**Bilag 6**

**Teknisk analyse**

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:** |  |
| Alle |  |
|  |  |
| **Beskrivelse:** |  |

Indeholder den tekniske analyse.

**Versionshistorik**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Dato** | **Initialer** | **Beskrivelse** |
| 0.1 | 20/05-2025 | CLB | Dokument oprettet |
| 1.0 | 21/05-2025 | CLB | Indsat fra rapport, og rettelser som kilder og tabeller. |

Indholdsfortegnelse

[Indledning 4](#_Toc198728276)

[Overordnet arkitektur 4](#_Toc198728277)

[Backend 5](#_Toc198728278)

[ASP.NET 5](#_Toc198728279)

[Node.JS 5](#_Toc198728280)

[Spring Boot 6](#_Toc198728281)

[Frontend – Web App 8](#_Toc198728282)

[AngularJS 8](#_Toc198728283)

[ReactJS 8](#_Toc198728284)

[VueJS 9](#_Toc198728285)

[Frontend - Native App 11](#_Toc198728286)

[.NET MAUI 11](#_Toc198728287)

[Flutter 11](#_Toc198728288)

[React Native 12](#_Toc198728289)

[Programmeringssprog 14](#_Toc198728290)

[Frontend – Webapplikation 14](#_Toc198728291)

[JavaScript 14](#_Toc198728292)

[TypeScript 14](#_Toc198728293)

[Frontend Native applikation programmeringssprog 15](#_Toc198728294)

[Backend programmeringssprog 15](#_Toc198728295)

[Database 18](#_Toc198728296)

[MySQL 18](#_Toc198728297)

[MongoDB 18](#_Toc198728298)

[MsSQL 19](#_Toc198728299)

[Django 20](#_Toc198728300)

[Server integration med frontend 22](#_Toc198728301)

[REST API 22](#_Toc198728302)

[GraphQL 22](#_Toc198728303)

[WebSockets 23](#_Toc198728304)

[Object-Relational Mapping(ORM) 25](#_Toc198728305)

[Entity framework Core 25](#_Toc198728306)

[Dapper 26](#_Toc198728307)

[Overordnet teknologivalg 28](#_Toc198728308)

[Referenceliste 30](#_Toc198728309)

# Indledning

For at sikre en velfunderet teknisk løsning gennemføres en dybdegående analyse af potentielle teknologier, som kan anvendes til udviklingen af systemets hovedkomponenter: Frontend (web og native), Backend og Database. Formålet med analysen er at skabe et beslutningsgrundlag, der matcher både funktionelle og ikke-funktionelle krav som skalerbarhed, ydeevne, vedligeholdelse og sikkerhed.

Hver delkomponent er analyseret i forhold til udvalgte teknologier, og fordele og ulemper vurderes systematisk. Valg af teknologi er truffet på baggrund af kravspecificerede mål, iterativ udviklingsmetodik og teamets kompetencer.

# Overordnet arkitektur

Systemet er designet med en tredelt arkitektur bestående af:

**Web-App:** En webbaseret platform, hvor brugere kan indtaste arbejdstimer, se løndata og interagere med backend via API’er.

**Native-App:** En applikation til Windows med samme funktionalitet som web-appen, der også kommunikerer med backend via API’er.

**Backend:** En API-baseret backend, der håndterer databehandling, brugergodkendelse og integration med databasen.

# Backend

Backend-komponenten i Finance Tracker udgør systemets centrale forretningslogik og ansvarlige for databehandling, validering, autentificering og kommunikation med frontend og database. Det er derfor essentielt at vælge en teknologi, der er robust, skalerbar og let at integrere med andre systemkomponenter. Tre relevante teknologier er blevet analyseret: ASP.NET, Node.js og Spring Boot. Analysen er baseret på kriterier som ydeevne, udviklingsvenlighed, skalerbarhed og gruppens tekniske forudsætninger.

## **ASP.NET**

ASP.NET er et open-source webframework udviklet af Microsoft og en del af .NET-platformen[1]. Det anvendes til udvikling af webapplikationer, API'er og services. Frameworket understøtter en række NuGet-pakker, som giver adgang til avanceret funktionalitet som f.eks. sikkerhed (autentificering og autorisation), logging, og ORM-løsninger (fx Entity Framework).

**Fordele**

* ASP.NET understøttes direkte i undervisningen på 4. semester, hvilket betyder, at projektgruppen har forhåndskendskab til både frameworket og C#.
* Tæt integration med Visual Studio og Azure gør det nemt at sætte CI/CD op og implementere DevOps-principper.
* Et modent økosystem og omfattende dokumentation bidrager til hurtig fejlfinding og stabil udvikling.

**Ulemper**

* ASP.NET kan opleves som tungt og overstruktureret til mindre projekter med hurtige iterationskrav.
* Frameworket er tæt knyttet til Microsofts økosystem, hvilket kan reducere fleksibiliteten i forhold til platforme og deployeringsmiljøer.

