Forstudie

TDT4140 Programvareutvikling, vår 2019

Gruppe 29

Februar 2019

Innhold

1 Problembeskrivelse

Produkteier ønsker et bookingsystem for bordreservasjoner for sin restaurant Trippin' Tacos. Kunden skal ha mulighet å booke på nett, og ansatte skal kunne legge inn *walk in* kunder. I tillegg ønsker han at restauranteier skal kunne se statistikk over besøkende.

Gruppen har blitt enig om å ta for seg problemet helhetlig og skal dermed lage hele reservasjonssystemet som produkteier beskriver, til den grad produktbeskrivelsen blir tolket.

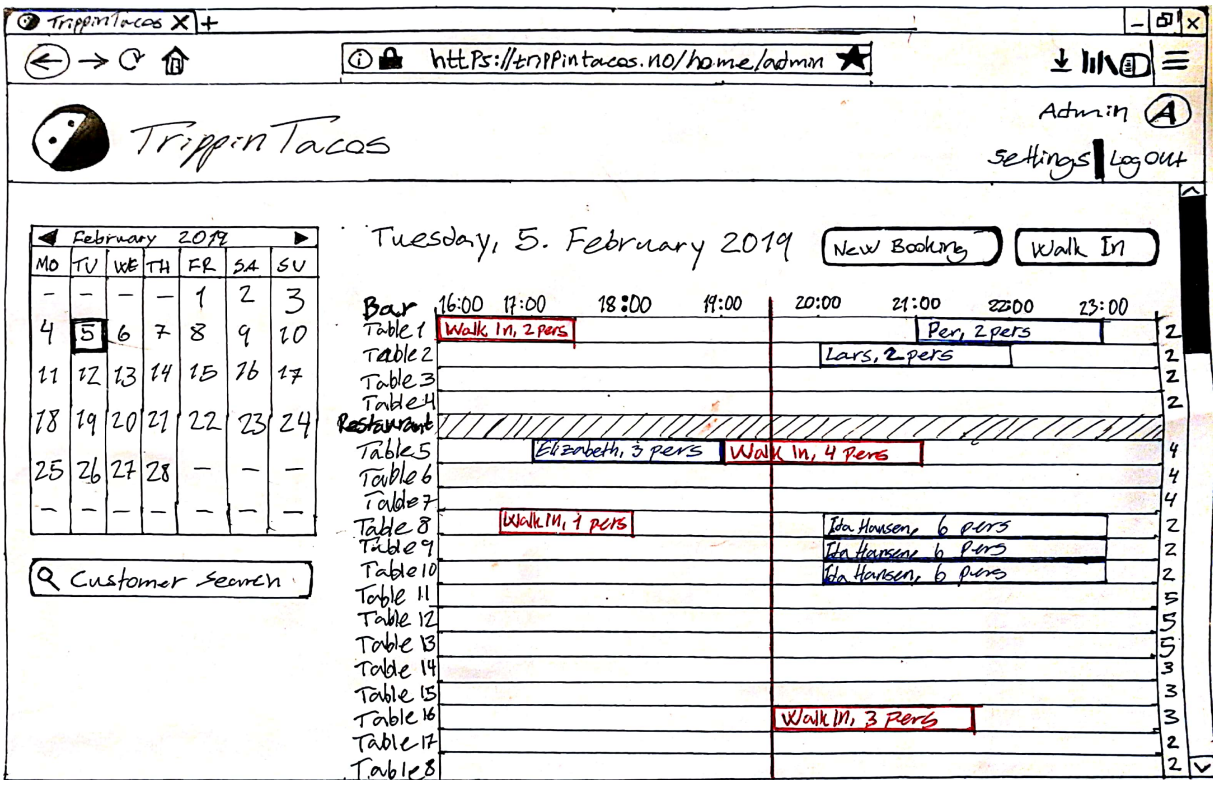
2 Foreløpig løsningsforslag

Vår løsning vil være et web-basert reservasjonssystem. Systemet vil være todelt med hver sin funksjonalitet: det ene omhandler bordreservasjoner, mens det andre håndterer eksisterende reservasjoner og besøksstatistikk. Gjestene skal enkelt kunne opprette og avbestille bordreservasjoner uten å ha kjennskap til hvordan restaurantens layout og booking-system fungerer. Restaurantens personvernerklæring skal være synlig for gjesten under bestillingen med ekstra fokus på punktene i EUs personvernforordning (GDPR). Servitørene skal også enkelt kunne opprette, endre og slette en bordbestilling på vegne av en gjest eller som en *walk in*. Servitørene vil derfor få utvidede rettigheter for å ha tilgang til alle reservasjoner som ligger i systemet til enhver tid, og kun admin har tilgang til statistikk.
Bordreservasjoner og *walk in* vil oppføre seg ganske likt i systemet, bortsett fra at *walk in* ikke er tilknyttet en person. Vi har definert en reservasjon til å bestå av:

- Tid og dato for besøket
- Varighet (standard er to timer)
- Antall gjester
- Navn og kontaktinfo
- Allergier eller en kommentar

