Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

Automatisert rapporteringstjeneste for bedriftsstatistikk
Systemdokumentasjon

Versjon 1.1

Dokumentet er basert på Systemdokumentasjon utarbeidet ved NTNU. Revisjon og tilpasninger til bruk ved IDER, DATA-INF utført av Carsten Gunnar Helgesen, Svein-Ivar Lillehaug og Per Christian Engdal. Dokumentet finnes også i engelsk utgave.

REVISJONSHISTORIE

Dato	Versjon	Beskrivelse	Forfatter
03/05/22	1.0	Opprettelse av dokument og førsteutkast.	Henrik Hammer, Anders Tuft Buanes, Markus Bjermeland, Mikael Aare
19/05/22	1.1	Oppdatering av figurer og formuleringer	Henrik Hammer, Anders Tuft Buanes, Markus Bjermeland, Mikael Aare

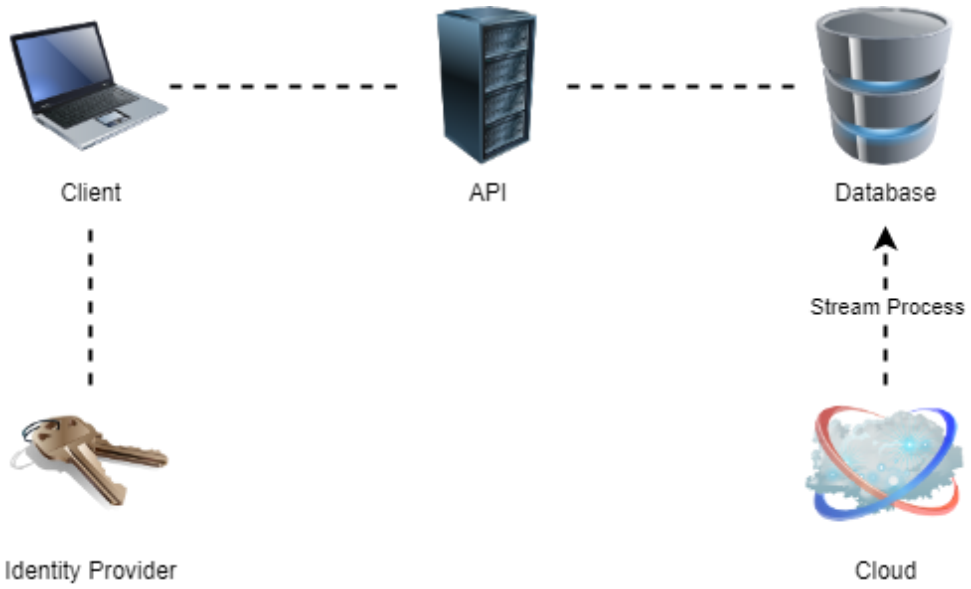
INNHOLDSFORTEGNELSE

1 INNLEDNING

Dette dokumentet er skrevet i sammenheng med bacheloroppgave ved Høgskulen på Vestlandet i vårsemesteret 2022. Hensikten med dokumentet er å gi en kort innføring i systemet som er utviklet. Dokumentet beskriver overordnet arkitektur, prosjektstruktur og tekniske detaljer om produktet som ble utviklet i prosjektet. Dette innebærer blant annet server-tjenester, sikkerhet, dokumentasjon av kildekode, og instruksjoner for installasjon og kjøring.

2 ARKITEKTUR

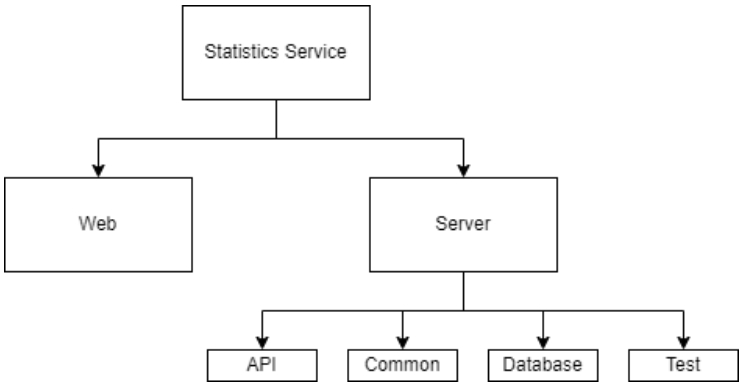
Løsningen tilbyr et brukergrensesnitt som konsumerer et programmeringsgrensesnitt for å fremvise statistikk. Kall til programmeringsgrensesnittet gjøres ved bruk av REST-prinsippene. Statistikken hentes fra en PostgreSQL-database som mottar data fra en skytjener. Figur 1 gir en overordnet beskrivelse av løsningens arkitektur.