## **Node.JS**

Node.js er et event-drevet, non-blocking runtime-miljø baseret på JavaScript og V8-motoren. Det er især velegnet til I/O-intensive applikationer og realtidskommunikation.

**Fordele**

* Høj ydeevne ved håndtering af mange samtidige forbindelser, hvilket matcher kravene til systemets API-endepunkter (op til 5000 requests/minut).
* Cross-platform support og stærkt open-source community gør det let at finde moduler og værktøjer.
* Fleksibelt og let vægtigt set up, særligt velegnet til hurtig prototyping og iteration.

**Ulemper**

* Mindre struktureret end f.eks. ASP.NET og kan derfor være mere udfordrende at vedligeholde i større projekter.
* Øget risiko for teknisk gæld ved brug af mindre modne biblioteker fra npm.

## **Spring Boot**

Spring Boot er et modulært og konfigurationslet Java-baseret framework, som bygger på Spring-økosystemet. Det er kendt for høj testbarhed og enterprise-grade skalerbarhed.

**Fordele**

* Understøtter Dependency Injection og løs kobling, hvilket forbedrer testbarhed og genanvendelighed.
* Spring Security muliggør robust autentificering og autorisation med minimal opsætning.

**Ulemper**

* Projektgruppen har ikke erfaring med Java eller Spring-økosystemet, hvilket indebærer en betydelig indlæringskurve.
* Udviklingsmiljøet og syntaksen kan virke fremmed og komplekst for C#/JavaScript-vante udviklere.

| **Teknologi** | **Fordele** | **Ulemper** |
| --- | --- | --- |
| **ASP.NET** | - Kendt af gruppen (bruges i undervisningen)  - Mange NuGet-pakker (autentificering, ORM)  - God integration med Microsoft-værktøjer | - Føles tungt til små projekter  - Begrænset til Microsoft-miljøer |
| **Node.js** | - Høj ydeevne til I/O og realtid  - Letvægts og fleksibelt  - Open source og stort community | - Mindre struktureret til større løsninger |
| **Spring Boot** | - Moderne, fleksibelt Java-framework  - God til testbarhed og sikkerhed via Spring Security | - Gruppen har ingen erfaring med Java  - Høj læringskurve |

Tabel 1, Backend teknologier

**Teknologivalg**

På baggrund af ovenstående analyse er ASP.NET valgt som backend-teknologi. Valget begrundes med følgende:

* Gruppen har eksisterende kompetencer og modtager samtidig undervisning i ASP.NET og C#, hvilket minimerer læringskurven og øger udviklingseffektiviteten.
* ASP.NET muliggør let integration med SQL Server og øvrige Microsoft-teknologier, som allerede er valgt til systemets database og hostingmiljø.
* Frameworket tilbyder høj sikkerhed, modenhed og understøttelse af enterprise-funktionalitet, hvilket er væsentligt for et finansielt system.

# Frontend – Web App

Frontend-komponenten i Finance Tracker skal sikre en brugervenlig, responsiv og effektiv brugeroplevelse til registrering af arbejdstimer og visning af løndata. For at imødekomme krav om hurtig responstid, komponentbaseret struktur og skalerbarhed, er tre moderne frontend-teknologier blevet analyseret: AngularJS, ReactJS og VueJS.

## **AngularJS**

AngularJS er et TypeScript-baseret frontend-framework udviklet af Google. Det anvender en struktureret arkitektur med deklarativ templating og stærk support til test og modulær opbygning.

**Fordele**

* Velegnet til store og komplekse applikationer med mange afhængigheder.
* Indbygget Dependency Injection understøtter høj testdækning og genanvendelighed.
* Mange funktioner og moduler er integreret direkte i frameworket, hvilket reducerer behovet for eksterne biblioteker.

**Ulemper**

* Stejl indlæringskurve og omfattende konventioner kræver længere oplæringstid.
* Kan opleves som for omfattende til mindre projekter.
* Potentielt længere load-tider grundet frameworkets kompleksitet.

## **ReactJS**

ReactJS er et komponentbaseret JavaScript-bibliotek udviklet af Meta, som anvender en virtuel DOM til hurtig rendering og UI-opdateringer[5].

**Fordele**

* Komponentbaseret arkitektur muliggør genbrug og modularisering, hvilket effektiviserer udviklingen.
* Virtuel DOM sikrer hurtig brugerinteraktion, hvilket understøtter ikke-funktionelle krav om <2 sek. svartid.
* Stort økosystem, aktivt community og bredt udbud af tredjepartsbiblioteker.
* Let integration med backend-API’er via fetch/Axios og understøttelse af moderne routing og state management.

