**Bilag 9**

**Frontend Web**

Finance Tracker

Semesterprojekt 4

Diplomingeniøruddannelsen i softwareteknologi

A blue logo with dolphins and a anchor

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Navn:** | **Studienummer:** |
| 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 |
|  |  |
| **Vejleder:** |  |
| Jung Min Kim |  |
|  |  |
| **Bilagsansvarlig:** |  |
| x |  |
|  |  |
| **Beskrivelse:** |  |

Dokumentation af frontend design og implementering for Web App.

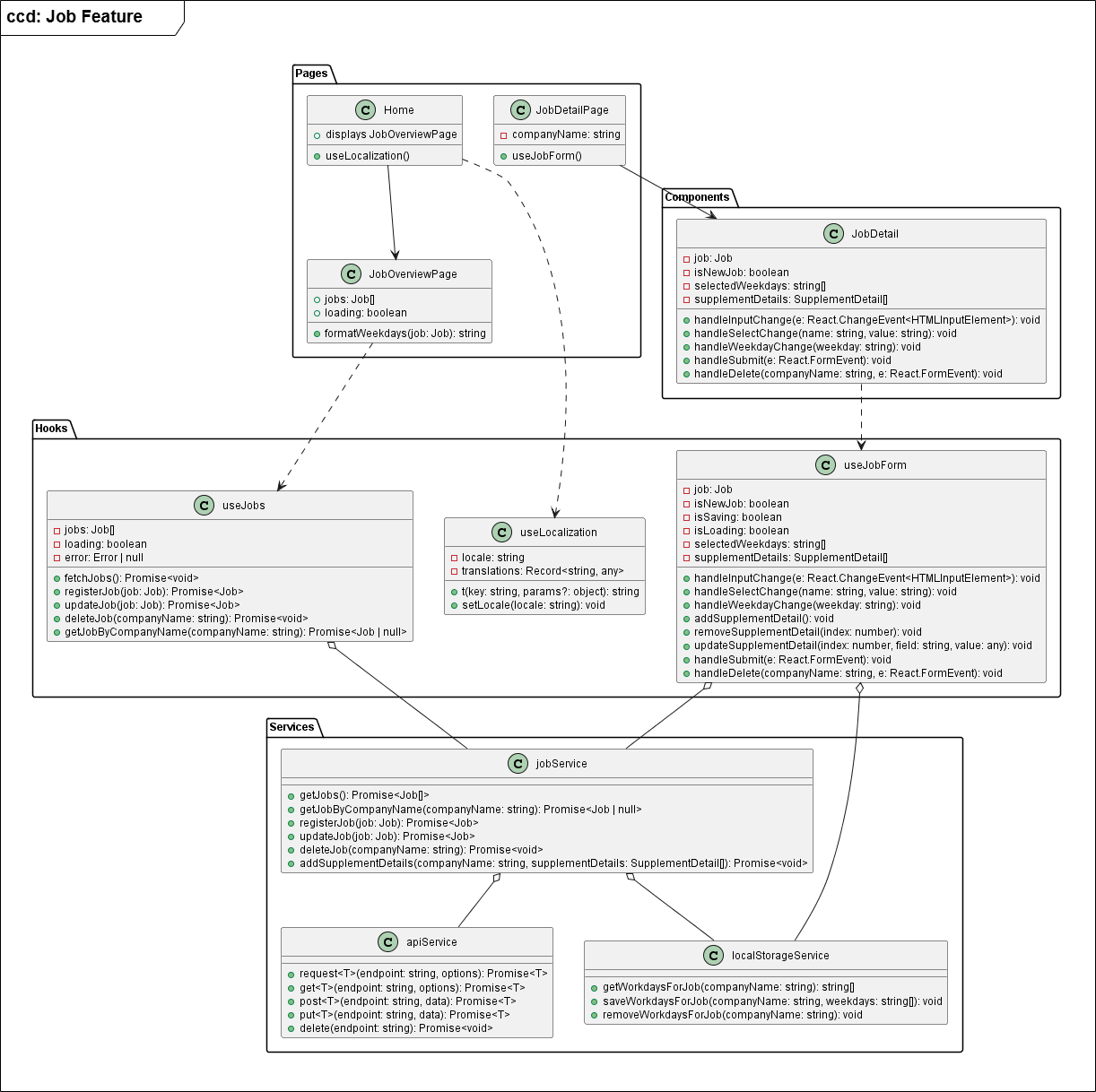
**Versionshistorik**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Dato** | **Initialer** | **Beskrivelse** |
| 0.1 | 20/05-2025 | CLB | Dokument oprettet |

