

H6 Svende forløb 2023 - gruppe 2

E-Scrap Solutions

Produktrapport



Skrevet af Casper Andersen og Kevin Frost



**DataTekniker med
speciale i programmering**
<https://www.zbc.dk/>
ZBC Ringsted

Title: E-Scrap Solutions

Tema: Elektronikaffald

Rapport type: Procesrapport

Projekt start: 11. apr. 2023

Antal gruppemedlemmer: 2

Vurdering. Fælles bedømmelse

Medlemmer:

Casper Andersen

Kevin Frost

Github Profiler:

Casper Andersen (@[Casp7654](#), @[Ndersorn](#))

Kevin Frost (@[Keviner123](#))

Sider: 37

Projekt afslutning: 8. maj 2023

Indholdsfortegnelse

| | |
|---|-----------|
| 1 Kravspecifikation (FURPS+) | 4 |
| 1.1 Introduktion | 5 |
| 1.1.1 Formål og Scope | 5 |
| 1.1.2 Definitioner, akronymer og forkortelser | 5 |
| 1.1.3 Reference | 6 |
| 1.1.4 Overblik | 7 |
| 1.2 Funktionalitet | 8 |
| 1.3 Usability | 15 |
| 1.3.1 Oversigt | 15 |
| 1.3.2 Effektivitet | 15 |
| 1.3.3 Brugervenlighed | 15 |
| 1.3.4 Dokumentation | 15 |
| 1.4 Reliability | 16 |
| 1.4.1 Tilgængelighed | 16 |
| 1.4.2 Server oppe tid | 16 |
| 1.4.3 Mekaniske dele | 17 |
| 1.5 Performance | 18 |
| 1.5.1 Svartid | 18 |
| 1.5.2 Throughput | 18 |
| 1.5.3 Opstarts- og nedlukning | 18 |
| 1.6 Supportability | 20 |
| 1.6.1 Kode dokumentation | 20 |
| 1.6.2 Nødvendig dokumentation | 20 |
| 1.6.3 Kodesprog og standarder | 20 |
| 1.7 Design constraints | 21 |
| 1.7.1 Grænsefladesprog og kildekode | 21 |
| 1.7.2 Bootstrap | 21 |
| 1.7.3 AdminLTE | 21 |
| 1.8 Interfaces | 22 |
| 1.8.1 Hardware interfaces | 22 |
| 1.8.2 Software interfaces | 22 |
| 1.8.3 Kommunikationsinterfaces | 22 |
| 1.9 Brugervejledning | 23 |
| 1.10 Teknisk produktdokumentation | 24 |
| 1.10.1 Deployment diagram | 24 |
| 1.10.2 Mockup design | 25 |

| | |
|---|-----------|
| 1.10.3 Use case diagram | 26 |
| 1.10.4 Flowcharts | 27 |
| 1.10.4.1 Api: Application | 27 |
| 1.10.4.2 Api: onEvent | 28 |
| 1.10.5 Entity Relationship Diagram | 29 |
| 1.10.6 Klassediagrammer | 30 |
| 1.10.6.1 Api: Package Overview | 30 |
| 1.10.6.2 API: Controllers | 31 |
| 1.10.6.3 API: Entities | 32 |
| 1.10.7 Component Creation | 33 |
| 1.11 System sekvens diagram | 33 |
| Figur 11 - System sekvens diagrammet for sorteringssystemet | 33 |
| 1.11.1 Logisk topologi | 34 |
| Figur 12 - Logisk topologi over vores sorteringssystem | 34 |
| 1.11.2 Netværksoversigt | 34 |
| 1.11.3 Enheder | 35 |
| 1.12 Testrapport | 36 |
| 1.12.1 Unit Tests | 36 |
| 1.12.1.1 Sorteringssystem tests | 36 |
| 1.12.1.2 Registreringssystem tests | 36 |
| 1.12.2 User Acceptance Tests | 37 |
| 1.13 Bilag | 39 |

1 Kravspecifikation (FURPS+)

1.1 Introduktion

Dette dokument beskriver alle kendte / forudbestemte funktionelle og ikke funktionelle krav, som er implementeret ved brug af kravspecifikations standard modellen FURPS+.

1.1.1 Formål og Scope

Formålet med dette dokument er at definere de krav der er blevet sat før og under udarbejdelse af produktet, samt information omkring test og vedligeholdelse.

Derudover indeholder dette dokument alt nødvendig information omkring både sorterings- og registreringssystemet samlet.

1.1.2 Definitioner, akronymer og forkortelser

Sorteringssystem: De samlede enheder der står for at differentiere og sortere elektroniske komponenter fysisk.

Registreringssystem: De samlede applikationer der står for at kunne registrere og håndtere elektroniske komponenter logisk.

YOLO/YOLO V8: Kunstig intelligens-bibliotek der bruger en forud trænet model til objektgenkendelse.

Control Center: Fysisk og logisk enhed der er mellem punkt / kontrol enhed for sorteringsssystemet.

Rest Api: Server applikations interface til datahåndtering for elektroniske komponenter

Web interface: Web applikations til visuel og statistisk repræsentation af elektroniske komponenter

ORM / TypeORM: Object relations manager bibliotek der bruges til at forbinde Rest Api og Database.

MariaDB: Database applikation

CET: Central European Time Zone

GPIO: General purpose Input Output

SBC: En enkelt processorplade, der allerede inkluderer alle de nødvendige komponenter til at fungere i produktion.

1.1.3 Reference

| Navn | Dato | Organization | Link |
|--------------------------|------------|------------------------------|---|
| Astro | 24/01/2023 | The Astro Technology Company | https://docs.astro.build/en/getting-started/ |
| AdminLTE | 02/07/2023 | ColorlibHQ | https://adminlte.io/docs/3.2/index.html |
| Chart.js | 10/02/2023 | chartjs | https://www.chartjs.org/docs/latest/ |
| Express | 29/04/2023 | OpenJS Foundation | http://expressjs.com/en/4x/api.html |
| TypeORM | 15/04/2023 | TypeORM | https://typeorm.io/ |
| Typescript code standard | 01/10/2012 | Microsoft | https://www.typescriptlang.org/docs/handbook |
| Python3 PyLint standard | 24/04/2023 | Python Software Organisation | https://pypi.org/project/pylint/ |
| YOLOv8 | 01/05/2023 | Ultralytics | YoloV8 |

1.1.4 Overblik

Der angives i dette dokument først de funktionelle krav til system i use-case format som brief udgave, der skal beskrive enkelte del-sekvenser systemet håndterer.

Derefter vil usability kravene blive beskrevet i tekstformat, som beskriver brugerens synspunkt på hvordan det er muligt at bruge systemet.

Dernæst vil der være et reliability afsnit, som skal give et overblik over hvad den forventede fejlrate samt angive service information og et performance afsnit, som vil beskrive hvad der kan forventes af ydeevne ud af det systemerne.

Derefter vil der være et supportability afsnit der skal give information omkring påkrævet dokumentation der skal vedligeholdes, samt kodesprog og standarder der skal overholdes.

Det næste afsnit vil indeholde design constraints, hvilket angiver hvilke ikke-funktionelle krav der måtte være til systemet, for at opretholde en strømlinet brugeroplevelse, samt forhold til opbevaring af data og sikkerhed i systemet.

Dernæst vil der være et afsnit omkring Interfaces, som skal give et overblik over hvilke hardware og software interfaces der er blevet brugt i produkt udarbejdelsen.

Til at afrunde dokumentet vil der være beskrevet en teknisk produktdokumentation og en testrapport. Den tekniske produktdokumentation der vil indeholde alle brugte klasse-, deployment-, entity relation- og andre diagrammer med forklaring dertil, samt topologi for proof-of-concept (evt. prototype) som skal vedligeholdes ved ændring.

Testrapporten vil vise og forklare de User Acceptance Tests og Unit Test der er lavet, som også skal vedligeholdes og dertil tilføje flere ved viderebyggelse af projektet.

