Team 5 Scrumprosjekt

Sluttrapport

“Gidd”

Versjon <1.0>

Eivind Berger Nilsen, Ellen Mikkelsen, Ilona Podliashanyk, Karl Labrador, Lars-Håvard Holter Bråten, Robin Vold, Sergio Martinez, Sigmund Ole Granaas

Revisjonshistorie

Dato	Versjon	Beskrivelse av endring	Forfatter
15/04/21	0.1	Innledende arbeid	Lars-Håvard Bråten
22/04/21	0.2	Kapittel 7.1.1	Lars-Håvard Bråten, Robin Vold, Sergio Martinez
26/04/21	0.3	Videre arbeid	Eivind Berger Nilsen, Ellen Mikkelsen, Ilona Podliashanyk, Karl Labrador, Lars-Håvard Holter Bråten, Robin Vold, Sergio Martinez, Sigmund Ole Granaas
30/04/21	1.0	Ferdigstillelse av rapporten	Eivind Berger Nilsen, Ellen Mikkelsen, Ilona Podliashanyk, Karl Labrador, Lars-Håvard Holter Bråten, Robin Vold, Sergio Martinez, Sigmund Ole Granaas

1. Forord

Denne rapporten er skrevet i sammenheng med et gjennomført gruppeprosjekt i emnet IDATT2106 Systemutvikling 2 ved NTNU i vårsemesteret 2021.

Rammen for prosjektet ble satt av faglærer og produkteier gjennom et visjonsdokument, der målet for gruppen er å utvikle en webapplikasjon med navnet “Gidd” ved å ta i bruk Scrum-rammeverket for å optimalisere produktutviklingen.

I rapporten skriver vi om prosessen for gruppearbeidet, teori bak valgt teknologi og Scrum-rammeverket og dokumentasjon for systemet vi har utviklet.

Gruppen ønsker å takke vår produkteier Håkon Lien for godt samarbeid og høy tilgjengelighet til å svare på spørsmålene vi måtte ha. I tillegg ønsker vi å takke scrum-master og faglærer Grethe Sandstrak for god veiledning på standup-møtene.

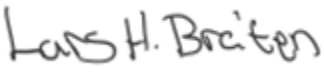
30.04.2021, Trondheim

Eivind Berger-Nilsen

Ellen Mikkelsen

Ilona Podliashanyk

Karl Labrador



Lars-Håvard Holter Bråten

Robin Vold

Sergio Martinez

Sigmund Ole Granaas

2. Oppgavetekst

Prosjektoppgaven var å utvikle webapplikasjonen “Gidd!” Oppgaven ble gitt i form av et visjonsdokument etter en felles introduksjon. Hensikten med applikasjonen var å legge til rette for at mennesker kunne møtes og opprettholde fysisk aktivitet ved å lage aktiviteter andre brukere av Gidd! kunne delta på. Webapplikasjonen skulle være tilgjengelig gjennom de vanligste nettleserne både på PC og mobil.

Applikasjonen skulle la brukere opprette profiler, lage aktiviteter, og melde seg på og av disse. Opprettelse av aktivitetene skulle inneholde en beskrivelse, maks antall deltakere, tid og sted for aktiviteten. Brukere skulle ha mulighet til å endre sitt eget passord, og de skulle ha et treningsnivå tilknyttet brukeren sin. Fulle aktiviteter skulle en venteliste, og oppretteren av en aktivitet skulle kunne velge nødvendig utstyr deltakerne skulle ha med.

Videre var det også ønskelig å kunne se aktiviteten på et kart samt været i nærområdet da aktiviteten pågikk. Aktiviteter skulle kunne filtreres og sorteres på for å hjelpe brukeren med å finne relevante aktiviteter. Oppretteren av aktiviteten skulle kunne avlyse og redigere på den i ettertid.

Brukere av systemet skulle kunne gi tilbakemeldinger på nettsiden. Plattformen burde også inneholde motiverende elementer, som for eksempel poengsystem, for å stimulere til økt aktivitet. Det var også ønskelig at brukere kunne bli markert til “trusted” da de etterhvert ble regnet som en god og trygg bidragsyter til plattformen. Brukere burde også ha muligheten til å logge inn gjennom kjente sosiale plattformer som Facebook og Google. Videre burde applikasjonen også kunne generere aktiviteter basert på brukeres tidligere historikk.

Produktet hadde også flere ikke-funksjonelle krav. Dette inkluderte en testdekning på minst 50% på tjenersiden. Produktet skulle være i samsvar med WCAG prinsipp 1, og ha god brukskvalitet. Videre skulle den være sikker, og minimum dekke OWASP punkt 1 og 3.

Oppgaven har endret seg ved at prioriteten på brukernes behov som skulle dekkes har endret seg i løpet av prosjektet, etter samtaler med produkteier der tema var sprint review og diskusjon angående prioriteringer i andre sprint, basert på der vi var i utviklingen. Endringene er i følgende tabell nedenfor.