Figur 1: Arkitekturskisse. Klienten kaller på API-et med autorisering fra Identity Provider. API-et gir tilgang til uthenting av statistikk fra databasen. Statistikken i databasen mottas fra en skytjener

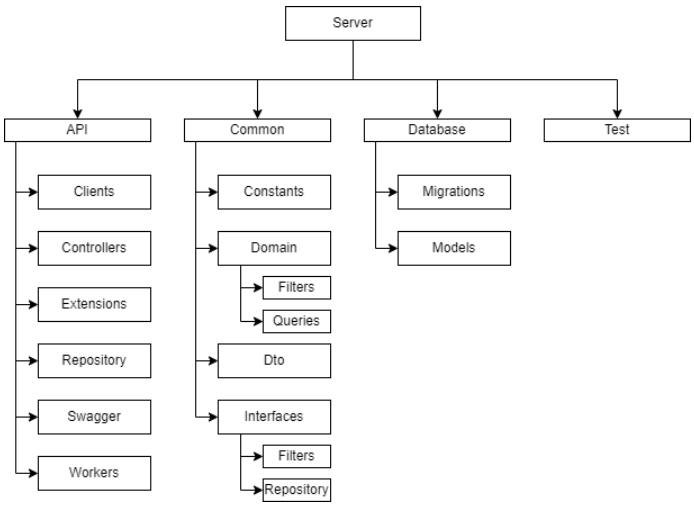
3 PROSJEKTSTRUKTUR

Løsningen er strukturert etter Clean Architecture-filosofien og MVC-arkitektur som begge vektlegger et klart skille mellom ansvarsområder. I praksis betyr dette at løsningen er delt inn i 5 prosjekter; Common, Database, API, Web og Test. Denne inndelingen er illustrert i figur 2.



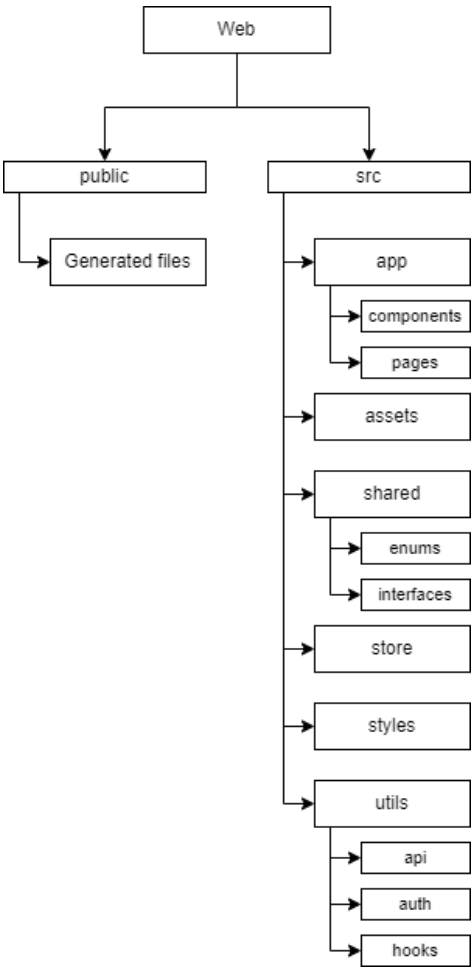
Figur 2: Illustrasjon av prosjektets overordnede struktur

Figur 3 gir en oversikt over serversiden av løsningen. Common definerer prosjektets domene, felles grensesnitt og forretningslogikk. Database definerer tabellene i databasen, samt *migrations* (endringer i modellen). API definerer programmeringsgrensesnittet og endepunktene som er tilgjengelig fra dette grensesnittet. Test definerer tester for prosjektet, dvs. eventuelle enhetstester, integrasjonstester og ende-til-ende tester.