1.2 Funktionalitet

Denne sektion beskriver alle de aktuelle funktionelle krav til systemet i use-case format, som angiver hvilke handlinger, som systemet skal udføre for at opfylde brugerens behov. Dette format har vi valg fordi use-case er et let forståeligt og standardiseret format, der giver en klar og præcis forståelse af kravet.

| | |
|---------------------------|---|
| Use-case | Differentiering mellem forskellige komponenttyper. |
| Id | 1 |
| Version | 1 |
| Aktør(er) | Sorteringssystemets Control Center |
| Trigger | Komponent bliver detekteret foran lysføleren |
| Pre-condition | Sorteringssystemer er aktivt og komponent ligger på transportbåndet, og vores machine learning model er trænet med komponentet |
| Post-condition | Komponentet er blevet genkendt, og kan køre videre på båndet. |
| Beskrivelse | Ved hjælp af vores machine learning model kan vi identificere et elektrisk komponent |
| Normalt forløb | <ol style="list-style-type: none">1. Komponent stopper foran sensoren2. Komponent bliver genkendt af vores model3. Komponentet kører videre på båndet |
| Alternativt forløb | Komponentet kører videre uden at være genkendt |

| | |
|---------------------------|--|
| Use-case | Sortering af komponenter per type. |
| Id | 2 |
| Version | 1 |
| Aktør(er) | Sorteringssystemets Control Center |
| Trigger | Komponent er blevet detekteret af vores machine learning |
| Pre-condition | Komponent er blevet genkendt, og komponentet har en placering i Web API |
| Post-condition | Komponent får tildelt placerings kategori og bliver lagt i den rigtige kasse. |
| Beskrivelse | Systemet sender en forespørgsel til API'en for at få information om, hvordan komponenten skal sorteres. API'en giver svar på, om komponenten skal genbruges, genanvendes eller sælges til andre formål. |
| Normalt forløb | <ol style="list-style-type: none"> 1. Systemet sender en anmodning til web API'et, hvor den spørger om hvilken kasse komponentet hører til 2. Kassen der skal opfange komponenterne bliver kørt over foran transportbåndet 3. Transportbåndet kører komponentet udover ned i kassen |
| Alternativt forløb | Komponentet bliver ikke genkendt, og vil derfor blive kørt over i kassen "ukendt" |

| | |
|---------------------------|---|
| Use-case | Afvejning enkelte komponenters faktiske vægt. |
| Id | 3 |
| Version | 1 |
| Aktør(er) | Sorteringssystemets Control Center |
| Trigger | Et komponent er landet i en sorteringskasse fra transportbåndet |
| Pre-condition | Et komponent er blevet sorteret og genkendt af vores system |
| Post-condition | Komponentens vægt kan lægges ind i systemet |
| Beskrivelse | <p>Vi vejer de forskellige komponenter, der primært består af metal, såsom kobber i en induktor, for at bestemme deres værdi.</p> <p>Vi gør dette ved at veje komponenterne når de lander i vores sorteringskasser</p> |
| Normalt forløb | <ol style="list-style-type: none"> 1. Komponent er detekteret af vores objektgenkendelse 2. Vores vejecelle nulstiller, for at gøre sig klar til den nye vægt. 3. Komponent falder ned i kassen og vejes |
| Alternativt forløb | Komponent vejer enten for meget, eller for lidt til at blive genkendt af vores vejecelle |

| | |
|-----------------------|--|
| Use-case | Registrering af enkelte komponenter. |
| Id | 4 |
| Version | 1 |
| Aktør(er) | Sorteringssystemets Control Center og Registreringssystemets Rest Api |
| Trigger | Sorteringssystemet har et vejet og genkendt komponent |
| Pre-condition | Et komponent er blevet genkendt af vores objektgenkendelse, og har fået tildelt en vægt. |
| Post-condition | Komponentet bliver oprettet i Rest API'et |
| Beskrivelse | For at kunne danne et overblik over vores registrerede komponenter, tilføj vi alle de komponenter vi kender med deres vægt til vores website. |
| Normalt forløb | <ol style="list-style-type: none">1. Komponentet er sorteret korrekt, og vi har vægten.2. Komponentet bliver oprettet via vores API |

| | |
|-----------------------------|---|
| Use-case | Tilgang til den enkelte komponents information. |
| Id | 5 |
| Version | 1 |
| Aktør(er) | Registreringssystemets Rest Api |
| Trigger | En forespørgsel på et komponent, dets type, model, placerings kategori eller anden information. |
| Pre-condition | Registreringssystemet har registreret mindst et komponents information |
| Post-condition | Forespørgeren har fået den forespurgt information |
| Beskrivelse | For at kunne bruge den registrerede information har vi brug for et interface til at kunne tilspørger efter nødvendig information |
| Normalt forløb | <ol style="list-style-type: none"> 1. Forespørgelsen bliver modtaget og routet til en given api rute. 2. Forespørgsels parametrene bliver tjekket for invaliditet. 3. Rest Api'et tilspørger databasen efter nødvendig data. 4. Rest Api'et giver forespørgeren et formateret svar tilbage med data og en type af hvilken datastruktur der er i svaret. |
| Alternativt forløb A | <ol style="list-style-type: none"> 1. Forespørgslen bliver modtaget men bliver afvist pga. den givne rute findes ikke i Rest Api'et. |
| Alternativt forløb B | <ol style="list-style-type: none"> 1. Forespørgelsen bliver modtaget og routet til en given api route. 2. Forespørgsels parametrene bliver tjekket for invaliditet. 3. Forespørgslen bliver afvist pga. invaliditet eller mangel på forespurgt information |

| | |
|-----------------------------|--|
| Use-case | Ændring af et komponents registreret værdier. |
| Id | 6 |
| Version | 1 |
| Aktør(er) | Registreringssystemets Rest Api |
| Trigger | En forespørgsel på at ændre information omkring et komponent, dets type, model, placeringskategori eller anden. |
| Pre-condition | Registreringssystemet har registreret mindst et komponents information |
| Post-condition | Forespørgeren får svar om ændringen af informationen kunne forekomme. |
| Beskrivelse | Det skal være muligt at kunne ændre den registrerede information omkring et komponent. |
| Normalt forløb | <ol style="list-style-type: none"> 1. Forespørgelsen bliver modtaget og routet til en given api rute. 2. Forespørgsels parametrene bliver tjekket for invaliditet. 3. Rest Api'et tilspørger databasen efter nødvendig data objekt. 4. Rest Api'et tilretter data objekt informationen. 5. Rest Api'et beder databasen om at opdateres. 6. Forespørgeren får et OK svar tilbage. |
| Alternativt forløb A | <ol style="list-style-type: none"> 7. Forespørgelsen bliver modtaget og routet til en given api rute. 8. Forespørgsels parametrene bliver tjekket for invaliditet. 9. Rest Api'et tilspørger databasen efter nødvendig data objekt. 10. Databasen har ikke det refererede data objekt. 11. Forespørgeren får et fejl svar tilbage. |

| | |
|---------------------------|--|
| Use-case | Liste oversigtsvisning over komponenter pr. tildelt placerings kategori (alle, genbrugbare, genanvendelige eller ”sælges til andet brug”). |
| Id | 7 |
| Version | 1 |
| Aktør(er) | Registreringssystemets Web interface, Rest Api og Database |
| Trigger | Web interfacet forespørger Rest Api'et om at få relevant data. |
| Pre-condition | Rest Api'et er tilgængeligt. |
| Post-condition | Web interfacet renderer en visuel præsentation af given data. |
| Beskrivelse | For at kunne vise hvilke komponenter der er i givne placerings kategorier, vil vi have en mulighed for at brugerne visuelt kan se en repræsentation af dataen fra database. |
| Normalt forløb | <ol style="list-style-type: none"> 1. Web interfacet forespørger efter relevant komponent data. 2. Rest Api'et modtager forespørgslen. 3. Rest Api'et forespørger databasen efter relevante data objekter. 4. Rest Api'et formaterer dataen i det efterspurgtrepræsentative objekt. 5. Rest Api'et sender repræsentative objekter tilbage til Webinterfacet. 6. WebInterfacet renderer efterspurgt data i en visuel præsentation |
| Alternativt forløb | <ol style="list-style-type: none"> 1. Web interfacet forespørger efter relevant komponent data. 2. Rest Api'et modtager forespørgslen. 3. Rest Api'et forespørger databasen efter relevante data objekter. 4. Rest Api'et fejler formateringen af det repræsentative objekt, ved mangel på dataen 5. Rest Api'et sender fejl svar tilbage med relevant fejlmeddeelse. 6. Webinterfacet logger fejlmeddeelse. |

1.3 Usability

Dette afsnit af dokumentet skal give et kort overblik fra brugerens synspunkt samt beskrive mulighederne for brugbarheden af systemet.

1.3.1 Oversigt

90% af alle brugerne skal kunne åbne og forstå backend interfaces oversigt inden for 5 minutter.

1.3.2 Effektivitet

En bruger af systemet skal kunne udføre alle opgaver i backend interfacet indenfor 10 minutter.

