



Universitetet i Sørøst-Norge
IT og informasjonssystemer

22. mai 2024

Bacheloroppgave

DugnadNett

Yusuf Salad Yusuf

Christian Alexander Ledaal

Kristian Otterstad Veggeland

Bakgrunn

Bakgrunnen for dette prosjektet er at vi tar bachelor i sommer 2024 og med det så trengte vi et prosjekt å jobbe med. I faget BOP3000 ved USN campus Bø kunne vi velge mellom flere prosjekter å jobbe med, alt fra prosjekter for bedrifter hvor de blir arbeidsgivere for oss til prosjekter hvor emne veileder blir arbeidsgiver. Men, vi kunne også velge å komme opp med et eget prosjekt, noe gruppemedlemmene Christian og Kristian allerede hadde ønsket og begynt å jobbe med. Prosjektideen deres hadde blitt godkjent av veileder og Yusuf ble med i gruppen etterpå.

En annen grunn for at vi valgte å jobbe med eget prosjekt, var fordi vi satte pris på muligheten for å selv kunne fremstille et prosjekt av vårt eget valg, hvor vi selv definerer krav, funksjonaliteter og gjennomføring. Dette lar oss sette alt det vi har lært gjennom studiet i praksis sånn som i APP2000 faget, men på en høyere standard og med mindre tid. Og med muligheten for å selv definere prosjektet så har vi også ekstrem frihet for hvordan vi løser problemstillingen, med valg av teknologi, verktøy, begrensninger, målsetting, prioriteringsliste og ambisjons grad.

Innholdsfortegnelse

Bakgrunn.....	2
Figurliste.....	5
Problemstilling.....	6
Administrasjon.....	7
Organisering	7
Roller og ansvar.....	7
Kristian:.....	8
Christian:	8
Yusuf:.....	8
Framdriftsplan	9
Verktøy og kommunikasjon	11
Discord	11
VSCode	11
Git og GitHub.....	11
ChatGPT.....	11
Utfordringer	12
Mangel på kompetanse.....	12
Forandring på løsningsarkitektur	12
Sikkerhet og personvern	12
Målsetting	13
Resultatmål.....	13
Effektmål.....	13
Kravspesifikasjoner	14
Funksjonelle krav.....	14
Ikke funksjonelle krav	15
Utviklingsmetode	16
Arbeidsmetodikk	16
WBS og Gant -Chart.....	17
WBS	17
Gantt-chart.....	18
Trello Board	19

Kontinuerlig testing med felles plan	21
Testing	21
Testplan	21
Testmal	22
Teknologier vi bruker	23
NodeJS	23
NextJS	23
PostgreSQL og Prisma	24
HostHatch	25
Systemdokumentasjon	26
Utplassering	26
Søk og filtrering	27
Identitets og Tilgangskontroll	29
Autentisering	29
Autorisasjon	29
Middleware	29
Optimaliseringer	30
Skeletons	30
Progressiv bilder	30
Bildekomprimering	30
Dugnader	30
Deltakelse	30
Gruppechat	31
Opprettelse	32
Innstillinger	32
Profilert	33
Konklusjon	34
Måloppnåelse	34
Hva som kunne ha blitt bedre	35
Websockets	35
Organisering av arbeid	35
Planlegging	36

Hva som mangler	37
Threads	37
Admin funksjoner	38
Anmeldelser	38
Erfaringer og utbytte	40
Erfaringer	40
Tekniske erfaringer	40
Samarbeid og kommunikasjon	40
Utbytte	41
Link til kildekode	41
Referanser	42

Figurliste

Figure 1 Work Breakdown Structure diagram	17
Figure 2 Gantt Chart	18
Figure 3 Trello Sprint 3	19
Figure 4 Trello Backlog Storage	20
Figure 5 Forbedring i ytelse etter overgang til HostHatch	25
Figure 6 Utplasseringsdiagram	26
Figure 7 Eksempel for tilstand I URL	27
Figure 8 Bilde av område filtering	28
Figure 9 Gruppechat	31

Problemstilling

I nåtiden er det vanlig å arrangere dugnader gjennom sosiale medier. Disse plattformene når ut til mange, men de har også noen begrensninger. Slik som mangel av funksjoner rundt deltakelse og organisering av dugnader. For eksempel kan informasjon lett drukne i mengden av andre innlegg på siden, og det er vanskelig å vite hvem som faktisk møter opp ettersom man kan vise interesse uten å faktisk forplikte seg. Som leder oss til vår problemstilling:

"Hvordan kan en webapplikasjon forbedre dugnadorganisering ved å overkomme begrensninger ved sosiale medier?"

For å dekke behovene ved dugnadorganisering, holder lager vi en egen webapplikasjon. Denne applikasjonen skal fokusere på å være direkte nyttig for å organisere dugnader. For eksempel inndeling av ulike dugnadsoppgaver, tilpassede varsler, feedback system og gode kommunikasjonskanaler. Vi tror dette kan hjelpe med utfordringene som dukker opp når man skal koordinere en dugnad. Selvfølgelig så bygger vi en fullstack applikasjon for å løse problemstillingen vår, som betyr at vi naturligvis har en frontend og en backend løsning som vi selv designer og bygger for å best kunne løse problemstillingen. Dette betyr naturligvis at vi må ta mange valg som ved valg, biblioteker og rammeverk som skal brukes for frontend. Database løsning, med valg av type database og hosting side, server logikk og design av api-er.

Administrasjon

Organisering

Vi er tre studenter som jobber med dette prosjektet. Siden ikke alle studentene går på samme linje og har like fag, vil mesteparten av arbeidet være gjort i isolasjon fra hverandre. For at vi skal kunne styre dette arbeidet har vi satt opp faste datoer der vi møter og diskuterer prosjektet. Vi går igjennom hva hver person har gjort siden hvert sitt møte og gir tilbakemelding på om alle i gruppen synes det er godt nok. Videre vil vi da diskutere hva som må gjøres videre, dele ut nye oppgaver og diskutere eventuelle problemer som gruppen eller medlemmer har.

I disse møtene har vi da planlagt arbeidet opp mot de oppgavene vi har. Som nevnt i arbeidsplanen vil vi ta i bruk sprinter for å organisere arbeidet. Dette betyr da at vi alltid ville jobbe imot et overordnet mål/ milepæl. Som nevnt tidligere vil vi da også ha et møte for hver sprint vi fullfører, disse vil da verifisere at alt av arbeidet er gjort i tråd med kravene vi har spesifisert i krav og rammebetingelser. Videre vil en også da etablere om alle gruppemedlemmer er fornøyde og om de eventuelt har noe bredere tilbakemeldinger de vil gi.

Roller og ansvar

I dette prosjektet har alle medlemmer tatt opp forskjellige roller. Dette er fordi prosjektet er stort og mye av arbeidet er avhengig av hverandre. Siden vi vil jobbe mye separat fra hverandre kan det kjapt oppstå problemer eksempelvis at en medlem må vente på to andre for å gå videre. For å løse dette har vi da laget spesifikke roller som da gir gruppemedlemmer ansvar for visse deler av prosjektet. Dette lar gruppemedlemmer jobbe spesifikt med sin del av prosjektet på en måte som ikke vil hindre andre medlemmer progresjon.

Kristian:

Det var naturlig at Kristian var gruppeleder for dette prosjektet. Han hadde tatt denne rollen i flere lignende prosjekter og var også godt kjent med utstyret og teknologien som ble brukt. Han fulgte opp alt arbeidet som ble gjort og hadde ansvar for å styre hvem som skulle gjøre hva. Videre hadde han ansvar for logging av både arbeid og møter. Databasen var hans hovedansvar i utviklingsfasen. Han styrte arkitekturen og var til stede når alle endringer ble gjort imot databasen med bruk av prisma. Videre jobbet han med flere andre deler av applikasjonen og hadde hovedansvaret for å skrive prosjektrapporten.

Christian:

Christian var den mest tekniske erfarne med teknologien som ble brukt i dette prosjektet. Derfor var han hovedutvikleren i dette prosjektet. Han styrte alt fra intern logikk i applikasjonen og forsikret at alt som ble implementert fungerte sammen i selve prosjektet. For å gjøre dette var han også hovedansvarlig for både deployment og maskinvaren som ble brukt. Testplanen var også Christian sitt ansvar og for at testerne gjennomførte testplanen og samlet inn resultatet etter hver test runde. Han deltok også en del i designfasen og hadde ansvar for at valgene som ble gjort i designfasen og senere passet kunne bli implementert med de verktøyene om teknologien som ble brukt.

Yusuf:

Yusuf hadde som hovedansvar designet og utseende til applikasjonen. Han genererte tidlig utseende for alle delene av applikasjonen og deltok når endringer måtte gjøres underveis i utviklingsfasen. Videre var han med på utviklingen av noen av delene av applikasjonen. Han deltok også en del i rapportskrivningen.

