Report

# Resume

Dele af vores samfund er udviklet sammen med internettet. Nu er det tid til at arbejdsstyrken mestre dette medium. Introducer Converge en freelancing platform skab for a forbedrer salg og køb af arbejdskraft. Med Converge kan en employer direkte vælge den mest kvalificerede freelancer til jobbet, og have en direkte kommunikation under hele udvikling. Converge vil give brugeren sin magt tilbage, og ikke lade beslutninger kun blive bestemt af højt rangerede CEOs.

Bliv din egen chef.

# Abstract

We’ve seen parts of our society evolve with the internet, now is the time for evolving the workforce to master this medium. Enter Converge, a freelancing platform aiming to improve the sale and purchase of work power. With Converge the employer can directly choose the most qualified freelancer for the job and be in close communication during the development of the task. Converge aims to give the power back to its users, and not let business be solely decided by high-ranking CEO’s.

Be your own boss.

## Versionshistorik

# Forord

Denne rapport er produktet af 7. semesters Bachelor projekt for 2 studerende på Aarhus University School of Engineering, der studerer til diplomingeniør i Informations- og Kommunikationsteknologi. Projektet er gennemført med vejledning fra Jesper Michael Kristensen på vintersemestret 2019/2020, som takkes for de gode råd og vejledning i forbindelse med udførelse af projektet. Rapporten er afleveret d. 18. december 2019, og står til eksamination d. 17. januar 2020. Jesper Tørresø takkes for de gode råd som støtte til den tekniske udfoldelse af projektet.

Victor B. Levesen, Philip H. Mikkelsen takkes for at være testpersoner, læse korrektur og støtte. I har været til stor hjælp.

Sammen med rapporten afleveres en tilhørende dokumentation, hvilket går mere i dybden med projektets detaljer. Dokumentationen er i form af en række bilag, bestående af, men ikke afgrænset til: kravspecifikation, arkitektur, design, testspecifikationer, udviklingsmiljø, udrulning og softwarens kildekode og opsætning.

# Læsevejledning

Der vil flere gange refereres til eksterne dokumenter, såvel som dokumenter, der vil kunne findes i dokumentationen. Alle referencer til både interne og eksterne dokumenter vil der henvises til referencelisten sidst i rapporten, og vil refereres med forfatter og udgivelsesår. Den refererede sektion vil være beskrevet på referencen i rapporten, f.eks. “DevOps (Converge-team, 2019, DevOps)”. I eksemplet henvises til et internt dokument, hvor forfatterne er Converge-teamet, hvilket består af medlemmerne af Bachelor gruppen, dokumentet er udgivet i år 2019 og referere til ordet DevOps i dokumentet for begreber, hvilket står i referencelisten bagerst i rapporten.

Interne som eksterne dokumenter vil blive behandlet ens, hvor referencer både kan bestå af artikler, hjemmesider, opslag osv. De interne dokumenter af Bachelor gruppen vil være markeret med et web-link til det opbevaringssted hvor dokumentet findes. Men det afleverede dokumentation kan også bruges. Grunden er at dokumentationen ikke er udgivet andre steder end ved afleveringen, og læseren skal have adgang til de givne dokumenter.

# Indledning

I følgende afsnit vil information omkring Converge projektet blive beskrevet, hvilket er vigtigt for at forstå projektets opbygning. Der ville blive beskrevet hvordan systemet håndtere det grundlæggende problem, som projektet prøver at løse.

## Problemformulering

Dette projekt skal bruges til at finde en måde at bringe arbejdsstyrken ind i det 21. århundrede. I dag har vi elbiler, nanoscience, sociale-media osv. Men det med at finde et arbejde er det samme det altid har været, man søger et arbejde, bliver ansat, fyret eller siger op, og om igen. Med internettet er så meget muligt, og for en hel speciel gruppe mennesker kunne der være en helt anden måde at leve og arbejde på. Svaret er freelancing, freelancing er ikke nyt, eftersom der altid har været brug for kontrakt arbejde. Men via. internettet er det nu muligt at bringe denne arbejdsform til helt nye højder. Men freelancing bringer nogle udfordringer og problemer med sig, noget som Converge-teamet ikke mener bliver løst tilstrækkeligt af eksisterende løsninger. Freelancing over internettet bringer en mur mellem folk, og selvom det er muligt at have mange kunder og være sin egen chef. Så er problemet at folk kun kan interagere kort med andre folk, eller næsten ikke, dette kan give grund til misforståelser og fortrudte projekter, som er endt i en fiasko.

Visionen for Converge er at bringe folk sammen om projekter. Converge vil ligne andre freelancing platforme, men vil have potentialet til så meget mere. Med en moderne app og server struktur, har Converge de bedste forhold for hurtige iterationer, for at rette fokusset mod det brugerne vil have. Converge, fokusere på kortvarigt og langvarigt projektarbejde, ved at tilbyde værktøjer for brugeren. Med Converge er det muligt at have et samarbejde mellem 2 brugere, en freelancer og en employer. Disse brugere arbejder sammen om et projekt hvor de har mulighed for at dele filer, stille spørgsmål på en naturlig og stream linet måde. Converge tilbyder også tekst chat, med inspiration fra Facebook Messenger, og muligheden for Video chat, så spørgsmål og misforståelser kan blive løst så hurtigt som muligt.

Converge er skabt som en hjemmeside rettet mod den almene iværksætter som udbyder (employer) og selvstændige kontrakt arbejder (freelancer), men udsteder også et Api, så firmaer kan bruge platformen på den måde der passer dem bedst. Med Converge er det muligt at oprette projekter til at få lavet både store eller små projekter, hvor betaling og materiale kan byttes på en sikker og nem måde, så brugeren får deres ønskede resultat uden at blive snydt.

## Domæne analyse

Til projektet er der opsat Personaer, som en afbildning af den målgruppe Converge projektet vil fokusere på, og tænker som de primære kunder.

Personerne er henholdsvist opdelt i 2 kategorier, slutbrugere og personale. Slutbrugere er kategoriseret som brugere, der hvor almene funktioner er gældende, dette er funktioner der er delt af de to slags slutbrugere, freelancers og employeers. Til at holde fokus på de forskellige brugere under udviklingen er der brugt personaer, til at give en fiktiv historie til en bruger som kunne være interesseret i systemet. Der er oprettet en til flere personae, for hver aktør og hver type person af målgruppen.

Til den anden kategori er personale, dette er inddelt i aktørerne, developer, supporter and Administrator. Disse aktører repræsenterer det interne personale i Converge-teamet og vil blive behandlet som var det en slutbruger. For flere detaljer for personalet se dokumentationen.

## Løsningsbeskrivelse

Som løsning til problemstillingen er der fremstillet et website (Converge-SPA), der gør det muligt at købe og sælge arbejdskraft. Mellem henholdsvist employer og freelancer. Til websitets koncept er der taget inspiration fra andre lignende sites, såsom Fiverr.com, Freelancer.com, worksome.dk osv. Dog er der med Converge mere fokus på samarbejde og kommunikation end disse platforme. Desuden er Converge fremstillet til at være så fremtidssikkert som muligt, og bruger det nyeste men stadig modne teknologi, for at brugeren får den mest behagelige oplevelse.

En given bruger kan være freelancer, employer eller begge dele. Selve forløbet starter med at en employer opretter et projekt personen eller hvad firmaet ønsker lavet. Til at starte med kan employeren definere en pris og indsætte en beskrivelse, samt uploade nogle filer til at forklare hvad personen gerne vil have lavet.

En freelancer kan derefter vælge at lægge sit bud og en kommentar på dette projekt, og employeren kan derefter vælge at acceptere buddet, eller tage en dialog med freelanceren. Efter at freelanceren er valgt, kan projektet starte, employeren fortæller freelanceren hvad der ønskes lavet, og freelanceren har nu mulighed for at spørge om alle de detaljer der skal til. Dette kunne være enten at bruge de interne projekt samarbejdsværktøjer eller ved at føre en dialog over tekst- eller videochat.

