Bacheloroppgave Dataingeniør

C - SYSTEMDOKUMENTASJON

Gruppe 062

Imran, Zaim Schau, Max Torre

Vår 2021

Revisjonshistorikk

Dato	**Versjon**	**Beskrivelse**	**Forfatter**
11/01/21	0.1	Oppsett av kapitler	Max T. Schau og Zaim Imran
26/04/21	1.0	Første utkast	Max T. Schau og Zaim Imran
03/05/21	1.1	Skrevet ferdig kapittel 10	Max T. Schau
15/05/21	1.2	Lagt til forside	Max T. Schau
18/05/21	2.0	Gjort klar for innlevering	Max T. Schau og Zaim Imran

Tabell 1: Revisjonshistorikk

Innhold

1 Introduksjon

Dette dokumentet er skrevet i forbindelse med bacheloroppgaven for Dataingeniør. Dokumentet har til hensikt å gi en overordnet beskrivelse av systemet som er utviklet. Leseren skal sitte igjen med en oversikt over hvordan systemet fungerer som en helhet.

Dokumentet har tatt utgangspunkt i malen "Systemdokumentasjon" av Ole Christian Eidheim med innspill fra Nils Tesdal gitt av NTNU.

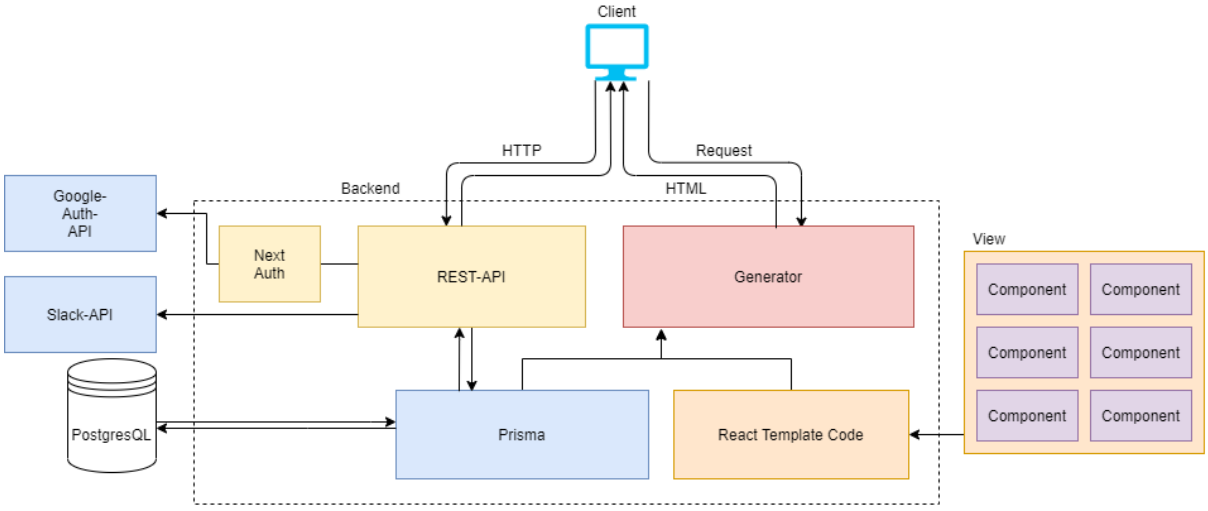
1.1 Valg

I systemdokumentasjonen er det foretatt enkelte valg fra teamet sin side som gjør at kapitlene avviker fra malen gitt fra NTNU.

Klassediagram er ikke tatt med da teamet ikke anså dette som relevant. Vi har ikke benyttet oss av et objektorientert språk slik at det heller ikke var relevant å inkludere et klassediagram. Man kan derimot få en helhetlig oversikt over de ulike entitetene og deres attributter, samt relasjonen, ved å lese kapittel 5 i dette dokumentet eller lese Vedlegg B Kravdokumentasjon kapittel 3 Domenemodell.

Vi har derimot valgt å inkludere et nytt kapittel "Sekvensdiagram" som viser aktiviteter som er kritiske for systemet.

