Høgskulen på Vestlandet

Fakultet for teknologi, miljø- og samfunnsvitenskap

Institutt for datateknologi, elektroteknologi og realfag

Utvikling av prosjektstyringsverktøy med tilhørende database

(Development of a project management tool with accompanying database)

Systemdokumentasjon

Versjon <1.1>

REVISJONSHISTORIE

Dato	Versjon	Beskrivelse	Forfatter
13.05.24	1.0	Dokumentopprettelse	Petter Tesdal
13.05.24	1.1	Revisjon	Kristoffer Fjeldstad Madsen

INNHOLDSFORTEGNELSE

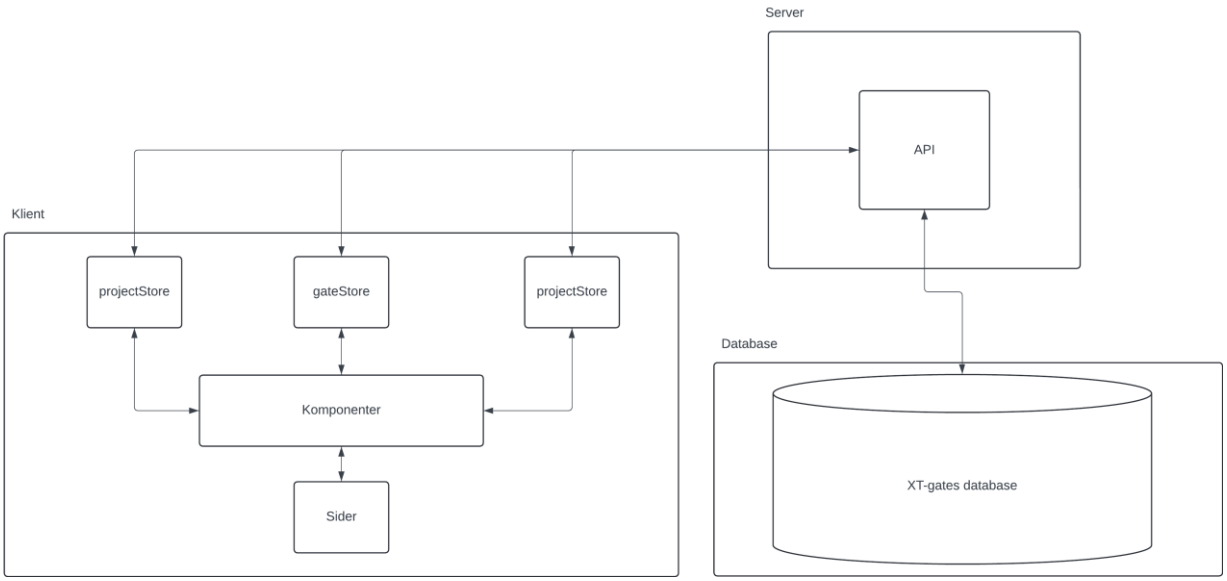
1 INNLEDNING

Dette dokumentet er skrevet som et støttedokument i forbindelse med bachelorprosjektet "*Utvikling av prosjektstyringsverktøy med tilhørende Database*" i faget DAT191. Hovedpoenget med dokumentet er å gi leseren en forståelse i hvordan prosjektet er satt opp, og hvordan dets arkitektur er satt opp.

Da prosjektes utvikling har hatt et stort fokus på frontend og det å knytte den opp til en database. Annen funksjonalitet gjennom en server har blitt lite utnyttet, da det ikke har vært tilstrekkelig med tid for det. Det er dermed lite diagrammer og utredninger for hvordan den fungerer i dette dokumentet.

2 Arkitektur

Prosjektet er laget i Nuxt og har dermed en struktur basert på reaktivitet. Det er også en mindre forskjell mellom server og klient da mye som tidligere måttet ha blitt definert i serveren har kunnet fint blitt definert i klienten, f.eks. routing. Den generelle arkitekturen vil da virke ganske simpel og er fremvist i figur 2-1.



Figur 2-1: Fremstiller en enkel arkitektur

Figuren i 2-1 har tatt utganspunkt i hvordan data behandles og sendes mellom komponenter og Stores, som så sendes videre til API og deretter lagres i Databasen.