1.3.3 Brugervenlighed

95% af alle brugere af systemet skal føle systemet intuitivt, kunne navigere og kunne se hvilke muligheder de har i backend interfacet inden for 1 minut.

1.3.4 Dokumentation

95% af al funktionalitet skal være mulig at slå op for alle brugere inden for 5 minutter.

1.4 Reliability

I dette afsnit vil der være angivet service information samt kendt information i forhold til fejrlate og oppetider for systemerne.

1.4.1 Tilgængelighed

Systemet skal have en generel oppetid på 98% for både WebInterface og mekaniske enheder. Brugere af systemet skal gøres opmærksomme på at der er et Servicevindue hver Onsdag og Fredag mellem klokken 6:00-10:00 CET hver morgen.

1.4.2 Server oppetid

Grundet produktets nuværende proof-of-concept tilstand, har vi ikke mulighed for at måle oppetid på hverken Rest Api'et eller mulige webshops, og har derfor valgt ikke at gisne om dette også. Dertil skal det forventes at vi kan opretholde en minimum på 90% oppetid over et års forløb, medtaget at der er aflagt et servicevindue til dette i samme tidsrum som forventet ved de andre delsystemer.

1.4.3 Mekaniske dele

Eftersom projektet involverer mekaniske enheder, er det vigtigt, at vi gør dem let udskiftelige. Heldigvis er dette en let opgave, da vi har valgt at bruger standard dele.

Mekanisk kontakt

Vores endestop i 3D-printeren er en KLS7-KW10, som er designet til at kunne tåle op til 1.000.000 tryk¹. Dette er en imponerende holdbarhed, især når man tager i betragtning, at vores printer kun udløser endestoppet én gang, når systemet genstarter.

Kontakterne er nemme at skifte ved at løsne 2 skruer og flytte kablet over i den nye

Strømforsyning

Strømforsyningen, som vi anvender til vores projekt, er en PRO-MCP M SNT 120W 24V 5A, som har en MTBF på over 500.000 timer². Dette er en imponerende holdbarhed og betyder, at strømforsyningen kan forventes at fungere fejlfrit i meget lang tid.

Siden vi bruger skrueterminaler, ville et komplet udskrift af strømforsyning ikke tage mere end et par minutter

Relæ

Vi anvender et SRD-05VDC-SL-C relæ til at kontrollere vores transportbånd, og dette relæ forventes at have en levetid på 10.000.000 operationer³. Da relæet slukkes og tændes for hver enkelt ting, der bevæger sig langs båndet, bliver det brugt meget hyppigt.

Vi vil hurtigt opdage eventuelle fejl, da båndet enten vil stoppe med at køre eller fortsætte med at køre på en måde, der signalerer en fejl.

¹ https://cdn.ozdisan.com/ETicaret_Dosya/528155_148230.pdf

² <https://media.automation24.com/datasheet/us/400778-datasheet.pdf>

³ <https://www.circuitbasics.com/wp-content/uploads/2015/11/SRD-05VDC-SL-C-Datasheet.pdf>

Stepper motor

Vi kunne ikke finde en estimeret levetid på præcis vores stepper motor, derfor går vi ud fra dette *"The typical lifetime of a stepper motor is 10,000 operating hours. This approximates to 4.8 years given that the motor operates one eight-hour shift per day. Motor lifetime may vary in regards to user application and how rigorously the motor is run.⁴"*

Det er nemt at fjerne stepper motoren, da den er fastgjort med 4 skruer, og den kan let udskiftes med en ny.

1.5 Performance

I dette afsnit vil det blive beskrevet den forventet svartid, hastighed og køretid af det komplette system

1.5.1 Svartid

Ethvert interface der benyttes i systemet skal have en gennemsnitlig svartid på enten under eller omkring 2 sekunder.

1.5.2 Throughput

Registreringssystemet skal kunne håndtere et minimum af 1 ny enhed per sekund.

Sorteringssystemet skal dertil også kunne håndtere et minimum af 2 nye enheder per sekund.

1.5.3 Opstarts- og nedlukning

Alle systemer skal kunne starte op på 2 minutter og være fuldt funktionelt, dertil skal der også være adgang til Webinterfacet for brugere.

⁴ <https://www.anaheimautomation.com/manuals/forms/stepper-motor-guide.php>

1.6 Supportability

I dette afsnit vil der blive beskrevet hvilke former for dokumentation der til hver en tid skal vedligeholdes samt hvilke kodestandarder der er og skal anvendes ved videreudvikling af systemet.

1.6.1 Kode dokumentation

Der skal udarbejdes kode dokumentation med indeholdt kode kommentarer for alle systemer og projektets leverancer.

1.6.2 Nødvendig dokumentation

Der er i dokumentet her indsat klassediagrammer, ER-diagrammer, flowcharts, SD-diagrammer og deployment diagram, der skal vedligeholdes og genindsættes ved ændring.

Der skal til hver en tid opretholdes tidsplaner og backlogs for alle nye tiltag for projektet.

Der skal angives en risikoanalyse i samarbejde med projektejerne.

1.6.3 Kodesprog og standarder

Programmeringssprog og teknologier vil blive valg efter behov og fulgte standarder for relaterede skal refereres til med hyperlinks. Links kan findes i afsnittet [1.1.3 Reference](#).

- Microsoft Typescript Standard
- Python 3 Pylint Standard

1.7 Design constraints

Dette afsnit består af de ikke-funktionelle krav til systemet, samt hvilke designskabeloner der er brugt og skal bruges ved fremtidig videreudvikling.

1.7.1 Grænsefladesprog og kildekode

Systemets grænseflade vil blive leveret på dansk, hvor kildekoden, samt kommentarer der i, skal og vil blive skrevet på engelsk. Dertil er al kommunikation mellem vores delsystemer på engelsk.

1.7.2 Bootstrap

I forhold til design af mulige webshops, har vi valgt at arbejde ud fra eventuelle design skabeloner baseret på Bootstrap⁵, da dette vil være med til at holde en forholdsvis rød tråd gennem alle vores brugergrænseflader.

1.7.3 AdminLTE

Design af backend dashboard/webinterfacet er udarbejdet efter AdminLTE skabelonen som er bygget ud fra Bootstrap CSS & Javascript biblioteket, og som også har design implementationer af ChartJS, som vi bruger til visuel repræsentationer af vores data.

⁵ <https://getbootstrap.com/>

1.8 Interfaces

Dette afsnit skal give et kort overblik over hvilke interfaces der er brugt til udarbejdelse af produktet, samt hvilke der er planlagt til brug i fremtiden.

1.8.1 Hardware interfaces

Alle mekaniske dele af registreringssystemet skal kunne afvikles på ethvert SBC (Single board computer) som har GPIO-pins, lignende Raspberry Pi eller Rock Pi enheder.

1.8.2 Software interfaces

Docker bruges til udviklingsmiljø for at holde sourcekoden konsekvent og understøtte ethvert system. JWT ønskes at blive brugt til autentificering af brugere i fremtiden.

1.8.3 Kommunikationsinterfaces

All kommunikation mellem systemer skal foregå gennem HTTPS protokollen, og skal være sikret med en brugers Token.

1.9 Brugervejledning

Størstedelen af vores system er automatiseret, men der er stadig nogle processer, der kræver opsætning for at sikre en glidende drift. For at få den bedste oplevelse af systemet, foreslår vi at man først ser [intro videoen](#).

Step 1:

Tænd for strømmen på strømforsyningen



Step 2:

Hvis båndet ikke starter af sig selv, kan du vente i 2 minutter. Hvis dette ikke løser problemet, kan det være en indikation på en fejl i ledningsnettet, og det vil kræve assistance fra en tekniker for at undersøge og løse problemet.



Step 3:

Når komponenten når ud foran den forreste fotosensor, skal det automatisk stoppe, og den korrekte sorteringsboks skal nu flyttes til det rigtige rum.