Behov	**Opprinnelig prioritet**	**Ny prioritet fra 2. sprint**
Vær og kartløsning	Lav	Høy
Spillifisering	Lav	Middels

3. Sammendrag

Denne rapporten gir et overblikk over produktet “Gidd!”som ble utviklet i perioden 12.04.2021-30.04.2021 av team 5 i faget Systemutvikling 2. Produktet er en applikasjon hvor brukere kan registrere aktiviteter og delta på disse. Gitt visjonsdokumentet finnes det allerede noen lignende løsninger, men de har andre fokusområder. Løsningen som ligner mest på vårt produkt er produktet “Spond”. Denne er derimot mer rettet mot lag og klubber, og ikke enkeltpersoner. Hovedmålet med teamets produkt er at det skal være lett for brukere å komme i fysisk aktivitet og møte nye mennesker, spesielt i en tid hvor tilbudet på arrangerte aktiviteter er begrenset.

Applikasjonen er lett og intuitiv å bruke, og gir forbrukere en enkel måte å komme i aktivitet på. Ingen installasjon er nødvendig, og det er enkelt og trygt å opprette en bruker på applikasjonen. Applikasjonen har som mål å komme med et tilbud hvor nordmenn, derav spesielt studenter kan møte nye mennesker, samtidig som man holder seg aktive. Derfor er applikasjonen utviklet for å forenkle opprettingen av lavterskels aktiviteter. Den implementerer også et poengsystem som skal motivere brukere til å delta på flere aktiviteter, samtidig som interessen for applikasjonen holdes oppe.

4. Innholdsfortegnelse

1. Forord **2**

2. Oppgavetekst **3**

3. Sammendrag **3**

4. Innholdsfortegnelse **5**

5. Introduksjon **6**

5.1 Begrepsavklaringer 6

6. Teori **7**

7. Valg av teknologi og metode **11**

7.1 Gjennomføring av scrum 15

7.1.1 Sprint 1 16

7.1.2 Sprint 2 17

8. Diskusjon og konklusjon **18**

5. Introduksjon

I denne rapporten dokumenteres og analyseres team 5 sitt forslag på løsning av et behov som spesielt har dukket opp i forbindelse med pandemien verden nå befinner seg i. Det siste årets restriksjoner har ført til at mange føler seg ensomme eller alene. Det har også blitt vanskelig for de fleste å komme i kontakt med nye folk eller å finne nye hobbyer. Produktet som blir beskrevet i dette dokumentet forsøker å løse dette problemet ved å tilby en arena hvor en bruker enkelt kan møte folk og samtidig få inn en treningsøkt eller dyrke sin hobby.

For å utvikle dette produktet har teamet tatt i bruk en agil utviklingsprosess. Teamet har også måttet ta i bruk designprinsipper som UX og Universal Design for å gjøre produktet brukervennlig. Fra et teknologisk ståsted har teamet brukt React og Java Spring for å produsere dette produktet. Videre har mange biblioteker blitt brukt for å implementere mer engasjerende funksjonalitet. Teorien bak teamets prosess og designprinsipper er beskrevet i kapittel 6 av denne rapporten. Kapittel 7 fortsetter med å beskrive teknologien som er brukt, og kobler dette mot teoridelen. Til slutt blir sluttresultatet drøftet i kapittel 8.

5.1 Begrepsavklaringer

API	Et programmeringsgrensesnitt som åpner for å aktivere spesifikke deler av et program. I webapplikasjonen har vi et REST API som overfører tilstanden til en etterspurt ressurs.
Burndown Chart	En grafisk representasjon som viser hvor raskt teamet arbeider seg gjennom oppgavene i en sprint
GitLab Runner	En applikasjon for for GitLab CI/CD for å kjøre bygge- og testjobber.
Dependency Injection	Et designmønster der et objekt mottar et annet objekt som den er avhengig av for å utføre sin oppgave
HTTP	Overføringsprotokoll som benyttes for å utveksle informasjon over internettet
Endepunkt	Inngangspunktet til en tjeneste
Spørring	En spørring returnerer et resultatsett ved oppslag mot tabeller i en SQL-spørring
Asynkront	Definisjon: “ikke samtidig”
Action	En redux action representerer en handling som skal gjennomføres. f.eks henting av aktiviteter
Reducer	En redux reducer tar imot data fra actions og gjør dataene tilgjengelig for resten av applikasjonen
Service	En Frontend service har ansvar for kommunikasjon med API-et
Dispatch	En dispatch er en melding fra en redux action til en redux reducer

6. Teori

Agil utvikling
Agil utvikling er en iterativ utviklingsmetodikk som setter fokus på kontinuerlig levering av produkt, læring og innovasjon, samt fleksibilitet til endringer. Utviklingsmetodikken er et alternativ til den tradisjonelle sekvensielle utviklingsmodellen. Sekvensiell utvikling legger stor vekt på grundig planlegging ved starten av et prosjekt, og gjennomføring av prosjektet basert på denne planleggingen. Gjennomføringen deles opp i tydelige faser hvor ulike utviklere og team står for hver sine deler av prosjektet. Kommunikasjon mellom de ulike partene skjer hovedsakelig gjennom grundig dokumentasjon.

Den agile utviklingsmetodikken løse mange av problemene med den sekvensielle modellen. En av ulempene med sekvensiell utvikling er at det ikke legger opp til håndtering av endringer som kan skje underveis i gjennomføringen. Den iterative strukturen i agil utvikling gjør prosjektgjennomføringen fleksibel for disse endringene. Her blir krav til hva som må være implementert innen en kort periode satt, istedenfor krav til full gjennomføring av prosjektet. Dette åpner for at endringer kan bli implementert underveis i prosessen etterhvert som nye idéer oppstår, eller nyttige erfaringer blir gjort.