**Ulemper**

* Håndtering af kompleks tilstand (state) kræver ekstra opsætning (Redux, Zustand mv.).
* Stejl indlæringskurve – især ved avanceret brug og vedligeholdelse.
* Hyppige ændringer og versionering i React-økosystemet kan medføre vedligeholdelsesudfordringer.

## **VueJS**

VueJS er et progressivt JavaScript-framework, som kombinerer ideer fra både Angular og React, men med fokus på simplicitet og lav indlæringskurve.

**Fordele**

* Enkel og intuitiv syntaks gør det let at lære og implementere.
* Reaktiv datamodel og kompakt struktur fremmer hurtig udvikling.
* Lille filstørrelse og høj performance giver kortere svartider.
* Understøtter indbygget state management via Vuex.

**Ulemper**

* Mindre community og økosystem end React og Angular.
* Færre tredjepartsmoduler kan føre til øget udviklingstid i komplekse projekter.
* Potentielle integrationsproblemer ved enterprise-skala projekter.

| **Teknologi** | **Fordele** | **Ulemper** |
| --- | --- | --- |
| **AngularJS** | - Struktureret arkitektur  - God testunderstøttelse  - Indbyggede moduler | - Stejl indlæringskurve  - Tungt til små projekter  - Længere load-tider |
| **ReactJS** | - Komponentbaseret og fleksibelt  - Hurtig DOM-opdatering  - Stort økosystem | - Kompleks state management  - Stejl kurve  - Hyppige ændringer |
| **VueJS** | - Let at lære  - Høj ydeevne  - Indbygget state management | - Begrænset support  - Færre biblioteker  - Svært at skalere op |

Tabel 2, Frontend - Web App, teknologier

**Teknologivalg**

ReactJS vælges som frontend-teknologi, da det kombinerer høj ydeevne, fleksibilitet og genanvendelighed. Den komponentbaserede arkitektur egner sig til modulopdeling af funktioner som tidsregistrering, lønsammenligning og realtidsopdateringer. Frameworkets brede anvendelse i industrien, kombineret med god dokumentation og integration med backend, gør det til et robust valg for dette projekt.

# Frontend - Native App

Udviklingen af en native app til Finance Tracker skal sikre en glidende brugeroplevelse med hurtig respons, offline-muligheder og integration med backend. Følgende tre cross-platform frameworks er analyseret: .NET MAUI, Flutter og React Native. Der lægges vægt på ydeevne, læringskurve, integration og fremtidig vedligeholdelse.

## **.NET MAUI**

MAUI (Multi-platform App UI) er Microsofts moderne cross-platform framework, der tillader udvikling af mobil- og desktopapplikationer på tværs af Android, iOS, Windows og macOS med én samlet kodebase i C#[2].

**Fordele**

* Projektgruppen undervises i både .NET og MAUI, hvilket reducerer indlæringsbehov og sikrer hurtigere produktivitet.
* MAUI tilbyder tæt integration med Azure og Visual Studio, hvilket letter deployment og DevOps.
* Native-lignende performance uden brug af JavaScript bridge.

**Ulemper**

* Som et nyere framework har MAUI endnu et mindre modent økosystem og begrænset community sammenlignet med konkurrenterne.
* Længere build-tider kan reducere feedback-hastigheden under udvikling.

## **Flutter**

Flutter er et open-source UI-framework udviklet af Google. Det anvender Dart og leverer applikationer med ensartet brugergrænseflade og høj ydeevne.

**Fordele**

* Ensartet og tilpasselig UI på tværs af platforme via en kraftfuld rendering engine.
* Velegnet til komplekse visualiseringer og dynamiske brugergrænseflader – ideelt til lønsammenligninger og datavisualisering.

**Ulemper**

* Dart er et relativt ukendt sprog for de fleste udviklere, og det øger indlæringsbehovet.
* Applikationer udviklet i Flutter har typisk en større binærstørrelse, hvilket kan være en ulempe på mobile enheder.

## **React Native**

React Native gør det muligt at udvikle mobile applikationer med JavaScript/TypeScript og deler arkitektoniske principper med ReactJS.

**Fordele**

* Kan potentielt dele komponenter og forretningslogik med den webbaserede frontend, hvilket mindsker udviklingstiden.
* Stort community og bredt udvalg af tredjepartsmoduler.

**Ulemper**

* Anvender en JavaScript bridge, hvilket kan føre til performanceudfordringer, særligt ved mange animationer og tunge dataflows.
* Debugging og vedligeholdelse kan være mere kompleks pga. lagdeling og tredjepartsafhængigheder.

| **Teknologi** | **Fordele** | **Ulemper** |
| --- | --- | --- |
| **.NET MAUI** | - God integration med .NET og Azure  - Native performance | - Relativt nyt og mindre community  - Længere build-tider |
| **Flutter** | - Høj ydeevne  - Ensartet UI på tværs af platforme | - Dart er ukendt  - Større app-størrelse |
| **React Native** | - Deler logik med ReactJS  – Stor community | - Lavere performance  - Kompleks debugging |

Tabel 3, Native app teknologier

**Teknologivalg**

.NET MAUI vælges som platform til native applikationen, da det giver den mest hensigtsmæssige balance mellem ydeevne, udviklingsproduktivitet og teknologisk synergi med resten af systemet. MAUI muliggør brug af kendt sprog (.NET/C#), integrerer gnidningsfrit med backend og Azure, og understøttes desuden i undervisningen.

