



Technical University  
of Denmark

---

# DTU Roadrunners

---

Cecilie Lindberg, s153869

Thomas Bech Madsen, s154174

Sebastian Roel Hjorth, s153255

Daniel Plaetner-Cancela, s154163

8. december 2017

# Indholdsfortegnelse

<b>1</b>	<b>Introduktion</b>	<b>7</b>
1.1	Problemstilling	7
1.2	Problemanalyse	8
1.3	Problemformulering	8
1.3.1	Supplerende spørgsmål	8
<b>2</b>	<b>Projektplanlægning</b>	<b>9</b>
2.1	Gruppekonspekt	9
<b>3</b>	<b>Analyse</b>	<b>11</b>
3.1	Kravspecifikation	11
3.1.1	Funktionelle krav	11
3.1.2	Non-funktionelle krav	12
3.2	User Stories	13
3.3	Tidsplan	14
3.3.1	Prioritering	15
3.3.2	Foreløb projektplan for 13-ugers	16
3.4	Domænemodel	16
3.5	ER analyse	17
3.6	Versionstyring	18
3.7	Hierarki	19
3.8	Sikkerhed	19
<b>4</b>	<b>Design</b>	<b>20</b>
4.1	Swagger	20
4.2	Arkitektur	20
4.2.1	Modellaget	21
4.2.2	Controller laget	21
4.2.3	View laget - API struktur	21
4.3	Deployment Diagram	23
4.3.1	Databasestruktur	24
4.4	Normaliseringsanalyse	25
4.4.1	Første normalform	26
4.4.2	Anden normalform	26
4.4.3	Tredje normalform	26
4.5	Sikkerhed	27
4.5.1	Oprettelse	27

4.5.2	Login . . . . .	28
4.5.3	JSON Web Token . . . . .	28
4.5.4	Hashing . . . . .	29
4.5.5	HTTPS . . . . .	29
4.6	Versionstyring i databasen . . . . .	29
4.7	Hierarki . . . . .	30
<b>5</b>	<b>Implementering . . . . .</b>	<b>31</b>
5.1	MVC i praksis . . . . .	31
5.1.1	Modellaget . . . . .	31
5.1.2	Controller laget . . . . .	31
5.1.3	View laget . . . . .	32
5.2	Konfiguration . . . . .	34
5.3	Asynkrone kald . . . . .	34
<b>6</b>	<b>Database . . . . .</b>	<b>34</b>
6.1	Database problem . . . . .	35
6.2	Versionering . . . . .	36
6.2.1	Stored procedures . . . . .	36
6.3	Sikkerhed . . . . .	39
6.3.1	Oprettelse af bruger . . . . .	39
6.3.2	Campusnet API adgang . . . . .	40
6.3.3	Hashing af kodeord . . . . .	40
6.3.4	Login . . . . .	40
<b>7</b>	<b>Vurdering . . . . .</b>	<b>42</b>
7.1	Revideret tidsplan . . . . .	42
7.1.1	Problemer og løsning . . . . .	42
7.2	Procesvurdering . . . . .	43
7.3	Sikkerhedsrapport . . . . .	43
7.3.1	Confidentiality (Fortrolighed) . . . . .	43
7.3.2	Integrity (Integritet) . . . . .	44
7.3.3	Availability (Tilgængeligt) . . . . .	44
7.3.4	Trusselvurdering . . . . .	44
7.3.5	Risikovurdering . . . . .	44
7.3.5.1	Sårbarhed HTTPS . . . . .	44
7.3.5.2	Server nedbrud . . . . .	45
7.3.5.3	Data håndtering . . . . .	46

7.3.6	Handlingsplan . . . . .	47
7.4	Test . . . . .	48
<b>8</b>	<b>Konklusion . . . . .</b>	<b>49</b>
8.1	Perspektivering . . . . .	50
8.2	Videre arbejde . . . . .	50
8.2.1	Dokumentation og videresendelse . . . . .	50
<b>9</b>	<b>Bilag . . . . .</b>	<b>52</b>
9.1	Tidsplan . . . . .	52

# Gloser

## Forklaringer

**System:** *Er det program som vi har lavet med database og api*

**Projekt:** *Er alt det som v har lavet, analyse, design, implementering og tidsplan med mere.*

**Category:** *En kategori beskriver den overordnet type af en component type, dette kunne være et hjul f.eks.*

**Component typer:** *Component typer er som componenter kan være, det kan være et specielt hjul type eller skrue type.*

**Component:** *Er en fysisk komponent type.*

**Document:** *Et document kan være alle slags filer som pdf, text, billeder eller en mathlap fil.*

**Elementer:** *Elementer er i dette tilfælde de forskellige ting man kan putte i databasen som et document og component.*

**null:** *Er ingenting*

**CAS:** *Henviser til Campusnet login*

**VCAPI:** *Henviser til systemet*

**ROADRUNNERS:** *DTU's økobilteam*

## Forkortelser

**API:** *Application Programming Interface*

**CAS:** *Campus Service*

**CIA:** *Confidentiality Integrity Availability*

**CRUD:** *Create, Read, Update, Delete*

**DTU:** *Danmarks Tekniske Universitet*

**DDoS:** *Distributed Denial of Service*

**DoS:** *Denial of Service*

**HTTP:** *HyperText Transfer Protocol*

**HTTPS:** *Hypertext Transfer Protocol Secure*

**ECTS:** *European Credit Transfer and Accumulation System*

**ER:** *Entity Delationship diagram*

**EER:** *Enhanced Entity Delationship diagram*

**IO:** *In/Out*

**JSON:** *JavaScript Object Notation*

**JWT:** *JSON Web Token*

**MVC:** *Model View Controller*

**OWASP:** *The Open Web Application Security Project*

**PERT:** *Program Evaluation and Review Technique*

**REST:** *Representational State Transfer*

**SOAP:** *Simple Object Access Protocol*

**SSL:** *Secure Sockets Layer*

**VCAPI:** *Version Control API*

**URL:** *Uniform Resource Locator*

**YAML:** *Yet Another Markup Language*

# 1 Introduktion

Dette projekt er et specialkursus, der laves i samarbejde med DTU Roadrunners og instituttet for Matematik og Computer Science i 13 ugers perioden. Projektet har til formål at konstruere et versionsstyringssystem, hvor gæster, ingeniører og underviserer kan se, gemme og redigere data og dokumentation om flere projekter. Det skal efterfølgende dokumenteres til fremtidige projekter, såsom kemi og andre institutioner på DTU.

Formålet er, at vi alle bliver i stand til at skabe et projekt helt fra bunden,- fra kravspecifikationerne der laves i samarbejde med DTU Roadrunners til et færdigt og testet produkt, hvori kreativiteten og projektplanlægning vægter højt for at opnå et produkt, vi kan være stolte af og en høj indlæringsprocess.

## 1.1 Problemstilling

Flere og flere benytter sig af teknologi til at dokumentere og ajourføre projekter, vi mener, at kommunikationen og arkiveringen af arbejdet kan forbedres digitalt og gøre processen mere effektivt, så interaktionen mellem de implicerede parter flyder lettere. Følgende problemstillinger opstilles:

- Hvordan fungerer kommunikation mellem slutbrugeren og systemet?
- Hvordan skal forskellige typer brugere benytte samme system?
- Hvad skal der ske, når man sletter data?
- Hvordan skal det være muligt at gå tilbage til tidligere dokumenter eller data?
- Hvordan kan vi sikre os, at der skabes integration mellem de forskellige typer brugere?
- Hvordan skal systemet dokumenteres til fremtidig udvikling?
- Hvordan skal systemet sammenkobles med DTU login system?
- Hvordan skal systemet kunne benyttes hybrid?
  - Web applikation
  - Mobilapplikation
- Hvordan skal hierarki strukturen opstilles mellem de forskellige elementer?

Svaret på disse spørgsmål vil have stor betydning for, hvordan man forbedrer og effektiviserer versionsstyringssystem og dokumentationen.

## 1.2 Problemanalyse

Hvilke positive følger har et versionsstyringssystem af dokumenter og elementer? Hvilke negative konsekvenser har et specialkursus uden undervisning og selvstudium?

Eftersom, at vi selv skal formulere projektet og dens læringsmål, er vi selv nødt til at skabe realistiske mål, både indenfor kursets rammer, men også hvad vi selv er i stand til opnå med vores nuværende viden og tid. Vi skal yderligere undersøge hvordan kommunikationen og interaktionen mellem brugere, kan forbedres og gøres mere effektivt, samt hvilken effekt disse forbedringer har for slutbrugeren.

## 1.3 Problemformulering

Hvordan kan man digitalisere versionsstyringen af DTU Roadrunners projekter til flere forskellige typer bruger, og efterfølgende få et hierarkisk struktur på de forskellige dele?

### 1.3.1 Supplerende spørgsmål

- Hvilke overvejelser skal vi have omkring lanceringen af produktet og hvad skal dokumenteres?
- Hvordan skal vi verificerer DTU brugere eksternt?
- Hvordan skal man planlægge til et selvstudium projekt?
- Hvordan håndteres sletning og oprettelse af tidligere dokumenter?
- Hvad skal de forskellige brugere have adgang til?



## 2 Projektplanlægning

Til dette projekt har vi valgt at arbejde meget ind over hinanden og parallelt for at få en fuld forståelse af hvad der sker, hvilket indebærer en form for pair programming fra Extreme Programming. Endvidere medfører denne type arbejdsproces også til færre fejl og bedre dokumenteret projektopbygning. Analyse- og den teoretiske del er diskuteret og udarbejdet fælles, for at få kendskab til det hele og få en konkret viden til de forskelliges ideer.

Vi som gruppe har alle haft forskellige styrker og svagheder, dette indebærer, at der hvor nogen har haft en styrke en anden en svaghed, er der blevet inkorporeret en inddækningsproces for at hindre denne svaghed fremover og gruppens ekspertise blev styrket. Dette betyder at alle i gruppen har været med i samtlige faser gennem projektet og endvidere har alle haft ansvar for de forskellige komponenter og dele. Samtidig giver dette en ligebyrdig arbejdsindsat fra alle gruppemedlemmer.





### 2.1 Gruppekонтракт

Til dette projekt er der lavet en forventningsafstemning, for at udforme en gruppekонтракт, med underskrifter samt kontaktoplysninger. Dette er med til at sikre, at forventningsafstemningen bliver overholdt og giver et endegyldigt flow i arbejdsprocessen. Skulle det ske, at et gruppemedlem ikke lever op til kontrakten og kravene, har vi mulighed for at henvise til gruppe kontrakten.

Gruppekонтракт indebærer og kan ses i tabel 1:

- Hvert gruppemedlem skal så vidt muligt møde op til hvert aftalt tidspunkt.
- Hvert gruppemedlem skal melde fra ved sygdom eller anden forhindring.
- Hvert gruppemedlem skal lave de udleveret lektier til hver gang, er det ikke muligt skal gruppemedlemmet udmelde det.
- Hvert medlem skal efterstræbe at påtage sig relevant ansvar i forhold til gruppens arbejde.
- Vi skal acceptere hinandens styrker og svagheder.

Tabel 1: Gruppekontrakt

Navn	studienummer	Telefon	Underskrift
Cecilie Lindberg	s153869	61603272	
Thomas Bech Madsen	s154174	20768949	
Sebastian Roel Hjorth	s153255	29929989	
Daniel Plaetner-Cancela	s154153	20722123	

## 3 Analyse

Følgende afsnit beskriver hvilke tanker og ideer, der er kommet ud fra projektbeskrivelsen, hvordan de forskellige aktiviteter og komponenter er blevet analyseret for at opnå et gennemarbejdet projekt, der kan skabe grundlag for et veldokumenteret projekt. Endvidere beskriver den også projektplanlægningen, hvordan vi som gruppe har tænkt os at prioritere og arbejde - dette medfører derfor, at alt der beskrives ikke nødvendigvis bliver gjort, men at der har været meget fokus på analysedelen for at fremadrette de næste processtadier.

### 3.1 Kravspecifikation

DTU Roadrunners ønsker en applikation, der lader brugere oprette dokumentation til de forskellige komponenttyper og komponenter med tilhørende versionsstyring. De forskellige brugertyper, skal have forskellig adgang, afhængig af hvilken rolle de får tildelt.

Følgende lister, beskriver de funktionelle og non-funktionelle krav til systemet. Efter som vi selv er opgavestillere, er vi derfor også nødt til at opstille vores egne krav. Disse krav blev defineret i samarbejde med DTU Roadrunners holdet.

