

Software Guidebook

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

1. Context	5
1.1. Rollen	5
1.1.1. Medewerker	6
1.1.2. Administrator	6
1.1.3. Ontvanger	6
2. Functioneel Overzicht	7
2.1. Functionaliteiten	7
2.2. User Stories	7
2.3. Services	8
2.3.1. AWS S3	8
2.3.2. Docker	8
2.3.3. EmailJS [Deprecated]	9
2.3.4. AWS SES	9
3. Kwaliteitsattributen	10
3.1. Functionality	10
3.2. Usability	10
3.3. Reliability	10
3.4. Performance	10
3.5. Supportability	11
3.6. +	11
4. Restricties	12
5. Principes	13
5.1. Code Kwaliteit	13
5.2. Git	13
5.3. AWS	13
6. Software Architectuur	14
6.1. System Context Diagram	14
6.2. Container Diagram	16
6.2.1. Front-End	16
6.2.2. Back-End	16
6.3. Component Diagram	17
6.3.1. Front-End	17
6.3.2. Back-End	18
6.4. Code Diagram	20
7. Code	21
7.1. Upload Router	21
7.2. AWS S3	24
7.3. Gevoelige informatie ophalen	26

7.4. Encryption	29
8. Data	30
8.1. Database structuur	30
8.2. Opslaglocatie	31
8.3. Verwijderen van data	31
8.4. Bestand limitaties	31
9. Infrastructuur Architectuur	32
9.1. Overzicht	32
9.2. Diagram	33
10. Deployment	34
10.1. Artifacts	34
10.2. Nginx	34
11. Werking en Ondersteuning	36
11.1. Lokale ontwikkeling:	36
11.2. Monitoring	37
11.3. Configuratie	37
12. Risicoanalyse	38
12.1. Beoordelingscriteria	38
12.2. Gevonden risico's	38
12.2.1. AWS services die de applicatie gebruikt liggen eruit	38
12.2.2. Door wetgeving in een ander land omtrent cryptografie krijgt de overheid van dit land toegang tot data	39
12.2.3. De (versleutelde) gevoelige informatie komt in verkeerde handen terecht	40
12.2.4. Connectie over publieke netwerken wordt niet middels encryptie gedaan	41
12.2.5. Gevoelige informatie wordt niet geheel verwijderd	42
13. Beslissing Logboek	43
13.1. T3 Stack	43
13.2. Docker	44
13.3. AWS	44
13.4. Elastic Beanstalk	44
14. Literatuurlijst	45

1. Context

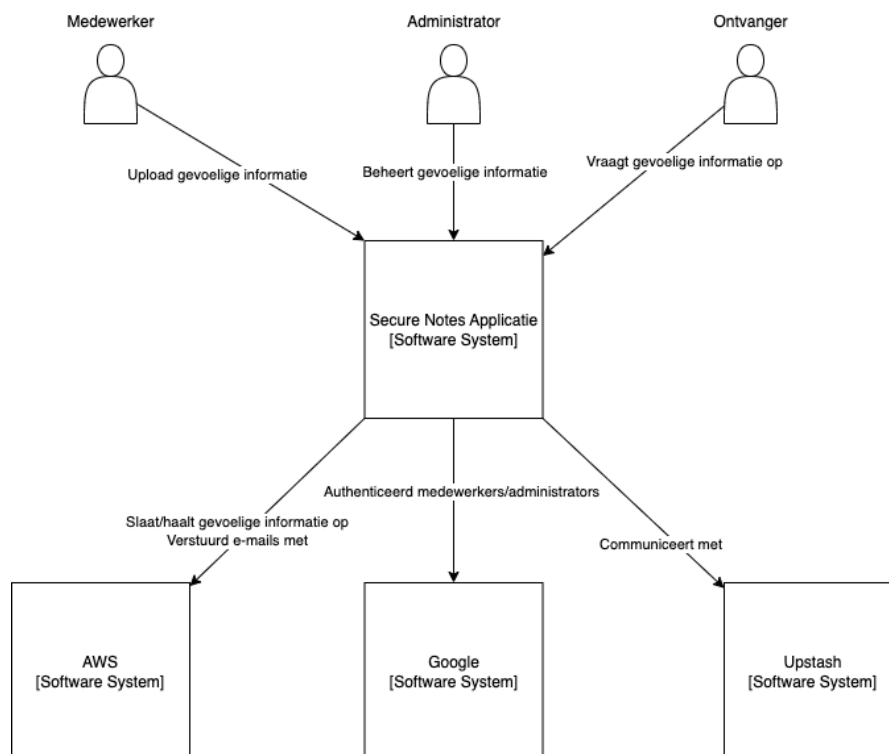
De oorspronkelijke context van SecureNotes.

Dit Software Guidebook is geschreven voor de webapplicatie "SecureNotes". Deze applicatie maakt het mogelijk voor medewerkers binnen het bedrijf Webbio om gevoelige informatie te sturen naar externe partijen, terwijl de strenge ISO 27001-beveiligingsnormen nageleefd worden.

Hedendaags werkt Webbio steeds meer met klanten gerelateerd met de overheid. Vandaar dat het van uiterste belang is om deze gevoelige informatie veilig te kunnen versturen. Dit is ook de directe aanleg om SecureNotes te realiseren.

1.1. Rollen

Binnen SecureNotes bevinden zich diverse rollen. Hoe deze rollen de webapplicatie kunnen beïnvloeden wordt onder in afbeelding 1 weergegeven in een context diagram.



Afbeelding 1: Context Diagram (volledige grootte in bijlagen)

1.1.1. Medewerker

Iedere medewerker binnen Webbio kan met een Google Workspace account gevoelige informatie uploaden en delen met externe partijen.

1.1.2. Administrator

Gebruikers die administrator rechten hebben in de Webbio Google Workspace zullen ook deze rol in de applicatie ontvangen. Een administrator kan net zoals een medewerker gevoelige informatie uploaden, echter zijn zij ook nog bevoegd om een overzicht van alle geüploade informatie te zien. Hiermee kunnen zij de geüploade informatie beheren.

1.1.3. Ontvanger

Dit zijn degene waarmee de gevoelige informatie gedeeld wordt. Een ontvanger kan door middel van een URL met de secret eenmalig de inhoud bekijken of downloaden. De gevoelige informatie zal daarna geheel verwijderd worden van de database.

Zie hoofdstuk 6.1 voor verdere uitleg over dit context diagram en daarbij horende externe systemen.

2. Functioneel Overzicht

Een overzicht van de functionaliteiten binnen Secure Notes.

2.1. Functionaliteiten

De SecureNotes applicatie bevat diverse componenten. De twee belangrijkste zijn voor het uploaden en ophalen van gevoelige informatie.

Het uploaden van informatie kan worden gedaan door een geautoriseerde Webbio medewerker. Deze upload kan in twee varianten komen: tekst of bestand. Daarbij moet er ook nog een vervaldatum van de upload worden geselecteerd en de e-mail van de ontvanger moet ingevuld zijn. Deze data wordt dan gecodeerd en naar de database of object storage geüpload. Zodra dit is voltooid wordt er automatisch een e-mail naar de ontvanger gestuurd met een melding dat er een upload klaar staat voor diegene. Daarnaast wordt er tegelijkertijd een secret weergegeven aan de medewerker, die diegene handmatig moet delen aan de ontvanger. Dit dient als een soort twee-stappen authenticatie. En hierdoor kan er niets met de gevoelige informatie gebeuren indien de upload naar het verkeerde e-mailadres is verstuurd.

Het ophalen van gevoelige informatie kan worden gedaan met de eerder genoemde e-mail. Hierin wordt een URL gegenereerd met een deel van de secret als parameter. Zodra er op de URL geklikt wordt, krijgt de gebruiker een melding waarin hij een secret in moet vullen. Dit is de secret die de medewerker heeft moeten delen. Indien deze twee secrets in combinatie komen kan de data gedecodeerd worden en gelezen/gedownload worden. Na het eenmalig ophalen wordt de gevoelige informatie geheel verwijderd uit de database.

Hiervoor is ook een uitgebreid ontwerp gemaakt waarop de applicatie is gebaseerd. Deze ontwerpen zijn terug te vinden in de bijlage: "Figma Design".

2.2. User Stories

Er zijn diverse user stories verwerkt binnen de applicatie. Deze user stories zijn samen met de opdrachtgever geprioriteerd door middel van MoSCoW. De

voortgang van alle user stories zijn terug te vinden op de backlog binnen de JIRA omgeving.

#	User Story	MoSCoW
1	Als medewerker wil ik gevoelige informatie kunnen delen met externe partijen	Must
2	Als medewerker wil ik gevoelige informatie kunnen uploaden	Must
3	Als externe partij wil ik de gevoelige informatie kunnen bekijken	Must
4	Als administrator wil ik een overzicht van alle systeem activiteiten	Could
5	Als opdrachtgever wil ik een veilige en goedwerkende applicatie	Should
6	Als medewerker/administrator wil ik kunnen inloggen op de applicatie	Must
7	Als medewerker wil ik een willekeurig gegenereerd secret kunnen ontvangen	Should

2.3. Services

SecureNotes maakt gebruik van diverse services om goed te kunnen functioneren.

2.3.1. AWS S3

Dit is waar alle gevoelige informatie in de vorm van bestanden veilig wordt opgeslagen. Ook kan er door middel van S3 een PresignedURL gegenereerd worden om de data later weer veilig op te halen.

