ObliGentle: Procesrapport

Svendeprøve ved
Kenneth Løvgren
&
Rasmus Ladefoged Wolffram

Aarhus Tech

Sune Koch Rønnow
sune@kochroennow.dk

8. december 2023

Titelblad

Deltagere	Sune Koch Rønnow
Projektnavn	ObliGentle
Skole	Aarhus Tech
	Hasselager Allé 2, 8260 Viby J
Projektperiode	13. november 2023 - 15. december 2023
Afleveringsdato	8. december 2023
Vejleder	Kenneth Løvgren & Rasmus Ladefoged Wolffram

Underskrifter

Sune Koch Rønnow Dato

Kenneth Løvgren Dato

Rasmus Ladefoged Wolffram Dato

Indhold

Kapitel 1

Formalia

1.1 Indledning

1.2 Læsevejledning

I forbindelse med svendeprøven er vi blevet pålagt at aflevere to rapporter, en om projektets proces og en om dets produkt. Dette er procesrapporten, som fokuserer på min overvejelser og refleksioner i forhold til projektet. Jeg antager, at læseren har en teknisk viden på niveau med en færdiguddannet datateknikker.

1.3 problemformulering

Lav en opgaveorganiserings-/kalenderapplikation, hvor brugere har mulighed for at styre forskellige typer af opgaver:

- **Aftaler**: Opgaver som bruger skal gøre på, eller inden et bestemt tidspunkt og som automatisk bliver færdiggjort på det pågældende tidspunkt.
- **Projekter**: Opgaver som brugeren selv har valgt, hvor brugeren kan notere fremgang og færdiggørelse.
- **Tjanser**: Tilbagevendende opgaver som brugeren skal have gjort ved lejlighed og hvor kun de vigtigste tjanser vises.

Applikationen skal laves som en *RESTful API* med *Crossplatform* brugerflader. Det skal være muligt for brugere at tilgå deres profil både fra et webinterface og en android app. Applikationen skal være modulært opbygget og rumme mulighed for, at jeg eller andre udviklere, kan udviklere videre på den. Applikationen skal i udgangspunktet sættes op online og have en *proof-of-concept tilgængelig.*

1.4 case

Tema/titel for	ObliGentle
Godkendt nummer	
Casebeskrivelsen	Det er en reel udfordring for mig, at jeg på den ene side godt kunne bruge et program til at hjælpe mig med at organisere mit privatliv, men på den anden side oplever jeg, at de fleste programmer enten er udviklet til arbejdsmarkedet eller fungerer som om de var. Dette fungerer ikke for mig, fordi jeg ikke gider at føle arbejdsdagen forsætter, efter jeg er kommet hjem. Derfor er idéen bag ObliGentle at lave en opgave organisator/kalender, som fokuserer på at hjælpe brugeren med det, som man har overskud til uden at den udskammer en for det man ikke nåede. Programmet skal kunne håndtere tjanser, ting som brugeren skal gøre, projekter, ting som brugeren gerne vil gøre, og aftaler, samt ting brugeren skal gøre på eller før et bestemt tidspunkt. Programmet skal hjælpe brugeren med at organisere sin opgaver, men hverken moralisere, direkte eller indirekte, om brugeren når nok eller optimerer sin produktivitet tilstrækkeligt. Derimod skal programmeret fejre, hvad det bliver gjort, og hjælpe med at prioritere det vigtigste, fremfor at alt skal gøres hver dag. Programmet kommer til at være baseret på personlige præferencer og ikke nyeste forskning. Det havde selvfølgelig været præferablet, men det er ikke realistisk at orientere sig i et nyt forskningsområde ved siden at man lave sin svendeprøve. Programmet forsøges dog at laves modulært og nemt foranderligt, så det i en potentiel fremtid ville kunne tilpasses forskningsfeltet[1].
Teknologier	Python & FastAPI JavaScript og React Native MongoDB
Fagområder som projektet dækker	Clientside programmering Serverside programmering GUI programmering Programmeringsmetodik Objektorienteret programmering Database programmering APP programmering
Valgfri specialefag som projektet dækker	

