Agora

Procesrapport

Svendeprøve efterår 2025

Stefan Lynge Hvilsom
13-11-2025

Titelblad

Techcollege Aalborg,
Struervej 70,
9220 Aalborg

Elev:
Stefan Lynge Hvilsom

Firma:
Skole oplæring

Projekt:
Agora

Uddannelse:
Datatekniker med special i programmering

Projektperiode:
20/10-2025 – 20/11-2025

Afleveringsdato:
13/11-2025

Fremlæggelsesdato:
20/11-2025

Vejledere:
Frank Rosbak
Lars Thise Pedersen

Index

Læsevejledning

Denne procesrapport bør læses i sammenhæng med produkltrapporten, for at opnå en fuld forståelse af Agora projektet. Anbefalingen er at starte med produktrapporten, da denne dokumenterer det færdige system og dens tekniske implementering.

Produktrapporten beskriver systemets arkitektur, funktionalitet og tekniske overvejelser der behandles i procesrapporten.

I procesrapporten følger du udviklingsforløbet fra ide til færdigt produkt. Rapporten er struktureret kronologisk og dokumenterer de beslutninger, udfordringer og justeringer der karakteriserede udviklingsprocessen.

Tekniske termer og forkortelser fortklares når de bliver nævnt. For specifikke implementeringsdetaljer henvises der til de relevante afsnit i produktrapporten.

Ved at læse rapporterne i den anbefalede rækkefølge opnår du en sammenhængende forståelse af både produktets endelige form og de processer der førte til dens realisering.

Forord

Dette projekt Agora repræsenterer udviklingen af et socialt medie med særlig fokus på europæiske værdier om databeskyttelse og bruger kontrol. Formålet med procesrapporten er at dokumentere den rejse, der har ført fra den oprindelige ide til det færdige produkt.

Gennem denne rapport vil læseren følge udviklingsprocessens forskellige faser, fra de indledende overvejelser om arkitektur og teknologi, gennem implementeringsfasen med dens udfordringer og løsninger, til de endelige tests. Rapporten beskriver de valg der er truffet undervejs, og begrunder disse ud fra tekniske og funktionelle overvejelser.

Agoraprojektet er udviklet som en platform, der kombinerer de sociale interaktioner som brugerne forventer fra et socialt medie, med den gennemsigtighed og datakontrol, der er nødvendig i dagens digitale samfund. Rapporten reflekterer over, hvordan disse tilsyneladende modstridende mål er forenet i denne løsning.

Til læseren ønsker jeg en interessant og inspirerende læseoplevelse.

Stefan lynge Hvilsom

Indhold

Casebeskrivelse

Der er kommet et stigende behov, for et europæisk socialt medie. Så vi kan være mere selvstændige, og ikke være afhængige af de amerikanske varianter. Da vi i den senere tid har oplevet i en større grad, at vi ikke stole på de amerikanske sociale medier. Dette har ført til en større mistro og misinformation. Og dermed skabe en platform der er mere transparent.

Dette ville kunne komme de 450 millioner europæere til gode, da vi ville kunne opbygge den til at bruge europæiske love og regler for internet brug.

Problemformulering

Hvordan kan et europæisk socialt medie bygges på europæiske værdier som privatliv og brugerkontrol for at gøre brugerne mere selvstændige?

Afgrænsning

Her præciserer vi projektet omfang og de prioriteringer, der er foretaget for at sikre fokuseret og realiserbar løsning.

Inkluderede elementer:

- Bruger registrering og autentifikation med JWT token
- Oprettelse og håndtering af opslag
- Realtid kommentarfunktion via SignalR
- GDPR kompatible funktioner
- Basis roller og adgangskontrol (bruger/admin)
- Responsiv webside i Svelte

Eksluderede elementer:

- Dedikeret profil med adgang til at slette konto eller se egne opslag
- Evnen til at følge personer eller selv blive fulgt
- Evnen til at like opslag
- Der er ingen dedikeret algoritme til at styre hvilket indhold man kan se
- Der er ingen søgefunktion
- Mobilapp til IOS og android (kun webbaseret løsning)
- Push notifikationer udenfor browseren
- Cookie banner
- Billede og video upload

Begrundelse for afgrænsning

Valgene er truffet for at sikre, at projektets kernefunktioner implementeres tilfredsstillende inden for den tildelte tidsramme. Fokus har været at lave et stabilt fundament, og flere af dde fravalg er lavet fra API side så der mangler bare implementering af dette.

