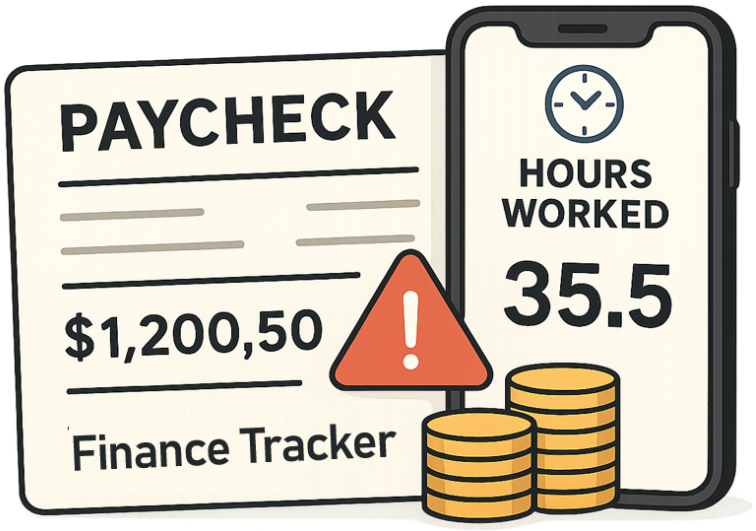
AARHUS UNIVERSITET

Institut for Elektro- og Computerteknologi
Diplomingeniøruddannelsen i softwareteknologi

Semester Projekt 4

Finance Tracker



Gruppemedlemmer	**Studienumre**
Ahmad Chiha	202305734
Ali Najafi	202307397
André Pelle Rashid	202306343
Christina Lavdal Braüner	202306583
Jahye Ali	202309135
John Nguyen	202209849
Khaled Rami Omar	202307853
Khizer Khan	201710674
Victoria Franca Edwards	202308509

SW-PRJ 4	
Gruppe nr:	11
Vejleder:	Jung Min Kim
Udfærdiget:	forårssemesteret 2025
Projekt periode	31/01/2025 - 30/05/2025
Afleverings dato:	30/05-2025 kl. 12.00
Antal tegn:	71997

Abstract

Finance Tracker is a digital solution designed to help students and young workers detect errors in their wages by tracking work hours, calculating expected pay, and comparing those calculations to actual payslips. The system consists of a React-based web application, a cross-platform mobile app built with .NET MAUI, and a shared ASP.NET Core backend with a Microsoft SQL Server database and a REST API secured by JWT authentication. Development followed an iterative, agile-inspired process focusing on modular architecture, cross-platform usability, and user-friendly design. The final implementation is delivered as a minimum viable product (MVP) that fulfills most of the specified requirements (implementing eight out of ten functional requirements and eight out of ten non-functional requirements), demonstrating solid functionality within the project's scope. Throughout the project, the team combined knowledge from multiple software engineering domains and utilized modern development tools. The resulting solution effectively addresses the problem of undetected wage discrepancies and aims to increase financial transparency and independence among the target user group.

Resumé

Finance Tracker er en digital løsning, der hjælper studerende og unge lønmodtagere med at opdage fejl i deres løn ved at registrere arbejdstimer, beregne forventet løn og sammenligne denne med faktiske lønsedler. Systemet består af en React-baseret webapplikation, en platformuafhængig mobilapplikation bygget med .NET MAUI, samt en fælles ASP.NET Core-backend med en Microsoft SQL Server-database og et REST API sikret med JWT-baseret autentifikation. Udviklingen fulgte en iterativ, agil-inspireret proces med fokus på modulær arkitektur, platformsuafhængighed og brugervenlige grænseflader. Den endelige implementering er leveret som et Minimum Viable Product (MVP), der opfylder størstedelen af de specificerede krav (implementerer otte ud af ti funktionelle krav og otte ud af ti ikke-funktionelle krav) og demonstrerer solid funktionalitet inden for projektets rammer. Gennem projektforløbet har gruppen kombineret viden fra flere softwarefaglige områder og anvendt moderne udviklingsværktøjer. Den resulterende løsning adresserer effektivt problemet med uopdagede fejl i lønudbetalinger og har til formål at øge økonomisk gennemsigtighed og uafhængighed hos målgruppen.

Forord

Denne rapport er resultatet af semesterprojekt på 4. semester af softwareteknologi ved Aarhus Universitet. Projektet er gennemført i perioden februar til maj 2025, og er udført i samarbejde af medlemmerne af gruppe 11.

Liste over forkortelser

2FA	Two-Factor Authentication
DAL	Data Acces Layer
EFC	Entity Framework Core
GDPR	General Data Protection Regulation
MVP	Minimum viable product
US	User Story

Indholdsfortegnelse

Indledning 1

Mange unge og studerende arbejder i timelønnet deltidsjob, hvor lønnen ofte afhænger af varierende arbejdstimer og forskellige tillæg. Desværre forekommer der hyppigt fejl i lønudbetalinger, hvilket kan skyldes administrative svigt, uklarheder i overenskomster eller manglende kendskab til lønregler. Ifølge Fagbevægelsens Hovedorganisation har hver anden lønmodtager i Danmark oplevet fejl på lønsedlen[1].

Fejl i lønudbetalinger kan medføre både økonomisk usikkerhed og stress, især blandt unge studerende, der ofte har begrænsede økonomiske ressourcer. Forskning viser, at økonomiske vanskeligheder ikke kun medfører mental stress, men også kan have alvorlige konsekvenser for fysisk sundhed[2].

Selvom der findes et tilgængeligt værktøj i form af Skats lønberegner, som kan anvendes til at beregne den forventede nettoløn baseret på månedsløn, mangler dette værktøj væsentlige aspekter. Det tilbyder ikke mulighed for timebaseret opfølgning og sammenligning mellem arbejdstimer og den faktiske lønudbetaling.[3]

1.1 Problemformulering

Unge og studerende oplever ofte fejl i deres lønudbetalinger, hvilket kan skyldes mangelfuld registrering af arbejdstimer, uklare lønforhold eller begrænset indsigt i, hvordan løn beregnes. Samtidig er lønsedler ofte komplekse og svære at gennemskue, hvilket gør det vanskeligt at opdage fejl og få et retvisende billede af sin økonomi.

På den baggrund undersøger projektet følgende:

- Hvordan kan en løsning understøtte brugere i at registrere arbejdstimer og tillæg med henblik på lønberegning?
- Hvordan kan brugeren gøres opmærksom på afvigelser mellem forventet og faktisk udbetalt løn?
- Hvordan kan løsningen bidrage til, at brugeren får bedre indsigt i sin samlede

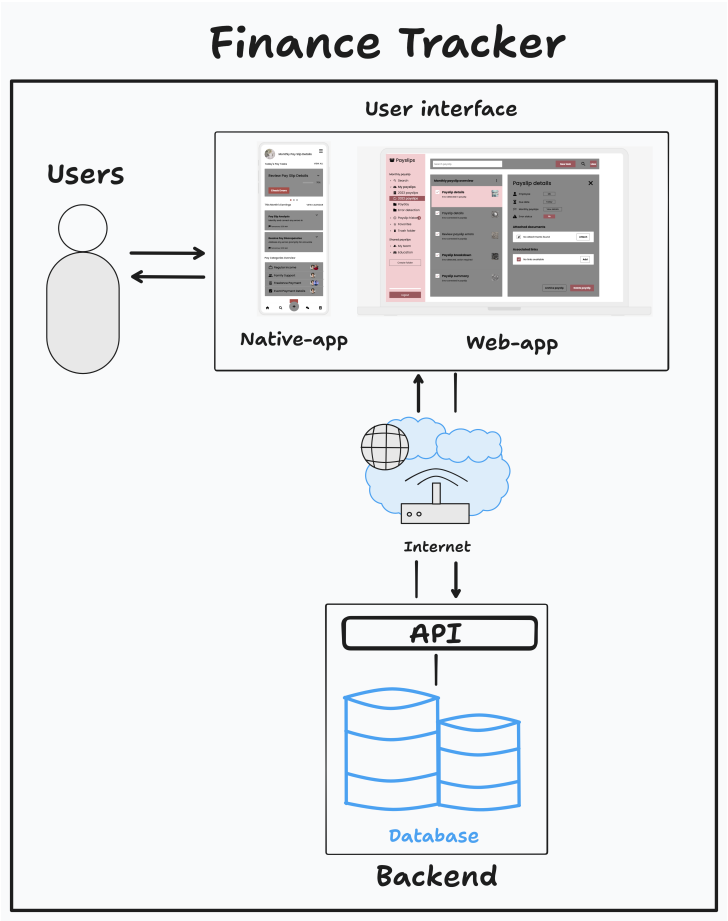
indkomst og overblik over relevante økonomiske grænser?

1.2 Systembeskrivelse

Finance Tracker er udviklet gennem en iterativ proces og består af:

- **Frontend**: En brugergrænseflade, hvor brugeren kan registrere arbejdstimer, se lønberegninger og sammenligne disse med egne lønsedler.
- **Backend**: Ansvarlig for håndtering af brugerdata, logik og kommunikation med databasen.
- **Database**: Varetager lagring og vedligeholdelse af applikationens data.

1.2.1 Rigt billede



Figur 1.1. Skitse af systemet Finance Tracker

Figur 1.1 viser en skitse for Finance Tracker systemet. Brugere interagerer med enten en native-app eller en web-app. Appen kommunikerer med backend via internettet.

Krav og prioriteringer 2

I dette kapitel præsenteres de krav, der stilles til løsningen, og hvordan de prioriteres.

2.1 User Stories (Funktionelle Krav)

User stories (US) beskriver brugerens behov samt ønsker til systemet. Hver US indeholder en kortfattet beskrivelse af, hvem der udfører en handling, hvad de vil opnå, og hvorfor det er vigtigt. Formålet er at gøre kravene mere forståelige og sikre, at løsningen skaber værdi for brugeren.

US1. Lønberegning

Som bruger
ønsker jeg at se en lønberegning for en lønperiode
så det kan sammenlignes med min aktuelle lønseddel.

US2. Lønsammenligning

Som bruger
ønsker jeg at systemet sammenligner min lønseddel med lønberegningen,
så jeg bliver opmærksom på afvigelser.

US3. Tillægstider
Som bruger
ønsker jeg at registrere tillæg for forskellige tidspunkter og ugedage,
så applikationen kan medtage dem i lønberegningen.

US4. Platformskompatibilitet

Som bruger
ønsker jeg at tilgå applikationen via en web app og en native app,
så jeg kan bruge appen i webbrowser eller som en lokal app.

US5. Oversigt over årsindkomst og SU-fribeløb

Som bruger

ønsker jeg at se en oversigt over min indkomst for året og resterende SU-fribeløb,

så jeg kan holde overblik over min økonomi.

US6. Advarsel ved overskridelse af SU-fribeløb

Som bruger

ønsker jeg at modtage en advarsel, hvis jeg nærmer mig mit SU-fribeløb,

så jeg kan reducere mine arbejdstimer og undgå tilbagebetaling af SU.

US7. Brugerprofil

Som bruger

ønsker jeg at oprette en brugerprofil

så jeg kan gemme mine personlige oplysninger.

US8. Registrering af arbejdsvagter

Som bruger

ønsker jeg at registrere mine arbejdsvagter

så applikationen kan lave en lønberegning.

US9. Eksport af lønberegning til PDF

Som bruger

ønsker jeg at kunne eksportere min lønberegning til en PDF-fil,

så jeg kan gemme den offline.

US10. Feriepengeberegning

Som bruger

ønsker jeg at kunne se mine optjente feriepenge for et år

så jeg kan få overblik over mine feriepenge.

2.2 Ikke-funktionelle krav

De ikke-funktionelle krav angiver rammerne for systemets kvalitet og drift.

1. Responstiden for en formularafsendelse må ikke overstige **1 sekund**[4].
2. Generering af en månedlig lønberegning må ikke tage mere end **1 sekund**[4].
3. REST API'et skal kunne håndtere op til **5000 samtidige requests pr. minut** uden at returnere HTTP 5xx-fejl[5].
4. Alle softwaremoduler skal have mindst **80%** code coverage[6].
5. Webapplikationen skal opnå en sikkerhedsscore på medium eller bedre ifølge ZAP-skalaen[7].
6. Applikationen skal understøtte både **dansk og engelsk**[8].
7. Farverne i brugergrænsefladen skal sikre et kontrastforhold på minimum **4.5:1** mellem tekst og baggrund[9].
8. Automatisk backup af brugerdata skal ske mindst hver **24. time**[10].
9. Systemet skal overholde **GDPR og nationale databeskyttelseslove**[11].
10. Systemet skal understøtte automatiserede **softwareopdateringer uden nedetid**[12].

2.3 Priotering af funktionalitet

Dette afsnit giver et overblik over, hvilke funktioner der er vigtigst for systemets kernefunktionalitet, og hvilke der kan udskydes til fremtidige versioner.

2.3.1 MoSCoW for user stories

Nedenstående tabel 2.1 viser, hvordan US er blevet prioriteret.

Tabel 2.1. Prioritering af krav baseret på MoSCoW-metoden

Prioritet	**Krav**
Must have	**US1**, **US2**, **US3**, **US4**, **US7**, **US8**
Should have	**US9**, **US10**
Could have	**US5**
Won't have	**US6**

2.3.2 FURPS+ tabel med MoSCoW-prioritering

Denne tabel opsummerer de ikke-funktionelle krav i systemet, prioriteret efter MoSCoW-metoden, med fokus på FURPS+ kriterierne.

Tabel 2.2. FURPS+ oversigt for Finance Tracker med MoSCoW-prioritering

Kategori	**Ikke-funktionelle krav**	**Prioritet**
Functionality	**IFK6**	**Should**
Usability	**IFK7**	**Should**
Reliability	**IFK4**	**Could**
	IFK5	**Must**
	IFK8	**Should**
Performance	**IFK1**	**Must**
	IFK2	**Must**
	IFK3	**Must**
Supportability	**IFK10**	**Could**
	IFK9	**Would not**

2.4 Afgrænsning

I udviklingen af Finance Tracker er der foretaget afgrænsninger for at sikre, at projektet forblev fokuseret og gennemført inden for de givne rammer. Prioriteringsmetoden MoSCoW har været anvendt til at identificere de funktionaliteter og kvalitetskrav, der var nødvendige for at realisere et brugbart MVP. MVP'en består af alle **MUST** krav.

Metoder og arbejdsproces 3

Dette kapitel beskriver de metoder og værktøjer, som gruppen har anvendt i forbindelse med udviklingen af systemet. Projektet er gennemført efter agile principper med inspiration fra SCRUM. Der er lagt vægt på iterativ udvikling, løbende evaluering og tydelig arbejdsfordeling. Kapitlet omfatter overvejelser om projektstyring, udviklingstilgang og anvendelse af planlægningsværktøjer. For yderligere dokumentation om processen henvises til Bilag 1.

3.1 Project management

Til projektets udførelse er SCRUM anvendt. Gruppen har organiseret arbejdet i sprints, hvor fremdrift, forhindringer og opgaver blev drøftet.

Gruppen har benyttet SCRUM, dog uden formel udpegning af SCRUM Master eller Product Owner. Ansvar for planlægning, retrospectives og opfølgning blev fordelt mellem medlemmerne. Rollerne som mødeleder- og referent gik på skift, hvilket styrkede erfaring med facilitering og bidrog til en flad struktur. Beslutningen om ikke at have en fast leder gav plads til initiativ og fælles ansvar, men skabte også udfordringer i pressede faser, hvor koordinering krævede mere struktur.

3.2 Agil udvikling

Udviklingsarbejdet har fulgt en agil og iterativ tilgang inspireret af SCRUM og ASE-modellen. Projektet blev opdelt i sprints med planlægning, udførelse og evaluering, og hvor der løbende blev designet, implementeret og testet. Denne tilgang har gjort det muligt at reagere hurtigt på ændringer i krav og prioriteringer.

3.3 Brug af tidsplaner, milepæle mm.

Gruppen anvendte Monday.com[13] til at planlægge og styre opgaver, sprints og deadlines. Taskboards og Gantt-diagrammer blev brugt til at skabe overblik og prioritere arbejde, mens opgaver blev opdateret og flyttet mellem kolonner alt efter status. Planen var fleksibel og blev tilpasset undervejs. Versionsstyring blev håndteret med Git og GitHub, hvor der blev arbejdet i branches og benyttet pull requests for at sikre kvalitet og gennemsigtighed.

Dokumentation og koordinering foregik via SharePoint og Overleaf, hvilket muliggjorde fælles redigering og struktureret lagring. Diagrammer blev lavet i Draw.io og PlantUML, gemt på Google Drev, og var tilgængelige for hele gruppen. Den daglige kommunikation skete primært via Messenger, hvilket understøttede hurtig afklaring og koordination. Samlet har værktøjerne understøttet transparens, fleksibilitet og kontinuerlig fremdrift gennem hele projektperioden.

Teknisk analyse 4

For at sikre, at de mest egnede teknologier anvendes i projektet, er der gennemført en omfattende analyse af de relevante muligheder for følgende: Database, Backend, Native App og Web App.

Formålet er at skabe et overblik over tilgængelige teknologier og danne beslutningsgrundlag for den endelige teknologistak. Overblikket opnås ved at opstille en liste for hver komponent, hvor forskellige teknologier sammenlignes på baggrund af gruppens kompetencer, og teknologiernes fordele og ulemper eller eventuelle begrænsninger.

Der henvises til bilag 4 for en mere detaljeret analyse.

4.1 Database

Tre databaser er analyseret: MySQL, MongoDB, Microsoft SQL Server.

4.1.1 Microsoft SQL Server

Microsoft SQL Server er en relationsdatabase[14].

Fordele

+ Understøtter ACID-transaktioner.
+ Integreres let med Entity Framework Core.
+ Gruppen har eksisterende erfaring med MSSQL.

Ulemper

– Højere ressourceforbrug end letvægtsdatabaser.
– Mindre fleksibelt uden for Microsoft.

4.1.2 MongoDB

MongoDB er en dokumentorienteret NoSQL-database[15].

Fordele

+ Fleksibel datamodel uden fast skema.
+ God horisontal skalerbarhed.
+ Velegnet til ustrukturerede eller varierende datatyper.

Ulemper

– Ikke fuldt ACID-kompatibel i alle scenarier.
– Kompleks håndtering af relationelle strukturer.

4.1.3 MySQL

MySQL er et open-source relationsdatabassystem[16].

Fordele

+ ACID kompatibel.
+ Understøtter integration med Entity Framework Core.

Ulemper

– Mindre egnet til dynamiske og hyppigt ændrede skemastrukturer.
– Begrænset funktionalitet ved komplekse datamodeller (f.eks. mange-til-mange med metadata).
– Mindre fleksibel ved integration med ikke-relationsbaserede komponenter (f.eks. dokument-API'er).

Microsoft SQL Server blev valgt som database, da den understøtter Entity Framework Core og er gennemgået i undervisningen, hvilket reducerer læringskurven og understøtter hurtigere udvikling.

4.2 Backend

Tre teknologier er analyseret og sammenlignet: ASP.NET Core, Node.js og Spring Boot.

4.2.1 ASP.NET Core

ASP.NET Core anvendes til udvikling af webapplikationer, API'er og bagvedliggende services[17].

Fordele

+ Understøtter moderne arkitektur via NuGet-pakker til autentificering og datatilgang.
+ Stærk typesikkerhed gennem C#.
+ Projektgruppen har erfaring med og modtager undervisning i teknologien.
+ Indbygget understøttelse af Dependency Injection (DI).

Ulemper

– Begrænset kompatibilitet med ældre .NET Framework-biblioteker.

4.2.2 Node.JS

Node.js er en serverside miljø baseret på JavaScript og drives af Googles V8-motor[18].

Fordele

+ Kører på Googles V8 JavaScript-motor.
+ Velegnet til håndtering af mange samtidige klienter.
+ Høj ydeevne og asynkron eventdrevet arkitektur.

Ulemper

– Begrænset erfaring.

4.2.3 Spring Boot

Spring Boot er et modulært og konfigurationslet Java-baseret framework[19].

Fordele

+ Understøtter DI.
+ Spring Security gør det muligt at autentificering og autorisation med minimal opsætning.

Ulemper

- Projektgruppen har ingen erfaring med Java.

ASP.NET Core blev valgt som backend-teknologi, da gruppen har erfaring med både C# og ASP.NET gennem undervisning.

4.3 Frontend – Native applikation

Tre frameworks er analyseret og sammenlignet: .NET MAUI, Flutter og React Native.

4.3.1 .NET MAUI

.NET MAUI (Multi-platform App UI) er et cross-platform framework[20].

Fordele

+ Projektgruppen har eksisterende erfaring og modtager undervisning i teknologien.
+ Integration med Visual Studio og øvrige .NET-værktøjer.

Ulemper

- Længere build-tider.