2.3.2. Docker

Voor lokale ontwikkeling is er een Docker omgeving opgezet volgens de Webbio standaarden. Hierop draait een MySQL database om lokaal data op te slaan en te testen.

2.3.3. EmailJS [Depreciated]

Een open source npm package om eenvoudig mails te versturen zonder enige complexe back-end te hoeven configureren. EmailJS is gebruikt om automatisch mails te versturen naar ontvangers van de gevoelige informatie.

2.3.4. AWS SES

Een vernieuwende email service dat gebruikt wordt in plaats van EmailJS. Hiermee regelt Amazon geheel het automatisch versturen van e-mails.

3. Kwaliteitsattributen

De niet-functionele eisen geformuleerd volgens FURPS+

3.1. Functionality

- De gevoelige informatie moet worden verwijderd uit de database zodra het eenmalig geopend of gedownload is.
- De gevoelige informatie moet worden verwijderd indien de vervaldatum is overschreden.
- Het encryptie algoritme moet tenminste voldoen aan de standaarden van AES 256.
- Voor de administrator kan er een activiteitenlog worden bijgehouden van alle IPs die de gevoelige informatie hebben opgehaald. Hiermee kan bijvoorbeeld het verkeer alleen beperkt worden tot NL door middel van een firewall binnen AWS.

3.2. Usability

- Er moet een twee-stappen verificatie worden geïmplementeerd waarbij de verstuurder apart de secret aan de ontvanger moet delen om de gevoelige informatie te kunnen bekijken.
- Een ontvanger heeft een maximaal aantal pogingen van drie om het juiste secret in te vullen.

3.3. Reliability

- De applicatie moet minimaal vijf gebruikers tegelijk kunnen ondersteunen.

3.4. Performance

- De maximale laadtijd van de applicatie moet een score van >90% hebben volgens de Google PageSpeed insights (About PageSpeed Insights, z.d.).

3.5. Supportability

- Er moet zowel de mogelijkheid zijn om een bestand als platte tekst te kunnen uploaden.
- Er mag maar maximaal één bestand worden geupload.
- Het moet de laatste versie van Google Chrome en Mozilla Firefox kunnen ondersteunen.
- Er is geen maximale grootte per bestand upload, echter mag de applicatie hier niet van leiden indien het een groot bestand is.
- Alle file types mogen geupload worden door de gebruiker.
- Authenticatie moet volledig plaatsvinden binnen Google Sign-In.
- De applicatie moet responsief zijn op zowel desktop als mobiel formaat.

3.6. +

- De user interface moet volledig in het Engels zijn.

4. Restricties

De restricties van de Secure Notes applicatie.

Tijdens het ontwikkelen van de applicatie zijn er een aantal restricties, deze restricties zijn als volgt:

- De applicatie is gebouwd op de T3 Stack.
- De database draait op Docker MySQL.
- Alle bestanden worden opgeslagen binnen een AWS S3 bucket.
- De infrastructuur van de applicatie wordt gehost op een bestaande AWS.
- De applicatie wordt ontwikkeld door een teamgrootte van één persoon.
- De code wordt gereviewd door ook maar één persoon tijdens pull requests.
- De ontwikkelaar maakt gebruik van onbekende technieken, zoals AWS.
- De ontwikkelaar besteedt ook een deel van zijn tijd aan het opleveren van schooldocumenten.
- De ontwikkelingstijd van de applicatie is ongeveer vijf maanden.
- De applicatie is ontwikkeld en opgeleverd binnen een beperkte tijd van ongeveer vijf maanden.

Grotendeels van de restricties beïnvloeden het uiteindelijk opgeleverde resultaat of beperken de mogelijkheid tot uitbreiding indien iemand anders dit project overneemt.

5. Principles

De principes waaraan gehouden zijn tijdens het ontwikkelen

5.1. Code Kwaliteit

- Alle code voldoet volledig aan de linting standaard binnen Webbio.
- De naamgeving van variabelen past camelCase toe.
- Alle Vitetest testen slagen.
- Alle credentials worden opgeslagen in environment variabelen en gevalideerd met een middleware.
- Code dient self-documenting te zijn anders zullen er opmerkingen bij komen
- Code bevat geen code smells.

5.2. Git

- Er wordt gebruikgemaakt van feature-based development.
- Iedere branch bevat de afkorting van de betreffende JIRA taak.
- Iedere commit beschrijving moet in een zin duidelijk de inhoud beschrijven.
- Er wordt gebruikgemaakt van een master- en development branch.
- Voor elke merge is een pull request nodig dat gecontroleerd is door een andere ontwikkelaar..

5.3. AWS

- Alle infrastructuur draait op de bestaande AWS-omgeving van Webbio.
- Er wordt een aparte IAM rol gebruikt voor het genereren van de PresignedURLs (aangezien de key gepubliceerd wordt).

6. Software Architectuur

De architectuur van de software weergegeven met behulp van het C4 model geschreven door Simon Brown (Brown, 2006).

(Zie de bijlagen voor de diagrammen in volledige grootte)

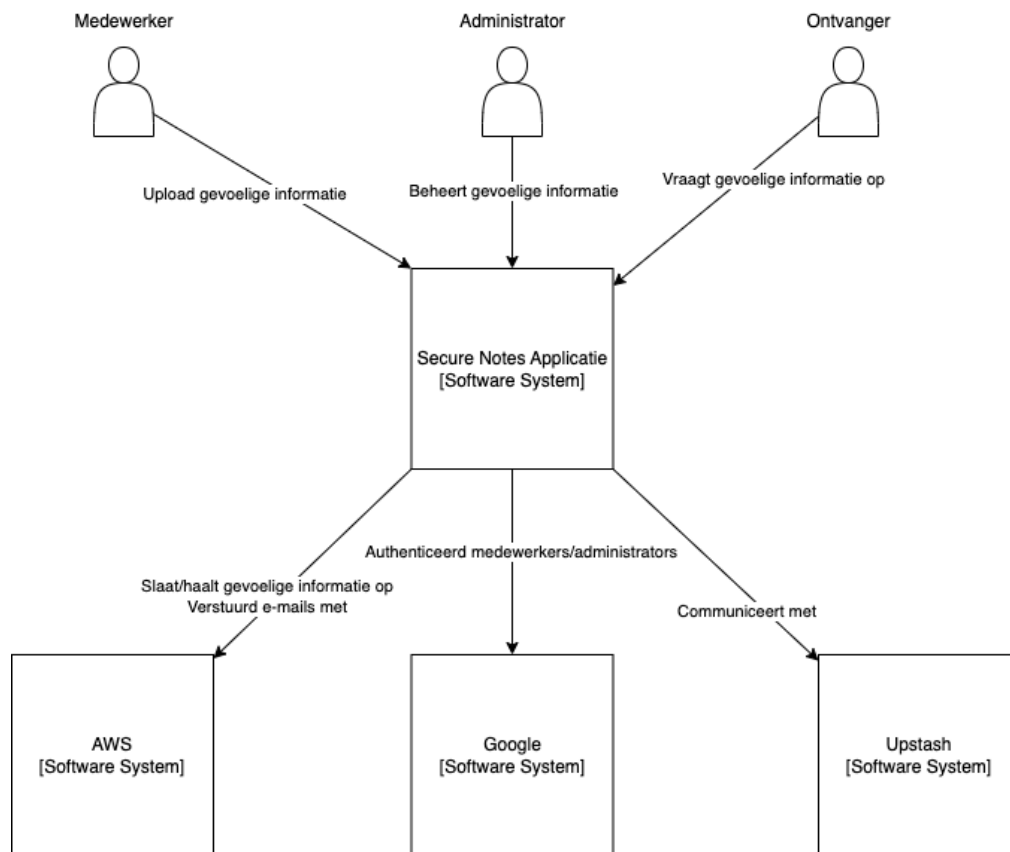
6.1. System Context Diagram

Onderstaand (*in afbeelding 2*) wordt weergegeven hoe het systeem “SecureNotes” communiceert met de rollen: medewerker, administrator en ontvanger. Daarnaast communiceert het systeem ook met externe systemen zoals AWS en Google.

De applicatie maakt gebruik van diverse services binnen AWS. Allereerst gebruikt het Amazon Simple Storage Service (S3), dit dient als een object storage voor alle geüploade bestanden. Daarnaast benut het Amazon Simple Email Service (SES) om eenvoudig e-mails te sturen naar andere gebruikers. En ten slotte wordt de gehele applicatie gehost op AWS via Elastic Beanstalk (EB), omdat Webbio al een bestaande AWS-omgeving heeft en deze is reeds ingericht volgens de ISO normen.

Ook wordt er Google gebruikt waarmee de Single Sign-On geconfigureerd wordt. Zodra een Webbio medewerker geauthenticeerd is door Google, dan worden de betreffende gebruikersinformatie opgehaald en naar de applicatie gestuurd.

Als laatste wordt er gebruikgemaakt van Upstash, een serverless data platform waarop Redis gedraaid kan worden om functionaliteiten, zoals een rate limit, toe te kunnen passen aan onze applicatie.



