ShopXpress

Procesrapport

TITELBLAD

Elev
Omidreza Ahanginashroudkoli

Projektnavn
ShopXpress

Uddannelse
Datatekniker med speciale i programmering

Projektperiode
06.nov.2023 – 08.dec.2023

Afleveringsdato
01.dec.2023

Fremlæggelsesdato
07.dec.2023

Vejledere
Frank Rosbak
Lars Thise Pedersen

Censor
(Navn)

Tech College Aalborg
Struevej 70,
9220 Aalborg

INDHOLDSFORTEGNELSE

FORORD

Procesrapporten indeholder information om de beslutninger, der er truffet for at opnå det bedst mulige resultat for produktet. Nederst i rapporten finder du projektets udviklingsproces i form af en dagbog/logbog. Den beskriver forløbet for hver projektdag, og hvilke udfordringer der blev mødt samt hvordan de blev håndteret.

LÆSEVEJLEDNING

Dette dokument skal læses sammen med Produktrapporten for at få det fulde overblik over produktet og processen bag skabelsen. Der vil undervejs forekomme diagrammer, figurer og tegninger, der anbefales at være opmærksom på disse illustrationer, da disse kan være vigtige for forståelsen af indholdet.

INDLEDNING

Dette projekt og processen startede med en oplevelse af, at den traditionelle tilgang til shopping er meget tidskrævende og ikke effektiv. Kampen med at navigere i overfyldte butikker, søge efter specifikke produkter og udholde lange betalingskøer kan være tidskrævende og frustrerende. Det skabte et problem, der kunne ses løst ved at tage moderne metoder og teknologier i brug. Som svar på disse udfordringer giver projektet en brugervenlig og sikker online shopping platform, som kan bruges af alle til hverdagsindkøb. Denne platform tilbyder ikke kun en bred vifte af produkter, men sikrer også et sikkert miljø, hvor brugerne kan udforske, udvælge og købe varer fra deres hjem.

CASE BESKRIVELSE

I det hurtige landskab i nutidens digitale æra, hvor teknologien udvikler sig hurtigt, fremstår tid som det mest kritiske element i menneskers liv. Efterspørgslen efter digitale løsninger er steget for de personer, der søger effektive måder at minimere tiden brugt på hverdagens opgaver. En markant tidskrævende indsats er fysisk shopping, som kræver navigation gennem trafikken for at finde den rigtige butik, udforske en bred vifte af produkter og udholde lange betalingskøer.
Disse tidskrævende udfordringer kan reduceres markant ved at udvikle en sikker og robust online platform, der tilbyder en bred vifte af produkter til forbrugerne, og som giver dem mulighed for frit at browse, søge, filtrere produkter efter kategori og tilføje ønskede produkter til deres indkøbskurv og købe med tilliden til deres datasikkerhed.