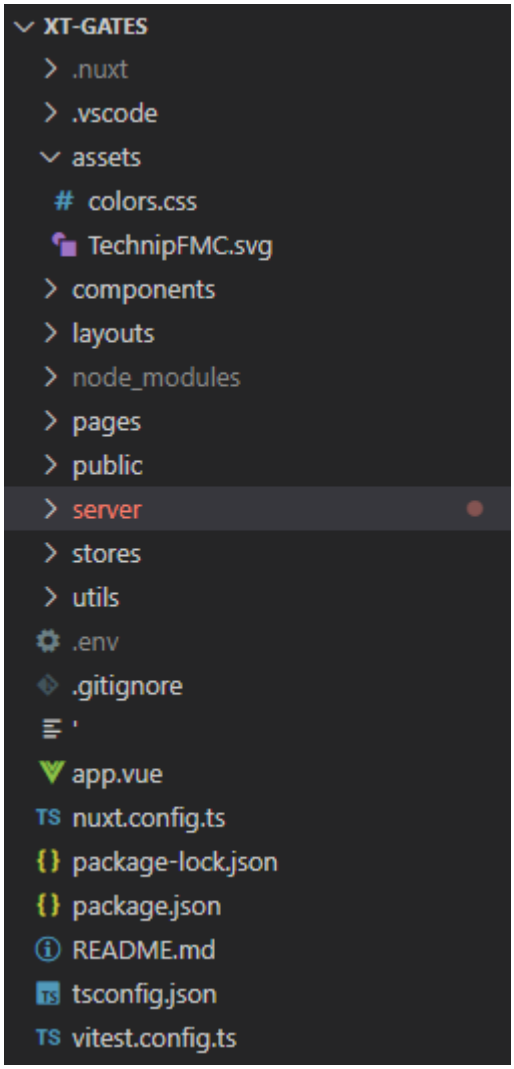
3 PROSJEKTSTRUKTUR

Prosjektets filstruktur er nokså komplisert og stort pga. Node og Nuxt sine biblioteker, så det å visualisere alle filene i hele prosjekt er ikke gjennomførbart. I stedet vises de filer som er laget og endret på av gruppen.

3.1 Generell struktur

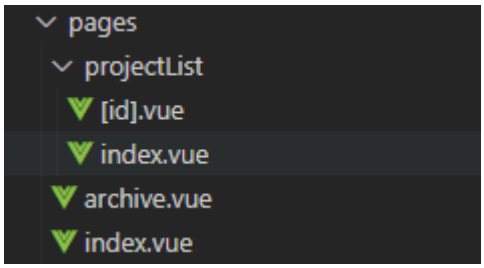
Den generelle strukturen til prosjektet er delt opp i filene vist i figuren under. Det som er verdt å merke her er package.json som fungerer som listen over alle avhengigheter prosjektet har. De fleste av pakkene listet der kommer fra Vue, mens 3 ekstra er lagt til.

App.vue er prosjektets hovedfil og fungerer som en indeks-fil i de fleste andre webapplikasjoner. Nuxt.config.ts-filen er også prosjektets innstillinger, og bestemmer f.eks hvor det kan skrues av og på utviklingsverktøy som kan brukes under kjøringen av applikasjonen.