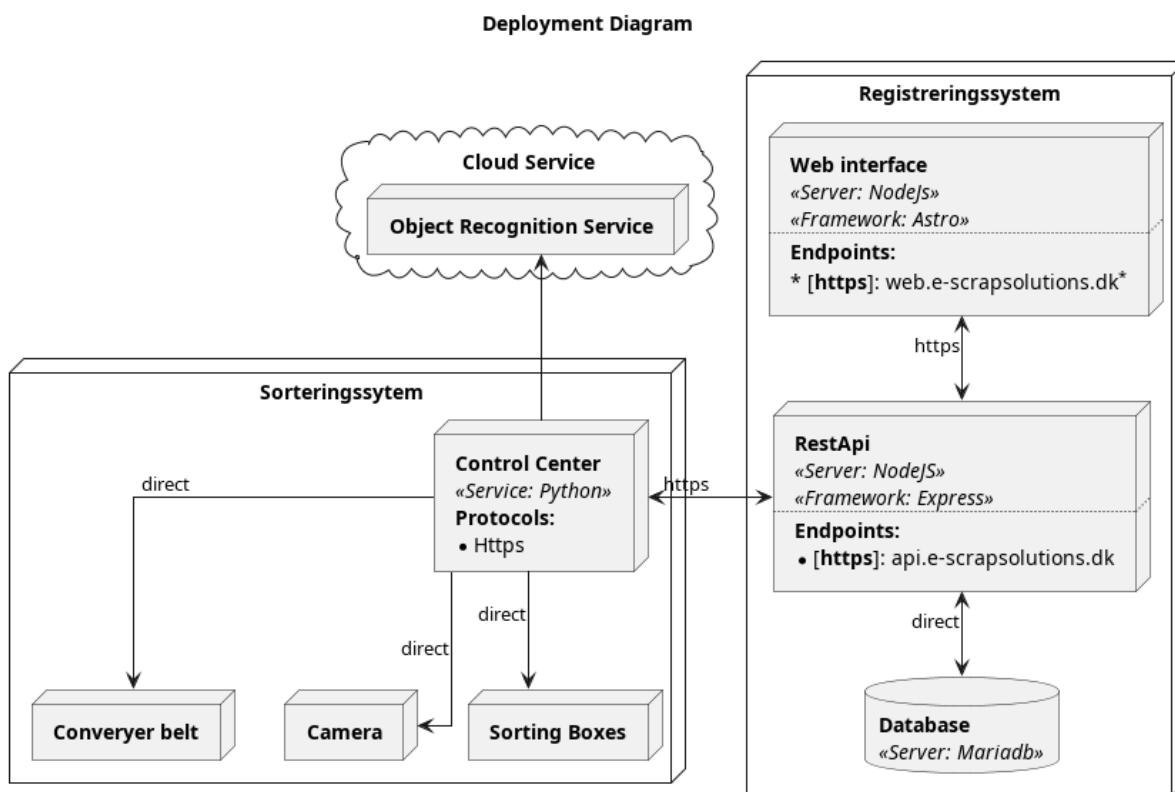
Step 4:

Nu vil komponentet blive placeret i sorteringskassen og registreret i systemet. Dette kan verificeres gennem webgrænsefladen

1.10 Teknisk produktdokumentation

Selve produktet er udarbejdet som værende et tredelt system. Et sorteringsanlæg, et komponent registreringssystem og en flerdels webshop til genbrugs- og materiale salg. Herunder vil der være information med passende diagrammer for at illustrere opsætningen og tanker omkring system designet.

1.10.1 Deployment diagram

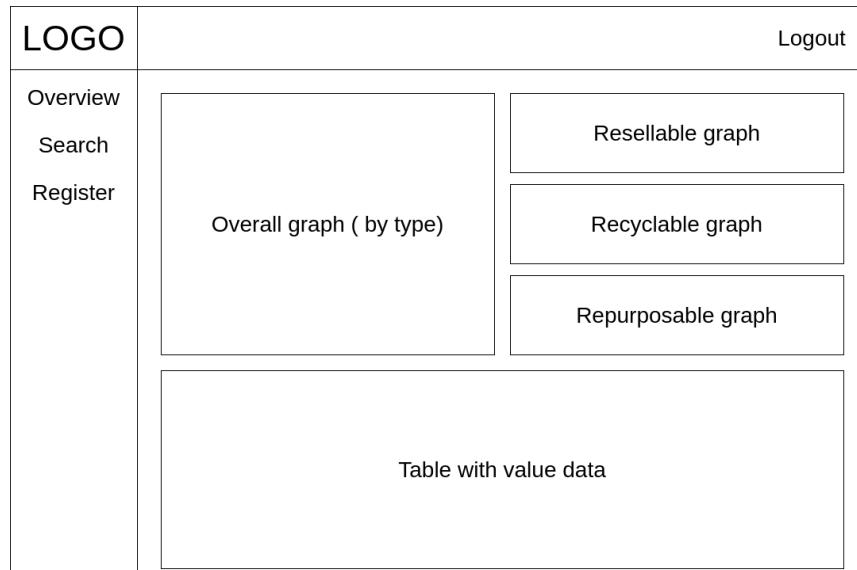


Figur 1 - Deployment diagram

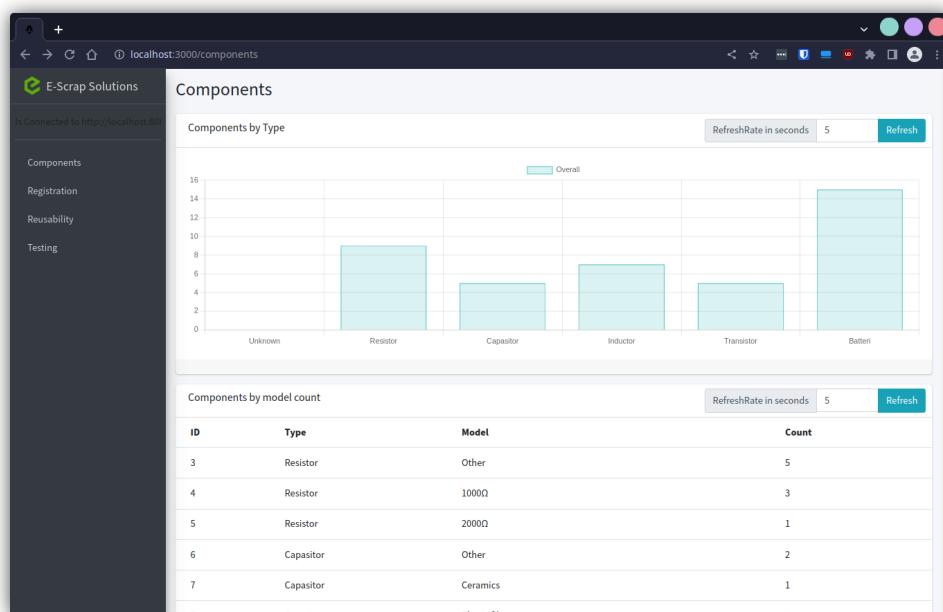
Først og fremmest vil vi præsentere vores deployment i vores større komplette system. Som det kan ses på *Figur 1*, så har vi valgt at udnytte at vi i gruppen har erfaring med flere forskellige teknologier, og med dem kan opsætte et bedre overblik over det vi gerne vil opnå. Vi har valgt at udnytte en Cloud Service til brug som Object Recognition Service, da dette kan være hårdt for ikke optimeret hardware. Dertil skal det siges, at man i forhold til et produktionsmiljø nok ville opsætte en lokal server specifikt til dette formål, da fjern forbindelsen godt kan give større risiko for fejl eller svartid.

1.10.2 Mockup design

Figur 2 Illustrerer de tanker vi havde til vores brugergrænseflade for web interfacet. Vi har holdt os godt til dette - som kan ses på Figur 3 -.På grund af projektets længde har vi ikke fået implementeret det hele endnu. Dog kunne vi sagtens have implementeret dette, men vi valgte at bruge tiden på at forfine vores sorteringssystem i stedet.