2.1 Layout og design

2.1.1 Servitører

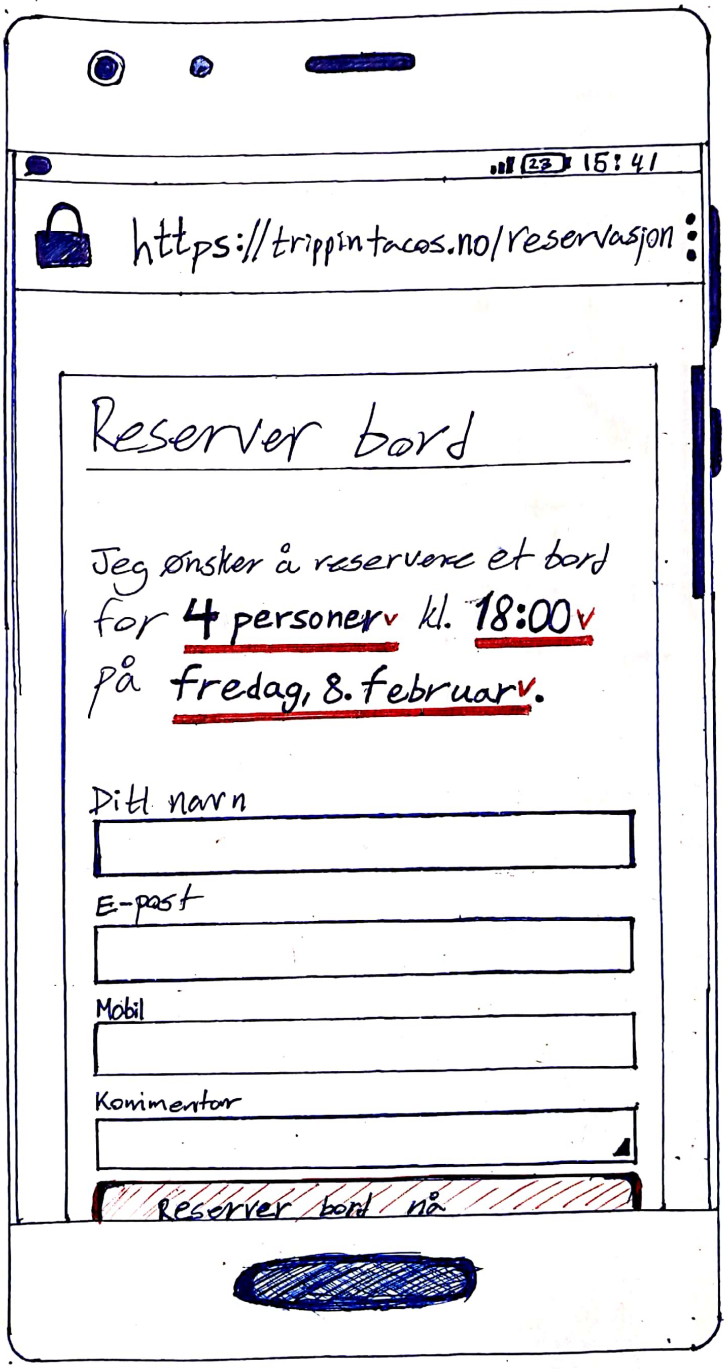


Figur 1: Eksempel på en desktop løsning for servitører

Figuren over viser et forslag til hvordan servitørenes system kan se ut. Vi ser for oss at hvert bord har en tidslinje som fylles med reservasjoner i løpet av en dag. Det skal kunne gå an å trykke på hver reservasjon for mer informasjon og for å vise alternativer om endring og sletting. På figuren ser du at reservasjoner og *walk in* vises på samme måte bare med forskjellige farger. Det er også mulighet for å bruke kalenderen til venstre for å velge dato. Vi ser for oss at restauranten har en passordbeskyttet bruker som alle servitørene deler for å slippe å bruke tid på å logge inn og ut av systemet når det er travelt.

