Semester Projekt 3:

Casino Chip Counter

Group members and student numbers:

Andreas Møller Gadgaard - 202306990

Emil Holm Riis - 202305265

Jesper Lund Pedersen - 202308221

Kristian Kirkegaard Trætholt Stausholm - 202308573

Maciej Tomasz Brylski - 201605868

Mikkel Mortensen - 202005937

Tobias Konrad Nielsen - 202007236

3. Semester Project

Group nr: SW 14

Softwareteknologi

Aarhus University

Supervisor: Danish Shaikh

Hand-in Date: March 24, 2025

Project Period: 26/09 - 13/12 - 2024

Softwaretechnology
Aarhus University
Softwaretechnology
Finlandsgade 24A
8200 Aarhus N

Title:

Semester projekt 3 - Casino Chip Counter

ECTS:

5 ECTS

Semester:

3. Semester

Project period:

26/09 - 13/12 - 2024

Project group:

SW 14

Group members and student numbers:

Andreas Møller Gadgaard - 202306990
Emil Holm Riis - 202305265
Jesper Lund Pedersen - 202308221
Kristian Kirkegaard Træholt Stausholm - 202308573
Maciej Tomasz Brylski - 201605868
Mikkel Mortensen - 202005937
Tobias Konrad Nielsen - 202007236

Supervisors:

Danish Shaikh

Ordforklaring

Forkortelse	Forklaring
AI	Kunstig intelligens
AM	Applikations model
BDD	Block Definition Diagram
CCC	Casino Chip Counter
DSI	Display Serial Interface
GND	Ground
GUI	Graphical User Interface
HW	Hardware
IBD	Internal Block Diagram
MISO	Master in slave out
MoSCoW	Must have, Should have, Could have and Won't have
MOSI	Master out slave in
OOP	Object-oriented programming
PSoC	Programmable system on chip
PWM	Pulse width modulation
RPi	Raspberry Pi v4
SD	Sequence Diagram
SPI	Serial peripheral interface
STM	State Machine Diagram
SW	Software
UC	Use Case

Forkortelse	Forklaring
UML	Unified modeling language
URL	Uniform resource locator
USB	Universal serial bus

Abstract og Resumé

Abstract

The purpose of this project is to demonstrate an understanding of the key processes in a project lifecycle and to develop a functional system based on the requirements specified in the project proposal. The focus has been on process management and problem-solving throughout various project phases.

SCRUM was used as the primary method for planning and organizing, with tasks divided into smaller, manageable parts through sprints. This provided the group with a clear overview of the project's tasks and priorities.

The developed system, **Casino Chip Counter**, is designed to sort casino chips and transfer data to a database, which users can utilize for graphical tracking of account activity. The development process followed an iterative approach, where each module was designed, implemented, and tested individually. Subsequently, the system was integrated and tested to ensure requirements were met and potential shortcomings identified.

The project concludes with an evaluation of the system's functionality, revealing that some planned features were not implemented. Despite this, the group successfully developed a prototype that demonstrates the core functionality of the system.

Resumé

Dette projekt har til formål at demonstrere en forståelse for de centrale processer i et projektforløb samt at udvikle et funktionelt system baseret på de krav, der er specificeret i projektoplægget. Fokus har været på processtyring og problemløsning gennem forskellige projektfaser.

SCRUM er blevet anvendt som metode til planlægning og organisering, hvor arbejdsopgaver blev opdelt i mindre, håndterbare dele gennem sprints. Dette har givet gruppen et klart overblik over projektets opgaver og prioriteringer.

Det udviklede system, **Casino Chip Counter**, er designet til at sortere jetoner og overføre data til en database, som brugeren kan anvende til grafisk at spore kontoaktivitet. Udviklingsprocessen fulgte en iterativ tilgang, hvor hvert modul blev designet, implementeret og testet separat.

Efterfølgende blev systemet integreret og testet for at sikre opfyldelse af krav og identificere eventuelle mangler.

Projektet afsluttes med en vurdering af systemets funktionalitet, som viser, at nogle planlagte funktioner ikke blev implementeret. Trods dette har gruppen formået at udvikle en prototype, der demonstrerer systemets kernefunktionalitet.

Contents

Forord 1

Dette 3. semesterprojekt er udarbejdet af 7 SW-studerende:

- Andreas Møller Gadgaard
- Emil Holm Riis
- Jesper Lund Pedersen
- Kristian Kirkegaard Træholt Stausholm
- Maciej Tomasz Brylski
- Mikkel Mortensen
- Tobias Konrad Nielsen

Projektet er gennemført i samarbejde med vores vejleder, Danish Shaikh, og er udarbejdet som en del af undervisningsfaget ***SW3PRJ3-01 Semesterprojekt 3***.

Vi har valgt dette projekt for at integrere vores viden fra 3. semester og drage nytte af erfaringerne fra tidligere semesterprojekter. I dette semester har vi desuden arbejdet med et egentligt styrings- og tidsplanlægningsværktøj, hvilket har givet os en dybere indsigt i, hvordan et projekt kan håndteres.

Vi vil gerne takke vores vejleder, Danish Shaikh, for hans værdifulde vejledning og støtte gennem hele projektforløbet. Derudover ønsker vi at rette en særlig tak til AU's Elektronik værksted for teknisk rådgivning og udlån af komponenter, som har gjort udførelsen af projektet muligt.

Projektet er afleveret March 24, 2025.

Ansvarsområder 2

2.1 Fordeling over ansvarsområder

Ansvars Område		
Sektion	**Primær**	**Sekundær**
HW-Design		
Aktuator	Andreas	
SW-Design		
GUI	Emil, Tobias	
Kamera	Emil, Tobias	
Sorterings algoritme	Maciej, Mikkel	
Database	Jesper, Kristian	
Modultest		
Aktuatorer og PSoC	Andreas	
GUI	Emil, Tobias	
Kamera	Emil, Tobias	
Sorterings algoritme	Maciej, Mikkel	
Database	Jesper, Kristian	
Integration		
Færdig prototype	Tobias	Alle
Acceptest	Alle	
Afslutning		

Ansvars Område (Fortsat)		
Sektion	**Primær**	**Sekundær**
Diskussion	Alle	
Resultater	Alle	
Konklusion	Alle	
Proces		
Kontakt person	Maciej	
Mødeleder	Emil, Andreas	
Scrum-master	Jesper	
Procesbeskrivelse	Emil, Jesper	
Product Owner	Alle	

Table 2.1. Ansvars tabel

Problemformulering 3

3.1 Problemformulering og indledning

Gruppen er kommet frem til følgende problemformulering:

- Hvordan kan vi udvikle et kompakt og brugervenligt system, der kan tælle, registrere og sortere casinojetoner efter værdi?

Projektet har ikke noget specifikt tema, men har nogle krav, som skal være en del af projektet. Disse krav fremgår i projektoplægget.[1.1.] Kravene består af følgende:

- Systemet skal interagere med omverdenen via sensorer og aktuatorer.
- Systemet skal have en brugervenlig brugerinterface.
- Systemet skal inkludere faglige elementer fra semesterets fag.
- Systemet skal anvende en indlejret Linux-platform og en PSoC-platform.

Målgruppe:
Projektets målgruppe er casinoer, der vil kunne bruge produktet til at automatisere optælling og sortering af jetoner efter værdi.

Projektets overordnede mål:
Brugeren skal kunne logge ind på en konto gennem en GUI. Det fremstillede system skal kunne tælle og registrere jetoner baseret på den QR-kode, som tildeles til hver jetonfarve. En algoritme skal kunne genkende hver QR-kode og tildele de korrekte værdier til brugerens konto, som befindes på en database.

Det iagttages dog at de primære brugere af systemet er casinoets spillere. Derfor vil systemet også være fokuseret på brugervenlighed, samt mindske casino medarbejdernes arbejdsbyrde.

Til opfyldning af kravene anvendes følgende komponenter:

- **Sensorer:**
  - 1x Raspberry Pi Camera Module v2

- **Aktuatorer:**
  - 2x Micro Servo SG90 (9 gram)
- **Indlejret linux enhed:**
  - Raspberry Pi 4 model B
- **PSoC:**
  - PSoC 5 LP CY8CKIT-059

Projektformulering

Brugeren skal kunne logge ind på deres brugerkonto gennem en browser-baseret GUI. Her præsenteres brugeren for fire valgmuligheder: "Deposit", "Withdraw", "Check balance" og "Log-out". Brugeren skal kunne benytte alle fire funktioner, men i projektrapporten tages udgangspunkt i muligheden "Deposit".

Brugeren kan derefter placere en jeton i Casino Chip Counter-sytemet. Systemet skal starte med at åbne port 1. Port 1 er styret af en servo motor (SG90), og giver adgang til kammeret med Raspberry Pi Camera Module. Jetonen falder på plads. Derefter lukkes port 1, og kameraet scanner jetonen. Scanningen identificerer jetonens QR-kode, tilskriver jetonens værdi, og sender den videre til algoritmen. Algoritmen gemmer på værdien. Når scanningen er fuldført, åbnes port 2 (som er styret af en anden SG90 motor), og jetonen føres videre til fysisk sortering. Algoritmen vil digitalt sortere alle værdierne og sende dataet videre til databasen.

Brugeren har mulighed for at hente information fra databasen. Når jetonbunken er blevet scannet og registreret, kan brugeren se, hvor mange jetoner der er registreret, ved at trykke på "Check balance". Her kan brugeren få vist:

- Antallet af sorterede jetoner
- Hvor meget der er vundet eller tabt
- Den aktuelle værdi på kontoen.
- Yderligere oplysninger om brugeren.

Jetonerne fremgår også "sorteret" på brugerens konto, hvilket giver et hurtigt overblik over status og beholdning.

Nedenstående er første udkast for systemopsætningen, vist som et rigt billede:

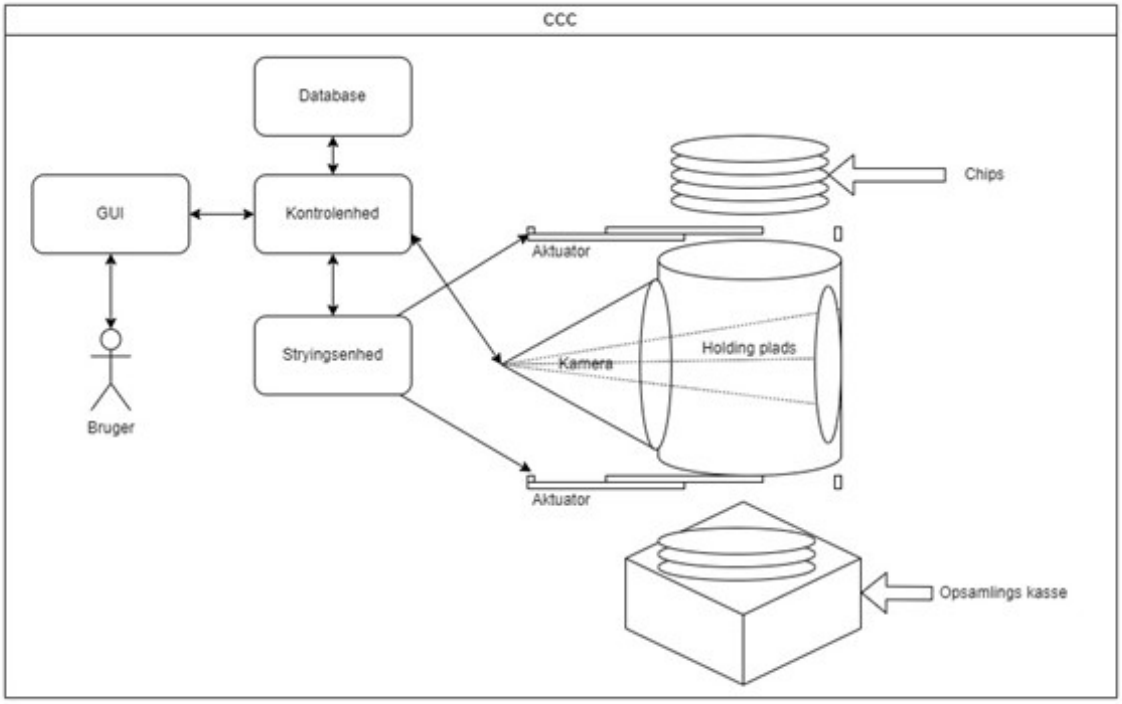

Figure 3.1. Rigt billede af CCC

3.1.1 Projektafgrænsning

Gruppen har ikke benyttet nogle ekstra ressourcer, udover det der er blevet udleveret af værkstedet. Da gruppen består udelukkende af SW-studerende, er der valgt at lægge mere fokus på softwaredelen af projektet. Oprindeligt var planen, at sorteringen skulle foregå gennem flere porte (flere aktuatorer som styrede forskellige porte), således at jetonerne ville blive sorteret fysisk. Dette har vi dog valgt at simulere i stedet.

Metoder og arbejdsproces 4

Dette afsnit tager udgangspunkt i bilaget Procesbeskrivelse.pdf[9], hvori der forekommer en dybere gennemgang af processen og arbejdsmetoden.

I projektet *Casino Chip Counter* har vi gennemført en struktureret arbejdsproces, der inkluderede planlægning, udvikling og projektstyring. Gruppen blev selvorganiseret og bestod af medlemmer med forskellige styrker, hvilket blev understøttet af Insight-profiler[13] for at optimere samarbejde og forståelse af gruppens dynamik. Vi udarbejdede en samarbejdsaftale[15] i begyndelsen af projektet, som sikrede klare rammer for gruppens arbejde og håndtering af potentielle konflikter.

Arbejdsmetoden var baseret på den agile udviklingsmetode Scrum[10], hvor opgaver blev opdelt i sprints og styret gennem værktøjet Trello. Scrum gav os mulighed for at arbejde iterativt og skabe et overblik over opgaver, som blev organiseret i en samlet *product backlog* og efterfølgende *sprint backlogs*. Rollen som Scrum master blev fastholdt af én person for at sikre stabilitet, mens *product owner*-ansvaret blev delt blandt gruppemedlemmerne, hvilket fremmede en fælles forståelse af opgaverne.

Projektplanlægningen inkluderede en tidsplan, der blev tilpasset undervejs for at tage højde for travlhed med andre fag. Vigtige milepæle blev understøttet af ugentlige gruppemøder[12] og vejledermøder på strategiske tidspunkter i projektet, såsom opstart, reviewrunder og accepttests.

GitLab blev brugt som central platform for kodeversionering og backup, hvilket var særligt vigtigt efter et medlem mistede en lokal kopi af koden ved tyveri. Derudover blev Microsoft SharePoint brugt til deling af dokumenter, og Google Drev til samarbejde om diagrammer.

Reviewprocessen bestod af to runder, hvor vi modtog feedback på problemformulering, kravspecifikation, systemarkitektur og valg af komponenter. Feedbacken var afgørende for at rette fejl og mangler, der ikke var tydelige internt i gruppen.

Gennem hele projektforløbet har vi haft en flad ledelsesstruktur med fælles beslutningstagning. Dette fremmede ejerskab og ansvar blandt alle gruppemedlemmer og bidrog til, at vi som gruppe nåede de opstillede mål.

Det har været svært for os at anvende scrums principper som studerende. Det er åbenlyst, at scrum er et værktøj, der er primært tiltænkt arbejde i industrien. Vi oplevede, at vi i løbet af semestret havde svært ved at arbejde intensivt i sprints, når vi samtidig har en fuld hverdag med undervisning og afleveringer, hvilket medførte, at vi endte med enten at nå meget lidt af det, der var på vores taskboard, eller at skubbe slutningen af sprintet, så vi kunne nå at færdiggøre flere af de igangsatte arbejdsopgaver.

Hen mod slutningen af semesteret, hvor vi havde haft den sidste undervisning og afleveret de sidste afleveringer i de øvrige fag, havde vi tid til at afsætte flere hele dage i træk til at arbejde intensivt på projektet. I denne periode havde vi en helt anden oplevelse af scrums sprintarbejde, hvor gruppens medlemmer løbende færdiggjorde arbejdsopgaver og satte sig på nye på taskboardet i et helt andet tempo. Vi føler, at det gav os et bedre, mere positivt indblik i, hvordan scrum-arbejdet fungerer, når det danner hele rammen for éns arbejdsdag.

Vi kunne godt have brugt, at der var blevet afholdt mere undervisning om scrum, så vi kunne anvende værktøjerne mere effektivt fra starten. Det gælder især, når der bliver lagt op til, at vi ifm. projektet arbejder mere selvstændigt. Meget af det materiale om agile udviklingsmodeller, der kan findes på nettet, er skrevet med virksomheder i øjemed, og var svært at arbejde med, når vi havde så lidt forforståelse for emnet, som vi havde. Der havde det været særligt oplagt at vi havde fået et oplæg om, hvordan man bedst arbejdede med det i en studiesetting - hvad det gav mening at bruge, og hvilke dele, der var lidt sværere at oversætte direkte fra erhvervslivet.

