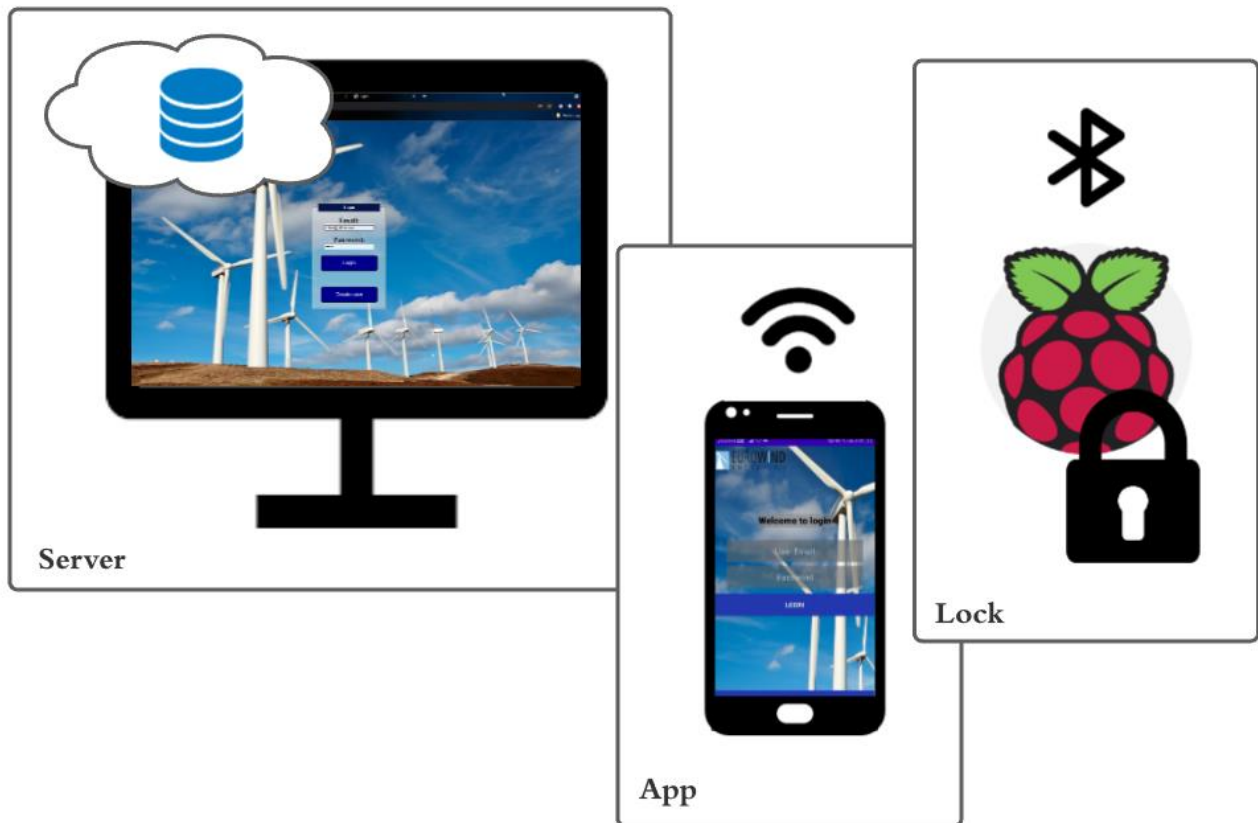


Online central styringsplatform

Bachelorprojekt

Aarhus Universitet, School of Engineering

Elektronik, Herning



Afleveret d. 15. dec. 2021.

Vejledere: Anders Lehmann.

Antal tegn: 71997.

Gruppemedlemmer:

Marc David Jensen Blunsdon
201708751

Jan Kastbjerg Schiermer
201701970

Resumé / Abstract

This report will describe the execution of a bachelor project, prepared over the 7th semester of Aarhus University's electronics bachelor program, for Eurowind energy in Hobro. The report will go through the project from its idea stage, all the way to a PoC (proof of concept) stage.

The project will be a PoC product, of an online central control platform, with a thereto belonging embed unit with lock, and a mobile application acting as a key, in compliance to Eurowinds desire of a PoC product, specifically designed according to their wishes and needs, to showcase for the company's employees and managers, and can be used to develop a product from a professional company, working with electronic locks.

The execution of the project has been done in accordance with the EUDP-model, which can be seen through the reports structure. The report goes through the phases: Pre Project, Launch Phase and Realisation Phase, and will in the end be concluded with Post Project, conclusion, and future thoughts.

Denne rapport vil beskrive udførelsen af et bachelorprojekt, udarbejdet over 7. semester på Århus universitets elektronik bacheloruddannelse, for Eurowind energy i Hobro. Rapporten vil gå igennem projektet, fra idé stadie til PoC (proof of concept) stadie.

Projektet vil være et PoC produkt, af en online central styringsplatform, med en dertilhørende embedded enhed med lås, og mobil applikation agerende som nøgle, i overensstemmelse med Eurowinds ønske om et PoC produkt der er specifikt designet efter deres ønsker og behov, til fremvisning af firmaets ansatte og ledelse, samt kan bruges til at få udviklet et produkt fra et professionelt firma, indenfor elektroniske låse.

Udførelsen af projektet er gjort i henhold til EUDP-modellen, hvilket vil kunne ses igennem rapportens struktur. Rapporten går igennem faserne: Pre Project, Launch Phase og Realisation Phase, for til sidst blive afsluttet med Post Project, konklusion og fremtidige tanker.

Indholdsfortegnelse

RESUMÉ / ABSTRACT	2
FORORD OG LÆSEVEJLEDNING	5
INDLEDNING OG PROBLEMFORMULERING	6
PRÆPROJEKT OG LAUNCH FASE	8
KRAV	8
AFGRÆNSNING	9
METODE OG PROCES.....	9
INDLEDENDE ANALYSE.....	12
<i>Website</i>	12
<i>Mobilapplikation</i>	15
<i>Embedded enhed med lås</i>	17
ARKITEKTUR	18
REALISERINGS FASE	20
WEBSITE.....	20
<i>Grafisk brugerflade</i>	21
<i>Django</i>	26
<i>RestAPI</i>	27
<i>Database</i>	29
<i>AWS</i>	32
<i>Sikkerhed</i>	33
MOBILAPPLIKATION	36
<i>Grafisk brugerflade</i>	36
<i>Gradle</i>	40
<i>RestAPI</i>	41
<i>Bluetooth</i>	42
<i>Fragmenter</i>	43
<i>Corutiner</i>	49
EMBEDDED ENHED MED LÅS	51
TEST.....	54
POSTPROJEKT	55
PRODUCT ACCEPTANCE	55
PRODUKTEVALUERING	56
PROJEKTEVALUERING	57
KONKLUSION.....	59
FREMTIDIGT ARBEJDE	61

REFERENCER..... 63

Forord og læsevejledning

Denne rapport omhandler samarbejdet med Eurowind [1]. Eurowind er en virksomhed, der tilbyder projekt planlægning, styring til opsætning af nye vindmølle og solcelle parker, samt har management og vedligeholdelse af disse parker.

Eurowind har hovedkontor i Hobro, og har afdelinger i Portugal, Sverige, Tyskland, Polen, Rumænien, Italien, og Spanien. De har desuden lige udvidet med en afdeling i USA, så Eurowind er en større virksomhed, som ekspanderer deres virksomhed løbende.

Udviklingsteamet, bestående af Marc Blunsdon og Jan Kastbjerg har under projektets forløb, haft vejledningsmøder med Anders Lehmann, som til dagligt er underviser på Aarhus Universitet, dagsorden og referat af disse møder er i Bilag 1.

Alle bilag vedrørende dette projekt og dens rapport kan findes i en zippet fil ved navn "Bilag", hvor dens indeholder er som følgende:

- Et dokument, som beskriver bilagsstrukturen
- En Procesrapport, som beskriver processen for projektet
- Bilag 1, en mappe med dagsorden og møder med vejlederen
- Bilag 2, en mappe Power Points fra møderne med Eurowind
- Bilag 3, en mappe med Pre Project rapporten
- Bilag 4, en mappe med Launch Phase rapporten og dens roadmap
- Bilag 5, en mappe med Realisation Phase og dens elementer
- Bilag 6, en mappe med feedback og fra Eurowind, lidt om sikkerheden i projektet, tanker omkring GDPR og Product Acceptance
- Bilag 7 indeholder den Psykologiske kontrakt
- Bilag 8 indeholder alt kode med filen ".git"

Rapporten tager læseren igennem, teamets fremgangsmåde med projektet, hvor hoved fokus har været at lave et "Proof of Concept", som visuelt kan vise det kunden ønsker.

Læseren præsenteres først for Pre Project og Launch Phase, som introducerer til de væsentligste overvejelser, og de indledende designvalg, og designbeskrivelser, for at afstemme produktets krav sammen med kunden.

Realisation Phase omhandler realiseringen af produktet, og præsenteres i 3 dele, website, mobilapplikation, og embedded enhed med lås. Hvert punkt præsenterer, de teknologier og elementer, som har være mest relevant for at udvikle produktet.

Til sidst præsenteres læseren for Post Project, konklusion og fremtidigt arbejde. Under Post Project vil der blandt andet være feedback fra kunden, resultaterne fra Product Acceptance og diskussion.

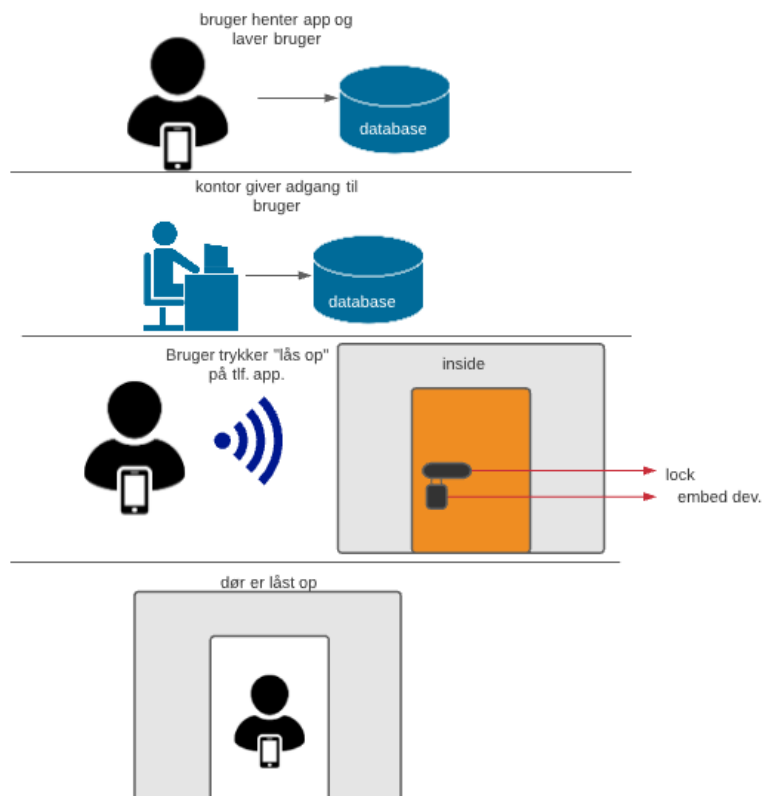
Indledning og Problemformulering

Eurowind vil have udviklet et produkt, der kan bruges som "proof of concept". Produktet skal kunne give adgang til diverse faciliteter, via en hjemmeside som administrerings- og kontrol platform. Herfra skal faciliteterne kunne låses op elektronisk, med en mobilapp agerende som nøgle. Denne mobilapplikation skal kunne fungere uanset hvor faciliteterne er placeret, samt føre logning over anvendelsen af faciliteter.

Oplægget for produktet, er at undgå fysiske nøgler, RFID-brikker o.l. skal udleveres, og da en mobil er almeneje, er det et oplagt objekt at bruge i stedet.

Eksisterende løsninger er enten for mangelfulde eller for komplekse, både hardware og software mæssigt, at sætte op, anvende og administrere. Hvilket gør det svært at gennemskue, og danne overblik over grundprincipperne, og funktionaliteterne i sådan et system.

Eurowind vil grundet disse problemer, gerne have udviklet et produkt som indeholder "bare minimum" funktionaliteter, for at have et gennemskueligt demonstrationsprodukt. Dette har, i samarbejde med Eurowind og udviklingsteamet, udmøntet sig, i et Rich Picture, som på Figur 1, der har været udgangspunktet i projektet.



Figur 1 Rich Picture fra bilag 3: Preproject

Projektet vil besvare følgende spørgsmål:

- Hvordan kan der laves et system, som undgår fysiske nøgler?
- Hvordan kan der udvikles et låsesystem, med centraliseret styring?
- Hvordan kan software, hardware opsætning og brugerfladen gøres mindre kompleks, i forhold til eksisterende løsninger?
- Hvordan vil der kunne udvikles et system som dækker Eurowinds administrative behov?

For at besvare spørgsmålene vil produktet bestå af 4 hovedelementer, der udvikles igennem projektet:

- Der skal udvikles en database, hvor der skal gemmes information omkring, hvem har adgang hvortil, samt hvor langtid.
- For at kunne styre databasen, skal der udvikles en hjemmeside, som skal være den centrale styring. Hjemmesiden skal kunne give adgang til registrerede brugere og til specifikke faciliteter i en sat mængde tid. På hjemmesiden skal der føres log over, af hvem og hvornår, en facilitet er blevet tilgået.
- Dertil skal der udvikles en mobil applikation. Applikationen skal bruges til at åbne en lås vha. Bluetooth, for at give adgang til en facilitet, og indsende log oplysninger til databasen.
- Selve låsen, der bliver udviklet i projektet, vil være et eksempel til, hvordan der kan kommunikeres mellem en mobil applikation og en lås.

Produktet vil blive udviklet over et enkelt semester, i det omfang der giver mening for projekt, i henhold til udviklingsmodellen EUDP [2].

Præprojekt og Launch fase

Dette afsnit omhandler planlægningen og de afklaringer, som var baseret på en række artefakter fra EUDP i faserne *Pre Project* (se bilag 3, Pre Project rapport) og *Launch Phase* (se bilag 4, Launch rapport). De væsentligste af disse artefakter er taget med i de følgende delafsnit.

Krav

I samarbejde med Eurowind, er der blevet udviklet nogle rammer for projektet i form af krav, disse krav er beskrevet i afsnittet *Requirements Analysis* (se Bilag 4, Launch rapport). Kravene har dannet projektets ramme for, hvad skal opfyldes for at produktet godkendes. Dette artefakt har været en af de vigtigste for projektet, og har været udarbejdet under hele Launch Phase forløbet, i samarbejde med både vejlederen og Eurowind.

Artefakterne *Preliminary Use Cases* og *System definition* fra *Pre Project rapport*, har været med til at udforme starten på *Requirements Analysis* fra *Launch Rapport* i Launch Phase.

Et mindre uddrag af kravene fra *Requirements Analysis*, er på Figur 2, hvor der er et kravs ID og en beskrivelse af kravet.

Kravs ID	Kravbeskrivelsen
WEB-1	Det skal være muligt at oprette nye brugere.
WEB-2	Det skal være muligt at slette brugere.
WEB-3	Det skal være muligt at logge ind på sin bruger.
WEB-4	Det skal være muligt at redigere i eksisterende brugere.

Figur 2 uddrag af "Requirement Analyse" (se bilag 4, Launch rapport)

Hvert krav er blevet tildelt til en kravsgruppe, hvilket er blevet gjort for at gøre det mere gennemskueligt, og nemmere at følge kravene, hvor der er følgende grupper:

- Webside, med præfikset WEB-.
- Mobilappen, med præfikset APP-.
- Embedded enhed, med præfikset EMB-.
- Låsen, med præfikset LOCK-.

Hvert krav har et tilhørende test ID, som henviser til dokumentet *Product Acceptance* (se Bilag 6), som beskriver hvordan kravet skal testes for at være godkendt, dette dokument har været udarbejdet af teamet og godkendt af Eurowind.

Afgrænsning

For at gøre det plausibelt, at kunne udføre projektet indenfor den givet tidsramme på ca. 3 måneder, har det være nødvendigt at lave nogle afgrænsninger, ved at vurdere hvad der giver mest værdi for projektet, at realiserer, hvilket er gjort i samarbejde med Eurowind og vejleder.

Da det endelige produkt er tilsigtet som et PoC (*Proof of Concept*), til fremvisning og inspiration, er det blevet vurderet, at funktionaliteterne af produktet, vil give den største værdi at fokusere på, hvor sikkerhed, brugervenlighed, robusthed og låsen er blevet nedprioriteret.

Ekstra sikkerhedsmæssige beslutning til produktet, vil kunne implementeres af Eurowind efter projektets færdiggørelse, da det ikke nødvendigvis vil give mere værdi til det endelige PoC-produkt.

Der er udfærdiget et dokument, med diverse sikkerheds undersøgelser (se Bilag 6, Sikkerhed), dokumentet beskriver de manglende og/eller implementerede sikkerheds beslutninger indenfor:

- Kommunikation mellem database og mobil applikation.
- Kommunikation mellem mobil applikation og den embedded enhed.
- fingerskanning.

Brugervenlighed har været en prioritet, indtil hvor produktet giver arkitektonisk mening, for at gøre anvendelsen af produktet nemmere for en potentiel bruger.

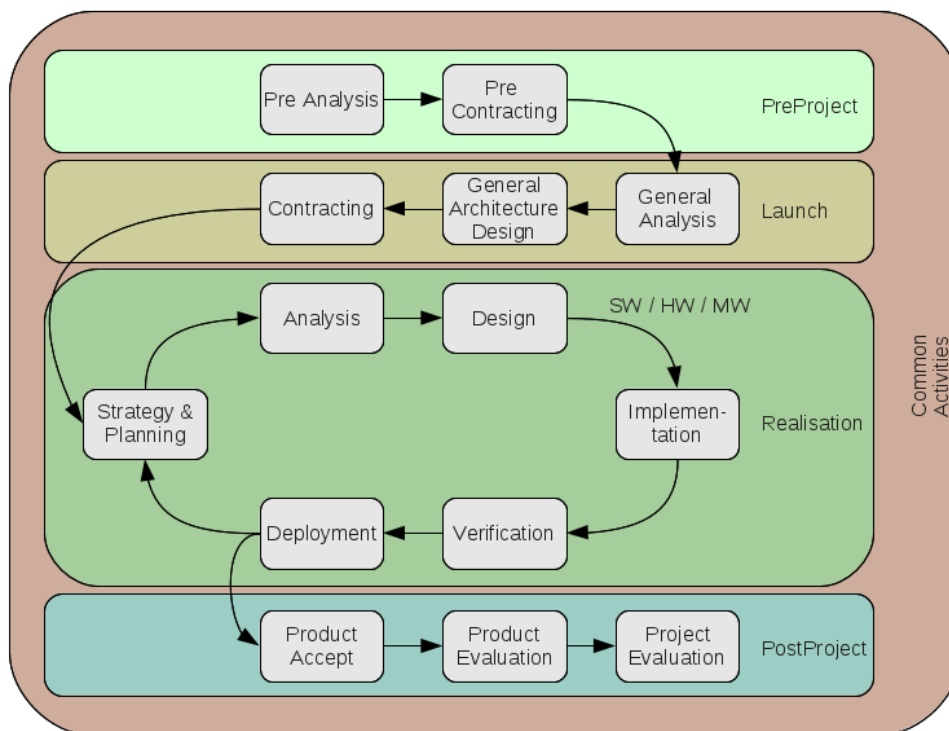
Ved nogle områder af produktet, er robustheden nedprioriteret, da det for eksempel ikke har været muligt at teste mobil applikation på en række af forskellige Android telefoner.

Låsen har ikke haft det største fokus, da låsen er et mockup af, hvordan de professionelt lavet elektroniske låse på markedet fungerer. Såfremt der i fremtiden findes en optimal elektronisk lås med API adgang, vil produktet kunne ændres over til sådanne en lås.

Metode og proces

Processen anvendt til at løse projektets tekniske del, har været i henhold til EUDP. EUDP går igennem forskellige faser, ment til at udvikle, designe og udfærdige et produkt, hvor realisation fasen er en iterativ fase.

De 4 faser på Figur 3, giver et overblik for hvornår faserne skifter, og hvad disse faser indeholder. De artefakter der giver værdi for projektet er blevet udvalgt og kan ses i Pre Project og Launch Phase rapporterne.



Figur 3 EUDP roadmap [3]

Metoderne anvendt til projektets styring, har hovedsageligt været afarter af Sprint [4], Gantt [5] og Kanban [6] i Excel til at styre opgaver og tidsplan.

