

# SW4PRJ4-01 Semesterprojekt 4

## Purchase4Less

Projektgruppe 5

Studerende	Studienummer
Mustafa Eren Sagirkaya	202209872
Shadi Abou Staite	202203522
Daahir Abukar Hussein	202209868
Ruben Gullborg Frandsen	202208849
Altaf Hussein Rezai	202209871
Mads Dittmann Villadsen	202209869
Arne Skou Pedersen	202110456
Nahom Melakehail Tadesse	201808281

**Vejleder:** Jakob Heldgaard Kristensen

**Anslag:** 71.278

## Resumé

Dette projekt, udviklet som en del af 4. semester på Aarhus Universitet, har skabt webapplikationen *Purchase4Less*, som hjælper brugere med at gøre deres daglige indkøb smartere og billigere. Applikationen giver brugerne mulighed for at oprette og organisere indkøbslister samt sammenligne priser fra forskellige supermarkeder, så de nemt kan finde det bedste tilbud og spare både tid og penge.

Projektet er udviklet iterativt ved hjælp af moderne værktøjer og teknologier. Frontenden er bygget i React og TypeScript med fokus på et brugervenligt design, mens backend er baseret på en solid MVC-arkitektur, der understøtter CRUD-operationer via en database.

Under udviklingen blev der lagt vægt på teamwork og struktur ved at bruge Kanban-værktøjer som Trello til projektstyring. Kombinationen af teoretisk viden fra studiet og praktiske udviklingsmetoder har resulteret i en applikation, der er fleksibel, skalerbar og klar til fremtidige forbedringer – herunder mere avancerede søgefunktioner. *Purchase4Less* demonstrerer, hvordan teknologi kan gøre hverdagen enklere og mere økonomisk for brugerne.

## Abstract

This project, developed as part of the 4th semester at Aarhus University, has created the web application *Purchase4Less*, which helps users make their daily shopping smarter and more affordable. The application allows users to create and organize shopping lists as well as compare prices from different supermarkets, enabling them to easily find the best deals and save both time and money.

The project was developed iteratively using modern tools and technologies. The frontend is built using React and TypeScript with a focus on user-friendly design, while the backend is based on a solid MVC architecture that supports CRUD operations using a database.

During development, emphasis was placed on teamwork and structure by using Kanban-tools like Trello for project management. The combination of theoretical knowledge from the study program and practical development methods has resulted in an application that is flexible, scalable, and ready for future improvements – including more advanced search functions. *Purchase4Less* demonstrates how technology can make everyday life simpler and more economical for users.

<b>1. Indledning .....</b>	<b>4</b>
1.1 Projektide.....	4
1.2 Problemformulering.....	5
1.3 Projektformulering.....	5
1.4 Ressourcer og projektafgrænsning.....	5
1.5 Arbejdsfordeling .....	6
<b>2. Kravspecifikation .....</b>	<b>7</b>
2.1 User Stories .....	7
2.2 Epics .....	8
2.3 Gherkin.....	9
<b>3. Afgrænsning .....</b>	<b>13</b>
<b>4. Metode og proces .....</b>	<b>15</b>
4.1 Metodeanvendelse .....	15
4.2 Gruppedannelse .....	16
4.3 Arbejdsmetode.....	16
4.4 Arbejdsfordeling .....	17
4.5 Tidsplan og iterationer.....	17
<b>5. Analyse .....</b>	<b>18</b>
5.1 Risikoanalyse .....	18
5.2 Teknologianalyse .....	19
<b>6. Arkitektur .....</b>	<b>21</b>
6.1 Systemarkitektur .....	21
6.1.1 Domænemodel .....	21
6.1.2 Klassediagram .....	22
6.1.3 Systemsekvensdiagrammer.....	23
6.1.4. ER-diagram.....	30
6.1.5. C4-model .....	31
6.2. Backend MVC-arkitektur .....	35
<b>7. Design &amp; implementering .....</b>	<b>36</b>
7.1. UI/UX.....	36
7.1.1. Designproces og Skitser .....	36
7.2. Design overvejelser .....	37
7.3. Database .....	38
7.4. User Stories .....	38
User Story 1: Søge efter en specifik vare .....	39
User Story 2: Oprette en ny indkøbsliste.....	41

User Story 3: Ændre en indkøbsliste .....	45
User Story 4: Sammenlign priser .....	48
7.5.    Yderligere implementeringer.....	51
7.5.1.    WebScraper .....	51
7.5.2.    CRON JOB .....	55
<b>8. Test .....</b>	<b>55</b>
8.1.    Teststrategi.....	55
8.2.    ZOMBIE-teststrategi.....	56
8.3.    Test af backend .....	56
8.4.    Integrationstest .....	56
<b>9. Resultater.....</b>	<b>57</b>
9.1.    Accepttestspecifikation .....	57
9.2.    Opsummering af resultaterne.....	57
<b>Diskussion .....</b>	<b>58</b>
<b>Konklusion.....</b>	<b>59</b>
<b>Fremtidigt arbejde.....</b>	<b>60</b>
<b>Litteraturliste .....</b>	<b>61</b>
<b>Bilag .....</b>	<b>61</b>
[6.1].....	Error! Bookmark not defined.

## Indholdsfortegnelse

## Forord

Denne rapport er udarbejdet som en del af 4. semesterprojektet på Aarhus Universitet af 8 studerende fra Diplomingeniøruddannelsen i Softwareteknologi. Formålet med rapporten er at dokumentere projektets udvikling, herunder de tekniske løsninger og metodiske tilgange, der er blevet anvendt. Rapporten fokuserer på de faglige kompetencer, samarbejdsprocesser og teknologier, der er anvendt til at opfylde projektets mål.

Projektet demonstrerer anvendelsen af teori og praksis gennem udviklingen af en webapplikation, der understøtter optimering af dagligvareindkøb. I rapporten beskrives design- og implementeringsvalgene, projektstyringsværktøjer og de læringsmål, der har været centrale for forløbet.

Rapporten er afleveret den 13-06-2024 og indeholder et samlet antal tegn: 71278.

# 1. Indledning

Her præsenteres gruppens projektide, problemformulering, projektformulering, ressourcer og afgrænsninger samt arbejdsfordeling.

## 1.1 Projektide

Gruppens projektidé fokuserer på at udvikle en webapplikation, Purchase4Less, der gør det muligt for brugere at oprette og administrere digitale indkøbslister samt finde de bedste tilbud på varer fra forskellige supermarkeder. Applikationen er designet til at gøre indkøbsoplevelsen mere effektiv ved at give brugerne et klart overblik over priserne fra supermarkederne, så de kan træffe bedre økonomiske beslutninger.

Ved at kombinere teknologier som React og TypeScript til frontenden samt en solid backend-struktur med CRUD-operationer og en velorganiseret database, kan Purchase4Less hjælpe brugere med både at spare tid og penge. Målet er at tilbyde en løsning, der gør indkøbsprocessen smartere og mere overskuelig, samtidig med at den tilpasses brugernes behov.

Nedenstående Figur 1 illustrerer gruppens projektide.



Figur 1: Illustration over projektide

## 1.2 Problemformulering

Hvordan kan man udvikle en brugervenlig og effektiv webindkøbsapp, der automatisk indhenter de bedste tilbud fra nærliggende butikker, så der skabes et produkt, der både sparer tid og penge for brugerne?

### Underspørgsmål:

- Hvordan kan man designe en intuitiv og letanvendelig brugergrænseflade, der gør det nemt for brugerne at tilføje varer til deres indkøbsliste?
- Hvordan kan man optimere backend-systemet, så det effektivt henter produkter og priser fra supermarkeder som Rema1000, Spar og Bilka?
- Hvordan kan man gennemføre grundig softwaretest for at sikre, at appen fungerer fejlfrit og leverer nøjagtige produktinformationer og korrekt prissammenligning?

## 1.3 Projektformulering

I takt med digitaliseringen af hverdagen opstår der et behov for smartere løsninger, der gør daglige rutiner som indkøb nemmere og mere effektive. Purchase4Less er udviklet som et svar på dette behov.

Applikationen giver brugerne mulighed for at oprette digitale indkøbslister og sammenligne priser på varer fra supermarkeder som Rema1000, Spar og Bilka. Formålet er at hjælpe brugerne med at finde de bedste tilbud og dermed spare både tid og penge.

Med en intuitiv brugergrænseflade og en backend, der hurtigt og præcist henter produkt- og prisdata, sigter projektet mod at gøre indkøbsprocessen enkel og overskuelig for brugerne. Gruppen har taget udgangspunkt i viden opnået fra kurserne på 4. semester for at sikre, at applikationen er pålidelig, brugervenlig og klar til videreudvikling.

## 1.4 Ressourcer og projektafgrænsning

Projektet er baseret på de teknologier, metoder og værktøjer, som gruppen er blevet introduceret til og har opnået erfaring med gennem 4. semester og tidligere semestre. I projektforløbet er der lagt vægt på at anvende denne viden i praksis og samtidig kunne løse de opstillede problemstillinger.

Gruppen har modtaget støtte og vejledning fra vejleder Jakob Heldgaard Kristensen, som gennem hele processen har bidraget med vejledning i forhold til udviklingsmetoder og hjælp til at håndtere udfordringer.

Projektet er udført inden for rammerne af følgende krav og afgrænsninger, som sikrer en fokuseret og relevant anvendelse af de kompetencer, vi har opnået på 4. semester:

- Udvikle applikationer med grafiske brugergrænseflader, databaser og netværkskommunikation
- Anvende teknikker, metoder og værktøjer til softwaretest
- Anvende objektorienteret analyse og design i systemudvikling

Disse krav definerer projektets omfang og sikrer, at det holder sig inden for de faglige rammer, som er blevet præsenteret i undervisningen.

## 1.5 Arbejdsfordeling

Hvert gruppemedlem har hver især ansvaret for forskellige områder af projektet.

Tabel over arbejdsfordeling									
Ansvarlig:	FÆLLES	Shadi	Ruben	Daahir	Arne	Altaf	Mustafa	Nahom	Mads
Rapport									
Indledning	X								
Kravspecifikation	X								
Afgræsning	X								
Metode og proces	X								
Analyse	X								
Arkitektur	X								
Design & Implementering	X								
User Story 1							X	X	
User Story 2		X		X					
User Story 3		X		X			X	X	
User Story 4		X		X					
User Story 5 (bilag)			X			X			
User Story 6 (bilag)			X			X			
User Story 7 (bilag)						X			X
User Story 8 (bilag)					X				X
User Story 9 (bilag)			X		X				X
User Story 10 (bilag)					X				X
Webscraper			X						
Test		X		X	X	X			
Resultater	X								
Diskussion	X								
Konklusion	X								
Fremtidigt arbejde	X								
Implementering									
Front-end	X								
Back-end	X								

Tabel 1: Projektets arbejdsfordeling



## 2. Kravspecifikation

Denne kravspecifikation beskriver de funktionelle krav og user stories for webapplikationen Purchase4Less.

### 2.1 User Stories

Ved udarbejdelsen af nedenstående user stories, er der taget udgangspunkt i principper og metoder beskrevet i bilag [6.1]:

**1. [Søge efter én specifik vare]**

Som en shopper  
vil jeg kunne søge efter én specifik vare,  
for at jeg kan vælge den billigste mulighed.

**2. [Oprette en ny indkøbsliste]**

Som en shopper  
vil jeg kunne oprette en ny indkøbsliste,  
så jeg kan have flere lister til forskellige behov.

**3. [Ændre en indkøbsliste]**

Som en shopper  
vil jeg kunne ændre i en indkøbsliste,  
så jeg kan tilpasse den med de varer, jeg har brug for at købe.

**4. [Sammenlign priser]**

Som en shopper  
vil jeg kunne få vist den samlede pris fra forskellige butikker,  
så jeg kan se, hvor jeg kan handle ind billigst.

**5. [Google Maps]**

Som en shopper  
vil jeg kunne få vist de nærmeste butikker indenfor en valgt radius, der har de billigst ønskede varer,  
så jeg kan spare tid på transport.

**6. [Notifikationer]**

Som en shopper  
vil jeg kunne få notifikationer om prisændringer på ønskede varer,  
så jeg kan købe til den bedste pris, på det rette tidspunkt.

**7. [Oprette bruger og login]**

Som en shopper  
vil jeg kunne oprette en profil,  
så jeg senere kan logge ind og gemme specifikke varer og indkøbslister.

**8. [Vælge specifikke butikker]**

Som en shopper  
vil jeg kunne filtrere specifikke butikker fra,  
så jeg ikke får vist butikker, som jeg ikke er interesseret i.

**9. [Se statistikker som administrator]**

Som en administrator  
vil jeg kunne tilgå en dashboard,  
så jeg kan få overblik over relevante statistikker.

**10. [Tilføje fødevarebutikker]**

Som administrator  
vil jeg kunne tilføje flere fødevarebutikker,  
så brugerne har flere produkter at vælge imellem.

## 2.2 Epics

Epics er overordnede kategorier, der bruges til at organisere relaterede user stories i et projekt. De repræsenterer større funktionelle områder eller mål, som hjælper med at strukturere udviklingsarbejdet og sikre, at alle user stories adresseres. Ved at opdele projektet i epics bliver det nemmere at prioritere, planlægge og implementere funktioner i et overskueligt og fokuseret workflow.

Ved udarbejdelsen af nedenstående epics, er der taget udgangspunkt i principper og metoder beskrevet i bilag [6.1]:

**Epic 1: Bruger login**

- US7: Oprette bruger og login

**Epic 2: Listehåndtering**

- US2: Oprette en ny indkøbsliste
- US3: Ændre en indkøbsliste
- US4: Sammenlign priser

**Epic 3: Søgning og filtrering**

- US1: Søge efter én specifik vare
- US5: Google Maps
- US8: Vælge specifikke butikker

**Epic 4:** Notifikationer

- US6: Notifikationer

**Epic 5:** Administrator muligheder

- US9: Se statistikker som administrator
- US10: Tilføje fødevarebutikker

## 2.3 Gherkin

Gherkin-syntaksen er en struktureret måde at beskrive accepttests på. Den benyttes til at definere testscenarier for softwarefunktioner, hvilket sikrer en fælles forståelse blandt udviklere, testere og brugere. Gherkin anvender et naturligt sprogformat kombineret med nøgleord som "Egenskab", "Scenarie", "Givet", "Når" og "Så". Dette gør det muligt at skrive klare og letforståelige tests, der fungerer som dokumentation og testcases.

Gherkin bruges i dette projekt til at sikre, at alle user stories fungerer som forventet og opfylder de stillede krav. Det gør testcases enkle og strukturerede, så funktionaliteten kan verificeres systematisk. Nedenfor findes Gherkin-baserede accepttestcases for hver user story, som gør det muligt at teste og validere systemets krav. Arbejdet er baseret på metoder fra Bilag [6.1] og værktøjet Cucumber.<sup>1</sup>

**User Story 1: Søge efter én specifik vare**

**Egenskab:** *Søgefunktion til at finde den billigste pris for en specifik vare.*

**Scenarie:** Brugeren vil finde ud af, hvor et givent produkt er billigst

- o **Givet:** Brugeren er logget ind.
- o **Givet:** Produktet "Minimælk" findes i databasen.
- o **Givet:** Brugeren har navigeret til "Find produkt"-siden.
- o **Når:** Brugeren indtaster "Minimælk" og trykker "Søg".
- o **Så:** Får brugeren en oversigt over, hvad minimælk koster i Bilka, Rema 1000 og SPAR.

**Scenarie:** *Det indtastede produkt findes ikke*

- o **Givet:** Brugeren er logget ind.
- o **Givet:** Produktet "Vin" ikke findes i databasen.
- o **Givet:** Brugeren har navigeret til "Find produkt"-siden.
- o **Når:** Brugeren indtaster "Vin" og trykker enter på sit tastatur.
- o **Så:** Modtager brugeren en besked med meddelelsen "Ingen resultater fundet."

---

<sup>1</sup> <https://cucumber.io/docs/gherkin/reference/>

**User Story 2: Oprette en ny indkøbsliste**

**Egenskab:** *Oprette en ny indkøbsliste og vise oversigt over eksisterende lister.*

**Scenarie:** Brugeren vil oprette en ny indkøbsliste

- o **Givet:** Brugeren er logget ind.
- o **Givet:** Brugeren har navigeret til "Mine shoppinglister"-siden.
- o **Når:** Brugeren trykker på "Opret ny indkøbsliste" knappen og indtaster navnet "Mandagsindkøb"
- o **Når:** Brugeren trykker på "Opret" knappen.
- o **Så:** Får brugeren en success-toast notifikation om, at indkøbslisten er oprettet.
- o **Så:** Ses en indkøbsliste med navnet "Mandagsindkøb" i oversigten over eksisterende indkøbslister.

**Scenarie:** Det indtastede indkøbsliste navn er tomt

- o **Givet:** Brugeren er logget ind.
- o **Givet:** Brugeren har navigeret til "Mine shoppinglister"-siden.
- o **Når:** Brugeren trykker på "Opret ny indkøbsliste" og ikke indtaster et navn
- o **Når:** Brugeren trykker på "Opret" knappen.
- o **Så:** Får brugeren en error-toast notifikation om at der er en fejl, og bliver prompted til at give et navn til sin indkøbsliste.

**User Story 3: Ændre en indkøbsliste**

**Egenskab:** *Ændre i en eksisterende indkøbsliste ved at tilføje, slette eller redigere antallet af et produkt.*

**Scenarie:** Brugeren vil tilføje et produkt til en indkøbsliste

- o **Givet:** Brugeren er logget ind.
- o **Givet:** Produktet "Minimælk" findes i databasen
- o **Givet:** Brugeren har navigeret til en oprettet indkøbsliste med navnet "Mandagsindkøb" og trykket på "Tilføj produkter".
- o **Når:** Brugeren indtaster navnet "Minimælk" og trykker på "Tilføj til liste" og vælger listen "Mandagsindkøb".
- o **Så:** Får brugeren en besked om, at varen er tilføjet, og brugeren kan se, at "Minimælk" er blevet tilføjet til indkøbslisten "Mandagsindkøb".

**User Story 4: Sammenlign priser**

**Egenskab:** *Sammenligne priser på specifikke varer fra forskellige butikker.*

**Scenarie:** Brugeren vil sammenligne priser for to varer

- o **Givet:** Brugeren er logget ind.
- o **Givet:** Produkterne "Banan" og "Minimælk" findes i databasen.
- o **Givet:** Brugeren har navigeret til "Find produkt" siden.
- o **Når:** Brugeren søger på "Banan".
- o **Når:** Brugeren tilføjer "Banan" til sin indkøbsliste.
- o **Når:** Brugeren søger på "Minimælk".
- o **Når:** Brugeren tilføjer "Minimælk" til sin indkøbsliste.
- o **Så:** Kan brugeren totalpris for produkterne i Bilka, Rema 1000 og SPAR.