Freelanceren kan nu arbejde på projektet og indlevere sit materiale i iterationer, for at få employerens kommentar og meninger. Hvis employeren siger god for det, så vil freelanceren anmode om betaling i bytte for the materiale personen/firmaet har fremstillet.

Employeren får en anmodning om at betale, hvis det vælges, så vil employeren modtage resultatet af projektet og freelanceren betaling for det arbejde gjort.

Til at klare dette er der brug for funktionalitet til at have brugere, projekter, tekstchat, betaling, osv.

Dette er løst ved at skære projektet op i 2 dele, en webapplikation (Converge-SPA) og en backend (Converge-cluster).

Converge-SPA er det som brugeren ser og interagere med, hvis webgrænsefladen er valgt. Hvis ikke, tilbyder Converge-cluster et API som kan det samme. Converge-cluster er der hvor al logik hører hjemme, det er kernen af produktet og indeholder alle de services der udgøre produktet Converge. Converge-cluster sørger for alt fra web-servere, til databaser, load-balancing, monitorering og meget mere.

## Afgrænsning

I dette afsnit vil de forskellige afgrænsninger til projektet blive præsenteret og beskrevet. Dette indebærer både interne, som eksterne. Samt interne og eksterne krav.

Følgende afsnit er gengivet fra dokumentation.

### Interne afgrænsninger

Interne afgrænsninger er bestemt Converge-teamet, og er nogle af de krav som teamet har sat sig selv for, til udformningen af produktet.

* Hjemmeside udviklet med Google Chrome
* Feature branching er valgt til at samarbejde om kode.
* Der skal være fokus på sikkerhed og stabilitet i produktet.
* Websitet skal være valgfrit at bruge, et API skal udstede samme funktionalitet.

### Eksterne krav

Eksterne krav beskriver nogle af de krav sat fra Converges omgivelser og den opgave stillet til udformningen af Converge produktet.

Nogle af kravene er beskrevet via. den antagelse at det er det der bedømmes på fra vejleder og sensors side.

* Produktet er unikt
* Grænseflader og protokoller beskrevet
* Krav er prioriteret
* Ikke-funktionelle krav er testbare
* Krav afspejler en detaljeret domæneforståelse
* Accepttest skal udføres, både med funktionelle og ikke funktionelle krav
* Testdata er entydigt formuleret (Sekvenser, testdata og forventet resultat)
* Udeståender for fejlede accepttest er beskrevet og en aktionsplan lavet.
* Den valgte arkitektur understøtter projektets mål
* Der argumenteres for designvalg, blandt alternativer
* Der anvendes en struktureret tilgang til fremstillingen af designet.
* Der argumenteres for designvalg
* Implementeringen følger designet
* Der anvendes konfigurationsstyring (feature-branching)
* Integrationstest er anvendt
* Modultest er anvendt
* Projektet er relateret til problemformuleringen
* Problemstillingen er entydig, har en tydelig sammenhængen med krav og understøttes evt. af delmål samt en plan for undersøgelse af disse.
* Den valgte arbejdsmetode er valgt konsekvent
* Resultater sammenlignes med relaterende produkter.

## Arbejdsområder

Dette projekt har været bygget op med fokus på egenskaber (features), og det er opstillet for at være så nemt som muligt at skifte fra at lave en service til en anden, og til at lave en egenskab for Converge-spa. Derfor betegnes det som om der er mange forskellige egenskaber for systemet. Til fordelingen vil der blive skrevet et primært medlem på en feature med ét, ettal (1.) og sekundær (2.) hvis personen har givet støtte, eller har lavet en mindre ændring.

Opdelingen er for webapplikationen Converge-SPA, Converge-cluster, DevOps Tooling og Proces. Proces har været lidt flydende, eftersom der kun er 2 medlemmer af gruppen, men teamet synes stadig at det var vigtigt at føre en struktureret arbejdsproces. Derfor har der været en fast projektleder og scrum master.

Arbejdsområder for Converge-team opdelt efter egenskaber

|  |  |  |
| --- | --- | --- |
| **Egenskab** | **Kasper** | **Samir** |
|  |  |  |
| Arkitektur | 1. | 1. |
| Design | 1. | 1. |
| Interface Design | 1. | 2. |
| Login & Registrering | 1. |  |
| Bidding | 1. |  |
| Collaboration | 1. |  |
| Projects | 1. |  |
| Chat | 2. | 1. |
| Payment | 1. |  |
| Projects | 1. | 2. |
| Profiles | 1. |  |
| Search | 1. | 1. |
| Unit tests | 2. | 1. |
| GUI tests | 1. | 2. |
|  |  |  |
| Arkitektur | 1. |  |
| Design | 1. |  |
| Audit | 1. |  |
| Authentication | 1. | 2. |
| Biddings | 1. | 2. |
| Broker | 1. | 2. |
| Chat | 1. | 1. |
| Collaboration | 1. | 2. |
| Files | 1. |  |
| Payments | 1. |  |
| Profiles | 1. | 2. |
| Projects | 1. | 2. |
| Users | 1. | 2. |
| Unit tests | 2. | 1. |
| Integrationtest | 2. | 1. |
| Systemtests | 2. | 1. |
|  |  |  |
| AutoAccepttest | 1. | 2. |
| Continuous Integration | 1. |  |
| Continuous Delivery | 1. |  |
| Monitoring | 1. |  |
| Tracing | 1. |  |
| Logging | 1. |  |
| Proxy | 1. |  |
|  |  |  |
| Projektleder | 1. |  |
| Scrum Master | 1. |  |
| Mødeleder | 1. | 2. |
| Referent | 2. | 1. |

# Krav

I dette afsnit vil rammerne for projektet gennemgås. Dette gøres ved at formulere problemstillingen som krav og user stories.

## Krav til systemet

At opstille solide krav for systemet på baggrund af problemformuleringen er den vigtigste del af udviklingsprocessen, uden denne fase, vil udviklingen hurtigt udvikle skævt i forhold til kundens oprindelige problemstillingen. Derfor har det været vigtigt at opstille krav inden projektets udarbejdelse gik i gang. Disse krav er opstillet af to typer, funktionelle og ikke-funktionelle. Hvor funktionelle krav er baseret på handling og er derved kategoriseret under aktører, personaer og beskrevet med user stories. Den anden type, ikke-funktionelle beskriver tekniske krav, såsom sikkerhed, anvendelse, vedligeholdelse osv.

De funktionelle krav er opsat under Epics og Initiatives, men først og fremmest er de prioriteret efter hvad problemstillingen ligger mest vægt på. Dette er gjort efter MoSCoW-metoden, hvilket bruger denne formulering, Must (skal), Should (burde), Could (kunne) og Won’t (ikke) have. De vigtigste for et MVP (Minimal Viable Product) er under Must hvilket følger lean udvikling.

**En bruger skal være en freelancer og/eller employer**  
For en bruger skal det være muligt at kunne gøre handlinger som enten en freelancer og/eller employer, dette relatere direkte til formålet med problemstillingen. Formålet med applikationen er at skabe en ny arbejdsbevægelse på internettet, derfor kræves det at en bruger skal være en freelancer og/eller employer.

**En employer skal kunne udgive projekter**  
En employer skal have muligheden for at kunne give forespørgsler for arbejdskraft, i dette tilfælde kaldes det projekter, og er den primære interaktionsform for Converge.

**En freelancer skal kunne tage projekter og udføre dem**  
Den nye arbejdskraft, freelancer skal kunne tage forespørgsler for en employer og udføre dem, derved tjene penge i bytte for ekspertise.

**En bruger skal kunne modtage og overføre penge**  
For at kunne leve af at bruge denne service, kræves det kan penge kan betales til systemet, og sikkert overført til andre brugere, og derefter udbetales.

**En bruger skal kunne chatte med en anden bruger**  
Fordi interaktion er så vigtig del af problemstillingen, er det nødvendigt at brugeren har mulighed for at snakke med andre brugere. Om det er tekst eller video chat, så for at få essensen skal brugeren som minimum kunne kommunikere med andre brugere over tekst chat.

