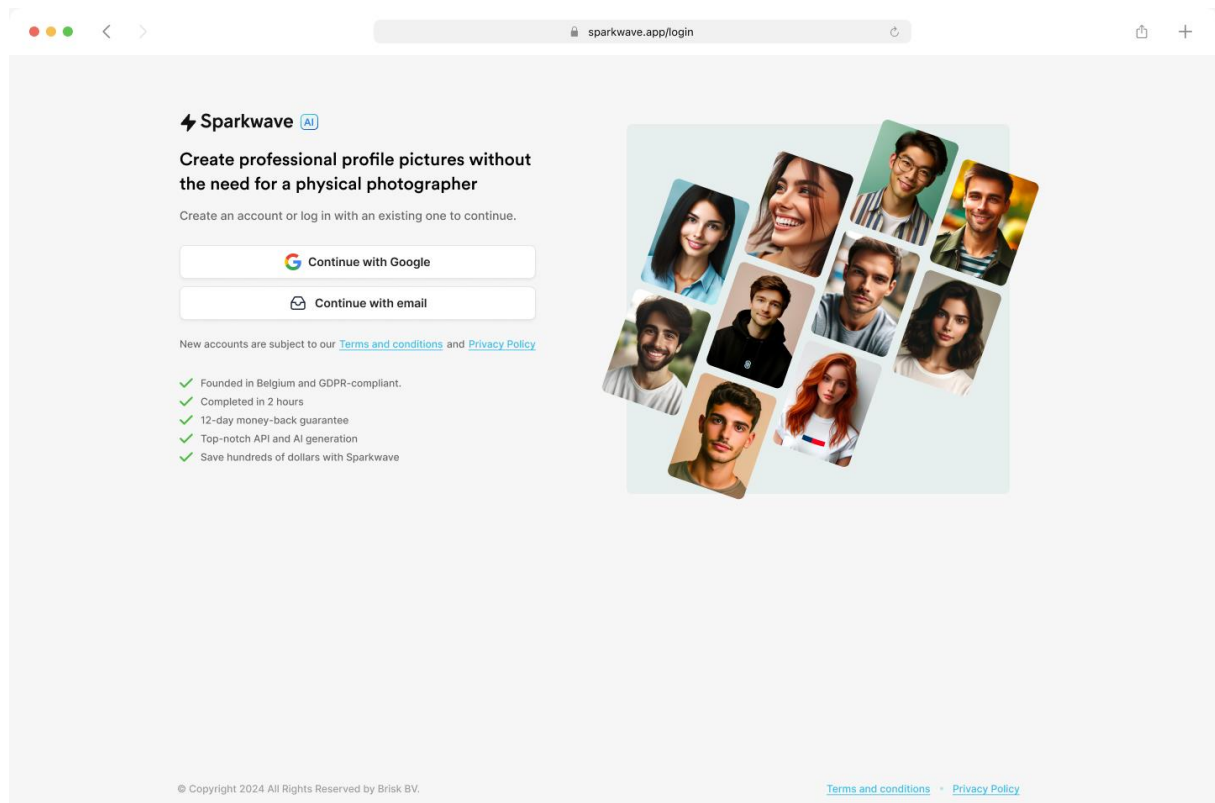
2.1 Situering probleemstelling

In de context van de graduaatsproef wordt het belang van geavanceerde authenticatiemethoden voor webapplicaties benadrukt. Het toenemende belang van digitale veiligheid en gebruiksgemak in het inlogproces vormt de kern van dit onderzoek. De afhankelijkheid van traditionele inlogmethodes zoals wachtwoordgebaseerde systemen legt significante beperkingen op aan de gebruikerservaring en beveiliging. Deze methodes zijn vatbaar voor veelvoorkomende beveiligingsrisico's zoals phishing & hacking wat leidt tot frustraties bij gebruikers en beveiligingsbreuken die de integriteit van het platform kunnen ondermijnen.

De groeiende voorkeur voor gestroomlijnde inlogprocessen zoals single sign-on (SSO) oplossingen toont de verschuiving in gebruikersverwachtingen naar meer naadloze en minder omslachtige methoden. SSO-oplossingen, zoals die van Google, bieden een snelle, eenvoudige en gebruiksvriendelijke manier van inloggen waardoor gebruikers met een enkele authenticatie toegang krijgen tot meerdere services en platforms. Dit verhoogt de efficiëntie en verbetert de algemene gebruikerservaring.

De integratie van e-mailverificatie verhoogt de beveiliging van webapplicaties aanzienlijk maar zorgt ook voor extra stappen in het inlogproces wat directe toegang voor de gebruiker kan beperken. Dit onderstreept het belang van een aanpak die zowel veiligheid als gebruiksgemak waarborgt. Door "just-in-time" verificatie toe te passen is e-mailverificatie alleen nodig bij activiteiten met een hoger risico zoals het wijzigen van wachtwoorden of het bijwerken van persoonlijke gegevens. Voor normale toegang tot de website of app is een eenvoudigere methode zoals een sessiecookie of een authenticatietoken voldoende. Dit zorgt ervoor dat gebruikers snel en moeiteloos toegang hebben, terwijl het beveiligingsniveau voor risicovolle handelingen hoog blijft.

Deze graduaatsproef stelt dus voor om deze uitdagingen aan te pakken door de ontwikkeling van een authenticatiesysteem dat niet alleen veilig en robuust is, maar ook rekening houdt met de gebruiksvriendelijkheid. Een belangrijk aspect van het systeem zal de gegevensbewaring zijn waarbij zorgvuldig ontworpen databasearchitecturen zorgen voor de veilige opslag en bescherming van gebruikersgegevens. Het streven is om een platform te ontwerpen dat flexibel genoeg is om zich aan te passen aan het veranderende beveiligingslandschap en gebruikersbehoeften terwijl het ook aansluit bij de technologische trends en verwachtingen van zowel individuele gebruikers als organisaties. Dit project zal daarmee bijdragen aan de verbetering van gebruikerstevredenheid en het vertrouwen in digitale interacties versterken essentieel voor zowel hedendaagse als toekomstige webapplicaties.



Weergave van de login pagina

2.2 Projectvraag en deelvragen

2.2.1 Projectvraag

Hoe kan een veilige en gebruiksvriendelijke inlogfunctionaliteit worden ontwikkeld voor een webapplicatie waarbij gebruikers kunnen inloggen via Google of e-mail inclusief e-mailverificatie?