**User Story 5: Google Maps**

**Egenskab:** *Søgefunktion for nærmeste butikker, der tilbyder de billigste varer baseret på brugerens lokation.*

**Scenarie:** Brugeren søger efter nærliggende butikker.

- o **Givet:** Brugeren er logget ind.
- o **Givet:** Brugeren har givet min browser rettigheder til at se min lokation.
- o **Givet:** Brugeren har navigeret til "Abonner på produkt" siden.
- o **Når:** Brugeren trykker på knappen "Find min placering"
- o **Når:** Brugeren indtaster en ønsket radius i "Radius" input feltet.
- o **Så:** Vil systemet vise en liste over de nærmeste butikker ud fra min nuværende lokation, indenfor den indtastede radius.

**User Story 6: Notifikationer**

**Egenskab:** *Notifikationer for prisfald på overvågede produkter.*

**Scenarie:** Brugeren vil gerne blive notificeret om, hvornår "Minimælk" koster under 8,- kr.

- o **Givet:** Brugeren er logget ind.
- o **Givet:** Produktet "Minimælk" eksisterer i databasen.
- o **Givet:** Brugeren har navigeret til siden "Abonner på produkter".
- o **Når:** Brugeren indtaster navnet "Minimælk" og mindsteprisen "8" kr. i input felterne og trykker på "Hold mig opdateret" knappen.
- o **Så:** Får brugeren en notifikation på applikationen, når "Minimælk" koster 8 kr. eller mindre i Bilka, Rema 1000 eller SPAR.

**User Story 7: Oprette bruger og login**

**Egenskab:** *Oprette brugerprofil og logge ind for at tilgå webapplikationens funktionaliteter.*

**Scenarie:** Bruger opretter en profil

- o **Givet:** Brugeren ikke er logget ind.
- o **Givet:** Brugeren er på forsiden.
- o **Når:** Brugeren trykker på "Log ind" knappen i sidebar'en.
- o **Når:** Brugeren bliver redirected til login siden og trykker på "Log ind"-knappen.
- o **Når:** Brugeren trykker "Log ind med Clerk"
- o **Så:** Vil systemet redirect mig til en Clerk log-in side, hvor brugeren kan oprette en profil med e-mail og adgangskode og derefter tilgå webapplikationen.

**User Story 8: Vælge specifikke butikker**

**Egenskab:** *Filtrering af uønskede butikker fra søgeresultater.*

**Scenarie:** Bruger filtrerer uønskede butikker fra

- o **Givet:** Brugeren er logget ind.
- o **Givet:** Minimælk findes i databasen.
- o **Givet:** Brugeren søger efter varen "Minimælk" på "Find produkt" siden.
- o **Når:** Brugeren klikker på "Filtrér butikker"-indstillingen og vælger specifikke butikker, der skal udelukkes.
- o **Så:** Viser systemet ikke længere resultater fra de valgte butikker.

**User Story 9: Se statistikker som administrator**

**Egenskab:** *Administrator kan se systemstatistikker, såsom aktive brugere og oprettede indkøbslister.*

**Scenarie:** Administrator ser systemstatistikker

- o **Givet:** Brugeren er logget ind med en bruger der har rollen "Admin"
- o **Givet:** Der er blevet trykket på "Opdater database" knappen i Admin panelet, så alle produkter er oprettet med deres aktuelle priser i databasen.
- o **Når:** Administratoren navigerer til sektionen "Admin" fra sidebar-menuen.
- o **Så:** Vises der en liste over antal brugere og indkøbslister i databasen.
- o **Så:** Vises der en graf over antal brugere og indkøbslister i databasen.
- o **Så:** Vises der en "Butiks sammenlignings" graf over samtlige produkters totalpris i Bilka, Rema 1000 og SPAR, samt gennemsnitsprisen for alle produkterne på tværs af Bilka, Rema 1000 og SPAR. Hver butiks totalpris procentvist i forhold til gennemsnitsprisen, den billigste og dyreste butik og den mulige besparelse imellem den dyreste og billigste butik.
- o **Så:** Vises der en "Populære produkter" graf, hvor der ses en søjle-graf over hvilke produkter der eksisterer på flest indkøbslister.

**User Story 10: Tilføje fødevarebutikker**

**Egenskab:** Administrator kan tilføje eller fjerne fødevarebutikker fra systemet.

**Scenarie:** Administrator tilføjer eller fjerner fødevarebutikker

- o **Givet:** Brugeren er logget ind med en bruger der har rollen "Admin"
- o **Givet:** Brugeren har navigeret til siden "Admin"
- o **Når:** Brugeren går til sektionen "Butiksadministration" og klikker på "Tilføj ny butik".
- o **Så:** Opdaterer systemet butikslisten i overensstemmelse hermed, og de nye butiksoplysninger afspejles for brugerne.

### 3. Afgrænsning

Projektets krav er specificeret i Tabel 2 ved anvendelse af Moscow prioritering overlagt FURPS modellen:

		Moscow			
FURPS Requirements		M	S	C	W
F	Functionality	Must	Should	Could	Wont
F1	User Story 1: [Søge efter én specifik vare]	x			
F2	User Story 2: [Oprette en ny indkøbsliste]	x			
F3	User Story 3: [Ændre en indkøbsliste]	x			
F4	User Story 4: [Sammenlign priser]	x			
F5	User Story 5: [Google Maps]			x	
F6	User Story 6: [Notifikationer]			x	
F7	User Story 7: [Oprette bruger og login]	x			
F8	User Story 8: [Vælge specifikke butikker]		x		
F9	User Story 9: [Se statistikker som administrator]		x		
F10	User Story 10: [Tilføje fødevarebutikker]			x	
U	Usability	Must	Should	Could	Wont
U1	Brugere skal kunne finde og gennemføre en specifik opgave (fx søg efter produkt) på under 3 klik.	x			
U2	Systemet skal vise bekræftelse (fx "Vare tilføjet") inden for 1 sekund efter brugerhandling.	x			
R	Reliability	Must	Should	Could	Wont
R1	Systemet skal opdatere API'et med ny data mindst én gang i timen og gemme tidspunktet for hver opdatering.		x		
R2	Systemet har automatisk backup på databasen			x	

<b>P</b>	<b>Performance</b>	<b>Must</b>	<b>Should</b>	<b>Could</b>	<b>Wont</b>
P1	Systemet skal sikre, at produkter kan hentes og vises fra API'et på under 2 sekunder, målt ved hjælp af relevante værktøjer.	x			
P2	Systemet skal kunne præsentere mindst 5 alternative produkter baseret på det valgte produkt inden for 1 sekund.			x	
<b>S</b>	<b>Supportability</b>	<b>Must</b>	<b>Should</b>	<b>Could</b>	<b>Wont</b>
S1	Fungere på en mobilenhed			x	
S2	Systemet skal være nemt at opdatere og vedligeholde		x		
<b>+</b>	<b>Security</b>	<b>Must</b>	<b>Should</b>	<b>Could</b>	<b>Wont</b>
+1	Personlige informationer			x	
+2	2 step verifikation			x	

Tabel 2: Moscow prioritering overlagt FUPRS modellen for projektets krav



## 4. Metode og proces

I dette afsnit beskrives projektets tilgang til metode og arbejdsproces, herunder valg af udviklingsmodeller, gruppedannelsesstrategier, arbejdsfordeling og anvendte værktøjer. Fokus har været på at skabe en struktureret og effektiv projektstyring ved at kombinere diverse iterative metoder.

Afsnittet dækker, hvordan disse metoder og processer understøttede samarbejdet, opgaveprioriteringen og tidsstyringen, hvilket har været afgørende for projektets fremdrift og resultater.

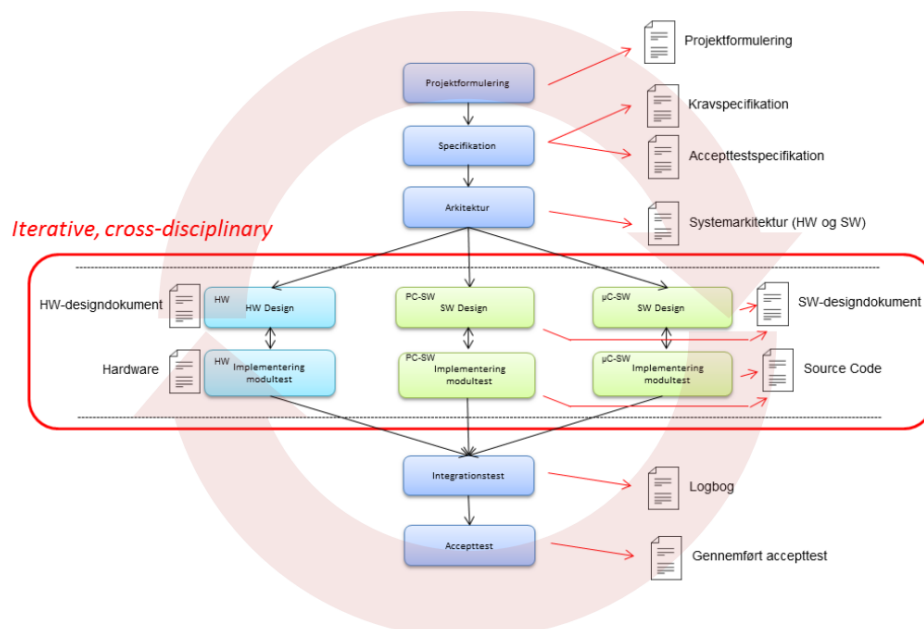
### 4.1 Metodeanvendelse

Projektet er udarbejdet ved brug af Trello, ASE-modellen og udviklingsmodellen UML. Dette har bidraget til en struktureret og effektiv arbejdsproces - dog kunne gruppen have brugt RUP-modellens struktur for en mere detaljeret tilgang.

Softwarearkitekturen er beskrevet gennem sekvensdiagrammer, en domænemodel og et klasse-, og ER-diagram. Sekvensdiagrammerne demonstrerer interaktionen mellem brugeren og systemets komponenter, under forskellige brugsscenarier. Domænemodellen skaber et overblik over systemets komponenter, mens klasse- og ER-diagrammet specificerer de forskellige klasser og deres funktioner/relationer i applikationen.

Ved at anvende ASE-modellen og de nævnte udviklingsmodeller har gruppen arbejdet iterativt, hvilket har hjulpet med at implementere ændringer løbende, og dermed forbedre gruppens endelige produkt.

Herunder ses ASE-modellen, som gruppen har fulgt i udviklingsprocessen.



Figur 2: ASE-Modellen<sup>2</sup>

<sup>2</sup> Kilde: Torben Gregersen - Ingeniørhøjskolen Aarhus Universitet. ASE.

## 4.2 Gruppedannelse

Projektarbejdet begyndte med gruppedannelsen, hvor der blev lagt vægt på at samle et team med forskellige kompetencer og erfaringer for at sikre en effektiv arbejdsproces.

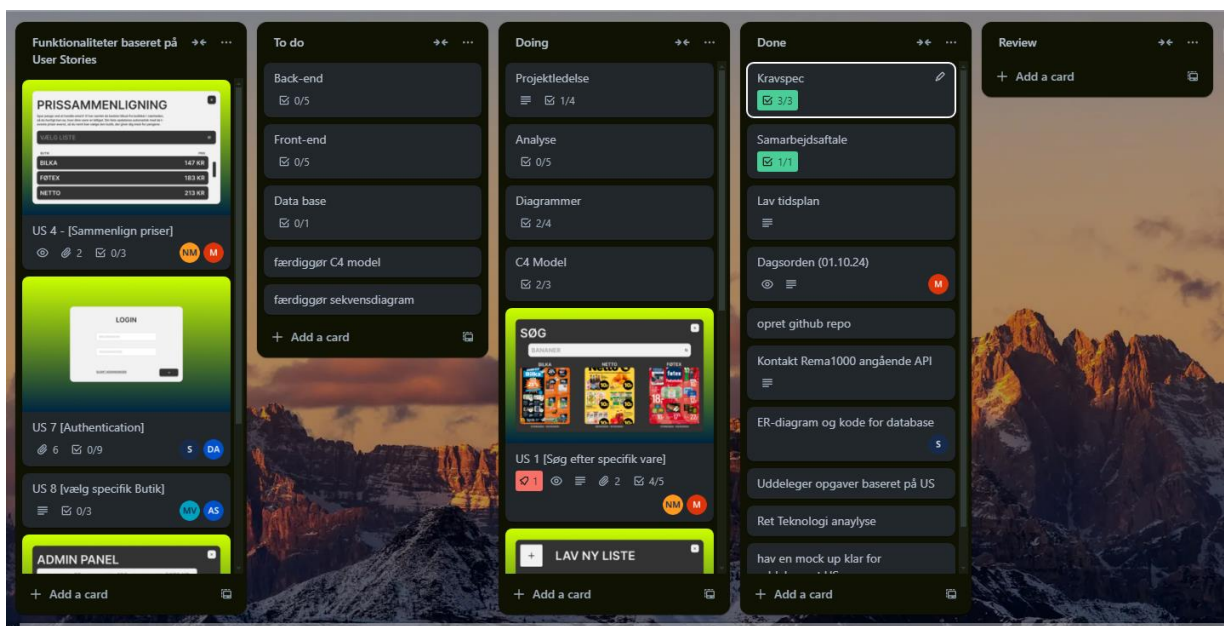
For at skabe en struktureret ramme for samarbejdet blev der udarbejdet en samarbejdsaftale, som alle medlemmer underskrev. Aftalen fastsatte klare regler for mødepligt, kommunikation og ansvarlighed, hvilket bidrog til en professionel tilgang til projektet.

## 4.3 Arbejdsmetode

Gruppen har anvendt Trello til at implementere arbejdsmetoden Kanban, hvilket har bidraget til at skabe et klart og overskueligt overblik over opgaver og deres status. På Trello har opgaverne været organiseret i forskellige kolonner, herunder "To Do," "Doing," "Done," og "Review," hvilket har gjort det nemt at følge projektets fremdrift.

Hver opgave er repræsenteret som et kort, hvor der kan tilføjes beskrivelser, deadlines, ansvarlige gruppemedlemmer samt kommentarer. Denne struktur understøttede effektiv kommunikation og opgavefordeling, således alle gruppemedlemmer altid havde adgang til opdateret information og et tydeligt indblik i både egne og andres arbejdsopgaver.

Gruppen har valgt at benytte arbejdsmetoden Kanban fremfor Scrum, da projektet krævede en agil og iterativ arbejdsmetode. Mens Scrum arbejder med faste sprintstrukturer og detaljeret planlægning, giver Kanban gruppen mulighed for kontinuerligt at prioritere og tilpasse opgaver uden at være bundet af en fastlåst tidsplan. Denne tilgang passede bedre til projektets dynamiske karakter og behovet for et konstant opdateret overblik over fremdriften. Ved at implementere Kanban i Trello, sikrede gruppen en struktureret og effektiv projektstyring, der understøttede både samarbejde og ressourcestyring.



Figur 3: Overblik over projektets Trello

## 4.4 Arbejdsfordeling

Arbejdsfordelingen i projektet blev organiseret ved at dele user stories mellem gruppens medlemmer. Gruppen blev opdelt i to-mandsgrupper, hvor hver gruppe fik ansvar for to user stories. Dette gav mulighed for at alle i gruppe kunne arbejde på både en frontend-, backend- og test del.

Kommunikationen blev håndteret primært gennem Discord, som blev brugt til diskussioner, problemløsning og koordinering. Desuden blev der holdt faste møder for at dele statusopdateringer og sikre, at alle arbejdede mod de samme mål. Vejlederen blev inddraget efter behov for at give feedback og vejledning.

## 4.5 Tidsplan og iterationer

En tidsplan blev udarbejdet tidligt i projektføreløbet og blev løbende justeret for at tage højde for ændringer og nye behov. Delmål blev fastsat for hver uge, hvilket gjorde det nemmere at holde styr på fremdriften og sikre, at deadlines blev overholdt.

På nedenstående figurer, kan den ideelle og endelige tidsplan for gruppens udviklingsproces ses. De grønne felter er de uger, hvor gruppen har haft planlagt arbejde, og de gule felter markerer semi-arbejde, hvor gruppens medlemmer har kunnet arbejde uden en reel planlagt aftale:

Ideelle Tidsplan for Gruppe 4									semi arbejde		forventet arbejde						
DEADLINES			Review 1 Deadline						Efterårsferie								Projekt deadline
Fase/uge	36	37	38	39	40	41	42	43	44	45	46	47	48	49		50	
Kravspecifikation																	
Accepttestspecifikation																	
Teknologianalyse																	
Risikoanalyse																	
Problemformulering																	
SW Arkitektur																	
SW-design																	
SW-implementering																	
Integrationstest																	
Accepttest																	
Konklusion																	
Individuelle konklusioner																	
Bilagsoversigt																	
Litteraturliste																	
Rapport skrivning																	

Figur 4: Den ideelle tidsplan for projektgruppe 4

Endelig tidsplan for Gruppe 4									semi arbejde		Reelt arbejde						
DEADLINES			Review 1 Deadline					Etterårsferie									Projekt deadline
Fase/uge	36	37	38	39	40	41	42		43	44	45	46	47	48	49		50
Kravspecifikation																	
Accepttestspecifikation																	
Teknologianalyse																	
Risikoanalyse																	
Problemformulering																	
SW Arkitektur																	
SW-design																	
SW-implementering																	
Integrationstest																	
Accepttest																	
Konklusion																	
Individuelle konklusioner																	
Bilagsoversigt																	
Litteraturliste																	
Rapport skrivning																	

Figur 5: Den endelige tidsplan for projektgruppe 4

Projektet blev ledet gennem en iterativ arbejdsmetode, hvor opgaver blev opdelt i mindre dele og løbende evalueret. Denne tilgang gjorde det muligt at reagere hurtigt på udfordringer og foretage nødvendige justeringer i processen. Trello blev, som nævnt tidligere, anvendt som et centralt værktøj til at organisere opgaver og fordele ansvar blandt gruppens medlemmer.

## 5. Analyse

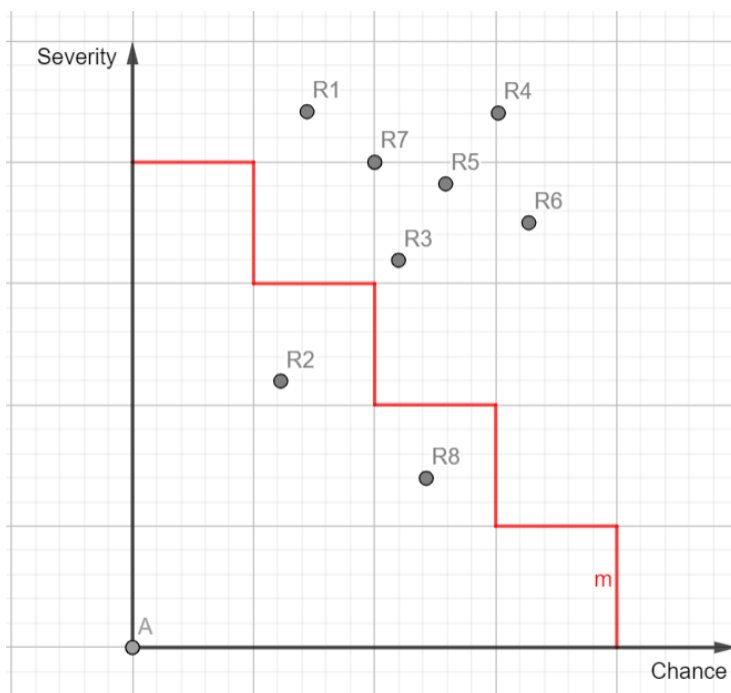
Analysen fokuserer på de udfordringer og løsninger, der opstod under projektet. Gennem risikoanalyser og vurdering af teknologivalg sikrede gruppen en struktureret tilgang til at håndtere problemer og skabe et effektivt system.