[-1] Eller mere individuelle behov, da folk nok organisere sig selv bedst på forskellige måder
[0] Eller mere individuelle behov, da folk nok organisere sig selv bedst på forskellige måder
[1] Eller mere individuelle behov, da folk nok organisere sig selv bedst på forskellige måder

Kapitel 2

Metode- og teknologivalg

2.1 Udvikling og planlægning

Valget af udviklingsmetode endte med at være en større udfordring for mig, end jeg egentlig havde antaget. På den ene side er jeg primært blevet oplært i *scrum* og har trives så godt i det, at jeg ikke rigtigt har søgt erfaringer i så mange andre; men på den anden side er det tvivlsomt, i hvilken grad en én-persons *scrum* proces overhovedet er mulig.

Hvorvidt en én-persons scrum proces overhovedet er mulig, er et forbløffende debatteret emne på nettet[1][2], og egentlig en debat, som jeg ikke ser nogen væsentlig grund til tage del i, men jeg vidste, at jeg ikke havde i sinde at udnævne mig selv hverken til *Scrummaster* eller *produktowner*, fordi det, i hvert fald i min læsning af scrum, er ret vigtigt, at disse er separate opgaver. Jeg forsøgte at gøre op med mig selv, hvad jeg egentlig på et dybere plan ønskede at få ud af Scrum, når det nu ikke var de faste roller: Det var i virkeligheden dens rod i agile udvikling, så det besluttede jeg mig for at fokusere på, i særdeleshed Agile Alliance 101[3].

Særligt fokuserede jeg på:s

- Holde daglige ajourføringer
- Dele projektet op i 3 iterationer/sprint
- Hvert sprint indebærer en planlægnings- og evalueringssession
- Indstille mig på, at processen kan ændres efter behov og de daglige ajourføringer og særlig sprintevalueringen skal danne et rum for dette

Derudover forsøgte jeg også at have de 12 principper[4][5], hvor princip 1 og 4 naturligvis blev ret frit fortolket, da jeg dårligt kan agere min egen kunde.Apar Jeg forsøgte endvidere at planlægge både for mit produkt og dokumentation og delte forløbet op i 3 sprints, hvor jeg havde 2 halve uger til start og slut, som fokuserede på helholdsvis at begynde og afslutte forløbet.

[1] https://www.scrum.org/forum/scrum-forum/36139/one-man-scrum-team-possible
[2] https://pm.stackexchange.com/questions/9081/a-one-man-project-methodology
[3] https://www.agilealliance.org/agile101/
[4] http://agilemanifesto.org/principles.html
[5] https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

Sidste men ikke mindst, så brugte jeg kanban-bræt for at holde overblik over opgaver. I slutningen af projektet, bevægede jeg mig lidt væk fra brættet og styrede det mere igennem logbogen, ajourføringerne og evalueringerne, men særligt i starten af projektet var det et vanvittigt godt redskab.

```
Daglige ajourføringer:
Hver formiddag afholdte jeg en daglig ajourføring, hvor jeg
    stillede følgende spørgsmål:
Daglig ajourføring
Holdes hver arbejdsformiddag, helst inden 10. Skriv kun ja,
    nej eller stikord. Hvis noget skal uddybes, så gør det
    nede i loggen.

Gårsdagen log: Fuldendt? Tilstrækkelig? Renskrevet?

Arbejdsproces: Opfølgninger? Blokeringer? Forandres?

Hvad er dagens prioriteter?
```

Disse er dokumenterede i logbogen. Ajourføringerne holdt samme form igennem hele processen. Dette skyldes, at jeg haft et svendeforløb før, hvor jeg lærte en del om, hvordan jeg arbejdede bedst med ajourføringerne.

```
Procesevalueringer:
Ved slutningen af hvert sprint, holdt jeg en
    procesevaluering, som endte med at tage denne form:
Evaluering ... af
Sidste evaluering
Seneste sprint
Tidsplanen
Produkt
Konklusion
```

De 3 evalueringer kan også findes i bilagsmaterialet. Procesevalueringerne skifter i højere grad form, hvor den første er helt andet format end de efterfølgende Sprintplanlægning: Sprintplanlægningen endte med at blive indtastet i kanban-brættet for at sikre mig, at det var ført ordentligt ajour.