#### 3.1.1 Funktionelle krav

De funktionelle krav[1] er opdelt ud fra de forskellige typer af brugere og er skrevet hierarkisk. En superbruger har for eksempel mulighed for alle nedenstående undergrene af brugere.

- Superbruger
  - skal kunne oprette admin
  - skal kunne redigere admin
  - skal kunne slette admin
  - skal kunne se admin
  - skal kunne se en liste af admins
- Admin
  - skal oprette projekt (fx økobil eller vindbil)

- skal kunne slette projekt
- skal kunne redigere projekt
- skal kunne se projekt
- skal kunne se en liste af projekter
- skal kunne oprette brugere
- skal kunne redigere brugere
- skal kunne slette brugere
- skal kunne se brugere
- skal kunne se en liste af brugere
- skal kunne oprette kategorier
- skal kunne slette kategorier
- skal kunne redigere kategorier
- skal kunne se kategorier
- skal kunne se en liste af kategorier
- Elev
  - skal kunne oprette dokumentation
  - skal kunne redigere dokumentation
  - skal kunne slette dokumentation
  - skal kunne se dokumentation
  - skal kunne se en liste af dokumentation tilknyttet et komponent
- Gæst
  - skal kunne se dokumentation, men ikke hvem der har oprettet det.
  - skal kunne se kategorier

### 3.1.2 Non-funktionelle krav

De non-funktionelle[2] krav til systemet er skabt for at få en god og gennemtestet brugeroplevelse - det er dog ikke endegyldigt, at alle kravene bliver overholdt endnu, men skal dokumenteres videre til fremtidig udvikling af projektet. Endvidere menes der med, at

applikationen skal være let at bruge, at der opstilles en række tests for at undersøge dette. Disse tests laves med den traditionelle usability testing metode på brugere, der arbejder med DTU Roadrunners til dagligt eller har det som kursus. Dette er med til at give os en struktureret test, så vi kan se hvor testbrugeren går i stå og hvad der volder problemer i applikationen.

- Applikationen skal være let at bruge (Usability)
- Applikationen skal være sikkert at bruge (Security)
- Applikationen skal kunne køres på Web og mobil (Platform compatibility)
- Applikationen skal kunne versioneres og logges (Documentation)
- Koden skal dokumenteres og kunne videreføres (Documentation/Scalability)

### 3.2 User Stories

Der er blevet lavet en række User Stories, som beskriver de funktionelle krav, som der tidligere er blevet skrevet i krav specifikationen[3]. En User Story bruges til at få en større forståelse for de krav, der er til systemet, og samtidig bliver de brugt til at komme op med en prioritering af kravene, og en estimeret tid som det ville tage at få denne User Story til at virke i praksis. De estimerede tider bliver brugt til at udarbejde en tidsplan. Et eksempel på den User Story kan ses i Tabel 2.

Tabel 2: User story

ID: 10
Navn: Admin - opret kategori
Beskrivelse: Som admin skal man kunne oprette en kategori
Krav: <ul style="list-style-type: none"><li>- Man er logget ind som admin</li><li>- Kategorien man opretter skal ikke findes i forvejen</li></ul>
Prioritering: høj
Estimering: 19 timer

Efter alle User Stories er lavet kan man bruge dem til at se, om det program man udvikler lever op til de krav, der blev opstillet i starten. De kan også bruges under selve

udviklingen, da man kan kigge på de krav, der er til en User Story og huske på, at disse skal være overholdt, før at selve User Story kan blive udført.

### 3.3 Tidsplan

Tidsplanen er oprettet i samarbejde med User Stories, de har været med til at danne grundlag for projektet og hvordan arbejdsindsatsen og timerne skulle udvælges. Derfor er hver aktivitet i tidsplanen blevet estimeret ud fra en User Story.

Tidsestimeringen til User Story dækker følgende kerneprincipper, analyse, design, implementering og test, samt den tilhørende dokumentation til alle faserne - dette indebærer rapportskrivning, kode, dokumentation og test. Det er altså ikke kun hvor lang tid det tager at kode den enkelte User Story, men tidsforbruget helt fra start til slut i alle faserne.

Til tidsestimeringen er der blevet benyttet PERT's analyseprincipper[4], som skaber en mere præcis estimering fremfor at tage gennemsnittet, da man vægter det realistiske mere end en optimistisk og pessimistisk vurdering. PERT formelen der benyttes er:

$$\frac{Optimistisk + 4 \cdot Realistisk + Pessimistisk}{6} \quad (1)$$

Estimeringsværdierne til de forskellige aktiviteter er mandetimer, det vil sige arbejder er en person 5 timer om en aktivitet, er den estimeret til 5 timer, men derimod er der to personer der arbejder 5 timer, er den estimeret til 10 timer - så formelen er:

$$\sum_{i=1}^n timer_i \quad (2)$$

Hvor  $n$  er antal personer der arbejder på aktiviteten og  $timer_i$  er personens individuelle timer. Dette har derfor medført til følgende endelige tidsplan i tabel 3. Endvidere ses det, at den totale estimeret tid går op til 1622 timer, hvilket er langt over den estimeret tid der er tilrettelagt 13 ugers for 4 personer:  $(\frac{timer}{ECTS_{point/uge}} \cdot n_{personer}) \cdot z_{uger} = (9 \cdot 4) \cdot 13 = 468$ . Det indebærer derfor, at følgende projekt ikke vil blive færdigt i 13-ugers perioden og kræver, at projektet dokumenteres grundigt til næste hold der skal arbejde på projektet.

Tabel 3: Tidsplan over aktiviteterne, med PERT estimeret tid og prioritet

Aktivitet (Navn)	Optimistisk	Realistisk	Pessimistisk	Estimeret	Prioritering
<b>Superbruger</b>					
skal kunne oprette admin	150	160	180	162	mellem
skal kunne redigere admin	60	80	100	80	lav
skal kunne slette admin	10	15	20	15	lav
skal kunne se admin	12	15	35	18	lav
skal kunne se en liste af admins	12	20	50	24	
<b>Admin</b>					
skal oprette projekt	200	250	400	267	meget høj
skal kunne slette projekt	50	75	90	73	lav
skal kunne redigere projekt	10	10	30	13	lav
skal kunne se projekt	12	8	10	9	meget høj
skal kunne se en liste af projekter	10	15	20	15	høj
skal kunne oprette brugere	70	90	120	92	høj
skal kunne redigere brugere	20	40	54	39	lav
skal kunne slette brugere	40	100	120	93	lav
skal kunne se brugere	22	34	42	33	høj
skal kunne se en liste af brugere	10	20	30	20	lav
skal kunne oprette kategorier	10	20	25	19	høj
skal kunne slette kategorier	32	54	66	52	lav
skal kunne redigere kategorier	10	15	20	15	lav
skal kunne se kategorier	30	20	30	23	lav
skal kunne se en liste af kategorier	5	10	15	10	mellem
<b>Elev</b>					
skal kunne oprette dokumentation	140	180	300	193	meget høj
skal kunne redigere dokumentation	25	40	60	41	høj
skal kunne slette dokumentation	30	55	80	55	høj
skal kunne se dokumentation	15	20	25	20	høj
<b>Gæst</b>					
skal kunne se dokumentation	140	200	220	193	meget lav
skal kunne se kategorier	20	50	60	47	meget lav
I alt	1145	1596	2202	1622	

### 3.3.1 Prioritering

Prioriteringen kommer til at vægte adskilt for alle de forskellige faser - hvilket betyder, at hvis en aktivitet har prioriteringen høj og en anden har lav, så betyder det ikke, at man laver den højest prioriterede aktivitet først, men at man først designer og analyserer den og efterfølgende designer og analyserer den næste. Dette medfører derfor en agil arbejdsproces fordi, at man til hver fase prioriterer de vigtige først, og gør det efterfølgende særskilt på hver aktivitet igennem forløbet. Derfor betyder det til sidst, at den højest prioriteret bliver først færdig men, at de alle bliver gennemarbejdet parallel.

Som det fremgår i tabel 3, under prioriteringens kolonnen, er elevs aktivitet og oprettelse af projektet sat til meget høj og er derfor også kerneprincippet for versionsstyringen i projektet. Gæsterne har fået en lav prioritet, da denne aktivitet ikke er ensbetydende for at kunne køre programmet, men er et *nice-to-have* i fremtiden. Det betyder derfor at oprettelse af projekt og dokumenter har den primære fokus i 13 ugers.

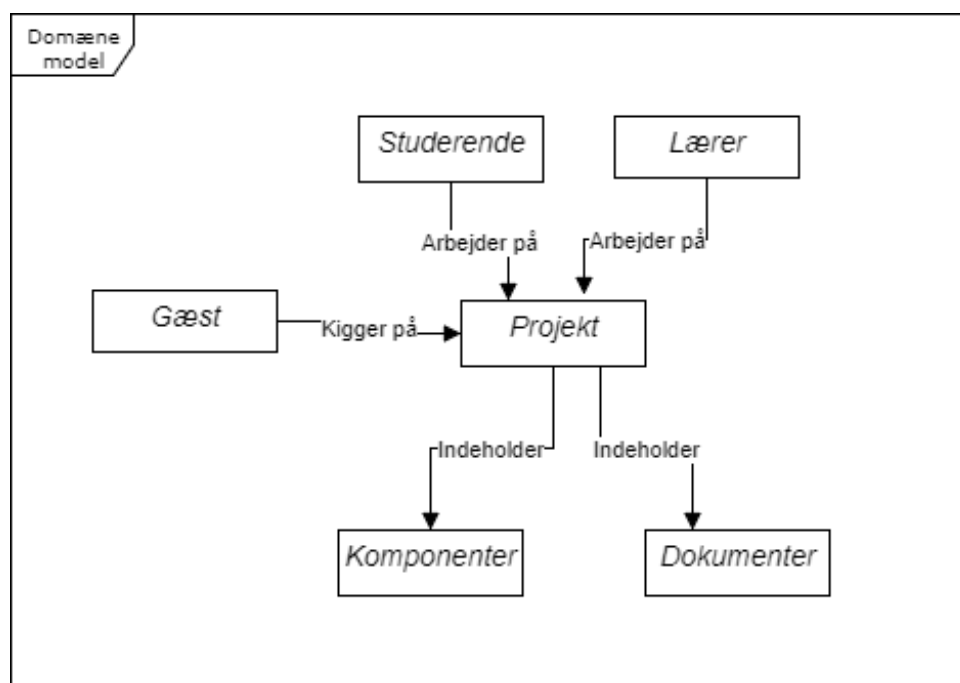
### 3.3.2 Foreløb projektplan for 13-ugers

Efter PERT analysen og tidsestimeringen er der blevet lavet en grundlæggende tidsplan, der har taget til grundlag for hvad vi har skulle nå i 13-ugers perioden og hvilke features vi gerne ville opnå før projektet blev skubbet videre til næste milepæl. Den generelle tidsplan kan ses i bilag i sektion 9.1.

Dette er en ønsket tidsplan, hvilket er noget vi vil prøve at opnå før projektet sendes videre. Det er ikke sikkert, at alt bliver lavet eftersom, at der kan komme problemer undervejs, men det er vigtigt, at alt dokumenteres, så det kan sendes videre næste runde.

## 3.4 Domænemodel

For at danne et overblik over hvilket domæne der skal arbejdes i, er der blevet lavet en domænemodel, som viser det domæne, der arbejdes i. Som det kan ses på Figur 1 arbejder både lærer og studerende på et projekt (evt. flere projekter) og, at disse projekter indeholder både dokumenter og komponenter. Disse komponenter kunne f.eks. være et hjul til en bil.



Figur 1: Domænemodel

Denne model har været udgangspunktet for hvilke elementer der skulle være i programmet, den er senere hen blevet udvidet så den passer bedre til programmet og det som

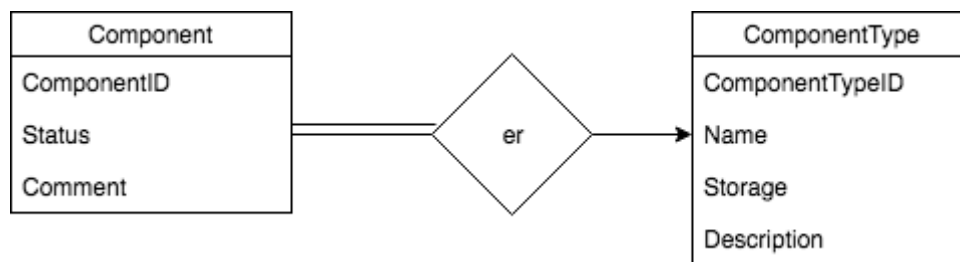


programmet skulle kunne. F.eks. er der i stedet for en lærer, en gæst og en studerende blevet lavet brugere, som så har forskellige roller i forhold til et projekt.