Vi havde tidligt i processen et møde med vores vejleder, hvor en stor del af tiden gik med, at vi blev sat ind i, hvordan man ud fra hans erfaring kunne arbejde med scrum. Det gav os et noget bedre udgangspunkt for det efterfølgende arbejde, som vi nødigt havde været foruden. Vi fik også god sparring fra vores vejleder i forhold til, hvad vores projekt bør indeholde ift. semestrets faglige indhold. Efterfølgende havde vi en mere selvstændig proces, end vi har haft på de tidligere semester. Vores vejledermøder handlede meget om, at vi gav en kort oversigt over, hvor vi var i processen lige nu, hvortil vores vejleder kunne komme med råd. Der var ikke samme behov for, at vi blev guidet igennem projektets enkelte faser løbende, som der har været på tidligere semestre, hvor vi har arbejdet med lineære udviklingsmodeller.

Selvom det ikke er alle dele af det agile procesarbejde, der har fungeret lige godt for os, har det givet gruppen værdifulde erfaringer inden for projektledelse, samarbejde og teknisk udvikling. Den flade ledelsesstruktur og den iterative arbejdsmetode har skabt en solid ramme for projektets succes og sikret, at gruppen i fællesskab har nået sine mål. I fremtiden kommer vi til at

arbejde videre med især planlægningsværktøjerne, som vi har lært at bruge til dette projekt. Taskboards[10.6.] er oplagte til at danne et fælles overblik over, hvor vi er i processen, når vi som travle studerende ikke nødvendigvis taler sammen om projektarbejdet dagligt. Det giver kun mening at arbejde med sprints på de tidspunkter, hvor vi har tid til, at det kan danne en ramme om vores hverdag. Det giver derfor ikke mening at anvende det i den form, vi har brugt det, når vi skal lave fjerde semesterprojekt. Baseret på vores positive oplevelse med det i slutningen af semestret, er det formegentligt en arbejdsform, flere af os kommer til at bruge, når vi når til at skrive bachelorprojekt.

Kravspecifikation 5

5.1 Systembeskrivelse

Casino Chip Counter (CCC) er et system designet til at tælle, registrere og sortere casinojetoner baseret på deres QR-koder. Systemet består af flere komponenter, der arbejder sammen for at opnå dette mål.

5.1.1 Hovedkomponenter

5.1.1.1 Raspberry Pi 4 Model B:

Funktion: Fungerer som den primære indlejrede Linux-platform, der håndterer hovedlogikken og kommunikationen i systemet, den har fået navnet "Kontrolenhed"
Rolle: Modtager data fra kameraet, behandler QR-koderne, og opdaterer brugerens konto i databasen.

5.1.1.2 PSoC 5 LP CY8CKIT-059:

Funktion: Bruges til at styre aktuatorerne, der åbner og lukker portene for jetonerne, den har fået navnet "Styringenhed".
Rolle: Modtager kommandoer fra Raspberry Pi via SPI og styrer servo motorerne præcist ved hjælp af PWM.

5.1.1.3 Raspberry Pi Camera Module v2:

Funktion: Scanner QR-koderne på jetonerne, den har fået navne "Kamera".
Rolle: Sender de scannede data til Raspberry Pi for videre behandling.

5.1.1.4 Micro Servo SG90:

Funktion: Styrer åbning og lukning af portene.
Rolle: Sikrer, at jetonerne kan falde på plads til scanning og derefter videre til opsamlingskassen. (dette er ikke fuldt implementeret, aktuatorne virker, men da der ikke er bygget en kasse til CCC

kan disse aktuatore virke overflødige, men de viser hvordan systemet kan fungere i en fremtidig version.

5.1.2 Systemets funktionaliteter

5.1.2.1 Login og Brugergrænseflade:

Brugeren logger ind via en browser-baseret GUI, hvor vedkommende præsenteres for valgmulighederne: "Deposit", "Withdraw", "Check balance" og "Log-out".

5.1.2.2 Jetonhåndtering:

Når brugeren vælger "Deposit" åbnes port 1, og jetonen kan placeres foran kameraet, hvorefter port 1 lukkes. Kameraet scanner jetonen og identificerer dens QR-kode, hvis værdi bliver gemt. Herefter åbner port 2, så jetonen falder ud af kammeret, port 2 lukkes igen, og til sidst åbnes port 1 igen, så CCC er klar til at modtage den næste jeton.

5.1.2.3 Datahåndtering og Opdatering:

Når brugeren er færdig med at scanne et antal jetoner, sendes summen af de scannede værdier til en MySQL database, hvor brugerens konto opdateres.

Når brugeren bruger funktionen "Withdraw", kan brugeren vælge et beløb. Herefter bliver databasen på lignende vis opdateret, så værdien bliver fjernet fra brugerens balance.

Brugeren kan til enhver tid tjekke deres balance via GUI'en, via funktionen "Check balance", som viser den aktuelle balance på kontoen. Her kan der også blive vist en oversigt over de jetoner, brugeren har indsat, og over dennes seneste transaktioner.

5.1.3 Aktør-kontekst Diagram

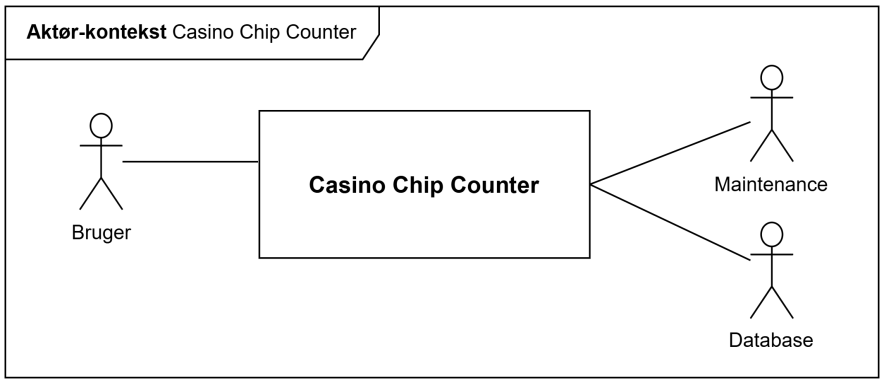

Figure 5.1. Aktør-kontekst Diagram

5.1.4 Use Case Diagram

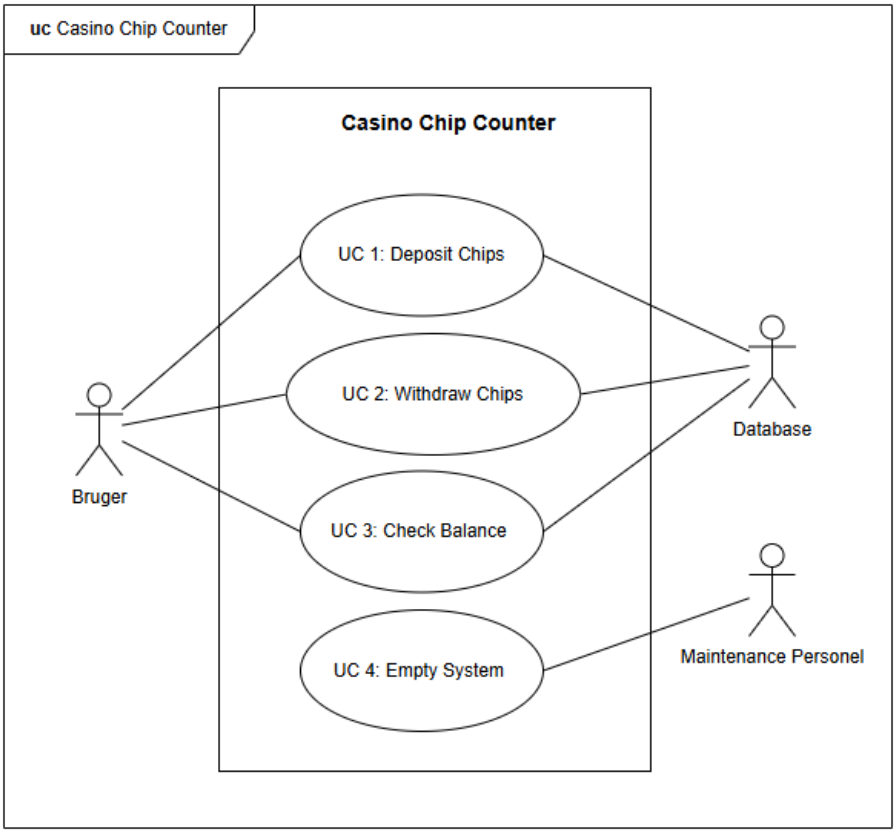

Figure 5.2. Use Case Diagram

5.2 Funktionelle krav

5.2.1 MoSCoW

Kategori	Beskrivelse
Skal	• Systemet **skal** være i stand til at læse QR-koder på jetoner. • Systemet **skal** være i stand til at sortere digitalt. • Brugeren **skal** være i stand til at logge ind. • Der **skal** være et brugerinterface, som brugeren kan interagere med. • Kameraet **skal** være i stand til at scanne en jeton og finde dens værdi. • Portene **skal** være i stand til at åbne og lukke automatisk. • Systemet **skal** have adgang til en web-baseret database, der indeholder brugeroplysninger og -balancer. • Brugeren **skal** have mulighed for at se deres balance. • Brugeren **skal** have mulighed for at stoppe systemet.
Bør	• Brugeren **bør** kunne få en fysisk eller digital kvittering af systemet til at få udleveret jetoner eller penge andetsteds. • Systemet **bør** have en tæller, der giver besked, når det nærmer sig maks. kapacitet. • Systemet **bør** have en tæller, der lukker for tilføjelse af jetoner, når det rammer maks. kapacitet.
Kan	• Brugeren **kan** få mulighed for at se anden information såsom balancehistorik. • Systemet **kan** få mulighed for at stoppe af sig selv efter en rum tid. • Systemet **kan** føre statistik på transaktioner, såsom daglige antal hævninger eller beløbssum.
Vil ikke	• Systemet **vil ikke** sortere jetonerne fysisk. • Systemet **vil ikke** kunne scanne mere end en jeton ad gangen. • Systemet **vil ikke** have en mekanisme for at hæve/få fysisk udleveret jetoner. • Systemet **vil ikke** oprette en bruger.

Table 5.1. Krav til systemet opdelt efter MoSCoW-prioritering

5.2.2 Use Case

Navn:	**Use Case 1: Deposit chips**
Mål:	Bruger indsætter jetoner i systemet og jetonernes værdi tilføjes til brugers konto.
Initiering:	Bruger
Aktører:	Primær: Bruger Sekundær: Database
Antal samtidige forekomster:	1
Prækondition:	Systemet er tændt og jetonbeholder er ikke fyldt. Systemet er på login skærm.
Postkondition:	Brugers kontobalance er opdateret. Systemet er klar til næste bruger.
Hovedscenarie:	1. Bruger logger ind på systemet • *[Ext 0]: Bruger indtaster forkerte oplysninger* 2. Systemet præsenterer bruger for 4 valgmuligheder: • Deposit, Withdraw, Check Balance og Log Out 3. Bruger vælger 'Deposit' 4. System viser mulighederne: • Done 5. Bruger indsætter jetoner i maskinens scanner 6. Jeton scannes af systemets kamera 7. Jeton flyttes til beholder • *[Ext 2]: Inaktivitet* 8. Bruger trykker Done 9. Brugerens kontobalance opdateres i databasen 10. Systemet går tilbage til login skærm

Udvidelser:	**[Extension 0: Bruger indtaster forkerte oplysninger]** 0.1: Bruger indtaster forkerte brugernavn &/eller password 0.2: Systemet udskriver "Forkert brugernavn &/eller password" 0.3: Systemet går tilbage til login skærm **[Extension 1: Bruger trykker på 'Log out']** 1.1: Bruger trykker på 'Log out'. 1.2: Systemet viser "You have been logged out." 1.3: Systemet går tilbage til login skærm. **[Extension 2: Inaktivitet]** 2.1: Efter 20 sekunder udskriver systemet automatisk: "Kontobalance er opdateret" 2.2: Brugeren logges ud. 2.3: Systemet går tilbage til login skærm.

Table 5.2. Use Case 1: Deposit chips

5.3 Ikke-Funktionelle krav

5.3.1 FURPS

Kategorier	Krav
Funcionality	• Systemet skal være i stand til at læse QR-koder på jetoner på maksimalt 2 sekunder.
Usability	• Brugeren skal være i stand til at logge ind med bruger ID og egen kode. • Systemets UI bør være intuitivt og nemt at navigere. • Brugeren skal have mulighed for at se deres balance før og efter udførelse af use case 1.
Reliability	• Systemet skal have en fejlrate på scanning af jetoner på under 5%.
Performance	• Systemet kan stoppe af sig selv efter 30 sekunders inaktivitet.
Supportability	• Systemet bør have en tæller, der giver besked, når det nærmer sig maksimal kapacitet.

Table 5.3. FURPS Krav

5.4 Afgrænsning

Forskellige problemstillinger opstod efter ide-fasen, hvor nogle af ideerne viste sig at være for tidskrævende og omfattende. Et eksempel på dette var ønsket om at lave en fysisk sortering. Ideen her var, at udover at systemet skulle være i stand til at sortere og gemme i en database, hvilke og hvor mange jetoner, der var blevet indsat, skulle den også fysisk sortere dem således, at det ved systemet også ville være muligt at hæve jetoner. Dog var problemstillingen her, at denne opsætning og implementering var væsentlig mere hardware-orienteret. Derfor blev valget at sortere jetonerne virtuelt i systemet, så de stadig blev aflæst og sorteret til brug af maintenance.

En anden idé var at bruge billedegenkendelse til at aflæse jetonerne. Gruppen havde begrænset viden indenfor emnet, det viste sig derfor at være en for stor udfordring at gå i gang med at træne et AI til billedgenkendelse fra bunden så tidsgrænsen blev en faktor. En anden mulighed for at bibeholde billedegenkendelse var at bruge et bibliotek, men for at få dette til at fungere,

skulle det gives en del data. Gruppen kom frem til, at hvis der alligevel blev brugt et bibliotek, ville det i stor grad minde om at bruge QR-kode scanning, bortset fra at, der ikke skulle laves en masse data til genkendelsen.

Desuden er det i projektformuleringen beskrevet, hvordan brugerens balancehistorik mm. skulle vises til brugeren et sted på deres profil. Dette blev tilsidesat, fordi gruppen kom frem til, at det ikke giver mening for et casino at vise statistikker til deres kunder på denne måde.

Analyse 6

6.1 Risikoanalyse

En af projektets faser var analysen af forskellige risici, der kunne opstå under projektets forløb. Gruppen har valgt at håndtere denne fase på to måder:

- Identifikation af risikofyldte elementer, som teknologiske- og procesmæssige forhold.
- Håndtering af fundne risici for at opnå projektets mål.

6.1.1 Teknologianalyse

Teknologianalysen fokuserer på værktøjer, platforme og komponenter, der kan anvendes til at bygge systemet. Formålet er at sikre, at de optimale komponenter anvendes.

6.1.1.1 Analyse af microcontroller

Til systemet skal der bruges en microcontroller. Denne skal kunne styre logikken i systemet. Et eksempel på en af funktionerne, er modtagelse af data fra et kameramodul. Derfor har gruppen undersøgt forskellige microcontrollere og deres kompatibilitet med kamera-moduler. Yderligere kigges der på hukommelseskapacitet og gruppens kendskab til microcontrolleren. Sammenligningen er vist i tabellen nedenfor:

Microcontroller	Kamera	Hukommelse	Kendskab
Raspberry Pi Zero W	Ja, stort udvalg	512 MB	Ja
Raspberry Pi 4	Ja, stort udvalg	1–8 GB	Ja (RPi), ikke 4
Mega2560	Begrænset	256 KB	Ja

Table 6.1. Sammenligning af microcontrollere.

Analysen viser, at en Arduino MEGA2560 ikke vil være det mest oplagte valg. Selvom gruppen tidligere har arbejdet med denne microcontroller, var det tilhørende kameramodul ikke tilgængeligt i Embedded Stock. Dette kunne skabe problemer senere i projektforløbet og kræve, at der blev dedikeret flere ressourcer for at sikre et tilfredsstillende resultat.

Analysen viser desuden, at Raspberry Pi-microcontrollere tilbyder flere valgmuligheder af komponenter i Embedded Stock. Selvom gruppen ikke har stor erfaring med Raspberry Pi-microcontrollere, giver det alligevel mening at vælge en af dem, da der arbejdes med Raspberry Pi i løbet af semesteret. Derudover har Elektronik værkstedet erfaring med Raspberry Pi-komponenter og kan derfor potentielt yde værdifuld support, hvis der opstår problemer, eksempelvis i implementeringsfasen.