4.3.2 Flutter

Flutter er et open-source UI-framework[21].

Fordele

+ Cross platform udvikling.

Ulemper

- Flutter anvender Dart. Et sprog gruppen ikke har erfaring med.

4.3.3 React Native

React Native gør det muligt at udvikle mobile applikationer med JavaScript/TypeScript[22].

Fordele

+ Kan potentielt dele komponenter og forretningslogik med den webbaserede frontend.

Ulemper

- Debugging og vedligeholdelse kan være mere kompleks som følge af lagdeling og tredjepartsafhængigheder.

.NET MAUI er valgt på grund af undervisningen på 4. semester. Derudover kan den integreres med Visual Studio og øvrige .NET-værktøjer.

4.4 Frontend - Webapplikation

Tre frontend-teknologier er blevet analyseret og sammenlignet: AngularJS, ReactJS og VueJS.

4.4.1 ReactJS

ReactJS er et komponentbaseret JavaScript-bibliotek[23].

Fordele

+ Understøtter genanvendelige komponenter og modulær opbygning.
+ Virtuel DOM sikrer hurtige brugerinteraktioner.
+ Projektgruppen har eksisterende erfaring og modtager undervisning i teknologien.

Ulemper

- Kompleks tilstandshåndtering og routing.

4.4.2 AngularJS

AngularJS er et TypeScript-baseret frontend-framework[24].

Fordele

+ Understøtter DI.
+ Mange funktioner og moduler er indbygget, hvilket reducerer afhængigheden af eksterne biblioteker.

Ulemper

- Gruppen har ingen praktisk erfaring med AngularJS.

4.4.3 VueJS

VueJS er et JavaScript-framework[25].

Fordele

+ Enkel syntaks og lav indlæringskurve.
+ Reaktiv datamodel muliggør effektiv UI-opdatering.
+ Lille filstørrelse og høj ydeevne.
+ Understøttelse af state management via Vuex.

Ulemper

– Færre tredjepartsmoduler og mindre community end React.
– Gruppen mangler erfaring.

ReactJS blev valgt som frontend-teknologi, da gruppen har erfaring med det, og det samtidig undervises i. Reacts komponentbaserede struktur og virtuelle DOM gør det velegnet til udvikling af en brugervenlig og responsiv løsning.

4.5 Programmeringssprog

I dette afsnit analyseres programeringssprog for de valgte frameworks i afsnit 4.

4.5.1 Backend

Backenden Bruger ASP.NET Core og derfor benyttes der også C# som det oplagte valg[26].

4.5.2 Frontend – Native applikation

Den native app udvikles i .NET MAUI med C# og Xaml. Dette sprog er oplagt til MAUI udvikling. Derfor er der ikke analyseret alternativer.

4.5.3 Frontend – Webapplikation

Webfrontend udvikles i ReactJS med TypeScript, et statisk typet supersæt af JavaScript.

4.5.3.1 JavaScript

JavaScript er et dynamisk, fortolket sprog, der kører direkte i browseren[27].

Fordele

+ Hurtig iteration uden kompilering.
+ Velegnet til dynamiske brugergrænseflader og realtidsinteraktioner.
+ Understøtter funktionel programmering og modulær opbygning.

Ulemper

– Mangler statisk typesystem, hvilket kan føre til runtime-fejl.
– Tillader løs kodeorganisation, hvilket kan give vedligeholdelsesudfordringer.

4.5.3.2 TypeScript

TypeScript er et statisk typet supersæt af JavaScript og skal transpileres til JavaScript før eksekvering[28].

Fordele

+ Tilføjer statisk typesikkerhed.
+ Giver bedre struktur via klasser, interfaces og namespaces.
+ Understøtter moderne objektorienteret udvikling.

Ulemper

– Kræver opsætning og build-pipeline.
– Udviklere skal forstå både JavaScript og TypeScript for at arbejde effektivt.

TypeScript vælges for at reducere fejl. Tailwind CSS anvendes til layout for en responsiv brugergrænseflade.

4.6 Integration mellem Server og frontends

Valget af kommunikationsprotokol mellem backend og frontent påvirker direkte, hvordan data udveksles og opdateres mellem klient og server. I dette afsnit er to kommunikationsteknologier analyseret : REST API og GraphQL.

4.6.1 REST API

REST (Representational State Transfer) er en udbredt arkitekturstil, der benytter HTTP-metoder til at kommunikere med serveren og returnere data i formater som JSON eller XML[29].

Fordele

+ Enkel at implementere og vedligeholde.
+ Stateless – hver anmodning behandles uafhængigt

Ulemper

– Fast struktur – klienten modtager ofte mere data end nødvendigt.

4.6.2 GraphQL

GraphQL er et query-baseret API, hvor klienten selv definerer hvilke data den ønsker[30].

Fordele

+ Klienten kan hente præcist de nødvendige data og undgår dermed over-fetching.
+ Understøtter komplekse og hierarkiske forespørgsler i én anmodning, hvilket reducerer antallet af API-kald.

Ulemper

– Har en stejlere læringskurve sammenlignet med REST.
– Kræver avanceret serveropsætning med hensyn til autorisation, caching og fejlbehandling.

REST API er valgt som integrationsteknologi, da det egner sig godt til strukturerede dataoperationer og klassiske CRUD-funktioner. Det er enkelt at implementere, bredt understøttet.

4.7 Object-Relational Mapping (ORM)

To ORM-løsninger er analyseret: Entity Framework Core (EF Core) og Dapper.

4.7.1 Entity Framework Core

EF Core er en ORM til .NET-platformen[31]. Det gør det muligt at interagere med databasen gennem C#-objekter frem for manuelle SQL-forespørgsler.

Fordele

+ Understøtter database-migreringer til versionering og strukturændringer.
+ Understøtter autentificering ved hjælp af identity entity framework core.
+ Eliminerer behovet for at skrive SQL.
+ Understøtter LINQ, hvilket giver en ensartet måde at forespørge data på.

Ulemper

– Introducerer performance-overhead grundet ekstra abstraktionslag.

4.7.2 Dapper

Dapper er et mikro-ORM udviklet af Stack Overflow, som fungerer som en tynd wrapper omkring SQL og ADO.NET[32].

Fordele

+ Ekstremt hurtig udførelse og lavt hukommelsesforbrug.
+ Giver fuld kontrol over SQL, hvilket gør det muligt at optimerede og præcise forespørgsler.
+ Minimal overhead gør det velegnet til performancekritiske scenarier.

Ulemper

– Kræver manuel vedligeholdelse af SQL, hvilket kan øge arbejdsbyrden og fejlrisikoen.
– Mangler indbygget objekt-tracking og lazy-loading.
– Komplekse relationer skal håndteres manuelt, hvilket øger kodekompleksitet.

Entity Framework Core vælges som standard ORM-løsning. EF Core forenkler arbejdet med datamodeller og reducerer mængden af redundent kode.

4.8 Opsummering af teknologivalg

Den valgte teknologistak er baseret på analysen af systemets krav.

Valgte teknologier:

- **Backend:** ASP.NET Core
- **Database:** Microsoft SQL Server
- **Frontend Web App:** React
- **Frontend Native App**: .NET MAUI
- **Programmeringssprog:** C#, Xaml, TypeScript
- **API-integration:** REST API
- **Databaseintegration:** Entity Framework Core

Arkitektur 5

Systemarkitekturen udgør den overordnede struktur for applikationen og beskriver samspillet mellem dens centrale komponenter. Afsnittet har til formål at give et teknisk overblik over systemets opbygning, der ligger til grund for design.

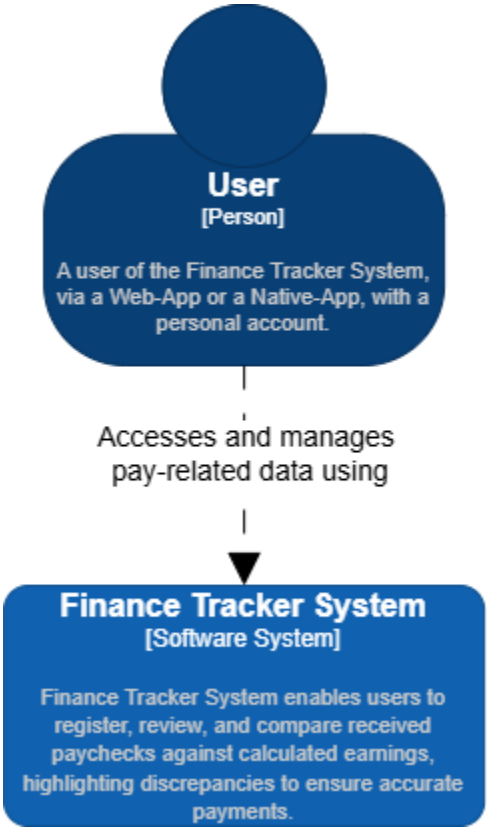
5.1 C4-Model: Systemarkitektur

For at kommunikere og dokumentere arkitekturen for Finance Tracker benyttes C4-modellen[33]. Modellen anvendes til at skabe et fælles overblik og sikre konsistent dokumentation. C4-modellen opdeler systemets arkitektur i følgende niveauer:

- **Level 1 – System Context Diagram:** Giver en overordnet beskrivelse af systemet og dets interaktion med brugere og eksterne systemer.
- **Level 2 – Container Diagram:** Viser systemets opdeling i logiske dele såsom frontend, backend og database.
- **Level 3 – Component Diagram:** Går i dybden med de interne komponenter i hver container (backend, web-app og native app).
- **Level 4 – Code Diagram:** Anvendes ikke i dette projekt.

Level 4 går i detaljer med kode, hvilket ligger uden for formålet med dette afsnit og projektets arkitekturbeskrivelse, og er derfor udeladt.

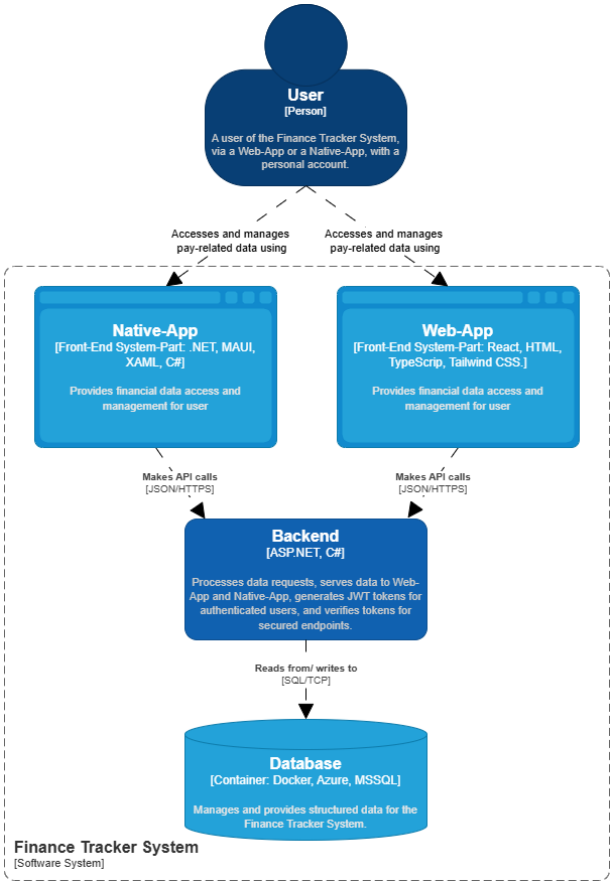
5.2 Level 1 - System Context diagram



Figur 5.1. System Context diagram for systemet

Figur 5.1 illustrerer systemets overordnede kontekst med fokus på aktøren og systemet. Formålet med dette niveau er at give et klart og præcist overblik over systemets placering i det omkringliggende miljø samt dets vigtigste kommunikationspartnere. Systemet har 1 aktør og interagerer ikke med eksterne systemer.

5.3 Level 2 - Container diagram



Figur 5.2. Container diagram: Finance Tracker system

Figur 5.2 viser container diagram - her ses systemets container og interaktionen imellem dem.

Systemet er opdelt i 4 hovedcontainere:

- **Web-App (ReactJS)**: Brugeren kan interagere med systemet via en browserbaseret løsning.
- **Native-App (.NET MAUI)**: En mobilapplikation med lignende funktionalitet som web-appen.
- **Backend (ASP.NET Core)**: Håndterer forretningslogik, databehandling og sikkerhed.
- **Database (MSSQL)**: Persistere brugerdata.

Da der udvikles både en native app og en webapp, vil en stor del af forretningslogikken være ens for begge applikationer. Ved at have en fælles backend kan forretningslogikken

defineres ét sted og deles af begge applikationer, hvilket betyder, at den samme logik ikke skal implementeres flere gange. Dermed overholdes **DRY-princippet** (Don't Repeat Yourself).

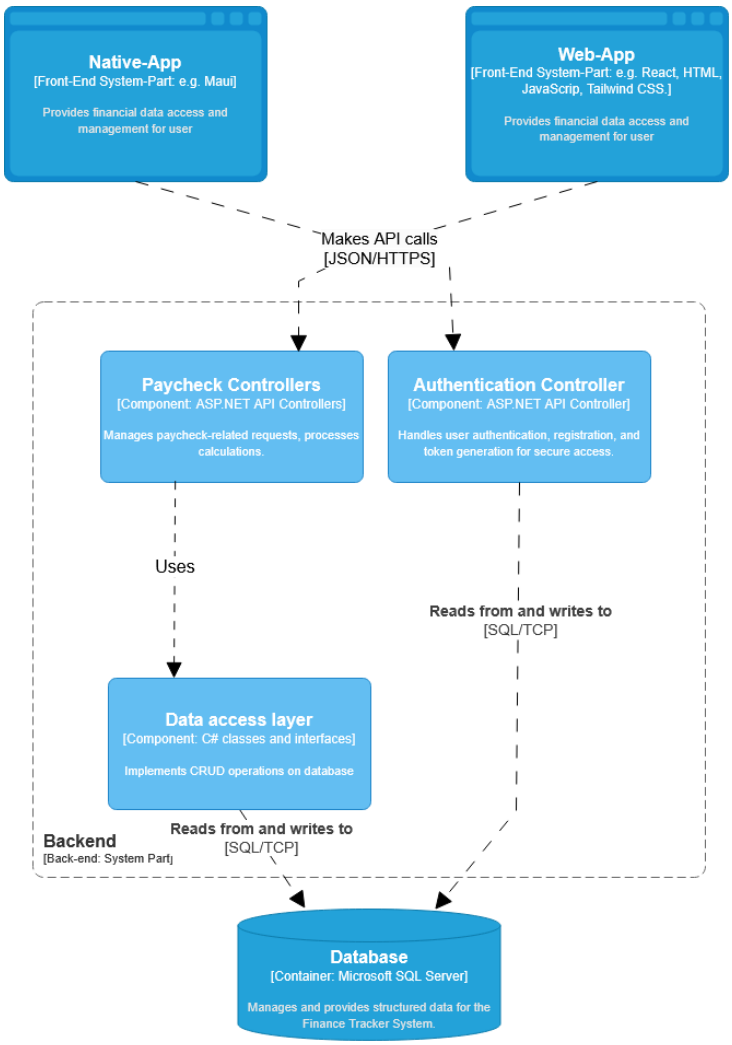
Denne arkitektur følger 3-tier stilen, som består af følgende lag:

Presentation tier (web- og native app)
Business logic tier (backend)
Database tier

Ved deployment placeres hver tier på sin egen fysiske maskine, hvilket gør det muligt at, at hver tier kan skaleres uafhængigt af de andre. Dette øger fleksibiliteten og gør det lettere at tilpasse ressourcer efter behov.

5.4 Level 3 - Component diagram

5.4.1 Backend component diagram



Figur 5.3. Component diagram for Backend

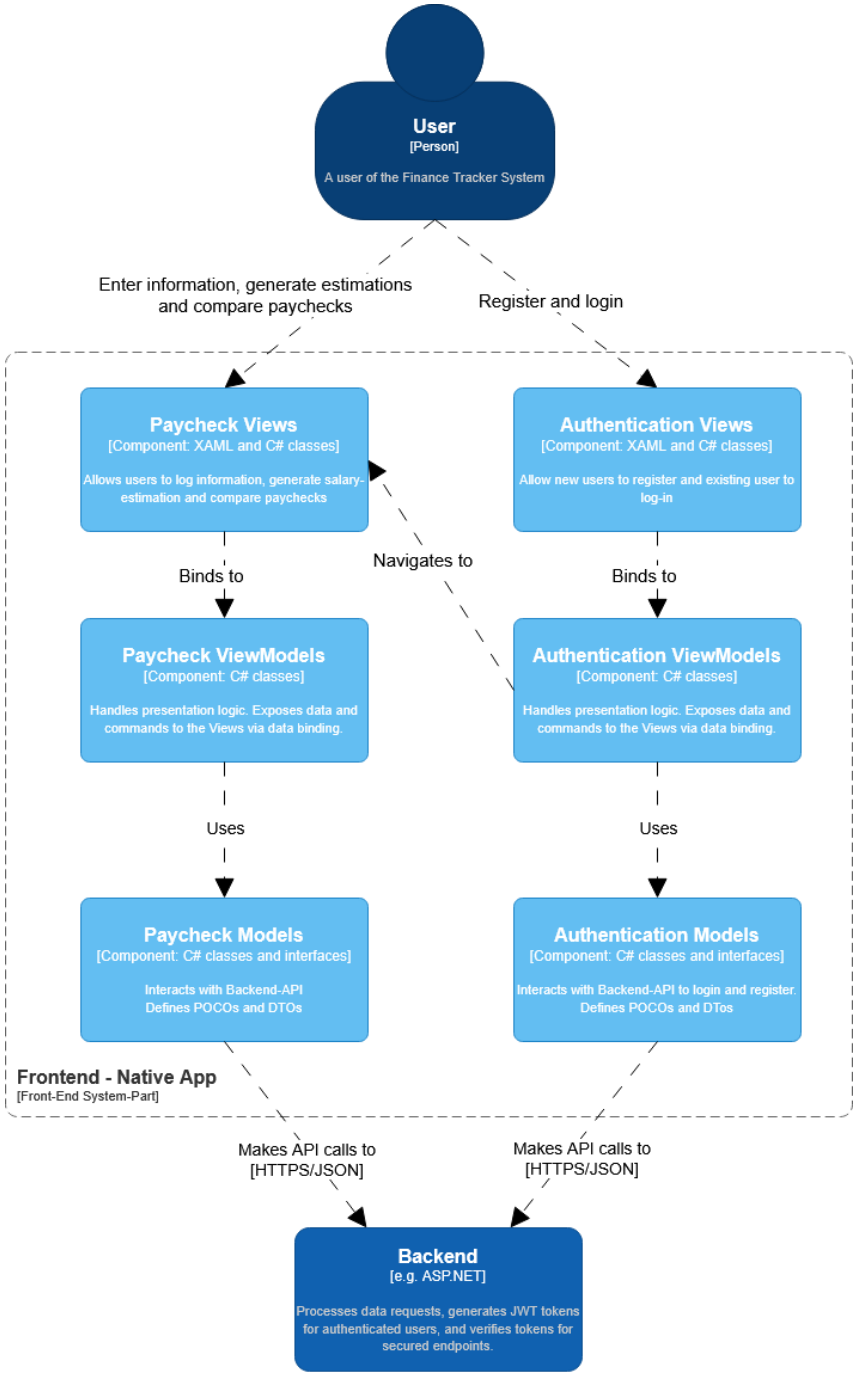
Figur 5.3 viser komponenter i Backend. **Authentication Controller** anvender ikke *Data Access Layer*, da registrering og login kræver brugen af *ASP.NET Core Identity*-komponent, som er designet til at håndtere brugerdata og sikkerhedsrelaterede operationer direkte mod databasen, og det er hverken muligt eller hensigtsmæssigt at omgå dem via det almindelige *Data Access Layer*.

Backend består af følgende komponenter:

- **Authentication Controller**: Modtager login requests og brugerregistrerings requests.

- **Paycheck Controllers**: Modtager autentificerede requests om data.
- **Data Access Layer**: Håndterer kommunikation med databasen og abstraherer databaseoperationer for resten af applikationen.