PROBLEMFORMULERING

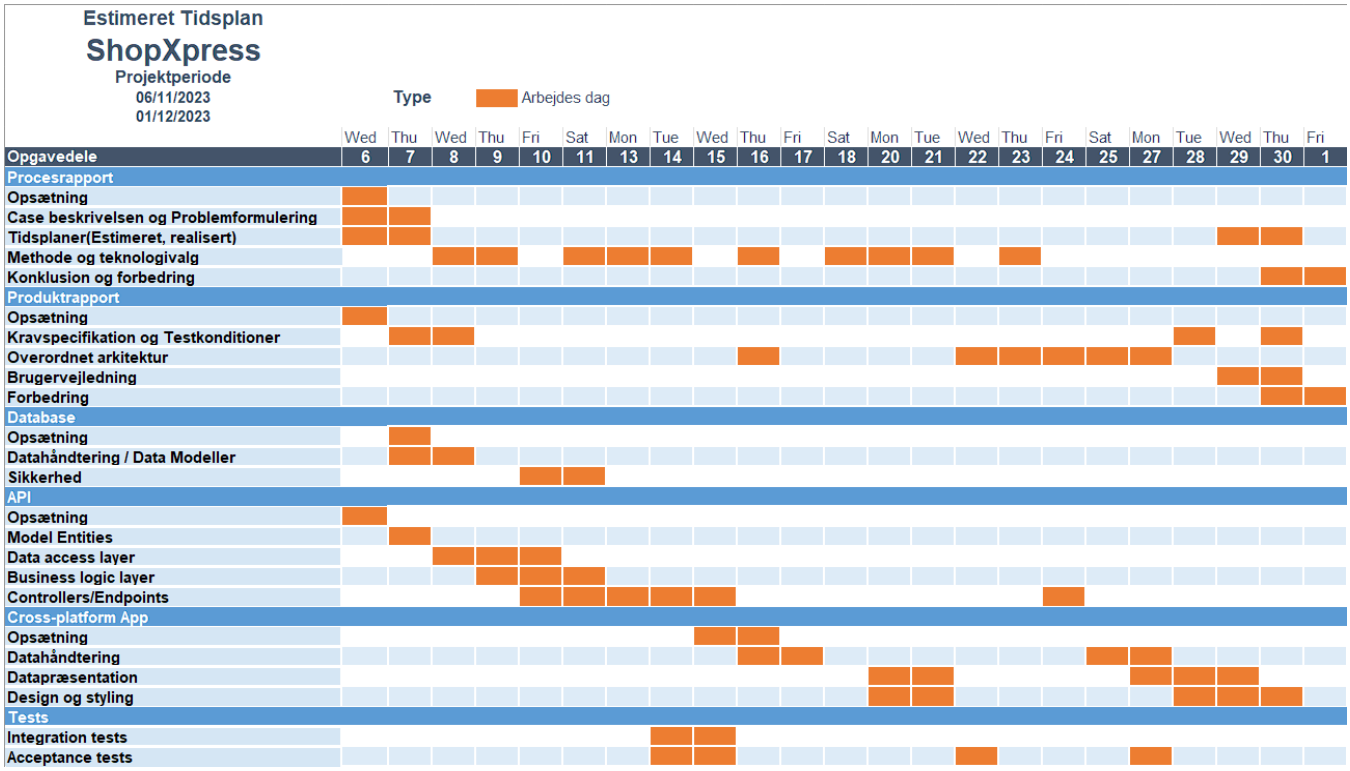
Hvordan kan man skabe en online shopping platform, der optimere bekvemmelighed, brugeroplevelse og sikkerhed, samtidig med at man effektivt løser udfordringerne ved produktopdagelse og opfylder forskellige forbrugerbehov?

PROJEKTPLANLÆGNING

Der er lavet en tidsplan for udførelsen af projektet. Derudover er der under udviklingen lavet en realiseret tidsplan, der kan læses om længere nede i rapporten. Hver eneste projektdag er der noteret i dagbogen om det udførte arbejde. Udover det, er der oprettet et Azure Board, der faciliteter funktioner til mere effektiv arbejdsstyring og hjælper med at opdele projektet i små opgaver.

ESTIMERET TIDSPLAN

Denne plan viser en oversigt over den estimerede tid for de forskellige opgaver som er involveret i oprettelsen af produktet. Den estimerede tidsplan blev udarbejdet i starten af projektet. Tidsplanen betragtes som et yderst vigtigt første skridt i projektets oprettelse, da det fører til ideudvikling og gennemgang af hvert enkelt komponent i projektet. Dette kan hjælpe udviklerne med at opnå en bedre forståelse af, hvilke teknologier der skal anvendes, og hvor meget tid der skal bruges på dem.



METODE- OG TEKNOLOGIVALG

Herunder findes de metoder og teknologier der er valgt i processen af dette projekt. Der vil være klare forklaringer, om hvorfor nogle blev valgt mens andre blev fravalgt.

Raw SQL vs. Entity framework Core

Rå SQL og Entity Framework Core er to forskellige tilgange til arbejde med relationelle databaser i .NET-applikationer. Hver har sine egne fordele og ulemper, og valget mellem dem afhænger af projektets behov og præferencer. EF Core giver en abstraktion på højt niveau til at arbejde med databaser, så du kan interagere med data ved hjælp af C#-objekter. Det kortlægger databasetabeller til C#-klasser og giver LINQ (Language Integrated Query) -understøttelse til forespørgsler i databasen.
EF Core giver sikkerhed for kompileringstid og stærkt indtastede forespørgsler, som rå SQL ikke gør, hvilket hjælper med at påpege fejlene tidligt i udviklingen. Den anden nyttige funktion i EF Core er, at den understøtter flere databaseudbydere, hvilket hjælper om en lettere udskiftning mellem databaser. Den største begrænsning ved at bruge EF Core er dens ydeevne, EF Core udfører langsommere, når de står over for komplekse forespørgsler sammenlignet med rå SQL. Selvom EF Core udfører langsommere end rå SQL, er det ikke en deal breaker for dette projekt, derfor er dette valgt og brugt som ORM i projektet.