Afbeelding 2: System Context Diagram

6.2. Container Diagram

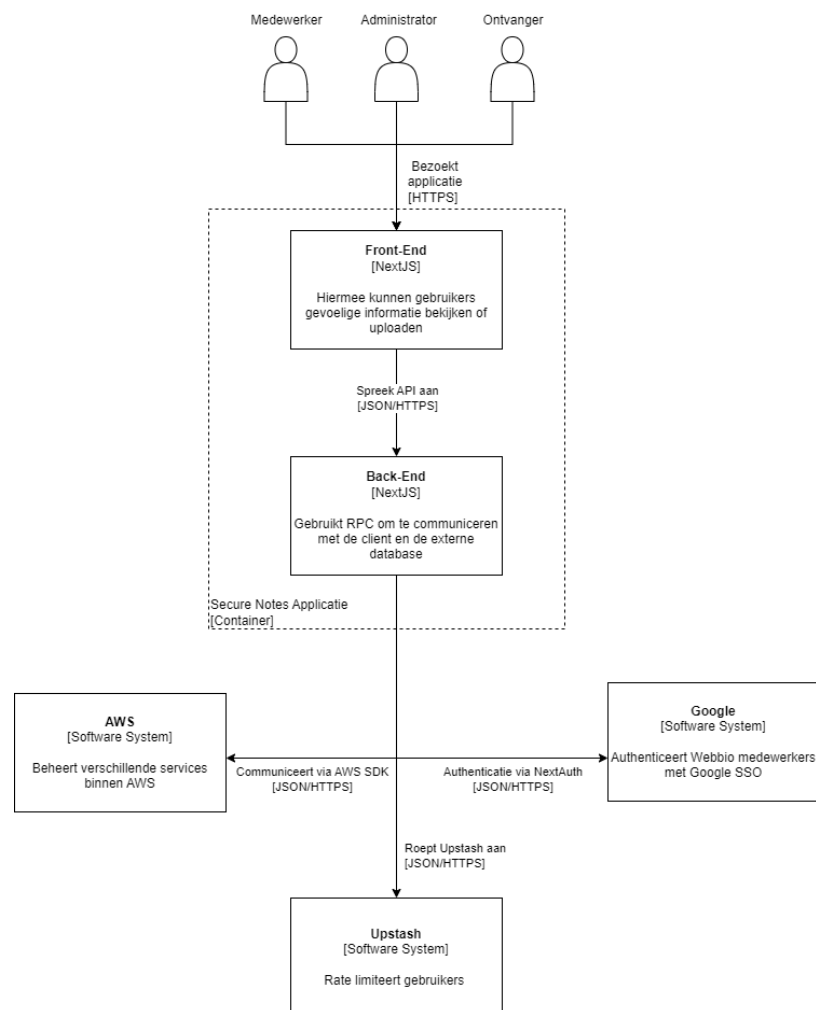
In afbeelding 3 is te zien hoe de Secure Notes applicatie onderscheiden kan worden tussen front- en back-end met behulp van NextJS.

6.2.1. Front-End

Een gebruiker kan de front-end benaderen door naar de betreffende website te gaan. Deze front-end kan dan met een API communiceren naar de back-end om gevoelige informatie op te slaan of op te halen afhankelijk van de rol van de gebruiker.

6.2.2. Back-End

Communiqueert met de verschillende externe systemen en zorgt dat de benodigde informatie opgehaald/opgeslagen wordt.



Afbeelding 3: Container Diagram

6.3. Component Diagram

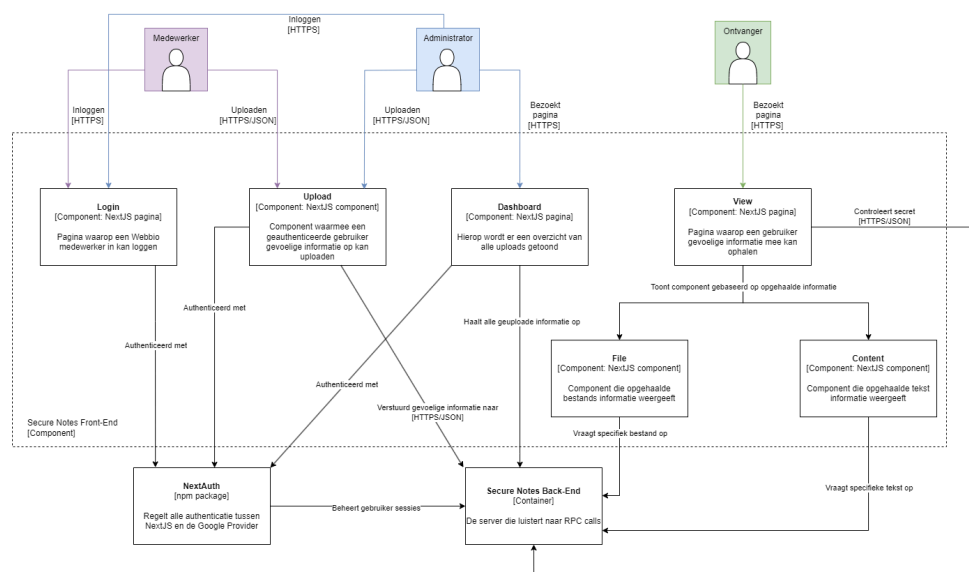
In afbeelding 4 en 5 wordt er naar de verschillende componenten gekeken binnen een container.

6.3.1. Front-End

Onderstaand (afbeelding 4) wordt het component diagram van de front-end getoond. Hieruit blijkt dat de medewerker/administrator kan inloggen met behulp van de NextAuth package.

Ook kan een gebruiker zichzelf authenticeren door een geldige sessie via NextAuth uit de back-end op te halen. Indien degene geauthenticeerd is kan de upload of dashboard pagina weergegeven worden (afhankelijk van rol). Het ophalen en uploaden van gevoelige informatie wordt allemaal gedaan in de back-end, deze endpoints worden met RPC aangeroepen.

Allerlaatst hoeft de ontvanger zich niet te authenticeren via NextAuth, maar wordt dit gedaan door een ontvangen secret. Deze secret wordt doorgestuurd naar de back-end om geauthenticeerd te worden. Zodra er een match is gevonden wordt de view pagina getoond met een bestand of tekst component. Dit is afhankelijk van het uploadtype.



Afbeelding 4: Component Diagram (Front-End)

6.3.2. Back-End

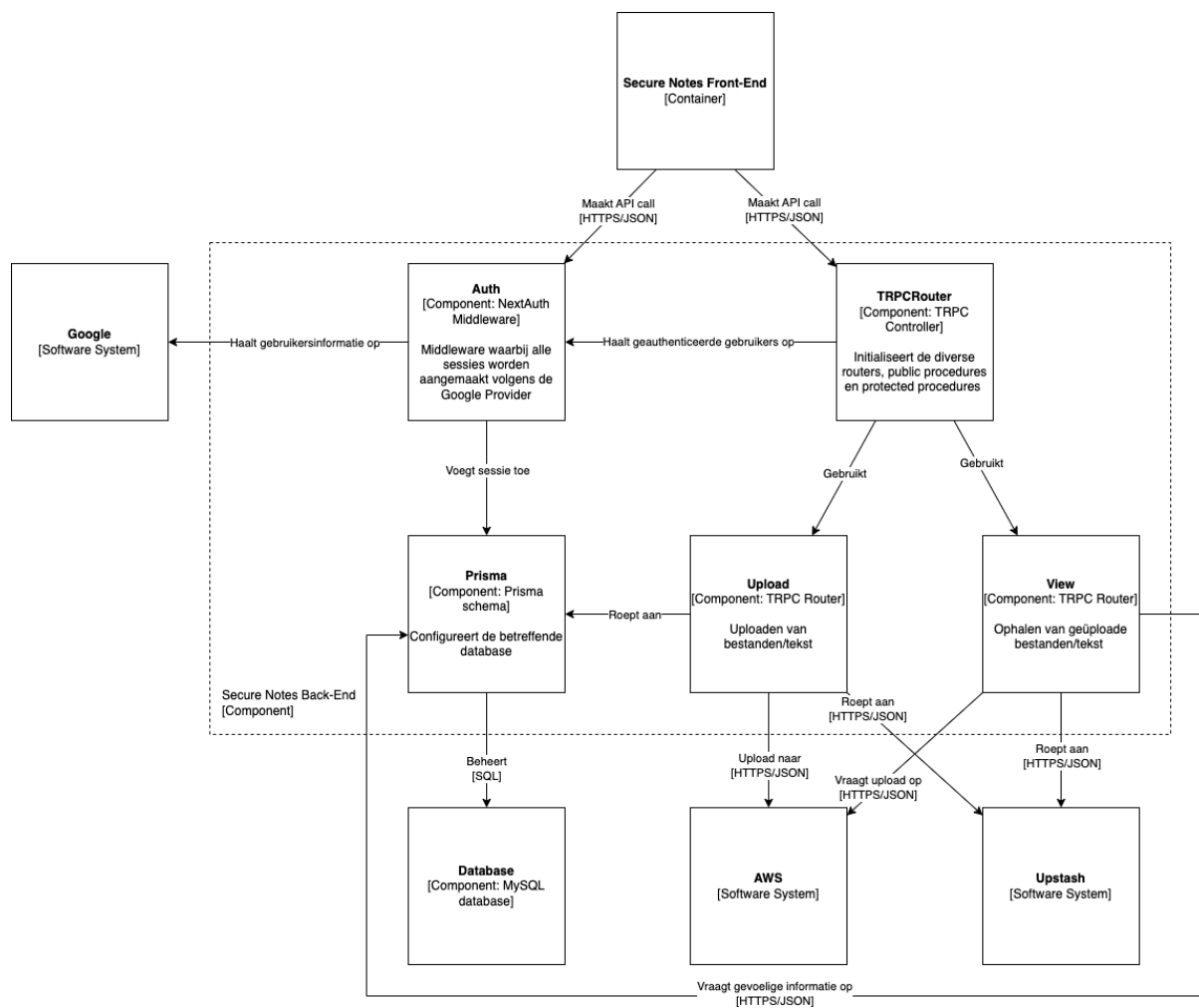
In afbeelding 5 is het component diagram van de back-end container weergegeven. Gebaseerd op de API call wordt de auth middleware aangeroepen of de TRPC controller. Deze auth middleware communiceert met de Google Provider om de gebruikersinformatie op te halen en maakt een sessie aan. Deze sessie wordt dan toegevoegd in de database door middel van Prisma.