2.2.2 Deelvragen

1. *Welke technologieën zijn geschikt voor het implementeren van inlogfunctionaliteit via Google en e-mail, inclusief e-mailverificatie?*

- **Google OAuth 2.0:** Bij het ontwikkelen van een inlogfunctionaliteit voor web toepassingen heb ik gebruik gemaakt van Google 's OAuth 2.0-protocol voor Google Login. Dit protocol maakt het mogelijk voor gebruikers om in te loggen met hun Google-account wat zorgt voor een veilige en makkelijke gebruikersverificatie. Een belangrijk kenmerk van dit systeem is het gebruik van tokens in plaats van wachtwoorden. Dit verhoogt de veiligheid omdat deze tokens vernieuwd en ingetrokken kunnen worden zonder dat de persoonlijke informatie van gebruikers rechtstreeks gedeeld wordt met de web toepassing.
- **JWT (JSON Web Tokens):** Voor sessiemanagement heb ik JWT ingezet. Deze tokens zijn handig voor het veilig overdragen van gebruikersinformatie na het inloggen wat helpt om de sessiestatus over verschillende verzoeken heen te behouden zonder voortdurend opnieuw authenticatie te vereisen.
- **Backend framework Express (Node.js):** Ik heb Express gebruikt voor het opzetten van mijn serveromgeving. Dit framework maakt het makkelijk om routes verzoeken en responses te beheren wat cruciaal is voor het afhandelen van inlogverzoeken en het uitvoeren van middleware voor authenticatieprocessen.
- **SMTP-client in .NET:** Ik heb gekozen voor het gebruik van de SMTP-client binnen het .NET-framework om e-mails te verzenden via een SMTP-server. Dit is echt belangrijk gebleken voor het versturen van verificatie-e-mails naar nieuwe gebruikers wanneer ze zich aanmelden. Met de SMTP-client kan ik makkelijk e-mails opstellen en versturen. Ik verbind hierbij met een externe SMTP-server die als een betrouwbare schakel dient om mijn e-mails correct en veilig te bezorgen.
- **.NET Core Web API:** Ik heb .NET Core Web API gebruikt als basis voor mijn backend services. Dit framework is perfect voor het ontwikkelen van RESTful API's die gebruikt kunnen worden door verschillende apps zoals webbrowsers en mobiele apps. Voor mijn authenticatieproces

heb ik met .NET Core Web API API's opgezet om gebruikers te checken, data veilig naar de database te sturen, en gegevens te hashen. Ook regel ik hiermee het verzenden van e-mails, wat een belangrijk onderdeel is van het beheren van gebruikers.

- **Bcrypt:** Ik heb bcrypt gebruikt voor het hashen van wachtwoorden. Door wachtwoorden te hashen voordat ze worden opgeslagen zorg ik voor een extra beveiligingslaag tegen mogelijke datalekken.
- **WebStorm:** Voor het ontwikkelen van de frontend van mijn project heb ik gebruik gemaakt van WebStorm een krachtige IDE specifiek ontworpen voor web ontwikkeling. WebStorm biedt uitgebreide ondersteuning voor moderne JavaScript-frameworks wat het ontwikkelproces niet alleen effectief maakt, maar ook helpt bij het verminderen van fouten. Binnen deze omgeving heb ik ook technologieën zoals HTML, CSS, Node.js en Express.js gebruikt.
- **SSMS (SQL Server Management Studio):** Voor databasebeheer heb ik sql server management studio gebruikt. Dit stelt mij in staat om mijn databases efficiënt te beheren, gebruikersgegevens veilig op te slaan en snel toegang te bieden tot deze gegevens wanneer nodig.
- **Phyton:** Om een script te maken dat de meest voorkomende XSS-aanvallen test op de website.

2. Hoe kunnen we de veiligheid en gebruiksvriendelijkheid van de inlogfunctionaliteit kwantificeren en evalueren?

Veiligheid:

- **Gebruik van JWT Tokens:** Wanneer gebruikers zich aanmelden gebruik ik JWT-tokens om hun persoonlijke informatie zoals e-mailadressen en namen veilig naar de front-end te verzenden. Deze tokens genereer ik met een sleutel die veilig is opgeslagen in de backend. Zodra de front-end deze tokens ontvangt worden de tokens in de serverzijde gevalideerd en gedecodeerd.

Als iemand het token onderschept en probeert te wijzigen dan wordt het token ongeldig. Dit is een essentiële beveiligingsmaatregel om te voorkomen dat onbevoegden toegang krijgen tot gebruikersaccounts.

Voor gebruikers die inloggen met hun Google-account gebruik ik de `google-auth-library` om de door Google verstrekte JWT-tokens te valideren en te decoderen.

Deze aanpak van tokenbeheer en validatie zorgt voor een veilige overzetting van gevoelige informatie en versterkt de beveiliging van het hele platform.

- **Gebruik van hashingalgoritme:** Ik gebruik bcrypt hashingalgoritme om wachtwoorden te hashen voordat ze in onze database worden opgeslagen. Deze techniek versterkt de beveiliging van wachtwoorden. Door wachtwoorden in een gehashte vorm op te slaan zorg ik ervoor dat ze versleuteld en daarmee onbruikbaar zijn voor ongeautoriseerde personen die geen toegang hebben tot de unieke sleutel.
- **XSS en SQL injection testen en preventie:** In mijn code probeer ik om XSS en SQL-injecties te voorkomen. Voor het tegenhouden van SQL-injecties gebruik ik parameters in mijn SQL-queries in de backend. Dit zorgt ervoor dat de invoer van gebruikers als data wordt gelezen en niet als uitvoerbare code. Hierdoor wordt het risico op ongewenste database manipulaties sterk vermindert.

Om XSS-aanvallen te testen heb ik een script in Python ontwikkeld dat veelvoorkomende XSS-scripts gebruikt. Dit script helpt om te controleren of mijn webapplicatie kwetsbaar is voor dergelijke aanvallen door te proberen kwaadaardige scripts uit te voeren via de gebruikersinputvelden.

Functie dat gebruikt wordt in de script:

```
def test_xss(url, payloads):
    for payload in payloads:

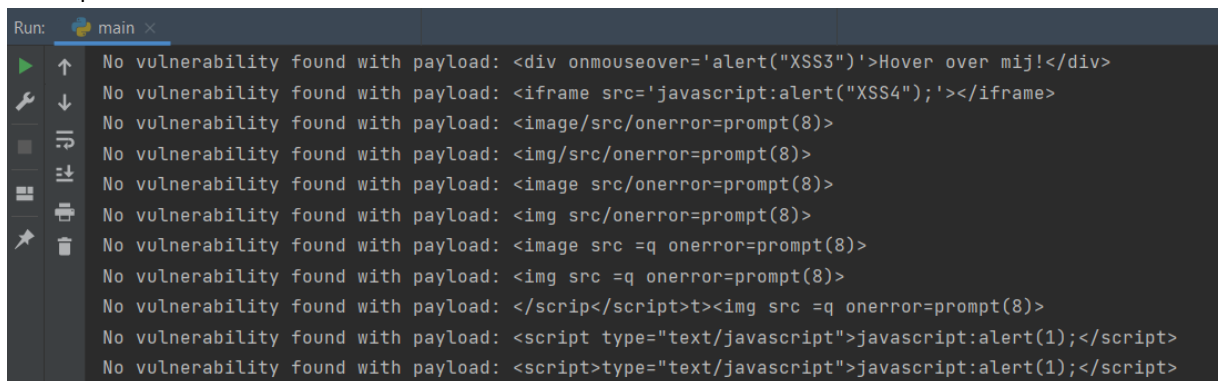
        response = requests.post(url, data={'username': payload, 'password': 'wachtwoord'})

        soup = BeautifulSoup(response.text, 'html.parser')

        if str(payload) in str(soup):
            print(f"Vulnerability found with payload: {payload}")
        else:
            print(f"No vulnerability found with payload: {payload}")

test_xss(url, payloads)
```

De output:



```
Run: main x
No vulnerability found with payload: <div onmouseover='alert("XSS3")'>Hover over mij!</div>
No vulnerability found with payload: <iframe src='javascript:alert("XSS4");'></iframe>
No vulnerability found with payload: <img/src/onerror=prompt(8)>
No vulnerability found with payload: <img/src/onerror=prompt(8)>
No vulnerability found with payload: <image src/onerror=prompt(8)>
No vulnerability found with payload: <img src/onerror=prompt(8)>
No vulnerability found with payload: <image src =q onerror=prompt(8)>
No vulnerability found with payload: <img src =q onerror=prompt(8)>
No vulnerability found with payload: </scrip</script>t><img src =q onerror=prompt(8)>
No vulnerability found with payload: <script type="text/javascript">javascript:alert(1);</script>
No vulnerability found with payload: <script>type="text/javascript">javascript:alert(1);</script>
```