ORM

EF Core er en ORM, hvilket står for ”Object-relational mapping”. Det er en metode, der gør det muligt at lave queries og manipulere data i en database ved brug af et object-oriented paradigm language. Her inkluderet C# der er et OOP (Object oriented programming) sprog.

Asp.net Core Web API vs. Flask

API ‘er er nødvendige for at bringe applikationer sammen, for at udføre en designet funktion bygget op omkring deling af data og udførelse af foruddefinerede processer. API fungerer som mellemmand og giver udviklere mulighed for at bygge applikationer, som kan interagere med hinanden. Fordelen ved at bruge API ‘er er, at det hjælper med at have bedre data kvalitet gennemgang og oprydning. API kan understøtte hurtigere og lettere datamigrering og det giver større fleksibilitet.

ASP.NET Core Web API er den valgte ramme til at bygge dette projekt. Det giver mulighed for at bygge HTTP services og afsløre RESTful API’er, der kan forbruges af forskellige klienter, såsom webapplikationer, mobilapps, eller andre services. ASP.NET Core Web API opfordrer til designet af RESTful API’er, som følger et sæt af principper for at skabe skalerbare og vedligeholdelige web services. Den høj ydeevne, fleksibiliteten i hostingmiljøer(Windows, macOS og Linux), sikkerhedsfunktionerne, det store aktive fællesskab med et stort udvalg af libraries og mit kendskab til både teknologien og sproget(C#), gør det til et fremragende valg til dette projekt.

Alternativ ramme er Flask. Flask er en let og fleksibel mikroramme til opbygning af webapplikationer i Python. På grund af det minimalistiske design er det derfor nemmere, at anvende designet som også giver muligheden for at udvælge komponenter til ekstra funktionalitet.

Flask er velegnet til at bygge RESTful API'er, hvilket gør det nemt at oprette endpoints og håndtere HTTP-metoder. Flask mangler indbyggede funktioner, da behovet for tredjepartsudvidelser vokser. Dette kan påvirke kompatibiliteten og kvaliteten af koden, hvilket gør det ikke den bedste framework for projektet.

N-layer architecture vs. Clean architecture

N-Layer og Clean Architecture er arkitektoniske patterns, der har til formål at strukturere og organisere kode på en måde, der fremmer vedligeholdelse, skalerbarhed og adskillelse af bekymringer. Projektet vil bruge den traditionelle tilgang, hvor applikationen er organiseret i lag, typisk inklusive Presentation, Business Logic, og Data Access layers. N-layer hjælper os med at adskille business logic fra Controllers. Disse adskillelser gør koden mere overskuelig, læsbar, testbar og mere vedligeholdelsesvenlig i forhold til et all-in-one layer. N-layer arkitektur er enkel og lige til at implementere, hvilket gør den til et godt valg for de fleste og mindre komplekse projekter.

Alternativet vil være Clean Architecture. Clean Architecture er en mere moderne tilgang inspireret af SOLID-principperne, Domain-Driven Design (DDD) og andre bedste praksisser. Clean arkitektur er meget modulær, hvilket gør det nemmere at tilføje, fjerne eller opdatere funktioner uden at påvirke hele applikationen. Ulempen ved at bruge Clean Architecture er kompleksiteten. For mindre projekter kan Clean Architecture ses som over-engineering, hvilket fører til unødvendig kompleksitet. Det kræver også mere indledende opsætning og udvikling, hvilket er mere tidskrævende og vil bremse processen.

Entity Configuration & Seeding

Entity-konfiguration giver os mulighed for at konfigurere tilknytningen mellem dine C#-entitetsklasser og de tilsvarende databasetabeller. Det bruges til at definere, hvordan egenskaberne for dine enhedsklasser skal tilknyttes databasekolonner, indstille relationer, specificere indekser og definere andre aspekter af databaseskemaet. Denne praksis er en del af Fluent API, som giver en mere overskuelig, fleksibel og programmatisk måde at konfigurere dine enhedsmodeller på i EF Core. Seeding sker kun en gang, hvilket er lige efter den første migrering af databasen. Denne service er kaldt af initialisering servicen, efter den har sikret databasen tilfældighed og migreret databasen. Initialisering servicen opdager selv om seeding er nødvendig. Det er baseret på de migrationer, der var på databasen inden initialisering startede.