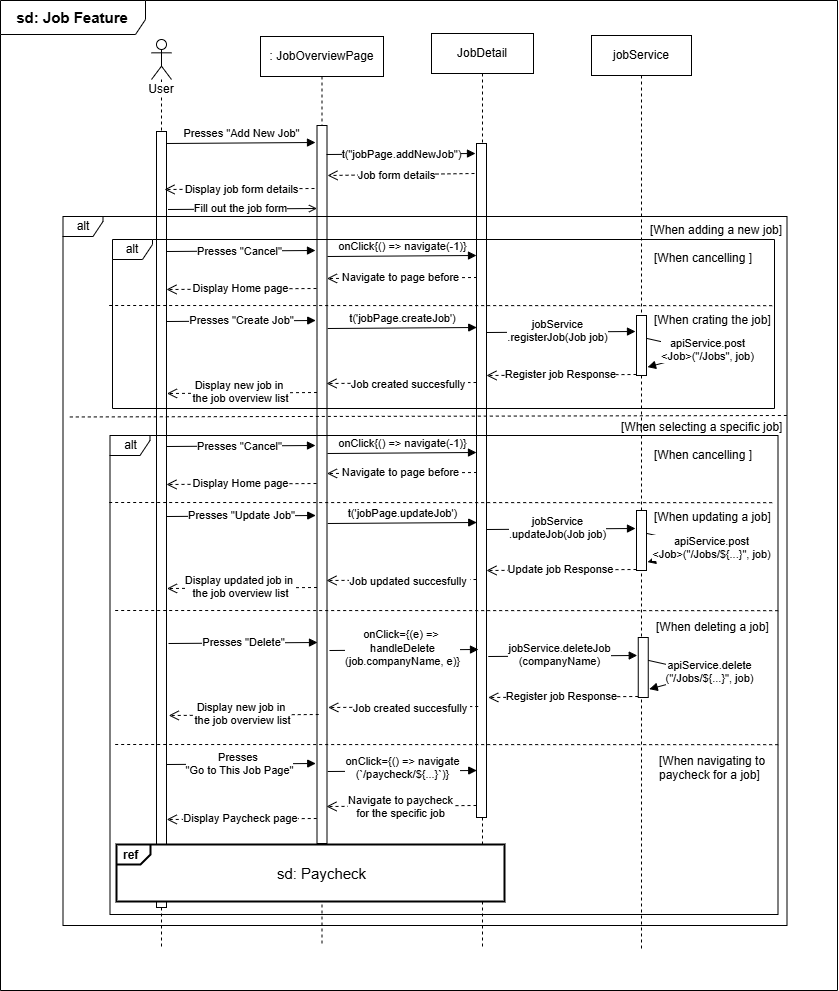
Indholdsfortegnelse

**No table of contents entries found.**

**Job Feature:**

  
*Figur 5: Job Feature Component Class Diagram*

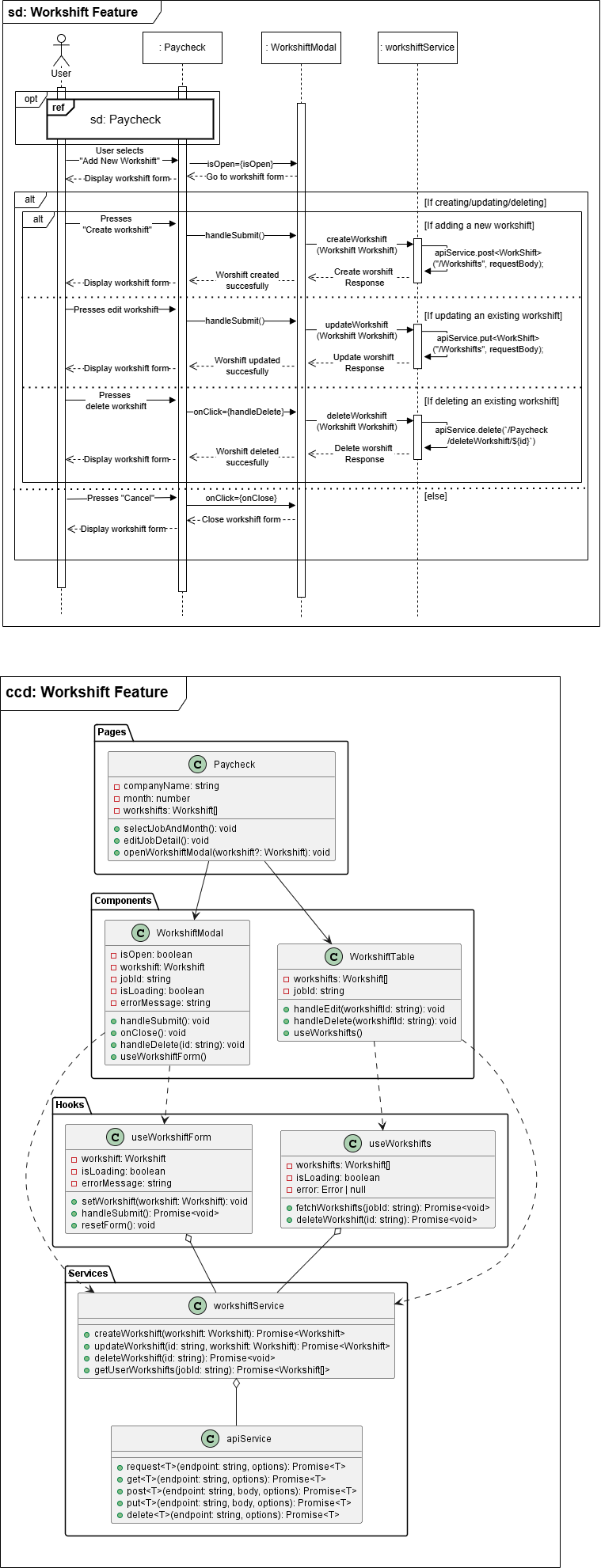
Diagrammet viser strukturen af en jobkomponent med sider, hooks, komponenter og tjenester. Siderne håndterer joboversigt og detaljer, mens hooks som useJobs og useLocalization styrer jobdata og lokaliseringsindstillinger. Komponenterne inkluderer jobdetaljer og en jobformular, og tjenesterne håndterer API-opkald og lokal lagring.

  