Framdriftsplan

Vi planlegger å jobbe med sprinter som da vil vare i to til tre uker. Disse sprintene vil da være koblet opp imot milepæler. Milepælene er grovt knyttet til de arbeidskravene som eksisterer i BOP 3000. Videre har vi også åpnet for å lage våre egne milepæler hvis det er nødvendig. Vi bruker kanban metodikk for å dele opp arbeidet mellom oss med bruk av flere trello boards. Videre bruker vi også et gantt chart for å vise hvordan vi tenker oss at arbeidet skal se ut og hvordan vi skal jobbe oss fremover. Her vil da arbeidet være delt opp i sprinter som da er knyttet til de milepælene vi har satt oss.

Milepæle	Beskrivelse	Endepunkt	Tidsbruk
Milepæle 1: Ferdig design og prototype	I denne perioden skal vi bestemme oss for hvordan gruppen mener prosjektet skal fungere, hva slags metoder eller teknologier som skal brukes og hvordan appen skal se ut. Videre vil vi også lage en veldig enkel prototype som ikke vil ha mye funksjonalitet slik at vi vet at designet fungerer.	25/02/2024	150 timer
Milepæle 2 : Hoved funksjonaliteter implementert	Her skal vi da implementere all hovedfunksjonalitet til appen som vi rekker i løpet av denne perioden. Videre vil vi også da jobbe med designet og forandre på de tingene som vi ikke likte fra forrige milepæl.	25/03/2024	250 timer
Milepæle 3 : Implementering av alle funksjonelle	Her vil vi da implementere funksjonaliteten som mangler for applikasjonen samtidig som vi da fikser på eller forandrer på feil fra forrige	28/04/2024	300 timer

krav og ikke funksjonelle krav	milepæl. Samtidig vil vi gå over applikasjonen vår og forsikre oss at den ikke bryter med de ikke funksjonelle kravene vi har spesifisert.		
Milepæle 4 : Testing og final release	Vi tester appen grundig og fikser alt det som gjenstår å gjøre. Vi vil også fullføre prosjektrapporten i denne perioden.	20/05/2024	200 timer

Verktøy og kommunikasjon

Discord

I vårt prosjekt bruker vi Discord for å holde kontakten og samarbeide. Vi har opprettet en egen server med flere tekst- og talekanaler til ulike formål. Dette gjør det enkelt å holde alle oppdatert og sørge for at viktig informasjon ikke blir borte. Vi bruker også Discords skjermdeling for å vise kode og demonstrere løsninger.

VSCode

VSCode er verktøyet vi bruker for koding i prosjektet. Alle i teamet har installert utvidelser som støtter de programmeringsspråkene vi jobber med. Vi bruker Code Together extensionen for pair programming, som gjør det mulig for flere utviklere å samarbeide i sanntid på samme kodebase. Vi bruker også Prettier for å få et standard kode format.

Git og GitHub

Git og GitHub er hvordan vi håndterer kildekoden i prosjektet vårt. Vi bruker Git til å kontrollere alle endringer i koden, hvor vi da kan alltid gå tilbake til tidligere versjoner om nødvendig. GitHub hoster kildekoden vår og gjør det mulig for alle i gruppen å samarbeide på en enkel måte. Vi har valgt å alltid pushe direkte til hovedgrenen. Dette muliggjør for flest endringer på kort tid, men medfører også høyere risiko for at feil kan oppstå. Vi bruker også GitHub Actions for å automatisere deployment prosessen.

ChatGPT

I prosjektet har vi brukt ChatGPT for å generere testdata for applikasjonen. Dette har vært nyttig i områder som dugnader hvor det er masse data som må fylles ut og generelt i andre database tabeller når vi trengte å teste ting ut med flere brukere. ChatGPT har hjulpet oss med å skape datasett som vi har brukt for ulike scenarioer og sørge for at applikasjonen fungerer som forventet under ulike forhold.

Utfordringer

Mangel på kompetanse

En utfordring vi kom utfor veldig fort og tidlig i prosjektet var mangel på kompetanse innad i gruppen. Med et gruppemedlem som ikke hadde hatt applikasjonsutviklings emne, var det tungt å komme rett ut i dette utviklingsprosjektet hvor det var store behov for kunnskap som skulle ha vært innlært. Dette gjorde det slik at han hele tiden hang etter og brukte mer tid på å lære enn å kode for prosjektet.

Forandring på løsningsarkitektur

Underveis i prosjektet så endret vi noe av løsningsarkitekturen som betyr at vi endret hva slags verktøy som ble brukt, dette kom i formen av endring på databasen. Hvor vi gikk fra å bruke en vanlig postgresql database som ble hostet på nettet til en "VM" database.

- Isolasjon
- Fleksibilitet
- Bærbarhet
- Ressursoptimalisering
- Testing og Utvikling

Sikkerhet og personvern

Det er strenge regler for behandling, bruk og spredning av personinformasjon og med dette er standarden for sikker behandling av personinformasjon på nett høy. Dette har medført at vi har hatt et behov for å tilegne oss mer og oppdatert kompetanse av GDPR, både i form av lover og praktisk "know how" av hvordan man bygger en sikker nettside med god beskyttelse for personinformasjon. Dette har naturligvis lagt på mer kompleksitet og tidspress for å komme i mål og har derfor vært en utfordring gjennom utviklingen av denne nettsiden.

Målsetting

Vi setter mål for å hjelpe oss med organisering, planlegging og gjennomføring av prosjektet. Det er vanskelig å vite hvor man skal uten gode mål, det blir som å sikte på i mørke nesten helt umulig. Uten mål så kan man ikke planlegge, du vet ikke hva du trenger og hvor god tid du har på deg osv. Uten mål så blir det også veldig vanskelig å måle eller kvantifisere i noen form, hvor langt man har kommet, hvor mye man har fått gjort og om man har fått til det som var planlagt osv. Så naturligvis så er det viktig for oss og generelt alle som driver med noe prosjekt av noe slag å sette mål for så og bruke disse målene som guides for å komme gjennom prosjektet.

Resultatmål

Resultatmål handler om å beskrive de konkrete og målbare resultatene eller produktene som skal oppnås etter hvor stor innsats eller aktivitet man har lagt i det. Disse målene fokuserer på resultatet av det man gjør, for eksempel antall enheter produsert, økning i inntekt, reduksjon i kostnader, antall kunder man solgt til eller hvor mange deals man har gjennomført. Resultatmål er vanligvis målbare, spesifikke og kvantifiserbare, noe som gjør dem lettere å evaluere og følge opp.

Effektmål

Effektmål går dypere enn resultatmål og ser på de langsiktige endringene eller virkningene av aktiviteter eller innsatser. Disse målene handler om å måle den reelle påvirkningen av det man gjør på samfunnet, organisasjonen eller målgruppen, og hvordan det bidrar til å oppnå overordnede mål. Effektmål kan omfatte endringer i holdninger, kunnskap, atferd, eller samfunnsmessige forhold, for eksempel å redusere fattigdom, øke utdanningsnivået, eller forbedre helseindikatorer. De kan være mer komplekse å måle enn resultatmål, da de ofte krever langsiktig oppfølging og analyse av ulike faktorer som kan påvirke resultatene.

Kravspesifikasjoner

Krav er en viktig del av utviklingen av et prosjekt. Det er her vi tar for hele prosjektbeskrivelsen og deler dem opp i mindre deler. Disse delene er også konkrete krav som både utviklere og kunder kan forstå og si seg enig om.

Funksjonelle krav

Funksjonelle krav er spesifikke handlinger som produktet må gjennomføre for å fungere. Disse er små og vil alltid beskrive kun en funksjon/handling til et system. De blir oftest presentert med bruk av brukerhistorier. Brukerhistorier er korte setninger som beskriver en handling som en bruker gjennomfører:

Krav nr.	Brukerhistorie for funksjonelle krav
1	Som en bruker vil jeg kunne se hvilke dugnader som er tilgjengelig, slik at jeg kan melde meg opp.
2	Som en bruker vil jeg kunne opprette dugnader, slik at jeg kan få hjelp med arbeidet.
3	Som en bruker vil jeg kunne spesifisere hva slags dugnader jeg vil se på basert på type og lokasjon.
4	Som en bruker vil jeg kunne kommunisere med andre deltagere og ledere for å organisere arbeidet og logistikken.
5	Som en bruker vil jeg kunne fjerne deltagere som ikke oppfører seg.
6	Som en bruker vil jeg kunne gi attest til at deltagere deltok på dugnader jeg ledet.
7	Som en bruker vil jeg kunne sende meldinger til andre brukere.
8	Som en bruker vil jeg kunne legge til andre brukere som venner.

9	Som en bruker vil jeg kunne følge med på hva mine venner holder på med.
---	---

Ikke funksjonelle krav

Ikke funksjonelle krav er ikke knyttet til selve funksjonaliteten til et produkt. Det er heller begrensninger som pålegges prosjektet for å sikre at produktets kvalitet måles opp til visse attributter som sikkerhet, ytelse og skalerbarhet. De sikrer at prosjektet ikke bare fungerer, men også møter brukerens behov. Vi bruker ikke brukerhistorier for å definere funksjonelle krav.

Krav nr.	Ikke funksjonelle krav
1	Applikasjonen skal være brukervennlig og følge andre relaterte webstandarder.
2	Applikasjonen skal ha en god ytelse slik at den brukes effektivt.
3	Applikasjonen skal oppfylle de kravene som er spesifisert i GDPR og henholde til lovverk relatert til personvern.