**Systemet skal have en web portal**  
For at brugere og ikke kun større firmaer skal kunne bruge systemet, så kræves det at der er en web app som brugeren kan interagere med. Det er også grænsefladen for systemet, så det er vigtigt at den er let at bruge, samt give de samme muligheder som Converge Api.

**En skal kunne registrere sig og kunne logge ind**  
For at en bruger kan beskytte sig selv kræves det at personen kan logges ind.

Kravene der ikke er et “Must”, kan findes i dokumentationen

## Funktionelle krav

Funktionelle krav er krav til systemet, der relaterer og beskriver de funktionelle elementer af systemet, dette er f.eks. adfærd. Til dette er der brugt agil software planlægning til at lave dette. Agil software planlægning bruger Initiatives, Epics og User Stories. Initiatives er samlingen for hele ideen bag denne nye problemstilling. I dette tilfælde er det at udgive en platform for freelancers. Epics er en samling af user stories og samler adfærd og ønsker omkring noget funktionalitet. I dette tilfælde er der oprettet en række Epics til at beskrive kravene opstillet i MoSCoW analysen, og de skal være tilstrækkelige til at løse det oprindelige problem fra problemstillingen. User stories beskriver et stykke adfærd relaterende til en funktionalitet og et ønske. Samlingen af User stories gør en funktionalitet.

Til udarbejdelsen af Epics, er der blevet taget udgangspunkt i MoSCoW analysen beskrevet i afsnit [6.1](#sec:system-requirements).

Epics:

* **E-01**: Login & Registering - tilgå systemet
* **E-02**: Employer flow - udbyder
* **E-03**: Freelancer flow - arbejdskraft
* **E-04**: Video chat - visuel kommunikation
* **E-05**: Text chat - tekstuel kommunikation
* **E-06**: Search - Søge funktionalitet
* **E-07**: DevOps Tooling - overblik og vedligeholdelse

Disse Epics beskriver hver især noget funktionalitet der retter sig mod problemstillingen. Grunden til at Epic 07 er med, er fordi udviklere og operations personale skal behandles med omhu, når produktet udvikles, det skal være muligt for personalet altid at have overblik og kunne vedligeholde applikationen. Det komplette overblik over user stories og Epics kan findes i dokumentation.

## Ikke-funktionelle krav

I systemet er der opstillet en række ikke-funktionelle krav, hvilket er inddelt i kategorier.

* Anvendelighed
* Sikkerhed
* Vedligeholdelse

De tre valgte kategorier er valgt fordi de understøtter brugeren og personalets oplevelse med systemet. Der er lagt vægt på sikkerhed og vedligeholdelse i forhold til personalet, og Anvendelighed i forhold til slutbrugeren. De ikke-funktionelle krav er i dokumentation under.

# Metode og proces

I dette afsnit vil processen bag projektets udførelsen blive beskrevet, i dette afsnit vil projektgruppens interne proces blive beskrevet, der bliver uddybet hvordan projektgruppen har arbejdet med Agil udvikling, scrum, arbejdsmetoder osv. Hele processen er beskrevet i detaljer i procesbeskrivelsen.

## Projektstyring

Det blev besluttet i starten af projektet, at gruppen skulle have en agil udviklingsproces, derfor blev der fastsat nogle roller, såsom projektleder og Scrum Master. Projektlederen har taget sig af alle de administrative opgaver, som f.eks. planlægning og styring af møder. Denne person har også haft ansvaret for kontakten til gruppens vejleder, og eksterne parter, såsom testpersoner. Det har været projektlederens ansvar at have overblik over udviklingen, samt at holde styr på tidsplanen og de forskellige deadlines. Projektledelsen har dog ikke taget beslutninger selv, da gruppen kun har to medlemmer, alle beslutninger har været taget fælles, men det har været projektlederens ansvar at have overblik og kontakt.

### Scrum

Der er i projektet benyttet Scrum som led i en agil udviklingsproces. Scrum er et agilt framework, der kan bruges til at holde overblik over udviklingsforløbet for et projekt. Med Scrum er det muligt at estimere ca. hvor lang tid en funktionalitet vil tage at fremstille, samt holde overblik over hvem laver hvad, samt i hvilket stadie de forskellige opgaver er i. Scrum har nogle ceremonier der indgår i at medlemmerne af gruppen er blokeret, lå lidt som muligt og kan arbejde sammen, hvor muligt.

Alle medlemmer af Scrum bruger Sprint boardet som deres primære opslagstavle, på denne tavle er der nogle kolonner til hvert stadie en opgave kan være i, fra backlog, todo, in progress, blocked, review, closed. I starten af et sprint vil opgaver være placeret i todo, efter de opgaver der er planlagt, er sat ind så arbejdsmængden passer med den givne sprintlængde. Resten giver sig selv. I gruppen blev et sprint på 2 uger brugt, dette er for at medlemmerne kunne nå at få færdiggjort en funktionalitet i dette tidsrum, gruppen besluttede tidligt at dette var en god længde. En opgave i Scrum er defineret som en task eller story og indeholder noget nyttigt information, såsom hvad der skal ske, hvilken user story eller arbejdsopgave den tilhøre samt hvor meget tid det regnes med at tage. Efter et par uger, er det muligt for gruppen at estimere hvor meget de kan nå på et sprint. Dette er specielt nyttigt til at estimere hvor meget der kan blive lavet, og hvornår deadlines kan sættes.

I Scrum er der en række forskellige roller. En af disse roller er en Scrum Master. Hvis primære opgave er at styre nogle af de møder der tit bruges til Scrum, som estimering, planning, daily scrum meeting osv. Samtidig med at holde styr på at Scrum boardet er i fin stand og at det er klart til de næste sprints. I dette tilfælde var det projektlederen som også tog rollen som Scrum Master, da der var en glidende overgang med disse.

En anden rolle i Scrum er Product Owner, hvis primære opgave er at vedligeholde scrum backlog, så opgaver er klar til estimering ved refinement og planning. En product owners vigtigste opgave er at kategorisere opgaver efter prioritet, og har normalt kontakt med kunden for at formidle hvad udviklere arbejder på og hvad der er vigtigst for kunden. I dette tilfælde var hele gruppen Product owners, det har været alle medlemmer af gruppen der har opdateret backloggen med nye opgaver samt prioriteringen af disse.

Den sidste rolle er udviklingsteamet, som alle gruppens medlemmer har været en del af. Inklusivt Scrum Masteren som har været et fungerende medlem. Udviklingsteamets fokus ligger på at få lavet de opgaver der ligger i todo til closed. Deres opgave er også at deltage i Scrum møder og give deres ekspertise til at give opgaver et estimat, noget der måske er for teknisk for en Product Owner.

### Møder

Der har i gruppen været forskellige typer møder, herunder interne og eksterne møder. De interne møder har været imellem gruppens medlemmer, og har været forbundet med Scrum, til f.eks. refinement, planning eller retrospektives. Normalt har der også været arbejdsmøder, hvor gruppen har siddet og arbejdet i samme lokale for at sparre. De eksterne møder er møder hvor andre en gruppens medlemmer har deltaget, om det er vejleder eller testpersoner. Gruppen har ugentligt haft et vejledermøde med gruppens vejleder, hvor et kort resume, og nogle gange en Sprint demo blevet vist, samt spørgsmål stillet og vejleder opdateres med status på projektet.

Testpersonmøder er også blevet holdt, Projektlederen har flere gange haft møder med eksterne parter, for at få deres indblik og ekspertise på hvordan en bruger vil bruge systemet. I løbet af hele projektet har testpersoner deltaget undervejs i flere stadier af produktets udvikling. Disse møder har tit været på tidspunkter hvor den givne testperson kunne, så har været om aftenen i hverdage, eller i weekenden. Disse har som regel varet i 1-2 timer og har konkluderet i en mundtlig præsentation for gruppen, såvel som et skriftligt dokument, med de vigtigste punkter fra testpersonen.

Til alle vejledermøder har der været en Mødeleder samt en referent hvis nødvendigt. Det har været mødelederens ansvar at få planlagt mødet og udsendt en mødeindkaldelse til de personer, der skulle deltage. Gruppens projektleder tog naturligt denne rolle. Mødereferater og mødeindkaldelser kan findes i dokumentationen.