6.1.1.2 Analyse af kameramoduler

Analysen af kameramodulerne er tæt knyttet til microcontrolleren, da de fleste microcontrollere allerede har et dedikeret kameramodul. Valget afhænger primært af kameramodulets opløsning, men også af implementeringens enkelhed i forbindelse med microcontrolleren.

Microcontroller	Kamera Modul	Opløsning
Raspberry Pi Zero W	Raspberry Pi Camera Module 2	1080p (1920x1080)
Raspberry Pi 4	Raspberry Pi Camera Module 2	1080p (1920x1080)
Arduino ATmega 2560	Arducam Mini Module	5 MP (2592x1944)

Table 6.2. Kamera Moduler for Forskellige Microcontrollere

Analysen viser at selvom opløsningen for Arduino-kameraet er større, er det ikke nødvendigt for projektet, da en opløsning på 1080p er tilstrækkelig. Kameramodulet skulle også købes ind. Potentielt flere gange hvis det gik i stykker. Det var derfor mere fordelagtigt at bruge kamera modulet for en RPi.

6.1.1.3 Analyse af styreenhed

Projektet stiller krav om anvendelse af en PSoC. I dette projekt er PSoC'en blevet valgt som styreenhed til aktuatorerne, der kontrollerer de porte, som åbner og lukker for jetonerne.

Fordele:

1. PSoC er fleksibel og kan let tilpasses til at styre flere aktuatorer.
2. Har indbygget PWM, som gør det muligt at styre aktuatorer med høj præcision.
3. Gruppen har erfaring med PSoC fra tidligere semestre, hvilket reducerer behovet for omfattende oplæring.

Ulemper:

1. PSoC'en kan være overflødig til dette projekt og bruges primært, fordi det er et krav i projektet.

2. Begrænset kompatibilitet med andre udviklingsværktøjer uden for Cypress-økosystemet, hvilket kan komplicere integrationen.
3. Længere udviklingstid som følge af PSoC'ens konfigurationsbehov og avancerede funktioner.

Analysen af PSoC'en gav gruppen en bedre forståelse af hvad denne skal bruges til i projektet.

6.1.1.4 Analyse af aktuator

Aktuatorer skal bruges til at styre portene i vores system. Derfor fokuseres der på at vælge en motor, der både er stærk nok til at flytte en fysisk port og samtidig lille nok, så systemets dimensioner ikke bliver for store. Motoren skal kunne også styres med en vis præcision. Gruppen har tidligere arbejdet med tre typer af elektriske motorer. Disse ses i nedenstående tabel:

Motor Type	Fordele	Ulemper
DC Motor	- Simpel og billig - Høj hastighed - Let at kontrollere	- Svær at kontrollere position præcist - Kan ikke holde en fast position - Mindre effektiv ved lav hastighed
Stepper Motor	- Præcis positionering - Kan dreje i små trin - Billig og enkel kontrol	- Begrænset hastighed - Kan vibrere ved lav hastighed - Mere kompleks styring end DC motor
Servo Motor	- Præcis kontrol over position - God til små bevægelser - Kompakt design	- Begrænset rotationsvinkel - Kan være dyrere - Kræver ofte en feedback kontrol

Table 6.3. Fordele og ulemper ved forskellige typer motorer

Analysen viser at en servo motor vil være passende i vores projekt. Den giver en præcis kontrol over positionerne, og samtidigt er kompakt nok, så systemets dimensioner ikke bliver for store.

6.1.1.5 Analyse af database

I denne del af projektet blev der overvejet tre forskellige database løsninger: MsSQL, MySQL og MongoDB. Hver løsning blev analyseret og følgende er konkluderet:

- **MsSQL** - En struktureret, relationsbaseret database, som er velegnet til systemer, der kræver høj dataintegritet og faste skemaer. Dog med fokus i .net framework.
- **MySQL** - En open-source relationsdatabase, som tilbyder stor fleksibilitet, god ydeevne og bredt fællesskab. Den er en af de mest populære databaser og bruges i mange applikationer.

- **MongoDB** - En NoSQL-database, som tilbyder fleksible, skema-frie lagringsmuligheder og er bedre egnet til store mængder ustruktureret data, eller når dataens struktur kan ændre sig over tid.

Valget af database blev primært baseret på den nødvendige skalerbarhed og fleksibilitet i systemet, samt dens kompatibilitet med Python. Analysen viste at en MySQL database kunne være mest passende for projektet.

6.1.1.6 Analyse af algoritme

Algoritmens rolle i projektet er at sortere mellem en mængde af værdier, og derefter sende disse værdier over til databasen. Dermed blev der lagt fokus på sorteringsalgoritmer. Gruppens gennemgang af algoritmer indsnævrede mulighederne til følgende to:

Algoritme	Fordele	Ulemper
Insertionsort	- Simpel at implementere - Effektiv for små datasæt - In-place sortering	- Langsom for store datasæt - Tager O(n^2) i værste tilfælde - Dårlig ydeevne med tilfældige data
Quicksort	- Meget hurtig for store datasæt - Gennemsnitlig tid O(n log n) - In-place sortering - Bedre ydeevne end Insertionsort for store datasæt	- Kan have dårlig ydeevne i værste tilfælde O(n^2) - Kræver en god partitioneringsstrategi

Table 6.4. Fordele og ulemper ved Insertionsort og Quicksort

6.1.1.7 Analyse af kodesprog

Under valget af kode sprog, har gruppen valgt at kigge på følgende sprog:

Sprog	Fordele	Ulemper
Python	- Brugervenlighed - Let at teste og debugge - Velegnet til prototyper	- Langsommere ydeevne - Dårlig til realtidskontrol
C++	- Høj ydeevne - Velegnet til kompleks logik - Kan håndtere realtidskrav	- Stejl indlæringskurve - Længere udviklingstid
C	- Meget effektivt for embedded systemer - Direkte hardwarekontrol	- Mangler moderne features som OOP - Fejl kan være svære at debugge

Table 6.5. Fordele og ulemper ved Python, C++, og C i systemdesign

Alle tre sprog er gruppen velkendt med, dermed er valget af kodesprog, baseret mest på kompleksiteten af sourcekoden. I løbet af semesteret har gruppen arbejdet med indlejeret software udvikling, hvor der blandt andet, arbejdes med threading. Til dette er det en fordel at arbejde med C++, som har en bedre håndtering af threads end Python.

6.1.2 Procesmæssig analyse

Her foretages en analyse af procesmæssige forhold. Der tages udgangspunkt i, hvilke dele af systemet der har den største risiko for, at noget kan gå galt, og hvor stor en påvirkning dette kan have på projektet.

Derudover tages der hensyn til faktorer som frafald i gruppen, uenigheder mellem medlemmerne under projektfaserne, tidspres og lignende. En risikomatrice bliver udarbejdet med det formål at hjælpe gruppen med at tilpasse arbejdsfordelingen, så projektmålene kan blive opnået.

Først opstilles en tabel over de faktorer, der kan påvirke projektet. Hver faktor tildeles et tal fra 1 - 10, både for sandsynligheden og konsekvensen. Påvirkning bliver derefter beregnet med formlen: $R = P \cdot C$, hvor P er "Sandsynlighed" og C er "Konsekvens".

Beskrivelse	Sand synlighed 1-10	Konsekvens 1-10	Påvirkning 1-100	Plan B
Scanning	6	10	60	Simulere scanning på RPi
Database	6	8	48	Lave en lokal database på RPi
Brugerinterface	2	7	14	Lave en tekstbaseret UI i konsollen
Sortering (algoritme)	4	10	40	Skifte til en alternativ algoritme, QuickSort
Aktuator (PSoC)	4	8	32	Simulere aktuator på RPi
Skærm	5	6	30	Skifte til alternativ løsning, PC
Utilstrækkelig afgrænsning	7	8	56	Afskæring af system moduler
Tidspres	8	10	80	Revurdering af tidsplanen, tilpasning af arbejdsbyrde mellem medlemmerne
Uenigheder i projektgruppen	6	5	30	Udpege en projektleder, som tager det endelige valg
Frafald i projektgruppen	2	9	18	Uddelegere projektarbejde mellem resterende medlemmer, tidpasning til tidsplanen

Figure 6.1. Overblik over projekt risici

Ud fra figur 6.1 laves en risiko matrice. Risikomatricen er farvekodet, for at fremhæve alvorligheden af risikovurderingerne.

Her tages der udgangspunkt i påvirkningsfaktoren. Afhængigt af hvor stor faktoren er, tildeles der en farve, som derefter kan aflæses på risikomatricen.

Påvirkningsrisiko	Påvirknings-scala
Low	0 - 15
Low - Medium	15 - 30
Medium	30 - 50
Medium - High	50 - 75
High	75 - 100

Table 6.6. Tabel over påvirkningsfaktoren

Risikomatricen ser dermed således ud:

		Sandsynlighed				
		Meget usandsynli	Usandsynlig	Mulig	Sandsynlig	Meget sandsynlig
Konsekvens	Ubetydelig	Low	Low	Low - Medium	Medium	Medium
	Mindre betydeligt	Low	Low - Medium	Low - Medium	Medium	Medium - High
	Moderat	Low	Low - Medium	Medium	Medium - High	Medium - High
	Betydelig	Low	Low - Medium	Medium	Medium - High	Medium - High
	Alvorlig	Low - Medium	Medium	Medium - High	High	High

Figure 6.2. Risikomatrice

Som eksempel aflæses påvirknings faktoren på "scanning". En sandsynlighed på 6 og en konsekvens på 10, giver en påvirkningsfaktor på 60. Dermed klassificeres dette modul som en "medium-high" prioritet. Dette betyder at gruppen, planlægger arbejdsfordelingen således, at modulet med sikkerhed kan implementeres efter forventningerne.

Denne fase har hjulpet markant med at opdele arbejdsbyrden mellem projektmedlemmerne. Dette har gjort det nemmere at planlægge og skabe et overblik over projektet.

6.2 Valg af embedded enhed

6.2.1 Raspberry Pi 4

Valget af mikrokontroller til systemet faldt på Raspberry Pi 4. Denne platform blev valgt på grund af dens omfattende kompatibilitet med forskellige kameramoduler, hvilket var afgørende for systemets funktionalitet som helhed. Derudover gjorde tilstedeværelsen af en DisplayPort det muligt at gøre systemet selvstændigt, da en skærm kunne tilsluttes direkte.

6.2.2 PSoC

I projektet er det et krav, at en PSoC-mikrocontroller indgår i løsningen. Til dette formål er *CY8CKIT-059 PSoC 5LP Prototyping Kit* blevet valgt, da projektgruppen allerede har erfaring med denne platform. I projektet vil PSoC'en blive anvendt til PWM-styring af to aktuatorer. Derudover vil den kommunikere med den Raspberry Pi, som fungerer som den indlejrede Linux-platform, via SPI.

6.3 Valg af grænseflader

6.3.1 Skærm

For at kunne have systemet stående alene uden behov for en pc el. lign. som GUI, vil der blive anvendt en skærm. Der blev oprindeligt ikke lavet en teknologianalyse for skærmen, da det blev antaget, at et hvilken som helst RPi-skærmmodul ville gå an.

Ved første version af moduldesignet benyttedes en Waveshare 4inch HDMI LCD-skærm. Den var forbundet med HDMI- og 24 GPIO-forbindelser til skærmen, for at den kunne opnå fuld display out- og touchinput in-funktionalitet. De mange ledninger medførte en ret ufleksibel forbindelse, og gjorde det svært at tilgå de øvrige porte på Raspberry Pi'en.

Fordi en Raspberry Pi 4 er valgt frem for Zero, har vi mulighed for at koble skærmen med et RPi displaykabel, hvilket gør koblingen en del mere simpel. Efter vejledning af Elektronikværkstede er skærmen blevet skiftet ud med en DFRobot 0678 7" DSI Capacitive Touchscreen, som kan forbindes vha. displaykabel.

6.3.2 Aktuator

En analyse af de tilgængelige aktuatorer førte til valget af servomotorer som den optimale løsning til opgaven. De konkrete servomotorer brugt i projektet er to *Tower Pro SG90* mikroservomotorer.

Disse blev valgt, da de både var let tilgængelige via Embedded stock og opfyldte de opstillede krav til projektet.

6.3.3 Kamera

Til skanning af QR koderne anvendes et kamera. Til dette formål er der valgt Raspberry Pi Camera V. 2.1. Dette valg er truffet, grundet kompatibiliteten mellem kameraet og RPi'en. Da kameraet er lavet til RPi'en, ville der ikke forekomme uforudsete komplikationer ved sammenkoblingen. Ydermere var RPi kameraet let tilgængeligt i Embedded stock.

6.3.4 Valg af brugerinterface

Gruppen har fastlagt specifikke krav for brugerinterfacet. Da der anvendes en touchskærm, skal brugergrænsefladen som minimum kunne oprette knapper og vise tekst. Gruppen har fra starten valgt at bruge HTML/JS, da det er en oplagt mulighed for at skabe et webbaseret brugerinterface. Dette suppleres også med Flask, siden der arbejdes med Python kodesprog.

6.4 Valg af database

Til at håndtere CCC´s brugerdata er der behov for en database. Det er valgt at denne database er en MySQL database. Som database type er MySQL et relationelt databasestyringssystem. Det bruges til at gemme, organisere og administrere data i tabeller med rækker og kolonner. Dette passer godt til CCC's behov. Databasesproget er MySQL. MySQL er valgt, da MySQL har et stort netværk, hvori det er "let" at finde support.

6.5 Valg af algoritme: Insert Sort

I vores projekt har vi valgt at implementere **Insert Sort** som primær sorteringsalgoritme.

1. **Simplicitet og implementering**: Insert Sort er en relativt simpel algoritme at implementere, og da vores system primært håndterer små mængder data ad gangen (jetonerne, som skal sorteres), er Insert Sort tilstrækkelig effektiv til vores formål. Den er let at forstå og debugge, hvilket er en vigtig fordel under udviklingsfasen, da vi kan sikre, at alle data behandles korrekt, før vi går videre til den næste fase.
2. **Tidskompleksitet for små datasæt**: Insert Sort fungerer godt med små datasæt, som er tilfældet i vores projekt. Den har en tidskompleksitet på $O(n^2)$, men da antallet af jetoner der skal sorteres er relativt lille, vurderes tidskompleksiteten til at være tilstrækkelig.

3. **Stabilitet**: Insert Sort er en stabil algoritme, hvilket betyder, at elementer med samme værdi bevarer deres oprindelige rækkefølge. Dette kan være nyttigt, hvis vi senere skal behandle data, hvor rækkefølgen af elementer er vigtig [16].

Selvom Insert Sort er valgt som primær algoritme, er der også overvejet en alternativ løsning, **Quicksort**, som skal bruges som en plan B. Dette er blevet uddybet i bilaget.[2.4.]

Algoritmen er i starten tiltænkt at skrives i C++, grundet brugen af C++ i Algoritmer og Datastrukturer faget. Dette er dog ændret til Python, for at gøre integrationsfasen lettere i fremtiden.

6.6 Valg af programmeringssprog

I projektet arbejdes der med flere forskellige programmeringssprog, hver valgt til specifikke formål.

Python anvendes primært til kommunikationen mellem modulerne og Raspberry Pi'en (RPi). Gruppen har valgt at tilpasse så mange moduler som muligt til Python, da sproget er effektivt og fleksibelt til integration og kommunikation med hardwarekomponenter.

HTML fungerer som grundlaget for brugergrænsefladen. Det bruges til at definere strukturen af interfacet, herunder knapper, overskrifter, tekstfelter og andre visuelle elementer.

JavaScript anvendes til at tilføje funktionalitet til HTML-elementerne, hvilket gør brugerinterfacet interaktivt og dynamisk.

C er brugt til programmering af PSoC'en, da dette er standardsproget i PSoC Creator.

Systemarkitektur 7

7.1 Block Definition Diagram

BDD er et diagram der viser den strukturelle opbygning af systemet. Her defineres blokke, og de atributter som hører med til dem. En signalbeskrivelse laves, som beskriver en detaljeret relation mellem blokkene. Dette danner et grundlag for hvordan gruppen skal tilgå IBD arkitekturen.