### 3.5 ER analyse

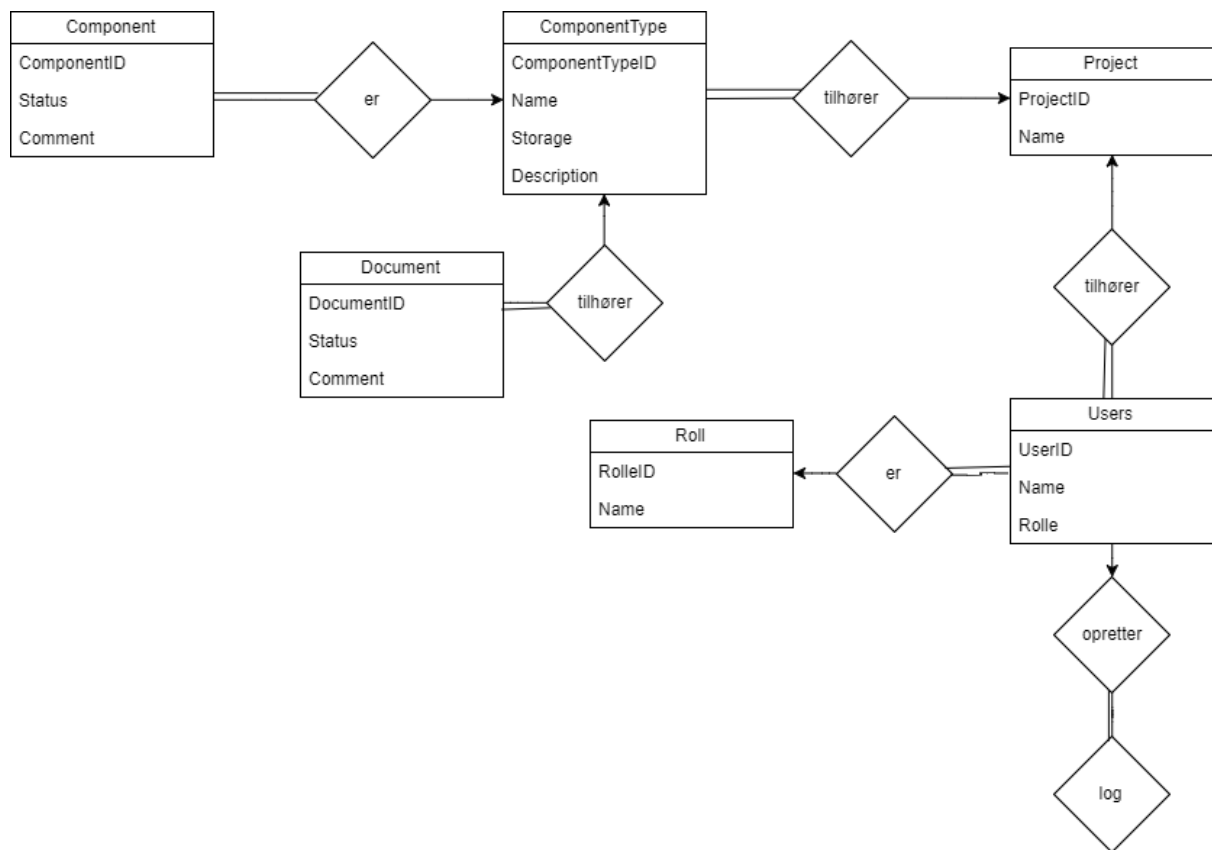
Ud fra kravspecifikationerne, er der blevet analyseret hvad systemet overordnet skal indeholde og hvilke relationer der efterfølgende skal dannes. Eftersom at det er i begyndelsen af analysefasen, er der ikke blevet fokuseret på normaliseringer og opdeling af de forskellige tabeller i første omgang, derimod kun de vigtigste egenskaber.

Som det ses på Figur 2, er systemets vigtigste egenskaber taget ud, nemlig at en *component* og en *componentType*. Det betyder at den eneste foreløbige relation er, at et *component*,  $\langle \text{er} \rangle$ , og *componentType*, hvor et *component* ikke kan eksistere uden et *componentType*. I et tidligt entitets relations analyse betyder det, at relationen beskrives med en diamant og et verbum, hvori den stærke relation understreges med dobbelt streg.



Figur 2: ER Analysedel 1

Efter første analysedel, er der efterfølgende blevet videreudviklet på det simple 'entity relations diagram', som det ses i Figur 3, dette indebærer igen at kigge på kravspecifikationerne, hvilket betød, at de resterende funktioner også skulle inkorporeres. Her er det vigtigt at se, at alle relationerne i sidste ende er tilknyttet et *Project*, hvilket betyder, at det er roden til de tilhørende elementer og de ikke kan leve uden. Desuden er *Users* også tilknyttet *Role*, hvor brugere har en tilhørende rolle, da det betyder, at man kan have forskellige roller afhængig af projektet man er oprettet til. Endvidere er *Document*, også relateret til *ComponentType* for at signalere, at man kan have flere Dokumenter, og dokumenter kan være alt fra PDF filer, matlab filer og andre slags filer, de er derfor også tilknyttet komponenterne.



Figur 3: ER Analysedel 2

Hvis man kigger på *log*, der er tilknyttet *Users*, står der, at en User <opretter> en log, hvilket i dette tilfælde betyder, at hver handling en user laver, opretter man noget til loggen - det er stadig for tidligt i analysefasen at fremhæve hvad der skal logges og derfor implementeres den indtil videre som en handling og ikke en tabel.

### 3.6 Versionstyring

Da der i systemet skal være versionsstyring på alle elementer i databasen (dokumenter, komponenttyper, projekt, komponenter osv.), er det blevet diskuteret hvordan sletning og tilbagerulning skulle håndteres i systemet. Da man i teorien gerne vil kunne slette elementer som man har lavet, men samtidig gerne ville kunne fortryde eller se en gammel en sletning, skal elementer aldrig rigtigt blive slettet fuldstændigt. En anden ting man skal tage højde for ved versionstyring er tilbagerulning. Tilbagerulning skal ske når man fortryder en eller flere ændringer og ønsker at gå tilbage til en tidligere version. Derfor er det vigtigt at vide hvilke versioner, der har været igennem tiderne og hvornår disse ændringer er sket, og hvilken slags ændring det var.

### 3.7 Hierarki

I systemet er det også ønsket, at man skal kunne se et hierarki over de forskellige komponenter. Dette vil sige, at man skal kunne se hvilke komponenter, der sidder sammen, og på denne måde bevæge sig rundt i f.eks. økobilen's sammensætning. Disse forhold i mellem komponenterne skal registres, så man senere hen kan se hvordan de arbejder sammen. Den primære fokus kommer dog til at være på versioneringssystemet.

### 3.8 Sikkerhed

Systemet skal indeholde en form for autorisering, da systemet kan indeholde arbejdstegninger og dokumentation, som ikke skal være frit tilgængeligt for alle. Systemet er delt op i følgende rettigheder hvor, at hver rank arver rettighederne fra den næste.

1. Superbruger
2. Admin
3. Elev
4. Gæst

Hver rank's rettigheder er vist i tabel 4

Tabel 4: Rettigheder for hver rank. Hver rank arver rettighederne fra den forrige.

Gæst	Læse dokumentation
Elev	Ændre dokumentation og kategorier
Admin	Ændre projekter og brugere
Superbruger	Ændre admins

Alle login skal være tilknyttet campusnet for at begrænse adgangen til DTU studerende.

## 4 Design

Følgende afsnit beskriver de forskellige valg projektet har bygget på fra analysefasen. Det indebærer systemets arkitektur og tilhørende sikkerhed.

### 4.1 Swagger

Vi har udarbejdet en swagger dokumentation[5] for at få et overblik over hvordan API'et skal se ud. Swagger er en JSON/YAML specifikation, som kan benyttes til at dokumentere et API's endpoints, samt hvilken data det håndterer. Swagger understøtter autogeneration af kodestruktur, som vi forsøgte at gøre brug af. Desværre bliver det hele generet i en enkelt fil og alle navne er uoverskuelige, da de er autogenereret. Den største fordel for os ved swagger er, at den også kan bruges til at generere en hjemmeside, som giver et overblik over api'et.

### 4.2 Arkitektur

Systemet's designvalg faldt på en REST-baseret[6] systemarkitektur, hvor de forskellige klientapplikationer kommunikerer med en serverapplikation via *stateless* netværkskald. Formålet ved at implementere det som en REST-baseret applikation med tilhørende API og ressourcer i stedet for statiske event handlinger så som SOAP. Valget er taget for at gøre systemet mere fleksibelt og hybrid på tværs af applikationer og platforme. De vil alle sammen kunne arbejde med den samme back-end, ved at benytte sig af REST kald der bliver valideret og udført efterfølgende i back-enden. Dette medfører, at logikken kun skal implementeres en gang, og efterfølgende kan implementeres via en webapplikation, en mobil applikation eller desktop applikation ved at implementere en front-end der kan sammenkoble forbindelsen.

Endvidere er back-enden faldet på MVC som det arkitektoniske mønster, der er baseret på tre lag, et modellag, der arbejder med *data access*, et controller-lag, der interagerer med en user gennem en række netværkskald, der så afgør, hvilket view brugeren skal præsenteres for. View-laget er modificeret lidt i forhold den semantiske betydning i MVC, da man normalt bliver præsenteret for en grænseoverflade, men med en REST-baseret tankegang, er det fremlagt med JSON og eventsne er JSON request.

### 4.2.1 Modellaget

Vores applikation designes på en simplificeret modificering af den generelle MVC struktur, frem for at præsentere hver tabel med en model, har vi valgt at designe databasen og bibeholde dem som den foretrukne repræsentation af sig selv. Dette indebærer, at når forretningslogikken skal modificere og redigere tabellerne, skal retur og afsendelsen sammenkobles med marshall objekter, der er med til transformere processen og kompleksiteten af tabellen. Et marshall objekt indeholder derfor strukturen på JSON objekterne, men gør det muligt at repræsentere det som objekt orienteret objekter. Endvidere forhindrer det i at når man skulle kalde en projektmodel, så ville den ikke returnere hele projektet i form af nedrivninger og lister, men kun den information man skulle bruge i gældende kald og derfor minimerede datamængden i hver returforsendelse.

### 4.2.2 Controller laget

Controller laget behandler efterspørgsler efter indhentning og ændring af data samt præsentation logikken for view-laget. Den overordnede designarkitektur for controller laget er, at de vedligeholder forretningslogikken og validerer de forskellige marshall objekter samt benytter sig af de rigtige databasekald uden, at der er risiko for SQL-injection eller man overtræder andre sikkerhedsnormer - for eksempel verificering af en brugers rettigheder.