Projektplanlægning

Projekt er planlagt over en periode på 4 uger med en samlet estimeret arbejdsdage på 25 dage. Planlægningen er opdelt i faser, der følger en logisk rækkefølge i udviklingen af systemet. *Se figur 1 for den fulde tidsplan*

Uge 1: Projektplanlægning og arkitektur

- Projekt opsætning og teknologivalg
- Datastruktur samt DatabaseKontext
- DTO'er og interfaces

Uge 2: Backend udvikling

- Services
- Utilities
- Opsætning af JWT samt SignalR
- Opsætning af controllers

Uge 3: frontend og rapport

- Implementering af login side
- Implementering af register side
- Implemtering af dashboard
- Produktrapport startes

Uge 4: Rapport og test

- Test og produktrapport færdiggøres
- Processrapport

Samlet estimeret arbejdsdage: 25 dage

Arbejdsfordeling

Som enkeltudvikler på projekter har jeg ansvaret for alle dele af udviklingsprocessen. Arbejdsfordelingen er organiseret efter faglige domæner:

Teknisk implementering

- Database design med Entity Framework Core
- .NET 9 Web API med controllers og services
- Svelte frontend med Typescript
- Azure deployment og konfiguration

- Vercel deployment

Dokumentation og kvalitetssikring

- Produktrapport med tekniske specifikationer
- Procesrapport med udviklingsdokumentation
- Test af enheder og integration

Projektstyring

- Tidsplanlægning og prioritering
- Kvalitetssikring og code review
- Vejledermøder og statusopdateringer
- Afleveringsforberedelse

Denne fordeling sikrer en helhedsorienteret tilgang til projektet, hvor tekniske og dokumentationsmæssige aspekter koordineres for at opnå et sammenhængende resultat.

Metode og teknologivalg

.NET Web API

Valget af .NET 9 som backend teknologi er baseret på en velovervejet evaluering af projektets specifikke krav og de tilgængelige teknologiers styrker.

Overvejelser angående Node.js som alternativ

Arkitektur og ydeevne

Node.js's event drevne, single-threaded arkitektur er rigtig god til køre mange I/O (input/output) applikationer med mange samtidige forbindelser. Imidlertid for et socialt medie som Agora, hvor CPU intensive operationer som billedbehandling, komplekse databaseoperationer og kryptografiske beregninger vil være almindelige. .NET's mulighed for at udnytte flere CPU-kerner samtidig giver en klar fordel for vores system.

Type sikkerhed og vedligeholdelse

:NET med C# har en indbygget og streng type sikkerhed, som fanger fejl allerede når koden kompileres. Dette betyder at mange fejl opdages, inden programmet overhovedet kører.

Typescript forbedrer typesikkerheden i Node.js, men den er valgfri og ikke lige så grundlæggende. Fejl kan kan derfor først opdages under kørsel.

For Agora er .NET's typesikkerhed vigtig, fordi den hjælper med at undgå fejl, der kunne påvirke mange brugere. Det gør igså koden lettere at vedligeholde og udvide over tid.

Django som alternativ

Django er et populært Python webframework, der tilbyder hurtig udvikling. Dog vurderes det ikke som det ikke som det som det optimale valg til Agora af følgende årsager:

Ydeevne

Pyton og Django har generelt en lavere ydeevne end .NET, det gælder især ved CPU tunge opgaver. Dette ville begrænse Agoras evne til håndtere mange brugere samtidig.

Type safety

Python er et dynamisk skrevet sprog, hvilket betyder færre fejl opdages under udvikling. .NET's strenge regler giver det en mere sikker kode.

Realtids funktionalitet

Django tilbyder ikke lige så god realtidskommunikation som .NET g'r med SignalR. Dette er afgørende for måden som Agora fungerer på.

Skalerbarhed

.NET's for AOT-kompilering og bedre ressourceudnyttelse giver en giver langsigtet skalerbarhed.
AOT betyder at .NET koden kompileres til maskinkode for programmet kører.
Selvom Django er fremragende til prototyper og mindre projekter, blev .NET valgt på grund af dens overlegne ydeevne og sikkerhed til at skalerer Agora til en enterprise størrelse.