Sprint har været brugt i realiseringsfasen, hvor sprintende har forløbet sig over én uges intervaller, hvor der er blevet sat, hvilke opgaver der skal laves, og hvem der skal lave hvad. Sprintet er blevet fælles planlagt om mandagen, for ugen frem, og hvad der blev nået den forgangne uge.

Der er blevet oprettet en opgaveliste for hvert delement (website + database, mobil applikation og embedded enhed med lås), et udklip kan ses på Figur 4, i projektet, hvori følgende har kunne sættes:

- *Target* (opgavens status)
- *Milestone or Activity* (opgavens titel)
- *Priority* (hvor vigtig er opgaven for projektet)
- *Result* (dokumentation og review status)
- *Time estimation*
- *Deadline*
- *Kommentar* (generelle kommentarer til opgaven)
- *Krav* (*Requirements Analysis* (se Bilag 4, Launch rapport))

Website + database roadmap

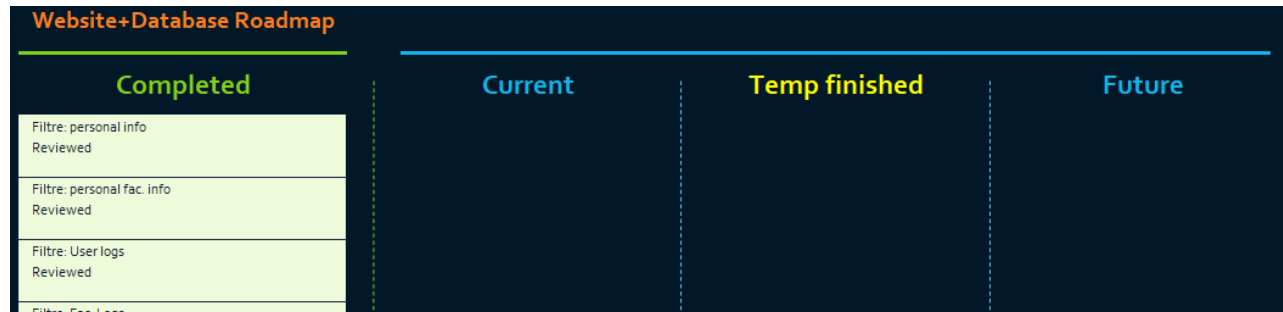
Total time:	199	days:	28,4				
Time left:	0						
Time spent:	199						
Target	Milestone or Activity	Priority	Result	Time estimation	Deadline (Week number)	kommentarer	krav
Completed	Filtre: personal info	1-High	Reviewed	5,00	39		WEB-4, WEB-8
Completed	Filtre: personal fac. info	1-High	Reviewed	5,00	39		WEB-4, WEB-8
Completed	Filtre: User logs	1-High	Reviewed	6,00	40		WEB-9, WEB-10
Completed	Filtre: Fac. Logs	1-High	Reviewed	1,00	40		WEB-9, WEB-10
Completed	admin bruger	1-High	Reviewed	10,00	38		WEB-2, WEB-11
Completed	csrf token inserts on all pages	1-High	Reviewed	1,00	39		
Completed	Ustertype control	1-High	Reviewed	1,00	39	only field user can access field user page etc.	
Completed	Login	1-High	Reviewed	10,00	38		WEB-3
Completed	LP: Office user dashboard	1-High	Reviewed	6,00	38		WEB-4

Figur 4 roadmap: opgaveliste udklip

Til hver opgaveliste, er der lavet et dynamisk Kanban diagram, se Figur 5 udklipet, som er linket til deres respektive opgavelister, og har følgende mulige statusser:

- *Completed*: Opgaven er fuldført.
- *Current*: Opgaven er igangværende.
- *Temp finished*: Opgaven mangler evt. dokumentation, review eller venter på udfærdigelse af en anden opgave.
- *Future*: Opgaver der mangler at blive påbegyndt.

Kanban diagrammet opdateres automatisk, ved ændringer i opgavelisten, og behøver derfor ikke direkte bruger indflydelse for at være opdateret.



Figur 5 roadmap: Kanban udklip

For at danne et samlet overblik og give en rød tråd igennem realiseringsfasen, er der blevet brugt et dynamisk Gantt diagram, se Figur 6 udklipet, som roadmap, baseret på en kompileret liste af opgaverne. Gantt diagrammet opdateres automatisk, ligesom Kanban'en dog ud fra den kompilerede opgaveliste.

Gantt diagrammet er opdelt i sprint perioderne, og i 3 faser:

- *Development*: Udvikling og implementering af projektet.
- *Test*: Fase til at udarbejde potentielle bugs, og få lavet integrations test af de endelige delelementer.

- **Report:** Periode udelukkende dedikeret til udfærdigelse af de endelige rapporter.

Ud over faser og sprint, er de listede opgaver i diagrammet også blevet farvekodet på følgende måde:

- Blå = website
- Grøn = mobilapplikation
- Rød = embedded enhed med lås

Website			timer:	475	project completion rate:		100%										
App			days:	68.4													
Embed			upper:	6.8													

Development										Test	Report			
Uge	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Timer i alt	5	43	34	63	53	20	88	90	31	40	0	0	0	0
Opp. 1	[Website] Opp AWS setup Kra: None	[Website] Opp admin bruger Kra: WEB-2, WEB-11	[Website] Opp Fire: personal info Kra: WEB-4, WEB-6	[Website] Opp Fire: User logs Kra: WEB-3, WEB-10	[Website] Opp update database doc Kra:	[App] Opp Research Kra: None	[App] Opp Analyse: Fragments Kra: file	[App] Opp Implementing: LoginFrag Kra: APP-1, APP-2, APP-3	[Website] Opp fire test API Kra:	[App] Opp Analyse: Finger scan Kra: APP-3				
Opp. 2	[Website] Opp Login Kra: WEB-3	[Website] Opp Fire: personal info Kra: WEB-4, WEB-6	[Website] Opp Fire: Fac. Logs Kra: WEB-3, WEB-10	[Website] Opp AWS deploy Kra: None	[Embed] Opp function: clean pairing record Kra: N/A	[App] Opp Analyse: Rest Api Kra: APP-2, APP-3, APP-4, APP-5, APP-7, APP-8	[App] Opp Implementing: ScanFrag Kra: APP-5	[Website] Opp sd certification Kra: WEB-12	[Embed] Opp Dokument: Skikshed Kra: N/A					
Opp. 3	[Website] Opp LP: Office user dashboard Kra: WEB-4	[Website] Opp csrf token inserts on all pages Kra:	[Website] Opp DMP: Give fac. Access Kra: WEB-4, WEB-6, WEB-7	[Website] Opp cron create user Kra: WEB-5		[App] Opp Analyse: Bluetooth Kra: APP-5, APP-6	[App] Opp Implementing: UnlockFrag Kra: APP-4, APP-5, APP-6, APP-7, APP-8	[Website] Opp fix url's in html and url.py's Kra:	[Embed] Opp Test: Integration test Kra: N/A					
Opp. 4	[Website] Opp LP: Field user dashboard	[Website] Opp User type control	[Website] Opp DMP: Remove fac. Access	[Website] Opp cron clean up		[App] Opp Analyse: general struktur Kra:	[App] Opp Implementing: SuccessFrag Kra:	[Website] Opp fix office cant edit admin						

Figur 6 roadmap: Gantt udklip

Gantt, Kanban og sprint er blevet brugt i både Launch Phase og Realisation Phase (se Bilag 4 & 5, roadmap).

En dybere forklaring af projektets bløde emner kan læses i procesrapporten (se Bilag 7, Procesrapport).

Indledende analyse

Dette afsnit vil beskrive de indledende analyser til projektets delelementer.

Opdelingen af afsnittet er grundet der igennem projektet, er blevet udarbejdet 3 produkter, en hjemmeside til kontrol, en elektronisk lås, og en mobil applikation som nøgle.

Website

Analysen for hvordan brugerne, skal interagere med hjemmesiden, er forgået i flere stadier, med afsæt i Preliminary Use Cases (se Bilag 3, Pre Project rapport), ledet over i "Usage Domain Analysis" (se Bilag 4, Launch rapport), hvor anvendelserne for hjemmesiden blandet andet specificeres, med først en liste over selve anvendelserne:

- Bruger login
- Bruger Log out
- Oprette ny bruger
- Se "Personal Information"
- Se "Facility access"

- Se "User logs"
- Se "Facility logs"
- Se "Create user code"
- "Edit user"
- "Edit facility access"
- "Add facility"
- "Edit facility"
- "Remove facility"

De brugertyper der skal anvende hjemmesiden er:

- Admin, vedligeholder server, database, hjemmeside, mobilappen, kan slette alle brugertyper og har adgang til alle hjemmesiden funktionaliteter
- Office User, har adgang til alle hjemmesidens funktionaliteter, har dog ikke direkte adgang til databasen, og kan kun slette brugertypen field.
- Field User, den almene bruger, som bruger både hjemmeside og mobil, kan se egen personlige informationer og hvilke faciliteter bruger har adgang til.

Admin og Office User har også adgang til at bruge mobilapplikationen, hvis der er givet adgang til en eller flere faciliteter.

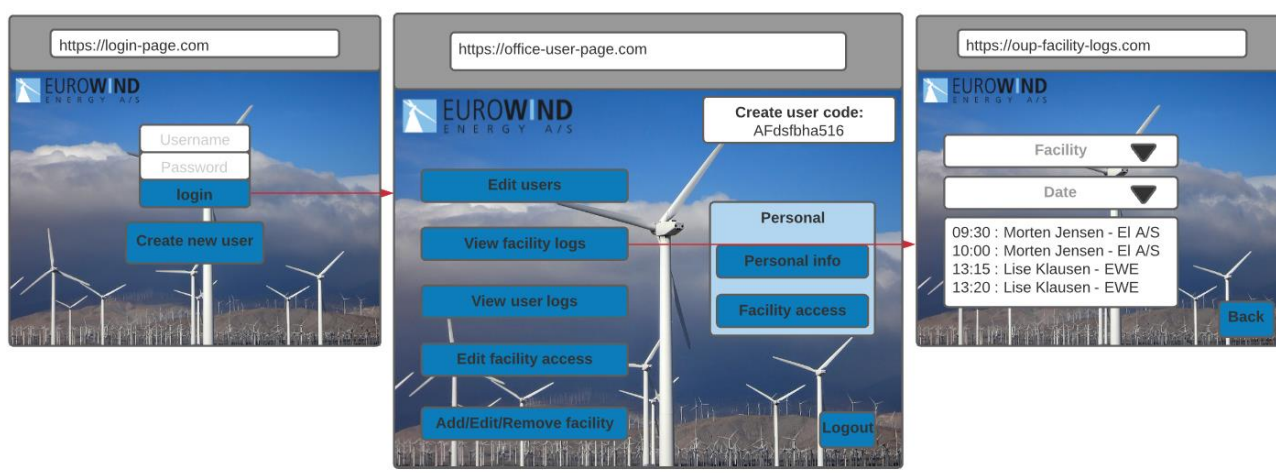
For hvert element i listen med anvendelser, har der været udviklet en "Use Case Definition", som udforsker hvordan flowet skal være for de enkelte anvendelser, på Figur 7 bemærkes, hvordan en anvendelse er beskrevet, resten kan læses i Bilag 4, Launch rapport.

Name	Brief description
Website – Se "Facilities logs"	Aktøren kan se logs fra de forskellige faciliteter, både hvem der har adgang og hvem der har haft adgang
Actor(s)	Precondition(s)
<ul style="list-style-type: none"> • Admin • Office User 	<ol style="list-style-type: none"> 1. Aktøren har en Office User eller er udvikler 2. Aktøren har tilgået hjemmesiden 3. Aktøren er logget ind på sin bruger
Basic flow of events	Alternative flow(s)
1. Aktøren klikker på "Facilities logs"	N/A
Postcondition(s) Successful completion	Postcondition(s) Unsuccessful completion
<ol style="list-style-type: none"> 1. Aktøren kommer ind på "Facilities logs" siden 2. Aktøren kan nu søge og se information vedrørende de forskellige faciliteter 	<ol style="list-style-type: none"> 1. Aktøren modtager en fejlbesked og har ikke adgang til at se "Facilities logs" informationen.

Figur 7 Use Case for se "facility logs" se Bilag 4, Launch rapport, s 11

Endvidere analyse af bruger interaktioner har foregået i Interface Analysis (se Bilag 4, Launch rapport), som er todelt i både "user interface" og "system interface". Hvordan en bruger skal kunne agere med hjemmesiden, hvad brugeren ser (user interface), og hvorledes hjemmesiden agere bagved, hvilke snitflader der er i systemet (system interface).

For at overvåge aktiviteten af en facilitet (hvem har tilgået faciliteten, og hvornår er faciliteten tilgået), skal der gemmes en log. For at se disse logs, vil en office eller admin bruger skulle logge ind på hjemmesiden, for herefter at trykke på "View facility logs", hvor der kan søges på faciliteter og dato, se Figur 8, som viser et eksempel med "View facility logs", resten kan læses i Bilag 4, Launch rapport.



Figur 8 View facility logs, Bilag 4, launch rapport, s 19

For at kunne udvikle hjemmesiden, dens funktionaliteter og implementere den på en server, er der blevet valgt en række teknologier og services.

Django [7] er valgt som framework til hjemmesiden, frem for alternativer som Laravel [8], da Django er bygget på Python, hvilket er en af de mest almene programmeringssprog i dag, er grundigt dokumenteret, har en lang række af færdig bygget biblioteker og plugins, og er et framework gruppen har arbejdet med før.

Pycharm [9] er IDE'en valgt til database og hjemmesideprogrammeringen, og har indbygget understøttelse af Django, hvilket yderlig har været et argument for at bruge Django og Pycharm.

Til *hosting* af hjemmesiden er der blevet brugt AWS lightsail [10], som er en cloud service med et fuldt prækonfigureret Linux system. Lightsail er blevet brugt da det er en *low cost* cloud server, som vil være nem at skalere, hvis der skulle være brug for det. I forhold til alternativer som fx Microsoft Azure [11] har AWS været billigere, samt har de første 3 måneder været gratis.

For at kunne forbinde udefra til hjemmesidens/serverens database, er der blevet sat en RestAPI op, som er brugt til at hente bruger og facilitets informationer, i hensyn til hvem har adgang til hvilke faciliteter, nøglen til de diverse låse og indsætte logs ind i databasen.

Som scheduler til oprydning af databasen, og daglig generering af koder, som vil blive dybere beskrevet i afsnittet *Website* side 20, er der blevet brugt CRON [12], da dette ligger som standard på Linux systemer, samt kan findes som et plugin designet til Django.

NGINX [13] er blevet brugt som "reverse proxy" fra cloud-serveren til hjemmesideserveren, for at dirigere trafik fra cloud serverens HTTP-port til HTTPS-porten, og sende header informationer, til Gunicorn [14] som er blevet brugt som hjemmeside server, da Django som server er anbefalet af Django selv ikke at blive brugt i produktion. NGINX står også for at vise de statiske filer der tilhører hjemmesiden.

Til kryptering af hjemmesidetrafikken er der blevet valgt SSL, med certifikater udbudt af Certbot Letsencrypt [15], da det er et gratis og sikkert alternativ til betalte SSL-certifikater udbudt af fx Google.

Mobilapplikation

Mobilapplikationen udvikles til en Android mobil da:

- Ingen på teamet har adgang til en iPhone og en Mac, og vil derfor ikke være muligt at teste produktet.
- Eurowinds fokus var ikke at udvikle et PoC, der passer til alle mobilplatforme.

Som i afsnittet *Website* side 12, har mobilapplikationen gennemgået de samme artefakter. En analyse for, hvordan brugerne skal interagere med mobilapplikationen, med afsæt i *Preliminary Use Cases* (Bilag 3, Pre Project rapport), ledet over i *Usage Domain Analysis* (se Bilag 4, Launch rapport), hvor anvendelserne for mobilapplikationen er:

- Installerer applikationen
- Bruger login
- Scan efter facilitet
- Forbind og lås faciliteten op

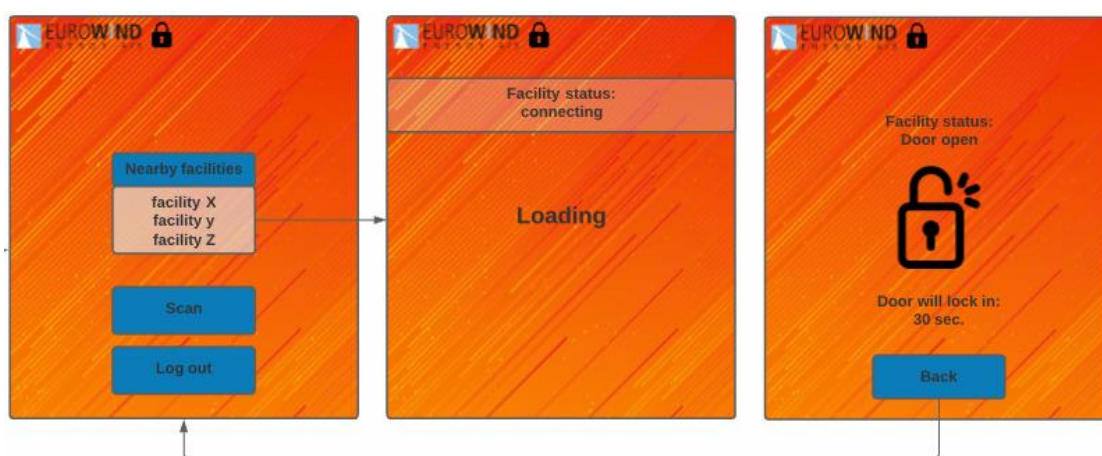
De tre aktør (admin, office og field user) kan anvende mobilapplikationen.

Her vil brugeranvendelsen "forbind og lås faciliteten op" af mobilapplikationen gennemgås, som på Figur 9.

Name	Brief description
Mobilappen – Forbind og Lås faciliteten op	Aktøren låser en facilitet op
Actor(s)	Precondition(s)
<ul style="list-style-type: none"> • Admin • Office User • Field User 	<ol style="list-style-type: none"> 1. Aktøren har installeret Appen 2. Aktøren har åbnet appen og login siden er tilgængelig 3. Aktøren har logget ind 4. Aktøren har aktiveret mobilens Bluetooth 5. Aktøren har skannet med Bluetooth 6. Aktøren har fået et skanningsresultat
Basic flow of events	Alternative flow(s)
1. Aktøren trykker på en af de fundene faciliteter	N/A
Postcondition(s) Successful completion	Postcondition(s) Unsuccessful completion
1. Facilitet XXX, låses op	1. Facilitet XXX lås, låses ikke op

Figur 9 Use Cases for forbind og lås faciliten op, se Bilag 4, Launch rapport, s 14

Her er der indtænkt brugervenlighed, da brugeren blot skal vælge en facilitet, som ønskes låst op, brugeren skal ikke tænke på at hente en nøgle, forbind til facilitet og klik låse op, brugeren skal have en besked, når facilitet er oplåst, se Figur 10.



Figur 10 User interface, se Bilag 4, Launch rapport, s 25

Disse artefakter har været brugbare under udviklingen af det endelige layout.

IDE'en brugt til at udvikle mobilapplikationen har været Android studio [16], da denne IDE er godt dokumenteret, er anbefalet af Google, gratis, og supporterer udviklingsproget Kotlin [17].

Programmet er skrevet i Kotlin, da Kotlin er designet til at skulle være nem og starte ud med, uden tidligere kendskab til mobil programmering, er godt dokumenteret, og anbefalet af Google. Kotlin kan desuden uden større vanskeligheder fungere sammen med Java [18] filer, som er sproget tidligere

normalt anvendt til mobil programmering, og er mere komprimeret i forhold til Java kode.

Embedded enhed med lås

Til et PoC produkt kan der bruges en Raspberry enheden, dog vil der i et produktionsprodukt, med fordel kunne bruges en anden embedded enhed, der er både fysisk og kapacitet mæssigt mindre, samt er mere strømbesparende, som Particles Argon enhed [19].

Som styrings enhed til låsen er der blevet valgt at gå med en Raspberry PI 3 B+ [20], da denne embedded enhed, vil kunne understøtte potentielle ændringer, da den kan køre et fuldt Linux system, understøtter Bluetooth [21] forbindelser, og har kapaciteter langt over hvad der skal bruges til låsen.

Fordelen ved at anvende Bluetooth er at alle mobiler har et Bluetooth modul, og da de fleste mennesker har en mobil, vil de potentielt alle have muligheden for at, have adgang til at skaffe sig en nøgle til en facilitet.

