

Onderzoeksverslag

SecureNotes

Veilige en Versleutelde Communicatie voor Externe Notities



Student: Alex Cheng (1634967)

Bedrijf Begeleider: Tom Klein

Docent Begeleider: Peter Schulz

Docent Assessor: Martijn Driessen



Opleiding: HBO-ICT (Web Development)

Schooljaar: 2023-2024

Versie: 1.0



Inhoudsopgave

1. Inleiding	4
2. Probleemstelling	5
2.1. Wat is de huidige situatie?	5
2.2. Wat is de gewenste situatie?	5
2.3. Wat is het verschil tussen de huidige en gewenste situatie?	5
3. Doelstelling	6
4. Vraagstelling	7
4.1. Hoofdvraag	7
4.2. Deelvragen	7
4.2.1. Welke encryptie mogelijkheden zijn er?	7
4.2.1.1. Motivatie	7
4.2.1.2. Relevantie	7
4.2.1.3. Onderzoeksmethode	7
4.2.2. Wat kan naast encryptie nog meer toegepast worden?	8
4.2.2.1. Motivatie	8
4.2.2.2. Relevantie	8
4.2.2.3. Onderzoeksmethode	8
4.2.3. Hoe kan een secret key het meest efficiënt en veilig gedeeld worden?	8
4.2.3.1. Motivatie	8
4.2.3.2. Relevantie	8
4.2.3.3. Onderzoeksmethode	8
4.2.4. Welke stack is het meest gepast bij deze opdracht?	9
4.2.4.1. Motivatie	9
4.2.4.2. Relevantie	9
4.2.4.3. Onderzoeksmethode	9
4.2.5. Hoe pas ik al bestaande cryptografische libraries toe?	9
4.2.5.1. Motivatie	9
4.2.5.2. Relevantie	9
4.2.5.3. Onderzoeksmethode	9
5. Resultaten	10
5.1. Welke encryptie mogelijkheden zijn er?	10
5.1.1. Wat is encryptie?	10
5.1.2. Hoe werkt encryptie?	10
5.1.3. Encryptie types	11
5.1.3.1. Symmetrisch	11
5.1.3.2. Asymmetrisch	11
5.1.4. Encryptie algoritmes	12
5.1.4.1. AES (Advanced Encryption Standard)	12

5.1.4.2. DES (Data Encryption Standard)	12
5.1.4.3. RSA (Rivest-Shamir-Adleman)	13
5.1.4.4. ECC (Elliptic Curve Cryptography)	13
5.2. Wat kan naast encryptie nog meer toegepast worden?	14
5.2.1. Hashing	14
5.2.2. Obfuscation	15
5.2.3. Masking	16
5.2.4. Two-Factor Authentication (2FA)	17
5.2.5. Access Control	18
5.3. Hoe kan een secret key het meest efficiënt en veilig gedeeld worden?	19
5.3.1. Wat is een secret key?	19
5.3.2. Delen van secret keys	19
5.3.2.1. Key Exchange Protocols	19
5.4. Welke stack is het meest gepast bij deze opdracht?	24
5.4.1. T3 Stack	24
5.4.2. MERN Stack	29
5.4.3. PERN Stack	33
5.4.4. Serverless	34
OR 5.5. Hoe pas ik al bestaande cryptografische libraries toe?	37
5.5.1. Crypto-js	37
5.5.2. Bcrypt	38
5.5.3. Cryptr	39
6. Discussie	41
6.1. Beveiligingsmaatregelen	41
6.2. Sleutelbeheer	43
5.3. Stack	44
5.4. Libraries	46
7. Conclusie	47
8. Literatuurlijst	48

1. Inleiding

Tegenwoordig zijn er steeds meer bedrijven met een ISO-27001 certificering. Een van deze bedrijven is ook Webbio. Webbio werkt regelmatig samen met de overheid, waardoor het erg belangrijk is om de ISO maatregelen te volgen, zodat de gevoelige informatie veilig uitgewisseld wordt (*Informatiebeveiliging (ISO 27001) - Cybersecurity & Privacy - Digitale ethiek en veiligheid - ICT, z.d.*).

Alle gevoelige informatie wordt op dit moment nog per mail verstuurd waardoor het in verkeerde handen zou kunnen vallen. Dit wordt gezien als een groot bedrijfsrisico. Om dit probleem op te lossen heb ik van Webbio de opdracht gekregen om een systeem te ontwikkelen dat in staat is om gevoelige informatie op een veilige en betrouwbare manier te kunnen delen met externe ontvangers, terwijl de naleving van de strenge beveiligingsnormen en het behoud van de vertrouwelijkheid van de gegevens gewaarborgd blijven.

Echter voordat dit systeem gerealiseerd kan worden zal er eerst een uitgebreid onderzoek plaatsvinden naar de verschillende beveiligingsstandaarden om te kijken hoeveel maatregelen toegepast moeten worden om de applicatie zo veilig en efficiënt mogelijk te maken. Daarbij zal er ook worden verdiept in de verschillende encryptie mogelijkheden, de alternatieven hiervan, hoe een secret veilig verwerkt kan worden en de uiteindelijke toolstack die het meest geschikt is voor deze applicatie. Gebaseerd hierop zullen er verschillende onderzoeksvragen opgesteld worden, aangevuld met de motivatie, relevantie en onderzoeksmethode. De resultaten van deze onderzoeken zullen gedocumenteerd worden in een onderzoeksverslag.

2. Probleemstelling

In dit hoofdstuk wordt de huidige situatie uitgelegd, met wat de gewenste situatie is en de verschillen hiertussen.

2.1. Wat is de huidige situatie?

Op dit moment staat Webbio tot de uitdaging om gevoelige informatie (wachtwoorden, environment variables, databases, etc.) veilig te moeten versturen naar externe partijen buiten de organisatie. Deze informatie wordt meestal per mail verstuurd en werd voorheen al gezien als een bedrijfsrisico, omdat het bij de verkeerde personen kan belanden. Echter is dit risico alleen maar groter en inzichtelijker geworden sinds Webbio een ISO 27001-gecertificeerd bedrijf is geworden. Vandaar dat er belangstelling is om dit bedrijfsrisico op te lossen en volledig te kunnen vermijden in de toekomst.

2.2. Wat is de gewenste situatie?

De gewenste situatie is een nieuw systeem dat in staat is om gevoelige informatie op een veilige en betrouwbare manier te kunnen delen met externe ontvangers, terwijl de naleving van de strenge beveiligingsnormen en het behoud van de vertrouwelijkheid van de gegevens gewaarborgd blijven.

2.3. Wat is het verschil tussen de huidige en gewenste situatie?

Momenteel is er nog geen systeem dat de gevoelige informatie op een veilige en betrouwbare manier kan verzenden naar externe partijen. De enige mogelijkheden op dit moment zijn niet versleuteld en hebben andere beperkingen zoals een maximale bestandsgrootte of accepteren alleen specifieke bestandstype. Bij de gewenste situatie zal het versturen van gevoelige informatie een stuk meer geautomatiseerd worden. Daarnaast is het de bedoeling dat de gevoelige informatie maar eenmalig bekeken of gedownload kan worden. Echter kunnen we door middel van dit onderzoeksverslag bekijken hoe we een balans kunnen vinden tussen veiligheid en gebruiksvriendelijkheid.

3. Doelstelling

Het doel van dit onderzoeksverslag is om inzicht te krijgen over diverse beveiligingsstandaarden die toegepast kunnen worden op het uiteindelijke systeem. Hierbij wordt er zowel gekeken naar hoe veilig, als hoe efficiënt de applicatie hiermee gemaakt kan worden.

Met de resultaten van dit verslag kunnen er concrete keuzes gemaakt worden tijdens het ontwikkelen van de uiteindelijke op te leveren producten. Ook kan dit onderzoeksverslag de gemaakte keuzes beargumenteren

4. Vraagstelling

De bovenstaande doelstelling vormt de basis van onze hoofdvraag en om deze hoofdvraag te beantwoorden is deze onderverdeeld in meerdere deelvragen. Na het beantwoorden van deze deelvragen kunnen deze gebruikt worden ter onderbouwing voor het beantwoorden van onze hoofdvraag.

Alle vragen in onderzoeksverslag worden beantwoord door middel van de ICT Research Methods (Bonestroo et al., 2018).

4.1. Hoofdvraag

De hoofdvraag voor dit onderzoek is: "**Welke beveiligingsstandaarden moet ik toepassen om mijn applicatie veilig en tegelijkertijd efficiënt te maken?**".

4.2. Deelvragen

Hieronder wordt per hoofdvraag de motivatie, relevantie en onderzoeksmethoden gegeven.

4.2.1. Welke encryptie mogelijkheden zijn er?

4.2.1.1. Motivatie

Een van de wensen van de opdrachtgever is om alle gevoelige informatie te encrypten. Daarom wordt eerst onderzocht hoe efficiënt en veilig deze beveiligingsstandaard is.

4.2.1.2. Relevantie

Er zijn tegenwoordig veel verschillende encryptie methodes. Om te achterhalen welke het beste past bij deze opdracht zal er onderzoek worden gedaan naar welke encryptie methodes er bestaan en deze vergelijken met elkaar.

4.2.1.3. Onderzoeksmethode

Voor deze deelvraag wordt er gebruikgemaakt van de Literature Study onderzoeksmethode. Er is namelijk online al veel informatie te vinden over encryptie.

4.2.2. Wat kan naast encryptie nog meer toegepast worden?

4.2.2.1. Motivatie

Voor dit onderzoek zal er onderzoek gedaan worden naar andere beveiligingsstandaarden naast encryptie. Uit de onderzochte standaarden wordt een overzicht gemaakt voor de opdrachtgever zodat deze kan kiezen welke toegepast kan worden binnen de applicatie zelf.

4.2.2.2. Relevantie

De hoofdvraag vraagt om de meest veilige en efficiënte oplossing en dus zal er een goede balans gevonden moeten worden. Er zouden bijvoorbeeld wel tientallen beveiligingsstandaarden kunnen worden toegepast, maar zou dit dan nog wel gebruiksvriendelijk en dus efficiënt zijn? Daarnaast is het niet geheel zeker of encryptie alleen veilig genoeg is.

4.2.2.3. Onderzoeksmethode

Ook hiervoor zal er gebruikgemaakt worden van Literature Study aangezien er hoogstwaarschijnlijk al veel onderzoek gedaan is naar beveiligingsstandaarden.

4.2.3. Hoe kan een secret key het meest efficiënt en veilig gedeeld worden?

4.2.3.1. Motivatie

Naast encryptie is nog een wens van de opdrachtgever om ook gebruik te maken van secret keys. Voordat dit toegepast kan worden moet er uitgezocht worden wat precies deze secret keys inhouden en hoe deze veilig gedeeld kunnen worden zonder het risico te lopen dat deze secrets in de verkeerde handen vallen.

4.2.3.2. Relevantie

Om informatie veilig te kunnen encrypten en decrypten moet er gebruik worden gemaakt van secret keys, echter moet er onderzocht worden hoe deze secrets het veiligst en efficiënt mogelijk met de externe partij gedeeld kunnen worden.

4.2.3.3. Onderzoeksmethode

Om deze onderzoeksvraag te beantwoorden zal er gebruik worden gemaakt van Literature Study.

4.2.4. Welke stack is het meest gepast bij deze opdracht?

4.2.4.1. Motivatie

Uiteindelijk mag de stack naar eigen keuze zijn van de ontwikkelaar. En dus is het belangrijk om te bepalen welke stack het meest passend is voor het ontwikkelen van de applicatie. Daarom zullen er diverse onderzoeken gedaan worden naar de populaire stacks op dit moment.

4.2.4.2. Relevantie

Hedendaags blijven programmeertalen en frameworks continu vernieuwen. En deze worden steeds efficiënter, veiliger en betrouwbaarder. Vandaar dat het van belang is om een afweging te maken over welke stack het meest geschikt is voor de uiteindelijke applicatie.

4.2.4.3. Onderzoeksmethode

Hiervoor zal de Prototyping methode van de Workshop strategie toegepast worden. Dit houdt in dat er voor iedere stack een simpel prototype ontwikkeld zal worden waardoor de sterkte- en zwaktepunten naar voren komen.

4.2.5. Hoe pas ik al bestaande cryptografische libraries toe?

4.2.5.1. Motivatie

Nadat er een passende stack is gevonden zal er gekeken worden naar al bestaande cryptografische libraries. Per library wordt er een poging gedaan om het toe te passen om daarna te vergelijken welke het meest geschikt is.

4.2.5.2. Relevantie

Er zullen vast al verschillende libraries rond cryptografie bestaan, echter zullen ze niet allemaal passend zijn bij de requirements van een opdracht. En dus zal er gekeken worden naar de bruikbaarheid van de huidige bestaande libraries voor mijn afstudeeropdracht.

4.2.5.3. Onderzoeksmethode

Om hierachter te komen zal er gebruik worden gemaakt van zowel Literature Study als Prototyping. Eerst wordt er onderzoek gedaan naar diverse libraries en daarna worden deze in een prototype getest.

5. Resultaten

In dit hoofdstuk staan alle resultaten van de eerder genoemde onderzoeken.

5.1. Welke encryptie mogelijkheden zijn er?

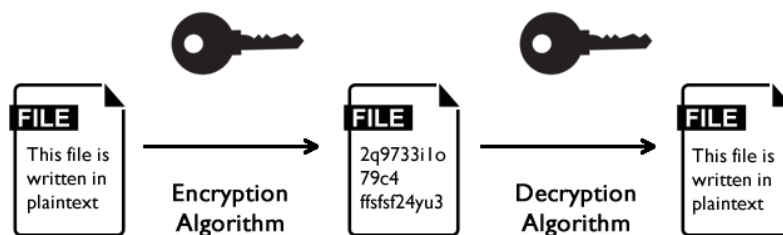
Voordat er in de verschillende encryptie mogelijkheden verdiept kan worden, is het eerst handig om überhaupt te begrijpen wat encryptie inhoudt en hoe encryptie precies werkt.

5.1.1. Wat is encryptie?

In het kort is encryptie een begrip uit de cryptografische wereld die staat voor het beveiligen van informatie of data door het gebruik van algoritmes. Deze algoritmes zorgen ervoor dat de inhoud zo vervormt wordt dat het alleen leesbaar is met een zogenaamde "secret key" (Google Cloud, z.d.). Je wilt encryptie toepassen om je data te beschermen tegen bijvoorbeeld malware of ongeautoriseerde gebruikers zoals hackers of cybercriminelen.