Realtids kommunikation

.NET SignalR tilbyder en komplet løsning til realtis kommunikation med:

- Automatisk genoprettelse af forbindelser ved netværksfejl
- Indbygget understøttelse af mange brugere Redis og Azure
- Integreret sikkerhed og adgangskontrol
- Strenge typeregler gennem .NET

Redis og Azure giver begge hurtige databaser til at håndtere realtids kommunikation.

Node.js altarnativer som Socket.IO kan også håndtere realtids kommunikation, men det kræver mere manuel opsætning for at opnå samme funktionalitet og giver større chancer for fejl. SignalR's indbyggede funktinor funktioner sparer på udviklingstiden og det giver en mere sikker kommunikation.

Økonomi

- .NET har en lavere kost da den udnytter ressourcer bedre.
- Node.js har en højere pris da den er mindre effektiv med udnyttelsen af sine ressourcer.
- Python er den dyreste løsning, da den er den skal bruge de fleste ressourcer for lignende løsning.

Konklusion af teknologi valg

Efter en omfattende evaluering blev .NET 9 valgt baseret på følgende punkter:

- Ydeevne: .NET er bedre til at håndtere CPU intensive arbejde.
- Skalerbarhed: Bedre muligheder for at udvide systemet og det har flere ressourcer til at håndterer sikkerhed.
- Pris: Det er bedre til at håndterer sine ressourcer, hvilket resultere i en lavere pris.
- Produktivitet: Integrerede værktøjer sikre at man kan skrive koden hurtigere, og den det strenge regler gør at der bliver lavet mindre fejl.

Dette valg gør at Agora kan håndtere at vækst og kompleksitet, men den leverer en responsiv og pålidelig brugeroplevelse.

Entity framework Core

Entity Framework Core er valgt som ORM (Objekt Relational Mapper) til at håndtere kommunikationen mellem .NET applikationen og databasen.

Hvor EF Core?

- **Automtisk SQL-generering: EF Coreoversætter C# kode til SQL, så man undgår at skrive manuelle SQL-forespørgsler.**
- **Model- struktur:** Database strukturen defineres gennem C# klasser, hvilket sikre mindre fejl.
- **Migrationer:** Med migrationer er det nemt at versionere sine databaseændringer, hvor EF Core generere automatiske SQL-scripts.

Fordele for Agora

- **Kodevenlig databasetilgang:** Arbejder med C# objekter i stedet for komplekse SQL-scripts.

- **Automatisk håndtering af relationer:** nem opsætning af many-to-many og one-to-many relationer.
- **Krydspltforms kapabilitet:** Understøtter SQL Server, PostgresSQL, SQLite mv.

Alternativer som Dapper blev overvejet, men EF Core blev valgt på grund af dens højerer produktivitet og indbyttede funktioner til en projektstørrelse som Agora.

SQL Server

Valget af SQL Server som databaseplatform er baseret på projektet behov for struktur, skalerbarhed, sikkerhed og integration med .NET.

Sammenligning med alternativer

MongoDB (NoSQL)

- **Fordele:** Fleksibelt schema struktur, og den er god til ustruktureret data.
- **Ulemper:** Svagere til at håndtere relationer.
- **Hvorfor ikke Agora:** Da sociale medier naturligt gør brug af relationelle databaser (brugere, opslag, kommentarer).

PostgreSQL

- **Fordele:** Open source, JSON-funktioner og den er hurtig.
- **Ulemper:** Mindre integreret imed .NET, og den har færre enterprise værktøjer.
- **Hvorfor ikke Agora:** SQL Server tilbyder bedre integration med Azure og Entity Framework.

SQL Servers stykre til Agora

, Relationel struktur

- **Klare relationer:** Nem modellering af bruger,følger, opslag, kommentar hierakier.
- **Data integritet:** Foreign key begrænsniger som sikre ensartethed.

Azure integration

- **Skalering:** Automatisk skalering i et cloudmiljø
- **Backup:** Automatiske sikkerhedskopier til Azure Storage.
- **Sikkerhed:** Det kan integreres med Azure Active Directory.

Ydeevne og skalerbarhed

Læse og skrive mønstre

Agora har et typisk socialt medie brugsmønster:

- **Tunge læseoperationer:** Visning af dashboards og kommentarer udgør hovedparten af belastningen.

- **Moderate skriveoperationer:** Oprettelse af opslag, kommentarer og likes forekommer hyppigt, men de er mindre ressoucekrævende.