Bluetooth på den embedded enhed, kan enten være Bluetooth Low Energy (BLE) eller Bluetooth classic, hvor de største forskelle er:

- BLE anvender mindre strøm
- BLE kræver flere opsætninger, som Services, characteristics, General Attribute Profile (GATT) og General Advertising Profile (GAP)
- Classic har en større dataoverførselshastighed

Fokusset har været et PoC med henblik på at anvende en allerede udviklet lås med Bluetooth API adgang. Derfor har det været prioriteret at sætte den elektriske lås og embedded enhed op, så hurtigt som muligt, hvilket er grundlaget for valget af Bluetooth classic. Det skal holdes i mente at, hvis en elektronisk lås skulle udvikles fra bunden, ville BLE være det mest optimale valg.

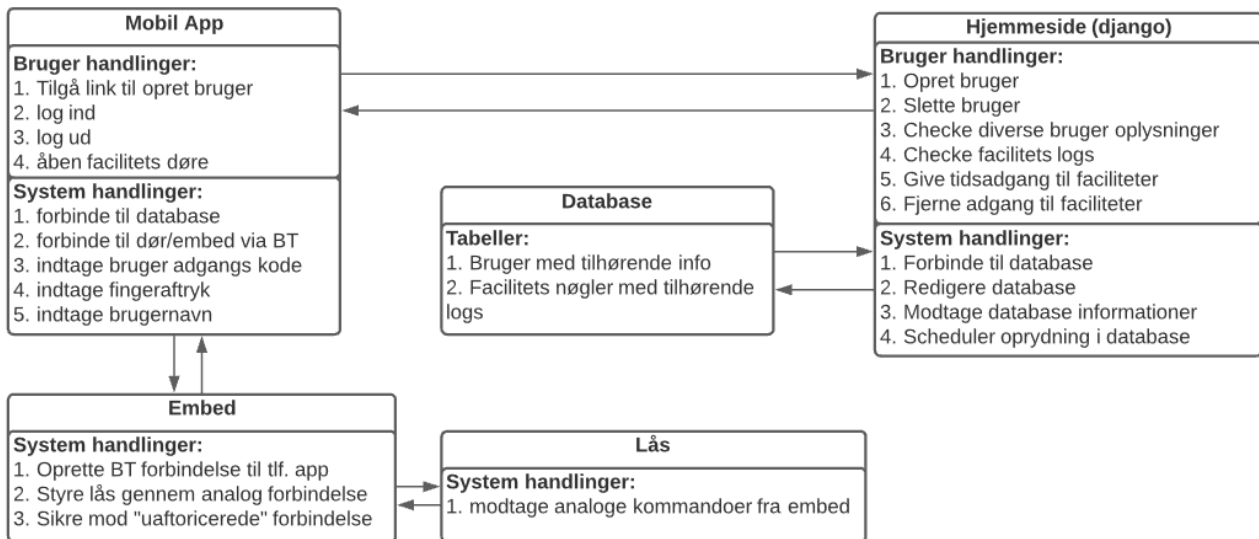
Da lås delen af projektet, er ment som et mockup af de professionelle låse, er der blevet brugt minimalt økonomisk på delene dertil.

Selve låsen brugt til projektet, er en elektrisk lås "Door solenoid" [22], styret af Raspberry PI'en. Hvis PoC skulle i produktion, og stadig være med en embedded enhed og lås, ville det have været bedre at bruge en magnetisk lås som fx "EL 582 Magnetlås" [23], da dette er en "rigtig" lås, hvor den anvendte "lås" er en slå. Magnetlåsen har forskellige måder at blive konfigureret på, alt efter hvordan man tilslutter strøm til den, den giver et feedback signal ved brug, og har også muligheden for at bruge en fysisk nøgle, i tilfælde af strømsvigt.

For at området "embedded enhed med lås" anses som virkende, skal enheden programmatisk styre IO's til låsen, håndtere Bluetooth pairing, forbindelse til mobilapplikationen, og igennem en scheduler, som CRON eller rc.local service filen på et linux system, selv starte disse programmer, efter opstart af Raspberry PI'en.

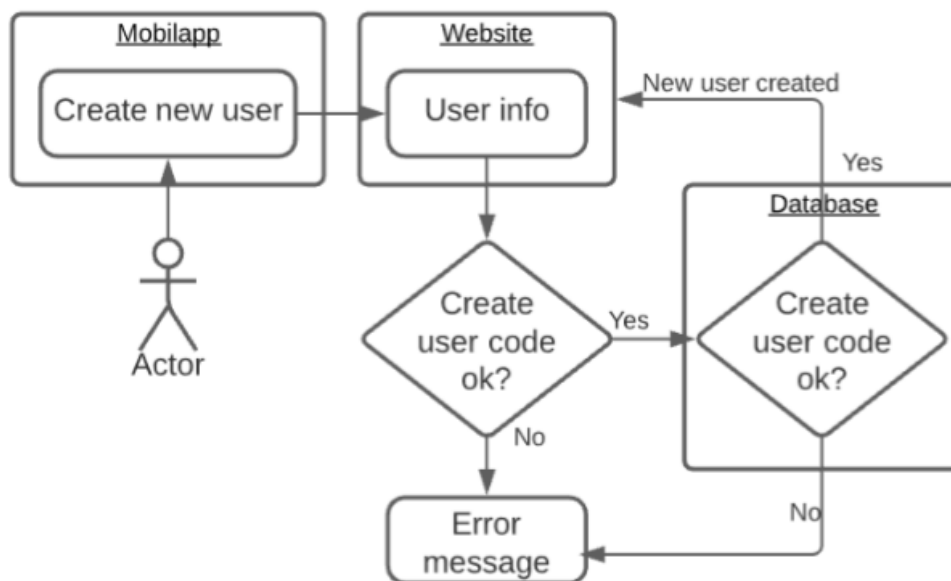
Arkitektur

Et større overblik af produktets snitflader, og hvilke handlingerne skal ske hvor, har været analyseret ved at bruge artefaktet "Problem Domain Analyse", se Figur 11, hvor en list af brugerhandlinger og system handlinger for hvert område (mobilapplikation, hjemmeside, database, embedded enhed med lås), pilene viser hvilke kommunikationsveje områderne imellem har.



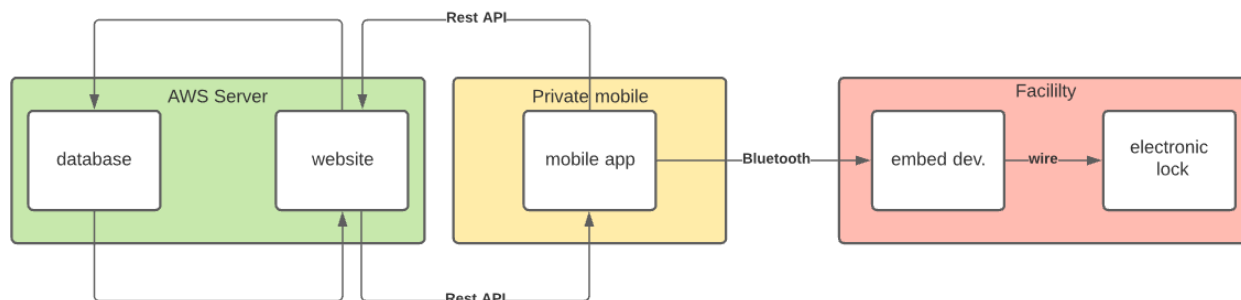
Figur 11 Problem domain analyse, Bilag 4, Launch rapport, s 4

Efterfølgende er artefaktet "System behavior" (se Bilag 4, Launch rapport, s 28-31), hvor produktets vigtigst system adfærd er blevet beskrevet, et eksempel kan ses på Figur 12, der beskriver hvad der sker, når en ny bruger skal oprettes. *Create user code* er en sikkerhed, for at udefrakommende, der ikke har noget med produktet at gøre, ikke kan oprette brugere, og på den måde overbelaste systemet.



Figur 12 System Behaviour Analyse, Bilag 4, Launch rapport, s 28

For at udpensle snitfladerne i produktet, med de udvalgte teknologier er Figur 13, blevet udviklet. Denne figur har mest været anvendt til møderne med Eurowind og vejlederen for at vise, hvilke elementer der skulle udvikles for PoC, og hvilke dele der vægtes højere end andre. Under hele projektets forløb har figuren, også haft funktionen at vise hvilke områder, der var færdig udviklet.



Figur 13 Interface design

På Figur 13 kan man se at Database og hjemmesiden hostes på en AWS-server, og der er forgå kommunikationen i mellem, hjemmesiden skal via RestAPI kunne kommunikere med mobilapplikationen. Mobilapplikationen skal installeres på en mobil, som kan kommunikere mellem mobilen og hjemmesiden via RestAPI over nettet, mobilapplikationen skal også via Bluetooth kunne kommunikere med embedded enhed (Raspberry Pi) på facilitets siden. Facilitets siden består af en embedded enhed og en elektronisk lås. Enheden skal kommunikere med mobilapplikationen, via Bluetooth, enheden skal kunne sendt et tændt signal til låsen.

Realiserings fase

Realiseringsfasen har forløbet sig over otte uger, hvor hver uge har været en Time Box med strategi og planlægning, analyse, implementering og test. Det er i denne fase selve udviklingen er gjort.

Afsnittet er delt op i fire delafsnit, website, mobilapplikation, embedded enhed med lås og test, hvert af disse delafsnit har nogle underpunkter, der beskriver de vigtigste elementer dertil.

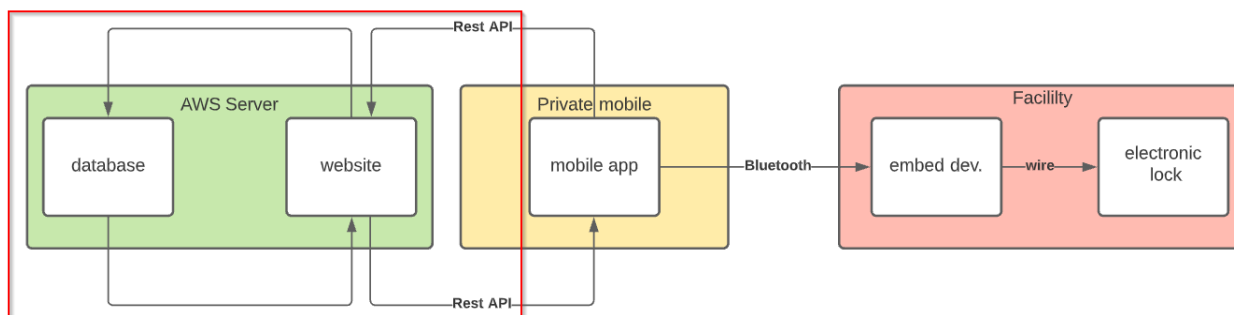
Der er til de forskellige delelementer vist kodelumper for at danne en ide, over hvorledes teamet har skrevet og struktureret koden, til projektet. Koden viser ikke det fulde billede. Cirka antal af linjer kode teamet selv har udviklet er som følgende:

- Website 3000 linjer plus HTML og Linux system filer.
- Mobilapplikation 1200 linjer.
- Embedded enhed med lås 150 linjer plus Linux system filer.

Alt kode vedrørende mobilapplikationen, embedded enhed, hjemmesiden og database er at finde i Bilag 8.

Website

Eurowind ønskede sig en platform, hvorfra adgangen til de forskellige faciliteter kunne styres, og der har kunne føres logs over dette, hvilket har udformet sig til en hjemmeside og database. Denne sektion vil tage læseren igennem implementeringen af databasen og hjemmesiden, som også kan ses på Figur 14, og er baseret på den tidligere analyse i afsnittet Website på side 12.



Figur 14 website afsnit

Emnerne der bliver gennemgået er:

- Grafisk brugerflade, hvor der kan dannes et overblik over hvordan hjemmesiden virker og ser ud.
- Django, som er frameworket brugt til hjemmesiden.

- Database, til brugere, faciliteter og logs.
- AWS, brugt til at lancere hjemmesiden.
- Sikkerhed, der er implementeret til hjemmesiden.

Implementering af hjemmesiden og databasen er foregået i Time Box 1-4 (se bilag 5, TB1-TB4).

Grafisk brugerflade

Hjemmesiden har overordnet tre forskellige brugerflader, én for hver brugertype (admin, office, og field). Hjemmesiden er struktureret sådan, at der udføres et tjek for, hvilken brugertype der logger ind på hjemmesiden, og alt efter hvilken brugertype brugeren er, sendes brugeren hen til brugertypens startside.

Brugertypen admin's startside er modificeret version af Django's egen admin template, se Figur 15, som viser hvilke elementer skal vises på "WEBSITE" fanen.

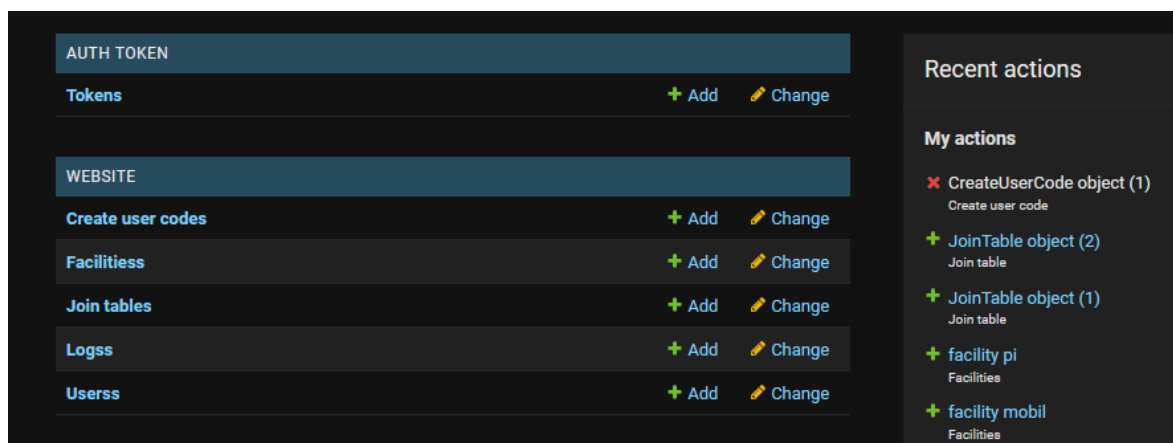
```

67 # The "users" section, applies a custom class
68 admin.site.register(Users, CustomUserAdmin)
69
70 # These use Django's standard templates
71 admin.site.register(Logs)
72 admin.site.register(Facilities)
73 admin.site.register(JoinTable)
74 admin.site.register(CreateUserCode)

```

Figur 15 Hjemmesidens admin.py (se Bilag 8, website)

Disse ændringer er som på Figur 16, hvor administratoren har adgang til at se, redigere, og slette alt.



Figur 16 Startside for en administrator

Startsiden for brugertypen office er som på Figur 17, hvor layoutet for startsiden vises, der er otte tilgængelige knapper og to informations bokse.



Figur 17 Starts siden for typen Office bruger

Øverste infobokse med "Home" stående i, giver information om hvor brugeren befinder sig på siden. Infoboksen med "Create user code: 71163303", viser den kode, som skal anvendelse til oprettelse af nye bruger. De otte knapper har følgende virkning:

- **Edit user** tager brugeren hen til en ny side, hvor brugeren kan finde andre brugere i form af to *dropdown* menuer (enten ved først at filtrere efter firma, eller blot at vælge ud fra alle brugere i systemet), hvorefter der kan redigeres i den ønskede brugers navn, firma, tlf. nummer, E-mail eller brugertype, brugertypen field kan opgraderes til brugertypen office, office kan ikke nedgradere andre office af sikkerhedsgrunde, grundet dette er det heller ikke muligt for office, at finde brugertyperne office og admin.
- **View facility logs** tager brugeren hen til en ny side, hvor brugeren kan sortere efter faciliteterne med en *dropdown* menu, enten ved først at

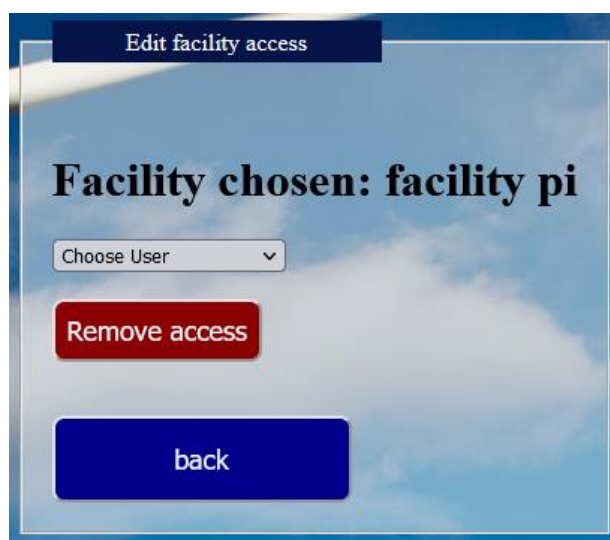
filtrere efter en lokation, eller en *dropdown* menu med alle faciliteter. Når en facilitet er valgt, kan alle logs for den valgte facilitet ses, som på Figur 18, hvor det er muligt at se navnet på den bruger som har tilgået facilitet, hvilket firma brugeren hører under, brugerens E-mail, og hvornår facilitet er tilgået.



Company	Person	Email	Date and time
ewe2	field	field@field.com	Nov. 9, 2021, 3:09 p.m.
ewe2	field	field@field.com	Nov. 10, 2021, 9:39 a.m.

Figur 18 facility logs for facility pi

- **View user logs** gør det samme som *View facility logs*, med undtagelsen af at sorteringen og visningen af logs er ud fra en valgt bruger.
- **Edit facility access** tager brugeren hen til en ny side, hvor der er tre nye valgmuligheder i form af tre knapper, *Give access*, *Remove access* og *Back*, hvor *Back* tager brugeren tilbage til startsideen. *Remove access* tager brugeren hen til en ny side, hvor brugeren kan vælge en facilitet via en *dropdown* menu, hvorefter brugeren tages til en ny side, se layout på Figur 19, hvor en bruger hvis adgang ønskes fjernet, kan vælges ud fra en *dropdown* menu, fjernelsen af adgangen færdiggøres ved tryk på *Remove access*.



Figur 19 layoutet for fjernelse af adgang til facilitet

Give access knappen tager brugeren videre til en ny side, hvor brugeren kan enten filtrere efter en lokation for at finde en facilitet, eller søge

igennem alle faciliteter. Når den facilitet som ønskes givet adgang til er valgt, tages bruger til en ny side, se sidens layout på Figur 20, hvor en bruger kan vælges med en *dropdown* menu, derefter kan en start og slutdato for adgangen vælges. Adgangen gives efter tryk på Give access, og hvis ingen start og slutdato vælges, vil der være permanent adgang til faciliteten.

Figur 20 Layoutet for at give adgang til en facilitet

- **Add/Edit/Remove facility** knappen tager brugeren hen til en ny side med fire knapper, *Back*, *Add facility*, *Edit facility* og *Remove facility*:
 - *Back* tager bruger tilbage til foregående side.
 - *Add facility* tager brugeren til en ny side, hvor det er muligt at tilføje en ny facilitet, felterne *Name*, *Location*, *Owner* og *Key* udfyldes, hvor facilitet tilføjes efter tryk på "Create facility".
 - *Edit facility* eller *Remove facility* sendes brugeren til en ny side, hvor brugeren kan filtrer efter facilitets lokation eller søge igennem alle faciliteter. Efter valg af facilitet, ved tryk på *Edit facility*, har brugeren mulighed for at redigere i *Name*, *Location* og *Owner*, hvor ændringerne gemmes ved tryk på "Accept edits".
 - Hvis der trykkes på *Remove facility* har brugeren mulighed for at se, hvilken facilitet der var valgt, og dens informationer *Name*, *Location* og *Owner*, ved trykke på "Remove facility", slettes faciliteten fra databasen, se Figur 21 for layoutet.



Figur 21 Remove facility layout

- **Personal info** tager brugeren til en nye side, hvor brugeren kan se egne oplysning, som navn, firma, telefonnummer og E-mail. Bruger har her, mulighed for at ændre telefonnummer, E-mail og kodeord.
- **Facility access** tager brugeren til en ny side, hvor brugeren kan se, hvilke faciliteter, brugeren har fået adgang til og i hvilke tidsrum adgangen er givet til.
- **Logout** knappen logger brugeren ud og sender brugeren hen til login siden.

Den røde boks på Figur 17 viser, hvilke knapper der er tilgængelig på startside for brugertypen field, som er de tre knapper:

- Logout
- Personal info
- Facility access

Disse knapper har samme funktionalitet, som de tilsvarende knapper ved brugertypen office.