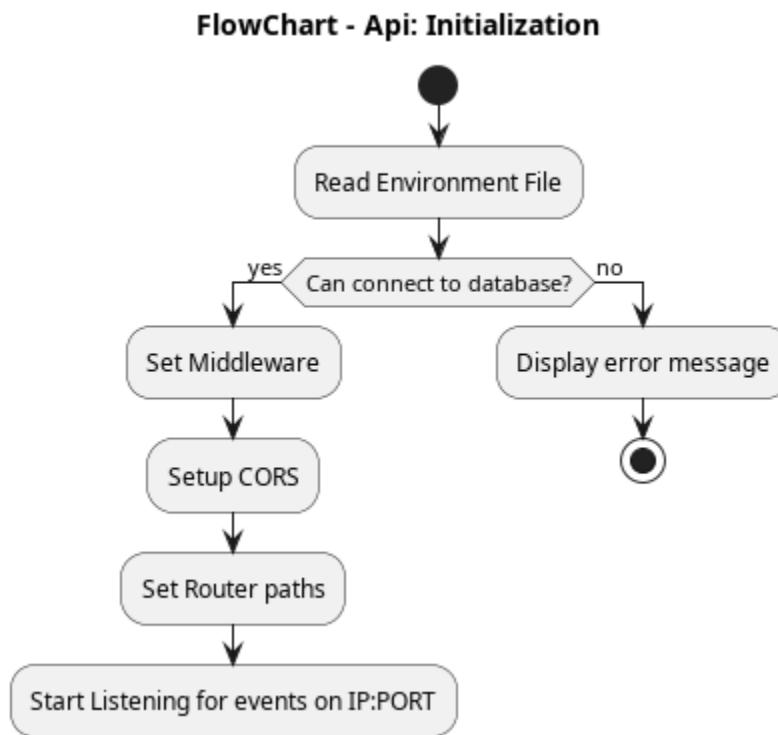
Figur 2 - Mockup of Web interface Design



Figur 3 -Web interface Design

1.10.4 Flowcharts

1.10.4.1 Api: Application



Figur 4 - Flowchart Api: Initialization

Som vist i *Figur 4* følger Rest Api'et, Express Js framework'ets normale *initialization*, ved at først at læse miljø variablerne, derefter sætte *CORS* og anden *middleware* håndtering af HTTP Request. Vi har så valgt at sætte et tilstandstjek ind efter læsningen af miljøvariablerne, da vi ikke mener at Rest Api'et bør starte hvis ikke der er adgang til database, da TypeORM kræver adgang fra *initializaion* for at kunne starte og holde synkroniseringen af data til databasen.

Vi har valgt at opdele forløbet ved "Start Listening for events on IP:PORT". Da vi her går fra at initialisere systemet. Det efterkommende forløb, som lytter på hændelser kan ses på *Figur 5*.

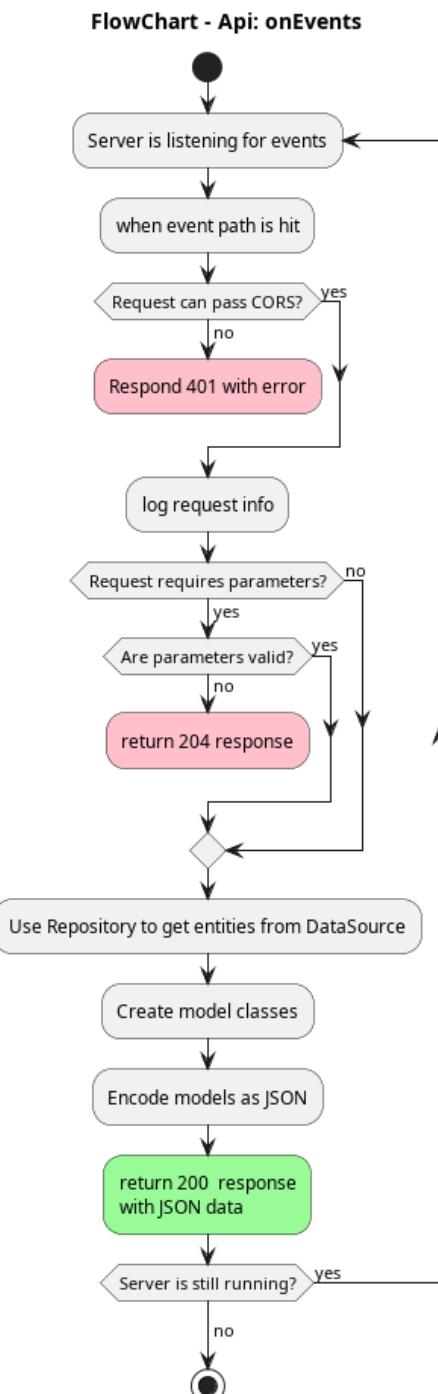
1.10.4.2 Api: onEvent

Figur 5 Illustrerer en normal håndtering af vores implementerede hændelsesforløb fra ExpressJS frameworkt. Dette forløb er ensartet hele vejen igennem vores system og dette er - efter ExpressJS egen dokumentation - den mest normale måde at håndterer hændelsesforløbet.

Forløbet er således, at vi ved *request* efter en specifik *route*, kan køre det, der kaldes for en *callback-funktion*. Denne funktionalitet giver os mulighed at opsætte kildekode strukturen komplekst og efter de design mønstre som er ønsket, og stadig opnå god svartid ved hændelsesforløbet.

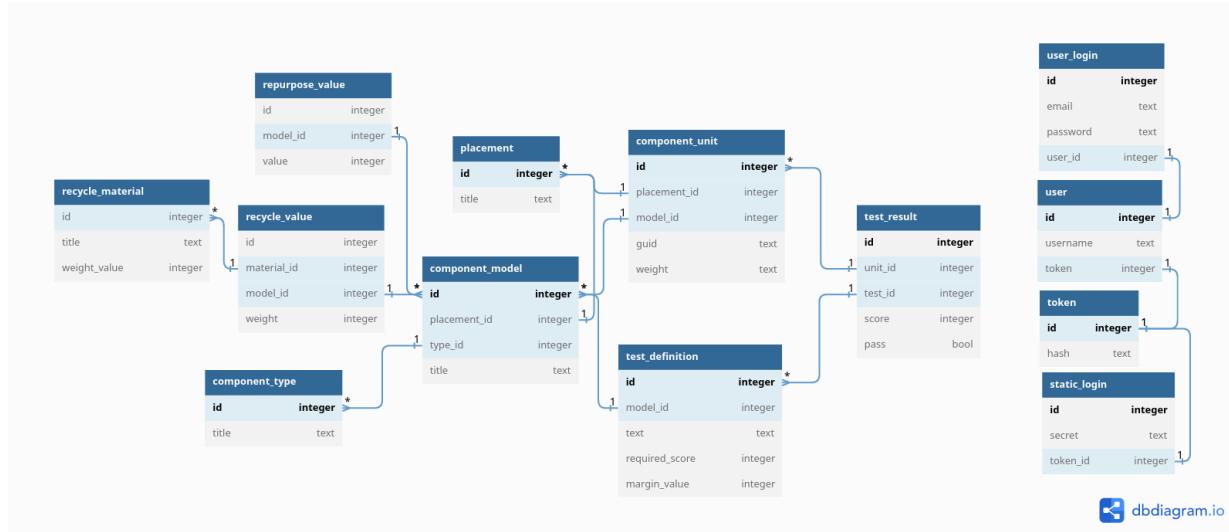
Det gør også at vi under hændelsesforløbet ikke skal genoprette klasser til at tilgå foreksempelvis vores *DataSource*, men vi kan i forløbet tilgå de instanser af denne og andre objekter/klasser vi har initialiseret under opstarts *initialization* hurtigere.

Selve hændelsesforløbet kører også uden for vores *main call stack*, hvilket gør at disse events udnytter NodeJs's tråd håndtering til at kunne håndtere flere hændelser på engang. Hvilket gør, at vi ikke kommer til at blokere vores main stack ved langsomme, eller store operationer.



Figur 5 - Flowchart Api: OnEvent

1.10.5 Entity Relationship Diagram



Figur 6 - Entity Relationship Diagram

I Figur 6 kan vi se hvordan vi har valgt at oprette / opretholde strukturen igennem vores system. Hvis vi først kigger på tabellerne *ComponentUnit*, *ComponentModel* og *ComponentType*, er disse det vi vil kalde for vores elektroniske komponent opdelt efter det der er selve den faktiske komponent enhed - *ComponentUnit* - som bliver registreret, hvilken type - *ComponentType* - enheden er af og hvilken model - *ComponentModel* - det er. Disse ekstra tabeller ville man kunne tillægge *ComponentUnit* men grundet vi har mange enheder med samme type eller model har vi valgt at opdele disse i deres egne tabeller.

For at snakke vores opsætning igennem de 3 faser der - efter registreringen i systemet - differentierer hvor den enkelte enhed skal ende hende i systemet, skal vi kigge på *Placement* tabellen. Denne tabel skal både virke som værende det opslag vores sorteringssystem skal lave for at finde ud af hvor en enhed af bestemt type skal hen, men også vise hvad den registrerede enheds nuværende placeringsskategori er.