2 Arkitektur



Systemarkitekturen i dette prosjektet består hovedsakelig av en delt tjener og to tredjeparts-API. Systemet bruker *Next.js* med hjelp av *React*, *Webpack* og *TypeScript*. Selve web-applikasjonen er delt opp i flere *Views*. Disse består av komponenter fra *Material-UI* og egenutviklede komponenter. Sammen utgjør de en side i systemet. Generatoren vil generere HTML-koden på tjeneren før det blir sendt til klienten. Dette gjør den ved å ta utgangspunkt i React-sidene, og populerer disse med data ved hjelp av rammeverket Prisma. Prisma kobler seg opp til PostgreSQL-databasen for å hente ut dataen.

Next.js brukes også som et REST-API for å manipulere data i en ekstern database. I likhet med generatoren vil tjeneren bruke Prisma for å generere spørringer for å hente ut data fra *PostgreSQL*-databasen. I tillegg benyttes det *NextAuth* sammen med *Google OAuth 2.0*-APIet for å autentisere og autorisere brukere av systemet. Dette skjer ved hjelp av ressurser i databasen som blir sammenlignet med informasjon om brukeren fra Google. Tilslutt brukes det *Slack-API* til å sende ut notifikasjoner til brukeren på Slack.

For å lagre dataen blir det benyttet en ekstern PostgreSQL-database.

3 Prosjektstruktur

```
src/
├── components/
│   ├── form/
│   └── views/
│       ├── ansatt/
│       ├── mine-ansatte/
│       ├── mine-oppgaver/
│       └── prosessmal/
├── fixtures/
├── lib/
├── node_modules/
│   └── .prisma/
│       └── client/
│           └── schema.prisma
├── pages/
│   ├── alle-ansatte/
│   ├── ansatt/
│   ├── api/
│   │   ├── auth/
│   │   ├── cron/
│   │   ├── employee/
│   │   │   └── [id]/
│   │   ├── employeeTasks/
│   │   ├── notification/
│   │   ├── phases/
│   │   └── tasks/
│   ├── mine-ansatte/
│   └── prosessmal/
├── prisma/
│   └── schema.prisma
└── utils/
```

Teamet har brukt konvensjonell prosjektstruktur for Next.js. Årsaken til at både norsk og engelsk benyttes er at alle sider under mappen “pages” vil fungere som en URI på systemet. For eksempel vil “alle-ansatte” peke til “/alle-ansatte” ettersom den er en undermappe i “pages”. I tillegg har alle komponenter tilknyttet sidene blitt lagt i mappen med “components” med tilsvarende navn for å holde det konsistent.