## Tidsplan

Til planlægningen af projekt blev der lavet en tidsplan i starten af projektet. Tidsplanen blev derefter justeret efter prioritering og deadlines. Tidsplanen er inddelt i to dele, udformning af produkt, og fremstillingen af dokumentation samt rapport skrivning.

## Roadmap

Roadmap er en plan over hvor lang de forskellige Epics er fra er være færdige, dette er ikke det samme som tidsplanen, men viser det resterende arbejde tilbage for de forskellige parter.

## Arbejdsmetode

I udviklingen af projektet har der for gruppen, været et par faktorer, som har været vigtige omkring den måde der blev arbejdet på.

Udviklingen har blandt andet været baseret på krav og user stories, og har derfor været udført vandret. Dvs. at en opgave både kunne været til Converge-SPA samt Converge-cluster. Dette har givet muligheden for alle gruppens medlemmer at have kendskab til både applikationen såvel som serveren.

Vigtigst af alt har gruppen fulgt arbejdsgange som man gør i en virksomhed. Derfor har den agile udviklings proces taget inspiration fra arbejdsgange fra Danske Bank Team Crypto. Grunden til dette er at have et solidt fundament for udviklingen og den mest afprøvede taktik til applikations udvikling, samt relationen mellem udviklingen og problemløsning.

Med en agil tankegang var det naturlige at adoptere en DevOps arbejdsgang, hvor gruppens medlemmer er tværfaglige, og gør alt fra opsætning, til udvikling og vedligeholdelse. Dette har gjort at med en agil udvikling og DevOps har man kunnet udnytte moderne værktøjer til at sikre den mest robuste og hurtigst iterative proces.

Dette har betydet at teamet har kunnet fokusere på hurtige iterationer med kvalitet som fokus. De værktøjer brugt som nævnt før at har givet muligheden for dette. Dette er især udført gennem test, her af: unittests, integrationstest, systemtest og auto accepttests.

## Værktøjer

Under udarbejdelsen af projektet er der blevet benyttet en række værktøjer. Værktøjerne består af software, der kan fungere som en hjælp til planlægning og administrationsopgaver. Derudover er der benyttet værktøjer, der har hjulpet udviklingen af softwaren til projektet, samt kvaliteten af udviklingsprocessen. Der er også brugt diverse værktøjer til at vedligeholde et sundt produktionsmiljø.

### Git/GitHub/GitLab

Git er et værktøj som gør det muligt at versionere sin applikationskode. Det gør det muligt at gå tilbage i tiden for en kode base (Repository) og udrulle det kode. GitOps er en praksis der gør at alt ens infrastruktur også er under versionering, dette er især tilfældet når Git bliver brugt til mere end bare applikations kode.

GitHub er en Repository Manager og er det sted Converge får hostet sit repository. GitHub tillader også andre funktioner ovenpå Git, såsom pullrequests og issues. Git er beregnet til at man kan arbejde parallelt, og det har Converge-teamet også brugt det til. Med Git kan man feature branche og med GitHub er det især en styrke, for så kan der foregå reviews og en Continuous Integration proces følges. Hvor koden den funktion under udvikling bliver afprøvet og testet inden den bliver flettet ind i masteren (produktion).

Til udviklingen af dette projekt er GitHub Flow brugt, som beskrevet tidligere går det ud på at oprette kort levede feature branches som gennemgår et review og bliver flettet ind i master. GitHub flow sikre at master branchen er nogenlunde stabil, samtidig med at ændringer hurtigt kan blive flettet ind. I forhold til et andet flow som Git Flow, som kræver flere parallelle branches og miljøer, så var GitHub flow er bedre bud til nogle relative små repositories, hvor hastighed er nøglen.

GitLab gør det muligt at køre kode fra en given Git branch. Med GitLab er det muligt at koble ens Git repository sammen med andre ressourcer, i dette tilfælde bliver Docker, Helm, Kubernetes brugt som ressourcer, og gør at kode kan blive udrullet automatisk, når koden er verificeret og testet.

### Docker/Kubernetes/Helm

Disse værktøjer bliver brugt til udrulning og produktionsmiljø. Med Docker er det muligt at pakke alt ens applikation skal bruge for at køre. Kubernetes bruges til at køre ens applikation og vedligeholde den automatisk. Helm bruges til at udrulle Docker på Kubernetes på en simpel og sikker måde, så det er muligt at rulle tilbage til en tidligere version, hvis noget går galt. Mere information om hvordan disse værktøjer arbejder sammen, kan ses i applikations udvikling i dokumentationen.

### Jaeger, ELK, Prometheus

Disse værktøjer er brugt til at monitorere produktionsmiljøet, men har hver deres fokus. Jaeger bruges specifikt til at spore forskellige kald, og dets relation mellem forskellige services. Dette har været brugbart for at kunne debugge produktionsmiljøet. ELK står for Elasticsearch, Logstash og Kibana og er brugt til at kunne danne statistikker for selve miljøet, det er et produkt man selv skal definere, men kan blive især kraftfuld, når det er bundet op på andre værktøjer. Dette har været brugt til at se antal fejl og hastigheder for de forskellige services. Prometheus er brugt til at monitorerer helbredet af ens miljø, i dette tilfælde har det været brugt til at se status på de forskellige services, antallet af dem og om de er oppe eller nede.

### Traefik

Traefik er den reverse-proxy brugt i Converge-cluster. En reverse-proxy er brugt til at route kald fra en ip-adresse til services. Med Traefik kan man med Converge-cluster hurtigt og nemt alle end points til lige præcis den service man ønsker. F.eks. users-service.api.converge-app.net som peger på users-service i clusteret.

### NextJS/Now

NextJS er den applikationsserver hvilket driver Converge-SPA. Med Now kan NextJS nemt udrulles på Zeits servere. Dvs. at Converge-SPA er udrullet på en anden server end Converge-cluster. Grunden til at dette har været valgt er for at udnytte de ekstra funktioner som Now giver brugt i konjunktion med NextJS. Med Now kan NextJS skalere næsten uendeligt.

### Stripe

Stripe er brugt til at håndtere online betalinger, og er brugt som et medium imellem de forskellige brugere i systemet. Stripe indeholder al den logik der skal til for at overføre penge mellem brugere, så Converge-teamet har valgt at bruge dette, da det giver muligheden for et rigtigt produktions klart værktøj for betalinger.

### MongoDB

Selvom Converge-cluster har muligheden for at understøtte forskellige databaser, så har gruppen valgt at bruge MongoDB. Mongodb er en nosql database til almen brug og er perfekt til det team ønsker. Mongodb kan bruges til at erstatte en traditionel database, og kan i de fleste situationer skalere bedre end en normal sql database.

### LaTeX og PlantUML

LaTeX er anvendt til at skrive dokumentation og fungere perfekt til at holde styr på referencer og bilag. PlantUML er brugt til at lave diagrammer og fungere godt sammen med LaTeX.

### Google Ressourcer

Converge bruger diverse Google ressourcer som Cloud DNS og Google Kubernetes Engine. Disse er valgt pga. deres integration med hinanden samt kendskab fra teamet. Der er diverse alternativer som Amazon Web Services eller Microsoft Azure. Men med en gratis startkapital fra Google var det nemt og hurtigt at starte med disse.

# Teknologiundersøgelse

I starten af dette projekt blev der udarbejdet en række teknologiundersøgelser, for at se hvilke muligheder der fandtes for at nå det mål der var fastsat til projektet. Efter en smule research, blev der fundet frem til hvilken teknologier der skulle bruges for at opnå målet med projektet. Projektet omhandler en Webapplikation og en applikationsserver, derfor skulle der specifik findes teknologier der egnede sig til at udviklingen af disse.

## React

Til at udvikle klientapplikationen er der blevet brugt udviklingsplatform React. React er et JavaScript-bibliotek til opbygning af brugergrænseflader. Grunden til at valget faldt over React, er fordi der skulle i dette projekt udvikles et dynamisk website. Derudover er det nemt og lære også er det samtidig enkelt og forstå. Den komponentbaserede tilgang, veldefinerede livscyklus og brug af bare almindelig JavaScript gør React meget let at lære, opbygge en webapplikation og understøtte den. Derudover giver React mulighed for at repræsentere hierarkiske data i en træstruktur, hvilket repræsenterer komponent-baseret design.