Automapper

AutoMapper er et kraftfuldt værktøj til at forenkle object mapping i C# applikationer, reducere gentagen kode og gøre koden mere vedligeholdelig. Det er især værdifuldt i scenarier som DTO (Data Transfer Object) mapping eller konvertering af domæneenheder for at se modeller i webapplikationer. Mens AutoMapper er et populært valg til objekt-til-objekt-mapping i C#, er der flere alternative tilgange, såsom Manual Mapping, man kan udføre objektmapping manuelt ved at skrive tilpasset kode for at kopiere værdier fra et objekt til et andet. Denne tilgang giver mere kontrol, men kan være mere tidskrævende og fejltilbøjelig, især for større og mere komplekse objektstrukturer. Jeg

valgte at bruge AutoMapper til at reducere gentagelses og boilerplate kode, for at gøre koden mere vedligeholdelsesvenlig.

Flutter vs. Maui

Flutter og .NET MAUI (Multi-platform App Ui) skiller sig ud som populære rammer for udvikling af mobileapps på trværs af platforme, der hver bringer deres egne styrker og overvejelser.
Flutter udmærker sig ved at levere meget tilpasselige og visuelt tiltalende brugergrænseflader takket være dens widget-baserede arkitektur. Flutter har et hurtigt community med mange pakker til forskellige funktioner. Flutters hot reload-funktion gør det muligt for udviklere at se effekterne af kodeændringer i realtid, hvilket fremskynder udviklingen og debugging-processen. Det er dog værd at bemærke, at Flutters større binære størrelse sammenlignet med full native apps kan resultere i lidt langsommere ydeevne.
På den anden side tilbyder .NET MAUI, udviklet af Microsoft og bruger programmeringssproget C#, en alternativ tilgang. fordel af integration med det bredere .NET ecosystem , hvilket giver en problemfri oplevelse for udviklere, der allerede er fordybet i Microsoft-teknologier. NET MAUI giver også "hot reload", så udviklere kan se ændringer i realtid under kodeudvikling.
På trods af synergien mellem mit eksisterende web-API og .NET MAUI inden for det samme økosystem, hælder min præference til Flutter baseret på min praktiske erfaring med .NET MAUI. En af grundene til denne præference er den mere omfattende fællesskabssupport og biblioteker, der er tilgængelige for Flutter, hvilket øger udviklingsfleksibiliteten og effektiviteten. Derudover har jeg stødt på udfordringer med .NET MAUI, herunder langsommere kompileringstider og forsinkelse i brugergrænsefladen.

Test

Test er en afgørende komponent i projektet for at sikre, at applikationen lever op til de fastsatte krav. White box test er lavet med en viden, om hvordan applikationen virker og kræver programmerings erfaring at lave. Disse test er bedst egnet til test af lavere lag som Unit og Integration. Sådanne test er oplagte at udfører automatisk gennem frameworks og andet software. Hensigten med denne slags teste er at sikre kvaliteten af applikationen. Denne type test udføres kun af udviklere i dette projekt. Black box-test udføres uden kendskab til, hvordan applikationen fungerer og kræver ingen programmeringserfaring. Denne type test er bedst egnet til test af højere lag som System. GUI og Acceptance. Testene skal teste applikationens funktionalitet og brugbarhed.

Unit

Unit test er implementeret med det formål at teste individuelle metoder. Det er en test der sikrer at metoden opfører sig som forventet. Det er bedst at teste både gyldige og ugyldige input, for at sikrer programmet håndterer ugyldige input uden at bryde sammen.

Integration

Integrations test er er implementeret med det formål at teste applikationens komponenter i sammenspil. Det tester at komponenterne der arbejder sammen, virker og producerer de forventede resultater. Der er lavet denne slags test til API controller endpoints. De arbejder sammen med authentication og authorisation middleware, repositories og databasen.

