TestMA1 - Mini Rapport

Til projektet har vi valgt JavaScript til backend-implementeringen. Dette valg tog vi, fordi JavaScript er et fleksibelt sprog, som kan bruges til både backend- og frontend-udvikling. Samtidigt er det et sprog, som alle i gruppen har erfaringer med. Vores kode er modulært og opdelt i flere filer, der indeholder specifikke funktioner og hjælpefunktioner, hvilket sikrer, at hver komponent har et klart defineret ansvar.

På backend-delen valgte vi også at bruge Knex.js til at administrere forbindelsen til vores MySQL-database. Knex er et fleksibelt værktøj, der understøtter migrationer og seeding, hvilket har gjort det nemt for os at håndtere database ændringer og sikre en konsistent opsætning på tværs af udviklingsmiljøer.

Vi har valgt at bruge JSON som outputformat. Vi valgte denne fremgangsmåde, da det er hvad vi er vant til at bruge og for at sikre kompatibilitet med den udleverede frontend.

Til testformål valgte vi Jest, da det er et populært og udbredt test framework med god support til både unit- og integrationstests for JavaScript. Derudover bruger vi Cypress til end-2-end test i frontend.

Teststrategi

Vi har primært brugt black-box testing i vores unit tests. Det betyder, at vi tester funktionerne ved at give dem input og sammenligne outputtet med det forventede resultat, uden at fokusere på, hvordan funktionen er implementeret. For eksempel tester vi, om CPR-numrene genereres korrekt, og om de overholder kravene til både dato og køn, som er beskrevet i opgavebeskrivelsen. Derudover har vi brugt forskellige black-box testing-teknikker, herunder *Equivalence Partitioning* samt *Boundary Analysis*.

Equivalence Partitioning
Ved tests af Equivalence Partitioning, har vi opdelt input i forskellige kategorier og testet repræsentative værdier fra hver kategori. For eksempel i testen af CPR-numre for mænd og kvinder har vi to partitioner. En for mandlige CPR-numre, som skal ende i ulige tal, og en for kvindelige CPR-numre, som skal ende med et lige tal. På samme måde tester vi også, om fødselsdatoer er gyldige baseret på CPR-numrene. Dette giver os mulighed for at sikre, at funktionen opfører sig korrekt på tværs af alle relevante input typer og scenarier.

Boundary Analysis
Udover Equivalence Partitioning har vi også benyttet os af Boundary Analysis i flere af vores tests. Et eksempel er testen af bulk-generering af personer, hvor vi tester grænseværdier ved at generere data for både minimum (2) og maksimum (100) antal personer. Vi tester også, at input under minimums-eller over maksimumsgrænser korrekt resulterer i fejlmeddelelser. Tilsvarende tester vi gyldige og ugyldige CPR-numre, hvor længden af CPR-numrene tjekkes for at sikre, at de nøjagtig har 10 cifre.

White-box testing

Vi har diskuteret brugen af white-box testing, men fandt ikke et behov baseret på projektets krav. White-box testing fokuserer på at teste den interne logik i funktionerne, som f.eks. loops eller kontrolstrukturer som if/else statements. Men da mange af vores funktioner handler om simpel datamanipulation eller generering af tilfældige data, vurderede vi, at der ikke var behov for white-box tests. Vi besluttede, at black-box tilgangen var tilstrækkelig til at sikre funktionaliteten.

Mocking og integrationstests
Undervejs i udviklingen har vi været i tvivl om, hvornår vi skulle bruge mocking, og hvornår det var nødvendigt at teste med rigtige data. I nogle af vores unit tests har vi valgt at mocke eksterne afhængigheder, såsom læsning af JSON-filen. Vi har benyttet mocking til at simulere forskellige scenarier, såsom manglende data eller ugyldige filformater.

På den anden side har vi brugt integrationstests til at teste funktionalitet, der er afhængig af databasen. Vi har blandt andet testet, om vores adressefunktion henter data korrekt fra databasen. Her har vi brugt Knex til at oprette og nedlægge databaseforbindelser under testene, hvilket har gjort det muligt at teste interaktionen mellem applikationen og databasen.

Designvalg og validering af adresser
Et specifikt design valg, vi traf under udviklingen, er at ændre vores adressefunktion til altid at returnere strings. Dette valg tog vi, fordi vi oplevede, at adresser kan have forskellige formater, der både indeholder tal og bogstaver. Ved at sikre, at vores funktion altid returnerer strings, har vi gjort det nemmere at håndtere og validere adressformaterne konsekvent.

Edge cases og negative scenarier
Vores tests dækker også en række edge cases og negative scenarier. For eksempel tester vi, om vores funktioner håndtere ugyldige input korrekt, eller hvordan de opfører sig, hvis de modtager manglende data. Vi har prøvet at være omhyggelige med at teste både positive og negative scenarier for at sikre, at applikationen kan håndtere alle tænkelige input. Dette inkluderer tests af både gyldige og ugyldige CPR-numre, ugyldige JSON-data og korrekt håndtering af mobiltelefonnumre med gyldige præfikser. At teste både positivt, negativt og de forskellige edge cases sikrer, at vores applikation er robust og kan håndtere forskellige scenarier, som kunne opstå under brug.