5.1.2. Hoe werkt encryptie?

Als we verder kijken naar deze algoritmes blijkt het dat encryptie zogenaamde leesbare tekst (plaintext) omzet in wartaal (ciphertext) door het gebruik van "cryptographic mathematical models" oftewel algoritmes (Cloudflare.com, 2023). Iedere algoritme is anders en gebruikt zijn eigen complexe wiskundige berekeningen, toch hebben zij allemaal een unieke decryption key nodig om de ciphertext weer terug te zetten naar plaintext (zie afbeelding 1). Ook is deze sleutel niet zomaar te kraken, want sleutels kunnen wel uit honderden, of zelfs duizenden karakters bestaan. Deze karakters worden door de computer gelezen in binary en noemen wij ook wel "bits". De sterkte van encryptie is afhankelijk van het algoritme en de complexiteit van de decryption key.



Afbeelding 1: Voorbeeld versleuteling van plaintext

5.1.3. Encryptie types

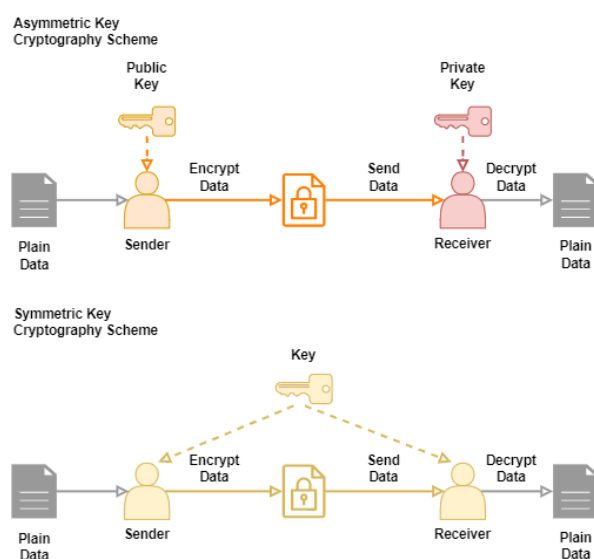
Er zijn diverse typen algoritmes, de meest voorkomende zijn: symmetrische en asymmetrische cryptografie (Marget, 2022).

5.1.3.1. Symmetrisch

Symmetrische encryptie, of ook bekend als "secret key encryption", gebruikt dezelfde sleutel voor zowel het encrypten en decrypten van de inhoud. Omdat er maar één sleutel wordt gebruikt is het coderen van de plaintext sneller en vandaar dat dit meestal wordt gebruikt tijdens het overdragen van grote hoeveelheden data. Echter loop je met symmetrische encryptie een risico dat zodra de secret key in de verkeerde handen valt, diegene ook direct toegang heeft tot de data.

5.1.3.2. Asymmetrisch

Asymmetrische encryptie, of ook bekend als "public-key encryption", gebruikt een combinatie van sleutels waarbij je een "public key" hebt voor encryptie en een "private key" hebt voor decryptie. De openbare sleutel kan met iedereen gedeeld worden om te encrypten, maar alleen de besloten sleutel kan de ciphertext daadwerkelijk lezen. Dit proces heeft meer computerkracht nodig en wordt daarom voornamelijk gebruikt voor het verzenden van kleinere data. Hierbij kan voorafgaand een veilig communicatiekanaal tot stand gebracht worden aangezien de private key dient als een soort authenticatie.



Afbeelding 2: Symmetrisch vs asymmetrische encryptie

5.1.4. Encryptie algoritmes

Onderstaand zijn per encryptie type een aantal populaire algoritmes onderzocht en vergeleken met ieders eigen voor- en nadelen.

5.1.4.1. AES (Advanced Encryption Standard)

AES is op dit moment een van de meest gebruikte algoritme dat symmetrisch encrypt (GeeksforGeeks, 2023). Het staat voornamelijk bekend om zijn veiligheid en efficiëntie. Vandaar dat het hedendaags door veel verschillende applicaties wordt gebruikt zoals voor data versleuteling, netwerkbeveiliging en beveiligde communicatie.

Een van de sterke punten met AES is dat het in drie verschillende sleutellengten kan komen: 128 bits, 192 bits en 256 bits. Een langere sleutel maakt het extreem rekenintensief voor een computer om het te kraken waardoor het bestendig is tegen de meest voorkomende cryptografische aanvallen, zoals brute-force, known-plaintext en chosen-plaintext (Cryptography Attacks: 6 Types & Prevention, 2022). Echter heeft dit algoritme nog steeds het nadeel van een symmetrische encryptie, waardoor de grootste uitdaging is om de sleutel veilig te kunnen delen met externe partijen. Hierdoor kan AES zelf geen authenticatie toepassen en moeten er aanvullende technieken worden geïmplementeerd als een beveiligde communicatie gewenst is.

5.1.4.2. DES (Data Encryption Standard)

Nog een ander symmetrisch algoritme is DES (Loshin & Cobb, 2021). Dit was vroeger een erg populaire algoritme, omdat het door de Verenigde Staten in 1977 werd goedgekeurd als de federale standaard. Sindsdien werd het door veel overheidsinstellingen gebruikt voor de beveiliging van gevoelige informatie.

Destijds was het een van de efficiëntste algoritmes voor de rekencapaciteit van toen. DES heeft in principe de deur geopend voor alle moderne cryptografie. Helaas is het grootste zwak punt dat het tegenwoordig niet meer aangeraden wordt, omdat de sleutellengte maar 56 bits is. Met deze korte lengte is het erg kwetsbaar tegen brute-force aanvallen aangezien de rekencapaciteit van computers nu veel sterker is.

5.1.4.3. RSA (Rivest-Shamir-Adleman)

RSA staat voor Rivest-Shamir-Adleman, dit zijn de achternamen van de drie uitvinders van dit algoritme (*Wickramasinghe, 2023*). Het is een veel gebruikte asymmetrisch algoritme waardoor het, zoals al eerder benoemd, met een public en private key werkt. Deze public key infrastructuur samen met de mogelijkheid tot een lange sleutellengte levert een erg sterke beveiliging op.

Vandaar dat RSA veel gebruikt wordt voor het creëren van digitale handtekeningen (*CISA, 2021*). Dit zijn net zoals handtekeningen op papier om de authenticiteit en integriteit van bijvoorbeeld digitale documenten en transacties te kunnen waarborgen. Behalve digitale handtekeningen wordt het algoritme ook gebruikt voor het beveiligen van mails, berichten en data. Echter heeft het ook een aantal zwakke punten, zoals dat de beveiliging van RSA erg afhankelijk is van de sleutellengte. Een langere lengte is veiliger, maar kost dan weer meer computerkracht en dit wordt nog intensiever met asymmetrische encryptie aangezien er meerdere sleutels moeten worden gecodeerd en gedecodeerd.

Daarnaast bestaan er zogenaamde "block ciphers" (*Sangfor Technologies, 2022*) die gebaseerd zijn op de hoeveelheid lengte data die het algoritme tegelijk kan verwerken. In het geval bij RSA is deze block cipher erg gelimiteerd en wordt er ook niet aangeraden om dit algoritme toe te passen op een grote hoeveelheid data.

5.1.4.4. ECC (Elliptic Curve Cryptography)

ECC is een modern algoritme dat asymmetrisch encrypt (*Sullivan, 2022*). Het algoritme is gebaseerd op elliptische krommen en heeft hierdoor zijn naam gekregen. Elliptische krommen is een term in de wiskunde en wordt gebruikt om meetkundig een optelling te kunnen definiëren. Het levert een sterke beveiliging op terwijl de sleutellengte relatief kort is.

Als we bijvoorbeeld ECC vergelijken met RSA heeft ECC maar een sleutellengte van 256 bits nodig om dezelfde beveiliging te hebben waarvoor RSA een sleutellengte van 3072 bits nodig heeft. Een sterk punt aan deze sleutellengte is dat het minder computerkracht nodig heeft. Toch is het erg schaalbaar, want wanneer de encryptie nog veiliger moet worden, kan de sleutellengte simpelweg verhoogd worden. Een zwaktepunt van ECC is dat het toepassen van dit algoritme erg uitdagend is,

aangezien het veel kennis eist over elliptische krommen en de betreffende parameters. Dit is ook een aanleiding dat veel systemen dit algoritme nog niet kunnen ondersteunen wegens gebrek aan compatibiliteit.

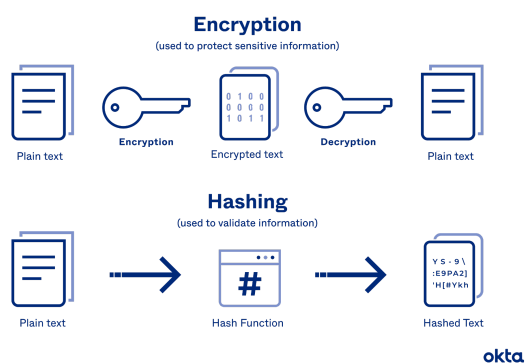
5.2. Wat kan naast encryptie nog meer toegepast worden?

Uiteraard zijn er nog andere methodes in de cryptografische wereld die diverse beveiligingsstandaarden met zich meebrengen. Hieronder zullen de bevindingen benoemd worden met een aantal alternatieven die toegepast zouden kunnen worden.

5.2.1. Hashing

Hashing is deels vergelijkbaar met encryptie aangezien het beide gegevens omzet in unieke tekenreeksen (Kok & Kok, 2023). Tijdens het hashen worden ook specifieke algoritmes toegepast. Echter wordt hierbij de plaintext omgezet in een zogenaamde "hash-value" of "hash". Deze uitvoer is altijd dezelfde lengte, ongeacht de lengte van de invoer. Toch heeft hashing andere eigenschappen waardoor het niet geheel identiek is aan encryptie (zie afbeelding 3).

Hashing is namelijk onomkeerbaar, dit houdt in dat zodra de inhoud veranderd is in een hash het ook niet meer ontcijferd kan worden. Ook is het deterministisch, hiermee wordt bedoeld dat dezelfde invoer van plaintext altijd dezelfde hash-waarde teruggeeft. Ondanks hierdoor is hashing veilig genoeg dat het niet rekenkundig reverse engineered kan worden. Hashing wordt hierdoor veel gebruikt voor het verifiëren van wachtwoorden, integriteit controleren van bestanden en om unieke identificaties te maken voor data.



Afbeelding 3: Encryption vs Hashing

5.2.2. Obfuscation

Obfuscation verwijst naar de techniek om iets opzettelijk onduidelijk, obscuur of moeilijk begrijpbaar te maken (*Hofheinz et al., z.d.*). Deze techniek wordt meestal gebruikt om gevoelige informatie, intellectuele eigendommen of code te beschermen voor onbevoegden.

Tegenwoordig wordt obfuscation voornamelijk toegepast in software beveiliging, cybersecurity of cryptografie. In het geval van cryptografie wordt obfuscation meestal toegepast op de algoritmes zelf. Hierdoor kan het moeilijker worden voor externe partij om te begrijpen wat er precies in het algoritme gebeurt en dit voorkomt reverse engineering of andere aanvallen.

Obfuscation is niet gebaseerd op complexe wiskundige algoritmes zoals bij encryptie, in plaats daarvan hernoemen ze bijvoorbeeld variabelen, voegen overbodige code toe of veranderen data structuren (*zie afbeelding 4*). Vandaar dat obfuscation als een waardevol hulpmiddel gezien kan worden. Het uiteindelijke doel hiervan is om het zo moeilijk mogelijk te maken om de inhoud te begrijpen, pogingen hiervan te vertragen of zelfs geheel af te schrikken.

Het wordt erg aangeraden om in combinatie te gebruiken met andere beveiligingsstandaarden en mag absoluut niet het gebruik van vertrouwde cryptografische algoritmen vervangen.

Original Source Code Before Control Obfuscation	Reverse-Engineered Source Code After Control Flow Obfuscation
<pre>public int CompareTo (Object o) { int n = occurrences - ((WordOccurrence)o).occurrences; if (n=0) { n=String.Compare (word, ((WordOccurrence)o).word); } return (n); }</pre>	<pre>private virtual int_a(Object A+0) { int local0; int local1; local 10 =this.a - (c) A_0.a; if (local10 !=0) goto i0; while (true) { return local1; } i1:local10= System.String.Compare(this.b, (c) A_0.b); goto i0; }</pre>

Afbeelding 4: Voorbeeld van obfuscated code

5.2.3. Masking

Masking wordt af en toe verward met obfuscation. Het is vergelijkbaar aangezien de inhoud bij data masking ook veranderd wordt met andere karakters of data, echter wordt masking meestal gebruikt voor andere doeleinden zoals testen met mock data. Hierbij wordt gevoelige informatie namelijk (deels) vervangen door neppe of anonieme informatie (Burchfiel, 2023).

Dit is handig voor een organisatie, omdat zij de masked gegevens alsnog kunnen gebruiken en delen zonder dat dit moeilijker te begrijpen wordt of dat er risico is om gevoelige informatie bloot te stellen aan ongeautoriseerde personen. Het is bijvoorbeeld ideaal wanneer er gewerkt wordt met gegevens zoals: namen, adressen, creditcardnummers, bsn nummers, telefoonnummers, etc.

Er bestaan diverse technieken om masking toe te passen (Cobb, 2022):

- **Substitution:** Hierbij wordt de inhoud vervangen met willekeurige waarden.
- **Permutation / Shuffling:** Hierbij wordt de volgorde van een waarde in een willekeurige volgorde gezet.
- **Scrambling:** Dit is hetzelfde als shuffling, maar wordt alleen toegepast op alfanumerieke tekens.
- **Masking Out:** Een deel van de waarde wordt alleen getoond.
- **Nullifying:** Vervangt de gehele waarde met een null waarde.
- **Date Shift:** Verandert datums door ze naar voren of achter te verplaatsen met een specifiek aantal dagen, maanden of jaren.
- **Variance:** Verandert financiële nummers of datums met een willekeurig percentage.

