NTNU

Kunnskap for en bedre verden

Bacheloroppgave dataingeniør

Systemdokumentasjon

Mikkel Ofrim, Oskar Remvang

Vår 2023

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
03.05.2023	0.1	Oppsett av mal	Mikkel Ofrim, Oskar Remvang
10.05.2023	1.0	Førsteutkast	Mikkel Ofrim, Oskar Remvang
18.05.2023	1.2	Revidering før innlevering	Mikkel Ofrim, Oskar Remvang

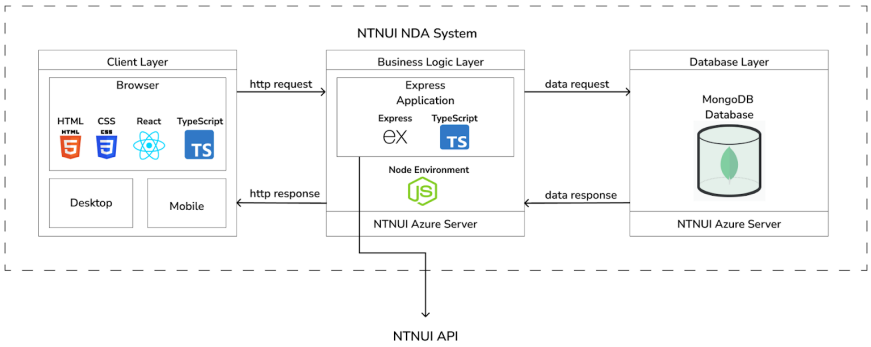
Innhold

1 Introduksjon

Dokumentet er skrevet i sammenheng med bachelorgruppe 68s bacheloroppgave, NTNUI NDA. Systemdokumentasjonen er ment å gi en helhetlig oversikt over systemets viktigste komponenter og virkemåter. Det inneholder derimot også noen mer detaljerte beskrivelser, blant annet angående databasemodell og server-endepunkt. Innholdet i dokumentet følger systemdokumentasjons-malen distribuert av NTNU, laget av Ole Christian Eidheim med innspill fra Nils Tesdal.

Denne systemdokumentasjonen inneholder ikke kapittelet om klassediagram som er inkludert i malen. Ettersom systemet ikke benytter seg av objektorienterte språk, bedømte gruppen det lite hensiktsmessig å benytte seg av klassediagram for dokumentasjon.

2 Arkitektur



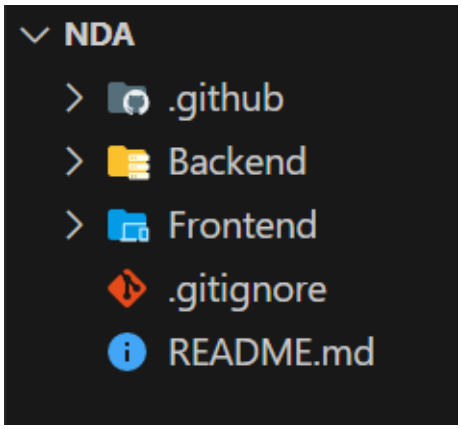
Figur 1: Arkitekturmodellen

Systemet er splittet inn i tre lag. Klientlaget utgjør Frontend-koden til prosjektet. Dette inkluderer alt av HTML, CSS og JavaScript som skal kjøres lokalt i klientens nettleser. Klient-koden er skrevet med React og TypeScript, og kommuniserer med businesslogikklaget gjennom HTTP forespørsler.

Businesslogikklaget er Backend-koden til systemet. Denne består av en Express applikasjon, som utgjør NDA APIet, skrevet i TypeScript. APIet kjører i et Node miljø som kjøres på en av NTNUI sine Azure servere. Express APIet kommuniserer med databaselaget som består av en MongoDB database. Denne inneholder all informasjon relatert til NDA systemet, og kjøres også på en av NTNUI sine Azure servere.

NDA APIet kommuniserer også med NTNUI APIet for brukerautentisering og henting av data.

