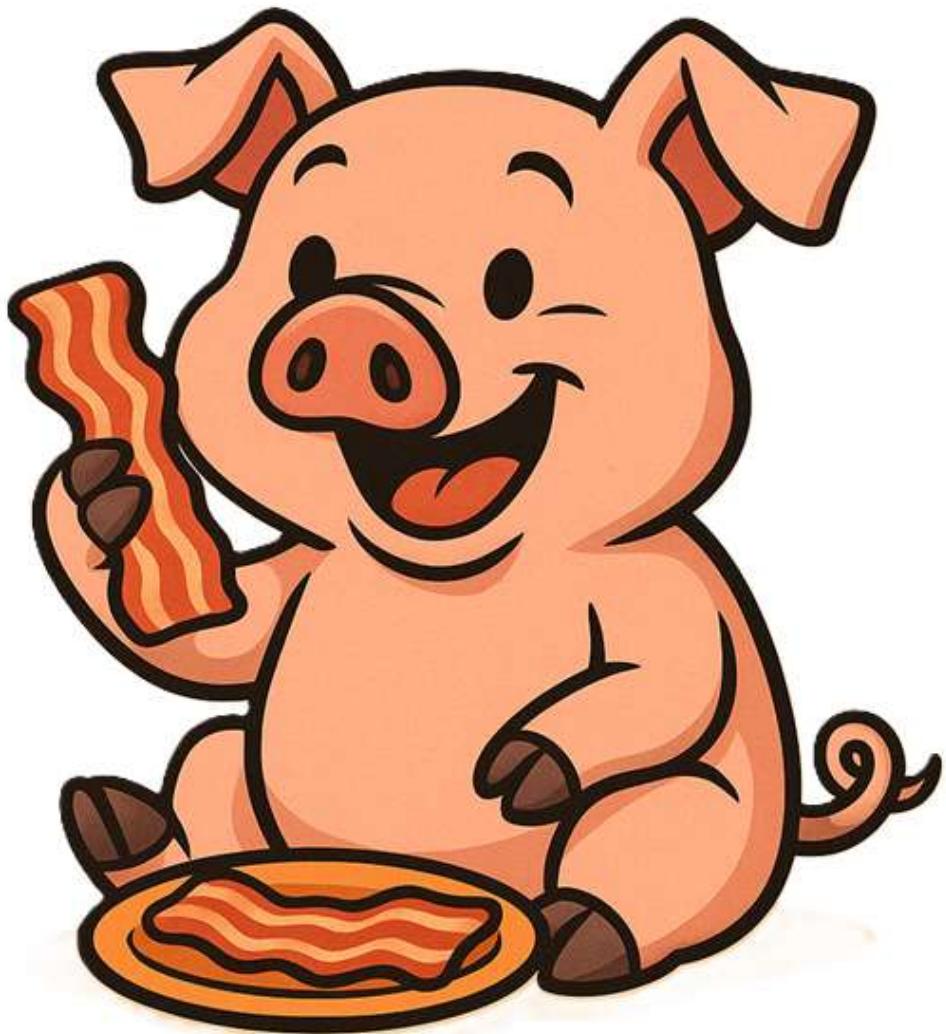


EKSAMENSPROJEKT

Agrisys – Grisedatasystem



DMU 2025 – EAMV

Henrik Bach Flensborg

Johannes Kizach

Martin Farsø

Antal tegn – 56496

Indholdsfortegnelse

1. Indledning.....	3
2. Metode	3
Azure DevOps og git	3
Vandfaldsmodel, radarmodel og hybridmodel.....	4
Scrum.....	7
MoSCoW.....	8
Unified Modeling Language.....	8
3-lags arkitektur.....	9
Kodemønstre.....	10
Sikkerhed – SQL Stored Procedures, Password Hashing.....	11
Normalisering af database	12
Test.....	12
AI.....	12
3. Kravspecifikation	13
MoSCoW Analyse – Agrisys' datasystem:	14
Use Case-diagram	15
KPI'er (Key Performance Indicators).....	15
4. Design	17
Guidesign.....	17
Databasedesign.....	19
Design Patterns	23
5. Implementering.....	29
Sprintstruktur	29
Scrum-forløb: Recap af Sprint 1-3	30
Sprint 1 – 05/05/2025 - 09/05/2025	30
Sprint 2 – 12/05/2025 - 16/05/2025	30
Sprint 3 – 19/05/2025 - 20/05/2025	31
6. Afprøvning	31
Testcheckliste	31
Fejlliste	33
Særligt komplekse funktioner	34

7. Konklusion og reflektion.....	36
8. Bibliografi	38
9. Bilag	40

1. Indledning

(Martin)

I takt med den teknologiske udvikling i landbrugssektoren stiger behovet for intelligente og automatiserede løsninger, der kan optimere både dyrevelfærd og produktivitet. Agrisys har specialiseret sig i automatiserede fodrings- og overvågningssystemer til svineproduktion. Virksomheden samarbejder tæt med landmænd for at levere systemer, som understøtter dataanalyse og effektiv drift.

Eksamensprojektet tager udgangspunkt i et konkret behov fra Agrisys: Udviklingen af et system, der kan importere fodringsdata fra Excel-filer, analysere forbruget og visualisere tendenser i dataene. Systemet skal desuden kunne generere advarsler, hvis en gris ikke har indtaget tilstrækkeligt foder over de seneste tre dage. Formålet med systemet er at give landmændene et brugervenligt og datadrevet værktøj, der kan bidrage til at identificere potentielle sundhedsproblemer hos dyrene i tide og dermed styrke den generelle dyrevelfærd og produktionskvalitet.

2. Metode

(Henrik, Johannes, Martin)

Vi bibruger vores gruppekontrakt fra tidligere projekter, da den hjælper til at opretholde en god arbejdsmoral og sikre at vi lever op til de forventninger vi stiller overfor hinanden. Derudover gennemgår vi koden i fællesskab, da det skaber stor værdi og forståelse for vores projekter, og sikrer at vi kan stå inde for projektet som helhed. Til slut indgår kodenstandarder også i kontrakten, der er med til at sikrer lighed mellem forskellige udviklernes kode (se bilag A).

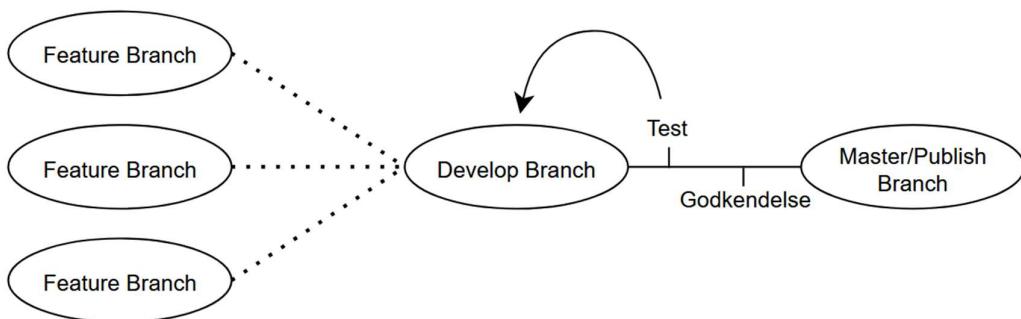
Azure DevOps og git

For at have en samlet platform til vores softwareudviklingsprocess, benytter vi os af Azure DevOps. Her kan vi gøre alt fra planlægning til bygning, test og deployment. Det er herigennem vi vil oprette vores repository til versionsstyring af vores program. Derudover benytter vi DevOps wiki, for at formidle vigtige informationer mellem udviklere. Det kan fx være vores stored procedures, forventede resultater på test osv. DevOps tilbyder også Kanban-board til oversigt over arbejdsopgaver i udviklingsfasen.



Figur 1 - Azure DevOps

Vi benytter os af vores egen git-struktur, baseret på Gitflow [1], til at sikre at alle kan arbejde på samme tid, uden at blokere hinanden eller ødelægge andres kode.



Figur 2 - Git Struktur

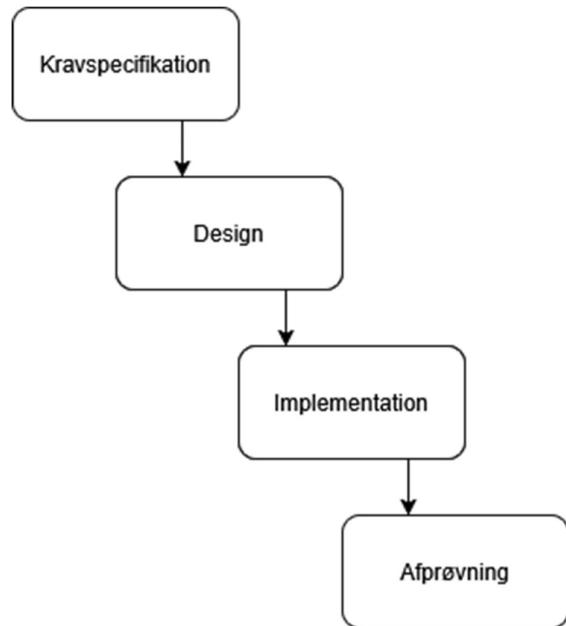
Vi har to branches som har uendelig levetid, develop og master [2]. Master branch er vores endelig produkt som er publiceret og er i “drift”. Vores develop branch er hvor vi udvikler og tilføjer vores ændringer, så vi ikke arbejder direkte i vores driftkode. Når der skal publiceres, gennemføres der test, for at sikre funktionalitet og kvalitet. Når test er gennemført, sikres programmet af minimum to medlemmer, før det endeligt bliver publiceret. Det sikrer at vi ikke ødelægger fungerende kode og at ingen har suverænt ansvar for vores produkt.

Til slut har vi vores feature branches, som hver har specifikke udgangspunkter og forventninger. De dækker over nye features, udvidelser eller rettelser af eksisterende kode. Vores feature branches er lokale, for vi kommer ikke til at arbejde på kryds af features. Det hjælper også til at mindske risikoen for merge conflicts.

Vandfaldsmodel, radarmodel og hybridmodel

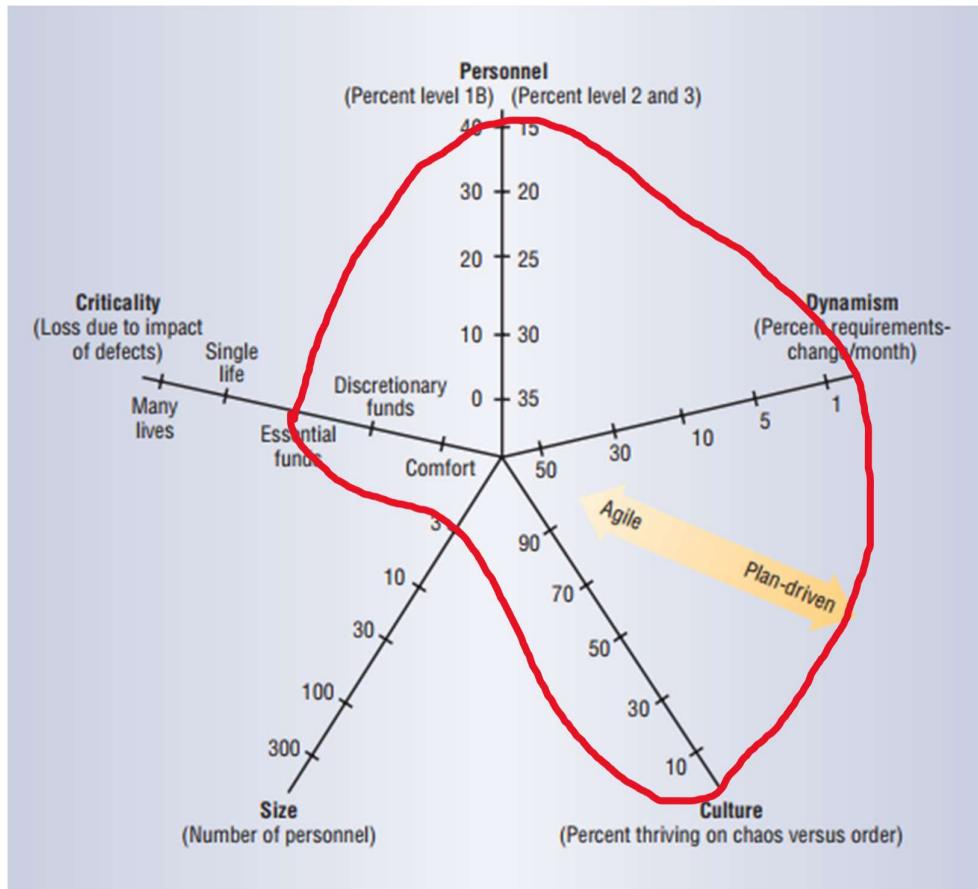
Vandfaldsmodellen er en traditionel tilgang til softwareudvikling, hvor arbejdet deles i klart adskilte trin, fx kravspecifikation, design, udvikling og afprøvning [3]. Metoden

anses for at være rigid og egner sig bedst til projekter hvor der ikke kommer overraskelser (nye krav, ændringer, budgetnedskæringer). Ligesom i et rigtigt vandfald er der kun én retning: fremad.



Figur 3 - Vandfaldsmodellens trin

Radarmodellen er et værktøj udviklet af Boehm og Turner [4] beregnet til at vælge en passende metode til et softwareudviklingsprojekt. Tanken er at man skal vælge enten en agil metode (Scrum) eller en traditionel (vandfaldsmodellen). Valget baseres på fem faktorer (se figur 4, fra [4]): Størrelse, risiko, kompetence, forandring, kultur.