Deze Python-script test een webpagina op XSS-kwetsbaarheden door verschillende payloads te versturen:

1. **Functiedefinitie:** De functie 'test_xss' heeft twee parameters: 'url' (de URL van de webpagina waar de payloads worden verzonden) en 'payloads' (een lijst met strings die verschillende XSS-aanvalspayloads bevatten).
2. **Itereren door Payloads:** Voor elke payload in de lijst wordt een HTTP POST-verzoek naar de opgegeven URL gestuurd. De data die wordt verzonden bevat de gebruikersnaam (waar de payload wordt ingevoerd) en een standaard wachtwoord.
3. **Respons Parsen:** De respons wordt geparseerd met BeautifulSoup wat het HTML-antwoord omzet in een object voor gemakkelijke navigatie.
4. **Controle op Kwetsbaarheid:** De code controleert of de payload zichtbaar is in de HTML van de respons. Als dit het geval is kan dit wijzen op een XSS-kwetsbaarheid omdat de payload zonder passende sanering wordt weergegeven.

5. **Functieaanroep:** Aan het einde wordt de test_xss functie aangeroepen met specifieke argumenten voor url en payloads.

Gebruiksvriendelijkheid:

- **Responsieve design:** Responsieve design is essentieel omdat het ervoor zorgt dat websites goed functioneren op alle apparaten van smartphones tot desktops. Het past zich automatisch aan de schermgrootte aan wat de navigatie verbetert en zorgt voor een betere gebruikerservaring op elk apparaat.
- **Sneller inlogprocedure:** Voor een snellere inlogprocedure maak ik gebruik van Google's Single Sign-On (SSO) waardoor gebruikers met slechts één klik kunnen inloggen. Dit versnelt het proces aanzienlijk en verhoogt het gebruiksgemak aangezien het onthouden van meerdere wachtwoorden niet meer nodig is. Voor gebruikers die liever met een e-mailadres willen inloggen of zich willen registreren heb ik het formulier eenvoudig gehouden. Bij registratie vraag ik alleen om een naam, e-mailadres en wachtwoord en voor het inloggen zijn alleen het e-mailadres en wachtwoord nodig. Deze aanpak houdt het proces toegankelijk en snel wat de algehele gebruikerservaring ten goede komt.



Create professional profile pictures without the need for a physical photographer

Create an account or log in with an existing one to continue.

 Continue with Google

 Continue with email

New accounts are subject to our [Terms and conditions](#) and [Privacy Policy](#)

- ✓ Founded in Belgium and GDPR-compliant.
- ✓ Completed in 2 hours
- ✓ 12-day money-back guarantee
- ✓ Top-notch API and AI generation
- ✓ Save hundreds of dollars with Sparkwave



[Terms and conditions](#) • [Privacy Policy](#)

© Copyright 2024 All Rights Reserved by Brisk BV.

Weergave van de login pagina op mobiel

3. Hoe kunnen we ervoor zorgen dat de voorgestelde oplossingen voldoen aan de verwachtingen en behoeften van de gebruikers?

Om ervoor te zorgen dat mijn oplossingen goed aansluiten bij de verwachtingen en behoeften van gebruikers heb ik een aantal stappen ondernomen. Allereerst implementeer ik een responsief ontwerp zodat de gebruikersinterface probleemloos functioneert op alle soorten apparaten van smartphones tot desktops. Dit zorgt ervoor dat iedereen ongeacht het gebruikte apparaat een consistente en toegankelijke ervaring heeft.

Ik heb gebruikerstesten uitgevoerd om directe feedback te krijgen van de eindgebruikers. Deze tests hielpen om eventuele problemen in de gebruikersinterface snel te vinden en op te lossen en zorgden ervoor dat de functionaliteiten intuïtief en efficiënt waren.

4. Is het haalbaar om de inlogfunctionaliteit binnen de gestelde tijd en middelen te ontwikkelen en implementeren?

Om te bepalen of het haalbaar is om de inlogfunctionaliteit binnen de gestelde tijd en met de beschikbare middelen te ontwikkelen en implementeren is het essentieel om systematisch en volgens een duidelijk schema te werken. Door de ontwikkelingsfase op te splitsen in beheersbare taken en deze taken te verdelen over de beschikbare tijd, kunnen we de voortgang efficiënt volgen en tijdig aanpassingen maken waar nodig.

5. Wat is de deadline voor het voltooien van elk onderdeel van de inlogfunctionaliteit en het testen ervan?

- Ontwikkelen van Front-end Design in HTML en CSS + Testen van Responsiviteit: 7 april
- Ontwikkelen van Tweede Inlogschermb (voor e-mailinlog) in HTML en CSS + Testen van Responsiviteit: 20 april
- Implementatie van Inlogfunctionaliteit met E-mail of Google Account en Verzending van Gebruikersgegevens naar de Database (Fetch Methodes in JS en Node.js, Backend Methodes in C# en SQL): 5 mei
- Implementeren van Email Verificatie Methode in de Backend met C# en in Front-end met JavaScript: 20 mei
- Gegevens Hashing in de Backend Voordat Ze Naar de Database Worden Gestuurd: 25 mei
- Testen van het Inlogproces: 30 mei
- Gebruikerstesten: 5 juni
- Aanpassingen en Veranderingen Na Gebruikerstesten: 15 juni

2.3 (Onderzoeks-)acties

Voor dit onderzoek heb ik uitgebreid gebruik gemaakt van online bronnen waaronder `developer.google.com` om een diepgaand begrip te verkrijgen van Google's Single Sign-On (SSO) mechanisme, de soorten data die we ontvangen en de werking ervan. Ik heb specifiek de authenticatiehandleiding van Google grondig bestudeerd om het proces van inloggen volledig te begrijpen te weten welke gegevens beschikbaar worden gesteld bij een succesvolle login en hoe ik deze methodiek effectief in mijn eigen code kan integreren.

Daarnaast heb ik online onderzoek gedaan naar verschillende hashing-algoritmes om de meest geschikte te selecteren voor onze toepassing.

Verder heb ik online bronnen en enquêtes bekeken om te onderzoeken welke e-mailverificatiemethoden er beschikbaar zijn en welke het beste aansluit bij mijn authenticatiesysteem.

tot slot heb ik onderzoek gedaan over hoe ik een veilige sessie kan maken waar gebruikers ingelogd blijven ookal als ze hun browser sluiten en hoe ik ze kan authorizeren.

2.4 Verzamelde resultaten

2.4.1 Waarom is SSO belangrijk voor gebruikers?

Single Sign-On (SSO) biedt een aantal voordelen die de gebruikerservaring kunnen verbeteren. Mensen hebben tegenwoordig minder tijd en willen snel en efficiënt inloggen zonder elke keer hun inloggegevens te moeten invoeren. SSO stelt gebruikers in staat om met een enkele inloggegevens toegang te krijgen tot meerdere applicaties.