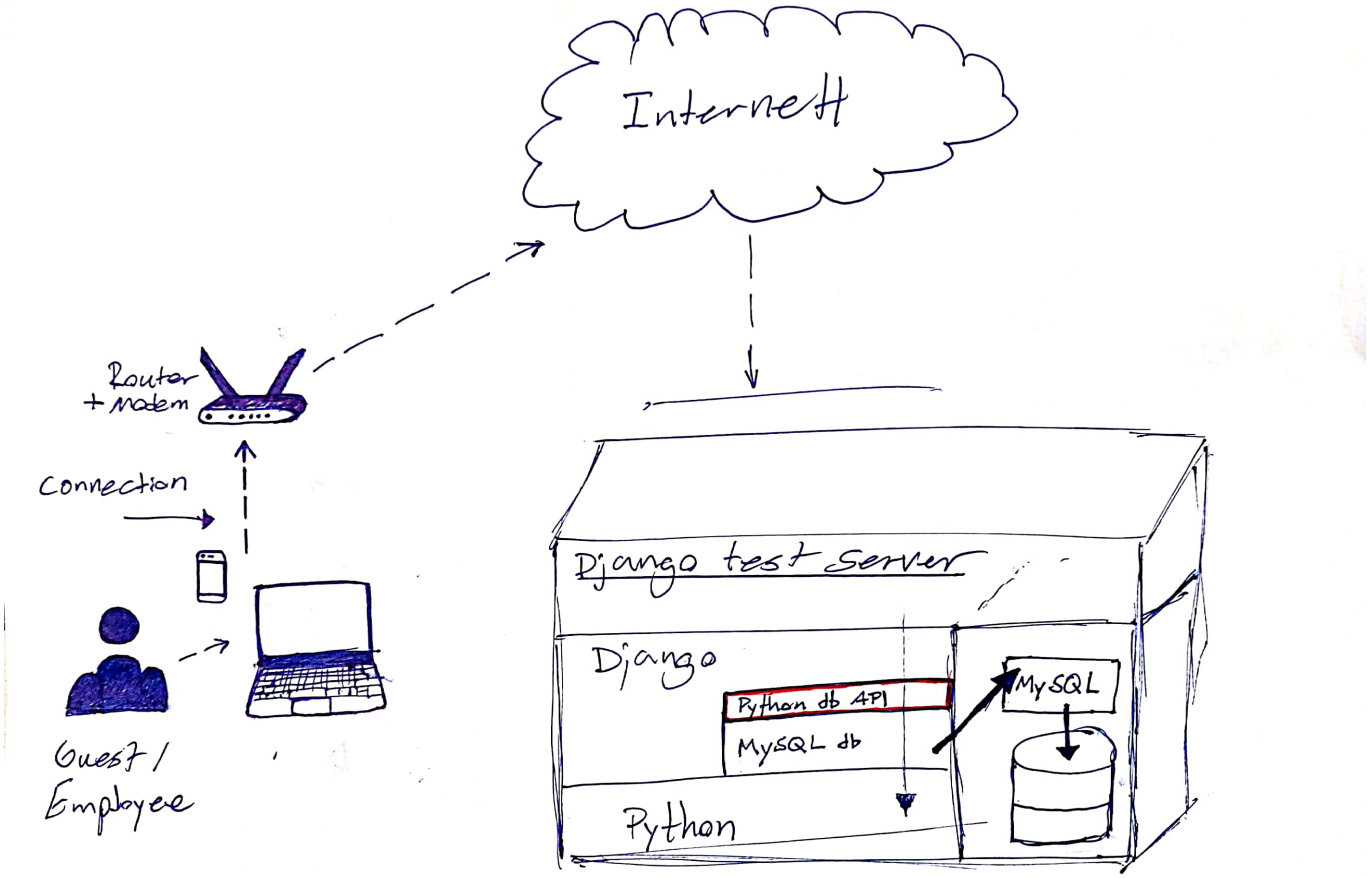
2.1.2 Gjestene

Figuren nedenfor burde være ganske selvforklarende. Gjesten fyller ut nødvendige felter for reservasjonen og avlsutter ved å trykke på 'bekreft'. Vi har valgt å bruke rullegardinmeny for å velge antall personer, klokkeslett og dato for besøket. For kontaktinformasjon og kommentar eller allergi har vi valgt å bruke tekstbokser. Layouten er ganske ren og enkel for å gjøre det lett å reservere et bord.