5.4.2 Native-App component diagram

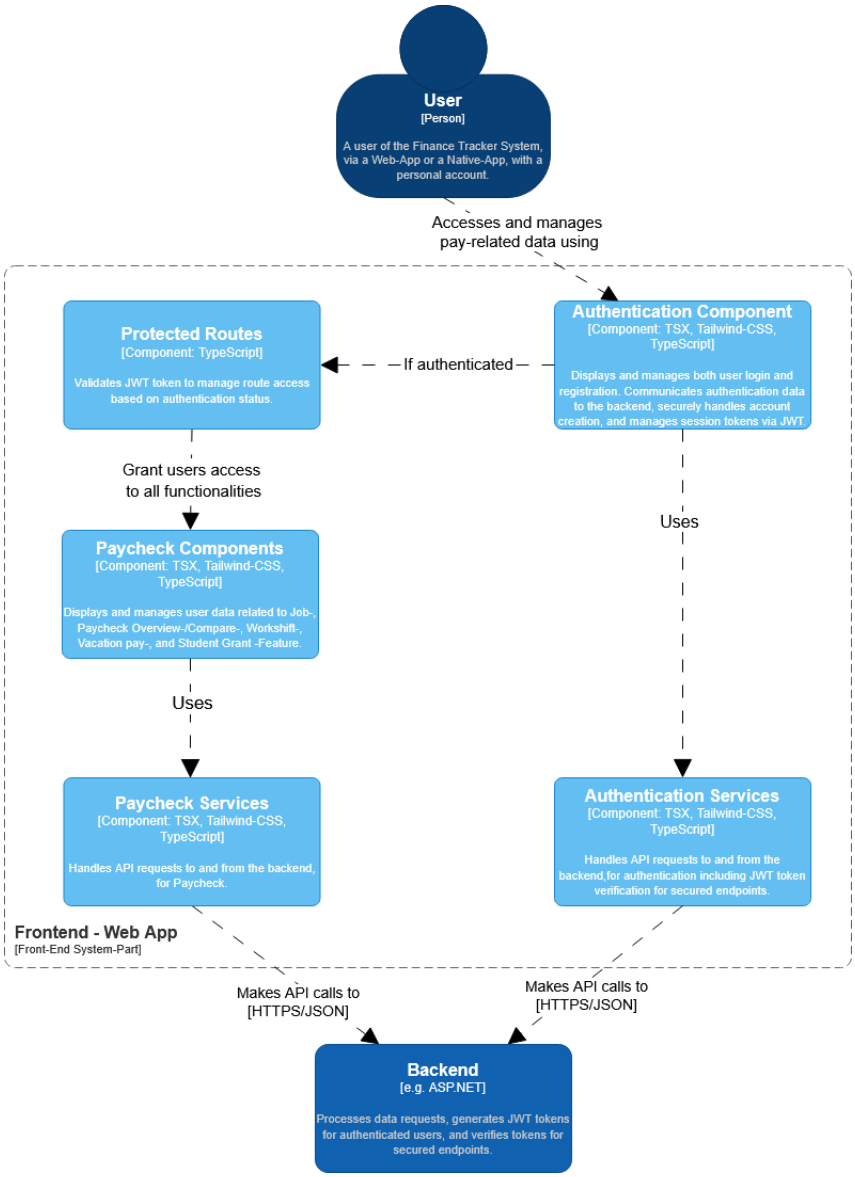


Figur 5.4. Component diagram for Native app

Native-applikationen følger MVVM arkitektur og består af følgende komponent-typer:

- **Views**: Definerer brugergrænsefladen.
- **ViewModels**: Indeholder præsentationslogik, håndterer brugerinteraktioner.
- **Models**: Interagerer med backend-API og definerer POCOs og DTOs.

5.4.3 Web-App component diagram



Figur 5.5. Component diagram for Web App

Web-applikationen består af tre primære komponenttyper:

- **Components**: Komponenter udgør brugergrænsefladen og præsenterer data for brugeren.

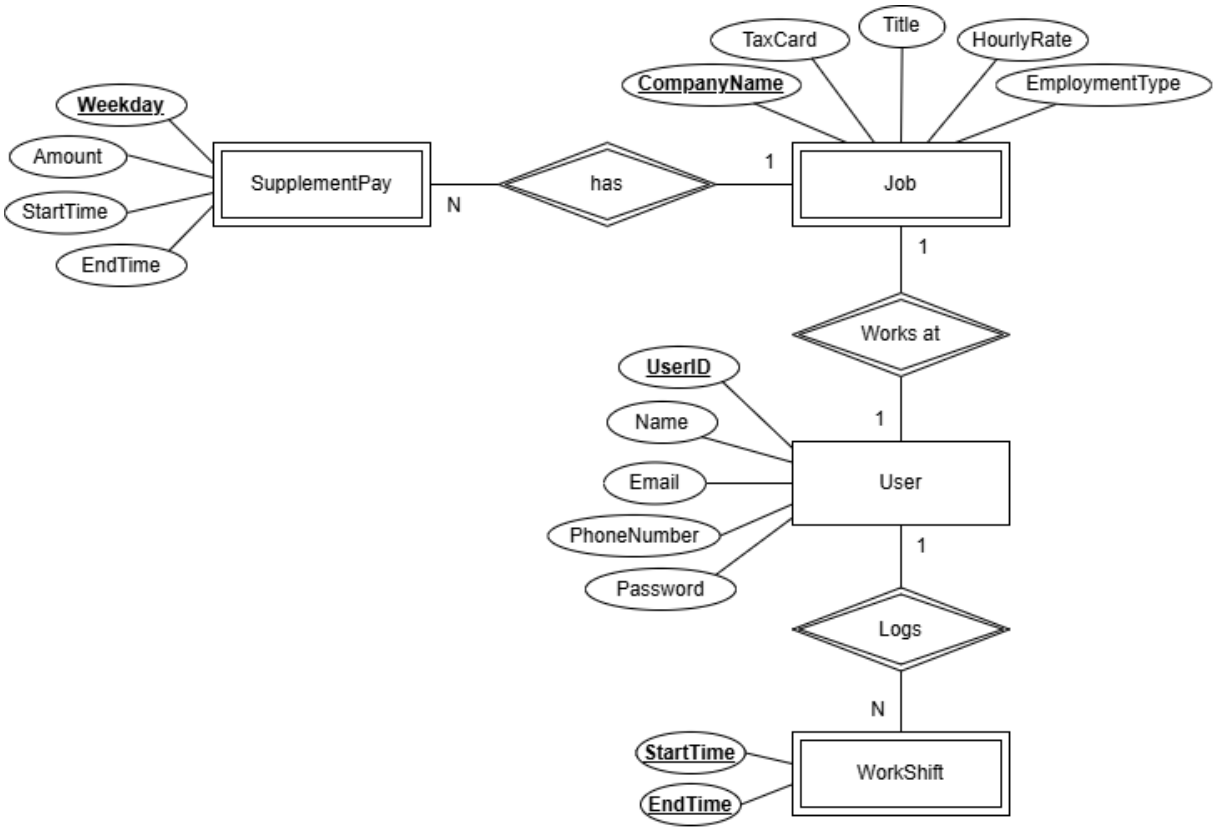
- **Protected Routes**: Begrænser adgang til sikre ruter baseret på autentificeringsstatus.
- **Services**: Håndterer REST-kommunikation med backend og kalder relevante endpoints.

Design 6

Dette afsnit beskriver det overordnede softwaredesign af systemet med fokus på at give en forståelse af, hvordan centrale komponenter fungerer og interagerer, baseret på systemets arkitektur. Designet er opdelt i fire hovedområder: Database, Backend, Native-app og Web-app.

6.1 Design af database

Dette afsnit beskriver datamodellen for Finance Tracker.



Figur 6.1. ER-diagram

Figur 6.1 viser ER-diagrammet, som bl.a. illustrerer systemets entiteter samt deres indbyrdes relationer. Diagrammet er udarbejdet ved brug af Chen-notation[34]. De følgende underafsnit giver en detaljeret beskrivelse af hver enkelt entitet.

Tabel 6.1. Oversigt over entiteter og primærnøgler

Entitet	**Primærnøgle (PK)**
User	UserID
Job	(CompanyName, UserID)
SupplementPay	(Weekday, CompanyName)
WorkShift	(UserID, StartTime, EndTime)

Tabel 6.1 viser primærnøglerne for hver entitet. Flere entiteter har sammensatte (komposite) primærnøgler, hvilket skyldes, at de er weak entities, der er afhængige af relationer til andre entiteter. I designet af databasen anvendes weak entities og composite keys for at sikre korrekt struktur og dataintegritet. Weak entities muliggør en naturlig modellering af afhængige data, der ikke kan eksistere uden tilknytning til en overordnet entitet, hvilket øger datakonsistensen.

6.1.1 User

Entiteten `User` er central i databasen og indeholder attributter relateret til brugerens login og profiloplysninger.

6.1.2 Job

Job-entiteten er en "weak entity", hvilket betyder, at den kræver en tilknytning til en `User` for at kunne eksistere. Den repræsenterer en brugers job og indeholder blandt andet oplysninger som brugerens timeløn.

6.1.3 SupplementPay

Entiteten `SupplementPay` kan indeholde op til syv poster per bruger, svarende til én per ugedag. Den lagrer tidsintervaller for tillæg på specifikke dage, eksempelvis lørdag kl. 15:00–22:00 med et tillæg på 30 kr. pr. time. `SupplementPay` er en weak entitet og kræver derfor tilknytning til et `Job`.

6.1.4 WorkShift

Entiteten `WorkShift` registrerer alle arbejdsvagter tilknyttet en bruger. Det er en "weak entity", da hver arbejdsvagt kræver en relation til en eksisterende `User` for at kunne

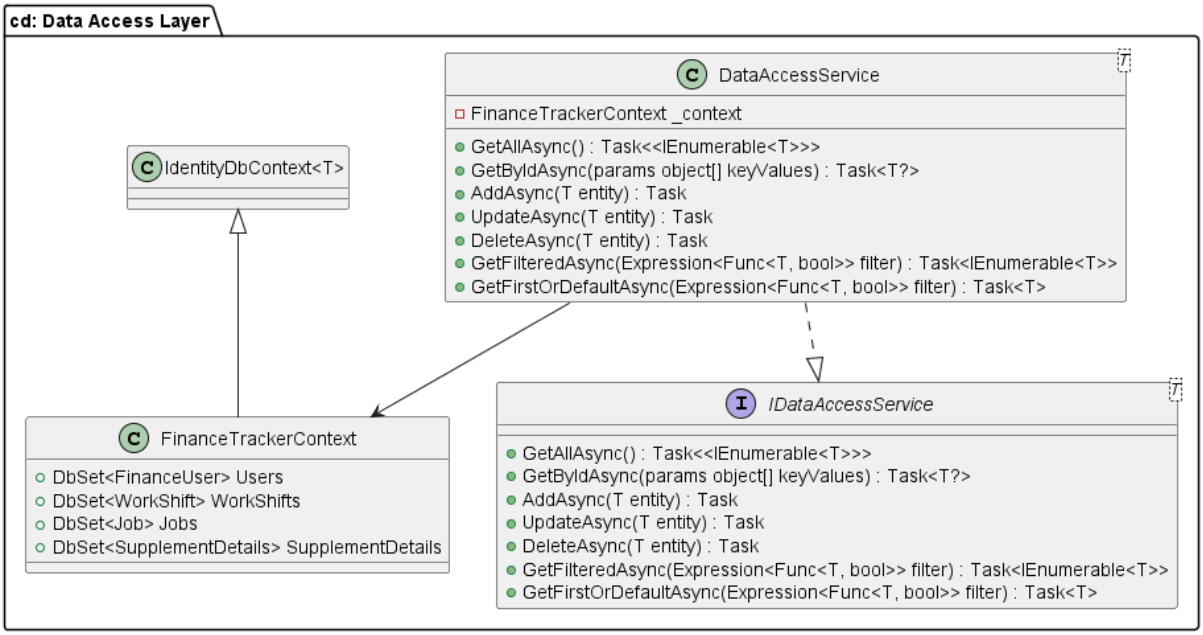
eksistere.

6.2 Design af Backend

I dette afsnit gennemgåes design for komponenterne:

- Data Access Layer
- Authentication Controller
- Paycheck Controllers

6.2.1 Data Access Layer



Figur 6.2. Klasse diagram for Data Access Layer

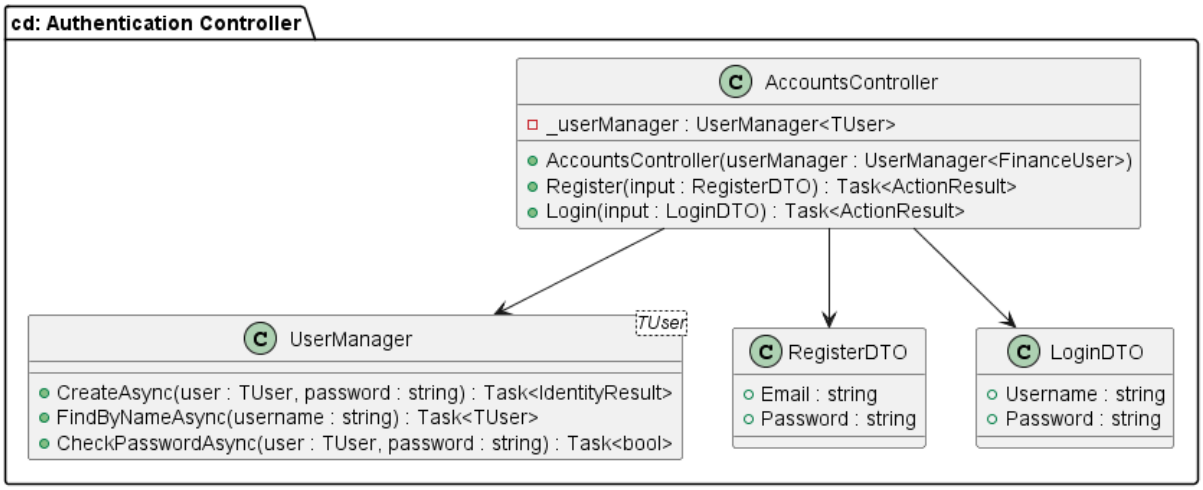
På figur 6.2 vises klassediagrammet for Data Access Layer(DAL), som udsteder et generisk interface som har en generisk implementering.

Designet følger *Repository Pattern*, hvor formålet er at adskille dataadgangen fra forretningslogikken. Hermed overholdes *Single Responsibility Principle*(SRP). DAL Interfacet sikrer lav kobling mellem DAL og **Paycheck Controllers** hvilket også forbedrer testbarheden. Den samme implementering kan genbruges til alle modelklasser ved blot at specificere den ønskede model som typeparameter `T` i interfacet.

Klassediagrammet er udarbejdet for at formidle designet mellem gruppemedlemmerne, så alle forstår, hvordan DAL kan anvendes af **Paycheck Controllers** i backenden.

6.2.2 Authentication Controller

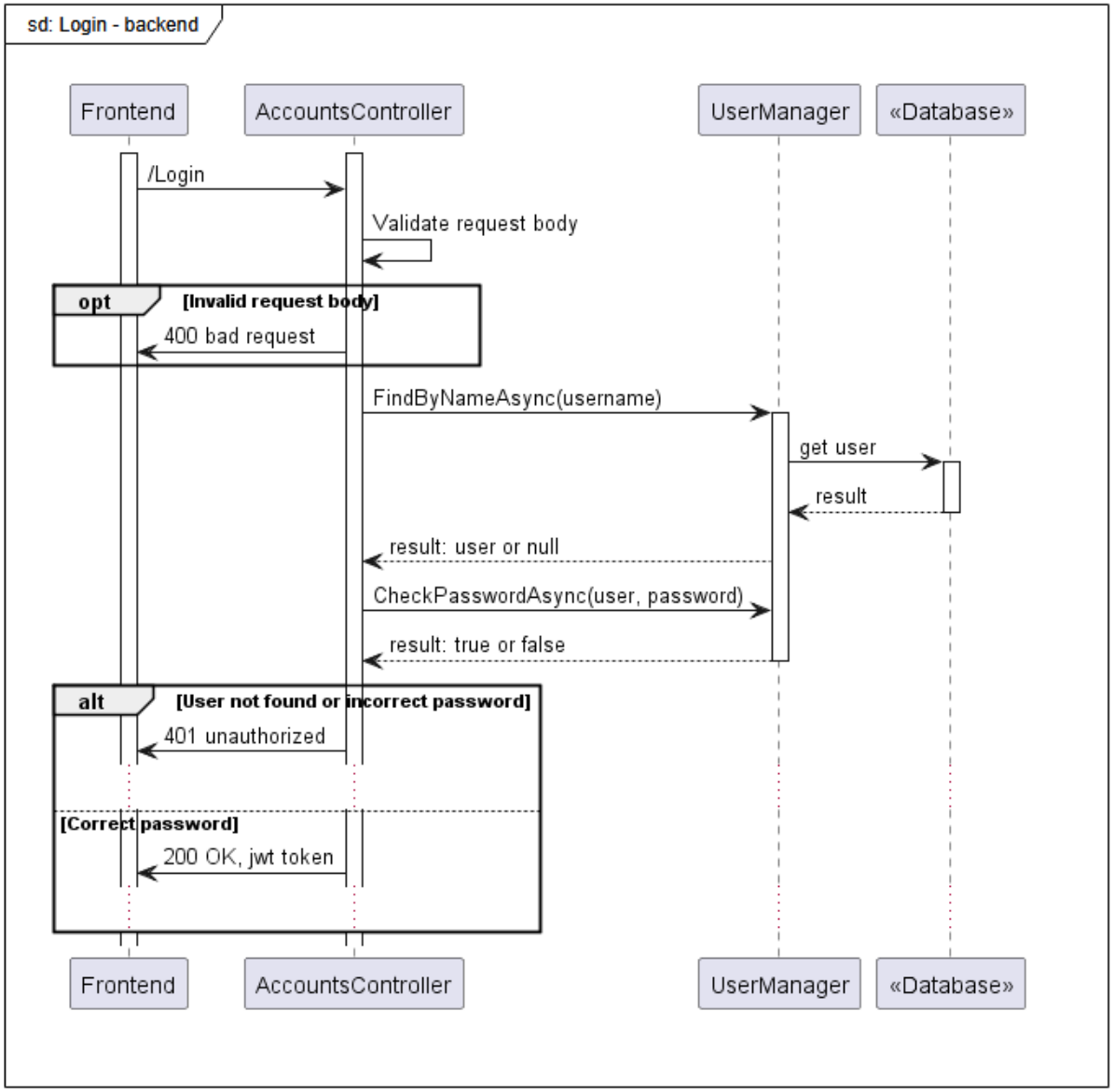
Authentication Controller-komponenten i backend-systemet er designet til at håndtere brugerautentificering og -registrering.



Figur 6.3. Klassediagram for Authentication Controller component

På figur 6.3 vises klassediagrammet for Authentication Controller component. `AccountsController`-klassen fungerer som grænseflade for login og brugerregistrering. Den anvender en ekstern `UserManager`-klasse til at håndtere den underliggende logik for brugeradministration, herunder oprettelse og validering af `FinanceUser`-identiteter. Ansvar separeres dermed mellem håndtering af forespørgsler og brugerlogik, hvilket resulterer i et design, der i højere grad overholder Single Responsibility Principle (SRP).

Authentication Controller tillader anonym adgang med henblik på at understøtte oprettelse og validering af `FinanceUser`-identiteter.

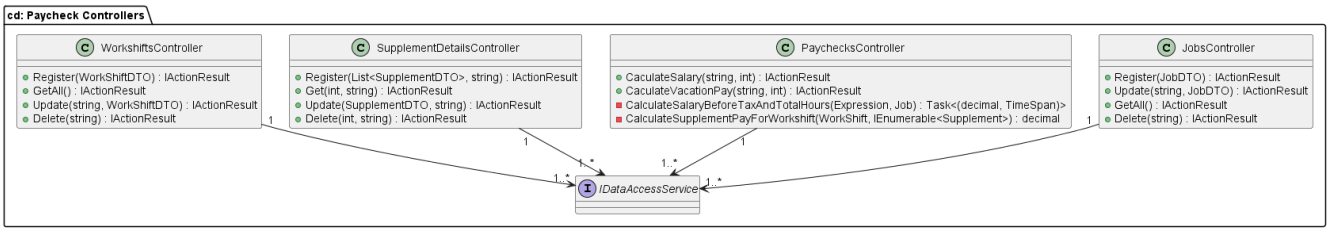


Figur 6.4. Sekvensdiagram for Login

Figur 6.4 viser sekvensdiagrammet for en loginforespørgsel fra en frontend. Diagrammet viser, hvordan `AccountsController` orkestrerer forespørgselsflowet og kalder metoder på `UserManager`, som interagerer direkte med databasen. Systemet benytter token-baseret sikkerhed, hvor en succesfuld loginproces resulterer i en adgangstoken, der identificerer brugeren i efterfølgende kald. Sekvensdiagrammet er anvendt som reference under implementeringen.

6.2.3 Paycheck Controllers

Paycheck Controllers-komponenten i backendens komponentdiagram indeholder flere ASP.NET Core API-controllere, der hver især er dedikeret til at håndtere specifikke forretningsoperationer relateret til lønbehandling.