Layout er skrevet med HTML og CSS, se Figur 22.

```

42 </fieldset>
43 <legend>Home page</legend>
44 <p style="..."> Create user code: {{ cu }} </p>
45 <button onClick="location.href = '{% url 'office_edit_user' %}'" style="..." id="edit_user">Edit user</button>
47 <button onClick="location.href = '{% url 'logs_facility_filter' %}'" style="...">View facility logs</button>
49 <br>
50 <button onClick="location.href = '{% url 'logs_user_filter' %}'" style="...">View user logs</button>
52 <button onClick="location.href = '{% url 'facility_access' %}'" style="...">Edit facility access</button>
54 <br>
55 <button onClick="location.href = '{% url 'facility' %}'" style="...">Add/Edit/Remove facility</button>
57 <br>
58 <br>
59 <button onClick="location.href = '{% url 'office_user_info' %}'" style="...">Personal info</button>
61 <button onClick="location.href = '{% url 'office_user_access' %}'" style="...">Facility access</button>
63 <br>
64 <br>
65 <button onClick="location.href = '{% url 'logout_user' %}'" style="...">Logout</button>
67 </fieldset>

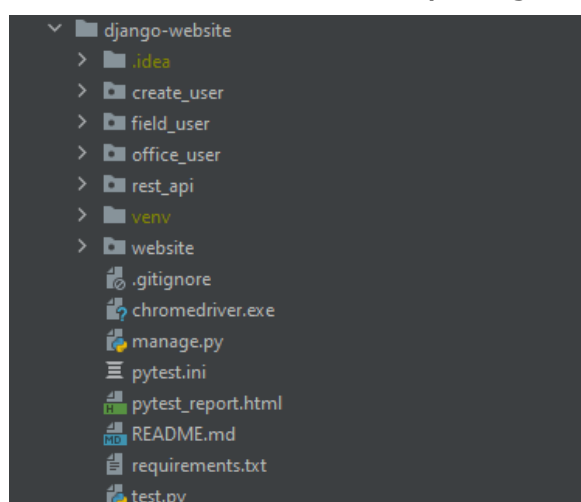
```

Figur 22 Office home layout (se Bilag 8, website)

Django

Som det første blev der udviklet en struktur for, hvordan udviklingen af hjemmesiden og databasen skulle forgå. For at sikre version styring blev Git [24] anvendt, hvor hvert medlem, har arbejdet på deres egen branch ud fra en branch lavet til de forskellige delemler (website, embedded enhed og mobil applikation) på GitHub [25].

Mappestruktur I Django har været således, at selve hjemmesiden med login og database er i mappen "website". Alt vedrørende oprettelse af ny bruger hører under "create_user". Hver brugertype hører under hver sin mappe, mappen "field_user" indeholder det til den almindelige bruger. Mappen "office_user" indeholder det til de bruger som skal administrere adgang til faciliteter og brugere, derfor er denne mappe også den største. Den sidste mappe "rest_api", indeholder det der skal anvendes til restAPI'en (se Figur 23).



Figur 23 overordenet mappestruktur website

Hver mappe har samme strukturform med views, urls, templates, static m.m.

I filen "views.py" er selve funktionalitet skrevet, hvor i filen "urls.py" er sammenkædningen mellem urls og views.py, Mappen "templates" indeholder alle HTML filerne, som er det siden viser til brugeren, mappen static indeholder alle de statiske elementer der anvendes som f.eks. billeder.

To mapper er lidt anderledes "rest_api" og "website".

- Mappen "rest_api" indeholder også en fil "serializers.py" som konvertere JSON om til et Python dictionary.
- Mappen "website" indeholder hovedapplikationen, hvor filen "settings.py" befinder sig, filerne "forms.py" og "models.py" som anvendes til databasen, det er også her filerne "asgi.py", "cron.py", "wsgi.py", som beskrives senere, befinder sig.

Settings.py filen er en af de vigtigste filer, det er blandet andet her de installeret applikation defineres, se Figur 24.

```
36 INSTALLED_APPS = [  
37     'website',  
38     'create_user',  
39     'field_user',  
40     'office_user',  
41     'rest_api',  
42     'rest_framework',  
43     'rest_framework.authtoken',  
44     'djoser',  
45     'django_crontab',  
46     'django.contrib.admin',  
47     'django.contrib.auth',  
48     'django.contrib.contenttypes',  
49     'django.contrib.sessions',  
50     'django.contrib.messages',  
51     'django.contrib.staticfiles',  
52 ]
```

Figur 24 settings.py (se Bilag 8, website)

Teamet har skrevet kodekommentarer så vidt muligt, at det er nødvendigt for at andre skal kunne forstå koden.

RestAPI

For at kunne få informationer fra databasen er der blevet lavet en restAPI på hjemmesiden. RestAPI'en gør det muligt at:

1. Få et token til at kunne bruge API funktionerne.
2. Hente bruger facilitets tilladelser.
3. Indsætte logs.
4. Slette token.

RestAPI'en er lavet ved hjælp af 2 Django python biblioteker:

- DjangoRESTframework (DRFW) V. 3.12.4 [26]
- Djoser V. 2.1.0 [27]

DRFW giver en standard template til at lave API'er i django, ved hjælp af python. For at bruge dette framework laves der 3 forskellige filer: URL, serializer og view, som vil styre endpoints for facilitets tilladelser og logs.

Serializer-filen indeholder en Python class for endpointene, hvor der defineres hvilke fields fra et SQL table der skal bruges, og serializer funktioner der er brugt til at formatere data mellem SQL og Python.

URL-filen indeholder URL adresserne til endpointene.

Views-filen indeholder Python classes for endpointene, hvori der kodes hvorledes interaktion med endpointene foregår, hvor flowet er som følgende:

1. Modtag data.
2. Behandl data.
3. Send data.

Et eksempel på flowet kan ses kode mæssigt på Figur 25.

```
53 class FacilityView(APIView):
54     def post(self, request):
55         post_data = UsersSerializer(data=request.data)
56         if post_data.is_valid():
57             data = post_data.data['userEmail']
58             pk = Users.objects.get(email=data)
59             list_fac = JoinTable.objects.all().filter(user=pk).values_list("facility__name", "facility__location",
60                                                                           "user__email", "user__company",
61                                                                           "user__name")
62             return Response({"status": "success", "list": list_fac}, status=status.HTTP_200_OK)
63         else:
64             return Response({"status": "error", "data": post_data.errors}, status=status.HTTP_400_BAD_REQUEST)
```

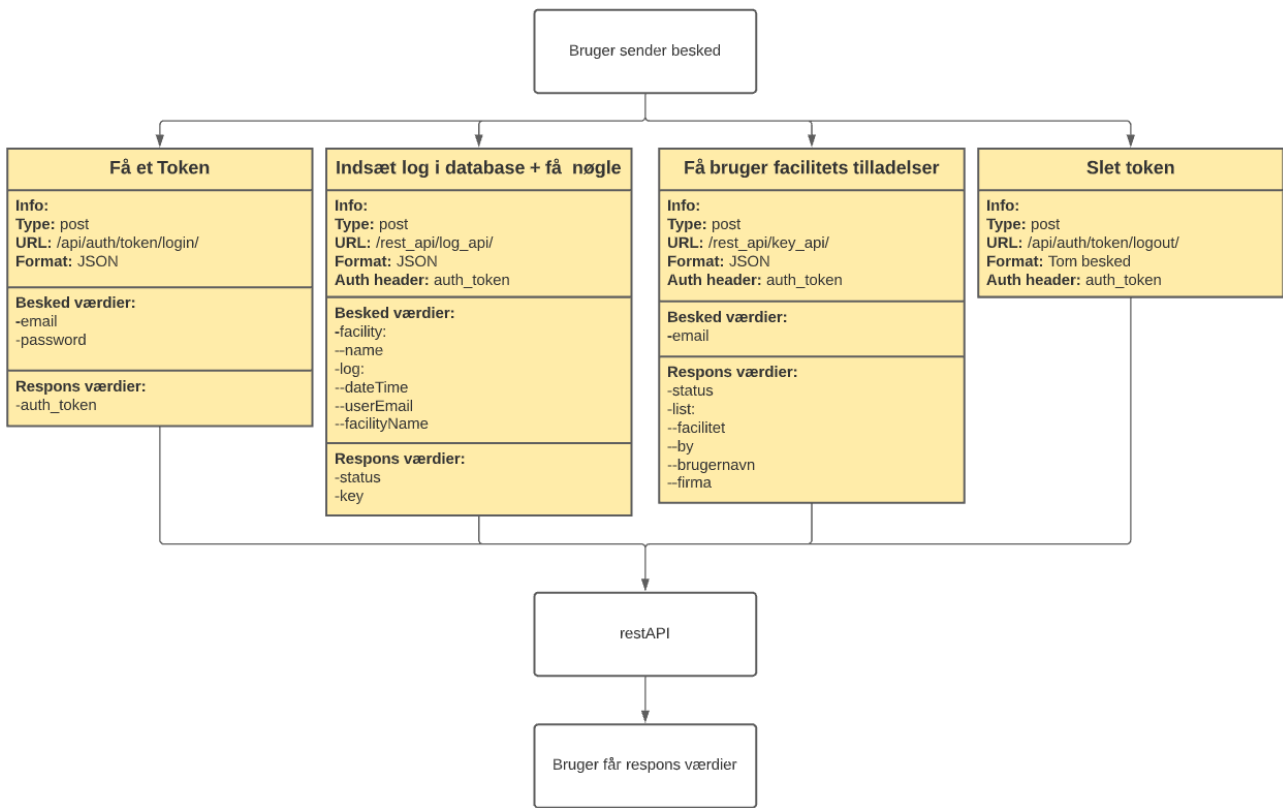
Figur 25 rest_api views.py (Bilag 8, website)

Djoser biblioteket giver muligheden for at bruge tokens til at bruge restAPI'en, hvilket gøres ved at have 2 endpoints. Et endpoint står for at give et token ved at sende login oplysninger til restAPI'en, mens det andet endpoint står for at slette tokens ved at sende tokenet til restAPI'ens logud endpoint. restAPI'ens tokens bruges til at kunne kontakte DRFW endpointene, hvor der højst kan være et enkelt token til hver bruger, på et hvilket som helst tidspunkt.

Protokollerne for restAPI'en kan ses på Figur 26, hvor:

- *Info*, beskriver om det er en POST eller GET request, URL endpointet der skal tilføjes til "https://www.control-center.xyz", beskedformatet, og hvad der skal være i headeren.
- *Besked værdier*, er de felter der skal være i beskeden, hvor "-" er et felt, og "--" er feltet i en liste, til det overstående "-".

- *Respons værdier*, er de værdier som vil blive returneret til brugerens request, og følger samme format som *besked værdier*.



Figur 26 restAPI protokol uml diagram

Hvordan restAPI'en specifikt er implementeret, kan ses i bilagene (se bilag 5, TB3, RAPI-auth & restAPI get key).

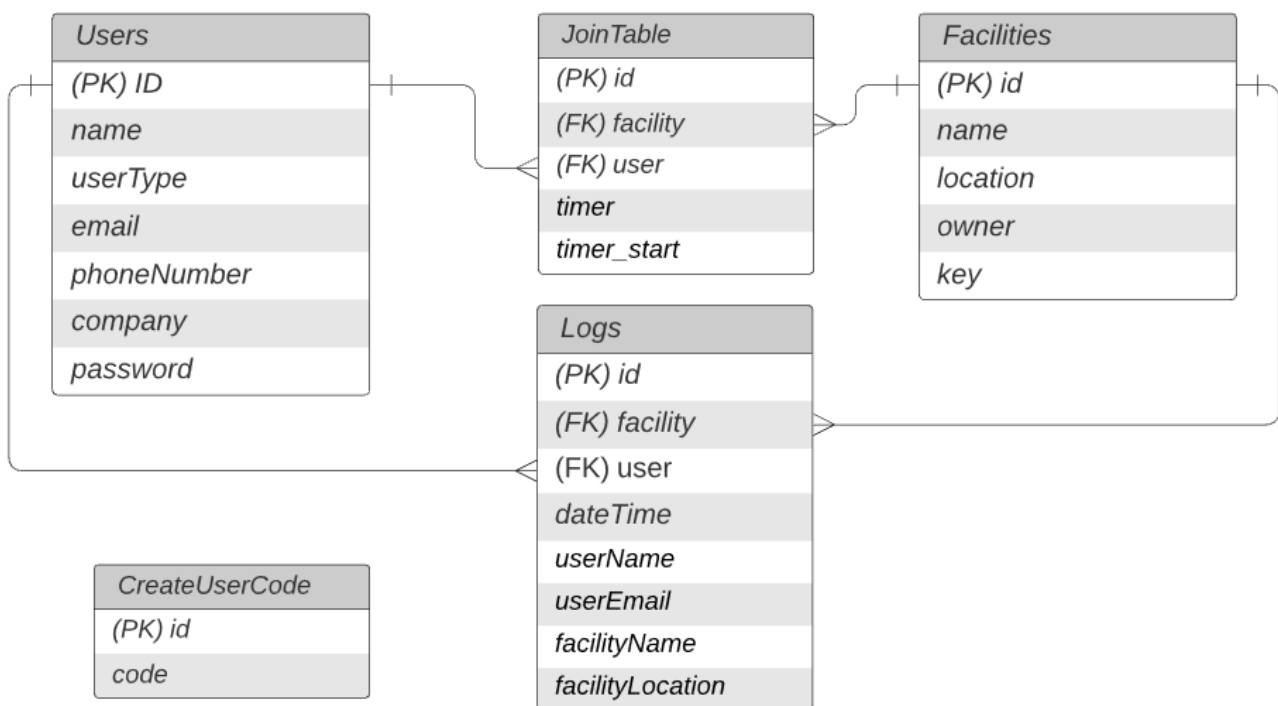
Database

Databasen og dens struktur er udviklet i starten af realiseringsfasens TB1 og senere i TB4 (se Bilag 5, TB1, TB1-DB struktur og TB4, TB4-DB struktur).

Databasen indeholder fem *tables*: *Users*, *Facilities*, *Logs*, *JoinTable* og *CreateUserCode*, hver *table* indeholder flere *fields* se Figur 27.

Mellem "Users" og "JoinTable" har vi en "one-to-many" relation, da vi har en bruger, som kan have adgang til flere faciliteter. Samme relation er der mellem "Facility" og "JoinTable", hvor en facilitet kan have flere brugere.

"Log" har samme relation til både "User" og "Facility" da en bruger kan have flere logs med samme facilitet, og en facilitet kan have flere forskellige brugere.



Figur 27 Database struktur (se Bliag 5, TB4, TB4-DB struktur)

Tabellen *Facilities* indeholder facilitets informationer og har disse fields:

- *id* er PK, dette felt er autogenereret
- *name* er facilitetens navn, dette felt er unikt
- *location* er facilitetens lokation
- *owner* er facilitetens ejer
- *key* er nøglen til faciliteten

Koden for tabellen *Facilities* er som på Figur 28, de andre tabeller kan ses i Bilag 8, website.

```

51 class Facilities(models.Model):
52     """
53     Model for facilities.
54     """
55     id = models.AutoField(primary_key=True)
56
57     name = models.CharField(max_length=80, unique=True)
58     location = models.CharField(max_length=120)
59     owner = models.CharField(max_length=80)
60     key = models.CharField(max_length=200)
61
62     def __str__(self):
63         return self.name
  
```

Figur 28 Facilities class (se Bilag 8, website)

Tabellen *Users* indeholder informationerne vedrørende en bruger, og har følgende *fields*:

- *ID* som er Primary key (PK), dette felt er autogeneret
- *name* er brugerens navn
- *email* er brugerens E-mail, dette felt er unikt
- *phoneNumber* er brugerens telefonen nummer
- *company* er virksomheden brugeren arbejder for
- *password* er brugerens kodeord, dette felt bliver krypteret med SHA256 [28] og saltet [29]
- *userType*, brugeren kan enten være field, office eller admin, alle oprettede brugere starter med at være field user

Tabellen *CreateUserCode* indeholder to felter:

- *id* er PK, dette felt er autogeneret
- *code* som er den kode, der skal anvendes ved oprettelse af nye brugere.

Der genereres en ny kode hver 24. time, med en scheduler (CRON).

Tabellen *JoinTable*, er den tabel der oprettes, når en bruger får adgang til en facilitet, og indeholder følgende felter:

- *id* er PK, dette felt er autogeneret
- *facility* er FK (til id) til tabellen *Facilities*
- *user* er FK (til ID) til tabellen *Users*
- *timer* er den tiden, adgangen til en facilitet ophører, hvis dette felt ikke er udfyldt eller NULL har bruger adgang bestandigt
- *timer_start* er tiden adgangen til en facilitet starter

For at sikre at en *JoinTable*, der er for gammel bliver slette, kører der en scheduler (CRON) hver dag, som tjekker feltet *timer* op mod dagens dato, se Figur 29.

```
28 def clean_access():
29     """
30     Cleans facility access
31     Removes all JoinTable that have expired
32     :return:
33     """
34     today = date.today() - timedelta(days=1)
35     yesterday = today.strftime("%Y-%m-%d")
36     JoinTable.objects.all().filter(timer=yesterday).delete()
37     pass
```

Figur 29 cron.py (se Bilag 8, website)

CRON scheduleren er sat op i setting.py filen som på Figur 30.

```

54 CRONJOBS = [
55     ('1 00 * * *', 'website.cron.gen_user_code'),
56     ('1 00 * * *', 'website.cron.clean_access')
57 ]

```

Figur 30 settings.py (se bilag 8, website)

Tabellen *Logs* indeholder informationer om, hvornår og hvilken facilitet er tilgået, og af hvem, og har felterne:

- *id* er PK, dette felt er autogeneret
- *facility* er en foreign key (FK), som refererer til PK (*id*) fra tabellen *Facilities*
- *user* er en FK, som refererer hen til en tabel *Users* PK (ID)
- *dateTime* er tidsstempet for, hvornår brugeren har tilgået nøglen
- *userName* er brugerens navn
- *facilityName* er facilitetens navn
- *facilityLocation* er facilitetslokationen

Sidste tre punkter gemmes uafhængigt af de to tabeller (*Users* og *facilities*) for at sikre, hvis brugeren eller facilitet slettes, består loggen stadig.

AWS

Inden der blev arbejdet på AWS serveren, skulle der først købes et domæne, hvilket blev gjort igennem GoDaddy [30], hvor domænet købt er *control-center.xyz* efter domænet var købt skulle alle name serverne sættes til dem der blev givet på lightsail [10] serverens DNS record.

For at serveren kører på AWS, efter AWS lightsail er betalt og startet op, skulle Django projektet klones fra GitHub, eller uploades med SSH forbindelse, hvorefter bibliotekerne fra projektets requirement fil skulle installeres, og til sidst installere Gunicorn [14], begge dele kunne gøres med PyPI [31].

Efter kloning og Gunicorn skulle der hentes et SSL-certifikat, hvilket blev gjort med Certbot [15]. Når Certbot var installeret kunne der køres et Certbot setup program, hvor der skulle gøres følgende:

- Indtaste email adresse
- Accepter "terms of service"
- Indtaste hjemmeside URL
- De udleverede txt records fra cerbot blev sat ind i lightsail DNS-servicen

Herefter skulle alle statiske filer i Django projektet, som fx billeder brugt til HTML'en, samles et sted, for derefter sætte NGINX på lightsail serveren op.