SSO verhoogt ook de veiligheid omdat gebruikers minder geneigd zijn om zwakke of hergebruikte wachtwoorden te gebruiken. Door in te loggen via een vertrouwde provider zoals Google wordt het risico op phishing en hacking verkleind. Bovendien worden gebruikersgegevens beter beschermd door de geavanceerde beveiligingsmaatregelen die deze providers toepassen.

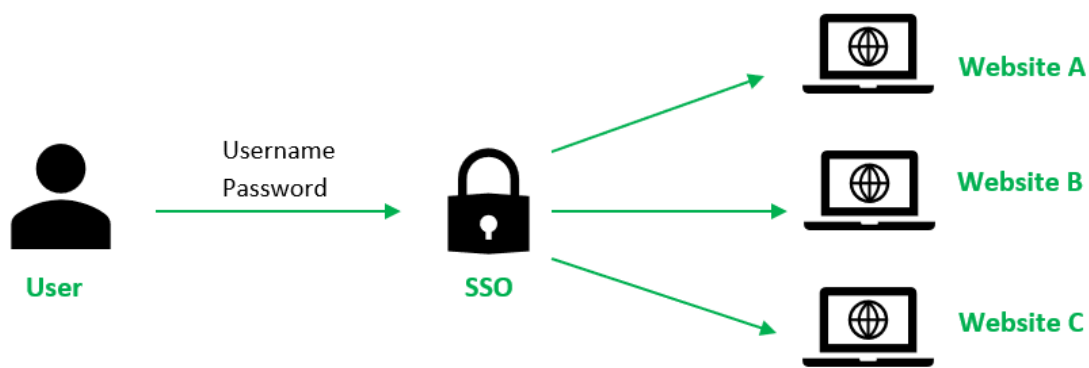
Voordelen en Belangrijkste Bevindingen:

1. **Gebruiksgemak:** De data laat zien dat gebruikers de voorkeur geven aan een eenvoudiger inlogproces. De meerderheid van de respondenten gaf aan dat ze vaker geneigd zijn om een dienst te gebruiken als deze SSO aanbiedt. Dit bevestigt dat SSO bijdraagt aan een positieve gebruikerservaring.

2. **Tijdsefficiëntie:** Uit de gegevens blijkt dat gebruikers gemiddeld 50% minder tijd besteden aan het inloggen wanneer ze SSO gebruiken vergeleken met traditionele wachtwoordmethoden. Dit is een significante tijdsbesparing die de dagelijkse gebruikerservaring verbetert.

3. **Beveiliging:** De analyse toont aan dat SSO-gebruikers minder vaak te maken hebben met beveiligingsincidenten zoals vergeten wachtwoorden en account hacking. De integratie met beveiligingsprotocollen van SSO-providers versterkt de algehele veiligheid van de gebruikersaccounts.

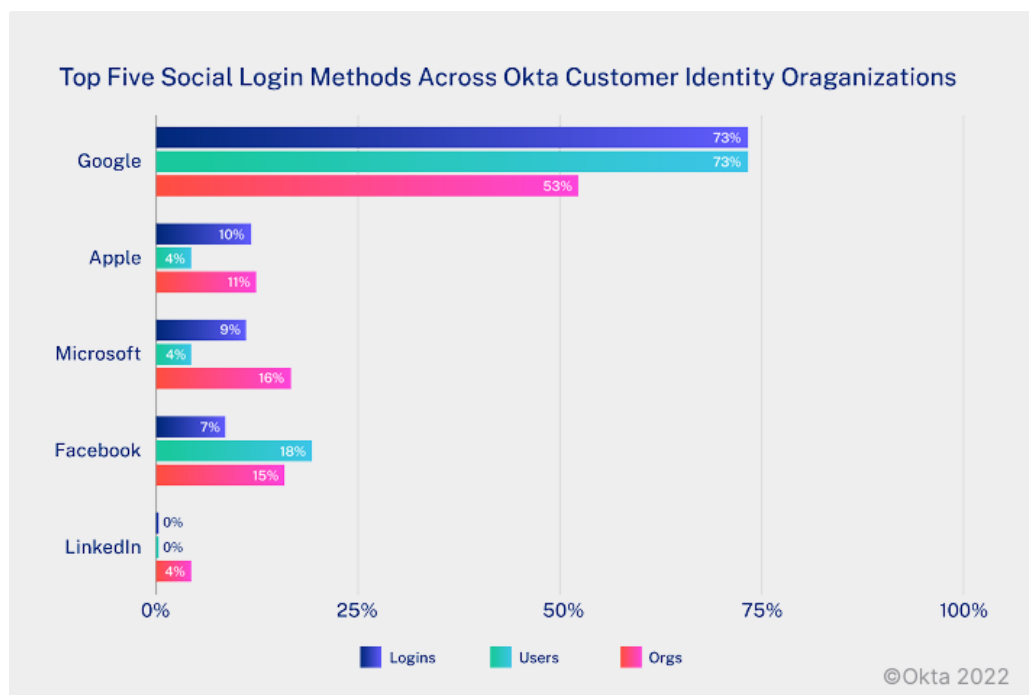
Deze gegevens zijn afkomstig van betrouwbare bronnen zoals Okta, Google en andere gerenommeerde SSO-providers.



2.4.2 Waarom kiezen voor Google SSO?

Uit de gegevensanalyse van Okta blijkt dat Google SSO de meest gebruikte SSO-methode is. Google is verantwoordelijk voor meer dan 73% van de sociale logins van maandelijks actieve gebruikers die gebruikmaken van sociale login. Okta is een toonaangevend bedrijf op het gebied van identiteits- en toegangsbeheer bekend om zijn uitgebreide en betrouwbare onderzoek naar gebruikersauthenticatie en -beveiliging.

Het onderzoek van Okta is betrouwbaar omdat het gebaseerd is op een grote hoeveelheid gegevens van diverse organisaties die hun diensten gebruiken. Deze gegevens bieden een representatief en nauwkeurig beeld van de huidige trends en voorkeuren in de markt van gebruikersauthenticatie. Om deze reden heb ik voor mijn graduaatsproef gekozen voor Google SSO