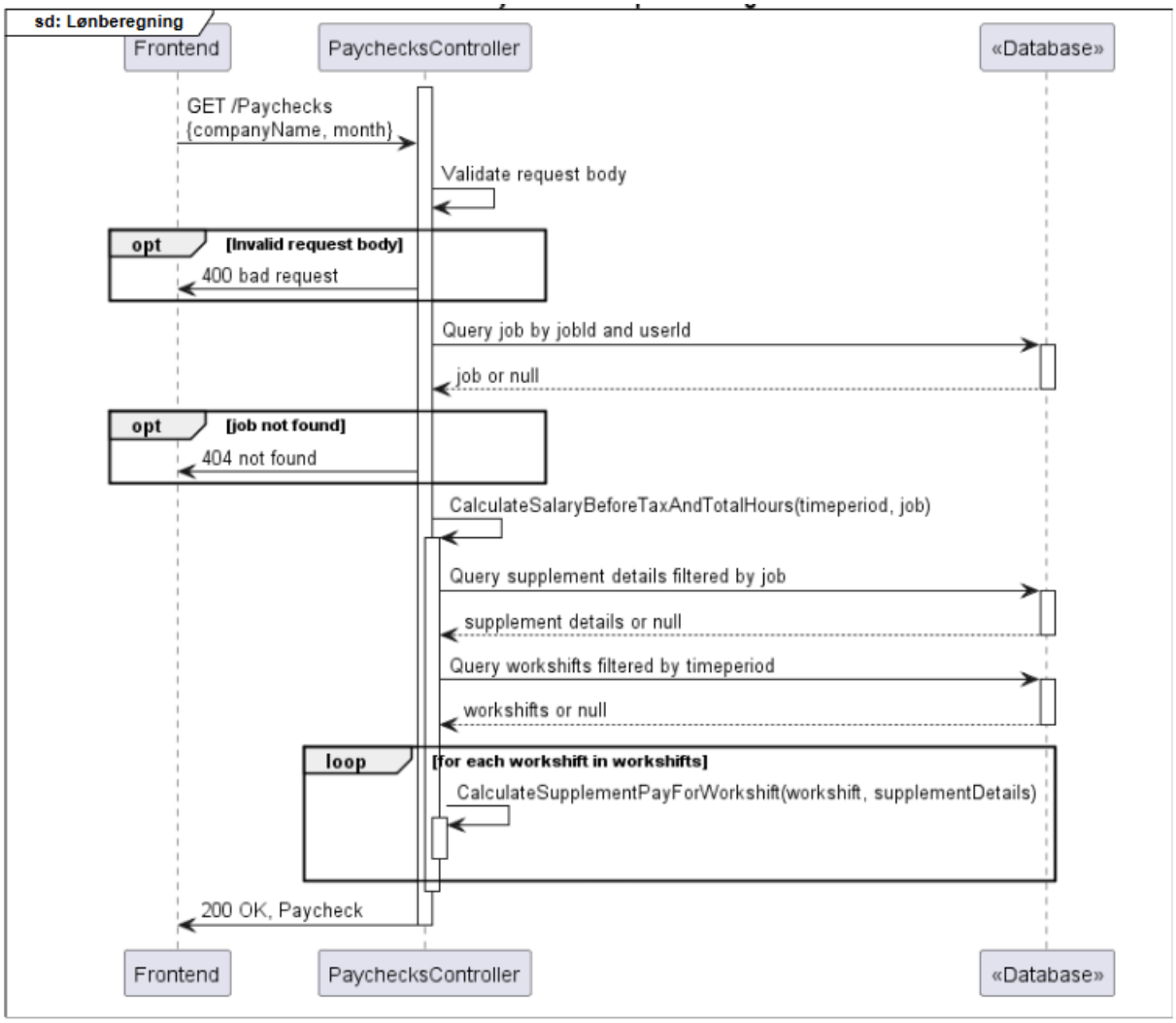
Figur 6.5. Paycheck Controllers class diagram

Figur 6.5 viser klassediagram for Paycheck Controllers. Dette designvalg sigter mod at opretholde **Single Responsibility Principle** (SRP) ved at sikre, at hver controller fokuserer på en veldefineret del af systemets funktionalitet.

- WorkshiftController - CRUD operationer for Workshift databasetabel
- SupplementDetailsController - CRUD operationer for SupplementDetails databasetabel
- PaychecksController - Bergne løn for lønperiode og feriepenge for et år.
- JobsController - CRUD operationer for Job databasetabel

For at garantere systemsikkerhed er alle API-endepunkter inden for denne komponent designet til at kræve autorisation, hvorved anonym adgang systematisk er blokeret. Figuren viser at hver controller anvender et eller flere `DAL` interface.

Den konkrete klasse `PaychecksController` har bl.a. ansvar for at udføre løn beregniner.



Figur 6.6. Sekvensdiagram for GET /Paychecks

Figur 6.6 viser sekvensdiagram lønberegning. Formålet med diagrammet er at illustrerer hvilke tabeller der bliver hentet data fra, hvilke tilfælde metoden skal early-exit og hvornår hjælpemetoder bliver kaldt. Dette har givet overblik og er blevet anvendt som en reference under implementeringen. Bemærk at DAL er blevet abstraheret væk for at forbedre overskueligheden i sekvensdiagrammet.

6.3 Design af Native Applikation

Dette afsnit beskriver designet af den native applikation, som er udviklet i `.NET MAUI`. Applikationen er struktureret efter MVVM-arkitekturen, der består af tre hovedkomponenter: *Views*, *ViewModels* og *Models*.

Viewet har ansvaret for at definere og præsentere brugergrænsefladen, mens *ViewModelen* indeholder de egenskaber og kommandoer, som Viewet binder sig til. Dette muliggør en klar adskillelse mellem UI, præsentationslogik og forretningslogik. *Modellaget* repræsenterer forretningslogik og data, typisk hentet fra en database eller et API. Denne struktur fremmer overskuelighed, testbarhed og genanvendelighed i koden.

Modellaget i den native applikation består af services, som interagerer med backendens API.

Da komponentdiagrammet for MAUI-appen (figur 5.4) følger MVVM-arkitekturen, fokuserer dette afsnit på samspillet mellem komponenttyperne: *Views*, *ViewModels* og *Services*.

Komponentdiagrammet er opdelt i to designafsnit. Sektion 6.3.1 beskriver designet af følgende komponenter:

- Authentication Views
- Authentication ViewModels
- Authentication Models

Sektion 6.3.2 fokuserer på designet af:

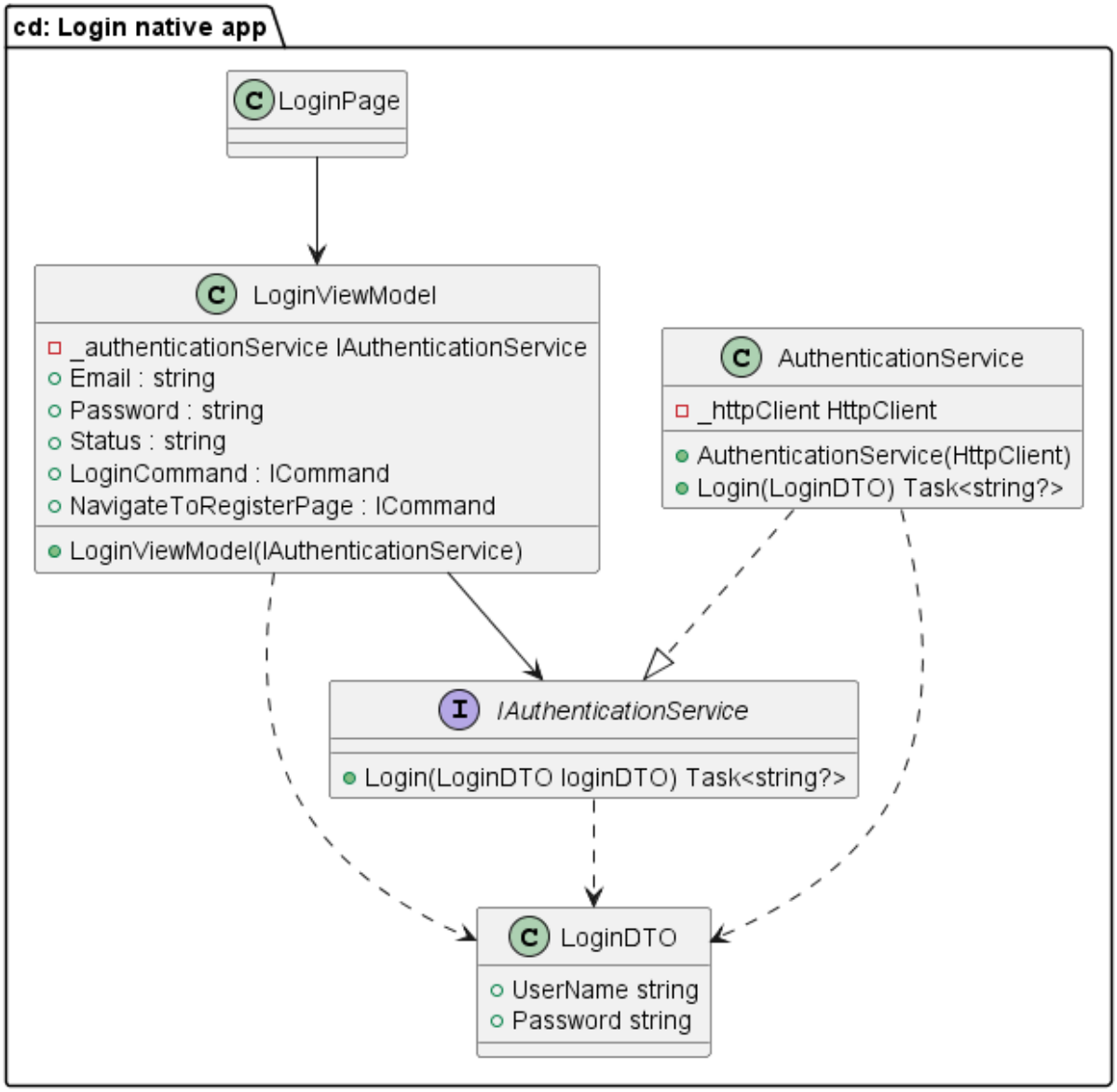
- Paycheck Views
- Paycheck ViewModels
- Paycheck Models

De tilhørende klassediagrammer følger Dependency inversion principle(DIP), hvor ViewModels er afhængige af serviceinterfaces, der implementeres af konkrete serviceklasser. Afhængighederne håndteres ved hjælp af dependency injection (DI), hvilket sikrer lav kobling og forbedret testbarhed.

6.3.1 Authentication Views, ViewModels og Models

Authentication-komponenternes ansvar omfatter login og brugerregistrering.

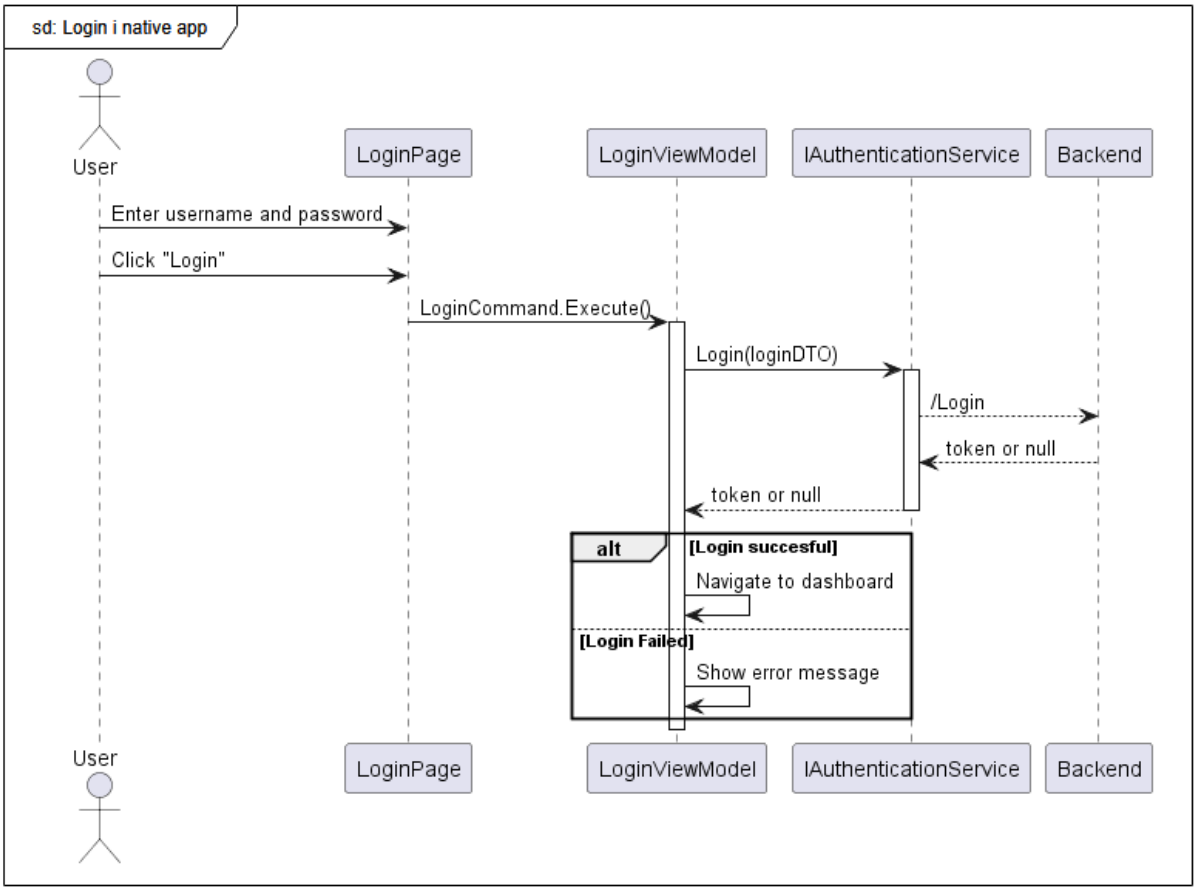
6.3.1.1 Login



Figur 6.7. Klassediagram for login-funktionalitet

Figur 6.7 viser klassediagrammet for login-funktionaliteten. Diagrammet illustrerer, hvordan klasser fra de forskellige MVVM-komponenter interagerer. `LoginViewModel` har properties og kommandoer, som `LoginPage` binder sig til. `AuthenticationService` injectes i `LoginViewModel`.

Klassediagrammet skabte en fælles forståelse af login-funktionaliteten i gruppen og muliggjorde parallel implementering. Én udvikler kunne implementere `LoginPage`, mens en anden arbejdede med `LoginViewModel`.



Figur 6.8. Login sekvensdiagram for native app

Figur 6.8 viser login sekvensen fra native app. Diagrammet viser hvordan brugerhandlinger på `LoginPage` udløser metodekald på `LoginViewModel` vha. databinding. Ved fejlet login-forsøg vises en statusmeddelelse til brugeren, baseret på ændringer i ViewModel'en, som automatisk opdaterer UI'en gennem databinding.

6.3.1.2 Brugerregistrering

Design af brugerregistreringen er vedhæftet som bilag 10 i form af et klassediagram.

6.3.2 Paycheck Views, ViewModels og Models

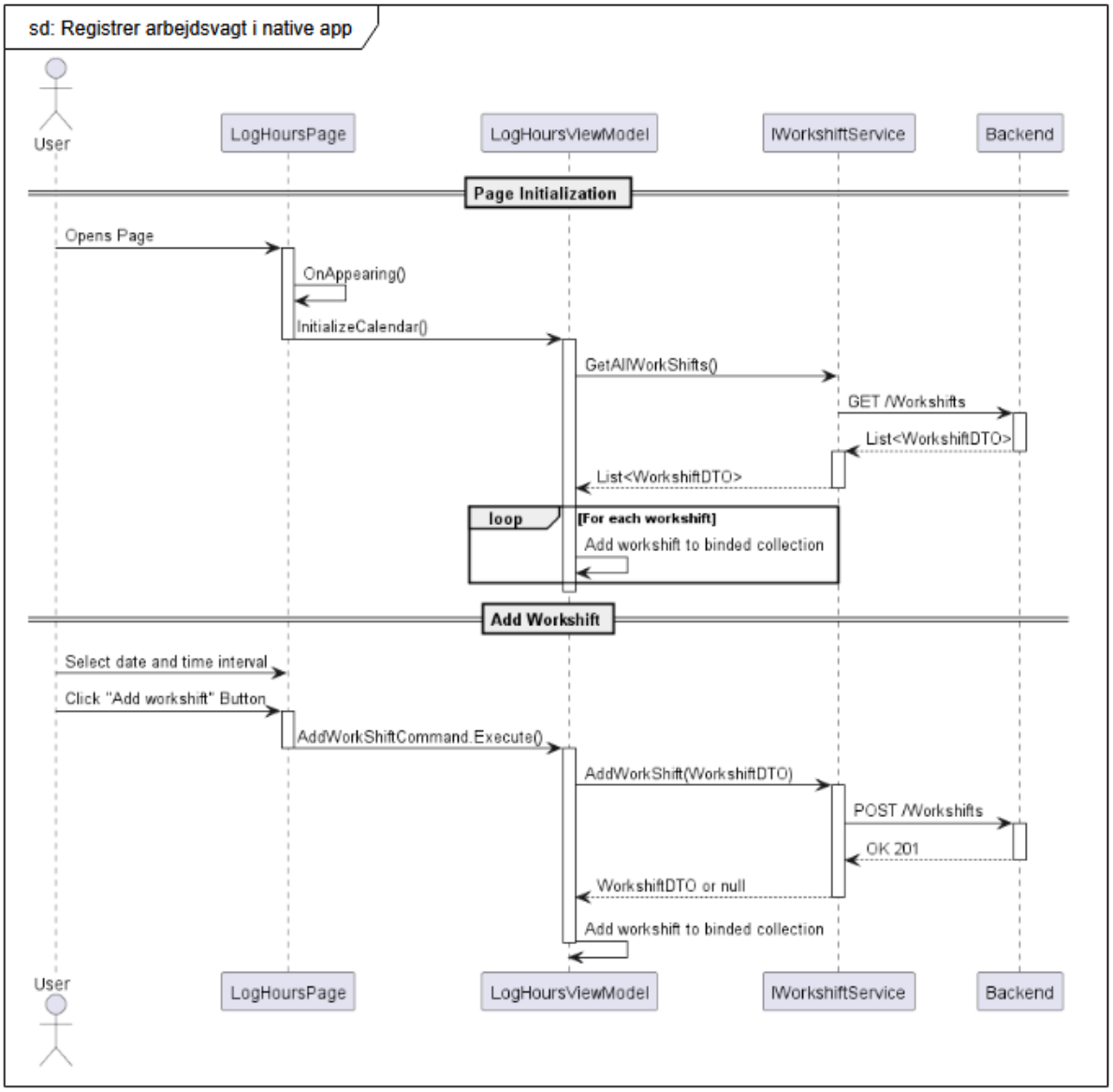
Dette afsnit beskriver det centrale design for de tre Paycheck-komponenter fra figur 5.4. Komponenterne har ansvar for mange funktionaliteter.

Tabel 6.2. Mapping af View til Funktionalitet

View	Funktionalitet(User-story)
LogHoursPage	US8
PaycheckPage	US1, US2 og US9
SupplementDetailsPage	US3
VacationPayPage	US10

Tabel 6.2 viser mapping mellem Views og User-stories. Det viser hvilke sider i `Paycheck Views` der er ansvarlige for forskellige User-stories. I dette afsnit beskrives design for LogHoursPage og PaycheckPage.