Figur 3: Illustrasjon av serverapplikasjonens struktur

Videre definerer Web klientsiden i løsningen. Klientsiden er hvor brukergrensesnittet er utviklet. Figur 4 gir en oversikt over strukturen på klientsiden.



Figur 4: Illustrasjon av klientens struktur

Web definerer React-prosjektet som er ansvarlig for klientsiden. Prosjektets kildekodemappe inneholder følgende mapper:

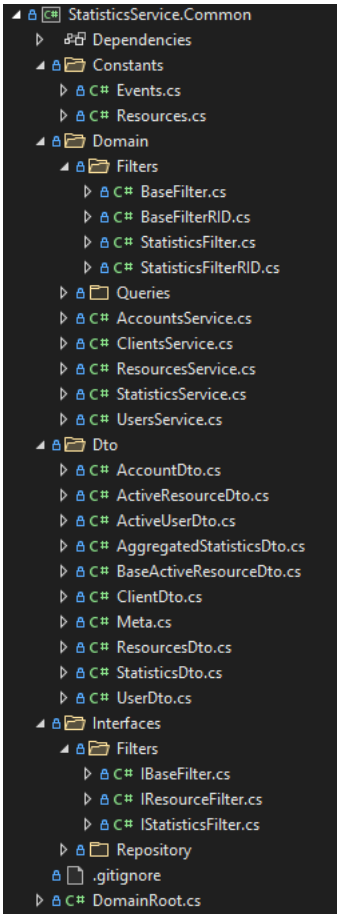
- App – inneholder sider og komponenter som brukes i prosjektet, samt filer for kjøring av klienten.
- Assets – inneholder statiske filer som bilder.
- Shared – inneholder interfaces og enums, brukt i hele prosjektet.
- Store – inneholder logikk for lokallagring og sentral tjeneste for å lagre eller hente verdier.

- Styles – inneholder stylingfiler som brukes i prosjektet.
- Types – automatisk generert mappe som inneholder definisjoner av TypeScript-typer.
- Utils – inneholder hjelpefunksjoner brukt i prosjektet, eks. funksjon for å gruppere tall med datoer for visning i grafer.

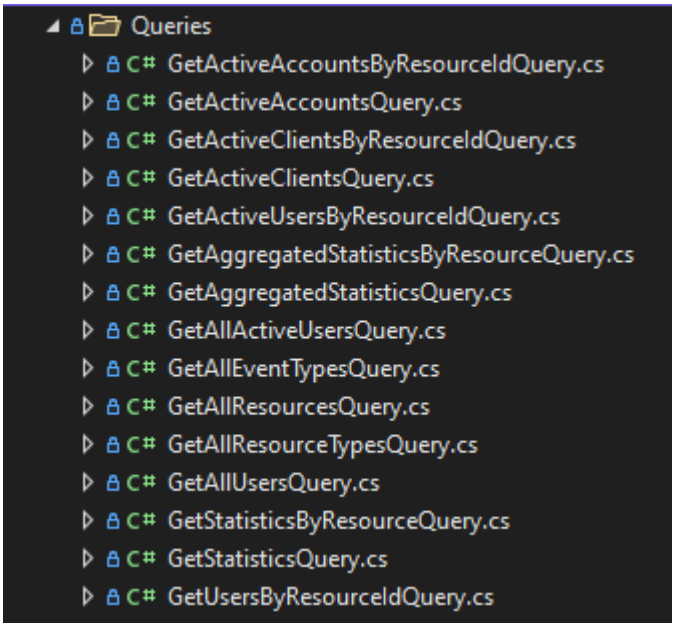
StatisticsService.Common inneholder mapper for følgende:

- Constants – konstanter brukt i løsningen.
- Domain – definisjoner av *filters* (innkapslinger for forespørselsparametere), *queries* (spørringer sendt fra API-laget ved datahenting) og *services* (tar hånd om spørringer etter instansiering).
- Dto – overførbare dataobjekter for dataoverføring til klientsiden.
- Interfaces – grensesnitt brukt i hele løsningen, blant annet for *filters* og *repositories*.

Oversikt over filstrukturen kan sees i figur 5 og 6.

