

Testrapport

# 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:** 2.0



**webbio.**

# Inhoudsopgave

<b>1. Inleiding</b>	<b>3</b>
<b>2. Tooling</b>	<b>4</b>
2.1. Vitest	4
2.2. AWS-S3-Client-Mock	4
2.3. Lighthouse	5
<b>3. Test Cases</b>	<b>6</b>
<b>4. Gevonden Fouten</b>	<b>7</b>
4.1. Response S3 Client	7
4.2. Upload file	7
4.3. Toegankelijkheid	8
<b>5. Code Coverage</b>	<b>9</b>
<b>6. Conclusie</b>	<b>10</b>
<b>6. Bronnenlijst</b>	<b>11</b>

# 1. Inleiding

Voor de applicatie Secure Notes is het van uiterste belang dat alles veilig en correct functioneert, aangezien er met gevoelige informatie gewerkt wordt. Om dit te garanderen zijn er verschillende integratie testen en unit testen geschreven om te controleren of de werking van verschillende componenten zoals gewenst is.

In dit testrapport heb ik de overweging genomen om voornamelijk alleen de back-end te testen, aangezien de front-end erg minimaal is met maar twee pagina's. Daarnaast is het complexe gedeelte van de applicatie verwerkt in de back-end. Hierbij zijn er namelijk diverse integraties met externe systemen zoals AWS S3 en Google SSO. De volledige infrastructuur is terug te vinden in de bijlage: "Software Guidebook".


Verder zal dit testrapport verdiepen in de gebruikte tooling, de behandelde test cases, eventuele fouten die ontdekt zijn en uiteindelijk een conclusie.

## 2. Tooling

Om het testen eenvoudig en overzichtelijk te houden zijn de volgende npm packages gebruikt:

### 2.1. Vitest

Vitest, afgeleid van “vulnerability testing” is een testing framework dat specifiek is ontworpen om de weerstand en robuustheid van een programma of systeem te evalueren (Fu & Capeletto, 2021). Het framework is erg goed in extreme situaties en onverwachte input te simuleren, waardoor het makkelijk te bepalen is hoe goed een softwaretoepassing kan omgaan met verschillende omstandigheden.

Ook raad de officiële documentatie van mijn gebruikte stack, de T3 Stack, Vitest aan om te gebruiken voor integratie testen met tRPC (TRPC  Create T3 App, z.d.). Dit is omdat Vitest erg gericht is op het identificeren van kwetsbaarheden en mogelijke systeemfouten. Daarnaast kan deze package perfect de invoer valideren die Zod normaal gesproken regelt.

Wegens deze Vitest uit te voeren kunnen ontwikkelaars en testers potentiële zwakke punten ontdekken en verhelpen voordat ze leiden tot kritieke fouten in de productieomgeving. Hierdoor draagt Vitest bij aan het verhogen van de algehele stabiliteit en betrouwbaarheid van softwaretoepassingen.

Mijn test bestanden zijn terug te vinden in `/tests/s3.test.ts` & `/tests/upload.test.ts` en deze worden automatisch gedetecteerd door Vitest zelf.

### 2.2. AWS-S3-Client-Mock

Daarnaast heb ik een npm package gebruikt genaamd AWS-S3-Client-Mock (Radzikowski, 2022). Zoals de naam al verkapt dient dit als een hulpmiddel voor het testen van de populaire AWS-SDK/Client-S3 package. Hiermee kunnen ontwikkelaars alsnog functionaliteiten van een echte AWS S3 Client testen, maar dan in een gesimuleerde versie.

## 2.3. Lighthouse

Ook heb ik gebruik gemaakt van Lighthouse, een ingebouwde ontwikkelingstool binnen Google Chrome. Hiermee is het mogelijk om verschillende statistieken van een webapplicatie te analyseren zoals de laadtijd, SEO kwaliteit, toegankelijkheid en nog veel meer handige informatie (Lighthouse Overview - Chrome for Developers, 2016).

De opdrachtgever had oorspronkelijk diverse eisen gesteld rondom de prestaties van de Secure Notes applicaties. Met behulp van Lighthouse heb ik dit kunnen evalueren en eventuele verbeteringen kunnen toepassen.

### 3. Test Cases

Onderstaand staan de verschillende testen die uitgevoerd en geslaagd zijn. Ieders beschreven met hun resultaat en de betreffende user story.