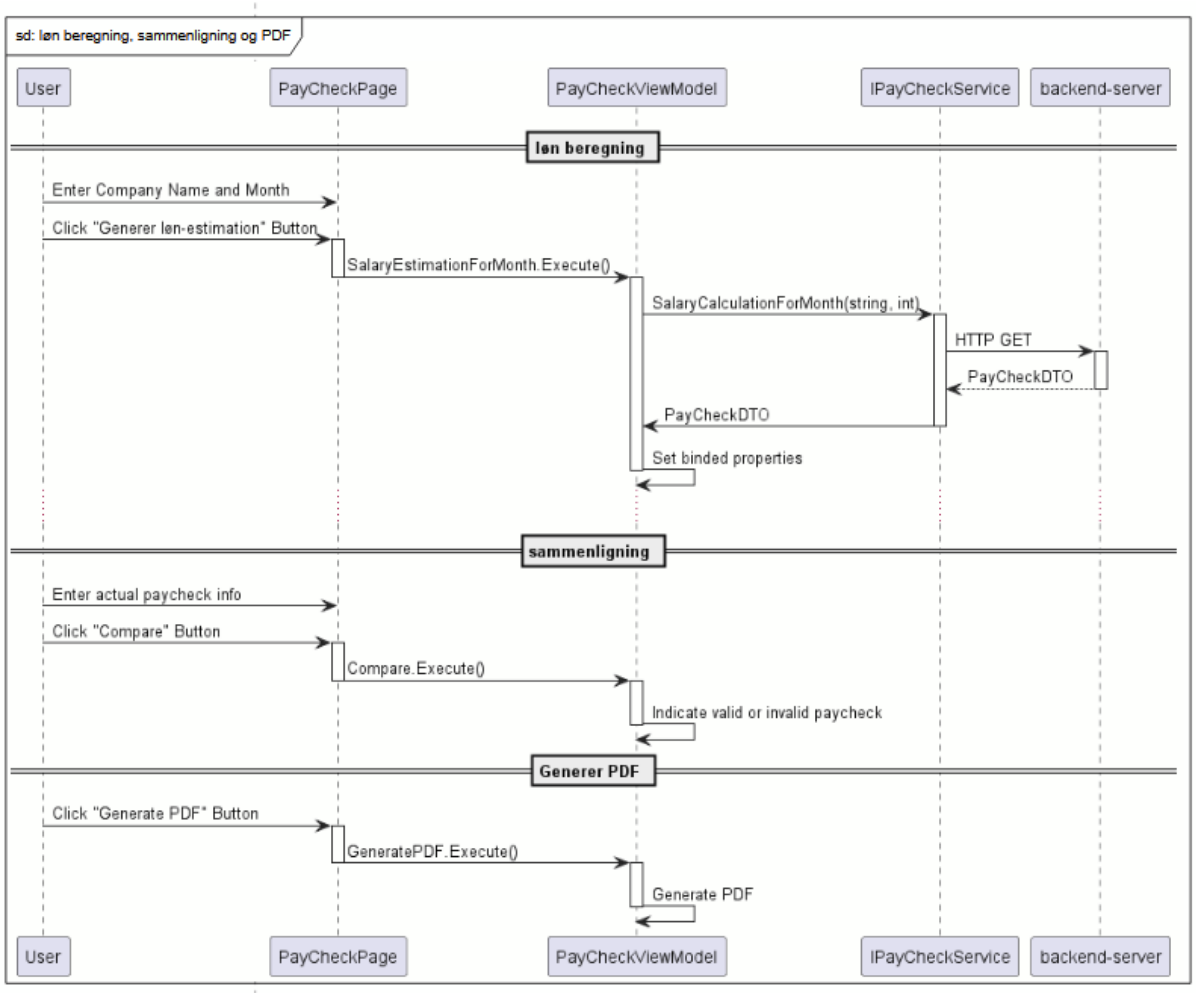
6.3.2.1 Registrering af arbejdsvagter(LogHoursPage)



Figur 6.9. Sekvensdiagram for registrering af arbejdstimer

Figur 6.9 viser sekvensdiagrammet for interaktionen med `LogHoursPage`, som anvendes til oprettelse og visning af arbejdsvagter. Brugerens handlinger på siden udløser kommandoer i `LogHoursViewModel`, der benytter `IWorkshiftService` til at tilgå backend og udføre læse- og skriveoperationer mod databasen.

6.3.2.2 Lønberegning, sammenligning og PDF-generering(PaycheckPage)



Figur 6.10. Sekvensdiagram for lønberegning, sammenligning og PDF-generering

Figur 6.10 viser sekvensdiagrammet for brugerens interaktion med `PaycheckPage`. Det er udelukkende lønberegningsfunktionen, der kræver interaktion med backend. Funktionerne til sammenligning og PDF-generering udføres i `PaycheckViewModel`, ved at benytte data, der allerede er hentet fra løn beregningen.

6.3.3 Design af navigation og routing

Navigationsdesignet i applikationen er baseret på en `Flyout`-menu, som strukturerer og præsenterer de vigtigste sider for brugeren i en overskuelig venstremenu. Hver sektion i menuen repræsenterer en selvstændig side og er defineret direkte i `Shell`-strukturen uden behov for manuel rute-registrering.

Navigationen håndteres automatisk af MAUI's `Shell`-system og understøtter både brugerinitieret navigation via menuvalg og programmatisk navigation i forbindelse med logout.

6.4 Design af Web applikation

Dette afsnit beskriver designet af webapplikationen, der er udviklet med React og TypeScript. Webklienten fungerer som en Single Page Application (SPA) med en komponentbaseret struktur hvilket gør at applikationen loader én gang i browseren og derefter dynamisk opdaterer indholdet. Den komponentbaserede tilgang indebærer, at funktionaliteten er opdelt i genanvendelige UI-komponenter, som hver især håndterer en afgrænset del af brugergrænsefladen og logikken. Applikationens SPA-struktur afspejler sig i designet ved, at man fokuserer på komponenternes sammensætning frem for separate side-layouts.

Webapplikationens struktur og komponenter tager udgangspunkt i det komponentniveau, der er beskrevet i component diagrammet (figur 5.5). Her fremgår det, hvordan UI-komponenter, state management og services tilsammen danner grundlaget for hele SPA'ens funktionalitet og interaktion med backend.

Component diagrammet bliver opdelt i 2 designafsnit:

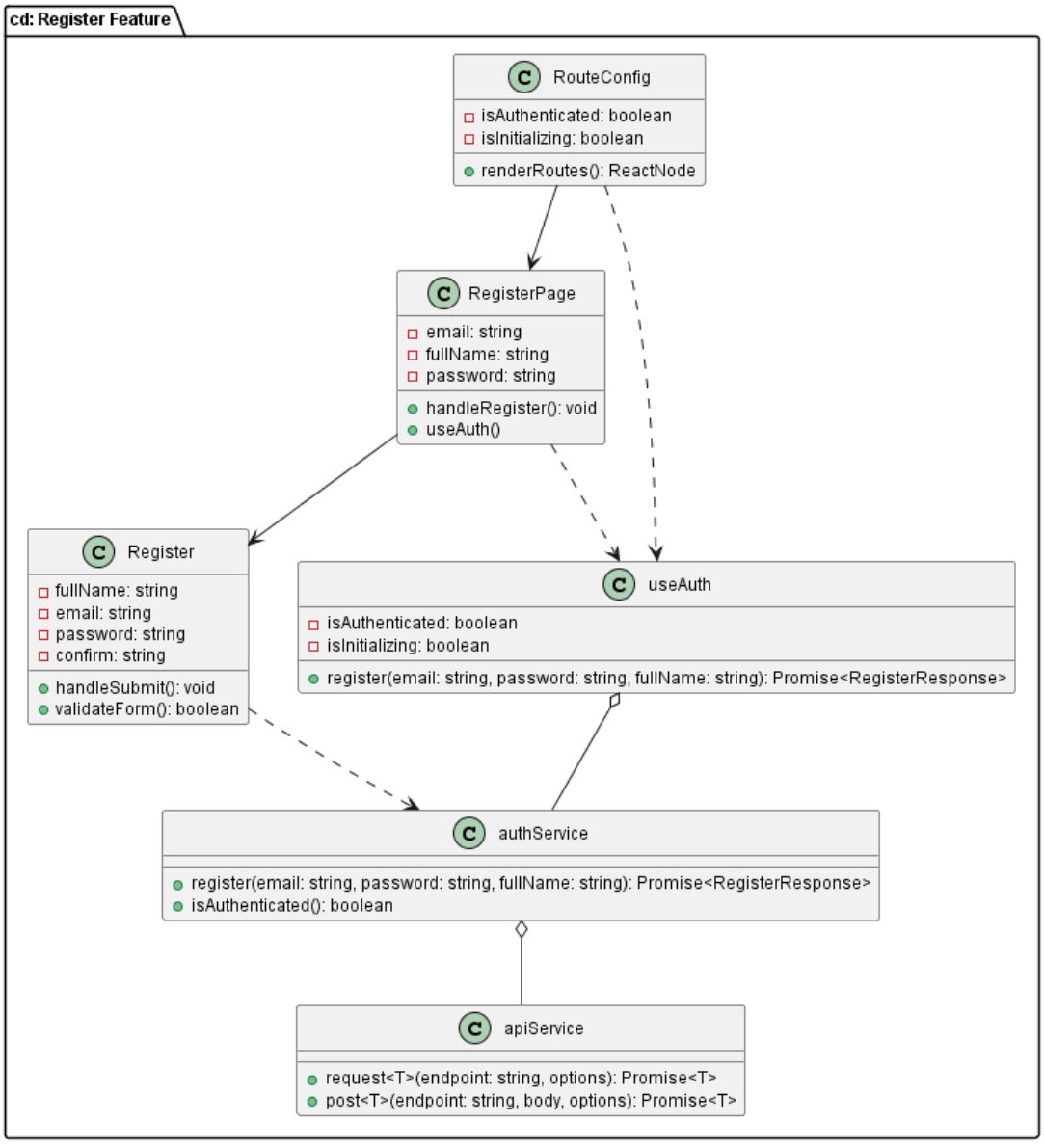
- **Authentication Features:** Authentication, Component, Authentication Services og Protected Route.
- **Paycheck Features:** Paycheck Components ,Paycheck Services.

6.4.1 Authentication Features:

Authentication Features har ansvar for to funktionaliteter, login og brugerregistrering, som opfylder US7. I dette afsnit beskrives design for begge funktionaliteter.

6.4.1.1 Registrer Bruger

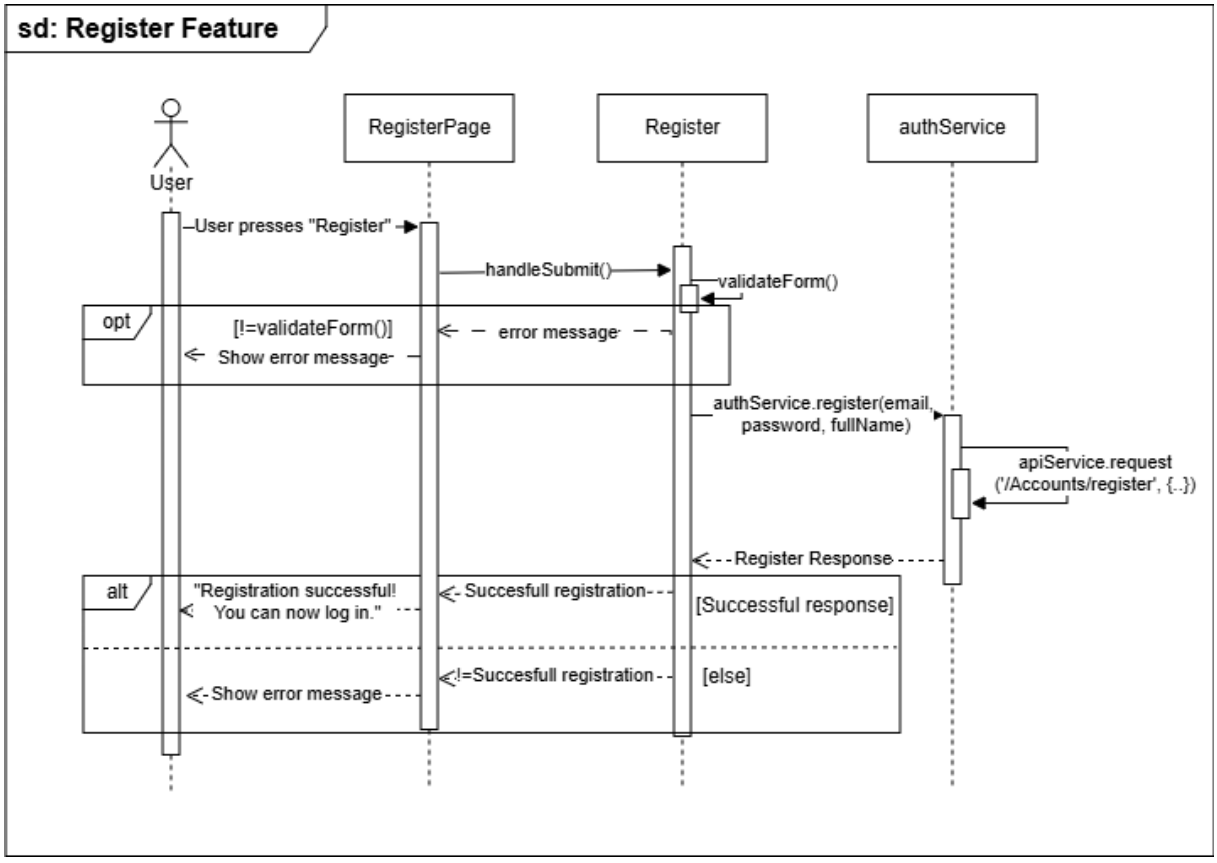
Register featuren er blevet implementeret jvf. Figur 5.5 hvor der anvendes Authenticationcomponent og authentication Services.



Figur 6.11. Register Feature Klasse Diagram

Figur 6.11 viser den strukturelle opbygning af registreringsfunktionen. RegisterPage hånd-

terer overordnet navigation og inputfelter, mens selve valideringen og brugerinteraktionen er isoleret i Register-komponenten. I modsætning til login omfatter registrering ekstra feltlogik som navn og passwordbekræftelse, hvilket stiller højere krav til lokal validering. Kommunikationslaget er abstraheret via useAuth og authService, hvilket muliggør lav kobling og genbrug på tværs af applikationen.

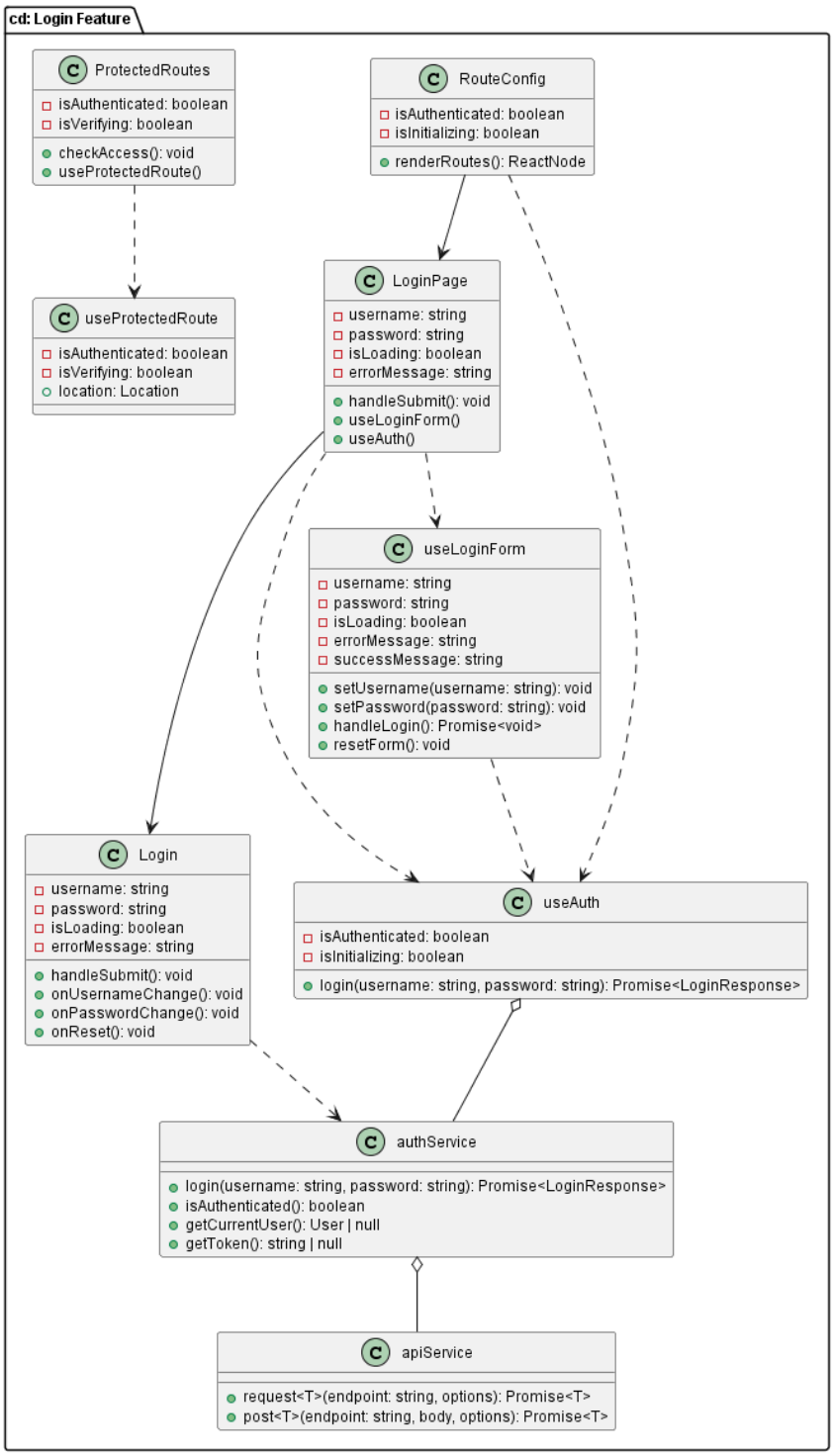


Figur 6.12. Register Feature Sekvens Diagram

Figur 6.12 illustrerer det sekventielle flow for en brugerregistrering. Før API-kald udføres lokal validering for at sikre datakvalitet. Kun ved korrekt input initieres registreringsanmodningen til backend. Ved positivt svar præsenteres brugeren for en bekræftelse, mens fejlmeddelelser håndteres lokalt. Diagrammet har bidraget til at definere kontrolstrukturen og sikre robust håndtering af både succes og fejltilfælde.

6.4.1.2 Login

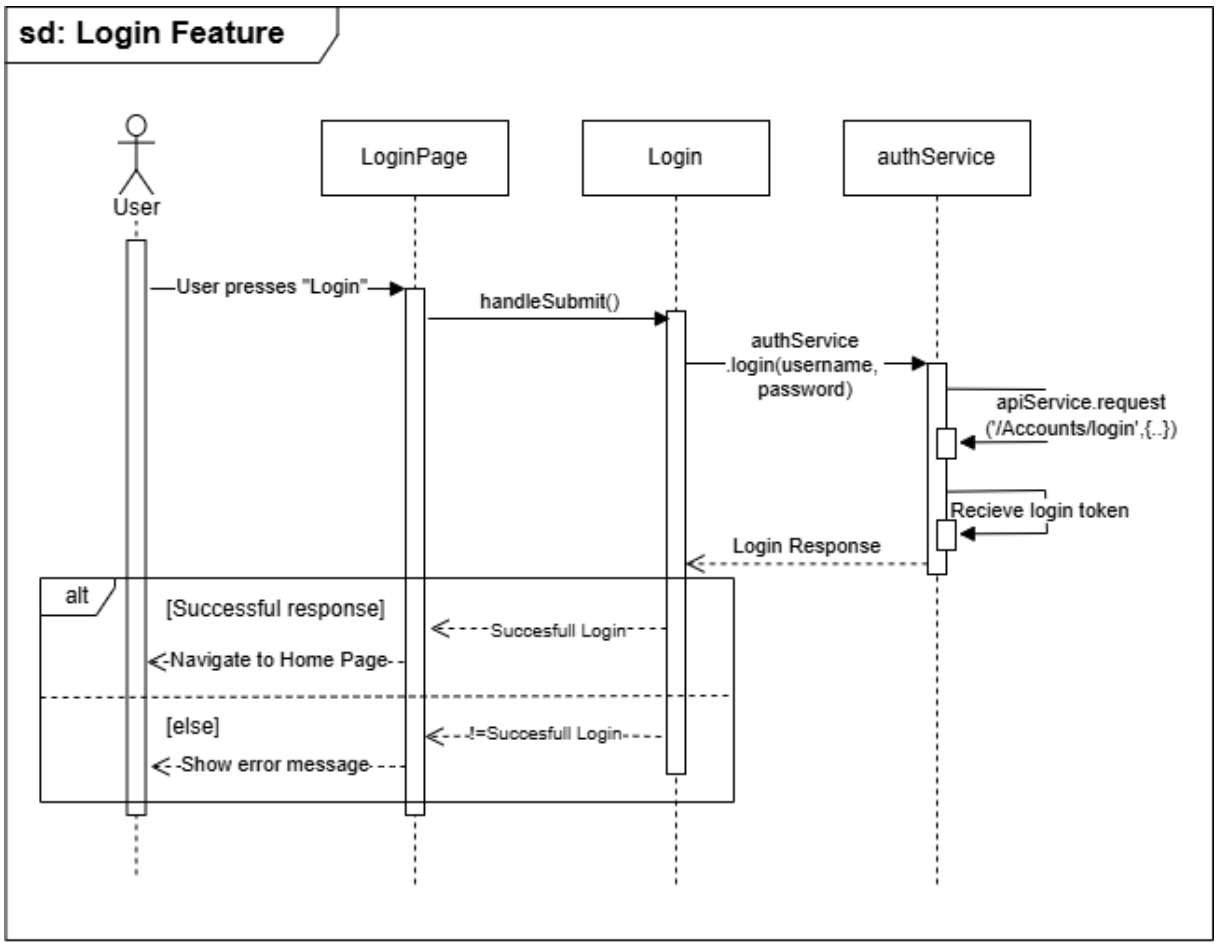
Login featuren er blevet implementeret jvf. Figur 5.5 hvor der anvendes Authenticationcomponent og authentication Services.