På figur 7.1 ses BDD'et for CCC:

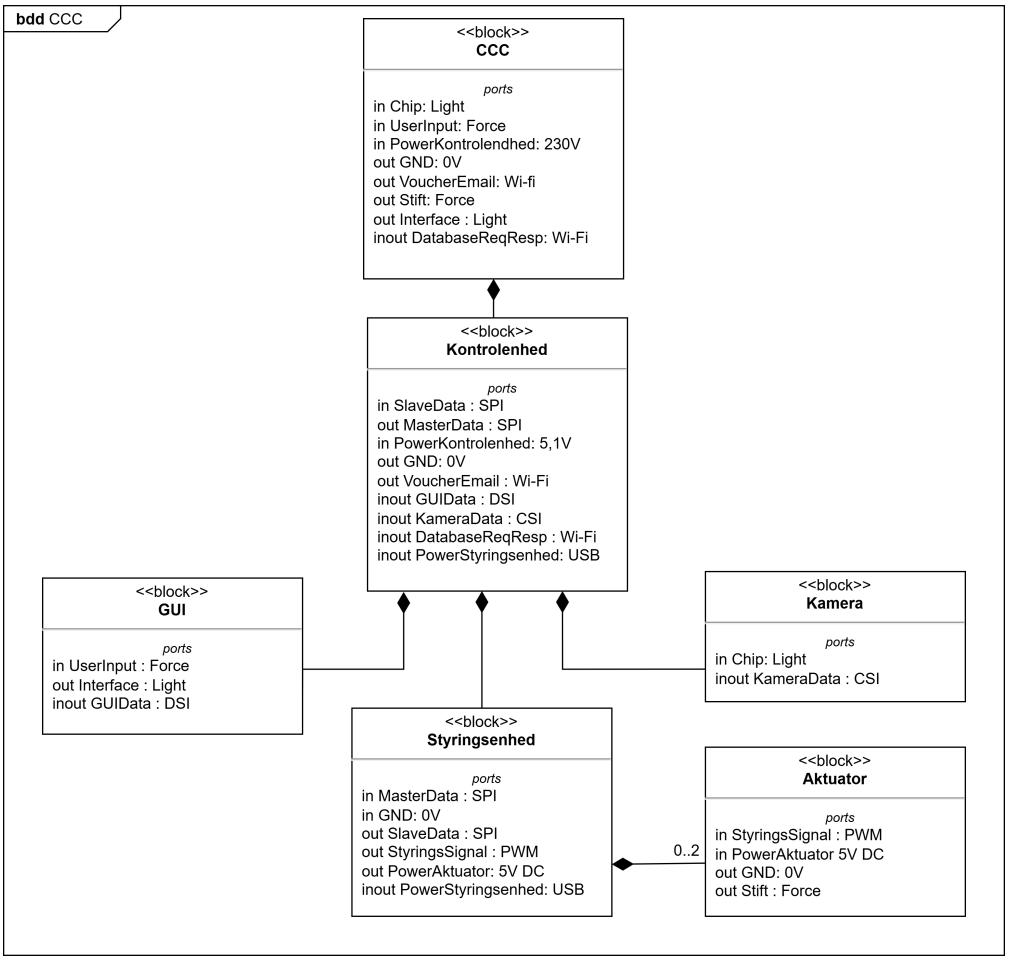

Figure 7.1. BDD af CCC

Blok-navn	Funktionsbeskrivelse	Signaler	Kommentarer
CCC	Beskriver systemet som helhed. Her modtages input fra brugeren og sender grænsefladeinformationer ud til brugeren.	In Chip: Light	
		In UserInput: Force	
		In PowerKontrolenhed: 230V	
		Out GND: 0V	
		Out VoucherEmail: Wi-Fi	
		Out Stift: Force	
		Out Interface: Light	
		InOut DatabaseReqResp: Wi-Fi	
Kontrolenhed	Håndterer signaler fra databasen, brugeren, chipstatus og sender kommandoer til kameraet og styringsenheden. Derudover kommunikerer den med alle andre blokke i systemet. Kontrolenheden får spænding fra elnettet igennem en adapter, så den får den nødvendige spænding, og leverer spænding til andre blokke i systemet.	inout PowerStyringsenhed: USB	RPi 4 Model B
		In SlaveData: SPI	
		Out MasterData: SPI	
		Inout KameraData: CSI	
		Inout GUIData: DSI	
		In PowerKontrolenhed: 5,1V	
		Out GND: 0V	
		InOut DatabaseReqResp: Wi-Fi	
		Out VoucherEmail: Wi-Fi	
GUI	Dette er touchskærmen som er brugerinterfacet for systemet. Her kan vælges forskellige muligheder, og der vises grafisk feedback.	Inout GUIData: DSI	DFRobot 0678 7" 800x480 TFT RPi DSI Capacitive Touchscreen V1.0. Denne blok får spændingsforsyning igennem DSI kablet.
		In UserInput: Force	
		Out Interface: Light	
Kamera	Kameraet bruges til scanning af jetoner, hvor værdien for den enkelte jeton sendes til kontrolenheden. Kameraet anvender CSI protokollen.	Inout KameraData: CSI	RPi-Camera Module V2. Denne blok får spændingsforsyning igennem CSI kablet.
		In Chip: Light	
Styringsenhed	Kontrollerer åbning/lukning til jetonbeholderen og styrer aktuatorernes bevægelser via PWM.	In MasterData: SPI	PSoC 5 LP CY8CKIT-059. Denne blok for spændingsforsyning igennem USB-forbindelsen.
		Inout PowerStyringsenhed: USB	
		In GND: 0V	
		Out SlaveData: SPI	
		Out Styringssignal: PWM	
		Out PowerAktuator: 5V DC	
Aktuator	Anvendes til at åbne og lukke til jetonbeholderen, så jetonerne kommer ind i systemet.	In Styringssignal: PWM	Servo Micro Motor 9G SG90.
		Out Stift: Force	
		In PowerAktuator: 5V DC	
		Out GND: 0V	

Figure 7.2. Blokbeskrivelse af BDD

7.2 Internal Block Diagram

IBD er et diagram der viser de interne forbindelser i systemet. Hvor BDD'et viser de overordnede forbindelser mellem blokkene, specificeres de interne forbindelser, med de korrekte porte, samt viser hvordan systemet interagerer med omverdenen. Visualisering af disse giver et godt overblik over, hvilke forbindelser der kan opstå fejl ved, dermed sikres en bedre design- og implementationsfase.

På figur 7.3 ses IBD'et for CCC:

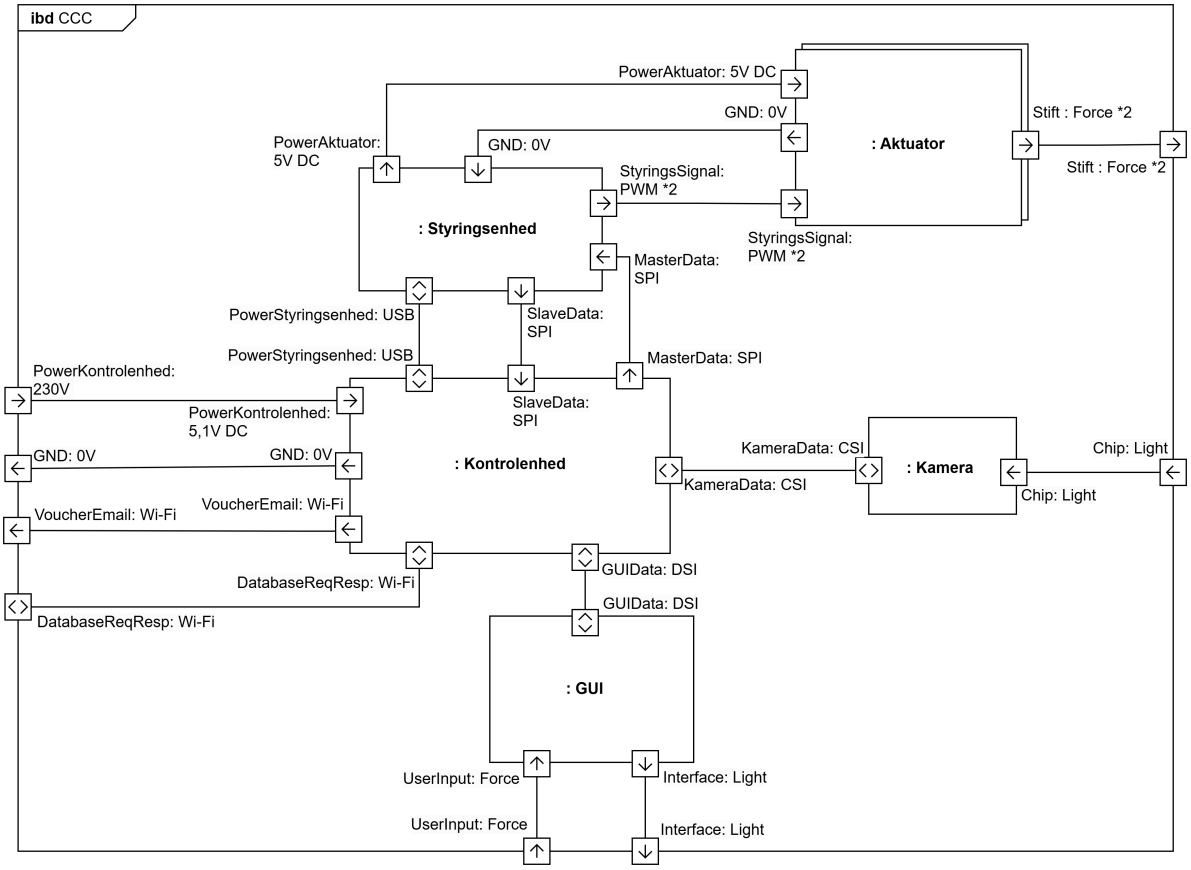

Figure 7.3. IBD af CCC

Der tilknyttes en signalbeskrivelse til systemets IBD, som en uddybende forklaring for, hvad de interne forbindelser på diagrammet anvendes til, som ses i tabel 7.1.

Signal-navn	Funktion	Område	Fysiske Porte	Kommentar
Kontrolenhed				
SlaveData: SPI	Modtager information fra styringsenheden (PSoC).	0-3,3V	MISO: GPIO9 CS: GPIO8	Anvender SPI protokollen. Logic converter konverterer 5V signal fra PSoC til 3.3V.
Database-ReqResp: Wi-Fi	Sender request til database om brugeroplysninger og modtager response fra database.	0-3,3V	-	-
GUIData: DSI	Sender grafisk information ud til brugeren og modtager brugerens input fra GUI.	0-3,3V	DSI	Anvender DSI protokollen. Logic converter konverterer 5V signal fra PSoC til 3.3V.
KameraData: CSI	Bruges til at styre kameraet i systemet. Leverer også spændingsforsyning til kameraet.	0-3,3V	CSI	Anvender CSI Protokollen.
MasterData: SPI	Sender kommandoer til styringsenheden.	0-3,3V	MOSI: GPIO10 SCLK: GPIO11	Logic converter konverterer 5V signal fra PSoC til 3.3V.
VoucherEmail: Wi-Fi	Sender voucher til brugeren via E-mail.	0-3,3V	-	-
PowerKontrolenhed: 5,1V DC	Spændingsforsyning med adapter fra 230V AC til 5,1V DC.	-	USB-C	Spændingsforsyning fra elnettet.
PowerStyrings-enhed: USB	Fungerer som spændingsforsyning 5V DC og fælles stel til styringsenheden.	-	USB-A	-
GND: 0V	Fælles stel.	-	USB-C	-
GUI				
GUIData: DSI	Sender grafisk information ud til brugeren og modtager brugerens input fra GUI.	0-3,3V	DSI	-
UserInput: Force	Registrerer tryk på touchskærmen som er brugerens interaktion med systemet.	0-3,3V	-	-
Interface: Light	Viser det grafiske interface for brugeren.	0-3,3V	-	-
Kamera				

KameraData: CSI	Kamera sender data for den jeton som er indsat i systemet.	0-3,3V	CSI	-
Chip: Light	Kameraet registrerer at en jeton er blevet sat ind i systemet til scanning.	0-3,3V	-	-
Styringsenhed				
MasterData: SPI	Modtager kommandoer fra kontrolenheden.	0-5V	MOSI: P1[7] SCLK: P1[5]	Logic converter konverterer 3.3V signal fra RPi til 5V.
SlaveData: SPI	Bruges til at overføre status fra styringsenhed til kontrolenhed.	0-5V	MISO: P1[6] CS: P1[4]	Logic converter konverterer 5V signal fra RPi til 3.3V.
Styringssignal: PWM	Styringssignal til systemets aktuatorer.	0-5V	P2[0] P3[0]	-
PowerStyrings-enhed: USB	Spændingsforsyning og fælles stel til styringsenheden.	0-5V	GND	-
PowerAktuator: 5V DC	Spændingsforsyning til aktuatorer.	0-5V	VCC	-
Aktuator				
Styringssignal: PWM	Aktuator modtager PWM signal, og styrer åbning / lukning af port.	0-5V	-	-
Stift: Force	Aktuator åbner eller lukker for porterne. Portene "styrer" en jeton som kommer ned til scanningskammeret. Når en jeton er kommet ned, lukker den første port, og den anden port åbner. Når jetonen har passeret port 2, vil denne port lukke, og port 1 vil vente på signal, om at åbne igen.	0-5V	-	-
PowerAktuator: 5,0V	Spændingsforsyning til aktuator.	0-5V	-	-
GND: 0V	Fælles stel.	-	-	-

Table 7.1. Signalbeskrivelse til yderligere forklaring af IBD

7.3 Sekvens Diagram

På figur 7.4 ses et SD diagram der beskriver hvad der sker i UC 1 "Deposit chips". Tilsvarende diagrammer for øvrige use cases findes i bilag.[4.6.]

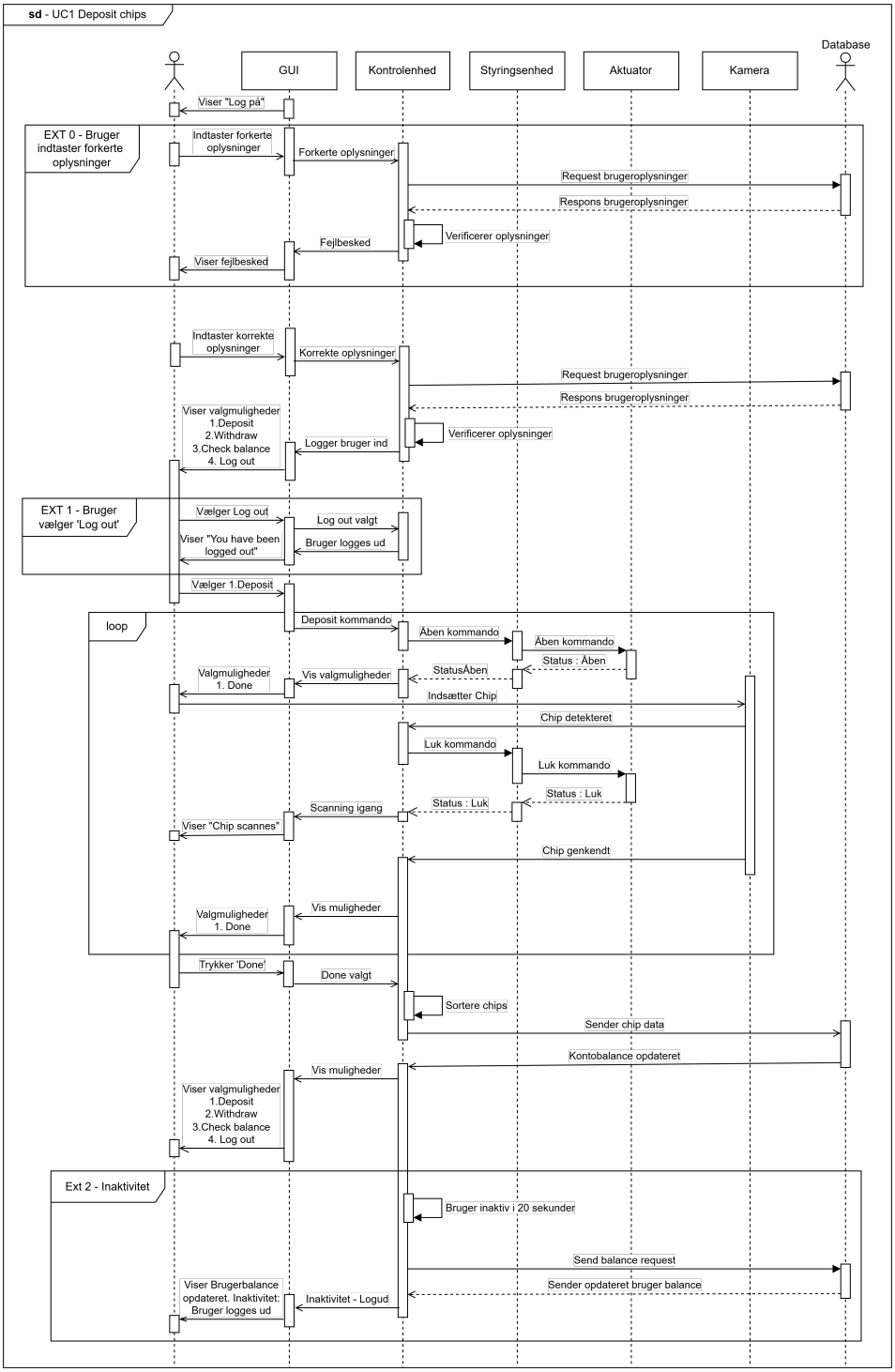

Figure 7.4. SysML SD UC1 - Deposit

Hardware 8

8.1 Hardware Design

8.1.1 Aktuatorer

Til projektet er der valgt at arbejde med Tower Pro's SG90 mikroservomotorer. Databladet for disse findes [7.3.].
Det fremgår i databladet, at aktuatorne styres med et PWM signal med en spænding på omkring 5V og en periode på 20ms. Ifølge databladet bør aktuatoren stå i -90° når signalet har en pulsbredde på 1ms, 0° når signalet har pulsbredde på 1,5ms og 90° når signalet har en pulsbredde på 2ms.