Indien een TRPC route benaderd moet worden zal deze door de controller komen. Deze controller regelt de verschillende TRPC procedures. De protected procedures worden geauthenticeerd gebaseerd op de gemaakte sessies binnen de NextAuth middleware.

Ook verwijst deze controller naar twee TRPC routers: upload en view. Binnen de upload component wordt het uploaden van bestanden of tekst geregeld. Hierbij wordt de algemene informatie opgeslagen via Prisma in de MySQL database. Daarnaast worden de bestanden veilig opgeslagen in een Amazon S3 bucket. Uiteindelijk wordt Amazon SES ook aangeroepen om automatisch een e-mail te versturen naar de betreffende ontvanger.

Indien de view router wordt aangeroepen zal het mogelijk zijn om gevoelige informatie op te halen vanuit de AWS-omgeving of de database.

Zowel de upload als de view router communiceren met Upstash. Hierbij wordt er middels een Redis database eerst gecontroleerd of er nog niet aan de rate limit is voldaan. Indien dit wel zo is wordt de gehele mutatie naar de AWS en Prisma niet uitgevoerd.



Afbeelding 5: Component Diagram (Back-End)

6.4. Code Diagram

Er is gekozen om voor mijn applicatie geen code diagram te maken. Zoals Simon Brown vermeldt in zijn C4 Model documentatie is het meestal overbodig om een code diagram toe te voegen, wegens het feit dat deze simpelweg gegenereerd kunnen worden.

Het kost erg veel tijd om het diagram constant up-to-date te houden indien de code verandert. Daarnaast zijn sommige genoemde componenten volledig gegenereerd tijdens het initialiseren van de T3 Stack en andere zelfgeschreven componenten zijn zo geschreven dat het self-documenting code is. Ten slotte is er bij deze applicatie goed gelet op modulaire ontwikkeling en dus zou er niet veel toegevoegde waarde vanuit de code diagrammen komen.

Voor uitleg over de belangrijkste code binnen de applicatie kan er gekeken worden naar hoofdstuk 7.

7. Code

Onderstaand wordt er verdiept in de cruciale code binnen de Secure Notes applicatie.

7.1. Upload Router

De upload router bevat een 'create' procedure die alle logica bevat om een nieuwe upload uit te voeren. Allereerst is dit een protected procedure waardoor het alleen bereikbaar is indien de gebruiker geauthenticeerd is door NextAuth. Deze authenticatie wordt geregeld binnen de TRPC middleware (zie afbeelding 6).

```
const enforceUserIsAuthenticated = t.middleware(({ ctx, next }) => {
  if (!ctx.session?.user) {
    throw new TRPCError({ code: 'UNAUTHORIZED' });
  }
  return next({
    ctx: {
      // infers the `session` as non-nullable
      session: { ...ctx.session, user: ctx.session.user }
    }
  });
});

export const protectedProcedure = t.procedure.use(enforceUserIsAuthenticated);
```

Afbeelding 6: Protected Procedure Middleware Code

Daarna wordt door middel van Zod, een validatie package (NPM: *ZoD*, z.d.), alle input gecontroleerd. Hierbij moest er ook gebruikgemaakt worden van een `refine()` om er zeker van te zijn dat content gevuld is indien het upload type "type" is en dat file gevuld is indien het upload type "file" is (zie afbeelding 7).

Indien de input goedgekeurd is, wordt de verloopdatum ingesteld en wordt daarbij gelijk een unieke secret gegenereerd. De secret wordt verdeeld in twee delen, een deel wordt uiteindelijk teruggegeven aan de gebruiker en het andere deel wordt automatisch verstuurd naar de ontvanger. Alleen indien de ontvanger het andere deel van de gebruiker ontvangt kan de gevoelige informatie bekeken worden. De werking hiervan zal in hoofdstuk 7.3 uitgelegd worden.

```

.input(
  z
  .object({
    type: z.union([z.literal('file'), z.literal('text')]),
    content: z.string().optional(),
    file: z
      .object({
        name: z.string(),
        size: z.number(),
        contentType: z.string()
      })
      .optional(),
    duration: z.string(),
    email: z.string()
  })
  .refine(
    (data) => {
      if (data.type === 'text') return data.content !== undefined && data.content.trim() !== '';
      else if (data.type === 'file') return data.file?.size !== 0;
      return true;
    },
    {
      message: "Either 'content' or 'file' is missing."
    }
  )
)

```

Afbeelding 7: Zod input validation

De data wordt dan opgeslagen in het upload model. In het geval dat de filetype “file” is, worden alle metadata van de bestanden nog opgeslagen in een aparte model. Hierbij is dan een relatie gedefinieerd om terug te verwijzen naar het upload model (zie afbeelding 8).

```

model Upload {
  id      String  @id @default(cuid())
  type     String  // File or Text
  content  String? // Type: Text -> Content will be filled
  createdAt DateTime @default(now())
  expiresAt DateTime
  user     User    @relation(fields: [userId], references: [id])
  userId   String
  file     File?
  secret   String  @unique
}

model File {
  id      String  @id @default(cuid())
  name     String
  size     Int
  contentType String
  upload   Upload @relation(fields: [uploadId], references: [id], onDelete: Cascade)
  uploadId String  @unique
}

```

Afbeelding 8: Prisma model schema's

Ten slotte wordt er nog met Amazon SES gecommuniceerd om een mail te versturen naar de ontvanger. Deze mail dient als een soort van notificatie met een url naar de overzichtspagina en bevat als parameter een deel van de secret (zie afbeelding 9).

```
export const sendEmail = async (to: string, secret: string, fromName: string, fromEmail: string): Promise<any> => {
  let sesClient;

  try {
    sesClient = new SESClient({ region: env.REGION });
  } catch {
    throw new Error(`Could not create SES client`);
  }

  try {
    const { Identities } = await sesClient.send(new ListIdentitiesCommand({}));

    if (!Identities?.includes(to || env.EMAIL_FROM)) throw new Error(`Email address ${to} is not verified`);
    if (!to.length || !secret.length || !fromName.length || !fromEmail.length) throw new Error(`Missing parameters`);
  } catch {
    throw new Error(`Could not verify identity`);
  }

  await sesClient.send(
    new SendEmailCommand({
```

Afbeelding 9: Amazon SES implementatie

7.2. AWS S3

Behalve dat de metadata van een bestand geüpload moet worden via Prisma, moet het daadwerkelijke bestand opgeslagen worden in een AWS S3 bucket. Deze buckets zijn namelijk specifiek gemaakt voor het veilig en efficiënt bewaren van objecten op de cloud.


Nadat het bestand op de front-end is toegevoegd aan de file input wordt het omgezet in FormData en doorgestuurd naar de back-end (zie afbeelding 10)

```
const uploadFile = async (currentFile: File, id: string): Promise<Response> => {  
  const formData = new FormData();  
  formData.append('file', currentFile);  
  formData.append('id', id);  
  
  const response = await fetch('/api/files/upload', {  
    method: 'POST',  
    body: formData,  
  });  
  
  return response;  
}
```

Afbeelding 10: Front-end code FormData

Zodra op de back-end moet de FormData worden afgelezen. Wegens de fetch request wordt de FormData getransformeerd in multipart/form-data. Dit content type kunnen wij lezen met behulp van een package genaamd formidable.

Daarna maken wij een nieuwe instantie aan van een S3Client door middel van de aws-sdk package. En met deze S3Client kunnen wij dan S3 aanroepen om een nieuw bestand toe te voegen aan de bucket (zie afbeelding 11 & 12).

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	 Schermafbeelding 2023-11-17 om 11.49.56.png	png	November 20, 2023, 14:33:34 (UTC+01:00)	393.0 KB	Standard

Afbeelding 11: Resultaat binnen AWS S3 bucket

```
client = new S3Client({
  region: env.REGION,
  credentials: {
    accessKeyId: env.AWS_ACCESS_KEY_ID,
    secretAccessKey: env.AWS_SECRET_ACCESS_KEY,
    sessionToken: env.AWS_SESSION_TOKEN
  }
});

command = new PutObjectCommand({
  Bucket: env.BUCKET_NAME,
  Key: `${fields.id[0]}/${files.file[0]?.originalFilename}`,
  Body: fs.readFileSync(files.file[0]?.filepath)
});

await client.send(command);
```

Afbeelding 12: Implementatie aws-sdk

7.3. Gevoelige informatie ophalen

Zoals in hoofdstuk 7.1 is benoemd kan de gevoelige informatie worden opgehaald door middel van de gegenereerde secret. Deze secret wordt opgedeeld in twee delen, het eerste deel bevindt zich in de url van de verzuurde mail en het andere deel moet de medewerker handmatig verzuren. Dit dient dan als een soort two-factor authentication om de kans dat de gevoelige informatie bij ongewensten terechtkomt te verkleinen.