Figur 4 - De fem faktorer i radarmodellen

Hvis vi ser på vores gruppe og projektet ift. de fem faktorer, så er størrelsen på holdet kun tre, og det peger på at bruge Scrum. Risikoen involverer naturligvis ikke tab af liv, men handler om at bestå en eksamen – det må vel svare til *essential funds* i en virksomhed, så her lander vi midt på skalaen. Vi er begyndere, så vores kompetence er per definition meget lav – det peger entydigt på vandfaldsmodellen (stram planlægning og klare mål giver færre fejl for begyndere). Der er ingen udefrakommende forandringer i løbet af projektet fordi det er en eksamen – så den faktor peger 100% på vandfaldsmodellen. Den sidste faktor er kulturen i gruppen: Trives man bedst med kaos eller med orden? Vi trives bedst med orden – så også her peger modellen på vandfaldsmodellen.

Den røde streg markerer resultatet af vores analyse, og som man ser er konklusionen at vandfaldsmodellen er mest hensigtsmæssig for os i det her tilfælde.

Der er dog undersøgelser der viser at det er mest almindeligt i praksis at blande de to modeller: den såkaldte hybridmodel [5]. Tanken er at man benytter sig af

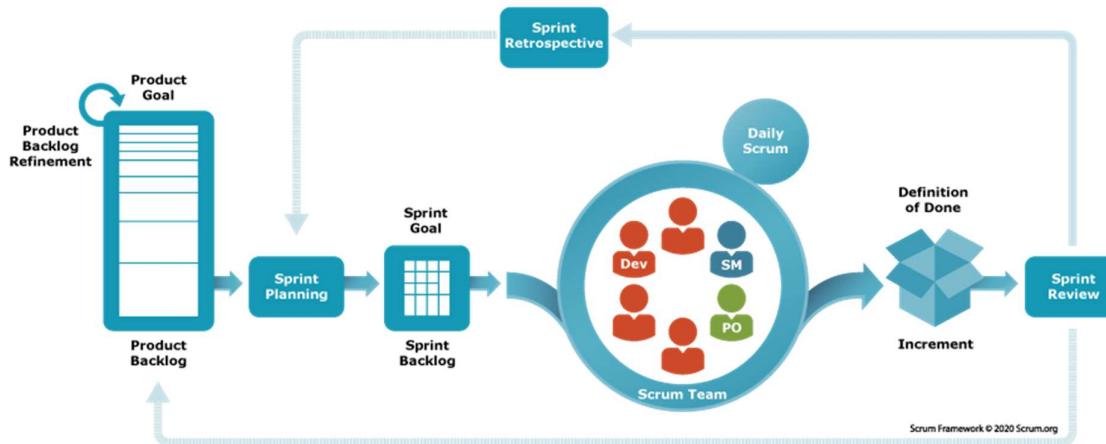
vandfaldsmodellen til kravspecifikation, design og afprøvning – men i udviklingsfasen benytter man Scrum. Vi har valgt at følge bruge hybridmodellen i vores projekt.

Scrum

Scrum er, modsat Vandfaldsmodellen, et agilt udviklingsframework, der anvendes til at strukturere og effektivisere samarbejdet i softwareudviklingsprojekter. Det bygger på iterative processer, hvor arbejdet opdeles i korte, tidsafgrænsede perioder kaldet sprints. Hvert sprint afsluttes med en gennemgang og evaluering af det udførte arbejde, hvilket sikrer løbende forbedring og tilpasning i projektforløbet [6].

I projektet vil vi anvende Scrum til at strukturere vores implementeringsfase. Vi inddeler arbejdet i sprints for at sikre kontinuerlig fremdrift, afklaring af opgaver og hyppig evaluering.

Vi vil benytte os af et traditionelt Scrum framework [7] med en Scrum Master og et developerhold, men vil ikke have en traditionel product owner.



Figur 5 - Sprintforløb

Vores sprinttilgang vil være meget traditionel, med sprintplanning, målsætning, daglige Scrum-møder, backlog og sprintreview. Vores sprints vil have en uges varighed, med forbehold for at det kan blive reduceret, hvis vi bliver begrænset i mængden af arbejdsopgaver. Den tilgang giver os fleksibilitet i opgaveløsningen under vores implementation, samtidig med at vi bevarer overblikket og sikrer løbende fremdrift og kvalitet i implementeringen.

Vi vil bruge et Kanban-board [8] til at holde overblik og styr på vores arbejdsopgaver under udviklingsfasen. Før hvert sprint opretter vi arbejdsopgaver som forventes færdige inden for de individuelle sprints. Hvis vi ikke når det ønskede resultat, overføres arbejdsopgaverne som en backlog og indgår i næste sprint.

Den konkrete brug af Scrum, som sprintstruktur og arbejdsdeling, vil blive uddybet i implementeringsafsnittet.

MoSCoW

For at udpense vores kravspecifikationer til projektet har vi valgt at bruge MoSCoW. MoSCoW er et analyseværktøj, der bruges til at rangordne krav, funktioner eller opgaver i projektstyring [9]. Akronymet står for:

- Must have: Absolut nødvendige krav for at løsningen fungerer.
- Should have: Vigtige, men ikke kritiske krav, der giver stor værdi.
- Could have: Ønskelige krav, der kan tilføjes, hvis tid og ressourcer tillader det.
- Won't have: Laveste prioritet, tages ikke med i denne iteration.

Metoden hjælper med at sikre, at alle forstår og accepterer prioriteringen af krav, og at udviklingsressourcer fokuseres på det mest forretningskritiske.

Unified Modeling Language

For at kunne præsentere vores projekt visuelt anvender vi forskellige UML-diagrammer. UML står for *Unified Modeling Language* og er et standardiseret visuelt sprog til at modellere software- og systemdesign [10]. Det bruges til at beskrive, specificere, visualisere og dokumentere forskellige aspekter af et system. Vi anvender tre UML-typer: Use-Case, Klasse og Sekvensdiagram og ER-diagram til visualisering af databasen (ER-diagrammet er dog ikke en del af UML-pakken).

Use Case-diagrammet bruges til at beskrive systemets funktionalitet set fra brugerens perspektiv. Det viser hvilke aktører (brugere eller andre systemer) der interagerer med systemet, og hvilke "use cases" (funktioner) de benytter. Formålet er at give et overblik over krav og funktionalitet uden at gå i tekniske detaljer.

Klassediagrammerne beskriver systemets struktur. Det viser klasserne i systemet, deres attributter, metoder samt relationer (som arv, association og komposition) mellem klasserne. Det bruges til at modellere de vigtigste objekter og deres sammenhænge i systemets softwarearkitektur.

Sekvensdiagrammet bruges til at illustrere, hvordan objekter kommunikerer med hinanden over tid i en specifik use case. Det viser rækkefølgen af beskeder/metodekald mellem systemets objekter, hvilket giver indsigt i programmets vej til et resultat.

Vores ER-diagram anvendes til at modellere databasedesign. Det viser entiteter, deres attributter og relationer. Selvom det ikke er en del af UML-standarden, bruges det ofte parallelt i systemudvikling til at designe den bagvedliggende datamodel.

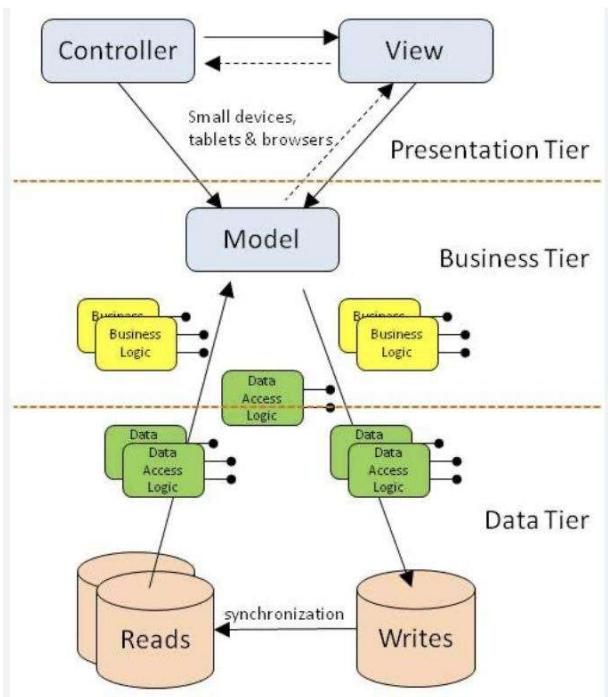
3-lags arkitektur

I udviklingen af applikationen anvender vi 3-lags arkitektur, som opdeler vores softwarearkitektur i tre lag:

- Præsentationslaget håndterer al interaktion med brugeren, såsom visning af data og input fra brugeren.
- Forretningslaget indeholder programmets kernefunktioner og logik. Her behandles data, beregninger og regler, som er centrale for systemets funktionalitet.
- Datalaget står for kommunikationen med databasen. Det sikrer, at al databaseadgang sker kontrolleret og struktureret.

Inden for præsentationslaget implementerer vi MVC-mønstret, for yderligere at strukturer koden:

- Model som repræsenterer applikationens data og forretningsregler.
- View som er ansvarlig for at præsenterer data.
- Controller håndterer brugerinput og opdaterer modellen og viewet med hinanden.



Figur 6 - MVC og trelags arkitektur

Kombinationen af arkitektur og designmønster, er en almen tilgang til 3-lags arkitektur, for at opnå en større separation og lettere vedligeholdelse [11], [12], [13].

Kodemønstre

Vi vil benytte såkaldte *design patterns* i vores kode. Design patterns er måder at løse hyppige kodeproblemer på og de blev første gang formuleret i 1994 [14], men er siden blevet uddybet og udviklet og vi forholder os primært til de mønstre der er beskrevet i bogen Head First Design Patterns fra 2021 [15]. Vi bruger som allerede nævnt MVC og derudover vil vi bruge Builder, Simple Factory, Singleton og Facade i vores program. Den præcise brug beskrives i designafsnittet, men her præsenterer vi kort hvert enkelt mønster.

MVC. Model-view-controller-mønsteret er beregnet til situationer hvor man har en GUI der skal vise noget data. Ideen er at *view* kun tager sig af selve de grafiske elementer og udseendet, *controller* sørger for brugerens interaktion med GUIen og sørger for at den information der kommer fra *model* kan ses i GUIen. Endelig sørger *model* for al interaktion med databasen og al nødvendig datamanipulation.

Builder. Hvis man har brug for at sende information fra fx databasen til modellen, men det afhænger af situationen præcis hvilke data man har brug for - så kan man have en række forskellige data-klasser som er næsten ens, men med små variationer. Eller man

kan have én klasse som er en *Builder* – det vil sige at den selv kan vælge de informationer der er relevante og bygge sig op med dem. Det smarte er at modtageren – fx modellen – kun skal forholde sig til én klasse og ved at den får netop det den skal bruge.

Simple Factory. Hvis man har mange betingelser i en klasse (if-then-else) så kan man nogle gange med fordel uddeletere betingelserne til en separat klasse. I stedet for at gøre A hvis betingelse X, og B hvis Z osv, så kan man samle alle mulige udfald i en anden klasse og så lade den oprindelige klasse indeholde en instans af den nye klasse. Når betingelserne styrer hvilken slags objekter man skaber, så kalder man sådan en hjælpeklasse for en Factory. Fordi den producerer de elementer man skal bruge. Fordelen er at alle detaljer om betingelserne og om hvordan de producerede elementer skal være er samlet ét sted – det letter vedligehold og ændringer.

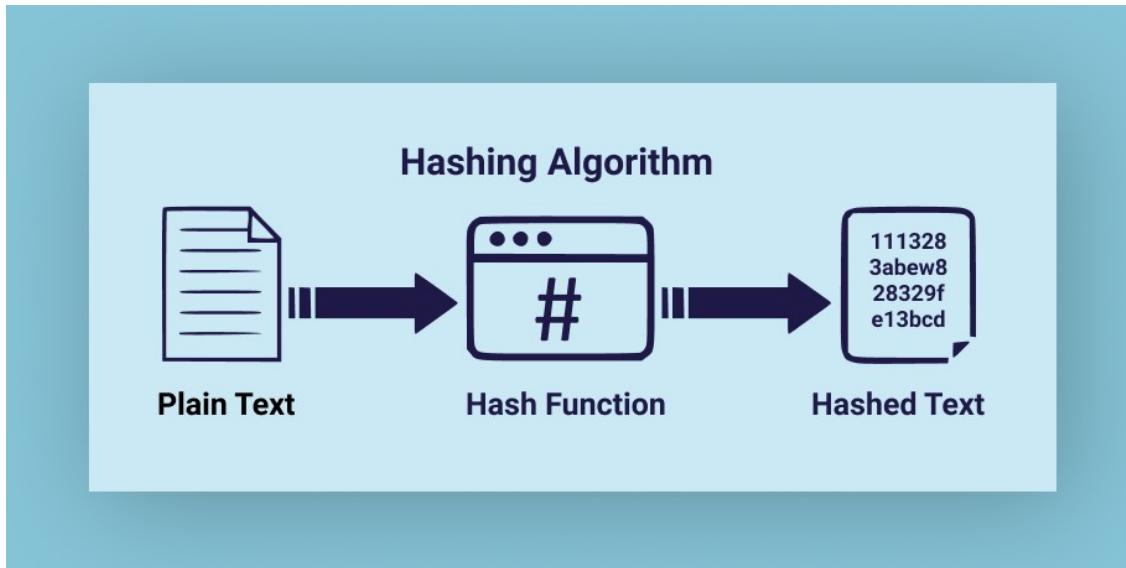
Singleton. Mønstret sikrer at der kun kan være én instans af klassen på samme tid når programmet kører.

Facade. Hvis en klasse har et kompleks interface med mange funktioner så kan det være en fordel at lave et simplere interface som en anden klasse kan forholde sig til i stedet. Det kan være en klasse holder styr på om en masse forskellige parametre er slået til eller ej gennem en række tjek-funktioner. Men hvis nu en klientklasse kun vil vide hvor mange der er slået til, men er ligeglads med hvilke det er – så kunne der implementeres en funktion som tjekker alle parametrene med tjek-funktionerne, tæller hvor mange der er slået til og returnerer antallet. Det er så præcis en facade man har lavet: Et simpelt interface som dækker over en større bagvedliggende kompleksitet.