*Figur 6: Job Feature Sequens Diagram*

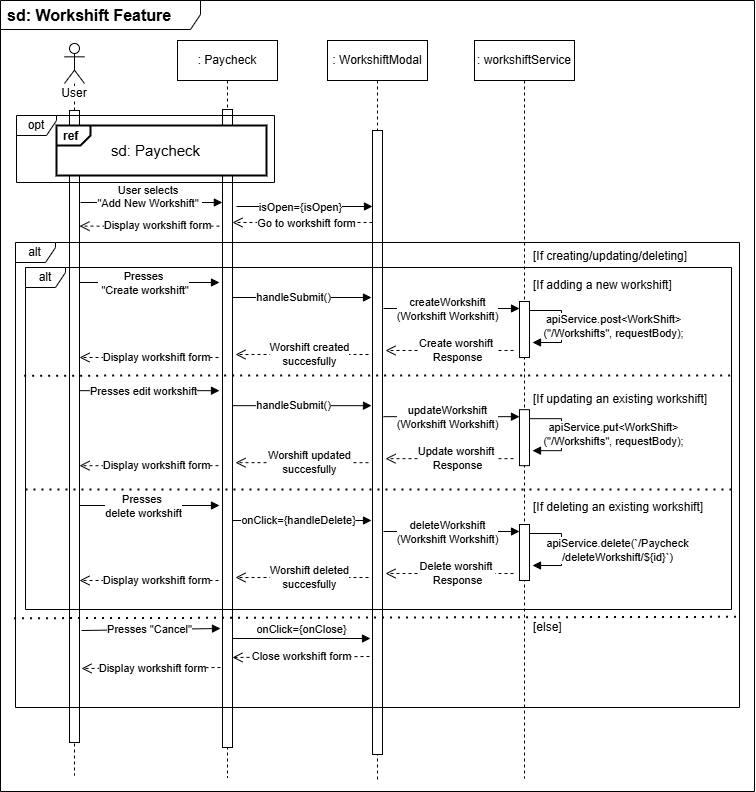
Diagrammet viser interaktionerne for jobs, hvor brugeren kan tilføje, opdatere, eller slette job. Når brugeren trykker på "Create Job", oprettes et job via JobService.registerJob(), og et succesfuldt svar returneres. Ved opdatering af job kaldes JobService.updateJob(), og ved sletning bruges JobService.deleteJob(). Brugeren kan også navigere til en lønseddel for et specifikt job, hvilket leder til Paycheck-siden. Alle handlinger involverer navigation mellem sider og kald til jobtjenesterne.