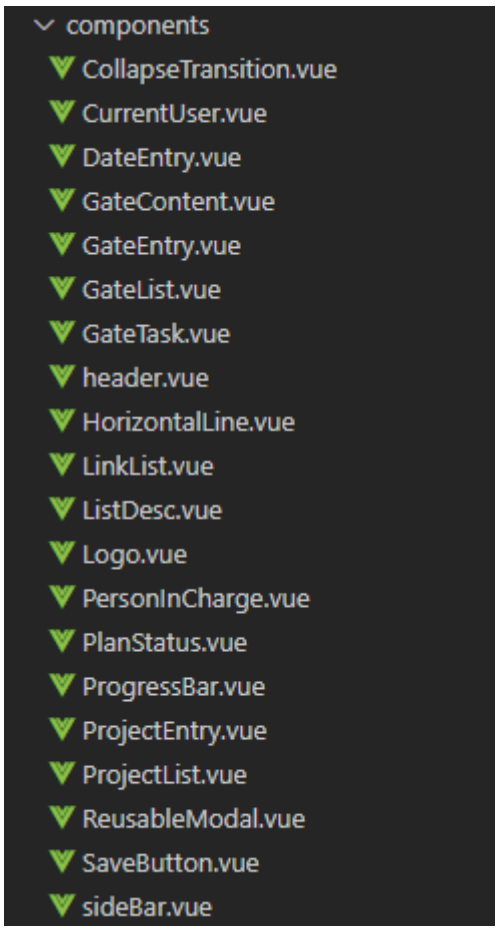
3.2 Pages

Denne mappen inneholder de forskjellige sidene som vises på løsningen. De blir routet automatisk gjennom Nuxt når de blir plassert her. Dette er dermed en mappe Nuxt søker etter for å finne og registrere.



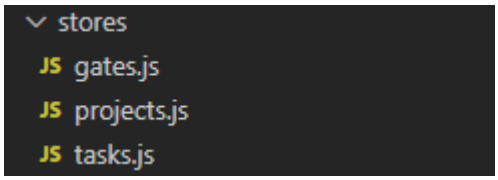
3.3 Components

Mappen inneholder alle komponentene i prosjektet. Disse er automatisk importert i komponentene slik at en komponent som er definert her, kan igjen gjenbrukes i en annen komponent. Dette er dermed også en mappe Nuxt har merket seg og søker etter.



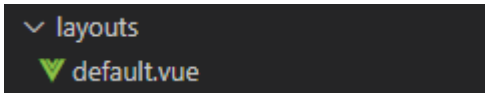
3.4 Stores

mappen er lagd i sammenheng med Pinia, som er knyttepunktet for alt av reaktivitet i applikasjonen.

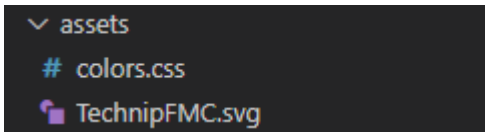


3.5 Layout

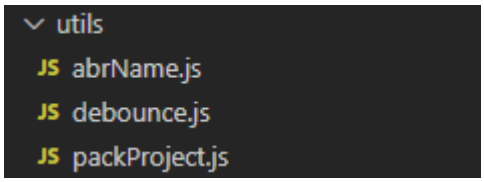
Mappen inneholder default layout for prosjektet og brukes på alle sidene