Dette er Agile development og jeg mener også, at man kan se en klar inspiration fra Scrum, om end det er underimplementeret, men det leder jo til et naturligt spørgsmål. Hvordan kan det være, at man ikke anvendte egentligvis Extreme Programming (XP), som er et agile framework, men som godt kan bruges af en person? Der er svaret simpelthen blot, at det er det her, som jeg vil have ud af det.

2.2 RESTful API Arkitektur

RESTful API kombinerer REST, som står for *Representational state transfer* , softwarearkitekturen med en API. RESTful API indeholder 5-6 arkitektoniske retnings-

linjer[6]:[7][8][9]

- Klient-server arkitektur igennem HTTP
- "Stateless"klient kommunikation, hvilket vil sige, at hver "*request*"er separat og der ikke gemmes data om klienten
- "*Cacheable*"data - for bedre performance for klientsiden og bedre skalering på serversiden.
- Ensartet interaktion med *API'en*, hvor data anmodes tydeligt og separat fra klientens repræsentation deraf
- *lag*-inddelt serverarkitektur, hvor data sendes igennem flere lag, usynligt for klienten.
- "*Code-on-demand*"er at serveren kan sende kode, som eksekveres af klienten. Denne retningslinje er dog bredt anerkendt, som værende valgfri og ikke nødvendig.

Med andre ord, så handler RESTful API om, at man giver klienter mulighed for let og tydeligt at anmode og indsende data igennem http-request, men man styrer selv behandlingen af dataen på serveren grundet sikkerheds- og skaleringshensyn. Data sendes frem og tilbage mellem server og klient og ansvaret for at repræsentere dataen placeres i højest mulige grad hos klienten. For at RESTful API faktisk er "*performance*-let og sikker, er det alfa-omega, at man sørger for at grænsefladerne fungerer, som de skal.

SOAP[10] er et ældre alternativ. SOAP få request igennem en eller flere "*application layer*-protokoler, så som *HTTP*, *SMTP* eller *TCP*, men returnerer altid et *XML* dokument. Kontra *RESTful API* så har *SOAP* flere indbyggede standarder og servicer, eksempelvis i forhold til sikkerhed, men er også markant tungere for serveren[11]. Af mere moderne alternativer eksisterer også *GraphQL*, som fokuserer på kun at overføre det korrekte data fremfor at minimere server ressourcer, og *RPC*[12], som fokusere på at køre kode lokalt på klienten, men bruger serveren som en "*dependency*", som om den var lokal og man kan have logikken på serveren[13]. Både GraphQL og RPC er ret spændende, men jeg tror, at det er korrekt valgt at fokusere på ressourceforbrug i dette tilfælde og SOAP virker dateret i forhold til andre alternativer.

Potentielt kunne det være spændende at se på at sammenligne projektet med et ellers lignende *GraphQL*-og *RPC*-projekt, men herfra hvor jeg står nu, virker *RESTful API* som det korrekte valg.

2.3 Software design: SOLID-principper

SOLID er en række principper for at lave den bedst mulige *objekt-orienterede programmering* som muligt og inkluderer fem principper.

[6] i ingen særlig rækkefølge
[7] https://www.redhat.com/en/topics/api/what-is-a-rest-api
[8] https://en.wikipedia.org/wiki/REST
[9] https://www.ibm.com/topics/rest-apis
[10] **S**imple **O**bject **A**ccess **P**rotocol
[11] https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest
[12] **R**emote **P**rocedure **C**all
[13] https://blog.bitsrc.io/not-all-microservices-need-to-be-rest-3-alternatives-to-the-classic-41cedbf1a907

SOLID-principper kan fuldt ud implementeres i *Python*[14] og *JavaScript*[15][16] , men ingen af dem har egentlige interface implantationer, som man eksempelvis finder C#. Begge bruger såkaldt "*duck typing*" i stedet for normativ eller strukturel typesystem. Ikke desto mindre kan interfaces sagtens implementeres, men det har jeg lige forklaret i dets eget afsnit.