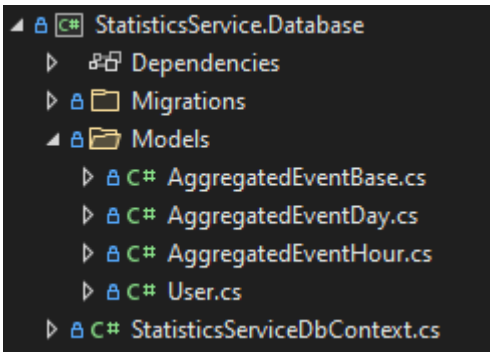


Figur 5: Illustrasjon av mappe- og filstruktur i StatisticsService.Common



Figur 6: Illustrasjon av filer i Queries-mappen i Common

StatisticsService.Database inneholder mapper for modeller og migrasjoner, samt en klasse som tilgjengeliggjør databasekonteksten for resten av prosjektet (se figur 7).

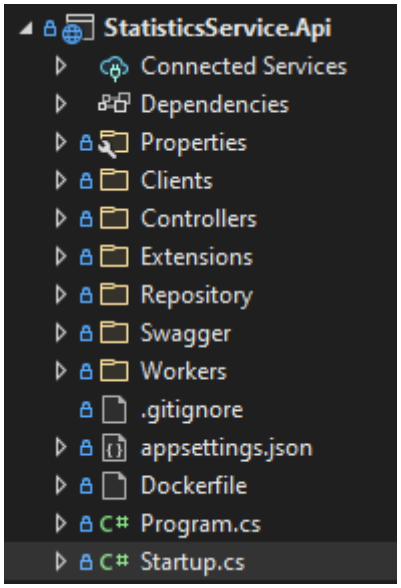


Figur 7: Illustrasjon av mappe- og filstruktur i StatisticsService.Database

StatisticsService.API (se figur 8) inneholder klasser for oppstart og konfigurasjon av programmet, samt mapper for følgende:

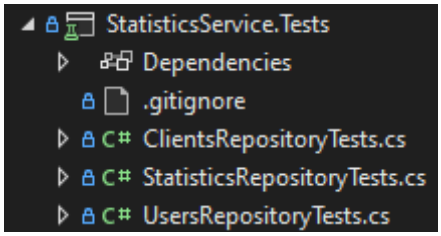
- Properties – inneholder oppstartsinnstillinger for programmet.
- Clients – inneholder hjelpeklasser for henting av data fra et eksternt API.
- Controllers – inneholder klasser som definerer alle endepunkter i programmeringsgrensesnittet.

- Extensions – inneholder diverse utvidelsesmetoder for spørringer, initialisering av autentisering, og CORS.
- Repository – inneholder klasser som definerer forretningslogikken mot databasen.
- Swagger – inneholder konfigurasjonsfiler for Swagger.
- Workers – inneholder klasser som definerer automatisk intervallbasert henting av data fra en ekstern tjeneste.



Figur 8: Illustrasjon av mappe- og filstruktur i StatisticsService.Api

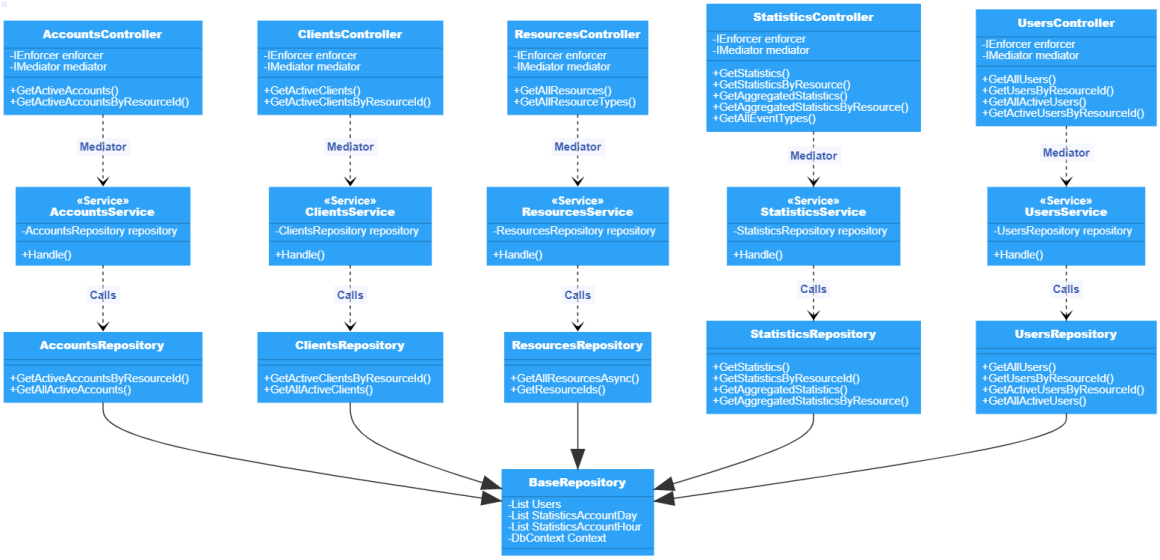
Avslutningsvis inneholder StatisticsService.Tests filer for enhetstesting av diverse klasser i prosjektet (se figur 9).