Agil utvikling legger også vekt på å utvikle et fungerende produkt med hyppige utgivelser, istedenfor å fokusere på grundig planlegging og dokumentasjon. Et problem som oppstår når et prosjekt avhenger av dokumentasjon for å kommunisere mellom parter, er at misforståelser kan oppstå. I tillegg er det lett for at overflødig dokumentasjon blir opprettet som i senere tid viser seg å ikke bli lest. Istedenfor å ha dedikerte team til ulike deler av prosjektet, benytter agil utvikling seg av kryssfunksjonelle team. Altså, består teamet som gjennomfører et prosjekt av all kompetanse som er nødvendig for gjennomføringen. I tillegg til at medlemmer da kommuniserer direkte, åpner organiseringen opp for læring da ikke én gruppe eller ett medlem ender opp med ansvar for all arbeid innenfor ett område.

Scrum
Scrum er et rammeverk brukt i agil utvikling. Gjennomføringen av utviklingsmetodikken tar utgangspunkt i prinsippet “Inspect and adapt”. Prinsippet fremhever læring da det setter lys på vurdering av arbeid som er gjort underveis i prosjektet og hvilke endringer som bør gjøres for å forbedre prosessen. Dette gjelder både for funksjonell implementasjon, samt organisering av teamet. I Scrum blir det tatt i bruk tre roller. Én produkteier, én Scrum master, og teamet. Produkteieren har ansvaret for å definere produktets funksjonelle krav, samt å sette opp prioriteringene over disse. Scrum masteren sitt ansvarsområde er å hjelpe produkteier og teamet til å bruker riktig Scrum-metodikk under gjennomføringen av et prosjekt. Scrum masteren fokuserer her på prosessen, og ikke teamets organisering eller arbeid. Teamet står for selve utviklingen av produktet. Teamet er kryssfunksjonell slik at all kompetanse som er nødvendig for gjennomføringen av prosjektet er tilstede i teamet.

Utviklingsmetodikken implementerer den iterative modellen gjennom sprinter. En sprint er en kortere arbeidsperiode som løper over én til fire uker. Hver sprint kjennetegnes med et sett med seremonier som gjennomføres. Disse inkluderer en sprintplanlegging i starten av hver sprint, samt gjennomføring av en sprint review og et retrospektiv på slutten av sprinten. Under sprintplanleggingen settes målet for sprinten, og hvilke funksjonelle krav som skal være implementert. De funksjonelle kravene blir spesifisert i en backlog for sprinten. Sprintens backlog tar utgangspunkt i prosjektets backlog som inneholder de funksjonelle kravene satt av produkteier for produktet, samt prioriteringene deres.

Underveis i sprintene kan ikke sprintens backlog endres. Dersom produkteier ønsker å gjøre endringer eller sette opp andre krav, må dette tas opp under sprintplanleggingen til neste sprint. Denne begrensningen kommer til fordel for både teamet og produkteier. For teamet betyr dette at de kan jobbe mot et konkret og tydelig mål hver sprint uten å måtte tenke på mulige endringer. Dette kommer også til fordel for produkteier som da alltid vil ha oversikt over hvilken funksjonalitet som arbeides med, samt vite at de med sikkerhet vil kunne for å foreslå endringer og nye funksjonelle ønsker ved start av neste sprint.

UX / IxD

Hvordan er system blir å oppfattes av dets brukere er et viktig poeng under utvikling. Fagfeltet som tar for seg brukeropplevelsen og aspektene med interaksjonen mellom bruker og system er UX, eller *User Experience.* Dette innebærer brukerens oppfatning av brukbarhet, effektivitet og enkelhet av systemet. UX kan videre deles inn i flere subset, og en av disse er interaksjonsdesign. Interaksjonsdesign som et fagfelt handler om å designe samhandlingen mellom bruker og systemer. Her er det altså viktig å forstå hvordan interaksjonen mellom bruker og system foregår, for å finne ut hvordan denne kan designes best mulig. Interaksjonsdesign tar utgangspunkt i noen prinsipper for design. Eksempler på disse er synlighet av funksjoner, tilbakemelding mellom system og bruker, og begrensninger av interaksjoner. Disse prinsippene vil legge til grunn for vurdering og sammenligning av ulike design.

Universell utforming

Universell utforming er fagfeltet som omhandler design av produkter og elementer slik at de er tilgjengelige for alle. Dette innebærer at et system skal legge til grunn for at brukerne i dere mottakergruppe har ulike behov, slik at produktet blir tilgjengelig for enhver bruker. Center for Universal Design ved North Carolina Statue University har definert syv prinsipper
ersal Design ved North Carolina State University definert syv prinsipper et system må ta utgangspunkt i for oppnå universell utforming. Prinsippene tar blant annet for seg at et system må være intuitivt å bruke, og tilby like muligheter for alle brukere uavhengig av deres ferdigheter. I tillegg skal et system tilby forståelig og godt kommunisert informasjon, som også tilbys i flere format.