Alle **SOLID**-principper kan der skrives og siges meget om, hvilket er blevet gjort. Jeg vil forsøge at forklare dem så simpelt som muligt, hvilket vil medføre, at jeg flere steder vil være reduktiv og oversimplificerende i min gennemgang. For en længere og væsentlig mere grundig gennemgang kan jeg anbefale følgende ressourcer[17][18][19].

2.3.1 Single-Responsibility Principle

Nok det simpleste af principper og kan i bund og grund reduceres til, at en klasse skal gøre en ting og ikke flere.

2.3.2 Open-Closed Principle

Det skal være muligt at udvide en klasse/module/software entitet, men ikke ændre den. Så det er godt og positivt at bygge videre på andet software, men vores klasser skal ikke kunne ændre hinanden.

2.3.3 Liskov Substitution Principle

Reglerne og bestemmelserne for en ting, skal også gælde for dens undertyper af samme ting. Hvad vigtigere er, så er det, at tingen ikke skal have en masse regler og bestemmelser, som ikke også er gældende for undertyperne.

2.3.4 Interface Segregation Principle

"interfaces"tilhører klasserne og omvendt. Derfor skal klasser ikke gøres afhængige af funktioner, som de ikke bruger. I stedet for skal interfaces underinddeles og gøres mere specifikke, så det passer til klasserne.

2.3.5 Dependency Inversion Principle

Dette er desværre et lidt misvisende navn, for det handler ikke om at vende afhængeligheder rundt, men afkoble dem og have abstraktioner i mellem.

2.3.6 "Duck typing" og interfaces

I normativ typesætning, som man finder vi blandt andet C++, C# og Java[20], eksplicit deklarerer deres type og så er den såkaldte "*duck typing*" kendt for, at hvis "*det går*

[14] https://realpython.com/solid-principles-python/
[15] https://dev.to/denisveleaev/5-solid-principles-with-javascript-how-to-make-your-code-solid-1kl5
[16] https://blog.logrocket.com/solid-principles-single-responsibility-in-javascript-frameworks/
[17] https://stackify.com/solid-design-principles/
[18] http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
[19] https://realpython.com/solid-principles-python/
[20] https://en.wikipedia.org/wiki/Nominal_type_system

som en and og ligner en and, så er det en and", så programmet bestemmer typen efter opførelse og hvad det ligner og ikke deklarationer. "*Duck typing*" adskiller sig fra strukturel typesystemet, som man finder i eksempelvis typescript, *Go* og *OCaml*[21] , ved kun noget, af den "*duck type*"'ede struktur evalueres, hvorimod der det strukturelle typesystem evalueres på hele strukturen for at se, om to er kompatible. Så selvom *Python* og *Javascript* ikke rigtigt har interfaces, så kan man lave klasser, som opfører sig som et interface, og som også vil virke som et *interface*. Så man kan lave en klasse, som man aldrig initierer og på den måde vil de agere som en abstrakt klasse.

SOLID-principperne står ret stærkt i *Objekt-Orienteret Programming (OOP)*. Personligt, sådant helt anekdotisk, er min oplevelse, at programmører oftere kritisere *OPP* end **SOLID**. Et direkte alternativt til **SOLID** er der dog i *GRASP*[22], som er et ældre *OPP* principsæt, men så kan man vel næsten lige så godt bruge **SOLID**. **SOLID** er også blevet bearbejdet, således, at det angiveligt også skulle omfavne funktionelt programming og multiparadigme *microservices*, men der er ikke helt enighed, om det rent faktisk er det mest fornuftige og *Poul Merson* har eksempelvis foreslået hans egne *IDEALS*-principper til *microservices*[23].