Acceptance
Acceptance test er den højeste lag af test og skal udelukkende udføres af brugere af systemet. Denne testtype bruges til at bekræfte, at applikationen opfylder brugernes krav, som specificeret i kravspecifikationen og løser brugerens problem, som beskrevet i problemformuleringen. Disse tests udføres som det sidste skridt i projektet, efter at al funktionalitet er implementeret, og test på de lavere niveauer er blevet gennemført og godkendt.

IDE'er
Et integreret udviklingsmiljø (IDE) er en softwareapplikation eller -platform, der giver et omfattende sæt af værktøjer og funktioner til at hjælpe computerprogrammører og softwareudviklere med at skabe, teste og administrere softwareapplikationer. Herunder findes de IDE' er der blevet valgt i udviklingen af dette projekt. Der vil være klare forklaringer, om hvorfor nogle blev valgt mens andre blev fravalgt.
Visual Studio er en kraftfuld og funktionsrig IDE, der tilbyder adskillige fordele til Web API-udvikling, herunder integrerede udviklingsværktøjer, web API-skabeloner, en kraftfuld kodeeditor, fejlfindingsværktøjer, test- og profileringsfunktioner, versionskontrolintegration, support til NuGet pakker, Azure-integration, udvidelsesmuligheder gennem plugins, et understøttende community. Baseret på disse funktioner og min erfaring med det, har jeg besluttet at udvikle web-API'en i Visual Studio.
Rider er en cross-platform IDE fra JetBrains designet til .NET udvikling. Det tilbyder robust support til .NET-økosystemet, intelligent kodeanalyse og fejlfinding og enhedstestning. Rider's alsidighed strækker sig til webudvikling og cloud-integration med fokus på kodekvalitet og præstationsprofilering. Det er et overbevisende alternativ til Visual Studio for .NET-udviklere.

Hjælpeprogrammer
Herunder findes de hjælpeprogrammer der er valgt i processen af dette projekt.

Azure
Microsoft Azure er en cloud computing platform og service skabt af Microsoft. Det giver en bred vifte af cloud-services, herunder computerkraft, lagring, netværk, databaser, machine learning og mere. Azure bruges på grund af de mange fordele, det giver for projektet. Nogle af de fordele, som benyttes til projektet er en fuldt styret relationel database service, der giver høj ydeevne, sikre og skalerbare databaser. Azure App Service, der gør det muligt at bygge, hoste og skalere webapplikationer og API'er. Azure DevOps, der hjælper med at skabe et planlægningsmiljø (Scrum) til at spore projektets fremskridt. Det inkluderer også services til versionsstyre koden. Det giver også Azure Blob Storage, der kan bruges til at gemme billeder og videoer til videreudvikling af projektet i fremtiden.
Et alternativ til Azure er Google Cloud Platform, der leverer cloud-services såsom computing, storage, databaser sammen med værktøjer til applikationsudvikling og implementering. Ulempen ved det er, at det er mindre stabilt og modent i forhold til Azure.

GIT
Git er et VSC (Version Control System) der bliver brugt af mange de populære platformer som Github og Azure DevOps. Git bruges til at spore ændringer i kildekoden under softwareudvikling. Det

hjælper flere udviklere med at samarbejde om et projekt ved at udgive en effektiv metode til at administrere og flette ændringer på. Git repository, eller repo, er en samling af filer og deres revisionshistorik, det kan være lokalt (på din computer) eller eksternt (på en server). Git tilbyder en masse funktioner. Nogle populære er, kloning af eksisterende repository, Commit et øjebliksbillede af dit arbejde på et bestemt tidspunkt, separat udviklingslinje ved at oprette en ny Branch for at arbejde på en ny funktion uden at påvirke hoved Branch, Merging kombinerer ændringer fra forskellige branches, Pull ændringer fra et fjernlager, Push ændringer til et fjernlager og en masse andet for at forbedre udviklingsmiljøet.

Swagger

Swagger er et fantastisk værktøj, der kan mange ting. Swagger giver et overblik over alle endpoints i API'et. Oplysningerne fra Swagger er informative og nyttige. Nogle af disse oplysninger er versionen af API, endepunkters path, forventede input og mulige output. Swagger kan desuden udføre test af API'en direkte i browseren. Dette hjælper med at teste endepunkter, der er under udvikling.