	Original production database		Development database with masked data
PATIENT NUMBER	113355	SCRAMBLING	100100
NAME	John West	SHUFFLING	Peter Church
ADDRESS	45 Broad Street	SUBSTITUTION	12 Johnson Square
CITY/STATE/ZIP	Sunnyview, CA 90261		Rochdale, CA 91331
SSN	778-62-8144		805-14-1893
DOB	10/07/1972	VARIANCE	02/18/1975
CREDIT CARD NUMBER	4145 1230 0000 0062	MASKING OUT	XXXX XXXX XXXX 0062
FILE	mri_results.pdf	NULLIFYING	NULL

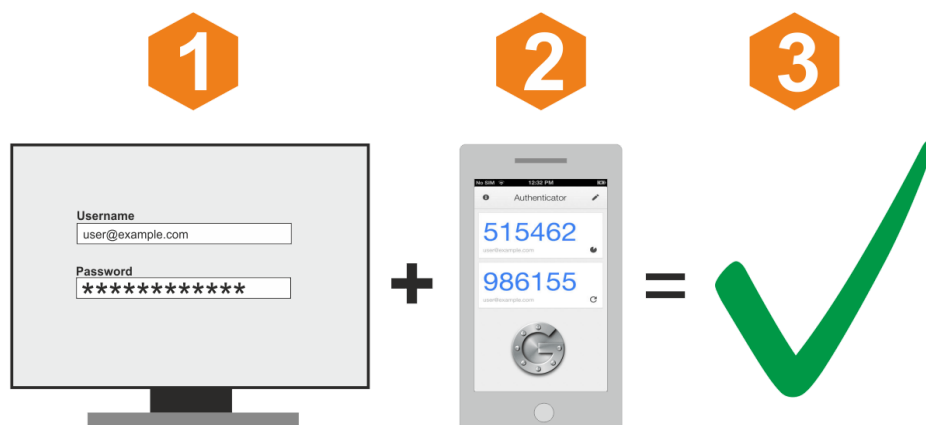
Afbeelding 5: Voorbeeld van masking technieken

5.2.4. Two-Factor Authentication (2FA)

Two-factor authentication, oftewel 2FA, is een beveiligingsstandaard die meestal gebruikt wordt voor het beveiligen van online accounts of digitale diensten (Microsoft Security, z.d.). Hierbij is het verplicht om twee vormen van identificatie te voltooien voordat er toegang gegeven kan worden. Het doel van 2FA is om het moeilijker te maken voor onbevoegde personen om toegang te krijgen tot bijvoorbeeld je account, zelfs al hebben ze toegang tot je wachtwoord.

Two-factor authenticatie werkt als volgt:

- **Eerste factor:** dit is hoogstwaarschijnlijk iets wat je al weet, zoals je gebruikersnaam en wachtwoord combinatie. Deze kun je simpelweg als eerst invullen tijdens het inloggen.
- **Tweede factor:** daarna komt de tweede factor, dit is iets unieks aan jou dat geheel los staat van jouw wachtwoord. Dit kan onder de volgende categorieën onderscheiden worden:
 - a) **Iets dat je hebt:** denk aan een SMS met eenmalige code, authenticator app zoals Google Authenticator (zie afbeelding 6) of hardware tokens die gegenereerd worden door een fysiek toestel zoals een YubiKey.
 - b) **Iets dat je bent:** denk aan biometrische informatie zoals vingerafdrukken of gezichtsherkenning.



Afbeelding 6: 2FA voorbeeld met authenticatie app

Af en toe wordt Two-Factor Authentication verward met Multi-Factor Authentication (MFA). Het grootste verschil is dat MFA een stuk uitgebreider en flexibeler kan zijn dan 2FA (AWS, z.d.). Het heeft namelijk de mogelijkheid om op meer dan twee factoren te authenticeren. Deze factoren kunnen nog meer criteria bevatten zoals je geolocatie of zelfs je gedragspatronen.

5.2.5. Access Control

Access Control wordt gebruikt om te controleren, beheren en reguleren wie toegang heeft tot systemen, data, middelen, etc, binnen een organisatie of computersysteem (Citrix, 2023). Het is namelijk een cruciaal onderdeel van informatiebeveiliging en speelt daarbij ook een belangrijke rol bij het beschermen van gevoelige informatie, het bewaken van integriteit en het garanderen dat alleen geautoriseerde personen toegang hebben tot specifieke bronnen.

Binnen Access Control zijn er vier verschillende types:

- **Attribute-Based Access Control (ABAC):** ABAC gebruikt attributen (zoals informatie van gebruikers, resources of de omgeving) om de toegangscontrole te beslissen. Hiermee heb je de controle en flexibiliteit.
- **Discretionary Access Control (DAC):** Binnen DAC beslist de eigenaar of administrator wie specifiek toegang heeft tot wat.
- **Mandatory Access Control (MAC):** Bij MAC wordt de toegangscontrole afgedwongen gebaseerd op strikte beveiligingslabels, zoals classificatie of beveiligingsniveau. Dit wordt vaak gebruikt binnen overheids- en militaire omgevingen.
- **Role-Based Access Control (RBAC):** Ten slotte is er RBAC, hiermee wordt toegang gegeven op basis van de bedrijfsfunctie van een gebruiker in plaats van zijn identiteit. Hiermee hebben gebruikers alleen toegang tot data die ook relevant is voor hun rol in de betreffende organisatie.

5.3. Hoe kan een secret key het meest efficiënt en veilig gedeeld worden?

In het geval dat er gevoelige informatie is opgeslagen moet het nog gedeeld worden met de externe partijen. Bij deze deelvraag zal er eerst kort worden herhaald wat secret keys precies zijn en daarna zal er onderzocht worden hoe deze veilig gedeeld kunnen worden met diverse prototypen.

5.3.1. Wat is een secret key?

Zoals al eerder benoemd in hoofdstuk 5.1. kunnen secret keys in verschillende vormen komen. Bij symmetrische algoritmes is dit dezelfde sleutel om de inhoud te encrypten of decrypten, terwijl bij asymmetrische algoritmes er een public- en private key bestaat. Zonder deze sleutels kan men geen data omzetten van plaintext naar ciphertext of andersom.

5.3.2. Delen van secret keys

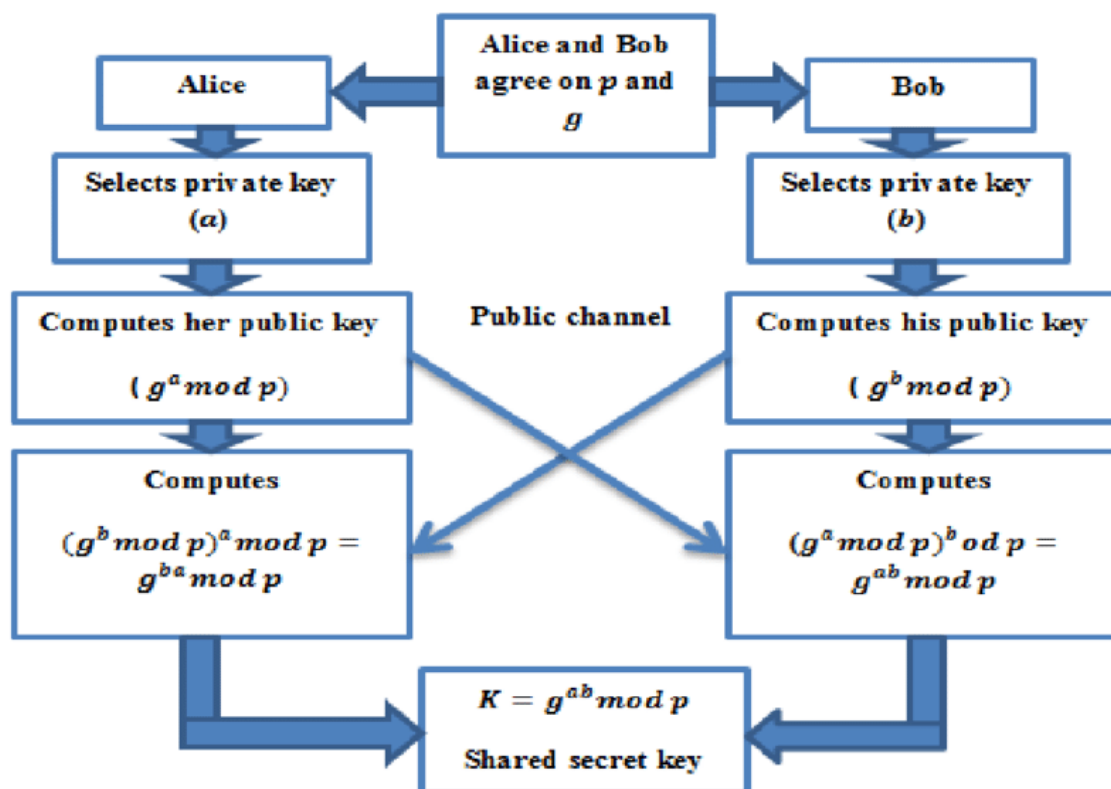
Het is van uiterste belang dat deze sleutels veilig gedeeld worden en niet gecompromitteerd worden. Daarom zijn er veel verschillende technieken bedacht om veilig en efficiënt deze secrets te kunnen delen, zoals:

5.3.2.1. Key Exchange Protocols

Een key exchange protocol, vaak ook wel gewoon key exchange genoemd, is een cryptografische methode om veilig een gedeelde secret key samen te stellen tussen twee of meer partijen (RnD, 2023). Deze gedeelde secret key bestaat uit diverse public keys en wordt meestal samengesteld door wiskundige algoritmes.

Een van de meest bekende key exchange protocols is het Diffie-Hellman Algorithm. Dit is een algoritme dat specifiek ontwikkeld is voor symmetrische encryptie, maar gebaseerd is op asymmetrische encryptie (*What is the Diffie-Hellman (DH) algorithm? / Security Encyclopedia, z.d.-b*). Hierbij hoeft de daadwerkelijke secret key nooit gedeeld te worden, in plaats daarvan genereren beide partijen een public- & private key. Deze public key kunnen zij delen met elkaar, terwijl ieders eigen private key geheim blijft. Ook spreken ze een aantal parameters met elkaar af, meestal zijn dit priemgetallen of wiskundige operaties.

Hiermee kunnen de partijen met hun eigen keys, de public key van de andere partij en de afgesproken parameters een gedeelde secret key berekenen (zie afbeelding 7). Zodra deze gedeelde sleutel vastgesteld is, kunnen deze partijen het gebruiken voor symmetrische encryptie. In principe gebruikt het Diffie-Hellman key exchange protocol zowel de sterktepunten van symmetrische als asymmetrische cryptografie om een veilige communicatie op te leveren.



Afbeelding 7: Voorbeeld van Diffie-Hellman Algorithm

5.3.2.2. Key Splitting

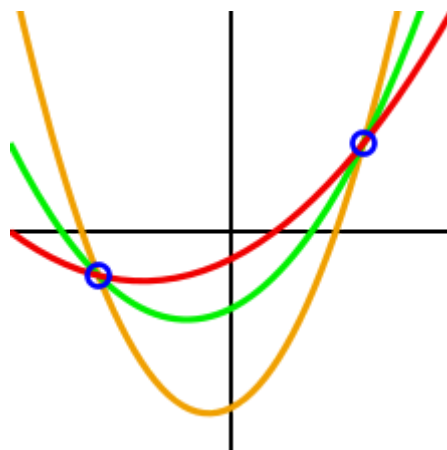
Key splitting, of ook al bekend als secret sharing of secret splitting, is een techniek in de cryptografische wereld om een secret op te splitsen in meerdere delen (*What is key splitting?* / *Security Encyclopedia*, z.d.). Deze gescheiden delen worden daarna verspreid naar diverse partijen of individuen. De oorspronkelijke secret kan daarna alleen nog maar gereconstrueerd worden indien er een specifiek aantal van deze delen gecombineerd wordt. De reconstructie hiervan wordt meestal gedaan door wiskundige berekeningen.

Door key splitting toe te passen verbetert de beveiliging. Indien een deel van de secret in de verkeerde handen valt, zal de oorspronkelijke secret nog geheim blijven.

Ook kan er een sterk algoritme toegepast worden zodat het moeilijker wordt om de oorspronkelijke secret te kraken nadat een deel van de secret bekend is gemaakt.

Een van de bekendste algoritmes waarmee key splitting wordt toegepast is het zogenaamde "Shamir's Secret Sharing". Dit algoritme is gebaseerd op polynomiale interpolatie, oftewel het is een methode waarbij je een onbekende waarde kan berekenen met minimaal twee bekende datapunten (*Technologies, 2021*).

In iets simpele termen, stel je voor dat je een grafiek hebt met een lijn die een deel van de secret moet representeren. Met alleen één datapunt weet je niet waar deze lijn heen gaat, hetzelfde geldt voor de secret, met maar een deel weet je niet de gehele secret. Hiervoor moet je dus nog een datapunt weten om de gehele lijn correct te kunnen visualiseren. In het geval van Shamir's Secret Sharing zullen het geen rechte lijnen zijn, maar polynomen (*zie afbeelding 8*).



Afbeelding 8: Polynoom met twee datapunten

Het voordeel van dit algoritme is dat het erg uitbreidbaar is (*Nugent, 2022*). In het geval dat de secret in meerdere delen wordt opgesplitst kan er simpelweg nog een datapunt aan de polynoom worden toegevoegd. Het biedt dus een flexibel en krachtig mechanisme voor het beveiligen van gevoelige informatie in verschillende toepassingen.