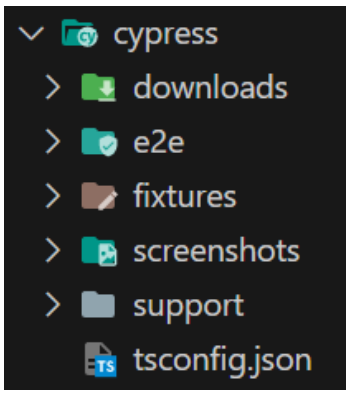
3 Prosjektstruktur



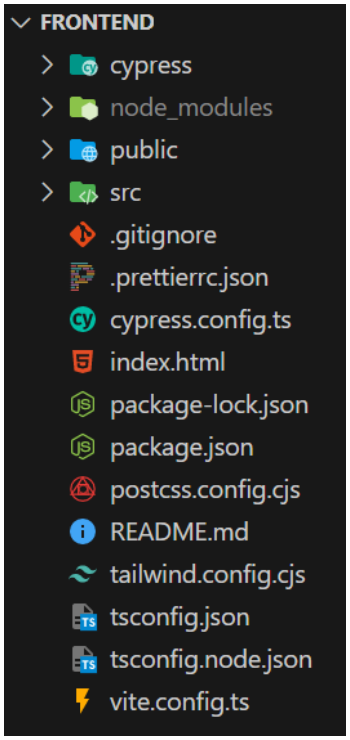
Figur 2: Overordnet filstruktur

Den overordnede filstrukturen i prosjektet splitter tjenerkode inn i Backend mappen og klientkode inn i Frontend mappen. Her defineres også CI workflows i .github mappen. Kun essensielle filer/-mapper vil bli nevnt og filer/mapper som er inkludert i både Backend og Frontend vil kun bli forklart en gang.

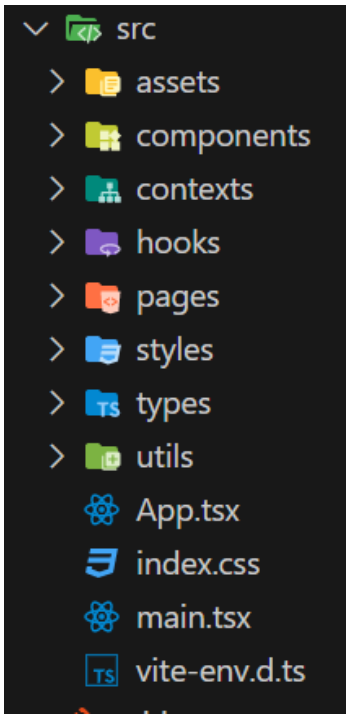
3.1 Klient



(a) Filstruktur i cypress testing-mappen



(b) Filstruktur i frontend (klient) mappen

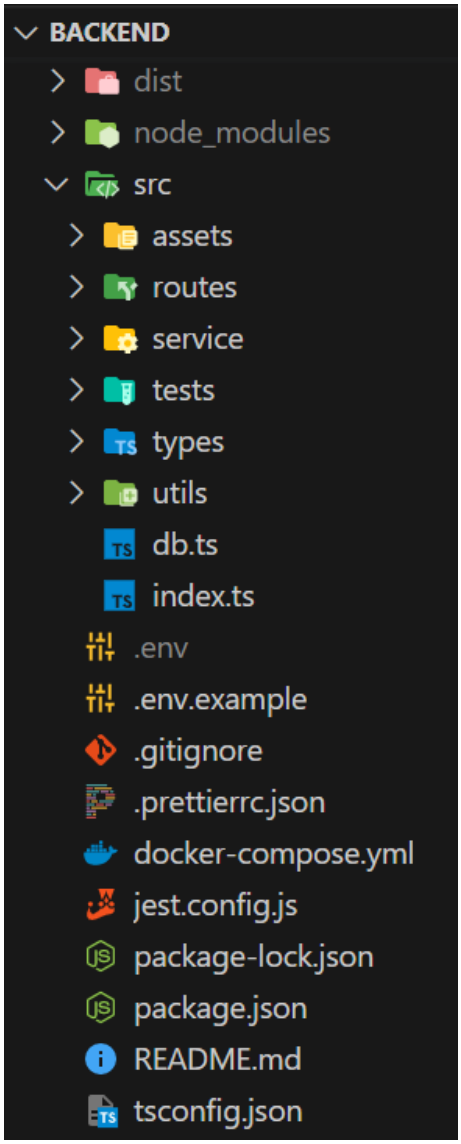


(c) Filstruktur i src mappen i klienten

Figur 3: Filstruktur i klienten

Frontend-mappen følger en konvensjonell React-filstruktur generert med Vite. Src mappen inneholder alt av eksekverbar JavaScript kode, samt HTML og CSS. Assets inkluderer statiske filer som bilder. Pages definerer de ulike sidene som vises på nettsiden, og components inneholder komponentene som utgjør disse sidene. Utils inneholder generelle funksjoner som ikke kan relateres til en spesifikk side eller komponent. Cypress mappen inneholder alt relatert til testing. Testene ligger i e2e mappen og fixtures definerer mock-responsdata. Utenom src og cypress består Frontend av diverse konfigureringsfiler, samt package.json filene som blant annet definerer eksterne Node-biblioteker.