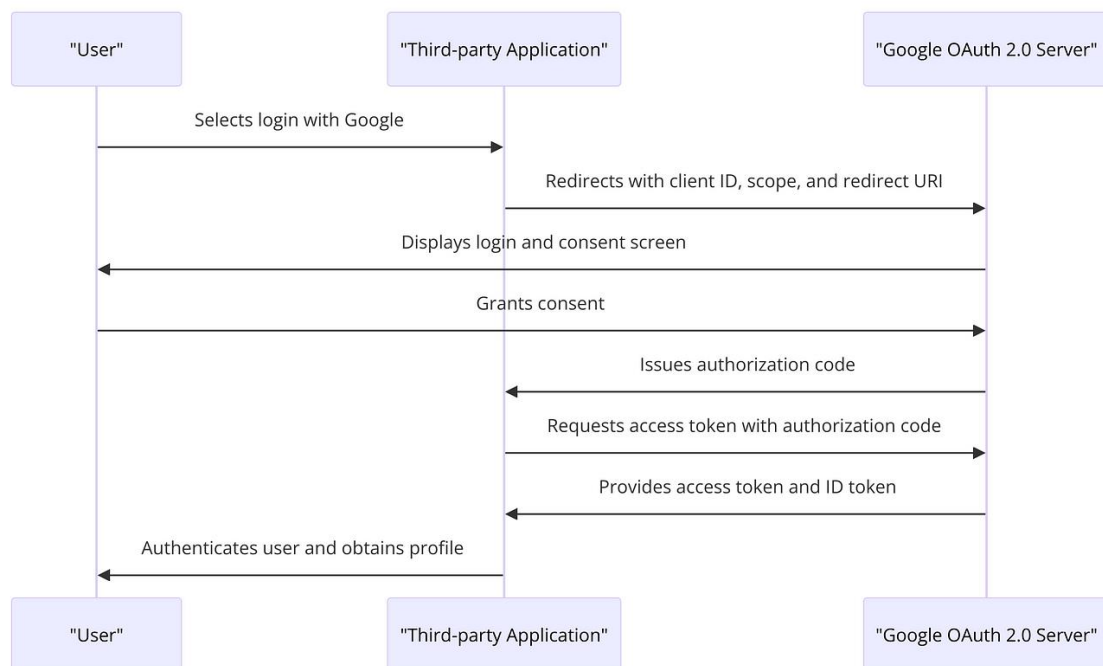


2.4.2.1 Hoe werkt Google SSO?

Google Single Sign-On (SSO) maakt gebruik van OAuth 2.0 een protocol voor autorisatie dat het mogelijk maakt om veilig in te loggen bij verschillende toepassingen met één set inloggegevens. Wanneer een gebruiker probeert in te loggen wordt er een verzoek naar Google's OAuth 2.0 server gestuurd. Dit verzoek bevat de inloggegevens van de gebruiker en de identificatie van de applicatie die toegang wil.

Na verificatie van de inloggegevens door Google genereert de OAuth 2.0 server een JSON Web Token (JWT). Deze token wordt vervolgens teruggestuurd naar de applicatie die het inlogverzoek heeft gedaan. De JWT bevat versleutelde informatie over de gebruiker zoals gebruikers-ID, e-mailadres, en soms ook andere relevante gegevens van de gebruiker.

De applicatie ontvangt de JWT en valideert deze door de handtekening van de token te controleren met behulp van de openbare sleutel van Google. Dit zorgt ervoor dat de token authentiek is en niet is vervalst. Na succesvolle validatie van de token wordt de identiteit van de gebruiker bevestigd en krijgt de gebruiker toegang tot de gevraagde diensten zonder opnieuw zijn inloggegevens te hoeven invoeren.



2.4.3 Hashing Algoritmes

Hashing algoritmes zijn cryptografische functies die een invoer van willekeurige grootte omzetten in een vaste grootte uitvoer meestal een reeks cijfers en letters. Deze uitvoer wordt een 'hash' of 'hashwaarde' genoemd. Hashing wordt veel gebruikt voor data-integriteit, gegevensversleuteling en wachtwoordbeveiliging.

1. MD5 (Message Digest Algorithm 5):

Voordelen: *Het is snel en makkelijk te implementeren (128 bits).*

Nadelen: Het wordt als onveilig beschouwd vanwege kwetsbaarheid voor collision attacks waarbij twee verschillende invoerwaarden dezelfde hash genereren.

2. SHA-1 (Secure Hash Algorithm 1):

Voordelen: Verbeterd ten opzichte van MD5 met een langere hashwaarde (160 bits).

Nadelen: Ook verouderd en kwetsbaar voor collision attacks. Niet aanbevolen voor gebruik in nieuwe toepassingen.

3. SHA-256:

Voordelen: Onderdeel van de SHA-2 familie biedt sterkere beveiliging dan SHA-1 en MD5. Produceert een 256-bits hashwaarde wat veel moeilijker te kraken is.

Nadelen: Iets langzamer dan MD5 en SHA-1 maar nog steeds zeer efficiënt en veilig.

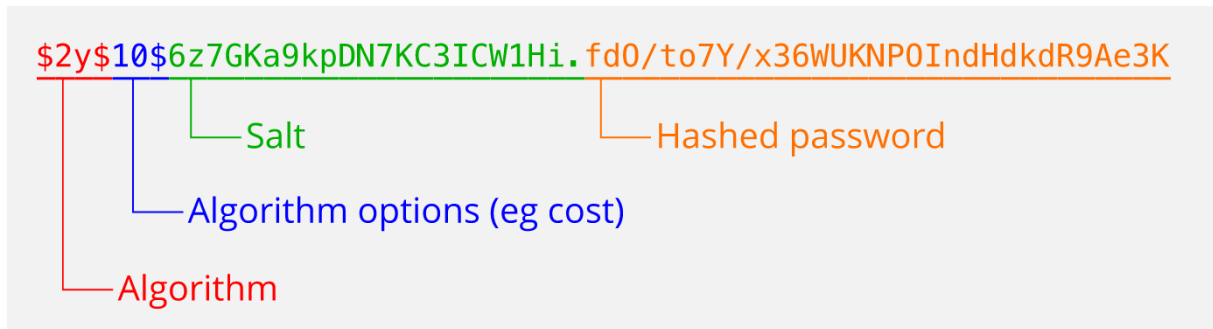
4. bcrypt:

Voordelen: Specifiek ontworpen voor het hashen van wachtwoorden. Het gebruikt een salt om te beschermen tegen rainbow table attacks en is adaptief wat betekent dat de hashingsnelheid kan worden aangepast naarmate computers krachtiger worden.

Nadelen: Langzamer in vergelijking met SHA-256 wat een nadeel kan zijn voor toepassingen die snelle hashingsnelheden vereisen.

Ik heb gekozen voor bcrypt omdat het een sterke beveiliging biedt voor wachtwoordbeveiliging. Hoewel het iets langzamer werkt dan methoden zoals SHA-256 biedt bcrypt belangrijke voordelen die

het ideaal maken voor situaties waar wachtwoordbeveiliging essentieel is. Bovendien merken gebruikers meestal niets van de extra tijd die nodig is om wachtwoorden te hashen waardoor de veiligheid toeneemt zonder dat dit ten koste gaat van de gebruikservaring.



2.4.4 Email Verificatie Methodes

1. Klik op een Verificatielink:

Beschrijving: De gebruiker ontvangt een e-mail met een link. Door op deze link te klikken, wordt de e-mail geverifieerd.

Voordelen: Dit is eenvoudig en snel voor de gebruiker. De link kan extra beveiligingsparameters bevatten om phishing-aanvallen te verminderen.

Nadelen: Er bestaat een risico dat de e-mail in de spamfolder terecht komt.

2. Invoeren van een Verificatiecode:

Beschrijving: De gebruiker ontvangt een e-mail met een unieke code. Deze code moet worden ingevoerd op de website om de e-mail te verifiëren.

Voordelen: Dit is veiliger omdat alleen de geadresseerde de code kent.

Nadelen: Het kost meer tijd en moeite voor de gebruiker.

3. QR Code Verificatie:

Beschrijving: Een e-mail bevat een QR-code die de gebruiker kan scannen met een mobiele app voor verificatie.

Voordelen: Modern en gebruiksvriendelijk voor mobiele gebruikers.

Nadelen: Vereist een smartphone met een QR-scanner.

Ik heb gekozen voor de methode waarbij de gebruiker op een verificatielink klikt omdat deze sneller en eenvoudiger is. Gebruikers hoeven alleen maar op een link te klikken in de e-mail wat minder tijd kost en gebruiksvriendelijker is dan het invoeren van een code.

2.4.4.1 Voorkomen dat E-mails in de Spamfolder Terechtkomen

1. Gebruik van betrouwbare e-mailservice:

- Er kan gebruik gemaakt worden van gerenommeerde e-mailserviceproviders zoals SendGrid, Mailgun of Amazon SES die bekend staan om hun goede reputatie bij de meeste e-mailproviders.