Figur 9: Illustrasjon av mappe- og filstruktur i StatisticsService.Tests

4 KLASSEDIAGRAM

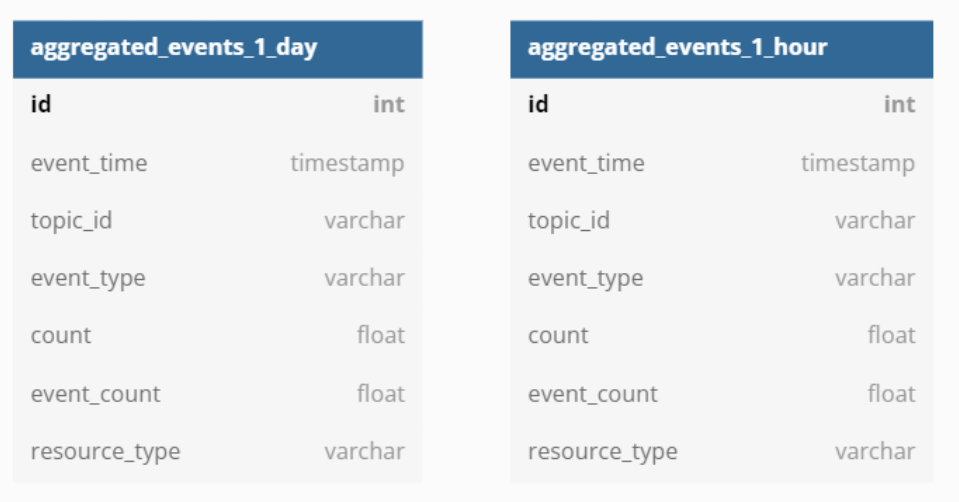
Det meste av forretningslogikk håndteres lagvis av kontrollere, *services* og *repositories.* Figur 10 skildrer hvordan de tre komponentene samhandler. Det figuren ikke viser er at kontroller-klassene arver fra ControllerBase-klassen i ASP.NET Core MVC-biblioteket. Hvilken *service*-metode som skal kalles fra kontroller håndteres av formidleren.



Figur 10: Klassediagram med relasjoner mellom kontrollere, Mediator-handlers (*services*) og *repositories*