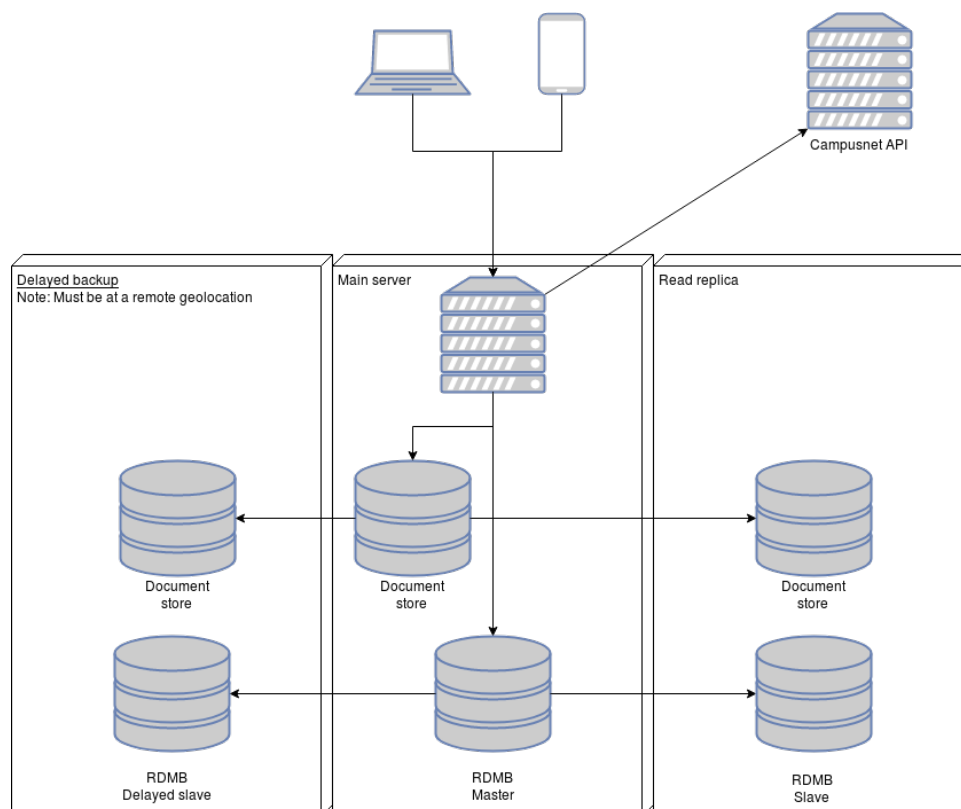
### 4.2.3 View laget - API struktur

Til dette projekt er designet af view laget, faldet på et at give en logisk og dynamisk overflade, nemlig at skabe ressourcer til hver metode og give handler til de forskellige brugere i systemet. Dette medfører derfor, at ressourcerne bygges videre på User Stories, da handlerne generelt arbejder med data, - at kunne læse, skrive, opdatere og slette data, den såkaldte CRUD implementering. Endvidere er ressourcerne kategoriseret og inddelt så man kun kan tilgå det, man har rettigheder til. I Tabel 5, ses de forskellige ressourcer versionsstyring systemet, den viser; hvilken type request, API endpointet og hvad der kræves af efterspørgslen. Tabel oversigten viser yderligere, hvad de forskellige brugere er tilknyttet, men den skal også læses som, at en slutbruger har rettigheder til de resterende brugeres API endpoints, og at en admin har adgang til en users API endpoints. Det medfører derfor også, at disse endpoints er sikre og kræver data validering og verificering på brugeren og brugerens rolle - hvilket vil blive beskrevet dybere i afsnittet om sikkerhed.

Tabel 5: API Ressourcer

Superbruger		
Type	Endpoint	request
POST	/admin	{data}
PUT	/admin/{adminid}	{data}
DELETE	/admin/{adminid}	
GET	/admin/	
GET	/admin/{adminid}	
Admin		
Type	Endpoint	request
POST	/projects/	{data}
PUT	/projects/{projectId}	{data}
DELETE	/projects/{projectId}	
POST	/users/	{data}
PUT	/users/{userid}	{data}
DELETE	/users/{userid}	
GET	/users/{userid}	
GET	/users/	
POST	/categories/	{data}
PUT	/categories/	{data}
DELETE	/categories/{categoryid}	
User		
Type	Endpoint	request
POST	/components/	{data}
PUT	/components/{componentTypeID}/{componentID}	{data}
DELETE	/components/{ComponentTypeID}	
POST	/components/{ComponentTypeID}	{data}
PUT	/components/{ComponentTypeID}	{data}
DELETE	/components/{componentTypeID}/{ComponentID}	
POST	/components/{componentTypeID}/{componentID}/documents	{data}
PUT	/components/{componentTypeID}/{componentID}/documents/{documentID}	{data}
DELETE	/components/{componentTypeID}/{componentID}/documents/{documentID}	
Guest		
Type	Endpoint	request
GET	/components/{componentTypeID}/{componentID}/documents/{documentID}	
GET	/components/{componentTypeID}/{componentID}/documents	
GET	/components/{componentTypeID}/{ComponentID}	
GET	/components/{ComponentTypeID}	query
GET	/components/	
GET	/categories/{categoryid}	
GET	/categories/	
GET	/projects/{projectId}	
GET	/projects/	

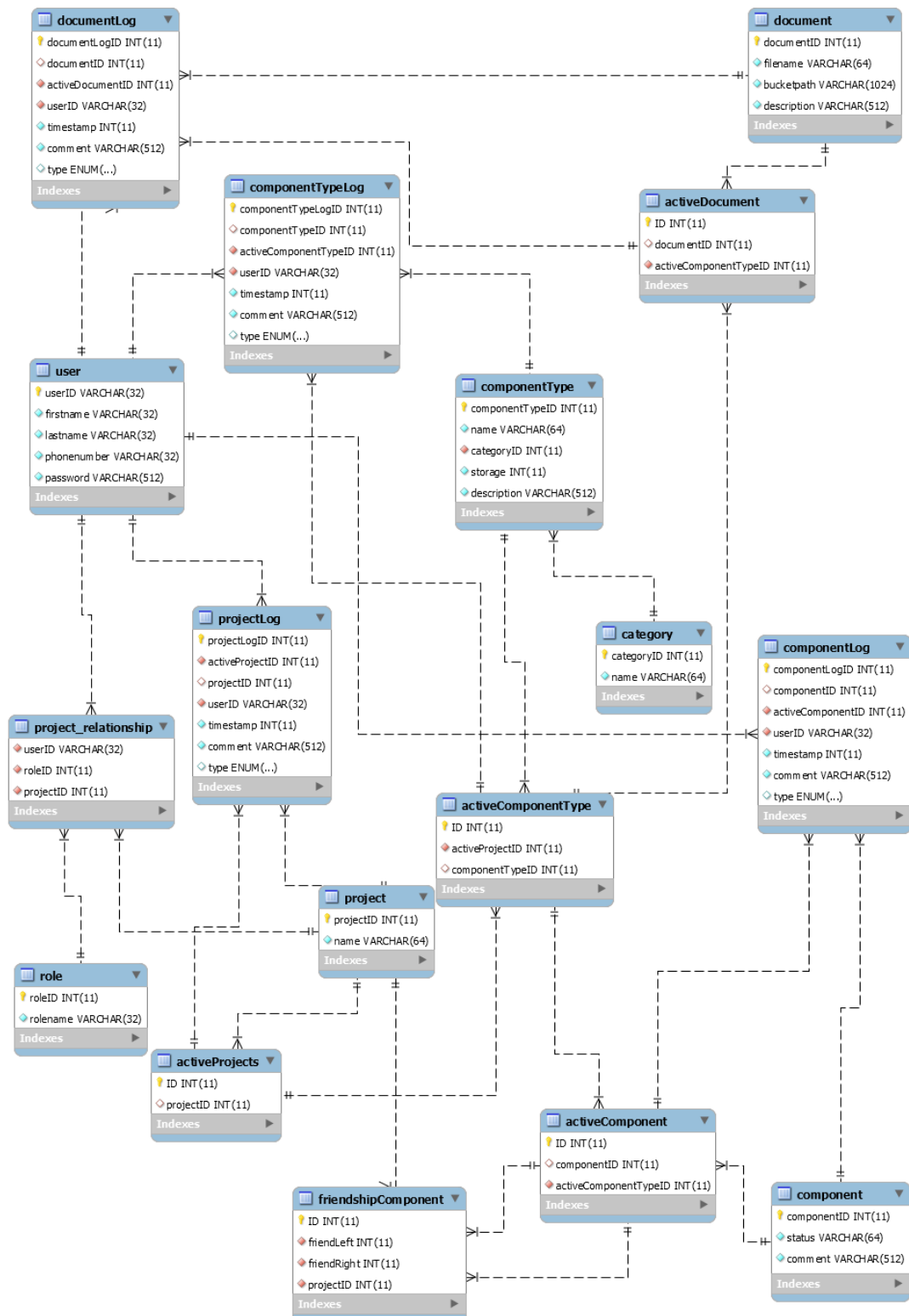
## 4.3 Deployment Diagram



Figur 4: Deployment diagram

Selve systemet skal bestå af tre server instanser for at maksimere tilgængeligheden og holdbarheden. De tre instanser består af hoved instansen, som kan ses i midten på Figur 4, read instansen, som kan ses til højre, og en backup instans, som kan ses til venstre. Både backup og read følger slave strukturen for hovedinstansen, men backup er en time forsinket i tilfælde af fejl. Read instansen bliver brugt i tilfælde af at hovedinstansen er overloaded. Det kan ske i tilfælde af, at mange store filer er i gang med at blive oprettet, hvilket foresager en stor mængde IO på hovedinstansen. Hvis andre forsøger at tilgå nogle andre filer fra serveren samtidig, så kan de i stedet hente dem fra read replika. Både MySQL og dokumenterne er replikeret.

### 4.3.1 Databasestruktur



Figur 5: EER-Diagram



I Figur 5, ses et EER diagram over databasen. I dens fulde udfoldelse kan den virke intimiderende; men frygt ikke, her bliver den forklaret.

I analyse afsnittet blev de basale database-elementer forklaret: Project, Componenttype, Component, Dokument og Category. De fremgår selvfølgelig i databasen med deres respektive navne.

Det er dog ikke muligt at se ud fra disse elementer, hvilke der er de nyeste, og hvilke der bare er gamle versioner. Vi har derfor lister af aktive elementer. Disse tabeller er navngivet 'active[navn på element]', f.eks. 'activeDocument'. Hver række i de aktive tabeller indeholder et id som altid kan benyttes til at referere til det nuværende aktive element. Denne række kaldes for aktivelementet. Denne løsning var nødvendig, fordi der aldrig må slettes noget fra databasen, så hvis man hurtigt vil finde ud af hvilke versioner, der er aktuelle skal man slå op i aktivlisten. Man behøver altså ikke slette noget, bare fjerne det fra listen af aktive elementer. Hver gang der bliver lavet en ændring, laves der et nyt element, og det nye element bliver så refereret af aktivelementet.

For at kunne følge med i ændringer bliver der lavet et log-entry, hver gang man ændrer et af elementerne i databasen. Loggen holder styr på, hvem der lavede ændringen, hvornår, en forklaring af ændringen og hvilken type ændring det var. Logs kan bruges til at rulle tilbage i tiden. Hvis man finder ud af, at man har lavet en fejl i systemet, kan man rulle tilbage til en tidligere version af elementet. For at rulle tilbage skal man kigge alle elementlogsne igennem, når man finder den log, der skal rulles tilbage, opdaterer man bare aktivelementet til at pege på det gamle element, som står i loggen før den angivne log. Dvs. man fortæller systemet hvilken log der skal rulles tilbage og så ruller den til versionen lige før det.

For at sikre, at det ikke bare er alle og enhver der kan ændre på databasen har vi brugere. En bruger er bundet til en rolle og et projekt. Det betyder, at brugeren kun kan arbejde på elementer i det projekt hvis deres rolle tillader det. Det er også krævet for at kunne lave logs, over hvad der er blevet ændret.

## 4.4 Normaliseringsanalyse

Der er blevet lavet en normaliseringsanalyse over databasen, for at se hvilke normalformer den lever op til. Disse normalformer er beskrevet nedenfor.

#### 4.4.1 Første normalform

Den første normalform kan hver tabelrække identificeres af en unik nøgle og indeholder kun atomare værdier.

Dette bliver overholdt i tabellerne, da de alle har en primær nøgle, den eneste tabel der ikke har en primær nøgle er `project_relationship`, dette har den ikke da den består af tre fremmed nøgler, dog skal det bemærkes at fremmed nøglerne tilsammen skal være unikke og derfor udgør en primærnøgle. Der er også kun atomare værdier i databasen, da der ingen steder bliver gemt flere værdier i et felt. En ting som kan bryde med den første normalform er, at der i nogle af tabellerne kan være en null værdier. Det er der dog besluttet, at i denne database er null værdier okay, da det er set som en nødvendighed at have null værdier de steder, hvor der er blevet slettet en ting, da disse ting kan blive bragt tilbage igen. Havde vi ikke tilladt null, så havde fremmed nøglerne ikke været mulig i log tabellen.

#### 4.4.2 Anden normalform

En database overholder anden normalform, hvis alle ikke primær attributter er afhængige af hele primær nøglen og første normalform er overholdt.

Da der kun er en tabel hvis primær nøgle ikke består af en enkelt attribut, er alle de andre i anden normalform. Den tabel som består af tre attributter i primær nøglen er `project_relationship`, den er også i anden normal form da der ikke er nogen andre attributter, som kun afhænger af en del af primær nøglen. Dette vil sige, at hele databasen er i anden normalform, hvis null værdier er godkendt i første normalform.

#### 4.4.3 Tredje normalform

En database er i tredje normalform hvis attributter afhænger af andet end primær nøglen og den samtidig er i anden normalform.

Et eksempel på en tabel som ikke er i tredje normalform kan ses i Tabel 6.