Figur 6.13. Login Feature Klasse Diagram

Figur 6.13 viser designet for login-funktionaliteten. LoginPage er ansvarlig for at opsamle brugerinput og benytter useLoginForm til at håndtere formularens tilstand og hændelser. Autentificering udføres gennem authService via useAuth, som desuden eksponerer loginstatus til resten af applikationen. Adgangsbegrænsede sider kontrolleres

af ProtectedRoute, der benytter tilstanden fra useAuth. Designet understøtter styring af sikkerhed. Validering og tilstandsstyring i useLoginForm sikre unødige API-kald.



Figur 6.14. Login Feature Sekvens Diagram

Figur 6.14 viser det logiske forløb i login-processen. Når brugeren indsender loginoplysninger, behandles disse gennem komponenthierarkiet og videresendes til backend. Ved succesfuld login opdateres klientens autentificeringstilstand, og brugeren navigeres videre. Ved fejl vises en passende besked.

6.4.2 Paycheck Features:

Paycheck Features har ansvar for flere funktionaliteter:

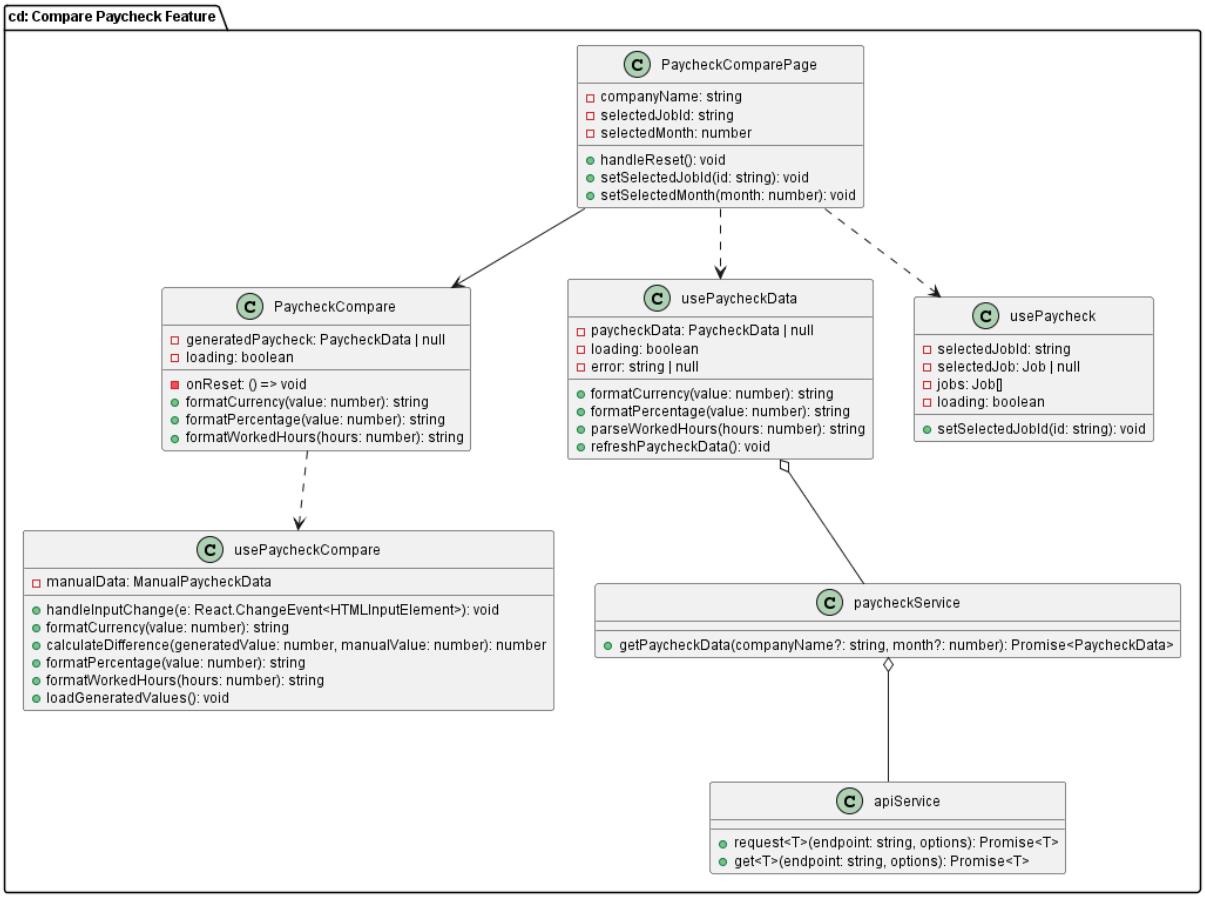
Tabel 6.3. Mapping af Component til Funktionalitet

Component	Funktionalitet(User-story)
Workshift	US8
Paycheck Overview	US1 & US9
Paycheck Commpare	US2
Job	US3
Vacation Pay	US10

I dette afsnit beskrives design for Paycheck Compare, Job og Workshift. Resten af funktionaliteter er beskrevet i bilag 5.

6.4.2.1 Løn Sammenligning

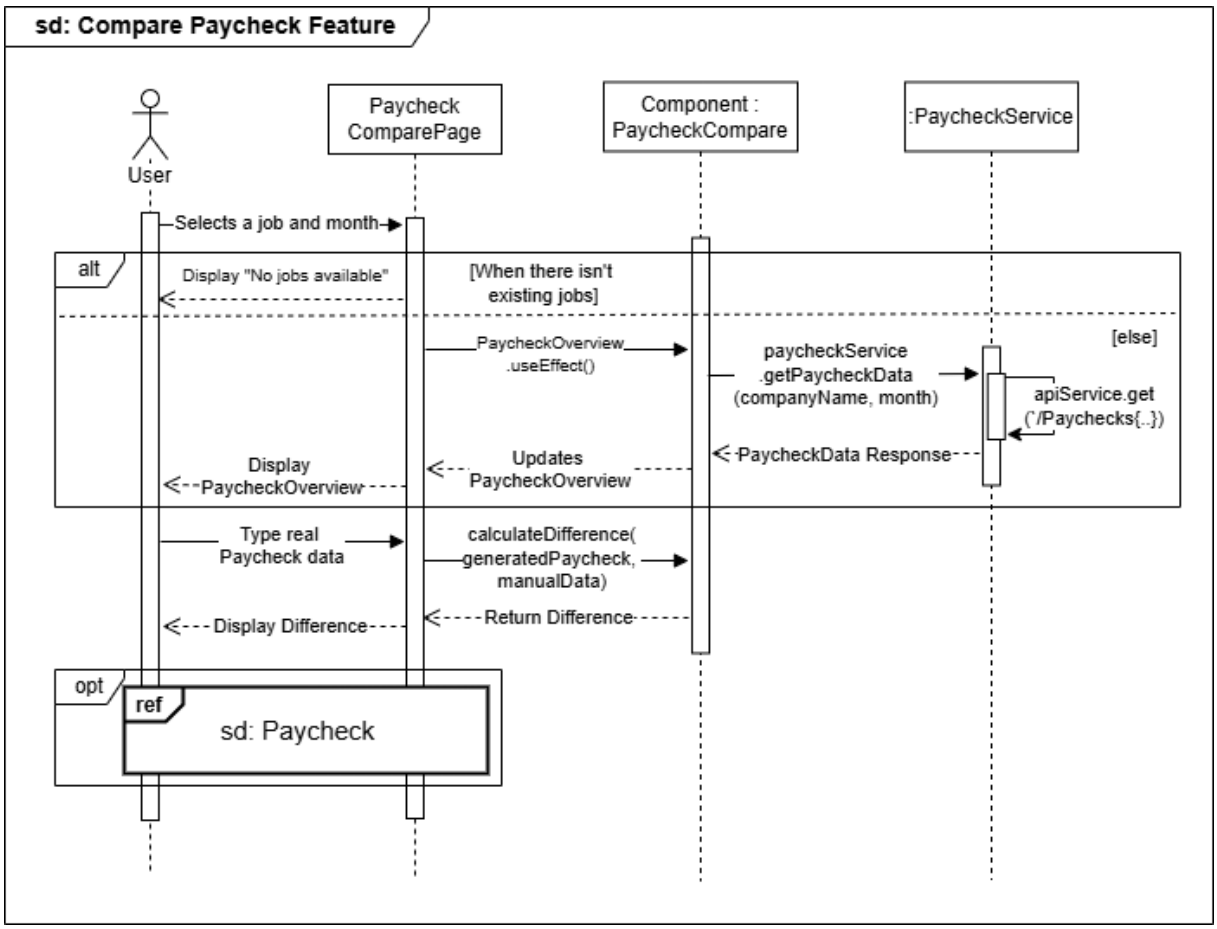
Paycheck Comparison featuren er blevet implementeret jvf. Figur 5.5 hvor den tilhører Paycheck Components og Paycheck Services.



Figur 6.15. Compare Paycheck Feature Klasse Diagram

Figur 6.15viser arkitekturen bag sammenligningsfunktionen. PaycheckComparePage styrer

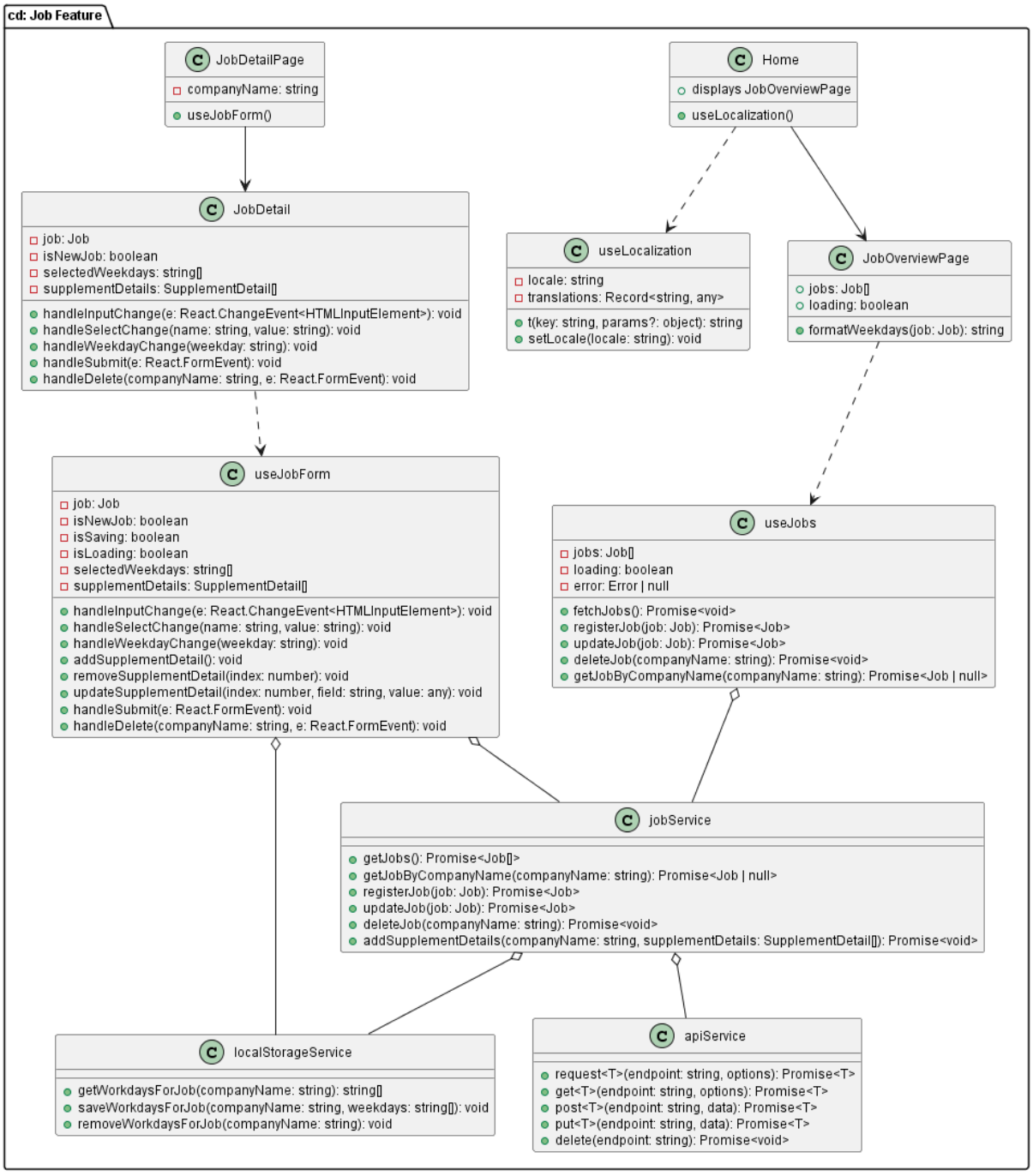
overordnede valg, mens PaycheckCompare og tilhørende hooks håndterer hhv. visning og beregning. API-adgangen er kapslet i paycheckService, hvilket adskiller præsentationslag og datalogik. Komponentansvaret er tydeligt opdelt, så både genererede og manuelle data behandles isoleret.



Figur 6.16. Compare Paycheck Feature Sekvens Diagram

Figur 6.16 fokuserer på brugerens interaktion med funktionen. Når et job vælges, forsøger systemet at hente eksisterende løndata. Derefter kan brugeren indtaste egne værdier, hvorefter forskellen beregnes og vises. Diagrammet afklarer, hvordan komponenterne samarbejder i et dynamisk flow baseret på både systemdata og brugerinput.

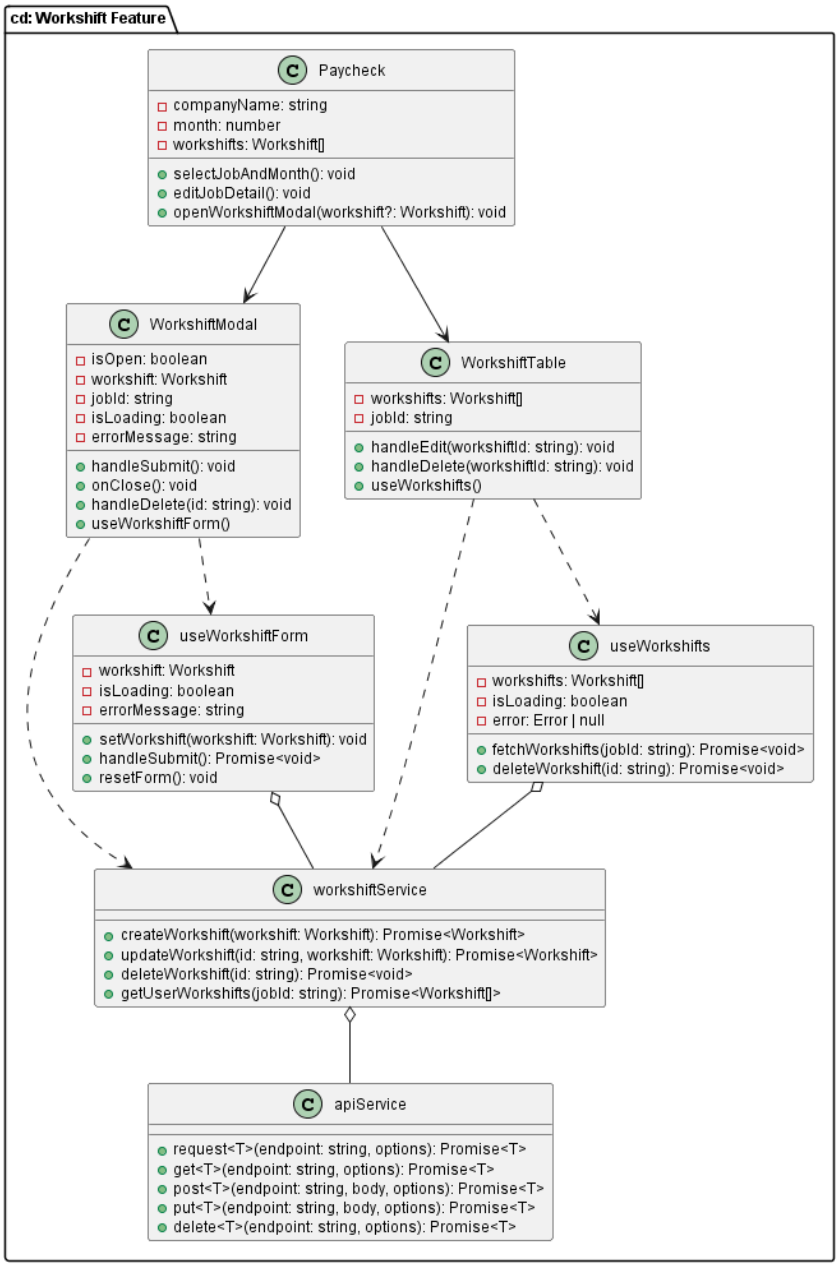
6.4.2.2 Job Administration



Figur 6.17. Job Feature Klasse Diagram

Figur 6.17 viser den strukturelle opbygning af jobfunktionen. JobDetailPage håndterer visning og redigering gennem useJobForm, som styrer tilstand og inputlogik. CRUD-operationer understøttes via useJobs, der kommunikerer med jobService. Funktionaliteten inkluderer håndtering af supplerende oplysninger som tillæg via supplementDetails, som bruges i lønberegning. Lokal lagring understøttes af localStorageService.

6.4.2.3 Arbejdsvagt Administration



Figur 6.18. Workshift Feature Klasse Diagram

Figur 6.18 8 viser strukturen for arbejdsvagtfunktionen. Paycheck håndterer overblik og åbner modalvinduer, mens WorkshiftModal og WorkshiftForm styrer oprettelse og redigering. CRUD-operationer varetages af workshiftService via useWorkshifts. Designet adskiller datalagring, præsentation og formularlogik.

Implementering 7

Dette kapitel beskriver den tekniske realisering af Finance Tracker. Kapitlet beskriver implemetering af systemet baseret på designarbejdet.

7.1 Implementering af Database

Til implementeringen af databasen er der anvendt Code First-tilgangen[35]. Ved at `FinanceTrackerContext`(Figur 6.2) arver fra `IdentityDbContext<T>` tilføjes en række predefinerede tabeller i databasen, når man laver en migration og anvender den på databasen. Her er det kun den predefinerede entitet `AspNetUsers` som anvendes. Tabellen svarer til `User` tabellen i ER-diagrammet på figur 6.1.

`IdentityDbContext` tager typeparameter `T`, som skal være klassen `IdentityUser` eller en klasse som arver fra `IdentityUser`. Hvis man har bruge for yderligere attributer på den predefinerede tabel `AspNetUsers` så kan man definere en klasse som arver fra `IdentityUser` og angive den som typeparameter `T`. Dermed tilføjes attributer til `AspNetUsers`.

Dermed kan vi etablere relationer til `Job` og `Workshift` fra `AspNetUsers(User)` ved at definere navigationsegenskaber på `FinanceUser`:

- ICollection<Job>
- ICollection<WorkShift>

I FinanceTrackerContext.cs defineres `DbSet<T>-properties`, hvor hver `DbSet<T>` bliver til en tabel i databasen ved en migration.

7.2 Implementering af backend

I dette afsnit bliver gennemgåes vigtige implementeringsdetaljer for komponenterne:

- Data Access Layer
- Authentication Controller

- Paycheck Controllers

7.2.1 Implementering af Data Access Layer

For at DAL kan kommunikere med databasen er der anvendt EFC, FinanceTrackerContext.cs(Figur 6.2) arver fra klassen IdentityDbContext<T> som er fra EFC. `DbSet<T>` properties på FinanceTrackerContext.cs repræsenterer databasetabeller og har metoder, som anvendes til at skrive og læse fra databasen. Disse metoder anvendes af DataAccessService til at implementere interfacet `IDataAccessService`.

Metoderne `GetFilteredAsync` og `GetFirstOrDefaultAsync` på `DataAccessService` tager imod en `Expression<Func<T, bool»`, som er en LINQ-udtrykstruktur i C#. Dette muliggør fleksibel og effektiv databaseforespørgsel, da udtrykket kan oversættes direkte til SQL af Entity Framework Core.

7.2.2 Implementering af Authentication Controller

Authentication Controller udsteder 2 endpoints:

- **POST /Login** – Udsteder et JWT-token ved succesfuld login.
- **POST /Register** – Registrerer en ny bruger i systemet.

Her er der anvendt den eksterne klasse UserManager<T>(figur 6.3) fra nuget-pakken Microsoft.AspNetCore.Identity. Denne klasse anvendes til at verficere brugernavn eksisterer, korrekt adgangskode og oprette nye brugere. Klassens metoder læser og skriver til AspNetUsers tabellen i databasen.