# Programmeringssprog

Valget af programmeringssprog har direkte indflydelse på udviklingstid, vedligeholdelse, fejlforebyggelse og mulighed for genbrug på tværs af systemets lag. Derfor er sprogvalget til både frontend og backend analyseret og udvalgt med fokus på stabilitet, skalerbarhed og udviklerkompetencer.

## **Frontend – Webapplikation**

Den webbaserede frontend i Finance Tracker er baseret på ReactJS, som typisk udvikles med JavaScript eller TypeScript (via JSX/TSX).

### **JavaScript**

JavaScript er et dynamisk, fortolket sprog, der kører direkte i browseren[7]. Det er velegnet til hurtig udvikling og understøtter asynkrone operationer, som er vigtige i et API-drevet system.

**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 i større projekter.

### **TypeScript**

TypeScript er et statisk typet supersæt af JavaScript og skal transpileres til JavaScript før eksekvering[6].

**Fordele**

* Tilføjer statisk typesikkerhed, som reducerer fejl og forbedrer refaktorering.
* 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 som primært sprog til frontend-webapplikationen, da det forbedrer datavalidering, vedligeholdelse og skalerbarhed. I kombination med HTML, CSS og Tailwind CSS opnås et moderne, responsivt og brugervenligt design med høj udviklingseffektivitet og genbrug.

## **Frontend Native applikation programmeringssprog**

Til udvikling af den native frontend anvendes .NET MAUI, som benytter C# som programmeringssprog.

**Fordele**

* Stærk typesikkerhed og compiler-validering.
* Let integration med backend skrevet i ASP.NET/C#.
* Mulighed for genbrug af modeller, valideringslogik og datatyper.

**Ulemper**

* Mindre community end visse JavaScript-baserede alternativer til native udvikling.
* Længere build-tider og større initial setup.

C# er det optimale valg, da det minimerer kompleksitet, forbedrer vedligeholdelse og muliggør en ensartet arkitektur mellem frontend og backend. MAUI med C# sikrer native-lignende performance og effektiv deling af kodebaser.

## **Backend programmeringssprog**

Backenden i Finance Tracker er udviklet i ASP.NET Core, og det primære sprog er C#.

**Fordele**

* Tæt integration med .NET-platformen og Microsoft Azure.
* Stærk ydeevne og optimeret garbage collection.
* Understøtter moderne objektorienteret og asynkron programmering.
* Udvidet sikkerheds- og valideringsinfrastruktur.

C# er et naturligt og strategisk valg til backend-udvikling med ASP.NET. Det giver stærk typesikkerhed, performanceoptimering og god understøttelse af kompleks forretningslogik – egenskaber der er afgørende i en finansapplikation med høj krav til dataintegritet og sikkerhed.

| **Komponent** | **Sprog** | **Fordele** | **Ulemper** |
| --- | --- | --- | --- |
| Webapp (Frontend) | TypeScript | - Statiske typer og bedre struktur  - God integration med ReactJS | - Kræver kompilering  - Øget kompleksitet |
| Native App (Frontend) | C# | - Stærk typesikkerhed  - Delbar kode med backend (.NET MAUI) | - Længere build-tider  - Relativt nyt framework (MAUI) |
| Backend | C# | - Tæt integration med ASP.NET  - Høj ydeevne og robusthed | - Mindre fleksibelt uden for Microsoft-økosystemet |
| Webdesign & layout | HTML/CSSTailwind CSS | - Simpelt og responsivt UI  - Hurtig styling med utility-classes | - Tailwind kræver tilvænning og build-step |
| API-kommunikation (Web) | JavaScript (via TS) | - Letvægt og browser-understøttelse | - Ikke typesikret alene  – kræver TypeScript for robusthed |

Tabel 4, Oversigt over programmeringssprog i systemet

Valget af programmeringssprog i Finance Tracker er truffet ud fra et helhedssyn på funktionalitet, udviklerkompetencer, systemarkitektur og vedligeholdelse. Ved at anvende TypeScript i webfrontend opnås forbedret strukturel kvalitet og lavere fejlrate, hvilket er afgørende for brugerinteraktion og datavisning. Til native applikationen og backend er C# valgt, hvilket sikrer tæt integration, høj ydeevne og en fælles teknologi-stack på tværs af systemets lag. Kombinationen af .NET MAUI og ASP.NET skaber et sammenhængende økosystem, som reducerer kompleksitet og øger udviklingseffektiviteten. Med HTML, CSS og Tailwind CSS opnås desuden en moderne og responsiv brugergrænseflade.