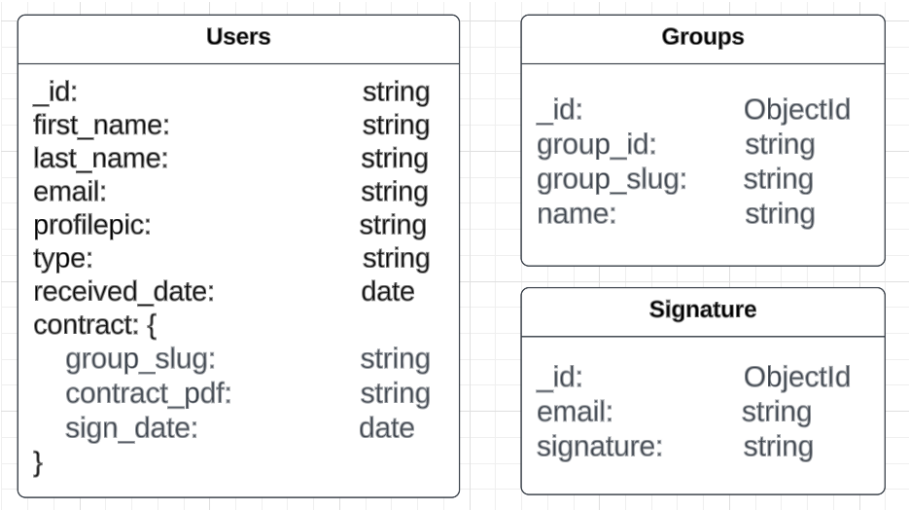
3.2 Tjener



Figur 4: Filstruktur i backend (tjener) mappen

Backend har alt av eksekverbar kode i src mappen. Src mappen inkluderer statiske filer i assets mappen, API-ruter i routes mappen, alt av logikk i service mappen, og TypeScript-typer i types mappen. .env-filen definerer hemmelige miljøvariabler som eksempelvis oppkoblings-URIen til databasen.

4 Databasemodell



Figur 5: Databasemodellen

"Users" inneholder data om alle brukere som har blitt tilsendt en taushetserklæring. Det er her viktig å nevne at dette ikke er informasjon tilhørende innloggede brukere, men brukere den innloggede skal kunne samhandle med. Her inngår både direkte informasjon tilknyttet bruker og all informasjon tilhørende kontrakten. _id attributten genereres generelt sett automatisk av MongoDB som en 12-byte hexadecimal string med typen ObjectId. User-dokumentene har derimot en 128-bit _id representert som en string, generert ved hjelp av npm pakken "uuid". Contract objektet inneholder attributten "group_slug". Denne brukes til å knytte en kontrakt opp mot en spesifikk gruppe.

"Groups" representerer en NTNUI gruppe eller utvalg. Dette inkluderer alle nåværende grupper og utvalg, altså de som eksisterer i NTNUI Medlemssystemet, samt historiske grupper og utvalg som er slettet fra NTNUI Medlemssystemet.

"Signature" holder en innlogget brukers opplastede signatur. Denne er representert i "signature" attributten som en base64 kodet string. Signaturen knyttes opp mot den innloggede brukeren ved hjelp av email.

5 Server-tjenester

5.1 Roller

NDA APIet brukes av tre ulike roller: Hovedstyret, Gruppeleder, og Styremedlem. APIet bestemmer ikke tilgang direkte ut fra disse rollene. Enkelte endepunkter returnerer derimot ulike resultater basert på rollen til brukeren som kaller det. Den eneste direkte tilgangbegrensningen ligger mellom innloggede og ikke-innloggede brukere. I sammenheng med de tre rollene nevnt tidligere er det Hovedstyret og Gruppeleder som har muligheten til å logge inn, mens Styremedlem ikke kan logge inn.

5.2 Endepunkter

APIet er inndelt i fem hovedruter som tar for seg ulike oppgaver/ressurser, alle begynnende med '/api': '/auth', '/user', '/group', '/pdf' og '/mail'.