### 5.1 Risikoanalyse

Under projektet gennemførte gruppen en løbende risikoanalyse for at forudse og håndtere mulige problemer, der kunne opstå. Analysen blev opdateret iterativt gennem hele projektet for at sikre, at gruppen var forberedt på nye udfordringer.

Gruppen startede med en brainstorm, hvor alle tænkelige risici blev identificeret og derefter rangeret på en skala fra 1 til 5, baseret på deres sandsynlighed og konsekvens. For at gøre det lettere at håndtere, blev risiciene opdelt i to kategorier: "Tekniske" og "Generelle". Herefter blev der udarbejdet planer for at forebygge (mitigation) og håndtere (contingency) de mest kritiske risici.

Risikoanalysen visualiseres i grafen på Figur 6. På grafen er risici over den røde linje markeret som højprioritetsområder, der krævede både mitigation- og contingency-planer.



Figur 6: Risikoanalyse

De væsentligste tekniske risici blev løst ved løbende at teste og kvalitetssikre koden. For generelle risici, som forsinkelser og kommunikationsproblemer, blev faste møder og klare deadlines afgørende for at holde projektet på sporet. Risikoanalysen hjalp med at minimere uforudsete problemer og sikrede, at projektet kunne gennemføres inden for tidsrammen.

Risikoanalysen er uddybet i bilag [2.2.1]

## 5.2 Teknologianalyse

I dette afsnit analyseres valget af teknologier til database, backend og frontend. For hvert område vurderes fordele, ulemper og det endelige valg. Øvrige teknologier er analyseret og dokumenteret i bilaget [2.2.2].

### Backend Framework teknologi analyse

Teknologi	Fordele	Ulemper	Begrundelse	Valgt
<b>ASP.NET Core</b>	<ul style="list-style-type: none"> <li>- Moderne, hurtig og skalerbar.</li> <li>- God integration med Azure.</li> </ul>	<ul style="list-style-type: none"> <li>- Kan føre til afhængighedsproblemer mellem versioner.</li> </ul>	Passer til krav om skalerbarhed og Azure-integration.	<b>Ja</b>
<b>Node.js</b>	<ul style="list-style-type: none"> <li>- Hurtigt, event-baseret.</li> </ul>	<ul style="list-style-type: none"> <li>- Single-threaded, hvilket kan være en udfordring ved CPU-tunge opgaver.</li> </ul>	Mindre egnet til CPU-tunge opgaver.	Nej
<b>.NET</b>	<b>Cross-platform:</b> Det er muligt at kunne køre webapplikationer på Windows, Mac OS og Linux	<b>Afhængighed af Microsoft:</b> Da .net primært udviklet og vedligeholdt af Microsoft, kan der være bekymringer om vendor lock-in selvom det er open source.	Mangler moderne cloud-funktioner som ASP.NET Core.	Nej

Tabel 3: Teknologianalyse for backend framework

### Frontend teknologi analyse

Teknologi	Fordele	Ulemper	Begrundelse	Valgt
<b>React</b>	<ul style="list-style-type: none"> <li>- Komponentbaseret arkitektur gør det nemt at genbruge kode.</li> <li>- Høj performance og real-time UI-opdateringer.</li> </ul>	<ul style="list-style-type: none"> <li>- Stejl læringskurve for nybegyndere.</li> </ul>	Fleksibel og ideel til interaktive brugergrænseflader.	<b>Ja</b>
<b>Angular</b>	<ul style="list-style-type: none"> <li>- Komplet framework med indbygget routing og state management.</li> <li>- TypeScript-integration for typesikkerhed.</li> </ul>	<ul style="list-style-type: none"> <li>- Kan være tungt og komplekst, især til mindre projekter.</li> </ul>	For omfattende til projektets behov.	Nej
<b>Vue.js</b>	<ul style="list-style-type: none"> <li>- Nem at lære og komme i gang med.</li> <li>- Real-time databinding</li> </ul>	<ul style="list-style-type: none"> <li>- Mindre community og færre ressourcer</li> </ul>	Begrænset ressourcer og community-støtte.	Nej

	gør UI-opdateringer intuitive.	sammenlignet med React og Angular.		
--	--------------------------------	------------------------------------	--	--

Tabel 4: Teknologianalyse for frontend

## Database teknologi analyse

Teknologi	Fordele	Ulemper	Begrundelse	Valgt
<b>SQL Server</b>	<ul style="list-style-type: none"> <li>- Fuld ACID-understøttelse.</li> <li>- Ideel til strukturerede data og komplekse forespørgsler.</li> </ul>	<ul style="list-style-type: none"> <li>- Mindre fleksibel ved ændringer i skema.</li> </ul>	Robust og ideel til strukturerede data og transaktioner.	<b>Ja</b>
<b>MongoDB</b>	<ul style="list-style-type: none"> <li>- Skemaløs fleksibilitet, ideel til ustrukturerede data.</li> <li>- God skalerbarhed.</li> </ul>	<ul style="list-style-type: none"> <li>- Mangler fuld ACID-understøttelse, hvilket kan give problemer ved transaktioner.</li> </ul>	Ikke egnet til strukturerede data og transaktioner.	Nej
<b>MySQL</b>	<ul style="list-style-type: none"> <li>- Velkendt og pålidelig.</li> <li>- Open-source og bredt understøttet.</li> </ul>	<ul style="list-style-type: none"> <li>- Kan mangle den skalerbarhed og fleksibilitet, som NoSQL-databaser tilbyder.</li> </ul>	SQL Server passer bedre til projektets kompleksitet.	Nej

Tabel 5: Teknologianalyse for database

## Opsummering

Område	Valgt Teknologi
<b>Backend</b>	ASP.NET Core
<b>Backend Sprog</b>	C#
<b>Frontend</b>	React
<b>Frontend Sprog</b>	JavaScript
<b>API</b>	RESTful API
<b>Database</b>	SQL Server
<b>Testing</b>	NUnit
<b>Datahåndtering</b>	Azure Data Studio

Tabel 6: Opsummering

I tabellen ovenfor præsenteres de valgte teknologier for de forskellige områder af projektet. Dokumentationen for de enkelte valg og begrundelser herfor kan findes i bilaget [2.2.2], hvor der redegøres detaljeret for hver teknologis styrker og svagheder, samt argumentationen for hvorfor de blev valgt.

## 6. Arkitektur

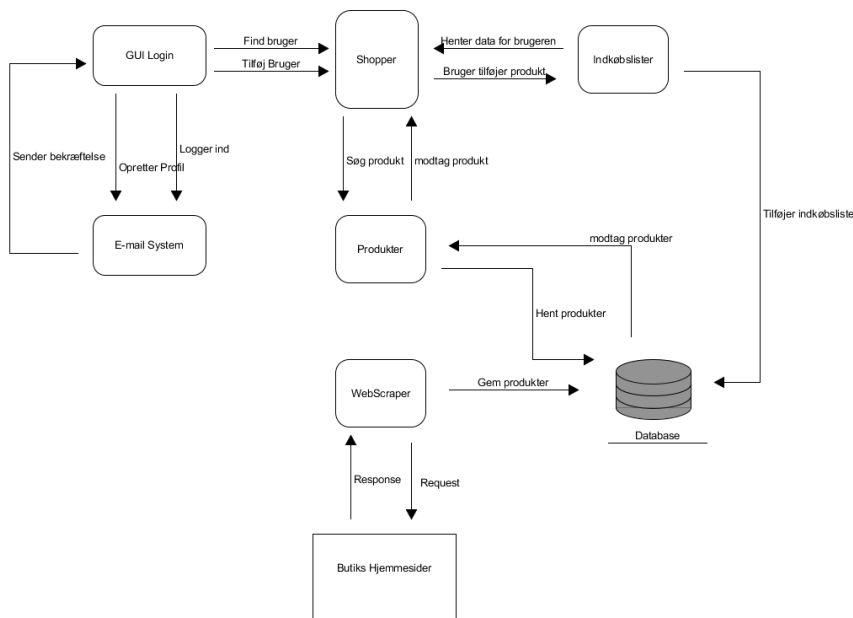
Arkitekturen beskriver, hvordan systemets forskellige komponenter er designet og organiseret for at sikre funktionalitet, skalerbarhed og brugervenlighed. Afsnittet dækker systemarkitektur, herunder domænemodellen, sekvensdiagrammer, klassediagram, ER-diagram og anvendelsen af C4-modellen, som tilsammen giver et detaljeret overblik over systemets struktur og interaktioner.

### 6.1 Systemarkitektur

I dette afsnit præsenteres systemarkitekturen, som danner grundlaget for projektets udvikling. Der fokuseres på centrale elementer som domænemodellen, sekvensdiagrammer, klasse- og ER-diagrammet samt anvendelsen af C4-modellen. Disse giver tilsammen et struktureret overblik over systemets design og opbygning.

#### 6.1.1 Domænemodel

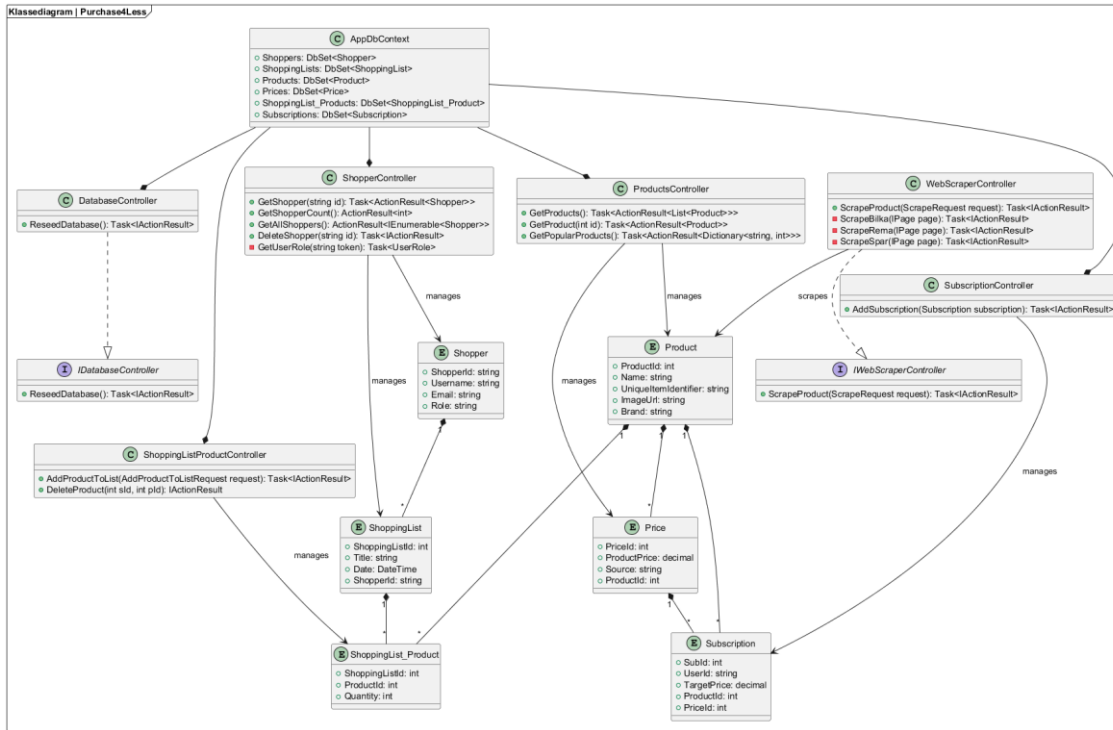
Nedstående domænemodel vist på Figur 7 beskriver de vigtigste begreber og relationer indenfor applikationsdomænet. Modellen hjælper med at forstå, hvordan data struktureres og interagerer i systemet.



Figur 7: Domænemodel for systemet

## 6.1.2 Klassediagramm

Klassediagrammet illustrerer systemets overordnede arkitektur og består af datamodeller, controllere og databasekontekst.



Figur 8: Klassediagram for systemet

## Datamodeller/Entities

- **Product:** Basismodel for produkter med information om navn, identifikator, billede og butik.
- **Price:** Håndterer prisinformation for produkter fra forskellige butikker.
- **Shopper:** Brugermodel med authentication og autorisationsdata.
- **ShoppingList:** Indeholder brugerens indkøbslister med titel og dato.
- **ShoppingList\_Product:** Sammenkoblingstabel mellem produkter og indkøbslister med mængdeangivelse.
- **Subscription:** Håndterer brugerens prisovervågninger med målpriser for specifikke produkter.

## Controllere

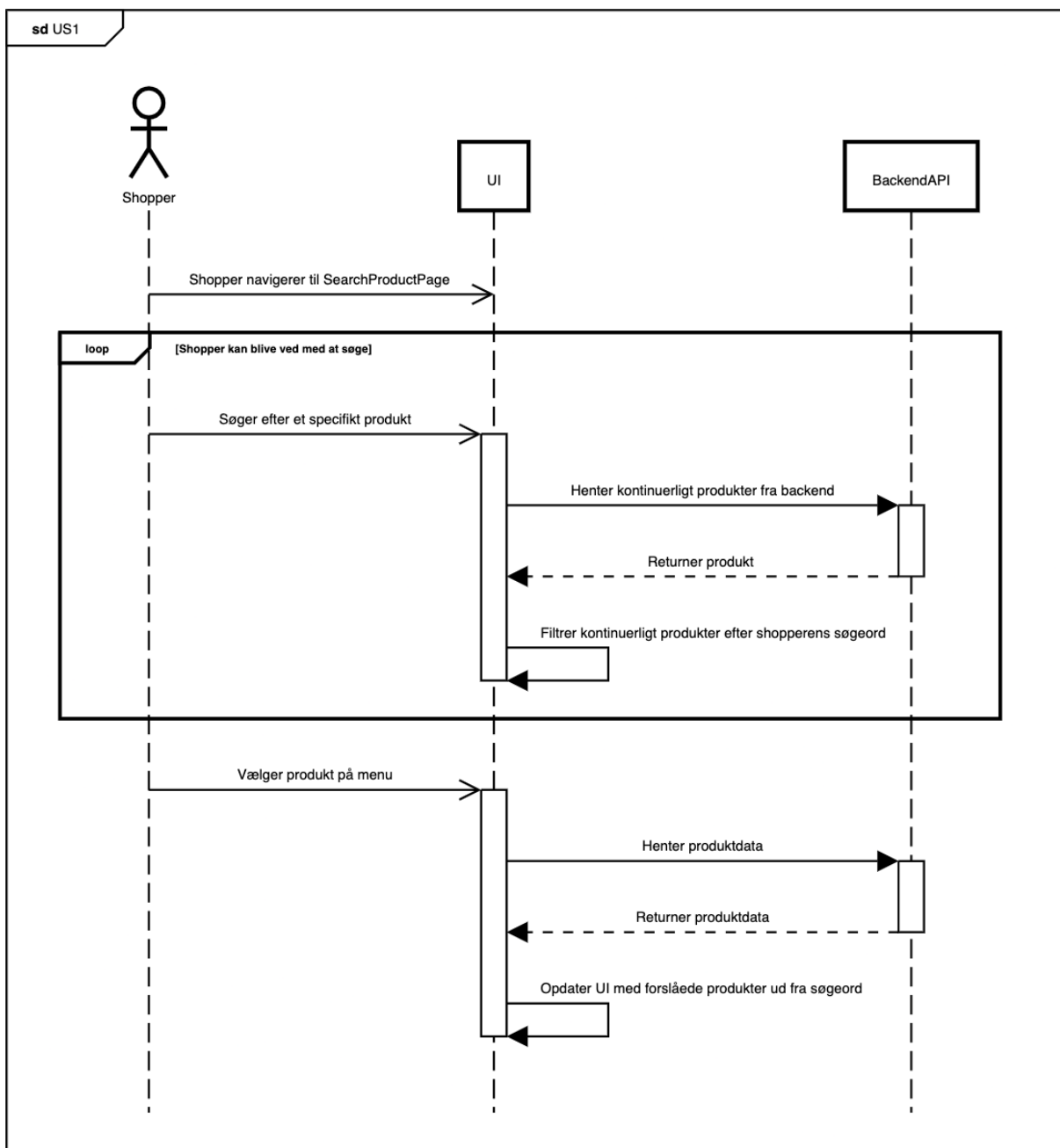
- **DatabaseController:** Administrerer databaseoperationer og seeding af testdata.
- **ProductsController:** Håndterer forespørgsler relateret til produkter og deres priser.
- **ShopperController:** Styrer brugerrelaterede operationer og autorisation.
- **ShoppingListProductController:** Administrerer tilføjelse og fjernelse af produkter på indkøbslister.
- **WebScraperController:** Implementerer webscraping fra forskellige supermarkeder (*Bilka, Rema 1000, Spar*)
- **SubscriptionController:** Håndterer oprettelse af prisovervågninger.



### 6.1.3 Systemsekvensdiagrammer

Systemsekvensdiagrammer viser interaktionen mellem brugere og systemet, samt systemets reaktioner på brugerhandlinger. I denne del er det bestemt, at gruppen går i dybden med specifikt user-story 1, 2, 3, og 4, da disse user-stories rummer de mest centrale brugerinteraktioner, og dermed også repræsenterer det mest komplekse dataflow mellem brugere og systemets komponenter. Dette gør disse user-stories særligt velegnede til at demonstrere og analysere systemets vigtigste funktioner og interaktioner.

#### User Story 1: Søge efter en specifik vare



Figur 9: US1 systemsekvensdiagram

Sekvensdiagrammet på Figur 9 viser interaktionen mellem shopperen og systemets komponenter, når shopperen søger efter et produkt. Processen kan inddeles i følgende tre trin:

**1. Navigation til siden "SearchProductPage".**

Shopperen navigerer til siden "SearchProductPage" via UI.