Op de front-end wordt een input bijgehouden met een React Hook. Zodra er op een knop wordt geklikt zal deze invoer in combinatie met de secret in de url naar de back-end worden gestuurd (zie afbeelding 13).

```
const [secretInput, setSecretInput] = useState<string>("");
const router = useRouter();

const handleView = (): void => {
  const key = router.query.key
  const secret = key + secretInput;

  mutate({ secret })
}
```

Afbeelding 13: Secret ophalen en verzuren

Op de back-end wordt deze ingevulde secret vergeleken met secrets van bestaande uploads. Zodra er een match is, worden de belangrijke informatie teruggestuurd (zie afbeelding 14).

```

.mutation(async ({ ctx, input }) => {
  const upload = await ctx.db.upload.findFirst({
    where: {
      secret: input.secret
    },
    select: {
      type: true,
      content: true,
      file: true,
      user: {
        select: {
          id: true
        }
      }
    }
  });

  if (!upload) {
    throw new Error('Upload not found');
  }
}

```

Afbeelding 14: Find upload query

De inhoud (content) zal gedecodeerd en daarna terug worden gegeven indien het type "text" is. Hoe het cryptografische gedeelte in elkaar zit wordt in hoofdstuk 7.4 verteld.

In het geval van een file upload zal er weer een nieuwe S3Client met aws-sdk worden aangemaakt. Met deze S3Client zal dan een PresignedURL worden aangemaakt. Dit geeft tijdelijk toegang tot de daadwerkelijke inhoud van het bestand (zie *afbeelding 15*). Ook hierbij moeten we nog de metadata decoderen.

```

if (upload.type === 'file') {
  let client;
  let command;
  let url;
  try {
    client = new S3Client({
      region: env.REGION,
      credentials: {
        accessKeyId: env.EXTERNAL_AWS_ACCESS_KEY_ID,
        secretAccessKey: env.EXTERNAL_AWS_SECRET_ACCESS_KEY
      }
    });
  } catch {
    throw new Error('Credentials are invalid');
  }

  if (!upload.file) throw new Error('Invalid file data');

  try {
    command = new GetObjectCommand({
      Bucket: env.BUCKET_NAME,
      Key: `${upload.user.id}/${upload.file.name}`
    });
  } catch {
    throw new Error('File not found');
  }
  try {
    url = await getSignedUrl(client, command, { expiresIn: 3600 });

    return {
      file: {
        name: decrypt(upload.file.name, input.secret),
        size: upload.file.size,
        contentType: upload.file.contentType
      },
      type: upload.type,
      url: url
    };
  } catch {
    throw new Error('Could not create url');
  }
}

```

Afbeelding 15: Genereer PresignedURL code

7.4. Encryption

Het coderen en decoderen van gevoelige informatie wordt gedaan met behulp van de ingebouwde “crypto” package binnen NodeJS. Deze crypto package biedt verschillende cryptografie toe met verschillende encryptie algoritmes. In mijn geval heb ik gekozen voor AES 256 (zie onderzoeksverslag voor uitleg over deze keuze).

Ik heb een helper functie geschreven waarmee ik verschillende inputs kan veranderen in cipher tekst en weer terug kan lezen in plain tekst indien de secret overeenkomt (zie afbeelding 16).

```
export const encrypt = (plaintext: string, secret: string): string => {
  const ciphertext = CryptoJS.AES.encrypt(plaintext, secret).toString();
  return ciphertext;
};

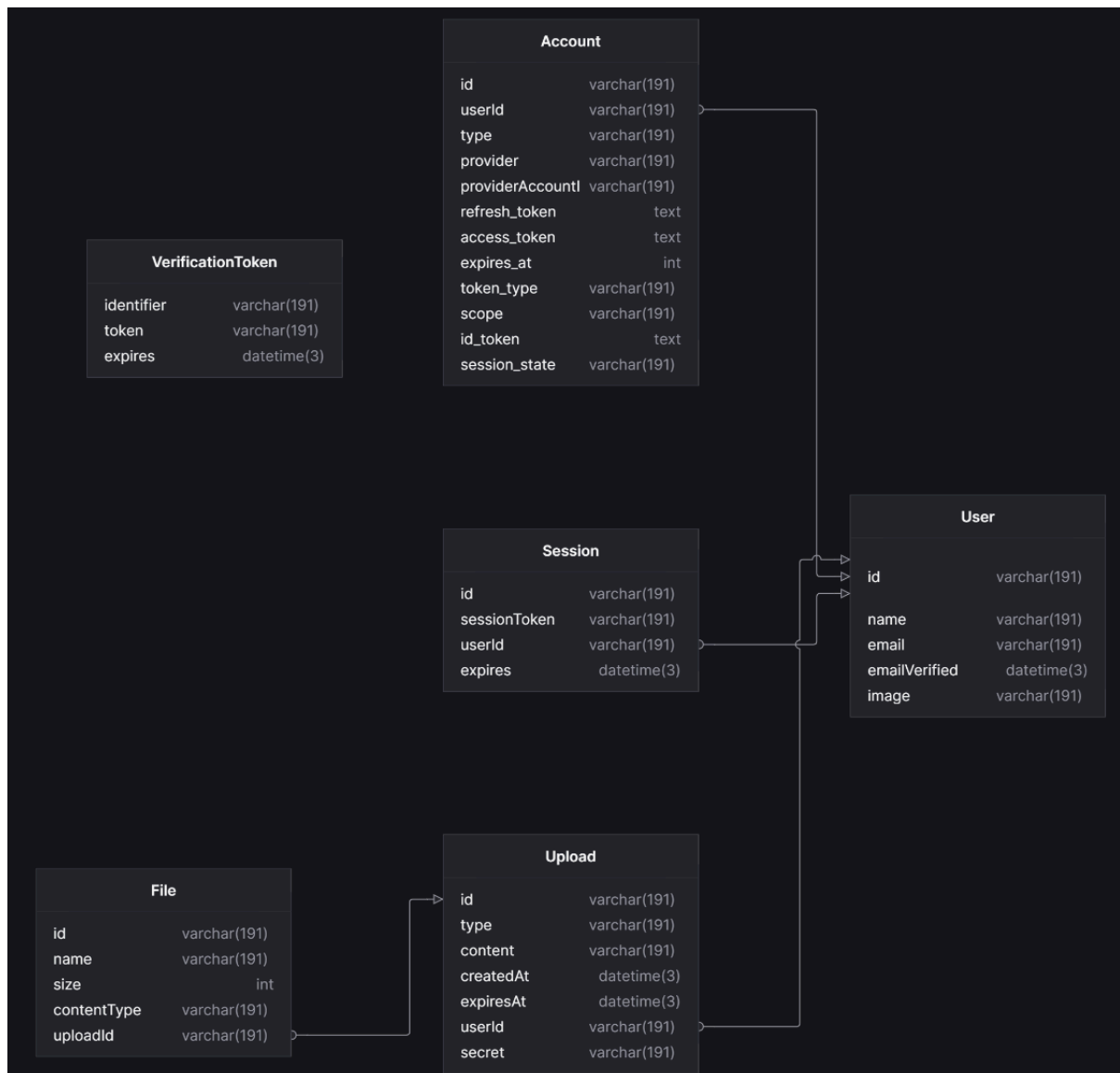
export const decrypt = (ciphertext: string, secret: string): string => {
  const bytes = CryptoJS.AES.decrypt(ciphertext, secret);
  const plaintext = bytes.toString(CryptoJS.enc.Utf8);
  return plaintext;
};
```

Afbeelding 16: Encryptie helper functie

8. Data

In dit hoofdstuk wordt beschreven hoe exact de data opgeslagen en gebruikt wordt.

8.1. Database structuur



Afbeelding 17: Gegeneerde afbeelding van huidige database structuur

De bovenstaande afbeelding (zie afbeelding 17) laat de databasestructuur zien binnen MySQL voor zowel de ontwikkel- als productieomgeving.

VerificationToken, Session, Account en User zijn standaard modellen gegenereerd door NextAuth en zijn verplicht voor de authenticatie. File en Upload zijn zelfgemaakte modellen die gerelateerd zijn met elkaar.

8.2. Opslaglocatie

Alle gevoelige informatie wordt op een MySQL database opgeslagen. Daarnaast worden de daadwerkelijk geüploade bestanden opgeslagen op een Amazon AWS S3 omgeving. Aangezien deze AWS omgeving van Webblio zelf is, blijft Webblio volledig de eigenaar van de data.

8.3. Verwijderen van data

Alle gevoelige informatie wordt volledig verwijderd zodra het eenmalig is bekeken of gedownload is. Indien het niet opgehaald is, wordt het na de ingestelde verlooptdatum alsnog volledig verwijderd.

8.4. Bestand limitaties

Rondom het uploaden van bestanden zijn er geen expliciete eisen. Iedere bestandstype mag geüpload worden. Daarnaast is er door de opdrachtgever geen specifieke maximale bestandsgrootte bepaald en dus zal deze gelijk zijn aan het limiet binnen AWS.

