Sluttrapport

Felleskomponent Topologi

Innhold

Innledning

Denne sluttrapporten oppsummerer arbeidet med prosjektet «Felleskomponent – topologi/delt geometri», («felleskomponenten») som er utført sommer/høst 2022.

Hensikten med rapporten er todelt: for det første fungerer den som en redegjørelse for arbeidet som er gjort, de teknologiske og designmessige valgene som er tatt og beskriver muligheter for forbedringer og videreutvikling av komponenten.

For det andre utgjør rapporten en oversikt over løsningen som er utviklet, den dokumenterer hvordan den fungerer og er tenkt brukt. Herunder diskuteres det hvilke bruksområder som støttes og hvordan dette gjøres.

Arbeidet med felleskomponenten er gjennomført som et samarbeidsprosjekt mellom tre kommersielle systemleverandører innenfor geomatikkbransjen i Norge, på oppdrag fra Oslo Havn og Kartverket. Prosjektets deltakere har vært:

- Geodata AS: Jarle Pedersen
- Norconsult Informasjonssystemer (NOIS): Andreas Bergstrøm Aarnseth og Lars Eggan
- Norkart AS: Atle Frenvik Sveen
- Kartverket: Andreas Røstad

Trondheim, november 2022

Bruksområde og hensikt

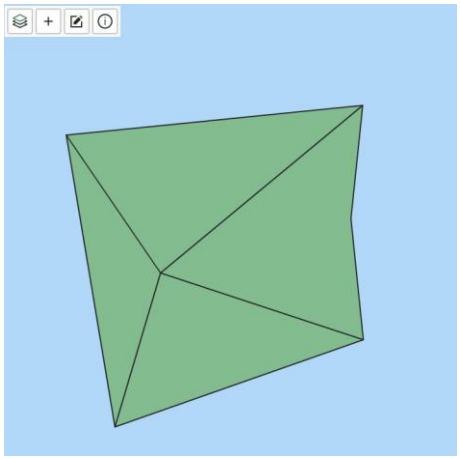
Bakgrunn

Produktspesifikasjonen for Havnedata[i] benytter som de andre FKB-datasettene delt geometri mellom avgrensingslinjene for flater.

Delt geometri er kun en utfordring for de klienter som skal oppdatere geometrien til flate- og linje-objekt som er berørt av dette. For visning av data, og for oppdatering av kun egenskapene til objektene trenger man ikke ta hensyn til dette.

I Havnedata prosjektrapport fra 2020 ble det konkludert fra gruppen at det var behov for en enklere måte å håndtere dette på. Hovedpunktene som det ble pekt på:

1. En felleskomponent (Open Source) som håndterer Flate-Linje topologi skrevet i .NET Core i C# som vil kunne kjøres på ulike operativsystem (Windows, Linux, Apple).
2. Tilgjengelig som OpenAPI tjeneste, men også som kildekode som kan benyttes av de som ønsker dette, også for egen implementasjon dersom dette er ønskelig.

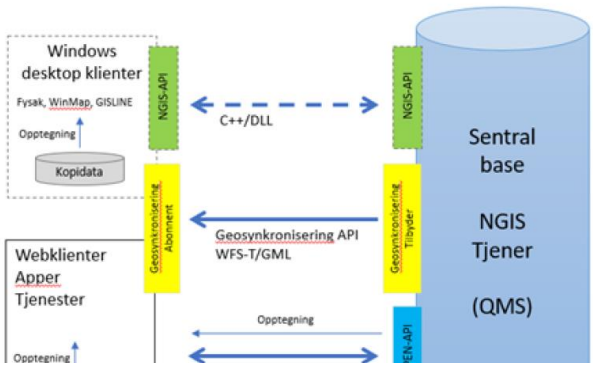


Figur 1: Animasjon som viser redigering på nodepunkt på delt grenselinje

Det samme ble også påpekt i rapporten *Referanseimplementasjon for digitalisering / innlegging av havnedata ved bruk av NGIS Open API*.

