**Bilag 11**

**Tests**

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:** |  |
| APR |  |
|  |  |
| **Beskrivelse:** |  |

Dokumentation for tests; unittests, integrationstest, modultests.

**Versionshistorik**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Dato** | **Initialer** | **Beskrivelse** |
| 0.1 | 20/05-2025 | CLB | Dokument oprettet |
| 0.2 | 21/05-2025 | APR | Test dokumenteret |

Indholdsfortegnelse

[Integrationsplan 2](#_Toc180551781)

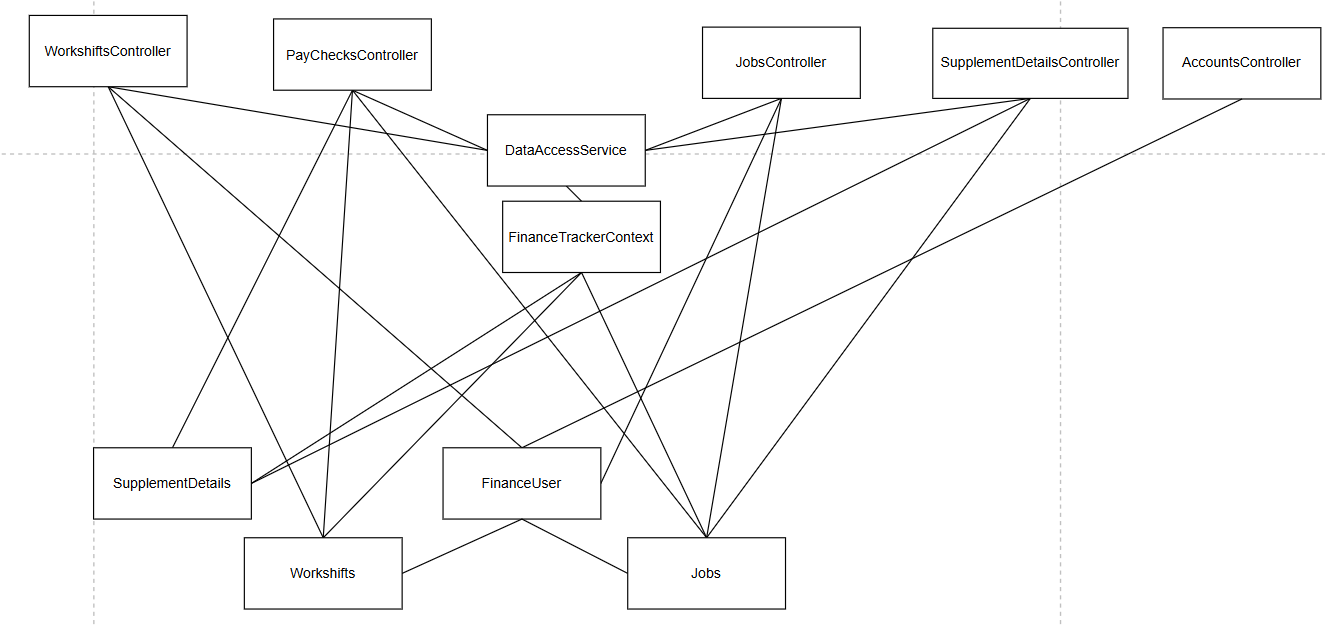
# Integrationsplan

For at sikre systemets funktionalitet og stabilitet er der anvendt en kombination af integrationstests og unittests. Testene har til formål at afdække fejl tidligt, validere grænseflader og sikre korrekt adfærd i både isolerede og integrerede komponenter. Strategien omfatter en struktureret integrationsplan samt målrettet brug af testdesignteknikker i enhedstestene.

## Integrationstest

Til integrationstest af backenddelen anvendes **Bottom-up Integration**. Denne strategi gør det muligt at teste komponenter fra de laveste niveauer og gradvist opbygge systemet, mens grænseflader mellem komponenterne verificeres undervejs. Hver ny komponent tilføjes først, når dens afhængigheder er korrekt implementeret og testet, hvilket bidrager til stabilitet og forenkler fejlfinding.