Personligt er min erfaring, at **SOLID** i langt højere grad indeholder nogle mål at aspirere i mod, men at man altid kan få ens kode til at blive mere og bedre **SOLID** og det særligt er anvendt som målestok, når man refactor ens kode.

2.4 Tech Stack: FAR(n)M

FARM-tech-stack står **FastAPI**, **React** og **MongoDB**. Jeg har tilføjet et lille n efter R'et, fordi jeg bruger **React Native**. **FARM**-techstack'en er langt fra at være bredt etableret, men er eksempelvis nævnt af *MongoDB*[24]. Ikke desto mindre er det en valid *tech-stack*, som tillader mig at arbejde med **Python** og **JavaScript** i et *MEVN/MERN* lignende stack. Stack'en er også valgt for at prøve en nyere og spændende tech-stack, som ikke er helt så etableret og som jeg har lidt mindre erfaring med.

2.4.1 Sprog: Python

Python bliver blot ved med a stige i popularitet og selvom det kan blive anset lidt som *datascience*- og begynderfokuseret sprog, har det en bred palette af anvendelighed.

På trods af, jeg selv er lidt af en **Python** fortaler, har jeg ikke brugt sproget i *full*- eller *web*-stack sammenhæng og derfor ønskede jeg at lave *backend*'en i **Python**.

Til projektet har jeg lavet et min nyt environment med *Anaconda*[25], som min package manager, for at undgå, at projektet løber ind i nogle problemer fra tidligere. Jeg endte dog med at reinstallere det, fordi jeg troede, at der var gået noget galt med miljøet, hvilket vidste sig at være en nedarvingsfejl fra *FastAPI-users' beanie* modul.

2.4.2 FastAPI

FastAPI[26] er både **F**'et og **A**'et i **FARnM**, så vigtig er den for stack'en.

[21] https://en.wikipedia.org/wiki/Structural_type_system
[22] https://en.wikipedia.org/wiki/GRASP_(object-oriented_design)
[23] https://www.infoq.com/articles/microservices-design-ideals/
[24] https://www.mongodb.com/developer/languages/python/farm-stack-fastapi-react-mongodb/
[25] https://www.anaconda.com/download/
[26] https://fastapi.tiangolo.com/

FastAPI er et framework til at bygge *API'er* i *Python* hurtigt, og det er blevet ret populært.

FastAPI er den primære grund til, at jeg gerne ville arbejde med denne stack.

2.4.3 Anvendte Python og FastAPI moduler

Modul og link	Beskrivelse	Anvendelse
Beanie[27]	Asynkron ODM for MongoDB baseret på pydantic	FastAPI-users dependant
FastAPI[28]	API framework	Til at lave API'en. Alt fra fouter, Reponse, exception, json encoding, CORS og Middleware med mere
Motor[29]	MongoDB driver	Sammen med asyncio til at oprette asynkron forbindelse MongoDB
Pymongo[30]	Endnu en MongoDB driver	Til at lave synkrone forbindelser.
uvicorn[31]	Asynchronous Server Gateway Interface web server	Til at kører min API
BSON[32]	Et BSON kodeks uafhængig af MongoDB	Til at generere ObjectId kompatible med MongoDb
datetime[33]	Inkludere en Datetime	Til at sætte dato og klokkeslæt på oprettelser og opdateringer
typing[34]	Type hints	til datamodeller, således de kan opdateresuden alle værdier sættes.
Pydantic [35]	Data validerings type hints	Datavalidering til datamodellerne
FastAPI_users[36]	Registrering og autentifikation til FastAPI	Håndtere bruger registering og autentifikation
HTTPX OAuth[37]	Asynkront OAuth	Til at lave 2faktor autentifikation
os[38]	Indbygget os modul	Til at få 2fa miljøet fra systemet

2.4.4 Sprog: JavaScript

Jeg brugte **JavaScript** som mit frontend sprog. Oprindeligt designet til at være et letanvendeligt *object scripting language* med fokus på HTML[39]. Det blev født, da *Netscape* bad en en scheme udvikler, Brendan Eich, om at skrive et scripting programmeringssprog, hvis syntaks mindede om Java's, som Netscape allerede havde fået implementeret i deres browser *Navigator*[40]. **JavaScript's** navn har ofte været en kilde til forvirring, men dets forbindelse til Java er ikke tættere end denne.