Til projektet ønskes aktuatorne at kunne skifte mellem -90° og 0°. PWM signalet sættes op på PSoC creator. PWM'en tilsluttes en clock med frekvensen 1MHz og sættes til at have en periode på 20.000. Der er valgt en 16-bit PWM fremfor en 8-bit, da 16-bit har bedre opløsning og giver mulighed for bedre finjustering.

Til at styre størrelsen af signalet pulsbredde, skal compare værdien (CMP value) varieres. Følgende formel bruges til at finde den korrekte CMP værdi:

$$C = t \cdot f$$

Hvor C er CMP værdien, t er den ønskede pulsbredde og f er clockfrekvensen.

De ønskede værdier indsættes i formlen:

$$C = \frac{1ms \cdot 1MHz}{1000} = 1000$$

$$C = \frac{1,5ms \cdot 1MHz}{1000} = 1500$$

De fundne værdier bruges i PSoC creator, og testes på aktuatorne. Ved test erkendes det at aktuatorne ikke helt drejer 90 grader som ønsket. Dette kan skyldes unøjagtigheder i både PSoC'ens PWM og aktuatorne. Efter eksperimentering, findes det at CMP værdierne 750 og 1700 giver den ønskede rotation.

Opsætningen af PWM'en kan ses på figur 8.1:

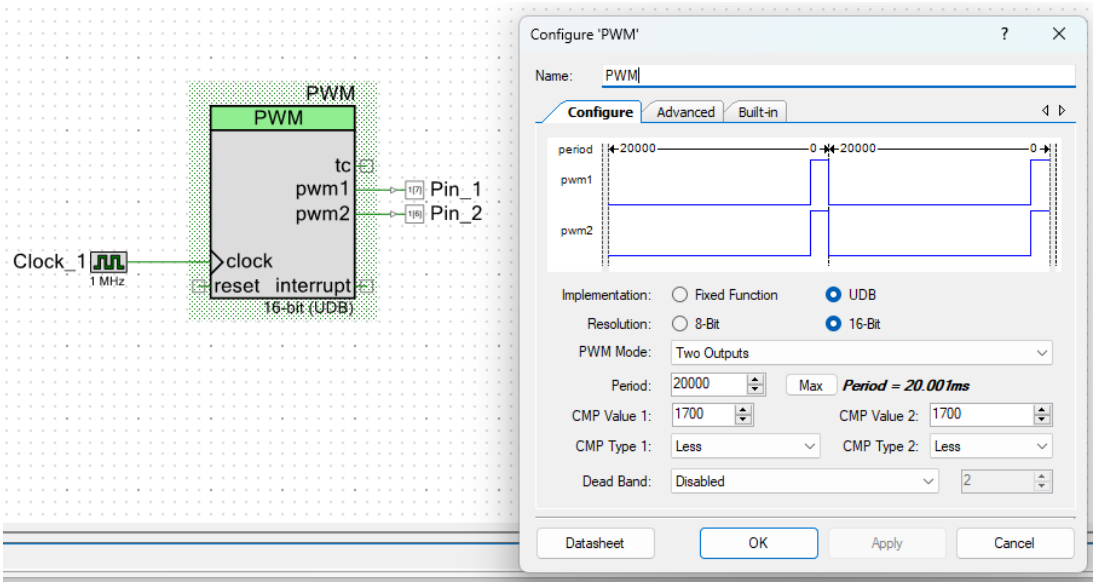

Figure 8.1. Opsætning af PWM

8.2 Hardware Implementering

8.2.1 PSoC og Aktuatorer

Til styring af PSoC'en fra en Raspberry Pi 4 anvendes SPI-protokollen som kommunikationsgrænseflade mellem de to enheder.

SPI-protokollen kræver, at PSoC'en konfigureres som SPI-slave, mens Raspberry Pi'en fungerer som SPI-master. Begge enheder understøtter alle fire SPI-modes. Mode 0 vælges, da det er standardindstillingen for begge komponenter, hvilket betyder, at både CPOL (Clock Polarity) og CPHA (Clock Phase) sættes til 0.

På figur 8.2 vises opsætningen af SPI-modulet i PSoC Creator. Figuren illustrerer relationen mellem signalerne MOSI (Master Out Slave In), MISO (Master In Slave Out), SCLK (Serial Clock) og CS/SS (Chip Select/Slave Select), når SPI anvendes i mode 0.

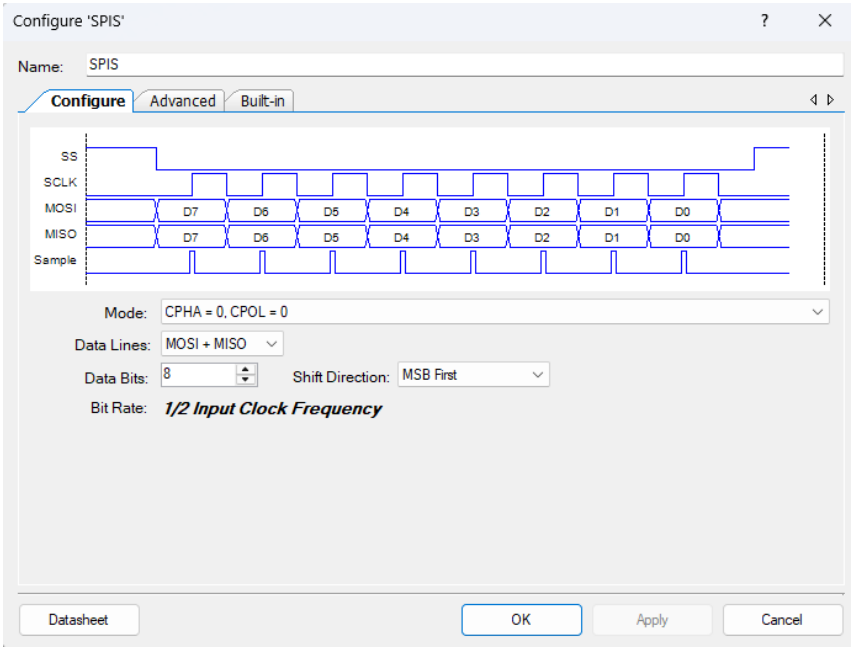

Figure 8.2. Opsætning af SPI-modulet i PSoC Creator med mode 0.

Figur 8.3 viser topdesignet fra PSoC Creator, hvor SPI-slave-modulet er indsat. Hver IO-pin (MOSI, MISO, SCLK, SS) er forbundet til en dedikeret GPIO på PSoC'en, som forbindes til Raspberry Pi'en med ledninger. For at sikre, at kommandoer fra Raspberry Pi'en håndteres korrekt, selvom PSoC'en er optaget af tidligere kommandoer, er et interrupt-komponent forbundet til SPI'ens `rx_interrupt`-signal. Dette gør det muligt at reagere på indgående SPI-data uden at gå glip af nogen kommandoer.

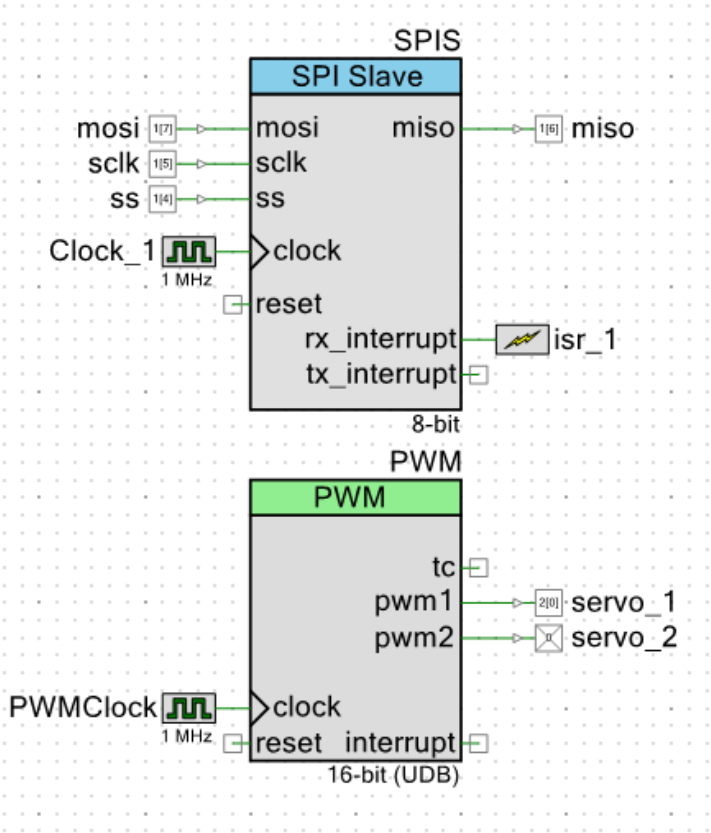

Figure 8.3. Topdesign fra PSoC Creator med SPI-slave og PWM-moduler.

8.2.1.1 Logic Converter

GPIO-pins på PSoC'en opererer ved 5V, mens Raspberry Pi'en benytter 3.3V. For at beskytte Raspberry Pi'en er det nødvendigt at anvende en logic converter. Til dette formål benyttes en 4-channel bidirectional logic converter, som vist i figur 8.4.

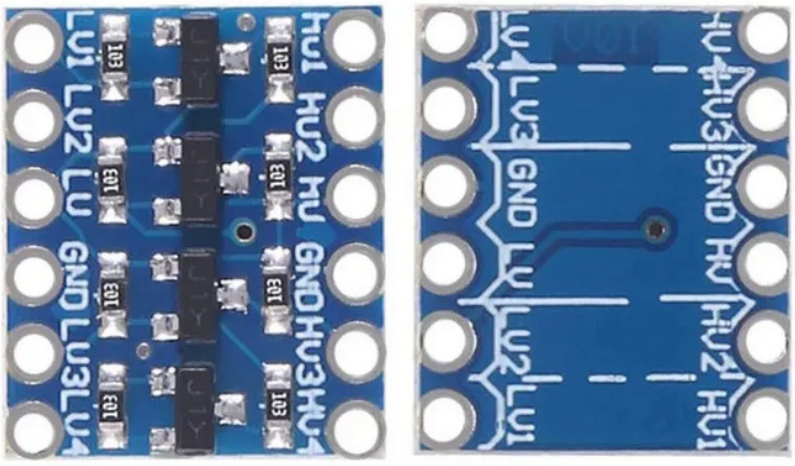

Figure 8.4. 4-kanals bidirektional logic converter

Logic converteren er enkel at bruge – PSoC'en forbindes til HV-siden, mens Raspberry Pi'en

forbindes til LV-siden. Denne opsætning sikrer, at Raspberry Pi'en ikke udsættes for for høje spændinger.

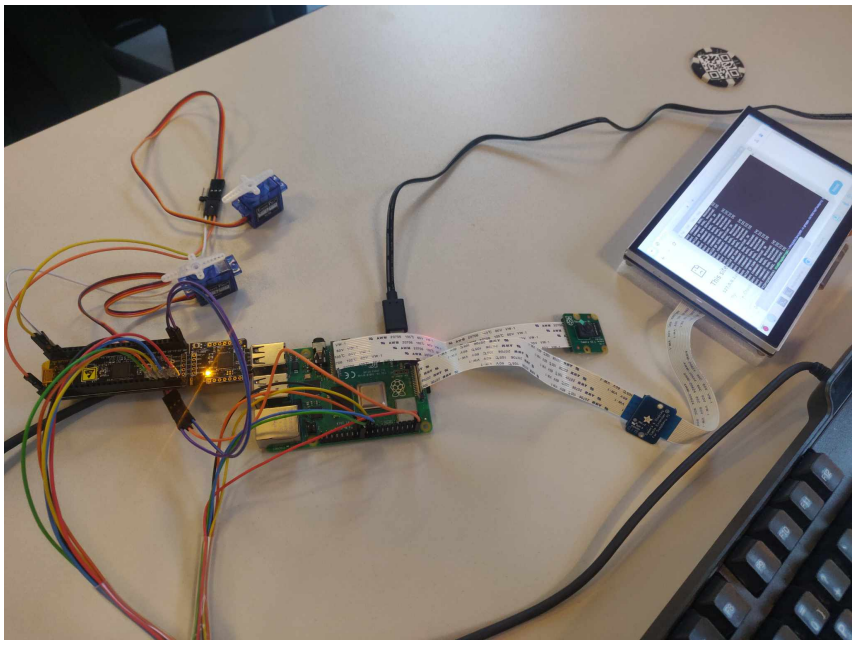

Figure 8.5. Test af PSoC kommunikation med aktuator

8.2.2 Raspberry Pi, kamera og GUI

De udvalgte komponenter til kamera og skærm er designet til at fungere problemfrit med Raspberry Pi som plug-and-play-enheder. Kommunikation mellem Pi, kamera og skærm kan håndteres ved hjælp af de tilgængelige biblioteker udviklet specifikt til formålet.

Software 9

9.1 Software Arkitektur

9.1.1 Domænemodel

Domænemodellen udarbejdes på baggrund af systemarkitekturen. I denne proces identificeres de forskellige blokke og attributter, der findes i systemet. Hver blok har sine specifikke attributter og forbindelser til aktører samt andre blokke, hvilket giver et overblik over den indledende kommunikation i systemet. Nedenstående figur 9.1 er domænemodellen for CCC. Modellen danner grundlaget for at gruppen kan fortsætte med at udvikle applikationsmodellen.

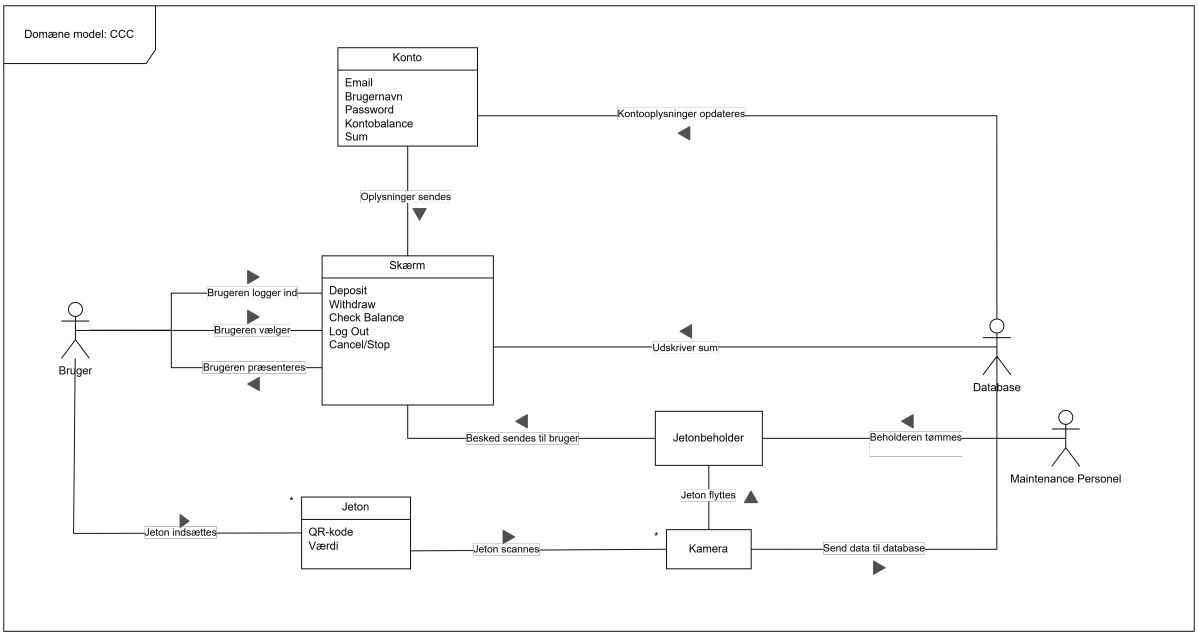

Figure 9.1. Domænemodel

9.1.2 Applikationsmodel

Applikationsmodellen er en videreudvikling af domænemodellen og beskriver systemets funktionalitet på et mere detaljeret niveau. Processen indebærer en analyse af de identificerede blokke og attributter, hvor der tilføjes metoder og interaktioner, som systemet skal understøtte.

9.1.2.1 Klassediagram

Klassediagrammer har til formål at vise hvilke funktioner og attributter ligger på de forskellige blokke. Gruppen starter med microcontrolleren, siden denne er central for systemet, og arbejder sig igennem alle relevante moduler. I rapporten præsenteres eksempler fra UC1, men de resterende kan findes i bilag [4.5.].