9. Infrastructuur Architectuur

In dit hoofdstuk wordt de infrastructuur van de applicatie beschreven.

9.1. Overzicht

De gehele infrastructuur bestaat uit meerdere AWS services die de applicatie ondersteunen. Allereerst wordt er middels CodePipeline een artifact gemaakt van de gehele applicatie die opgeslagen wordt in S3. Daarna wordt Elastic Beanstalk (EB) gebruikt om een Elastic Compute Cloud (EC2) instantie te starten waarop Amazon Linux 2023 draait. Deze EC2 instantie is door EB geconfigureerd om de eerder gemaakte artifact te bouwen binnen een Docker container en daarna met de database te verbinden die in een Relational Database Service (RDS) draait.

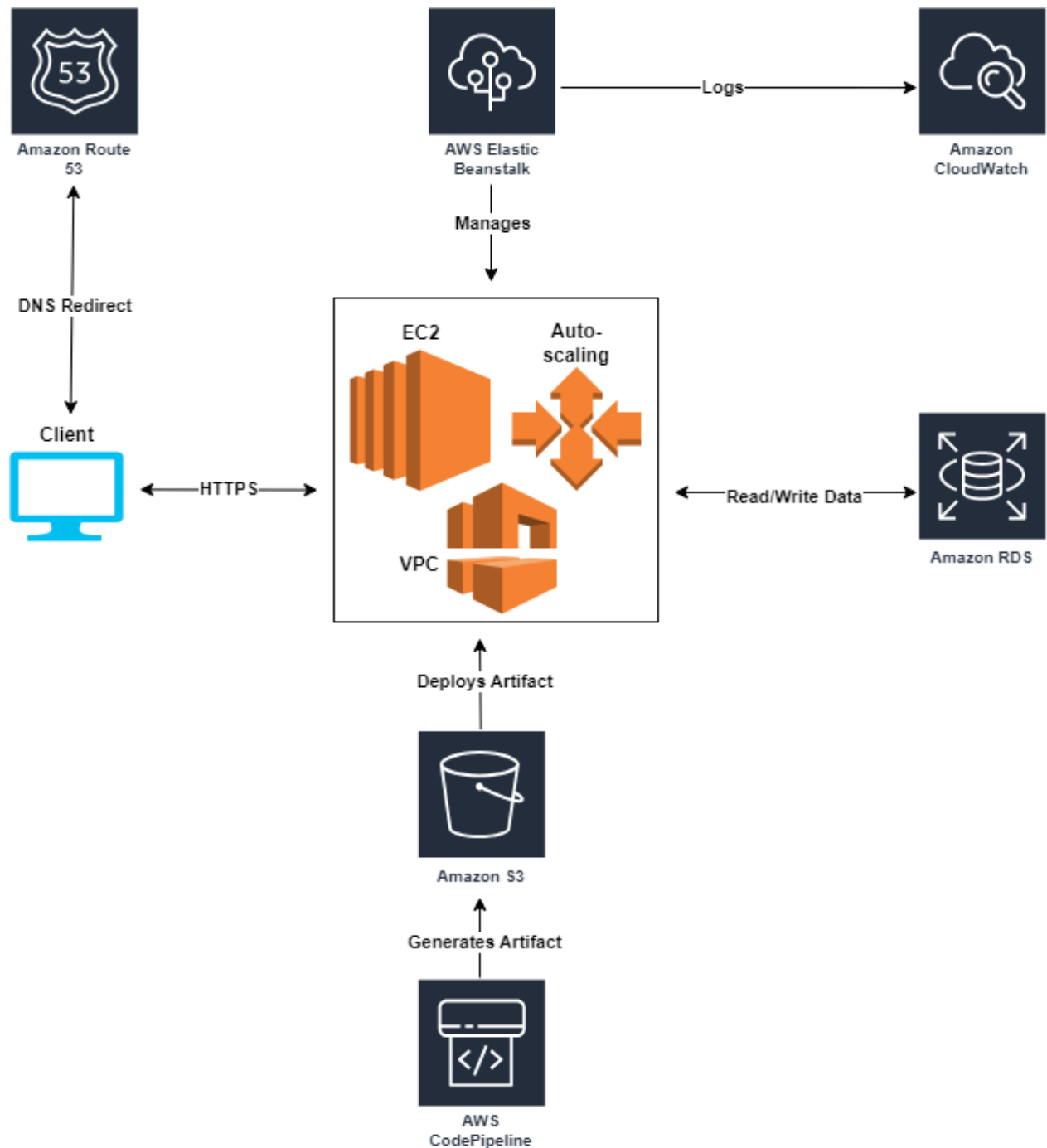
Verder wordt Elastic Beanstalk gebruikt om belangrijke instellingen, zoals de environment variables, load balancer en de instance type, in dit geval t2.micro, te beheren. En om gebruik te maken van Log Streaming om de huidige status en prestaties van de applicatie te kunnen bekijken. Deze logs worden geleverd door weer een andere service die verbonden is met EB genaamd Cloudwatch.

Ten slotte wordt er gebruikgemaakt van Route53 om een domein van Webbio aan de EC2 instantie te koppelen. Hierdoor is het mogelijk om eigen DNS records toe te voegen, zodat de applicatie niet alleen gelimiteerd is door de sandbox mode van Amazon Simple Email Service (SES).

Een geverifieerde domein is namelijk een verplichting om op productieniveau gebruik te kunnen maken van SES (Moving out of the Amazon SES sandbox - Amazon Simple email service, z.d.) en daarom is Route53 de geschikte kandidaat, aangezien hierdoor alle gebruikte services binnen AWS zijn.

9.2. Diagram

In de onderstaande afbeelding is de infrastructuur van de applicatie weergegeven.



Afbeelding 19: Architectuur Diagram

10. Deployment

Onderstaand wordt uitgelegd hoe de deployment van de applicatie tot stand komt.

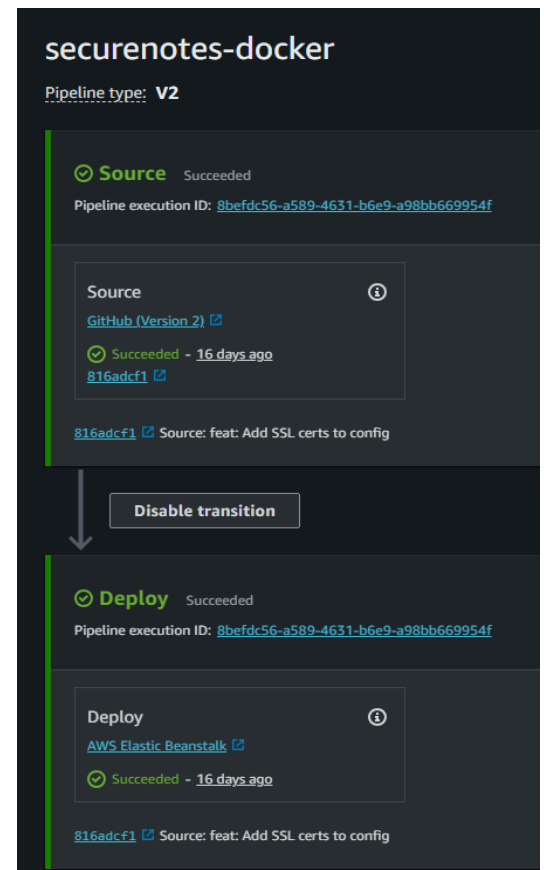
10.1. Artifacts

Zoals te zien is in afbeelding 19 van hoofdstuk 9.2 wordt er gebruikgemaakt van zogenaamde artifacts om de gehele applicatie te bouwen. Deze artifacts worden door CodePipeline gegenereerd vanuit de gekoppelde GitHub repository en opgeslagen in S3.

Zodra de gehele artifact klaar is met genereren wordt er gecommuniceerd met Elastic Beanstalk en zal de opgeslagen artifact worden gebouwd binnen een Docker container op een EC2 instantie.

Deze docker container leest dan de Dockerfile waarin alle aanwijzingen staan hoe exact het gedeployed moet worden.

Hiervoor is er ook een overzicht dat het gehele proces bijhoudt (zie afbeelding 20).



Afbeelding 20: Overzicht CP

10.2. Nginx

Zodra de applicatie middels EB is gedeployed maakt de website gebruik van HTTP. Echter is het gewenst om een beveiligde verbinding te hebben met de gebruiker en dus zal er HTTPS moeten worden toegepast. Hiervoor wordt Certbot gebruikt, een tool die certificaten kan genereren en configureren binnen Nginx zodat browsers de identiteit van webserver kunnen verifiëren (*CertBot 2.7.0.Dev0 documentation*, z.d.). Certbot stelt daarmee automatisch de nginx.conf bestand in zodat er een certificaat opgehaald wordt vanuit Let's Encrypt, een geauthenticeerde

certificeringsinstantie gelanceerd door Mozilla en EFF. Ook configureert Certbot automatische redirects indien de gebruiker naar HTTP in plaats van HTTPS gaat in de browser.

Het enige nadeel hiervan is dat de gehele nginx configuratie gereset wordt indien de EC2 instantie verdwijnt. Dit kan gebeuren wanneer de instantie crasht, maar ook bijvoorbeeld wanneer er te veel gebruikers op de applicatie zijn en EC2 automatisch nieuwe instanties toevoegt wegens autoscaling.