3.6 Assets

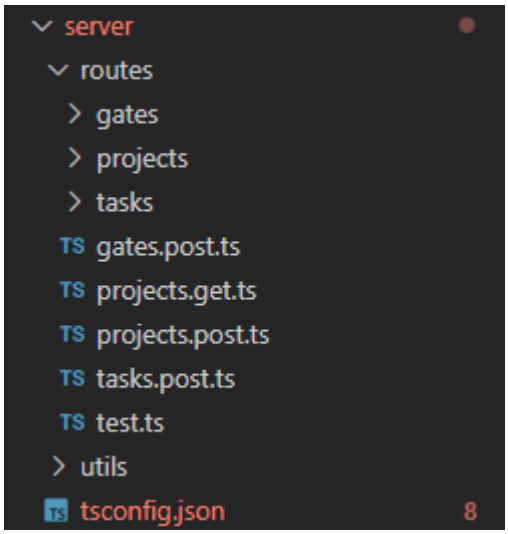


3.7 Utils



3.8 Server

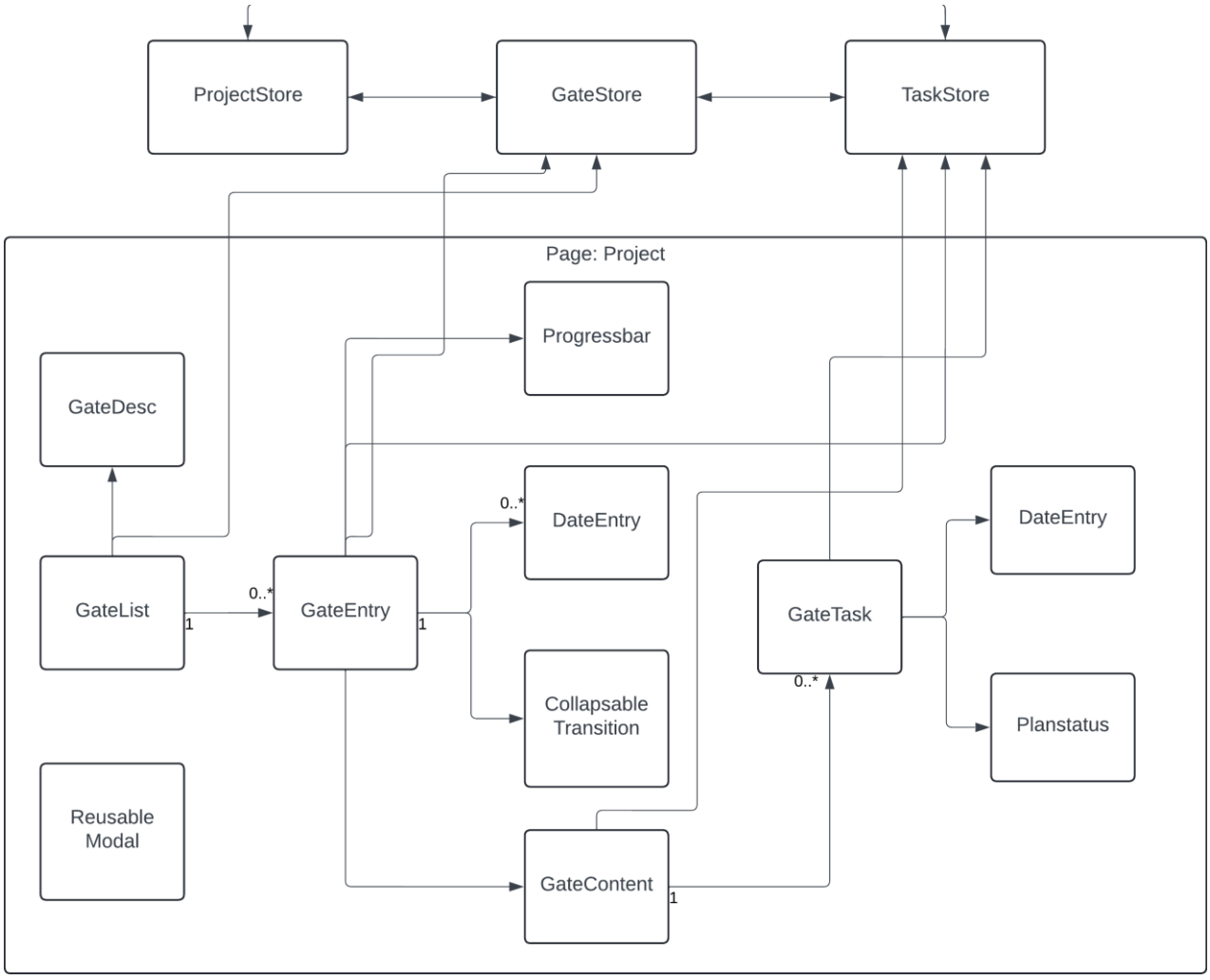
Serverstrukturen er desidert den største strukturen, og trenger ekstra arbeid da strukturen er viktig for hvordan de forskjellige API endepunktene kalles på.