Tabel 6: Ikke i tredje normalform

componentID	componentTypeID	componentTypeName
1	1	Søm
2	2	Metalplade
3	1	Søm

Denne tabel er ikke i tredje normalform, da man ud fra componentTypeID kan se hvilket navn componentTypen har, det vil sige at en attribut afhænger af noget andet en primærnøglen, som i dette tilfælde er componentID. For at få tabellen i tredje normalform er den blevet delt op i to tabeller. Disse to tabeller vil komme til at se ud som i Tabel 7 og 8.

Tabel 7: Component	
componentID	componentTypeID
1	1
2	2
3	1

Tabel 8: ComponentType	
componentTypeID	name
1	Søm
2	Metalplade

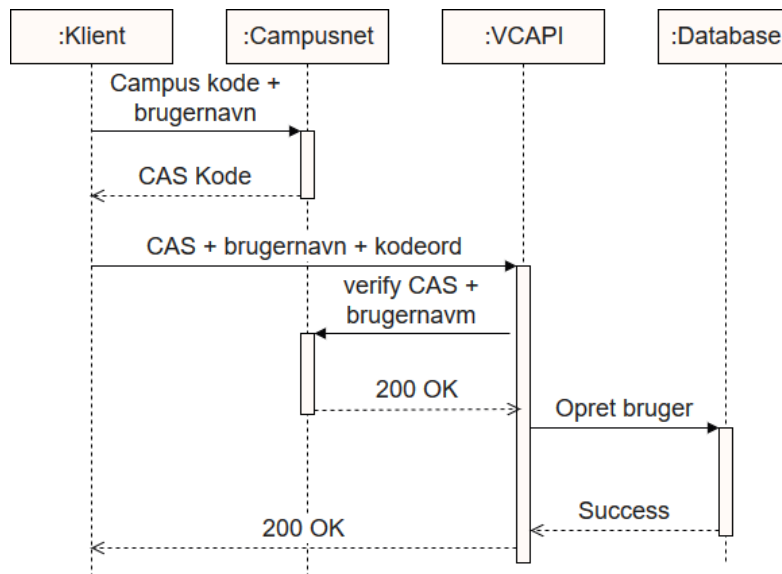
Nu er de i tredje normalform, da der ikke længere er noget som afhænger af andet en primær nøglen. Dette er blevet gjort på alle tabeller, som ikke var i tredje normalform, for at få databasen til at overholde tredje normalform.

## 4.5 Sikkerhed

Følgende afsnit beskriver sikkerheds designvalget af systemet, blandt andet hvordan man opretter sig og hvordan man verificere om man er logget sikkert ind.

### 4.5.1 Oprettelse

Oprettelse i systemet skal benytte sig af Campusnet API [7] hvor CAS-koden, som man kan få tildelt via deres API, skal sendes sammen med brugernavn og kodeord ved oprettelse. Brugernavnet skal være det samme, som der bliver brugt på kampusnet, men kodeordet kan være forskelligt. CAS-koden giver kun begrænset adgang til ens campusnet profil, så derfor er det mere sikkert end at give adgang til end ens faktiske campusnet login og vi har derfor valgt at bruge kun den til validering. CAS-koden bliver brugt til at verificere, at man går på DTU ved at benytte koden til at tilgå UserInfo for API.



Figur 6: Et diagram over oprettelsesprocessen

Adgangen kræver, at man sender både CAS-koden og brugernavn med i forespørgslen hvilket resulterer i en 200 OK svar hvis de to passer sammen. Hvis svaret er 200 OK, så bliver brugeren oprettet(givet den ikke allerede findes i forvejen). Når brugeren er oprettet starter de med rank gæst, og det er op til de kursusansvarlige tildele dem en højere rank.

#### 4.5.2 Login

Login fungerer ved, at man sender sit brugernavn samt kodeord af sted hvorefter det verificeres. Hvis login er succesfuld, så skal der oprettes en JWT, som indeholder ens brugerid. Denne token skal sendes med hver efterspørgsel hvor, at serveren verificerer brugeren's rettigheder ved brug af brugerid'et i token.

#### 4.5.3 JSON Web Token

JSON Web Token[8] bliver brugt til autorisering, da det er et simpelt format, som er understøttet på mange platforme. Selve token kan i princippet indeholde rettighederne for brugeren til hvert projekt udover brugernavnet, men for at holde det simpelt og for at ændringer i brugerrettigheder tager kraft med det samme, så skal den for nu kun indeholde brugerid'et. JWT har et audience felt, som fortæller hvor den udstedte token er gyldig. Dette felt skal være ens på tværs af serverne. Der er også en udløbstid for token, hvilket er sat til seks timer. Et login er påkrævet igen efter seks timer.

#### 4.5.4 Hashing

Alle kodeord skal gemmes i hashet format i databasen. Da database adgang er frit tilgængelig for alle, som kommer til at arbejde på API'et, så er det meget vigtigt at alle kodeord bliver gemt i databasen i hashed format og, at der benyttes et salt på hvert password. Salt skal være forskelligt fra bruger til bruger for at undgå en studerende opretter et *rainbow table*, som kan benyttes for alle brugere i databasen.

#### 4.5.5 HTTPS

Systemet skal benytte HTTPS når det kommer i produktion, da alle filer er sendt i plaintext. Hvis HTTPS ikke er benyttet, så kan filerne blive opsnapet i transit. Derudover er følsomme informationer så som CAS-koder, kodeord og brugernavne også sendt i plaintext.

### 4.6 Versionstyring i databasen

Som det kan læses lidt om i analyse afsnittet så er versionstyring vigtigt i dette projekt, derfor er designet også vigtigt. Måden det er besluttet at håndtere versionstyring er at elementer der en gang har været i systemet aldrig bliver slettet igen.

I stedet for at slette elementer er der i stedet lavet en liste over aktive elementer, og hvis man ønsker at slette noget bliver den blot fjernet fra listen. Dette er det samme hvis man ønsker at opdatere et element, så bliver det gamle element blot erstattet med det nye. For at brugerne ikke selv skal huske hvilket element der er det aktive, så er det blevet besluttet at i listen over aktive elementer er et aktivt id og så en reference til det element som er det aktive. Dette betyder at selvom man opdaterer et element så vil det ligne at den beholder det samme id som den havde før. Der er dog blevet oprettet et nyt element med et nyt id, men dette id bliver gemt i den aktive liste, hvis et element bliver slettet så peger referencen blot på null i stedet.

Tilbagerulning er også en vigtig del af versionstyring, dette skulle der derfor også laves en plan for. Her er det blevet besluttet at man kan se i loggen over alle de ændringer der er sket for et element. I loggen står der hvem, hvad og en kommentar på den ændring der har været. Hvis man ønsker at gå tilbage til en tidligere version skal man vælge den log som er den sidst log som man ønsker at rulle tilbage. Det vil sige at hvis man kan se at navnet et blevet ændret i en log, og man ikke ønsker at det skulle være sket, så er det den log man skal vælge. Samtidig bliver alle de andre ændringer der evt. er sket

efterfølgende også rullet tilbage. I loggen skal der gemmes hvilket aktivelement der er ændret på og hvordan det så ud før. Med det menes der, at loggen skal gemme de to id'er på det aktivelement og id'et på det gamle element, da disse skal bruges hvis der skal rulles tilbage.

## **4.7 Hierarki**

U databasen er der lavet en tabel, som skal gemme på forholdene i mellem to komponenter. Tabellen indeholder to komponenter, som har et indbyrdes forhold, hvordan dette forhold er kan man ikke se, blot at de har et. Det der bliver gemt i tabellen er to id'er, som er to aktive komponenter.

## 5 Implementering

Systemet er implementeret ved brug af frameworket ASP.NET, lavet til at køre på runtimen .NET Core II og er skrevet i C#. Frameworket bygger på reflection hvor, at alle klasser, som nedarver fra **Controller**, bliver registreret som et endpoint, kaldet en controller, i api'et. Hvordan controllerne bliver tilgået er defineret ved brug af C#'s attributter<sup>1</sup> enten på controlleren eller dens metoder. Et præfix til alle metoderne kan defineres ved brug af **Route(path)** attributten. Vi har benyttet det til at skabe et nested hierarki, så man kan følge f.eks. hvilket projekt man er i. Attributterne **HttpGet(endpoint)**, **HttpPost(endpoint)**, **HttpPut(endpoint)** er blevet brugt til hver metode og definerer hvilken ressource, og metode der bruges for at kalde metoden. Frameworket har også support for konfigurationsfiler og autorisering.

### 5.1 MVC i praksis

Systemet kan ses som en Model-View-Controller struktur hvor API'et udgør vores view, modellerne udgør det som bliver sendt frem og tilbage via API og dem som behandler modellerne er vores controllers(endpoints).

#### 5.1.1 Modellaget

Modellaget er designet til brug af JSON. Modellerne bliver modtaget som C# objekter i controllerne. Det bliver gjort automatisk af ASP.NET, så API'et kan også automatisk understøtte XML, men dette er ikke testet. Vi har lavet en attribut **VerifyModelState** for at verificere, at modellerne er korrekte når de bliver modtaget. Korrektheden af modellerne er defineret ved attributterne på klasserne, så som **Required**. De modeller eller controllerne modtager kan komme fra efterspørgslen's body, rute eller query. Hvor parameterne kommer fra bliver beskrevet ved brug af attributterne **FromRoute** eller **FromQuery**.

#### 5.1.2 Controller laget

Controllers har ansvar for adgangskontrol og kommunikation med databaselaget. Alle endpoints på nær login og oprettelse kræver, at man har en gyldigt JWT token. ASP.NET kan verificere det automatisk ved brug af **Authorize** attributten. Hvis JWT token ikke

---

<sup>1</sup>Attributter har formatet `[Type(...)]` og kan bruges til at tilføje metadata til elementer

er gyldig, så bliver status koden 401(Unauthorized) sendt tilbage og controlleren bliver aldrig kaldt. Alle controllers benytter sig af dependency injection, hvilket automatisk er håndteret af ASP.NET. Controllerens constructor tager imod et interface og ASP.NET sørger for, at den rigtige klasse bliver brugt. Vi har gjort brug af dette til primært for database adgang, men også hash konfiguration for f.eks. LoginController. Alle vores controllers har adgang et *projectId* parameter siden vores rute er hierarkisk opbygget. Da adgang er pr. projekt, så benyttes projekt id til at hente brugerens adgang for det specifikke projekt. Hvis brugeren har adgang til at udføre handlingen, så foretager controlleren et database kald, som udfører handlingen. Det meste er håndteret i databasen og vores controllers er bare et tyndt lag ovenpå, som håndterer rettighederne.

### 5.1.3 View laget

Vores view lag er den samlede data man kan hente ned ved brug af API'et. Vores API er RESTful og kan det benyttes til at vise den data, som bliver tilgået ved f.eks. en hjemmeside eller app.

For at vise hvordan kommunikationen fungerer er der lavet et eksempel på hvordan man opretter et nyt dokument og på hvordan man opdaterer et komponent.

Det første eksempel som kan ses i Listing 1 er et eksempel på en post metode, som bliver brugt når man lægger et dokument op første gang. Da filen man gerne vil ligge op bliver gemt på serveren i et andet filsystem, kan man ikke se selve filen, kun filnavnet. Det sidste er en kommentar, som bliver brugt i loggen, så man kan se hvorfor de forskellige ændringer er blevet lavet. Hvis alt går godt med efterspørgslen vil klienten modtage 201(Ok), ellers vil klienten modtage en anden fejlkode der reflekterer hvad der gik galt.

Listing 1: Create document request

---

```
curl
-H "Content-Type: application/json"
"Authorization: Bearer
    eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aW1lYWQiOiJ0b211Ym9keSI6Im5iZiI6MTUxMjI5Nz
U5MiwiaXhwIjoxNTEyMzE5MTkyLCJpYXQiOiE1MTIyOTc1OTIsImF1ZCI6InJvYWRydW5uZXItYXBpIn0.03s
XJjFCNzUA0deoa4sSyk_hscG3GZIjY1PBdbZ_VM"
-X POST
-d '{ "model" : {
    "filename" : "Arbejdstegning",
    "description": "Dette er en arbejds tegning over en motor"
  },
  "comment" : "Jeg har uploadet en arbejdstegning" }'
```



```
http://domain.dk/api/project/{projectId}/componentType/  
{componentTypeId}/documents/
```

---

#### Response

HTTP/1.1 201 OK

Date: Mon, 03 Dec 2017 15:21:22 GMT

Content-Type: text/plain; charset=utf-8

Server: Kestrel

Transfer-Encoding: chunked

Location: http://domain.dk/api/project/{projectId}/  
componentType/{componentTypeId}/documents/{documentId}

---

Et eksempel på et PUT metoded kan ses i Listing 2, disse metoder bliver brugt til update, delete og rollback funktioner. Denne PUT metode bliver brugt til at opdatere et komponent. Som det kan ses, bliver der sendt en model hvori der er en status og en kommentar, så man også ved hvilket komponent det er. Den anden kommentar er til loggen, så man kan skrive hvorfor man har opdateret noget på komponentey. En anden ting som man skal lægge mærke til her er den url der bliver brugt, der skal der tilføjes det komponentID som man ønsker at opdatere. Reponse er i dette tilfælde 200 hvis det er gået godt.