Sikkerhed – SQL Stored Procedures, Password Hashing

For at styrke sikkerheden i vores applikation, anvender vi stored procedures frem for direkte SQL-forespørgsler fra applikationslaget. Ved at benytte stored procedures, begrænses vi adgangen til vores underliggende tabeller og minimerer risikoen for SQL-injektion. Ved at opdele rettigheder til forskellige procedurer uden direkte adgang til tabellerne, opnår vi samtidig en mere kontrolleret og sikker datatilgang [16].

I håndteringen af brugeraldagsgoder anvender vi jBCrypt. Det er en algoritme som er designet til at være adaptiv og inkluderer en indbygget salt (sikrer at ens passwords får forskellige hashes) [17]. Ved at implementere jBCrypt sikrer vi at adgangskoder ikke gemmes som tekst, men som en hash, og det betyder at de er beskyttet mod angreb på ens database, fordi et eventuelt læk ikke vil give angriberne koderne, men bare hash-versionerne af koderne og endda med iblandet salt (en ekstra hash som genereres og gemmes sammen med kodeordets hash) [18].



Figur 7 - Password Hashing

Normalisering af database

For at få data i tredje normalform, skal man sikre sig at der er en unik primærnøgle i hver tabel, og at der ikke er transitive afhængigheder imellem attributter som ikke er primærnøgler. Det betyder at værdierne er afhængige af primærnøglen, og kun af primærnøglen [19].

Test

Til verifikation og validering af vores software har vi benyttet en systematisk tilgang til test, som vi opdeler i overordnede testkategorier. De kategorier vil udspringe direkte af vores kravspecifikation og dækker henholdsvis over funktionalitet relateret til brugerhåndtering og sikkerhed, visuelle komponenter og widgets, brugerinteraktion samt systemets respons og output. Den systematiske test vil være struktureret som en checkliste, som vil blive udarbejdet til slut i forløbet.

AI

I vores eksamsprojekt har vi anvendt ChatGPT som en tutor til at besvare spørgsmål og guide os gennem udfordringer, som man normalt ville søge svar på via Stack Overflow. Det har bidraget til hurtigere problemløsning og en bedre forståelse af komplekse emner.

3. Kravspecifikation

(Henrik, Martin)

Her beskrives de tekniske krav og specifikationer til den database, der skal anvendes i projektet. Fokus er på, hvilke data systemet skal kunne håndtere, hvordan de skal struktureres, samt hvilke funktionelle krav der stilles til programmet i relation til databasen.

Datasættet fra Agrisys består af to separate tabeller: Animal Data og Visit Data. Vi betragter dem som uafhængige datasæt, da de ikke har en direkte relation til hinanden, bortset fra felterne *Responder* og *Location*, som findes i begge. Af den grund vurderer vi Animal Data som et udsnit af en større datamængde, formentlig Visit Data, hvor grisens vægt før og efter testfasen er blevet tilføjet. Visit Data er derimod et dynamisk datasæt der modtager nye data fra dag til dag, om grisens spisemønstre og herunder antal gange den besøger en foderstand på en dag. Visit Data vil derfor udvikle sig over tid, og efterhånden vokse sig større, når data for flere målingsdage bliver tilføjet.

Databasen skal oprettes i Microsoft SQL Server (MSSQL), og det er et krav, at data kan importeres fra Excel-filer via applikationen.

Derudover skal systemet understøtte to slags brugere: almindelige brugere og superbrugere. Det implementeres via en separat brugertabel i databasen, som indeholder information om brugertype og sikrer adgangsoplysninger gennem hashing af adgangskoder.

MoSCoW Analyse – Agrisys' datasystem:

Must Have:

Krav	Begrundelse
Import af data fra Excel	Grundlæggende for systemets funktionalitet
Gemme data i MSSQL database	Data skal kunne tilgås, analyseres og visualiseres
Adgangskontrol med roller	Nødvendig for sikkerhed og brugervenlighed
Visualisering af data i JavaFX	Visualisering er kernen for produktet
Tilpasning af Dashboard	Ændre de forskellige KPI'er der kan tilgås af alm. bruger.
Advarselsfunktion	Kritisk funktion ift. dyrevelfærd, drift og overvågning
Eksport af data	Skal kunne eksportere KPI-resultater til CSV

Should Have:

Krav	Begrundelse
Listevisning af alle grise med advarsler	Understøtter advarselsfunktionen, men med større overblik
Simulation af data	Indgår som bonus til eksamensprojektet, men bliver nedprioriteret, da det ikke er essentielt for projektet.

Could Have:

Krav	Begrundelse
Sortering og filtrering i Dashboards	Øger brugbarhed
Søgefunktion	Kvalitetsforbedring, men ikke essentiel
Printvenlig visning af grafer	Ekstra værdi, men ikke nødvendig for funktionalitet
Logbog	Mulighed for at logge/dokumentere behandling/tjek af sulten gris
Brugerdefinerede KPI'er	Øge funktionalitet og være mere brugerspecifikt

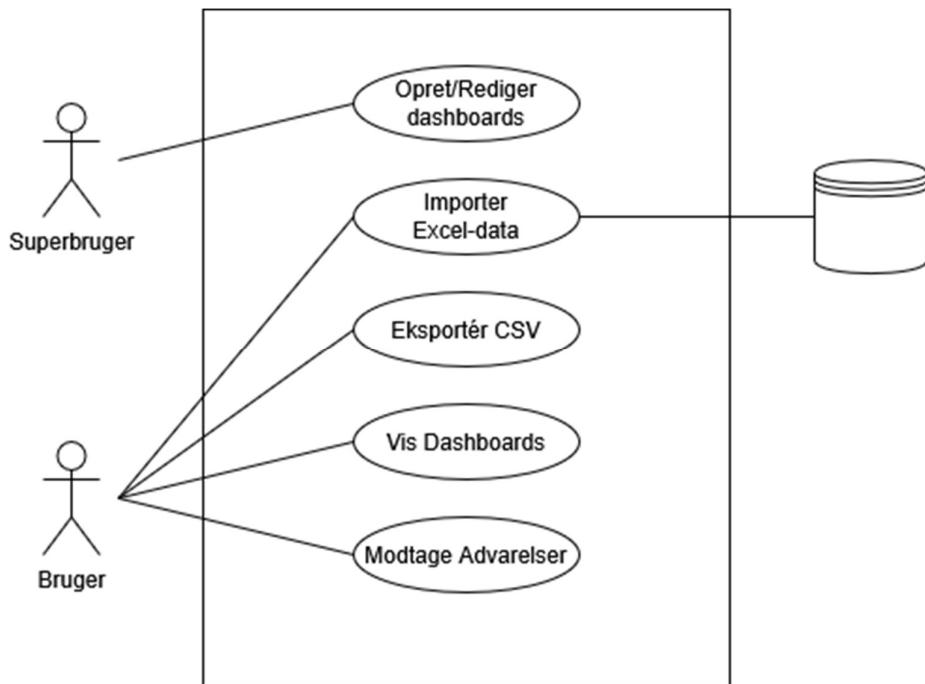
Will Not Have:

Krav	Begrundelse
Online brugerlogin / Cloud-integration	Ikke nødvendig for funktionalitet af programmet
Mobilversion / Responsivt Design	Fokus er på desktop med JavaFX
Realtid Datafeed fra sensorer	Ikke realistisk med vores datastrøm.

MoSCoW-analysen for Agrisys' datasystem identificerer de vigtigste krav til systemet opdelt i fire prioritetskategorier. Must Have dækker over de essentielle funktioner, som

fremgår af eksamenskravene. Det vil sige systemets grundlæggende drift, sikkerhed og brugeroplevelse. Det er her vi vil prioritere og tage udgangspunkt i vores projekt. Hvis vi har nok tid, vil vi kigge videre i Should Have, hvor der indgår visuelle forbedringer og bonusprojekt til eksamen. Vi vil se bort fra Could Have og Will Not Have, da det ikke er realistisk at implementere inden for tidsrammen.

Use Case-diagram



Figur 8 - Use case diagram

Use case-diagrammet viser et system til håndtering og visualisering af data via dashboards. Der er to typer brugere: Superbruger og Bruger. Superbrugeren har adgang til de funktioner som brugeren har adgang til. Derudover kan superbrugeren oprette eller redigere dashboards. En almindelig bruger kan se dashboards, modtage advarsler, importere fra Excel og eksportere data som CSV-filer.

KPI'er (Key Performance Indicators)

For at opnå en helhedsvurdering af grisens trivsel, adfærd og produktivitet anvendes en række nøje udvalgte KPI'er, som bygger på både Animal Data og Visit Data. Med KPI mener vi data, som vi gerne vil trække ud af databasen og fremvise i vores GUI. KPI'erne kan veksle i form. Nogle er blot et enkelt nøgletal, andre er udregninger, som vi udfører med SQL-kode som stored procedures. I vores vandfaldfase besluttede vi os for at hente 17 KPI'er fra databasen. Efter implementeringen endte vi med i alt 26 KPI'er.

Nedenfor ses listen over dem:

Animal Data

1. Foderudnyttelse baseret på FCR (Feed Conversion Rate)

Måler forholdet mellem foderforbrug og vægtøgning. FCR fungerer som indikator for foderets effektivitet og kan anvendes til at identificere sygdom eller dårligt foder.

2. Gennemsnitlig FCR

Giver landmanden et samlet billede af effektiviteten i forhold til fodring og vægtøgning.

3. Gennemsnitlig FCR per lokation

Viser effektiviteten på de enkelte fodringsstationer.

4. Antal grise per lokation

Giver overblik over, hvor mange grise der befinner sig på en specifik lokation.

5. Gennemsnitlig daglig vægtøgning

En vigtig indikator for sund vækst og for, om grisen følger den ønskede vækstkurve.

6. Kønsfordeling af grisene

Anvendes bl.a. fordi orner ofte ikke fortsætter i målingerne.

Visit Data

7. Totalt foderindtag

Gør det muligt at sammenligne grise over tid, mellem grupper eller stationer og identificere eventuelle problemer.

8. Totalt foderindtag per dag per lokation

Giver overblik over det daglige foderforbrug på de enkelte lokationer.

9. Gennemsnitligt foderindtag per besøg

Indikerer hvor meget foder en gris indtager per besøg. Kan afsløre ændringer i adfærd, som kan være tegn på sygdom eller stress.

10. Totalt foderindtag per dag per gris

Bruges til at sikre, at den enkelte gris trives og følger den forventede vækstkurve.

11. Daglige besøg per lokation

Viser aktivitetsniveauet for hver lokation.

12. Gennemsnitlig besøgstid

Giver et nøgletal for den typiske varighed af et besøg ved foderstand.

13. Gennemsnitlig besøgstid per lokation

Giver indsigt i forskelle mellem lokationers brugsmønstre.

14. Gennemsnitlig besøgstid per gris

Kan bruges til at identificere grise, som eventuelt har problemer med fødeindtag eller adgang til foderstanden.

Advarselsbaserede KPI

15. Grise med negativ FCR

Identifierer grise, som har tabt vægt under målingerne.

16. Advarsel ved lavt foderindtag over 3 dage

Udløser en alarm, hvis en gris har indtaget under en fastsat minimumsgrænse.

Der vises både foderindtag og grisens respondernummer.

17. Advarsel ved stort fald i foderindtag på lokation

Giver besked, hvis der observeres markant fald i foderindtag på en lokation over de seneste 3 dage.

KPI'er tilføjet under implementering

18. FCR-distribution

Visualiserer fordelingen af FCR-værdier blandt grisene. Hjælper med at identificere typiske eller afvigende tendenser.

19. Distribueret gennemsnitlig vægtforøgelse

Viser vægtforøgelse i fordelingsform, så man lettere kan se de mest typiske vækstniveauer.

20. Antal grise

Simpelt nøgletal, der viser, hvor mange grise der optræder i datasættet.

21. Antal lokationer

Viser antallet af unikke lokationer i datasættet.

22. Totalt foderforbrug

Giver samlet overblik over foderforbruget i det valgte datasæt.

23. Procentdel grise med positiv FCR

Simpelt nøgletal.

24. Procentdel grise med negativ FCR

Simpelt nøgletal.

25. Gennemsnitlig vægt

Viser gennemsnitsvægten for grisene i datasættet.

26. Antal grise over 82 kg

Viser, hvor mange grise der potentielt er slagteklares. Vægten er udledt af tal fra Danish Crown [20].