## ASP.NET Core

Til at udvikle en dynamisk applikation, er der behov for at holde styrer på interaktionerne på klientapplikationen, derfor blev der besluttet at der var behov for en server. I gruppen blev der gjort mange overvejelser over hvilken platform serveren skulle udvikles i, dog faldt valget over at serveren skulle udvikles i ASP.NET. Da det opfyldt de behov og funktionalitet der var behov for i dette projekt. ASP.NET er en webudviklingsplatform, der giver en omfattende softwareinfrastruktur og forskellige tjenester, der kræves for at opbygge robuste webapplikationer. Der er flere gode grunde til at bruge ASP.NET, når man udvikler webapplikationer. Høj hastighed, lave omkostninger og enorm sprogstøtte. ASP.NET er indbygget i det velkendte Windows-servermiljø, hvilket kræver mindre opsætning og konfiguration end andre webudviklingsplatforme, der skal installeres og konfigureres separat. ASP.NETs gør det let at finde online ressourcer. Den primær grund for valget, er at ASP.NET er et framework til C# og det var et ønske fra gruppens side at det skulle være indenfor programmeringssprog C#. Derudover var gruppen enige om at det skulle være nemt og udvikle services. På denne måde kunne projektets behov opfyldes.

# Arkitektur

I dette afsnit vil der blive gennemgået en overordnet arkitektur for hele systemet. Afsnittet starter med en gennemgang på systemniveau og arbejder sig ud til de individuelle moduler såsom klientapplikation og server.

## Systemarkitektur

På nedstående figur ses systemarkitekturen, som er repræsenteret i form af en domænemodel over systemet Converge.

Figur viser de krav systemet har, med kravene fra problemformuleringen taget i betragtning. Kravspecifikation og domænemodellen er brugt til at finde frem til en række funktionaliteter og hvordan de er afhængige af hinanden. Dette medfør til at hvordan håndtering af de forskellige funktionaliteter sker, såsom: brugeren, login, signup, dashboard, chat, indstillinger, portfolio, kategorier, betaling, søgning og filer. Ud fra disse informationer er der blevet udarbejdet et komponentdiagram over komponenter som ses på nedstående figur, dette er dog en konceptuel betegnelse men viser et overblik over hvordan de forskellige elementer i systemet er koblet sammen og interagerer.

Figur viser et komponentdiagram over det aktuelle system under udvikling i forskellige høje funktionalitetsniveauer. Hver komponent er ansvarlig for et klart mål inden for hele systemet og interagerer kun med andre væsentlige elementer på et behov-til-kendskabsbasis. Derudover kan vi se ud fra, at præsentations komponenterne ligger i en samlet pakke og grunden til at de gør det er, fordi det er her brugeren interagerer.

Disse funktionaliteter ligger op til at benytte model-view-controller (MVC) arkitekturen, der står for model, view og controller. Model repræsenterer business logik, view viser brugergrænseflade og controller, som er med til at håndterer brugeranmodningen. I dette tilfælde er MVC brugt på flere måder, både i vores services og med hele vores system. På serviceniveau bruger vi det i ASP.NET Core, med controllers, models og views (json/swagger). På system niveau, bruger vi det, som at applikations services er controllers, MongoDB som model og React/NextJS som View.

## Klientapplikation

Arkitekturen for klientapplikationen er udarbejdet med domænemodel og user stories i tankerne. Applikationen er udviklet som en SPA (Single Page Application), der overholder slutbrugerens ønske om en hurtig og brugervenlig applikation, dette er specielt vigtigt eftersom produktet er rettet mod professionelle. Herudover fungerer klientapplikationen, som grænsefladen for serveren og grænsefladen for de brugere der benytter applikationen.

Klientapplikationen følger Komponent-baseret design, hvilket maksimerer kode genbrug, da mindre komponenter let kan genbruges andre steder. Til at håndtere webapplikationens tilstand, anvendes en ekstern store, som giver muligheden for at dele data mellem de forskellige komponenter, dette vælges på den baggrund at de forskellige komponenter skal kunne kommunikere på en flydende og sammenhængende måde.

## Server applikation

Sammenhængen mellem Webapplikationen og Web-serveren kan beskrives som en MVC-løsning, hvor Webapplikationen fungere som view og Web-serveren som Model-Controller. Dog er dette en grov simplificering af interaktionen, men beskriver forholdet mellem de forskellige parter. Webapplikationen skal kommunikere med Web-serveren, som ikke kun er en enkelt server, men en samling af dem som arbejder parallelt, kaldet et cluster. Dette cluster fungerer som en web-server, og Webapplikationen ved ikke bedre. Dette er valgt for at give muligheden for at arbejde parallelt på de forskellige services, samt at fremstille serveren efter Domæne-Drevet-Design (DDD), hvilket grupperer funktionalitet efter relevans for forretningsdomænet: krav og kontekst.

Web-serveren har en primær funktion: At behandle anmodninger med passende forretningslogik. Adgangen til denne forretningslogik foregår igennem forskellige adresser på Web-serveren. Disse adresser udgår den udstillede Web-server API. Det skal siges at Web-serveren og Converge-Cluster er andet til samme formål, dog hvor Web-serveren beskriver selve ideen fra et overbliks perspektiv, og Converge-Cluster for det egentlige produkt.

Derudover vil Web-serveren ved behandling af anmodninger gemme resultater i et permitteringslag, enten som filer eller rækker i en database. Denne information skal kunne hentes til den funktion der har brugt for det, og som har adgang. De forskellige services (selvstændige applikationer) i Web-serveren, så er der lag: Et samlingslag, Et præsentationslag (API), et forretningslag (forretningslogik), og et dataadgangslag. Disse tre lag placeres i en teknologisk sammenhæng, hvor ASP.NET Core platformen er valgt. Denne platform har programmerbar routing, der samler alle kald til de individuelle services.

Resten af afsnittet indeholder uddrag fra dokumentation.

**Samlingslaget** samler de underlæggende applikationer under en udgang, dette gør det muligt for Web-appen at nemt kunne skifte mellem de forskellige services og end points. Samlingslaget må ikke indeholde forretningslogik, og skal bare beskytte de bagvedlæggende services.

**Præsentationslaget** præsenterer et JSON API til omverdenen. Dette API forventes at modtage JSON indpakket data ved anmodninger, og ingen data ved forespørgsler. Præsentationslaget er tyndt og indeholder kun lige nok data til at verificere indholdet, samt at verificere klienters anmodningsrettigheder. Det forventes at hvis en anmodning er valideret, vil den blive sendt videre til forretningslaget, hvilket kan håndtere dette. Alt i alt vil det sige at data kan modtages og returneres i dette lag, men kun det overfladiske er valideret, resten er bestemt i forretningslaget.

**Forretningslaget** indeholder services, der agerer på klienters forespørgsler i systemets domæne, her bliver forespørgsler håndteret baseret på forretningslogik. Dette kan f.eks. være kommunikation til andre services, for at få deres ressourcer eller videresende begivenheder til interesserede parter.

**Dataadgangslaget** er bygget op omkring 2 forskellige services, enten en Http klient eller en databaseadgangsklient. Til en httpklient er der brugt et fælles bibliotek der kan tilgå systemets ressourcer, dette gør at to forskellige services kan tale sammen, selvom man skriver C# kode. Til en databaseadgangsklient er der brugt repositories, ved et repository beskriver man hvordan databasen skal tilgås. Repositories udgiver en grænseflade som er implementeret med den klient kode der skal til for at kommunikere med database. I dette tilfælde er MongoDBClient brugt, pga. databasetypen som er MongoDB.