SQL Server er særligt velegnet til denne balance mellem læs og skriv, da dens række baseredestorage og effektive caching håndterer dashboard visning optimalt og samtidig med at den kan svare hurtigt ved nye opslag.

Skalering

Fase 1:

I de indladende fase anvendes en enkelt database instans, som håndterer både læse og skriveoperationer. Dette giver en simpel og omkostningseffektiv opsætning, der er optimal til projektets tidlige stadier.

Fase 2:

Når antallet af brugere vokser, indføres ekstra databasekopier, der er i at håndtere forespørgsler til dashboards og kommentarer. Dette frigør den originale database til at koncentrere sig om skriveoperationer som nye opslag og redigeringer.

Fase 3:

Ved meget stor vækstopdeles databasen i regionale dele, så brugere i f.eks. Europe og Nordamerika har hver deres database. Dette vil reducere ventetid og det vil belaste netværker mindre, når data hentes tæt på brugerne.

Svelte

Ved projektstart stod Agora overfor et afgørende valg af frontend teknologi. Efter omhyggelig evaluering blev SVelte valgt. Denne beslutning var baseret på en detaljeret sammenligning med de to primære alternativer: Vue.js og Blazor.

DOM vs Virtual DOM

For at forstå teknologivalget er det essentielt at forstå forskellen mellem traditionel DOM og Virtual DOM tilgangen.

- **Virtual DOM:** Vue.js bruger en Virtual DOM som er en abstraktion til den rigtige DOM. Når data ændres, opfdateres der først i den virtuelle DOM, som derefter sammenlignes med den rigtige DOM for at identificere minimale ændringer. Denne tilgang tilføjer et ekstra lag af kompleksitet, og dette vil derfor sænke hastigheden.

- **Kompilering:** Svelte eliminerer virtual DOM helt. Under kompilering analyserer Svelte komponenterne og generer en optimeret JavaScript kode, der direkte manipulerer DOM'en. Denne tilgang fjerner unødvendigt brug af ressourcer og det resulterer i hurtigere og mere effektiv kode.

Sammeligning Svelte vs Vue.js

Arkitektoniske forskelle

Vue.js følger den traditionelle virtual DOM baserede model, der kræver en runtime i browseren for at håndtere reaktivitet og komponentopdateringer.

Svelte derimod kompilerer komponenter til optimeret vanilla JavaScript, der arbejder direkte med DOM'en idem nogle mellemliggende lag.

Reaktivitet og State Management

Vue.js kræver specifikke funktioner og wrappers for at gøre data reaktive.

Svelte derimod gør al state automtisk reaktiv gennem sin kompileringstilgang.

Dette resulterer i en mere intuitiv udviklingsoplevelse kode.

Udvikler produktivitet

Vue.js tilbyder en velstruktureret, men omfattende komponentmodel med separate sektioner for template script og styles. Svelte forenkler denne model markant og reducerer mængden af kode, der kræves for at implemtere de samme funktioner
Her kan vi se en Svelte implementering fra dashbord

```
<script>
  let newPost = '';
  let posts = [];

    async function createPost() {
const content = newPost.trim();
if(!content) return;
}
</script>

<textarea bind:value={newPost}></textarea>
<button on:click={createPost}>Post</button>

```

Vi kan her se en lignende implementering i Vua.js

```
<template>
  <textarea v-model="newPost"></textarea>
  <button @click="createPost">Post</button>
</template>

<script>
export default {
  data() {
    return {
      newPost: '',
      posts: []
    }
```

```
13.   },
14.   methods: {
15.     async createPost() {
16.       const content = this.newPost.trim();
17.       if (!content) return;
18.
19.     }
20.   }
21. }
22. </script>
23.
```

I Agora dashboard ses Sveltes fordele tydeligt:

- **Ingen data() funktion:** Variabler deklareres direkte.
- **Inden methods sektion:** Funktioner defineres frit
- Direkte bind**:**value vs Vues v-model.

Denne forenkling reducerer kode meget og gør koden mere læsbar og vedligeholdelsesvenlig. Især i komplekse komponenter som der findes i dashboard med de mange variabler og funktioner.

Implemetering i Agora

Projektet specifikke krav

Agora som er en social medieplatform med behov for høj responsivitet og realtidsinteraktion gjorde ydeevne til en afgørende faktor. Svelte's kompileringstilgang viste sig ideel for dette projekt.