Postman

Postman giver mulighed for at oprette og udføre API-requests for at teste funktionaliteten af en API. Collections i Postman er grupper af gemte anmodninger, organiseret til bedre workflowstyring. Det kan også deles mellem teammedlemmer, hvilket gør det nemmere at samarbejde om API-udvikling og -testning. Postman er et alsidigt værktøj, der forenkler processen med at arbejde med API'er.

VÆSENTLIGE ELEMENTER (FRA PRODUKTRAPPORTEN)

Entity framework Core

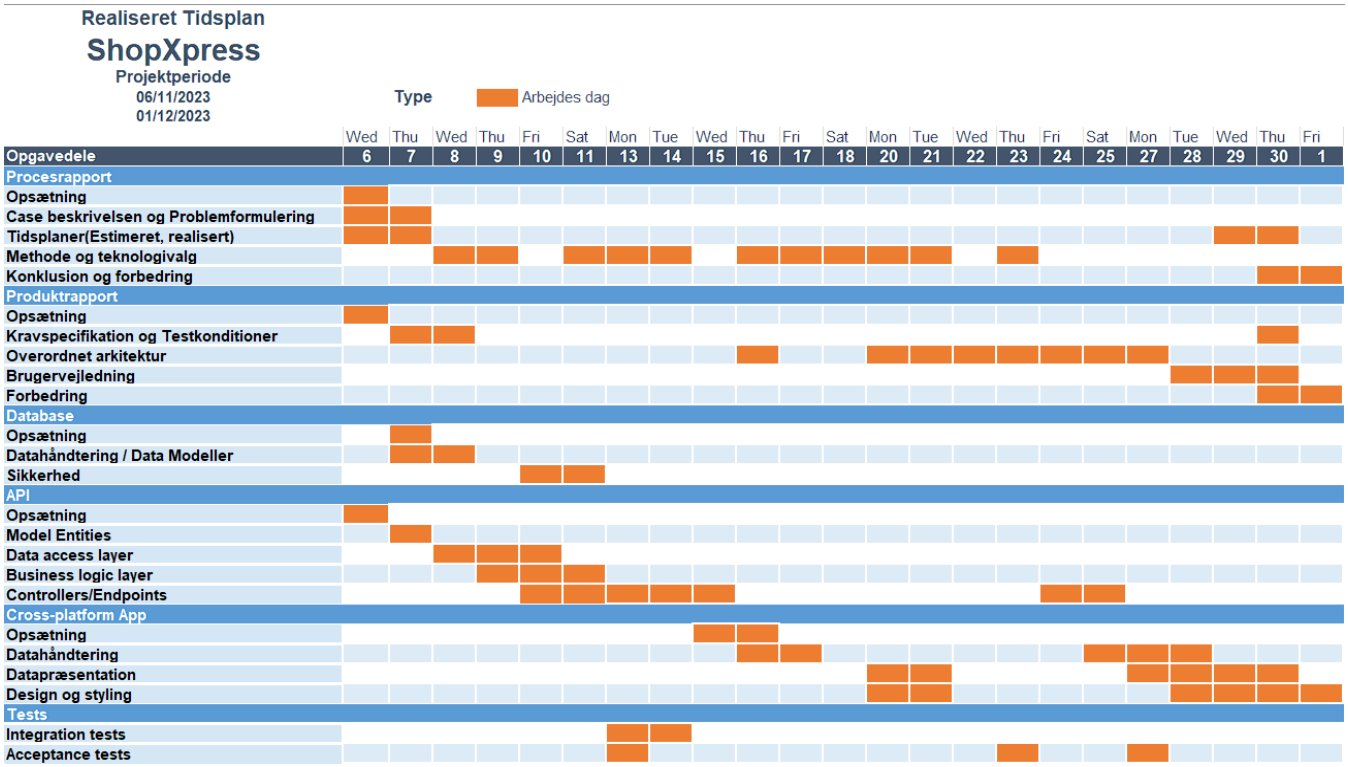
Entity framework Core spillede en vigtig rolle i skabelsen af projektet. Evnen til at abstrahere kompleksiteten af databaseinteraktioner ved at tillade os at arbejde med C#-objekter, skåne os fra de rå SQL queries, og derudover forbedre kodelæsbarhed og vedligeholdelse. Brug af ORM (Object-Relational Mapping) til at skabe databasetabeller og relationer baseret på C# klasserne. Derudover konfigurerer de komplekse relationer mellem entiteter og håndhæver forskellige begrænsninger på vores egenskaber (data) ved at bruge IEntityTypeConfiguration, leverer separate filer fra vores C#-modelklasser, hvilket forbedrer læsbarheden og vedligeholdelsen. At kunne bruge migreringer til at anvende ændringerne til databasen, i stedet for at skrive manuelle scripts og have potentielle fejl i databasen.
Disse robuste funktioner forbedrer i bund og grund vores udviklingsproces betydeligt, hvilket fremmer øget produktivitet. Entity Framework Core står som et fundament i vores projekts arkitektur og tilbyder et sæt værktøjer, der ikke kun forenkler komplekse databaseoperationer, men også bidrager til den overordnede effektivitet og succes af vores udvikling.