JavaScript er dog et af de, hvis ikke det, mest populære programmeringssprog til webudvikling og var derfor et oplagt valg til at udvikle min *frontend*.

2.4.5 Framework: React Native

Er et open-source UI-framework udviklet og driftet primært af Facebook/Meta[41]. **React Native** er baseret *React*, *facebook's* UI-framework til webudvikling, men målrettet mod at kunne køre "native" på mobile platforme.

[3] https://pypi.org/project/beanie/
[4] https://pypi.org/project/fastapi/
[5] https://pypi.org/project/motor/
[6] https://pypi.org/project/pymongo/
[7] https://pypi.org/project/uvicorn/
[8] https://pypi.org/project/bson/
[9] https://pypi.org/project/DateTime/
[10] https://pypi.org/project/typing/
[11] https://pypi.org/project/pydantic/
[12] https://pypi.org/project/fastapi-users/
[13] https://pypi.org/project/httpx-oauth/
[14] https://www.w3schools.com/python/module_os.asp
[15] https://pypi.org/project/beanie/
[16] https://pypi.org/project/fastapi/
[17] https://pypi.org/project/motor/
[18] https://pypi.org/project/pymongo/
[19] https://pypi.org/project/uvicorn/
[20] https://pypi.org/project/bson/
[21] https://pypi.org/project/DateTime/
[22] https://pypi.org/project/typing/
[23] https://pypi.org/project/pydantic/
[24] https://pypi.org/project/fastapi-users/
[25] https://pypi.org/project/httpx-oauth/
[26] https://www.w3schools.com/python/module_os.asp
[27] https://pypi.org/project/beanie/
[28] https://pypi.org/project/fastapi/
[29] https://pypi.org/project/motor/
[30] https://pypi.org/project/pymongo/
[31] https://pypi.org/project/uvicorn/
[32] https://pypi.org/project/bson/
[33] https://pypi.org/project/DateTime/
[34] https://pypi.org/project/typing/
[35] https://pypi.org/project/pydantic/
[36] https://pypi.org/project/fastapi-users/
[37] https://pypi.org/project/httpx-oauth/
[38] https://www.w3schools.com/python/module_os.asp
[39] https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html
[40] https://web.archive.org/web/20200227184037/https://speakingjs.com/es5/ch04.html
[41] https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

Mit oprindelige valg var *Ionic*, men endte med at fravælge det, da ionic er ret langsomt og, selvom det formentlig er brugbart til at lave et *proof-of-concept* som dette, så tænkte jeg, at det formentlig var bedre bruge et mere industriegnet framework – også selvom jeg havde mindre erfaring.

2.4.6 Anvendte JavaScript og React Native modul

Modul og link	Beskrivelse	Anvendelse
react[42]	Bibliotek til at lave UI	Brugt det til at lave UI
react-native[43]	React Native biblioteket til at lave native UI	Brugt det til at lave UI
axios[44]	"Promise"-baseret http-klient	Brugte det til mine frontend API kald
react-native-modal[45]	Modal komponent, som håndterer pop-ups	Til når der opdateres eller skabes et nyt "task"-element
expo-constants[46]	Giver informationer om konstanter	Til at få højde på statusbaren på android enheder
react-navigation/botto tabs[47]	bottom tab navigator	Bottom-tab navigatiosnbaren.
react-navigation/native	React native's navigations håndtering	native Til at lave min "navigations container"
react-navigation/stack[4	App navigation ved at "stack"'e sider oven på	Til at lave min "stack" navigation

2.4.7 MongoDB

Jeg har arbejdet med **MongoDB**[50] før og føler også, at *NoSQL* database imødekommer projektets behov for at kunne blive udbygget modulært og over længere tid. Det er bare en kæmpe fordel ikke at skulle have defineret en database, hvis man senere beslutter sig for, at produktet skal kunne rumme noget helt andet.