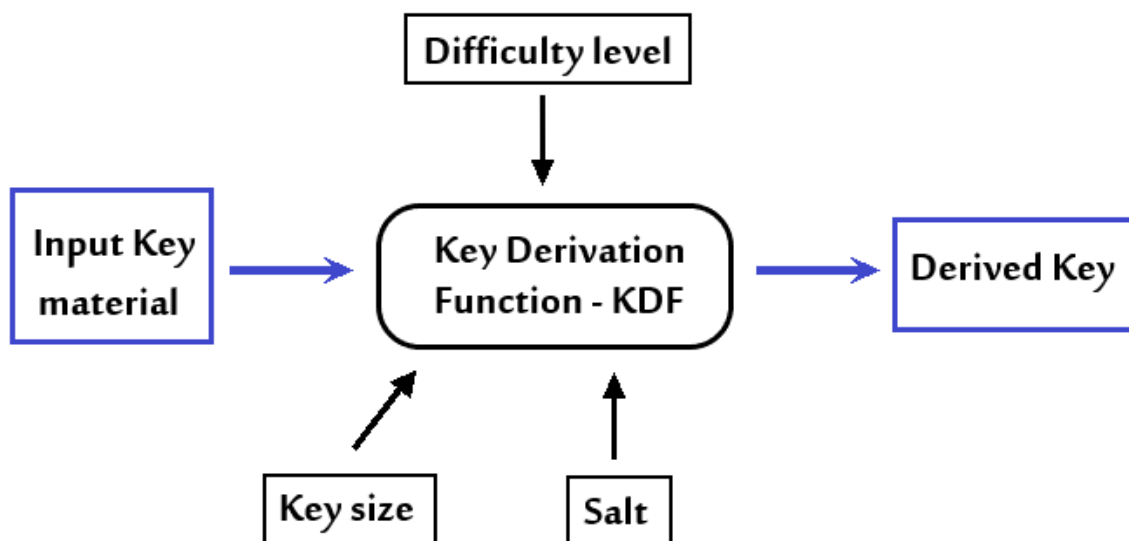
5.3.2.3. Key Derivation Function

Key Derivation Function, of KDF, is een algoritme waarbij er een of meerdere cryptografische sleutels afgeleid worden uit een initiële sleutel (*Wikipedia contributors, 2023*). Het uiteindelijke doel hierbij is om de bestaande sleutel te

transformeren en uit te breiden, zodat iedere sleutel gebruikt kan worden voor specifieke cryptografische doeleinden.

Een aantal belangrijke punten om te weten bij KDF:

- **Master Key:** De oorspronkelijke sleutel waarop de afgeleide sleutels op gebaseerd zijn heet de "master key".
- **Salts & Parameters:** Meestal accepteert het algoritme ook nog extra variabelen zoals salts of parameters om de nieuwe sleutels unieker te maken.
- **Voorspelbaarheid:** Het is mogelijk om de voorspelbaarheid aan te passen binnen KDF. Het kan zoals bij hashing altijd dezelfde uitvoer hebben wanneer de invoer hetzelfde is of het kan ook onvoorspelbare uitvoer genereren.
- **Flexibiliteit:** Ook is het erg veelzijdig en aanpasbaar, het ondersteunt namelijk verschillende invoerformaten, sleutellengtes en zelfs configuraties om te voldoen aan de beveiligingsvereisten van verschillende cryptografische systemen en protocollen.



Afbeelding 9: Key Derivation Function

5.3.2.4. Password-Authenticated Key Agreement

Password-Authenticated Key Agreement (PAKE) is een cryptografisch protocol waarmee twee partijen, meestal een client en een server, een gedeelde secret key kunnen vaststellen via een onveilig communicatiekanaal (Green, 2020). Dit wordt gedaan op basis van een gemeenschappelijk wachtwoord. Het wachtwoord zelf hoeft hierbij niet blootgesteld te worden. De PAKE-protocol is ontworpen om een

veilige secret key overeenkomst te waarborgen en tegelijkertijd bescherming te bieden tegen afluisteren of zogenaamde offline dictionary attacks (brute-forcing met veel voorkomende wachtwoorden).

Daarnaast is het belangrijk om te weten dat er twee vormen PAKE-protocollen zijn (Hao & Van Oorschot, 2022):

- **Balanced:** De eerste is balanced, hierbij gaat PAKE ervan uit dat beide partijen hetzelfde wachtwoord gebruiken om de gedeelde secret key te authenticeren.
- **Augmented:** Daarnaast is er augmented, dit is een variatie op de eerder benoemde client en server relatie. Hierbij slaat de server geen data op dat gerelateerd is aan de gemeenschappelijke wachtwoorden. Dit levert een verbeterde beveiliging op, terwijl de bruikbaarheid en efficiëntie behouden blijven.

Elk van deze vormen heeft diverse methoden om PAKE toe te passen, echter zijn de meeste gebaseerd op de eerder benoemde Diffie-Hellman Algorithm.

5.4. Welke stack is het meest gepast bij deze opdracht?

Bij het zoeken naar een geschikte stack voor deze afstudeeropdracht is er voornamelijk gekeken naar factoren zoals de functionaliteiten, hoe up-to-date het is en of Webbio er al ervaring mee heeft om genoeg ondersteuning te kunnen bieden. Vanwege het laatste punt vallen direct al frameworks met talen zoals .NET, PHP, Ruby en Python af.

Bij Webbio worden de applicaties tegenwoordig namelijk met TypeScript en Next.js gemaakt. Daarbij heeft het team van Webbio ook ervaring met JavaScript en React aangezien TypeScript en Next.js hierop gebaseerd zijn. Door gebruik te maken van een framework dat Webbio zelf ook specialist in is kunnen zij de applicatie onderhouden nadat de stageperiode is afgelopen.

Tijdens dit onderzoek is er verdiept in de volgende stacks:

5.4.1. T3 Stack

Allereerst, de T3 Stack, dit is een erg moderne web development stack waarbij de nadruk wordt gelegd op eenvoudigheid, modulariteit en full-stack type safety (*Create T3 App, z.d.*). Deze stack maakt gebruik van de volgende libraries:

- **Next.js:** Een populaire open-source framework gebaseerd op React om webapplicaties te ontwikkelen. Het vereenvoudigt webdevelopment met functionaliteiten zoals dynamische server-side rendering, automatische code splitting en simpele file based routing, waardoor developers snel en eenvoudig een website of applicatie kunnen bouwen (*Next.js by Vercel - the React framework, 2023*).
- **TypeScript:** Een open-source high-level programmeertaal ontwikkeld door Microsoft (*JavaScript with syntax for types., z.d.*). TypeScript is een superset van JavaScript met extra functionaliteiten, een van de meest opmerkelijke is het type systeem dat hiermee toegevoegd wordt.
- **Tailwind CSS:** Dit is een CSS framework dat ontworpen is om het coderen te versnellen door middel van zijn utility-first fundamentals. Hierdoor hoeft er

geen custom CSS geschreven te worden en kunnen utility-classes direct op de HTML elementen toegepast worden (*Tailwind CSS, 2023*).

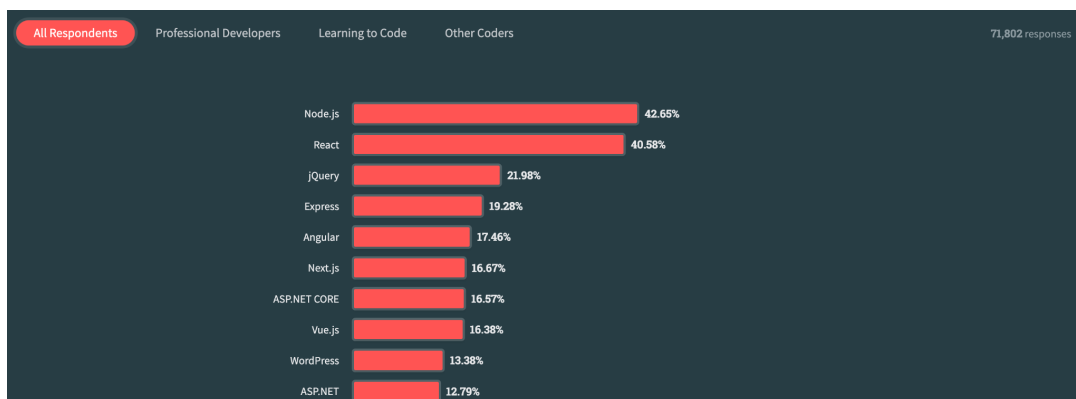
- **Prisma:** Een Node.js & Typescript Object-Relational-Mapping (ORM) tool gemaakt om database toegang en beheer eenvoudiger te maken (*Prisma, z.d.*).
- **tRPC:** TypeScript Remote Procedure Call, oftewel tRPC, is een library waarmee je een API tot stand kan brengen die net zoals in TypeScript type-safety garandeert tussen de API en de client (*TRPC, 2001*).
- **NextAuth.js:** Daarnaast maakt deze stack nog gebruik van NextAuth, dit is een open-source authenticatie library voor Next.js applicaties. Het biedt een uitbreidbare en eenvoudig te gebruiken authenticatie oplossing en ondersteunt daarom ook veel authenticatie methoden zoals OAuth, passwordless email authenticatie, JSON Web Tokens, Active Directory en meer (*Collins & NextAuth.js, 2023*). OAuth ondersteunt diverse providers zoals Google, een van de requirements van deze applicatie.

Omdat deze stack zoveel libraries bevat, betekent niet dat je er ook gebruik van hoeft te maken. Zoals al eerder benoemd is het erg modulair en dit is daarom een pluspunt van de T3 Stack.

Daarnaast zijn nog een aantal pluspunten dat de learning curve vrij steil is, want veel libraries zijn simpelweg uitbreidingen van andere frameworks die ik tijdens mijn studie op de HAN heb benut. In het geval dat ik vast kom te zitten kan ik altijd terecht bij de actieve community binnen Next.js, TypeScript, etc.

Ook zijn alle technologieën binnen deze stack zeer geschikt voor zowel kleinschalige als grote projecten. En deze technologieën worden ook nog goed onderhouden.

Ten slotte wordt deze stack steeds populairder aangezien het gebruik maakt van moderne libraries, neem bijvoorbeeld een kijkje naar Next.js, uit een vragenlijst georganiseerd door Stack Overflow (*Moretti, 2023*), die als doelgroep zowel professionele als beginnende ontwikkelaars had, is gebleken dat de populariteit van Next.js dit jaar gestegen is van #11 naar #6 (*zie afbeelding 10*). Daarnaast staat het ook in de top drie van meest gewenste frameworks van het jaar.



Afbeelding 10: Top 10 meest populaire web frameworks & technologieën

Nadat er een goed beeld is gecreëerd over wat T3 Stack te bieden heeft, is er een poging gedaan om een daadwerkelijk prototype te bouwen. Het doel hierbij was om alle zes de libraries te verwerken om erachter te komen hoe uitdagend het is om een simpele applicatie te ontwikkelen.

Het creëren van een T3 applicatie ging heel makkelijk. Met een simpele NPM command in de Terminal kon ik een boilerplate code opzetten met de packages naar keuze (zie afbeelding 11).

```
npm create t3-app@latest

/ _ | _ \ _ | / _ \ _ | _ | _ | _ | / _ \ _ | _ | _ | _ |
| ( _ | / _ | / _ \ | | _ | _ | _ | _ | _ | _ | / _ \ _ |
\ _ | _ | _ | / _ \ _ | _ | _ | _ | _ | / _ \ _ | _ |

? What will your project be called? (my-t3-app) my-t3-app
? Will you be using JavaScript or TypeScript? TypeScript
Good choice! Using TypeScript!
? Which packages would you like to enable? nextAuth, prisma, tailwind, trpc
? Initialize a new git repository? (Y/n) No
Sounds good! You can come back and run git init later.
? Would you like us to run npm install? (Y/n) Yes
Alright. We'll install the dependencies for you!
```

Afbeelding 11: Installatie nieuwe T3 applicatie

De boilerplate code bevat al heel veel bestanden die geheel geconfigureerd zijn om met elkaar te kunnen werken (zie afbeelding 12). Ook was er een uitgebreide documentatie die uitlegde waarvoor elke file directory voor diende.

Afbeelding 12: File Structure

```
.
├── public
│   └── favicon.ico
├── prisma
│   └── schema.prisma
├── src
│   ├── env.mjs
│   ├── pages
│   │   ├── _app.tsx
│   │   └── api
│   │       ├── auth
│   │       │   └── [...nextauth].ts
│   │       └── trpc
│   │           └── [trpc].ts
│   └── index.tsx
├── server
│   ├── auth.ts
│   ├── db.ts
│   └── api
│       ├── routers
│       │   └── example.ts
│       ├── trpc.ts
│       └── root.ts
├── styles
│   └── globals.css
├── utils
│   └── api.ts
├── .env
├── .env.example
├── .eslintrc.cjs
├── .gitignore
├── next-env.d.ts
├── next.config.mjs
├── package.json
├── postcss.config.cjs
├── prettier.config.mjs
├── README.md
├── tailwind.config.ts
└── tsconfig.json
```

Nadat de installatie voltooid was, kon er een begin worden gemaakt om de daadwerkelijke applicatie te bouwen. Als prototype hiervoor had ik het idee om een soort van simpele Twitter clone te maken. Hiermee zou ik dan een gebruiker eerst laten authenticeren en daarna berichten laten schrijven en lezen via een API.

De authenticatie binnen NextAuth heb ik gedaan met Google, omdat ik bij de uiteindelijke afstudeeropdracht ook een Google SSO moet gebruiken. Dit was vrij eenvoudig, ik hoefde alleen te specificeren welke authenticatie service ik gebruikte en daarna had ik toegang tot de gegevens van de ingelogde gebruiker (zie afbeelding 13).

```
import NextAuth from "next-auth"
import GoogleProvider from "next-auth/providers/google";
import { env } from "../../env.mjs";

export const authOptions = {
  providers: [
    GoogleProvider({
      clientId: env.GOOGLE_CLIENT_ID,
      clientSecret: env.GOOGLE_CLIENT_SECRET,
    }),
  ],
}
export default NextAuth(authOptions)
```

Afbeelding 13: NextAuth code

Daarna met behulp van de gegenereerde boilerplate code was het vrij eenvoudig om een API endpoint te schrijven en om ermee te communiceren (zie afbeelding 14).

Natuurlijk moet de data ergens vandaan gehaald worden. Hiervoor moest ik alleen maar een schema aanpassen in een Prisma bestand. Ook dit was al geheel geconfigureerd tijdens de installatie. Ook ontdekte ik dat Prisma een hele simpele ingebouwde database visual editor heeft genaamd Prisma Studio.

```

export const postsRouter = createTRPCRouter({
  getAll: publicProcedure.query(async ({ ctx }) => {
    const posts = await ctx.db.post.findMany({
      take: 100,
      orderBy: [{ createdAt: "desc" }],
    });








    return posts;
  }),
});

const { data } = api.posts.getAll.useQuery();

```

Afbeelding 14: Simpele GET endpoint

Nadat ik de back-end heb opgezet was het tijd voor de front-end. Dit was goed te doen, omdat er veel documentatie en voorbeelden over Next.js met TailwindCSS te vinden waren. Uiteindelijk heb ik de volgende pagina ontwikkeld waarop een gebruiker kan inloggen om berichten te plaatsen en daarnaast ook berichten van andere gebruikers kan lezen (zie *afbeelding 15*).