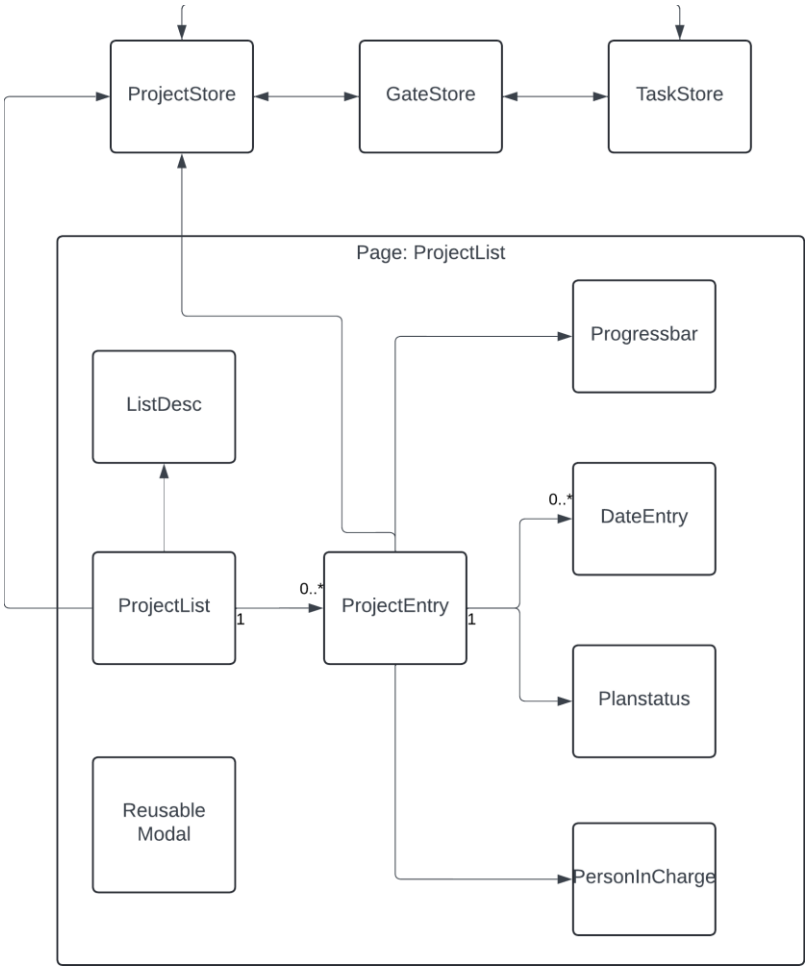

server
routes
gates
projects
tasks
gates.post.ts
projects.get.ts
projects.post.ts
tasks.post.ts
test.ts
utils
tsconfig.json
8

.

4 KLASSEDIAGRAM

Med dette prosjektet er det vanskelig å sette oppe et klassisk klassediagram, da det ikke tas ibruk noen klassiske klasser, men heller komponenter. Det er derfor gjort et forsøk på å visualisere disse komponentene og deres koblinger.