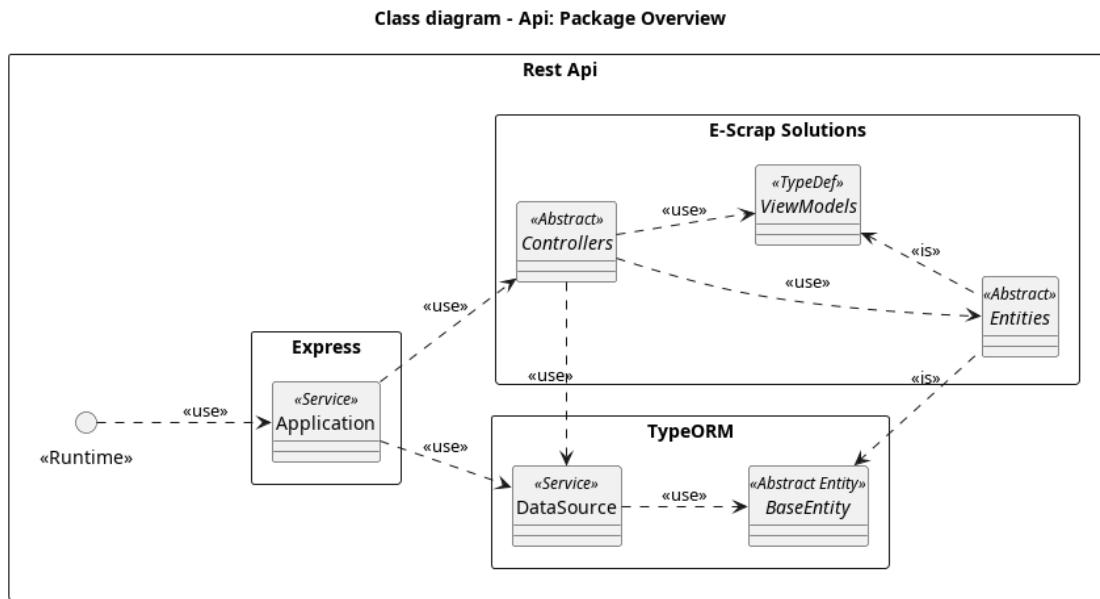
TestDefinition tabellen har kendskab til hvilken model enheden er af, da det skal beskrive hvilke fysiske / elektroniske tests der skal udføres på disse for at give et *TestResult* tilbage på den faktiske enhed.

RecycleValue indeholder hvilken værdi komponent modellen har i denne fase, samt med opslag på hvad for materialer - *RecycleMaterial* - komponentet indeholder, skal kunne give

den værdi - ud fra *ComponentUnit* tabellens felt *weight* - der er pr. enhed ved genanvendelse af denne. Det samme kan siges omkring tabellen *RepurposeValue*, dog uden materiale opslaget, da vi regner med at salget går på hele enheden i denne fase.

1.10.6 Klassediagrammer

1.10.6.1 Api: Package Overview



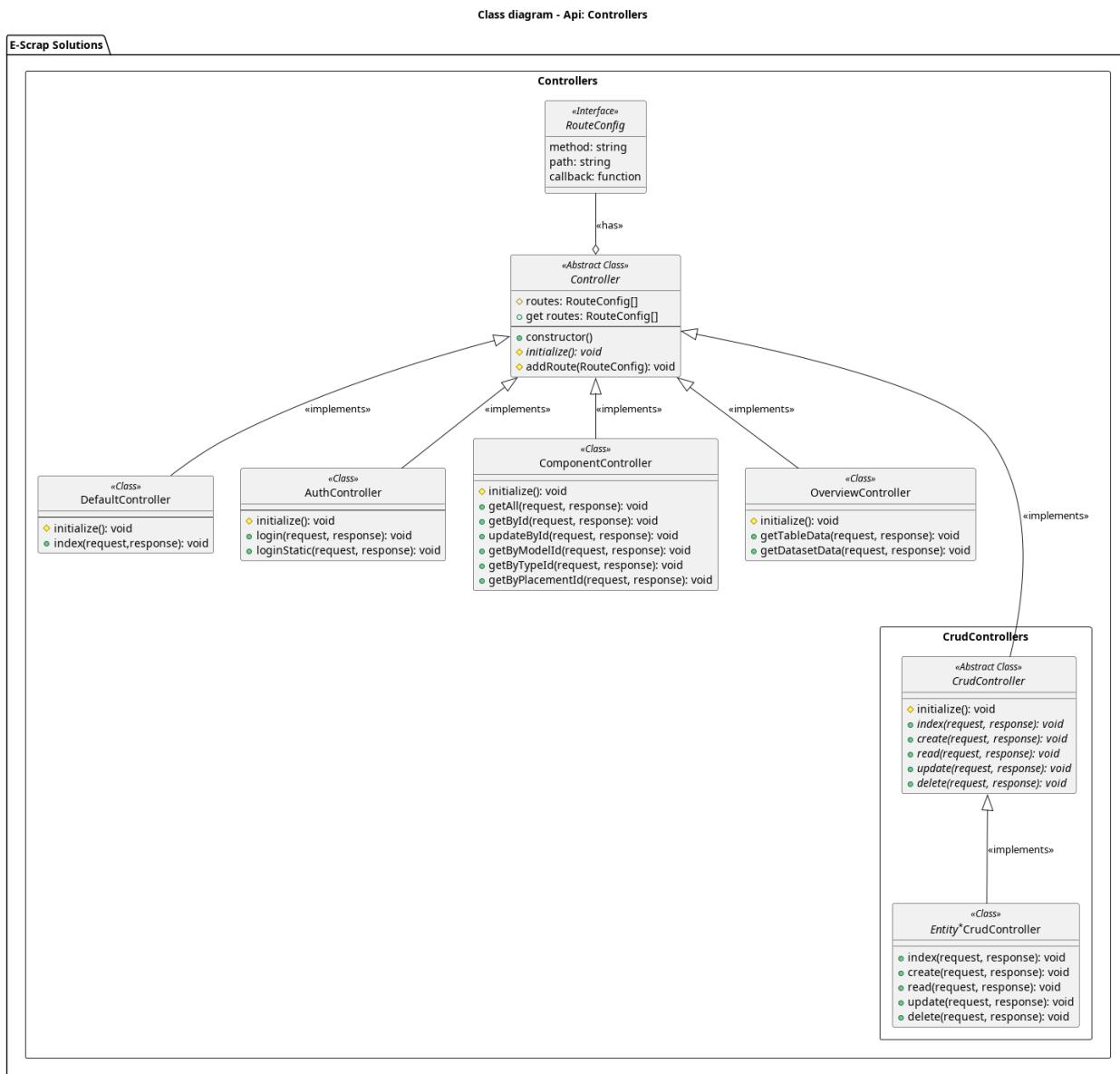
Figur 7 - klassediagram for Rest API'et: Package Overview

Som det kan ses på Figur 7, har vi delt vores klassediagram op, for at bedre kunne illustrere hvordan vores Rest API er struktureret - hvad der er under vores løsning og hvad vi implementerer af biblioteker. I dette billede udvider vi TypeORM bibliotekets *BaseEntity* klassen, for at udnytte TypeORM's *DataSource* klasses funktionalitet til at automatisk oprette og synkronisere Entity klasser med databasen.

Vi opretter også en *Abstrakt klasse* vi kalder for Controller hvis *routes property* liste Express *Application* klassen burger, ved hjælp af Dependency Injection, til at oprette hændelses kald til vores implementerede Controller klasser.

Vores Controllers kender til *DataSource* klassen fra Express Applikationens HTTP request parameter, der bliver tilsendt vores Controller klasser ved hændelses affyring, og opretter *ViewModel* objekter som sendes tilbage til forespørgeren ved Express Applikations HTTP response parameteren.