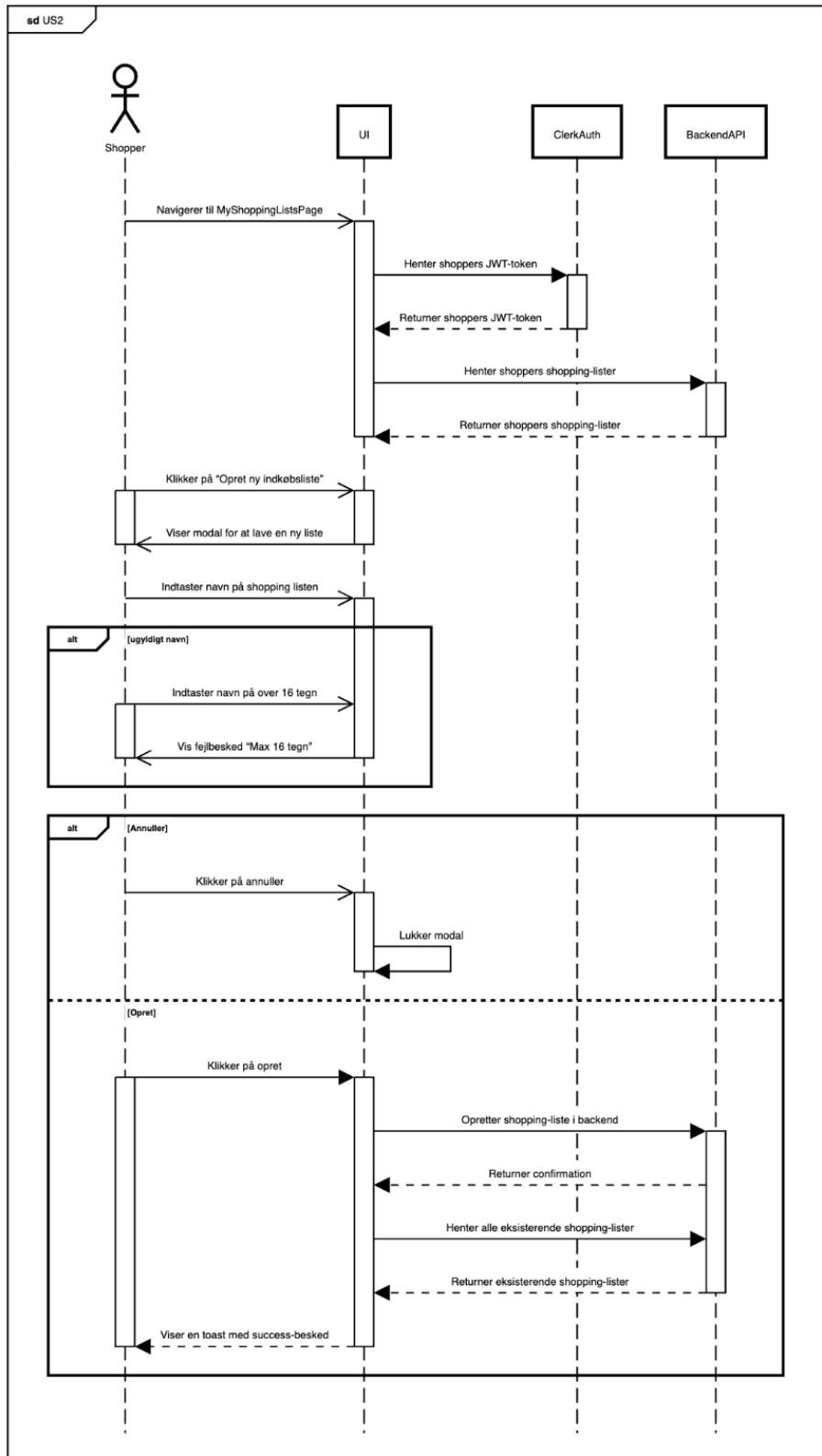
**2. Søgning efter et specifikt produkt.**

Shopperen indtaster et søgeord i søgefeltet. UI sender denne forespørgsel til BackendAPI, der kontinuerligt returnerer produkter, der matcher søgeordet. UI filtrerer og opdaterer løbende resultaterne baseret på shopperens input.

**3. Valg af produkt fra søgeresultaterne.**

Shopperen vælger et produkt fra de foreslåede resultater. UI henter yderligere produktdata fra BackendAPI'et for at give en mere detaljeret visning. UI bliver opdateret for at vise de valgte produktoplysninger.

## User Story 2: Oprette en ny indkøbsliste



Figur 10: US2 systemsekvensdiagram

Sekvensdiagrammet på Figur 10 viser interaktionen mellem shopperen og systemet ved oprettelse af en ny indkøbsliste. Processen kan inddeles i følgende fem trin:

**1. Navigation til siden "Mine shoppinglister".**

Shopperen navigerer til "Mine shoppinglister"-siden via UI. Her hentes shopperens JWT-token for autentificering, så systemet kan validere, hvem der anmoder om data.

**2. Hentning af eksisterende shopping lister for brugeren.**

Når shopperens JWT-token er bekræftet, kontakter systemet BackendAPI'et for at hente alle eksisterende shoppinglister, der er tilknyttet denne bruger. Disse data vises på siden, så shopperen kan se sine eksisterende lister.

**3. Visning af modal til oprettelse af en ny liste.**

Når shopperen klikker på "Opret ny indkøbsliste", åbnes en modal, hvor shopperen kan indtaste et navn til sin nye liste.

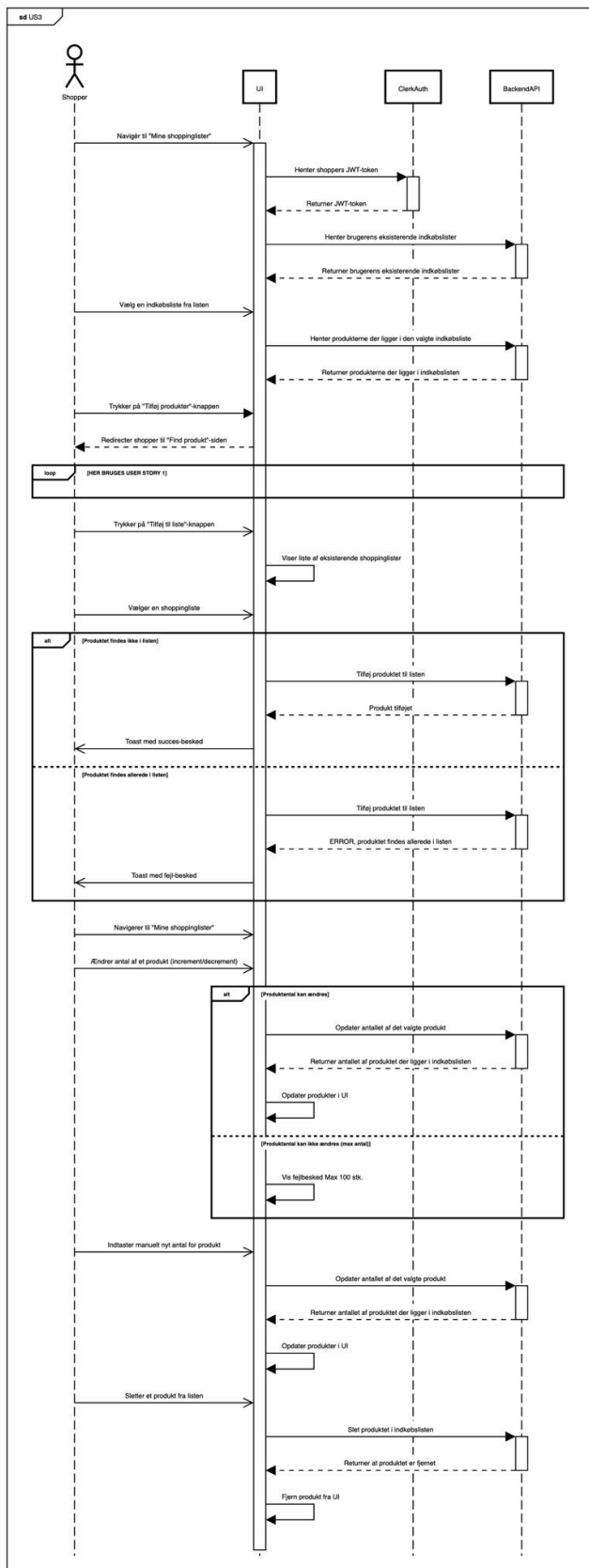
**4. Validering af det indtastede listenavn.**

Shopperen indtaster navnet på den nye liste. Systemet validerer input, og hvis navnet overstiger 16 tegn, vises en fejlmeddelelse: "Max 16 tegn".

**5. Oprettelse af listen**

Når shopperen klikker på "Opret", sendes dataene til BackendAPI'et, som opretter den nye shoppingliste. Efter vellykket oprettelse sendes en bekræftelse tilbage, og systemet opdaterer listen over eksisterende shoppinglister. Shopperen får samtidig vist en "toast"-besked for at bekræfte, at listen blev oprettet med succes.

## User Story 3: Ændre en indkøbsliste



Figur 11: US3 sekvensdiagram

Sekvensdiagrammet på Figur 11 viser interaktionen mellem brugeren og systemets komponenter ved redigering af en indkøbsliste. Denne interaktion kan inddeles i 6 overordnede processer:

**1. Navigation til "Mine shoppinglister".**

Brugeren navigerer til "Mine shoppinglister"-siden via UI'et. Systemet henter og validerer brugerens JWT-token for at sikre korrekt adgang. Derefter indlæses brugerens eksisterende indkøbslister fra BackendAPI'et og præsenteres i UI'et.

**2. Valg af en specifik shopping liste.**

Brugeren vælger en ønsket indkøbsliste fra den viste oversigt. Systemet henter alle produkter, der er tilknyttet den valgte indkøbsliste fra BackendAPI'et og viser disse i UI'et, så brugeren kan redigere listen.

**3. Navigation til "Find produkt".**

Brugeren trykker på "Tilføj produkt"-knappen og navigeres til "Find produkt"-siden. Her vil US1 (Søg efter produkt) starte, hvor brugeren vil vælge et produkt.

**4. Tilføj produkt til en shoppingliste.**

Brugeren trykker på "Tilføj til liste"-knappen. UI viser en modal med en liste over brugerens shoppinglister. Brugeren vælger den ønskede shoppingliste.

**4.1. Succes ved tilføjelse**

Hvis produktet ikke allerede findes i listen, vil produktet tilføjes i databasen, og et modal med en succesbesked vil vises.

**4.2. Fejl ved tilføjelse**

Hvis produktet allerede findes i listen, vil produktet ikke tilføjes, og et modal med en fejlbesked vil vises.

**5. Ændring af produktantal i shoppinglisten.**

Brugeren navigerer til "Mine shoppinglister"-siden. Her kan brugeren ændre antallet af et specifikt produkt ved enten at trykke på "+" eller "-" knapperne eller ændre på antallet manuelt. Metoderne fungerer således:

**5.1. Ved at bruge "+" eller "-" knapperne:** Brugeren kan increment eller decrement antallet af produktet med 1. Systemet opdaterer produktets antal i backend og UI'et. Hvis det nye antal er gyldigt, sker ændringen gennemføres succesfuldt, og UI'et opdateres.

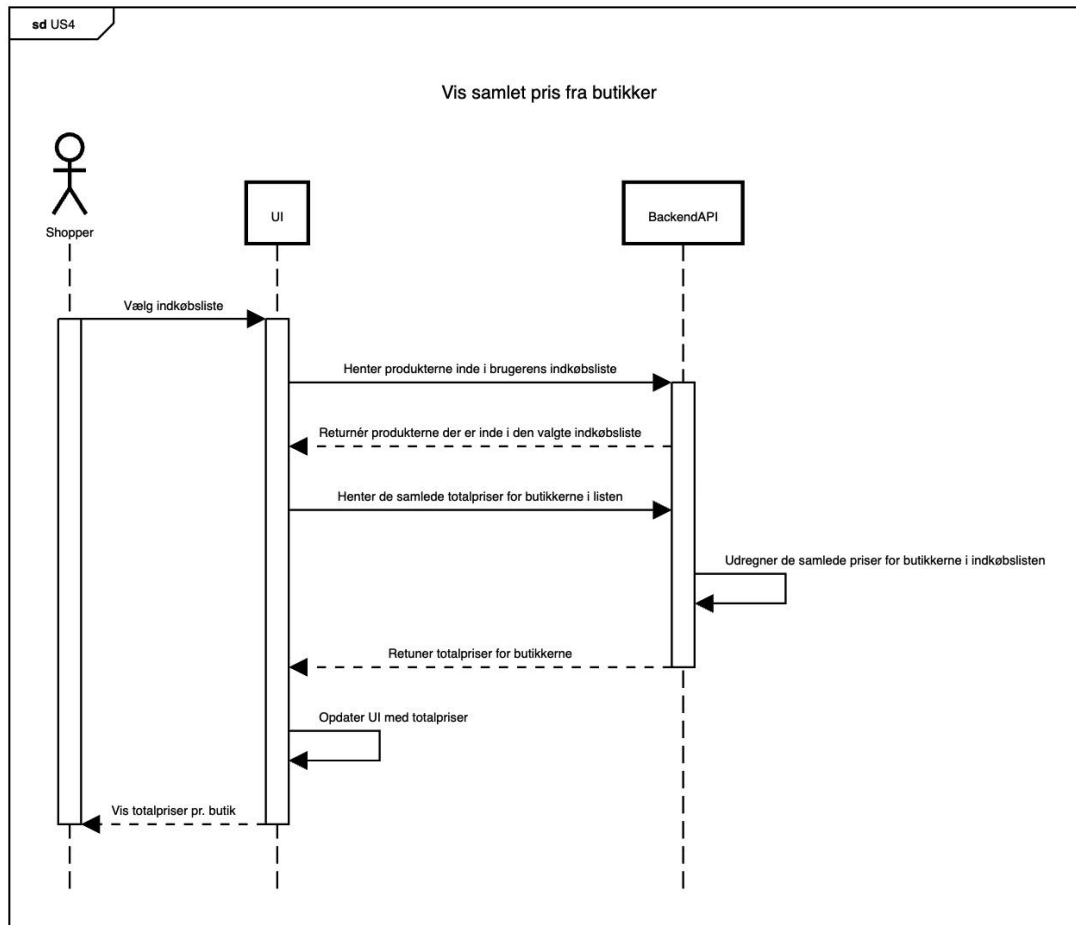
**5.2. Hvis det nye antal overstiger maksimumgrænsen (100 stk.):** Ændringen afvises, og UI'et viser en fejlmeddelelse: "Max 100 stk."

**5.3. Ved manuelt at indtaste et nyt antal:** Brugeren kan skrive det ønskede antal direkte i inputfeltet. Systemet validerer inputtet og opdaterer produktets antal i backend og UI'et, hvis det er gyldigt.

**6. Sletning af produkt fra shoppinglisten.**

Brugeren kan vælge at fjerne et produkt fra listen ved at trykke på fjern knappen eller ved at trykke på "-" knappen indtil antallet af produktet er 0. Systemet opdaterer backend for at slette produktet og opdaterer UI'et for at fjerne produktet fra oversigten.

## User Story 4: Sammenlign priser



Figur 12: US4 sekvensdiagram

Sekvensdiagrammet på Figur 12 illustrer, hvordan systemet beregner og viser de samlede priser fra forskellige butikker, baseret på en valgt indkøbsliste med forskellige produkter. Processen kan inddeles i følgende fire trin:

#### 1. Valg af indkøbsliste

Shopperen vælger en ønsket indkøbsliste fra UI'et. Systemet sender en anmodning til BackendAPI'et for at hente produkterne tilknyttet den valgte liste.

#### 2. Hentning af produkter

BackendAPI'et returnerer alle produkter fra den valgte indkøbsliste sammen med deres priser fra de relevante butikker.

#### 3. Udregning af samlede priser

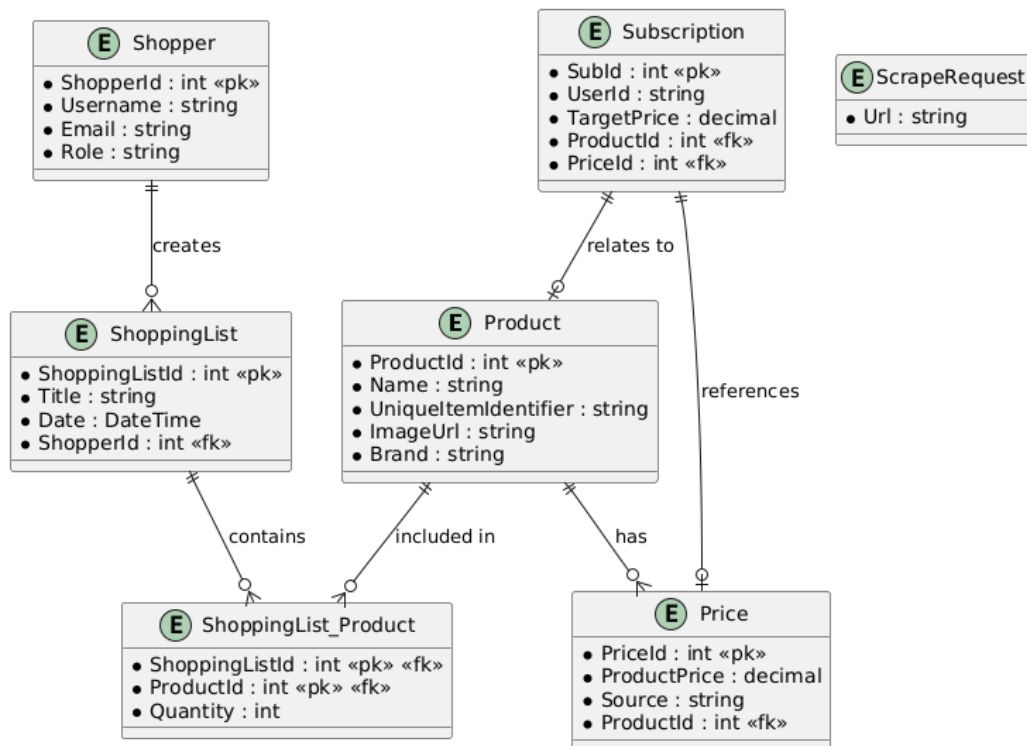
UI sender en anmodning til BackendAPI'et om totalpriserne for butikkerne i listen. BackendAPI'et beregner de samlede priser for produkterne pr. butik og returnerer dem.

#### 4. Visning af samlede priser i UI

UI'et opdateres for at vise totalpriserne pr. butik.

### 6.1.4. ER-diagram

Projektets ER-diagram illustrerer strukturen og relationerne mellem de centrale entiteter i systemet Purchase4Less. Nedenfor ses projektets ER-diagram, efterfulgt af en forklaring af diagrammets elementer og deres sammenhænge.



Figur 13: ER-diagram for Purchase4Less

Diagrammet fremhæver følgende centrale elementer:

**Shopper** fungerer som udgangspunktet for systemet, hvor brugerne kan oprette og administrere indkøbslister.

**ShoppingList** er knyttet til en shopper og kan indeholde flere produkter via relationen til ShoppingList\_Product.

**Product** repræsenterer de varer, der kan tilføjes til indkøbslister, og er forbundet til Price, hvilket gør det muligt at knytte flere priser til et produkt.

**Price** indeholder prisoplysninger og angiver kilden, såsom en butik, hvor produktet er fundet.

**ShoppingList\_Product** skaber en dynamisk forbindelse mellem produkter og shoppinglister og angiver mængden af hvert produkt i listen.

**Subscription** hjælper brugere med at holde styr på prisændringer og målpriser.

**ScrapeRequest** indsamler data fra eksterne kilder for at holde priser og produkter opdaterede.

For yderligere dokumentation refereres der til bilag [1.1.2].