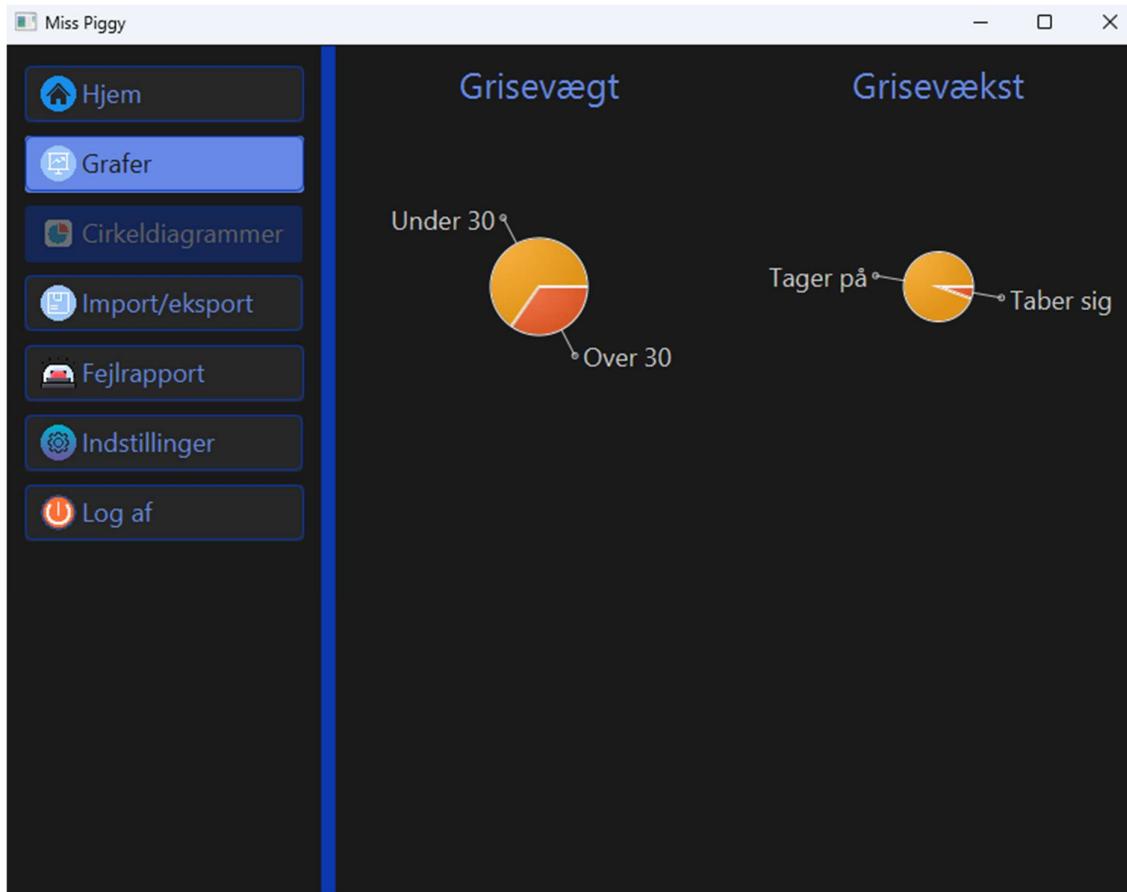
4. Design

(Henrik, Johannes)

Guidesign

Der skal vises en del data, så det er vigtigt at der er plads i GUlen. Problemet med at have flere sider (meget plads) er at brugerne kan miste orienteringen. En fin løsning på det problem ser man i Azure DevOps, hvor der er menupunkter i venstre side og

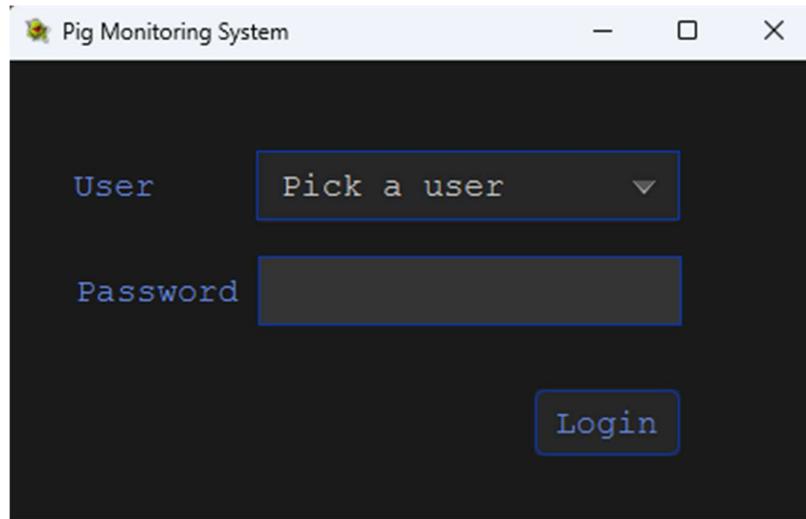
indholdet af de enkelte punkter vises i et større felt i højre side. Det giver muligheden for at have mange sider, men man farer aldrig vild fordi menuen i siden viser hvor man er, og hvor man kan komme hen. Derfor valgte vi at opbygge vores GUI baseret på Azures intuitive design.



Figur 9 - En prototype lavet i scene-builder

Menuen i venstre side bruger både ikoner og tydelige navne, og indholdsfeltet i højre side viser så den aktuelle side – her er det menupunktet Cirkeldiagrammer. Det er klart at vi skal tilpasse antallet af menupunkter og navne osv. ift. de aktuelle krav og organisere visning af data på en hensigtsmæssig måde. Men grundlæggende kan det her enkle design udvides og tilpasses efter behov (der kan tilføjes eller fjernes menupunkter) og man kan desuden – ligesom i DevOps – tilføje underpunkter og på den måde udvide den flade struktur til en hierarkisk.

Udseendet (font og farver) er bestemt i en css-fil som er knyttet til rodscenen.



Figur 10 - Login-vinduet

GULen vil bestå af to vinduer, fordi der skal være en login-funktion og det er praktisk at det foregår i et separat vindue. Hovedvinduet bliver så kun tilgængeligt hvis man har det rigtige brugernavn og kodeord.

Databasedesign

For at kunne starte et datadrevet projekt er en velfungerende database en absolut nødvendighed. Data skal ikke blot kunne opbevares, men også kunne trækkes ud via SQL, så det kan anvendes og visualiseres i vores program. Til projektet fik vi stillet en større mængde data til rådighed fra Agrisys, som vi skulle anvende til at populere vores database.

Fra Agrisys modtog vi to tabeller: **Animal Data** og **Visit Data**.

Animal Data indeholdt 343 unikke grise (identificeret ved deres *Responder*), hver med deres egne måledata som:

Location (grisens unikke foderstation), Sex, FCR, StartWeight, TotalFeedIntake, WeightGain, EndWeight og CompletedDaysInTest.

Tabel 1: Animal Data

Location	Number	Responder	Sex	FCR	Start weight (kg)	Total feed intake (kg)	Weight gain (kg)	End weight (kg)	Completed Days In Test
2	283258	984000009	Female	0,59	29,5	7,3	12,3	41,8	5

Figur 11 - Oversigt over tabellen Animal Data

Visit Data indeholdt oplysninger om grisens aktivitet i en given tidsperiode. Ud over Location og Responder fik vi også data som:

VisitDate (den konkrete dato for målingen), Duration (hvor længe gris'en havde opholdt sig ved foderstationen den dag) og FeedAmount (hvor meget foder gris'en indtog i løbet af et besøg ved en foderstand).

Table 2: Visit Data

Number	Location	Responder	Date	Duration	Feed amount (g)
138312	2	9840000983	24-10-25	36:00	11
133370	2	9840000932	23-10-25	05:40	2

Figur 12 - Oversigt over tabellen Visit Data

Visit Data bestod af over 37.000 unikke rækker, og vi havde derfor brug for en effektiv metode til at importere data. Det løste vi ved at anvende et batch insert, hvor man indsætter mange rækker på én gang, i stedet for én ad gangen. Det gjorde processen markant hurtigere – en opgave, der ellers kunne have taget flere timer, blev nu gennemført på få sekunder.

Da projektets krav var at data skulle importeres fra et Excel-ark, blev selve databaseopbygningen et vigtigt diskussionsemne. Ud fra vores viden om databasedesign overvejede vi at opbygge databasen efter principperne i 3. normalform (3NF). Det indebærer at data ikke må forekomme unødig flere gange, og at der ikke må være transitive afhængigheder mellem attributter.

Dog viste det sig, at datasættet fra Agrisys ikke var struktureret i 3NF. Det rejste spørgsmålet om vi skulle normalisere data, eller beholde den oprindelige struktur. I første omgang planlagde vi at følge en streng 3NF-struktur (hvor man sikrer sig mod redundans og afhængigheder), men i løbet af planlægningen opdagede vi at der ikke eksisterede nogen reel relation mellem data i *Animal Data* og *Visit Data* – udover at *Responder*, *Location* og *Number* gik igen.

Der var fx ingen sammenhæng mellem *CompletedDaysInTest* i *Animal Data* og antallet af dagsregistreringer per gris i *Visit Data*. Samtidig var der heller ikke noget sammenfald mellem *TotalFeedIntake* i *Animal Data* og summen af *FeedAmount* fra *Visit Data*. Derfor måtte vi gentænke vores databaseopbygning, da det ikke gav mening at forsøge at skabe relationer (*foreign keys*) mellem tabellerne, når datagrundlaget ikke understøttede det.

I stedet fokuserede vi på at skabe så konsistente og rene tabeller som muligt:

I **Animal Data** fjernede vi kolonnen Number, da den var transitivt afhængig af *Responder*, som vi i stedet gjorde til primærnøgle (PK). Vi fjernede også FCR, da den værdi kunne beregnes ud fra WeightGain og TotalFeedIntake, og derfor også var transitivt afhængig. WeightGain var derudover også transitivt afhængig af StartWeight og EndWeight, så derfor fjernede vi også den kolonne fra databasen.

Tabel 1: Animal Data

Location	Responder	Sex	Start weight (kg)	Total feed intake (kg)	End weight (kg)	Completed Days In Test
2	984000009	Female	29,5	7,3	41,8	5

Figur 13 - Opdateret Animal Data tabel

I **Visit Data** tilføjede vi en ny kolonne *VisitID* som primærnøgle og fjernede *Number*, da den igen var transitivt afhængig af *Responder*. Responder kan ikke selv bruges som primærnøgle i Visit Data, fordi hver gris optræder mange gange.

Table 2: Visit Data

Visit ID	Location	Responder	Date	VisitDate	Feed amount (g)
1	Location	9840000983	28-10	36:00	11
2	Reponter	9840000093	28-10	05:40	2

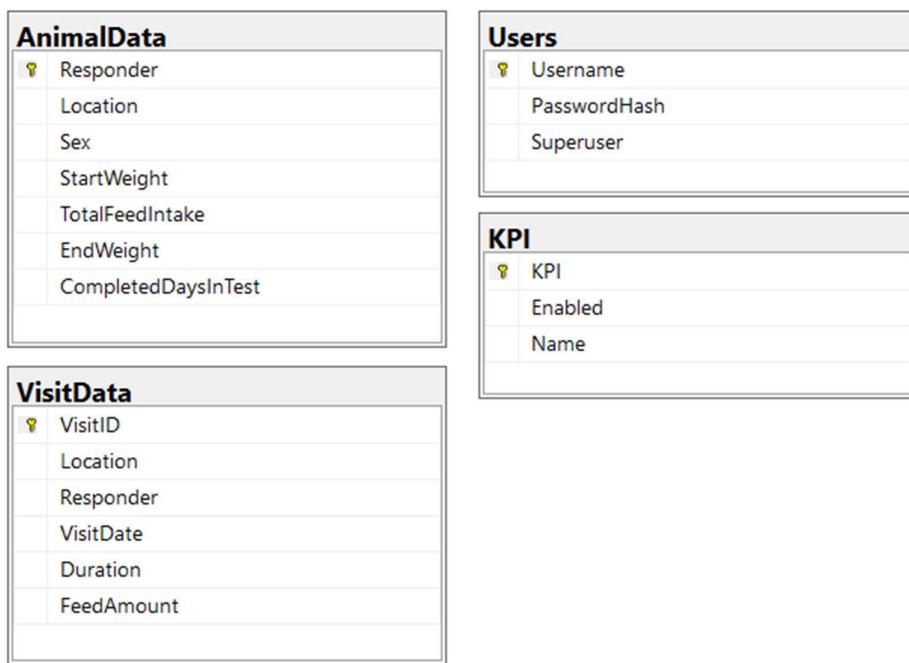
Figur 14 - Opdateret Visit Data tabel

Udover data fra Agrisys tilføjede vi to ekstra tabeller til vores database: Users og KPI.

Users tjener det overordnede formål at muliggøre autentificering og autorisation i systemet. Den indeholder essentielle informationer om hver enkelt bruger, herunder brugernavn og en adgangskode repræsenteret ved en kryptografisk hash samt en markør for, hvorvidt brugeren er tildelt superbrugerstatus. Det muliggør en differentieret adgangskontrol, hvor visse funktioner og administrative rettigheder udelukkende er tilgængelige for brugere med udvidede privilegier. Anvendelsen af hash-funktioner til

lagring af adgangskoder er i overensstemmelse med gældende sikkerhedsstandarder for projektet og bidrager til beskyttelsen af logindata.

Tabellen KPI (Key Performance Indicators) har til formål at definere og administrere de nøgletal, vi har udvalgt som indikatorer for grisenes performance og udvikling i de tildelte data fra Agrisys. Hver post i tabellen repræsenterer en enkelt KPI, der er karakteriseret med et navn samt en binær aktiveringsstatus (Enabled), der angiver, om indikatoren aktuelt er i brug. Ved at strukturere KPI'er i en særskilt tabel sikres en høj grad af fleksibilitet og skalerbarhed i systemets analysemaessige dimension. Det muliggør at relevante nøgletal nemt kan aktiveres eller deaktiveres efter behov, uden at det kræver strukturelle ændringer i den øvrige datamodel. Tabellen understøtter dermed en dynamisk og konfigurerbar tilgang til dataevaluering, hvilket er centralt for vores projekt, da fremvisning af måleparametre skal kunne justeres af superbrugerne gennem vores GUI.



Figur 15 - ER-Diagram for databasen

Det er bemærkelsesværdigt, at der i den nuværende datamodel ikke er defineret nogen eksplisit relationer mellem tabellerne. Fraværet af fremmednøgler, som danner grundlaget for referentiell integritet i relationelle databaser, betyder at sammenhænge mellem data udelukkende er implicitte og ikke håndhæves strukturelt af databasen. Efter at vi blev bekendte med indholdet af vores data fra Agrisys blev en ukonventionel opbygning dog godkendt af gruppen, da vi, som tidligere forklaret, ikke fandt nogen datarelation mellem tabellerne, som synes nødvendige at strukturere i SQL. I relation til

tredje normalform (3NF) kan det overvejes, om attributten Location bør placeres i en separat Responder-Location-tabel. Det udspringer af at hver Responder i det nuværende datasæt kun er knyttet til én Location, mens en Location godt kan indeholde flere Responders. I 3NF-sprog er det altså en en-til-mange-relation.