User Story	Wat is er getest?	Wat was de uitkomst?
SN-14	Poging om iets te uploaden zonder ingelogd te zijn.	De API geeft een "unauthorized" terug.
SN-14, SN-16	Gebruiker upload een tekst naar een mock database.	Text wordt geüpload in de mock database en API geeft een secret terug (om te delen met ontvanger).
SN-14, SN-16	Gebruiker upload een bestand naar een mock database.	Bestand wordt geüpload in de mock database, maakt een relatie tussen het bestand en de upload en API geeft een secret terug (om te delen met ontvanger).
SN-15	Secret genereert met de salt van de environment variables.	De willekeurige gegenereerde secret bevatte de salt als hex waarde.
SN-15	Secrets zijn ieders uniek gegenereerd.	Meerdere secrets zijn gegenereerd en kwamen allemaal niet overeen.
SN-16	Encryptie en decryptie van een string.	De string die eerst gecodeerd werd naar cipher tekst en daarna omgezet naar plain tekst kwam overeen met de oorspronkelijke input
SN-20	File ophalen vanuit mock S3 omgeving.	Mock bestand wordt teruggegeven door S3 Client.
SN-20	Tekst ophalen vanuit mock Prisma omgeving.	Mock upload wordt teruggegeven door Prisma vanuit MySQL database.
SN-20	Niet bestaande bestanden ophalen.	Een verwachte foutmelding werd teruggegeven.
SN-14	File uploaden op mock S3 omgeving.	Mock bestand wordt toegevoegd in de S3 bucket.
SN-5	De performance van de applicatie is getest met Lighthouse	Er was een minimale eis gesteld van >90%, maar mijn applicatie heeft een resultaat van 100% behaald.

## 4. Gevonden Fouten

Tijdens het schrijven en testen van de testcases zijn een aantal fouten naar boven gekomen. De volgende fouten zijn ontdekt tijdens het testen:

### 4.1. Response S3 Client

Het toevoegen van data geeft altijd een leeg object terug indien een PutObjectCommand request met de betreffende parameters correct wordt meegestuurd.

In mijn code ging ik er oorspronkelijk vanuit dat er enige response of callback terug werd gegeven. Daarom controleerde ik of response niet undefined was, in plaats van op een response code of status message te checken. Natuurlijk zou dit gewoon door de if statement komen, aangezien het niet undefined is.

Pas tijdens het verwerken van de expect() functie van Vitest kwam ik erachter dat de actual response een leeg object was en hiervoor heb ik dus mijn code aan moeten passen.

### 4.2. Upload file

De code voor het uploaden van een bestand was vergelijkbaar met het uploaden van tekst, echter ontdekte ik pas in mijn test dat de gemaakte interface (/types/IUpload) niet geheel up-to-date was.

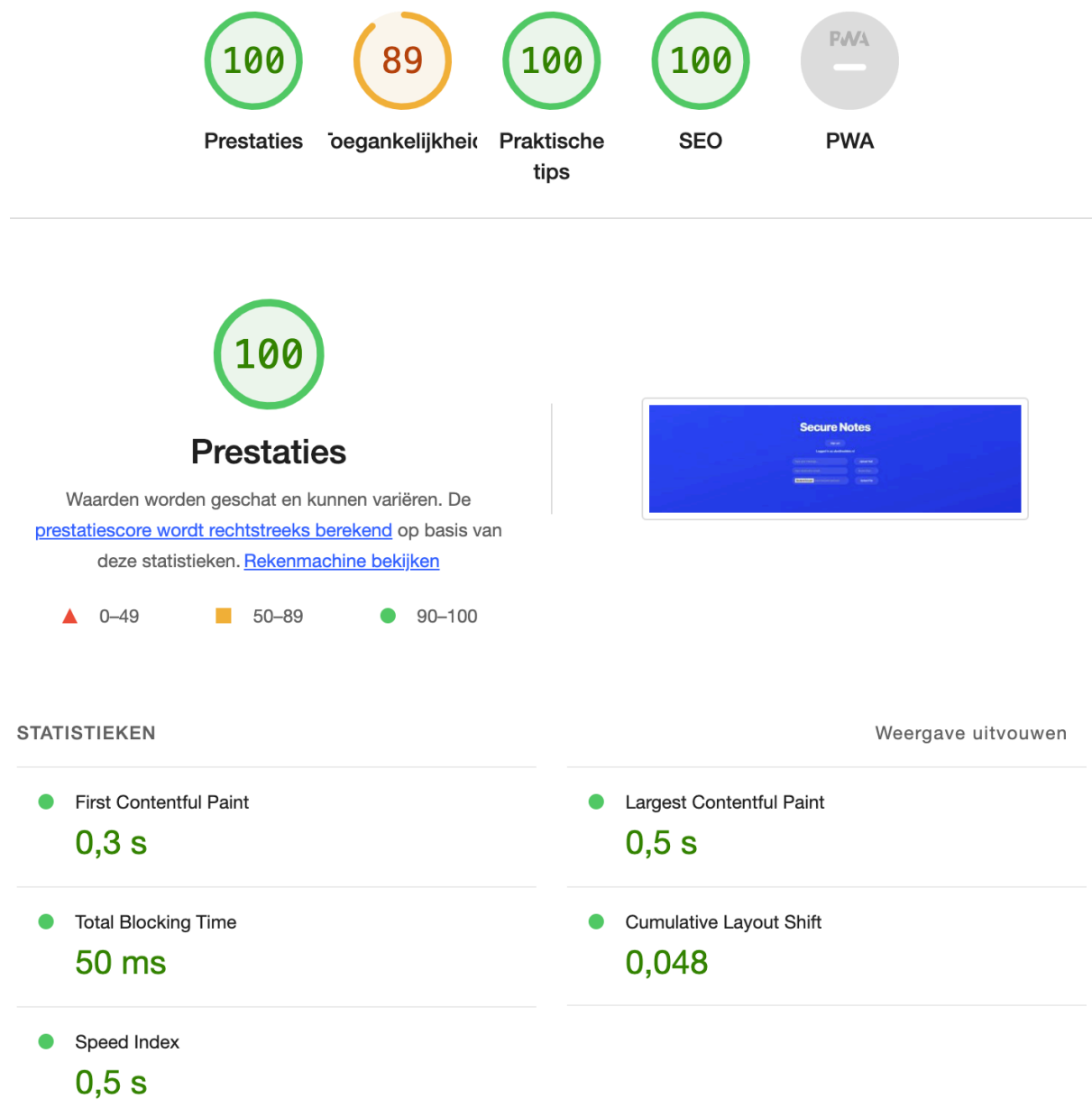
Tijdens het invoeren van verschillende inputs door middel van de inferProcecedureInput() ben ik erachter gekomen dat een aantal velden die niet verplicht waren, eigenlijk wel verplicht moeten zijn. Dit was eenvoudig op te lossen door simpelweg de waarde in het interface bestand te veranderen van optioneel naar vereist.