NGINX blev sat op til at gøre følgende:

- Alt HTTP trafik routes til HTTPS, som set på Figur 31
- HTTPS tager brug af Certbot SSL-certifikatet
- HTTPS header informationen sendes over til Gunicorn

- NGINX peger hen til Djangos samlede static filer

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;

    server_name control-center.xyz www.control-center.xyz;

    return 301 https://control-center.xyz$request_uri;
}
```

Figur 31 AWS cloudserver's nginx custom default fil

Når NGINX var sat op, kunne Unicorn sættes i gang, ved at pege Unicorn hen til Djangos wsgi.py fil [32], og sætte hvilken port der skal startes op på (local host port 8000).

En dybere forklaring af AWS-implementeringen kan læses i bilag (se bilag 5, TB4, AWS-Deployment).

Sikkerhed

Sikkerhedsmæssigt har følgende emner været taget til eftertanke:

- Sikring af hjemmeside trafik
- Sikring af bruger login
- Sikring af hvad brugertyper kan og kan se
- Sikre at uautoriserede personer ikke kan lave en bruger

For at sikre hjemmesidetrafikken er der blevet brugt SSL-certifikater fra Certbot og NGINX hvor implementeringen af disse kan læses i afsnittet AWS side 32.

For bruger login på hjemmesiden, er der blevet brugt Djangos standard login template, med nogle få overrides til template. De overrides der er blevet lavet er hvor en bruger bliver dirigeret hen, og hvad "user name" der skal bruges. Som standard bruger Django en standard brugertype, dog da der til projektet er blevet lavet en egenudviklet brugertype, er logintemplaten sat til at skulle bruge denne i stedet, hvor "user name" er blevet til "user email".

For at sikre en "field user" ikke kan gå ind på "office user's" dashboard, og bruge de dertil hørende funktioner, er der blevet lavet en dekorater, se Figur 32, til alle views, som tjekker brugerens brugertype, hvor hvis typen er lavere rangeret end hvad der prøves at blive tilgået, vil brugeren blive routet tilbage til deres egen brugertypes dashboard. Hierarkiet for de forskellige brugertyper, fra højest til lavest, er som følgende:

1. Administratorer
2. Office users
3. field users

```
13 # custom decorator
14 def user_controller(function):
15     """
16     Checks the user type, and redirects if wrong type
17
18     :param function:
19     :return:
20     """
21     @wraps(function)
22     def wrap(request, *args, **kwargs):
23         usertype = request.user.userType
24         if usertype == "Admin" or usertype == "Field user":
25             return function(request, *args, **kwargs)
26         elif usertype == "Office user":
27             return redirect('office_user_home')
28         else:
29             return redirect('front_page')
30
31     return wrap
```

Figur 32 field_user/views.py (se Bilag 8, website)

For at databasen ikke kan blive spammet med nye bruger, af personer med hensigt på at ødelægge hjemmesiden, skal der bruges en "create user code", som set på Figur 33.

Create new user

Create user code:

Email adress:

Name:

PhoneNumber:

Company:

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

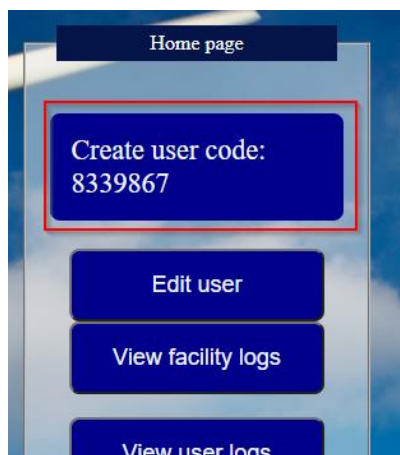
Password confirmation:

Enter the same password as before, for verification.

Back Sign up

Figur 33 create new user page

Koden brugt til at lave nye brugere, kan kun blive udgivet af enten en office eller admin bruger, som set på Figur 34, og bliver fornyet dagligt, igennem CRON [12] scheduleren, og er sat til at være pseudo-randomiseret af Pythons random funktion [33]

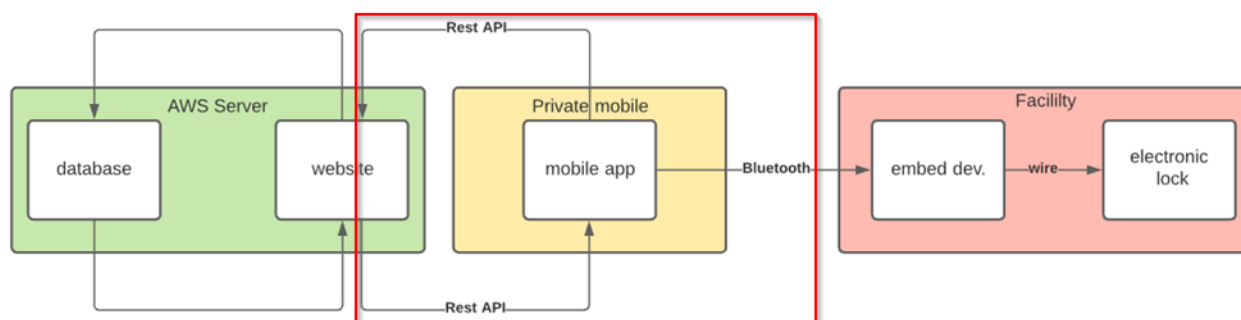


Figur 34 create user code

En overvejelse der har været, kryptering af nøglen/koden til låsene sendt via restAPI'en, dog blev der enighed om at med HTTPS-krypteringen på restAPI trafikken, vil det ikke være nødvendigt at yderlig kryptere disse nøgler/koder.

Mobilapplikation

Da ønsket fra Eurowind har været at der ikke skal udleveres fysiske nøgler til låsene, er der blevet udviklet en mobil applikation og embedded enhed med lås til dette, hvor denne sektion vil tage læseren igennem implementering af mobilapplikationen, som også kan ses på Figur 35, og er baseret på den tidligere analyse i *Embedded enhed med lås* på side 15.



Figur 35 mobilapplikation afsnit

Emnerne der bliver gennemgået er:

- Grafisk brugerflade, hvor der kan dannes et overblik over hvordan applikationen virker og ser ud.
- Gradle, som bestemmer fil opsætningen og hjælper med at kompilere programmet.
- RestAPI, som gør kontakt til hjemmesidens database mulig.
- Bluetooth, som gør kontakt med den embedded enhed mulig.
- Fragmenter, som er en vigtig del i hvordan programmet er sat op.
- Coroutines, som er hvordan resurse krævende opgaver er blevet håndteret.

Implementeringerne af mobilapplikationen er forgået i Time Box 5-8 (se bilag 5, TB5-TB8).

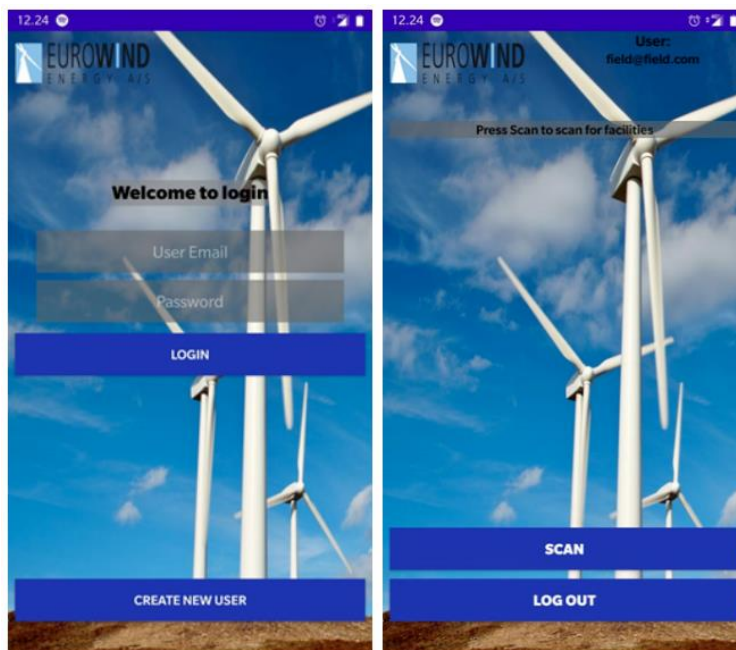
Grafisk brugerflade

Mobilapplikationens grafiske brugerflade har været udviklet med tankerne "keep it simpel" altså holde det simpelt, opfyldt ved at udvikle en applikation, med så få bruger interaktioner som muligt.

Ved start af applikationens kan brugeren se to knapper (LOG IN og CREATE NEW USER), en infoboks og to felt (se Figur 36, tv.). Hvis brugeren ikke er registeret som bruger af systemet, kan brugeren trykke på knappen "CREATE NEW USER", som dirigerer brugeren hen til hjemmesidens "Create user" side. Ellers udfyldes de to felter, ved at indtaste E-mail og kodeord, efterfulgt af tryk på knappen "LOG IN".

Efter brugeren har logget ind, kan brugeren se hvem brugeren er logget ind som, to knapper og en infoboks (se Figur 36, th). Brugeren kan gøre en af de to følgende:

- Trykke på knappen "LOG OUT", som logger brugeren ud og tager brugeren tilbage til login.
- Trykke på Knappen "SCAN", som starter Bluetooth modulet og skanner efter andre Bluetooth enheder.



Figur 36 tv. Login, th. efter login

På Figur 37 vises en lille kodelump fra Login fragmentet, som observerer responskoden efter logininformationen er sendt, hvis koden er 200, sendes både E-mail og Token med videre til næste fragment.

```
92 // Check response
93 viewModel.myResponse.observe(viewLifecycleOwner, { response ->
94 // check if login successful
95 if(response.code() == 200) {
96 // save token for onDestroy
97 tokenModel.setToken(response.body()?.auth_token.toString())
98 val test = tokenModel.getToken()
99 Log.d( tag: "ViewModel", test.toString())
100 // Create bundle
101 val bundle = Bundle()
102 bundle.putString("UserEmail", userEmail.text.toString())
103 bundle.putString("Token", response.body()?.auth_token.toString())
104 // Redirect to ScanFrag
105 Navigation.findNavController(view).navigate(R.id.action_loginFrag_to_scanFrag, bundle)
106 }
```

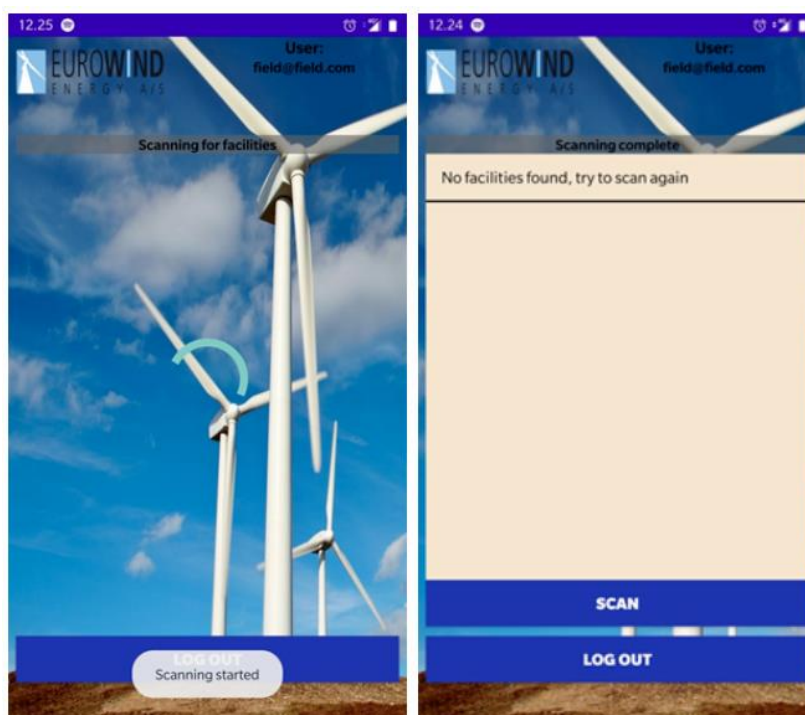
Figur 37 Login fragment (se Bilag 8, mobile)

Efter brugeren har startet skanningen for Bluetooth enheder, vises brugeren en cirkulær *loading bar* som indikere overfor brugeren, at mobilen arbejder (se

Figur 38, tv), derudover vises en Toast besked med "Scanning started", brugeren har mulighed for at trykke på knappe "LOG OUT", for at stoppe processen og logge ud.

Der er to mulige udfald af resultat af Bluetooth skanningen, enten er andre enheder fundet, ellers ingen enheder fundet.

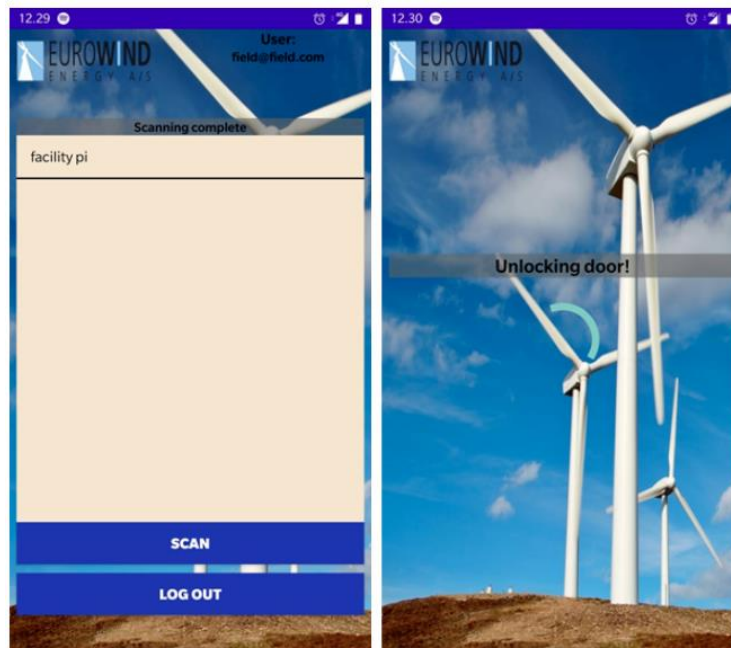
Hvis ingen enheder kan findes, informeres brugeren med beskeden *"No facilities found, try to scan again"*, og har mulighed for at trykke på knappen "SCAN" eller knappe "LOG OUT" (se Figur 38, th).



Figur 38 tv. Efter tryk på Scan, th. ingen enheder fundet

Ved fund af andre enheder, vises en liste med de fundne enheder, brugeren har derefter tre valgmuligheder, enten trykke på knapperne "SCAN" eller "LOG OUT", eller trykke på den facilitet, som ønskes oplåst (se Figur 39, tv).

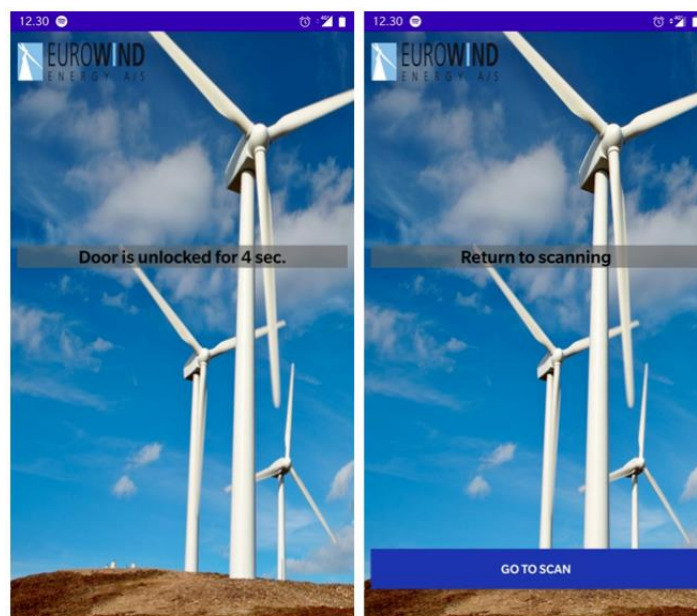
Når brugeren har trykket på en facilitet, vises brugeren en cirkulær *loading bar* og en infoboks med beskeden *"Unlocking door!"*, så brugeren ved at mobilen arbejder, brugeren har ingen mulighed for at påvirke denne proces (se Figur 39, th).



Figur 39 tv. enhed fundet, th. Unlocking sekvens

Efter sekvensen med at låse facilitetsdøren op, og alt er forgået korrekt, oplåses låsen og brugeren vises en ny besked på mobilapplikationen, som fortæller brugeren at låsen er låst op i en givet tid, denne tid tæller ned, for at informere brugeren, hvor langtid brugeren har til at åbne døren (se Figur 40, tv). Ved fejlet forsøg på at åbne låsen, informeres brugeren og brugeren logges ud.

Når nedtælling er færdig, vises brugeren en besked og har mulighed for at trykke på knappen "GO TO SCAN" (se Figur 40, th) som tager bruger tilbage til siden vist på Figur 36, th.



Figur 40 tv. låsen oplåst, th. afslutningen

Gradle

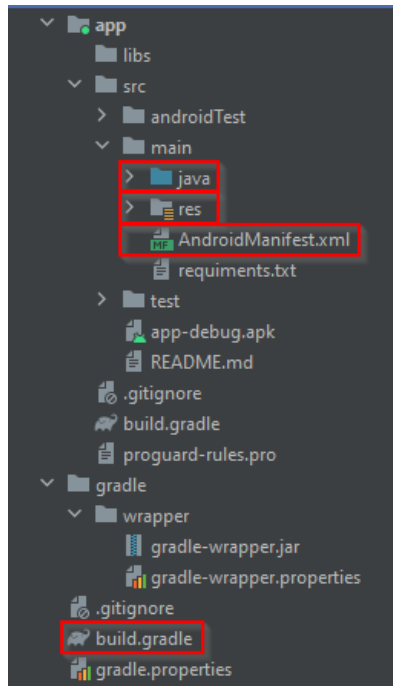
Ligesom hjemmesiden er styret af Django [7] frameworket, er mobilapplikationen styret af Gradle build system [34], som har dens egen mappe struktur, som igennem Android studio gør det nemt at inkorporere nye biblioteker til brug i mobilapplikationen, se udsnip af filen på Figur 41. Gradle gør det nemmere at kompilere koden til en apk fil, der kan installeres på en telefon. Gradle styrer selv den korrekte rækkefølge, filerne skal kompileres i, uden brug for bruger interaktion, under denne proces, samt under genkompilering, spilder Gradle ikke tid på at kompilere hele projektet forfra, men kun de ændrede filer.

```
49      //retrofit
50      implementation 'com.squareup.retrofit2:retrofit:2.9.0'
51      implementation 'com.squareup.retrofit2:converter-gson:2.6.0'
52      implementation 'com.squareup.okhttp3:logging-interceptor:4.5.0'
```

Figur 41 build.gradle (se Bilag 8, mobile)

Mappestrukturen for gradle er sat op som på Figur 42, hvor de vigtigste mapper under udvikling er:

- java mappen, som indeholder Kotlin filerne, som er koden der bestemmer hvordan applikationen skal agere.
- res mappen, indeholder billeder og filerne der styrer layoutet af applikation.
- AndroidManifest.xml filen, indeholder de forskellige tilladelser for applikation.
- build.gradle filen, indeholder de biblioteker der skal bruges i applikation.



Figur 42 Gradle mappe struktur

RestAPI

RestAPI'en på hjemmesiden, bliver taget i brug 4 gange af mobilapplikationen, ved login, log ud, skanning og låse op.

Ved login bliver restAPI'en brugt til at sende loginoplysningerne til "login" endpointet af restAPI'en, hvor hvis brugeroplysningerne er korrekte, bliver der returneret et token og statuskode 200. Hvis der modtages en 200 status, bliver loginet godkendt, og sender brugeren videre til scannings siden i applikationen, hvor hvis andet end 200 modtages bliver der givet en fejl meddelelse til brugeren.

Ved skanning bliver restAPI'ens "key_api" endpoint kontaktet, ved at indsende bruger email og token, hvorefter der bliver returneret en liste, med faciliteter brugeren har tilladelse til at kunne tilgå. Listen der bliver returneret fra endpointet, bliver brugt til at filtrere hvilke bluetooth enheder brugeren kan se i skanningslisten.

Når en facilitet bliver forsøgt åbnet, igennem applikationen, bliver restAPI'ens "log_api" endpoint kontaktet i starten af processen. Endpointet skal bruge facilitets navn, tidsstempel og brugernavn, hvorefter koden der skal bruges til faciliteten, bliver returneret, så applikationen kan fortsætte processen med at låse faciliteten op.

Når der logges ud af applikationen, bliver restAPI'ens "logout" endpoint kontaktet, der sørger for at slette tokenet, udleveret under login, fra databasen. Det eneste der skal sendes til dette endpoint er brugerens token.

En mere detaljeret forklaring af restAPI'en og dens protokoller, kan læses i afsnittet *RestAPI* side 27.

For at bruge restAPI'en igennem telefonen, bliver der brugt biblioteket "retrofit" [35], som kan give et java interface ind til en HTTPS API som f.eks. restAPI'en på hjemmesiden, samt bruges biblioteket "converter-gson" [36] til serialisering af JSON-formatet brugt til restAPI'en.

For at bruge retrofit og converter-gson, skal bibliotekerne skrives ind i gradle filen, og følgende filer skal udfærdiges:

- Modelfactory.kt til hver applikations side der skal kontakte restAPI'en.
 - Klasse til at interface mellem viewmodel og programkoden.
- Viewmodel.kt til hver applikations side der skal kontakte restAPI'en.
 - Klasse der bruges til at håndtere data til og fra restAPI'en.
- Constant.kt
 - Fil der indeholder konstanter som f.eks. URL-adresser.
- Repository.kt
 - Klasse der indeholder forud deklARATIONER for funktioner beregnet til at kontakte restAPI'en.
- Post.kt, se udsnit på Figur 43
 - Data klasser over hvad der skal være i beskederne sendt til restAPI'ens endpoints.