For å oppnå universell utforming må konkrete tiltak gjøres underveis i en utviklingsprosess. Her gjelder det å aktivt integrere prinsippene bak universell utforming under planleggingen av design og struktur på produktet. I tillegg kan enkelt tiltak som å tilby informasjon i flere formater implementeres underveis for å øke den universelle utformingen til produktet. En ringvirkning med å implementere løsninger med høy grad av universell utforming, er at disse får som resultat av implementasjonen høyere brukbarhet. Videre vil også søkemotoroptimaliseringen øke, da crawlerne som bestemmer indeksingen hos søkemotorene kan ansees som brukere med svært begrenset mulighet for å forstå innhold, spesielt video og grafisk innhold.
En annen god grunn for å utvikle god universell utforming er at det er lovpålagt. Blant annet må en nettside oppfylle 35 av 61 kriterier i WCAG standarden.

Sikkerhet
Et viktig punkt i utviklingen av webapplikasjoner, er å gjøre dem sikre. For å oppnå dette må potensielle sårbarheter i applikasjonene identifiseres, og mottiltak for disse implementeres. OWASP er en internasjonal, idealistisk organisasjon. Organisasjonen jobber med å identifisere sårbarheter, og publiserer årlig de ti mest vanlige angrepsmetodene. Noen av disse inkluderer SQL injections, Cross-site-scripting, Cross-site-request-forgery og Sensitive data exposure. SQL injections er en angrepsmetode hvor brukerinput utnyttes til å konstruere spørringer mot databasen som får tak i informasjon brukere egentlig ikke skal ha tilgang til. SQL injections kan oppstå i situasjoner hvor et brukerinput ikke blir validert før det brukes i en spørring. Cross-site-scripting, eller XSS er en annen sårbarhet som er et resultat av ikke-validerte brukerinput. Her prøver en angriper å eksekvere egne scripts i en nettside, ved å utnytte at nettsiden for eksempel viser frem brukerinput på siden. Slike scripts kan brukes til å for eksempel omdirigere en bruker til en ondsinnet side. Cross site request forgery, eller CSRF er et angrep hvor en bruker blir tvunget til å gjennomføre uønskede handlinger på en side de allerede er innlogget på. Her utnytter angriperne at en bruker allerede er autentisert på en side. En siste sårbarhet som kan trekkes frem er Sensitive data exposure. Denne oppstår i situasjoner hvor systemer ikke krypterer data godt nok, slik at data blir tilgjengelig for andre.

Domain driven design
Domenedrevet design er et programmeringsdesign som går ut på å sentralisere logikken til koden som samsvarer med problemet produktet prøver å løse. Et viktig prinsipp er at domenet skal isoleres fra resten av koden som håndterer de tekniske problemstillingene og implementasjonsdetaljene. Et domene er en gruppering av kunnskap og regler. I sammenheng med utviklingsarbeid som tar bruk av Scrum kan domenet bestå av løsningene på problemstillingene til produkteier, aktuelle userstories og deres akseptansekrav. Userstories representerer kjernen i hvordan et produkt skal fungere. User stories beskriver hvordan en bruker kan benytte seg av produktet, hvilke handlinger de kan utføre, samt forventningene en bruker har mens den bruker produktet.

Den største oppgaven til domain driven design er å sentralisere og strukturere koden rundt problemet som skal løses. Dette vil gjøre det lettere å gjøre endringer som har en effekt på

hvordan et produkt fungerer og hvordan de sentrale entitetene samhandler med hverandre. Koden som definerer hvordan produktet fungerer skal aller helst fungere i isolasjon og kunne testes separat fra de tekniske delene av applikasjonen. Dette er essensielt for å kunne verifisere at produktets kjerne fungere slik det skal til enhver tid. Domain driven design prøver å skille «business»-koden og den tekniske implementasjonen som må til for å leverer et digitalt produkt. Dette gjøres for fordi designmønsteret ikke ønsker å være påvirket av hvordan et produkt er implementert, men heller hvordan det fungerer.

GDPR

GDPR er en personvernforordning som ble vedtatt av EU i 2016, og nå er en del av det norske lovverket siden 2018. Forordningen ser på hvordan personvern ivaretas ved innsamling og digital behandling av personvernopplysninger. GDPR reglementet gjelder for alle organisasjoner som behandler opplysninger til europeiske brukere, uansett om organisasjonen selv er en del av EU.

GDPR-reglementet spesifiserer blant annet hvordan data skal samles inn, lagres og prosesseres. En viktig regel er at data kan kun samles inn når en bruker har gitt direkte samtykke til dette. Før dette skal brukeren også opplyses om hvordan dataen blir prosessert. I tillegg er det et krav om at integriteten og konfidensialiteten til personopplysninger bevares. GDPR har også tydelige rammer for hva som utgjør samtykke fra en bruker. Det skal være tydelig for en bruker når de gir samtykke til lagring og prosessering av personlige opplysninger, samt at hvilke personopplysninger det er snakk om skal være informert om på dette tidspunktet. En bruker har også krav om å trekke tilbake samtykket til enhver tid. Dette innebærer at alle lagrede opplysninger blir slettet.

Kontinuerlig integrasjon

Kontinuerlig integrasjon er en praksis for å automatisere testing og bygging av arbeid og kildekode som er gjort av flere utviklere. Bruk av kontinuerlig integrasjon er høyst relevant i kombinasjon med verktøy for versjonskontroll. I praksis tillater kontinuerlig integrasjon flere utviklere å merge sammen kildekode hyppigere. I det en utvikler legger til sitt arbeid bygges systemet opp og tester blir kjørt automatisk. Arbeidet blir da umiddelbart validert og en tilbakemelding blir gitt til utvikleren om hvordan systemet reagerte på endringen. Kontinuerlig integrasjon sikrer altså at små feil blir oppdaget så snart de inntreffer, og kan da rettes opp i snarere.