For at beskytte endspoints som ikke tillader anonym adgang er der konfigureret authentication middleware i program.cs. Derefter er der anvendt tagget `[Authorize]` ved endpoints som kræver autentificeret bruger.

Ved succesfuldt login bliver der tilføjet claims i token, som returneres til klient. Disse claims anvendes til at identificere bruger ved request til beskyttede endpoints. Det er muligt at udtrække brugerens Id med følgende kodeudsnit:

Listing 7.1. Udtræk bruger Id

```
1  var UserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
```

Beskyttede endpoints kan hermed identificere brugeren vha. claim. For at generere nye tokens ved succesfuld login, anvendes en **Signing Key**, som er defineret som en miljøvariabel. Denne bruges sammen med algoritmen `HmacSha256` til at oprette `signingCredentials`:

Listing 7.2. Instantiering af signingCredentials

```
1 var signingCredentials = new SigningCredentials(
2     new SymmetricSecurityKey(
3         System.Text.Encoding.UTF8.GetBytes(_configuration["JWT:SigningKey"])),
4     SecurityAlgorithms.HmacSha256);
```

Derefter oprettes en række `claims`, som beskriver brugerens identitet:

Listing 7.3. Instantiering af claims

```
1 var claims = new List<Claim>
2 {
3     new Claim(ClaimTypes.Name, user.UserName),
4     new Claim(ClaimTypes.Email, user.Email),
5     new Claim(ClaimTypes.NameIdentifier, user.Id)
6 };
```

Til sidst oprettes et JWT-token objekt:

Listing 7.4. Generering af JWT-token med konfigurationsdata og signatur

```
1 var jwtObject = new JwtSecurityToken(
2     issuer: _configuration["JWT:Issuer"],
3     audience: _configuration["JWT:Audience"],
4     claims: claims,
5     expires: DateTime.Now.AddSeconds(17000),
6     signingCredentials: signingCredentials);
```

Dette JWT-token sendes tilbage til klienten og bruges til efterfølgende autentificering. `signingCredentials` sikrer, at autentificeringsmiddleware på serveren kan validere, at det modtagne token faktisk er udstedt af serveren selv.

7.2.3 Implemetering af Paycheck controllers

Eftersom API-controllere `WorkshiftsController`, `JobsController` og `SupplementDetailsController` udfører CRUD operationer på deres tilsvarende database-entitet bliver de ikke gennemgået da deres implementeringer anses som

trivielle. Dette afsnit beskriver implementeringsdetaljer for den konkrete API-controller PaychecksController.cs fra figur 6.5. API-controlleren eksponerer endpoints:

- GET /Paychecks - Beregner løn
- GET /Paychecks/VacationPay - Beregner feriepenge for et år

Ved bergning af løn bliver Hjælpemetoden `CalculateSalaryBeforeTaxAndTotalHours` kaldt, som beregner løn før skat og total antal arbejdstimer for en lønperiode. Denne metode kalder `CalculateSupplementPayForWorkshift` for hver arbejdsvagt i en lønperiode, dette kan ses på figur 6.6. Baseret på løn før skat og totale arbejdstimer kan der yderligere udledes resterende information for lønberegningen.

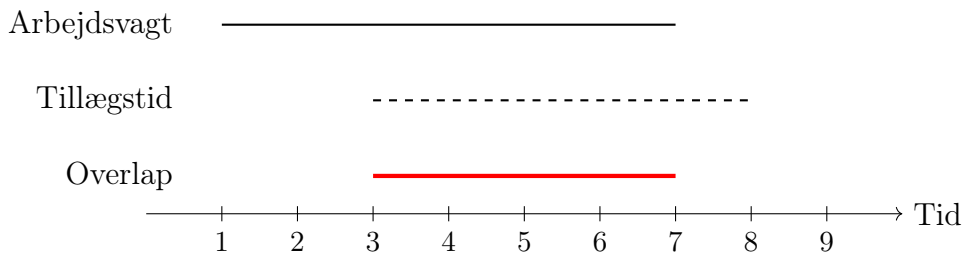
For at sikre korrekt lønberegning er det nødvendigt at tage højde for tillægstider.

Tabel 7.1. CalculateSupplementPayForWorkshift

Metode	Parameter	Retur
`CalculateSupplementPayForWorkshift`	`Workshift, List<SupplementDetails>`	`Decimal`

Tabel 7.1 beskriver hjælpemetoden `CalculateSupplementPayForWorkshift` i Paychecks-Controller. Metoden returnerer tillægsløn for en arbejdsvagt.

Ved beregning af løn bliver `CalculateSupplementPayForWorkshift` kaldt for hver arbejdsvagt i den givne lønperiode(Se figur 6.6). Metoden er central i systemets forretningslogik.



Figur 7.1. Illustration af overlap mellem arbejdsvagt og tillægstid

På figur 7.1 visualiserer problemet, der skal findes overlap(rød) mellem en arbejdsvagt og et tillægstidsrum. Formålet med algoritmen er nemlig at finde overlappet mellem arbejdsvagt tidsrum og tillægstidsrum og derefter gange tillægsløn med overlappet.

Listing 7.5. Beregning af tillægsløn baseret på overlap mellem vagt og tillægstidspunkt

```
1       var start = workShift.StartTime > supplementDay.StartTime ? workShift.StartTime :
            supplementDay.StartTime;
2       var end = workShift.EndTime < supplementDay.EndTime ? workShift.EndTime :
            supplementDay.EndTime;
3
4       if (start >= end) return 0;
5
6       decimal hoursWorked = (decimal)(end - start).TotalHours;
```

Det korte kodeudsnit på listing 7.5 viser hvordan overlappet kan findes. Det sker ved at:

- Initialisere start til at være den største at de 2 starttider
- Initialisere end til at være den mindste at de 2 sluttider

Hvis start tiden er over eller lig med slut tiden, så er der ikke overlap.

7.3 Implementering af Frontend i Native App

NuGet-pakker: Projektet anvender bl.a. `CommunityToolkit.MVVM` til at understøtte MVVM-arkitekturen i MAUI. Dette forenkler databinding og brugen af kommandoer.

7.3.1 Authentication

Dette afsnit beskriver centrale implementeringsdetaljer for authentication-komponenterne vist i komponentdiagrammet (figur 5.3).

7.3.1.1 Login

Ved et succesfuldt login er det nødvendigt at gemme det modtagne JWT-token fra serveren på en sikker måde, så det kan anvendes i efterfølgende requests for at autentificere brugeren.

`Login`-metoden i `AuthenticationService.cs` er ansvarlig for at sende login-anmodningen til backend og gemme token i `SecureStorage`. Klassediagrammet for login-funktionaliteten kan ses i figur 6.7.

I de efterfølgende HTTP-anmodninger læses token fra `SecureStorage` og tilføjes som en autorisationsheader, så serveren kan identificere brugeren.

Til dette formål er klassen `AuthHeaderHandler` implementeret. Den arver fra `DelegatingHandler` og overrider metoden `SendAsync`, så headeren automatisk tilføjes:

Listing 7.6. Tilføjelse af JWT-token til HTTP-anmodning i MAUI-klient

```
1 var token = await SecureStorage.GetAsync("auth_token");
2 if (!string.IsNullOrEmpty(token))
3 {
4     request.Headers.Authorization =
5         new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);
6 }
7 return await base.SendAsync(request, cancellationToken);
```

Denne override resulterer i at forespørgsler lavet med en `HttpClient` inkluderer `Authorization`-headeren.

Handleren registreres via DI i `Program.cs` med følgende konfiguration til en vilkårlig service:

Listing 7.7. Registrering af HTTP-klient med autorisations-handler

```
1 builder.Services.AddHttpClient<IPayCheckService, PayCheckService>()
2     .AddHttpMessageHandler<AuthHeaderHandler>();
```

Derved inkluderes autorisationsoplysninger automatisk i alle anmodninger til beskyttede endpoints.

7.3.2 Paycheck

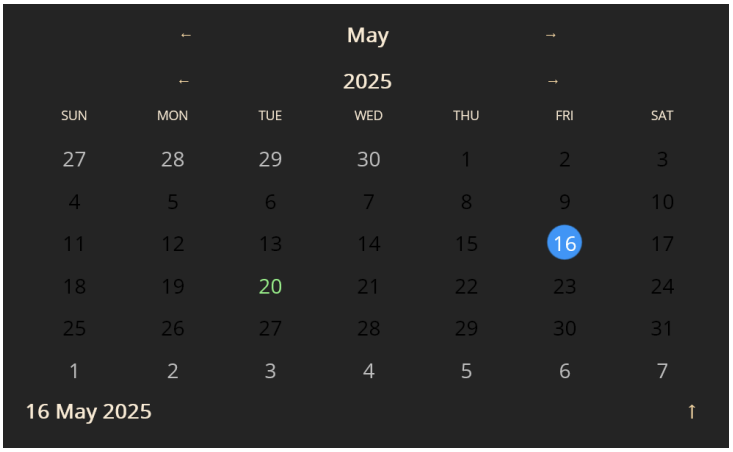
Dette afsnit beskriver nøgleaspekter ved implementeringen af Paycheck-komponenterne, jf. komponentdiagrammet (figur 5.3).

7.3.2.1 Registrering af arbejdsvagter

Da MAUI ikke har en indbygget kalendarkontrol, er `Plugin.Maui.Calendar` anvendt(NuGet-pakke). Denne pakke giver mulighed for at visualisere events med en kalenderkontrol og kan integreres med MVVM-mønsteret.

I `LogHoursPage` bindes propertyen `Events`, som er af typen `EventCollection<Event>`, til kalenderkontrollen. Når en arbejdsvagt registreres, vises den på kalendarkontrollen, ved at

tilføje arbejdsvagten til `Events` propertien. Kalendarkontrollen og `EventCollection<Event>` typen er fra pakken `Plugin.Maui.Calendar`.



Figur 7.2. Kalendervisning med arbejdsvagter

Figur 7.2 viser kalenderkontrollen. Ved åbning af siden skal vagter indlæses fra database og vises på kalenderen. Til dette er Lifecycle metoden `OnAppearing` på `LogHoursPage` overriden, så den kalder `InitializeCalendar()` i `LogHoursViewModel`. Metoden henter arbejdsvagter og tilføjer dem til en binded collection som vist i følgende kodeudsnit:

Listing 7.8. Indlæsning og gruppering af vagter i kalenderen

```
1  public async Task InitializeCalendar()
2  {
3      var workshifts = await _workshiftService.GetAllWorkShifts();
4      foreach (var workshift in workshifts)
5      {
6          var date = workshift.StartTime.Date;
7          Events[date] = new List<string>
8          {
9              $"{workshift.StartTime:HH:mm}-{workshift.EndTime:HH:mm}"
10         };
11     }
12 }
```

`Events`-propertyen indekseres med datoer, ved indexet kan der indsættes en string som repræsenterer en arbejdsvagt, kalendarkontrolleren opdaters pga. databinding.

7.3.2.2 Lønberegning

For at brugeren kan se lønberegningen er metoden `SalaryCalculationForMonth()` defineret i `PaycheckViewModel.cs`. Denne metode har tagget `[RelayCommand]`, ved at tilføje dette tag bliver der automatisk genereret en ICommand property som hedder metodenavnet + Command postfix, så for `SalaryCalculationForMonth()` vil der oprettes en ICommand ved navn `SalaryCalculationForMonthCommand`. Denne ICommand kalder `SalaryCalculationForMonth()` hvis den triggers fra Viewet. Fra Viewet kan en knap nu binde til denne ICommand.

ViewModelen indeholder bl.a. følgende properties:

Listing 7.9. Properties i PaycheckViewModel.cs

```
1  [ObservableProperty]
2  private decimal salaryBeforeTax;
3  [ObservableProperty]
4  private double workedHours;
5  [ObservableProperty]
6  private decimal amContribution;
7  [ObservableProperty]
8  private decimal tax;
9  [ObservableProperty]
10 private decimal salaryAfterTax;
```

Properties i listing 7.9 er markeret med attributten `[ObservableProperty]`. Bemærk, at properties starter med små bogstaver. Når `[ObservableProperty]` anvendes, genereres der automatisk en offentlig property med stort begyndelsesbogstav samt den nødvendige kode til at understøtte `INotifyPropertyChanged`-interfacet.

For eksempel vil feltet `salaryBeforeTax` automatisk generere en offentlig property ved navn `SalaryBeforeTax`. Når værdien ændres, vil den genererede kode automatisk udløse en `PropertyChanged`-begivenhed, hvilket gør det muligt for UI'et at opdatere sig dynamisk, når data ændres i ViewModelen. Dette reducerer boilerplate-kode og sikrer en mere vedligeholdelsesvenlig implementering af databinding.

Resultat af metoden `SalaryCalculationForMonth` bliver sat til properties på listing 7.9, som resulterer i at Viewet opdaters og viser resultat af løn beregningen.

7.3.2.3 Sammenligning

Efter bruger har hentet en løn beregning, er det nu muligt for bruger at sammenligne med en akutel lønseddel. Bruger indtaster oplysninger for sin aktuelle lønseddel og trykker på en knap, hermed indikeres om der er afvigelser eller ej. Det er implmenteret ved at definere `[ObservableProperty]` properties på PaycheckViewModel som har typen `Microsoft.Maui.Graphics.Color`, derefter er yderligere er der defineret `string` [ObservableProperty] properties som binder til inputfelter hvor bruger indtaster oplysninger om akutelle lønseddel. Metoden `[RelayCommand] Compare()` på `PaycheckViewModel` sammenligner properties binded til inputfelter med properties fra listing 7.9, som holder på løn beregning og indikerer afvigelser ved at sætte enten rød eller grøn farve på Color properties. Color properties kan bindes til Label kontrollers baggrundsfarve.

7.3.2.4 PDF-generering

Generering af PDF-dokumenter er implementeret ved hjælp af `iText7`-biblioteket. Funktionaliteten gør det muligt for brugeren at eksportere en lønseddel som en PDF-fil, der indeholder løn-beregningen.

PDF-filen oprettes ved hjælp af klasserne `PdfWriter`, `PdfDocument` og `Document` fra iText7, hvor indholdet struktureres med overskrifter, afsnit og tekstformatering. Den færdige fil gemmes lokalt på brugerens enhed.

Når PDF'en er genereret, modtager brugeren en bekræftelse via en dialogboks i brugergrænsefladen.

7.4 Implementering af frontend i Web App

I dette afsnit gennemgås centrale implementeringsdetaljer for komponenternerne med fokus på struktur, ansvar og samspil. Formålet er at belyse den tekniske opbygning og sikre forståelse for komponenternes funktion og integration.

7.4.1 Værktøjer

Frontend-projektet er opbygget med følgende værktøjer og biblioteker:

- Vite (Build-tool)
- Tailwind CSS (Styling)
- Fetch API (HTTP-klient)
- NPM (Pakkehåndtering)

7.4.2 Implementering af Authentication Features:

Registrering er implementeret via `RegisterPage`, som anvender `Register`-komponenten til formularvisning. Brugerdata valideres lokalt og sendes til backend via `authService.register()` og `apiService.post()`.

7.4.2.1 Login

Login sker via `Login`-komponenten, der håndterer input og kalder `authService.login()` med brugerens data. JWT-token gemmes i `localStorage`. Adgangskontrol håndteres af `ProtectedRoutes`, og routing defineres i `RouteConfig`.

7.4.3 Implementering af Paycheck Features

7.4.3.1 Løn Sammenligning

Funktionen er bygget op omkring `PaycheckCompare`, som henter løndata og tillader brugeren at sammenligne med egne tal. Beregninger sker via `usePaycheckCompare`, og visning er responsiv med farvekodning. Formhooks håndterer loading-, fejl- og success-tilstande lokalt, hvilket forbedrer brugerfeedback og reducerer behovet for global state.

7.4.3.2 Job Administration

Jobfunktionaliteten anvender `JobOverviewPage` og `JobDetail`, med datahåndtering gennem `useJobForm`. CRUD-operationer udføres via `jobService`, som benytter `apiService` til backend-integration.

7.4.3.3 Arbejdsvagt Administration

Vagter vises i `WorkshiftTable` og håndteres gennem `WorkshiftModal` og `useWorkshiftForm`. Data hentes, oprettes og opdateres via `workshiftService`. Tidsformatering håndteres lokalt, og bruger-ID hentes fra `authService`.

7.5 Deployment

Dette afsnit beskriver, hvordan de forskellige komponenter i FinanceTracker-applikationen bliver deployed og gjort tilgængelige for brugerne.

7.5.1 Database

Databasen er hostet i Microsoft Azure ved hjælp af Azure SQL Database-tjenesten. Databasen kører i en serverløs konfiguration med prisniveauet Free – General Purpose – Serverless: Gen5, 2 vCores[36].

7.5.2 Backend

Backend-applikationen deployes automatisk på Render via Docker. Når ændringer pushes til main-branchen på GitHub-repositoriet, bygger Render automatisk et Docker-image baseret på Dockerfile'en i FinanceTracker-mappen og kører det som en web-service.

7.5.3 Web-app

Web-applikationen deployes på Netlify gennem en automatiseret proces defineret i netlify.toml filen[37]. Når koden pushes til repository'et, kører Netlify byggekommandoen:

```
1    npm run typecheck \&\& npm run build
```

som kontrollerer typerne og bygger applikationen til produktionsbrug. Den færdige app publiceres fra dist-mappen.

Netlify-konfigurationen håndterer også API-redirects til backend-serveren på Render, samt sikrer at alle frontend-ruter fungerer korrekt i single-page applikationen.

7.5.4 Native-app

Native-applikationen publishes ved hjælp af Visual Studio 2022. Programmet versionsstyres, og nødvendige filer oprettes automatisk under build-processen.

Der er oprettet et lokalt signeringscertifikat til autentificering af MSIX-pakken, hvilket giver tilladelse til at downloade og køre applikationen. Dette var nødvendigt for at sikre, at alle brugere kunne installere og anvende appen.

Test 8

For at sikre høj kvalitet og pålidelighed i systemet er der gennemført en række tests. Dette kapitel redegør for de anvendte testmetoder – herunder enhedstests, integrationstests og accepttests.

8.1 Modultest

Modultest er en testmetode, hvor individuelle enheder af koden testes isoleret for at sikre, at de fungerer korrekt. Det hjælper med tidlig fejlfinding og øger kodekvaliteten.

8.1.1 Backend

En stor del af testarbejdet er blevet udført ved hjælp af Swagger, som har fungeret som både dokumentation og interaktivt testværktøj for API'et. Swagger gjorde det muligt at teste hvert endpoint under ensartede betingelser. En væsentlig fordel ved Swagger var dets debug-muligheder, der gav detaljerede fejlbeskeder ved mislykkede kald. Dette gjorde det nemt at identificere fejl og årsagen til dem.

8.1.2 Database

En stor del af testarbejdet blev udført ved hjælp af Azure Data Studio. Dette var det oplagte valg til at simulere data og teste, om data blev gemt som forventet, når DAL blev kaldt. Derudover blev der anvendt en database seeder til at initialisere databasen med testdata, som kunne bruges til at validere funktionalitet og sikre, at ændringer i koden ikke introducerede fejl i databaselogikken. Seederen blev især brugt til at teste datamodeller, relationer og validering af input.

8.1.3 Native-App

En stor del af testarbejdet blev udført ved hjælp af en lokal database, som gjorde det muligt at simulere virkelige scenarier og sikre, at applikationen reagerede korrekt på

forskellige typer input. Målet var at sikre, at funktionerne gav det forventede output, når de interagerede med databasen.

8.1.4 Web-App

En stor del af testarbejdet blev udført ved hjælp af en lokal server, der simulerede backendens REST API. Det gjorde det muligt at udvikle og teste CRUD-operationer i webklienten. Serveren brugte samme endpoints og datastruktur som den endelige backend, hvilket sikrede nem integration. Test blev udført manuelt via fetch()-kald, hvor både succes og fejlscenarier kunne simuleres og UI-feedback valideres.

8.2 Integrationstest

8.2.1 Backend og Database

Til integrationen mellem server og database blev der anvendt Entity Framework Core. Efter instansiering af ORM'en i programmets Main-metode, blev Swagger anvendt til at teste forskellige HTTP-kald, for at sikre at dataen fra request body korrekt blev gemt i databasen. For at validere resultaterne og undgå falske positive blev Azure Data Studio brugt som sekundær kontrol, hvor det blev verificeret, at dataen faktisk blev indsat og opdateret korrekt i databasen.