Utviklingsmetode

Arbeidsmetodikk

Det ble tidlig bestemt at vi skulle ta i bruk Kanban i dette prosjektet. Kanban er en populær metodikk for utvikling grunnet dens fleksibilitet og skalerbarhet (Wrike. u.å.). Den visualiserer arbeidsflyten gjennom et kanban-brett og deretter kan en enkelt identifisere problemer i prosessen og forbedre arbeidsflyten over tid. Dette fokuset på kontinuerlig forbedring gjør Kanban spesielt egnet for dynamiske og iterative utviklingsprosesser som er vanlige innen softwareutvikling. I tillegg tilbyr Kanban en mer lettvektig tilnærming enn tradisjonelle rammeverk som Scrum, noe som gjør det lettere å tilpasse seg endringer og redusere unødvendig overhead.

Gruppemedlemmer hadde tidligere erfaringer med at Scrum kunne være tungvint å bruke, spesielt med planlegging av sprinter og estimering av tidsbruken (Rehkopf, M. u.å.). Denne erfaringen ledet dem til å foretrekke Kanban, da det ga dem mer autonomi og mindre rigide strukturer. Ved å kombinere Kanban med sprinter, fikk gruppen det beste ut av begge metodikker. Sprinteren fungerte som mindre prosjekter, der gruppemedlemmene kunne fokusere på spesifikke oppgaver, mens Kanban-brettet fortsatt tillot dem å håndtere uforutsette hendelser og jobbe med andre oppgaver som måtte oppstå mellom sprintene. Dette skapte en smidig og fleksibel utviklingsprosess som passet måten gruppen jobbet.

WBS og Gant -Chart

WBS

I starten av prosjektet ble gruppen enig om å bruke WBS (Work Breakdown Structure). Dette lot gruppen kjapt dele arbeidet inn i mindre deler som en kunne jobbe imot iterativt. Arbeidet ble delt inn i fire grupper som fulgte etter hverandre og som fungerte som en form for milepæler for gruppen. Det er her viktig å peke ut at innholdet i disse gruppene ikke var satt fast i stein når det ble fullført. Mye av arbeidet som tilhørte eksempelvis designfasen måtte forandres senere under utviklingen. Hovedpoenget med vært WBS var at det lot gruppen kjapt visualisere hva som skulle gjøres hvor og ga oss en god innsikt i starten av hva som måtte gjøres først.

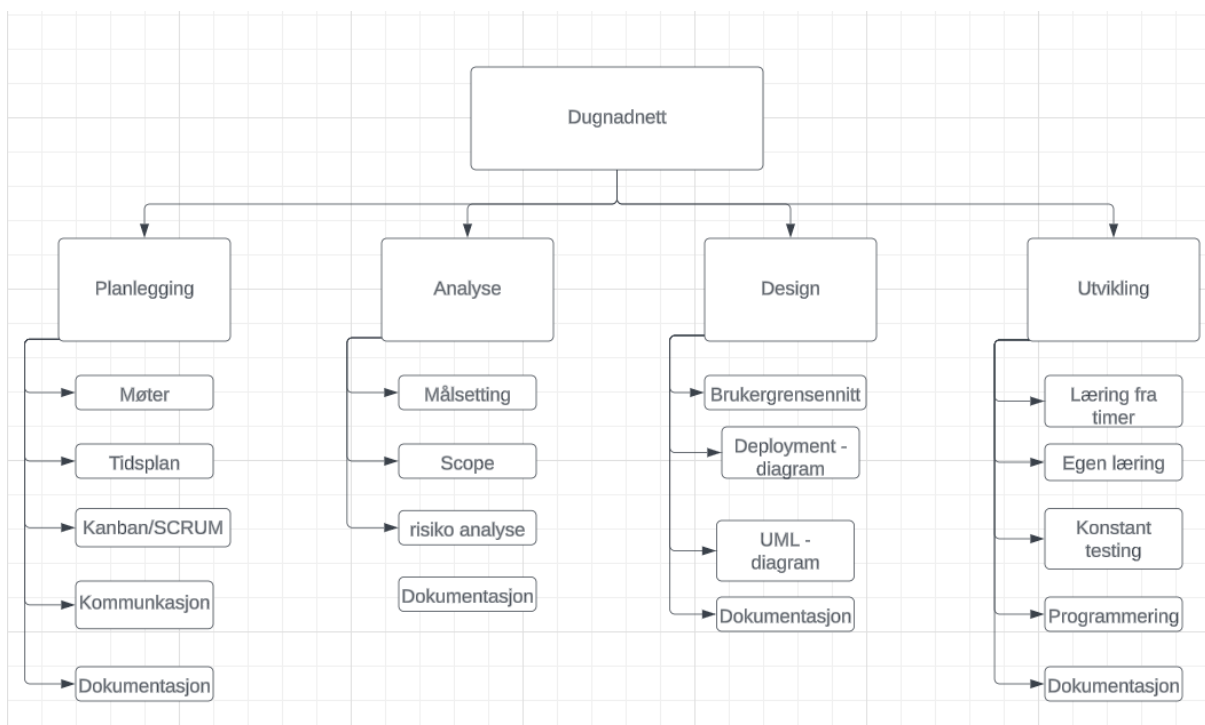


Figure 1 Work Breakdown Structure diagram

Gantt-chart

Etter vi lagde wbs ble det bestemt at vi burde ta i bruk et gantt-chart for visualisere vårt wbs. Et Gantt-chart er et visuelt verktøy som brukes til å planlegge, koordinere og spore tidsplanen for et prosjekt. Det viser tidslinjen for ulike oppgaver eller aktiviteter i prosjektet, samt deres avhengigheter og rekkefølgen de skal utføres i. Hvert oppgave element i Gantt-diagrammet representerer en spesifikk aktivitet, og dens lengde indikerer hvor lang tid det tar å fullføre. Med dette på plass ble da sprintene gruppert inn i fasene som ble spesifisert i vårt wbs og som ble gjennomført plassert i kronologisk rekkefølge. Dette gjorde det også mulig å koble de forskjellige arbeidsfasene inn mot sprintene våre og ga innsikt i hvordan vi lå i forhold til den planlagte fremdriftsplanen til prosjektet. Gantt charter vårt ligger på siden under:

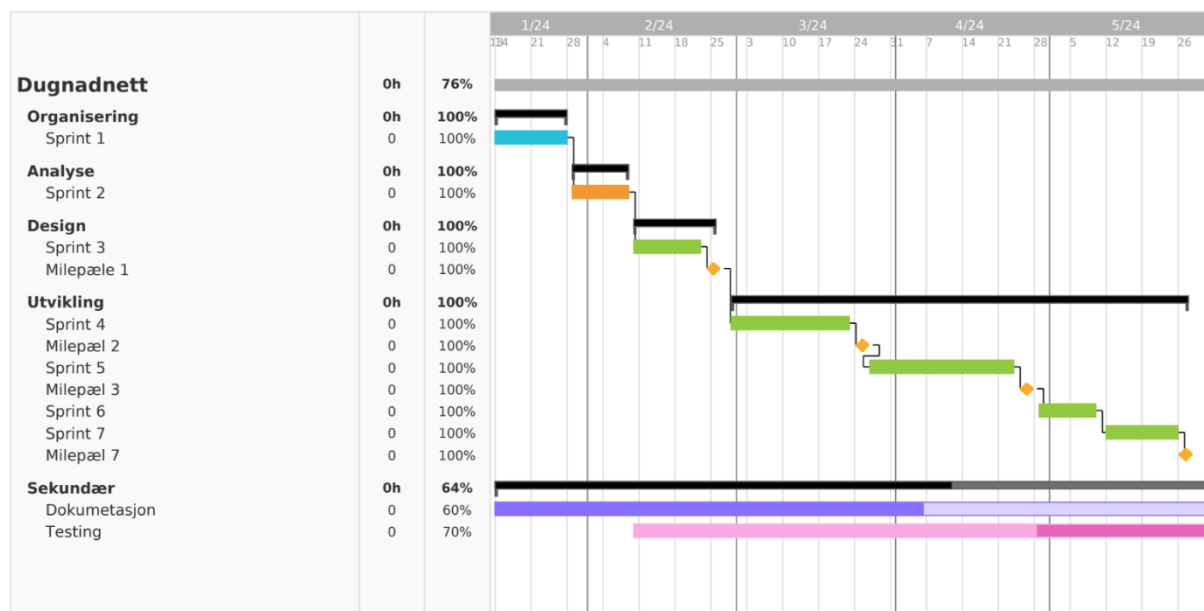


Figure 2 Gantt Chart

Trello Board

Trello ble brukt for å styre hvem som gjorde hva. Det er en webbasert samarbeidsplattform som organiserer prosjekter i virtuelle tavler (Trello. u.å.). Hver Trello-tavle fulgte oppsettet til kanban brett, hva som skal gjøres, hvem som gjør det og om det er gjort. Som nevnt tidligere jobbet vi også med sprinter. De varte i to uker med et unntak og inneholde alle oppgavene som gruppemedlemmer jobbet med i denne tidsperioden. her kan vi se hvordan et av disse brettet så ut med sprint 3:

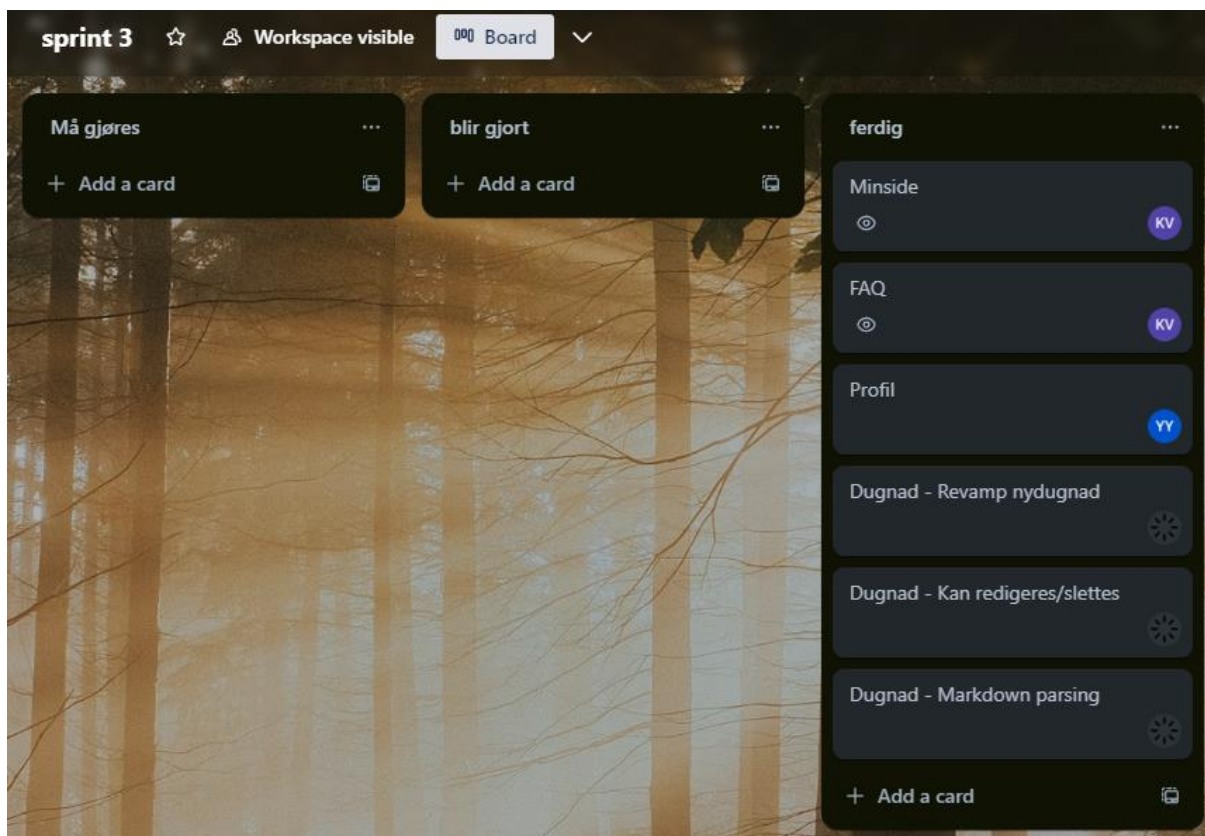


Figure 3 Trello Sprint 3

Vi hadde også en ekstra tavle kalt backlog storage. Denne ble laget fordi det ikke alltid ble gjort i sprintene. Grunnene til dette var flere, blant annet at noen gruppemedlemmer ikke rakk alt eller at det ble bestemt at en burde gjøre noe annet i stedet. Tavlen inneholdt da alt som måtte gjøres i prosjektet som sto igjen og var delt inn i grupper basert på type arbeid. Tavlen så slik ut i slutten av prosjektet:

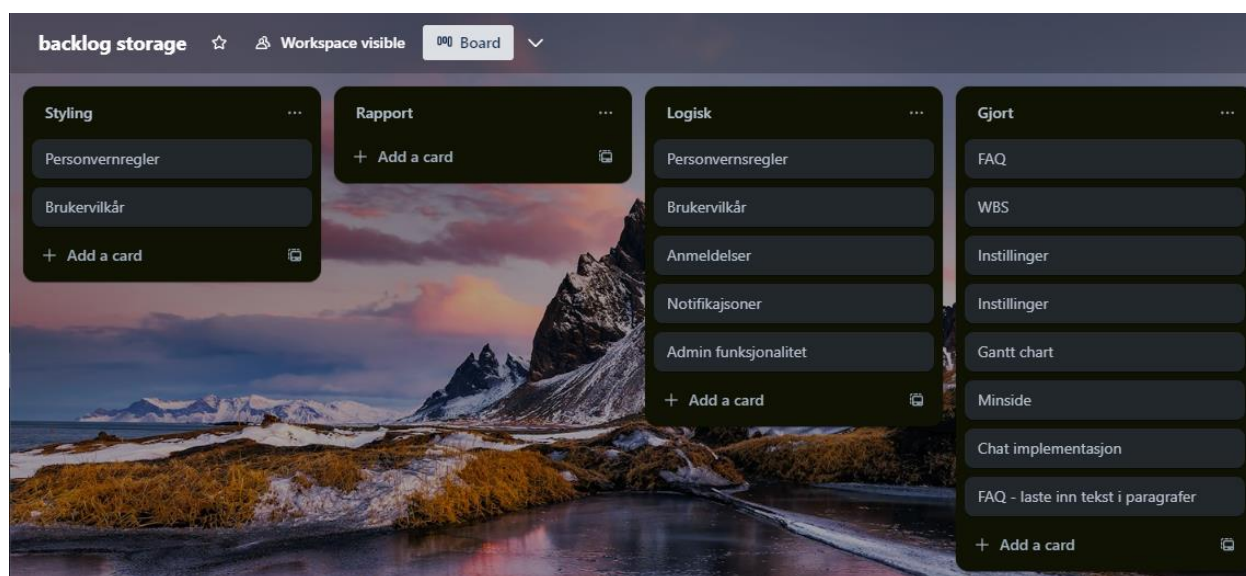


Figure 4 Trello Backlog Storage

Kontinuerlig testing med felles plan

Testing

I dette prosjektet ble det mye testing. En del av denne testingen var bare enkel kode testing gjort av en eller flere gruppemedlemmer. Denne typen testing sjekket kun om de delene av applikasjonen som arbeidet med fungerte. Videre er dette kanskje godt nok for enkle funksjoner, men hvis en skal utvikle en applikasjon burde en teste på flere måter. For å løse dette problemet valgte vi å ta i bruk brukertesting. Brukertesting er når du lar en person som ikke har noe tidligere kjennskap til funksjonaliteten den skal teste (Omniconvert. 2024). Videre vet de ikke hvordan programvare henger sammen. Brukertesting lot en teste flere ting enn funksjonaliteten. Den vil også gi et innblikk i brukervennligheten til applikasjonen, om utseende egner seg, bugs og feil en ikke ville ha oppdaget selv og forslag om hvordan applikasjonen kunne forbedres.

Testplan

Testplanen vi brukte var egnet til det som ble testet. For å enkelt gjøre hele prosessen ble det lagd en mal som alle testene fulgte. Etterpå ble resultatene logget slik at de kunne sammenlignes etterpå. Brukeren ble gitt et enkelt skjema med spørsmål. Disse spørsmålene generere enkle svar, eksempelvis ja/nei eller 1-6.

Testingen ble gjennomført med enkle utstyr og verktøy. Vi brukte discord til å kommunisere med testerene. Testene ble gjennomført med bruk av google docs der de skrev ned svarene sine. Hver tester ble introdusert til applikasjonen av en av utviklerne. Den ble da logget inn og registrert på nettsiden. Videre gikk også gruppemedlemmene gjennom test strukturen og hvordan testen skulle gjennomføres. Etter testen var gjennomført sendte testerene dokumentet inn til gruppemedlemmet.

Spørsmålene ble delt opp i to typer, BV (brukervennlighet) og A (akseptanse). BV spørsmålene var ja nei spørsmål som spurte om bruker mente om applikasjonen var brukervennlig. Akseptanse spørsmålene skulle svare på brukerhistorier som var egnet

for å teste funksjonaliteten til applikasjonen. Disse spørsmålene ble rangert fra 1-6, noe som ga mer innsikt i hvor fornøyd testeren var.

Testmal

ID	Spørsmål	Svar	Andre kommentarer
BV1	Spørsmål 1	Ja.	Ingen kommentarer.
A1	Spørsmål 2	3	Ingen kommentarer.

Teknologier vi bruker

NodeJS

NodeJs er et Javascript Runtime miljø som gjør det mulig for utviklere å jobbe med javascript på serversiden av applikasjonen (OpenJS Foundation. u.å.). Dette gjør det slik at utviklere kan skrive server-side kode ved hjelp av javascript. En annen ting som NodeJs har og brukes ofte for er commonS modul pakkesystemet, dette pakkesystemet brukes for å organisere og gjenbruke kode i form av moduler.

NodeJs kommer også med NPM(Node package manager), dette er et stort pakke håndteringssystem for javascript. NPM gjør det enkelt å installere, administrere og dele pakker av kode, og det spiller en viktig rolle i Node.js-utviklingsprosessen.