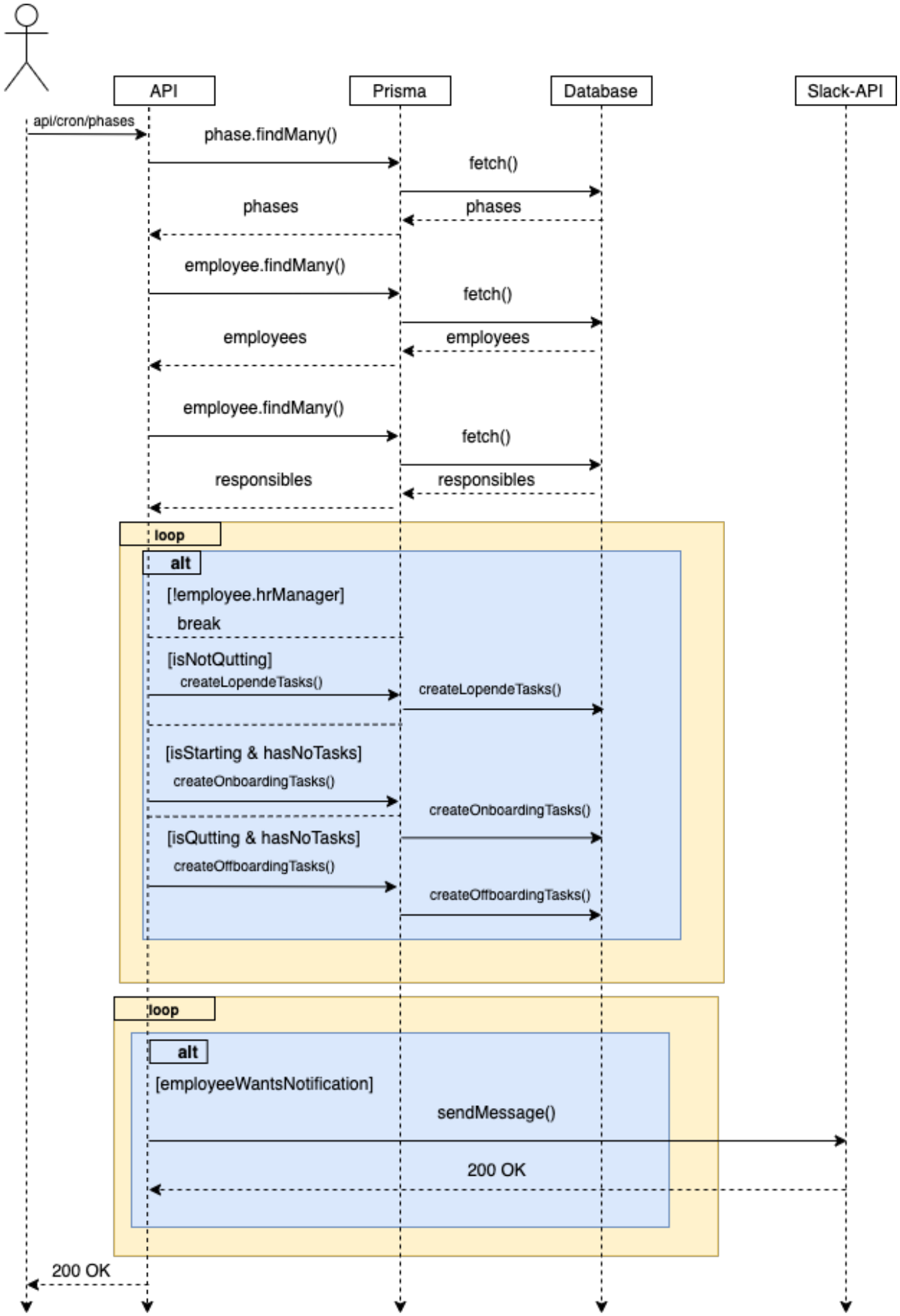
Ellers inneholder “fixtures” et sett med .yml-filer som benyttes for å laste inn data til databasen for å enkelt kunne teste applikasjonen.

“Lib” inneholder filer knyttet til Prisma og autentisering.

Videre inneholder “utils” diverse filer som benyttes av flere av prosjekts komponenter.