Desuden er **MongoDB**'s *NoSQL* også mere fleksibel og selvom ObliGentle ikke kommer til at blive stort nok til, at jeg for alvor kan gøre brug af *NoSQL*'s skaleringsmuligheder, så er det rart at arbejde med og ville være vigtig, hvis jeg beslutter mig for senere at ville færdigudvikle ObliGentle.

Der er åbenlyse alternativer til *FARM*-stack'en, så som *MEVN*[51] - eller *MERN*[52]-stack. De kunne begge have fungeret godt, og som nævnt før, så skyldes det i højere grad en præference for at bruge Python og FastAPI, end at der er noget i vejen med de andre stack'er. *LAMP*[53] -stack er også oplagt at nævne, men vil helst ikke arbejde i PHP. Et spændende alternativ kunne være *PERN*[54]-stack'en, men ville gerne arbejde med en *NoSQL* database. R*uby on Rails* havde også været en spændende mulighed.

Alt i alt er jeg dog tilfreds med stack'en som den blev. *MEVN* var den anden store mulighed for mig, men med *Python* vandt *FARM* bare for mig.

2.4.8 GitHub

Helt fra starten af projektet har jeg benyttet **Github** til versionering. Jeg brugte VS code's indbyggede værktøjer til at forbinde til **GitHub** det meste af tiden, men da jeg skulle ligge to repositories sammen, så måtte jeg også bruge noget *Git Bash*.

[26] https://www.npmjs.com/package/react
[27] https://www.npmjs.com/package/react-native
[28] https://www.npmjs.com/package/axios
[29] https://www.npmjs.com/package/react-native-modal
[30] https://www.npmjs.com/package/expo-constants
[31] https://www.npmjs.com/package/@react-navigation/bottom-tabs
[32] https://www.npmjs.com/package/@react-navigation/
[33] https://www.npmjs.com/package/@react-navigation/stack
[34] https://www.npmjs.com/package/react
[35] https://www.npmjs.com/package/react-native
[36] https://www.npmjs.com/package/axios
[37] https://www.npmjs.com/package/react-native-modal
[38] https://www.npmjs.com/package/expo-constants
[39] https://www.npmjs.com/package/@react-navigation/bottom-tabs
[40] https://www.npmjs.com/package/@react-navigation/
[41] https://www.npmjs.com/package/@react-navigation/stack
[42] https://www.npmjs.com/package/react
[43] https://www.npmjs.com/package/react-native
[44] https://www.npmjs.com/package/axios
[45] https://www.npmjs.com/package/react-native-modal
[46] https://www.npmjs.com/package/expo-constants
[47] https://www.npmjs.com/package/@react-navigation/bottom-tabs
[48] https://www.npmjs.com/package/@react-navigation/
[49] https://www.npmjs.com/package/@react-navigation/stack
[50] https://www.mongodb.com/try/download/community
[51] **M**ongoDB, **E**xpress.js, **V**ue.JS & **N**ode.js
[52] **M**ongoDB, **E**xpress.js, **R**eact.js & **N**ode.js
[53] **L**inux, **A**pache, **M**ySQL & **P**HP
[54] **P**ostgreSQL, **E**xpress.js, **R**eact.js & **N**ode.js

Det var målet at få brugt det flittigt og det lykkes ikke for mig. Men jeg mistede ikke noget data, så den del er ok.

2.4.9 MongoDB Compass

MongoDB Compas har jeg brugt til at interagere direkte med databasen. Den er ikke fantastisk, men er rar at have til at fejlfinde, teste og gøre ting, som jeg ikke ønsker at skrive ind i programmet.

Jeg brugte **MongoDB Compass** til at oprette databasen og collections samt inspicere databasens indhold.