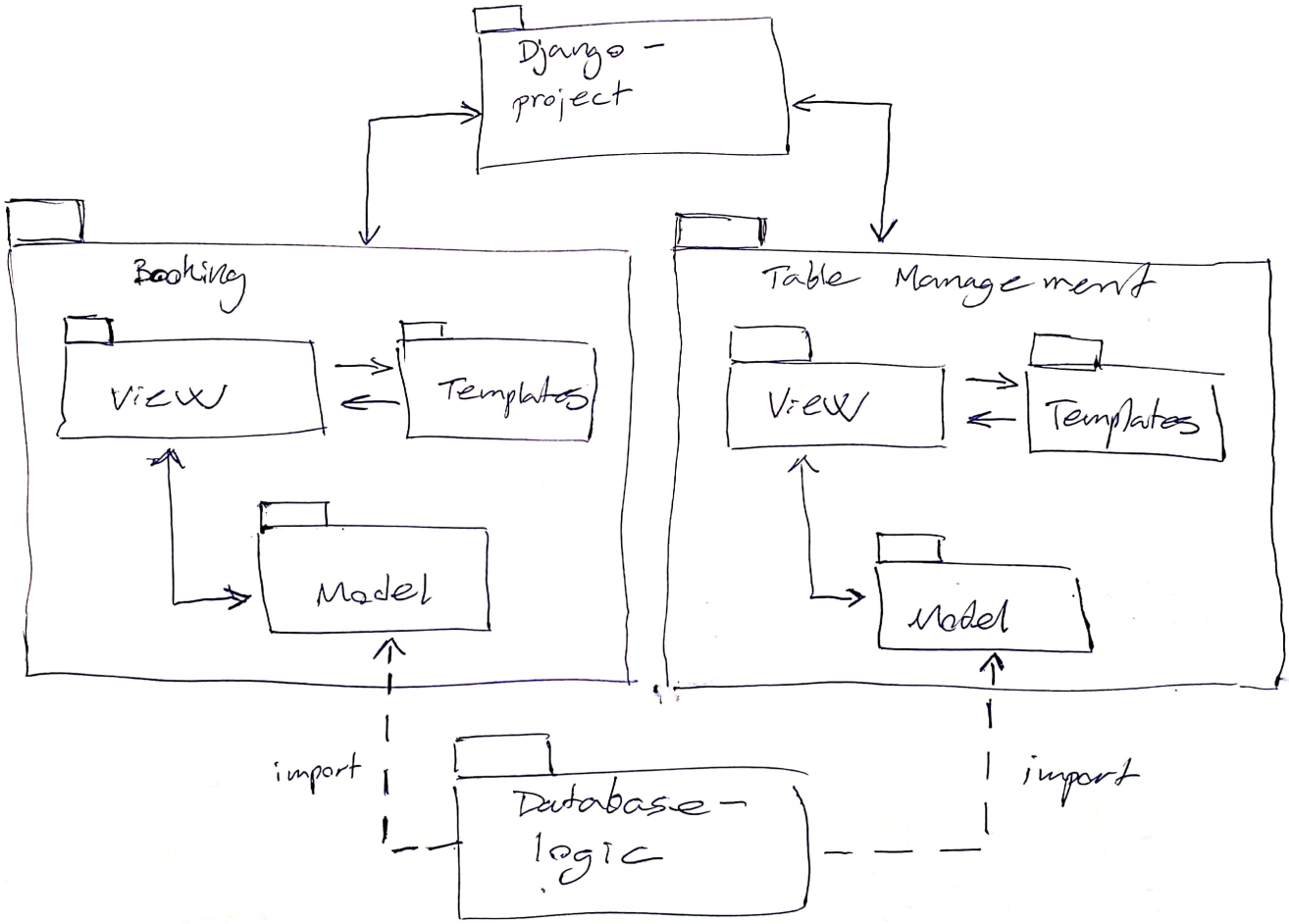


Figur 2: Eksempel på en mobilløsning for gjestene

2.2 Arkitekturskisser



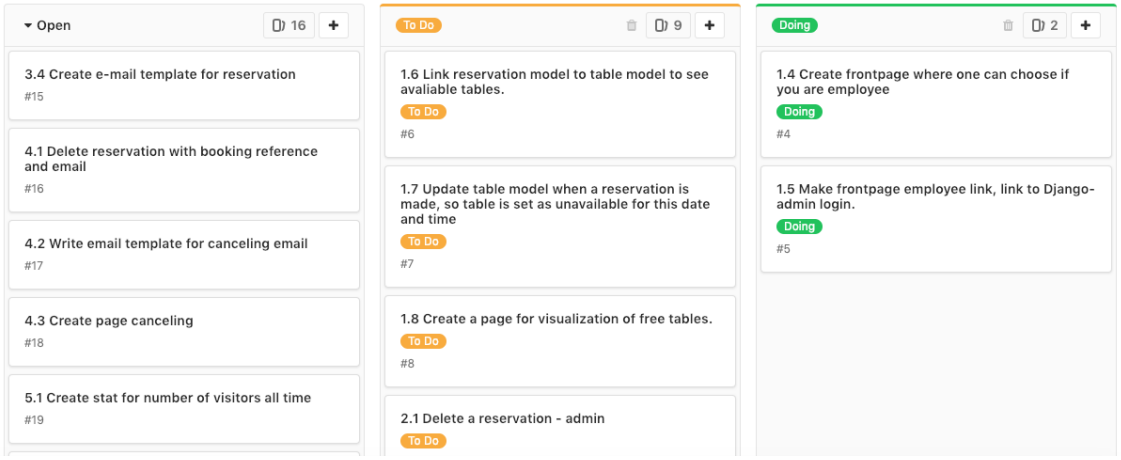
Figur 3: *Deployment*-diagram av løsningen



Figur 4: *Component*-diagram av løsningen

2.3 Brukerhistorier og issues i GitLab

Etter å ha fått oppgaven av produkteier hadde vi et møte for å liste opp brukerhistorier i et spreadsheet. Brukerhistoriene har en ID, et navn, en prioriteringsrangering (mellom 1-100) og en kort beskrivelse om hva brukerhistorien skal gjøre. Vi har også gitt hver brukerhistorie et estimat over hvor lang tid vi tror det tar å gjøre de ferdig. Vi gikk gjennom brukerhistoriene med produkteier på et senere tidspunkt, og da han var fornøyd med prioritering og funksjonaliteter delte vi brukerhistoriene opp i mindre oppgaver som vi kaller tasks.



Figur 5: Her er brukerhistoriene delt opp i tasks og lagt inn i GitLab. Disse er blandt de vi vil ha med i første sprint.

3 Teknologisk plattform

3.1 Programmeringspråk

Vi har bestemt at Django rammeverket skal brukes som backend. Grunnlaget for dette er at flere medlemmer på gruppen har kunnskap innen Django, samt mye innebygd funksjonalitet, som gjør det lettere og bedre å lage et bra produkt. Produkteier var også likegyldig til hva vi skulle bruke, så da falt Django som et naturlig valg.

Python er hovedprogrammeringsspråk, fordi Django er et Python rammeverk. I tillegg vil vi bruke HTML, JavaScript og CSS for front-end.