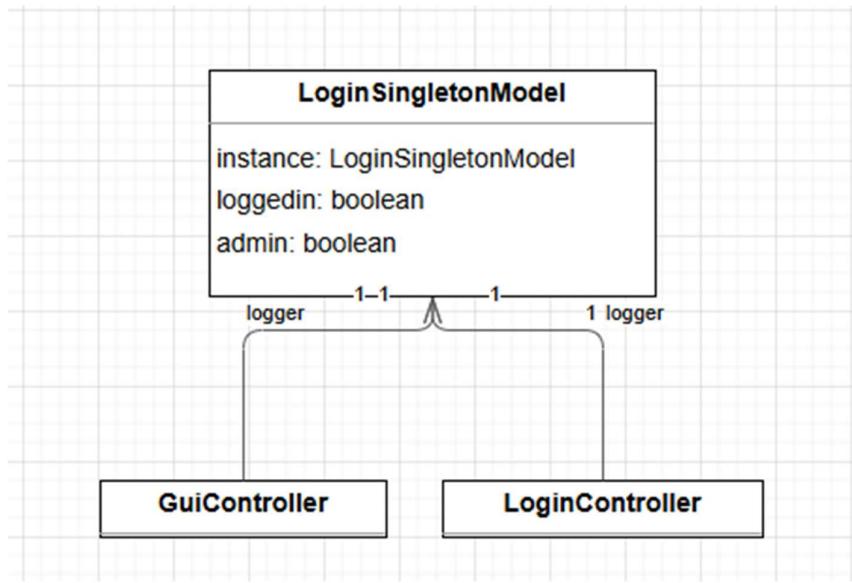
Det er dog vigtigt at understrege, at vores datasæt kun repræsenterer et udsnit af en større datamængde. Animal Data-tabellen dækker udelukkende et testudsnit, hvor dyr er blevet vejet og får beregnet FCR, mens Visit Data-tabellen indeholder kontinuerligt indsamlede foderdata på daglig basis. Den begrænsning i datagrundlaget gør det usikkert, om relationen mellem Responder og Location reelt er statisk. Hvis én Responder på et tidspunkt flyttes til en ny Location, vil den foreslædede normaliserede struktur blive brudt. Det ville medføre enten en kompleks omstrukturering af databasen eller – endnu værre – en rigid arkitektur, hvor en landmand ikke let ville kunne ændre en gris' placering uden at støde på tekniske begrænsninger. Derfor vurderer vi at gentagelsen af Location i både Animal Data og Visit Data ikke nødvendigvis er et brud med god databasemodellering. I stedet ser vi det som en praktisk løsning, hvor tabellerne tjener forskellige formål og har deres egne kontekster.

Design Patterns

Vi besluttede at bruge en række mønstre i designet af koden og i det her afsnit gennemgår vi dem og illustrerer med simple klassediagrammer. Mønstrene er: Singleton, Builder, Simple Factory, MVC, og Facade.

Singleton

Login skal foregå i et separat vindue, og her skal brugernavn og kodeord kunne verificeres. Det sker i klassen LoginSingletonModel – som anvender statiske metoder fra klassen PasswordUtils for at kontakte databasen og lave de relevante tjek. Hvis man har adgang, så åbnes det egentlige vindue styret af GuiController – men nu skal GuiController jo vide om det er en administrator eller en almindelig bruger der har logget sig på (fordi kun en administrator kan ændre på hvilke grafer der vises). Det løses ved at både LoginController og GuiController har den samme instans af LoginSingletonModel-klassen. Så kan GuiController tjekke om en boolean i LoginSingletonModel er sand eller falsk, og så ved man hvilken slags bruger der er logget ind.



Figur 16 - Singleton-mønsteret

Der konstrueres kun en instans af LoginSingletonModel én gang – nemlig når man aktiverer login-vinduet. Det egentlige GUI-vindue har samme instans, men opretter ikke en ny (det er netop pointen med Singleton-mønsteret).

Builder

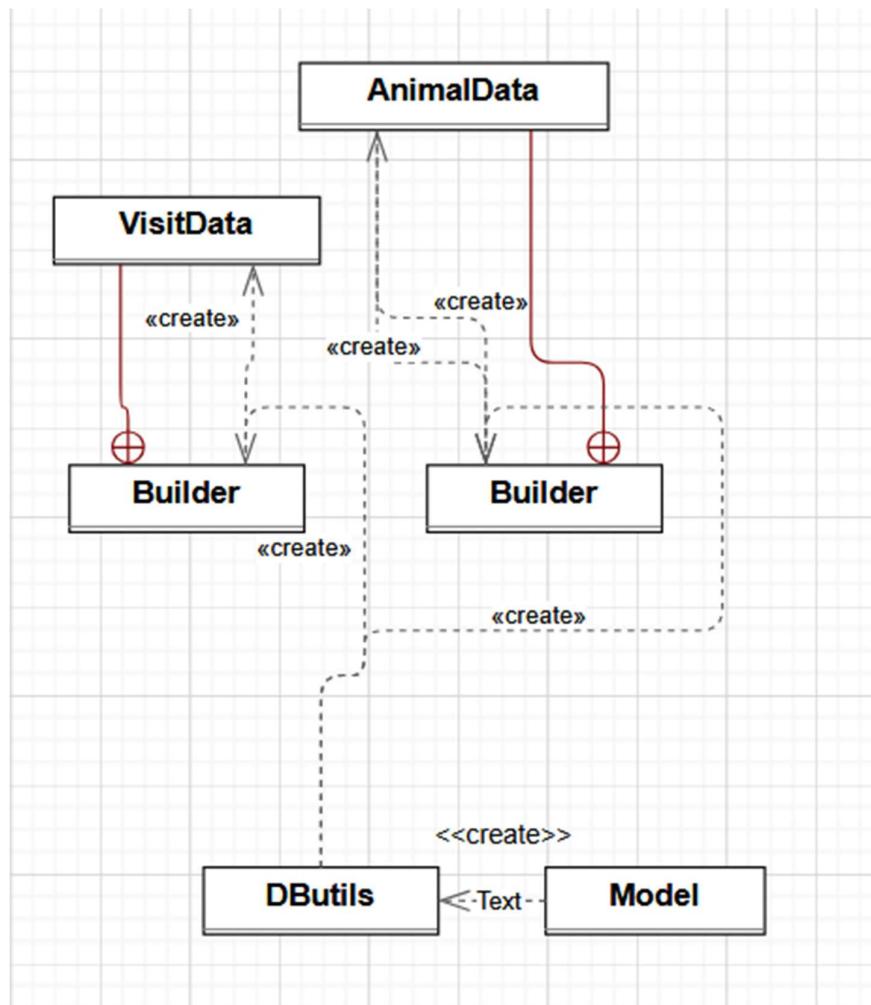
Under planlægningsfasen vurderede vi oprindeligt, at KPI-objekterne kunne konstrueres ved hjælp af specialiserede og overloadede konstruktører. Med specialiserede konstruktører menes der, at konstruktørerne i klasserne AnimalData og VisitData, kun ville indeholde præcis de variable som et givent objekt behøvede. Det virkede under planlægningen hensigtsmæssigt, da vi kendte den fremgangsmåde fra vores forrige projekt, og vi samtidig vidste hvad hvert KPI-objekt skulle indeholde af attributter hentet fra databasen. Senere i planlægningsfasen ændrede vi dog holdning, da vi besluttede at alle beregninger samt sortering og filtrering skulle foregå i SQL. Det medførte et øget antal parametre og større kompleksitet i objektopbygningen.

Det blev hurtigt tydeligt, at KPI-objekterne havde mange fællestræk, men også en vis variation i den data, der skulle hentes fra databasen. Mange KPI'er delte et fælles sæt af primitive typer (særligt long, int og double), men skulle stadig levere særlige udtræk fra databasen.

Den kompleksitet gjorde den planlagte tilgang med overloadede og specialiserede konstruktører uhensigtsmæssig, da Java ikke længere kunne kende forskel på konstruktørerne. Problemet opstod når konstruktørerne skulle deles om ens primitive typer. Fx ville en 'long' type ofte optræde, da mange af vores KPI'er henviser til en responder. Samme long ville derfor flere gange blive forbundet med en double eller en int, og det ville skabe tilpas meget lighed i de specialiserede konstruktører, at Java ikke

længere kunne skelne den ene konstruktører fra den anden. Derfor kunne vi udelukke den fremgangsmåde som en valid metode til at opbygge vores KPI-objekter med.

Et andet alternativ til vores builderpattern kunne også være oprettelse af mange modelklasser, der hver især ville indeholde én konstruktør, hvilket ville gøre den overordnede kodenestruktur vanskelig at overskue. Et tredje alternativ ville være at oprette én meget lang konstruktør i hhv. AnimalData og VisitData klasserne med alle instansvariabler som parametre. Det ville dog føre til lav læsbarhed og mange null-værdier i DButils metoderne, hvor KPI'ernes stored procedures kaldes fra. I en sådan 'samlende' konstruktør ville vi skulle indsætte alle parametre vi ville kunne ønske fra vores data, og derefter aktivt fravælge nogle af dem, når et objekt skulle opbygges i DButils.



Figur 17 - Builder pattern design

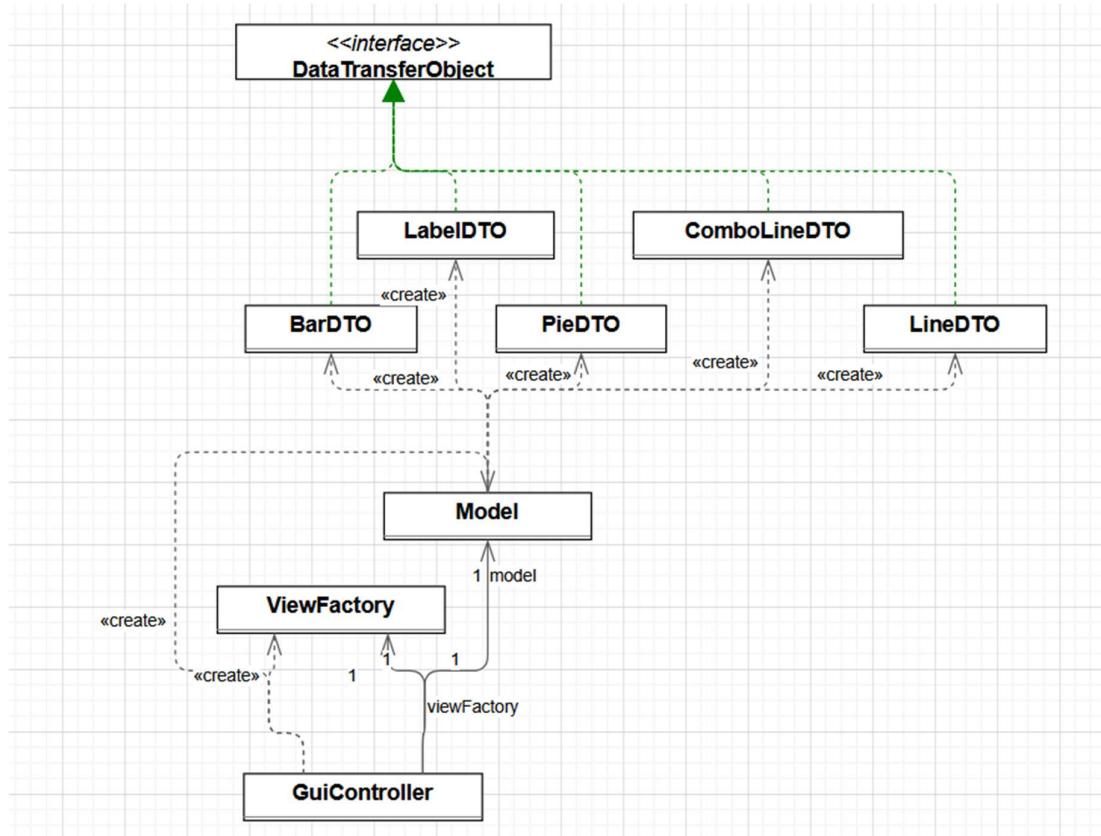
Ovenstående diagram illustrerer vores implementering af Builder Pattern, hvor klasserne VisitData og AnimalData hver især indeholder en intern Builder, som bruges til at konstruere deres respektive objekter. Builder-klassen samler de nødvendige

parametre til at opbygge objekter fra databasen. Som et sæt legoklodser gør builderen det muligt at skabe objekter med stor lighed, men samtidig tilpasse dem individuelt i DButils ved blot at angive de relevante værdier.

Vi valgte at anvende Builder Pattern til konstruktionen af KPI-objekterne, da det gjorde det muligt at opbygge dem trinvist og fleksibelt. Tilgangen har øget både læsbarheden og genanvendeligheden af koden og har vist sig særligt velegnet i en kontekst, hvor objekterne deler primitive typer, men kræver individuelle konfigurationer. Derudover har dette mønster været meget anvendeligt under implementeringen, da det er meget nemt at tilføje ‘legoklodser’ til builderen, når vi så behovet for flere KPI’er.

Simple Factory

De forskellige KPIer skal visualiseres på en række forskellige måder, og for at løse det problem bruger vi et DataTransferObject-interface som så implementeres af de forskellige DTOer vi har brug for. Model-klassen har så funktioner til at levere hver enkelt KPI. Funktioner som GuiController (der har en instans af Model) så kan kalde for at få de data der skal vises.