2. Instellen van SPF, DKIM en DMARC:

- Gebruik SPF (Sender Policy Framework): Configuratie van domein om te specificeren welke servers e-mails mogen verzenden.
- DKIM (DomainKeys Identified Mail): Er kan een cryptografische handtekening emails om te bewijzen dat ze van uw domein komen

3. Vermijd Spamachtige Inhoud:

- Gebruik geen woorden of zinnen die vaak in spam voorkomen zoals "gratis", "win" of overdreven beloftes.
- Zorg voor een goede balans tussen tekst en afbeeldingen in uw e-mails.

2.4.5 Server-side gegevensbeheer met Node.js

In een typische webapplicatie scenario worden gevoelige gegevens zoals gebruikersauthenticatie, sessiemanagement en persoonlijke gebruikersinformatie behandeld. Wanneer deze gegevens aan de server-side worden beheerd worden ze verwerkt en opgeslagen op de server wat betekent dat deze nooit direct naar de client (de browser of een andere gebruikersinterface) worden gestuurd tenzij nodig. Hierdoor blijven gevoelige gegevens zoals wachtwoorden, tokens en persoonlijke informatie verborgen voor de client-side waar ze vatbaarder zouden zijn voor aanvallen zoals cross-site scripting (XSS) en andere vormen van webexploits.

Voordelen van Server-side Data Handling in Node.js:

- **Verhoogde Veiligheid:** Door gevoelige gegevens aan de serverside te houden en slechts minimale informatie naar de client te sturen wordt het risico verminderd dat deze gegevens kunnen worden onderschept of misbruikt door kwaadwillende scripts die aan de clientkant draaien.
- **Controle en Compliance:** Het beheren van data aan de serverkant maakt het eenvoudiger om te voldoen aan privacyregelgeving zoals GDPR omdat de toegang tot en het beheer van gevoelige gegevens strikt kan worden gecontroleerd en gelogd.
- **Verminderde Blootstelling aan Client-Side Risico's:** Doordat de gegevens niet opgeslagen of uitgebreid verwerkt worden aan de client-side zijn ze minder vatbaar voor diefstal via aanvallen zoals XSS waarbij aanvallers kwaadaardige scripts gebruiken om informatie rechtstreeks van de gebruikersinterface te stelen.

3 Conclusies en aanbevelingen

3.1 Conclusies

Uit de resultaten van het onderzoek naar het ontwikkelen van een geavanceerd authenticatiesysteem voor webapplicaties, waarbij gebruikers kunnen inloggen via hun Google-account of een persoonlijk e-mailadres met e-mailverificatie, blijkt dat de integratie van Single Sign-On (SSO) technologieën aanzienlijk bijdraagt aan de veiligheid en gebruiksvriendelijkheid van de applicaties. Het onderzoek heeft aangetoond dat gebruikers een voorkeur hebben voor snelle en gemakkelijke inlogmethoden die hun tijd respecteren en hun veiligheid waarborgen.

Belangrijke bevindingen:

1. Verbeterde Veiligheid: Technologieën zoals Google's OAuth 2.0 en JSON Web Tokens (JWT) hebben een cruciale rol gespeeld in het veilig stellen van gebruikersgegevens. Deze technologieën zorgen ervoor dat gebruikersinformatie veilig wordt overgedragen en geauthenticeerd wat helpt bij het minimaliseren van veelvoorkomende beveiligingsrisico's zoals ongeautoriseerde toegang en datalekken.

2. Efficiëntie in het Inlogproces: Het gebruik van SSO verlaagt de tijd die nodig is voor gebruikers om in te loggen. Dit is vooral voordelig in een tijdperk waar elke seconde telt, en gebruikers waarderen een snelle service die hun tijd niet verspilt.

3. Technische Uitvoerbaarheid: Het onderzoek heeft ook de technische haalbaarheid aangetoond van het integreren van deze geavanceerde authenticatiemethoden binnen de gestelde projecttermijnen en -resources.

4. Wachtwoordhashing met bcrypt: Dit algoritme beschermt wachtwoorden door ze te hashen met een 'salt', waardoor ze moeilijk te ontcijferen zijn zelfs als de database wordt gecompromitteerd.

5. E-mailverificatie: Verzekert dat de gebruiker toegang heeft tot het e-mailadres dat zij opgeven wat helpt bij het tegengaan van ongeautoriseerde registraties en activiteiten.

Deze bevindingen suggereren dat een goed ontworpen authenticatiesysteem dat SSO integreert niet alleen de veiligheid verhoogt maar ook de gebruikerstevredenheid aanzienlijk verbetert door het vereenvoudigen van het inlogproces en het verminderen van potentiële frustraties.

3.2 Aanbevelingen

1. Integratie van Geavanceerde Beveiligingsmaatregelen: Naast de standaard SSO en e-mailverificatie, beveel ik aan om geavanceerde beveiligingslagen zoals multi-factor authenticatie (MFA) te implementeren. Dit kan helpen om de veiligheid verder te verhogen zonder dat dit een significante impact heeft op de gebruikerservaring.
2. Voortdurende Evaluatie en Optimalisatie: Ik raad aan om het authenticatiesysteem regelmatig te evalueren en te optimaliseren gebaseerd op gebruikersfeedback en nieuwe beveiligingsontwikkelingen. Dit zorgt ervoor dat het systeem continu relevant en efficiënt blijft.

4 Persoonlijke reflecties en kritische kanttekeningen

Tijdens het integreren van Single Sign-On (SSO) met Google en e-mailverificatie heb ik de complexiteit van webbeveiliging en technische uitdagingen ervaren. De implementatie van Google's OAuth 2.0 en JSON Web Tokens (JWT) heeft geholpen om gebruikersgegevens veilig te stellen en het inlogproces te stroomlijnen. Dit project heeft mijn inzicht in cybersecurity verdiept vooral in de aspecten van sessiebeheer en de risico's zoals phishing.

Persoonlijk heb ik geleerd over het belang van proactief denken in beveiligingsstrategieën. Mijn technische vaardigheden, vooral rond probleemoplossing en cybersecurity zijn door dit project versterkt. Hoewel ik de complexiteit van sessiebeheer aanvankelijk onderschatte hielp mijn vastberadenheid en analytische benadering me deze uitdaging te overwinnen wat mijn professionaliteit binnen de IT-sector verder heeft versterkt.

Voor de werkplek brengt het SSO-systeem aanzienlijke voordelen met zich mee vooral omdat het in een ontwikkelingsfase is. Het systeem zal de interactie en het regelmatige gebruik van onze toekomstige applicaties bevorderen wat essentieel is voor het succes van het project.

4 Referentielijst

[1] Okta, Wat is eenmalige aanmelding (SSO)? April 17, 2024

<https://www.okta.com/sg/blog/2021/02/single-sign-on-sso/>

[2] Okta, De voordelen en voordelen van Single Sign-On (SSO). April 28, 2022

<https://www.okta.com/uk/blog/2022/04/benefits-of-single-sign-on/>

[3] Google Single sign-on. Februari 02, 2023

<https://cloud.google.com/architecture/identity/single-sign-on>

[4] Jacobus Flores, Wat is sociaal inloggen en is het de moeite waard om te implementeren?