Brukertester

Brukertester brukes for å finne ut noe om systemet som utvikles eller for å finne forbedringer. De brukes typisk for å finne ut om brukergrensesnittet og informasjonsarkitekturen er godt satt opp. De vil gjerne bli utført gjennom hele utviklingsprosessen, der de for hver iterasjon vil bli mer lik sluttproduktet.

7. Valg av teknologi og metode

Jira og Confluence
Jira er et verktøy for å organisere og planlegge digitalt arbeid, mens Confluence er et dokumentasjonsverktøy. Begge er utviklet av Atlassian. Programmene ble brukt av teamet for å planlegge og holde oversikt over oppgaver som måtte gjøres under sprintene, samt til å lage en Wiki. Jira og Confluence er fullstendig integrert med hverandre. Verktøyene er mer komplisert enn tradisjonelle verktøy, som issue-tracking i GitLab og GitLab Wiki, men tilbyr til gjengjeld mer funksjonalitet teamet har benyttet seg av. Dette inkluderer for eksempel burndown charts og integrerte table of contents med mer.

GitLab
Teamet har brukt GitLab for versjonskontroll av programvare.Teamet valgte å benytte seg av to repositories for å skille arbeid på frontend og backend. For kontinuerlig integrasjon ble det satt opp en egen GitLab-runner, slik at tester kunne kjøres etter teamets behov.

Gitflow
I en agil prosess vil kode måtte integreres i versjonskontroll i et høyt tempo. Dette problemet løste teamet ved å bruke GitFlow. Dette innebærte å gjøre aktiv utvikling på en annen gren enn den som inneholdt produksjonskode. Grenene teamet brukte for å fullføre issues, ble navngitt etter issue-koder fra Jira. Grenene ble da automatisk lenket til issues i Jira. Dette ga teamet oversikt over hvilken kode som hørte til forskjellige user-stories.

Google Forms
Teamet valgte å holde tilbakemeldinger fra teammedlemmer i sprint-retrospektivene anonyme for å gjøre det lettere å gi ærlige tilbakemeldinger. Til dette brukte vi Google Forms.

Scrum poker
For å kartlegge hvor lang tid teamet trodde hver user-story kom til å ta, brukte vi Scrum-poker. Ut i fra dette brukte vi gjennomsnittet av alle svarene som et estimat. Dette hjalp teamet med å prioritere user-stories ut i fra et forhold mellom verdi og tidsbruk.

React
React er et komponentbasert javascript-bibliotek for å lage webapplikasjoner. Teamet valgte å benytte React over Vue fordi noen teammedlemmer hadde erfaring med teknologien. Vue og React har ingen betydelige forskjeller når det gjelder hva du kan produsere, så valget er av liten betydning.

Redux
Redux er et javascript-bibliotek som gjør det lettere å gi komponenter tilgang til global data. Ved bruk sammen med React, gir dette komponentene mulighetene til å reagere dynamisk på

endringer som skjer i applikasjonen. Redux gir et sterkt rammeverk for hvordan data skal flyte i applikasjonen og hvor unntakshåndteringer skal skje.

Pigeon
Pigeon er et lett javascript-bibliotek for kartvisning. Dette bruker OpenStreetMap.

Spring, Spring-boot, Spring-MVC, og Spring security
I dette prosjektet brukte teamet Spring-boot, et rammeverk som bygger på Java Spring for å lettere bygge applikasjoner. Spring er et stort prosjekt i Java som bygger på to prinsipper, inversion of control og dependency injection. Et av fundamentene til Spring er muligheten for å la Spring være ansvarlig for ha kontroll på, og sende dependencies inn i koden. Med dette kunne teamet produsere komponenter som ikke inneholdt spesifikke implementasjoner av systemet. Dette ga også muligheten for å lettere dele systemet inn i lag, for å bedre definere hvilken oppgave forskjellige deler av koden har. Dette går hånd i hånd med valget om å basere applikasjonslogikken rundt Domenedrevet Design. Alt i alt ga dette teamet muligheten til å teste alle lagene av koden separat for å forsikre at hver komponent fungerer slik som planlagt.

Spring MVC
Spring MVC er et underprosjekt fra Spring for å bygge applikasjoner som er basert på model-view-controller arkitekturen. I applikasjonen ble spring MVC brukt for å kontrollere HTTP-laget(MVC laget). Mer spesifikt ble Spring MVC brukt i kontroller-klassene og for å implementere REST api-et.

Spring DATA JPA
Spring data JPA er et verktøy brukt for å lage repositories som skal snakke med datalaget i en Spring applikasjon. Spring data gjør det mulig å abstrahere bort spesifikke kall mot et lagringsmedium til fordel for et metode-språk som er basert på attributtene til modellene brukt i applikasjonen. Dette gjorde at teamet kunne abstrahere bort databaseimplementasjonen og ga muligheten til å la spring DATA JPA selv bygge queries basert på hvilken database som ble brukt. Dette gjorde det enklere å mocke bort databaselaget, og å bytte database etter behov.