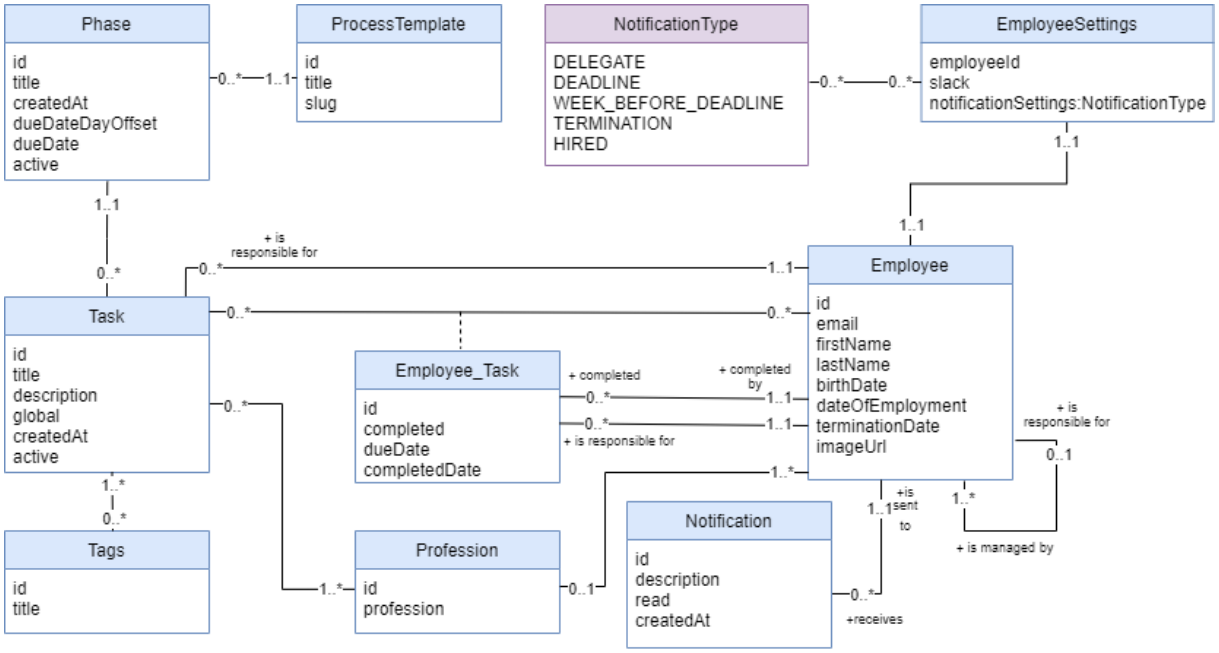
4 Sekvensdiagram

4.1 Automatisk opprettelse av oppgaver



Sekvensdiagrammet viser overordnet hvordan oppgaver automatisk blir opprettet. Det er en planlegger som kaller på endepunktet. Her vil det bli gjort en del sjekker for å verifisere at det skal opprettes oppgaver. Metodenavnene i diagrammet samsvarer ikke med hva de faktisk heter da dette kun er en abstrahert versjon for å vise frem det på et overordnet plan.

5 Databasemodell



Til å begynne med er relasjonen mellom prosessmal, fase og oppgave relativ selvforklarende der prosessmalene kan ha null til mange faser, som igjen kan ha null til mange oppgaver.

Enhver oppgave vil ha null til mange stikkord som er med på å forklare hva oppgaven går ut på. I tillegg vil enhver oppgave enten ha en eller flere yrker knyttet til seg. Årsaken til dette er at noen oppgaver vil gjelde for alle yrker, mens andre oppgaver vil eksempelvis kun gjelde designere, og ikke teknologer.

En ansatt fungerer som bruker-entiteten i databasen. Entiteten har en rekursiv relasjon til seg selv ettersom en ansatt kan ha ansvaret for en annen ansatt. En ansatt kan også være ansvarlig for enhver oppgave som blir opprettet og har derfor en relasjon til denne.

Systemet tar også i bruk varslinger som blir knyttet til hver ansatt. Hver notifikasjonstype ligger lagret som en enum[1] da disse fungerer som konstante streng-variabler. I tillegg er det en relasjon til yrke for å forklare hva slags rolle brukeren har i bedriften. En oppgave fra malen kan være knyttet til flere ansatte, mens en ansatt kan ha flere oppgaver knyttet til seg. Ettersom vi får en mange-til-mange-relasjon er det nødvendig med en koplingstabell.

Databasemodellen blir relativt kompleks med tanke på relasjonene mellom ansatt og koplingstabellen mellom ansatt og oppgave. Relasjonene til ansatt beskriver hvem som eventuelt har fullført oppgaven og hvem som er ansvarlig for en individuell oppgave. Dette blir beskrevet med gode attributtnavn i databasen for å unngå forvirring.