Det tilhørende **dependency tree** viser, hvordan systemet er opbygget i lag. Nederst findes FinanceTrackerContext, som anvendes af de generiske repositories (DataAccessService<T>), der videre benyttes af controllerlaget. Domænemodeller som Jobs, WorkShifts og SupplementDetails fungerer som datagrundlag uden forretningslogik og kræver derfor ikke selvstændige tests. Controllerne – herunder JobsController, WorkshiftsController, SupplementDetailsController, PaychecksController og AccountsController – udgør øverste lag i arkitekturen og kalder underliggende lag via dependency injection.



Figur 1, Dependency Tree over de forskellige moduler i systemets backend. Her kan det ses hvordan de bliver opdelt i forskellige lag: Controller, DataAccessService, FinanceTrackerContext og Model klasserne helt nederst.

Der er udarbejdet en **bottom-up integrationsplan**, hvor testsekvensen starter med databasen (FinanceTrackerContext), efterfulgt af repositories og domæneobjekter, og afsluttes med integration af controllerlaget. Komponenter fra ASP.NET Core Identity såsom UserManager<FinanceUser> og SignInManager<FinanceUser>, som anvendes i AccountsController, er ikke en del af selve integrationstestplanen.

Identity-laget er bevidst udeladt fra integrationstesten, da det bygger på Microsofts ASP.NET Core Identity-framework, som anses for værende veletableret og grundigt testet. I stedet anvendes stubs og mocks for afhængigheder til Identity i integrationstests, mens den fulde funktionalitet – herunder login og brugeroprettelse – dækkes i systemtesten, hvor integrationen vurderes samlet.

Dette valg reducerer kompleksiteten i integrationstestforløbet og fokuserer testindsatsen på de komponenter, der er udviklet specifikt til Finance Tracker.

Tabel 1, Tabeloversigt over integrationsplanen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | FinanceTracterContext | Domæne-objekter | DataAccesService | Controllers |
| 1 | T |  |  |  |
| 2 | X | T |  |  |
| 3 | X | X | T |  |
| 4 | X | X | X | T (Jobs / Workshifts / SupplementDetails / Paychecks / Accounts) |

## Unittest

Til unit tests er der anvendt en systematisk tilgang med fokus på både funktionel og strukturel testdækning. Testene omfatter hovedsageligt egne metoder og services, herunder DataAccessService<T>, beregninger i PaychecksController og valideringslogik i controllerne.

Der er anvendt **ZOMBIE-huskereglen** som rettesnor for at sikre, at tests dækker centrale fejltyper. Dette inkluderer eksempelvis tests af, hvordan repository-metoder håndterer null-værdier (Zero), om metoder kaldes i korrekt rækkefølge (Order), og om uautoriseret adgang håndteres med passende fejlkoder (Authorization og Broken constraints).

Inputvalidering er testet ved brug af **Equivalence Partitioning (EP)** og **Boundary Value Analysis (BVA)**, særligt i forbindelse med DTO’er som RegisterDTO og WorkShiftDTO. Her er der opstillet både gyldige og ugyldige eksempler på f.eks. e-mail-formater, datoer og tidsintervaller for at sikre dækkende tests. Testdata inkluderer både typiske cases, grænseværdier og klart ugyldige input for at afdække potentielle valideringsfejl.

Derudover anvendes både **Black Box-** og **White Box-teststrategier** afhængigt af konteksten. Til endpoints og controller-metoder benyttes Black Box-test, hvor fokus ligger på input-output uden kendskab til den interne implementation, f.eks. ved test af HTTP-responskoder. Til logik med beregninger, som eksempelvis i lønberegning og tillæg, anvendes White Box-test, hvor branches og edge cases dækkes ud fra den faktiske kode, herunder beregning af skat, feriepenge og arbejdstimer.

En bevidst kombination af flere testdesignteknikker sikrer høj testdækning, robusthed og kvalitet i de testede komponenter. Unittests er implementeret med xUnit og indgår som en integreret del af den automatiserede CI-pipeline.