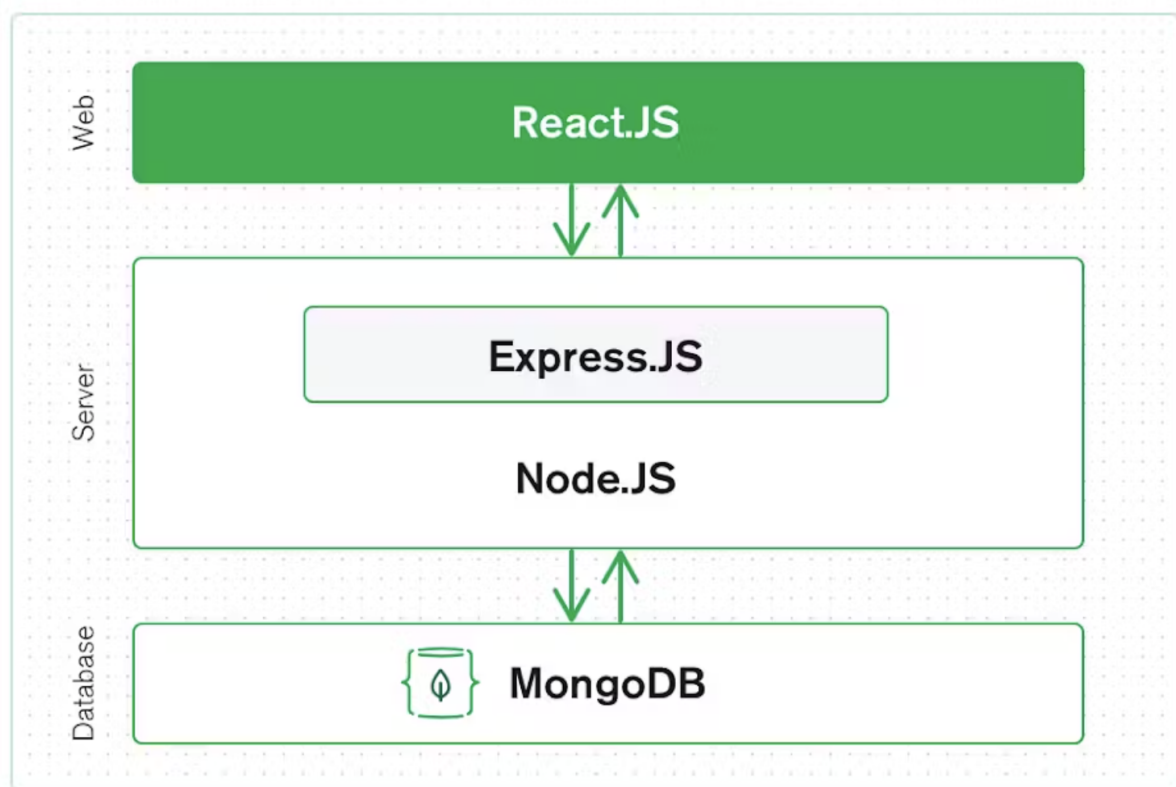
	<div>Sign Out</div> <div>  <input type="text" value="Type your message..."/> <div>Send</div> </div>
	<div>  <div> @Alex Cheng • a few seconds ago hoi </div> </div>
	<div>  <div> @Alex Cheng • a few seconds ago dasdasd </div> </div>
	<div>  <div> @Throwaway Account • 3 hours ago asdsadad </div> </div>
	<div>  <div> @A • 4 hours ago asdad </div> </div>
	<div>  <div> @A • 4 hours ago adasda </div> </div>
	<div>  <div> @A • 4 hours ago asda </div> </div>

Afbeelding 15: Uiteindelijke prototype

Mijn uiteindelijke oordeel van deze stack is dat het voor nieuwe developers goed te doen is. Er is veel documentatie te vinden, zelfs al is dit een best wel nieuwe stack. Ik liep vrijwel nergens tegen behalve dat ik moeite had met het wennen aan de TypeScript conventies. Gelukkig hebben diverse collega's van Webbio hier ervaring mee en hebben zij mij hiermee kunnen ondersteunen.

5.4.2. MERN Stack

Daarna heb ik me verdiept in de MERN Stack. Een van de meest populaire JavaScript stacks dat bestaat uit **M**ongoDB, **E**xpressJS, **R**ead en **N**odeJS (*MongoDB, z.d.*). Binnen deze stack gebruik je React om de front-end te ontwerpen, MongoDB als je database en Express met Node om te communiceren tussen de front- en back-end (zie afbeelding 16).



Afbeelding 16: Overzicht lagen binnen MERN

React, Node en Express waren een verplichting om te gebruiken bij DWA en vandaar dat ik me hier niet in ga verdiepen om opnieuw kennis over op te doen. Echter is MongoDB wel interessant om te benoemen. Dit is namelijk een NoSQL database en dit houdt in dat het een compleet andere structuur heeft in vergelijking tot SQL (*MongoDB, z.d.-b*). De NoSQL structuur bestaat namelijk uit een of meerdere collecties, iedere collectie bevat documenten en deze documenten bevatten

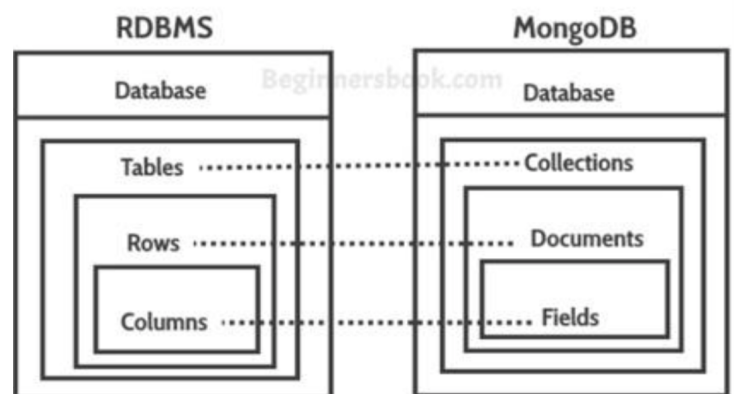
velden. Uiteindelijk wordt de data in deze velden opgeslagen op een JSON-achtige manier (zie afbeelding 17).

```
{
  "_id": "5f788a2a904ced85fe71231c",
  "username": "Kenerz",
  "firstname": "Ken",
  "password": "$2a$04$aUvLjoRAvazzj6YX7K2eEfUt9PHVVgr",
  "adress": {
    "street": "Stallstigen 93",
    "city": "Hargshamn",
    "state": "Sweden",
    "zip": "74073"
  },
  "games": ["Overwatch", "Valorant", "Minecraft"]
}
```

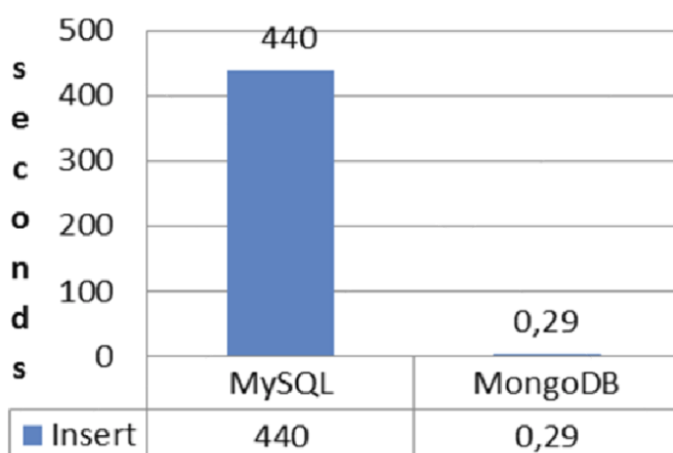
Afbeelding 17: Voorbeeld van een veld binnen MongoDB

In principe zijn de collecties, documenten en velden vergelijkbaar met tabellen, rijen en kolommen binnen een SQL database (zie afbeelding 18).

Wegens deze overzichtelijke data structuur is het ook wetenschappelijk bewezen dat MongoDB veranderingen in de database sneller uitvoert (Krishnan et al., 2016).



Afbeelding 18: Naamgeving



Een van de onderzoeken die zij hebben gedaan was het vergelijken hoe lang elke database erover doet om 10,000 gebruikers te inserten. MySQL doet er 440 seconden over, terwijl MongoDB dit doet in 0,29 seconden (zie afbeelding 19).

Afbeelding 19: NoSQL vs SQL

Allerlaatst is de schaalbaarheid een groot verschil tussen NoSQL en SQL databases. NoSQL databases zoals MongoDB zijn horizontaal schaalbaar, terwijl SQL databases verticaal schaalbaar zijn. Dit houdt in dat MongoDB meer apparaten, bijvoorbeeld servers, nodig heeft om het gehele systeem krachtiger te maken. Ondertussen krijgt SQL meer kracht door een al bestaand systeem te verbeteren (bijvoorbeeld een verbeterde processor).

Nu ik een goed beeld heb gekregen van wat een NoSQL database zoals MongoDB te bieden heeft, kan ik weer een prototype ontwikkelen. Hierbij is mijn plan om simpele CRUD applicatie op te zetten. De data van deze applicatie wil ik opslaan met een database in MongoDB Atlas, een platform waarin je Mongo databases kan beheren. Nadat ik dit werkend heb gekregen wil ik vanuit een API gemaakt door Express en Node gaan communiceren en de data op een React front-end tonen.

```
import express from "express";
import cors from "cors";
import "./loadEnvironment.mjs";
import records from "./routes/record.mjs";

const PORT = 5050;
const app = express();

app.use(cors());
app.use(express.json());

app.use("/record", records);

// start the Express server
app.listen(PORT, () => {
  console.log(`Server listening on port ${PORT}`);
});
```

Als eerst begin ik met de back-end en wil ik hiermee een lokale Express server draaiend krijgen. Dit bleek vrij eenvoudig te zijn wegens de uitgebreide documentatie die online over Node & Express servers te vinden waren.

Afbeelding 20: Express server

Nadat ik de server werkend heb gekregen, heb ik een verbinding gemaakt naar een MongoDB database.

Met deze verbinding heb ik de mogelijkheid om door middel van diverse routes te communiceren met de database.

Afbeelding 21: Database connectie

```
import { MongoClient } from "mongodb";


const connectionString = process.env.ATLAS_URI
const client = new MongoClient(connectionString);

let conn;
try {
  conn = await client.connect();
} catch(e) {
  console.error(e);
}

let db = conn.db("test");

export default db;
```

Daarna heb ik me gefocust op de front-end, hiervoor heb ik gekozen voor een simpele Single Page Application met Bootstrap (zie afbeelding 22).

			Create Record
Record List			
Name	Position	Level	Action
Alex Cheng	Developer	Intern	Edit Delete
Rowan Paul Flynn	Lead Developer	Junior	Edit Delete
Joeri Smits	Head of Development	Senior	Edit Delete

Afbeelding 22: Front-end pagina

Op deze pagina wordt er een simpele GET endpoints aangesproken en deze resultaten worden in een `.map()` gegooid om een voor een getoond te worden. Daarnaast spreekt iedere knop (create record, edit en delete) een endpoint aan zodra erop geklikt wordt. Deze functionaliteit wordt gedaan door middel van een fetch functie waarmee je kan specificeren wat voor type request het is samen met een eventuele body (zie afbeelding 23).

```
await fetch("http://localhost:5050/record", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify(data),
})
.catch(error => {
  console.log(error);
});
```

Afbeelding 23: Fetch functie

Afbeelding 24: Opgeslagen data

Binnen de database wordt het dan in de collectie opgeslagen zoals rechts te zien is (zie afbeelding 24)

Ik vond het erg makkelijk om een app te bouwen volgens de MERN stack. Er waren namelijk veel voorbeelden die online te vinden waren.

```
_id: ObjectId('651fbc73a751fe2df685812f')
name: "Alex Cheng"
position: "Developer"
level: "Intern"
```

```
_id: ObjectId('651fbc83a751fe2df6858130')
name: "Rowan Paul Flynn"
position: "Lead Developer"
level: "Junior"
```

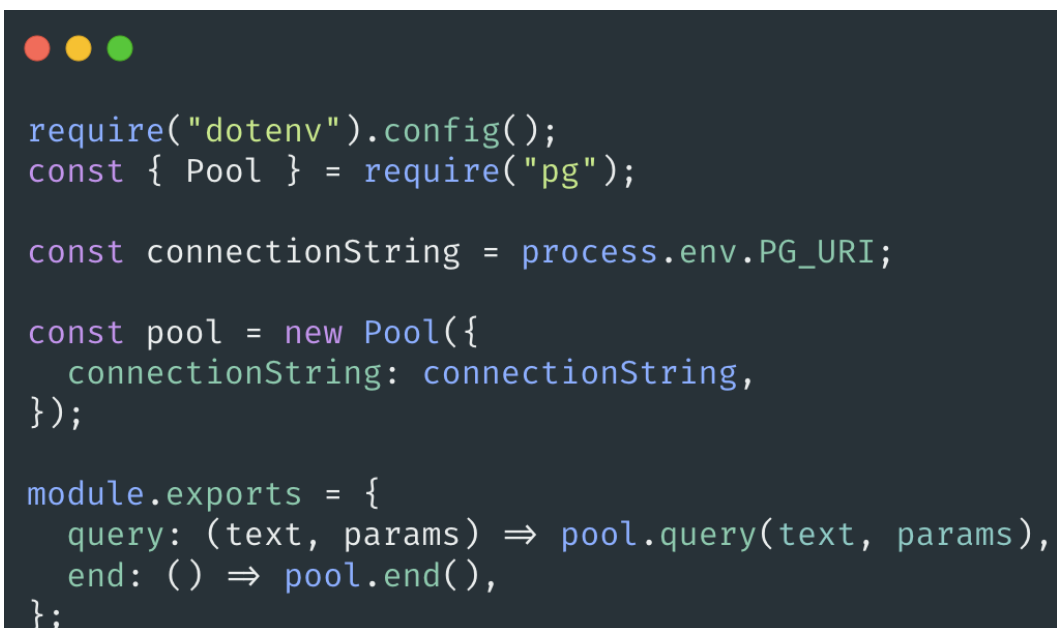
```
_id: ObjectId('651fbce1a751fe2df6858131')
name: "Joeri Smits"
position: "Head of Development"
level: "Senior"
```


5.4.3. PERN Stack

Naast MERN bestaat er ook PERN. Deze stack is erg vergelijkbaar met MERN. Het gebruikt dezelfde frameworks, maar maakt gebruik van een andere database. In plaats van MongoDB benut het namelijk PostgreSQL. Dit is een open-source relational database management system (RDBMS) en het staat betekent om de robuustheid, uitbreidbaarheid en ondersteuning voor SQL, waardoor het een erg populaire keuze is voor het opslaan en ophalen van gegevens (*PostgreSQL: about, z.d.*). Een bekend alternatief, MySQL, is niet open-source en wordt daarom minder gebruikt in tegenstelling tot PostgreSQL.