3.2 Programmeringsomgivelse

Som programmeringsovgivelse(r) ser vi det opp til hver enkelt på gruppen å bruke den/de har mest erfaring med og den/de er mest komfortabel med, da vi ikke har noen spesielle krav til funksjonalitet annet enn at det er kompatibelt med programmeringsspråkene vi skal bruke. De fleste har erfaring med bruk av PyCharm, men noen foretrekker Visual Studio Code.

3.3 Databaseteknologi og språk

Når det kommer til databaseteknologi og databasespråk, har Django innebygd støtte for databaser. Vi har bestemt oss for å bruke NTNUs MySQL servere, da lokal SQLite er standard. Ettersom Django har støtte for SQL-spørringer vil vi skrive til MySQL i Python. For å se innholdet i databasen vår, vil vi bruke MySQL workbench og/eller phpMyAdmin.

Da ingen på gruppen har veldig stor erfaring med andre databaser innen Django enn Djangos standard SQLite, vil vår plan B her være å bruke denne, dersom MySQL skulle vise seg å ikke være samarbeidsvillig og møte våre krav.

3.4 Tredjepartskomponenter og kode

Som nevnt vil vi bruke Django og MySQL workbench som tredjeparter. Disse ser vi på som sikre komponenter å bruke, da de kommer fra verifiserte kilder. Ellers kommer vi til å søke på nettet etter hjelp ved behov. Koden vi da finner kommer vi til å være kritiske til og lese nøye igjennom før vi tar lærdom av den.

3.5 Nettverksprotokoller

Vi kommer ikke til å sette opp noen protokoller selv, dette gjør Django for oss. Django bruker HTTP og TCP, derfor vil dette brukes.

3.6 Oppsett for automatisk release

For oppsett av automatisk release, vil vi passe på at ”Master branchen” på GitLab til alle tider har et fungerende produkt, slik at vi er sikre på at koden som ligger der trygt å publisere.

3.7 Hvor skal produkteier og andre laste ned og teste koden?

Produkteier og andre interesserte vil få linken til ”Master branchen”, slik at de alltid kan få pullet og testet kode som vi mener virker.

3.8 Testing

3.8.1 Developer testing

I gruppen har vi utnevnt en testansvarlig. Denne personen har det overordnede ansvaret for testing. I følge [1] er unit testing ”the process of testing program components, such as methods or object classes”. Testansvarlig skal

skrive egen kode som sikrer spesifikke metoders integritet. Dette vil foregå etter normene beskrevet i [1] for enhetstesting. Selv om ett medlem er ansvarlig, vil samtlige på gruppen teste sin egen kodes funksjonalitet fortløpende.

Systemtesting:
Systemtesting er helhetlig testing etter alle komponentene er satt sammen.[1] Dette kommer til å gjøres jevnlig, for å sikre at funksjoner og metoder fungerer sømløst. Det overordnede ansvaret her vil falle på testansvarlig.

3.8.2 Release testing

Release testing er å teste en spesifikk release av systemet som er ment for å tas i bruk. [1]. I utgangspunktet skal ikke utviklerteamet ha ansvaret for release testingen. Testansvarlig er mindre involvert i kodebidrag for å kunne teste fra et nøytralt ståsted.

3.8.3 User testing

Under user testing er det opp til brukere å teste systemet. Det finnes tre typer user testing i følge [1]; ”Alpha testing”, ”Beta testing” og ”Acceptance testing”. Under Alpha testing jobber brukere tett med utviklerteamet, mens under Beta testing gis det ut en release av systemet til brukertestere. En akseptansetest er ”generalprøven” før systemet lanseres. Her bestemmer brukerene om systemet er klart for release eller ikke.

Vi kommer ikke til å utføre Alpha testing, ettersom vi ikke har muligheten til å jobbe tett opp imot brukertestere. Vi kommer derimot til å utføre Beta testing, med et utvalg forskjellige personer med forskjellig ferdighetsnivå og datakunnskaper.

Akseptansetesten kommer til å bli tatt med produkteier før hver sprint er gjennomført.