```
3  data class PostLogin (  
4      val email: String?,  
5      val password: String?,  
6      val auth_token: String?  
7  )
```

Figur 43 Post.kt (se Bilag 8, mobile)

- RetrofitInstance.kt
 - Et objekt hvor beskeden der skal sendes overordnet struktureres og kreeres.
- restAPI.kt
 - Forud funktion deklARATIONER, som indeholder hvordan de individuelle beskeder sendt til restAPI'ens endpoints skal struktureres.

Bluetooth

Der er nogle tilladelser, brugeren skal give, for at kunne bruge mobilens Bluetooth modul, hvilket er følgende tilladelser:

- Mobilens placering (krævet fra Android 9 og opefter)
- Bluetooth modulet er tændt (dette kan gøres uden brugertilladelse, men det er god stil at spørge)

Når mobilapplikation åbnes første gang, bliver brugeren af mobil spurgt om tilladelse til placering. Hver gang Bluetooth model skal anvendes udføres der et tjek for om Bluetooth modulet er tændt.

På Figur 44 , kan man se implementering af tjek af tilladelser.

```

132 private fun startBT() {
133     Log.d( tag: "sf_startBt", msg: "function started")
134     val permissionCheck = ContextCompat.checkSelfPermission(
135         requireActivity(),
136         Manifest.permission.ACCESS_FINE_LOCATION
137     )
138     if (permissionCheck == PackageManager.PERMISSION_GRANTED){
139         Log.d( tag: "sf_startBt", msg: "permissions granted")

```

Figur 44 Tilladelser for BT (se Bilag 8, mobile)

I fragmentet (

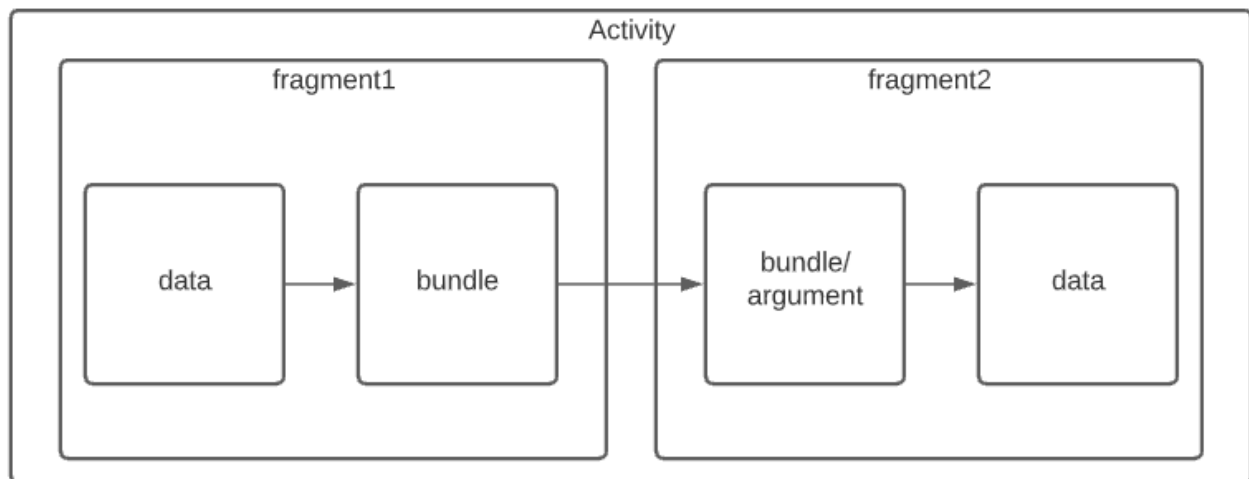
Fragmenter side 43) der står for at forbinde og låse låsen op, bliver der i hensyn til Bluetooth delen, først lavet en Bluetooth socket, med en *hardcoded* UUID, hvor den opretter en direkte u-krypteret forbindelse til en MAC-adresse, modtaget fra skan fragmentet.

Når der er oprettet en u-krypteret forbindelse, bliver nøglen, modtaget igennem restAPI'en, sendt igennem forbindelsen, for derefter at vente på en retur besked, med en status kode i.

Når retur beskeden er modtaget, bliver Bluetooth forbindelsen lukket, ved at lukke input og output strømmene, og til sidst lukke socket'en.

Fragmenter

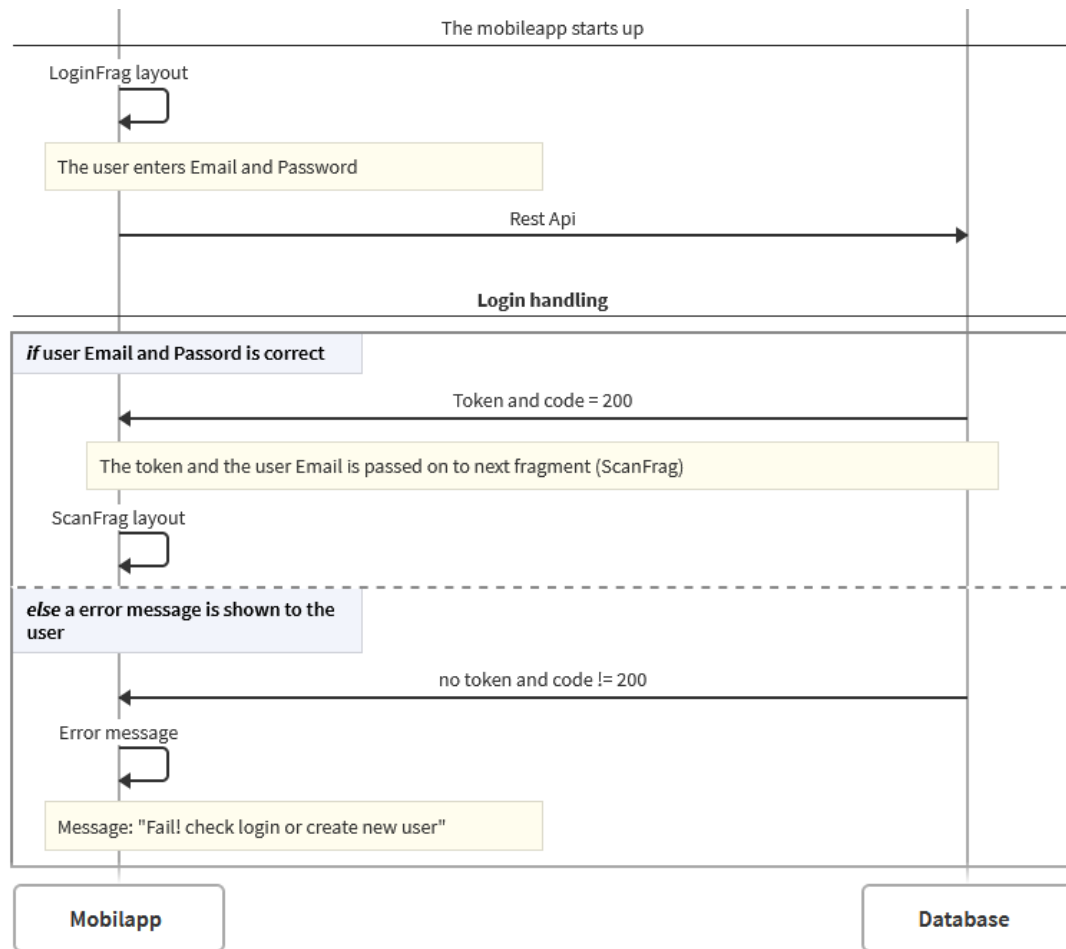
Idéen bag fragmenter er så det er nemmere at dele data, imellem flere forskellige sider på en app. Måden der deles mellem fragmenter, er at de fragmenter som kører i same activity ("process"), hvor data som skal passeres mellem fragmenterne, pakkes i et bundle, der som et argument sendes med til det næste fragment, hvorefter bundlet kan "udpakkes" og anvendes, se Figur 45.



Figur 45 eksempel på fragments

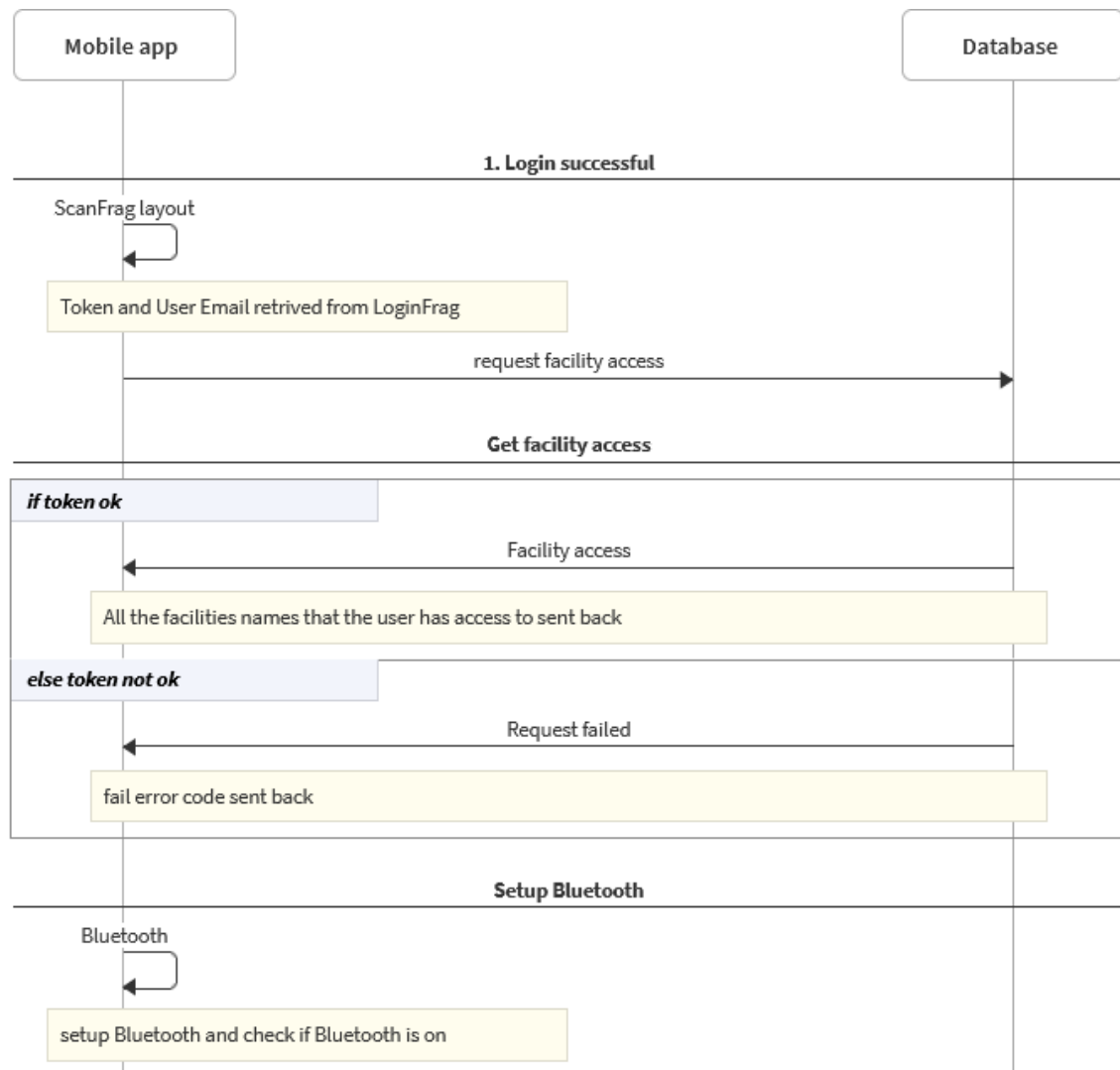
Denne mobilapplikation har fire fragmenter, hvor der skal passeres data imellem de forskellige fragmenter, de fire fragmenter er:

- **LoginFrag** kan dirigere brugeren hen til hjemmesidens "create new user" side. Ellers håndteres brugerens login, ved at sende de indtastede data med restAPI, hvor dataen valideres, hvor hvis valid sendes et token tilbage, hvorefter et bundle pakkes med token og brugerens E-mail, se Figur 46.



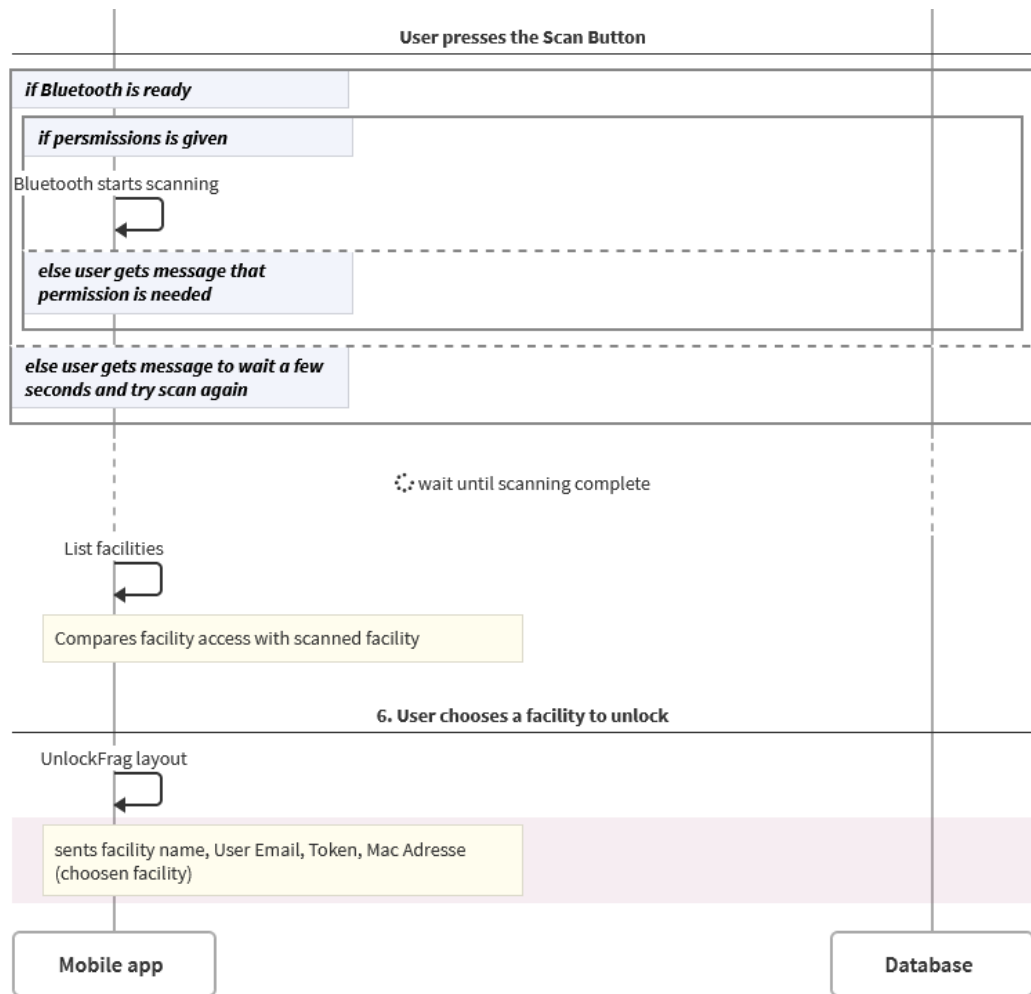
Figur 46 Login fragment

- **ScanFrag** modtager et bundle fra LoginFrag, henter brugerens information om hvilke faciliteter der er givet adgang til, for så at sætter Bluetooth op, se Figur 47.



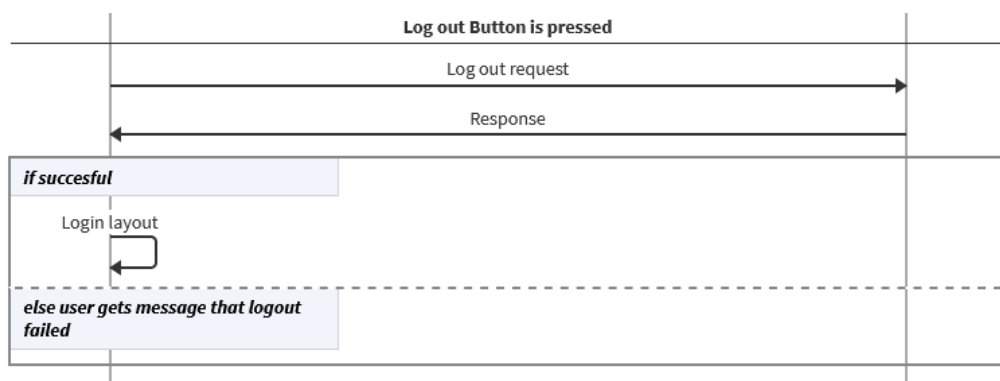
Figur 47 Scan fragment del 1

Efter Bluetooth opsætningen, håndteres skanning med Bluetooth, tjekker om der er givet adgang til Bluetooth, derefter skannes der efter enheder i nærheden, filtrere enhederne, så dem brugeren ikke har adgang til, ikke vises, se Figur 48.



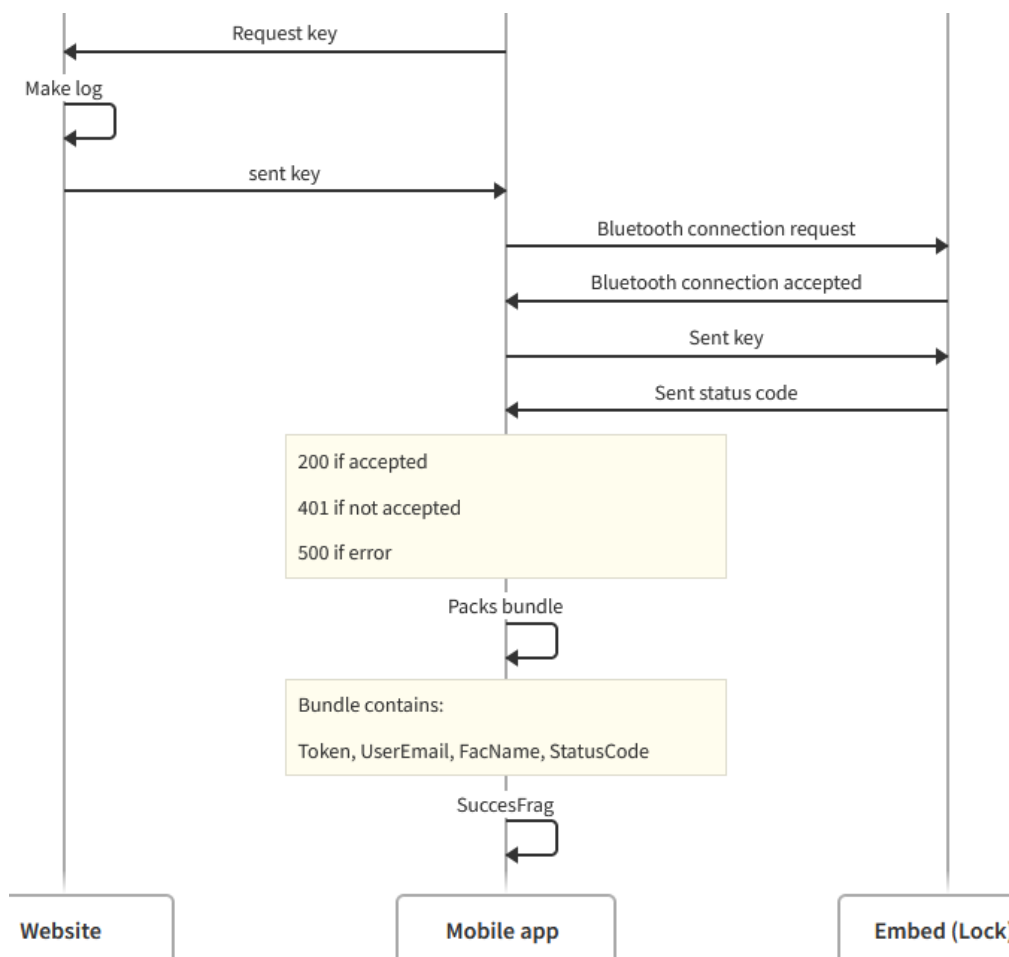
Figur 48 Scan fragment del 2

Når brugeren har valgt en facilitet, som skal låses op, pakkes et bundle med token, brugerens E-mail, navnet på valgt facilitet og MAC-adressen på den valgte facilitet. ScanFrag håndterer også bruger log out, se Figur 49.



Figur 49 Scan fragment del 3

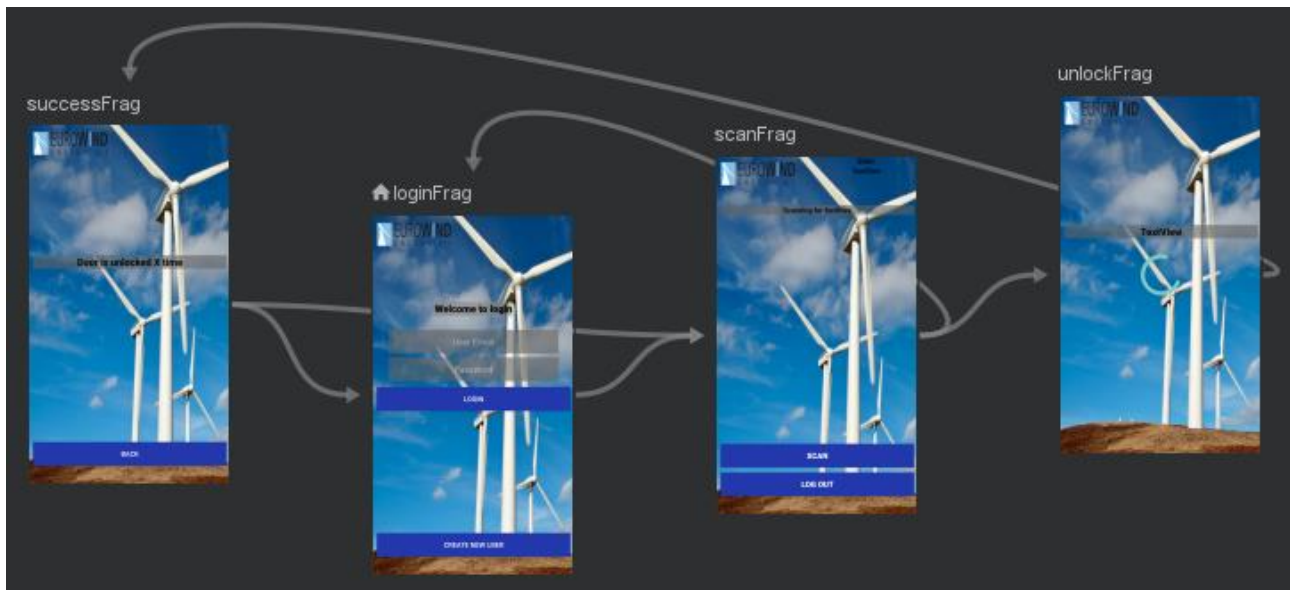
- **UnlockFrag** håndterer Bluetooth forbindelsen, henter nøglen med restAPI fra hjemmesiden, sender en Bluetooth besked med nøglen, pakker modtaget statuskode i en bundle og sender bruger videre til næste fragment, se Figur 50.



Figur 50 UnlockFrag swimlane

- **SuccesFrag** modtager en statuskode fra UnlockFrag, og agerer ud fra disse statuskoder:
 - 200 = success, giver en status til brugeren, døren er åben i 30 sekunder. Når timeren er udløbet, vil en knap blive synlig, for at kunne dirigere hen til ScanFrag.
 - 401 = forkert kode (nøgle) er sendt, brugeren logges ud og sendes til LoginFrag
 - 500 = fejl, der er enten sket en fejl på enhed eller i appen, brugeren sendes tilbage til ScanFrag

Hvordan fragmenterne er forbundet til hinanden, kan ses på Figur 51.



Figur 51 Autogeneret navigation diagram

Corutiner

Når telefon applikationens brugerflade vises på skærmen, kaldes det et "frame", og for at vise ændringer på brugerfladen, skal framen opdateres. Hvor hurtigt brugerfladens frames opdateres, kaldes en "framerate", og når det er hvor mange gange den opdateres pr. sekund, kaldes det "frames per second" eller "FPS". FPS'en på telefoner i dag er typisk 60 eller højere.

Hvis der på telefon bliver trukket på for mange resurser, til at køre f.eks. Bluetooth-forbindelsen og restAPI kaldende, vil applikationen prioritere baggrundsfunktionerne fremfor brugerfladen, hvilket vil kunne gøre at brugerfladen ikke bliver opdateret, hvilket kaldes for "skipped frames".

Ved brugerfladen for skanning kan skipped frames have den konsekvens, at de fundne devices i nærheden ikke bliver vist, og derfor ikke kan trykkes på, og derved åbne op for en facilitet.

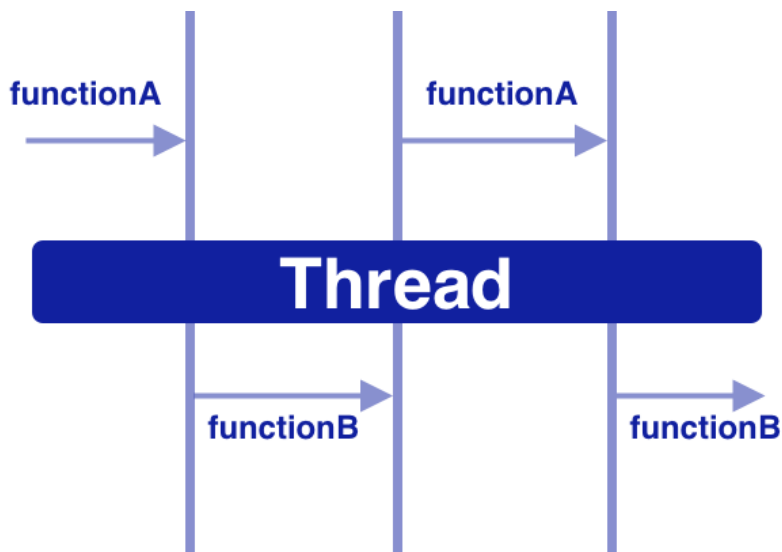
En anden problematik der kan være ved at køre mange opgaver på samme tråd, på samme tid, vil være at applikationen potentielt kan fryse, hvis f.eks. et kald til restAPI'en tager længere tid end planlagt, hvilket vil stoppe programmet, indtil restAPI kaldet er færdigt, og i yderste tilfælde kan applikation eller mobilen lukke ned pga. mangel på resurser.

For at undgå skipped frames og at appen fryser, bliver der brugt "coroutines" [37]. Coroutines bruges til at kunne køre asynkrone opgaver, undgå uønskelige blokeringer på hovedtråden af programmet, og lette arbejdsbyrden for hovedtråden. I en normal applikation er der hovedsageligt 3 tråde som kan bruges til at køre coroutines:

- Default, bruges normalt til CPU intensivt arbejde.
- Main, bruges normalt til let arbejde.
- IO, bruges normalt til netværk og disk adgang.

Medmindre andet specificeres, vil applikationen køre på main tråden.

En af fordelene ved at bruge coroutines er at programmets eksekverings orden af funktioner kan styres, ved at suspendere andre funktioner, som set på Figur 52, hvor funktion A bliver suspenderet mens funktion B eksekveres, for derefter fortsætte funktion A, hvilket minimerer risikoen for at applikation fryses i længere tid.



Figur 52 coroutine suspend [38]

Hovedforskellene mellem at bruge enten coroutines eller threads, er at:

- coroutine opgaverne ikke behøver altid at køre på samme tråd, men kan blive sat på "pause" og sat til at køre på en anden tråd, for så at blive startet igen.
- Coroutiner kører kun i applikationens levetid, hvilket minimerer chancen for "memory bleeding", hvis rutinen ikke lukkes ordentligt ned.
- Coroutiner er et forholdsvis lille objekt i java's memory management heap (JVM) [39], og bruger en forholdsvis lille mængde hukommelse, hvor threads styres af operativsystemet, og bruger en væsentlig større mængde resurser CPU mæssigt.

Til dette projekt anvendes coroutinerne i fragmentet *UnlockFrag*, ved at have en Bluetooth coroutine til at køre på IO tråden, mens restAPI kaldet og applikationens hovedfunktion kører på MAIN tråden, hvor coroutines kan startes som set på Figur 53.