Spring security
Spring security er Spring sitt rammeverk for autentisering og sikkerhet. Teamet har brukt spring security for å konfigurere sessions slik at brukere kan autentiseres. Videre har Spring security blitt brukt til å konfigurere tilgang til apiet. Alle endepunkter unntatt registrering og verifisering krever at brukeren som prøver å nå dem har en valid session.

JAVA Criteria
Java criteria er et API som gjør det lettere å gjennomføre spørringer i database. I dette prosjektet er Criteria brukt for å implementere søk filtrering sortering og pagination.

Docker
Docker er et virtualiseringsverktøy som kjører programvare i isolerte containere. I prosjektet har det blitt brukt for å kjøre opp kartdata for norge. Disse dataene brukes for å konvertere stedsnavn til koordinater og motsatt.

React+redux arkitektur
Teamet har valgt å bruke redux for å håndtere dataflyten i applikasjonen. Redux har et helt klart rammeverk for hvordan data skal håndteres. De fleste handlinger som involverer data i applikasjonen skjer asynkront. For at den visuelle delen av applikasjonen ikke skal påvirkes av dette, kan den ikke være direkte knyttet til eventuelle kall som kjøres mot api-et. En redux dataflyt innebærer derfor å sende data i sirkel. Dette fungerer svært godt sammen med react, fordi react er bygd for å reagere dynamisk på endringer i data enhver komponent er knyttet til. Dette vil skje uavhengig av andre komponenter, slik at det kun skjer oppdateringer der det er nødvendig.

Brukeren har kun muligheten for å sende “actions”. Enten vil brukeren gjøre det selv, eller så vil det gjøres når komponenter lastes inn. Dette vil håndteres av “action” metoder. “Actions” er ansvarlige for å sende korrekte dispatches med eller uten nyttelast til en “reducer” og å kommunisere med eventuelle “services”. Implementasjonen av en “service” er uvisst for en action metode. I vårt tilfelle vil services kun kommunisere med eksterne api-er. En action vil sende dispatches til reduceren underveis i operasjonen så global data kan oppdatere brukeren på hvordan en eventuell transaksjon utvikler seg.

Et viktig prinsipp for react-redux er immutabilitet i data, siden de begge bruker shallow equality checking. For at oppdateringsmodellen i React skal fungere effektivt, kan den ikke konstant sjekke hvordan dataen den er bundet til ser ut. Når en dispatch blir sendt til en reducer, vil den sjekke hvilken type det er, og enten returnere samme dataobjekt hvis det ikke trengs noen endringer, eller et nytt dataobjekt med endringer. Dette kan da sjekkes opp mot det originale dataobjektet ved hjelp av shallow equality checking, som er veldig effektivt når dataobjektene blir store. På denne måten trenger ikke systemet å lete etter endringer i data. Brukere vil få en umiddelbar oppdatering straks nye data er globalt tilgjengelig i applikasjonen. I tillegg gjør denne prosessen det enklere å holde styr på hva som gjør endringer i global data, noe som er tryggere.

Den overordnede arkitekturen teamet har fulgt på frontend er en MVP(Model-view-presentrer). I dette følger det at model(redux+services) står for datahåndtering i applikasjonen, presenter(Containers) inneholder logikk for hva som skal vises til brukeren og view(react) styrer hvordan dette skal vises til brukeren.

Trelagsmodellen
Systemet teamet har laget følger trelagsmodellen. En webfrontend(view) et REST-API(controller) og en database(Model). Disse er tre forskjellige logiske og fysiske lag. Frontend er brukerens inngang til applikasjonen, denne kan kun kommunisere med kontrolleren, som inneholder bussiness/domenelogikken for hvordan teamets tjeneste skal fungere. Controlleren vil se hva brukeren ber om og selv gjøre valg for hvilken data som skal hentes fra en eventuell database(eller andre datamedium) og hvordan den skal behandles. Denne dataen vil da bli formatert i henhold til hvilke regler controlleren er programmert til. Til slutt vil dette sendes tilbake til brukerens view, som er ansvarlig for hvordan dataen blir presentert.

Firelagsmodellen i Spring
Api-et er delt inn i fire lag. Dette er Presentasjonslaget (HTTP/REST-controller) Business-laget (services/userstories), persistence (Repo/DAO) og database. Kontrollere er ansvarlig for kommunikasjon gjennom HTTP. De vil bare kalle services for å håndtere kall til forskjellige endepunkter. Serviceklasser er ansvarlig for å velge hvilken userstory som skal kalles for å utføre riktig handling. De er også ansvarlig for å gi hver userstory et objekt den kan bruke for å kommunisere med et datalagringsmedium. Teamet har valgt å basere business-logikken på domenedrevet design. Teamets implementasjon er ikke fullverdig og fokuserer kun på utføring av user stories. User stories i dette laget er direkte knyttet til hvordan user stories ble beskrevet i planleggingsprosessen og skal representere enhver handling brukeren kan gjøre i applikasjonen. I denne modellen, vil ikke user stories ha noen annen tilknytning til andre lag enn service. De deler på domeneobjekter med persistence laget, for å unngå unødig kompleksitet.

Det har vært eksperimenter med en dypere implementasjon av domeneobjekter og separasjon fra andre lag, men koden ble unødvendig kompleks og vanskelig å jobbe med, noe som gikk i stor grad i mot agile prinsipper. Problemdomenet teamet jobber med er ikke stort eller komplekst nok for å gjøre det nødvendig med en slik implementasjon. Denne løsningen prøvde å ta høyde for problemer teamet ikke kom til å støte på i løpet av prosjektet og ble forkastet.