Produktivitet

Under udviklingen viste det sig, at Svelte's enkle komponentmodel og automatiske reaktivitet gjorde mere effektivt. Dette gjorde at der var mere tid til at fejlfinde vores komponenter så brugeren kan god oplevelse.

Konklusion på teknologi valg

Efter en omhyggelig evaluering blev Svelte valgt som Agora's frontend teknologi

I forhold til Vue.js

- Mere direkte og effektiv DOM kompilering
- Mindre kompleks state management
- Renere komponent arkitektur
- Forbedret produktivitet

Svelte's unikke kompileringstilgang, kombineret med dens enkle udviklingsmodel, viste sig at være den optimale løsning for Agora's behov. Platformens krav om høj ydeevne og effektive udvikling blev alle bedre opfyldt gennem dette valg.

Den praktiske erfaring underprojektet bekræftede, at Svelte leverer på sine løfter om en mere direkte og effektiv tilgang til frontend udvikling, hvilket har gavnet både udviklingsprocessen og den endelige brugeroplevelse.

Væsentlige elementer fra produktrapporten

Overordnet system arkitektur

Agorasystemet følger en lagdelt arkitektur baseret på på SOLID principperne, hvilket sikre en vedligeholdesesvenlig, testbar og skalerbar kodestruktur.

Systemet er organiseret i fire primære lag:

- **Presentation layer:** Controllere der håndterer http forespørgsler.
- **Business layer:** Services med forretningslogik og domæneregler.
- **Data access layer:** AppDbContext og Entity Framework
- **Domain layer:** Modelle rog DTO'er der definerer systemets domænemodel.

Database design med Entity Framework Core

Databasen er designet med selv-referende relationer i både Posts og Comments, hvilket muliggør at hierakiske strukture uden ekstra tabeller.

Designvalg inkluderer:

- **Many-tomany relationer:** For User-Role og Follower
- **Soft delete:** Implementeret gennem IsDeleted flaget til GDOR.
- **Unikkebegrænsninger:** Som UserId/PostId i Like modellen.
- **Automatiske tidsstempler:** For CreatedAt og UpdatedAt.

Sikkerhed

JWT autentikation

Systemet emplementerer en komplet JWT baseret autentifikationsløsning med:

- Access token med en konfigurerbar levetid.
- Refresh token gemt som http only cookie

- Rolle baseret autorisation med claim
- Automatisk token fornyelse I frontend.

GDPR

Agora implenterer flere GDPR-krav igennem:

- Right to be forgotten via deaktiveringsproces
- Data export funktion til brugerdata
- Autmatisk sletning af inaktive brugere efter 6 måneder.
- Dataminimering gennem DTO'er.

Realtidsfunktionalitet med SignalR

SignalR implementerer realtids kommunikation til:

- Øjeblikkelige opdateringer af opslag og kommentarer.
- Brugerspecifikke grupper baseret på UserId.
- Integration med eksisterende autentifikation.
- Event baseret kommunikation i services.

Frontend akitektur i Svelte

Frontenden følger en SPA (Single Page Applikation) arkitektur med:

- Reaktiv state management uden unødvendig kode.
- Automatisk token fornyelse med refresh logik.
- Tema system med localStorage.
- Responsivt design med CSS

Teststrategi

Unit tests

- Model validering for Posts og Users.
- Forretningslogik i services.
- FluentAssertions for læsbare tests

Contract Tests

- API endpointed validering gennem http anmodninger
- Komplette brugerscenarier fra registrering til logout
- Fejlhåndtering og statuskoder

Kritiske tekniske implementeringer

AutoMapper og DTO'er

Systemet bruger AuitoMapper til at sikre dataminimering ved kun aty eksponere nødvendige data gennem DTO'er. Dette adskiller klar datakontrakter fra modellerne.

Dependency injection

Hele systemet er bygget omkring dependency injection, hvilket gør det:

- Skalerbart gennem klare ansvarsområder.
- Vedligeholdelsesvenligt med løst koblede komponenter

Fejlhpndtering og logging

Systemet implementerer konsistent fejlhåndtering med:

- Strukturerede fejlmeddelser.
- Sikkerhedslogning uden at lække systemoplysninger.

Disse elementer udgør sammen den tekniske kerne der muliggør Agora's funktionalitet som et sikkerhedsorienteret, europæisk socialt medie.