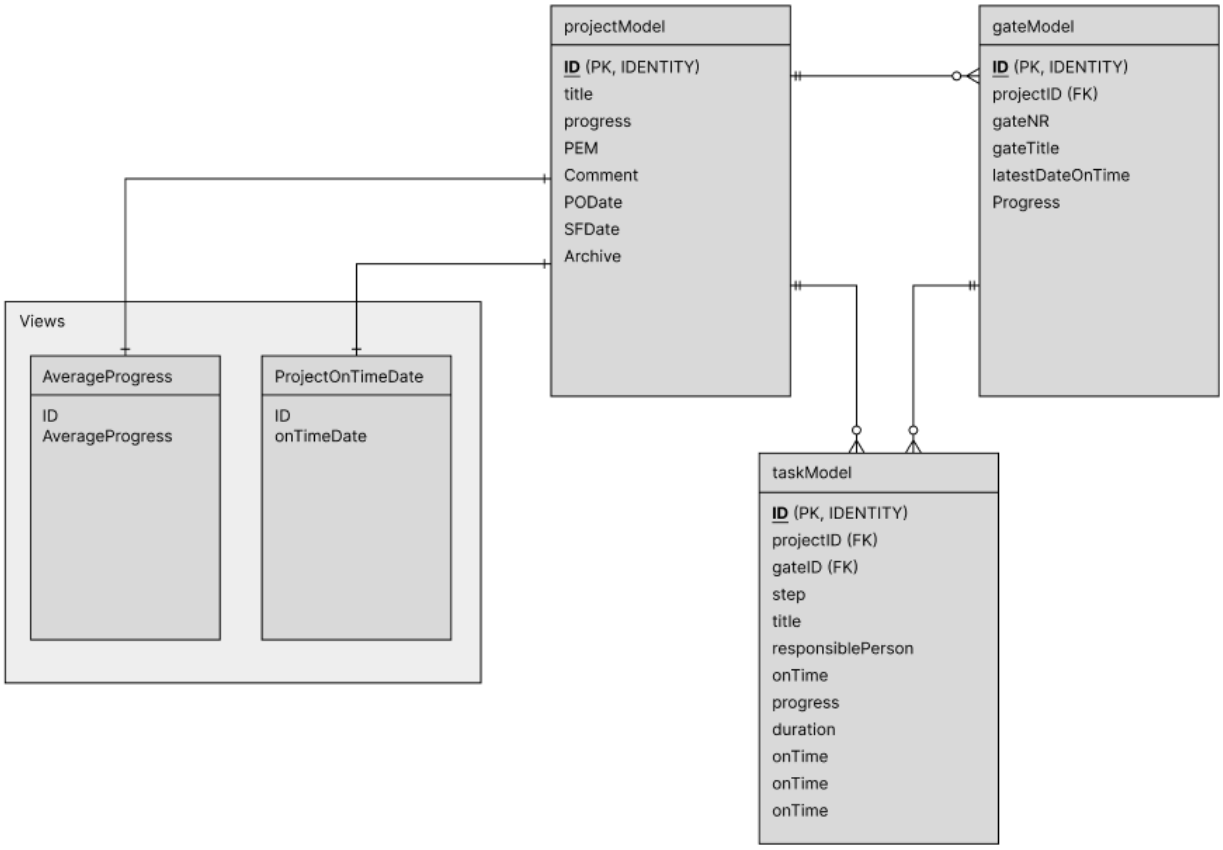



ProjectStore
GateStore
TaskStore
Page: ProjectList
Progressbar
ListDesc
0..*
DateEntry
ProjectList
1
0..*
ProjectEntry
1
Planstatus
Reusable
Modal
PersonInCharge

2

5 DATABASEMODELL

Den følgende figuren er et kråkefotdiagram som representerer databasen knyttet til applikasjonen. I tillegg til de tre tabellene er det inkludert to views som produserer konkrete variabler brukt i applikasjonen.



Figur 5-1: Kråkefotdiagram

3

6 SIKKERHET

Sikkerheten til webapplikasjonen er relatert til verifisering, autentisering og SQL-injeksjoner. Oppdragsgiver har et lukket system hvor webapplikasjonen vil bli kjørt, som man får tilgang til via Single Sign-On (SSO). Dette bidrar til at dersom brukere ønsker tilgang til webapplikasjonen må de benytte seg av en Windows PC som er godkjent i deres system, samt logge seg inn på en bruker i deres system. SSO bidrar dermed til verifisering av brukere.

SSO brukerne er ikke direkte koblet til webapplikasjonen og det er derfor ikke autentisering av ulike brukere. Resultatet er at alle brukere i webapplikasjonen har samme muligheter til å utføre endringer. Det er dermed ingen diskriminering mellom en eventuell *Admin-bruker* og *Bruker*. Dersom en ønsker å gi begrenset tilgang til en bruker er ikke dette en mulighet.

Utviklingen av produktet har ikke hatt fokus på beskyttelse mot SQL-injeksjoner. SQL-injeksjoner innebærer at en angriper uten autorisert tilgang manipulerer spørringer sendt til databasen og får tilgang til sensitiv data som angriperen ikke skulle hatt tilgang på. Dette er identifisert som et problem, men er derimot ikke blitt gjort iverksatt tiltak for å sikre mot uønskede angrep.

4

7 INSTALLASJON OG KJØRING

Løsningen er sterkt avhengig an Node og Npm for å fungere, da disse brukes for å kontrollere pakkene som må hentes og brukes for å kjøre serveren. I tillegg til disse er Nuxt noe som er nødvendig og kan installeres gjennom en enkel npm install på prosjektet og skal ikke være noe problem.

Ettersom npm fungerer veldig bra og enkelt skal denne kommandoen være alt du trenger. Dessverre vil det ikke være knyttet opp mot databasen, da denne er knyttet gjennom hemmelige variabler i en .env fil som ikke er lagt til i repositoriet. Hvis denne er ønskelig i etterkant er det mulig å ta kontakt med gruppen. Ellers er det en alternativ løsning lengre nede.

De siste avhengighetene som ikke kommer med gjennom Nuxt er Pinia og vue-draggable

For å installere programmet må brukeren:

1. Laste ned Node

https://nodejs.org/en/download

Steg- for steg forklaring:

https://www.geeksforgeeks.org/installation-of-node-js-on-windows/

2. Hente repositorien og klone fra fra github

https://github.com/h584903/XT-Gates.git

3. Finne fram til mappelokasjonen i en terminael
4. Laste ned nødvendige pakker gjennom kommandoen:

```
npm install
```

5. Start dev applikasjonen

```
npm run dev
```

Det er mulig å hente prosjektet fra løsningens frontend-vert Vercel. At denne linken er funksjonell fremover i tid er tvilsomt, men for øyeblikket er den kjørbar og tilgjengelig.