5 DATABASEMODELL

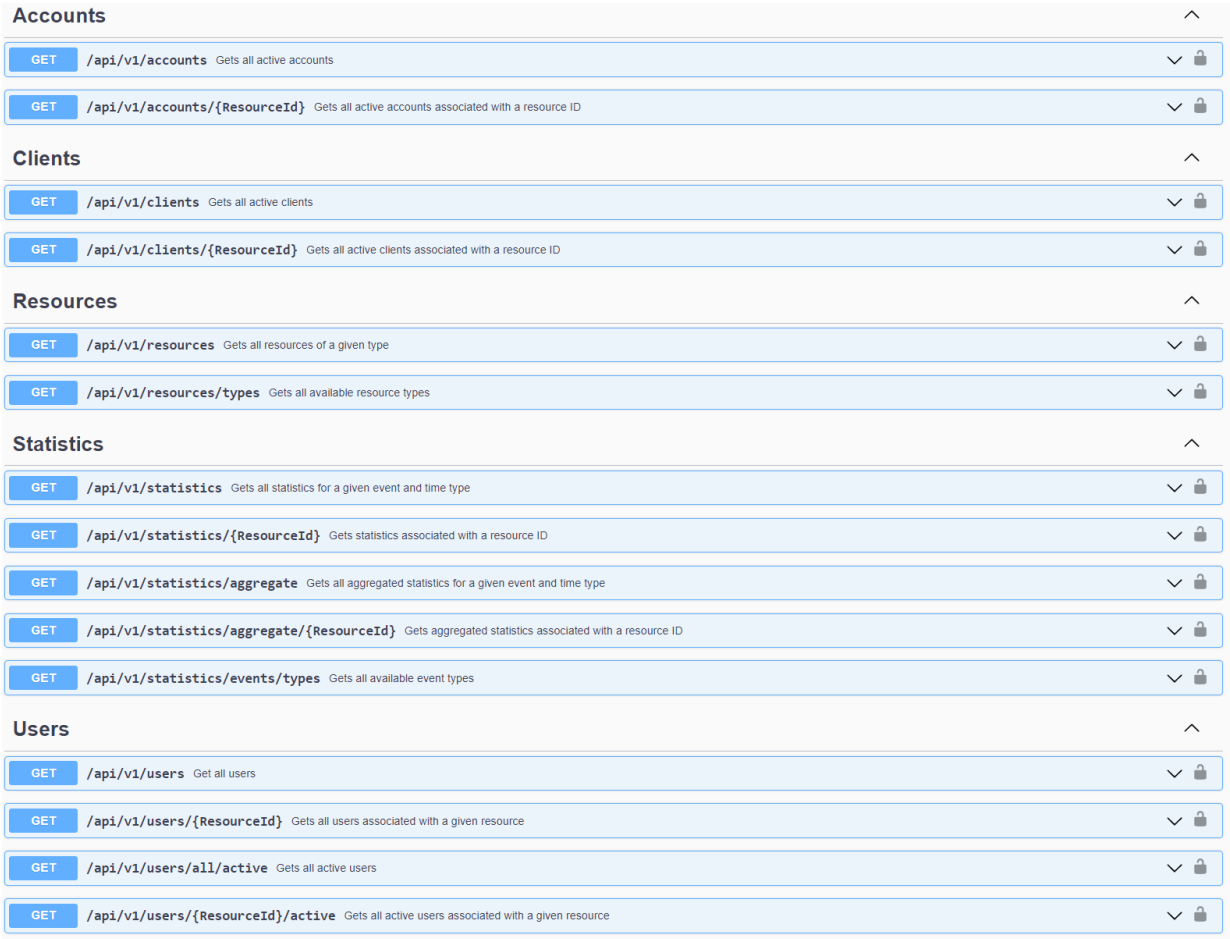
Databasen ble designet av oppdragsgiver. Det er to tabeller som vist i figur 11, henholdsvis gruppert på dags- og timesbasis. Dataen er gruppert på topic_id (gjenspeiler hver klient) og på event_type (som er for eksempel utbetalinger, innbetalinger, kontoutskrifter og lignende).



Figur 11: Illustrasjon av tilgjengelige tabeller i databasen

6 SERVER-TJENESTER

REST-serveren tilbyr endepunktene listet i figur 12.



Figur 12: Illustrasjon av tilgjengelige endepunkter i programmeringsgrensesnittet

Endepunktene tilgjengeliggjør følgende ressurstyper:

- Accounts – leverer daglige tall, og et samlet tall for en periode over hvor mange hendelser som har hatt ett unikt kontonummer tilknyttet seg.
- Clients – leverer daglige tall, og et samlet tall for en periode over hvor mange hendelser som har hatt unike ressurs IDer tilknyttet seg.
- Resources – leverer alle ressurser (ID, navn og type) basert på ressurstype, samt et endepunkt for tillate ressurstyper.