Realiseret tidsplan

Se figur1 i bliq

Uge 1: Projektplanlægning or arkitektur (20/10-24/10)

- Problemformuleringen og casebeskrivelsen udarbejdes.
- Kravsspecifikation færdiggjort.
- Git repositories oprettet for frontend og backend.

- .NET Web API og Svelte projekt opsat.
- Databasemodellern og DbContext implementeres.
- DTO'er og interfaces færdiggjort.

Planen fuldt alle aktiviteter gennemført.

Uge 2: Backend udvikling (27/10-31/10)

- AutoMapper og PasswordHasher implementeret.
- SignalR Hub og services udviklet.
- Alle services færdiggjort med forretningslogik.
- JWT konfiguration og dependency injection opsat.
- Admin system og rolle baseret autorisation.

Planen fuldt, backend komplet implementeret.

Uge 3: Frontend (03/11-07/11)

- Login og registreringssider implementeret i Svelte.
- Dashboard komponent med state management udviklet.
- Azure App Service og SQL Database konfigureret.
- Frontend deployed til Vercel
- Brugervisning og sletningsfunktionalitet implementeret.
- Produktrapport påbegyndt

Uge 4: Dokumentation og afslutning (07/11-12/11)

- Produktrapport færdiggjort med tekniske specifikationer.
- Procesrapport udarbejdet.
- Endelige rettelser og kvalitetssikring gennemført.
- Komplet dokumentation afsluttet

Planen fulgt og dokumentation er færdiggjort.

Afvigelser fra planen

Positive afvigelser:

- Frontend udvikling gik hurtigere end forventet.
- Azure/Verdel deployment proces ukompliceret
- GDPR funbktionalitet implementeret tidligere end planlagt.

Mindre justeringer:

- Dokumentationen kræver fokus i uge 4
- SignalR integration tog ekstra opmærksomhed i uge 2.

Negative afvigelser:

- 12/11 er der en rettelse angående azure. Mine 700 student credit blev slugt op og de er forsvundet på omkring 7 dage.

Konklusion

Agora projektet har succesfuldt demonstreret, hvordan et europæisk socialt medie kan opbygges på europæiske værdier som privatliv og brugerkontrol. Gennem en velplanlagt og struktureret udviklingsproces er der skabt en platform, der kombinerer essentiel social medie funktionalitet med streng databseskyttelse og gennemsigtighed.

Projektet har opfyldt de oprindelige mål ved at levere en komplet løsning med brugerrigistrering, autentifikation via JWT tokens, oprettelse og håndtering af opslag, samt realtids kommentarfunktioner gennem SignalR. Platformen implementerer GPDR kompatible funktioner inklusive data export og ”Right to be forgotten”, hvilket styrker brugerkontrollen over egne data.

Den tekniske implementering med .NET 9 backend og Svelte frontend har vist sig at være et optimalt valg. .NET’s stærke typesikkerhed og skalerbarhed sammenmed Svelte’s kompileringstilgang uden virtual DOM har resulteret i en højtydende vedligeholdesesvenlig platform. Integrationen af Entity Framework Core og SQL Server har sikret en robust datalagring med europæiske datalokalisering.

Projektets afgrænsning har været afgørende for at opnå en fokuseret og realiserbar løsning inden for den tildelte tidsramme. Ved at prioritere kernefunktionaliteten er der skabt et solidt fundament, der kan udvides fremadrettet med yderligere funktioner som følgesystem, likes og profilmuligheder.

Agora repræsenterer et væsentligt skridt med mere selvstændige europæiske digitale platforme, der respekterer brugerens privatliv og datakontrol. Platformen arkitektur og implementering viser, at det er muligt at skabe konkurrencedygtige sociale medier baseret på europæiske værdier uden at gå på kompromis med funktionalitet eller brugervenlighed.

Diskussion

Valg af teknologier

Valget af .NET 9 som backend teknologi har vist sig at være optimalt for Agora's formål. .NET's stærke typesikkerhed har reduceret runtime fejl mærkant, og dens integration med Azure har muliggjort en problemfri deployment. SignalR har leveret pålidelig realtidskommunikation, hvilket er afgørende for en social medieplatform.
Svente som frontend teknologi har demonstreret klare fordele til traditionelle frameworks. Dens kompileringstilgang har resulteret i hurtig indlæsnings tider, som hjælper til med en mere responsiv brugeroplevelse.