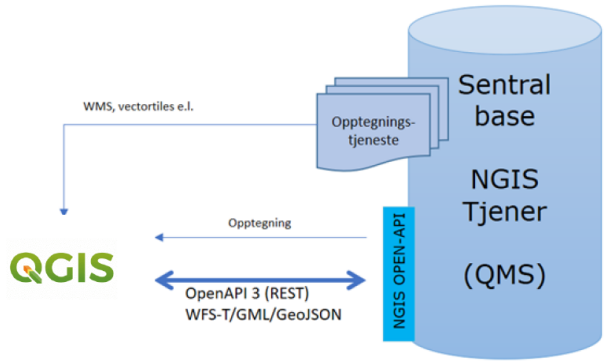
Det ble også sagt at man i dette arbeidet skulle man ikke bare kunne håndtere Havnedata, men også ta høyde for håndtering av andre datasett som er tilgjengelig via NGIS OpenAPI og Sentral FKB.

Sentral FKB er forvaltningssystemet for kartdata der kommunene, Kartverket og andre parter oppdaterer kartdata direkte i en felles database.



Figur 2: Oversikt over metoder klienter kan kommunisere med NGIS på.

For Havnedata er det tidligere blitt implementert en QGIS-plugin. Denne er nå oppdatert med redigering av flater med delt geometri ved hjelp av topologi felleskomponent.



Figur 3: Bruk av QGIS opp mot NGIS Open API

Målsetning og hensikt

Hovedhensikten har vært å lage en løsning som gjør at man fra en enkel klient som f.eks. QGIS eller en enkel WEB-klient kunne redigere på flater med delt geometri uten å implementere kompleks klient-kode, topologi-felleskomponenten skal i samspill med NGIS OpenAPI ta seg av kompleksiteten.

Prinsippet om Open Source har stått sentralt, dette er en felleskomponent som skal være tilgjengelig for alle interesserte parter.

Finansiering

Prosjektet har blitt finansiert av Oslo Havn i tillegg til noe egeninnsats fra partene.

En stor takk til Oslo Havn.

Praktisk bruk

Hovedformålet med felleskomponenten er å hjelpe utviklere som skal lagre objekter med NGIS Open API. Objekter som skal opprettes eller redigeres med NGIS Open API følger ett sett med regler for topologi, som veldig kort kan oppsummeres ved at flater ikke har egen geometri, men refererer til en eller flere kurver. Flere flater kan også referere til samme kurve. Dette kalles delt geometri.

Denne modellen samsvarer ofte ikke med simple features-modellen som brukes i mange enkle klienter. I simple features-modellen har alle flater sin egen geometri. Felleskomponentens metoder skal understøtte «enkle» kartklienter, med støtte for å håndtere simple features, til å håndtere kravene til delt geometri som NGIS Open API setter.

For å understøtte denne arbeidsflyten tilbyr felleskomponenten fire metoder

- **CreateGeometry**: Metode for å opprette et NGIS Open API objekt gitt en GeoJSON geometri
- **CreatePolygonFromLines**: Metode for å opprette et eller flere NGIS Open API objekter med en flate, gitt et sett med kurver og et eller flere senterpunkter
- **EditLine:** Metode for å redigere på et sett med NGIS Open API objekter ved å sende inn endringer på node- eller linjegeometri-nivå
- **EditPolygon:** Metode for å redigere på et sett med NGIS Open API objekter ved å sende inn endringer på polygon-nivå

Disse metodene tar som input og returnerer GeoJson Features som kan hentes ut fra og sendes inn til NGIS Open API. Felles for alle metodene er følgende terminologi:

- Feature: Et NGIS Open API objekt (også omtalt som en «Feature»), vanligvis det objektet som endres.
- AffectedFeatures: En liste med NGIS Open API objekter som er berørt av operasjonen. Brukeren er selv ansvarlig for å finne og sette låser på disse.