Metode	**Endepunkt**	**Beskrivelse**	**Tilgang**
POST	/auth/login	Logger inn bruker med konto fra NTNUI Medlemssystemet	Ikke-innlogget
GET	/auth/logout	Logger ut bruker	Ikke-innlogget
GET	/auth/checktoken	Validerer accessToken og refreshToken for å se om bruker er logget inn	Ikke-innlogget
GET	/user/loggedInUser	Henter brukerinformasjon fra NTNUI Medlemssystemet med bruk av accessToken	Innlogget
GET	/user/users?groupslug=	Henter alle brukere fra NTNUI Medlemssystemet og NTNUI NDA gitt gruppe. Returnerer kun medlemmer fra NTNUI Medlemssystemet for Gruppeleder	Innlogget
POST	/user/doesUserExist	Sjekker om bruker eksisterer i NDA-systemet	Innlogget
PUT	/user/resetContractByEmail	Erstatter den gamle kontrakten med en ny når en ny kontrakt blir generert for personen	Innlogget
PUT	/user/updateSignature	Oppdaterer en brukers opplastede signatur	Innlogget
POST	/user/getSignature	Henter en brukers opplastede signatur	Innlogget
POST	/user/addUser	Legger til en bruker i NDA-systemet	Innlogget
POST	/user/isUserSigned	Sjekker om en bruker har signert en kontrakt	Ikke-innlogget
GET	/user/getContractByUserId?userId=	Henter kontrakt basert på bruker id	Ikke-innlogget
POST	/user/getContractByUserEmail	Henter kontrakt basert på email	Innlogget
POST	/user/getAllSignedContractsFromGroup	Henter alle signerte kontrakter innad en gruppe	Innlogget
PUT	/user/updateContract	Oppdaterer kontrakten som er lagret i databasen til den nye versjonen	Ikke-innlogget

GET	/group/groups	Henter alle grupper fra NTNUI Medlemssystemet og NTNUI NDA. Dersom det eksisterer grupper i Medlemssystemet som ikke eksisterer i NDA blir disse lagt til i NDA-systemet	Innlogget
POST	/mail/sendEmail	Sender bekreftelsesmail med en kopi av den ferdig signerte kontrakten etter ferdig signering av kontrakt	Ikke-innlogget
POST	/pdf/editContract	Setter avsender og mottakers navn, rolle og gruppe, samt avsenders signatur inn i kontraktmal og returnerer ny kontrakt	Ikke-innlogget
POST	/pdf/editContractForReceiver	Setter mottakers signatur inn i mottatt kontrakt og returnerer ny kontrakt	Ikke-innlogget

6 Sikkerhet

For å sikre systemet mot de vanligste risikofaktorene har teamet valgt å ha hovedfokus på OWASP Top Ten 2021 når det gjelder sikkerhet. Dette er en liste med de ti mest kritiske sikkerhetshullene for web-applikasjoner[1]. Listen publiseres av The OWASP Foundation og kommer ut med en hyppighet på rundt 3-4 år slik at punktene holdes relevante. Kun punkter gruppen har forholdt seg til er inkludert her.

A01:2021-Broken Access Control

I NDA-systemet skilles det i hovedsak mellom innloggede og ikke-innloggede brukere. Innlogging skjer gjennom NTNUI Medlemssystemet. Autentisering opp mot Medlemssystemet skjer med en pakke som er utviklet av NTNUI. Den returnerer en accessToken og en refreshToken dersom login var vellykket. Alle sensitive endepunkter i NDA-systemet bruker et authorization middleware, supplert av NTNUI, for å sjekke gyldigheten av accessToken og refreshToken. Det er kun endepunkter som er nødvendige og trygge å ha åpne for ikke-innloggede brukere som ikke går gjennom authorization middlewaret.

A02:2021-Cryptographic Failures

NDA-systemet lagrer ingen sensitiv informasjon, dermed er heller ingen spesielle hensyn tatt innenfor kryptering av overført data. Systemet blir hostet på NTNUI sitt domene som bruker HTTPS, som vil si at all nettverkstrafikk allerede er kryptert.

A03:2021-Injection

Klientkoden til systemet er skrevet i React. React kommer med en mekanisme som heter JSX (JavaScript XML), som brukes til å laste inn dynamiske data i DOMen. JSX saniterer input automatisk slik at injection angrep ikke er mulig. NDA-systemet har konsekvent brukt JSX istedenfor manuell manipulasjon av DOMen med f.eks .append() eller innerHTML.