[1]Datatype bestående av navngitte, konstante variabler

6 Server-tjenester

6.1 Endepunkter

Metode	Endepunkt	Beskrivelse	Rettigheter
POST	/auth/	OAuth-endepunkt for å håndtere innlogging ved hjelp av Google	Ansatt
POST	/cron/phases	Endepunkt som en planlegger kaller på for å gjøre daglige oppgaver tilknyttet systemet	Planlegger
GET	/employees/	Henter ut alle ansatte	Innlogget ansatt
GET	/employees/:id/	Henter ut ansatt basert på id	Innlogget ansatt
GET	/employees/:id/ notifications	Henter ut notifikasjonene til en ansatt	Innlogget ansatt
PUT	/employees/:id/ settings	Endring av innstillingene til en ansatt	Innlogget ansatt
POST	/employeeTasks/	Oppretter en ansattoppgave	Innlogget ansatt
PATCH	/employeeTasks/	Ved hjelp av et action kan man gjøre endringer på flere employeeTasks i samme forespørsel	Innlogget ansatt
GET	/employeeTasks/ :id/	Henter ut en ansattoppgave	Innlogget ansatt
PUT	/employeeTasks/ :id/	Gjør endring på en ansattoppgave	Innlogget ansatt
POST	/notifications	Oppretter en varsling	Innlogget ansatt
PUT	/notifications/ :id/	Gjør endring på en varsling	Innlogget ansatt
POST	/phases/	Oppretter en fase	Innlogget ansatt
GET	/phases/:id/	Henter ut en fase	Innlogget ansatt
PUT	/phases/:id/	Gjør endring på en fase	Innlogget ansatt
DELETE	/phases/:id/	Sletter en fase	Innlogget ansatt
GET	/processTemplates/	Henter ut alle prosessmaler	Innlogget ansatt
GET	/professions/	Henter ut alle yrker	Innlogget ansatt
GET	/tags/	Henter ut alle tagger	Innlogget ansatt
POST	/tasks/	Oppretter en oppgave	Innlogget ansatt
GET	/tasks/:id/	Henter en oppgave	Innlogget ansatt
PUT	/tasks/:id/	Gjør endring på en oppgave	Innlogget ansatt
DELETE	/tasks/:id/	Sletter en oppgave	Innlogget ansatt

6.2 Rettigheter

Det er i hovedsak kun to rettigheter for endepunktene i systemet. Man må altså være autorisert for å få tilgang til systemet. Endepunktet man benytter for å logge inn (auth) vil kun fungere for ansatte som ligger lagret i databasen. Dermed er det kun brukere som per definisjon er ansatt hos Blank som har tilgang.

Ellers er rettigheten *innlogget ansatt* brukere med et gyldig token som blir sendt med i forespørselen.

Endepunktet med rettigheten planlegger er ment for en cron-jobb som kjører periodisk. Her er

det en delt, hemmelig nøkkel mellom slik at det er kun planleggeren som kan benytte seg av endepunktet.

7 Sikkerhet

For å ivareta sikkerheten har teamet tatt utgangspunkt i OWASP Top Ten fra 2017[1]. OWASP Top Ten er en liste over de ti mest kritiske sikkerhetsrisikoene i applikasjoner. Den blir regelmessig oppdatert for å sørge for at utviklere til enhver tid kan være oppdatert på de risikoene som kan være.

For å gå gjennom hvordan sikkerheten er ivaretatt vil det bli gått gjennom de aktuelle risikoene fra OWASP, og hvordan dette er løst.

1. Injection

For å forhindre injections har teamet benyttet seg av et ORM-rammeverk som sender data som parametere istedenfor å kjøre rå SQL-setninger. På den måten vil det ikke være mulig å kjøre SQL-injections på systemet.

2. Broken Authentication

Det er blitt tatt i bruk et eksternt bibliotek (Next-Auth) for å håndtere autentiseringen på nettsiden. Dermed er det biblioteket som håndtere all form for autentisering og tokens. Det vil derfor ikke være mulig å aksessere systemet uten å være autentisert. Ved bruk av Google OAuth sørger man for at e-posten man logger inn med må være fra Blank-domenet slik at ingen andre enn de ansatte kan få tilgang.

Det er blitt tatt i bruk JWT for å videre autentisere den innloggede brukeren.

3. Sensitive Data Exposure

Det blir i utgangspunktet ikke lagret sensitiv data i databasen. Systemet skal derimot koble seg til Blanks internsystemer der det kan ligge sensitiv data. Derfor vil systemet ikke hente ut mer data enn nødvendige, og ingen andre enn verifiserte brukere har tilgang til systemet.