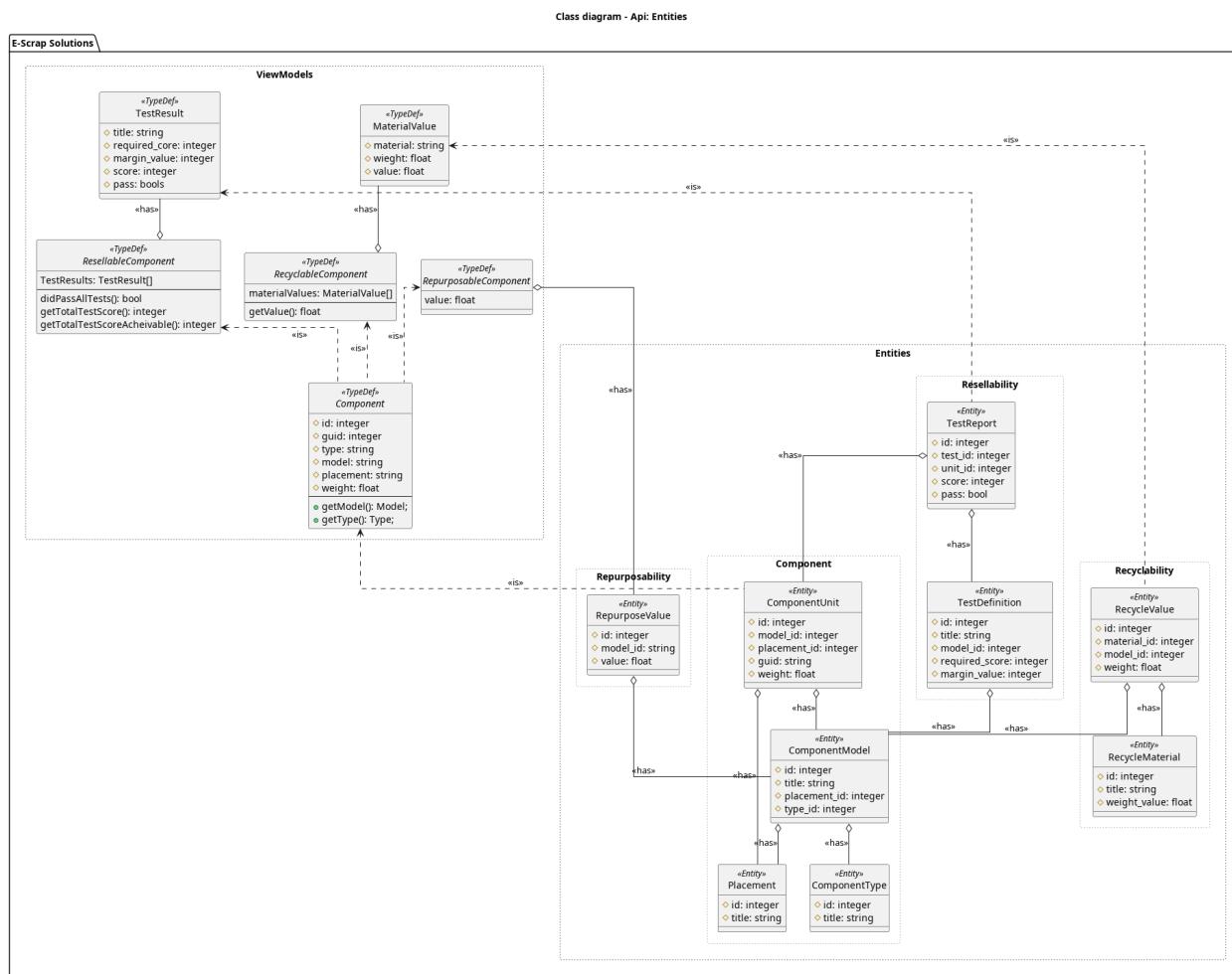
1.10.6.2 API: Controllers



Figur 8 - klassediagram for Rest Api'et: Controllers

I Figur 8 bliver der vist en oversigt over de tilgængelige controller og hvordan den abstrakte klasse er implementeret. Dette giver os muligheden for at kunne *Dependency Injecte* vores Controller klasser ind i Express Applikationen som vist på figur Figur 7.

1.10.6.3 API: Entities



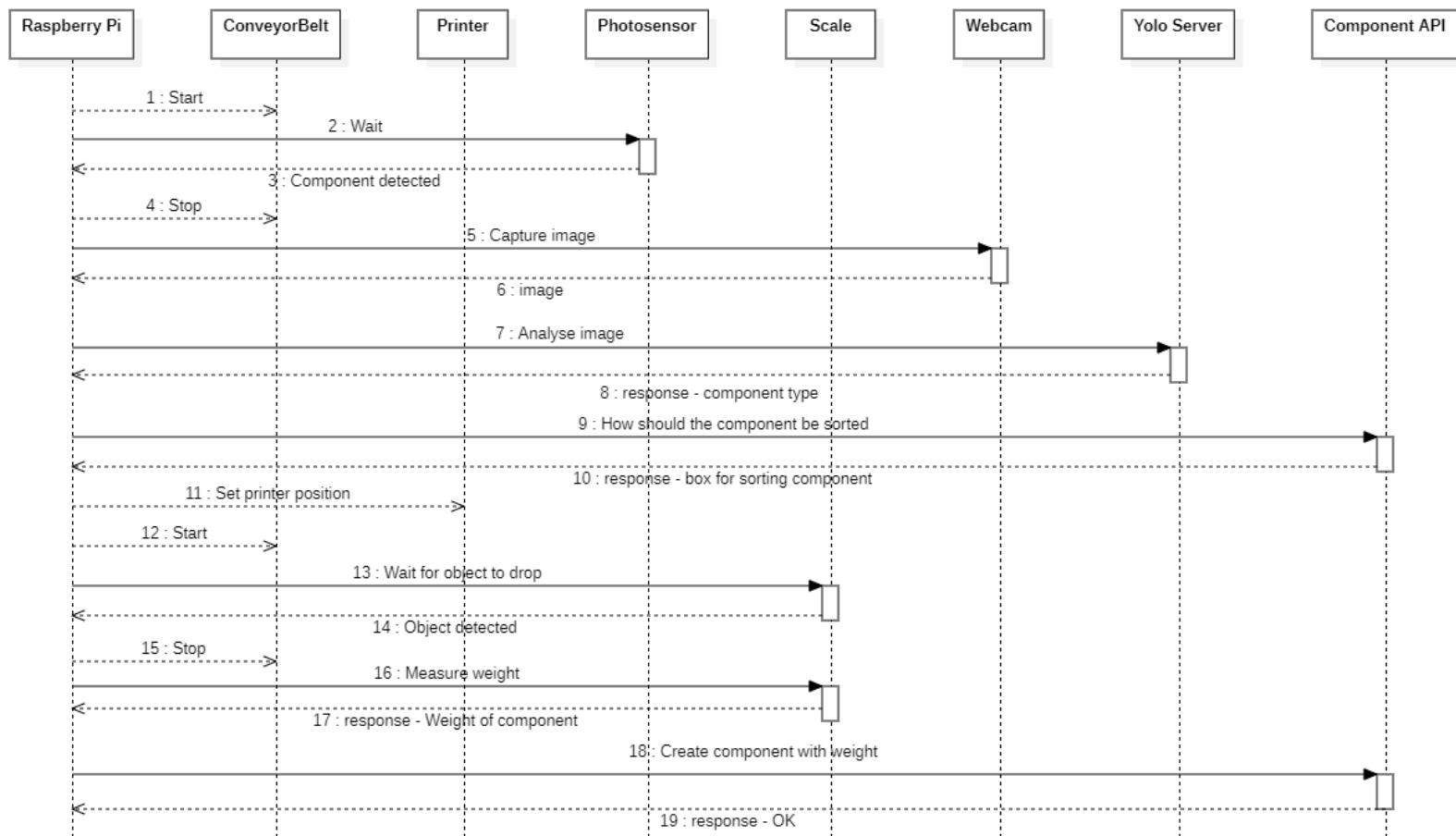
Figur 9 - Klassediagram for Rest Api'et: Entities

Figur 9 viser en illustration af, hvordan vores entiteter er opsat i vores Rest API, både for vores registrerede elektroniske komponenter, brugere og de repræsentative *ViewModels*. De viste *ViewModels* findes kun i systemet under hændelsesudførelse, da disse kun er type definitioner som bruges i webinterfacet. Vi har valgt at oprette og håndtere dem i Rest Api'et, da det giver bedre ydeevne at holde alt databehandlingen ud af Webinterfacet.

1.10.7 System Sekvensdiagram for Component Creation

Dette afsnit er indsat her for at forhåbentligt give en bedre forståelse af hvordan systemerne interagerer mellem hinanden i det forløb vi kalder for *Component Creation*. Vi råder til at der er før dette afsnit læst og forstået afsnittene [1.10.4 Flowcharts](#), [1.10.5 Entity Relationship Diagram](#) samt [1.10.6 Klassediagrammer](#).