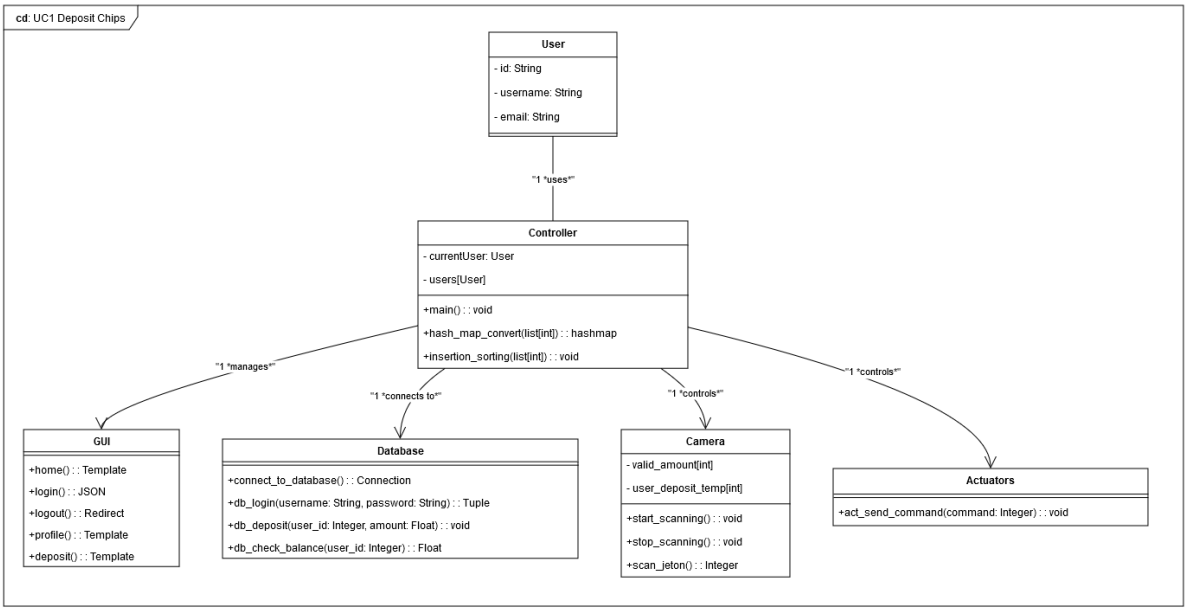

Figure 9.2. Klassediagram for UC1

9.1.2.2 Sekvensdiagram

Klassediagrammerne bruges i forbindelse med sekvensdiagrammer til at supplerede designet, med korrekte funktioner. Klassediagrammerne giver et statisk overblik over systemets struktur, mens sekvensdiagrammer fokuserer på den dynamiske adfærd. Disse opdateres løbende med klassediagrammerne.

I rapporten præsenteres eksempler fra UC1, mens de resterende kan findes i bilag [4.6.].

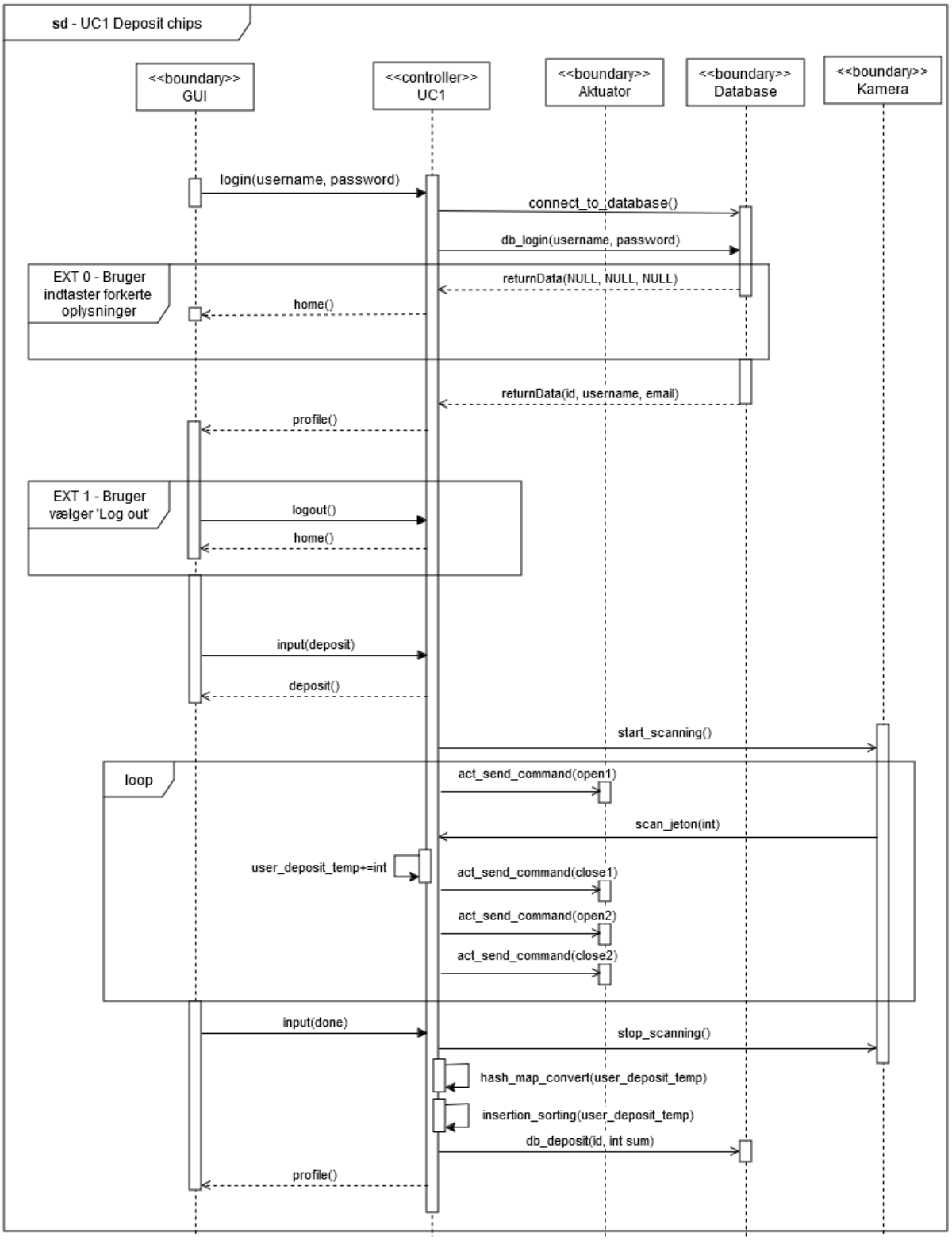

Figure 9.3. Sekvensdiagram for UC1

9.1.2.3 State machine diagram

STM bruges til at vise hvordan algoritmen skal kunne fungere som en seperat modul på figur 9.4.

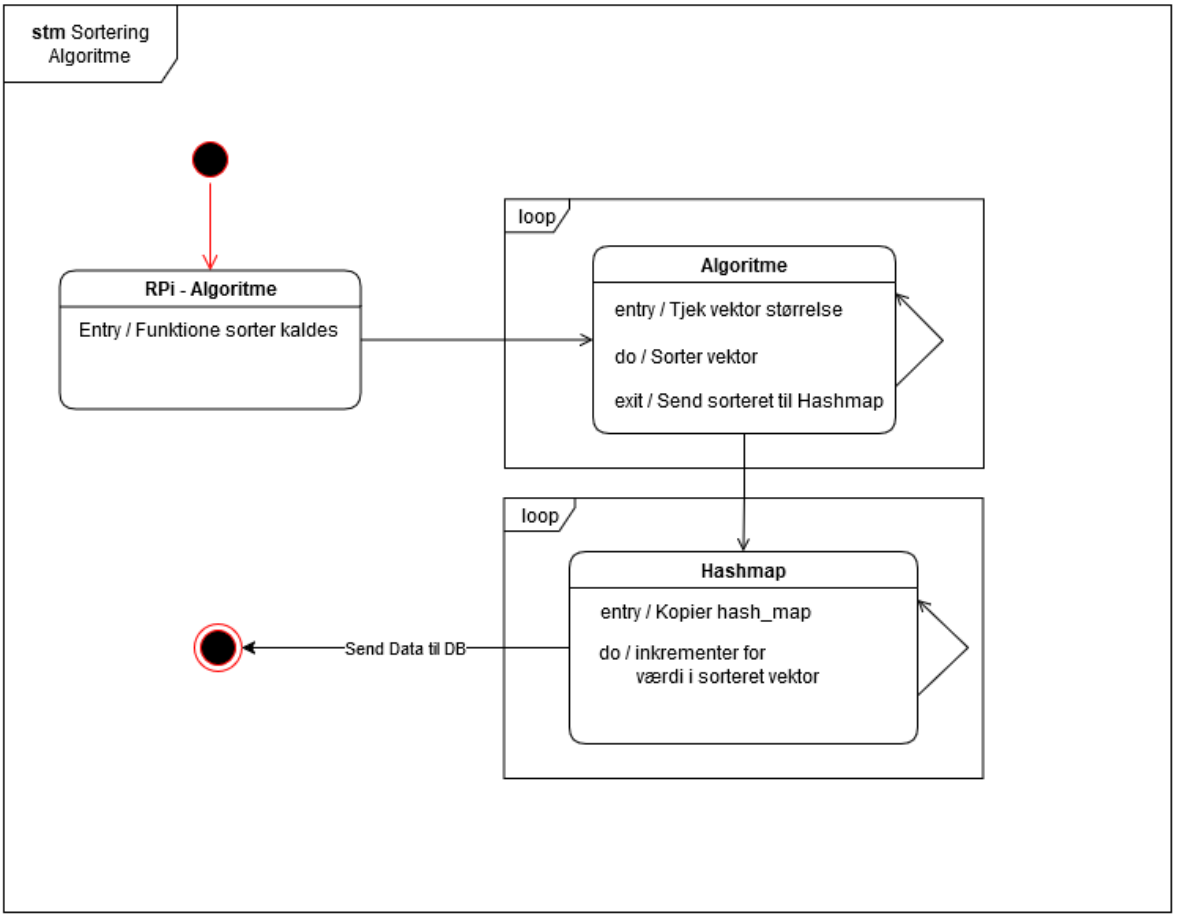

Figure 9.4. STM for algoritmen

9.2 Software Design- og implementation

9.2.1 Algoritme

Algoritmen består at to funktioner:

- insertion_sorting()
- hash_map_convert()

Funktionen **insertion_sorting()** er sorteringalgoritmen. Denne skal kunne iterere gennem en vektor og derefter sortere værdierne fra laveste til højeste. Først defineres to globale variabler:

- **sorting_vector**, som skal indeholde de værdier, der modtages via scanning.
- **temp_sort**, som midlertidigt skal gemme de sorterede værdier.

Algoritmen starter med at tage det andet element i **sorting_vector** og fortsætter til slutningen. Det aktuelle element, der sammenlignes, gemmes i variablen **temp**. Derefter sammenlignes **temp**

med de andre elementer i vektoren, og elementerne rykkes, indtil **temp** placeres på den korrekte position.

Når sorteringen er færdig, bliver det endelige sorterede resultat kopieret over til **temp_sort** for videre brug.

9.2.1.1 Tidskompleksitet

Tidskompleksiteten afhænger af hvordan dataene er organiseret i **sorting_vector** før algoritmen begynder. I værste tilfælde, er tidskompleksiteten på $O(n^2)$. Dette er tilfældet, hvis dataet er sorteret i omvendt rækkefølge (fra størst til lavest). Algoritmen vil skulle gøre følgende for hvert element i listen:

- Sammenligne hvert element med tidligere elementer.
- Flytte elementer der er større end det aktuelle.

Dermed er resultatet summen af operationer: $1 + 2 + 3... + (n - 1)$, hvilket svarer til:

$$\frac{n(n-1)}{2}$$

I bedste tilfælde er dataet allerede sorteret i **sorting_vector**. Her vil algoritmen stadig iterere gennem listen, men der vil ikke være behov for at flytte noget. Algoritmen sammenligner kun elementerne, der udføres n-1 sammenligninger i alt. Derfor er tidskompleksiteten $O(n)$. Fordi dataet er tifældigt organiseret, må der regnes med den værste tidskompleksit på $O(n^2)$,

hash_map_convert()
Variablen **hash_map** er en struktur, der indeholder antallet af hver jetonværdi, der er blevet scannet. Funktionen opretter en midlertidig kopi af **hash_map** og itererer gennem den liste, der indeholder værdierne fra scanningen. Hvis et element findes i listen, opdateres den midlertidige kopi af mappen ved at øge tælleren for det tilsvarende element. Når processen er færdig, udskrives det samlede resultat af opdateringerne fra mappet. Denne fremgangsmåde sikrer, at det oprindelige **hash_map** forbliver uændret under opdateringsprocessen.

9.2.2 PSoC

På PSoC'en kører et konstant loop, der kontinuerligt opdaterer compareværdierne i det PWM komponent, der styrer servomotorerne. Loopet bliver afbrudt af en interrupt rutine, der er styret af `rx`-interruptet i SPI komponentet.

9.2.3 Kamera

Til brugen af kameraet importeres biblioteket Picamera2, der er lavet til at tilgå kameraet digital. Kameraet bliver i Funktionen `picam2.capture_array()` bruges til at opfange et QR-frame fra.

Til at analysere QR-koder er funktionen `decode` importeret fra pyzbar biblioteket. Denne funktion muliggøre aflæsningen af den data, der befinder sig i framet af QR-koden. Hermed kan en værdien fra QR-koden parses til den ønskede værdi-type.

Begge af disse instanser er placeret i en while-løkke, til at kunne gentagende opfange QR-kode. Dette er laver med dei tanke, at denne while-løkke kun skulle igangsættes når GUI's /deposit-route tilgås.

9.2.4 GUI

Til GUI bliver anvendt en webbrowserbaseret klient som front end.

Back end for interfacet er en Python-server, der er opsat vha. Python Flask framework.

Flask-Login anvendes for at skabe et "lukket" back end-miljø, hvor brugeren som udgangspunkt kun har adgang til loginsiden, medmindre de er logget ind. Til det formål bruges en række af Flask-Logins indbyggede metoder.

9.2.4.1 Back End med Flask

Til back end-implementering anvendes Flask framework.

Flask-funktionen `render_template(template)` bliver brugt til at sætte den HTML-template, der skal indlæses, når brugeren tilgår en given route.

`jsonify()`-funktionen bruges til indpakning af data til JSON-format, så det kan sendes mellem server og klient.

`request` bliver brugt til håndtering af, hvilken metode, der bliver kaldt for en route, så der kan håndteres både POST- og GET-metoder, hvor det er relevant. De steder, hvor brugeren indtaster variable data, bliver `request` også brugt til at læse input-dataet. Det er det, der tillader GUI'en at have bl.a. loginformen og custom amount ved withdraw.

Derudover bruges `redirect()` og `url_for()` til omdirigering.

9.2.4.2 Flask-Login

Flask-Login anvendes til håndtering af brugere og sessioner. For at kunne anvende Flask-Login kræves det, at der bliver oprettet en `User`-klasse. `User`-klassen kan have forskellige variabler efter behov, men til dette projekt anvendes som udgangspunkt `id` og `username`. Til test af modulet bliver der også lavet variabler til `password` og `balance`, men i den endelige implementering forventes det, at de bliver fjernet til fordel for at have dem liggende i databasen.

Brugerens login-session bliver varetaget af en `login_manager`. Brugeren kan logges ind og ud ved brug af Flask-Login-funktionerne `login_user(user)` og `logout_user(user)`. Når brugeren er logget ind, sætter Flask-Login dem automatisk som `current_user`, så deres attributter kan kaldes, eksempelvis `current_user.id` eller `current_user.balance`. Er der ikke logget nogen bruger ind, bliver default-brugeren `UserMixIn` anvendt.

Så længe der ikke er en aktiv `current_user`, kan app routes med konditionen `@login_required` ikke tilgås. Da alle vores routes ud over loginsiden er afhængige af `current_user`, bruges konditionen for alle andre routes end vores base path og `/home`, der viser loginsiden.

9.2.5 Front End

Front end-klienten er implementeret som en række forskellige HTML-templates, der bliver render'et ved de relevante routes. I HTML-filerne er opsat `forms` og `buttons` som brugeren kan lave input igennem i webklienten, som bliver sendt som POST- og GET-metoder som en del af HTTP-protokol.

En del af Flasks template-funktionalitet bruger Jinja. Det tillader os at parse variable data fra serveren til klienten til rendering, eksempelvis i `/profile`-routen.