5. Broken Access Control

Ettersom ingen andre enn ansatte fra Blank kan logge inn i systemet er det ikke satt opp begrensninger for en innlogget bruker. Dermed har en innlogget bruker tilgang til systemet i sin helhet.

7. Cross-Site Scripting (XSS)

Rammeverket React er utviklet slik at det løser dette problemet som en standard. Ettersom Next.js er React-rammeverk vil dette problem være løst. React vil nemlig "escape" alle tags som kommer inn i systemet. Det er fortsatt mulighet for en bruker å linke til en "ond" side da det ikke er lagt til begrensninger på hva slags linker som kan bli lagt til. Likevel er systemet ment som et internsystem for pålitelige ansatte og alle linker blir vist i sin helhet slik at brukere kan se at linken peker til en side som ikke ser trygg ut.

9. Using Components with Known Vulnerabilities

For å forhindre at dette skjer har det blitt lagt fokus på å gjøre grundig forarbeids før man benytter seg av eksterne biblioteker og komponenter i prosjektet. På den måten har man sørget for at ethvert importert bibliotek skal være sikkert, og ikke utsette systemet for noen risikoer.

8 Installasjon og kjøring

8.1 Avhengigheter

Avhengighet	Versjon
Git	2.3
Node	15.6 eller høyere
Yarn	1.22.10 eller høyere
PostgreSQL	13.2

Installasjonsmanualen for Git for Linux, Windows og MacOS kan finnes her:

`https://git-scm.com/book/en/v2/Getting-Started-Installing-Git`

For å verifisere at Git er installert kan man skrive følgende inn i terminalen:

```
git --version
```

Da er det forventet å få skrevet ut tilsvarende:

```
git version 2.30.1
```

Node kan lastes ned og installeres fra følgende link:

`https://nodejs.org/en/download/`

For å verifisere at Node er installert kan man skrive følgende inn i terminalen:

```
node --version
```

Da er det forventet å få følgende:

```
v15.6.0
```

Installasjonsmanual for Yarn finnes på følgende link:

`https://classic.yarnpkg.com/en/docs/install`

For å verifisere at Yarn er installert kan man skrive inn følgende inn i terminalen:

```
yarn --version
```

Da er følgende forventet å bli skrevet ut:

```
1.22.5
```

En PostgreSQL-database er også nødvendig for å kunne lagre all dataen. Denne kan kjøres lokalt på datamaskinen med for eksempel Docker. Dersom Docker er installert kan man enkelt opprette databasen ved å kjøre følgende kommando fra rot-mappen:

```
yarn createdb
```

8.2 Miljøvariabler

Videre må man også legge inn diverse miljøvariabler for at systemet skal fungere som normalt. Under kommer en liste over de ulike og en kort forklaring på hver enkelt variabel.

DATABASE_URL:

URL benyttet for å få tilgang til databasen.

CRON_SECRET:

En tilfeldig generert streng som benyttes av planleggeren for å sikre at ingen andre kan gjøre kall til endepunktet som utfører de daglige oppgavene.

JWT_SECRET:

En tilfeldig generert nøkkel-streng som brukes for å generere og validere JWT-tokens.

SLACK_TOKEN:

Dette blir generert av Slack for å kunne aksessere deres API´er, og for å kunne sende meldinger og se oversikt over medlemmer i et workspace [2].

GOOGLE_ID:

For å kunne benytte seg av innlogging med Google ved hjelp av OAuth 2.0 er det nødvendig å aksessere Google sitt API. Her må man lage en applikasjon, og hente ut ulike legitimasjoner fra Google API Console.

GOOGLE_SECRET:

Må i likhet med GOOGLE_ID hentes ut fra Google API Console for å bruke innlogging med Google.

8.3 Oppsett

Først må prosjektet klones fra GitHub:

```
git clone https://github.com/Zenjjim/Trak.git
```

Deretter kan man installere alle avhengigheter ved å skrive følgende:

```
cd Trak
yarn
```

8.4 Oppsett av database

For å laste inn de ulike entitetene inn i databasen gjøres følgende:

```
npx prisma generate
yarn push
```

8.5 Innlastning av data

Dette er kun testdata som kan benyttes for å se hvordan systemet vil fungere i praksis.