Det er viktig at dersom Vercel brukes, er man underforstått med at dette er uten en skikkelig backend, og på en server som knyttes svært dårlig til serveren som gjør løsningen tregere enn på oppdragsgivers lokaler.

https://xt-gates.vercel.app/projectList

5

8 DOKUMENTASJON AV KILDEKODE

Gruppen har ikke hatt noen implementasjon av dokumentering av koden utenfor vanlige kommentarer. Bakgrunnen bak dette er gruppens manglende erfaring med dokumentasjon av prosjekter strukturert etter Nuxts prosjektstruktur og filtyper.

En god del eksempler fra koden blir gått gjennom i rapporten, som vil gi et bedre innsyn i hvordan koden ser ut og fungerer.

Liste over komponenter:

- **CollapseTransition.vue**
  Komponent for å kollapse og vise en div.
- CurrentUser.vue
- **DateEntry.vue**
  Blir matet en dato, som så formateres riktig utifra lokalet klienten tar ibruk.
- **GateContent.vue**
  Looper gjennom tasks som skal være i gaten med den ID'en som blir matet til komponenten.
- **GateDesc.vue**
  Beskrivende tekst i en gate for å vise hva verdiene i en task er.
- **GateEntry.vue**
  Hvordan en enkelt gate skal være strukturert og metoder for å endre innholdet i en spesifikk gate.
- **GateList.vue**
  En komponent som looper gjennom gates og oppretter et GateEntry for hver loop.
- **GateTask.vue**
  En komponent som fremstiller en task, er ment for å brukes i en gate.
- **Header.vue**
  Oppretter en header for siden.
- **HorizontalLine.vue**
  Lager en horisontal linje som er formatert riktig.
- **LinkList.vue**
  Listen over prosjekter som skal linkes til, opprettes utifra prosjekter som er aktive.
- **ListDesc.vue**
  Liste med beskrivende tekster formatert slik at de passer med listen under.
- **Logo.vue**
  Logoen til TechnipFMC brukt i Header.vue.
- **PersonInCharge.vue**
  Tar inn et navn og forkorter det til kun forbokstaven fornavn og hele etternavnet
- **PlanStatus.vue**
  Fremstiller et lys basert på verdien som mates inn i komponenten, 0 blir gult og 1 blir grønt.
- **ProgressBar.vue**
  Tar inn et tall mellom 0 og 100 og fremstiller en bar som er fyllt opp med den prosent verdien.
- **ProjectEntry.vue**
  Hvordan et enkelt prosjekt skal være strukturert og metoder for å endre innholdet i et spesifikt prosjekt.
- **ProjectList.vue**
  En komponent som looper gjennom prosjekter og oppretter ProjectEntry for hver loop.
- **ReusableModal.vue**
  En komponent for å åpne og lukke en dialogboks.
- **SideBar.vue**
  Tar inn komponenten LinkList.vue for å gi lenker i sidebare.
- **TaskDesc.vue**
  Deskriptiv tekst som legges til i tasks for å beskrive tall og tekster formatert riktig.

6

Liste over Stores

- **Projects.js**
  Store som brukes til å knytte reaktive variabler til store mellom komponenter. Inneholder meste av metoder for å knytte sammen front-end og back-end for et prosjekt.
- **Gates.js**
  Store som brukes til å knytte reaktive variabler til store mellom komponenter. Inneholder meste av metoder for å knytte sammen front-end og back-end for en gate.
- **Tasks.js**
  Store som brukes til å knytte reaktive variabler til store mellom komponenter. Inneholder meste av metoder for å knytte sammen front-end og back-end for en task.