Server applikationen skal agere som en fælles front, men have forskellige komponenter i maven. Nogle som præsenterer et nemt at bruge grænseflade, og andre lidt mere komplekse som bruges inde i selve systemet. I dette tilfælde kan det f.eks. være til Users Service. Hvilket er en af byggestenene for systemet, det beskriver noget om brugeren og indeholder et id som bruges i næsten alle andre services. Users service bliver f.eks. brugt af authentication-service som skal udstede et API til registrering og login. Dvs. at Users service primært er længere nede i hierarkiet end authentication-service.

# Design

I dette afsnit vil der være en gennemgang af projektets individuelle moduler, samt en beskrivelse af de overvejelser og valg der er blevet taget i forhold til Converge platformen.

## Converge SPA

Dette afsnit vil indeholde en beskrivelse af designet for Converge applikationen. Der kan læses mere uddybende om dette i afsnit i dokumentation.

Converge-SPA er implementeringen af tidligere nævnt Webapplikation og er fremstillet ved rangerer komponenter efter de forskellige domæner fra domænemodellen, her kan ses i pakke-diagrammet, hvordan de forskellige komponenter er grupperet, samt I hvilket hierarki de hører til.

For at tilbyde brugeren flere forskellige sider, er der brugt en speciel komponent kaldet en page. Pagen er roden for en vis funktionalitet, om det er til login, kollaboration osv. Under denne page, bruges der et layout, hvilket er en opsætning der går igen side for side. Denne opsætning viser f.eks. en navigationsbar til brugeren, så brugeren nemt kan navigere mellem de forskellige funktionaliteter. Selve applikationen er delt op i to parter, et hvor brugeren er logget ind og et hvor brugeren ikke er. Derfor er der definere to forskellige opsætninger, et hvor brugeren har adgang til login og registrering, og et hvor brugeren har adgang til bruger specifikt funktionalitet.

Til at styre tilstanden i applikationen er der brugt et Flux mønster, hvilket definerer en ekstern database i applikationen til at persistere brugerens adfærd og data mellem de forskellige sider. Det giver f.eks. muligheden for at brugeren dynamisk kan indlæse data fra web-serveren uden bruger oplevelsen bliver begrænset at bruge sådan et mønster er ikke nødvendigt for fuldførelsen af Converge-SPA, men gør koden lettere at vedligeholde, og undgå et problem der opstår ved komponent-baseret design kaldet event-bubbling.

Til at få adgang til Converge-cluster er der brugt web-browserens fetch Api, for at give adgang til dette Api er der brugt Axios, hvilket tilbyder et asynkront request, response model. Sammen med Redux (implementering af Flux), gør det at brugerens grænseflade dynamisk kan opdateres via. Asynkrone metoder. Brugerens brugeroplevelse bliver altså ikke låst mens data fra Converge-cluster hentes, eller opdateres.

Til at tilbyde brugeren en responsiv oplevelse, så Converge-SPA ser pænt ud og fungere på både pc, tablet og smartphone er Material-UI anvendt, hvilket implementerer Googles Material Design standarder.

## Converge-Cluster

I dette afsnit beskrives designet af Web-serveren eller nærmere implementeringen Converge-cluster. Afsnittet præsenterer den overordnede opbygning af Converge-cluster efterfulgt af en kort beskrivelse af de individuelle servicer, der udgør Converge-cluster. For dybdegående detaljer henvises til dokumentation.

Komponentdiagram for opbygningen af Converge-cluster

Komponentdiagram for opbygningen af Converge-cluster

På figur  [10.1](#fig:component-converge-cluster) vises et komponentdiagram over Converge-clusters overordnede struktur. Hver komponent har sin egen tilhørende database og applikations server. Hver database er MongoDB og hver applikations run time er ASP.NET Core.

Disse services kan kalde hinanden og er tilgængelige igennem Samlingslaget præsenteret i arkitekturen for Web-serveren.

Hver service har en lignede opbygning, i de grænseflader og pakker de har. Men har forskellige klasser alt efter hvilket domæne der er tale om. F.eks. users-service, har en Users Controller til at agere som præsentationslag, et Users Service til at fungere som forretningslag, og Users Repository som fungere som dataadgangslag.

Konceptuelt klassediagram for en service

Konceptuelt klassediagram for en service

På figur [10.2](#fig:conceptual-class-service) kan det bemærkes at opbygningen for en service passer med eksemplet nævnt tidligere med users-service. I de følgende afsnit vil de gængse elementer for den konceptuelle klasse blive beskrevet.

### Startup

Startup indeholder opsætningen for ASP.NET Core konfigurationen. Startup er den klasse der beskriver hvordan applikationen skal startes op, og hvordan hvert kald skal håndteres. Dette sker henholdsvist i to metoder:

* **Configure** opsætter anmodningspipeline og bliver kørt for hver anmodning.
* **ConfigureServices** opsætter de komponenter hver anmodning skal have adgang til, om det er til database eller autentifikation osv.

Startup er den klasse som udvikleren manipulerer til at konfigurere ASP.NET Core. Ved compile-time bliver frameworkets vigtigste dele opsat, såsom routing og initialisere af diverse klasser såsom services.

### Controller

Controller er grænsefladen mellem omverdenen og forretningslaget, og omhandler som regel logik til at omdanne klientdomænet til systemdomænet, samt at validere indholdet af anmodningen. Controlleren sørger for at klienten får den bedste oplevelse ved at sende data tilbage, om det var en succesfuld anmodning eller en fejl, så håndtere Controlleren dette. Samtidig konfigureres Controlleren også til at bestemme hvem der skal have adgang til den gængse ressource, dette gøres ved at validere en JWT-token og eventuelt en ressources tilknyttede bruger id.

### Service

En service er forretningslaget og bestemmer flowet for en anmodning, om hvordan anmodning skal lykkedes og i hvilken rækkefølge, eller hvordan det skal fejle, i et fejlscenarie. Servicen har adgang til både Http Klienter og Database Repositories til adgang til enten andre services, eller til den tilknyttede database.

### HttpClient

En HttpClient bestemmer hvordan man fra C# kan tilgå et Web API med JSON fra en service til en anden. Dette gøres ved at kalde et API på samme måde som f.eks. Web Applikationen Converge-SPA ville, dette sker dog indenfor clusteret så dataene er ikke nødvendigvis beskyttet af TLS (HTTPS). Dette er valgt grundet kompleksitet og performance.

### Repository

Et Repository er en klasse der har adgang til en Database eller anden persisteringstype, og har et interface som det udsteder som grænseflade, men en implementering der afhænger af den type database brugt. Dvs. at grænsefladen er uafhængigt af selve databasen, men selve implementeringen ikke er. Dette gør det muligt at instantiere dette lag hvor nødvendigt, f.eks. ved unit eller integrations tests.

Disse er bare nogle af klasser brugt, men beskriver den generelle struktur for applikationen.

## Services

I de følgende afsnit vil de forskellige services blive beskrevet i alfabetisk rækkefølge.

### Audit

Audit service er en service der indeholder et paper-trail for brugerens adfærd. Dvs. at hvis en bruger initierer en betaling, vil det blive vist på personens audit. Dette gør det muligt at opnå auditability, så personer kan blive bekræftet i deres handlinger.

### Authentication

Authentication service er en service der gør det muligt at blive registeret som bruger i systemet, og logge ind og modtage et JWT token. Authentication service bruges ved registrering og login. Hvilket gør det muligt at bruge JWT token til at tilgå person specifikke ressourcer i andre services. Dette er gjort ved at have en delt hemmelighed mellem de andre services, så personens token kan blive verificeret og godkendt. Denne token indeholder brugeres ID, hvilket kan hentes fra de andre services.

### Biddings

Biddings service er en service der indeholder de bud forskellige freelancere har lagt på et givent projekt. Disse bud er transiente og afhænger af projekts service  [10.3.10](#sec:projects-service). Når en freelancer er valgt gennem sit bud, vil det ikke længere være muligt at oprette bud til det pågældende projekt.

### Broker

Broker service håndterer det endelige resultat fra et projekt. Denne broker gør det muligt at uploade diverse dokumenter og anmode om betaling for det udførte arbejde. Derefter kommer projektet i en tilstand hvor employeren kan betale i bytte for det indleverede materiale. Denne service er afhængig af kollaboration service  [10.3.6](#sec:collaboration-service) for at kunne notificere de involverede brugere, henholdsvis freelancer og employer.