Om dit te voorkomen bevindt er in de github repository een '.ebextensions' folder. Deze wordt automatisch tijdens het deployen herkend door EB en hierbinnen worden standaardwaarden meegegeven die tijdens het initialiseren van de EC2 instantie direct toegepast worden (zie *afbeelding 21*).

```
1  files:
2    '/etc/nginx/nginx.conf':
3      owner: root
4      group: root
5      mode: '000644'
6      content: |
7        # Elastic Beanstalk Nginx Configuration File
8
9        user  nginx;
10       worker_processes  auto;
11       error_log  /var/log/nginx/error.log;
12       pid        /var/run/nginx.pid;
13       worker_rlimit_nofile  200000;
14
15       events {
16         worker_connections  1024;
17       }
18
19       http {
20         include        /etc/nginx/mime.types;
21         default_type   application/octet-stream;
22
23         access_log     /var/log/nginx/access.log;
24
25
26         log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
27                           '$status $body_bytes_sent "$http_referer" '
28                           '"$http_user_agent" "$http_x_forwarded_for"';
29
30         include conf.d/*.conf;
```

Afbeelding 21: Code voorbeeld .ebextensions

11. Werking en Ondersteuning

In dit hoofdstuk wordt uitgelegd hoe de applicatie werkt en ondersteund wordt.

11.1. Lokale ontwikkeling:

Voordat er lokaal gewerkt kan worden aan de bestaande applicatie zijn er een aantal vereiste:

- NodeJS 18.X of hoger
- Docker
- Lokale MySQL database
- AWS CLI

Zodra al deze vereisten zijn geïnstalleerd en geconfigureerd, kunnen de volgende stappen worden gevolgd.

1. **Clone** de bestaande code vanuit GitHub.
2. Voeg een **.env bestand** toe met de waardes vanuit `.env.example`.
De AWS credentials zelf hoeven niet meegegeven te worden in de .env, maar worden automatisch herkend in de code indien AWS CLI is geconfigureerd.
3. Voer **npm install** uit om alle packages te installeren.
4. Voer **npx prisma generate** uit om de database schema te scaffolden.
5. Nadat alles is ingesteld kan de applicatie lokaal bekeken worden door **npm run dev** uit te voeren.
6. De applicatie draait nu op <http://localhost:3000/>.

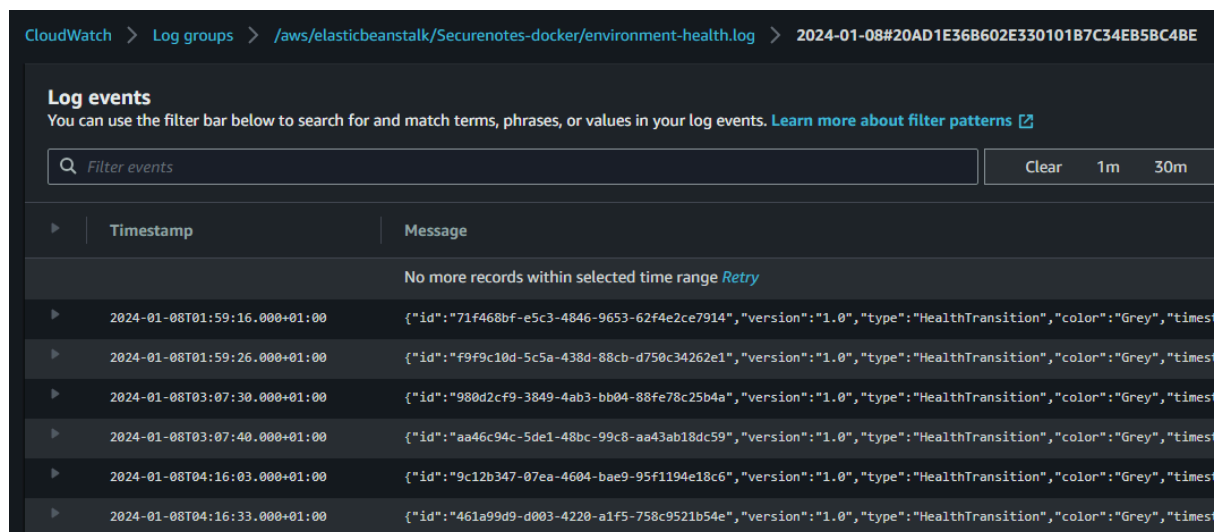
Extra opmerkingen:

- Indien er een testrapport gemaakt zal moeten worden met alle code coverage kan er **npm run test** worden uitgevoerd.
- Indien de database overzichtelijk bekeken en beheerd wilt worden kan er **npx prisma studio** uitgevoerd worden.
- Indien er veranderingen gemaakt worden aan de database in `schema.prisma` moet er **npx prisma db push** uitgevoerd worden om de database tabellen opnieuw te genereren en alles up to date te houden.

11.2. Monitoring

Om de draaiende applicatie te monitoren kan er gebruikgemaakt worden van CloudWatch binnen AWS. Op de EC2 instantie is namelijk Log Streaming ingesteld waarmee er real-time alle logs te zien zijn (zie *afbeelding 22*).

Daarnaast geeft CloudWatch nog andere nuttige informatie weer zoals de huidige performance van de applicatie, uptime, gezondheid, etc. Deze omgeving is dus erg handig om problemen mee te diagnosticeren.



Afbeelding 22: CloudWatch logs

11.3. Configuratie

Alle huidige AWS diensten binnen de Webbio omgeving zijn al ingesteld. Indien hier veranderingen aan moeten worden toegevoegd, kan er naar de configuratiepagina op Elastic Beanstalk verwijst worden. Alle wijzigingen activeren automatisch een nieuwe deployment met de meest actuele wijzigingen. Het kan daarom eventjes duren voordat de wijzigingen daadwerkelijk live staan.

12. Risicoanalyse

Hieronder vindt de risicoanalyse plaats van de applicatie.

Tijdens het ontwikkelen en testen is er gekeken naar potentiële kwetsbaarheden of risico's die zouden kunnen plaatsvinden. Iedereen heeft een tegenmaatregel om dit geheel te voorkomen of om de schade te beperken indien het toch gebeurt.

12.1. Beoordelingscriteria

Er wordt gekeken naar de volgende factoren:

- **Risico type:** Onder welke soort risico valt het? (integriteit, vertrouwelijkheid of beschikbaarheid)
- **Getroffen item:** Welke categorie wordt beïnvloed door het gevonden risico? (data of proces)
- **Bestaande maatregelen:** Zijn er inmiddels al bestaande maatregelen tegen het risico ondernomen?
- **Kwetsbaarheid:** Wat voor kwetsbaarheid neemt het risico met zich mee?
- **Impact:** Hoeveel impact heeft het risico? (laag, matig of hoog)
- **Waarschijnlijkheid:** Hoe waarschijnlijk is het dat het risico zich per ongeluk of opzettelijk voorkomt? (laag, matig of hoog)
- **Risicobehandeling:** Gedetailleerd plan om het risico geheel te voorkomen of schade te beperken.
- **Resterende impact:** Hoeveel impact heeft het risico na de risicobehandeling? (laag, matig of hoog)
- **Resterende waarschijnlijkheid:** Hoe waarschijnlijk is het dat het risico voor gaat komen na de risicobehandeling? (laag, matig of hoog)

12.2. Gevonden risico's

12.2.1. AWS services die de applicatie gebruikt liggen eruit

- **Risico type:** Beschikbaarheid, het uitvallen van AWS-services beïnvloedt de beschikbaarheid van de applicatie.

- **Getroffen item:** Data, dit heeft invloed op de gegevens en het verwerken van de gegevens.
- **Bestaande maatregelen:** AWS beschikt over vele maatregelen zoals back-up strategieën, maar deze staan nog uit.
- **Kwetsbaarheid:** Indien een of meerdere AWS-services uitvallen is er een kans op gegevensverlies of een niet geheel functionerende applicatie.
- **Impact:** Hoog, de impact van het uitvallen van diverse AWS-services is hoog aangezien het de beschikbaarheid van SecureNotes en de toegang tot belangrijke gevoelige informatie kan beïnvloeden.
- **Waarschijnlijkheid:** Middelmatig, de waarschijnlijkheid van het uitvallen van een AWS-service kan variëren, dit is afhankelijk van de stabiliteit van de huidige infrastructuur en de configuratie van de applicatie.
- **Risicobehandeling:** Hiervoor kan er een overweging worden gemaakt om redundante AWS-services op te zetten, back-up- en herstelplannen te implementeren, en actief gebruik te maken van monitoring- en waarschuwingssystemen om snel te reageren op een eventuele uitval.
- **Resterende impact:** Laag/Middelmatig, afhankelijk van de effectiviteit van de genomen maatregelen.
- **Resterende waarschijnlijkheid:** Laag, met de genomen risicobehandeling kan de waarschijnlijkheid van het risico lager worden.

12.2.2. Door wetgeving in een ander land omtrent cryptografie krijgt de overheid van dit land toegang tot data

- **Risico type:** Integriteit, dit beïnvloedt de integriteit van alle gevoelige informatie door toegang te geven aan de overheid.
- **Getroffen item:** Data, dit heeft invloed op de categorie "data", omdat het risico de toegang tot en mogelijk de controle over de data kan beïnvloeden.
- **Bestaande maatregelen:** Gevoelige informatie wordt encrypted opgeslagen.
- **Kwetsbaarheid:** Het risico hiervan is het eventueel blootstellen van de gebruikte data-encryptie strategie aangezien de gevoelige informatie publiekelijk wordt.
- **Impact:** Hoog, hierdoor kan gevoelige informatie in verkeerde handen vallen en mogelijk de reputatie van Webbio beschadigen.