MongoDB Compass er **MongoDB's** *native GUI*, og jeg vidste, at de havde de begrænsede funktioner, som jeg havde brug for og derfor valgte jeg blot at bruge dem.

2.4.10 OneNote

Til at føre og holde styr på min noter brugte jeg **OneNote**[55]. Det er en fantastisk hjælp at kunne holde styr på tanker om projektet på farten.

Logbøgerne er siden hen blevet skrevet over i LaTeX, men det har været guld værd i arbejdsprocessen.

2.4.11 Swagger UI

Jeg brugte **Swagger UI**[56] til at teste min API.

SwaggerUI kommer som en del af **FastAPI** og kan teste ens **API** fra browseren. Jeg har tidligere brugt *Postman* og har haft gode erfaringer med den.

2.4.12 TexMaker og MikTek

En *opensource* LaTeX-editor (TexMaker[57]) og package-manager (MikTeX[58]). Der er et væld af alternativer, men disse er jeg vandt til og glade for at bruge.

2.4.13 Visual Studio Code

VSC er nærmest en industristandard og den vi har brugt mest på skolen. Den kan godt føles lidt bloatet til tider, men det er klart den, som jeg har mest erfaring med og fordi jeg eksperimenterede med andre ting, ønskede jeg ikke også at gøre det med mit *editor*-valg.

Jeg kunne formentligt med fordel af gjort brug af en fuld IDE, særligt i forhold til *React Native*, men kan simpelthen ikke lide at programmere på den måde. Så for mig er **VSC** en god mellemvej.

[55] https://www.onenote.com/Download
[56] https://swagger.io/tools/swagger-ui/
[57] https://www.xm1math.net/texmaker/
[58] https://miktex.org/

Kapitel 3

Tidsplan

3.1 Estimeret tidsplan

Tidspunkt	Projekt	Dokumentation
13-21/11 Opvarmning Præsprint		Lav *case*, *problemformulering*, *Kravsspecifikation*, *tidsplan*, *mockup* og send dem til godkendelse
13-21/11 Startsprint Sprint 1	Lav API og Test API'en Lav en rudimentær frontend	Begynd begge rapporterne og skitser en overordnet struktur. Skriv første udkast til metode- og teknologiafsnittet
22-28/11 Fremstød Sprint 2	Gør frontend færdig API på nettet og sørg for kan tilgås af programmet Sørg for frontend kan gøres fra både tablet og web	Beskriv løbende program features, til produktrapporten, og erfaringer og udfordringer til procesrapporten
29/11-5/12 Slutspurt Sprint 3	Program testning Ryd op i koden og sikre der er kommenteret tilstrækkeligt Lav en liste over mulige *features* Frasorter fra feature-listen, hvad der ikke kan nås, implementer, hvad der kan	Beskriv testene, features (både kasserede og implementerede)
6-8/9 Dokumentationsr Postsprint	Sidste test af programmeret kører Eksporter filer og sikre dig, at de virker på en anden enhed	Færdiggør og aflever rapporterne

3.2 Realiseret tidsplan

Kapitel 4

indhold

Bilag A

Foreløbig literaturliste

Dette er ikke en begrænsning af fremtidige anvendte ressourcer, men beskrivelse af allerede anvendte ressourcer.

A.1 Internet ressourcer

- atlassian.com
- baeldung.com
- code.visualstudio.com
- christophergs.com
- codevoweb.com
- colorhexa.com
- ctan.org
- devwithdave.co.uk
- geeksforgeeks.org
- github.com
- fastapi.tiangolo.com
- flaticon.com
- ionicframework.com
- merriam-webster.com
- mongodb.com
- multimediedesigneren.dk
- ordbogen.com
- ordnet.dk
- overleaf.com
- panabee.com
- purepng.com

- pythontutorial.net
- scrumguides.org
- shortcut.com
- stackexchange.com
- thefreedictionary.com
- thesaurus.com
- realpython.com
- vuejs.org
- whois.domaintools.com
- wikipedia.org