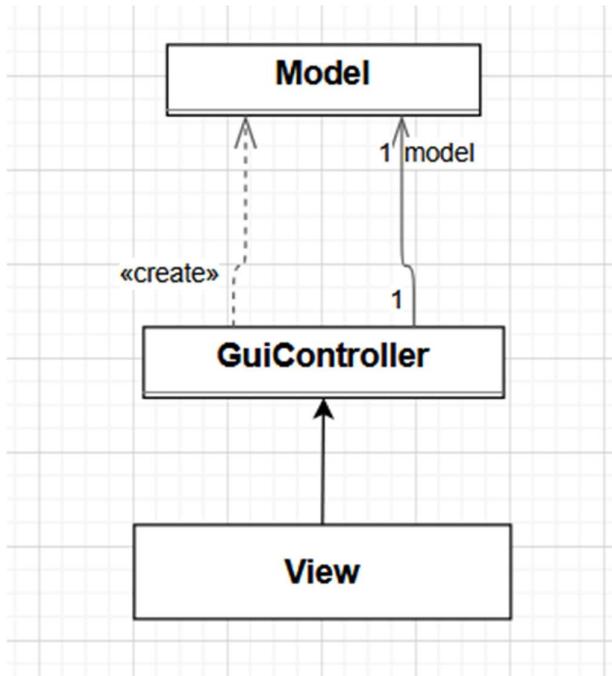


Figur 18 - Klassediagram for DTOer der sendes fra model til GUI

Grunden til at det er smart at bruge et DataTransferObject-interface er at GuiController så kan tage ethvert element der kommer fra Model og sende det ind i sin ViewFactory og så sørger den for resten. Resten er at skabe det relevante plot, eller label og returnere det som en Node – som GuiController-klassen så kan vise. ViewFactory har bare én public metode, `createView()`, som netop tager et DataTransferObject som input. Kodden er et eksempel på det mønster der kaldes Simple Factory.

Model-View-Controller

Mønsteret er ret simpelt og det handler egentlig bare om at holde visningen af data og opbevaring og manipulation af data adskilt. Vores egentlige view er fxml-filen i javafx-projektet, gui-view.fxml, og kontrollen af hvad der sker når brugeren trykker på knapper, rykker med scrollbars osv, er styret af GuiController-klassen.



Figur 19 - Model-view-controller design

GuiController har så en instans af klassen Model – og det er i Model at al interaktion med databasen foregår, og at al processering af data foregår. Resultatet er at View er helt adskilt fra Model, og begge interagerer med Controller-klassen.

Vi har også en login-view.fxml fil og her er der tilsvarende et MVC-mønster, men nu med LoginController og LoginSingletonModel som *Controller* og *Model*.

Der var imidlertid et sted i designet hvor vi delvist bryder med MVC-mønsteret, nemlig med klassen ComboChart. Problemet den klasse løser er at vi har tre grafer hvor vi lader brugeren selv vælge én specifik gris eller foderstation, og så viser vi grafen for den valgte. Det kan løses på to måder: Enten skal vi hente data fra databasen hver gang man vælger en ny gris/lokation, eller også skal vi hente data én gang for alle og så lade klassen opbevare data selv. Vi valgte den sidste løsning for at reducere antallet af kald til databasen, men det betyder jo at ComboChart-klassen både er en slags graf (view), reagerer på brugerinput (controller) og har data (model) – så ComboChart er 3-i-en.

Facade

Man kan diskutere om vi egentlig bruger et vaskeægte Facade-mønster – men der er en række KPI'er som bare skal vises som et navn og et tal, fx "Number of pigs 343". På GULens forside viser vi ni af dem. Her kunne GuiController kalde en funktion i modellen for hver KPI – men i stedet implementerer Model-klassen et simplere interface for GuiController, hvor man bare kan kalde én funktion som så leverer alle de relevante KPI'er i en liste.

```
//KeyNumbers Metode
public List<DataTransferObject> getKeyNumbers(){ 1 usage  ↳ Mfarsoe *
    List<DataTransferObject> keyNumbers = new ArrayList<>();
    keyNumbers.add(getAvgFcrDTO());
    keyNumbers.add(getAvgDurationDTO());
    keyNumbers.add(getPigCountDTO());
    keyNumbers.add getLocationCountDTO();
    keyNumbers.add getTotalFeedDTO();
    keyNumbers.add getPositiveFCRDTO();
    keyNumbers.add getNegativeFCRDTO();
    keyNumbers.add getAvgWeightDTO();
    keyNumbers.add getPigsAboveWeightDTO();
    return keyNumbers;
}
```

Figur 20 - Facade-funktionen getKeyNumbers()

Ideen med Facade-mønsteret er netop at skjule en underliggende kompleksitet for klient-klassen i tilfælde hvor klienten (her GuiController) ikke behøver at have viden om

den kompleksitet. Det er derfor vi kan sige at vores funktion `getKeyNumbers()` implementerer Facade-mønsteret (selvom vi ikke bruger en separat Facade-klasse).

5. Implementering

(Martin)

Som beskrevet i vores metodeafsnit vil vi under implementeringsfasen gå over til at benytte Scrum. Scrum giver os en fleksibel men struktureret ramme, som gjorde det muligt at arbejde målrettet og reagere hurtigt på ændring og behov.

Sprintstruktur

Vi arbejdede i korte, fokuserede sprint, hvor hvert sprint havde en tidsramme og klart definerede mål. Martin fik rollen som SM (Scrum Master) og fik til opgave at organisere hvert sprint, samt at fungere som facilitator i vores daglige standups. Vores udviklerhold bestod af alle medlemmer af gruppen, da vores størrelse ikke understøtter en SM som ikke også indgår i udviklerholdet. Da projektet udføres som en del af en eksamensopgave, har vi valgt ikke at udpege en traditionel Product Owner. I stedet fungerer alle gruppemedlemmer som fælles product owners, da vi alle har en lige stor interesse i projektets succes og beståelse af eksamen.

Hvert sprint bestod af følgende faser (se bilag B):

- Sprint Planning – Her blev sprintmål definerede og arbejdsopgaver blev udvalgt fra den overordnede opgavemængde eller backlog. Arbejdsopgaverne blev konkretiseret og forventninger til hinandens arbejdsindsats blev afstemt.
- Daily Standup – Vi planlagde daglige standups kl. 14.00, men valgte at afholde dem fleksibelt efter behov og tilgængelighed. De standups fungerede som korte statusmøder, hvor vi drøftede fremdrift, forhindringer og koordinerede med hinandens opgaver og flyttede prioriteringer, så arbejdet ikke blev blokeret.
- Udviklingsarbejde – Selve udviklingsarbejdet blev organiseret gennem et Kanban-board, som fungerede som det centrale planlægningsværktøj. Det gav os et visuelt overblik over opgavernes status og hjalp os med at synliggøre fremdrift og testparathed.
- Sprint Review og Retrospective – Ved afslutningen af hvert sprint evaluerede vi resultaterne og gennemgik hvilke opgaver der var løst, hvilke der udestod, samt hvordan vi som team har fungeret. Review-delen fokuserede på produktet, men retrospective fokuserede på processen og samarbejdet.

Scrum-forløb: Recap af Sprint 1-3

Sprint 1 – 05/05/2025 - 09/05/2025

Første sprint havde til formål at etablere de centrale rammer for projektet. Herunder udviklingen af den grundlæggende GUI-struktur, login-funktionalitet, ViewFactory, import af data og opbygningen af databaseforbindelser, stored procedures og model.

Sprintet blev sat i gang med 71 opgaver på Kanban-boardet. Fokus var på en god opstart af projektet, arbejdsstruktur i form af brugen af Kanban og kodeprocessen generelt. For ikke at stå stille i udviklingen, holdt vi daglige standups efter behov, hvilket gav os fleksibilitet uden at miste overblik. Arbejdsopgaverne blev fordelt ligeligt, og der var en god dynamik i teamet. For at undgå blokeringer, fokuserede vi først på databasedesignet og de udvalgte KPI'ers stored procedures.

Resultater for sprintet:

- 66 opgaver blev færdiggjort
- 5 opgaver blev overført til backloggen
- Skabt solidt grundlag for udviklingsfasen
- Forøget forståelse for brugen af Kanban og Scrum-metoden

Teamet arbejdede effektivt og med god energi. Samarbejdet var præget af en positiv og åben tone. Dog var der en udfordring med manglende opdatering af Kanban-boardet, hvilket førte til forvirring omkring opgavestatus. Det blev identificeret i vores sprint review og vi aftalte at prioritere løbende opdateringer i fremtidige sprint.

Sprint 2 – 12/05/2025 - 16/05/2025

Formålet med sprint 2 var at videreudvikle KPI-funktionaliteten og tilføje flere widgets og KPI'er. Der blev i alt planlagt yderligere 32 arbejdsopgaver, herunder KPI'er 18-26, scroll-funktion i ViewFactory samt funktioner til import af Excel og eksport til CSV.

På trods af fravær hos Henrik midt i sprintet blev arbejdsopgaver løst effektivt ved fleksibel planlægning. Daglige standups blev gennemført og én dag blev afsat til systematisk test. Arbejdet med KPI'er og GUI fortsatte og nye DTO'er blev introduceret til at understøtte nye visuelle visninger af KPI'er i form af AreaDTO og Combo-visninger.

Resultater for sprintet:

- 32 opgaver blev færdiggjort
- 3 opgaver blev overført til backloggen
- Funktionalitet af programmet blev testet
- Størstedelen af projektet blev vurderet færdigt

Der var god fremdrift og øget bevisethed om Kanban-boardets værdi. Løbende blev det opdateret, hvilket gav klart overblik over status og testklar funktionalitet. En væsentlig

læring var dog testprocessens svagheder, da næsten 60 opgaver endte med at være klar til test samtidigt. Det belastede testprocessen og øgede risiko for fejl. Vi aftalte derfor at oprette reelle testcases og gennemføre test med forventede resultater i et mere struktureret forløb.

Sprint 3 – 19/05/2025 - 20/05/2025

Sprint 3 var et kort sprint med målet at færdiggøre de sidste opgaver fra backloggen. Opgaverne bestod primært af import/export-funktionalitet, som var påbegyndt i sprint 2.

Sprintet omfattede kun tre opgaver, men blev behandlet med samme struktur og fokus som tidligere sprints. Johannes havde delvist fravær, men det blev håndteret uden problemer. Ét dagligt standup blev det til i sprintet, og fungerede som et status-checkup for at sikre at tidsplanen ville blive overholdt.

Resultater for sprintet:

- Alle 3 backlog-opgaver blev løst
- Implementeringsfasen blev officielt afsluttet, og backloggen var tom.

Sprint 3 viste, at vores tilgang til projektet havde vist sig at være effektivt og at alle opgaver blev løst rettidigt. Fokus skiftede herefter naturligt over mod rapportskrivning og dokumentation. Der blev ikke identificeret yderligere forbedringer, da sprintet var både kort og præcist gennemført.

6. Afprøvning

(Martin)

Ud fra vores krav til programmet kan vi opdele vores test i 4 grupper: Bruger, View, Input, Output (se bilag xx). Bruger dækker over brugerhåndtering og sikkerhed. View dækker over alt det visuelle som brugeren vil blive præsenteret for. Herunder indgår der de forskellige widgets (KPI'er) som bliver vist på forskellig vis. Input er al interaktion med programmet, hvor brugeren har valg de skal foretage, eller at programmet afventer input fra brugeren, før en handling kan udføres. Til slut har vi output som dækker over de forskellige output fra programmet.

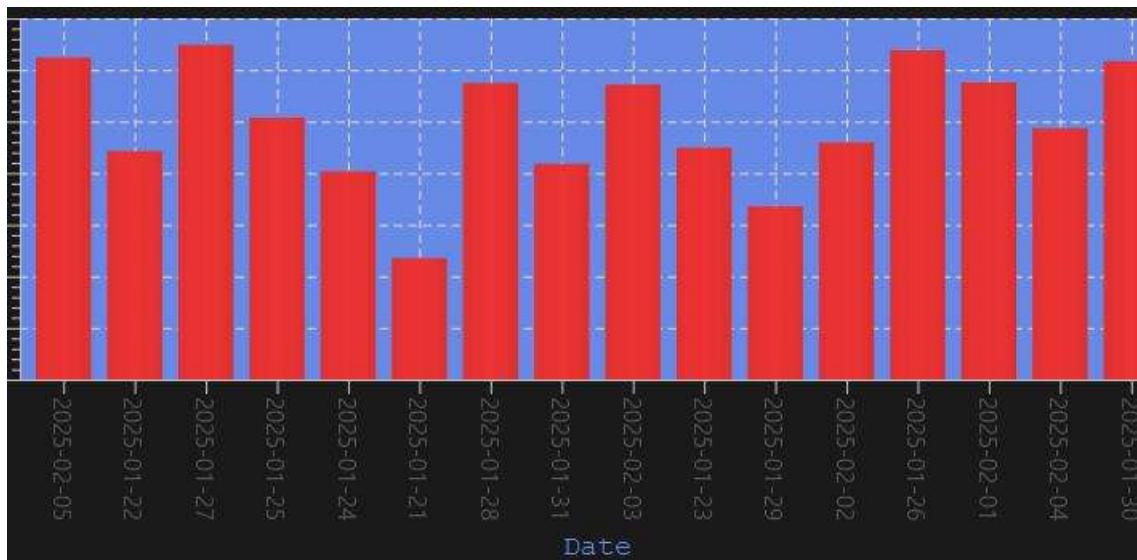
Vi har igennem vores netværk fået en certificeret tester fra Nuuday, til at teste vores software. Der er blevet udført erfaringsbaserede testteknikker som udforskende test og checkliste baseret test ud fra vores systemkrav. Alle test blev udført d. 22/05/2025.

Testcheckliste

Checklisten er udarbejdet ud fra den kravliste der er blevet oprettet i forbindelse med vores test og dækker over alle de krav og forventninger der er til vores software.