Job-funktionaliteten er en komplet jobadministrationsløsning med CRUD-operationer og supplement-detaljer, designet med en todelt struktur: JobOverviewPage viser en tabeloversigt med sortering og filtrering, mens JobDetail håndterer formular-baserede jobdetaljer. Systemet følger SOLID-principperne med adskilt UI og datahåndtering, bruger Custom Hooks (useJobs, useJobForm) og Composition pattern for genbrug af UI-komponenter samt Repository, Observer, Strategy, Composite og Factory patterns for fleksibel datahåndtering, realtidsopdateringer, jobtyper og skatteberegning.

**Workshift Feature:**

  
*Figur 7: Job Feature Component Class Diagram}*

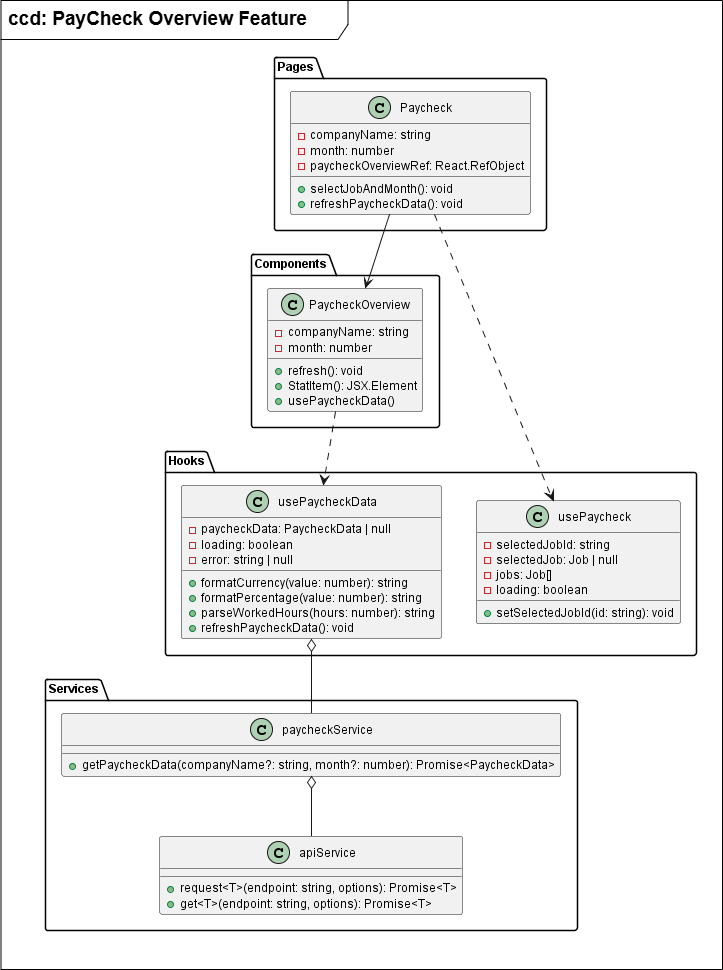
Diagrammet viser en komponentstruktur til håndtering af arbejdstimer. Siderne inkluderer en "Paycheck" side, der viser oplysninger om arbejdstimer og månedsløn. Komponenterne omfatter en modal og tabel til visning og håndtering af arbejdstimer, som interagerer med hooks som useWorkshiftForm og useWorkshifts for at håndtere data og brugerinput. Tjenesterne, som WorkshiftService, håndterer oprettelse, opdatering, sletning og hentning af arbejdstidsdata gennem API-kald.