### Chat

Chat service implementerer en mindre version af en Messenger type chat, hvor to brugere kan skrive frem og tilbage mellem hinanden.

### Collaboration

Collaboration service er en tynd skal omkring domænet kollaboration. Dvs. at brugerne, kan ligge hvad som helst ind i servicen ud fra det interface udstedt, det skal dog være JSON. Converge-SPA kan derfor vælge hvilken type begivenheder den vil tillade mellem henholdsvis freelancer og employer. Dette gør det muligt at genbruge den samme service til nemt at kunne sende både fil referencer, betaling og simple beskeder til et projekt. Collaboration service går efter less-is-more, ved at implementere den simpleste validering muligt for et JSON Api. Grundet at funktionalitet skal være så dynamisk som muligt. Dette har dog en ulempe for en Web Klient. Men det ville være trivielt at lave en ny service som et interface for dette, for at blandt andet give det et mere konkret interface. Collaboration er en af de mest primitive byggeblokke for systemet, og skal med videre udvikling gemmes længere og længere ned i systemet.

### Files

Files service håndterer upload af filer, og kommunikation til Google Storage Bucket. En reference i form af et media link vil blive gemt i en database, så de brugte filer kan ses, og de ubrugte slettes. Denne service bliver brugt af både Collaboration service  [10.3.6](#sec:collaboration-service) og Profile service

### Payments

Payments service er nok den tungeste service i Converge-cluster og holder styr på hvilke konti er tilknyttet til tredjeparten Stripe, samt de betalinger tilhørende de forskellige konti. Payments service tilbyder også et Webhook til Stripe for at opdatere systemet internt, når en begivenhed for en konto sker.

### Profiles

Profiles service bruges til at indeholde information om selve profilen, det er alt fra referencer til tidligere erfaringer og jobs, samt en general portfolio.

### Projects

Projects service bliver brugt til at indeholde information om det gældende projekt og er relativt simpel, det en indeholder bare et ID om hvilket projekt det er samt en employer som udbyder og en freelancer hvis et bud er valgt. Projects service ligner meget users service  [10.3.11](#sec:users-service) pga. den simplicitet grænsefladen har.

### Users

Users service er en service der binder et ID sammen med noget personlig information om en bruger. f.eks. Et navn, e-mail osv. Users service sammen med Projects service bruges som Source of Truth, og hvis andre services er i tvivl, kan de spørge disse services om den gældende bruger og projekt eksistere og er gyldig.

# Systemets grænseflader

Følgende afsnit omhandler brugergrænseflader af systemets moduler. Der vil være en beskrivelse af Converge applikationen brugergrænseflader, samt server applikationen, med fokus på de mange services som systemet består af.

## Converge SPA

Grænsefladen til Converge-SPA er i form af et website, og bruger et dynamisk frontend bibliotek React til at tilbyde en Desktop lignende brugervenlighed. Normalt er websider statiske og vil ikke kunne reagere på brugerens interaktion uden at skulle genindlæse siden. React gør at det html hjemmesiden er bygget på kan opdateres igennem DOM-api. Derfor skrives hele Converge-SPA i React.

Selve grænsefladen blev designet i Figma, før udviklingen begyndte, dette var for at tilbyde hurtige iterationer over hvilken funktionalitet der var vigtigt.

Converge-SPA består af et hierarki af sider indeholdende de forskellige funktioner, på nedenstående diagram beskrives de forskellige muligheder for brugerens navigation.

Hver side er tilpasset til at skalere til en fuld desktop, en tablet eller en smartphone. Dette er for at tilbyde en glidende oplevelse fra de forskellige enheder. Selve designet følger et mørkt tema, med kontrast af hvidt og blåt til at vise Converge’s personlighed og karakter. Sammen med Material standarden fra Google, viser Converge et stilrent design, genkendeligt fra mange andre produkter på internettet, som Gmail, Facebook osv. Dog med dets, egen kvalitet og karakter.

## Server applikation

Server applikationen består af en række API ‘er, som bruges til at lagrer og udlevere data til brugeren. De opbevarede data kan webbrowsere hente via protokollen HTTP eller HTTPS. Converge systemet benytter disse API ‘er benyttes til at kommunikere med serveren. Den måde det fungerer på er at en klient sender en HTTP-anmodning til serveren, serveren returnerer et svar til klienten. svaret kan indeholde oplysninger om den anmodning brugeren ønsker. På nedstående figur ses kommunikationen mellem brugeren og serveren.

Først og fremmest er et JSON Api anvendt for at give adgang til systemdomænet. Der kunne været brugt hvilken som helst grænseflade, men HTTP protokollen er brugt i forvejen på internettet, og har relativ god performance, mod læsbarhed. Desuden kan denne protokol udnyttes af andre tredjeparter, til at lave deres egen brugergrænseflade.

Converge-SPA bruger HTTP kald til et samlingspunkt i Converge-cluster, dette samlingspunkt er defineret af Traefik, og er relativ ud fra hvilken ressource der ønskes adgang til. F.eks. users-service, kan tilgås via. users-service.api.converge-app.net/api/users. Dette gør det muligt at have et unikt end point hvor hver service. Dette er specielt rart, eftersom den reverse proxy brugt kan være så tynd som muligt, uden at skulle indeholde regler for routing af de forskellige kald.

De individuelle services tilbyder en API beskrivelse af deres eget Api på servicenavn.api.converge-app.net/swagger. Med swagger er det muligt at fra sin egen browser kalde de forskellige services, det er også her API ‘et er dokumenteret i detaljer.

# Vedligeholdelse

I dette afsnit vil vedligeholdelse af Converge blive beskrevet. Vedligeholdelse i denne kontekst er alt fra videreudvikling til simpel drift. For at forstå hvorfor vedligeholdelse er vigtigt, skal det forstås at Converge-cluster er en gruppering af applikationer, der ikke nødvendigvis hænger sammen. For at forstå hvorfor noget gik galt, og det generelle helbred af systemet, kræves der nogle værktøjer som normalt ikke er relevante når der tænkes en enkelt selvstændig server applikation.

De værktøjer brugt er delt op i kategorier, som drift, monitorering og udvikling. Til drift bruges Kubernetes, med Kubernetes kan en applikation automatisk skalerer eller blive gendannet efter behov. Dette er specielt nyttige for relativt ustabile applikationer, da de vil genstarte automatisk. Til monitorering vil de forskellige services blive bundet sammen, hvor spor, logs og andre system information vil kunne spores i deres respektive applikationer. Med værktøjer som ELK, Jaeger eller Prometheus er dette muligt. Til udvikling er der brugt Git, GitLab, Docker og Helm. Git bruges til at versionerer kode, GitLab til at køre denne kode og danne det eventuelle kode kørende på Converge-cluster, ved at pakke det med Docker. Derefter udruller det med Helm på Kubernetes.

# Tests

I de følgende afsnit vil de forskellige tests, der er lavet til systemet blive beskrevet, herunder unit tests, integrations tests, modul tests og system test (automatisk accept test). Slutteligt vil udførelsen af accepttesten blive beskrevet.

## Unit Tests

I dette afsnit vil det blive beskrevet, hvordan de to enheder i systemet er blevet unit testet. Først er den overordnede fremgangsmåde beskrevet, hvorefter der følger en beskrivelse af de opnåede resultater.

Formålet med unit tests er at teste funktionalitet med så lille granularitet som muligt. f.eks. en klasse eller en funktion, dette er gjort under isolation, så ingen effekt fra omkring værende miljø har nogen effekt. Denne test suite er den største mængde tests, eftersom det skal være muligt at kunne identificere et problem på den mindst mulige tværsnit. Denne form for tests vil blive kørt automatisk når software blive skubbet til Git.

I udviklingen af unit tests har der ikke været stor fokus på code-coverage, noget man normalt ville have for en større kode base. Men da meget af de koder skrevet har været bundet op på det framework brugt, så er det eneste der har været nyttigt at teste været forretningslaget mellem frameworket og database eller cluster.