Testcase	Testtrin	Forventet resultat	Resultat
Login med admin-bruger	Vælg 'admin' og indtast korrekt kodeord	Login lykkes og admin-view vises	✓
Login med almindelig bruger	Vælg 'bruger' og indtast korrekt kodeord	Login lykkes og bruger-view vises	✓
Forkert kodeord	Vælg bruger og indtast forkert kodeord	Fejlmeddeelse vises: 'Access Denied'	✓
Korrekt kodeord matcher kun én bruger	Indtast korrekt kodeord for 'admin' men vælg 'bruger'	Fejlmeddeelse: 'Access Denied'	✓
Manglende valg af bruger	Indtast kodeord uden at vælge bruger	Fejlmeddeelse: 'No user chosen'	✓
Brugerdata er hashed	Tjek lagret kodeord i database	Kodeord gemt som hash (bcrypt)	✓
Log af	Klik 'Log ud'	Login-skærm vises og session afsluttes	✓
Admin ser Settings, bruger gør ikke	Log ind som admin og derefter som bruger	Settings kun synlig for admin	✓
Data vises i widgets	Log ind og kontroller visning af widgets	Data vises korrekt	✓
Feedback ved knaptryk	Klik på knapper i UI	Visuel feedback vises	✓
Kodeord skjult	Tjek kodeordsfelt ved login	Indtastning vises som ****	✓
Vælg bruger fra liste	Klik på bruger-drop-down og vælg bruger	Bruger vælges korrekt	✓
Indtast kodeord	Indtast kodeord i feltet	Input accepteres og skjules	✓
Import af Excel-fil	Klik 'Importer' og vælg gyldig fil	Data importeres til systemet	✓
Admin ændrer view	Log ind som admin og skift visning	View opdateres	✓
View bevares efter brugerskift	Skift view, log af og ind igen	Samme view vises efter login	✓
Skift mellem faner	Klik på forskellige faner	View ændres i henhold til valgt fane	✓
Eksporter CSV med KeyNumbers og Alarm-data	Klik 'Export' og vælg CSV	CSV downloades med korrekte data	✓
Import til database via program	Importer datafil via program	Data lagres i database korrekt	✓

De fundne fejl var i forbindelse med læsbarhed og korrektur. Der var én lidt mere kritisk fejl, i form af forkert sorteret data.



Figur 21 - KPI 10 før korrektion

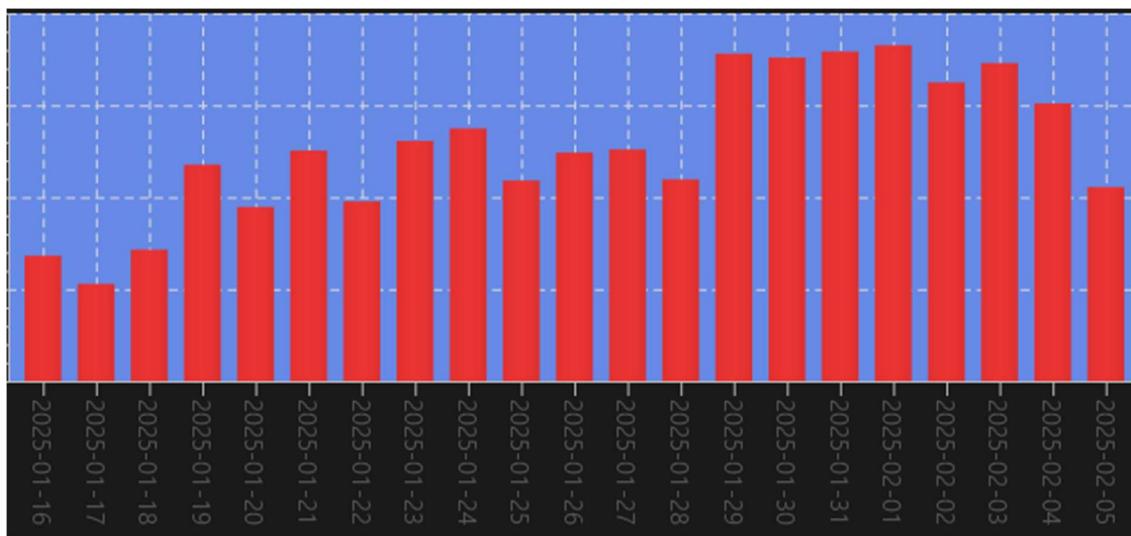
Data fremstår korrekt, men er ikke sorteret efter dato. Eftersom alle vores KPI'er bliver bygget på samme måde, identificerede vi at problemet måtte være fra den stored procedure der henter data.

Fejlliste

Fejl	Beskrivelse	Løsning	Priority
Forkert Sorteret KPI 10	Datoerne fremstår ikke kronologisk.	Ændres i stored procedure i database	Medium
Keynumbers skrift	Average feed duration står med stort.	Ændres i database	Low
Pig Overview	Distribution skal med småt i KPI 18 og KPI 19	Ændres i model	Low
FCR per pig	Feed Consumption Rate skal forkortes til FCR	Ændres i model	Low
Avg feed per visit per day per pig	Avg feed amount skal med stort	Ændres i model	Low
Manglende dataentries i widgets	Data fremstår ikke fuldkommen i view – Det sker ved resizing af vinduet.	ViewFactory bør justeres så data ikke forsvinder	Low

Vi har i alt seks defekter som alle er visuelle fejl og mangler. Vi vurderer at vi laver ændringerne til database og model, men vi ikke påvirker ViewFactory så sent i vores eksamsprojekt, da der potentielt kan opstå andre ændringsrelaterede fejl. Ved eventuel vedligeholdelse af et reelt forløb, ville man prioritere det og finde en løsning.

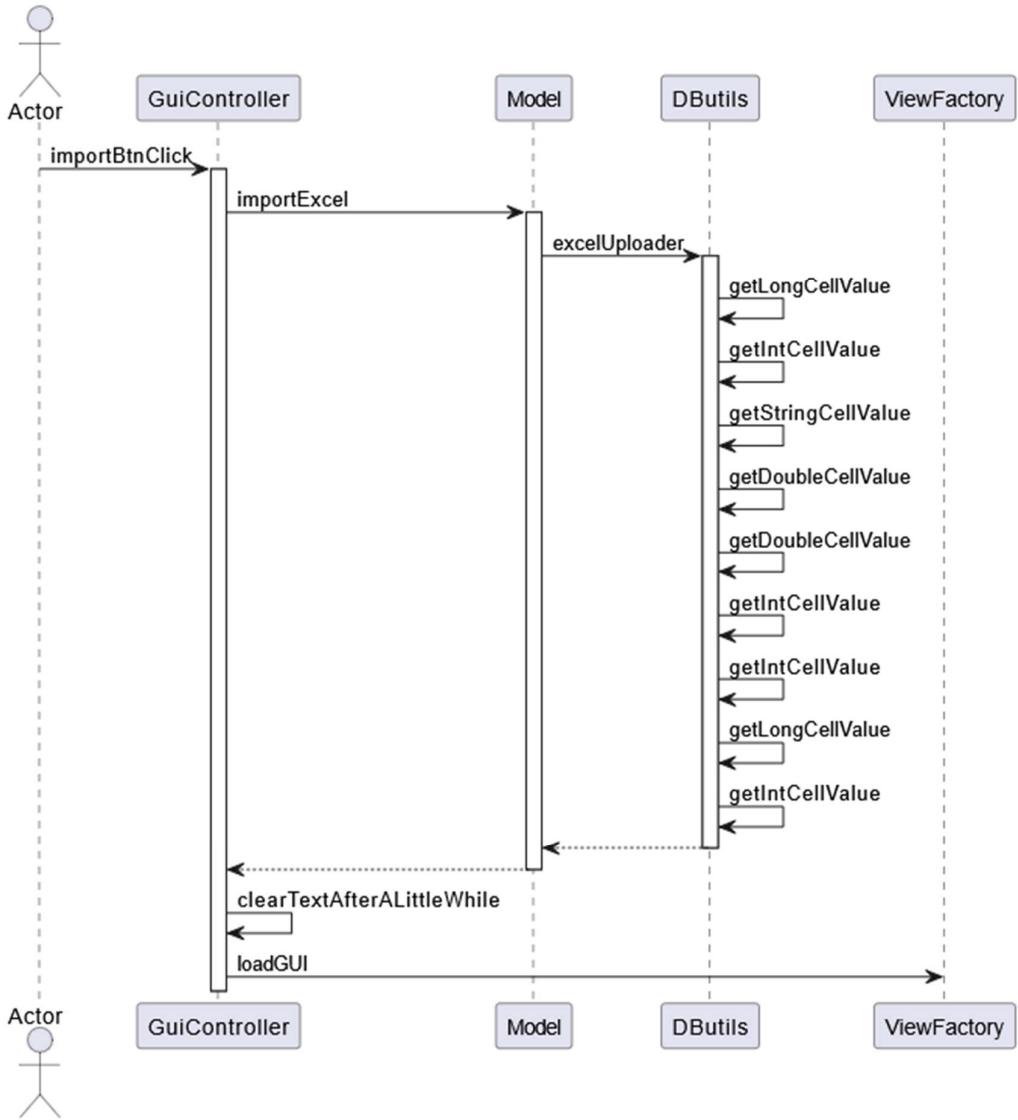
Ved vores mere kritiske fejl med det fejlsorterede data, fandt vi frem til at data, blev sorteret forkert når den blev hentet ned. Så vi ændrede den pågældende stored procedure til at sortere efter visitdato og problemet blev løst.



Figur 22 - KPI 10 efter korrektion

Særligt komplekse funktioner

Import af Excel-fil er en lidt mere kompleks operation end de andre fordi formatet Excel ikke er direkte understøttet af JDK. I stedet bruges der Apache API, som muliggør læsningen af de individuelle cellers værdier. Sekvensdiagrammet viser hvordan den kalder sig selv 9 gange, for at kunne oversætte hver celle til variabler der kan bruges i den stored procedure der kaldes.



Figur 23 - Sekvensdiagram over importExcel()

DButils forholder sig kun til resultatet af processen og returnerer derfor kun en boolsk værdi tilbage til modellen. Modellen tager den boolske værdi og sender en tekststreg til Controlleren, alt efter hvilket resultat blev returneret fra DBUtils. Til slut kaldes der en service metode clearTextAfterALittleWhile(), som renser labeltekst, og loadGUI() som henter alle værdier ned og opbygger vores grafiske brugerflade igen.

Under vores checklistetest, indgik der test af import fra Excel og det forløb uden nogen problemer.

7. Konklusion og reflektion

(Henrik, Martin)

Vores Scrum-tilgang blev ikke blot et styringsværktøj, men også en løbende læringsproces, hvor både produkt og proces blev forbedret fra sprint til sprint. Hvert sprint afsluttedes med en evaluering, hvor vi identificerede styrker og forbedringsområder. Reflektionen blev aktivt brugt til at justere vores arbejdsmetode i de efterfølgende sprints.

Vi har med stor fordel anvendt en hybrid tilgang mellem vandfaldsmodellen og Scrum, hvilket har bidraget til en strømliniet og effektiv udviklingsproces. Den grundige planlægning i projektets indledende fase – inspireret af vandfaldsmodellen – har vist sig særdeles værdifuld, da vores oprindelige idéer og struktur i høj grad har holdt stik gennem hele forløbet. I implementeringsfasen har vi taget udgangspunkt i Scrum, hvor de lidt friere rammer og løbende sprint-forløb har givet os mulighed for agilt at angribe opståede problemer og hurtigt finde passende løsninger. Det har gjort os i stand til at tilpasse og forbedre systemet løbende uden at miste overblikket over helheden. Som et konkret eksempel indførte vi et par nye graftyper i implementeringsfasen og ændrede visualiseringen af flere KPI'er.

Undervejs har vi været opmærksomme på potentielle udfordringer og har formået at identificere og løse problemer tidligt, hvilket har styrket projektets samlede kvalitet. Vores ønsker til database voksede mere end først forventet, hvilket betød, at en stor del af vores logik blev placeret her – særligt i form af stored procedures. Det har dog vist sig at være en effektiv løsning i forhold til systemets performance og vedligeholdelse. Samtidig har vi haft stort fokus på at etablere en solid og gennemtænkt arkitektur med udgangspunkt i et godt MVC-design inden for rammerne af vores 3-lagsmodel. Vores tidlige erfaringer og tekniske forståelse fra uddannelsen har været afgørende for, at vi har kunnet udvikle en funktionel og brugervenlig platform, som lever op til projektets målsætninger.

Arbejdsprocessen har desuden været præget af en god og konstruktiv stemning, hvor der var plads til både faglig sparring og uformel dialog. Vi har prioriteret at skabe tid til snak og godt humør, hvilket har været med til at opretholde en positiv arbejdskultur gennem hele projektet. Den gode tone og det gode samarbejde har haft stor betydning for både trivsel og produktivitet, og har styrket vores evne til at arbejde sammen som et effektivt team, hvor alle har løftet i flok for at skabe et tilfredsstillende resultat.

Vores program opfylder alle opstillede krav i casen med undtagelse af bonusopgaven omkring simulering af drift. Systemet kan læse data fra en Excel-fil og gemme dem i en MSSQL-database, hvilket opfylder kravene til datahåndtering. Brugerrollerne er implementeret med adskillelse mellem superbruger (administrator) og almindelig

bruger, hvor superbrugeren har adgang til at oprette og tilpasse dashboards og KPI-widgets, mens den almindelige bruger kan tilgå og overvåge grisenes fodringstilstand.

Dashboards og KPI-widgets er realiseret ved brug af JavaFX, hvor fodringsmønstre og vægtøgning visualiseres. Derudover indeholder systemet en advarselsfunktion, som registrerer og informerer brugeren, hvis en gris har spist under et defineret minimum de seneste tre dage, og brugeren kan se en samlet liste over alle advarsler i GULen.

Endelig indeholder systemet også import/eksportfunktioner, der gør det muligt at importere data direkte fra Excel og gemme analyser som CSV-filer.