AFGRÆNSNING

Hvis jeg havde mere tid til projektudvikling, ville jeg forbedre Flutter-appen ved at inkorporere flere funktioner for at forbedre sikkerheden, optimere ydeevnen, forfine æstetikken og forbedre brugervenlighed. Derudover ville jeg udvikle en dedikeret webapplikation til administratorer, hvilket forenklede databasehåndteringsopgaver. Næste skridt til at forbedre projektet ville være implementering af en Blob Storage til at gemme produktbilleder og videoer. Jeg vil have udviklet implementeringen af automatiseret test i API'et, herunder både unit-test og integration-test, for at sikre at alle funktionaliteter fungerer efter hensigten.Til sidst vil jeg afsætte mere tid til at forbedre API'en ved at introducere yderligere controllers og endpoins, såsom at skabe en specialiseret controller til at administrere lokationer og tilbyde brugerne mere fleksibilitet ved at have mere end én lokation.

REALISERET TIDSPLAN

Denne plan viser et overblik over den realisert tid for de forskellige opgaver, der er involveret i skabelsen af produktet. Realisert tidsplan kan sammenlignes med den estimeret tidsplan, for at vise nøjagtigheden af den indledende estimering, der blev foretaget i starten af produktionen.



KONKLUSION

Det endelige produkt løser for det meste det problem, der er beskrevet i problemformuleringen. Brugere kan med fordel bruge produktet til at udforske en bred vifte af produkter og uden besvær foretage køb, alt sammen fra deres hjem. Dette løser behovet for traditionelle fysiske butikker, herunder at navigere i trafik, navigere i omfattende produktudvalg og udholde lange kassekøer. Både forbrugere og administratorer kan trygt engagere sig i systemet, forsikret om, at deres data er sikkert opbevaret, og at systemet opretholder et højt niveau af tilgængelighed og stabilitet.
Igennem udviklingsprocessen stødte jeg på forskellige udfordringer vedrørende både frontend og backend aspekter af projektet. API'et viste sig at være mere komplekst og ressourcekrævende end først forventet. Jeg stod over for problemer med funktionaliteten af CartsController og OrdersControllers endpoints, der ikke fungerede, som det skulle. For at løse disse udfordringer, brugt ekstra tid på at fejlfinde og løse problemerne. Dette involverede tilføjelse af Services til mine controllere og implementering af Transitions. Disse problemer resulterede i, at der blev brugt mindre tid på frontend-applikationen. På trods af tilbageslagene bidrog disse problemer til forbedringen af systemets stabilitet og funktionalitet.

LOGBOG

-Uge 45-

Mandag - 06.nov.2023
Intro, information omkring forløbet og svendeprøven.
Oprettelse af projektstyring og versionstyring (Azure DevOps Board og Azure Repo Git).
Opstart på Estimeret tidsplan, tegne systemoversigt og database diagrammer.

Tirsdag - 07.nov.2023
Videreudvikling af problemformuleringen og case beskrivelse, og færdiggøre af Estimeret tidsplan.
Fortsæt med at arbejde på Azure DevOps projektstyring.
Opstart af oprettelse af kravspecifikation, tilføjede krav med forklaring til forbrugere og administratorer.

Onsdag - 08.nov.2023
Talt med Lars omkring case beskrivelse, problemformuleringen og projektets kravspecifikation.
udvikling af egenskabskonfigurationer og seed produkternes data.

Torsdag - 09.nov.2023
Talt med Frank omkring case beskrivelse og problemformuleringen og fik det godkendt.
Fortsæt med at arbejde på modellerne og begyndte at arbejde på repositories og UnitOfWork.
Videreudvikling af kravspecifikation, og opstart af metode og teknologivalg afsnittet i procesrapporten.