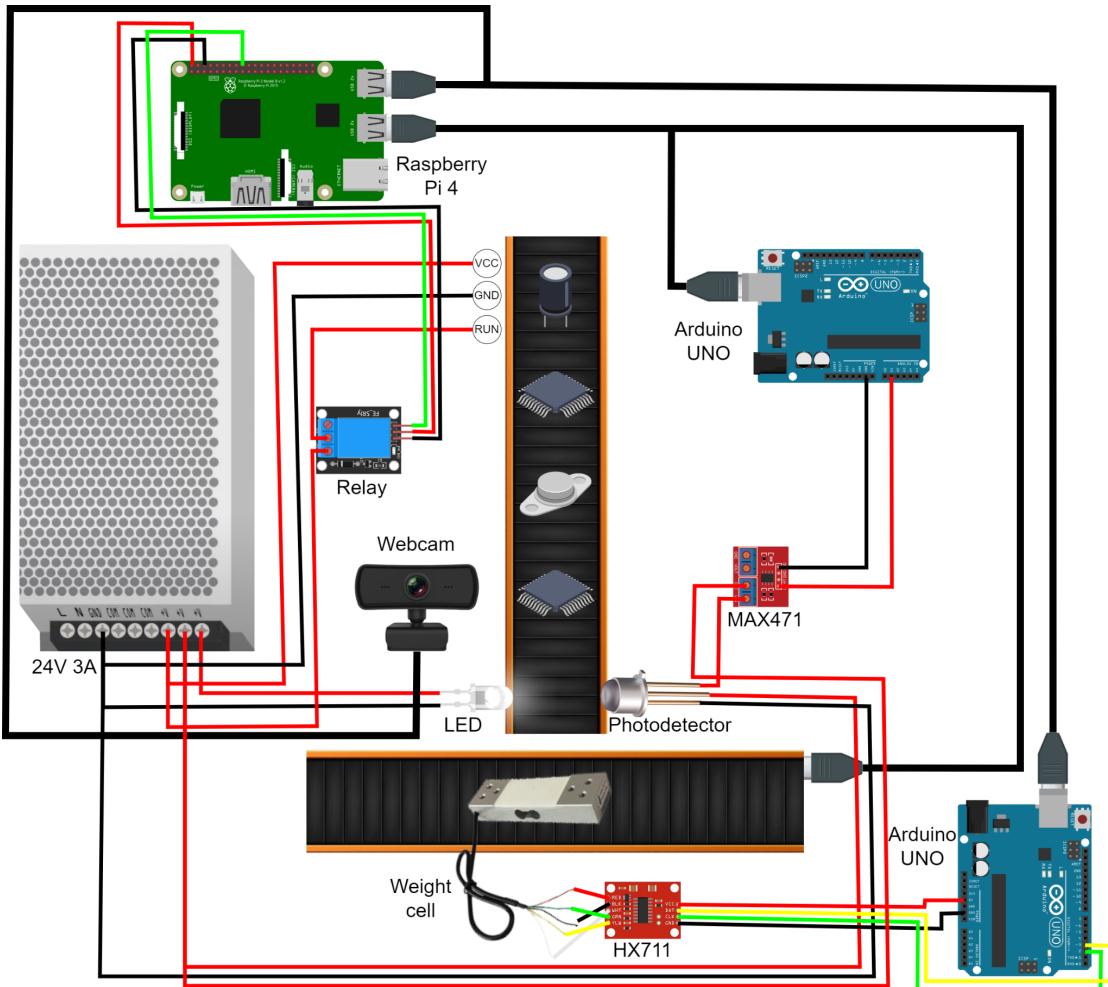
Ved *Component Creation* mener vi egentlig *Component Registration*, men grundet at det handler om interaktionen mellem sorterings- og registreringssystemet ved oprettelse af ny enhed i databasen, synes *Component Creation* titlen egner sig bedre.



Figur 10 - System sekvens diagrammet for sorteringssystemet

Figur 10 illustrerer den operationelle funktion af vores sorteringssystem i forhold til oprettelsen af komponenter i vores API og processen med at sende billeder til vores YOLO-server, der er ansvarlig for objektgenkendelse.

1.11.1 Logisk topologi



Figur 11 - Logisk topologi over vores sorteringssystem

Figur 11 viser hvordan vores opbygning af det fysiske sorteringsystem er opbygget i vores proof-of-concept.

1.11.2 Netværksoversigt

Router: 192.168.1.1

Netværk: 192.168.1.0/24

SSID: Linksys00103

Password: nhBMsvENThiNg4

1.11.3 Enheder

| Title | Hostname / endpoint | Ip:port | System & Framework | Source-kode sprog | Operativsystem |
|-----------------|---------------------|---------------|--------------------------|-------------------|----------------|
| Control Central | centralpi | 192.168.1.108 | Native Python | Python | RaspbianOS |
| Google Api | /upload | 34.90.55.55 | FastAPI | Python | Ubuntu |
| Web interface | - | 192.168.1.2 | NodeJs, Astro, Chart.JS | TypeScript | Docker |
| Rest Api | - | 192.168.1.2 | NodeJs, Express, TypeORM | TypeScript | Docker |
| Database | - | 127.0.0.1 | MariaDB | MySQL | Docker |

1.12 Testrapport

Dette afsnit omhandler de tests der er beskrevet i processraportten og skal til enhver tid holdes opdateret.

1.12.1 Unit Tests

Herunder er der vist et status-overblik over køрte unit tests opdelt per delsystem.

1.12.1.1 Sorteringssystem tests

Hver klasse i sorteringssystemet har fået tilføjet en unit test, der giver os mulighed for at bekræfte funktionaliteten, selv når der foretages ændringer i kildekoden.

Alle testene bliver lagt i undermappen /tests

1.12.1.2 Registreringssystem tests

Som der kan ses herunder på *Figur 12* er der på nuværende tidspunkt lavet unit test for de delte funktioner der er i mappen *src/functions*, da de klasser er de mest nødvendige funktionaliteter for at opretholde et minimums funktionalitets niveau. På *Figur 12* kan der ses hvordan vi har skrevet unit tests for disse funktioner.

```
> ess-regapp-api@1.0.0 test
> jest --verbose

PASS  tests/functions/getPlacementByComponentModel.test.ts
  getComponent Tests
    ✓ Should return null if unable to get Placement (4 ms)
    ✓ Should return Placement if input is correct (2 ms)

PASS  tests/functions/getComponent.test.ts
  getComponent Tests
    ✓ Should return null if unable to get Component (3 ms)
    ✓ Should return Component if input is Correct (4 ms)

Test Suites: 2 passed, 2 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        3.106 s
Ran all test suites.
```

Figur 12 - Unit tests for src/functions

1.12.2 User Acceptance Tests

Vi har herunder lavet UAT for hver af vores use-cases fra vores projektrapport.

| Differentiering mellem forskellige komponenttyper | Resultat | | Use-case:1 |
|---|----------|-----|------------|
| Acceptkriterie | Ja | Nej | Kommentar |
| Stopper komponentet foran sensoren? | x | | |
| Bliver komponentet genkendt af kameraet? | x | | |
| Kører komponentet videre på båndet? | x | | |

| Sortering af komponenter per type | Resultat | | Use-case:2 |
|--|----------|-----|------------|
| Acceptkriterie | Ja | Nej | Kommentar |
| Når komponentet bliver godkendt, bliver der givet information om hvilken komponenttype det er. | x | | |
| Bliver kassen der skal opfange, at komponenterne kørt ind foran transportbåndet? | x | | |
| Transportbåndet kører komponentet ned i den rigtige kasse. | x | | |

| Afvejning af enkelte komponents faktiske vægt | Resultat | | Use-case:3 |
|---|----------|-----|------------|
| Acceptkriterie | Ja | Nej | Kommentar |
| Når komponentet falder i den rigtige kasse, bliver der givet information omkring hvad denne enhed vejer | x | | |

| | | | |
|--|----------|-----|------------|
| Registrering af enkelte komponenter | Resultat | | Use-case:4 |
| Acceptkriterie | Ja | Nej | Kommentar |
| Når informationen af komponentens vægt og type er angivet, bliver komponentet registreret i Rest Api'et. Er dette komponent registreret med korrekt information | x | | |

| | | | |
|--|----------|-----|------------|
| Tilgang til den enkelte komponents information | Resultat | | Use-case:1 |
| Acceptkriterie | Ja | Nej | Kommentar |
| Kan man ved forespørgsel på et komponent få oplyst den rigtige information? | x | | |

| | | | |
|---|----------|-----|------------|
| Ændring af et komponents registeret værdi | Resultat | | Use-case:1 |
| Acceptkriterie | Ja | Nej | Kommentar |
| Kan man ved forespørgsel om at ændre et komponent få oplyst om, hvorvidt handlingen blev gennemført? | x | | |

| | | | |
|---|----------|-----|------------|
| Oversigtsvisning af komponenter, opdelt i placerings kategorier | Resultat | | Use-case:1 |
| Acceptkriterie | Ja | Nej | Kommentar |
| Får man vist et overblik over alle komponenter og deres registrerede placering I Web interfacet? | x | | |

1.13 Bilag

Eventuelle bilag kan findes i vores [Github Repository](#) under *docs* mappen.