Som sagt så brukes NodeJs for server-side utvikling: Node.js brukes til å bygge server-side applikasjoner og tjenester. Utviklere kan skrive backend-logikk og håndtere nettverk forespørsler ved hjelp av JavaScript, og dermed oppnå en enhetlig kodebase på både klient- og serversiden.

NextJS

Next.js er et rammeverk for React som tar utviklingen av nettapper til et nytt nivå ved å tilby et stort og veldokumentert bibliotek av verktøy for bruk i utviklingen. NextJs er bygget opp på React og tilbyr en rekke funksjoner og fordeler som gjør det til et populært valg blant utviklere (Vercel, Inc. u.å.).

En av de mest fremtredende funksjonene til Next.js er automatisk routing. Mens du med React vanligvis må håndtere routingen selv, tar Next.js seg av dette for deg. Dette sparer tid og energi, samtidig som det gir en enkel og konsistent måte å håndtere routing på. Next.js tilbyr også muligheten til å tilpasse routingen gjennom egendefinerte konfigurasjonsfiler, slik at utviklere fortsatt har kontrollen når det er nødvendig. I tillegg til automatisk routing, tilbyr Next.js andre funksjoner som er designet for å forenkle

utviklingsprosessen og forbedre ytelsen til nett applikasjonene dine. Dette inkluderer innebygd støtte for server-side rendering (SSR) og statisk side generering (SSG), som bidrar til å optimalisere ytelsen til nettstedet ditt og forbedre brukeropplevelsen.

Next.js har også et sterkt økosystem med tilleggsverktøy og plugins som kan hjelpe deg med å bygge og skalere nettsidene dine. Dette gjør det til et allsidig og kraftig verktøy for å bygge moderne nettsider. Alt i alt tilbyr Next.js en enkel og effektiv måte å bygge React-apper på, samtidig som det gir avanserte funksjoner og ytelse fordeler som kan hjelpe deg med å ta nett applikasjonene dine til neste nivå.

PostgresSQL og Prisma

Vi bruker PostgreSQL som database og Prisma som ORM. Object–relational mapping (ORM) gjør det lettere å snakke mellom applikasjonen og databasen, ved å være et oversettelses lag for å konvertere data fra applikasjonen til data som databasen ønsker.

PostgreSQL er en SQL basert database som har god funksjonalitet og ytelse. Den takler avanserte datatyper, komplekse spørringer og transaksjoner. PostgreSQL er også god på å håndtere store datamengder og samtidige brukere i sanntid som alle har forskjellige transaksjoner. (PostgreSQL Global Development Group, u.å.)

Prisma hjelper oss å slippe å styre med komplisert SQL kode. Det gir oss type sikkerhet, automatiske migrasjoner og et enkelt kontrollpanel for å håndtere CRUD operasjoner Create, Read, Update og Delete. (Prisma Data, Inc. u.å.) Prisma lar oss sette opp datamodeller som vi kan endre på i etterkant uten mye problemer. Dette fører til redusert tid på utvikling og vedlikehold. I tillegg har Prisma sin dokumentasjon vært veldig utfyllende og hjelpsomt gjennom hele utviklingstiden.

HostHatch

Vi valgte å benytte HostHatch som leverandør av hostingtjenester. Vi leier en server som befinner seg i Oslo, hvor vi hoster en WebSocket server, en Next.js nettside og PostgreSQL databasen vår. Vi benytter oss av HostHatch sin plan kalt NVMe 4 GB, som koster \$6.00 per måned. Planen inkluderer følgende ressurser:

2 AMD EPYC kjerner

4 GB DDR4 RAM

20 GB NVMe-lagring

1 TB båndbredde

Overgang fra Serverless til Virtuell Maskin

I starten av prosjektet benyttet vi oss av serverless hosting via Vercel. Selv om dette var en enkel og skalerbar løsning, opplevde vi merkbare forsinkelser på grunn av cold-start tider. Cold start refererer til tiden det tar for en serverless funksjon å starte opp fra inaktiv tilstand. Denne latensen ble merkbar og påvirket brukeropplevelsen negativt. For å forbedre ytelsen og redusere latens, bestemte vi oss for å migrere til en virtuell server hos HostHatch. Etter migreringen observerte vi forbedringer i responstider og generell ytelse. Her er noen av benchmark resultatene som viser forbedringen:

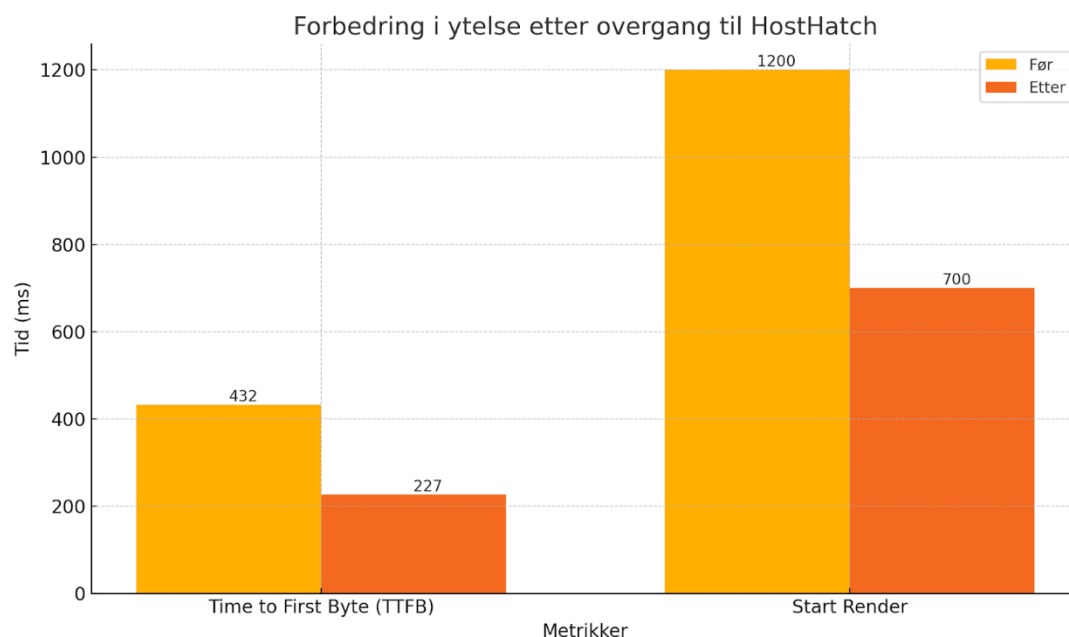


Figure 5 Forbedring i ytelse etter overgang til HostHatch

Systemdokumentasjon

Utplassering

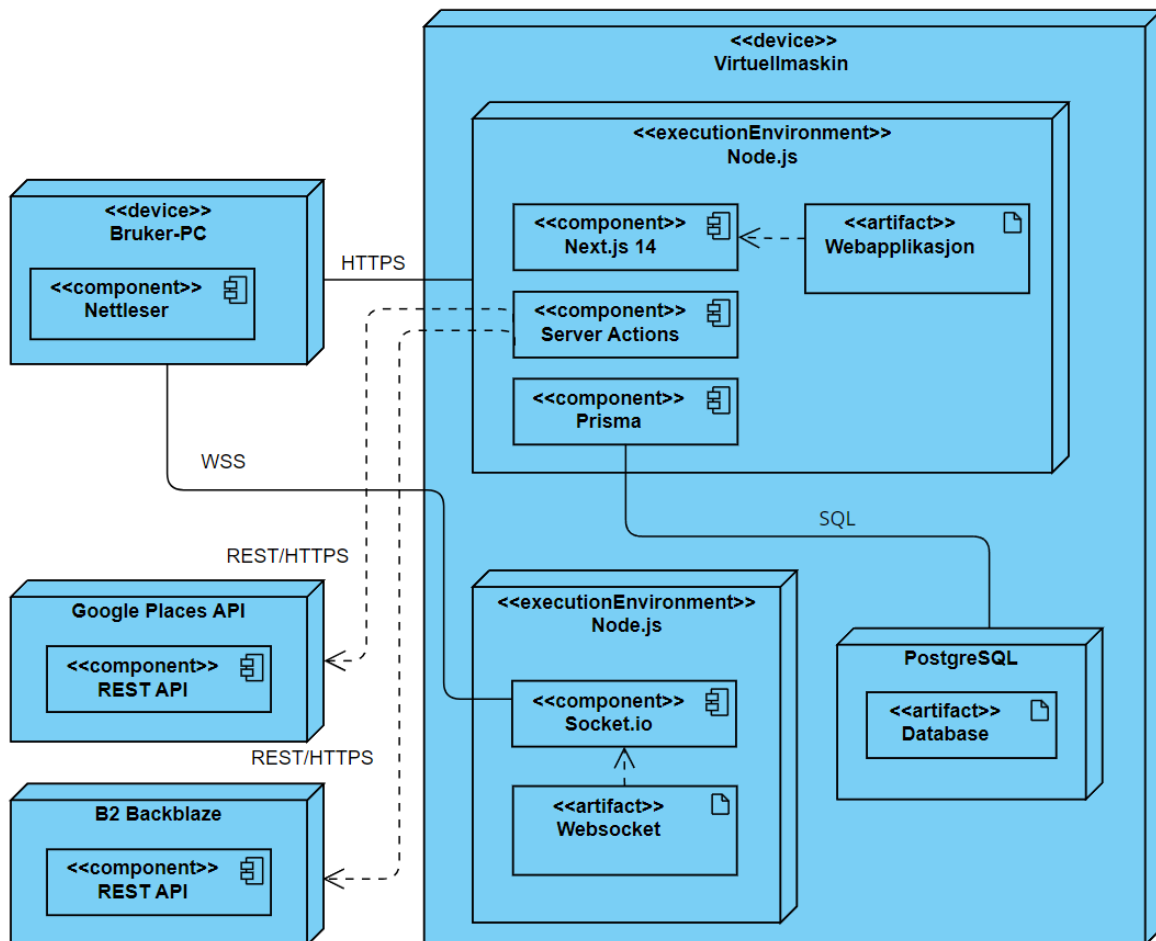


Figure 6 Utplasseringsdiagram


Vi har valgt en monolittisk arkitektur for applikasjonen vår. I denne arkitekturen kjører alle hovedkomponentene: Next.js applikasjonen, WebSocket serveren og PostgreSQL database på en enkelt virtuell maskin plassert i Oslo. Dette gir oss et enklere oppsett og gjør det lettere å teste og distribuere nye funksjoner raskt. Eksterne tjenester som Backblaze for bildelagring og Google Places API for kartdata integreres via API kall. Denne strukturen passer godt til vårt behov og muliggjør effektiv utnyttelse av våre ressurser.

Vi bruker kontinuerlig utplassering ved at hvert GitHub commit utløser et shell script lagret på serveren, som bygger applikasjonen på nytt og deployer den. Dette gjelder også for WebSocket serveren. PM2 muliggjør denne løsningen slik at vi kan ha flere server på en VM samt andre operasjoner parallelt uten å forstyrre eksisterende prosesser.

Søk og filtrering

På vår utforsk side kan brukere søke og filtrere etter spesifikke dugnader basert på sine preferanser. Filtringen kan gjøres via søkeord, kategorier og område*, mens sorteringen kan velges mellom eldste eller nyeste publiseringer. Når brukeren har valgt noen preferanser, vil disse automatisk bli lagt til i URLen. Dette gjør at brukeren kan dele eller navigere frem og tilbake uten å miste den valgte tilstanden.

En typisk URL for utforsk siden med filtrering og sortering kan se slik ut:



```
dugnadnett.no/utforsk?sort=publisert&categories=Fysisk%2CFriluftsliv
```

Figure 7 Eksempel for tilstand i URL

Dette er også den anbefalte måten å lagre tilstand i Next.js 14, som bruker Server Actions. Det kan være utfordrende å kalle en server action når informasjonen er fragmentert fra flere små klient komponenter som ikke er i et direkte parent/child forhold. Selv om dette kan løses med en global state manager, blir dette en bedre løsning.

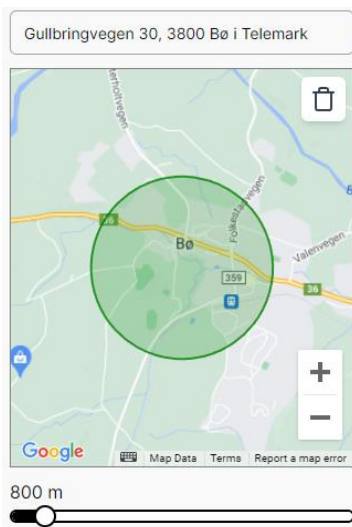


Figure 8 Bilde av område filtering

Filtreringen for områder har blitt delvis implementert. Vi har geotagget lokasjoner lagret i databasen, slik at når en bruker oppretter en dugnad, velger de et område. På utforsk siden kan brukerne søke etter steder ved hjelp av Google Places API og bruke et kart der en justerbar sirkel med en radius på 1,5 km vises rundt det valgte stedet. Dette filteret vil imidlertid ikke bli lagt til i URL-en og vil da ikke påvirke de faktiske filtreringene, ettersom andre prioriteringer tok over mot slutten av prosjektet.

Søkeboksen for søketermer har en 300 ms debounce for å unngå å spamme databasen med forespørsler ved hvert tastetrykk. I stedet venter det 300 ms for å se om det kommer flere tastetrykk. Hvis det ikke gjør det, vil søket utføres.

Identitets og Tilgangskontroll

Autentisering

Når brukere registrerer seg eller logger inn, bruker vi Auth.js for autentisering. En viktig del av dette er håndtering av passord. For å sikre passordene lagrer vi dem ikke direkte i databasen. Vi hasher dem først ved bruk av bcrypt. Bcrypt gjør passordene vanskelig å dekryptere. Under innlogging sammenligner vi det inngåtte passordet med det hashede passordet i databasen for å verifisere brukerens identitet.

For validering av innlogg og registrering bruker vi Zod på både klient og serversiden. Zod lar oss definere regler for hvilken type data vi vil ha og sikrer at disse reglene blir fulgt. Dette gjør at dataene som sendes til serveren er korrekte og i riktig format. Ved å bruke Zod på begge sider kan vi definere valideringsreglene på et sted.

Autorisasjon

Autorisasjon er viktig for å sikre at bare autoriserte brukere kan utføre visse handlinger. Hver dugnad i applikasjonen har en eier som er identifisert med en ownerId som er lagret i dugnad tabellen. Denne ownerId gir eieren rettigheter som å godta eller avslå forespørsler fra andre brukere som vil delta i dugnaden. Eieren kan også redigere informasjon om dugnaden, sette den som ferdig eller slette den fra applikasjonen.

Middleware

I vår applikasjon bruker vi middleware for å sjekke om brukeren er autentisert og autorisert til å se beskyttede sider. Vi definerer ruter som API, Public og Auth, med disse kan middleware sjekke om ulike tilganger basert på brukerens innlogget status. Middleware vil først se om brukeren er logget inn ved å hente ut en autentiserings token fra den nåværende sesjonen. Hvis den ikke eksisterer og brukeren har prøvd å komme til en beskyttet rute, vil brukeren bli omdirigert til innloggingssiden. I samme tråd hvis brukeren er logget inn og har prøvd å komme seg til en Auth route (login/register), vil brukeren bli omdirigert til utforsk siden.

Optimaliseringer

Skeletons

I applikasjonen har vi brukt skeleton til utforsk siden. Dette er en representasjon av hvordan UI ser ut uten noen data. I tillegg har vi en liten rullende effekt på det når det laster inn. Dette brukes blant annet av YouTube og mange content heavy plattformer som har mange bilder. Skeletons gir en bedre brukeropplevelse ved å vise en midlertidig layout mens innholdet lastes inn. Dette reduserer følelsen av ventetid for brukerne.

Progressiv bilder

Progressiv bilder brukes i prosjektet ved at vi på utforsk siden laster inn bilder som er 500px brede. Når man klikker på en av disse kortene, vil man komme inn til en dugnad hvor 500px bilde fra utforsk blir lastet inn med en gang fra cachen. I bakgrunnen vil det samtidig hentes et 900px bilde som byttes ut når det er ferdig lastet inn. Dette gir en rask opprinnelig visning av bildet, som deretter erstattes med en høyoppløselig versjon og brukes blant annet av Facebook.

Bildegomprimering

Vi har implementert bildegomprimering ved at bildene blir konvertert til WebP format ved opplasting ved hjelp av Sharp. Vi lagrer også bildene i to størrelser 500px og 900px for bruk i progressive bilder. Dette reduserer filstørrelsen betraktelig uten at det går ut over bildekvaliteten. Dette utgjør en stor forskjell i lastetid og er standard på nettsider som man laster opp bilder på.

Dugnader

Dugnader i applikasjonen vår vil ha informasjon som inkluderer tittel, bilde, kategorier, dato, beskrivelse, deltakerliste, kart over området og gruppechat. Hver dugnad har en unik dugnadId som bestemmer hvilken data av dugnad som vises. Dette håndteres via dynamiske ruter.

Deltakelse

Brukere kan be om å delta i en dugnad ved å klikke "Be om å bli med" knappen, dette vil sende en forespørsel videre til den som organiserer dugnaden. Eier av dugnaden

kan deretter enten godkjenne eller avslå forespørselen. Når en bruker blir godkjent blir de lagt til i deltakerlisten og får tilgang til gruppechatten. Der kan de snakke med andre deltakere i sanntid.

Hvis en bruker skulle ha blitt avslått vil det ikke være mulig for dem å kunne sende en ny forespørsel. Knappen sin tekst bytter til "Ikke godkjent" og blir deaktivert. Dette sørger for at brukeren ikke kan sende gjentatte forespørsler om deltakelse.