4 Standarer for koding, dokumentasjon og bruk av GitLab

4.1 Kodekonvensjoner

4.1.1 Variabelnavn

I henhold til standarder for koding er det enighet om å bruke beskrivende variabelnavn. Disse skal også skrives på engelsk, ettersom språkets dokumentasjon er skrevet på engelsk.

For oversiktlighets skyld planlegger vi å skrive variabelnavn med små bokstaver og å skille med understrek dersom variabelnavn er sammensatt av flere ord. Dette faller under Pythons konvensjoner.

4.1.2 Funksjonsnavn

Funksjoner kommer til å bli skrevet med små bokstaver på starten av hvert påbegynte ord, og i motsetning til variabelnavn så skal funksjonsnavn sammensatt av flere ord skilles med stor bokstav [1]. Denne formen for å skille ord på er også kjent som ”camelCase”.

4.1.3 Ryddighet og forståelig kode

For å opprettholde ryddighet i koden vil vi føye oss etter Pycharms krav rundt syntaks og formatering.

For at det skal være enkelt å fortsette påbegynt arbeid planlegger vi å legge ved korte, konsise kommentarer i koden.

4.2 Dokumentasjon

Hovedsaklig kommer vi til å benyttes oss av dokumentasjon som allerede ligger på nettet for å hanskes med problemer, i tillegg til en enkel oversikt for bruk av Git.

4.3 Git og GitLab

4.3.1 Plattform og kodedeling

For å arbeide sammen på samme kodebase blir Git[1] brukt som versjonskontrollsystem. Git-repsitorien vi kommer til å benyttes oss av er GitLab [2]. Dette verktøyet kommer ikke bare til å brukes som kodebase, men tilbyr også annet nyttig funksjonalitet som oversikt over issues, som beskrevet i kapittel 2.3. Flere er allerede kjent med Git og GitLab fra tidligere arbeid, noe som gjør det enklere å hjelpe de resterende på gruppa med å gjøre seg kjent med vektøyene.

4.3.2 Branches

For å bestandig ha et produkt som fungerer kommer vi til å benytte oss av branches i Git. Som nevnt, kommer vi til å ha én ”master-branch” der kun fungerende kode ligger. I tillegg vil vi ha en ”dev-branch”, som er en prototype til master-branchen med lavere terskel for merging av ny kode.

For å sikre at utestet og ny kode skrevet av forskjellige personer ikke kolliderer under utvikling, vil det være hendig å lage branches basert på hvilke brukerhistorier som gjøres. Når en brukerhistorie er fullført kan den merges med dev-branchen.

[1] Se `https://git-scm.com/about` for mer informasjon.
[2] Se `https://about.gitlab.com/` for mer informasjon

Vi kommer også til å fjerne muligheten til å pushe rett inn i master, grunnen til dette er for å forhindre at ødeleggende kode feilaktig blir dyttet inn i branchen. Dette tenker vi kan være hendig da flere på gruppen ikke er kjent med Git, samtidig som det er greit å få egen kode sett over av andre før det blir brukt i den endelige versjonen.

4.3.3 Commits

Når det kommer til Commits, vil vi ha det som praksis at all kode som skrives skal pushes til GitLab ved endt arbeidsøkt. Å commite så ofte som mulig har fra tidligere erfaring har vist seg å være veldig hendig.

Når det gjøres commits vil vi også at det skal legges ved en kort og konsis beskrivelse av hva som er gjort. Disse meldingene skal skrives på engelsk.

4.4 Historikk og prosjektsamarbeid

For å holde oversikt over prosjektsamarbeidet kommer vi til å bruke GitLab sitt "Issue board". Dette verktøyet tilbyr en oversiktlig og intuitiv måte å fremstille fremgangen på brukerhistorier og andre ting som eventuelt skal gjøres. Se gjerne 2.3 for mer informasjon.