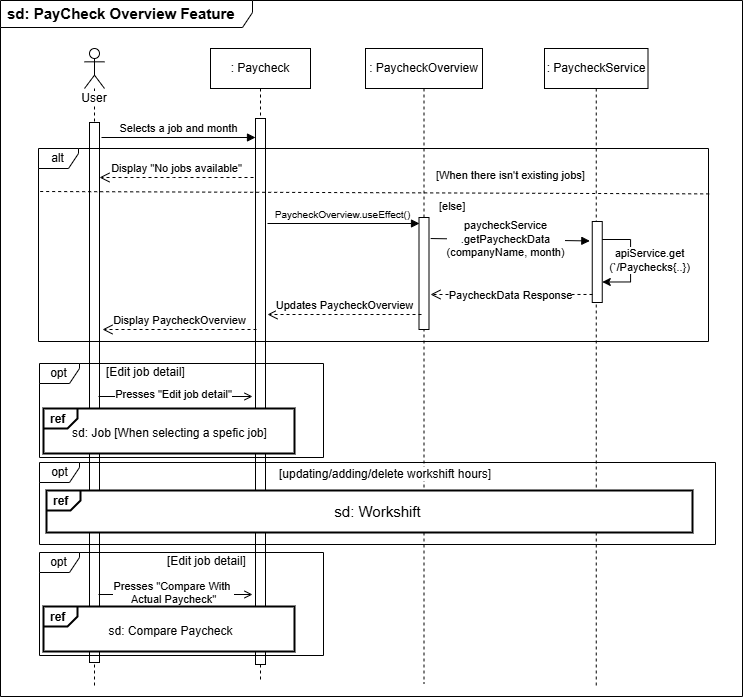
  
*Figur 8: Workshift Feature Sequance Diagram*

Diagrammet viser sekvensen af handlinger til håndtering af arbejdstimer (workshifts). Brugeren kan vælge at tilføje, opdatere eller slette arbejdstimer gennem en modal. Når en bruger trykker på "Create workshift", oprettes en ny arbejdstid via apiService.post. Hvis brugeren redigerer en eksisterende arbejdstid, opdateres den via apiService.put, og hvis de sletter en arbejdstid, sker det med apiService.delete. Alle handlinger involverer kommunikation med workshiftService, som håndterer API-opkald for at oprette, opdatere eller slette arbejdstimer. Hvis brugeren afbryder handlingen, lukkes formularen.

Workshift-funktionalitenten er designet med en todelt struktur: WorkshiftTable viser en tabel oversigt over arbejdsvagt med data, start/slut tid og arbejdstimer, mens WorkshiftModal håndterer oprettelse og redigering af individuelle vagt. Systemett følger Single Responsibility Principle med adskilt UI og datahåndtering, bruger Custom hoks til state management, og implementerer Observer pattern gennem toast-notifiaktioner og bekræftelsesdialogue. Det følger også Strategy pattern for tidsberegninger, Fatory pattern for workshift-objekter, og Command pattern for CRUD-operationer.