Fredag - 10.nov.2023
Opstart på Bussines Logic Layer af API'et. Udvikling af DTO klasser for hver modelklasse, oprettelse af Authentication Service med funktioner.
Videreudvikling af metode og teknologivalg afsnittet i procesrapporten.

Søndag - 12.nov.2023
Opstart på API'et controller. Oprettet endpoints for AccountsController og CategoriesController.

-Uge 46-

Mandag - 13.nov.2023
Videreudvikling af API'et kontrollere. Oprettet endpoints for CategoriesController, ProductsController, og OrdersController.
Videreudvikling af teknologivalg afsnittet i procesrapporten.

Tirsdag - 14.nov.2023

Videreudvikling af API'et controllers, tilføjede endpoints til CartsController, OrdersController og ProductsController. Rettede fejl i DTO-klasser.

Onsdag - 15.nov.2023
Opstart på Flutter appen, konfigureret strukturen og filerne.
Videreudvikling af API'et controllers og endpoints for hver controller. Foretaget ændringer i dto-klasser og enhedskonfigurationer.
Opstart på ShopXpress.Tests, konfiguration af test-services og autentificering og autorisation.

Torsdag - 16.nov.2023
Videreudvikling af integration test til ProductsController. Tilføjede metoder for hver endpoint i ProductsController. Tilføjet faste brugerdefinerede temaer til Flutter-appen, såsom appbar, bundark, chip, knap, tekstfelt og teksttemaer. Tilføjede konstantværdier for farver, api-URL'er, billeder, enums, størrelser og tekster. Sidst tilføjet http-hjælperklasse til CRUD api-anmodninger.

Fredag - 17.nov.2023
Videreudvikling af integration test, implementere CRUD-testmetoder til ProductControllerTests for hver responstype og fikset fejl i modelklasser.

-Uge 47-

Mandag - 20.nov.2023
Opstart på database arkitektur og dokumentation i produktrapporten.
Arbejdet på metode og teknologi valg afsnittet i procesrapporten.
Videreudvikling af flutter appen.

Tirsdag - 21.nov.2023
Foretaget ændringer til DTO-klasser for at indeholde korrekte egenskaber. Rettede fejl i controllere relateret til roller. Seedede roller til databasen.
færdiggjort database arkitektur og dokumentation i produktrapporten.
Undersøgt on Flutter.

Onsdag - 22.nov.2023
rettede fejl i seedede kategorier.
Opstart på API sektion i produktrapporten.

Torsdag - 23.nov.2023
Arbejdet videre på væsentlige elementer og API sektioner i begge rapporter.

Fredag - 24.nov.2023
Løste problemer med CartsController ved at implementere CartService og gjort den klar til at blive publiceret.
Arbejdet på test og sikkerhedssektioner i produktrapporten.

-Uge 48-

Mandag - 27.nov.2023
Arbejdet på versionsstyring, projektstyring og hostssektioner i produktrapporten.
Rettede fejl i OrderService og OrdersController ved at implementere transaktioner.
Undersøgelser og videreudvikling af datahåndtering i appen.

Tirsdag - 28.nov.2023
Opstart på brugervejledninger og transaktionsafsnittet i produktrapporten.
Arbejdet på app sektion i produktrapporten.
Rettede fejl i OrderService ved at implementere transaktioner.
Videreudvikling af datapræsentation i appen.

Onsdag - 29.nov.2023
Arbejdet på brugervejledninger til Swagger i produktrapporten.
Implementeret realisert tidsplan til procesrapporten.
Videreudvikling af datapræsentation i appen.
Hostet API, SQL Server og SQL databasen.
Oprette databasediagram til produktrapporten.

Torsdag - 30.nov.2023
Arbejdet på brugervejledninger og app sektion til Flutter appen i produktrapporten.
Videreudvikling af app sider og datahåndtering.
Arbejdet på afgrænsningsafsnittet i procesrapporten.
Arbejdet på konklusionsafsnittet i begge rapporter.

Fredag - 01.dec.2023
Gennemgik begge rapporter til sidste finpudsning.
Videreudvikling af appen, skubbede de sidste ændringer til git inklusive Flutter APK-filen. Jeg kommer til at bruge weekenden til at afslutte Flutter-appen.