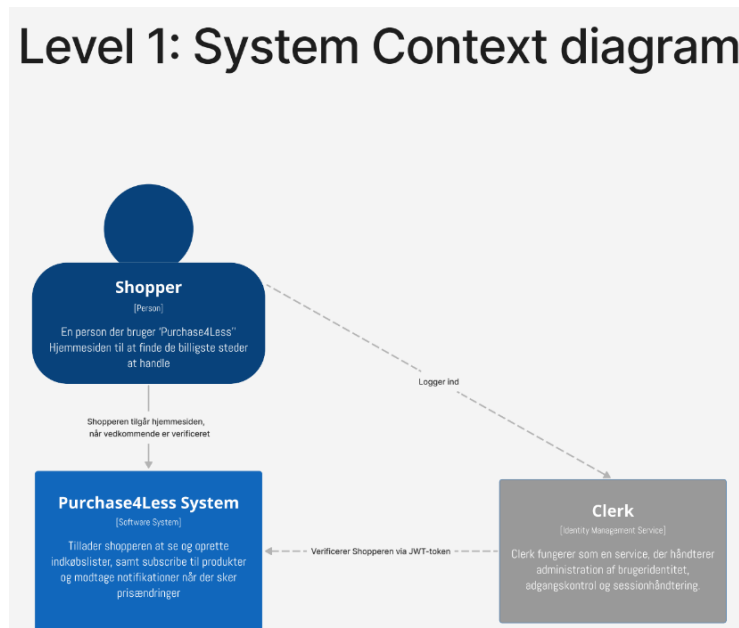


### 6.1.5. C4-model

Dette afsnit beskriver C4-modellen. De forskellige diagrammer er til for at zoome ind på de enkelte dele af systemet og give et mere detaljeret indblik i, hvordan systemet er opbygget og opsat.

#### System kontekst diagram

System konteksts diagrammet<sup>3</sup> på nedenstående Figur 14 illustrerer, hvordan en shopper (bruger) interagerer med applikationen for at udføre forskellige funktioner. Efter at have logget ind, kan shopperen benytte systemets søgefunktion og oprette indkøbslister.



Figur 14: C4-Model Level 1: System konteksts diagram for systemet

I System konteksts diagrammet fremgår det, at shopperen interagerer med webapplikationen, som benytter Clerk til at verificere den enkelte shopper. Clerk fungerer som et centralt system til brugeridentifikation, autentifikation og adgangskontrol, der sikrer, at brugerne kun har adgang til deres egne indkøbslister og autoriserede data og funktioner.

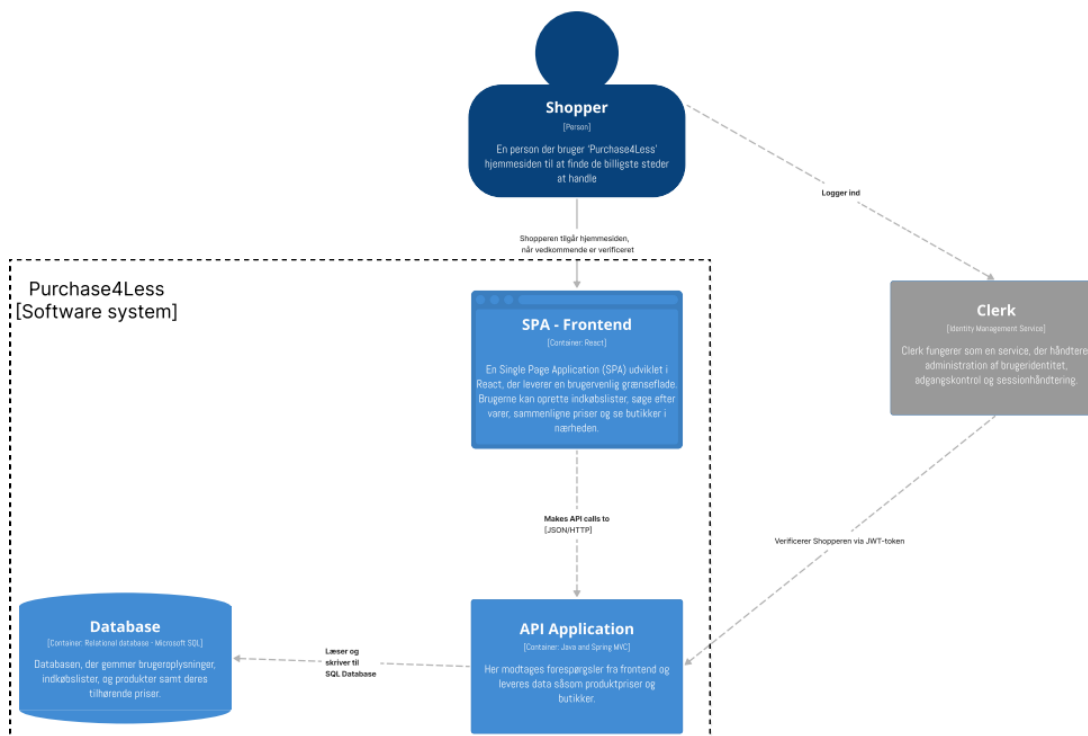
#### Container diagram

Containerdiagrammet<sup>4</sup> på Figur 15 viser de forskellige "containers" som er en del af systemet, såsom frontenden, API applikationen og databasen. Diagrammet giver et overblik over de vigtigste containers' ansvar og illustrerer, hvordan de kommunikerer med hinanden.

<sup>3</sup> <https://c4model.com/diagrams/system-context>

<sup>4</sup> <https://c4model.com/diagrams/container>

## Level 2: Container diagram



Figur 1515: C4 Model Level 2: Container diagram for systemet

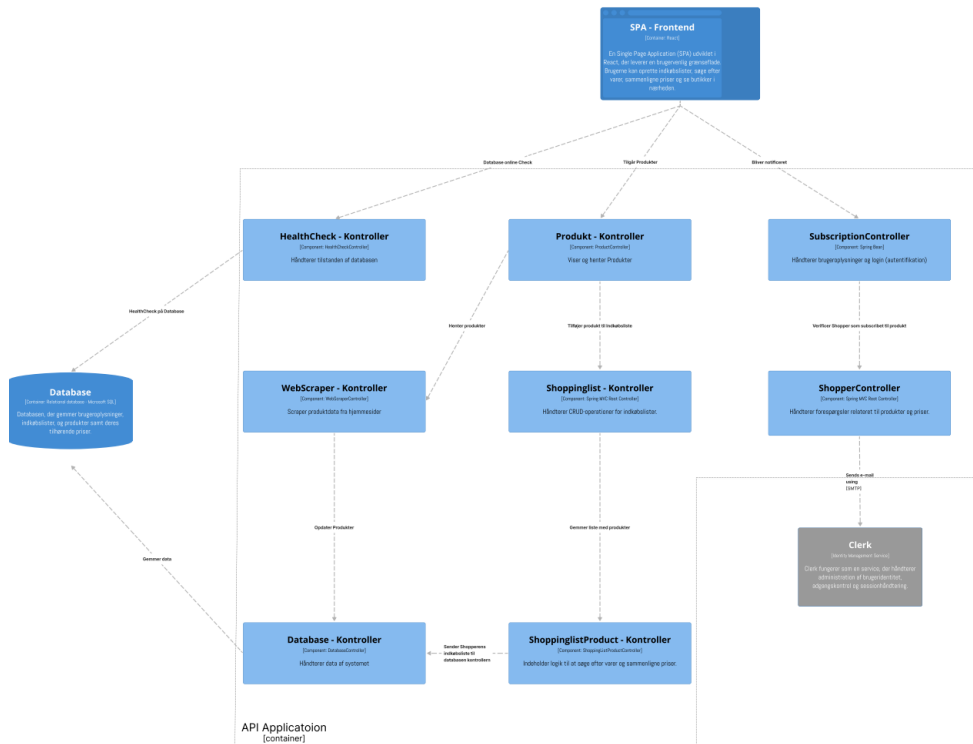
Systemet består af en SPA-frontend, der kommunikerer med en API-applikation, som håndterer dataudveksling med databasen.

Frontend 'en kommunikerer med API-applikationen via RESTful API-anmodninger, hvor den sender anmodninger om data og modtager svar i form af JSON-objekter. API'en fungerer som et mellemlid mellem frontend 'en og databasen og håndterer dataudvekslingen. Den udfører de nødvendige CRUD-operationer på databasen, som opbevarer information om produkter, priser og indkøbslister.

### Component diagram API

Component diagrammet<sup>5</sup> på Figur 16 viser, hvordan frontenden interagerer med API-kontrollerne. Det viser, hvordan systemet strukturerer databehandling og autentifikation gennem Clerk og databasen.

Level 3: Component diagram



Figur 1616: C4 Model Level 3: Component diagram for systemets API

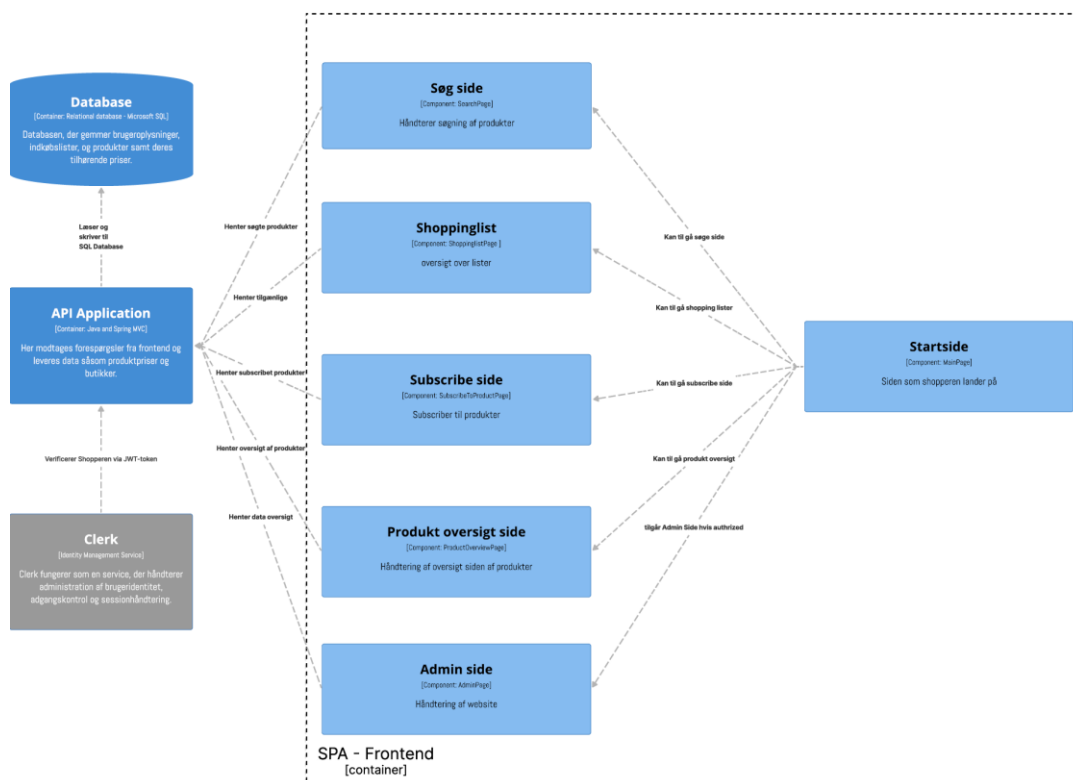
Diagrammet viser, hvordan systemets controllers arbejder sammen for at sikre en effektiv datahåndtering og funktionalitet. Controllers som ShoppingListController og ProductController kommunikerer med databasen via API-kald for at hente og opdatere data, mens AuthController håndterer brugerautentificering gennem Clerk. Disse komponenter interagerer for at sikre, at data fra brugerne bliver valideret, gemt og præsenteret korrekt i systemet. Yderligere detaljer kan findes i bilag [1.1.4.2].

<sup>5</sup> <https://c4model.com/diagrams/component>

### Component diagram frontend

Nedenstående diagram viser systemets frontend-komponenter under SPA – Frontend. Startside fungerer som landingsside, mens de øvrige komponenter giver brugeren mulighed for at søge efter produkter, administrere shoppinglister, abonnere på produkter, få en produktoversigt og tilgå administrative funktioner.

### Level 3: Component diagram



Figur 1717: C4 Model Level 3: Component diagram for systemets frontend

Frontend består af sektioner som Startside, Søg side, Shoppinglist, Subscribe, Produkt oversigt og Admin side. Disse arbejder sammen via backend for at sikre funktionalitet som produktsøgning, listeadministration, abonnementer og datastyring.

## 6.2. Backend MVC-arkitektur

Projektets backend er opbygget med MVC-arkitekturen (Model-View-Controller), som giver en tydelig struktur og gør systemet lettere at vedligeholde og udvide.

- **Model:** Modellerne i projektet repræsenterer data og reglerne for, hvordan dataene fungerer. For eksempel håndterer modeller som ShoppingList, Product og Price data som indkøbslister, produkter og priser. De sikrer også, at dataene hænger sammen, og hvordan de gemmes i databasen via Entity Framework Core.
- **Controller:** Controllerne fungerer som bindeled mellem frontend og backend. De modtager forespørgsler fra frontend (som at tilføje et produkt til en indkøbsliste), udfører det nødvendige arbejde og sender svaret tilbage. Et eksempel er ShoppingListProductController, som står for funktioner som at tilføje produkter, opdatere mængder eller beregne priser.
- **View:** I dette projekt fungerer frontend som visningen (View), hvor data, der sendes fra backend via JSON, bliver vist til brugeren gennem React-komponenter.

Databasekonteksten (*AppDbContext*) fungerer som bro mellem modeller og databasen. Ved at bruge MVC-arkitekturen har projektet en stærk struktur, hvor de forskellige dele af systemet arbejder godt sammen. Det gør det lettere at tilføje nye funktioner, rette fejl og sikre, at systemet kan tilpasses fremtidige behov.

## 7. Design & implementering

Design og implementering af projektet omfatter både backend, database og frontend, som tilsammen sikrer en fuldt funktionel og sammenhængende applikation. Dette afsnit beskriver, hvordan systemets forskellige dele er blevet realiseret, herunder arkitektur, API-endpoints og specifikke funktioner som webscraping.

### 7.1. UI/UX

Designet af Purchase4Less fokuserer på brugervenlighed, visuel æstetik og en forbedret brugeroplevelse, med nøgleelementer og principper præsenteret i dette afsnit.

#### 7.1.1. Designproces og Skitser

For at sikre at designet adresserede alle brugerbehov, blev der udarbejdet skitser baseret på User Stories. Disse skitser, som ses på Figur 18, fungerede som et fælles udgangspunkt for det grafiske arbejde og hjalp med at definere det visuelle udtryk for brugerrejser og sikre, at alle krav fik et dertilhørende visuelt user-interface.



Figur 1818: Initiale skitser baseret på User Stories

Her ses hvordan de grundlæggende idéer blev visualiseret for nøglefunktioner såsom login, produktoversigt mm.

Der blev gjort brug af Figma for at skabe et mere detaljeret og visuelt overblik over applikationen, samtidig med at det blev nemt at justere og afprøve forskellige designidéer, uden at skulle implementere dem i kode.



Figur 1919: Iterativ design af Hero-sektion

På Figur 19 ses udviklingen af Hero-sektionen, fra den oprindelige skitse til det endelige produkt.

Hero-sektionen fungerer som applikationens 'onboarding' og introduktion for brugeren. Derfor blev den designet med et fokus på at fange brugerens opmærksomhed. Dette opnås ved hjælp af et dynamisk slideshow, der fremviser billeder af almindelige danske dagligvarer. Derudover giver call-to-action-knapperne 'Start søgning' og 'Lær mere' brugerne en nem og intuitiv indgang til deres rejse gennem applikationen.

For yderligere design dokumentation henvises til bilag [7.3].

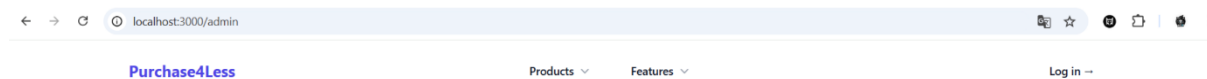
## 7.2. Design overvejelser

Designet af webapplikationen har naturligvis gennemgået flere ændringer i takt med, at projektet har nærmet sig deadline, og forståelsen af det overordnede system er blevet større.

En af de mest markante ændringer var beslutningen om at skifte fra en standard header-menu, som vist i Figur 20, til en sidebar-menu, som ses i Figur 21. Denne ændring var motiveret af ønsket om at give brugeren en følelse af at arbejde med en interaktiv applikation, hvilket en sidebar oftest giver en oplevelse af.

En undersøgelse fra 2017, *Horizontal Attention Leans Left*<sup>6</sup>, viser, at brugere i gennemsnit bruger 80% af deres tid på at se på den venstre halvdel af en webside, mens kun 20% af tiden bruges på den højre halvdel. Denne indsigt understøttede beslutningen om at placere vigtige navigations- og interaktive elementer i venstre side af skærmen, hvor brugernes opmærksomhed er mest koncentreret. Ved at implementere en sidebar-menu placeres de vigtigste navigationsmuligheder på venstre side af skærmen, hvilket gør dem mere synlige og lettere at tilgå for brugeren.

<sup>6</sup> Nielsen Norman Group. (n.d.). *Horizontal Attention Leans Left*. Accessed December 12, 2024, from <https://www.nngroup.com/articles/horizontal-attention-leans-left/#:~:text=Summary%3A%20Web%20users%20spend%2080,users'%20efficiency%20and%20company%20profits.>



Figur 2020: Original menu i header



Figur 2121: Opdateret menu i sidebar

## 7.3. Database

Dette afsnit beskriver implementeringen af projektets database, som understøtter funktioner relateret til shoppere, produkter og indkøbslister. Databasen er udviklet med Entity Framework Core og ASP.NET Core for effektiv datahåndtering og integration med API-endpoints, der muliggør CRUD-operationer mellem backend og frontend. Databasen er oprettet på en SQL Server for robust lagring og skalerbarhed, mens Azure Data Studio og Swagger er brugt til dataadministration og endpoint-test.

## 7.4. User Stories

For at illustrere, hvordan projektets database, backend og frontend arbejder sammen, præsenteres her udvalgte user stories. Disse user stories viser den praktiske implementering af systemets kernefunktionalteter, herunder backend-API-endpoints og deres integration med frontend. Fokus vil være på, hvordan disse user stories er blevet omsat til konkrete funktioner, der understøtter shoppere, indkøbslister og produktdata. For at se resultater og eksempler på funktionaliteten af de specificerede user-stories, henvises der til resultat-delene under hver user story i bilag [7.1.1].