Implementeringsudfordringer

Sikkerhed og GDPR

Implementeringen af GDPR-krav som "Right to be forgotten" og data export funktionalitet tilføjede kompleksitet til systemets arkitektur. Den valgte løsning med "soft delete" og dataminimering gennem DTO'er har dog vist sig effektiv til at balancere funktionalitet og overholde GDPR.

Realtidsfunktionalitet

Integrationen af SignalR for realtidsopdateringer af kommentarer og opslag introducerede udfordringer vedrørende forbindelse styring og fejlhåndtering. Disse blev løst gennem omhyggelig implementering af genoprettelsesmekanismer og fejlhåndtering.

Begrænsninger og fremtidige udvidelser

Projektet afgrænsning var nødvendig for at op en fokuseret plan, men efterlader plads til fremtidige forbedringer:

- **Manglende profilstyring:** Implementering af en dedikeret profilside ville styrke brugeroplevelsen.

- **Begrænset social interaktion:** Tilføjelse af følgesystem og likes ville engagement.
- **Mobiloptimering:** En dedikeret mobilapp ville udvide platformens rækkevidde.

Økonomiske overvejelser

Platformens enkle og rene design har fokuseret på essentiel funktionalitet, hvilket gør den tilgængelig for et bredt brugergrundlag. Den manglende algoritmiske dashboard sætter fokus på kronologisk visning og brugerkontrol, hvilket stemmer overens med platformens værdier og gennemsigtighed.

Konkluderende vurdering

Agora projektet har succesfuld demonstreret, at det er muligt at udvikle et konkurrencedygtigt socialt medie baseret på europæiske værdier. De tekniske valg har vist sig solide, og platformen leverer en pålidelig og sikker oplevelse. Selvom der er plads til forbedringer og udvidelser, repræsenterer Agora et væsentligt bidrag til udviklingen af mere etiske og brugercentrerede digitale platforme.
Projekter understøtter idéen om, at europæiske værdier om privatliv og datakontrol ikke uforenelige med moderne sociale medier.