Wegens deze test heb ik voorkomen dat tijdens productie benodigde gevoelige informatie ontbreekt en eventuele fouten worden gecreëerd bij andere componenten die deze informatie nodig hebben, zoals bij het ophalen van data.

## 4.3. Toegankelijkheid

Uit de resultaten van de Lighthouse test (zie afbeelding 1) bleek de applicatie goed te presteren op allerlei categorieën, behalve op toegankelijkheid. Dit was wegens het gebrek aan labels voor de verschillende inputs op de pagina. Hierdoor zou bijvoorbeeld een schermlezer niet correct kunnen functioneren.

Dit probleem is om simpelweg labels toe te voegen voor iedere inputs en is al opgepakt in het toekomstige design van Secure Notes.



Afbeelding 1: Lighthouse resultaten



## 5. Code Coverage

Middels Vitest is er een code coverage rapport gegenereerd dat de statistieken van alle geteste code overzichtelijk weergeeft. Hierdoor kan er in een oogopslag gekeken worden hoe effectief de testen zijn.

In het rapport zijn twee details gemeten: function en line coverage. Function coverage kijkt naar de hoeveelheid functies die getest zijn en line coverage kijkt naar de hoeveelheid regels die uitgevoerd zijn.

Bij het testen van SecureNotes zag het rapport er als volgt uit:

File	Function Coverage	Line Coverage
upload.tsx	100%	93%
view.tsx	100%	89%
Totaal	100%	91%

**Tabel 1:** Code Coverage

Hierbij is er gestreefd naar een minimale score van 80% voor iedere meting. En dit blijkt uit tabel te zijn behaald.

## 6. Conclusie

In dit testrapport hebben we Vitest, AWS-SDK-Client-Mock en Lighthouse benut om de kwaliteit van de Secure Notes applicatie te evalueren en eventueel te verbeteren.


Dankzij het gebruik van Vitest zijn diverse componenten getest in verschillende situaties. Hierbij zijn er sommige fouten in de code geïdentificeerd om de werking van de applicatie te corrigeren.

Daarnaast heeft AWS-SDK-Client-Mock ervoor gezorgd om een beter inzicht te krijgen in de prestaties van de applicatie onder verschillende omstandigheden door de gesimuleerde versie.

Ook heeft Lighthouse een bijdrage geleverd aan het verbeteren en analyseren van de prestaties binnen de applicatie door mij nuttige inzichten te geven over de huidige resultaten van Secure Notes.

Uiteindelijk zijn alle geschreven tests geslaagd met een stevige code coverage rapport, waardoor dit heeft geleid tot een meer stabiele en fouttolerante applicatie.

## 6. Bronnenlijst

- Fu, A., & Capeletto, M. (2021). Vitest. <https://vitest.dev/>. <https://vitest.dev/>
- TRPC  Create T3 App. (z.d.). Create T3 App.  
<https://create.t3.gg/en/usage/trpc#sample-integration-test>
- Radzikowski, M. (2022). AWS-SDK-Client-Mock. github.com.  
<https://github.com/m-radzikowski/aws-sdk-client-mock>
- Lighthouse Overview - Chrome for Developers. (2016, 27 september).  
Chrome for Developers.  
<https://developer.chrome.com/docs/lighthouse/overview/>