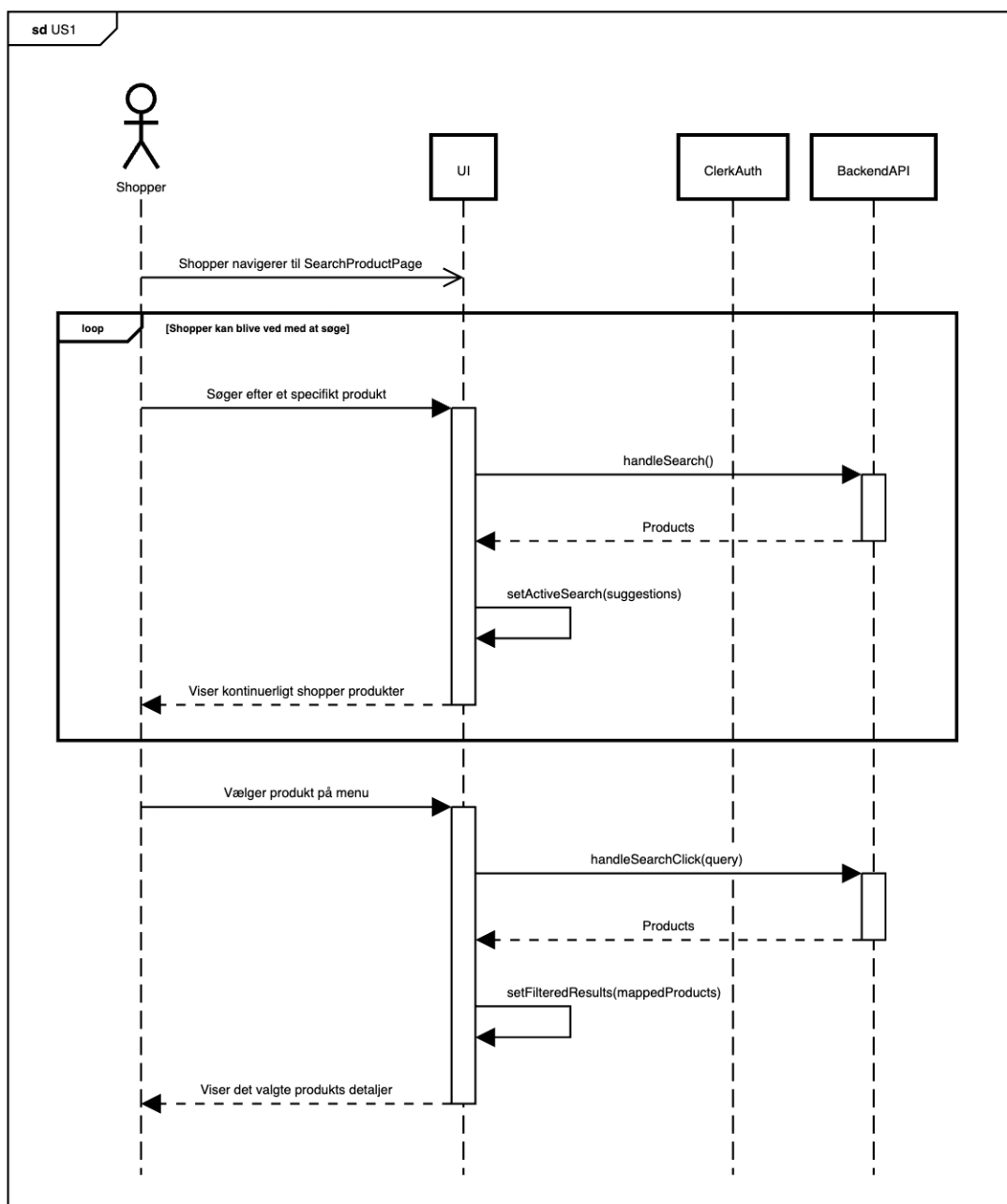


## User Story 1: Søge efter en specifik vare

### Beskrivelse

Denne user story omhandler implementeringen af funktionaliteten, der giver en shopper mulighed for at søge efter specifikke varer. Når en shopper navigerer til søge-siden, hentes systemets produkter fra backend. Systemet tillader derefter gentagne søgninger efter specifikke produkter, hvor forslag returneres og vises dynamisk. Implementeringen sikrer, at shopperen kan finde og vælge de billigste muligheder hurtigt og problemfrit, understøttet af backend-validering og en brugervenlig grænseflade.

Implementeringen af denne user-story er skitseret med følgende sekvensdiagram:



Figur 2222: US1uddybet sekvensdiagram

Sekvensdiagrammet på Figur 22 viser interaktionen mellem shopper og UI og systemets komponenter, når shopperen skal søge efter en specifik vare. Denne interaktion kan inddeles i tre hovedtrin, som kan ses forklaret i bilag [7.1.1].

### Backend

ProductsController.cs håndterer alle HTTP-forespørgsler relateret til produkter. Controlleren fungerer som en central del af API'en, hvor den behandler HTTP-anmodningen GET. Den udfører operationer på data-laget ved at interagere med databasen gennem ApplicationDbContext og returnerer relevante dat. Et eksempel på denne funktionalitet er metoden GetProduct, der henter et specifikt produkt fra databasen baseret på dets unikke ID. Nedenstående kodeudsnit viser denne funktionalitet.

```
[HttpGet("{id}")]
2 references
public async Task<ActionResult<Product>> GetProduct(int id)
{
    try
    {
        var product = await _context.Products
            .Include(p => p.Prices)
            .FirstOrDefaultAsync(p => p.ProductId == id);

        if (product == null)
        {
            return NotFound();
        }

        return Ok(product);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal server error: {ex.Message}");
    }
}
```

Nedenstående tabel viser det relevante endpoint og dets anvendelse til at gøre denne user story mulig:

Endpoint	HTTP Metode	Beskrivelse
/api/products	GET	Henter alle produkter inklusiv deres tilknyttede priser.

Tabel 7 7: Brugt endpoint for US1

## Frontend

Frontenden for denne user story er centreret omkring SearchProductPage-komponenten. Her præsenteres brugeren for en grænseflade, der gør det muligt at søge efter produkter og sammenligne deres priser.

Søgeflowet er implementeret med følgende trin:

- **Automatiske søgeforslag:** Når brugeren skriver i søgefeltet, fetches der data fra backend baseret på inputtet. Søgeresultater grupperes efter uniqueItemIdentifier og vises som forslag. Søgefeltet bruger PlaceholdersAndVanishInput-komponenten, som dynamisk viser placeholder-tekster og understøtter realtids-fetching fra backend.
- **Valg af produkt:** Brugeren kan vælge et produkt fra listen ved enten at klikke eller bruge pilene til navigation.
- **Produktresultater:** Efter valg fetches detaljerede produktdata, herunder priser fra forskellige butikker, relevante billeder og logoer.

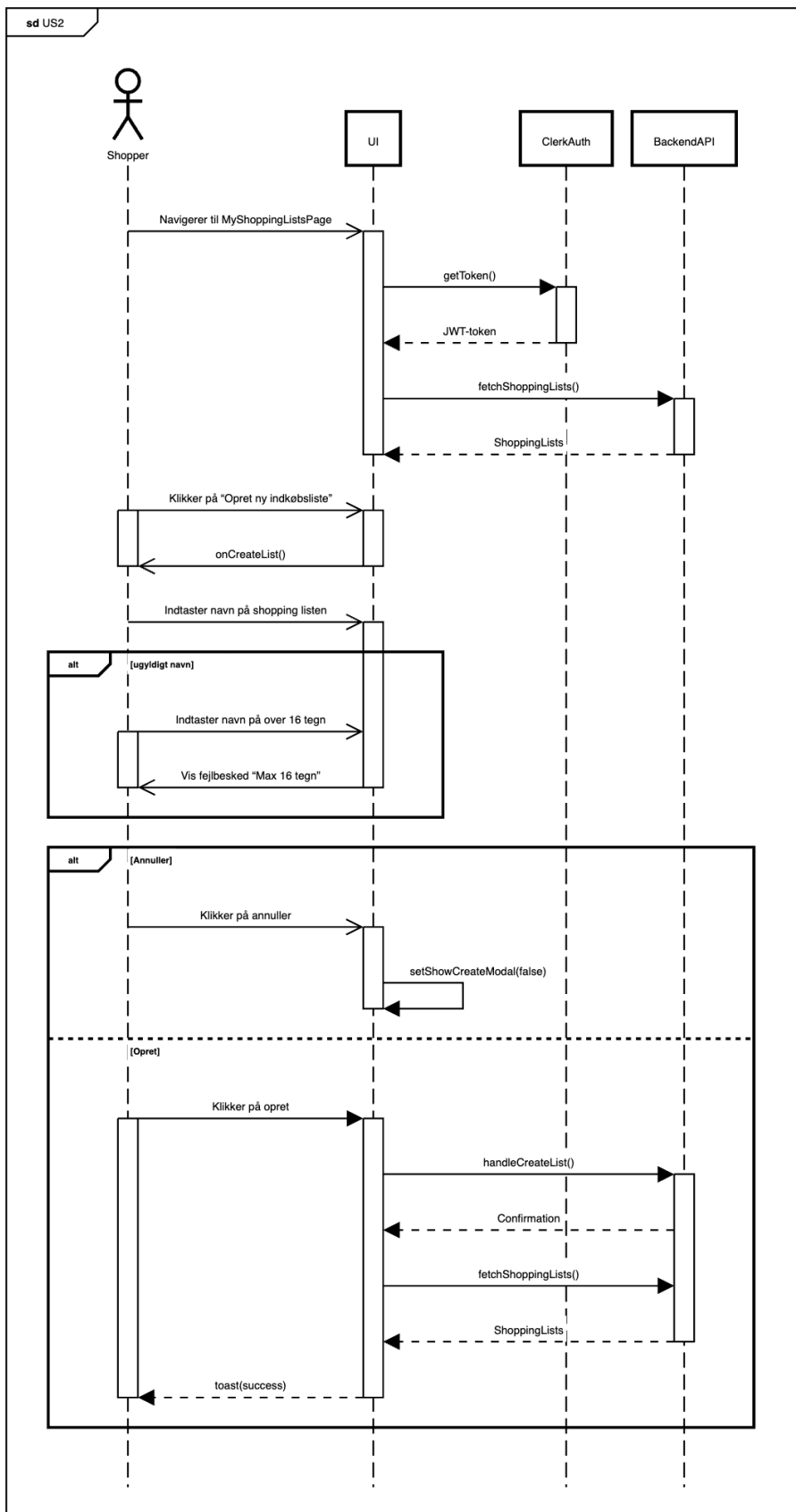
## User Story 2: Oprette en ny indkøbsliste

### Beskrivelse

Denne user story omhandler implementeringen af funktionalitet til oprettelse af en ny indkøbsliste, som giver brugerne mulighed for at organisere deres indkøb effektivt ved at oprette flere lister til forskellige behov. Formålet med denne funktion er at give brugerne fleksibilitet i planlægningen af deres indkøb, f.eks. ved at kunne adskille daglige behov fra større indkøb eller specifikke projekter.

Når en bruger vælger at oprette en ny indkøbsliste, kan de angive en titel for listen, som gemmes i systemet. Backend sikrer, at den oprettede liste knyttes til den aktuelle bruger og kun er tilgængelig for denne, når brugeren er logget ind. Hvis der opstår fejl under oprettelsen, som f.eks. manglende oplysninger eller andre valideringsproblemer, vil systemet give brugeren en passende fejlbesked, der muliggør rettelse og et nyt forsøg.

Implementeringen af denne user-story er skitseret med følgende sekvensdiagram:



Figur 2323: US2 uddybet sekvensdiagram

Sekvensdiagrammet på Figur 23 viser interaktionen mellem shopperen og systemets komponenter, når shopperen skal oprette en ny indkøbsliste. Denne interaktion kan inddeles i fire hovedtrin, som kan ses forklaret i bilag [7.1.1].

### Backend

Backend-delen er implementeret med en dedikeret `ShoppingListController`, der håndterer CRUD-operationer for shoppinglister. Controlleren inkluderer et POST-endpoint, `/api/shoppinglist`, som muliggør oprettelsen af en ny indkøbsliste. Når frontend sender en anmodning med de nødvendige data for en ny indkøbsliste, tilføjer `CreateShoppingList`-metoden listen til databasen og returnerer en bekræftelse sammen med den oprettede liste.

```
[HttpPost]
public ActionResult<ShoppingList> CreateShoppingList(ShoppingList shoppinglist)
{
    _context.ShoppingLists.Add(shoppinglist);
    _context.SaveChanges();

    return CreatedAtAction(nameof(getShoppingList), new { id = shoppinglist.ShoppingListId }, shoppinglist);
}
```

```
POST /api/shoppinglist
Content-Type: application/json
Body: {
  "title": "Indkøb til dagligvarer",
  "shopperId": "123"
}
```

Figur 2424: Eksempel på POST-anmodning til oprettelse af indkøbsliste

Denne funktionalitet sikrer, at lister oprettes med en unik ID, en tilknytning til den aktuelle bruger via `ShopperId`, og en automatisk genereret oprettelsesdato. Dette gør det muligt at oprette og gemme flere lister for hver bruger.

Derudover er det designet sådan, at hver liste kun er tilgængelig for den bruger, der har oprettet den, og er beskyttet via brugerens autentificering og autorisation i systemet. Endpointet returnerer statuskoden 201 Created, hvis oprettelsen er vellykket, og en passende fejlbesked, hvis der opstår problemer under processen.

Nedenstående tabel viser de tilgængelige endpoints og deres anvendelse i applikationen:

Endpoint	HTTP-metode	Beskrivelse
<code>/api/shoppinglist</code>	GET	Hent alle shoppinglister.
<code>/api/shoppinglist</code>	POST	Opret en ny shoppingliste
<code>/api/shoppinglist/{id}</code>	PUT	Opdater en eksisterende shoppingliste.
<code>/api/shoppinglist/{id}</code>	DELETE	Slet en eksisterende shoppingliste.

Tabel 88: `ShoppingListController` endpoints

## Frontend

Frontend-delen for implementeringen af funktionen til oprettelse af en ny indkøbsliste er en integreret del af siden MyShoppingListsPage. Her vises brugerens eksisterende indkøbslister i en sidebar-komponent, ShoppingListSidebar, der giver mulighed for at oprette, redigere og slette shoppinglister.

### Visning og interaktion

Sidebaren viser en oversigt over brugerens indkøbslister med deres titler. Øverst fremgår brugerens navn efterfulgt af teksten "indkøbslister". Hver liste vises som en klikbar række med muligheden for at redigere titlen eller slette den specifikke liste. Nederst i sidebar-komponenten findes en knap "Opret ny indkøbsliste", som åbner et inputfelt, hvor brugeren kan angive en titel for den nye liste. Når titlen er indtastet, gemmes listen som en tom indkøbsliste i systemet.

Koden nedenfor viser funktionen, der kaldes, når brugeren klikker på knappen for at oprette en ny liste:

```
<button
  className="w-full p-2 text-white bg-gradient-to-r from-indigo-500
  to-purple-600 rounded-md hover:from-indigo-600 hover:to-purple-700"
  onClick={onCreateList}
>
  <PlusIcon className="h-5 w-5 inline-block mr-2" />
  Opret ny indkøbsliste
</button>
```

### API-integration

Når brugeren bekræfter oprettelsen af en ny liste, sendes en anmodning til backend via en POST-metode. Dette gøres i funktionen handleCreateList på siden MyShoppingListsPage, der sender data som liste-titel og user-ID til backend. Hvis oprettelsen lykkes, opdateres listen dynamisk i sidebar-komponenten.

### UI og brugeroplevelse

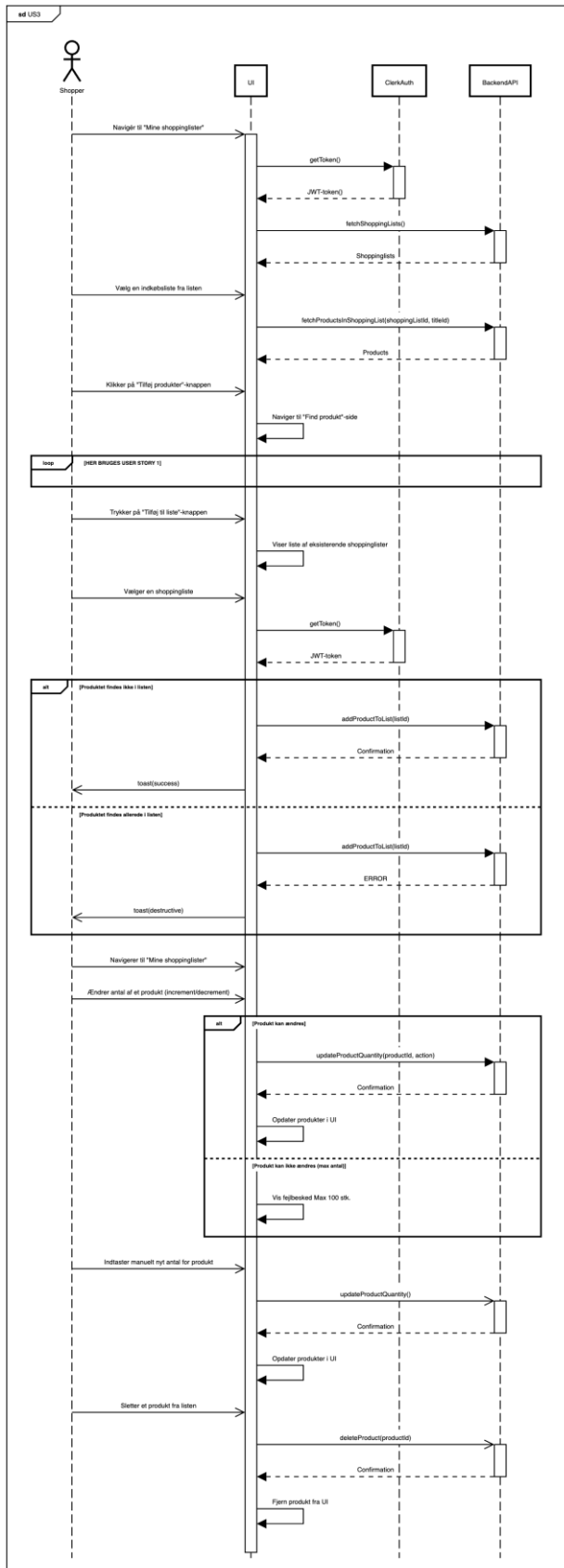
- Brugeren får visuel feedback i form af en toast-notifikation, der viser, om listen blev oprettet succesfuldt, eller om der opstod en fejl.
- For at sikre en intuitiv oplevelse fremhæves aktive lister med en gradient baggrundsfarve, mens inaktive lister vises med en mere dæmpet stil.
- En fejlbesked vises, hvis brugeren forsøger at indsende en tom titel eller en titel, der overstiger 16 tegn.

## User Story 3: Ændre en indkøbsliste

### *Beskrivelse*

Denne user story omhandler implementeringen af funktionen til at redigere en indkøbsliste ved at tilføje, fjerne eller ændre antallet af varer. Formålet er at give brugerne mulighed for at opretholde en opdateret og relevant liste, der matcher deres aktuelle indkøbsbehov.

Implementeringen af denne user-story er skitseret med følgende sekvensdiagram:



Figur 2525: US3 uddybet sekvensdiagram



Sekvensdiagrammet på Figur 25 viser interaktionen mellem shopperen og systemets komponenter, når shopperen skal ændre en indkøbsliste. Denne interaktion kan inddeles i følgende syv hovedtrin som kan ses forklaret uddybende i bilag [7.1.1].