Gruppechat



Figure 9 Gruppechat

Gruppechat er implementert med Socket.io for sanntidskommunikasjon. Når nye meldinger sendes, vil chatten automatisk scrolle ned for å vise de nyeste meldingene, med mindre brukeren har scrollet opp for langt. I så fall vil en klikkbar tekst boble vises, som informerer om at det er nye meldinger nedenfor. Hvis brukeren scroller opp til toppen av chatten, vil det lastes inn opptil 25 tidligere meldinger, og brukeren vil forbli i sin nåværende posisjon i chatten. Etter at en dugnad er fullført blir meldingene arkivert og kan aksesseres fra Fullførte dugnader på Min side.

Opprettelse

Hver enkelt bruker kan lage og organisere en dugnad. Under ruten /dugnad/opprett på nettsiden blir man presentert med et skjema hvor man skal fylle ut tittel, beskrivelse, dato, bilde, område og kategorier. I likhet med login/register har dette også validering med Zod. Bildet er det eneste feltet som ikke er påkrevd for å opprette dugnaden. Hvis bildet ikke velges, vil et repeterbart CSS mønster fra Hero Patterns dukke opp i stedet.

Som nevnt i en tidligere seksjon oppretter vi to bilder i ulik størrelse for optimaliseringer. Deretter blir bildene lastet opp til Backblaze sin server i Amsterdam. Backblaze tilbyr 10 GB gratis lagring og veldig rimelige priser for ekstra lagring, derfor valgte vi denne løsningen i stedet for en mer standard S3 bønne fra AWS. Til slutt når bildene har blitt vellykket lastet opp lagrer vi bildeurl i databasen.

Innstillinger

I innstillinger har vi profilinnstillinger og brukerinnstillinger som lar brukerne tilpasse ulike parametere. Under profilinnstillinger kan man bytte eller sette et profilbilde. Det er enkelt å laste opp et nytt bilde for å oppdatere profilen. Brukerne kan også skrive en bio. Denne bioen vises på profilen og gir en kort beskrivelse av brukeren. Under det er et sted hvor man kan legge til ferdigheter. Ferdighetene gir andre brukere en ide om hva man er god på.

Ved brukerinnstillingene kan man endre e-postadresse og passord. Endring av passord kan gjøres ved å først skrive inn det nåværende passordet og deretter det nye passordet man ønsker. Dette er en veldig standard måte å gjøre det på og oppstår i de fleste applikasjoner med brukerkontoer. Siden det er flere måter å få tilgang til en konto uten å ha logget seg inn ved f.eks session kapring eller på en felles pc hvor man glemmer å logge seg av.

Profiler

Hver profil kan sees av hvem som helst innlogget bruker. På en profil ser man profilbildet og når kontoen ble opprettet. Det er også en liste av følgere som viser hvem som følger profilen. Hvis du allerede følger en bruker ser du en slutt å følge knapp, ellers ser du en følg knapp. Det finnes også en send melding knapp, men funksjonaliteten bak den er ikke implementert.

Bio og ferdigheter vises også på profilen, som nevnt før. Bioen gir en kort beskrivelse av brukeren og ferdighetene viser hva brukeren er god på. URLen er basert på en UUID som er unik for hver bruker. Dette betyr at hver profil har en unik adresse som er enkel å dele.

Konklusjon

Måloppnåelse

Vi har jobbet med dette prosjektet en stund nå og har kommet i mål med mange av kravene. Her er en oppsummering av hva vi har fått til.

Når det gjelder de funksjonelle kravene har vi oppnådd flere av dem. Brukerne kan først av alt se de tilgjengelige dugnadene og melde seg på dem. Dette var viktig fordi det gir oversikt og gjør det lettere å delta. Brukerne kan også opprette egne dugnader for å få hjelp med arbeid. Vi har også klart å implementere en filtreringsfunksjon som lar brukerne spesifisere type og delvis lokasjon for dugnadene de vil se på.

Kommunikasjonen mellom deltagere og ledere fungerer nå slik at de kan organisere arbeidet og logistikken. Dessuten kan eieren av dugnaden fjerne deltagere som ikke oppfører seg ordentlig.

Men, ikke alt har gått som planlagt. Vi har ikke fått til funksjonen hvor brukerne kan gi attest til deltagere som deltok på dugnadene. Denne mangler fortsatt. Direktemelding funksjonen ble heller ikke implementert, så brukerne kan ikke sende meldinger til hverandre direkte, men kan i gruppechat. Det var også meningen at brukerne skulle kunne legge til andre som venner, men vi byttet strategi og gikk for følgere i stedet, som ble implementert. Kravet om å følge med på hva vennene driver med ble ikke gjennomført.

Når det gjelder de ikke funksjonelle kravene, har vi hatt bedre suksess. Applikasjonen er brukervennlig og følger de nødvendige webstandardene. Ytelsen er god og det gir en effektiv brukeropplevelse. Vi har også sørget for at GDPR kravene er for det meste oppfylt og at applikasjonen overholder personvernlovgivningen.

Alt i alt har vi nådd mange av målene våre selv om vi ikke har klart alt. Flere viktige funksjoner er på plass og applikasjonen fungerer godt innenfor de tekniske kravene vi satte oss.

Hva som kunne ha blitt bedre

Her skal vi peke ut deler av prosjektet der vi kunne ha gjort en bedre jobb. Prosjektet har gitt oss mye nye erfaringer. Vi har lært mye nye teknologier og lært mer om utvikling, deployment og vedlikehold av en nettplattform. Videre har vi også lært mer om hvordan en kan jobbe sammen i grupper. Derfor viser vi her til tre punkter der vi mener selv som gruppe at vi kunne ha gjort en bedre jobb og vi vil ta med oss disse erfaringene videre fremover.

Websockets

Vi kunne ha forbedret websockets delen ved å implementere den fra starten av, i stedet for å legge den til senere på en begrenset måte. For øyeblikket oppretter vi en ny forbindelse hver gang man går inn i en gruppechat, og avslutter den når man går ut. Dette innebærer å stadig opprette og avslutte forbindelser for å joine kanaler.

Vi burde ha integrert en overordnet websocket for hele applikasjonen. Dette ville tillatt oss å inkludere flere kanaler for private meldinger, gruppechatter, notifikasjoner, og sanntid oppdateringer på forespørsler for dugnader. En slik løsning ville gjort systemet mer fleksibelt og redusert behovet for unødvendige tilkoblinger.

Organisering av arbeid

Gruppen ble tidlig enige i hvordan prosjektet skulle gjennomføres. Gruppen kom inn i prosjektet med forskjellige erfaringer og kjennskap til fagstoffet. Derfor måtte gruppen ta hensyn til dette og gi ut roller basert på dette. Videre var det også viktig at enhver gruppemedlem skulle lære mye under arbeidet og derfor ble det satt opp muligheter for hvert gruppemedlem å prøve ut deler av arbeidet som ikke tilhørte dens rolle. Dette ga

gruppen en god fleksibilitet og åpnet opp for at gruppemedlemmer kunne samarbeide når det passet og at en alltid kunne spørre om hjelp.

Denne organisering fungerte godt i starten når det var mye som skulle gjøres. Etterhvert som prosjektet fortsatte, oppsto det flere problemer.

1. Det første var at det ikke alltid var like lett å få tak i andre gruppemedlemmer. Som nevnt tidligere jobbet hvert gruppemedlem mye for seg selv. Grunnen til dette var at gruppemedlemmene hadde forskjellige rutiner og dagsplaner. Derfor brukte de bare discord for å kommunisere med hverandre. Det var ofte vanskelig å få tak i andre gruppemedlemmer når en jobbet med prosjektet siden en visste ikke når de andre jobbet og om de kunne hjelpe deg.
2. Det andre problemet var arbeidsdelingen. Hvert gruppemedlem fikk velge oppgaver selv så lenge det hang sammen med den delen av applikasjonen de jobbet med. Dette ble gjort for å begrense antall ganger en måtte vente på en annens arbeid for å gå videre med sitt eget. Dette fungerte fint så lenge hvert gruppemedlem klarte å jobbe seg gradvis fremover. Problemet var når en eller flere satte seg fast i utviklingsprosessen. Dette bremset ofte utviklingsprosessen og i kombinasjon med punkt en førte det ofte til flere dagers pause før andre medlemmer ble klar over det og kunne hjelpe til.

Planlegging

Planlegging av prosjekt var en kontinuerlig prosess. Mesteparten av grunnstrukturen ble bestemt helt i starten av prosjektet. Milepæler ble satt opp som mål som skulle jobbes mot og sprinter ble fullført mellom dem. Ut ifra dette rammeverket ble planer lagt i henhold til hvor i prosjektet gruppen var og med hensyn til andre faktorer eksempelvis personlige hendelser eller andre fag. Denne løse strukturen ga gruppen god fleksibilitet og tillot tilpassing til av når arbeidet ble gjennomført.

Planlegging fungerte for det meste godt i løpet av prosjektet. Likevel oppsto det problemer underveis. Alle disse problemene oppsto som grunnet eksterne hendelser og mangel på rutiner for dette. Den løse strukturen ga gruppemedlemmene mye kontroll over eget arbeid. Problemet med dette var når en eller flere gruppemedlemmer ikke kunne nå sine deadliner. Her måtte da andre gruppemedlemmer ta opp arbeidet til den andre. Dette var ofte problematisk siden hver av gruppemedlemmene ofte jobbet med sin egen del av prosjektet, noe som førte til manglende kjennskap hvis en måtte jobbe i en annen del.