---

#### Listing 2: Update component request

---

#### curl

-H "Content-Type: application/json"

"Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm9keSI6Im5iZiI6MTUxMjI5NzU5MTUzXhwIjoxNTEyMzE5MTUzOTI6ImF1ZCI6InJvYWRydW5uZXItYXBpIn0.03sXJjFCNzUA0deoa4sSyk\_hscG3GZIjY1PBdboZ\_VM"

-X PUT

-d '{ "model" : {  
          "status" : "Skrue ikke i brug",  
          "comment": "Runde skurer"  
      },  
      "comment" : "Jeg har fjernet skruen, bliver ikke brugt mere"}'

http://domain.dk/api/project/{projectId}/componentType/  
{componentTypeId}/components/{componentId}

---

#### Response

HTTP/1.1 200 OK

Date: Mon, 03 Dec 2017 15:21:22 GMT  
Content-Type: text/plain; charset=utf-8  
Server: Kestrel  
Transfer-Encoding: chunked

---

Begge request har deres header tilfælles, da man for at kunne uploade og ændre skal have en bruger, og denne bruger skal have de rigtige rettigheder til det projekt. Derfor bliver der sendt en token med i headers, som bliver kaldt for Authorization, den bliver brugt til at være sikker på, at brugeren har de rettigheder som man skal have.

## 5.2 Konfiguration

Konfigurationen af API'et så som database og hash parameters findes henholdsvis i filerne appsettings.json og settings.json. Filerne er registreret ved startup og kan tilgås ved brug af `Configuration.GetSection` og kan tildeles kontrollers ved at registrere i `ConfigureService` ved brug af `services.Configure` f.eks. Får `LoginController` hash settings ved brug via Dependency Injection af

```
services.Configure<JWTOptions>(Configuration.GetSection("TokenSettings"));
```

## 5.3 Asynkrone kald

Systemet benytter sig af asynkrone kald, hvilket er supporteret af ASP.NET. Det fungerer ved, at metoder defineret med `async` kan kaldes med `await`. Dette foresager i mange tilfælde, at kaldet er non-blocking og derved gør det muligt at servicere flere brugere ad gangen.

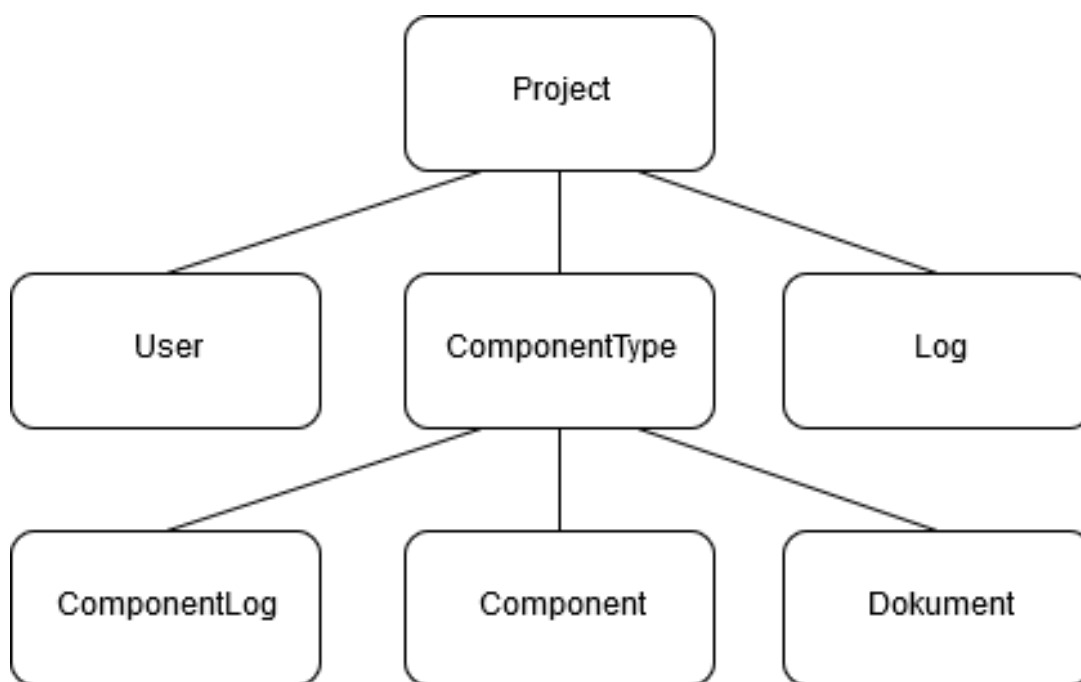
## 6 Database

MySQL's officielle C# connectors er ikke supporteret på .Net Core II, så systemet benytter i stedet den uofficielle `MySqlConnection`. For at forbinde til databasen bruges `DatabaseConnector`, som sørger for, at den rigtige connection string bliver brugt. Connection string er defineret i `appsettings.json` og indeholder ip, port og login information. Connection string bliver givet til `MySQLConnector` via dependency injection og `MySQLConnector` bliver givet til repositorerne, også ved brug af dependency injection. `DatabaseConnector` er registreret som en singleton og alle repositories deler derved den samme instans. Dette er for at undgå at skulle autorisere med databasen for hver

request. Når serveren bliver lukket ned, så bliver `DatabaseConnector`'s `Dispose` automatisk kaldet. `DatabaseConnector` har metoden `Create`, som returnerer en ny forbindelse til databasen, som kan benyttes til queries. `Dispose` er ikke automatisk kaldet her, så systemet gør brug af C#'s `using` scope-statement, som automatisk kalder `Dispose` når objektet går ud af scope. Bliver `Dispose` ikke kaldet, så kan `MySQLConnector`'s interne connection pool løbe tør, selvom `Connection` ikke bliver brugt.

## 6.1 Database problem

Før den nuværende implementation af systemet havde vi lavet en løsning der brugte MongoDB og NodeJS-Mongoose. Source koden til denne implementation kan nu findes i repositoret VCAPI-Legacy[9]. MongoDB virkede som et ideelt valg til databasen, da den naturligt passede til vores struktur. MongoDB virker lidt som et filsystem, og det vil derfor være nemmere at tilføje filer. Her er en model der viser database strukturen:



Figur 7: MongoDB model

I MongoDB har man hvad der kaldes 'dokumenter' som svarer til et entry i databasen. Dokumenter kan have sub-dokumenter. Sådan var det meget intuitivt at forstå hvad der hørte til hvad. Det var dog svært at lave logsystemet da det skulle referere rundt i databasen. Løsning var, at alle projekter havde en 'projekt-log' som holdt styr på alle meta-ændringer, såsom ændringer af navne på MongoDB dokumenterne. Hver komponenttype havde en log som holdt styr på reelle ændringer i komponenttypen og dens

underdokumenter 'components' og 'documents'. Når en ændring bliver lavet i et subdokument, håndterer MongoDB det som en ændring i hoveddokumentet og alle dets subdokumenter. Der var derfor bekymring om at systemet ville være langsomt under pres, da der kunne forekomme store træer af indlejrede subdokumenter. Det ville betyde, at alle dokumenter i et projekt skulle 'opdateres' af MongoDB for hver lille ændring.

Da databasen næsten var klar fandt vi dog et kritisk problem med databasen. Da MongoDB er et noSQL sprog, er der ikke nogen måde at sikre, at en række operationer bliver udført atomisk. Det ville betyde, at hvis der skete en fejl på serveren mens to operationer på databasen blev udført ville man risikere, at kun den første operation blev udført og den anden 'glemt'. Et eksempel kunne være når der skal oprettes en log, efter der er sket en ændring på et dokument. Der var kun en lille chance for, at det kunne ske, men det ville være katastrofalt for systemet, hvis der ikke var sammenhæng.

Vi tog derfor en beslutning om at starte forfra med en .NET løsning med MySQL database. MySQL har 'transactions' som løser vores problem ved at sikre at alle operationer bliver udført eller rullet tilbage, hvis der sker en fejl. Beslutningen blev taget på det belæg, at databasen skulle bruges til andre studerendes arbejde og vi ikke ville være skyld i spildt tid eller arbejde. Da systemet er så centralt i versionsstyrings projektet er det også vigtigt, at det virker godt og pålideligt, så andre kan arbejde videre med det.

Det var en drastisk beslutning, men vi havde lært meget om systemet fra vores forrige implementering og vi var derfor hurtigt i gang med at implementere det nye system. Vi havde allerede en klar forståelse af hvad der skulle være i databasen.

## 6.2 Versionering

For at få versionering implementeret er der blevet brugt en række stored procedures. De stored procedures er dem som sikrer, at den versionstyring vi har designet bliver overholdt.

### 6.2.1 Stored procedures

For at sikre for, at databasen er atomar, er der oprettet en række stored procedures, der skal bruges til at oprette, opdatere, slette og rulle tilbage. Nogle af disse stored procedures er beskrevet nedenfor.

Da dette system altid skal kunne rulles tilbage, kan man ikke slette de forskellige dele, derfor er der blevet lavet f.eks. Listing 3, som sletter et dokument. Det er samme princip,

som der bliver brugt i alle slette procedures. Som det kan ses bliver et dokument ikke slettet, men i stedet bliver den active document tabel opdateret til ikke længere at pege på et dokument men i stedet for at pege på NULL. Dette bliver gjort, så hvis en sletning bliver fortrudt, kan man stadig finde det dokument der ellers var blevet slettet. Det sidste der bliver gjort er at lave en log over den sletning, som er blevet lavet, i denne log bliver det gamle dokument gemt, så man senere kan gå tilbage og finde det gamle dokument.

Listing 3: SQL Delete stored procedure

---

```
CREATE DEFINER='root'@'localhost' PROCEDURE 'deleteDocument'(  
    IN activeID INT,  
    IN userID VARCHAR(32),  
    IN logComment VARCHAR(512),  
    OUT err INT)  
BEGIN  
    DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING  
    BEGIN  
        SET err = 1;  
        ROLLBACK;  
        END;  
        SET err = 0;  
        START TRANSACTION;  
            SELECT documentID INTO @oldDocumentID  
            FROM activeDocument WHERE ID = activeID;  
            UPDATE activeDocument SET documentID = NULL  
            WHERE ID=activeID;  
            INSERT INTO documentLog  
            VALUES(NULL, @oldDocumentID, activeID, userID,  
            UNIX_TIMESTAMP(NOW()), logComment, 'deleted');  
            COMMIT;  
        END
```

---

En anden stored procedure, som er blevet lavet er tilbagerulnings funktionen. Et eksempel på denne kan ses i Listing 4. Dette eksempel viser hvordan en komponent bliver rullet tilbage. Det første der bliver gjort, er at finde de forskellige id'er på den nuværende komponent og den man gerne vil rulle tilbage til samt id'et på den aktive komponent man ønsker at ændre. Disse id'er skal bruges til at opdatere den aktive komponent tabel samt loggen. Det kan ses, at den aktive komponent bliver sat til den tidligere id, som der er fundet i loggen, og at der samtidig bliver lavet en ny log, hvor i man kan se, at der er

blevet rullet tilbage, og fra hvilken komponent man har gjort det.