Na het onderzoeken van de verschillen tussen NoSQL en SQL heeft het mij erg nieuwsgierig tot hoeverre de functionaliteiten van elkaar verschillen. Dus als prototype hiervoor ga ik mijn CRUD applicatie herschrijven en uitbreiden om compatibel te maken met een SQL database.

Hiervoor heb ik allereerst een PostgreSQL database aangemaakt en de oude connection string vervangen. De verbinding wordt uiteraard niet meer gedaan door de eerder genoemde MongoClient, maar wordt nu geregeld door de "node-postgres" package (zie afbeelding 25).



```
require("dotenv").config();
const { Pool } = require("pg");

const connectionString = process.env.PG_URI;

const pool = new Pool({
  connectionString: connectionString,
});

module.exports = {
  query: (text, params) => pool.query(text, params),
  end: () => pool.end(),
};
```

Afbeelding 25: Verbinden met PostgreSQL

Iets wat mij direct opviel is dat SQL een andere syntax gebruikt ten opzichte van NoSQL en dus moest ik al mijn endpoints herschrijven om aan de SQL queries te kunnen voldoen (zie afbeelding 26).

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in a light-colored font and represents a JavaScript function named 'getRecords'. It uses an arrow function syntax and a database pool to execute a SQL query. The query is 'SELECT * FROM records'. The function checks for an error and throws it if present, then sends the records as JSON with a 200 status code.

```
const getRecords = (req, res) => {  
  pool.query('SELECT * FROM records', (error, records) => {  
    if (error) {  
      throw error  
    }  
    res.status(200).json(records.rows)  
  })  
}
```

Afbeelding 25: GET query om alle records op te halen

Nadat ik mijn API heb refactored werkte de gehele applicatie vlekkeloos. Ik hoefde niks extra's te veranderen aan de front-end aangezien de endpoints gelijk zijn gebleven.

Mijn ervaring met de PERN stack was soortgelijk als het werken met de MERN stack. Er was voldoende informatie over PostgreSQL te vinden, maar ze waren niet geheel duidelijk voor het verwerken in een Node / Express omgeving. Ook moest ik weer erg wennen aan alle SQL queries. Ten slotte heb ik niks gemerkt tussen de performance van de twee soorten databases, hoogstwaarschijnlijk omdat ik niet zoals het eerder genoemde onderzoek de data met grootschalige hoeveelheden aanpas. Uiteindelijk gaat mijn voorkeur toch naar MERN, want het verwerken van een MongoDB database ging iets soepeler en de structuur van een NoSQL database vind ik een stuk overzichtelijker.

5.4.4. Serverless

Ten slotte wil ik me gaan verdiepen in serverless webapplicaties. De reden hiervoor is dat het niet een echte stack is, maar wel een eventuele vereiste voor het kiezen van een stack. Ook heeft Webbio tegenwoordig al diverse applicaties serverless draaien, bijvoorbeeld op hun eigen AWS-omgeving.

Om te beginnen wat precies een serverless webapplicatie? Dit is een architectuurtype dat gebruikmaakt van cloud computing-services om de behoefte van traditionele serverbeheer en de voorzieningen hiervan te minimaliseren (*Amazon Web Services, z.d.*).

Een aantal kenmerken van het werken met een serverless webapplicatie zijn:

- **Geen server management:** Binnen een serverloze architectuur wordt het beheer van servers, besturingssystemen en infrastructuur weggenomen van de ontwikkelaar. Hierdoor kan diegene zich volledig focussen op het schrijven van code.
- **Pay-per-use:** Dit serverless computing introduceert een nieuw factureringsmodel, namelijk gebaseerd op betalen per gebruik. Hierbij worden ontwikkelaars alleen gefactureerd voor de daadwerkelijke computerkracht die verbruikt wordt in plaats van het moeten betalen voor bijvoorbeeld inactieve servers.
- **Stateless:** Serverless webapplicaties zijn ontworpen om stateless te zijn. Dit houdt in dat ze geen persistente gegevens tussen verzoeken opslaan. In plaats daarvan wordt alleen alle noodzakelijke informatie opgeslagen in externe services, zoals een database. Dit zorgt voor snellere schaalvergroting en vergroot ook de fouttolerantie.
- **Platform:** Er bestaan diverse cloud service providers die hun eigen serverless platform aanbieden. Iedere hebben hun eigen functionaliteiten en diensten. Een aantal bekende voorbeelden zijn: AWS Lambda (Amazon Web Services), Azure Functions (Microsoft Azure) en Google Cloud Functions (Google Cloud). Ontwikkelaars kunnen hun platform selecteren gebaseerd op eigen behoeften en voorkeuren.
- **Schaalbaarheid:** Deze serverless platforms zijn ontworpen om applicaties automatisch te schalen gebaseerd op het huidige verkeer. Wanneer er een plotselinge toename is van het verkeer, zorgt het platform voor extra middelen om de belasting aan te kunnen. Wederzijds, wanneer het verkeer daalt, worden de middelen ook verminderd. Hierdoor worden zowel de operationele kosten als de presentaties optimaal gehouden.

Nu er een helder beeld is gemaakt over wat serverless precies inhoudt, zal er immers weer een prototype ontwikkeld worden. De cloud service provider die hiervoor

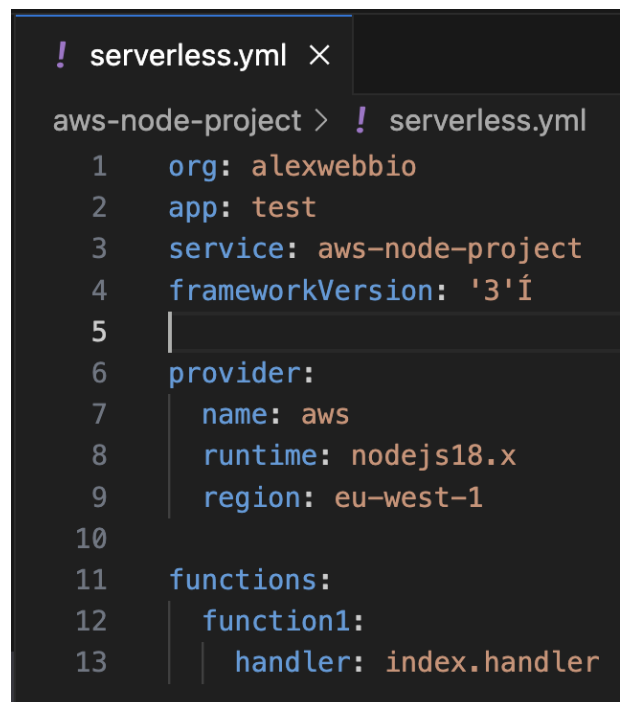
gebruikt gaat worden zal AWS zijn, omdat Webbio deze ook benut en is dit het grootste hosting platform ter wereld.

Daarnaast zal er gebruik worden gemaakt van de Serverless Framework, aangezien AWS zelf zoveel verschillende services heeft en het voor een ontwikkelaar zonder enige ervaring snel erg overweldigend kan zijn.

De Serverless Framework is een open-source framework dat het deployen en beheren van een serverless applicatie eenvoudiger maakt (*Serverless: Develop & monitor apps on AWS Lambda, z.d.*). Het neemt namelijk vele complexiteiten weg en daarnaast biedt het mogelijkheden aan zoals configuraties voor schaalbaarheid, event triggers en mogelijkheid om diverse plug-ins toe te voegen.

Dit framework heb ik globaal geïnstalleerd door middel van NPM. Daarna heb ik dit moeten configureren met mijn AWS gegevens, echter heeft dit veel tijd gekost omdat er geen duidelijke documentatie was hoe dit precies geregeld moest worden.

Nadat ik dit toch uiteindelijk werkend heb gekregen, heb ik een nieuwe serverless service kunnen creëren. Hierbij moest er een yaml bestand gedefinieerd worden met alle betreffende instellingen (zie afbeelding 26).



```
! serverless.yml ×
aws-node-project > ! serverless.yml
1  org: alexwebbio
2  app: test
3  service: aws-node-project
4  frameworkVersion: '3'
5
6  provider:
7    name: aws
8    runtime: nodejs18.x
9    region: eu-west-1
10
11  functions:
12    function1:
13      handler: index.handler
```

Afbeelding 26: Serverless.yml

Nadat deze configuratie correct is ingesteld, verwijst ik in de functies variabelen welke endpoints er aangeroepen moeten worden. Ten slotte kan ik dit op AWS deployen door 'serverless deploy' in de terminal te typen. Dit genereert automatisch een API url die benaderd kan worden door de gebruiker.

Ik vond het behoorlijk uitdagend om serverless te implementeren binnen mijn webapplicatie zelfs al maakte ik gebruik van een framework om dit juist eenvoudiger

te maken. AWS biedt zoveel alternatieve manieren aan om het bovenstaande prototype te bereiken, echter is het niet duidelijk welke manier het beste is.

OR 5.5. Hoe pas ik al bestaande cryptografische libraries toe?

Ik ben op npm gaan zoeken naar de meest populaire cryptografische libraries. Gebaseerd volgens de wekelijkse downloads op NPM zijn de meest populaire libraries op dit moment (per 2 oktober 2023) als volgt:

#	Naam	Wekelijkse Downloads	Versie	Laatst geupdate
1	crypto-js	5.835.510	4.1.1.	2 jaar geleden
2	bcrypt	1.251.583	5.1.1.	2 maanden geleden
3	cryptr	98.883	6.3.0.	2 maanden geleden

Tabel 1: Top 3 meest populaire cryptografische libraries

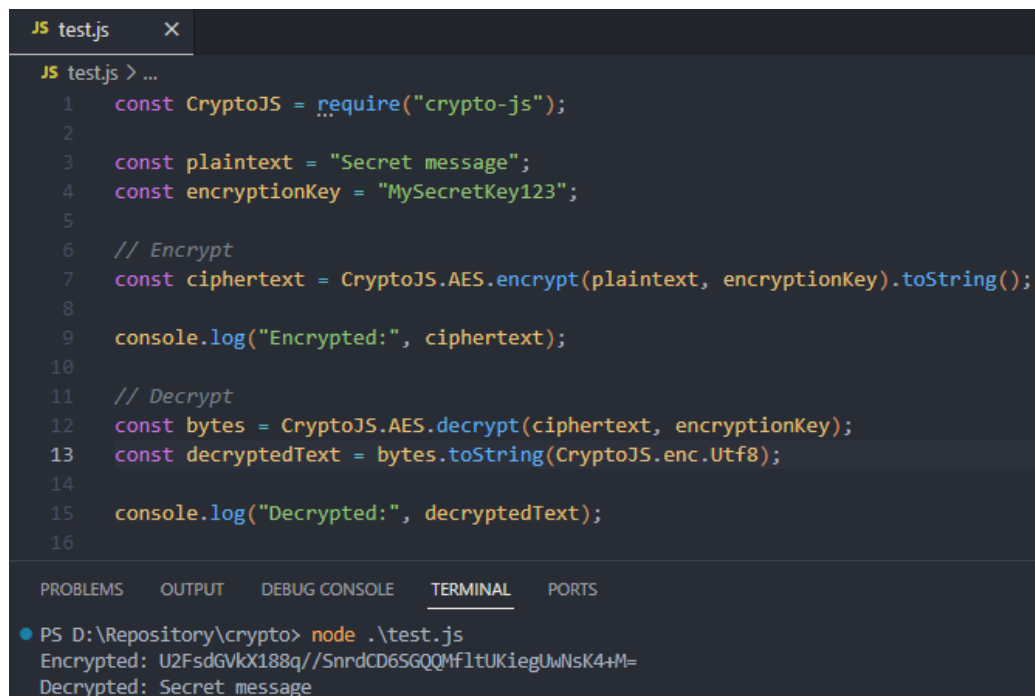
5.5.1. Crypto-js

Allereerst de crypto-js library, deze is ontwikkelt om een heleboel cryptografische functies te kunnen uitvoeren. Hierdoor kan een ontwikkelaar eenvoudig encryptie, decryptie, hashing en andere veiligheidsmaatregelen toepassen in hun eigen JavaScript applicatie (*CryptoJS, z.d.*).

Ook maakt het gebruik van diverse cryptografische algoritmes die al eerder in dit onderzoeksverslag benoemd zijn, zoals AES, DES en SHA-256. Hierdoor is het op een veelzijdige tool voor het beveiligen van gegevens en communicatie binnen webapplicaties. Deze library draagt bij aan de robuustheid van vele web- en serverside projecten en is daarom al jaren een vertrouwde keuze in de crypto community.