Den samlede sprogstrategi understøtter systemets funktionelle og ikke-funktionelle krav, giver stærk fremtidssikring og fremmer vedligeholdelse og samarbejde i udviklingsteamet.

# Database

Databasen er en central komponent i Finance Tracker, da systemet skal kunne håndtere følsomme løndata, brugerintegration, statistikker og historik. Både performance, dataintegritet og strukturel fleksibilitet spiller en stor rolle for valget af databaseteknologi. Fire løsninger er analyseret: MySQL, MongoDB, MsSQL og Django ORM.

## **MySQL**

MySQL er en velkendt open-source RDBMS (Relational Database Management System), der anvender strukturerede skemaer til at gemme data i tabeller[12].

**Fordele**

* God til relationelle data og transaktioner med klare datatyper.
* Høj ydeevne ved læse-/skriveoperationer.
* Udbredt og veldokumenteret – mange undervisningsressourcer og biblioteker.

**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 (fx dokument-API’er).

MySQL er en hurtig og velunderstøttet relationsdatabase, som egner sig godt til systemer med faste, relationelle datastrukturer. Den er dog mindre fleksibel ved ændringer i datamodel og kompleks datarelation. Derfor er den ikke ideel til et system med mange forbundne og potentielt dynamiske datatyper som i Finance Tracker.

## **MongoDB**

MongoDB er en dokumentorienteret NoSQL-database, hvor data gemmes som JSON-lignende dokumenter (BSON), hvilket giver fleksibilitet i skemastrukturen[13].

**Fordele**

* Fleksibel datamodel – ideel til hurtig udvikling og skiftende datatyper.
* Horisontal skalerbarhed og høj skriveperformance.
* Egnet til prototyper og ustrukturerede datamængder.

**Ulemper**

* Ikke fuldt ACID-kompatibel, hvilket kan give dataintegritetsproblemer ved finansielle transaktioner.
* Relationelle afhængigheder er komplekse at modellere og vedligeholde.
* Mindre intuitivt til forespørgsler med mange JOIN-lignende behov.

MongoDB tilbyder høj fleksibilitet og hurtig datalagring, men mangler fuld ACID-kompatibilitet. Dens dokumentbaserede struktur er uhensigtsmæssig til komplekse relationer som dem, der opstår ved lønsedler og historik. MongoDB vurderes derfor uegnet til en finansiel applikation med krav om dataintegritet.

## **MsSQL**

Microsoft SQL Server (MsSQL) er en enterprise-grade relationel databaseplatform udviklet af Microsoft[11]. Den tilbyder avanceret sikkerhed, transaktionshåndtering og tæt integration med .NET-teknologier.

**Fordele**

* Tæt integration med ASP.NET og Entity Framework.
* Understøtter ACID-transaktioner, views, stored procedures og komplekse forespørgsler.
* Stærk værktøjsunderstøttelse (SQL Server Management Studio, Azure Data Studio).

**Ulemper**

* Kræver konfiguration af licenser og ressourcer (selv om Express-versionen er gratis).
* Større ressourceforbrug sammenlignet med letvægtsdatabaser som SQLite eller MySQL.
* Mindre fleksibel ved deployment på ikke-Windows platforme (selvom understøttelse er blevet bedre med Docker og Linux).

Microsoft SQL Server er en robust og skalerbar database med stærk transaktionssikkerhed og god integration med .NET-økosystemet. Den understøtter komplekse relationer og er velegnet til finansielle systemer. Derfor vurderes MsSQL som det mest egnede valg til projektets krav og arkitektur.

## **Django**

Django ORM er ikke en database i sig selv, men et objekt-relationsmapper-framework der ofte bruges sammen med PostgreSQL, SQLite eller MySQL. Det nævnes her, da det kan være relevant i projekter, hvor Python anvendes.

**Fordele**

* Hurtig prototypning og høj udviklingshastighed.
* Mulighed for at skifte mellem databaser (abstraktion).
* Velegnet til testbarhed og små projekter.

**Ulemper**

* Begrænset synergi med .NET- og C#-teknologistack.
* Ikke egnet til tunge finansielle systemer uden dybere tilpasning.
* Mindre oplagt i kombination med ASP.NET-arkitekturen.