**Paycheck Overview Feature:**

  
*Figur 9: Paycheck Overview Feature Component Class Diagram*

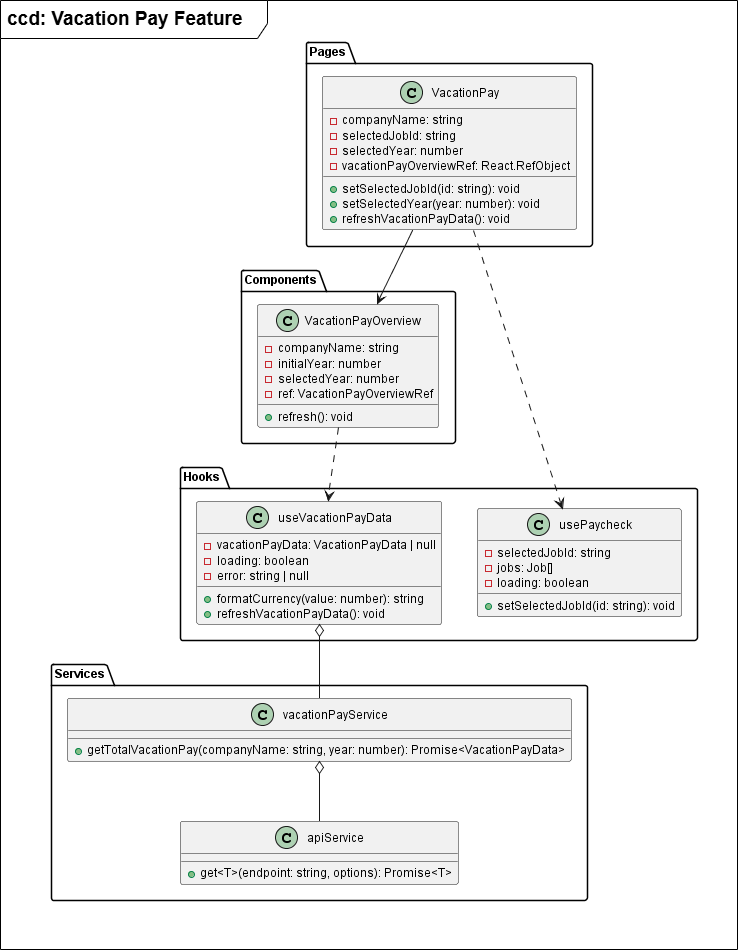
  
*Figur 10: Paycheck Overview Feature Sequence Diagram*

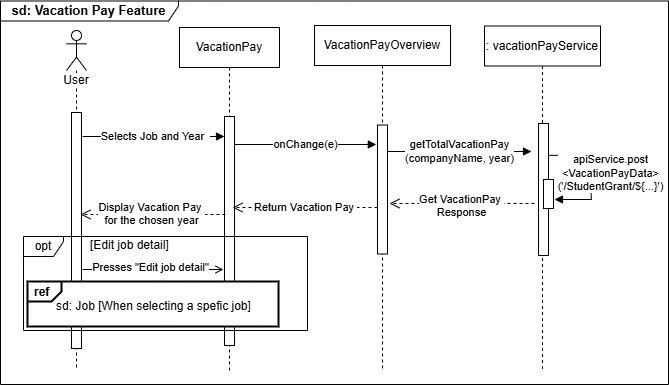
Paycheck-funktionaliteten er designet med en todelt struktur: PaycheckOverview viser en grid-baseret oversigt over løndetaljer med ikoner og statistikker, mens Paycheck håndterer job-valg og månedsfiltrering. Systemet følger Single Responsibility Principle med adskilt UI og datahåndtering, bruger Custom Hooks (usePaycheck, usePaycheckData) til state management, og implementerer Observer pattern gennem toast-notifikationer. Det følger også Strategy pattern for valuta- og procentformatering, Factory pattern for paycheck-objekter, og Command pattern for data-hentning og opdateringe

Diagrammet viser sekvensen af handlinger, når en bruger sammenligner lønsedler. Når brugeren vælger et job og en måned, kontrollerer systemet om eksisterende jobdata findes. Hvis der ikke findes job, vises en meddelelse om, at ingen job er tilgængelige. Hvis jobdata findes, hentes løndata fra paycheckService, som kalder API-tjenesten for at få de nødvendige oplysninger. Derefter beregnes forskellen mellem de genererede og manuelle løndata og vises som resultat.

Paycheck Compare-funktionaliteten er designet med en tre-kolonne struktur der sammenligner manuelle input med genererede løndetaljer og viser forskelle. Systemet følger Single Responsibility Principle med adskilt UI og beregningslogik, bruger Custom Hooks (usePaycheckCompare) til state management og beregninger, og implementerer Observer pattern gennem realtids opdateringer. Det følger også Strategy pattern for forskellige beregningsmetoder, Template Method pattern for sammenligningslogikken, og Composite pattern for opbygning af sammenligningselementer.