Als prototype heb ik geprobeerd een korte string te coderen met een symmetrisch algoritme (*zie afbeelding 27*). Hierbij heb ik gebruikgemaakt van het AES algoritme. Na het importeren van de library heb ik een secret key gemaakt (in dit geval een simpele string) en de betreffende tekst die gecodeerd moet worden. Daarna gaf ik deze variabelen mee met de encrypt() functie om het in een ciphertext te veranderen. Zoals er in de onderstaande console gezien kan worden komt er een

reeks willekeurige karakters uit. Daarna kan ik de ciphertext weer terugzetten in plaintext met de `decrypt()` functie. In het geval dat de secret key niet overeenkomt geeft de output niks terug. Uiteindelijk vond ik het vrij eenvoudig om crypto-js werkend te krijgen.



```
JS test.js x
JS test.js > ...
1  const CryptoJS = require("crypto-js");
2
3  const plaintext = "Secret message";
4  const encryptionKey = "MySecretKey123";
5
6  // Encrypt
7  const ciphertext = CryptoJS.AES.encrypt(plaintext, encryptionKey).toString();
8
9  console.log("Encrypted:", ciphertext);
10
11 // Decrypt
12 const bytes = CryptoJS.AES.decrypt(ciphertext, encryptionKey);
13 const decryptedText = bytes.toString(CryptoJS.enc.Utf8);
14
15 console.log("Decrypted:", decryptedText);
16

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS D:\Repository\crypto> node .\test.js
Encrypted: U2FsdGVkX188q//SnrdCD6SGQQMf1tUKiegUwNsK4+M=
Decrypted: Secret message
```

Afbeelding 27: Crypto-js code & output

5.5.2. Bcrypt

Bcrypt is een veel gebruikte cryptografische library ontworpen om voor het veilig hashen van wachtwoorden of andere gevoelige informatie. Het is speciaal ontworpen om rekenintensief te zijn, waardoor het goed bestendig is tegen aanvallen zoals een brute force of dictionary attack (Patel, 2023).

Bcrypt gebruikt een algoritme genaamd Blowfish cipher ontwikkeld door Bruce Schneier in 1993 als alternatief voor het DES algoritme (GeeksforGeeks, 2023). Vandaar dat de naam "bcrypt" verwijst naar "Blowfish crypt".

Het toepassen van deze library was vrij eenvoudig. Allereerst genereerde ik een willekeurige salt met een lengte van 10. Deze salt gebruikte ik dan met het wachtwoord dat gecodeerd moest worden om een hash van te maken. Uiteindelijk

kon ik verschillende inputs vergelijken met deze hash door middel van de `compare()` functie (zie afbeelding 28).

Echter viel mij een cruciaal probleem gelijk op met deze library. Het heeft alleen de mogelijkheid om gegevens te hashen, oftewel dit is een eenzijdig proces. Hierdoor zou ik bijvoorbeeld niet mijn gecodeerde gegevens later kunnen ophalen om met de externe partij te delen en zou het hoogstwaarschijnlijk niet geschikt zijn voor mijn doeleinden.



```
const bcrypt = require('bcrypt');

// Example password to hash
const passwordToHash = 'MySecurePassword123';

// Generate a salt and hash the password
bcrypt.genSalt(10, (err, salt) => {
  if (err) {
    throw err;
  }

  bcrypt.hash(passwordToHash, salt, (err, hash) => {
    if (err) {
      throw err;
    }

    // Store the hash in your database, not the plain password
    console.log('Hashed Password:', hash);

    // Later, when you need to verify a password
    const enteredPassword = 'MySecurePassword123'; // Password entered by the user

    bcrypt.compare(enteredPassword, hash, (err, isMatch) => {
      if (err) {
        throw err;
      }

      if (isMatch) {
        console.log('Password is correct');
      } else {
        console.log('Password is incorrect');
      }
    });
  });
});
```

Afbeelding 28: Bcrypt code

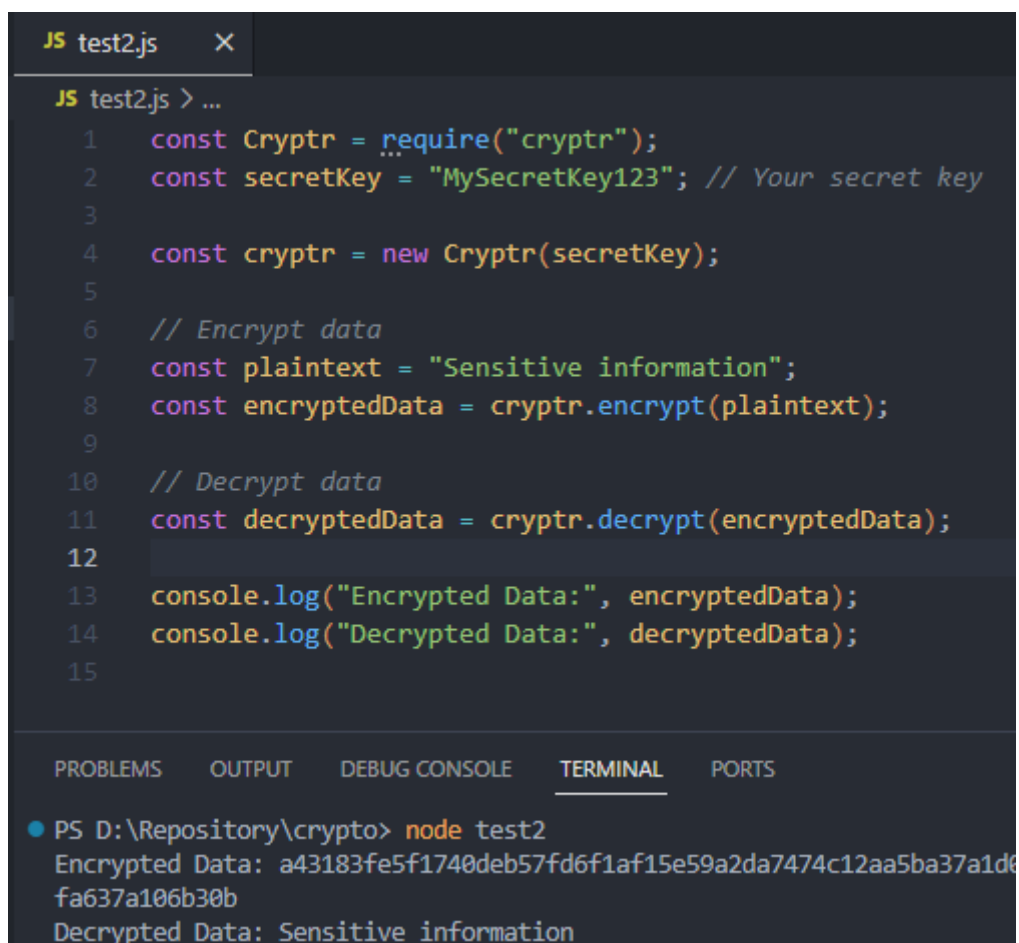
5.5.3. Cryptr

Ten slotte, hebben wij cryptr, dit is een iets minder populaire library, maar staat alsnog in onze top 3. Cryptr heeft als doel om codering en decodering eenvoudiger te maken in webapplicaties (*Plug & play authentication that fits your business, z.d.*). Het maakt vele complexiteiten binnen cryptografie namelijk abstracter, waardoor het makkelijker wordt voor ontwikkelaars om beveiligingsmaatregelen in hun applicatie

op te nemen. Vandaar dat het vaak wordt gebruikt bij front-end ontwikkeling voor het versleutelen van gevoelige informatie zoals persoonlijke informatie of gebruiker credentials. Ook heeft het al een geheel werkende integratie met SSO providers zoals Google Workspace en Azure AD, echter kost dit wel geld.

Het encryptie algoritme dat gebruikt wordt hierbij is AES-256-GCM. Dit is een combinatie van het AES-256 algoritme en Galois/Counter Mode (GCM). GCM is een modus die zowel encryptie als authenticatie van gegevens biedt, waardoor het een veilige en efficiënte keuze is voor cryptografische bewerkingen (*IBM documentation, z.d.*).

De implementatie hiervan was erg vergelijkbaar met de implementatie van crypto-js. Hierbij geef je ook simpelweg de plaintext mee in een encrypt() functie om een ciphertext te creëren. Daarna kan deze weer ontcijferd worden met de decrypt() functie. In plaats van de secret key per functie mee te geven wordt deze direct gedefinieerd (zie afbeelding 29).



```
JS test2.js  X
JS test2.js > ...
1  const Cryptr = require("cryptr");
2  const secretKey = "MySecretKey123"; // Your secret key
3
4  const cryptr = new Cryptr(secretKey);
5
6  // Encrypt data
7  const plaintext = "Sensitive information";
8  const encryptedData = cryptr.encrypt(plaintext);
9
10 // Decrypt data
11 const decryptedData = cryptr.decrypt(encryptedData);
12
13 console.log("Encrypted Data:", encryptedData);
14 console.log("Decrypted Data:", decryptedData);
15

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● PS D:\Repository\crypto> node test2
Encrypted Data: a43183fe5f1740deb57fd6f1af15e59a2da7474c12aa5ba37a1d6
fa637a106b30b
Decrypted Data: Sensitive information
```

Afbeelding 29: Cryptr code & output

6. Discussie

In dit hoofdstuk zal ik mijn oordeel geven over de resultaten van de verschillende onderzoeken. Er zal voornamelijk gekeken worden naar hoeveel toegevoegde waarde het heeft voor de ontwikkeling van SecureNotes.

6.1. Beveiligingsmaatregelen

Ik heb veel verschillende beveiligingsmaatregelen onderzocht, van zowel diverse encryptie algoritmes als andere cryptografische alternatieven. Hieronder mijn oordeel:

- **Encryptie Algoritme:** Zowel symmetrische en asymmetrische encryptie algoritmes zijn erg geschikt voor het versleutelen van gevoelige informatie, maar naar mijn mening overtreft symmetrische encryptie asymmetrische encryptie vanwege de efficiëntie en eenvoud ervan. Dit is wegens het feit dat symmetrische encryptie een enkele sleutel gebruikt voor zowel encryptie als decryptie. Hierdoor wordt er minder rekenkracht vereist en biedt dit een superieure snelheid, dit zal voornamelijk cruciaal zijn tijdens het uploaden van grote bestanden (denk aan databases).

Bovendien wordt asymmetrische encryptie meestal gebruikt om data langdurig te beveiligen, dit is het tegenovergestelde met wat wij binnen SecureNotes proberen te bereiken. De data zal namelijk direct verwijderd worden na het bekijken/downloaden hiervan en is dus maar eenmalig beschikbaar. Ook is een van de requirements om het bestand te verwijderen indien het langer dan zeven dagen niet bekeken blijft.

Daarnaast is er in de resultaten van mijn onderzoek al benoemd dat algoritmes zoals RSA nog niet geschikt zijn voor de datatype die ontvangen zullen worden binnen mijn applicatie.

Hoewel asymmetrische encryptie zijn voordelen heeft, is symmetrische encryptie in deze toepassingen de efficiëntere en effectievere keuze voor robuuste gegevensbeveiliging. Het enige nadeel dat het heeft is dat er

binnen symmetrische encryptie geen authenticatie plaatsvindt, maar dit probleem kunnen we met andere beveiligingsmaatregelen oplossen.

- **Hashing:** Ik vind dat hashing niet echt passend is voor mijn SecureNotes applicatie. De oorspronkelijke data moet nog gelezen kunnen worden door andere ontvangers, echter is dit met hashing niet meer mogelijk nadat iets gehashed is.
- **Obfuscation:** In de context van mijn afstudeeropdracht zal obfuscation niet erg relevant zijn. Ik ga namelijk geen eigen encryptie algoritme schrijven die obfuscated moet worden. Daarnaast is het ook niet handig om dit toe te gaan passen op mijn eigen code, omdat dan Webbio hierdoor moeilijker mijn code kan begrijpen en onderhouden.
- **Masking:** Ook masking zal voor mijn doeleinden niet erg relevant zijn. Het testen met nep gegevens zal niet gedeeld hoeven te worden met externe partijen, maar volledig binnen de organisatie blijven.
- **Two-Factor Authentication (2FA):** 2FA vind ik wel een sterke beveiligingsmaatregel die van pas zal kunnen komen binnen mijn applicatie. Het zou datgene wat mist binnen symmetrische encryptie, authenticatie, kunnen opvangen.

Er zou een soort van code gegenereerd kunnen worden zodra een Webbio medewerker gevoelige informatie uploadt en deze code zal dan gedeeld moeten worden met de externe partij. Deze code wordt dan gebruikt als authenticatie van de ontvanger en alleen met deze code heeft de ontvanger toegang tot de betreffende informatie.

Bij de resultaten van 2FA was ook kort Multi-Factor Authentication (MFA) benoemd. MFA vind ik persoonlijk iets te veel voor mijn applicatie. Er zijn namelijk al twee soorten controles om te verifiëren dat het de juiste ontvanger is: werknemer bevestigt de e-mail ontvanger en de authenticatiecode. Als er nog meer verificaties toegevoegd worden, vind ik dat het de gebruiksvriendelijkheid verliest.

Uiteindelijk heb ik een weight decision matrix opgesteld om de alternatieven te vergelijken (zie tabel 2). Dit is een beoordelingstechniek om een reeks keuzes te evalueren met een reeks criteria (*FreshBooks, z.d.*). Hierbij wordt alles geëvalueerd op

een schaal van 1-5, waarbij hoe hoger hoe beter is. Mijn matrix gaat kijken naar de beveiliging, bruikbaarheid voor mijn applicatie en de benodigde computerkracht. Daarna zal er een totaal aantal uitkomen die weergeeft hoe krachtig een beveiligingsmaatregel is.

Beveiligingsmaatregel	Beveiliging	Bruikbaarheid	Computerkracht	Totaal
Gewicht	5	3	2	
Symmetrische encryptie	5	5	2	44
Asymmetrische encryptie	5	4	1	39
Hashing	4	1	4	31
Obfuscation	3	1	5	28
Masking	2	2	5	26
Two-Factor Authentication	4	5	3	41

Tabel 2: Weight Decision Matrix - Beveiligingsmaatregel

Uit deze tabel valt te zien dat symmetrische encryptie en two-factor authentication het meest geschikt zijn voor het ontwikkelen van SecureNotes.

6.2. Sleutelbeheer

Tijdens het onderzoek heb ik ook gekeken naar hoe ik het meest efficiënt en veilig een secret key kan delen. Hierover zal ik net zoals in hoofdstuk 6.1 mijn mening geven:

- **Key Exchange Protocols:** Een key exchange protocol blijkt uit mijn onderzoek een erg veilige techniek te zijn. Al helemaal volgens het Diffie-Hellman algoritme. Echter is dit niet de efficiëntste manier aangezien zowel de zender als de ontvanger verschillende variabelen moeten vaststellen. Het zou tijd kosten om meerdere sleutels te genereren en daar vanuit een gedeelde sleutel mee te berekenen. Daarnaast zou het eventueel nog meer tijd kosten in het geval dat een van de twee partijen niet bekend is met deze techniek.
- **Key Splitting:** Dit is een veilige en behoorlijk efficiënte manier om een secret te delen, echter hoeft de secret alleen maar eenmalig gedeeld te worden met

een partij. Vandaar dat key splitting niet geheel toegepast zal kunnen worden binnen mijn applicatie.