A06:2021-Vulnerable and Outdated Components

Eksterne komponenter kan komme med ukjente sikkerhetsmessige risikoer. For å prøve å minimere sannsynligheten for sikkerhetshull gjennom eksterne komponenter har gruppen gjort flere ulike tiltak. Først og fremst har fokuset vært på å bruke få eksterne komponenter. Dersom problemet kan utvikles selv, innenfor rimelig tid, har gruppen heller gjort dette. I tillegg har kun veldokumenterte og vedlikeholdte komponenter blitt tatt i bruk. Begge gruppens medlemmer måtte lese seg opp på og godkjenne komponentene før de eventuelt ble brukt.

A07:2021-Identification and Authentication Failures

Som tidligere nevnt, avhenger NDA-systemet av NTNUI Medlemssystemet for login og autorisering gjennom NTNUI APIet. NDA-systemet forholder seg kun til brukers accessToken og refreshToken som returneres i form av en JWT (Json Web Token). Disse lagres i cookies men har en kryptert signatur så serveren kan sjekke om de har blitt endret på eller ikke.

A08:2021-Software and Data Integrity Failures

Systemet er avhengig av en rekke tredjeparts biblioteker i form av npm pakker. Sikkerhetshull i disse kan potensielt utnyttes til å angripe systemet. For å unngå dette har vi vært sparsommelige og forsiktige med hvilke npm pakker vi har innført. Videre forebygging kan gjøres gjennom CI/CD, f.eks ved hjelp av npm audit. Dette overlates til videreutviklerne av systemet.

7 Installasjon og kjøring

7.1 Avhengigheter

Avhengighet	Versjon
Git	2.3
Node	16

Git kan installeres her: https://git-scm.com/downloads

Versjon kan sjekkes med følgende kommando:

```
git --version
```

Node kan installeres her: https://nodejs.org/en/download

Versjon kan sjekkes med følgende kommando:

```
node --version
```

Det er også nødvendig å kjøre en MongoDB database. Dette kan gjøres lokalt på maskinen ved hjelp av Docker, eller gjennom Atlas, en skybasert MongoDB database.

En lokal instans av databasen kan settes opp og kjøres med Docker ved å kjøre følgende kommando i Backend-mappen

```
npm run db
```

Denne databasen kan stopped ved å kjøre følgende kommando, også i Backend-mappen

```
npm run stop-db
```

Hvis du isteden ønsker å kjøre databasen med hjelp av Atlas kan du finne instrukser på hvordan du kan gjøre det her:

https://www.mongodb.com/docs/atlas/getting-started/
https://account.mongodb.com/account/login

For å kunne sende mailer er det nødvendig å kjøre en mail-server. Her finnes det gratis alternativer som fungerer bra for utvikling. Så lenge du finner noe som gir deg tilgang til alle feltene som brukes under så vil det fungere.

7.2 Miljøvariabler

For at systemet skal fungere må man legge til en .env fil i Backend mappen til prosjektet. Denne inneholder ulike miljøvariabler listet her:

SMTP_HOST: URI til mail-tjener
SMTP_PORT: Porten som SMTP kjører på, generelt 587
SMTP_USERNAME: Brukernavn til autentisering i mail-tjener
SMTP_PASSWORD: Passord til autentisering i mail-tjener
SMTP_SENDER: Mailadressen som vil stå som avsender
NTNUI_TOOLS_API_URL: "https://dev.api.ntnui.no/"hvis prosjektet skal kjøres i utviklingsmodus, la stå tom hvis produktet skal kjøres i production
DB_URI: MongoDB oppkoblingstring

MONGO_INITDB_ROOT_USERNAME, MONGO_INITDB_ROOT_PASSWORD: Dette er login informasjon til rotbruker for kjøring av database i Docker

Du kan finne mer detaljerte instrukser og informasjon om hvordan du kan sette opp en fungerende .env-fil i README.md-filen i backendmappen til systemet.