For å laste inn dataen kjøres følgende:

```
yarn loaddata
```

[2]Et område bestående av kanaler hvor medlemmer kan sende meldinger til hverandre

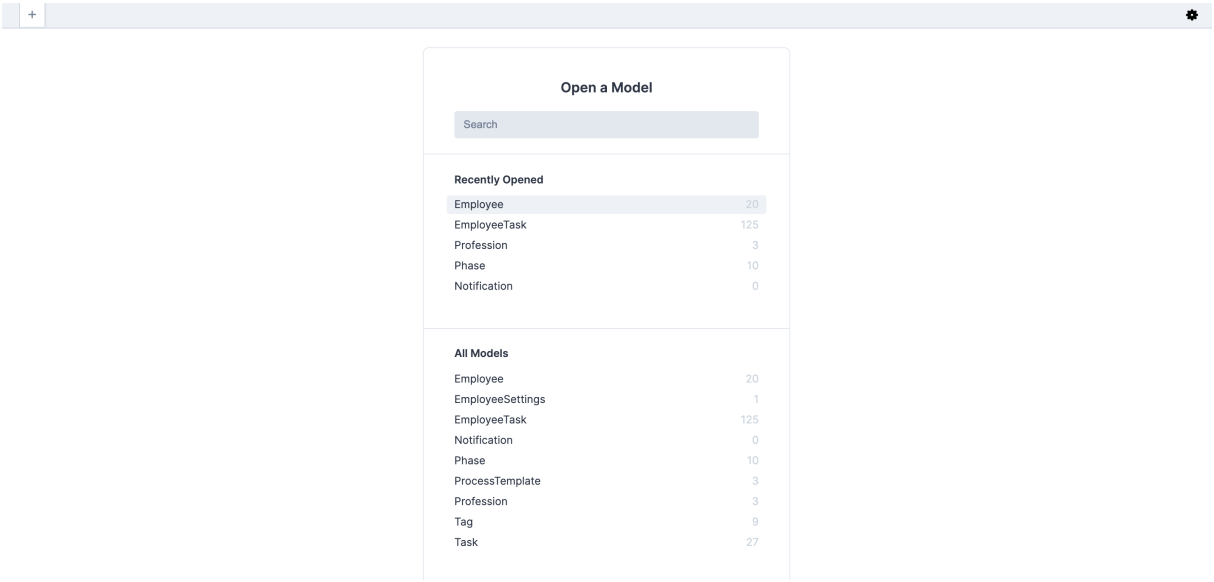
8.6 Opprette egen bruker

For å kunne logge inn må man ha en bruker inne i databasen med Google e-posten. Dette gjøres ved første å kjøre:

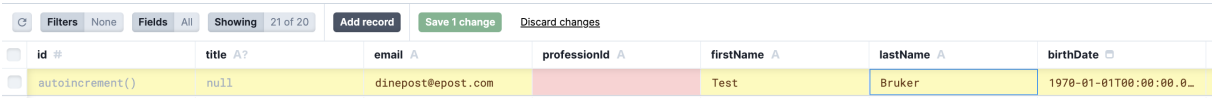
```
npx prisma studio
```

Da skal nettleseren åpne *localhost:5555* for deg, men om dette ikke skjer må du manuelt skrive inn adressen i nettleseren din. Dette åpner et panel til databasen din.

Øverst til venstre er det et pluss tegn. Trykk på den og velg *Employee*.



Dette skal åpne Employee-tabellen fra databasen. Her skal du legge til en bruker ved å trykke på *Add Record* og fylle inn feltenen *email*, *firstName*, *lastName* og *profession*.



Når alle feltene er fylt kan man lagre endringen ved å trykke på **Save 1 change**. Da skal man kunne logge inn med denne brukeren.