Hva som mangler

I denne delen vil vi diskutere de elementene som vi dessverre ikke fikk tid til å implementere i prosjektet. Til tross for vår innsats og engasjement, var det flere funksjoner og forbedringer som måtte utsettes på grunn av tidsbegrensninger. Vi vil her presentere en oversikt over disse komponentene og forklare hvorfor de ikke kom med applikasjonen.

Threads

Underveis i utviklingsprosessen ble gruppemedlemmene enige om at dugnadsiden burde inneholde flere komponenter. Den første av disse var en chat funksjon som alle dugnadsmedlemmer kunne bruke til å kommunisere med hverandre om hva som helst. Denne ble deretter tatt med i prosjektet og lagt til applikasjonen. Deretter ble det også pekt ut at man kanskje burde kunne ha mer fokuserte samtaler om mer spesifikke ting, for eksempel transport til og fra dugnad. Dette ledet til ideen om å implementere en funksjon for samtaletråder (threads).

Threads-funksjonaliteten ble utviklet av teamet i slutten av utviklingsperioden. Alt fra UI-komponentene til server actions ble fullført, og funksjonen passerte enkle tester. Problemet var tidsbegrensningene som oppsto i midten av april. Alle gruppemedlemmene var enige om at de måtte prioritere andre deler av prosjektet som ikke var fullført. Videre var det også viktig at noen av gruppemedlemmene begynte å

skrive rapporten. På grunn av disse prioriteringene, ble den endelige integrasjonen og omfattende testing av threads-funksjonaliteten utsatt.

Admin funksjoner

Det ble ganske tidlig oppdaget at dugnadnett måtte ha visse roller for som styrte innholdet på applikasjonen. Siden dugnadnett skulle la brukeren starte og delta i dugnader i tillegg til at de skulle kunne kommunisere med hverandre, måtte det eksistere personer eller programvare som kunne moderere dette. Gruppemedlemmene ble kjapt enige i starten av prosessen at den letteste løsningen å implementere var forskjellige admin roller. Disse ville eksistere på forskjellige nivåer som ga dem større kontroll over plattformen. Tenkte roller var:

1. Rootrollen ville kun utviklerne ha tilgang til. Den ville gi tilgang til alle admin funksjonaliteter Alt fra slette meldinger, threads og kontoer til tildeling av admin funksjonalitet til fjerning av den fra kontoer.
2. Adminrollen ville blitt delt ut til kjente brukere av applikasjonen som var aktive, De ville gjøre mesteparten av selve moderasjonen. Eneste forskjellen mellom admin og root er at root ville kunne fjerne admin funksjoner fra andre kontoer, noe som admin rollen ikke tillot.
3. Den siste rollen tilhørte dugnadseieren. Han skulle ha muligheten til å moderere sinne alt innad i sine egne dugnader.

Grunnen til at denne funksjonaliteten ikke ble utviklet var tid. Etterhvert som utviklingsperioden strakk seg ut, var gruppemedlemmene opptatt med andre deler av prosjektet. Da det nærmet seg slutten av utviklingsprosessen ble det tatt en avgjørelse om at andre deler av prosjektet måtte prioriteres. Derfor ble det aldri gjort noe arbeid på denne delen av applikasjonen.

Anmeldelser

Dugnaden ble utviklet fordi gruppen syntes organisering over nettet med sosiale medlemmer manglet mye. En av de mest sentrale kritikkene som ble tatt opp var at det

ikke eksisterte et godt system som pekte ut hvor gode organiseringene var. Videre var påliteligheten både til organisasjonen og deltakerne viktige. Det ble derfor avgjort at dugnaden burde ha et system som ga et godt innblikk i begge to samtidig som det ga gode incentiver for alle medlemmer av dugnadnett.

Tenkte løsningen for dette var at det skulle eksistere to forskjellige typer for rangeringer i dugnaden. Den første ville påvirke eieren av dugnaden og ville basere seg på hva deltakerne tenkte om dugnaden i etterkant av dens fullføring. Den andre ville være en rangering som så på alle brukere og baserte seg på hvor mange dugnader de hadde fullført i forhold til hvor mange de hadde meldt seg på. Den siste delen ville bli bekreftet av dugnads eieren.

Denne delen av prosjektet ble heller ikke utviklet. Det ble som nevnt tidligere i de andre delene som manglet, gjort et valg av gruppen at de måtte prioritere andre deler av prosjektet.

Erfaringer og utbytte

Erfaringer

Gruppen har sammen lært mye i løpet av dette prosjektet. Alle gruppemedlemmene kom inn i prosjektet med varierende ferdigheter og bidro til at prosjektet ble fullført. Det var mye som måtte læres, både fra selve arbeidet og fra hverandre. Derfor deler vi opp erfaringene opp i to deler. Den første delen er de tekniske erfaringene som blir tatt med videre. Den andre er samarbeid og kommunikasjon erfaringer.

Tekniske erfaringer

Det var mye nytt som måtte læres i dette prosjektet. Gruppen valgte som nevnt tidligere å ta i bruk nye teknologier og løsninger. I tillegg var det også varierende erfaring knyttet til programmering og prosjekter knyttet til det. Derfor lærte alle gruppemedlemmer mye nytt samtidig som at det varierte litt hvordan det ble gjort.

Next JS var en spennende ny teknologi for gruppen. Flertallet av gruppen hadde allerede en del erfaring med React, noe som førte til at det så spennende ut på forhånd. Det var noen problemer i starten, men med flere feil plukket gruppen det opp. Alt i alt ville gruppen si seg fornøyd med Next JS og vil ta med seg denne teknologien videre fremover.

Prisma var også et nytt for flertallet av gruppen. Det var greit at en av gruppemedlemmene allerede hadde erfaring med bruk av det. Etter en litt treg start gikk det veldig greit. Med Prisma kommuniserte vi med en PostgreSQL database. De fleste i gruppen hadde brukt det før og syntes derfor at prisma så ut som en god løsning for kommunisering med databasen og applikasjonen vår.

Samarbeid og kommunikasjon

Gruppen fungerte godt i dette prosjektet. Det var flere grunner til dette. Først og fremst ble gruppen tidlig enig om en hva slags gruppe dynamikk som passet for gruppen. Den

løse strukturen med fokus på egne oppgaver imot deadliner passet gruppen godt. Videre var også hele gruppen motivert til å legge inn et arbeid. Det var noen problemer med denne strukturen. Som nevnt tidligere oppsto det problemer når denne strukturen møtte opp mot unntakssituasjoner eller eksterne faktorer. Gruppen tar derfor med seg at en må ha bedre rutiner for slike hendelser. Disse rutinene burde eksempelvis peke ut sekundære ansvarlige personer som kan plukke opp arbeidet til en annen raskt og fullføre den. Videre ble gruppen enig om at den harde skillen mellom ansvarsområder ikke var så lurt. Den begrenset gruppens medlemmers evne til å jobbe effektivt og raskt i det området de hadde jobbet med selv. Derfor ville det vært lurere å mikse oppgavene mer og heller ha flere møter i starten av prosjektet.

Det var også mye vi lærte om kommunikasjon. Dette skjedde for det meste i de to ukentlige møtene gruppen hadde. Alle medlemmene deltok når de kunne og var flinke til å informere om hvor de var, hvor lang tid det ville ta å komme seg videre og hva de trengte fra resten av gruppen.

Utbytte

Det er mye gruppen tar med seg fra dette prosjektet. Alt fra bredere tekniske ferdigheter innad i webutvikling til bedre organisasjonsforståelse og bedre Kommunikationsferdigheter.

Link til kildekode

<https://github.com/Zekima/dugnadnett>

Referanser

Code With Antonio (2023, 31. desember) Next Auth V5 - Advanced Guide [Video]. Youtube. <https://www.youtube.com/watch?v=1MTyCvS05V4>

PostgreSQL Global Development Group. (u.å.). About PostgreSQL. Hentet 18. mai 2024, fra <https://www.postgresql.org/about/>

Prisma Data, Inc. (u.å.). Why Prisma ORM?. Hentet 18. mai 2024, fra <https://www.prisma.io/docs/orm/overview/introduction/why-prisma>

Rehkopf, M. (u.å.). Kanban vs. scrum: which agile are you?. Hentet 18. mai 2024 fra <https://www.atlassian.com/agile/kanban/kanban-vs-scrum>

Trello. (u.å.). Learn Trello board basics. Hentet 18. mai 2024 fra <https://trello.com/guide/trello-101>

Wrike. (u.å.). What Is Kanban? The Ultimate Guide to Kanban Methodology. Hentet 18. mai 2024 fra <https://www.wrike.com/kanban-guide/what-is-kanban/>

Setter, M. (2020, 4. juni). What Is User Acceptance Testing (UAT): Meaning, Definition. <https://usersnap.com/blog/user-acceptance-testing-right/>

Omniconvert. (2024, 12. mai). What is usertesting?. <https://www.omniconvert.com/what-is/user-testing/>

OpenJS Foundation. (u.å.). Introduction to Node.js. Hentet 18. mai 2024 fra <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>

Vercel, Inc (u.å.). Documentation. Hentet 18. mai 2024 fra <https://nextjs.org/docs>