7.3 Oppsett

Prosjektet kan klones fra GitHub ved å skrive:

```
git clone https://github.com/NTNUI/nda.git
```

Avhengigheter må installeres både i frontend og i backend. Dette kan gjøres ved å navigere til og skrive følgende kommando mens man er i den respektive mappen:

```
npm i
```

7.4 Kjøring av applikasjonen

Express APIet kan startes ved å skrive følgende:

```
cd Backend
npm run serve
```

Klient koden kan startes ved å skrive:

```
cd Frontend
npm run dev
```

Både front- og backend må kjøre samtidig for at systemet skal fungere.

7.5 Innlogging

For å logge inn i systemet er man avhengig av å ha informasjonen til en bruker i NTNUI dev miljøet. Her kreves telefonnummer og passord til en bruker med medlemskap i Hovedstyret eller lederrolle i en gruppe. Dev-miljøet er stadig i endring, så det er ikke mulig å gi login-informasjon med garanti om at den eksisterer i fremtiden.

8 Dokumentasjon av kildekode

Koden er grundig dokumentert med baktanke om å gjøre det lettest mulig å videreutvikle og opprettholde systemet. Gruppens visjon var å gjøre dokumentasjonen så minimal og intuitiv som mulig, samtidig som den måtte gi grundig nok dekning til å gi en god forståelse av systemet. Det ble dermed brukt mye ressurser på å skrive selvdokumenterende kode. Kommentarer er vedlagt i kildekoden der det var absolutt nødvendig, men utenom dette er koden skrevet på en modulær måte som gjør den lett å lese. Ved hjelp av Prettier følger også all kode i prosjektet en standard-formatering som forhåpentligvis vil gjøre det enda lettere å tolke koden.

9 Kontinuerlig integrasjon og testing

9.1 Kontinuerlig integrasjon

Kontinuerlig integrasjon gjøres gjennom GitHub ved hjelp av GitHub Actions. På hver push til master og dev branchene i prosjektet bygges og testes systemet automatisk. Dette gir økt selvsikkerhet om at koden som går ut på de sentrale branchene fungerer som den skal.

CI prosedyren er definert i en ci.yaml fil i .github/workflows mappen. I denne filen defineres ulike jobber som må utføres, for eksempel bygging og testing av koden. GitHub Actions tilbyr forhåndslagde jobber som simplifiserer denne prosessen, for eksempel bygging av Node kode.

9.2 Testing

Testing gjøres ved hjelp av to ulike rammeverk. For Backend bruker vi Jest til å skrive Unit-tester. For Frontend brukes Cypress til å skrive End-to-end tester. Hovedfokuset har ligget på testing av Frontend med End-to-end tester. Dette kommer av at End-to-end tester gjennom cypress både kan brukes til å teste klient kode og tjener kode. Cypress krever at systemet kjøres lokalt før testene kan kjøres og tester derfor at endepunktene faktisk fungerer. Endepunkter kan også enkelt mockes for hver test som gjør det enkelt å velge fritt om man ønsker å teste tjeneren eller ikke.

9.3 Kjøring av tester

Backend tester kan kjøres ved å gå inn i Backend mappen og skrive:

```
npm run test
```

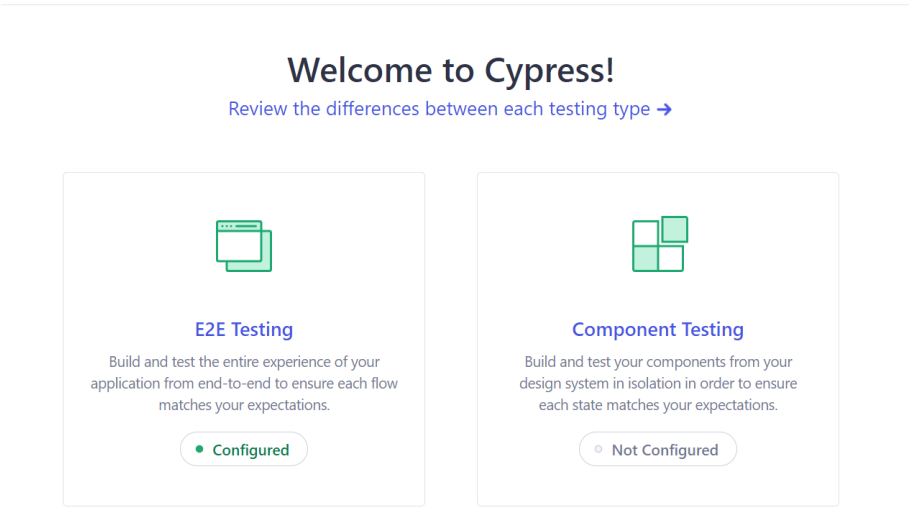
Før man kan kjøre Frontend tester må man starte applikasjonen lokalt slik vist i kapittel 7.5. Frontend tester kan kjøres på to ulike måter. Den første måten kjører testene uten oppstart av Cypress GUI:

```
npm run cypress:run
```