9.3 Funktionsbeskrivelser

Funktionsnavn	Parametre	Returværdi	Beskrivelse
insertion_sorting(sorting_vector: List[T])	List[T]	List[T]	Modtager en liste af ints fra kameraet, som bliver placeret i et hashmap, der har key der er værdien af inputtet, og value som er antallet af en given key der er indsat.
hash_map_convert(map_list: List[int])	List[T]	List[T]	Modtager en liste af ints fra kameraet, der bliver til en sorteret liste ved brug af insertion sort.

Table 9.1. Funktionsbeskrivelser for sortingalgorithm.py

Funktionsnavn	Parametre	Returværdi	Beskrivelse
connect_to_database()	Ingen	MySQLConnection	Opretter og returnerer en forbindelse til databasen.
db_login(username: str, password: str)	str, str	Tuple(int, str, str, str)	Autentificerer bruger og returnerer ID, brugernavn, email og brugertype.
generate_qr_code(data: str)	str	bytes	Genererer og returnerer en QR-kode som PNG-byteobjekt.
send_email_with_qr(user_email: str, amount: float)	str, float	Ingen	Sender en email med en QR-kode for udbetaling til angivet email.
db_deposit(user_id: int, amount: float)	int, float	Ingen	Indsætter beløb på brugerens konto og logger transaktionen.
db_withdraw(user_id: int, amount: float, user_email: str)	int, float, str	Ingen	Trækker beløb fra brugerens konto, logger transaktionen og sender email med QR-kode.
db_check_balance(user_id: int)	int	str	Returnerer brugerens aktuelle saldo som en formatteret streng.
main_menu(user_id: int, username: str, user_email: str, user_type: str)	int, str, str, str	Ingen	Viser hovedmenuen og håndterer brugerens valg.

Table 9.2. Funktionsbeskrivelser for database.py.

Funktionsnavn	Parametre	Returværdi	Beskrivelse
initialize_chip_hash()	Ingen	Ingen	Initialiserer en hash-map i sessionsdata, som gemmer antallet af chips af hver værdi.
load_user(user_id: str)	str	User	Henter en bruger fra det globale 'users' dictionary baseret på et bruger-ID.
home()	Ingen	HTML	Returnerer HTML for startsiden (login).
login()	Ingen	JSON/HTML	Håndterer brugerlogin, validerer data mod databasen og logger brugeren ind. Returnerer en JSON-respons med loginstatus.
logout()	Ingen	Redirect	Logger brugeren ud og nulstiller servoer til deres standardtilstand. Redirecter til startsiden.
profile()	Ingen	HTML	Returnerer HTML for brugerens profilside.
deposit()	JSON/Ingen	JSON/HTML	Ved GET starter QR-scanning; ved POST opdateres brugerens depositum baseret på scannede QR-koder og gemmer data i databasen og sessionen.
stop_scanning()	Ingen	Redirect	Stopper QR-scanningsprocessen og sender brugerens depositum til '/profile/deposit' for behandling. Redirecter til profilen.
withdraw()	Ingen	HTML	Håndterer brugernes anmodning om at hæve penge og opdaterer balancen i databasen.
check_balance()	Ingen	HTML	Returnerer HTML for siden, der viser brugerens balance.
start_scanning()	Ingen	Ingen	Starter kamera og scanner efter QR-koder. Håndterer validering af scannede værdier og styrer aktuatorsekvenser for chipindsættelse.

Table 9.3. Funktionsbeskrivelser for pyhthonserver.py

Funktionsnavn	Parametre	Returværdi	Beskrivelse
act_send_command(command: int)	int	Ingen	Sender en kommando til PSoC via SPI-protokollen og udskriver svaret fra PSoC. Anvendes til at styre servoer.

Table 9.4. Funktionsbeskrivelser for actuatorcontrol.py

Funktionsnavn	Parametre	Returværdi	Beskrivelse
sendData()	Ingen	Ingen	Sender brugerens loginoplysninger til serveren via en POST-forespørgsel. Håndterer svar fra serveren og omdirigerer til passende side ved succes eller viser fejlbeskeder ved fejl.

Table 9.5. Funktionsbeskrivelser for login-side JavaScript-koden

Test og resultater 10

10.1 Modultest

Her er en kort gennemgang af modultest. De fulde modultestfindes i bilagsmappen *test*[5].

10.1.1 Kamera

Testcase-ID	1
Testscenarie	Aflæs QR-kode
Test trin	qrReaderPiCamera2.py bliver kørt og QR-kode sættes foran kamera
Forudsætninger	Der er forbindelse mellem kamera og PythonScript
Test data	Kan ses i bilag [5.2.]
Forventet resultat	Terminalen udskriver værdien af QR-koden
Faktisk resultat	Terminalen udskriver værdien af QR-koden
Test status - OK/Fail	OK
Kommentar	NA

Table 10.1. Test af "Aflæs værdi af QR-kode"

10.1.2 Algoritme

Testcase-ID	1
Testscenarie	Sorter jetoner
Test trin	Insertion_sort() bliver kaldt på den indsamlede data fra scan_jetoner()
Forudsætninger	Dummy funktion til indsamling af data brugt
Test data	Kan ses i bilag [5.4.]
Forventet resultat	Den scannede data bliver sorteret
Faktisk resultat	Den scannede data bliver sorteret
Test status - OK/Fail	OK
Kommentar	NA

Table 10.2. Test af "Insertion_sort()"

Testcase-ID	2
Testscenarie	Optæl antal jetoner for hver værdi
Test trin	hash_map_convert() bliver kaldt på den indsamlede data fra scan_jetoner()
Forudsætninger	Dummy funktion til indsamling af data brugt
Test data	Kan ses i bilag [5.4.]
Forventet resultat	Den scannede data bliver inddelt i værdier og hvor mange der er
Faktisk resultat	Den scannede data bliver inddelt, man kan se hvor mange jetoner der er af hver værdi
Test status - OK/Fail	OK
Kommentar	NA

Table 10.3. Test af "hash_map_convert()"

10.1.3 Database

Testcase-ID	1
Testscenarie	Log in
Test trin	Log in forsøges i Dummy Python script
Forudsætninger	Der er forbindelse mellem databasen og PythonScript
Test data	Kan ses i bilag [5.5.]
Forventet resultat	Brugeren logges ind
Faktisk resultat	Brugeren logges ind
Test status - OK/Fail	OK
Kommentar	NA

Table 10.4. Test af "Log in"

Testcase-ID	2
Testscenarie	Deposit
Test trin	Der indsættes 1337 på brugeren konto
Forudsætninger	Der er forbindelse mellem databasen og PythonScript og brugeren er logget ind
Test data	Kan ses i bilag [5.5.]
Forventet resultat	Brugerens balance bliver korrekt opdateret
Faktisk resultat	Brugerens balance bliver korrekt opdateret
Test status - OK/Fail	OK
Kommentar	NA

Table 10.5. Test af "Deposit"

Testcase-ID	3
Testscenarie	Withdraw succes
Test trin	Der hæves 1337 på brugeren konto
Forudsætninger	Der er forbindelse mellem databasen og PythonScript og brugeren er logget ind
Test data	Kan ses i bilag [5.5.]
Forventet resultat	Brugeren modtager en email med en voucher med værdi 1337, og brugerens balance bliver korrekt opdateret
Faktisk resultat	Brugeren modtager en email med en voucher med værdi 1337, og bliver brugerens balance bliver korrekt opdateret
Test status - OK/Fail	OK
Kommentar	NA

Table 10.6. Test af "Withdraw succes"

Testcase-ID	4
Testscenarie	Withdraw failed
Test trin	Der hæves 7000 på brugeren konto
Forudsætninger	Der er forbindelse mellem databasen og PythonScript og brugeren er logget ind
Test data	Kan ses i bilag [5.5.]
Forventet resultat	Der vises en besked til brugeren "Error: Insufficient balances or unable to fetch balance"
Faktisk resultat	Der vises en besked til bruger "Error: Insufficient balances or unable to fetch balance"
Test status - OK/Fail	OK
Kommentar	NA

Table 10.7. Test af "Withdraw failed"

Testcase-ID	5
Testscenarie	Check Balance
Test trin	Menuen "Check Balance" vælges
Forudsætninger	Der er forbindelse mellem databasen og PythonScript og brugeren er logget ind
Test data	Kan ses i bilag [1]
Forventet resultat	Der vises en besked til brugeren "Your current balance: xxx"
Faktisk resultat	Der vises en besked til brugeren "Your current balance: xxx"
Test status - OK/Fail	OK
Kommentar	NA

Table 10.8. Test af "Check Balance"

10.1.4 Motor/Aktuator

Testcase-ID	1
Testscenarie	Test af aktuator med UART
Test trin	Der sendes nye CMP-værdier til servomotorerne vha. UART
Forudsætninger	PSoC er opsat til UART. Alle forbindelser sidder i de korrekte pins
Test data	Kan ses i bilag [5.3.]
Forventet resultat	Servomotorerne roterer 90 grader
Faktisk resultat	Servomotorerne roterer 90 grader
Test status - OK/Fail	OK
Kommentar	NA

Table 10.9. Test af aktuator med UART

Testcase-ID	2
Testscenarie	Test af aktuator med SPI
Test trin	Der sendes nye CMP-værdier til servomotorerne vha. SPI fra RPi
Forudsætninger	PSoC er opsat til SPI. Alle forbindelser sidder i de korrekte pins
Test data	Kan ses i bilag [5.3.]
Forventet resultat	Servomotorerne roterer 90 grader
Faktisk resultat	Servomotorerne roterer 90 grader
Test status - OK/Fail	OK
Kommentar	NA

Table 10.10. Test af aktuator med SPI

10.1.5 Brugerinterface

Testcase-ID	1
Testscenarie	Loginside
Test trin	Brugeren tilgår localhost:8080 i webbrowser
Forudsætninger	Serveren er aktiv
Test data	Kan ses i bilag [5.1.]
Forventet resultat	Webbrowseren viser indholdet i loginwebsite-template
Faktisk resultat	Webbrowseren viser indholdet i loginwebsite-template
Test status - OK/Fail	OK
Kommentar	NA

Table 10.11. Test af loginside

Testcase-ID	2
Testscenarie	Login
Test trin	Brugeren input'er valid logindata i formen
Forudsætninger	Serveren er aktiv
Test data	Kan ses i bilag [5.1.]
Forventet resultat	Webbrowseren viser en login succes-notifikation og omdirigerer brugeren til profilsiden
Faktisk resultat	Webbrowseren viser en login succes-notifikation og omdirigerer brugeren til profilsiden
Test status - OK/Fail	OK
Kommentar	I testen bliver fejlscenariet med invalid logindata også testet OK

Table 10.12. Test af valid login

Testcase-ID	3
Testscenarie	Deposit
Test trin	Brugeren vælger deposit, og vælger en mængde at deposit'e
Forudsætninger	Serveren er aktiv. Brugeren er logget ind.
Test data	Kan ses i bilag [5.1.]
Forventet resultat	Brugeren bliver omdirigeret til deposit-siden og bliver præsenteret for en række muligheder. Den valgte mængde bliver tillagt brugerens balance.
Faktisk resultat	Brugeren bliver omdirigeret til profilsiden og bliver præsenteret for en række muligheder. Den valgte mængde bliver tillagt brugerens balance.
Test status - OK/Fail	OK
Kommentar	I testen bliver omdirigering med "Back"-knap også testet OK

Table 10.13. Test af deposit

Testcase-ID	4
Testscenarie	Withdraw
Test trin	Brugeren vælger withdraw, og vælger en mængde at withdraw'e
Forudsætninger	Serveren er aktiv. Brugeren er logget ind.
Test data	Kan ses i bilag [5.1.]
Forventet resultat	Brugeren bliver omdirigeret til withdraw-siden og bliver præsenteret for en række muligheder. Den valgte mængde bliver taget fra brugerens balance.
Faktisk resultat	Brugeren bliver omdirigeret til withdraw-siden og bliver præsenteret for en række muligheder. Den valgte mængde bliver taget fra brugerens balance.
Test status - OK/Fail	OK
Kommentar	I testen bliver withdraw med custom value også testet OK. I testen bliver omdirigering med "Back"-knap også testet OK

Table 10.14. Test af withdraw

Testcase-ID	5
Testscenarie	Check balance
Test trin	Brugeren vælger check balance
Forudsætninger	Serveren er aktiv. Brugeren er logget ind.
Test data	Kan ses i bilag [5.1.]
Forventet resultat	Brugeren bliver omdirigeret til check balance-siden, hvor deres balance bliver displayet.
Faktisk resultat	Brugeren bliver omdirigeret til check balance-siden, hvor deres balance bliver displayet.
Test status - OK/Fail	OK
Kommentar	I testen bliver omdirigering med "Back"-knap også testet OK

Table 10.15. Test af check balance

Testcase-ID	6
Testscenarie	Logout
Test trin	Brugeren vælger logout
Forudsætninger	Serveren er aktiv. Brugeren er logget ind.
Test data	Kan ses i bilag [5.1.]
Forventet resultat	Brugeren bliver logget ud og omdirigeret til loginsiden.
Faktisk resultat	Brugeren bliver logget ud og omdirigeret til loginsiden.
Test status - OK/Fail	OK
Kommentar	NA

Table 10.16. Test af valid login

10.2 Integrationstest

Når alle modulerne er blevet designet og testet, skal der udføres integrationstest. For dette projekt bliver integrationen udført bottom up, altså kobles de individuelle moduler på ét ad gangen og færdigintegreres med kontrolenheden, før det næste modul forsøges integreret.

10.2.1 Kontrolenhed og GUI

Det er mest nærliggende at integrere GUI med kontrolenheden først, fordi den i forvejen er koblet til kontrolenheden ifm. modultest. Backend-server for GUI er genimplementeret, så det kommer til at være det program, der agerer koblingspunktet til de øvrige moduler i kontrolenheden. I testmodulet bliver deposit-input fra frontend til backend parset som int-værdier, der bliver tillagt dummy-brugernes balance. I den endelige implementering skal deposit-dataet ikke komme i form at brugerinput, så deposit-knapper og –forms bliver fjernet fra frontend. Dataet fra deposit-routens POST-metode bliver opbevaret i en temp list, der kan parses til algoritme og database senere.

10.2.2 Kontrolenhed og kamera

Kamerakoden bliver implementeret som en funktion i kontrolenheden. I deposit-routens GET-metode bliver tilføjet en thread, der starter kamerafunktionen og sætter et aktivt event, så kameraet begynder at scanne, når deposit-routen tilgås. Den scannede data bliver tilføjet til temp-listen.

Logout-knappen på frontend bliver ændret, så den router brugeren til en ny /stopscanning-route, hvor kameraeventet stoppes, temp-listen bliver sendt videre til /deposit-routen, og brugeren bliver

redirected tilbage til profilsiden.

Kameratråden bliver kaldt med copy_current_request_context, så den også er en del af brugerens loginsession. Fordi tråden er en daemon-thread, bliver den fjernet, når koden er færdig med at eksekvere.

10.2.3 Kontrolenhed og database

Funktionerne fra databasemodulet importeres. Brugerens indtastede login-data bliver brugt i databaselogin-funktionen, der returner valide værdier, hvis login er verificeret, og null-værdier, hvis det ikke er verificeret. Ved succesfuldt login bliver brugerens data gemt i et User-objekt, der bliver sat til current user i flask login.

I /withdraw- og /deposit-route'erne bliver input-værdi(er) summet, og databasens deposit- og withdraw-funktioner bliver kaldt med værdien og current_user.id, så brugerens balance bliver opdateret på databasen. På lignende vis bliver databasens db_check_balance()-funktion kaldt alle steder, hvor brugerens balance er relevant.

10.2.4 Kontrolenhed og algoritme

Algoritmens funktioner importeres i kontrolenheden, og bliver kaldt i /deposit-routen's POST-metode, så hashmap og sorteret liste over de indsatte jetoner bliver udskrevet i terminalen. De læste hashmap-værdier bliver tilføjet til et session hashmap, som holder styr på, hvor mange af hver type jeton er blevet deposit'et i sessionen, med henblik på at holde dem op imod en kapacitetstæller.

Da kapacitetstælleren ikke er implementeret, udfører session-hashmap'et desværre ikke nogen funktion i systemet for nu. Der opstår også det problem, at sessionen bliver afsluttet ved logout, og jetontællerne derfor ikke persister fra én brugersession til en anden. For at få dette til at fungere efter intentionen, ville det kræve, at hashmap'et blev gemt på en måde, der spænder over flere sessioner, eksempelvis ved at gemme det i databasen eller i en lokal json-fil, der bliver åbnet ved sessionens start. Dette er dog ikke forsøgt implementeret i projektets nuværende form.

Da det fra starten er blevet fravalgt at forsøge at sortere de fysiske jetoner, har sorteringsalgoritmen kun til formål at simulere, hvordan jetonerne ville blive sorteret. Derfor er der ikke oprettet en tilsvarende fysisk sorteringsfunktion.