8.2.2 Frontend og Backend

Til integration mellem maui og backend er der brugt klassen HttpClient i C#. Denne klasse bliver injected in i de klasser som står for at kalde HTTP kald til backend. Når den korrekte URL er defineret og servicen er registreret i main, så blev det testet ved hjælp af funktionaliteten fra MAUI og Azure Data Studio blev igen brugt som sekundær kontrol, hvor samme fremgangsmåde som i Bakcend og database blev brugt. Til integration mellem webapplikationen og backend blev JavaScript-funktionen fetch() anvendt til at foretage HTTP-kald. Disse kald blev udført direkte i de komponenter, der håndterer dataindhentning fra backend-API'et. Når den korrekte URL var defineret, og routing samt datahåndtering var sat op i React-komponenterne, blev funktionaliteten testet gennem webapplikationens brugergrænseflade. Som sekundær kontrol blev Azure Data Studio brugt

til at bekræfte, at de forventede ændringer skete i databasen – samme fremgangsmåde som beskrevet under backend- og database.

8.3 Accepttestspecifikation

Dette afsnit beskriver, hvordan test af både funktionelle og ikke-funktionelle krav udføres, samt hvilket resultat der forventes for hver test. Specifikationen bidrager til at gøre kravene mere testbare og anvendelige.

8.3.1 Accepttestspecifikation for funktionelle krav

Accepttest specifikation for alle funktionelle krav med brug af Gherkin baseret på user stories[38].

US1: Lønberegning

Baggrund:

- **Givet** at brugeren er logget ind
- **Og** har registreret arbejdsvagter for en lønperiode

Scenarie: Beregne løn (Webapplikation & Native app)

- **Når** brugeren ønsker at se en lønberegning for en lønperiode
- **Så** navigerer bruger til lønberegnings siden
- **Og** angiver hvilken måned løn-estimeringen skal genereres for
- **Og** klikker på beregn knappen
- **Så** skal systemet systemet præsentere en løn-beregning til brugeren

US2: Lønsammenligning

Baggrund:

- **Givet** at brugeren er logget ind
- **Og** har en lønseddel for en specifik måned
- **Og** har registreret arbejdstimer for den måned

Scenarie: Sammenligne lønseddel (Webapplikation & Native app)

- **Når** brugeren navigerer til sektionen "Lønsammenligning"
- **Så** vælger brugeren den relevante måned
- **Og** indtaster den udbetalte løn
- **Så** trykker brugeren på "Sammenlign"
- **Så** skal systemet vise en sammenligning mellem den aktuelle lønseddel og systems lønberegning
- **Og** indikere om der er en afvigelse

8.3.2 US3: Tillægstider

Baggrund:

- **Givet** at brugeren er logget ind
- **Og** har registreret job
- **Og** brugeren har følgende eksempel tillægsinformation:

Ugedag	Tillæg	Tidsrum
Mandag	30 DKK	18-22

Scenarie: Registrer tillægstid (Webapplikation & Native app)

- Når brugeren vil registrere tillægsinformationer
- Så navigerer brugeren til tillægsformularen
- Og indtaster tillægsinformation ind
- Og trykker på gem knappen
- Så skal systemet gemme informationen og vise en bekræftelse til brugeren

US4: Platformskompatibilitet

Baggrund:

- **Givet** at brugeren har en enhed med internetforbindelse

Scenarie: Åbne applikationen (Webapplikation & Native app)

- **Når** brugeren tilgår applikationen via web eller mobil
- **Så** skal alle funktioner være tilgængelige på begge platforme

US5. Oversigt over årsindkomst og SU-fribeløb

Baggrund:

- **Givet** at brugeren er logget ind

Scenarie: Se SU-fribeløb (Webapplikation & Native app)

1. **Når** brugeren navigerer til sektionen for SU-fribeløb
2. **Så** vises en oversigt over brugerens samlede indkomst og resterende fribeløb

US6: Advarsel ved overskridelse af SU-fribeløb

Baggrund:

- **Givet** at brugeren er logget ind
- **Og** at brugerens årsindkomst nærmer sig SU-fribeløbsgrænsen

Scenarie: SU-fribeløb overskredet (Webapplikation & Native app)

- **Når** brugerens indkomst overstiger 80% af det gældende årsfribeløb baseret på deres studiestatus.
- **Så** modtager brugeren en advarsel i både webapplikationen og den native app.
- **Og** advarslen indeholder information om det resterende fribeløb og konsekvenserne ved overskridelse.
- **Og** brugeren får anbefalinger til at justere sin arbejdstid for at undgå tilbagebetaling af SU.

US7: Brugerprofil

Baggrund:

- **Givet** at brugeren ikke har en brugerprofil på applikationen

Scenarie: Oprettelse af brugerprofil (Webapplikation & Native app)

- **Når** brugeren ønsker at oprette en brugerprofil
- **Så** trykker bruger på "Registrer"knappen på login siden
- **Så** navigerer applikationen til registreringssiden
- **Så** Indtaster bruger sine oplysninger

- **Og** Bekræfter oprettelse af brugerprofil
- **Så** logger bruger succesfuldt ind

US8: Registrering af arbejdsvagter

Baggrund:

- **Givet** at brugeren er logget ind
- **Og** har registreret et job

Scenarie: Registrering af arbejdsvagt (Webapplikation & Native app)

- **Når** brugeren ønsker at registrere en arbejdsvagt
- **Så** navigerer bruger til registreringssiden
- **Og** indtaster dato og tidsrum for arbejdsvagten
- **Og** trykker på "Registrer"knappen
- **Så** indikerer systemet succesfuld registrering til bruger

US9: Eksport af lønberegning til PDF

Baggrund:

- **Givet** at brugeren er logget ind
- **Og** har genereret en lønberegning for en lønperiode

Scenarie: Eksportere lønberegning som PDF (Webapplikation & Native app)

- **Når** brugeren ønsker at eksportere lønberegningen
- **Så** navigerer bruger til lønberegnings siden
- **Og** klikker på knappen "Eksportér som PDF"
- **Så** skal systemet generere en PDF med den aktuelle lønberegning
- **Og** gemme den på brugers enhed

US10: Feriepengeberegning

Baggrund:

- **Givet** at brugeren er logget ind

- **Og** har registreret arbejdstimer for en periode

Scenarie: Se optjente feriepenge for et år (Webapplikation & Native app)

- **Når** brugeren navigerer til feriepengeoversigten
- **Og** vælger et år fra en dropdown-menu
- **Så** skal systemet beregne og vise de optjente feriepenge for det valgte år
- **Og** vise totalbeløbet samt evt. detaljer pr. måned eller lønperiode

8.3.3 Accepttestspecifikation for ikke funktionelle krav

Denne tabel dækker de ikke-funktionelle krav, som systemet skal opfylde. Testene fokuserer på systemets performance, pålidelighed og supportabilitet. Tabellen giver et klart overblik over, hvordan disse krav testes, og hvilke resultater der forventes.

Tabel 8.1. Accepttest for ikke funktionelle krav.

Krav	**Test**	**Forventet resultat**
IFK1	Webapplikation: Testes vha. Lighthouse og Chrome DevTools Perfomance. Tid måles fra formularindsendelse til visuel feedback. Native app: Måles manuelt med stopur, fra formularindsendelse til bekræftelse	Responstiden overstiger ikke 1 sekunder.
IFK2	Brugeren indtaster timer og tillæg og trykker "Beregn". Svartiden på lønberegningen måles.	Svartiden overstiger ikke 1 sekunder.
IFK3	Brug belastningstestværktøj k6 til at simulere 5000 API-requests pr. minut.	Systemet håndterer 5000 (100 gange 80%) requests pr. minut uden fejl.
IFK4	Kør kode-dækningstest via pipeline for at sikre ved hjælp af Github, at testdækningen er mere end 80%.	Efter commit change vises der i pipeline at testdækningen er 80%.
IFK5	Kør en sikkerhedsscanning i ZAP.	Sikkerhedsscoren er på maximum Medium i ZAP-skala.
IFK6	Brugeren skifter sproget i UI fra dansk til engelsk og omvendt. Brugergrænsefladen opdateres korrekt uden fejl.	Dansk tekst bliver til engelsk tekst og omvendt.
IFK7	Undersøg kontrastforhold i UI med `webaim.org/resources/contrastchecker`. Tekst har mindst 4.5:1 kontrastforhold til baggrund.	Kontrastforhold på mindst 4.5:1.

Tabel 8.2. Accepttest for ikke funktionelle krav, forsat.

Krav	**Test**	**Forventet resultat**
IFK8	Undersøg om den aktuelle deployment prisplan (pricing tier) for databasen inkluderer automatiske backups, der foretages mindst hver 24. time.	Der foretages backup hver 24. time eller oftere.
IFK9	Gennemgå systemets samtykkefunktioner samt adgang, rettelse og sletning af data. Sammenhold med GDPR.	Systemet overholder GDPR og nationale databeskyttelsesregler.
IFK10	Undersøg om backend og webapp deployment prisplaner inkluderer mulighed for softwareopdatering uden nedetid	Der understøttes softwareopdatering uden nedetid.

Resultater 9

Resultaterne i det følgende stammer fra accepttesten udført den 22. maj. Derudover blev US9 og IFK6 testet separat den 27. maj, da disse ikke var klar ved den første testgennemgang. Videodokumentation for US9 findes i bilag 7 og 9, og for IFK6 i bilag 8 og 9. Tabel 9.1 viser resultaterne for de funktionelle krav, mens tabel 9.2 viser resultaterne for de ikke-funktionelle krav.

Tabel 9.1. Accepttesttabel over funktionelle krav baseret på user stories.

Krav	**Observation**	**Vurdering**
US1	Lønberegning resultat vises.	OK
US2	5. måned vælges, afvigelse vises i rød.	OK
US3	Jobinfo og tillæg gemmes med bekræftelse.	OK
US4	Webfunktioner virker i browser.	OK
US5	Ikke implementeret.	FAIL
US6	Ikke implementeret.	FAIL
US7	Brugerprofil oprettes og login fungerer.	OK
US8	Vagt registreret 20. maj, gemmes korrekt.	OK
US9	En pdf downloades og den viser lønberegningen.	OK
US10	Viser feriepenge for valgt år med detaljer.	OK

Tabel 9.2. Accepttest for ikke-funktionelle krav

Krav	**Observation**	**Vurdering**
IFK1	Meget kort svartid, ca. 0,1–0,3 sekunder.	OK
IFK2	Svartiden på lønberegning er under 1 sekunder.	OK
IFK3	Ingen requests modtog statuskode 5xx.	OK
IFK4	Testdækningen på Github er mere end 80%.	FAIL
IFK5	En score på medium er opnået.	OK
IFK6	Sproget skifter på alle sider ved toggle knap.	OK
IFK7	Kontrastforhold er over 4.5:1.	OK
IFK8	Databasen er deployet i Azure med pricing tier: *Free – General Purpose – Serverless: Gen5, 2 vCores.* Backup sker hver 12. time.	OK
IFK9	Ikke implementeret.	FAIL
IFK10	Render og netlify sikrer automatiserede opdateringer uden nedetid.	OK

Diskussion 10

I dette kapitel reflekteres der over, hvad der fungerede, og hvad der ikke fungerede efter gennemførelsen af accepttesten, samtidig med at projektets opfyldelse af kravene og potentielle fejlkilder samt mulige løsninger diskuteres.

10.1 Opfyldelse af krav

Ud af de definerede user stories blev alle, **med undtagelse af US5 og US6**, implementeret og godkendt i accepttesten.

US5 – Der blev arbejdet på denne US, men gruppen nåede ikke i mål med implementeringen. Dette skyldes primært projektets *MoSCoW-prioritering*, hvor US5 havde lavere prioritet end mere kritiske funktioner og derfor blev nedprioriteret.

US6 – Da denne US var afhængig af US5, blev den hverken påbegyndt eller testet.

Herudover fejlede to ud af ti ikke-funktionelle krav i accepttesten: **IFK4** og **IFK9**.

IFK4 – Der blev ikke gennemført løbende testning af koden i projektets forløb. Først i projektets afslutning blev der udarbejdet unit tests, men på dette tidspunkt var det for sent at opnå målet om 80% testdækning, da fokus var flyttet til dokumentation. Gruppen har erkendt vigtigheden af at prioritere tests i selve implementeringsfasen i fremtidige projekter.

IFK9 – Gruppen havde begrænset viden om GDPR og databeskyttelseslovgivning, hvilket vanskeliggjorde vurderingen af overholdelse og de nødvendige tekniske og organisatoriske tiltag.

Set i forhold til projektets problemformulering er målet opfyldt: Det er nu muligt at identificere fejl i lønsedler ved at sammenligne den faktiske lønseddel med en beregnet løn.

Konklusion 11

Projektets formål var at udvikle en løsning, som kan hjælpe unge og studerende med at opdage fejl i deres lønudbetaling ved at sammenligne den faktiske lønseddel med en beregnet løn baseret på registrerede arbejdstimer og tillæg. På baggrund af accepttesten vurderes det, at projektets målsætninger i overvejende grad er opfyldt. Otte ud af ti user stories blev implementeret og godkendt, mens to ikke blev færdiggjort som følge af prioritering ud fra MoSCoW-metoden.

Ligeledes blev otte ud af ti ikke-funktionelle krav opfyldt. De to resterende krav fejlede grundet manglende teknisk testdækning samt begrænset viden om GDPR og databeskyttelse.

Det udviklede system giver brugeren mulighed for at registrere arbejdstimer og tillæg samt beregne den forventede løn. Funktionen til at identificere afvigelser mellem den udbetalte og den beregnede løn adresserer direkte både første og andet spørgsmål i problemformuleringen. Samtidig skaber løsningen grundlag for øget indsigt i egen økonomi og nemmere identifikation af fejl i lønsedler, hvilket understøtter tredje spørgsmål i problemformuleringen.

Projektet blev gennemført med en agil og iterativ tilgang inspireret af SCRUM og ASE-modellen. Arbejdet blev organiseret i sprints med løbende planlægning, implementering og evaluering. Denne metode har generelt fungeret tilfredsstillende og bidraget til fleksibilitet i forhold til ændrede krav undervejs i forløbet.

Fremtidigt arbejde 12

Finance Tracker er udviklet med fokus på skalerbarhed og fremtidig videreudvikling. Oplagt fremtidigt arbejde er at implementere de 4(2 IFK'er og 2 User-stories) krav som fejlede fra accepttesten.

Næste skridt er at integrere med SU og SKAT, så brugerens oplysninger hentes automatisk og manuel indtastning undgås.

Implementeringen af to-faktor-autentifikation (2FA) bidrage til øget datasikkerhed. Muligheden for nulstilling af glemt adgangskode via e-mail-validering vil ligeledes forbedre brugervenligheden og mindske risikoen for permanent kontotab.

Disse fremtidige tiltag øger systemets funktionalitet og sikkerhed, og også understøtte dets overordnede formål: at styrke unges økonomiske indsigt og evne til at identificere fejl i lønseddler.

Litteratur

[1] Fagbevægelsens Hovedorganisation, "Tjek din lønseddel: Hver anden finder fejl," 2021, set: 14-02-2025. [Online]. Available: https://fho.dk/blog/2021/11/19/tjek-din-loenseddel-hver-anden-finder-fejl/

[2] S. E. Jakobsen, "At have for få penge kan gøre dig fysisk syg," 2024, set: 25-02-2025,. [Online]. Available: https://videnskab.dk/kultur-samfund/at-have-for-faa-penge-kan-goere-dig-fysisk-syg/

[3] Skat, "Regn ud, hvad du får udbetalt," 2025, set: 21-05-2025. [Online]. Available: https://skat.dk/borger/aarsopgoerelse-forskudsopgoerelse-og-indkomst/forskudsopgoerelse/regn-ud-hvad-du-faar-udbetalt

[4] J. Nielsen, "Time scales of ux," 2024, set: 21-05-2025. [Online]. Available: https://jakobnielsenphd.substack.com/p/time-scale-ux

[5] Arman, "A comprehensive guide to api load testing," 2024, set: 21-05-2025. [Online]. Available: https://testfully.io/blog/api-load-testing/

[6] Microsoft, "Use code coverage to determine how much code is being tested," 2024, set: 21-05-2025. [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/test/using-code-coverage-to-determine-how-much-code-is-being-tested?view=vs-2022&tabs=csharp

[7] OWASP Foundation, "Owasp risk rating methodology," 2025, set: 21-05-2025. [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[8] Apple Developer, "Supporting multiple languages in your app," 2025, set: 21-05-2025. [Online]. Available: https://developer.apple.com/documentation/xcode/supporting-multiple-languages-in-your-app

[9] A. Tandon, "The ultimate guide to contrast ratio in ui design," 2023, set: 21-05-2025. [Online]. Available: https://medium.com/design-bootcamp/the-ultimate-guide-to-contrast-ratio-in-ui-design-2502d42442c4

[10] EJ Phillips, "How often do i really need to backup data?" 2024, set: 21-05-2025. [Online]. Available: https://www.proactive-info.com/blog/data-backup

[11] S. Y. Yacoob, "7 gdpr principles every business must follow for compliance," 2024, set: 21-05-2025. [Online]. Available: https://www.cookieyes.com/blog/gdpr-principles/

[12] Microsoft, "Automating software updates," 2025, set: 21-05-2025. [Online]. Available: https://techcommunity.microsoft.com/blog/coreinfrastructureandsecurityblog/automating-software-updates/323183

[13] Monday, "So how did monday.com come to be?" 2025, set: 21-05-2025. [Online]. Available: https://monday.com/p/about/

[14] Microsoft, "Sql server technical documentation," 2025, set: 21-05-2025. [Online]. Available: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17

[15] MongoDB, "What is mongodb?" 2025, set: 21-05-2025. [Online]. Available: https://www.mongodb.com/docs/manual/

[16] Oracle, "Mysql 8.4 reference manual," 2025, set: 21-05-2025. [Online]. Available: https://dev.mysql.com/doc/refman/8.4/en/

[17] Microsoft, “Asp.net core documentation,” 2025, set: 21-05-2025. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/

[18] Nodejs.org, “Node.js v24.1.0 documentation,” 2025, set: 21-05-2025. [Online]. Available: https://nodejs.org/docs/latest/api/

[19] Spring, “Spring boot,” 2025, set: 21-05-2025. [Online]. Available: https://spring.io/projects/spring-boot

[20] Microsoft, “.net multi-platform app ui documentation,” 2025, set: 21-05-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/

[21] Flutter, “Flutter documentation,” 2025, set: 21-05-2025. [Online]. Available: https://docs.flutter.dev/

[22] Meta Platforms, Inc., “React native,” 2025, set: 21-05-2025. [Online]. Available: https://reactnative.dev/

[23] ——, “React,” 2025, set: 21-05-2025. [Online]. Available: https://reactjs.org/

[24] Google, “What is angular?” 2025, set: 21-05-2025. [Online]. Available: https://angular.dev/overview

[25] Vue.js, “The progressive javascript framework,” 2025, set: 21-05-2025. [Online]. Available: https://vuejs.org/

[26] Microsoft, “C# language documentation,” 2025, set: 21-05-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/

[27] Mozilla, “Javascript guide,” 2025, set: 21-05-2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

[28] TypeScript, “Typescript documentation,” 2025, set: 21-05-2025. [Online]. Available: https://www.typescriptlang.org/docs/

[29] L. Gupta, “What is rest?” 2025, set: 21-05-2025. [Online]. Available: https://restfulapi.net/

[30] GraphQL, “Introduction to graphql,” 2025, set: 21-05-2025. [Online]. Available: https://graphql.org/learn/

[31] Microsoft, “Entity framework core,” 2025, set: 21-05-2025. [Online]. Available: https://learn.microsoft.com/en-us/ef/core/

[32] ZZZ Projects, “Welcome to learn dapper,” 2024, set: 21-05-2025. [Online]. Available: https://www.learndapper.com/

[33] S. Brown, “C4 model for visualising software architecture,” 2020, set: 21-05-2025. [Online]. Available: https://c4model.com/

[34] L. Stebelski, “Chen erd notation – what is it and how to use it?” 2020, set: 21-05-2025. [Online]. Available: https://vertabelo.com/blog/chen-erd-notation/

[35] R. Miller, “Code first to a new database - ef6,” https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database, 2020, set: 27-05-2025.

[36] Microsoft Azure, “Azure sql database single database pricing,” 2025, set: 27-05-2025. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/azure-sql-database/single/

[37] E. Williamson and J. Lengstorf, “A step-by-step guide: Deploying on netlify,” 2016, set: 27-05-2025. [Online]. Available: https://www.netlify.com/blog/2016/09/29/a-step-by-step-guide-deploying-on-netlify/

[38] Cucumber, “User story,” 2025, set: 21-05-2025. [Online]. Available: https://cucumber.io/docs/terms/user-story/

Bilagsoversigt

Bilag nr.	Navn
01	Procesbeskrivelse
02	Samarbejdskontrakt
03	Mødereferater
04	Teknisk Analyse
05	Frontend Web
06	Tests
07	US9-test Native-app
08	IFK6-test Native-app
09	US9 og IFK6 test Web-app
10	Design af Native-app
11	Kode