8. Bibliografi

- [1] »atlassian.com,« 2025. [Online]. Available: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>.
- [2] »scmgalaxy.com,« 2025. [Online]. Available: <https://www.scmgalaxy.com/tutorials/a-successful-git-branching-model>.
- [3] A. Adetokunbo og B. A. Adenowo, »Software Engineering Methodologies: A Review of the Waterfall Model and ObjectOriented Approach,« *International Journal of Scientific & Engineering Research*, Juli 2013.
- [4] B. Boehm og R. Turner, »Using risk to balance agile and plan-driven methods,« *Computer*, pp. 57-66, Juni 2003.
- [5] »The Waterfall Model with Agile Scrum as the Hybrid Agile Model for the Software Engineering Team,« *10th International Conference on Cyber and IT Service Management (CITSM)*, 2022.
- [6] K. Schwaber og J. Sutherland, »The Scrum Guide. Scrum.org.,« 2020. [Online]. Available: <https://Scrumguides.org/Scrum-guide.html>.
- [7] »scrum.org,« 2025. [Online]. Available: <https://www.scrum.org/resources/what-scrum-module>.
- [8] J. Dalton, »Kanban Board,« i *Great Big Agile*, Berkeley, CA, Apress, 2019.
- [9] »agilebusiness.org,« 2025. [Online]. Available: <https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html>.
- [10] R. Martin, UML for Java Programmers, New Jersey: Prentice Hall, 2003.
- [11] »learn.microsoft.com,« 2025. [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>.
- [12] H. Christensen, »Klient-server og tre-lags-arkitekturen,« Datalogisk Institut, Aarhus, 2011.
- [13] »criticaltechnology.blogspot.com,« 2011. [Online]. Available: <https://criticaltechnology.blogspot.com/2011/09/mvc-in-three-tier-architecture.html>.

- [14] E. Gamma, R. Helm, R. Johnson og J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
- [15] E. Freeman og E. Robson, *Head First Design Patterns*, Sebastopol, CA: O'Reilly Media, 2021.
- [16] »learn.microsoft.com,« 2025. [Online]. Available: <https://learn.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver16>.
- [17] »mindrot.org,« 2025. [Online]. Available: <https://www.mindrot.org/projects/jBCrypt>.
- [18] N. Provos og D. Mazières, »A Future-Adaptable Password Scheme,« *Proceedings of the FREENIX Track:1999 USENIX Annual Technical Conference*, Juni 1999.
- [19] T. Connolly og C. Begg, *Database systems: a practical approach to design, implementation, and management*, Pearson Education, 2005.
- [20] »danishcrown.com,« 2025. [Online]. Available: <https://www.danishcrown.com/dk/kontakt/presse-og-nyheder/nyheder/optimal-slagtevaegt-er-gaaet-op/>.

9. Bilag

Bilag A:

Gruppekontrakt

Medlemmer: Martin Farsø, Johannes Kizach & Henrik Flensborg

Kontraktindhold

Overholdelse af aftaler:

- Gruppemedlemmer forpligter sig til at overholde indgåede aftaler.

Ansvar og engagement:

- Alle skal løfte deres del af projektet.
- Der forventes tydelig kommunikation, hvis der opstår afvigelser fra det planlagte.

Gennemgang og forståelse:

- Alt, der afleveres, skal være gennemgået og forstået af alle gruppemedlemmer.

Kodestandarder:

- Kodekommentarer og brugerinteraktion i GUI'en skal være på dansk.
- Navngivning af metoder og klasser skal være på engelsk og følge camelCase.

Det følger, at flere punkter kan tilføjes løbende med fuld enighed fra alle gruppemedlemmer.

"A verbal contract isn't worth the paper it's written on." – Samuel Goldwyn

Dato:

Underskrift:

Bilag B:

Sprint Planning - Sprint 1

Sprint Information

Startdato: 05-05-2025

Slutdato: 09-05-2025

Scrum Master: Martin Farsø

DevTeam: Henrik Bach Flensborg, Johannes Kizach, Martin Farsø

Sprint Mål

Målet for dette sprint er at få projektet sat godt i gang, ved at få de grundlæggende funktioner til at virke. På kanbanboardet fremgår der arbejdsopgaver, som forventes færdig i sprintet.

Arbejdsopgaverne dækker over følgende:

- Gui
- Login
- Overview Stack
- Pig Stack
- Location Stack
- Settings Stack
- View Factory
- Database
- Import af data
- KPI stored procedures
- Model
- DTO
- SP -> DTO

Totalt: 71 Arbejdsopgaver

Backlog Items

Ingen backlog items, da det er vores første sprint.

Tilgængelighed

Henrik Bach Flensborg - Vil være tilgængelig alle dage.

Johannes Kizach - Vil være tilgængelig alle dage.

Martin Farsø - Vil være tilgængelig alle dage.

Risici og afhængigheder

Grundet afhængighederne af database design og stored procedures, fokuseres der på den del først.

Aftaler for Sprinten

Daily standups kl. 14.00 efter behov og tilgængelighed.

Review og Retrospective dato: Afholder vi i sammenhæng med næste sprints opstart.

Vi vil efter hvert daily standup holder vi eventuelle udviklingsmøder efter behov.

Noter og kommentarer

Ingen noter eller kommentarer.

08-05-2025 – Daily Standup

Henrik

- Arbejdet med KPI'er og ændringer.
- Vi gennemgår dem alle igen og tager udgangspunkt.

Johannes

- Opdatering af Kanban med opgaver.
- Løbende implementering af KPI'er i view.

Martin

- Arbejdet med model og fodret KPI'er til DTO'er (færdig med 7).
- Mangler afklaring på hvilke views KPI'erne skal bruge.

09-05-2025 – Daily Standup

Henrik

- Opdateret Stored Procedures efter i går.
- Oprettet KPI 17.
- KPI 8 skal datosorteres.
- DBUtils:
- Opret 2 hashmaps:
- kpild -> Title
- kpild -> boolean
- Udkommentering af bekræftelsesbeskeder.

Johannes

- Organiserer og tweaker som en gal.
- KPI'er med scroll:
- Overvejer mulighed for fullscreen-visning.
- Justerer størrelsen på bar-grafer med scroll.

Martin

- Arbejder fortsat med KPI'er – mangler 5.
- KPI 5 konverteres til lineDTO.
- KPI 10 ændres til Combo-visning.

Sprint Review & Retrospective - Sprint 1

Sprint Information

Startdato: 05-05-2025

Slutdato: 09-05-2025

Scrum Master: Martin Farsø

DevTeam: Henrik Bach Flensborg, Johannes Kizach, Martin Farsø

Sprint Review

Vi opnåede i store træk det forventede resultat. Vi har enkelte udestående, men vores planlægning af arbejdsopgaver og indgangsvinkel til projektet har vist sig fornuftigt.

Arbejdsopgaver løst: 66

Udestående arbejdsopgaver efter sprint: 5

Ændringer i Product Backlog

Vi tilføjer udestående arbejdsopgaver til vores product backlog med høj prioritet.

ID	Beskrivelse
32	Model - KPI 15 - Alarm - LowFCR
34	Model - KPI 16 - Alarm - LowFeedConsumption
35	Model - KPI 17 - Alarm - LowFeedConsumptionLocation
69	GUI - Items til GUI
71	DBUtils - Hashmap

Sprint Retrospective

Hvad gik godt?

Godt struktureret arbejde, med klare forventninger og overskuelige arbejdsopgaver. Samarbejdet fungerer godt, der er en hyggelig atmosfære og godt humør.

Hvad kunne være bedre?

Vi skal være bedre til at holde vores Kanban board opdateret. Arbejdsopgaver forsvinder når de ikke flyttes til deres aktuelle fase.

Forslag til forbedringer

Bedre navngivning af arbejdsopgaver.

Aftalte handlinger og eksperimenter til næste sprint

Vi aftaler at vi alle prøver at være mere opmærksomme på de arbejdsopgaver vi sidder med og holder dem opdateret i kanban.

Sprint Planning - Sprint 2

Sprint Information

Startdato: 12-05-2025

Slutdato: 16-05-2025

Scrum Master: Martin Farsø

DevTeam: Henrik Bach Flensborg, Johannes Kizach, Martin Farsø

Sprint Mål

Fortsat implementering af KPI'er. Yderligere 32 arbejdsopgaver tilføjet til kanban som dækker over følgende:

- KPI Opdateringer og tilføjelser i DB, Model og GUI
- Nye KPI'er 18-26
- ViewFactory opdateres med scroll
- Import af Excel og export af CSV

Totalt: 35 Arbejdsopgaver

Backlog Items

ID	Beskrivelse
32	Model - KPI 15 - Alarm - LowFCR
34	Model - KPI 16 - Alarm - LowFeedConsumption
35	Model - KPI 17 - Alarm - LowFeedConsumptionLocation
69	GUI - Items til GUI
71	DBUtils - Hashmap

Tilgængelighed

Henrik Bach Flensborg - Vil være fraværende d. 13/14/15 i normal tidsrum.

Johannes Kizach - Vil være tilgængelig alle dage.

Martin Farsø - Vil være tilgængelig alle dage.

Risici og afhængigheder

Ingen store afhængigheder på trods af fravær. Henrik er tilgængelig og vil kigge på de forskellige arbejdsopgaver uden for normal tidsrum.

Aftaler for Sprinten

Daily standups kl. 14.00 efter behov og tilgængelighed.

Review og Retrospective dato: Afholder vi i sammenhæng med næste sprints opstart.

Vi vil efter hvert daily standup holder vi eventuelle udviklingsmøder efter behov.

Noter og kommentarer

Grundet fraværet allokerer vi én dag til systematisk test af færdige arbejdsopgaver.

12-05-2025 – Daily Standup

Henrik

- Opdateret hashmap.
- Rapportskrivning:
- Beskrivelse af builder pattern og valg heraf.
- Små kodeændringer.

Johannes

- Kigget på plots og nye DTO'er.
- Afventer ændringer i koden.

Martin

- Arbejder på KPI'er 15, 16 og 17.
- Skal lave getters og setters til hashmaps.
- Kigget på plots – ser fornuftigt ud.
- Introducerer AreaDTO og tilføjer nye KPI'er med ny DTO.

13-05-2025 – Daily Standup

Henrik

- Nye KPI'er – App breakdown fixet (grundet null-værdier i hashmaps).
- Alle KPI'er klar til modellen.
- Fortsætter med rapporten indtil næste update.

Johannes

- Færdiggør koblinger mellem model og view.
- Arbejder med alignment af labels og generel GUI.

Martin

- Kigget på PasswordUtils, sat hash ind i database.
- KPI 18 og 19.
- DBUtils: upload af hashmap og enable i modellen.

Sprint Review & Retrospective - Sprint 2

Sprint Information

Startdato: 12-05-2025

Slutdato: 16-05-2025

Scrum Master: Martin Farsø

DevTeam: Henrik Bach Flensborg, Johannes Kizach, Martin Farsø

Sprint Review

Solidt sprint med projektet stort set færdigt. Få mangler som stort set er færdige og bare mangler få justeringer.

Arbejdsopgaver løst: 32

Udestående arbejdsopgaver efter sprint: 3

Ændringer i Product Backlog

Vores product backlog blev renset i sprintet, og tre nye arbejdsopgaver er blevet tilføjet efter sprintet.

ID	Beskrivelse
78	Import Export Stack
99	Import Excel
103	Export af CSV

Sprint Retrospective

Hvad gik godt?

Kanbanboardet blev opdateret løbende og det gjorde det muligt at se hvor vi var klar til test og gjorde det muligt at få overblikket på projektet. Samarbejdet var igen rigtig positivt og alle har en god tilgang til projektet.

Hvad kunne være bedre?

Mere struktureret tilgang til test. Vi endte med omkring 60 arbejdsopgaver, som var klar til test, før de blev testet.

Forslag til forbedringer

Oprette testcases med forventede resultater til mere struktureret test. Og test løbende når arbejdsopgaver bliver færdige.

Aftalte handlinger og eksperimenter til næste sprint

Eftersom implementeringsfasen snart er overstået, vil vi gradvis gå mere i gang med rapporten. Derfor ser vi ingen grund til at ændre vores sprintstruktur.

Sprint Planning - Sprint 3

Sprint Information

Startdato: 19-05-2025

Slutdato: 20-05-2025

Scrum Master: Martin Farsø

DevTeam: Henrik Bach Flensborg, Johannes Kizach, Martin Farsø

Sprint Mål

Få implementeret de sidste udestående arbejdsopgaver fra vores backlog.

Totalt: 3 Arbejdsopgaver

Backlog Items

ID	Beskrivelse
78	Import Export Stack
99	Import Excel
103	Export af CSV

Tilgængelighed

Henrik Bach Flensborg - Vil være tilgængelig alle dage.

Johannes Kizach - Vil være delvis fraværende mandag d. 19.

Martin Farsø - Vil være tilgængelig alle dage.

Risici og afhængigheder

Ingen risici eller afhængigheder

Aftaler for Sprinten

Daily standups kl. 14.00 efter behov og tilgængelighed.

Review og Retrospective dato: Afholder vi d. 22/05 som afslutning på implementeringsfasen.

Vi vil efter hvert daily standup holder vi eventuelle udviklingsmøder efter behov.

Noter og kommentarer

20-05-2025 – Daily Standup

Henrik

- Rapport overhaul:
- Databasekrav
- Databasedesign
- Scrum i metode
- Scrum og opbygning af hvert sprint
- KPI-opdateringsliste gennemgået.

Johannes

- Implementeret CSV-funktionalitet i GUI.

Martin

- Opsætning og eksport af CSV.

Sprint Review & Retrospective - Sprint 3

Sprint Information

Startdato: 19-05-2025

Slutdato: 20-05-2025

Scrum Master: Martin Farsø

DevTeam: Henrik Bach Flensborg, Johannes Kizach, Martin Farsø

Sprint Review

Færdiggjort implementeringsfasen. Det var et kort sprint, men det lykkedes os at færdiggøre efter tidsplanen.

Arbejdsopgaver løst: 3

Udestående arbejdsopgaver efter sprint: 0

Ændringer i Product Backlog

Tom backlog 😊

Sprint Retrospective

Hvad gik godt?

Vores samarbejde og struktur viser sig gentagende at komplimentere hinanden godt.

Hvad kunne være bedre?

Vi ser ikke noget kunne have været bedre i det korte sprint.

Forslag til forbedringer

Ingen forbedringer.

Aftalte handlinger og eksperimenter til næste sprint

Færdiggøre scrumfasen og påbegynder rapportskrivning.