- **Waarschijnlijkheid:** Laag, voorlopig zijn alle stakeholders nog in Nederland, maar de waarschijnlijkheid zou kunnen verhogen indien er met klanten uit andere landen wordt samengewerkt.
- **Risicobehandeling:** Overweging om juridisch advies in te winnen om te evalueren hoe er kan worden voldaan aan de wetgeving in het betreffende land zonder de vertrouwelijkheid van de data in gevaar te moeten brengen. Bijvoorbeeld door de locatie van de gegevensopslag te wijzigen om te voldoen aan de wetgeving.
- **Resterende impact:** Laag/Middelmatig, na het implementeren van de bovengenoemde maatregel kan de impact variëren, maar het doel is om deze zo laag mogelijk te houden.
- **Resterende waarschijnlijkheid:** Laag/Middelmatig, hetzelfde argument als bovenstaand.

12.2.3. De (versleutelde) gevoelige informatie komt in verkeerde handen terecht

- **Risico type:** Vertrouwelijkheid, dit risico gaat omtrent de vertrouwelijkheid van gevoelige informatie.
- **Getroffen item:** Data, omdat het betrekking heeft op de beveiliging en toegang tot gevoelige gegevens.
- **Bestaande maatregelen:** Gevoelige informatie wordt encrypted opgeslagen.
- **Kwetsbaarheid:** Het risico kan kwetsbaarheden blootleggen in de beveiligingsmaatregelen die zijn genomen om gevoelige informatie te beschermen
- **Impact:** Hoog, omdat het lekken van gevoelige informatie tot financiële schade, juridische consequenties en reputatieschade kan leiden.
- **Waarschijnlijkheid:** Middelmatig, de waarschijnlijkheid van dit risico kan variëren, afhankelijk van de kwaliteit van de beveiligingsmaatregelen en de mogelijke bedreigingen.
- **Risicobehandeling:** Overwegen om de beveiligingsmaatregelen te versterken, regelmatige beveiliging audits uit te voeren, trainingsprogramma's voor medewerkers te implementeren en een incidentresponsplan op te stellen voor het geval er toch een datalek plaatsvindt.

- **Resterende impact:** Laag/Middelmatig, na het implementeren van geschikte maatregelen zal er gestreefd worden naar een lagere resterende impact, maar deze kan nooit volledig worden geëlimineerd.
- **Resterende waarschijnlijkheid:** Laag/Middelmatig, na het nemen van de benoemde risicobehandelingen zou het doel zijn om de waarschijnlijkheid van dit risico te verlagen, maar zal er alsnog alert blijft moeten worden op nieuwe bedreigingen en pas de beveiligingsmaatregelen dienovereenkomstig aan.

12.2.4. Connectie over publieke netwerken wordt niet middels encryptie gedaan

- **Risico type:** Vertrouwelijkheid, een gebrek aan encryptie kan leiden tot het onderscheppen van gevoelige informatie tijdens communicatie.
- **Getroffen item:** Data, omdat het risico betrekking heeft op de beveiliging van de gegevens die via de verbinding worden verzonden.
- **Bestaande maatregelen:** De applicatie maakt gebruik van HTTPS.
- **Kwetsbaarheid:** Het risico kan kwetsbaarheden blootleggen in de vertrouwelijkheid van gegevens, omdat onversleutelde communicatie gevoelig is voor afluisteren en gegevenslekken.
- **Impact:** Hoog, vooral als gevoelige informatie, zoals inloggegevens, persoonlijke gegevens of vertrouwelijke bedrijfsinformatie, wordt onderschept en misbruikt.
- **Waarschijnlijkheid:** Middelmatig, de waarschijnlijkheid van dit risico kan variëren en is afhankelijk van de gebruikte netwerken, vooral op openbare wifi-netwerken.
- **Risicobehandeling:** Verplicht medewerkers en klanten om gebruik te maken van een VPN.
- **Resterende impact:** Hoog, na het gebruik maken van een VPN zal de impact alsnog hoog zijn indien de gevoelige informatie toch onderschept wordt.
- **Resterende waarschijnlijkheid:** Laag, na het gebruiken van een VPN zal het waarschijnlijk een stuk lager worden, maar zou alsnog aanwezig kunnen zijn.

12.2.5. Gevoelige informatie wordt niet geheel verwijderd

- **Risico type:** Integriteit, omdat het betrekking heeft op de integriteit van gevoelige informatie die mogelijk niet correct wordt verwijderd.
- **Getroffen item:** Proces, omdat het risico verband houdt met het proces van het verwijderen van gevoelige data.
- **Bestaande maatregelen:** Alle gevoelige informatie wordt verwijderd indien het eenmalig bekeken of gedownload wordt. Anders heeft het ook nog een verloopdatum van maximaal zeven dagen.
- **Kwetsbaarheid:** Het risico kan leiden tot het achterlaten van resterende kopieën van de data, die kwetsbaarheden in het verwijderingsproces kunnen tonen.
- **Impact:** Hoog, omdat het onjuist verwijderen van gevoelige informatie kan leiden tot datalekken, inbreuken op de privacy en wettelijke gevolgen.
- **Waarschijnlijkheid:** Middelmatig, vele configuraties zijn uitbundig getest om te valideren dat alle gegevens zorgvuldig en geheel verwijderd worden.
- **Risicobehandeling:** Een duidelijke procedure opstellen met het verwijderen van gevoelige data, bijvoorbeeld cronjobs die regelmatig controleren of er geen gevoelige informatie nog rondzweeft.
- **Resterende impact:** Hoog, indien er alsnog gevoelige informatie rondzweeft kan dit alsnog leiden tot de eerder genoemde consequenties.
- **Resterende waarschijnlijkheid:** Laag, indien het op een reguliere basis wordt gecontroleerd en uitbundig getest is, is de waarschijnlijkheid hiervoor laag.

13. Beslissing Logboek

Onderstaand worden alle belangrijke keuzes die gemaakt zijn voor de applicatie toegelicht.

13.1. T3 Stack

Voor de technologie van de applicatie is de T3 Stack gekozen. Dit is een erg moderne web development stack waarbij de nadruk wordt gelegd op eenvoudigheid, modulariteit en full-stack type safety. Het maakt gebruik van de libraries:

- Next.JS
- TypeScript
- Tailwind.CSS
- Prisma
- tRPC
- NextAuth.js

De libraries passen goed bij dit project wegens argumenten zoals: met de combinatie van NextJS en tRPC is het erg flexibel voor front- tot back-end ontwikkeling. Daarnaast wordt Google SSO, een vereiste van de opdrachtgever, ondersteund middels NextAuth.

Deze technologieën zijn in de beginnende fase van het project vergeleken met andere technologieën en hieruit werd geconcludeerd dat de T3 Stack de meest geschikte kandidaat was.

Toch was de voornaamste reden om de T3 Stack te kiezen dat het al wel bekend was binnen Webbio. Hierdoor zou er na de overdracht van de applicatie nog goede ondersteuning kunnen worden behouden.

13.2. Docker

Voor de applicatie is ervoor gekozen om gebruik te maken van Docker, omdat het ervoor zorgt dat de applicatie verpakt kan worden in de vorm van containers.

Hierdoor kan zowel de client als de server van de applicatie eenvoudig onderhouden en uitgevoerd worden. Ook vereenvoudigt dit de sprong van lokale ontwikkeling tot deployment van de applicatie aangezien dezelfde Dockerfile configuratie uitgelezen kan worden.

13.3. AWS

AWS is niet alleen gebruikt omdat het een vereiste was van dit project, maar ook omdat AWS zoveel flexibiliteit biedt met de hoeveelheid diensten. Tijdens dit project zijn er vele verschillende diensten van AWS gebruikt. Hierbij is het ideaal om alles op een plek te hebben.

13.4. Elastic Beanstalk

Een service binnen AWS waar een specifieke afweging voor is genomen om te gebruiken was Elastic Beanstalk. AWS heeft namelijk diverse diensten rondom het deployen van een applicatie, maar Elastic Beanstalk is gekozen omdat het grotendeels van de configuratie al verzorgde. Daarnaast regelde Elastic Beanstalk in principe ook de verbinding met de EC2 instantie. Omdat alles eenvoudig te configureren is, maakt dit de applicatie ook erg makkelijk overdraagbaar. Ten slotte is Elastic Beanstalk erg gebruiksvriendelijk voor ontwikkelaars die nog weinig of geen ervaring hebben rondom deployment binnen AWS.

14. Literatuurlijst

- Brown, S. (2006). The C4 model for visualising software architecture.
<https://c4model.com/>
- NPM: ZoD. (z.d.). npm. <https://www.npmjs.com/package/zod>
- About PageSpeed Insights. (z.d.). Google for Developers.
<https://developers.google.com/speed/docs/insights/v5/about>
- Moving out of the Amazon SES sandbox - Amazon Simple email service.
(z.d.).
<https://docs.aws.amazon.com/ses/latest/dg/request-production-access.html>
- CertBot 2.7.0.Dev0 documentation. (z.d.).
<https://eff-certbot.readthedocs.io/en/latest/intro.html>