- Statistics – leverer to typer ressurser, samt et informasjons endepunkt:
    - Statistics – leverer hele objekter fra databasen, med all info for alle datapunkter.
    - Aggregated Statistics – leverer daglige summer av alle hendelser for en periode.
    - Types – gir tilbake en liste over tillate hendelsestype til bruk under henting av de to andre ressursene.
- Users – leverer to typer ressurser:
    - Users – leverer alle brukere fra databasen (dvs. brukernavn, tilknyttet ressurs og registreringsdato).
    - New Users – leverer daglig sum av registrerte brukere, samt et tall for perioden (referert til som *active* i figur 10).

7 SIKKERHET

Løsningen bruker eksklusivt SSL (HTTPS-protokoll) for kommunikasjon mellom klient og server. Noe som hindrer innsyn i kommunikasjon av en tredjepart. Videre autentiseres alle brukere med access tokens i henhold til OAuth 2.0 standarden. Access- og refresh-tokens er generert av IdentityServer 4. Dette sikrer systemet, da all forretningslogikk relatert til innlogging og registrering er utført av en kompetent tredjepart. På serversiden blir disse tokenene validert mot en rekke faktorer for å verifisere gyldigheten av tokenet.
Løsningen er beskyttet mot SQL-injeksjonsangrep ved bruken av Entity Framework Core, samt ved å ikke tillate brukerinput til serveren utover gyldige forespørselsparametere.

Løsningen er beskyttet mot XSS-angrep ved å ikke vise bruker-input i applikasjonen, dersom dette hadde vært tilfellet, ville TypeScript påtvunget korrekt type til brukerens input, så et <script>-tag ville blir returnert som et tekstelement, ovenfor et script.

8 INSTALLASJON OG KJØRING

Programmeringsgrensesnittets viktigste avhengigheter er:

- Microsoft.NET.Sdk.Web et SDK som inneholder funksjonalitet for ASP.NET Core.
- MediatR – et bibliotek som tillater implementering av mediator-mønsteret i ASP.NET.
- Microsoft.EntityFrameworkCore – en ORM som tillater databasetilkobling, samt skriving av spørringer i C#.
- Swashbuckle.AspNetCore – et bibliotek som produserer Swagger-dokumentasjon fra XML-kommentarer i kildekoden.

I tillegg til disse bruker programmet oppdragsgivers private NuGet-pakker. Disse pakkene er ikke offentlig tilgjengelig, og dermed kan ikke løsningen kjøres uten tilgang til pakkene. Videre kan ikke programvaren benyttes uten gyldig innlogging mot oppdragsgivers autentiseringstjenester. Dersom oppdragsgivers pakker er installert, samt gyldig autentisering er oppnådd, kan løsningen kjøres.

For å installere og kjøre programmeringsgrensesnittet må følgende gjøres:

1. Clone prosjektet fra git remote
2. Naviger til src/StatisticsService.API
3. $ dotnet restore
4. $ dotnet run

Alternativt:

1. Clone prosjektet fra git remote.
2. Start prosjektets *solution*-fil (.sln) i Visual Studio.
3. Trykk på startknapp for prosjekt.

Swagger-grensesnittet vil åpnes i et nytt nettleservindu (https://localhost:5001/swagger/index.html).

Klientapplikasjonens viktigste avhengigheter er:

- React – et bibliotek for utvikling av applikasjoner på klientsiden.
- TypeScript – et supersett av JavaScript som legger til typesikkerhet. Dette gir økt sikkerhet til språket og hindrer typiske tolkningsfeil ved bruk av JavaScript.

- Axios – et bibliotek for utførelse av HTTP-forespørsler.
- React-Query – et React-bibliotek for henting, mellomlagring og fornyelse av data hentet over HTTP.
- Chart.js - et bibliotek for å lage grafer i brukergrensesnittet.
- oidc-client-ts - en pakke som tilrettelegger for bruken av OIDC-kompatible identitetstjenester.
- Zustand – et bibliotek for å skape sentrale lagringsobjekter i JavaScript. Det tilbyr alternativer for å mellomlagre i minne, sesjon eller permanent lokallagring.

For å starte klientapplikasjonen må følgende gjøres i en ny terminal:

1. Clone prosjektet fra git remote (gitt at det ikke allerede er utført)
2. Naviger til web/StatisticsService.Web
3. $ npm install
4. $ npm start

Grensesnittet er tilgjengelig på https://localhost:3000.