```
83 //response okay -> launch coroutines
84 ➦ CoroutineScope(Dispatchers.IO).launch { connectBt() }
```

Figur 53 UnlockFrag.kt (se Bilag 8, mobile)

Embedded enhed med lås

Den embedded enhed står for at være et interface mellem mobilapplikationen og låsen, ved at kunne etablere en Bluetooth (BT) forbindelse til mobilapplikationen, og have fysisk forbindelse til låsen igennem enhedens GPIO'er.

Ved start af enheden, bliver Python programmet der styrer GPIO's og BT startet op, igennem rc_local filen der ligger på Linux, som en baggrunds proces.

Programmets flow kan ses på Figur 55, hvor der startes ud med et 3 sekunders delay, for at være sikker på at Raspberry pi'en (RP) er startet op, for derefter at lave en BT server opsætning, hvor enhedens BT-navn bliver sat, der bliver lavet en socket, og der bliver lavet BT advertismment, se Figur 54, for at mobiltelefoner kan finde den embedded enhed.

```
28     bluetooth.advertise_service(server_sock, "SampleServer", service_id=uuid,  
29                               service_classes=[uuid, bluetooth.SERIAL_PORT_CLASS],  
30                               profiles=[bluetooth.SERIAL_PORT_PROFILE]  
31                               )
```

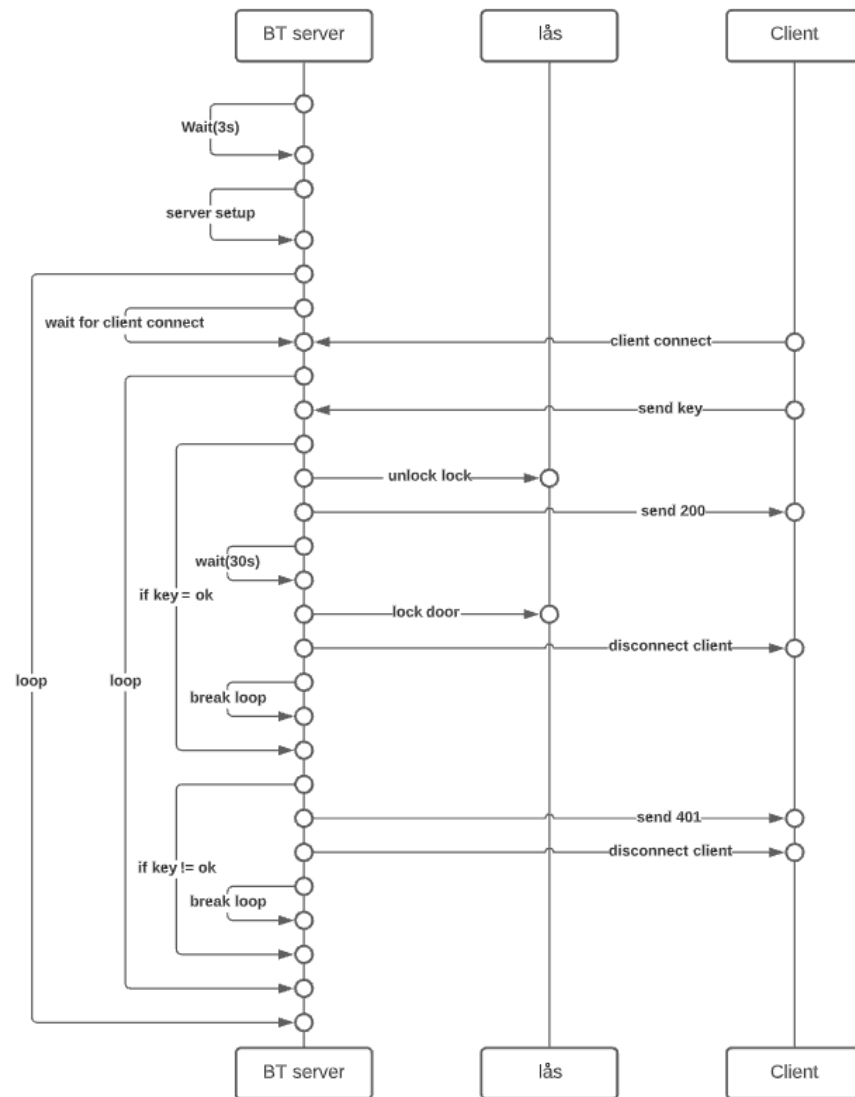
Figur 54 bt_class.py (se Bilag 8, embed)

Efter BT server opsætning, bliver der gået ind i et uendeligt loop, hvor i der er en blokerende funktion, som venter på indkommende klient forbindelser. Når en klient opretter forbindelse, bliver der kørt et nyt loop som venter på at der kommer en besked fra klienten.

Når der bliver modtaget en besked fra klienten, bliver der tjekket op om beskeden er ens med den gemte nøgle kode for låsen, hvor protokollen for dette er som følgende:

- Kodens er ens:
 - Handling: lås låsen op, send retur besked, og afbryd forbindelse til klient.
 - Retur besked: 200
- Kodens er ikke ens:
 - Handling: send retur besked og afbryd forbindelse til klienten
 - Retur besked: 401
- Noget gik galt på den embedded enhed:
 - Handling: send retur besked og afbryd forbindelsen til klienten.
 - Retur besked: 500

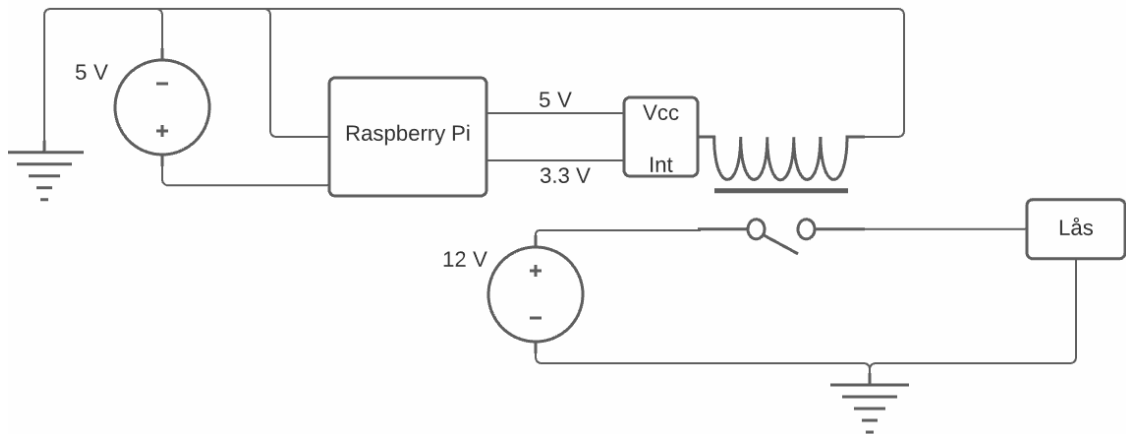
For hver gang forbindelsen til klienter afbrydes, bliver parringsregistret på den embedded enhed ryddet, for at minimere chancen for parrings fejl, eftersom der igennem applikationen ikke bliver parret med enheden, men på enheden står mobilapplikationen som parret.



Figur 55 embed bluetooth swimlane

Scheduleren brugt til at starte programmet op er som før nævnt rc_local filen, som kører dens kommandoer igennem hver gang RP'en bliver startet op. I stedet for rc_local, ville det have været mere optimalt at bruge Linux's CRON [12] scheduler, da rc_local kun kører ved opstart, men CRON vil kunne sættes til at periodiske tjekke om et program stadig kører, og vil kunne genstarte programmet, hvis den ikke kører.

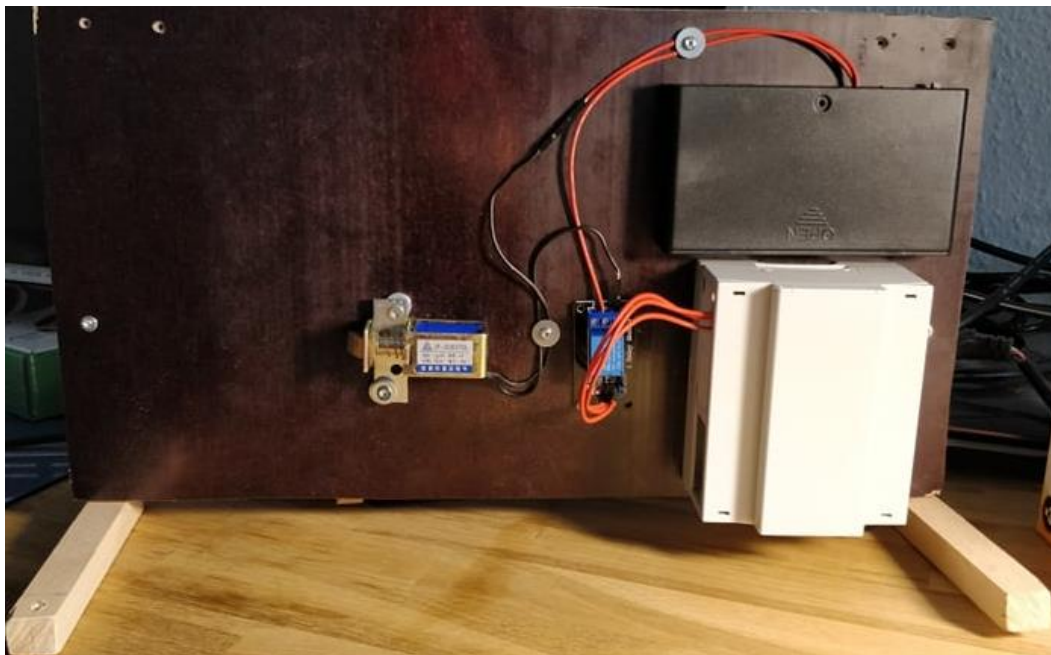
RP'en har en forsyning på 5 V og sender 5 V forsyning (Vcc) ud til relæet. Til at styre relæet (tænd/sluk), anvendes en GPIO med 3.3 V (Int), derudover går der en ledning mere mellem relæet og RP (fælles ground). Når der er strøm i relæet, bliver låsen tilsluttet 12 V, fra en batteriholder, som låser låsen op disse er forbundet til fælles ground. Kredsløbsdiagrammet er som på Figur 56.



Figur 56 kredsløbsdiagram

Der er ingen strømmålinger lavet, da Eurowind ikke er interesseret, hvor meget strøm der trækkes, da hoved idéen er at produktet skulle placeres ude ved transformere, hvor der er masser af strøm.

Selve opsætningen af PoC delen, er som på Figur 57, hvor udstyret er monteret på en træplade med to træben. Denne montering gør produktet nemmere at stille på et bord og præsentere overfor andre.



Figur 57 PoC setup

Test

Hver arbejdsopgave i realiseringsfase (se Bilag 5, alle TB), har et afsnit med test, hvor det der er udviklet, er blevet testet modulært, før integration som et samlet system (PoC), hvorefter hele systemet er testet samlet.

Test af de enkelte dele, har forgået således:

- Test af Website (hjemmesiden og databasen), har forgået i to fase, før udrulningen på AWS-serveren og efter. Før udrulningen har test af hver arbejdsopgave været manuelt udført på *local host*, senere er testene udført med automatiske test, da i takt med det udviklet system, blev større og krævede mere omfattende og repetitive test, gjort med værktøjet Selenium [40] (se bilag 5, TB4). Efter udrulningen har testene været fokuseret mere omkring opsætning af AWS-serveren og systemet som helhed på AWS-serveren, se en af unittestcasene med Selenium på Figur 58.