8.7 Starte applikasjonen

Til slutt gjenstår det kun å starte applikasjonen:

```
yarn dev
```

9 Dokumentasjon av kildekode

Det har blitt lagt fokus på god dokumentasjon av kode for å sikre at det skal være enkelt for Blank å potensielt videreutvikle systemet. Derfor har komponenter og endepunkter blitt dokumentert ved hjelp av TSDoc.

I tillegg har teamet hatt fokus på å såkalt clean-code. Dette er prinsipper som stammer fra boken "Clean Code" av Robert C. Martin. Teamet har på denne måten strebet etter at koden i seg selv skal fungere som en dokumentasjon. På den måten skal man så langt det lar seg gjøre få en god forståelse for hva koden gjør ved å lese variabelnavn, funksjonsnavn osv.

Dokumentasjonen kan finnes her:

`https://zenjjim.github.io/Trak/docs/index.html`

10 Kontinuerlig integrasjon og testing

10.1 Kontinuerlig integrasjon

Teamet har implementert kontinuerlig integrasjon ved hjelp av GitHub Actions. Dette er implementert slik at for hver branch i prosjektet, vil systemet bli bygget og testene bli kjørt. Dermed sikrer man at koden er av en minimumskvalitet da det ikke er mulig å slå sammen en branch inn til hovedbranchen dersom byggingen eller testene feiler.

Måten dette skjer på er definert i de ulike .yml-filene som ligger i *.github/workflows*-mappen. Her ligger det totalt tre filer som vil bli kjørt hver for seg.

I tillegg har det også blitt implementert kontinuerlig utrulling ved enhver push til dev-branchen. Så lenge byggingen og testene kjører uten feil vil systemet bli rullet ut på en Heroku-tjener.

10.2 Testing

10.2.1 Hvordan er det blitt testet?

Teamet har sammen med oppdragsgiver blitt enige om å teste vesentlige endepunkter på tjeneren, og dermed nedprioritere klient-tester. Testene blir kjørt opp mot en dedikert test-database som blir populert med data ved hjelp av designmønsteret fabrikk.

Det er blitt lagt fokus på å teste at vi får de responsene vi forventer, og at systemet også reagerer korrekt når brukeren forsøker å gjøre ting hen ikke skal. Det vil si at det for eksempel ikke er mulig å slette noe som ikke eksisterer.

All files

80.45% Statements 325/404 **45.42%** Branches 129/284 **79.1%** Functions 53/67 **78.65%** Lines 269/342

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
__tests__/factories		100%	76/76	100%	13/13	100%	8/8	100%	52/52
__tests__/utils		100%	3/3	100%	0/0	100%	1/1	100%	1/1
src/lib		64.29%	18/28	18.75%	3/16	100%	3/3	66.67%	16/24
src/pages/api		87.76%	43/49	62.5%	10/16	71.43%	5/7	86.36%	38/44
src/pages/api/cron		58.18%	64/110	25.22%	29/115	54.55%	12/22	58.42%	59/101
src/pages/api/employeeTasks		93.75%	30/32	76.67%	23/30	100%	5/5	92.59%	25/27
src/pages/api/phases		82.93%	34/41	39.47%	15/38	100%	7/7	80%	28/35
src/pages/api/tasks		81.82%	36/44	56.52%	26/46	77.78%	7/9	78.38%	29/37
src/utils		100%	21/21	100%	10/10	100%	5/5	100%	21/21

For å se testrapporten i mer detalj kan man benytte seg av følgende link:

`https://zenjjim.github.io/Trak/coverage/lcov-report/index.html`

10.2.2 Hvordan kjøre testene?

For å kjøre testene lokalt er det nødvendig å ha tilgang til en ekstern PostgreSQL-database. Teamet anbefaler å benytte seg av Docker for å ha en dedikert test-database.

Dersom Docker er installert har teamet laget et script som oppretter et nytt Docker-image, laster inn tabellene og kjører testene. Dette kan kjøres ved å skrive følgende inn i terminalen:

```
yarn test
```

11 Referanser

Referanser

[1] *OWASP Top Ten.* URL: https://owasp.org/www-project-top-ten/.