7.1 Gjennomføring av scrum

Daglige standup-møter i regi av faglærer har blitt utført av samtlige teammedlemmer hver dag. De daglige standup-møtene har fungert for å orientere teammedlemmer om hva en selv jobbet med, og hva hvert medlem tenker å jobbe med samme dag. Medlemmer forteller da også hvis noe hindrer dem i å gjøre det planlagte arbeidet.

For å bestemme hvilke arbeidsoppgaver som skulle gjennomføres hver sprint, tok teamet utgangspunkt i en backlog. Backlogen representerer teamets mål for hver sprint. Hver historie i backloggen er et delmål som skal ende opp som funksjonalitet for en bruker. Disse elementene flyttes fra backlogen når den oppfyller kravene vi har definert i hver story. Kravene skal beskrive hva som skal være nødvendig for at funksjonaliteten skal fungere i produksjon. Teamet har fulgt en sterk kobling mellom grener i repositoriet og hvert element i backloggen. Grener vil kun merges tilbake i development-branchen når koden fyller kravene for hva som ansees å være ferdig. På denne måten forsikrer teamet seg om at all kode som merges inn i det sentrale repositoriet alltid skal representere funksjonalitet som er klar for sluttbrukerne. Dette fører til en iterativ prosess der vi holder fokuset på produktet slik vi har planlagt det for hver sprint. Etter hvert sprint ble det holdt et sprint retrospective

For hver sprint gjennomføres en sprint review. Her viser teamet resultatet av sprinten ovenfor prosjekteieren. Det er her produkteier kan komme med innsigelser. Gjennom sprint reviewen vi hadde den 22.04.2021 fikk vi positiv respons fra produkteier.

Etter hver sprint ble det gjennomført en retrospektiv hvor gruppemedlemmene gikk sammen og reflekterte over prosessen og gjennomført arbeid. Positive og negative erfaringer ble tatt opp, og teamet ble enig om hva som skulle videreføres til neste sprint og hva som var nødvendig å endre.

7.1.1 Sprint 1

Sprint 1 ble iverksatt tirsdag 13/04-2021, og startet med definering av et overordnet sprintmål og hva som skulle til for å nå det. Sprintens overordnede mål ble av teamet bestemt til å være å produsere en MVP som dekket applikasjonens basisfunksjoner. Teamet bestemte også at applikasjonens infrastruktur skulle være skalerbar. Applikasjonens basisfunksjoner ble definert som en navigerbar nettside i tillegg til akseptansekriteriene for brukerhistoriene som omhandlet opprettelse, sletting, visning, påmelding og avlysning av aktiviteter, samt brukerregistrering og pålogging. For å oppnå dette, måtte teamet først lage en domenemodell, og deretter produsere arkitektur ut fra denne for både front og backend. I tillegg til dette ble kontinuerlig integrasjon implementert med vår egen runner. Denne testet koden og oppdaterte javadoc på GitLab pages for hver push. Teamet måtte også opprette en database. Første iterasjon wireframes og brukertester skulle også produseres i løpet av Sprint 1.

Teamet bestemte seg for å jobbe kontinuerlig med wiki og dokumentasjon gjennom hele prosjektprosessen slik at man unngikk at dokumentering stjal for mye tid i prosjektets sluttfase. Etter at alle oppgavene var definert, prioritert og estimert ved hjelp av Scrum Poker, ble de lagt inn i backlogen for sprint 1.

Målene for sprinten kan sies å ha blitt fullført til en grei grad. Nettsiden var navigerbar, og så bra ut. Likevel manglet produktet noen av basisfunksjonene som var definert i sprintmålene. Komponentene som var nødvendige for at basisfunksjonene skulle fungere var på plass, men de hadde ikke blitt sydd sammen. Teamet investerte mye tid i den underliggende arkitekturen for prosjektet. Arbeidsfordelingen var også ofte gjort slik at hvert teammedlem bare skulle implementere deler av en user story. Som resultat av disse tingene var mye av produksjonen som hadde blitt gjort fortsatt ikke presentabel når sprinten var over.

Sprint review ble holdt den 22/04-2021. Teamet fikk her vist både scrum master og produkteier resultatet av arbeidet med sprinten. Både scrum master og produkteier var fornøyd med fremgangen i utviklingen, og mente at teamet hadde lagt et godt grunnlag for å lykkes.

Sprint retrospektiv ble holdt rett etter sprint review, og teamet kom med tilbakemeldinger til hverandre om hva man burde fortsette å gjøre, hva man burde begynne å gjøre, og hva man burde slutte å gjøre fremover. Dette gjorde at teamet innførte endringer som vi tok med oss inn til Sprint 2. En av disse endringene var å innføre jobbing i par slik at teammedlemmene skulle ha bedre oversikt over prosjektstrukturen. Det ble også besluttet at teammedlemmene som jobbet med en user story i større grad skulle prøve å i implementere den i hele stacken.