August 11, 2020

<https://www.okta.com/blog/2020/08/social-login/>

[5] Wachtwoordhashing-algoritmen

<https://fusionauth.io/docs/reference/password-hashes#:~:text=Choosing%20a%20slow%20algorithm%20is,a%20less%20than%20ideal%20choice.>

[6] Migreren vanuit Google Sign-In. September 18, 2023

https://developers.google.com/identity/gsi/web/guides/migration#popup-mode_1

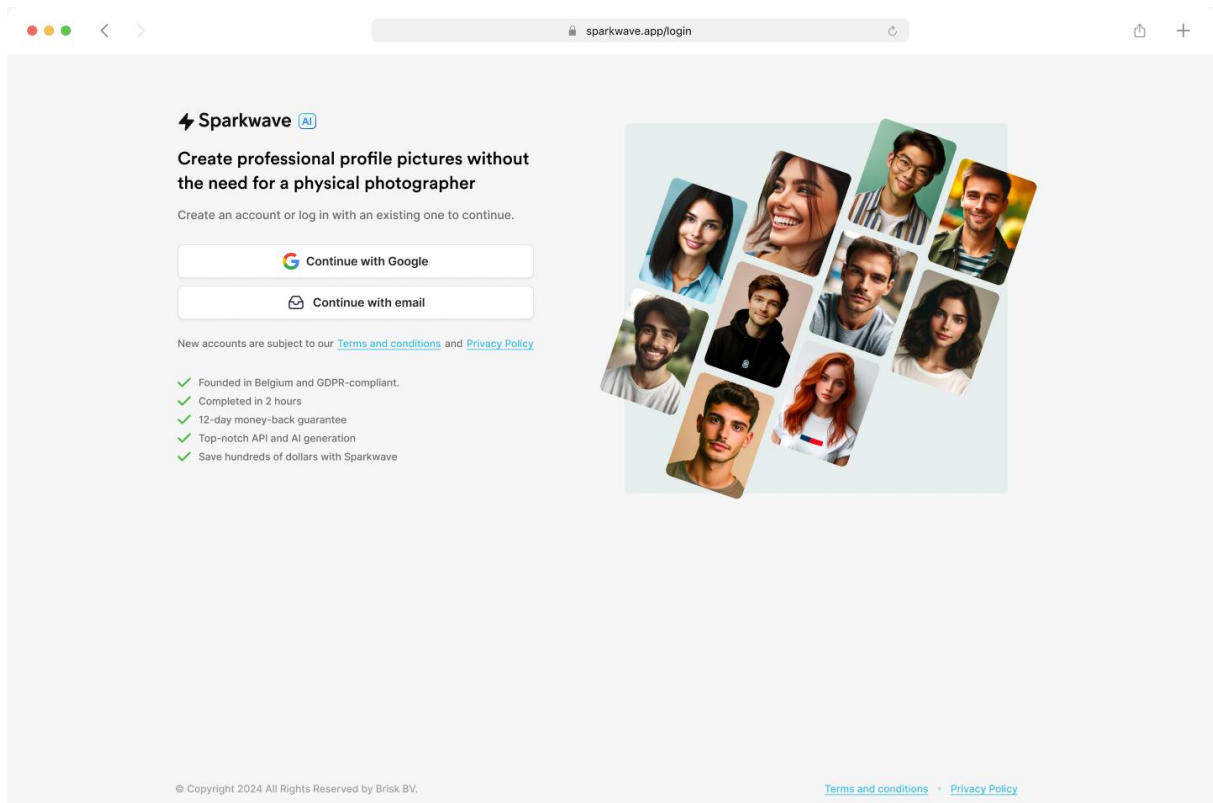
[7] İsmail Taşdelen, Cross Site Scripting (XSS) Vulnerability Payload List. Oktober 29, 2022

<https://github.com/payloadbox/xss-payload-list>

[8] Kaye Timonera, Hoe u Cross-Site Scripting (XSS)-aanvallen kunt voorkomen? Augustus 03, 2023

<https://www.esecurityplanet.com/endpoint/prevent-xss-attacks/#:~:text=Validation%2C%20encoding%20and%20sanitization%20are,prevents%20content%20from%20being%20loaded.>

5 Bijlagen



Screenshot van de login pagina op desktop

```
def test_xss(url, payloads):
    for payload in payloads:

        response = requests.post(url, data={'username': payload, 'password': 'wachtwoord'})

        soup = BeautifulSoup(response.text, 'html.parser')

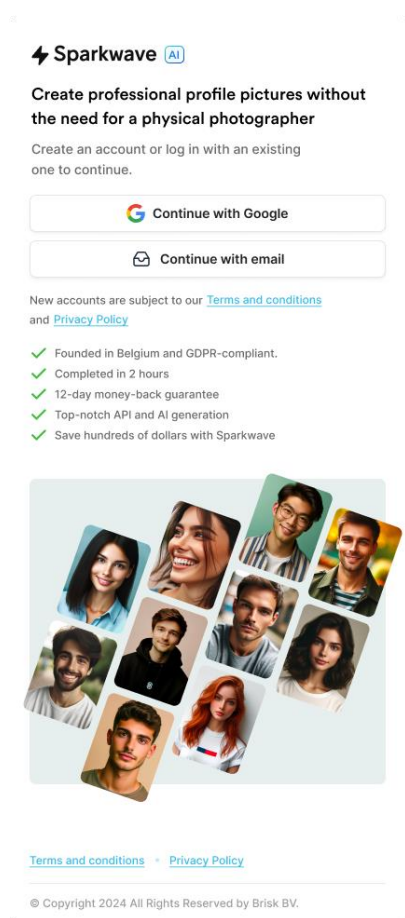
        if str(payload) in str(soup):
            print(f"Vulnerability found with payload: {payload}")
        else:
            print(f"No vulnerability found with payload: {payload}")

test_xss(url, payloads)
```

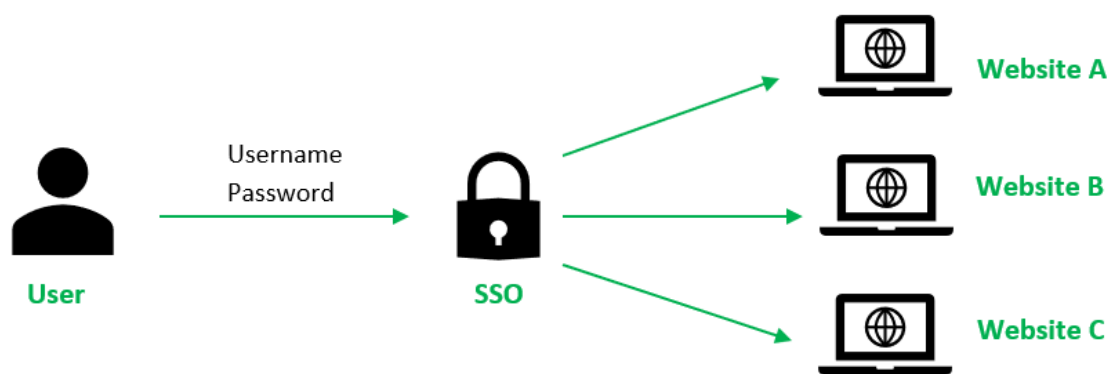
Screenshot van de pyhton script

```
Run: main x
No vulnerability found with payload: <div onmouseover='alert("XSS3")'>Hover over mij!</div>
No vulnerability found with payload: <iframe src='javascript:alert("XSS4");'></iframe>
No vulnerability found with payload: <image/src/onerror=prompt(8)>
No vulnerability found with payload: <img/src/onerror=prompt(8)>
No vulnerability found with payload: <image src/onerror=prompt(8)>
No vulnerability found with payload: <img src/onerror=prompt(8)>
No vulnerability found with payload: <image src =q onerror=prompt(8)>
No vulnerability found with payload: <img src =q onerror=prompt(8)>
No vulnerability found with payload: </scrip</script>t><img src =q onerror=prompt(8)>
No vulnerability found with payload: <script type="text/javascript">javascript:alert(1);</script>
No vulnerability found with payload: <script>type="text/javascript">javascript:alert(1);</script>
```