5 Risikoanalyse

5.1 Tekniske risikoer

Det er hovedsakelig tre tekniske utfordringer som på forhånd er identifisert som de viktigste å være oppmerksomme på: at koden ikke er konsistent og lett gjenbrukbar, at det oppstår problematikk i synkronisering mellom Django, SQL og Git, samt problemer knyttet til databasens kapasitet. Antakeligvis vil det gå greit med de to sistnevnte, ettersom dette er enormt populære systemer som brukes mye i samhandling. I forbindelse med kode planlegger vi å på forhånd tenke nøye gjennom det overordnede systemet med objektorienterte briller før vi begynner å skrive koden. Ved å dermed øke muligheten for smidighet vil det også bli lettere å tilpasse seg nye brukerkrav.

5.2 Organisatoriske og mellommenneskelige risikoer

Når det kommer til de organisatoriske rammene er det meste av strukturen fastsatt i forbindelse med prosjektets natur. Ettersom dette er et studentprosjekt foreligger ikke restrukturering eller finansielle påvirkninger på lik linje med en tilsvarende situasjon i arbeidslivet.

I forbindelse med menneskelige risikoer er henholdsvis to potensielle utfordringer identifisert; at medlemmer blir overbelastet av arbeid i andre sammenhenger i ugunstige perioder, samt sykdom/fravær som går ut over gruppen

sin generelle fremdrift. For å unngå førstnevnte situasjon er strategien å kommunisere om fremtidig utilgjengelighet, samt å synkronisere kalendere. Det er generelt lite å gjøre noe drastisk med sykdom og fravær, men gruppen har som mål å holde hverandre oppdatert på det ethvert medlem arbeider med gjennom daily scrum [2].

5.3 Estimat- og kravspesifikke risikoer

Det er naturlig å forvente utfordringer rundt tidsbrukestimater for en gruppe som gjør sitt første prosjekt innenfor systemutvikling. For å unngå problematikk og for å overholde tidsfrister er planen å ikke ta vann over hodet i forbindelse med problemløsning, eksempelvis å i første omgang fokusere på kjernefunksjonalitet fremfor estetikk. Når det kommer til endring av kravspesifikasjon er vi nødt til å være smidige og diskutere med kunden og gruppa hva som er innenfor rimelighetens grenser.

5.4 Tabell over risikoer

Risiko	Sannsynlighet	Effekttyngde	Håndtering
Gruppemedlemmer er syke i kritiske situasjoner	Moderat	Alvorlig	Kommunikasjon
Tidsbruken for utviklingen av programvaren underestimeres	Høy	Alvorlig	Øke arbeidsmengde/omprioritering
Endringer i kundekrav slik at større systemforandringer må gjennomføres	Høy	Tolererbart	Dialog med produkteier
Databasen kan ikke bearbeide forventet mengde pågang	Lav	Tolererbart	Endre database
Mangler i gjenbrukt kode må repareres/endres før videre bruk	Moderat	Tolererbart	Tilpasse kodens smidighet
Programvareverktøy kan ikke integreres med tredjepartssystemer	Moderat	Tolererbart	Fortløpende tilpasning
Kunden forstår ikke omfanget av spesifikasjonsendringer	Høy	Tolererbart	Dialog med kunden
Nedprioritering av faget slik at mer arbeid tilfaller resterende medlemmer	Høy	Tolererbart	Dialog med relevante medlemmer
Mengden feilsøking og reparasjon av defekt kode underestimeres	Lav	Tolererbart	Økt arbeidsmengde

Tabell 1: Tabell over riskoer

6 Releaseplan

Sprint 1 (Codename Arctic Fox)
Start date: 11.02.2019
End date: 27.02.2019
Goal: Employees can make reservations

Importance	ID	Estimate
100	1	9
90	2	5
80	3	3
70	4	3
60	5	7

Sprint 2 (Codename Black Hawk)
Start date: 06.03.2019
End date: 30.03.2019
Goal: Make feedback integrated in the system

Importance	ID	Estimate
50	6	6
10	7	2
7	8	3
2	9	5
1	10	3

Figur 6: Releaseplan

Releaseplanen består av 2 sprinter med kodenavn, varighet og mål vist i figur 6. Det ligger ingen andre brukerhistorier i backloggen, da alle brukerhistoriene våre fikk plass i releaseplanen.

Referanser

[1] Ian Sommerville. *Software Engineering*. Perason education limited, 2016.

[2] Henrik Kniberg. *Scrum and XP from the Trenches*. Lulu. com, 2015.