### Backend

Backend-delen er implementeret med en dedikeret ShoppingListProductController, der håndterer alle operationer relateret til produkter i indkøbslister. Controlleren indeholder følgende nøglefunktioner:

Brugeren kan administrere produkter på en indkøbsliste via API'et. Produkter tilføjes med POST /api/ShoppingListProduct, hvor en UniqueId-identifikator sikrer korrekt gruppering. Antallet kan justeres med PUT /api/ShoppingListProduct/increment eller /decrement, eller opdateres direkte med PUT /api/ShoppingListProduct/updateQuantity. Sletning af produkter håndteres med DELETE /api/ShoppingListProduct, hvor indkøbslisten og produkt-ID'et angives som parametre.

I kodeudsnittet nedenfor ses et eksempel på en metode til opdatering af produktmængden i en liste:

```
[HttpPut("updateQuantity")]
public IActionResult UpdateProductQuantity([FromQuery] int sId, [FromQuery] int pId, [FromQuery] int newQuantity)
{
    var shoppinglistproduct = _context.ShoppingList_Products
        .FirstOrDefault(slp => slp.ShoppingListId == sId && slp.ProductId == pId);

    if (shoppinglistproduct == null)
    {
        return NotFound();
    }

    if (newQuantity == 0)
    {
        _context.ShoppingList_Products.Remove(shoppinglistproduct);
    }
    else
    {
        shoppinglistproduct.Quantity = newQuantity;
    }

    _context.SaveChanges();
    return NoContent();
}
```

Nedenstående tabel viser de tilgængelige endpoints og deres anvendelse i applikationen:

Endpoint	HTTP-metode	Beskrivelse
/api/ShoppingListProduct	POST	Tilføj et produkt til en indkøbsliste.
/api/ShoppingListProduct/increment	PUT	Øg antallet af et produkt i en indkøbsliste.
/api/ShoppingListProduct/decrement	PUT	Reducer antallet af et produkt i en indkøbsliste.
/api/ShoppingListProduct/updateQuantity	PUT	Opdater antallet af et produkt til en bestemt værdi.
/api/ShoppingListProduct	DELETE	Fjern et produkt fra en indkøbsliste.

Figur 2626: ShoppingListProductController endpoints

## Frontend

Frontend-funktionen til at ændre en indkøbsliste er fordelt på SearchProductPage.tsx og MyShoppingListsPage.tsx og understøtter tilføjelse, redigering og sletning af produkter. På SearchProductPage kan brugeren søge efter produkter og tilføje dem til en valgt indkøbsliste. Ved klik på "Tilføj til liste" åbnes en modal med brugerens indkøbslister, hvor en liste vælges. En POST-anmodning med shoppingListId og UniqueItemIdentifier sendes til backend, som håndterer tilføjelsen. På MyShoppingListsPage vises produkterne som en liste med navn, billede, mængde og priser fra forskellige butikker. Brugeren kan justere mængden via "+" og "-"-knapper eller fjerne produkter helt fra listen. Koden nedenfor viser funktionen, der udføres, når et produkt tilføjes til en indkøbsliste.

```
const addProductToList = async (listId: number) => {  
  const response = await fetch('http://localhost:5000/api/ShoppingListProduct', {  
    method: 'POST',  
    headers: { Authorization: `Bearer ${token}`, 'Content-Type': 'application/json' },  
    body: JSON.stringify({ shoppingListId: listId, uniqueItemIdentifier: selectedUniqueItemIdentifier }),  
  });  
};
```

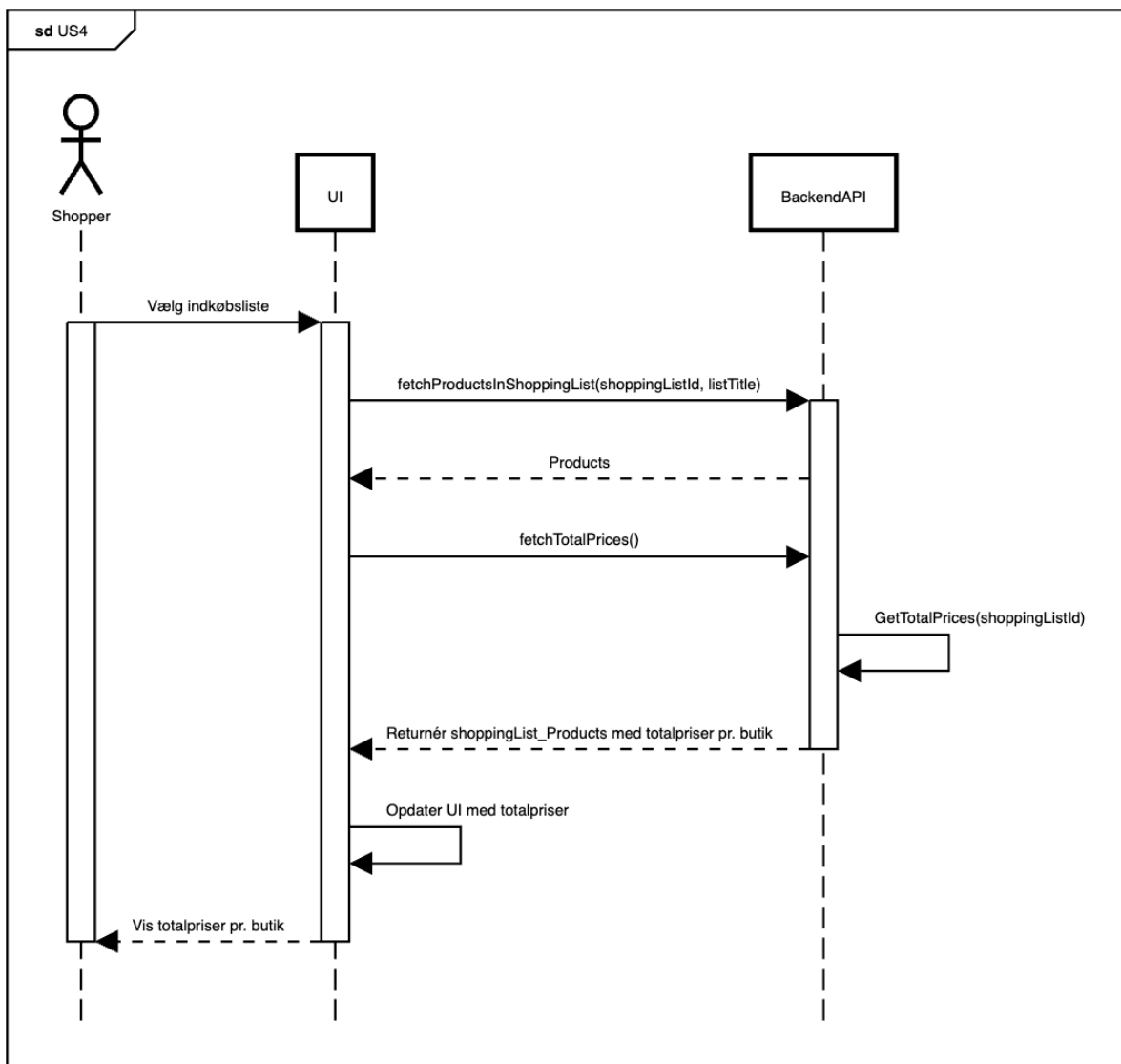
I MyShoppingListsPage sendes PUT- eller DELETE-anmodninger via funktionerne updateProductQuantity og deleteProduct ved redigering eller sletning af produkter. Ændringer opdateres øjeblikkeligt på siden, og brugeren modtager beskeder om succes eller fejl. Ved sletning eller ændring til 0 vises en bekræftelsesboks for godkendelse. For yderligere dokumentation af resultaterne for US3 henvises der til bilag [7.1.1]

## User Story 4: Sammenlign priser

### Beskrivelse

Denne user story giver brugeren en oversigt over samlede priser fra forskellige butikker, så de nemt kan finde det billigste sted at handle og optimere deres indkøb.

Implementeringen af denne user-story er skitseret med følgende sekvensdiagram:



Figur 27 27: US4 uddybet sekvensdiagram

Sekvensdiagrammet på Figur 27 viser interaktionen mellem shopperen og systemets komponenter, når shopperen skal ændre en indkøbsliste. Denne interaktion kan inddeles i 3 hovedtrin, som er forklaret uddybende i bilag [7.1.1].

### Backend

For at understøtte denne funktion er der blevet oprettet et GET-endpoint i `ShoppingListProductController /api/ShoppingListProduct/totalPrices/{shoppingListId}`. Dette endpoint beregner den samlede pris for hver butik baseret på produkterne i en valgt indkøbsliste.

Endpointet `GetTotalPrices` udfører følgende trin for at beregne de samlede priser:

1. Alle produkter i den valgte indkøbsliste hentes fra databasen sammen med deres priser og detaljer.

```
var products = await _context.ShoppingList_Products
    .Where(slp => slp.ShoppingListId == shoppingListId)
    .Include(slp => slp.Product)
    .ThenInclude(p => p.Prices)
    .ToListAsync();
```

2. Produkterne grupperes baseret på deres unikke identifikator.

```
var groupedProducts = products
    .GroupBy(p => p.Product.UniqueItemIdentifier);
```

3. For hver produktgruppe identificeres alle butikker, hvor produktet er tilgængeligt.

```
var sources = group
    .SelectMany(item => item.Product.Prices)
    .Select(p => p.Source)
    .Distinct();
```

4. Totalprisen for hver butik sættes til at starte på 0 kr.

```
var totals = new Dictionary<string, decimal>();

foreach (var source in sources)
{
    if (!totals.ContainsKey(source))
    {
        totals[source] = 0;
    }
}
```

5. For hver butik hentes priserne for dets produkt i hver produktgruppe.

```
var prices = group
    .SelectMany(item => item.Product.Prices)
    .Where(p => p.Source == source)
    .Select(p => p.ProductPrice);
```

6. Antallet af produkter i en gruppe hentes fra det første produkts Quantity-værdi, da det er ens for alle i gruppen. Priserne ganges med antallet, og resultatet lægges til den samlede pris for hver butik. Dette gentages for alle butikker fra step 3.

```
var quantity = group.First().Quantity;
foreach (var price in prices)
{
    totals[source] += price * quantity;
}
```

7. Resultatet returneres som en liste af TotalPrice-objekter. Hvert objekt indeholder navnet på en butik og den samlede pris for alle produkter i den valgte indkøbsliste, beregnet for den pågældende butik. Priserne afrundes til 2 decimaler.

```
public class TotalPrice
{
    public string Source { get; set; }
    public decimal Total { get; set; }
}

var result = totals.Select(t => new TotalPrice
{
    Source = t.Key,
    Total = Math.Round(t.Value, 2)
});

return Ok(result);
```

Den fulde kode til endpointet GetTotalPrices kan findes i bilaget [5.3.1] i filen ShoppingListProductController.cs.

### Frontend

Frontend-delen er integreret i MyShoppingListsPage, hvor komponentet TotalPrices bruges til at vise de samlede priser.

Priser fra forskellige butikker vises med logoer, navne og den samlede pris. Butikken med den laveste pris fremhæves med en grøn ring og tekst, så den nemt kan identificeres. Komponenten vises kun for indkøbslister med produkter for at bevare en ren brugergrænseflade.

Når en bruger vælger en indkøbsliste, henter MyShoppingListsPage priserne fra backend ved hjælp af GET-anmodningen til /api/ShoppingListProduct/totalPrices/{shoppingListId}. Dataene sendes derefter til TotalPrices-komponenten for at blive vist.

## 7.5. Yderligere implementeringer

I dette afsnit gennemgås de resterende essentielle tekniske løsninger, der er med til at sikre, at applikationen fungerer effektivt. Det omfatter automatisering af opgaver og dataindsamling.

### 7.5.1. WebScraper

En central funktion i Purchase4Less er evnen til at indsamle priser fra forskellige supermarkeder, hvilket opnås gennem en avanceret web scraping-løsning. Systemet er bygget til at håndtere scraping af produktdata fra forskellige danske dagligvarekæder ved hjælp af en række specialiserede scrapere. Dette afsnit beskriver implementeringen af web scraping-løsningen, som har til ansvar at indsamle relevant data direkte fra supermarkedernes egne hjemmesider.

#### Implementering

Back-end-delen af web-scraperen er implementeret i C# ved hjælp af PuppeteerSharp, som er en .NET-port af det standard JavaScript-baserede webscraping-bibliotek Puppeteer. PuppeteerSharp blev valgt som den primære scraping-engine, da det tilbyder flere fordele out-of-the-box, herunder:

- Muligheden for at køre i headless mode, hvilket resulterer i hurtig og effektiv scraping.

- Support til at scrape indhold, der genereres dynamisk af JavaScript.
- Understøttelse af skærbilleder, hvilket kan bruges til debugging.

Systemets scraping-funktionalitet er implementeret gennem et samspil mellem tre hovedkomponenter: *WebScraperController*, *ProductSeeder* og *DatabaseController*.

### *Product Seeder*

I *ProductSeeder.cs* defineres alle URL'er sammen med deres tilhørende unikke identifikatorer for hvert produkt. Denne struktur fungerer som en "mapping", der forbinder URL'er med deres respektive produktidentifikatorer. Dette er afgørende for systemets evne til at genkende og sammenligne de samme produkter på tværs af forskellige supermarkeder.

```
public static readonly List<(string Url, string UniqueIdentifier)> RemaProducts = new()
{
    ("https://shop.rema1000.dk/varer/304000", "Banan"),
    ("https://shop.rema1000.dk/varer/21464", "Minimælk"),
    ("https://shop.rema1000.dk/varer/61508", "Solsikkerugbrød"),
    ("https://shop.rema1000.dk/varer/304070", "Æble"),
    ("https://shop.rema1000.dk/varer/220017", "Æg 8 Øko"),
}
```

Den fulde implementation af *ProductSeeder* kan ses i bilag [5.2.2].

### *WebScraperController*

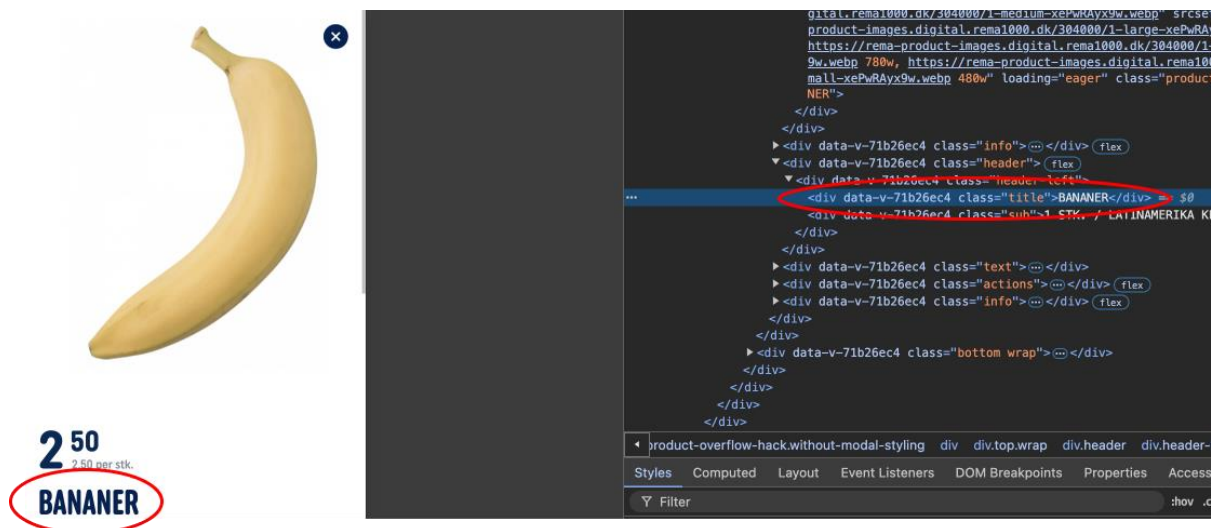
Den specifikke web scraper-funktionalitet er implementeret i *WebScraperController.cs*, som styrer indsamlingen af produktdata fra forskellige supermarkeds kæder. For hver kæde er der udviklet specialiserede scraping-metoder, der håndterer de unikke HTML-strukturer på de respektive hjemmesider. Disse metoder anvender JavaScript til at udtrække den nødvendige produktinformation.

I *WebScraperController* bliver der automatisk identificeret, hvilken butik-scraper der skal kaldes, baseret på URL'en, som vises i nedenstående kodeudsnit.

```
// Bestem hvilken scraping logik der skal bruges baseret på URL
if (request.Url.Contains("bilkatogo.dk"))
{
    return await ScrapeBilka(page);
}
else if (request.Url.Contains("rema1000.dk"))
{
    return await ScrapeRema(page);
}
else if (request.Url.Contains("spar.dk"))
{
    return await ScrapeSpar(page);
}
```

Web scraperen anvender JavaScript selectors til præcist at identificere og udtrække specifikke elementer fra websiden. Som illustreret i Figur 28, henter systemet data fra Rema 1000's hjemmeside, hvor produktnavnet i HTML-strukturen er markeret med attributten 'data-v-71b26ec4'. Dette match er blevet implementeret i JavaScript-koden, som vist i nedenstående kodeudsnit, hvilket sikrer at netop produktinformation der repræsenterer "produktnavn" udtrækkes fra siden.

```
private async Task<IActionResult> ScrapeRema(IPage page)
{
    try
    {
        var name = await page.EvaluateExpressionAsync<string>(
            "document.querySelector('div[data-v-71b26ec4].title').innerText"
        );
    }
}
```



Figur 2828: Rema 1000 hjemmesidens klassenavn for en banan i inspect tools

I kodeudsnittet nedenfor ses et eksempel på den information, der bliver returneret for enhver given butik. Hvis værdierne ikke findes eller er tomme, returneres standardværdier som "Ikke fundet" for navnet og "Ukendt mærke" for brandet. Værdien for "Store" fastsættes af den specifikke web scraper baseret på URL'en, således at "Bilka" eksempelvis altid returneres, hvis URL'en indeholder "bilka.dk". Priserne behandles af metoden ExtractPrice, der renser prisdata ved at fjerne unødvendige tegn og sikrer, at en gyldig pris altid returneres, selv hvis prisen ikke er korrekt angivet i det oprindelige data.

```
return Ok(new
{
    name = name?.Trim() ?? "Ikke fundet",
    brand = brand?.Trim() ?? "Ukendt mærke",
    store = "Bilka",
    image = image?.Trim() ?? "",
    price = ExtractPrice(price ?? "0")
});
```

Den fulde implementation af WebScraperController() kan ses i bilag [5.2.1]

### *DataBaseController*

DatabaseController's SeedProducts metode står for at rydde eksisterende produktdata, iterere gennem hver butiks produktliste, kalde den respektive web scraper, og gemme de scrapede data i databasen. For hver butik gentages dette mønster, hvor den eneste forskel er den specifikke datakilde fra URL-listen i ProductSeeder.cs.