API-Dokumentasjon

Felleskomponentens formål er å lette arbeidet med å implementere en kartklient som benytter NGIS Open API til å lagre data. Spesielt gjelder dette klienter som forholder seg til simple features og ikke har noe forhold til delt geometri.

I lys av dette er det på sin plass å presentere en oversikt over noen prinsipper som kan ligge til grunn når en slik klient skal utvikles. Når det er sagt: alle klienter vil benytte forskjellig teknologi, terminologi og ha forskjellige begrensninger, så dette er ikke ment som noen «howto» på hvordan en slik klient skal utvikles.

Generelt kan operasjoner i en slik klient opp mot NGIS Open API deles inn i 13 kategorier

1. Hente og vise geometri for et eller flere objekter
2. Hente og vise egenskaper for et eller flere objekter
3. Opprette nye punkt-geometrier
4. Opprette nye linje-geometrier
5. **Opprette nye flate-geometrier**
6. Opprette nytt objekt med geometri og egenskaper
7. Redigere egenskaper på et objekt
8. Redigere punkt-geometri
9. **Redigere linje-geometri**
10. **Redigere flate-geometri**
11. Slette punkt-objekt
12. **Slette linje-objekt**
13. **Slette flate-objekt**

Av disse er det kun fem operasjoner som innebærer arbeid med delt geometri, og det er disse tilfellene felleskomponenten kan avhjelpe arbeidet med. I praksis er ikke støtte for sletting av geometrier implementert i felleskomponenten. Det betyr at det er i tilfellene opprette nye flate-geometrier, samt å redigere på linje- og flate-geometri at en klient kan ha nytte av å benytte felleskomponenten pr i dag.

I det følgende gis en gjennomgang av hvordan felleskomponenten understøtter disse operasjonene. Felleskomponenten kan eksponeres både som et REST-API og som en C#-modul som kan installeres via NuGet og brukes i annen C#-kode*. Begge måtene å bruke funksjonaliteten på eksponerer de samme metodene, og bruksområdet skal være relativt likt. En mapping mellom metodenavn og http-endepunkter er presentert i Tabell 1. Denne dokumentasjonen er agnostisk til hvilken metode som brukes for å kommunisere med felleskomponenten.

Opprettelse av objekter

1. Opprette et punkt- eller linje-objekt

Punkt- og linje- objekter har ikke referanser til andre objekter, og trenger ikke noen spesiell logikk for å opprettes. Enhver klient kan dermed sende disse direkte til NGIS Open API. Metoden *CreateGeometry* støtter likevel innsending av objekter med punkt- eller linje-geometri. Felleskomponenten vil da autogenerere en UUID som settes som LokalId (om denne ikke er satt) samt sette action=Create.

2. Opprette et flate- objekt ved å sende inn et polygon-objekt

For å opprette et flate-objekt med NGIS Open API må man sende inn både en linje, som avgrenser flaten, samt et flate-objekt som refererer til denne. For å avhjelpe dette kan en klient sende inn et

flate-objekt, med geometri, til *CreateGeometry*-metoden i felleskomponenten. Denne vil da returnere minst to AffectedFeatures: Flate-objektet, et linje-objekt for flatens ytre avgrensning, samt et linje- objekt for hvert hull i flaten. Flate-objektet vil få satt referanser til de opprettede linje-objektene. Alle returnerte objekter vil få opprettet LokalId (om denne ikke er satt) samt satt action=Create og flate-objektet vil få generert et flatepunkt.

3. Opprette et eller flere flate-objekter ved å sende inn et eller flere linje-objekter

Hvis en flate skal ha referanser til flere enn en linje, typisk hvis man skal legge til en flate som deler grenselinje med en eksisterende flate, kan man opprette et flate-objekt ved å sende inn referanser til linjene flate-objektet skal bestå av. Metoden CreatePolygonFromLines implementerer denne funksjonaliteten. Denne metoden tar inn en liste av linje-objekter og prøver å bygge flater fra disse. Alle flater som kan dannes returneres fra metoden. I tillegg er det mulig å angi et eller flere punkter, som i så fall indikerer et punkt innenfor hver flate.