API endpoints

TASKS

- Task.post.ts

  Oppretter en ny task i databasen

- Tasks/comment/[id].ts

  Oppdaterer kommentaren for en task

- Tasks/completeDate/[id].ts

  Oppdaterer complete date for en task

- Tasks/progress/[id].ts

  Oppdaterer progress for en task

- Tasks/responsiblePerson/[id].ts

  Oppdaterer responsiblePerson for en task

- Tasks/step/[id].ts

  Oppdaterer step for en task

- Tasks/title/[id].ts

  Oppdaterer tittel for en task

- Tasks/[id]delete.ts

  Sletter en task

- Tasks/order.ts

  Oppdaterer rekkefølgen av tasks

PROJECTS

- Project.get.ts

  Henter ut alle prosjektene i databasen

- Project.post.ts

  Oppretter et nytt prosjekt i databasen

7

- Projects/[id].delete.ts
  Sletter et prosjekt
- Projects/comment/[id].put.ts
  Oppdaterer kommentar
- Projects/PEM/[id].put.ts
  Oppdaterer PEM
- Projects/POdate/[id].put.ts
  Oppdaterer POdate
- Projects/SFdate/[id].put.ts
  Oppdaterer SFdate
- Projects/title/[id].put.ts
  Oppdaterer prosjekt tittel

GATES

- Gates/[id].delete.ts
  Sletter en gate
- Gates/[id].get.ts
  Henter alle gates som tilhører prosjekt med id
- Gates/order.put.ts
  Oppdaterer rekkefølgen av gates
- Gates/title/[id].put.ts
  Oppdaterer gate tittel
- Gates/progress/[id].put.ts
  Oppdaterer progress til gate
- Gates.post.ts
  Oppretter en gate og justerer tall
- Gates/lastDate/[id].put.ts
  Oppdaterer lastDate til gate i databasen

8

9

9 KONTINUERLIG INTEGRASJON OG TESTING

Tidlig i utviklingsperioden ble det forsøkt å sette opp Nuxt test utils for Unit Testing. Dette ble derimot ikke videre implementert og tatt i bruk. Begrunnelsen for dette var at gruppen mente det var tilstrekkelig å benytte seg av konsoll. Dette innebar å skrive til konsoll når data ble lagt til, sjekke hvor data var tilgjengelig, hva som skjedde når dataen ble endret eller slettet, og feilmeldinger med eventuelle feil. Ved bruk av utskrift til konsoll har gruppen testet underveis som kode ble implementert og etter implementering av kode. De meste av utskrift til konsoll er fjernet fra koden ettersom ikke var ønsket å ha det ved overlevering til kunden.

Gjennom utviklingsperioden har gruppen benyttet seg av kontinuerlig integrasjon ved bruk av GitHub Flow. GitHub Flow består blant annet av et innebygd Kanban-brett. Her har det blitt opprettet iterasjoner avhengig av hvilken iterasjon utviklingsperioden befant seg i henhold til planen satt i Gant-Diagrammet. I hver iterasjon kan det opprettes issues (steg), som er oppgaver som må gjøres i koden. For eksempel:

“ProjectList/[id]: Endre tittel”

For oppgaven ble det lagt til en beskrivelse:

“Oppdragsgiver ønsker å kunne endre tittel for et prosjekt. Når man befinner seg i et prosjekt /projectList/[id] må det være mulig å endre tittel. Bruk en form for editor, put-request til database og oppdater store for prosjekt. Se endring av task sin progress for inspirasjon”

Om en utvikler ønsker å arbeide med oppgaven tilegner den oppgaven til sin bruker og kan deretter opprette en gren. Etter at utvikleren har fullført oppgaven i grenen sin sender den en “Pull-request” (forespørsel) til GitHub. Når forespørselen er sendt har gruppen en regel om at en av de andre utviklerne må se over endringene som har blitt gjort og gi eventuelle tilbakemeldinger dersom det oppdages problemer eller forbedringsområder. Dersom det ikke er nødvendig med tilbakemeldinger eller forbedringer kan utvikleren som har sett over koden godkjenne forespørselen. Det var ikke lov å godkjenne sin egen forespørsel. Dersom grenen som det ble sendt en forespørsel for har problemer med hovedgrenen, oppstår det en “merge conflict” (konflikt). Om dette problemet oppstår er det utvikleren som sendte forespørselen sitt ansvar å fikse konflikten. Ved ingen konflikter og gjenstående problemer ble koden fra grenen lagt til i hovedgrenen.

Totalt i prosjektet ble det opprettet omtrent 230 grener som har blitt flettet inn med hovedgrenen.

10

10 REFERANSER

11