### Fremgangsmåde

For at finde frem til hvilke test cases der skulle være for hvert enkelt element, er der blevet brug en fremgangsmåde, hvor man kigger på hvad er en offentlig grænseflade, og tester alle de scenarier der er for dette stykke kode. Til de individuelle test cases er der fulgt den såkaldte metode, “Arrange, Act, Assert”, dette bruges for at give de forskellige tests samhørighed. Alt i alt er der noget kode der er frifundet fra tests, dette kan være til at bygge ui, eller fra frameworket selv.

#### Converge-SPA

Til Converge-spa er der testet de services hvor der i forvejen er meget lidt forretningslogik. Det giver ikke mere end omkring 50 tests.

#### Converge-cluster

Til Converge-cluster er der testet de individuelle services, da det er det eneste kode urørt af framework kode, og er det kode med størst risici for at fejle. Der er flere hundrede tests for Converge-cluster spredt ud over alle de anvendte services. Grunden til at frameworket brugt til de forskellige services ikke er testet, er fordi de bliver testet af integrationstests. Hvilket vi har afgjort til at være godt nok til dette projekt.

## Integrationstests

I dette afsnit vil projektets brug af automatiserede integrationstests blive beskrevet. De implementerede integrationstests er blevet kørt på CI serven sammen med unit tests.

Integrationstests er kun kørt for Converge-cluster

### Converge-cluster

Integrationstest forstås af projektgruppen, som en test der primært tester grænsefladen mellem klasser (komponenter), i en ideel udviklingsproces ville man have en løbende integrationstest, men med naturen af dette projekt som kun bruger en udgivelse af produktet, fandt man det ideelt at lave den en enkelt gang.

Projektgruppen har valgt at teste relationen mellem services, eftersom med den arkitektur valgt for systemet passer med den granularitet som man normalt laver integrationstests for. Dvs. at man ikke tester kode direkte, men tester med grænsefladen af de forskellige services.

Denne form for tests køre når koden er udgivet på en anden branch end masteren og tester med de komponenter, og deres komponenters komponenter, så den enkelte funktion kan testes i et miljø der ligner produktion så meget som muligt.

## Systemtest

I dette afsnit vil de udførte systemtest for systemet blive beskrevet. Der differentieres mellem automatiseret systemtest, der bruges i forbindelse med subsystemtest, og en manuelt systemtest accepttest, der bruges til at teste hele det samlede system.

### Automatiseret Systemtest

Systemtest og i projektets forståelse og brug, automatisk-accepttests. Ved brug af diverse værktøjer som Selenium og Python er der opstillet tests som tester Converge-SPA i forhold til de Epics og user stories, samt ikke-funktionelle krav. Dette er for at sikre at hele systemet fungere, efter en ny udgivelse. Systemtest kan blive startet af alle CI/CD-jobs og er vigtig for at den nye udgivelse virker, og hvis ikke rullet tilbage.

### Accepttest

Nedenfor vil accepttesten blive beskrevet. Accepttesten i sig selv vil ikke være inkluderet, men kan findes i dokumentationen.

Accepttesten er udført hver gang der testes med en testperson, samt med vejleder før projektets aflevering. Hvis der har været fejl eller mangler fra accepttest vil de være dokumenteret i udeståender og en handlingsplan lavet og oprettet under ZenHub.

Testbrugers kommentarer i forhold til accepttest er også taget til forbehold under udviklingen og har hjulpet med at bane vejen for den retning produktet skulle udvikles.

# Test brugere

I dette afsnit vil brugen af test brugere blive beskrevet. Test brugere er eksterne parter som har hjulpet med udviklingsprocessen ved at teste produktet under udviklingen. Testpersoner har både været folk med erfaring for freelancing platforme, som læ personer med ingen erfaring.

Det var vigtigt at have begge, både for ekspertviden, men også for at få et friskt førstehåndsindtryk.

Der har været brugt en række testpersoner, deres erfaringer er beskrevet i dokumentationen under testpersoner. Test brugere har været en uvurderlig ressource, og har hjulpet med at prioritere krav efter hvad der var vigtigst. Testpersoner har været brugt under hele udviklingen, men brugt især i starten hvor designet var udformet, dette var vigtigt, for på det tidspunkt var den meget lettere at iterere over funktionalitet. Test brugere har derefter været brugt løbende og til sidst ved udførelsen af accepttesten.

# Udeståender

I dette afsnit vil udeståender blive beskrevet. Opsætning vil være et nummer, en kort beskrivelse og et link til det oprettede problem på GitHub, under den funktionalitet der giver problemer, hvis ikke vil den ligge på Converge-SPA og have et label unknown.

# Diskussion

Projektet havde til formål at skabe et moderne arbejdes platform ind i det 21. århundrede, der kunne bruges af bruger der ønsker at enten arbejde selvstændig med projekter og have frihed under eget ansvar eller få løst kort eller langvarigt projekter, efter deres ønske. På denne måde skulle det være muligt at både få løst og tage projekter, uanset hvor brugeren befandt sig i verden. Det endelige resultat endte ud med en web platform, hvor det består af en front-end, back-end og en database. Front-enden er udarbejdet som SPA (single page application). Derudover er front-enden er serverløs det betyder at man ikke har en server som man skal holde styr på eller kører, men det gør andre såsom: Google cloud. Back-enden er udarbejdet i ASP.Net core og databasen er MongoDB. Herudover består systemet af en række værktøjer (Docker, Kubernetes, Helm, Jaeger, Traefik osv.), der gør udvikling nemt og samtidig har det gjort at målet er nået.

Her kan der diskuteres om alle mål og forventninger er opfyldt for webapplikationen Converge. Ud fra gruppens ambitiøse mål og forventninger er alle målene opfyldt og derfor fungere efter hensigten. Dog skal det siges at der er fortaget nedskæringer i funktionalitet, hvor gruppen var enig om hvilken funktionalitet der var vigtige for en prototype. For eksempel kan en bruger ikke få tilsendt en ny kode, logge ind med google, samt andre små funktionaliteter. Disse funktionaliteter kan brugeren godt undvære i første om gang, dog med tanker om at tilføje dem, hvis der videreudvikles på systemet.

En anden mål som gruppen havde for systemet, var at benytte forskellige værktøjer til at gør udviklingen nemmer og samtidig opnå målet. Disse værktøjer har haft hver deres formål og har dermed opfyldt gruppens behov på hver deres måde. For eksempel er Docker, Kubernetes, Helm blevet brugt til udrulning af produktionsmiljøet, Jaeger er blevet brugt som spores værktøj, der finder spore mellem de forskellige relationer som service har og der findes yderligere værktøjer der er blevet benyttet til udvikling af Converge platformen afsnit(xxx).

# Konklusion

Formålet med dette projekt var at udvikle en platform, der skulle gøre det muligt at arbejde på en ny måde. Med Converge er det muligt for en udbyder at kunne købe arbejdskraft og ekspertise til et ønskes produkt eller område.

Converge har været udviklet med både slutbrugeren og udvikleren i centrum. For at give udvikleren den bedste mulighed for at lave det bedste produkt. Converge har også været udviklet efter moderne startup metoder, og har haft fokus på at få et Minimal-Viable-Produkt i luften. Dette har været gjort med moderne teknologier og arbejdsmetoder. Med en decentral struktur drevet af en agil tilgang til softwareudvikling.

Converge har været designet fra grunden op med fokus på at inddrage brugeren, derfor har rigtige testpersoner været anvendt til at pege produktet i den rigtige retning, samt at kvalitet sikre produktet. Converge, har gået fra design til produktion, og er med få ændringer klar til at rulle en lille smule early adopters ind.

Softwaren har været testet, med unittests, integrationstests samt systems tests og et automatisk accepttest system. Ikke nok til en 100% fyldestgørende test, men nok til at der er en vis tillid til systemet. Accepttest anses som godkendt, med enkelte fejl meldt som udeståender, med tilhørende aktionsplan.

Det kan konkluderes, at projektet er veludført, og det udviklede produkt er fyldestgørende, med de vigtigste krav implementeret for at opnå visionen og løsningen på problemstillingen.