Denne metoden vil bygge flater hvis det er mulig, og returnere et eller flere flate-objekter med geometri og referanser satt. Metoden håndterer hull i flater og sørger for at orienteringen på disse er riktig. For at metoden skal klare å bygge flater av angitte linjer er det viktig at koordinatene for nodepunktene samsvarer. Dette kan løses i klient ved bruk av snapping-funksjonalitet.

Redigering av objekter

For å redigere geometrier kan man enten fokusere på å la brukeren redigere på linje-geometrier, og så sørge for å oppdatere tilgrensende linjer og flater, eller man kan redigere på flate-geometrier og oppdatere tilhørende og tilgrensende linjer og flater. Førstnevnte tilnærming er å anbefale, da dette vil redusere behovet for låsing av objekter.

Metoden *EditLine* tilbyr funksjonalitet for å redigere på linjer. Metoden kan fungere i to moduser:

1. Klienten sender en endring på node-nivå
2. Klienten sender en endring på linjegeometri-nivå.

Her er det beste valget avhengig av funksjonalitet klienten tilbyr, men siden metoden for å ta inn endringer på linje-geometri-nivå kun aksepterer en endring av gangen vil node-nivå være å anbefale hvis klienten har støtte for dette.

1. Redigere et linje-objekt på node-nivå

Ved å redigere på node-nivå vil man for hver endring på en linje-geometri sende en forespørsel til felleskomponentens *EditLine*-metode med beskrivelse av denne endringen. Her er man altså avhengig av at en klient kan si fra så fort en bruker har satt inn, slettet eller flyttet på en node. Felleskomponenten vil så beregne en ny linje-geometri, samt flytte på alle andre linjer som berører noden (hvis den er et nodepunkt) og endringer på alle flater som bruker en av de påvirkede linje-objektene som avgrensningslinje. For at dette skal fungere må klienten sende med en liste med linje-objektets avhengigheter. Flere redigeringer på et linje-objekt kan sendes til felleskomponenten før man lagrer resultatet mot NGIS Open API, men det anbefales å bruke en form for temporærlager i klienten, slik at man hele tiden kan sende felleskomponenten riktig sett med AffectedFeatures. AffectedFeatures-responsen av kall *n* til felleskomponenten kan brukes som AffectedFeatures på kall *n+1* til felleskomponenten, og AffectedFeatures-respons fra siste kall til felleskomponenten kan brukes som input til NGIS Open API sin saveFeatures-metode.

2. Redigere et linje-objekt på linjegeometri-nivå

Ved redigering på linjegeometri-nivå vil man ikke sende inn endringer på node-nivå, men sende inn en representasjon av linje-geometrien slik den så ut før og etter redigeringen. Her overlater man

mer beregning til felleskomponenten, men hvis ikke klienten har en enkel måte å bestemme hvilken node en bruker har redigert på kan denne metoden være til nytte. Bortsett fra hva man sender inn fungerer dette kallet helt likt som redigering på node-nivå, og man benytter også her metoden *EditLine*.

3. Redigere et flate-objekt

Som man ser av avsnittene om å redigere på linje-objekter vil redigering av en grenselinje også endre flate-objektene som avgrenses av denne. Når man jobber med delt geometri er dette typisk brukerens mentale modell også: man redigerer på linje-objekter og flatene oppdaterer seg slik at de reflekterer de nye avgrensingslinjene.

I noen tilfeller kan man dog ha interesse av å la brukeren redigere direkte på flate-geometriene. Dette kan være tilfeller der en flate kun har en avgrensingslinje som ikke deles med andre flater, eller i applikasjoner der brukeren ikke har noe forhold til delt geometri.