Metoden sikrer, at tidligere produktdata ryddes for at opretholde høj datakvalitet, før nye produkter og priser tilføjes fra de respektive butikker. Hver scraping-operation er indkapslet i en try-catch blok, så eventuelle fejl logges uden at afbryde den samlede proces. Dette design giver robusthed, idet systemet fortsætter, selv hvis enkelte scraping-operationer fejler.

Re-seed processen kan køres manuelt gennem API'et som det ses i Admin panelet, eller automatisk via et planlagt CRON-job, hvilket giver fleksibilitet i forhold til opdateringsfrekvensen.

Den fulde implementation af Seed metoden fra DataBaseController() kan ses i bilag [5.2.3]

### *Risiko*

Dog er der visse udfordringer forbundet med løsningen. En væsentlig risiko er afhængigheden af eksterne faktorer, såsom ændringer i butikkernes CSS-klasser, der kan påvirke dataudtrækningen. Derudover er systemet udsat for risikoen for at blive IP-bannet, da der sendes mange requests hurtigt til forskellige supermarkeder. Dette kræver forsigtighed og på længere sigt kunne opsætning af et proxy-netværk eller VPN blive nødvendigt, for at sikre kontinuerlig drift.

### *Skalering*

Web scraperens opsætning gør systemet skalerbart og vedligeholdelsesvenligt. Et eksempel sås da gruppen tilføjede SPAR-butikken lang tid efter Bilka og Rema 1000 var integreret. For at tilføje SPAR var det blot nødvendigt at definere en ekstra butik i WebScraperController, finde de relevante CSS-klasser og indtaste en række URL'er.



## 7.5.2. CRON JOB

I Purchase4Less systemet er det afgørende at kunne tilbyde brugerne aktuelle og pålidelige prissammenligninger på tværs af forskellige supermarkeder. For at opretholde denne funktionalitet blev et automatiseret CRON-job implementerede, der regelmæssigt opdaterer produkt info og priser. Dette er særligt vigtigt, da supermarkedernes priser ofte ændrer sig, og manuel opdatering ville være både tidskrævende og upålideligt. Det implementerede CRON-job har til formål at sikre, at systemet automatisk indhenter nye priser hver søndag ved midnat, hvilket giver brugerne pålidelige prissammenligninger i starten af hver ny uge.

Yderligere dokumentation ift. CRON-job implementation kan ses i Bilag [7.4]

## 8. Test

Dette afsnit beskriver de tests, der er udført for at verificere, at webapplikationen *Purchase4Less* fungerer som forventet og opfylder de specificerede krav. Der beskrives de forskellige testmetoder og resultater for testprocessen.

### 8.1. Teststrategi

For at sikre kvaliteten af applikationen er der blevet lavet en kombination af enhedstest, integrationstest og systemtest.

- **Enhedstest:** Enhedstest bruges til at sikre, at metoder og funktioner i applikationens controllere fungerer korrekt isoleret. Testene simulerer afhængigheder som databaser og API'er ved hjælp af mocking og in-memory databaser.
- **Integrationstest:** Integrationstests har primært været manuelle og brugt en kombination af bottom-up og top-down tilgange for at kunne sikre nye funktioner fungere sammen med eksisterende funktionalitet. Integrationstestene blev udført på følgende måde:
  - En ny branch i GitLab blev oprettet for hver ny funktion eller side, der skulle implementeres.
  - Funktionen blev udviklet og testet isoleret på sin egen branch.
  - Når funktionen fungerede korrekt, blev den merged med main-branch, hvorefter den egentlige integrationstest begyndte.
  - Det blev sikret, at alle eksisterende funktioner stadig fungerede korrekt efter merge.
  - Den nye funktion blev testet for at bekræfte, at den fungerede som forventet efter merge.

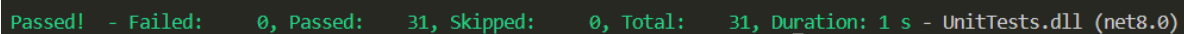
De fleste integrationstest fulgte en bottom-up tilgang, hvor grundlæggende funktioner som brugeroprettelse og login blev testet først. Enkelte funktioner, som oprettelse af en shopper, blev testet med en top-down strategi for at sikre korrekt funktionalitet i en overordnet kontekst og samarbejde med systemets øvrige dele.

- **Systemtest:** Systemtesten blev udført med en forsøgsperson, der gennemgik accepttest og ikke-funktionelle krav på hjemmesiden, som blev dokumenteret.

## 8.2. ZOMBIE-teststrategi

Der blev anvendt ZOMBIE-teststrategi til at test forskellige komponenter og funktioner i applikationen. Alle test med fuld beskrivelse kan ses i bilag [3]. Dokumentation af ZOMBIE test strategien kan ses i bilag [6.6.2]. Nedenfor er nogle eksempler på hvordan ZOMBIE blev benyttet i dette projekt:

- **Zero:** Test af tomme eller nul input for at sikre systemet fejlhåndter korrekt: For eksempel blev der testet for et specifikt produkt, hvis det ikke eksisterede, skulle den kunne håndtere fejlen.
- **One:** Test med enkelt input for at sikre, korrekt funktionalitet med minimumsdata. For eksempel blev der testet, hvor en specifik Shopper blev hentet baseret på ID.
- **Many:** Test med flere inputs for at sikre, at systemet kan håndtere store mængder data og komplekse operationer. Her blev der bl.a. testet ved at hente alle eksisterende produkter i en liste.
- **Boundary:** Test af grænsetilfælde for at sikre, at systemet håndterer ekstreme værdier korrekt. F.eks. er der blevet lavet en grænse for når man ønsker at hente et specifikt produkt.
- **Interface:** Test af integration mellem systemkomponenter for at sikre, at data bliver overført og behandlet korrekt. F.eks. testes der for specifikke produkter at retur værdien stemmer overens med den forventede data.
- **Exceptional behavior:** Test af systemets fejlhåndtering og undtagelser for at sikre, at applikationen reagerer korrekt i tilfælde af fejl. F.eks. ved test for ikke eksisterende produkter.



```
Passed! - Failed: 0, Passed: 31, Skipped: 0, Total: 31, Duration: 1 s - UnitTests.dll (net8.0)
```

Figur 2929: Gennemførte tests

På Figur 29 ses at alle test er gennemført korrekt.

## 8.3. Test af backend

Test for backend-komponenterne er beskrevet detaljeret i bilag [3.1].

## 8.4. Intergrationstest

Der er fokuseret på at sikre korrekt integration mellem backend, database og frontend. Unittests blev prioriteret for backend, mens integrationstest blev udført løbende efter færdiggørelse af større funktionaliteter. Testene kontrollerede, om produkter eksisterede i systemet og kunne vises korrekt, hvilket sikrede, at Web scraperen leverede korrekte data. Integrationstestene afslørede hurtigt fejl og forbedrede systemets robusthed.

For yderligere dokumentation henvises der til bilag [3.1].

## 9. Resultater

I dette afsnit præsenteres resultaterne af gruppens arbejde med udviklingen af Purchase4Less webapplikationen. De opnåede resultater gennemgås i forhold til de opstillede mål og krav.

### 9.1. Accepttestspecifikation

Nedenfor følger en gennemgang af accepttesten baseret på tidligere definerede Gherkin-scenarier. En mere detaljeret gennemgang med yderligere dokumentation findes i bilag [4.1].

MoSCoW	User Story	Vurdering
<b>Must</b>	User story 1: Søge efter én specifik vare	Godkendt
<b>Must</b>	User Story 2: Oprette en ny indkøbsliste	Godkendt
<b>Must</b>	User Story 3: Ændre en indkøbsliste	Godkendt
<b>Must</b>	User Story 4: Sammenlign priser	Godkendt
<b>Could</b>	User Story 5: Google Maps	Godkendt
<b>Could</b>	User Story 6: Notifikationer	Delvist Godkendt
<b>Must</b>	User Story 7: Oprette bruger og login	Godkendt
<b>Should</b>	User Story 8: Vælge specifikke butikker	Ikke Godkendt
<b>Should</b>	User Story 9: Se statistikker som administrator	Godkendt
<b>Could</b>	User Story 10: Tilføje fødevarerbutikker	Ikke godkendt

Tabel 99: Accepttestspecifikation

### 9.2. Opsummering af resultaterne

Det fremgår af Tabel 9 at alle funktionelle *Must*-krav er opfyldt. De ikke- og delvist godkendte funktionelle krav er enten blevet delvist implementeret eller ikke implementeret. De ikke-funktionelle krav kan ses i bilag [3.2].

**US1:** Bekræftede udvikling af søgefunktion for specifikke vare som tillader shopperen at kunne finde ønskede produkt.

**US2 og US3:** Bekræftede at shopperen kan oprette en indkøbsliste og tilføje produkter til den. Dertil har shopper også mulighed for at kunne tilgå deres indkøbsliste og redigere den.

**US4:** Bekræftede at shopperen har mulighed for at sammenligne priser af et produkt.

**US5:** Bekræftede at shopperen har mulighed for at tilgå google maps og se butikker i nærheden.

**US7:** Bekræftede at shopperen kan oprette bruger og logge ind ved benyttelse af Clerk.

**US9:** Bekræftede, ved login som administrator, får admin adgang til en adminside, hvor statistikker om antal shoppere, indkøbslister samt butikssammenligninger af produktpriser vises.

**US6:** Delvist godkendt, da shopper ikke bliver notificeret ved prisændring.

**US8 og US10:** Ikke accepteret. Administrator har ikke mulighed for at kunne tilføje butikker eller filtrere fra adminsiden hvilke butikker, der skal være aktive under søgning for en shopper.

For yderligere dokumentation af resultater og eksempler henvises der til bilag [7.1.1]

## Diskussion

Udviklingsprocessen for vores projekt har givet en række værdifulde resultater og erfaringer.

Der er blevet udarbejdet en funktionel webapplikation, der adresserer de centrale krav i projektets problemformulering. Alle "Must"-krav er blevet implementeret med succes, hvilket sikrer, at brugerne kan søge efter varer, oprette og ændre indkøbslister, samt sammenligne priser mellem butikker. Disse funktioner udgør kernen af systemet og er essentielle for at levere værdi til brugerne.

I processen valgte gruppen at prioritere brugerrelaterede funktioner frem for admin-funktionalitet, da det blev vurderet, at dette havde størst betydning for slutbrugerne. Denne prioritering var nødvendig for at sikre en stabil løsning inden for den givne tidsramme. I processen oplevede gruppen tidspres, da der ikke var taget tilstrækkeligt hensyn til arbejdsbelastningen fra andre fag i semesterets sidste halvdel. Dette resulterede i, at mindre kritiske funktioner som "Tilføj fødevarebutikker" og "Avanceret filtrering" blev nedprioriteret, hvilket var årsagen til, at visse "Should"- og "Could"-krav ikke blev fuldt implementeret.

En af de største udfordringer, gruppen stødte på under udviklingsprocessen, var manglende erfaring med Git workflow. I projektets indledende faser opstod gentagne merge-konflikter, da arbejdet ikke blev organiseret i branches. Dette skabte ineffektivitet og frustration blandt gruppemedlemmerne. Problemet blev løst efter et møde med vejlederen, der introducerede en struktureret tilgang til brugen af Git branches, hvilket markant forbedrede workflowet og reducerede konflikter.

En anden udfordring var opdelingen af opgaver inden for kategorier som frontend, backend, database og test. Selvom denne struktur var designet til at skabe overblik og effektivitet, blev det vanskeligt at holde opgaverne adskilt, da mange var tæt kobledede, hvilket resulterede i forsinkelser og udfordringer i samarbejdet.

Gruppen løste dette ved at ændre tilgangen til opgavedelingen og i stedet tage udgangspunkt i user stories. Hver user story blev tildelt til mindre teams af 2 personer, som så fik ansvaret for at implementere hele funktionaliteten fra frontend til backend og database. Denne tilgang gav teams ejerskab over deres opgaver og reducerede afhængigheder, hvilket resulterede i en mere effektiv og overskuelig udviklingsproces.

Vi er særligt tilfredse med implementeringen af søge- og sammenligningsfunktionen, som vi anser for en af de mest centrale og vellykkede dele af projektet. Funktionerne er intuitive for brugerne og leverer præcise resultater, hvilket blev bekræftet gennem accepttestene. Derudover er integrationen af

brugeroprettelse og login en anden vigtig milepæl, som på trods af kompleksiteten ved at implementere en tredjeparts identity management-service, har været både udfordrende og særdeles lærerigt.

Et andet område, vi er stolte af, er den implementerede web scraping-løsning, som adresserede en væsentlig udfordring tidligt i udviklingsprocessen. Oprindeligt forventede vi at kunne hente produktinformation fra diverse dagligvarebutikkers API'er, men da dette viste sig ikke at være muligt, måtte vi udvikle en alternativ løsning. Dette førte til, at vi opbyggede en web scraper fra bunden, hvilket krævede en dyb forståelse af teknologien og en grundig vurdering af, hvilke dagligvarebutikker det var muligt at scrape data fra.

Denne proces var ikke blot en teknisk løsning, men også en mulighed for at praktisere og anvende gode softwareprincipper i praksis. Vi fokuserede på at gøre løsningen skalerbar, så den fremadrettet kan tilpasses og udvides med flere butikker, hvis behovet skulle opstå. Samlet set repræsenterer web scraping-løsningen både en kreativ og teknisk succes, som har været afgørende for projektets funktionalitet.

Vores teknologivalg har generelt fungeret godt i udviklingen af applikationen. Frontend er implementeret i React, hvilket har givet os mulighed for at arbejde med TypeScript og opdele UI i genanvendelige komponenter. Dog oplevede vi en læringskurve med React, da det var flere gruppemedlemmers første gang med frameworket og ikke alle i gruppen var konsekvente i at opdele deres sider i komponenter undervejs. Dette blev i starten anset som noget, der kunne løses til sidst, men det har ført til hængepartier, som kunne være undgået ved at følge best practice fra begyndelsen.

På backend-siden har .NET Frameworket været et stærkt valg. Det veldokumenterede framework har gjort det nemt at finde løsninger på udfordringer, og brugen af eksterne biblioteker som Quartz til CRON-job og Puppeteer Sharp til web scraping har udvidet funktionaliteten betydeligt. Derudover har arbejdet i C# været en fordel for gruppen, da sproget minder om C++, som vi har erfaring med fra tidligere semestre.

## Konklusion

Ud fra produktet og rapporten kan projektgruppe 5 konkludere, at der er blevet udviklet en funktionel webapplikation, der succesfuldt adresserer problemformuleringen om at udvikle en brugervenlig og effektiv løsning til prissammenligning af dagligvarer. Systemet understøtter kernefunktioner som brugerlogin, oprettelse og redigering af indkøbslister samt muligheden for at finde butikker tæt på brugerens lokation. Derudover er en automatisk sammenligningsfunktion implementeret, som giver brugerne aktuelle priser på 26 unikke dagligvarer fra Bilka, Rema 1000 og SPAR.

Ved at prioritere "Must"-kravene har vi sikret, at de vigtigste funktioner er blevet implementeret med succes, hvilket gør Purchase4Less til en komplet og værdiskabende løsning for slutbrugerne. Trods udfordringer som tidsbegrænsninger og enkelte uafsluttede funktioner, herunder avanceret filtrering og tilføjelse af nye butikker, har projektet opnået de væsentligste mål om at spare tid og penge for brugerne gennem en effektiv og brugervenlig webapplikation.

Samarbejdet i gruppen har overordnet set været positivt og har givet værdifuld indsigt i projektstyring og den agile udviklingsproces. Udviklingen af Purchase4Less har været en lærerig oplevelse, der ikke kun har styrket vores forståelse af softwareudvikling, men også givet erfaringer, som vil være til stor gavn i fremtidige projekter.

## Fremtidigt arbejde

Fremtidigt arbejde på appen Purchase4Less, ville som start være fokuseret fuld implementering af alle user stories. Derefter kunne der fokusere på flere områder for at forbedre brugeroplevelsen og udvide appens funktionalitet. En vigtig opgave kunne være at implementere en avanceret anbefalingsalgoritme, der baserer sig på brugerens købshistorik og præferencer. Denne algoritme vil kunne tilbyde personaliserede tilbud og rabatter, hvilket øger relevansen af de produkter, der vises for brugeren, og skaber en mere skræddersyet indkøbsoplevelse.

For at forbedre tilgængeligheden på tværs af enheder kunne appen porteres til mobilversioner ved hjælp af React Native. Ved at bruge dette cross-platform framework kan appen tilpasses både Android- og iOS-brugere. Dette ville sikre en mobilvenlig oplevelse og gøre appen tilgængelig for et bredere publikum.

En mulighed for at generere indtægter kunne være at sælge forbrugernes købsdata til butikkerne. Butikker kunne betale for at få adgang til data om kundernes præferencer og købsmønstre, hvilket giver dem mulighed for at målrette deres markedsføring mere præcist. Denne tilgang vil skabe en indtægtskilde, samtidig med at brugerne får en mere skræddersyet shoppingoplevelse, baseret på deres tidligere købsadfærd.

## Litteraturliste

- **C4 Model.** (n.d.). *Software System*. Tilgået den 11. oktober 2024, fra <https://c4model.com/abstractions/software-system>
- **Garfinkel, S. L.** (2002). *Practical Unix & Internet Security*. Tilgået den 8. december 2024, fra <https://simson.net/ref/2002/stanfordPTL.pdf>
- **Aeternity.** (n.d.). *Container Scroll Animation*. Tilgået den 11. december 2024, fra <https://ui.aceternity.com/components/container-scroll-animation>
- **Aeternity.** (n.d.). *Placeholders and Vanish Input*. Tilgået den 12. december 2024, fra <https://ui.aceternity.com/components/placeholders-and-vanish-input>
- **shadcn.** (n.d.). *Toast Component Documentation*. Tilgået den 12. december 2024, fra <https://ui.shadcn.com/docs/components/toast>
- **Nielsen Norman Group.** (n.d.). *Horizontal Attention Leans Left*. Tilgået den 11. september 2024, fra <https://www.nngroup.com/articles/horizontal-attention-leans-left/#:~:text=Summary%3A%20Web%20users%20spend%2080,users'%20efficiency%20and%20company%20profits>
- **Cucumber.** (n.d.). *Gherkin Reference*. Tilgået den 2. december 2024, fra <https://cucumber.io/docs/gherkin/reference/>

## Bilag

[1.1.2] ER-Diagram

[1.1.4.2] C4-model

[2.2.1] Risikoanalyse

[2.2.2] Teknologianalyse

[3] Test

[3.1] Test beskrivelse

[3.2] Ikke-funktionelle krav

[4.1] Accepttest

[5.2.1] WebScraperController

[5.2.2] ProductSeeder

[5.2.3] Seed-Method fra DataBaseController

[5.3.1] TotalPrice - endpoint

[6.1] Adfærdsdrevet udvikling af Jacob Heldgaard

[6.6.2] Zombie Testing

[7.1.1] User Stories

[7.3] UX

[7.4] CRON-job implementation og konklusion