9 DOKUMENTASJON AV KILDEKODE

Løsningen er hovedsakelig dokumentert i koden, hvor det er nødvendig. Brukergrensesnittet er betydelig mindre dokumentert enn programmeringsgrensesnittet. Brukergrensesnittet er, som nevnt, skrevet i React og TypeScript, noe som gjør JSDocs overflødig, ihvertfall for typesetting i kommentarer. Tyngre funksjoner er dokumentert, da de kan være noe tunglest, eksempelvis figur 13.

```ts
/**
 * Sums an array that contains numeric values, optionally through a property.
 * @param arr {array}
 * @param item {string}
 * @returns {number}
 */
// eslint-disable-next-line @typescript-eslint/no-explicit-any
export const sumArray = (arr: Array<any> | undefined, item?: string): number => {
  if (arr == undefined) return 0
  return item ? arr.reduce((sum, x) => sum + x[`${item}`], 0) : arr.reduce((sum, x) => sum + x, 0)
}
```

Figur 13: Eksempel på en JavaScript-funksjon som er dokumentert grunnet lesbarhet

Programmeringsgrensesnittet følger samme filosofi, alt som er vendt utad (dvs. mot endepunkter og Swagger) er nøye dokumentert for korrekthet og leselighet, eksempelvis figur 14 og 15. Alle metoder i repositoriet har standard XML dokumentasjon påført, med beskrivelser av funksjon, inndatatyper og forventet resultat. Klasser, metoder og typer som brukes i endepunkter vil automatisk bli satt inn i Swagger-dokumentasjon.

```cs
/// <summary>
        /// Gets all active accounts
        /// </summary>
        /// <remarks>Returns a list of dates with counts of all active accounts within requested date range.</remarks>
        [HttpGet]
        public async Task<ActionResult<ActiveResourceDto>> GetActiveAccounts([FromQuery] BaseFilter filter)
        {
            var accounts = await mediator.Send(new GetActiveAccountsQuery(filter));

            if (accounts == null)
                return NotFound();

            return Ok(accounts);
        }
```

Figur 14: Eksempel på XML dokumentasjon i et endepunkt

```
public class BaseFilter : IBaseFilter
  {

      /// <summary>
      /// An ISO datestring defining the start of a requested period. Defaults to previous 7 days.
      /// </summary>
      public DateTime FromDate { get; set; }

      /// <summary>
      /// An ISO datestring defining the end of a requested period. Defaults to today.
      /// </summary>
      public DateTime ToDate { get; set; } = DateTime.Now;

      /// <summary>
      /// The amount of matching values to skip before returning. Starts at the end date.
      /// <br/><br/> Example: Take 2 returns a list of the last two matching elements.
      /// </summary>
      public int? Take { get; set; }

      /// <summary>
      /// The amount of matching values to skip before returning. Starts at the end date.
      /// <br/><br/> Example: Skip 2 returns a list with all elements preceeding the last two elements.
      /// </summary>
      public int Skip { get; set; } = 0;

      public BaseFilter()
      {
          FromDate = ToDate.AddDays(-6).Date;
      }
  }
```

Figur 15: Eksempel på XML dokumentasjon i en innkapslingsklasse for parametere

10 KONTINUERLIG INTEGRASJON OG TESTING

Prosjektet har to ulike utviklingsmiljøer, henholdsvis demo og produksjon. Kontinuerlig integrasjon-oppsettet fungerer ved at man sender inn pull-forespørsel mot hovedgrenen. Den nye koden blir sjekket for konflikter mot hovedgrenen, og blir derav åpen for gjennomgåing av kode. Etter den har blitt godkjent blir den flettet sammen med hovedgrenen. Dermed starter en automatisk byggeprosess som sjekker at det ikke er noen feil i koden og at testene er vellykket. Den vil da bli kjørt ut til demo miljøet, og etter godkjennelse av administrator kjørt ut i produksjon.

Testene som er skrevet for prosjektet er hovedsakelig enhetstester skrevet i MSTest, alle ferdigutviklede metoder i *repositories* er testet, noe som gir høy testdekning i de mest sentrale funksjonene i programmeringsgrensesnittet. For å manuelt kjøre testene, kan man bruke “dotnet test” kommandoen eller bruke Visual Studios “Test Explorer”.

11 REFERANSER

Ingen referanser.