Listing 4: SQL Rollback stored procedure

---

```
CREATE DEFINER='root'@'localhost' PROCEDURE 'rollbackComponent'(  
    IN logID INT,  
    IN userID INT,  
    IN commentParam VARCHAR(512),  
    OUT err INT)  
BEGIN  
    DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING  
    BEGIN  
        SET err = 1;  
        ROLLBACK;  
        END;  
        SET err = 0;  
        START TRANSACTION;  
        SELECT activeComponentID INTO @activeID  
        FROM componentLog WHERE componentLogID = logID;  
        SELECT componentID INTO @oldComponentID  
        FROM activeComponent WHERE ID = @activeID;  
        SELECT componentID INTO @ID  
        FROM componentLog WHERE componentLogID = logID;  
  
        UPDATE activeComponent SET componentID = @ID  
        WHERE ID=@activeID;  
  
        INSERT INTO componentLog  
        VALUES (NULL, @oldComponentID, @activeID, userID,  
        UNIX_TIMESTAMP(NOW()), commentparam, 'rollback');  
        COMMIT;  
    END
```

---

En ting alle stored procedures, som kan ændre noget i databasen har tilfælles, er en out parameter, denne parameter kan enten være 0 eller 1 eller -1 eller id hvis det er en create stored procedure. Hvis out er 0, så er det gået godt og hvis den er 1, så er der sket en fejl undervejs. Grunden til at der bliver brugt -1 eller id i create er, at man så på den måde med det samme kan se hvilket element man har lavet, hvorimod -1 er, hvis der er sket en fejl. Hvis der sker en fejl undervejs så bliver der ikke lavet nogle ændringer overhoved, dette er netop for at holde vores database atomar. Hvis dette ikke blev gjort kunne

man risikere at have en database, som ikke stemte overens på tværs af tabeller og dette skal undgås, det er derfor, at der bliver kaldt rollback hvis der sker en fejl. Databasen ville så se ud som den gjorde før stored proceduren fejlede. Dette kunne også være hvis databasen brød ned eller forbindelsen forsvandt undervejs.

## 6.3 Sikkerhed

API'et er blevet implementeret med fokus på sikkerhed. Alt database adgang sker gennem stored procedures og ved brug af en MySQL bruger, som ikke har rettigheder for at opdatere eller slette data, da alt skal versioneres.

### 6.3.1 Oprettelse af bruger

Oprettelse af bruger fungerer ved at sende en POST request til `/api/signup` med et JSON, der har felterne `username`, `firstname`, `lastname`, `password` og `CAS` kode. Ved oprettelse foretages der et asynkront kald til campusnet's api for at bekræfte opretterens identitet. I tilfældet af, at `CAS` koden og brugernavnet ikke passer sammen, så returnerer API'et 401. Hvis `CAS` og brugernavn passer, så foretages der et database kald via procedures `createUser`. `CreateUser` tager imod fornavn, efternavn, `password` og brugernavn. `CAS` koden bliver ikke gemt efter verificering. Følgende er et eksempel på en efterspørgsel om oprettelse af bruger:

Listing 5: Request for oprettelse

---

```
curl
-H "Content-Type: application/json"
-X POST
-d '{  "username" : "sXXXXXX",
      "password" : "passwrđ",
      "firstname" : "Some",
      "lastname" : "one",
      "CASCODE"  : "AOGIJeaEFas"
    }'
http://domain.dk/api/signup
```

---

Response  
HTTP/1.1 200 OK  
Date: Mon, 04 Dec 2017 09:47:34 GMT  
Server: Kestrel

### 6.3.2 Campusnet API adgang

Der er behov for to ekstra headers udover CAS koden for at få adgang til campusnet's api. Disse parametre kan anskaffes ved at registrere din app til at få adgang til campusnet's api. For økobil har vi fået parameterne i tabel 9.

x-appname	Opslagsystem for økobil
x-token	3ddfc095-5a62-4162-a058-5bc3784e36d7

Tabel 9: Adgang til campusnet api

### 6.3.3 Hashing af kodeord

Alle kodeord bliver hashet ved brug af hash funktionen PBKDF2 HMACSHA512[10]. Algoritmen er konfigureret til at bruge en hash størrelse på 512 bytes, et (for nu) hard-coded salt `oihergoi23r092ugwe` og at køre 10000 iterationer. Ideelt ville alle brugere have tilknyttet deres eget salt, som er gemt i en separat database tabel.

### 6.3.4 Login

Login fungerer ved at sende ens kodeord og brugernavn til `api/login` hvorefter en JWT token bliver returneret i body. Fejl kode 401 bliver returneret i tilfælde af ugyldigt login. Når `LoginControlleren` modtager en login efterspørgsel, verificerer den først login via `MySQLUserRepository`. `MySQLUserRepository` hasher kodeordet og sammenligner det med det hashede password, som ligger i databasen. I tilfældet af det er et match, så returnerer den true og ellers false. Hvis brugeren har givet et gyldigt kodeord, så genererer `LoginController` en `SecurityTokenDescriptor` som indeholder et `Claim` af typen `ClaimTypes.NameIdentifier`. Claims er en række af key value pairs som brugerens får krav på. I tilfældet af typen `ClaimTypes.NameIdentifier` så har brugeren krav på at kalde sig selv for det, som står i værdien for det claim. Vi kunne have gemt brugerens rettigheder til hvert projekt i brugerens claim, men for at simplificere systemet, så tjekker vi i stedet for databasen ved hver request ved brug af værdien i `ClaimTypes.NameIdentifier`. Havde vi gemt brugerens claim's i sin token, så ville vi også skulle tage caching i betragtning og tage højde for hvad der sker hvis brugerens adgang blev ændret indenfor de 6 timer hvor token er gyldig i. Når claims er blevet sat



så sættes tokens udløbstid til 6 timer. Audience, som er en betegnelse for hvor token er gyldig bliver sat til audience værdien fra settings.json. Til sidst bliver token signeret ved brug af HmacSHA256 hvor secretKey også kommer fra settings.json. Når token er signeret, så bliver den sendt tilbage til brugeren, som så skal benytte den til fremtidige efterspørgsler. Følgende er et eksempel på en cURL request for login:

---

Listing 6: Request for login

---

```
curl
-H "Content-Type: application/json"
-X PUT
-d '{
    "username" : "Somebody",
    "password" : "asdeasd"
}'
http://domain.dk/api/login
```

---

## 7 Vurdering

I de følgende sektioner vil vi beskrive vurderinger på projektet og hvad vi kan videreføre applikationen - der medfører derfor dokumentation på sikkerhed og test.

### 7.1 Revideret tidsplan

Undevejs i forløbet har vi revideret tidsplanen løbende, hvis vi er stødt ind i problemer som har taget lang tid at løse, har nogle af de andre dele i projektet fået en mindre prioritering. Dette er blandt andet gået ud over hjemmesiden, da denne ikke er blevet implementeret, da vi valgte at have fokus på at få API'en til at fungere.

En anden ting som heller ikke er blevet implementeret fuldstændig er det fil system som skal gemme de dokumenter som brugerne ligger op. Dette betyder at selve metoden til at uplaode et dokument er blevet lavet og testet, men dokumentet bliver ikke gemt på serveren, men de bliver dog registreret i databasen, bucketpath har bare ikke nogen værdi, som senere kan bruges til at læse filen.

Hjemmesiden er der ikke blevet arbejdet på i dette projekt, som det ellers var planen i starten. Dette er grundet nogle uforudsete problemer og en ændring af prioriteringer i forhold til de originale prioriteringer. Grunden til, at vi har valgt ikke at arbejde på hjemmesiden er at vi synes det var vigtigere at have en gennemtænkt og gennemarbejdet backend og database, end at vi senere hen opdagede at der var noget der ikke fungerede i backenden og vi ville blive nød til at lave den om.

#### 7.1.1 Problemer og løsning

Det største problem vi stødte på undervejs i projektet var i forhold til databasen, dette problem satte os flere uger bagud i forhold til den originale tidsplan. Det gjorde det fordi at vi blev nød til både at ændre hvilken database vi brugte fra MongoDB til MySQL. Samtidig ændrede vi også det sprog vi skrev backenden i fra Javascript til C#.

Da vi ændrede sproget til backenden opstod der et nyt problem, da vi ikke kunne få .dot core 2 til at fungere på mac, hvilket gik ud over en person i gruppen. Dog valgte vi at forstætte med dette, for ikke at komme endnu længere bagud i tidsplanen, da vi allerede var kommet langt bagud. I stedet for at der sad en der ikke kunne arbejde, arbejde vi sammen i par, så alle stadig arbejdede på projektet.

## 7.2 Procesvurdering

Alle i gruppen har formået at opbygge et rigtigt godt arbejdsmiljø, hvor samtlige gruppe-medlemmer har kunne styrke sig indenfor alle områder af studiet og forløbet. Det betyder at projektet konstant har været under hård udvikling og der samtidig er blevet revideret til de forskellige implementeringer undervejs og jævnført diskuteret hvad der kunne forbedres eller ændres. For eksempel versionstyringen af dokumenter og hvad der skulle gøres for at rulle tilbage efter en sletning.

Vigtigst af alt, har projektforløbet gjort at gruppen kom frem til en endegyldig plan for databasen og derfor besluttede i samarbejde at ændre strukturen på databasen og derfor også implementeringssproget på databasen og serveren, gruppen somhelhed mente at forsinkelsen for projektet var mindre relevant for denne handling.

## 7.3 Sikkerhedsrapport

Til dette projekt, er der blevet lavet en sikkerhedsrapport, som er med til at give en dokumenteret handlingsplan til fremtidige justeringer og sårbarheder. Det betyder at projektet ikke er færdigt, og der derfor mangler implementationer for at sikre systemet fuldt ud. Sikkerhedsrrapporten er desuden blevet opstillet efter *CIA [11]* standardene. De følgende underafsnit vil opstille en overordnet opsummering af CIA standarderne for projektet.

### 7.3.1 Confidentiality (Fortrolighed)

Eftersom at der bliver arbejdet med forskellige brugere og personfølsom data på både brugere og projektdokumenter, er det vigtigt at denne type data holdes fortroligt og adskilt for uvedkommende. Endvidere når man sletter data, så skal denne type data ”gemmes” væk og ikke være tilgængelig, men systemet skal stadig opbevare det til fremtidig tilbage rulning. Det her er et uhyrlig vigtigt da økobilen er et konkurrencepræget projekt og konkurrenterne derfor ikke skal have mulighed for at tilgå personfølsomme eller projektfølsomme data. Dette indebærer også til fremtidig brug af andre projekter, for eksempel kemi når der forskes i større projekter om hvad der skal ajourføres til uvedkommende gæster.

### **7.3.2 Integrity (Integritet)**

Er en bruger ikke tilknyttet et projekt, skal det ikke være muligt at arbejde på adskilt data man ikke er berigtiget til at arbejde med. Integritetten skal derfor være at man kun får lov at arbejde med data og projekter man er fortrolig med. Det er derfor også gældende at rollen man får udleveret til projektet opholder disse krav.

### **7.3.3 Availability (Tilgængeligt)**

Eftersom at det her er universitets- og projektmateriale der skal være åben heletiden til kurserne og forskere. Så er det vigtigt at programmet heletiden kører stabilt og og kan håndtere så mange brugere som muligt og holde sig ajour med log og store filer uden nedbrud.

### **7.3.4 Trusselvurdering**

Udfra CIA, kan der opstilles en række trusselvurderinger, deres sårbarheder og umiddelbare vurdering på hvor stort truslen og problemet er. Projektets største trussel er derfor faldet på om man kan lave SQL injektions, at udgive sig for at være en bruger både med rolle eller godkendt til tilhørende projekt og yderligere om applikationen kører stabilt. Sårbarhederne er meget enkelte, men alligevel meget kritiske da en bruger forventer at alt automatiseres og tjekkes af systemet, yderligere forventer brugeren også at der skabes integritet og at hans data håndteres sikkert.

### **7.3.5 Risikovurdering**

Risikovurdering der er blevet benytte til sikkerhedsrapporten er OWASP DREAD[12] (Damage Potential, Reproducibility, Exploitability, Affected Users, & Discoverability) model, der giver en en vurdering på de trusler ikke gennemførte testen i trusselvurderingen da det er trusler der er blevet håndteret. Derimod er derfor blevet opstillet følgende trusler og derefter bliver der opstillet en handlingsplan til de forskellige sårbarheder der blevet formået at dukke frem.

#### **7.3.5.1 Sårbarhed HTTPS Manglende HTTPS på hele systemet**

Trussel	D	R	E	A	D	Vurdering
JWT hijacking	7	6	9	9	10	8.2

### JWT hijacking

**Damage** Er sat til en af de højeste, da hele enkelte brugere kan ødelægges og en hacker kan optræde som en anden bruger ved at stjæle hans Json Web Token.

**Reproducibility** Er relativt let at lave, da man bare skal være logget på samme system og man skal data-sniffe en brugers token.

**Affected users** Kommer man først ind og kan få hijacked en token, vil det ramme alle de brugere der er på nettet og andre brugere man kan sende til. Det kan betyde at man også kan få adgang til en superadmin eller admin og derfor have adgang til hele projektet.