- **Key Derivation Function:** Ook dit vind ik een zeer veilig en efficiënt alternatief, maar helaas is het niet geschikt voor mijn context. Ik heb namelijk maar een sleutel nodig die maar een specifiek cryptografisch doel heeft, het (de)coderen van de gevoelige informatie.
- **Password-Authenticated Key Agreement:** Van alle genoemde alternatieven vind ik dit nog het meest efficiënt. Hierbij kun je volgens de balanced vorm hetzelfde wachtwoord vaststellen en daarmee een gedeelde secret key genereren. Echter zal het niet nodig zijn om een gedeelde secret key te maken wanneer er gebruikgemaakt zal worden van een symmetrisch encryptie algoritme.

Alle genoemde methoden om een secret key te beheren en te delen zijn veilig en efficiënt, maar geen optie past binnen mijn eigen applicatie en dus zal ik geen van deze methodes toe kunnen passen.

5.3. Stack

Dit was een van de belangrijkste onderzoeken voor mijn afstudeeropdracht. De stack die ik ga gebruiken is deels afhankelijk van de ondersteuning die Webbio mij hierbij kan bieden. Daarnaast is het ook cruciaal dat Webbio het later kan gaan onderhouden zodra de opdracht is afgerond. In dit hoofdstuk ga ik kijken naar de onderzochte stacks.

- **T3 Stack:** Als eerst had ik de T3 Stack onderzocht. Dit vind ik een zeer geschikte stack aangezien het veel documentatie heeft voor een erg moderne stack. Ook is het voor iemand die nog geen ervaring heeft met deze stack eenvoudig te leren, vele libraries zijn simpelweg uitbreidingen van andere frameworks die ik tijdens mijn studie op de HAN al heb gebruikt. En ten slotte heeft Webbio ook al ervaring met deze stack, waardoor mijn collega's mij kunnen helpen indien ik vastloop.
- **MERN:** Daarna had ik MERN onderzocht. Ook dit vind ik een geschikte stack aangezien er reeds veel documentatie en voorbeelden bestaan over het gebruik van deze stack. Hierbij heeft Webbio ook al ervaring mee, maar werken zij tegenwoordig meestal met Next.js in plaats van React.js en

TypeScript over JavaScript. Ten slotte maakt een NoSQL database het ook geschikte kandidaat om grote hoeveelheden gevoelige data op te slaan.

- **PERN:** Vergelijkbaar met MERN zou dit eventueel ook een geschikte stack kunnen zijn voor mijn afstudeeropdracht. Er bestaat veel documentatie over de PostgreSQL database, echter gaat mijn voorkeur toch naar een NoSQL database zoals MongoDB aangezien deze krachtiger zijn en de structuur hierbinnen een stuk overzichtelijker dan een SQL database.
- **Serverless:** Allerlaatst heb ik een serverless alternatief met AWS onderzocht. Dit vond ik erg uitdagend om werkend te krijgen, aangezien de AWS-omgeving zo uitgebreid en complex is. Uiteindelijk heb ik het toch werkend kunnen krijgen door middel van een framework, echter heeft dit heel veel tijd gekost om de configuratie goed in te stellen. Daarnaast ben ik erachter gekomen dat zonder dit framework je alle code binnen de AWS-omgeving moet typen. Dit maakt het niet flexibel in het geval dat je de applicatie zou willen aanpassen, want dan zou je zowel je eigen code als de code binnen AWS moeten aanpassen. Als ontwikkelaar wil je het liefst alle code op een plek hebben.

Omdat er zoveel veelbelovende alternatieven zijn heb ik weer een weight decision matrix opgesteld om te bepalen welke stack het meest geschikt is voor mijn opdracht (zie tabel 3). Als eerst kijk ik naar de hoeveelheid documentatie dat er over de betreffende stack te vinden was, daarna kijk ik naar de simpelheid van de inhoud en ook kijk ik hierbij naar de complexiteit die ik heb ervaren tijdens het ontwikkelen van mijn prototype, ten slotte ga ik oordelen over de schaalbaarheid die iedere stack aanbiedt.

Stack	Documentatie	Simpelheid	Schaalbaarheid	Totaal
Gewicht	3	4	5	
T3 Stack	5	4	5	56
MERN	5	5	3	50
PERN	5	4	3	46
Serverless	4	2	5	45

Tabel 3: Weight Decision Matrix - Stack

Volgens deze weight decision matrix blijkt het dat de T3 Stack het meest geschikt is als stack voor dit afstudeerproject. Het bevat goede documentatie, is redelijk eenvoudig om te leren en is erg schaalbaar voor in de toekomst.

5.4. Libraries

Ten slotte heb ik gekeken naar diverse cryptografische libraries die al bestaan. Dit was voornamelijk om een helder beeld te creëren over wat hedendaags verwacht wordt van de meest populaire libraries en hoe deze daadwerkelijk toegepast worden. In principe waren alle libraries goed te begrijpen en eenvoudig te implementeren. Dit was wegens de verschillende code voorbeelden die al bestonden en de documentatie hiervan.

In het geval van bcrypt bevatte het alleen hashing en dit maakt het niet geschikt voor SecureNotes. De andere twee libraries, crypto-js en cryptr, bevatten allebei symmetrische encryptie mogelijkheden. crypto-js biedt ondersteuning van vele encryptie algoritmes zoals AES, DES, MD5, SHA-256, etc. terwijl cryptr alleen AES-256-GCM gebruikt. Cryptr is daarom ook ontworpen om een simpele en gebruiksvriendelijke library te zijn voor het coderen en decoderen van data. In tegenstelling hiervan wilt crypto-js juist een uitgebreide en veelzijdige library zijn die een breed aanbod van cryptografische functies bevat.

Zelfs al staan zowel crypto-js als cryptr in de top drie meest populaire cryptografische library op dit moment, crypto-js is op simpelweg nog een stuk populairder en heeft daarom ook een grotere community die gebruikers kunnen ondersteunen. Vandaar dat ik voor het beveiligen van de gevoelige informatie toch gebruik zal maken van crypto-js.

7. Conclusie

Ten slotte kijken we terug naar de hoofdvraag: “Welke beveiligingsstandaarden moet ik toepassen om mijn applicatie veilig en tegelijkertijd efficiënt te maken?”.

In dit onderzoeksverslag zijn diverse beveiligingsstandaarden onderzocht, maar het werkelijke antwoord is dat de benodigde beveiligingsstandaarden volledig afhankelijk zijn van de gewenste applicatie. In mijn geval zal mijn applicatie symmetrische encryptie en two-factor authentication toepassen.

Deze maatregelen passen binnen de doeleinden van mijn applicatie en werken efficiënt in combinatie met elkaar. Het wordt minder efficiënt en tegelijkertijd minder gebruiksvriendelijker door juist meer beveiligingsstandaarden toe te voegen.

Daarnaast zal mijn applicatie gebruikmaken van de T3 Stack om een applicatie te ontwikkelen die deze beveiligingsstandaarden toe te passen.

8. Literatuurlijst

- Informatiebeveiliging (ISO 27001) - Cybersecurity & Privacy - Digitale ethiek en veiligheid - ICT. (z.d.).
<https://www.nen.nl/ict/digitale-ethiek-en-veiligheid/cyber-privacy/informatiebeveiliging>
- Bonestroo, W.J., Meesters, M., Niels, R., Schagen, J.D., Henneke, L., Turnhout, K. van (2018): ICT Research Methods. HBO-i, Amsterdam. ISBN/EAN: 9990002067426.
Available from: <http://www.ictresearchmethods.nl/>
- What is encryption and how does it work? | Google Cloud. (z.d.). Google Cloud.
<https://cloud.google.com/learn/what-is-encryption>
- Cloudflare.com. (2023). What is encryption? Cloudflare.
<https://www.cloudflare.com/nl-nl/learning/ssl/what-is-encryption/>
- Marget, A. (2022). Data encryption: How it works & Methods used. Unitrends.
<https://www.unitrends.com/blog/data-encryption>
- GeeksforGeeks. (2023). Advanced Encryption Standard AES. GeeksforGeeks.
<https://www.geeksforgeeks.org/advanced-encryption-standard-aes/>
- Cryptography Attacks: 6 Types & Prevention. (2022, 26 augustus). Packetlabs.
<https://www.packetlabs.net/posts/cryptography-attacks/>
- Loshin, P., & Cobb, M. (2021). Data Encryption Standard (DES). Security.
<https://www.techtarget.com/searchsecurity/definition/Data-Encryption-Standard>
- Understanding Digital Signatures | CISA. (2021, 1 februari). Cybersecurity and Infrastructure Security Agency CISA.
<https://www.cisa.gov/news-events/news/understanding-digital-signatures>
- Sangfor Technologies. (2022, 25 april). What is a block cipher and how does it work. SANGFOR. <https://www.sangfor.com/glossary/cybersecurity/what-is-block-cipher>
- Sullivan, N. (2022). A (Relatively easy to understand) primer on elliptic curve cryptography. The Cloudflare Blog.
<https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/>
- Kok, A., & Kok, A. (2023). Wat is hashing en hoe werkt het? NordVPN.
<https://nordvpn.com/nl/blog/hash/en/>
- Hofheinz, D., Malone-Lee, J., & Stam, M. (z.d.). Obfuscation for Cryptographic Purposes. iacr.org. <https://eprint.iacr.org/2006/463.pdf>
- What is two-factor authentication (2FA)? | Microsoft Security. (z.d.).
<https://www.microsoft.com/en-ww/security/business/security-101/what-is-two-factor-authentication-2fa>

- What is Multi-Factor Authentication? - MFA explained - AWS. (z.d.). Amazon Web Services, Inc. <https://aws.amazon.com/what-is/mfa/>
- What is access control? - Citrix. (2023). Citrix.com. <https://www.citrix.com/solutions/secure-access/what-is-access-control.html>
- Burchfiel, A. (2023). Data masking vs encryption: Are you using the right data security tool? tokenex. <https://www.tokenex.com/blog/ab-what-is-data-masking-data-masking-vs-encryption-for-data-security/>
- Cobb, M. (2022). Data masking. Security. <https://www.techtarget.com/searchsecurity/definition/data-masking>
- RnD, D. (2023, 1 januari). Cryptographic key Exchange - deep RND - medium. Medium. <https://deeprnd.medium.com/cryptographic-key-exchange-5eb9e926edb0>
- What is the Diffie–Hellman (DH) algorithm? | Security Encyclopedia. (z.d.-b). <https://www.hypr.com/security-encyclopedia/diffie-hellman-algorithm>
- What is key splitting? | Security Encyclopedia. (z.d.). <https://www.hypr.com/security-encyclopedia/key-splitting>
- Technologies, K. (2021, 13 december). A beginner's guide to Shamir's secret sharing - Keyless Technologies - Medium. Medium. <https://medium.com/@keylesstech/a-beginners-guide-to-shamir-s-secret-sharing-e864efbf3648>
- Nugent, D. (2022, 4 oktober). Shamir's secret sharing: Explanation and visualization — blog — Evervault. Evervault. <https://evervault.com/blog/shamir-secret-sharing>
- Wikipedia contributors. (2023). Key Derivation Function. Wikipedia. https://en.wikipedia.org/wiki/Key_derivation_function
- Green, M. (2020, 21 januari). Let's talk about PAKE. A Few Thoughts on Cryptographic Engineering. <https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/>
- Hao, F., & Van Oorschot, P. C. (2022). SoK: Password-Authenticated Key Exchange. iacr. <https://eprint.iacr.org/2021/1492.pdf>
- Create T3 App. (z.d.). Create T3 App. <https://create.t3.gg/>
- Next.js by Vercel - the React framework. (2023). <https://nextjs.org/>
- JavaScript with syntax for types. (z.d.). <https://www.typescriptlang.org/>
- Installation - Tailwind CSS. (2023). Tailwind CSS. <https://tailwindcss.com/docs/installation>
- Prisma. (z.d.). Prisma | Next-generation ORM for Node.js & TypeScript. <https://www.prisma.io/>

- TRPC - Move fast and break nothing. End-to-end TypeSafe APIs made easy. | TRPC. (2001, 19 september). <https://trpc.io/>
- Collins, I. & NextAuth.js. (2023). NextAuth.js. <https://next-auth.js.org/>
- Moretti, F. (2023). Next.js Revolution - The framework popularity rises in 2023. DEV Community.
<https://dev.to/franciscomoretti/nextjs-revolution-the-framework-popularity-rises-in-2023-5356>
- MongoDB. (z.d.). How to use MERN Stack: A complete guide.
<https://www.mongodb.com/languages/mern-stack-tutorial>
- MongoDB. (z.d.-b). What is NoSQL? NoSQL databases explained.
<https://www.mongodb.com/nosql-explained>
- Krishnan, H., Elayidom, M., & Santhanakrishnan, T. (2016). MongoDB – A comparison with NoSQL databases. ResearchGate.
https://www.researchgate.net/publication/327120267_MongoDB_-_a_comparison_with_NoSQL_databases
- PostgreSQL: about. (z.d.). The PostgreSQL Global Development Group.
<https://www.postgresql.org/about/>
- Build your first serverless web app | Amazon Web Services. (z.d.). Amazon Web Services, Inc. <https://aws.amazon.com/serverless/build-a-web-app/>
- CryptoJS - CryptoJS. (z.d.). <https://cryptojs.gitbook.io/docs/>
- Patel, H. (2023, 27 maart). Password hashing in Node.Js with BCrypt - LogRocket blog. LogRocket Blog.
<https://blog.logrocket.com/password-hashing-node-js-bcrypt/>
- GeeksforGeeks. (2023). Blowfish algorithm with examples. GeeksforGeeks.
<https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/>
- Plug & play authentication that fits your business. (z.d.). Cryptr.
<https://www.cryptr.co/>
- IBM documentation. (z.d.).
<https://www.ibm.com/docs/en/zos/2.3.0?topic=operation-galoiscounter-mode-gcm>
- FreshBooks. (z.d.). What is a weighted decision matrix & how do you use it?
<https://www.freshbooks.com/en-ca/hub/other/weighted-decision-matrix>