```
13 class LoginTestCase(unittest.TestCase):
14     """ ... """
24
25     def setUp(self):...
31
32     def test_login_field(self):...
60
61     def test_login_office(self):...
89
90     def test_login_admin(self):...
118
119     def tearDown(self):...
```

Figur 58 test.py (se Bilag 8, website)

- Test af Embedded enhed med lås, hvor først er de forskellige dele (låsen, relæet og RP) testet efter om de virker, hver for sig, før delene er sat sammen, hvorefter en samlet test af systemet er udført. For at teste BT delen af RP, er en ekstern mobilapp "Serial Bluetooth" anvendt, disse test er dokumenteret i Bilag 5, TB4-BT.
- Test af mobilapplikationen, har været test af de forskellige fragmenter, hver for sig, og dokumenteret (se Bilag 5, TB7-LoginFrag, ScanFrag, SuccessFrag og UnlockFrag), efterfulgt af en større integrationstest af hele mobilens applikation og PoC samlet

Per aftale med Eurowind, er hver del (Website, Embedded enhed med lås og Mobilapplikation) af projekt leveret, ved dels færdiggørelse, efterfulgt af en større levering af produktet som et færdigt PoC.

For at levere PoC som færdig, er en større test af produktet som helhed udført, baseret ud fra *Launch Phase* rapportens afsnit *Product Acceptance* (se Bilag 3), som er dokumenteret (se Bilag 6, Product Acceptance), hvori der beskrives krav, teststeps, testresultater og om kunden (Eurowind) har accepteret.

Postprojekt

Postprojektet omhandler afslutning af projektet, overlevering af produktet til kunden og evalueringen af projektet, produktet, forløbet og diskussion.

Product Acceptance

For at få godkendt produktet af Eurowind, blev der holdt et møde med én af vores kontaktpersoner fra Eurowind, Søren G. Jeppesen, hvor produktet i sin helhed blev vist frem og demonstreret, hvorefter dokumentet *product acceptance* (Bilag 6, product acceptance), blev gennemgået.

Product acceptance dokumentet, som set i udklipet på Figur 59, består af:

- *testID*, er ID'et på *acceptance* testen.
- *kravID*, er hvilket krav testen opfylder.
- *Teststeps*, er hvilke steps der skal gøres i testen.
- *Testresultat*, er resultatet for testen.
- *Kunden*, er om kunden har godtaget testen.

TestID	KravID	Teststeps	Testresultat	Kunden
T-1	WEB-1	1. Klik på "Create new user". 2. Udfyld indtastningsfilterne 3. klik på "Create user"	Den nye bruger er oprettet i databasen.	Accepteret
T-2	WEB-2	1. Log in på "office" brugeren 2. klik på "Edit user" 3. Vælg den bruger som der skal slettes 4. klik på "Edit user" 5. klik på "Delete user"	Den pågældende bruger er slettet i databasen.	Accepteret

Figur 59 product acceptance udklip

Alle product acceptance testene blev bestået, på nær testen med fingerskan, da dette blev udeladt af produktet, som er forklaret hvorfor i bilag 6, sikkerhed, afsnit fingerskan.

Overleveringen af projektet til Eurowind, vil ske kort efter eksamen, da det fysiske af produktet skal bruges til fremvisning derunder. Til overleveringen vil alle dokumenter og kode ligge på Github [25], og de fysiske elementer vil også her bliver overleveret.

Udviklingsprisen for produktet blev lavere end hvad der var forventet (se Bilag 4, Launch rapport, afsnit *quotation*), hvor prisen har været:

- Forventet pris: 440 kr.
- Endelig pris: 270 kr.

Grunden for den lavere pris er:

- AWS er gratis de første 3 mdr.

- Der blev gået med den billige elektroniske lås, fremfor den dyre magnet lås.
- Det var ikke nødvendigt med et batteri til den embedded enhed, da teamet havde en powerbank til at ligge.
- Der blev ikke oprettet en Google play store konto, men bliver derimod brugt Github i stedet, til at dele mobilapplikationens apk (installations) fil (kan findes i Bilag 8, mobile, app-debug.apk).

Produktevaluering

Alle krav fra requirements (se Bilag 4, Launch Rapport, afsnit requirements analysis) blev opfyldt, på nær fingerskan kravet, som ikke ville kunne implementeres, som et ekstra sikkerhedslag. Hvis fingerskan implementeres skulle det være for brugervenlighed og ikke sikkerhed.

Eurowind har både accepteret produktet, og forklaring på fingerskan, og er ydersttilfreds med PoC, Kim Moberg (kontakt person fra Eurowind) var imponeret over, hvor langt projektet var nået.

Med Søren G. Jeppesen (kontaktperson fra Eurowind) blev der diskuteret om, hvorledes man kunne udvide hierarkiet af brugertyper yderlig, så det var muligt at give en bruger af systemet tilladelse til og give adgang til en facilitet eller facilitetsgruppe, hvor teamet mener det er muligt, at gå tilbage og implementere dette.

Eurowind har fået det produkt, de havde ønsket, på trods af den korte udviklings tid, med dette taget i betragtningen, er Teamet er stolte over produktet som PoC, dog for at PoC produktet vil kunne tages i produktion, vil der skulle laves en del ekstra, hvilket vil blive berørt i

Fremtidigt arbejde side 61.

Ved møde med Søren har der været forhørt efter om projektet eventuelt vil kunne tilpasses Hobro idrætsforening, da det udfærdigede PoC produkt, ville kunne opfylde mange af idrætsforeningens ønsker og mangler, efter projektets afslutning.

Projektevaluering

Samarbejdet mellem Eurowind og teamet har været tilpas, med et minimum antal af møder under realiseringsfasen, hvor der kun har været møder, når en af de overordnede emner (website, mobil applikation og embedded enhed med lås) er blevet "færdiggjort" og klar til "levering", hvilket har været planen mellem Eurowind og teamet fra starten.

Ved alle møderne med Eurowind, har teamet været fysisk til stede i Hobro, dette har været med til at skabe en bedre kommunikation mellem parterne. Møderne omhandlede i mindre grad det tekniske, i forhold til det overordnet system som helhed.

Fremdrift af projektet har været drevet af god planlægning, som har været essentielt for projektet, hvilket har bidraget til at projektet, som PoC, har kunne nå i mål, uden for mange sene nætter og arbejde i weekenden. Teamet har nydt godt af at mødes om morgen, tage en snak om, hvad der skulle arbejdes på i løbet af dagen, for derefter at arbejde, hver for sig.

Teamet har fulgt EUDP-modellen, hvor artefakterne er dokumenteret i Bilag 3-5, anvendelsen af denne model har givet teamet en rød tråd fra projektets start til slut.

Der har været anvendt to udviklingsplaner, en for launch fasen og en for realiseringsfasen, de to første har været fulgt som planlagt uden nogen nævneværdige ændringer. Planen for realisering har været mere omfattende, med nogle ændringer især vedrørende mobilapplikationen, dette var dog forventet, da ingen i teamet havde udviklet eller haft teori omkring mobilapplikationer. Teamet har haft travlt, men ved hjælp af disse to udviklingsplaner har teamet været i stand til at følge planen og nå i mål med projektet.

Eurowind har været tilfreds med projektet og dens forløb, og har flere gange snakket om samarbejde og job efter projektet. Eurowind har opnået det de ville med projektet, for dem har det udover selve projektet, været en mulighed for at observere, hvad en elektronik ingeniør kan, og hvordan en elektronik ingeniør vil tilgå et nyt projekt. De ser det positivt, at vores mentalitet med nye emner er, at "springe ud i det", og lærer mere om emnet, yderlig er de imponeret over mængden af dokumentation, der er blevet lavet.

Mængden af kommunikation mellem Eurowind og teamet har været tilpas, med plads til flere møder. Eurowind har haft en travl periode, så nogle af de aftalte møder har været aflyst, ellers har det kun været muligt at holde møder med enten Kim Moberg eller Søren G. Jeppesen individuelt, dette har været suboptimalt, men teamet har tilpasset sig efter omstændighederne.

Der har i starten været snakket med firmaerne Assa Abloy [41] og Abus [42], omkring muligt samarbejde angående låsen til projektet, dog har disse firmaer ikke været interesseret i dette, grundet deres egne interne sikkerhed og markeds interesse.

Teamet har være yderst tilfreds med samarbejdet internt i teamet, og projektets forløb.

Konklusion

Der er blevet udviklet en mobil applikation, som vil kunne installeres på Android telefoner. Applikationen vil agere som nøgle, hvilket gør at Eurowind ikke vil behøve at udlevere en fysisk nøgle, som både skal hentes og afleveres på et kontor, som potentielt vil kunne forsvinde. Mobilapplikationen er blevet lavet så brugervenligt som muligt, ved at fjerne mængden af mulige knapper og funktioner. Mobilapplikationen kontakter en Raspberry pi 3 B+, ved hjælp af Bluetooth, som styrer låsen igennem en fysisk forbindelse.

Som Online central styringsplatform er der udviklet en hjemmeside og database som er udrullet på en AWS-server. Hjemmesiden agerer som selve styringsplatform, hvor der kan oprettes nye brugere til systemet, og hvor adgangen til faciliteter kan styres. Hjemmesiden indeholder de funktionaliteter som er nødvendige, for at den vil være anvendelig for Eurowind.

Hardware mæssigt er produktet gjort mindre kompleks, i forhold til andre produktet på markedet. PoC kræver ikke at den embedded enhed har internetadgang, men skal kun forsynes med strøm. Som et simpelt setup er de eneste fysiske objekter den embedded enhed, relæ, lås og batterier.

For at mindske brugerfladen, og samtidig dække Eurowinds administrative behov, er der opsat en restAPI, som uddelegerer facilitetsnøglerne og laver en log med brugerens oplysninger, samt er der opsat en CRON scheduler som rydder op i adgangstilladelser til faciliteter. På grund af dette er der på brugerfladen kun nødvendigt at have følgende funktionaliteter:

- Oprette, redigere og slette bruger.
- Se bruger informationer, og hvor en bruger har adgang til.
- Oprette, slette, styre adgang til faciliteter.
- Se logs over hvem der har tilgået hvilke faciliteter og hvornår.

Alle Eurowinds krav er blevet opnået, på nær at bruge fingerskanning til mobilapplikationen, som et ekstra sikkerhedslag, da måden fingerskanning er sat op på i Android telefoner, ikke er ment til dette.

Det udviklet PoC kan anvendes til det Eurowind har haft i tængerne, nemlig at vise ideen om en online styringsplatform, til både medarbejder og bestyrelse, men også overfor de firmaer som udbyder lignende løsning.

Eurowind har været interesseret i hvordan dokumentationen af sådanne et projekt kan udføres, og bliver overleveret til Eurowind gennem et Github repository, sammen med alt kode udviklet under projektet. Dette kan give ideer til, hvordan de eventuelt skulle strukturere deres fremtidige projekter

Projektstyringsmetoden, har været gjort en smule anderledes, end ved andre projekter lavet af teamet, ved at simultant bruge opgave lister, Kanbans og Gantt diagrammer i samme projekt, hvilket har haft en stor positiv effekt på

projektudførelsen, da det gav en god struktur, et godt flow til arbejdsgangen og et godt overblik af projektet.

Sikkerhedsmæssigt er der implementeret brugerlogin med SHA256 kryptering på passwords, og SSL-certifikat til HTTPS-trafikken. For at sikre uautoriserede brugere ikke kan oprettes, er der blevet implementeret en "create user code", der skiftes dagligt. Følgende brugertype hierarki, fra lavest til højest, er implementeret: field, office, admin, hvor office-brugeren vil kunne udlevere "create user code" og styre adgangen til faciliteter.

Som helhed kan det konkluderes at projektet er blevet færdiggjort til et tilfredsstillende niveau, for både Eurowind og teamet.

Fremtidigt arbejde

Der vil ved en videreudvikling af PoC, skulle adresseres nogle emner som:

- GDPR, teamet har haft mulighed for at snakke med Eurowinds advokat (Se bilag 6, *svaret på GDPR spørgsmål*), hvor deres svar til om der må gemmes brugeroplysninger var: *"..., at personoplysningerne ikke må opbevares i et længere tidsrum end det, der er nødvendigt til det formål, hvortil de er indsamlet."*. Dvs. at der skal findes en tidsfrist for sletningen af brugerdata også dem der gemmes i loggen, som pt. gemmes permanent. Advokaterne svarede også på om det var nødvendigt med en samtykkeerklæring, ved brugeroprettelsen for at kunne lovligt gemme brugerinformation, som kunne lede hen til en specifik bruger, til det var deres svar: *"... i jeres tilfælde er der tale om behandling af personoplysninger som er nødvendig af hensyn til en retlig forpligtigelse, altså en forpligtigelse som det følger af lovgivningen. Derfor er det ikke nødvendigt for jer at lave en samtykkeerklæring, og I behøver den derfor ikke."*, derfor skal der ikke umiddelbart udvikles en samtykkeerklæring. Dette emne er dog mere kompleks og burde tjekkes igennem af en advokat inden en eventuel udrulning af produktet.
- Bluetooth forbindelsen, har lige nu ingen kryptering af forbindelsen mellem den embedded enhed og mobilapplikationen, dette gør systemet sårbart over f.eks. "Man in the middle attack" [42].
- Databasen kan gøres mere sikker, ved f.eks. at adskille hjemmesiden og databasen, så de befinder sig på hver af deres egen server, og opdele elementer i f.eks. Docker containere.
- Hjemmesidens sikkerhed kan udbedres ved blandt andet, flere SSL-certifikater og rotere mellem dem, så hvis en er komprimeret, ville det ikke automatisk give adgang til hjemmesiden. Det er også muligt at opdele hjemmesiden i flere elementer, hvor hvert element har egne roterende SSL-certifikater.
- Designet på hjemmesiden kan gøres mere brugervenlig, ved blandt andet at implementere Bootstrap, så hjemmesiden dynamisk tilpasser sig flere platformes skærmstørrelse.
- Mobilapplikationen kan udvikles mere robust, ved blandt andet at indføre flere tjeks og håndtering af dem, og yderlig test af mobilapplikationen, med flere forskellige Android versioner. Det vil kunne hvis nødvendigt, udvides med en applikation til Iphone, for udvide til flere mobile platforme.

Der er flere mulige scenarier for videre udvikling af PoC:

- Eurowind eller et andet firma udvikler PoC videre med API adgang til en eksisterende lås.
- Eurowind eller et andet firma udvikler PoC med henblik på at udgive egen produkt

- Udviklingsteamet start egen virksomhed, med henblik på at videre udvikle produktet.

Referencer

- [1] »Eurowind Energy,« [Online]. Available: <https://eurowindenergy.com/DK.aspx>. [Senest hentet eller vist den 09 12 2021].
- [2] »EUDP,« 01 12 2021. [Online]. Available: http://eudp.net/index.php/Main_Page.
- [3] »EUDP roadmap,« [Online]. Available: <http://eudp.net/index.php/Roadmap>. [Senest hentet eller vist den 24 11 2021].
- [4] »Sprint,« [Online]. Available: https://en.wikipedia.org/wiki/Scrum_Sprint. [Senest hentet eller vist den 25 11 2021].
- [5] »Gantt,« [Online]. Available: https://en.wikipedia.org/wiki/Gantt_chart. [Senest hentet eller vist den 25 11 2021].
- [6] »Kanban,« [Online]. Available: <https://www.projectmanager.com/blog/a-quick-guide-to-kanban-cards>. [Senest hentet eller vist den 25 11 2021].
- [7] »Django,« [Online]. Available: [https://en.wikipedia.org/wiki/Django_\(web_framework\)](https://en.wikipedia.org/wiki/Django_(web_framework)). [Senest hentet eller vist den 25 11 2021].
- [8] »Laravel,« [Online]. Available: <https://en.wikipedia.org/wiki/Laravel>. [Senest hentet eller vist den 25 11 2021].
- [9] »Pycharm,« [Online]. Available: <https://www.jetbrains.com/pycharm/>. [Senest hentet eller vist den 25 11 2021].
- [10] »Lightsail,« [Online]. Available: <https://aws.amazon.com/lightsail/>. [Senest hentet eller vist den 26 11 2021].
- [11] »Azure,« [Online]. Available: https://sentia.com/dk/teknologi/microsoft-azure/?utm_source=adwords&utm_medium=ppc&utm_term=microsoft%20azure&utm_campaign=G:+Azure&hsa_cam=10368884565&hsa_mt=e&hsa_ver=3&hsa_src=g&hsa_ad=454139174593&hsa_net=adwords

&hsa_tgt=kwd-340246634088&hsa_acc=95106. [Senest hentet eller vist den 26 11 2021].

- [12] »CRON,« [Online]. Available: <https://en.wikipedia.org/wiki/Cron>. [Senest hentet eller vist den 26 11 2021].
- [13] »NGINX,« [Online]. Available: <https://www.nginx.com/resources/wiki/>. [Senest hentet eller vist den 26 11 2021].
- [14] »Gunicorn,« [Online]. Available: <https://en.wikipedia.org/wiki/Gunicorn>. [Senest hentet eller vist den 26 11 2021].
- [15] »Certbot,« [Online]. Available: <https://certbot.eff.org/pages/about>. [Senest hentet eller vist den 26 11 2021].
- [16] »Android studio,« [Online]. Available: https://en.wikipedia.org/wiki/Android_Studio. [Senest hentet eller vist den 26 11 2021].
- [17] »Kotlin,« [Online]. Available: [https://en.wikipedia.org/wiki/Kotlin_\(programming_language\)](https://en.wikipedia.org/wiki/Kotlin_(programming_language)). [Senest hentet eller vist den 26 11 2021].
- [18] »Java,« [Online]. Available: [https://da.wikipedia.org/wiki/Java_\(programmeringssprog\)](https://da.wikipedia.org/wiki/Java_(programmeringssprog)). [Senest hentet eller vist den 26 11 2021].
- [19] »Particle Argon,« [Online]. Available: <https://docs.particle.io/argon/>. [Senest hentet eller vist den 26 11 2021].
- [20] »Raspberry PI,« [Online]. Available: https://da.wikipedia.org/wiki/Raspberry_Pi. [Senest hentet eller vist den 26 11 2021].
- [21] »Bluetooth,« Bluetooth SIG, [Online]. Available: <https://www.bluetooth.com/>. [Senest hentet eller vist den 26 11 2021].
- [22] »Solenoid lås,« [Online]. Available: <https://elektronik-lavpris.dk/p140845/jf-s0837dl-elektrisk-doerlaas-12vdc-1a/>. [Senest hentet eller vist den 26 11 2021].
- [23] »Magnetlås,« [Online]. Available: https://lavpris-laase.dk/magnetlaase-592/el-582-magnetlaas-dorn-50-mm-p430?gclid=CjwKCAjw7fuJBhBdEiwA2ILMYQvjMez_pVTFQ1f3NIcenmor9iz

O_8-Lou4GCogFPGCXHZ4pCe2jkxoCg-UQAvD_BwE. [Senest hentet eller vist den 26 11 2021].

- [24] »GIt,« [Online]. Available: <https://git-scm.com/>. [Senest hentet eller vist den 11 30 2021].
- [25] »GitHub,« 18 11 2021. [Online]. Available: <https://github.com/>.
- [26] »Django rest framework,« [Online]. Available: <https://pypi.org/project/djangorestframework/>. [Senest hentet eller vist den 29 11 2021].
- [27] »Djoser,« [Online]. Available: <https://pypi.org/project/djoser/>. [Senest hentet eller vist den 29 11 2021].
- [28] »SHA-2,« [Online]. Available: <https://en.wikipedia.org/wiki/SHA-2>. [Senest hentet eller vist den 10 12 2021].
- [29] »salting,« [Online]. Available: [https://en.wikipedia.org/wiki/Salt_\(cryptography\)](https://en.wikipedia.org/wiki/Salt_(cryptography)). [Senest hentet eller vist den 10 12 2021].
- [30] »GoDaddy,« [Online]. Available: <https://dk.godaddy.com/>. [Senest hentet eller vist den 30 11 2021].
- [31] »PyPI,« [Online]. Available: <https://pypi.org/>. [Senest hentet eller vist den 30 11 2021].
- [32] »wsgi,« [Online]. Available: https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface. [Senest hentet eller vist den 30 11 2021].
- [33] »Python random funk.,« [Online]. Available: <https://docs.python.org/3/library/random.html>. [Senest hentet eller vist den 30 11 2021].
- [34] »Gradle,« [Online]. Available: https://docs.gradle.org/current/userguide/what_is_gradle.html. [Senest hentet eller vist den 01 12 2021].
- [35] »Retrofit,« [Online]. Available: <https://square.github.io/retrofit/>. [Senest hentet eller vist den 01 12 2021].

- [36] »GSON,« [Online]. Available: <https://github.com/square/retrofit/tree/master/retrofit-converters/gson>. [Senest hentet eller vist den 01 12 2021].
- [37] »Coroutines,« [Online]. Available: https://developer.android.com/kotlin/coroutines?gclid=Cj0KCQiA-qGNBhD3ARIsAO_o7ymodwXYtDjT0X61o5hAAt6VcHuX5ns_yMeMJVUspscGNyp0FMNuKBwaAjWdEALw_wcB&gclsrc=aw.ds. [Senest hentet eller vist den 02 12 2021].
- [38] »coroutine suspend,« [Online]. Available: <https://www.raywenderlich.com/1423941-kotlin-coroutines-tutorial-for-android-getting-started>. [Senest hentet eller vist den 02 12 2021].
- [39] »JVM,« [Online]. Available: <https://www.javatpoint.com/memory-management-in-java>. [Senest hentet eller vist den 02 12 2021].
- [40] »Selenium,« [Online]. Available: <https://www.selenium.dev/>. [Senest hentet eller vist den 04 12 2021].
- [41] »Assa Abloy,« [Online]. Available: <https://www.assaabloyopeningsolutions.dk/da/local/dk/>. [Senest hentet eller vist den 10 12 2021].
- [42] »MITM (WIKIPEDIA),« [Online]. Available: https://en.wikipedia.org/wiki/Man-in-the-middle_attack. [Senest hentet eller vist den 07 12 2021].
- [43] »Abus,« [Online]. Available: <https://www.abus.com/us/Home-Security>. [Senest hentet eller vist den 10 12 2021].