For å avhjelpe i slike situasjoner tilbyr felleskomponenten *EditPolygon*-metoden. Her sender man inn en representasjon av en flate *før* og *etter* redigering, så vil felleskomponenten analysere seg frem til hvilke grenselinjer og eventuelle andre flate-objekter som påvirkes av denne endringen. Merk at denne operasjonen ikke egner seg til redigering av flate-objekter der man potensielt kan ha veldig mange avhengigheter.

Oversikt over APIet

```
public interface ITopologyImplementation
{
    TopologyResponse CreateGeometry(CreateGeometryRequest request);
    IEnumerable<TopologyResponse>
CreatePolygonsFromLines(CreatePolygonFromLinesRequest request);
    TopologyResponse EditLine(EditLineRequest request);
    TopologyResponse EditPolygon(EditPolygonRequest request);
}
```

Kodeblokk 1: ITopologyImplementation-interfacet

Hver av de fire metodene som beskrives over eksponeres gjennom *ITopologyImplementation*-interfacet, beskrevet i Kodeblokk 1, og eksponeres som http-endepunkter som beskrevet i Tabell 1.

Metode	http-endepunkt	http-metode
CreateGeometry	/createGeometry	POST
CreatePolygonFromLines	/polygonFromLines	POST
EditLine	/editLine	POST
EditPolygon	/editPolygon	POST

Tabell 1: Oversikt over metoder i felleskomponenten og hvilke http-endepunkt de samsvarer med.

Alle metodene i APIet returnerer et *TopologyResponse*-objekt (med unntak av at CreatePolygonFromLines som returnerer ett objekt pr genererte flate), som beskrevet i Kodeblokk 2. Dette objektet inneholder en liste av *NgisFeatures*, kalt AffectedFeatures, som er objekter påvirket av operasjonen, samt et boolsk flagg IsValid, som settes til *false* om man ikke kan danne gyldige objekter på bakgrunn av forespørselen.

Hvis IsValid er true kan AffectedFeatures-responsen sendes til NGIS Open API for lagring. Unntaket her er ved opprettelse av objekter, der klienten selv må sørge for å få satt nødvendige egenskaper (som angitt ved datasettets og objekttypens skjema) før innsending.

```
public class TopologyResponse
{
    public List<NgisFeature> AffectedFeatures { get; set; }
    public bool IsValid { get; set; }
}
```

Kodeblokk 2: TopologyResponse-klassen

De fire metodene tar inn litt forskjellig input-data, som beskrevet i det følgende.

```
public class CreateGeometryRequest
{
    public NgisFeature Feature { get; set; }
    public List<NgisFeature> AffectedFeatures { get; set; } = new ();
}
```

Kodeblokk 3: CreateGeometryRequest-klassen

CreateGeometry tar inn en CreateGeometryRequest, som beskrevet i Kodeblokk 3. Her er det kun Feature som er relevant, dette er et objekt som inneholder geometrien brukeren har tegnet i klienten.

```
public class CreatePolygonFromLinesRequest
{
    public List<NgisFeature> Features { get; set; } = new();
    public List<Point>? Centroids { get; set; }
}
```

Kodeblokk 4: CreatePolygonFromLinesRequest-klassen

CreatePolygonFromLines-metoden tar inn en CreatePolygonFromLinesRequest, gjengitt i Kodeblokk 4. Her er Features en liste med linje-objekter man vil danne flater fra, og centroids er en liste med senterpunkter som kan utelates. Merk at NgisFeatures i Features-lista må inneholde linje-geometrier.

```
public class EditLineRequest
{
    public List<NgisFeature>? AffectedFeatures { get; set; }
    public NgisFeature Feature { get; set; }
    public NgisFeature? NewFeature { get; set; }
    public EditLineOperation? Edit { get; set; }
}
```

Kodeblokk 5: EditLineRequest-klassen

EditLine tar inn en EditLineRequest (Kodeblokk 5). Denne klassen har to valgfrie parametere, der en av dem må være satt. Feature-attributtet representerer objektet med linje-geometri slik det så ut *før* redigeringsoperasjonen. AffectedFeatures-lista er en liste over alle direkte og indirekte referanser til den redigerte linja. Disse kan hentes fra NGIS Open API sin getFeature-metode med references satt til «all».