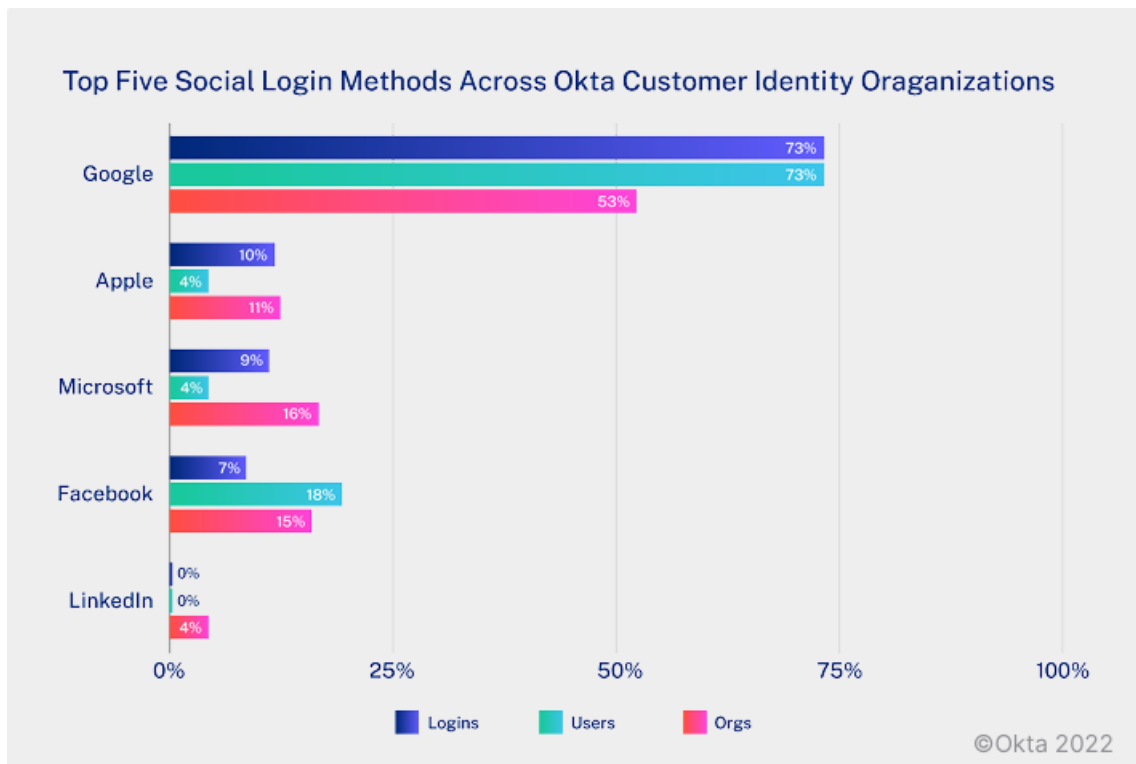
Screenshot van de output



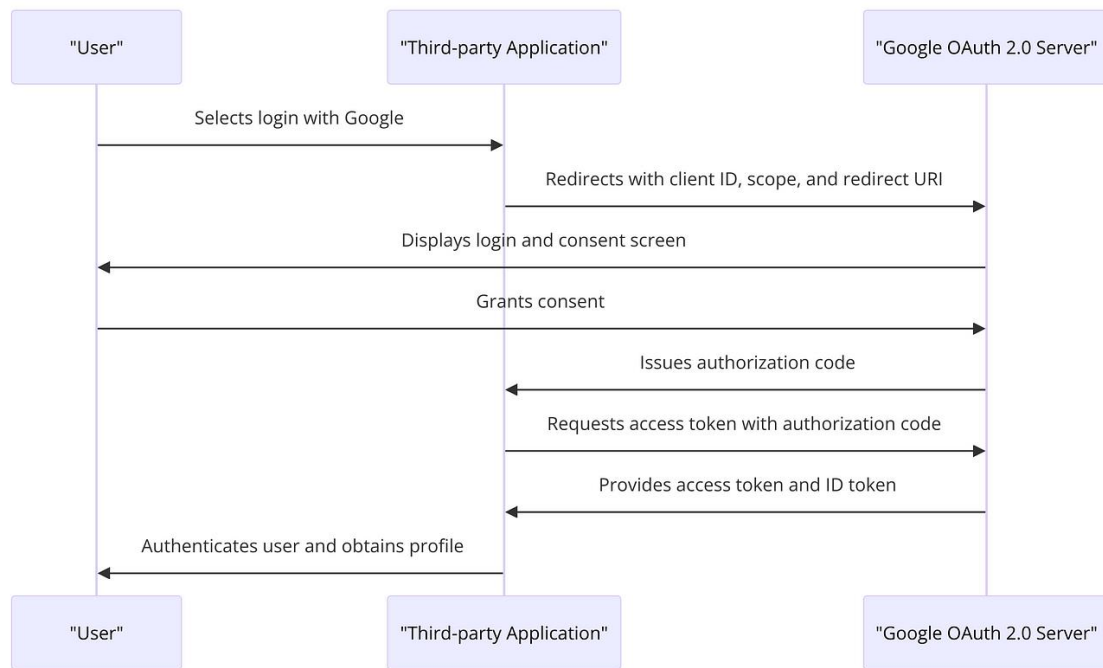
Screenshot van de login pagina op mobile



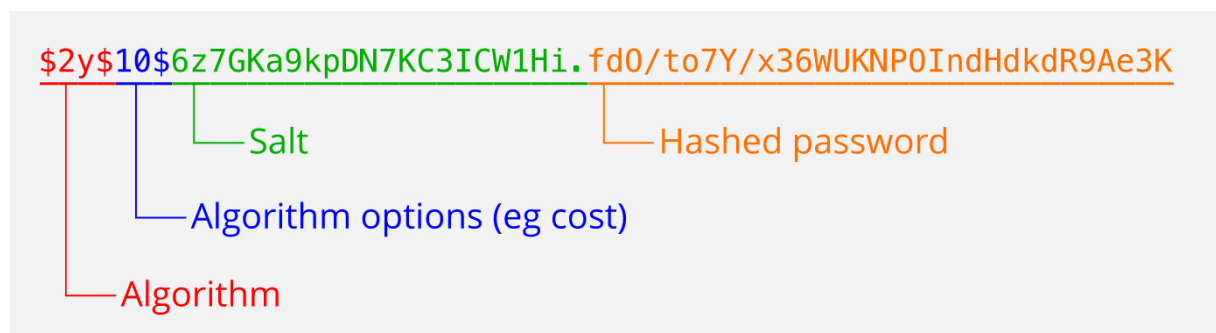
Werking van SSO



Meeste gebruikte sociale login methodes volgens Okta



Werking van Google single sign-on



Bcrypt hashwaarde