Den andre måten åpner Cypress GUIen som viser nettsiden for hvert steg i hver av testene:

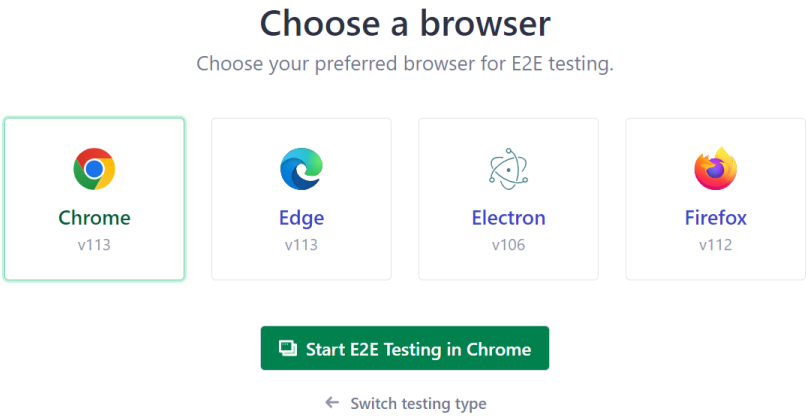
```
npm run cypress:open
```

Her må man først velge E2E Testing ettersom det er End to End tester vi har konfigurert Cypress til å kjøre.



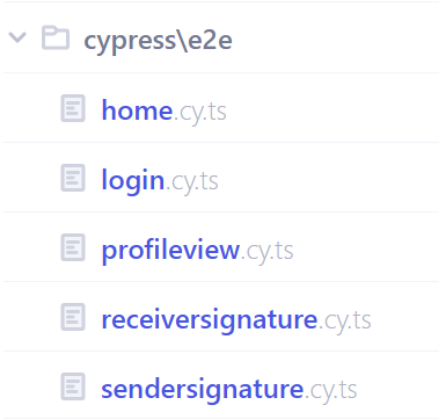
Figur 6: Valget mellom E2E og component testing

Deretter får man mulighet til å velge nettleser å teste i. Valget her er vilkårlig ettersom prosjektet kan kjøres i alle de gitte nettleserene.

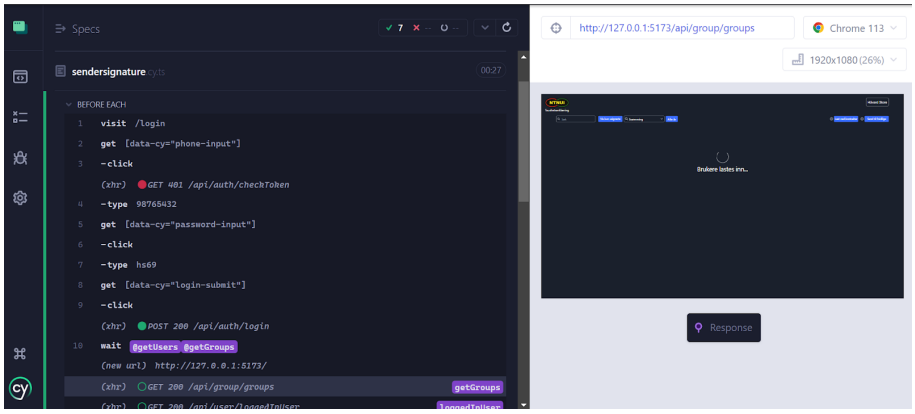


Figur 7: Valget mellom forskjellige nettlesere

Til slutt får du muligheten til å velge hvilken "Spec" du skal kjøre. En Spec er kun en gruppe med tester. Vi har hovedsakelig inndelt testene etter hvilken side de tilhører. Specen kjøres ved å dobbeltklikke på fila.



Figur 8: Valget mellom forskjellige specs



Figur 9: Testresultatene gis oversiktlig i listeform på venstre side, mens høyre side viser klienten for hvert steg i testene.

10 Referanser

1. OWASP Top Ten — OWASP Foundation. https://owasp.org/www-project-top-ten/. Åpnet 8. mai 2023.