Django ORM tilbyder hurtig udvikling og fleksibilitet, men passer ikke ind i en C#/ASP.NET-baseret stack. Da teamet ikke anvender Python eller Django, og teknologien ikke understøtter direkte integration med Microsofts platforme, fravælges denne løsning.

| **Database** | **Fordele** | **Ulemper** |
| --- | --- | --- |
| MySQL | - Hurtig til relationelle forespørgsler  - Udbredt og veldokumenteret | - Begrænset til faste skemaer  - Svær ved komplekse relationer |
| MongoDB | - Fleksibel struktur  - Høj skriveperformance | - Ikke ACID-kompatibel  - Besværlig håndtering af relationer |
| MsSQL | - Tæt integration med .NET  - ACID-sikker og skalerbar | - Licens- og ressourcekrævende  - Mindre fleksibel ved non-MS stack |
| Django ORM | - Hurtig udvikling og testbar  - Database-uafhængigt | - Ikke egnet i .NET-stack  - Lav performance ved kompleksitet |

Tabel 5, Sammenligning af databaser

På baggrund af analyse og systemkrav vælges Microsoft SQL Server (MsSQL) som database til Finance Tracker. Det skyldes især behovet for stærk dataintegritet, relationshåndtering og synergi med ASP.NET-backenden. MsSQL tilbyder en skalerbar og sikker løsning med avancerede forespørgselsmuligheder og tæt integration med .NET og C#, hvilket muliggør effektiv udvikling og vedligeholdelse. MongoDB og MySQL fravælges pga. henholdsvis manglende ACID-kompatibilitet og lavere fleksibilitet til komplekse relationelle strukturer. Django ORM vurderes irrelevant i .NET-konteksten.

# Server integration med frontend

Valget af integrationsteknologi mellem backend og frontend er afgørende for systemets ydeevne, skalerbarhed og brugeroplevelse. Det har direkte betydning for, hvordan data udveksles, opdateres og håndteres mellem klienten og serveren. I denne analyse vurderes tre populære teknologier: REST API, GraphQL og WebSockets.

## **REST API**

REST (Representational State Transfer) er en arkitekturstil, der benytter HTTP-metoder som GET, POST, PUT og DELETE til kommunikation med serveren[10]. Data returneres typisk i JSON-format.

**Fordele**

* Enkel at implementere og vedligeholde.
* Bredt understøttet i næsten alle moderne programmeringssprog og frameworks.
* Stateless karakter og mulighed for caching kan øge ydeevne ved gentagne forespørgsler.

**Ulemper**

* Kan medføre over-fetching eller under-fetching, da datastrukturen er fastlagt på serveren.
* Ikke optimeret til realtidskommunikation, hvilket kan være en begrænsning ved løbende opdateringer.
* Hver forespørgsel behandles individuelt, hvilket kan medføre øget latenstid i sekventielle kald.

REST er en stabil og gennemtestet integrationsteknologi, der er let at implementere og vedligeholde. Det er ideelt til standard CRUD-operationer og struktureret datatilgang, men har begrænsninger i forhold til realtidsbehov og fleksibilitet i datavalget.

## **GraphQL**

GraphQL er et query-baseret API, hvor klienten selv definerer hvilke data den ønsker[9]. Teknologien anvender én samlet endpoint til forespørgsler og mutationer.

**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.
* Effektiv datahåndtering i frontend-komponenter med varierende behov.

**Ulemper**

* Har en stejlere læringskurve sammenlignet med REST.
* Kræver avanceret serveropsætning med hensyn til autorisation, caching og fejlbehandling.
* Komplekse forespørgsler kan øge belastningen på serveren og kræver god databaseoptimering.

GraphQL egner sig godt til komplekse og dynamiske brugergrænseflader med behov for præcis datastyring. Det reducerer netværkstraffik men kræver en højere grad af serverlogik og datamodeloptimering, som kan være uhensigtsmæssigt i denne projektfase.

## **WebSockets**

WebSockets er en fuld-dupleksprotokol, der muliggør vedvarende, tovejs-kommunikation mellem klient og server uden behov for gentagne HTTP-anmodninger[8].

**Fordele**

* Ideel til realtidsfunktionalitet, f.eks. ved lønopdateringer og arbejdstidstracking.
* Hurtigere end REST og GraphQL ved hyppige dataændringer.
* Mindre netværksbelastning, da data kan push’es direkte til klienten uden polling.

**Ulemper**

* Mere kompleks at implementere korrekt og sikkert, især ift. autentificering og forbindelseshåndtering.
* Vedvarende forbindelser kræver korrekt styring af ressourcer for at undgå ydeevneproblemer.
* Ikke altid nødvendigt ved traditionelle forespørgsel/svar-interaktioner.