10.2.5 Kamera og aktuatorer

Funktionen til styring af aktuatorerne bliver importeret, så act_send_command()-funktionen kan kaldes inde i kamerafunktionen. Aktuatorerne bliver kaldt i en rækkefølge, hvor der opstår en slags "slusesystem" til jetonerne.

10.2.6 Samlet integrationstest

Når først alle modulerne er blevet integreret, laves en samlet integrationstest.

Ved login på GUI kan der kun logges ind med de brugere, der er oprettet i databasen, og ikke eksisterende brugere kan ikke logge ind. Ved logout bliver brugeren logget ud, og kan på grund af flask-login's @login_required-funktionalitet ikke tilgå de "låste" routes såsom deposit længere, kun loginsiden.

Ved deposit kan brugeren scanne en jeton foran kameraet, og aktuatorernes "slusesystem" aktiverer. Når man trykker "done", bliver brugerens balance på databasen opdateret, og der bliver i terminalen udskrevet hashmap og sorteret liste over de scannede jetoner. Kameraet stopper også med at scanne og registrere jetoner.

Ved "withdraw" og "check balance" kan man tilsvarende opdatere og tilgå brugerbalancen fra databasen via GUI'et.

Systemet har altså en succesfuld integration, hvor de forskellige moduler kan sende og tilgå data og signaler til hinanden og kontrolenheden efter intentionen.

Det samlede system kan ses på figur 10.1.

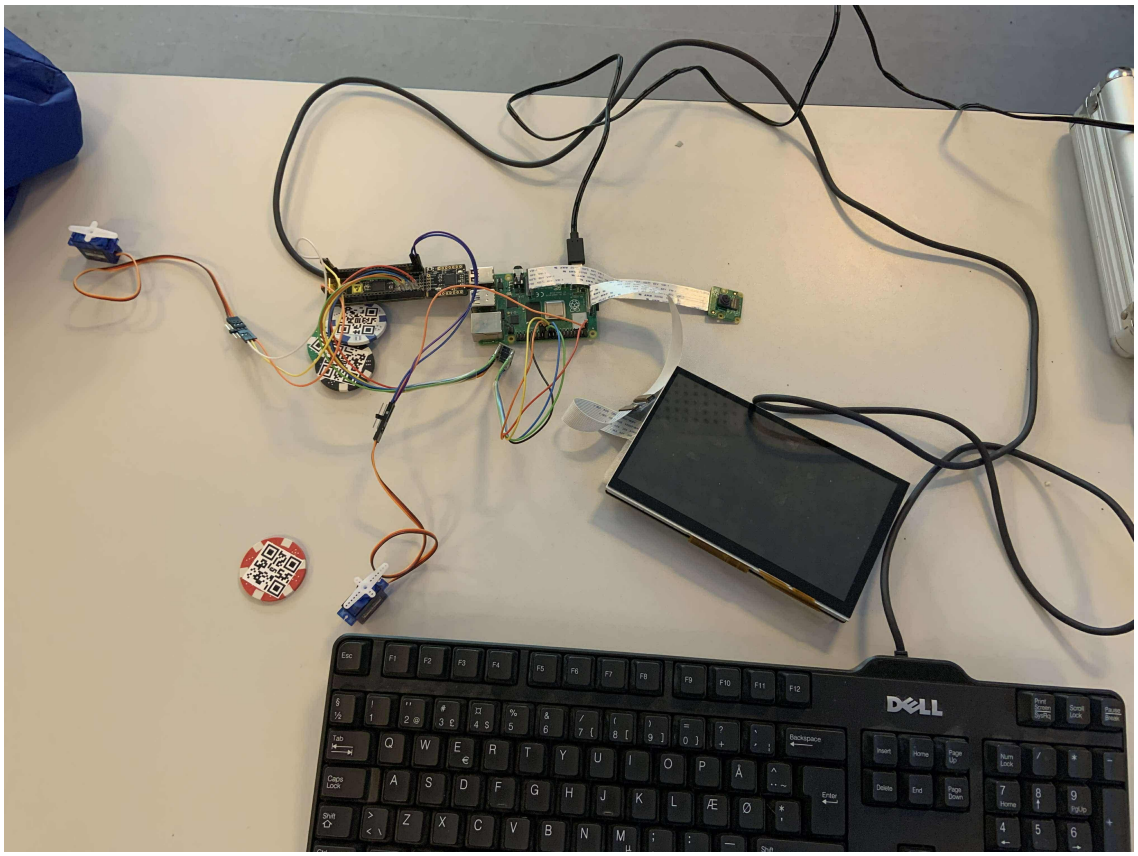

Figure 10.1. CCC - Version 1.0

Accepttest 11

Accepttesten for projektet, *Casino Chip Counter*, blev udarbejdet for at sikre, at systemet lever op til de funktionelle og ikke-funktionelle krav, der blev fastlagt i starten af projektet. En detaljeret gennemgang af alle dele af accepttesten kan findes i det fulde testdokument[1.6.].

11.1 Kort opsummering af accepttesten

Usecase	Vurdering
1. Deposit chips	Delvis OK
2. Withdraw chips	OK
3. Check Balance	OK
4. Empty System	Fail

Table 11.1. Funktionelle krav

Krav	Vurdering
Systemet sorterer digitalt	OK
Kamera kan scanne QR koder	OK
Porte kan åbne og lukke automatisk	OK
Systemet har webbaseret database	OK
Systemet lukker for deposit ved max kapacitet	Fail
Systemet har brugerinterface	OK
Bruger kan logge ind	OK
Bruger kan stoppe system	OK
Bruger kan se balance	OK
Bruger får kvittering på e-mail	OK
Bruger kan se balancehistorik	Fail
System stopper automatisk ved inaktivitet	Fail
Systemet fører statistik	Fail
Systemet scanner på maks 2 sekunder	Fail
UI er intuitiv	Fail
Bruger kan se balance før og efter brug	OK
Fejlrate på maks 5%	OK

Table 11.2. Ikke-funktionelle krav

Diskussion 12

12.1 Diskussion af resultater

Flere resultater fra accepttesten blev ikke godkendt, primært fordi flere tiltænkte funktioner ikke blev implementeret. Dette skyldes forskellige prioriteringer og udfordringer under projektets udvikling.

Ved projektets start var gruppen ikke sikre på, hvor omstændige semesterets andre kurser ville være. Derfor blev der truffet beslutninger om projektet, der senere ville vise sig at være for ambitiøse i forhold til tiden der var til rådighed.

En af de oprindelige designidéer var en automatisk logout-funktion efter 20-30 sekunders inaktivitet. Denne blev dog nedprioriteret, da den ikke var essentiel for hovedscenarierne, da det er en extension. Grundet tidspres blev der ikke arbejdet videre med denne funktionalitet.

En anden funktionalitet, der blev nedprioriteret, var logning af antallet af indsatte jetoner fordelt på deres værdi. Dette skulle bruges til maintenance, som er beskrevet i Use Case 4[1.5.]. Denne funktion viste sig dog at være udfordrende at implementere, især med hensyn til at spore jetoner på tværs af brugersessioner. For at fokusere på mere centrale use cases blev funktionen droppet, selvom dele af den stadig er synlige, fx i `maintenance.html`-templaten og sessionens hash-map-struktur.

Vi har efterfølgende tilpasset diagrammerne i applikationsmodellen for at reflektere systemets faktiske adfærd. De tidligere versioner af diagrammerne er stadig tilgængelige i bilagene.[4.5.[4.6.]

Overordnet set er resultatet af accepttesten som forventet. Langt størstedelen af de ikke-accepterete punkter er nogen, som er blevet nedprioriteret i løbet af projektes forløb. Hovedparten af produktets kernefunktionalitet er implementeret i tilfredsstillende grad.

12.2 Diskussion af proces

Gruppen har været mangelfuld i review af eget materiale. Dette har medført en del af rettelser i de sidste arbejdsprocesser. Hvor nogle fundamentale ting blev rettet / forbedret. Dette kunne være undgået med en review struktur, der satte nogle gruppe medlemmer som reviewere af lavet arbejde. En anden ide kunne være, at arbejde sammen som gruppe om de første faser, dermed sikre at alle var indforstået og havde bedre overblik over systemet og dennes sammensætning. Dette ville være tidskrævende, og grundet større tidspres i løbet af semesteret, kan denne løsning også være med til at give mere arbejde end gavn.

Kommunikationen mellem medlemmerne har været god, hvilket gjorde det nemmere at samarbejde på tværs af faserne. Der har været et godt fremmøde samt engagement i projektet. Dette medførte dermed en fint integration af systemet, selvom ikke alle funktionaliteter er blevet implementeret. Overordnet anses projektet ikke som en negativ oplevelse.

Konklusion 13

Konklusion

Projektet *Casino Chip Counter* har haft som mål at designe og implementere et funktionelt system, der kan tælle, registrere og sortere casinojetoner baseret på QR-koder. Med fokus på praktisk anvendelse af teknologier og metoder fra semesterets undervisning har gruppen med succes gennemført en iterativ udviklingsproces, hvor den agile arbejdsmetode Scrum blev anvendt som ramme.

Systemet blev udviklet med en Raspberry Pi 4 som kontrolenhed og en PSoC 5 LP som styreenhed. Kombinationen af Python og Flask muliggjorde en effektiv integration af moduler samt udviklingen af en brugervenlig, webbaseret GUI. En insertion sort algoritme blev anvendt til sortering af jetoner. En MySQL-database blev anvendt til at håndtere brugerdata.

Selvom visse funktionaliteter, såsom fysisk sortering af jetoner, ikke blev implementeret, demonstrerer prototypen de centrale krav og funktionaliteter specificeret i projektoplægget. Accepttesten bekræfter, at systemet opfylder sine kerneformål, omend der er plads til forbedringer og udvidelser i fremtidige iterationer.

Projektet har givet gruppen værdifulde erfaringer inden for projektledelse, samarbejde og brug af en iterativ arbejdsmodel. Anvendelsen af Scrum gav både positive og udfordrende indsigter, hvilket har styrket gruppens forståelse for planlægning og arbejdsmetoder. Til fremtidige projekter vil gruppen særligt fokusere på at udnytte de planlægningsværktøjer og iterative processer, som vi har lært at bruge effektivt i løbet af projektet.

Fremtidigt arbejde 14

I løbet af projektet har der været en del afgrænsninger som konsekvens af forskellige problemstillinger. Dette betyder, at der er rig mulighed for at videreudvikle på systemet.

Som det første fremtidige arbejde ville det være oplagt at implementere rettelser af de problemer, der medførte fejl i accepttest, eftersom mange af dem er småfejl i tekst eller rækkefølge af logikken.

I sin nuværende form er det svært at opstille og teste meget af funktionaliteten, især den tiltænkte sluse med aktuatorerne. Det ville give mening at konstruere en kasse til prototypen, som ville give et bedre overblik over, hvordan systemet opfører sig, og som ville gøre det nemmere at transportere.

Som det næste ville det være en prioritet at få udarbejdet den ikke-implementerede funktionalitet fra use cases 1-3, heriblandt automatisk log out ved inaktivitet.

Dette lægger op til efterfølgende at tage hul på UC4. Det medfører videreudvikling på den simulerede sortering, herunder ordentlig tracking af indsatte jetoner på tværs af sessioner og virtuelle jetoncontainere med en bestemt kapacitet.

Når først den simulerede sortering og de øvrige funktionaliterer er færdigudviklede, ville en mere ambitiøs udvikling på længere sigt være at genbesøge projektkonceptets vinkel med fysisk sortering, herunder aktuatorkontrol til omfordeling imellem forskellige containere med bestemte jetonværdier.

Efterfølgende ville udgangspunktet være i MoSCoW, hvor man vil starte fra should og bevæge sig videre i prioriteringen. Herfra kan overvejes og arbejdes på andre potentielle ønsker en kunde kunne have for produktet.

Bilag 15

Bilagsoversigt

- **Projekt**
  1. **Bilag 1** Kravspecifikation
     1.1. Oplæg til semesterprojekt 3 (LGJ).pptx
     1.2. Projektformulering v. 1.pdf
     1.3. Kravspecifikation v. 1.pdf
     1.4. Fully-Dressed Use Cases v. 1.pdf
     1.5. Fully-Dressed Use Cases v. 2.pdf
     1.6. Accepttestspecifikation.pdf
     1.7. Rigt Billede.pdf
  2. **Bilag 2** Analyse
     2.1. Moscow analyse.pdf
     2.2. Risikoanalyse og teknisk analyse.pdf
     2.3. Risikomatrice.pdf
     2.4. Algoritme Analyse.pdf
  3. **Bilag 3** Systemarkitektur
     3.1. BDD og blokbeskrivelse.pdf
     3.2. IBD og Signalbeskrivelse.pdf
  4. **Bilag 4** Diagrammer
     4.1. Aktør Kontekst.pdf
     4.2. UC Diagram CCC.pdf
     4.3. STM - Sortering Algoritme.pdf
     4.4. Domænemodel CCC.pdf
     4.5. Klassediagrammer.pdf
     4.6. Sekvensdiagrammer.pdf
  5. **Bilag 5** Test
     5.1. GUI Modultest.pdf
     5.2. Kamera Modultest.pdf
     5.3. Aktuator Modultest.pdf
     5.4. Algoritme Modultest.pdf
     5.5. Database Modultest.pdf
     5.6. Integrationstest.pdf
     5.7. Accepttest.pdf
  6. **Bilag 6** Source Code
  7. **Bilag 7** Datablade
     7.1. DFRobot 0678 touchscreen datasheet.pdf
     7.2. DFRobot 0678 touchscreen datasheet2.pdf
     7.3. Infineon CY8CKIT-059 PSoC 5LP Prototyping v01.pdf
     7.4. raspberry-pi-4-datasheet.pdf
     7.5. SG90-datasheet.pdf
  8. **Bilag 8** Misc
     8.1. QR-kode-print.pdf

- **Proces**

9. **Bilag 9** Procesbeskrivelse.pdf
10. **Bilag 10** Scrum
    - 10.1. Sprint 1 Retrospective.pdf
    - 10.2. Sprint 1 Retrospective.pdf
    - 10.3. Sprint 1 Retrospective.pdf
    - 10.4. Sprint 1 Retrospective.pdf
    - 10.5. Scrumboard
    - 10.6. Trello - Sprint 1 Taskboard
    - 10.7. Trello - Sprint 2 Taskboard
    - 10.8. Trello - Sprint 3 Taskboard
    - 10.9. Trello - Sprint 4 Taskboard
11. **Bilag 11** Mødeindkaldelse
    - 11.1. Gruppemøde 20-09-2024.docx
    - 11.2. Gruppemøde 27-09-2024.docx
    - 11.3. Gruppemøde 04-10-2024.docx
    - 11.4. Gruppemøde 11-10-2024.docx
    - 11.5. Gruppemøde 01-11-2024.docx
    - 11.6. Gruppemøde 08-11-2024.docx
    - 11.7. Gruppemøde 15-11-2024.docx
    - 11.8. Gruppemøde 22-11-2024.docx
    - 11.9. Dagsorden skabelon.docx
12. **Bilag 12** Mødereferater
    - 12.1. Gruppemøde 13-09-2024.docx
    - 12.2. Vejledermøde 17-09-2024.docx
    - 12.3. Gruppemøde 20-09-2024.docx
    - 12.4. Gruppemøde 27-09-2024.docx
    - 12.5. Gruppemøde 4-10-2024.docx
    - 12.6. Gruppemøde 11-10-2024.docx
    - 12.7. Gruppemøde 01-11-2024.docx
    - 12.8. Gruppemøde 08-11-2024.docx
    - 12.9. Gruppemøde 15-11-2024.docx
    - 12.10. Gruppemøde 22-11-2024.docx
    - 12.11. Vejledermøde 29-11-2024.docx
    - 12.12. Referat Skabelon.docx
13. **Bilag 13** Insight Profiler
    - 13.1. AndreasGadgaard - 29 Supporterende Hjælper (Klassisk).pdf
    - 13.2. EmilHolmRiis-45-Inspirerende-Motivator-Adaptiv (2).pdf
    - 13.3. JesperLundPedersen - 31 Koordinerende Supporter.pdf
    - 13.4. KristianStausholm - 30 Hjælpende Supporter (Klassisk) (3).pdf
    - 13.5. MaciejBrylski - 133 Kreativ Observerende Koordinator (Klassisk).pdf
    - 13.6. MikkelMortensen - 31 Koordinerende Supporter (Klassisk).pdf
    - 13.7. TobiasKonradNielsen - 14 Koordinerende Observatør (Fokuseret).pdf
14. **Bilag 14** Tidsplaner
    - 14.1. 3. Semesterprojekt Tidsplan v. 1.pdf
    - 14.2. 3. Semesterprojekt Tidsplan v. 2.pdf
15. **Bilag 15** Samarbejdskontrakt.pdf

Referencer

16. Weiss, M. A. (2014). *Data Structures and Algorithm Analysis.* (pp. 292–295). Pearson.