Ved redigering i node-modus, som beskrevet i avsnittet «1. Redigere et linje-objekt på node-nivå», settes Edit-attributtet (se beskrivelse av dette under). Ved redigering på linjegeometri-nivå settes NewFeature-attributtet til representasjonen av objektet slik det ser ut *etter* redigeringen. Merk at felleskomponenten kun støtter en node-endring for hver forespørsel, også når man redigerer på linjegeometri-nivå.

```
public class EditLineOperation
{
    public EditOperation Operation { get; set; }
    public int NodeIndex { get; set; }
    public Coordinate? NodeCoordinate { get; set; }
}
```

Kodeblokk 6: EditLineOperation-klassen

EditLineOperation-klassen (Kodeblokk 7) brukes ved redigering på en linje-geometri i node-modus, og er en kompakt måte å beskrive endinger på en linje-geometri på. Operation-attributtet kan settes til en av tre muligheter: «Edit», «Insert» eller «Delete». Denne operasjonen angir hva som skal skje med noden på den (0-indekserte) indeksen satt med NodeIndex-attributtet. Delete vil slette noden, Edit vil sette verdien til noden til NodeCoordinate, mens Insert vil sette inn en ny node med verdien til NodeCoordinate. Merk at ved Insert vil den nye noden settes inn til *venstre* for eksisterende node. For å gi en linje en ny start-node brukes en Insert på NodeIndex=0, og for å sette inn et nytt endepunkt på en linje med 3 noder brukes en Insert på NodeIndex=3.

```
public class EditPolygonRequest
{
    public List<NgisFeature>? AffectedFeatures { get; set; }
    public NgisFeature Feature { get; set; }
    public Polygon EditedGeometry { get; set; }
}
```

Kodeblokk 7: EditPolygonRequest-klassen

Ikke overraskende bruker EditPolygon metoden EditPolygonRequest-klassen (Kodeblokk 7). Denne samsvarer mye med en EditLineRequest. AffectedFeatures er alle direkte og indirekte referanser, Feature er det redigerte flate-objektet *før* redigering og EditedGeometry er geometrien til det redigerte objektet etter redigering. Merk at også her støttes kun en operasjon per forespørsel.

Utviklingsmiljø og teknologi

Felleskomponenten er utviklet i C# på Microsoft .NET -plattformen. REST-API-funksjonalitet er implementert ved bruk av ASP.NETCore[ii]-biblioteket, som legger til rette for enkelt oppsett av http endepunkter med definert input og output som serialiseres til og fra JSON. Det er også satt opp funksjonalitet som automatisk genererer en OpenApi-dokumentasjon av APIet basert på koden. For enhetstesting har biblioteket XUnit blitt benyttet. Løsningen er utviklet ved bruk av git som versjonskontrollsystem og koden ligger på GitHub[iii] med en åpen lisens (MIT License).

Som topologimotor i systemet benyttes biblioteket NetTopologySuite (NTS)[iv], en åpen kildekode-implementasjon av en rekke konsepter innenfor geografisk IT, som lesing og skriving av GeoJSON, håndtering av de geografiske primitivene punkt, linje og polygon, samt overlapps- og buffer-operasjoner. NTS er en port av java-biblioteket JTS, som brukes i komponenter som GeoServer og også er portert til C++ som GEOS, og benyttes i PostGIS. Dermed ansees NTS som et kraftfullt, komplett og godt testet bibliotek for å jobbe med geografiske data på .NET-plattformen.