WebSockets giver markante fordele i applikationens realtidsfunktioner, såsom live lønberegning og arbejdstidsopdatering. Det kræver dog avanceret håndtering af forbindelser og ressourcer. Teknologien anbefales som supplement, ikke erstatning for REST.

| **Teknologi / Protokol** | **Fordele** | **Ulemper** |
| --- | --- | --- |
| REST API | - Enkel at implementere  - Bred support og standardiseret  - Understøtter caching | - Over-/under-fetching  - Ikke realtidsoptimeret  - Øget latenstid |
| GraphQL | - Præcise forespørgsler  - Mindre antal API-kald  - Effektiv til dynamisk UI | - Høj kompleksitet  - Ressourcetung ved komplekse queries |
| WebSockets | - Realtidskommunikation  - Lav netværksbelastning  - Hurtig feedback | - Sværere at implementere og sikre  - Ressourcestyring er kritisk |

Tabel 6, Sammenligning af integrationsteknologier

Den mest hensigtsmæssige tilgang til integration mellem frontend og backend i Finance Tracker er en kombination af REST API og WebSockets. REST anvendes som hovedprotokol til strukturerede dataoperationer og CRUD-funktionalitet, mens WebSockets anvendes målrettet til realtidsbaserede funktioner, der kræver øjeblikkelig opdatering og respons. GraphQL fravælges grundet øget kompleksitet, manglende erfaring i gruppen og manglende behov for dens avancerede funktionalitet i denne iteration af systemet.

# Object-Relational Mapping(ORM)

I udviklingen af Finance Tracker skal backend-komponenten kunne kommunikere effektivt og sikkert med databasen. Dette gøres traditionelt via SQL, men ORM-teknologier tilbyder et højere abstraktionsniveau, som kan øge udviklingshastigheden og forbedre vedligeholdelsen. To ORM-løsninger er analyseret: Entity Framework Core (EF Core) og Dapper.

## **Entity framework Core**

Entity Framework Core er Microsofts officielle ORM til .NET-platformen[3]. Det tillader udviklere at arbejde med databasen gennem C#-objekter i stedet for eksplicitte SQL-forespørgsler.

**Fordele**

* Understøtter database-migreringer, hvilket gør det nemt at versionere og opdatere skemaer.
* Integreres let med sikkerhedsfunktioner som rollebaseret adgangskontrol og autentificering.
* Forenkler udviklingen ved at eliminere behovet for at skrive manuelle SQL-forespørgsler.
* Understøtter LINQ, hvilket giver en konsistent måde at håndtere data på.

**Ulemper**

* Har performance-overhead pga. det ekstra abstraktionslag.
* Genererede SQL-forespørgsler kan være mindre optimerede end håndskrevne.
* Kræver forståelse af koncepter som tracking, context-lifecycle og migrations.

EF Core er velegnet til større systemer med komplekse domænemodeller og krav om høj sikkerhed og skema-kontrol. Det reducerer udviklingstid og understøtter moderne softwarepraksis som migrations og LINQ. Det anbefales som standardvalg i systemer med tæt .NET-integration og behov for hurtig udvikling og høj vedligeholdelseseffektivitet.

## **Dapper**

Dapper er et mikro-ORM udviklet af Stack Overflow, som fungerer som en tynd wrapper omkring SQL og ADO.NET[4].

**Fordele**

* Ekstremt hurtig udførelse og lavt hukommelsesforbrug.
* Giver fuld kontrol over SQL, hvilket muliggør 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.

Dapper egner sig bedst til performancekritiske moduler med få og gentagne forespørgsler, hvor man ønsker fuld SQL-kontrol. Det kræver dog mere kode og øger kompleksiteten. Det anbefales som supplement til EF Core i performancefølsomme dele af systemet – eksempelvis statistik- eller rapporteringsmoduler.

| **Teknologi** | **Fordele** | **Ulemper** |
| --- | --- | --- |
| **EF Core** | - Automatisk database-migrering  - Høj sikkerhedsintegration  - Mindre behov for SQL | - Performance-overhead  - Mindre kontrol over queries |
| **Dapper** | - Høj ydeevne  - Fuld kontrol over SQL  - Lavt hukommelsesforbrug | - Ingen tracking  - Manuel vedligeholdelse  - Øget kompleksitet ved relationer |

Tabel 7, Sammenligning af ORM-teknologier

På baggrund af analysen vælges Entity Framework Core som standard ORM-løsning i Finance Tracker-systemet. EF Core tilbyder en god balance mellem udviklingshastighed, sikkerhed og funktionel fleksibilitet. Med understøttelse af database-migreringer, LINQ og indbygget integration til .NET-platformen forenkles både udvikling og vedligeholdelse af datatilgangslag.

EF Core muliggør en høj grad af abstraktion og sikrer konsistens i datahåndtering på tværs af hele systemet, samtidig med at det understøtter vigtige ikke-funktionelle krav som adgangskontrol og validering. Selvom Dapper tilbyder højere performance og kontrol i udvalgte tilfælde, vurderes EF Core at være den mest robuste og fremtidssikrede løsning til det overordnede systemdesign.

Ved performancekritiske dele af applikationen, såsom statistikgenerering eller avancerede filtreringer, kan Dapper anvendes som supplement, hvor det giver målbar værdi – men EF Core forbliver den primære ORM-løsning.