7.1.2 Sprint 2

Sprint 2 ble satt i gang torsdag 22/04-2021 og ble fullført fredag 30/5-2021. Sprintens hovedmål var å levere et ferdig produkt innen tidsrammen, inkludert de deloppgaver som ikke ble ferdig under Sprint 1. Oppgaver som ikke var en del av sprint 1 var også inkludert i sprintmålet. Disse var søk, filtrering og sortering av aktiviteter, venteliste på aktiviteter, spillifisering av produkt, kart og kalendervisning av aktiviteter, endring av brukerinformasjon og aktivitetsinformasjon. Ettersom infrastrukturen til produktet ble ferdigstilt i sprint 1, var det relativt lett å implementere de resterende brukerhistoriene. Under sprint 2 ble det også jobbet en del med dokumentajon, da spesielt med sluttrapporten og eventuelle mangler i wikien.

Målene for sprinten har blitt gjennomført. Vi har klart å gjennomføre de fleste kravene gitt av produkteier med noen unntak. De teamet ikke fikk gjennomført nedprioriterte teamet grunnet tidsrammen, og at dette var funksjonalitet vi ikke anså som nødvendig for å få et godt produkt.

Vi implementerte også flere av løsningene på problemene vi gjennomgikk på retrospektiven vi hadde i slutten av første sprint. Blant annet ble det i større grad tatt i bruk parprogrammering, spesielt ved litt vanskeligere oppgaver. Videre jobbet også flere av medlemmene seg gjennom hele brukerhistorien. Videre er vi fornøyde med prioriteringene vi gjorde med userstoriesene vi valgte å gjennomføre. Test og dokumentasjonsskrivinga vi gjorde underveis i prossessen har vi også vært fornøyd med. Når det gjelder forbedringspotensialer vi har tatt ut fra retroperspektivet kan man spesielt trekke ut at tidsplanleggingen burde vært bedre, og at teamet må være mer forsiktig med å merge til hovedgrenen på “GitLab”.

8. Diskusjon og konklusjon

Som nevnt i kapittel 6. er det viktig at et system er utformet slik at det er mottakelig for enhver bruker i mottakergruppen. Gjennom prosjektet holdt teamet av den grunn et fokus på implementasjonen av prinsippene i universell utforming. For at bilde-elementer skulle være tilgjengelig på flere format ble alt-tags lagt til. I tillegg ble størrelser på bilde og elementer valgt for å være mest mulig synlig og tilgjengelig for brukerne. Videre er tjenesten tilgjengelig på flere forskjellige enheter, som mobil, nettbrett og PC.

Prosessen teamet fulgte var den agile utviklingsprosessen Scrum. Det ble holdt standup-møter hver morgen og en retrospektiv på slutten av hver sprint. Standup-møtene hjalp teamet å ha oversikt over hva som ble gjort til enhver tid og hvor det trengtes arbeidskraft eller hjelp. Retrospektiven som ble gjennomført etter sprint 1 ga teamet mye nyttig data om hvordan prosessen kunne tilpasses for å forbedres. Endringene som ble innført som resultat av dette hjalp teamet både med å utvikle kompetanse, og å øke effektiviteten i gruppen.

Teknologien som ble valgt gjorde jobben den skulle gjøre bra. Både Java Spring og React er rike rammeverk som inneholder mange utvidelser. Ulempen med å velge disse rammeverkene var at de var nye for flere i gruppa. Læringskurven til teknologiene gjorde at teamet utviklet noe tregere i sprint 1, da tid måtte investeres til læring. Dette ble hjulpet veldig av prosessen som ble fulgt, da endringene som ble innført etter sprint review 2 gjorde det mye lettere for medlemmene å spre, samt tilegne seg kompetanse. I sprint 2 ble fordelene med de valgte teknologiene tydeligere. Etterhvert som teammedlemmene ble flinkere, kunne redskapene som teknologien tilba brukes for å produsere bra kode effektivt.

Teamet fungerte meget bra sammen, flere av teammedlemmene var kjent fra før av, noe som gjorde at det var lett å kommunisere. Det var også et stort spekter av forskjellige kompetanser på teamet, noe som gjorde at teammedlemmene kunne lære mye av hverandre. Dette ble lagt ekstra merke til i sprint 2 da parprogrammering ble innført. Teammedlemmene sine mål for prosjektet stemte også bra overens fra starten av, noe som gjorde det lettere å komme til enighet i avgjørelser om både produkt og prosess.

Produktet teamet endte opp med å utvikle fungerte bra som en løsning på problemstillingen gitt. Oppgaven var veldig fri, i den forstand at alt som ikke var essensiell funksjonalitet hadde lav prioritering. Dette tillot teamet å ha en dialog med produkteier, og derfra beslutte hvilken funksjonalitet som skulle prioriteres. Sluttproduktet møtte målene som ble satt ved starten av sprint 2. Likevel møtte ikke produktet alle spesifikasjonene som var spesifisert i visjonsdokumentet. Flere av tilleggsfunksjonene som venner, grupper og chat ble ikke implementert, men også noen funksjonelle krav, derav brukertilbakemelding, at brukere kan få status som “trusted” og innlogging gjennom kjente sosiale plattformer. Disse funksjonene måtte droppes til fordel for funksjonalitetene med høyere prioritet satt av produkteier. Teamet prioriterte gjennom hele utviklingsprosessen at koden som ble produsert skulle være av bra

kvalitet, og dette førte til at ikke alle ønskede funksjoner ble ferdig. I dialogene teamet hadde med produkteier så virket det likevel som om han var veldig fornøyd.