Arbeidet med felleskomponenten har under utviklingsperioden foregått med utgangspunkt i et åpent git-repository der prosjektdeltakerne har hatt skrivetilgang. Pull-request (PR) metodikken, der endringer utføres i egne branches og merges over i main-branch etter gjennomgang av andre prosjektdeltakere har som hovedregel blitt fulgt. Det har også blitt etablert Continuous Integration-oppsett via Github Actions. Dette betyr at for hver ny endring på main-branchen (det vil si ved hver merged PR) har alle enhetstester blitt kjørt og APIet har blitt rullet ut til en testserver. Testserver har blitt etablert i Microsoft Azure i NOIS sitt miljø, og depolyes vha Docker.

Selv om felleskomponenten er tenkt distribuert som et REST-API som kan benyttes fra en rekke klienter over http er kodebasen organisert slik at det skal være enkelt å ta den i bruk fra annen kode på .NET-plattformen. Den enkleste måten å fasilitere dette på vil være å distribuere komponenten som en NuGet pakke, der en binærversjon av koden distribueres til et sentralt register. Dette har dog ikke blitt gjennomført.

I det hele kan det sies at teknologivalgene for felleskomponenten har vært gjort med fokus på å benytte åpen kildekode, benytte komponenter som er i bruk av mange aktører og er kjent for mange, samtidig som de benytter en plattform som alle leverandørene benytter og har et forhold til. Ved å eksponere funksjonaliteten over et HTTP REST-API vil ingen klientimplementasjoner utelukkes fra å benytte funksjonaliteten.

Det kan nevnes at vi har brukt C# på .NET 6.0 som nå er Open Source.

Fremdrift og kommunikasjon

Arbeidet med felleskomponenten har foregått som et samarbeidsprosjekt mellom tre kommersielle systemleverandører, som hver har stilt med en eller flere prosjektdeltakere. Dette betyr at arbeidsformen ikke har vært typisk for et kommersielt softwareutviklingsprosjekt, med et samlokalisert team. Teamet har vært distribuert på både lokasjon og tid, siden ingen av prosjektdeltakerne har jobbet 100% på dette prosjektet. Slik sett kan teamets arbeidsform si å minne mer om åpen kildekode-utvikling, som ofte preges av distribuerte team som samarbeider digitalt.

Teamet gjennomførte en fysisk workshop ved arbeidets start, der vi jobbet for en samordnet problemforståelse og for å skissere en løsning. I tiden etter dette har teamet hatt ukentlige statusmøter over Teams, der vi har diskutert fremgang, hindringer og nye ideer. I tillegg til dette har det vært gjennomført statusmøter annenhver uke med oppdragsgiver, der fremgang har blitt presentert.

Koordineringen av arbeidet har skjedd ved bruk av GitHub issues[v], et lettvekts alternativ til oppgavestyringssystemer som Jira eller Azure Devops. Til teamets formål ha dette fungert godt. Vi har også benyttet kommunikasjon via GitHub Pull Requests og tale/chat via Teams og Slack.

En av hovedutfordringene med et arbeid som dette e ikke nødvendigvis å komme frem til den tekniske løsningen, men å designe gode interfacer og arbeide frem en omforent problemforståelse i teamet. I lys av dette var det å samles fysisk til en workshop en god løsning, da man fikk rom til å diskutere problemstillingen og enes om hva denne er og hvordan den kan løses.

Parallelt med utvikling av felleskomponenten har det foregått et utviklingsløp for å ta i bruk funksjonaliteten av denne i en QGIS-plugin som utvikles av NOIS på oppdrag fra samme oppdragsgiver. Det å ha nær kontakt og til dels overlapp med teamet som utvikler denne gjør at man får bedre innsikt i hvordan komponenten brukes og kan brukes, og kan sies å ha lettet utfordringen.