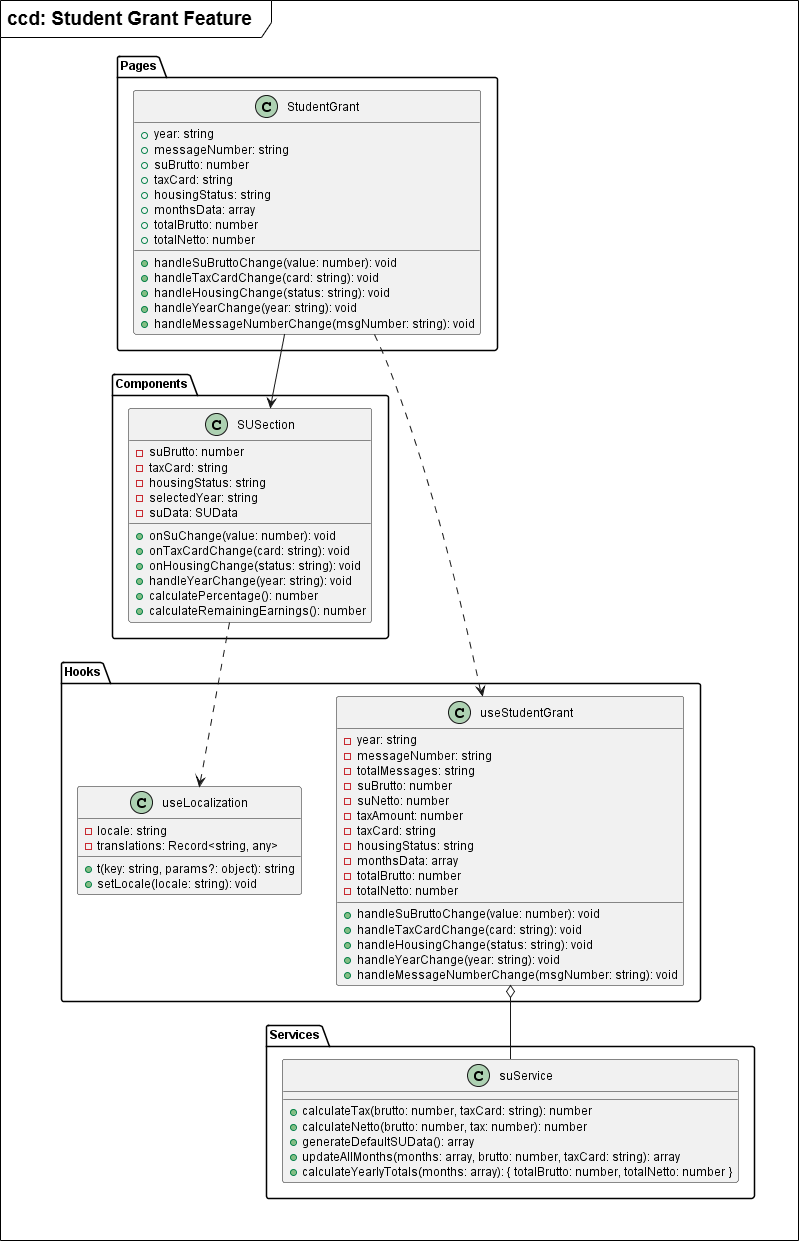
**Vacation Pay Feature:**

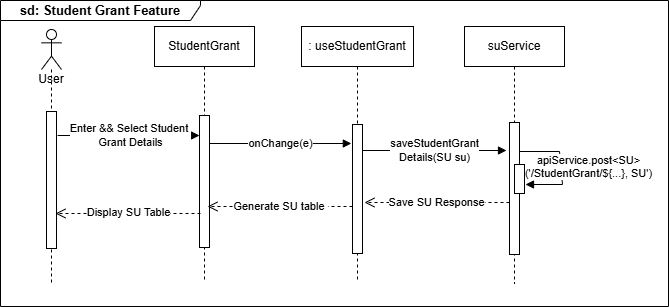
  
*Figur 13: Vacation Pay Feature Component Class Diagram*

*Figur 14: Vacation Pay Sequence Diagram*

Feriepenge-funktionaliteten er designet med en oversigtsvisning der viser total feriepenge for et valgt år, med mulighed for at skifte mellem forskellige år. Systemet følger Single Responsibility Principle med adskilt UI og datahåndtering, bruger Custom Hooks (useVacationPayData) til state management, og implementerer Observer pattern gennem realtids opdateringer. Det følger også Strategy pattern for feriepengeberegninger, Factory pattern for feriepenge-objekter, og Composite pattern for opbygning af årsoversigten.

**Student Grant Feature:**

  
*Figur 15: Student Grant Feature Component Class Diagram*

*Figur 16: Student Grant Feature Sequence Diagram*

Student grant (SU)-funktionaliteten er designet med en tabelbaseret oversigt over månedlige SU-beløb og en konfigurationssektion til at justere parametre. Systemet følger Single Responsibility Principle med adskilt UI og beregningslogik, bruger Custom Hooks (useStudentGrant) til state management, og implementerer Observer pattern gennem realtids opdateringer. Det følger også Strategy pattern for forskellige beregningsmetoder baseret på boligstatus og skattekort, Factory pattern for SU-objekter, og Composite pattern for opbygning af månedlige beregninger.