# Overordnet teknologivalg

På baggrund af den gennemførte tekniske analyse er den endelige teknologistak for Finance Tracker sammensat med fokus på høj ydeevne, skalerbarhed, sikkerhed og effektiv udvikling. Valgene er truffet med hensyn til både de funktionelle og ikke-funktionelle krav samt projektgruppens tekniske kompetencer.

Den valgte backend-teknologi, ASP.NET Core, muliggør en robust og sikker API-arkitektur med god performance og tæt integration med Microsoft-økosystemet. Til datalagring er Microsoft SQL Server valgt for at sikre fuld ACID-kompatibilitet og effektiv håndtering af relationelle data.

Frontend-teknologierne er valgt med fokus på modularitet og brugeroplevelse. ReactJS benyttes til webapplikationen grundet dets fleksible komponentmodel og hurtige DOM-opdatering, mens .NET MAUI giver mulighed for genbrug af forretningslogik i en samlet, native-lignende løsning på tværs af mobile platforme.

Der anvendes C# og TypeScript som primære programmeringssprog, hvilket skaber en konsistent og typesikker kodebase i både frontend og backend. Til integration mellem frontend og backend benyttes primært REST API, med mulighed for WebSockets i realtidskritiske scenarier.

Til databaseadgang anvendes Entity Framework Core som standard ORM, da det sikrer en høj grad af struktur, understøtter migreringer og minimerer kompleksiteten i datatilgangen. Dapper kan benyttes selektivt i performancefølsomme områder.

Samlet set udgør den valgte teknologistak en moderne, velafbalanceret og fremtidssikret løsning, der understøtter både udviklingsproces og systemkrav i projektets nuværende og fremtidige faser.

| **Komponent** | **Valgt teknologi** | **Begrundelse** |
| --- | --- | --- |
| **Backend** | ASP.NET Core | Høj ydeevne, stærk sikkerhed, .NET-integration |
| **Database** | Microsoft SQL Server | ACID-kompatibel, velegnet til relationelle datastrukturer |
| **Web-Frontend** | ReactJS | Fleksibel komponentstruktur, hurtig DOM-opdatering |
| **Native-Frontend** | .NET MAUI | Én fælles kodebase, tæt kobling til backend |
| **Programmeringssprog** | C#, TypeScript | Stærk typesikkerhed, synergi mellem frontend og backend |
| **API-integration** | REST API | REST for enkelhed |
| **ORM** | Entity Framework Core (EF Core) | Struktur, migrations, .NET-kompatibilitet. Dapper som supplement ved behov |

Tabel 8, Oversigtstabel

# Referenceliste

[1] Microsoft, “ASP.NET documentation”, Microsoft Learn, [Online]. Available: <https://learn.microsoft.com/en-us/aspnet/core/>. [Accessed: 21-May-2025].

[2] Microsoft, “.NET Multi-platform App UI documentation”, Microsoft Learn, [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/maui/>. [Accessed: 21-May-2025].

[3] Microsoft, “Entity Framework Core”, Microsoft Learn, [Online]. Available: <https://learn.microsoft.com/en-us/ef/core/>. [Accessed: 21-May-2025].

[4] Stack Overflow Engineering, “Dapper: A simple object mapper for .NET”, GitHub, [Online]. Available: <https://github.com/DapperLib/Dapper>. [Accessed: 21-May-2025].

[5] ReactJS.org, “React The library for web and native user interfaces”, [Online]. Available: <https://reactjs.org/>. [Accessed: 21-May-2025].

[6] TypeScript Team, “TypeScript Documentation”, [Online]. Available: <https://www.typescriptlang.org/docs/>. [Accessed: 21-May-2025].

[7] Mozilla Developer Network (MDN), “JavaScript”, [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. [Accessed: 21-May-2025].

[8] Mozilla Developer Network (MDN), “The WebSocket API (WebSockets)”, [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API>. [Accessed: 21-May-2025].

[9] GraphQL Foundation, “Introduction to GraphQL”, [Online]. Available: <https://graphql.org/learn/>. [Accessed: 21-May-2025].

[10] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, Dissertation, Univ. of California, Irvine, 2000. [Online]. Available: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. [Accessed: 21-May-2025].

[11] Microsoft, “SQL Server technical documentation”, Microsoft Learn, [Online]. Available: <https://learn.microsoft.com/en-us/sql/sql-server/>. [Accessed: 21-May-2025].

[12] Oracle, “MySQL 8.0 Reference Manual”, MySQL Documentation, 2025, [Online]. Available: <https://dev.mysql.com/doc/refman/8.0/en/>. [Accessed: 21-May-2025].

[13] MongoDB Inc., “What is MongoDB?”, [Online]. Available: <https://www.mongodb.com/docs/manual/>. [Accessed: 21-May-2025].