**Discoverability** Det er let at finde ud af om man kan briste det her, man skal blot kigge i browseren og se om der står HTTP eller HTTPS i URL'en

**7.3.5.2 Server nedbrud** Manglende mulighed for høj parallelitet, ydelse og lavt hukommelsesbrug.

Trussel	D	R	E	A	D	Vurdering
DDoS	2	10	10	10	8	8

### DDoS

**Damage** Skaden der kan forekomme i et DDoS angreb er ikke skadeligt, men derimod irriterende da man ikke har mulighed for at tilgå systemet. Der mistes ikke data og derfor sættes risikoen til noget af det laveste.

**Reproducibility** Er sat til det højeste, da man i dag let kan købe DDoS servicer online der tilbyder denne handling. Endvidere hvis serveren ikke er kraftig nok, kan en person med en hurtig forbindelse selv emulere et DoS angreb med et program.

**Exploitability** Som beskrevet før, så kan man købe servicen online og er derfor let at fremprovokere.

**Affected users** Alle brugere bliver påvirket af DDoS, eftersom at hele serveren går ned eller bliver uhyrlig langsom.

**Discoverability** Grunden til at denne handling ikke er sat til det absolut højeste, skyldes at man nogengange ikke ved om ens egen forbindelse er påvirket eller om det er serveren.

#### 7.3.5.3 Data håndtering Mangelende data eller nedbrud kan forekomme

Trussel	D	R	E	A	D	Vurdering	
SQL nedbrud	10	2	2	10	10	6.8	Mellem
Dokument sletning	8	8	4	7	5	6.4	Mellem

#### SQL nedbrud

**Damage** Hvis en SQL servers går ned og alt data slettes, så er fejlen kroatisk risikabel og kan nedbryde hele systemet.

**Reproducibility** Er sat relativt lavt, da det kræver en teknisk viden indenfor hvordan man sender persistent trusseldata der kan nedbryde eller slette databasen. Eftersom at systemet er sikret imod SQL-injection er truslen derfor ikke så let at reproducere.

**Exploitability** Denne handling kræver tekniske egenskaber og er derfor svær at udnytte.

**Affected users** Vil denne sårbarhed åbnes og udnyttes, er truslen sat til det kritiske - da der ikke er mulighed for at genskabe systemet og derfor vil alt data gå tabt.

**Discoverability** Systemet er nede og man har ikke mulighed for at tilgå noget og alt data er væk.

#### Dokument sletning

**Damage** Hvis et dokument slettes, så er der ikke mulighed for at få det tilbage - kun tidligere versioner da de også vil blive gemt i systemet, derfor er truslen ikke sat til det højeste, men den er stadig kritisk.

**Reproducibility** Har man adgang til serveren, har man derfor også adgang til filsystemet og har derfor mulighed for at slette det, om det er ondsindet eller sket ved en fejl. Dette er derfor en uhyrlig kritisk fejl og er let at reproducere - da der ikke er nogen validering endnu.

**Exploitability** Systemet har ikke nogen login validering endnu, men kræver alligevel en teknisk viden hvordan systemet fungerer - derfor er den ikke sat høj.

**Affected users** Er sat relativt højt, slettes en fil eller et dokument, påvirkes brugerne kun der arbejder på det og derfor ikke alle - den er stadig kritisk.

**Discoverability** Er kun brugere der arbejder på selve filen eller der tilgår den, den kan derfor være svær at se om der er sket datatab.

### 7.3.6 Handlingsplan

Ud fra analysen og udførte tests kan der konkluderes at systemet stadigvæk er sårbart og kræver yderligere testing, før det kan lanceres. Derimod kan vi komme med en handlingsplan og hvad vi vil anbefale før det sendes ud som et færdigt produkt eller til næste hold der skal videreudvikle projektet.

Anbefalingen for at håndtere session hijacking og data-sniffing er at benytte HTTPS på hele systemet, både front-end og back-end, netop for at kryptere den trafikerede data. For at virke troværdige kræves det af man får udstedt et SSL certifikat af en troværdig certifikatudbyder.

Der er yderligere lavet en række anbefalinger til hvordan vi kunne håndtere DoS. Et DoS er et angreb på et system, server eller lignende, med det formål at gøre servicen ubrugelig. Et DDoS angreb er det samme som et DoS angreb, men udføres af mere end en maskine (derfor distribueret).

Der er ingen måder at stoppe et DoS angreb på, men man kan gøre nogle ting for at forsøge at mindske effekten af et såsom at have en meget stor bandwidth. Andre ting man kan gøre er at have noget monitorerings software der holder øje med hvordan forskellige ip'er "opfører" sig og udelukke dem der "opfører" sig skadeligt. Dette kan dog resultere i udelukkelse af uskyldige ip'er hvis softwaren tager fejl. Man kan også benytte sig af tredjeparts services såsom cloudflare, der tilbyder DDoS beskyttelse samt en række andre fordele.

Handlingsplanen for data, er designet men ikke implementeret og kræver derfor en plan der skal videreføres - den er beskrevet detaljeret i Sektion 4.3. Hovedsageligt kræver det at man kører master-slave til både databasen og serveren hvor man håndterer filerne.

Et af de scenarier og sikkerhedsbrister, man tit tager for givet, er social engineering. Det er ikke noget vi kan teste da vi ikke har fysisk lanceret systemet endnu og har en server kørende. Vi kan derimod stadig komme med en løsning til dette for at forhindre dette når produktet lanceres.

For eksempel, hvis en uvedkommende ondsindet person benytter sig af social engineering

og opfører sig som en intern student eller medarbejder på DTU - så har personen mulighed for få fat i server oplysninger og derfor beskadeliggøre systemet. Handlingsplanen til dette at vise billede identifikation for validere at man er den rette bruger.

## 7.4 Test

Under hele implementeringen er der lavet en række manuelle test. Dette vil sige, at hver gang der er blevet implementeret noget nyt, er dette blevet testet for at se om det virkede efter hensigten. Der er dog ikke blevet lavet nogle test script for at teste det der er blevet lavet, det har været op til os selv at være sikker på at det der er blevet lavet fungere som det skal.

SQL har heller ikke gennem gået nogle test script, da det er blevet implementeret med stored procedures. Dette betyder, at hvis der skulle opstå en fejl undervejs, bliver tingene automatisk rullet tilbage, vi har derfor ikke fundet det nødvendigt at teste om de har fejl, da disse fejl alligevel ikke vil komme ind i databasen.



## 8 Konklusion

Ser vi på projektet som helhed, har vi formået at analysere og designe en database - der er gennemtestet og bearbejdet i fællesskab for et lancérbart system, som forventes at ville kunne benyttes med stor success. Endvidere har det hér også medført, at systemet er blevet forsinket netop af den årsag, at vi kommer svaghederne i forkøbet, ved at analysere designvalgene og gennemteste implementationerne grundigt igennem og diskutere hvilke mulige forcer og svagheder der kan forekomme undervejs forløbet.

For at skabe et overblik over applikationens bestanddele, er der blevet opstillet konkrete krav til systemet, som lå til grund for vores kravspecifikationer, User Stories og den tilhørende projektplanlægning og agile udvikling. For at kunne redegøre for de specifikke krav og deres status, er der blevet samlet dokumentation undervejs, til de forskellige agile faser - analyse, design, implementation og test. Dette resultere i, at projektet let kan sendes videre, ved at have en dokumenteret procesvejledning og detaljeret sikkerhedsrapport der kan bruges til vejledende milepæle til lanceringen af systemet og en form for tjekliste til hvad mangler.

Systemet har yderligere fået designet og modelleret en REST-baseret netværksarkitektur, med tilhørende API ressourcer, der skal benyttes til kommunikation mellem front-end systemerne, herunder hjemmeside og mobilapplikation. Al implementation er ikke blevet udviklet færdigt endnu, men helheden og datastrukturen er blevet grundigt diskuteret for at få den bedst mulige applikationsdesign, der formår at lettere gøre rettelser og videresende til vedligeholdelse af udvikling.

Endvidere er der også blevet designet et versioneringssystem der gør det muligt at tilføje, slette og opdatere på en ikke destruktiv måde, med tilhørende datalogging. Dette medfører, at man altid kan vende tilbage til tidligere stadier i systemet og kan holde sig opdateret på hvem har medført hvilken handling og hvorfor.

Sikkerhed har også fyldt meget, og der er derfor blevet analyseret og designet hvordan systemet skal lanceres og en plan for hvordan applikationen kan lave back-up på de forskellige filer, dokumenter og database. Yderligere også hvordan vi kan benytte sikker login med rollefordeling. Indtil videre er login til DTU's api det eneste der er blevet implementeret. Deployment af systemet er kun blevet designet, da det ikke er lancérbart endnu.

## 8.1 Perspektivering

Planen herfra er at videreudvikle systemet i 3-ugers, hvor der skal udvikles en hjemmeside og færdiggøre API'et. Endvidere skal der også laves testing af User Interfacet i form af brugertest, med tilhørende dokumentation. Systemet forventes også at skulle lanceres på en server, og derfor vil denne del også fylde en del.

## 8.2 Videre arbejde

Det er stadig en masse, som skal foretages før systemet er fuldt fungerende. De fleste er krav til, at produktet kan sættes i produktion. F.eks. skal API'et dokumenteres bedre end den swagger specifikation, som blev udarbejdet i design fasen. Derudover skal der være bedre kontrol over de brugere som er tilknyttet systemet og deres rettigheder. Systemet mangler også en måde at kunne vise de forskellige komponenters relationer til hinanden. F.eks. hvilke komponenter motoren har. Dette er relativt simpelt at tilføje og kræver bare nogle ekstra tabeller i databasen, samt et udvidelse af api'et til at supportere det. Til sidst burde der integreres bedre sikkerhed i systemet så som at gemme et unikt salt til hver bruger udover at have det hard-kodet. Secrets, så som krypteringsnøgler, SSL certifikater og campusnet tokens skal fjernes fra github og gemmes eksternt i en fil.

### 8.2.1 Dokumentation og videresendelse

Der skal udarbejdes en komplet dokumentation af API'et for, at den kan videregives. Til dette kan en viderebyggelse swagger specifikationen fra design fasen benyttes, da den gør det relativt nemt at dokumentere og vise de forskellige endpoints med deres modeller. Derudover skal alle Secrets gemmes og videreføres.

## References

- [1] wikipedia functional. [https://en.wikipedia.org/wiki/Functional<sub>r</sub>requirement](https://en.wikipedia.org/wiki/Functional_requirement).
- [2] wikipedia non-functional. [https://en.wikipedia.org/wiki/Non-functional<sub>r</sub>requirement](https://en.wikipedia.org/wiki/Non-functional_requirement).
- [3] User stories uml. <http://www.uml.org/>.
- [4] K.K. Khandelwal B.C. Punmia. Project planning and control with pert and cpm. 2016.
- [5] Swagger specification. <https://swagger.io/specification/>.
- [6] Leonard Richardson. Restful web apis: Services for a changing world. 2013.
- [7] DTU api (03/12/2017).
- [8] Json web tokens. <https://jwt.io/>.
- [9] DTU api (04/12/2017).
- [10] Pbkdf2s. <https://en.wikipedia.org/wiki/PBKDF2>.
- [11] CIA. <http://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA>.
- [12] OWASP DREAD. [https://www.owasp.org/index.php/Threat<sub>Risk</sub>Modeling](https://www.owasp.org/index.php/Threat_Risk_Modeling).

## 9 Bilag

### 9.1 Tidsplan

- **Uge 0**
  - Introduktion til projekt
- **Uge 1**
  - Kravspecification
  - Projektstyring
  - User stories
- **Uge 2**
  - Design
  - Møde
  - Flowchart
  - Deployment diagram
- **Uge 3**
  - Brugersystem
  - Login
- **Uge 4**
  - Komponenttyper
  - Komponenter
  - Dokumentations genstande
  - Hierarki
  - Kategori
- **Uge 5**
  - Komponenttyper
  - Komponenter
  - Dokumentations genstande

- Hierarki
  - Kategori
  - Database færdig
- **Uge 6**
  - API
- **Uge 7**
  - API
- **Uge 8**
  - API
- **Uge 9**
  - Front-end (Hjemmeside)
- **Uge 10**
  - Front-end (Hjemmeside)
- **Uge 11**
  - Produkt færdigt
- **Uge 12**
  - Rapport
- **Uge 13**
  - Rapport færdig