Referencer

Teknologisk dokumentation

ASP.NET Core documentation

https://learn.microsoft.com/en-us/aspnet/core/

Entity Framework Core documentation

https://learn.microsoft.com/en-us/ef/core/

Svelte documentation

https://svelte.dev/docs

SignalR documentation

https://learn.microsoft.com/en-us/aspnet/core/signalr

Databeskyttelse

GitHub documentation

https://docs.github.com/

Azure App Service documentation

https://learn.microsoft.com/en-us/azure/app-service/

Vercel documentation

https://vercel.com/docs

Software biblioteker

AutoMapper documentation

https://docs.automapper.org/

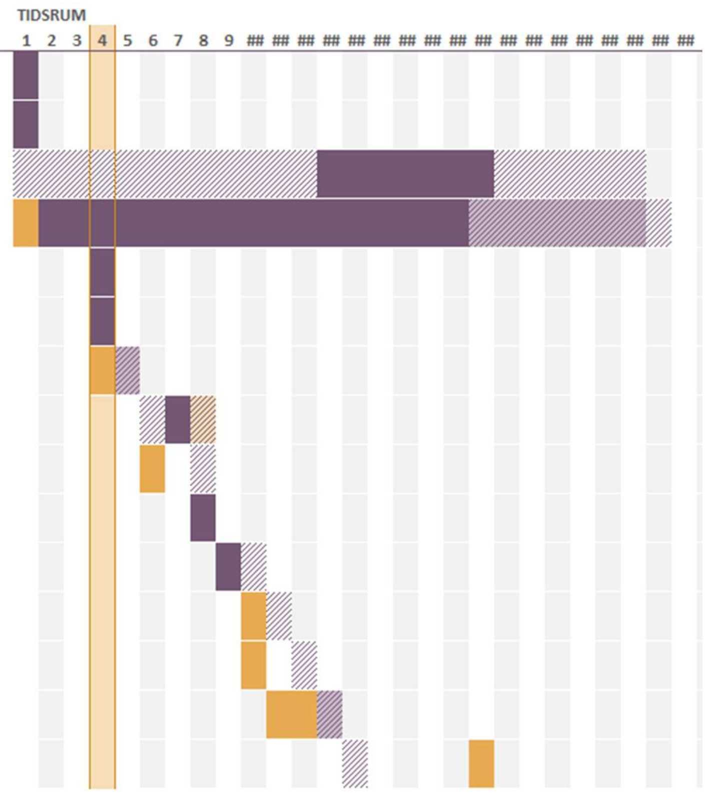
FluentAssertions documentation

https://fluentassertions.com/

Bilag

Bilag A: Tidsplan

AKTIVITET	STARTDATO FOR PLAN	VARIGHED FOR PLAN	FAKTISK STARTDATO	FAKTISK VARIGHED	PROCENTDEL FULDFØRT
Casebeskrivelse	1	1	1	1	100%
Problemformulering	1	1	1	1	100%
Rapport	1	25	13	7	100%
Tidsplan	2	25	1	25	75%
Models	4	1	4	1	80%
EF Config	4	1	4	1	80%
DTO/Interfaces	5	1	4	2	80%
Services	6	2	7	2	80%
Utilities	8	1	6	1	80%
Opsætning af JWT	8	1	8	1	100%
Controllers	9	2	9	1	80%
Front-Login	11	1	10	1	80%
Front-Register	12	1	10	1	100%
Front-Dashboard	13	1	11	3	80%
Testing	14	1	19	1	100%



Figur 1tidsplan

Bilag 2: Dagbog

Dagbog

- **20/10-2025**
  - I dag har jeg lavet problemformuleringen og casebeskrivelsen. Vi har samtidig også fået en overordnet forklaring på svendeprøveforløbet.

- **21/10-2025**
  - I dag hart jeg færdiggjort min kravspecifikation, samt lavet git til både frontend og api, Hvori jeg har lavet et nyt webapi project i .net samt et nyt svelte project. Jeg har også påbegyndt min tidsplan
- **22/10-2025**
  - I dag har jeg lavet min tidsplan, og lavet issues til min API så det er nemmere at komme i gang og følge hvad jeg mangler. Jeg har ikke lavet issues til min frontend endnu da der går lidt tid før jeg kigger på den.
- **23/10-2025**

  - I dag har jeg lavet mine modeller, der kan komme ændringer på et senere tidspunkt hvis jeg finder ud af at jeg mangler noget. Jeg har også lavet min dbcontext så alle mine relationer er rigtige, igen dette kan ændres hvis der er noget der skal bruges lidt anderledes i frontend. Jeg er også påbegyndt mine DTO'er, så jeg separerer mine modeller fra min service.
- **24/10-2025**
  - I dag har jeg færdiggjort De sidste DTO'er jeg manglede og alle mine interfaces, så det vil være nemmer at lave mine services og også nemmer at tilføje mere i fremtiden.
- **27/10-2025**
  - I dag har jeg lavet alle mappers til mine DTO'er og Modeller, samt lavet min passwordhaser så vi kan sikre brugernes informationer bedre.
- **28/10-2025**
  - I dag har jeg lavet min SignalR hub samt user,role og postservice
- **29/10-2025**
  - I dag har jeg færdiggjort mine services, samt opsat appsætting til jwt, jeg har også sat min program.cs op så der kun mangler controllers.
- **30/10-2025**
  - I dag har jeg færdiggjort alle mine controllers samt tilføjes admin user og manager til rollelisten, da den fremadrettet vil være admin only.
- **31/10-2025**
  - I dag satteJeg mine rappporter nogenlunde op og gjorde klar til at skrive dem, samt påbegyndte mit frontend projekt.
- **3/11 2025**
  - I dag har jeg lavet mine login og register sider, samt sat min api op til at køre htttps
- **4/11-2025**
  - Her begyndte jeg min dashboard side dette trak lidt tænder, men jeg fik sammensat hvad jeg tænkte var okay.
- **5/11-2025**
  - I dag oprettede jeg min azure til API og sql på azure, jeg fik også oprettet min frontend til Vercel, så jeg kunne teste det, og her slåssede jeg lidt med at få brugeren vist med navn, og slet knappen til at vise sig.
- **6/11-2025**
  - I dag fik jeg afsluttet produktet og vist navne på alle posts. Og jeg fik også påbegyndt min produktrapport.
- **7/11-2025**
  - Her startede jeg produktrapporten
- 8/11-2025
  - Fortsatte med produktrapport
- 9/11-2025
  - Fortsatte produktrapport

- 10/11-2025
    - Aflsuttede produktrapport og startede minm procesrapport
- 11/11-2025
    - Procesrapport i fuld gang
- 12/11 2025
    - Afsluttede procesrapporten og rettede fejl i begge rapporter