Teamet valgte også tidlig å dele opp oppgavene i steg, slik at man til enhver tid prioriterte de mest grunnleggende problemene først. Dette er en fordel i prosjekter av en smidig natur, med en gitt tids- og kostnadsramme. Man kan da avslutte arbeidet når en av disse rammene møtes, men fortsatt ha et fungerende produkt. Denne tilnærmingen har også medført at man parallelt har kunnet utvikle klienter som bruker felleskomponenten. Denne arbeidsformen har mye til felles med diverse smidige utviklingsmetodikker som brukes mye i softwareutvikling.

Teamets medlemmer er også enige i at det har vært en nyttig erfaring å samarbeide på tvers av selskaper som i det daglige er konkurrenter. Innsikt i hvordan andre tenker, problemstillinger, erfaringer og måter å løse problemer på har vært givende for alle parter.

Utfordringer og erfaringer

Systemutvikling som fag er interessant. Hvis man vet nøyaktig hvordan en utfordring skal løses i kode har man allerede løst den og trenger ergo ikke utvikle noe. Av dette følger det at all systemutvikling innebærer å løse problemer som ikke er løst før. Som videre betyr at man er nødt til å støte på utfordringer man ikke så på forhånd. Her vil vi diskutere noen av utfordringene vi har møtt under arbeidet, slik at andre forhåpentligvis kan lære av disse. Vi vil også diskutere noen erfaringer vi har gjort oss, både rent teknisk og av mer organisatorisk art.

Rent teknisk er felleskomponenten «enkel». Gitt en input-state skal output-state genereres. Mye av utfordringen med arbeidet har vært å definere hva input- og output-state skal være. Hovedprinsippene har vært at input skal bestå av det en enkel klient skal kunne generere ved hjelp av NGIS Open API og at output skal være noe en enkel klient skal kunne vise og lagre til NGIS Open API.

I arbeidet med å definere dette har det vært til stor hjelp å ha et parallelt løp med utvikling av nettopp en komponent (QGIS-plugin). På toppen av dette har systemleverandørene også kjørt parallelle løp med utvikling av egne klienter, noe som har medført flere tilnærminger til arbeidet og satt lys på flere bruksområder.

Når input-og output-state er definert dreier utviklingsarbeidet seg om å implementere funksjoner som transformerer input til output. Denne løsningen skal være korrekt, rask og ikke minst lesbar for andre utviklere.

Slikt utviklingsarbeid passer godt til konseptet testdreven utvikling, der med skriver automatiserte tester som kjører koden som er skrevet. Utfordringen med dette er å ha gode testdata som dekker alle tilfeller som kan oppstå. En erfaring vi gjorde oss tidlig var at det ikke holder å teste på enkle geometrier som fungerer fint for å illustrere utfordringen. Man er nødt til å dekke andre tilfeller som kan oppstå. Så lenge man tar hensyn til dette har testdreven utvikling vært til stor hjelp. Ikke bare som en støtte til å utvikle koden, men også som en støtte når man videreutvikler og utvider. Med automatiserte tester på plass har man et sikkerhetsnett som gjør at man ikke innfører feil i eksisterende kode. I tillegg har det muliggjort refaktorering («omskriving») av koden for å gjøre den mer oversiktlig og lesbar.

Det å la et API vokse «organisk» basert på en oppgavedefinisjon er en tilnærming til API-design som står i kontrast til et løp der man på forhånd definerer et strikt API og så implementerer kode som oppfyller dette. Begge tilnærminger har sine fordeler og ulemper, men i dette prosjektet har vi stor tro på at en organisk tilnærming var den rette. Mye fordi en stor del av oppgaven i seg selv var å definere hvordan et slikt API skal se ut.

I retrospekt ser vi dog at man nok kunne brukt mer tid på å først definere grensesnittene, før man investerte for mye tid på å skrive kode. Så lenge dette hadde vært utført av det samme teamet ville man fortsatt oppnådd stor grad av eierskap til prosessen og produktet, noe som kan være en utfordring hvis API-design og API-utvikling splittes mellom forskjellige team. På den annen side, det ligger mye gevinst og læring i smidig utvikling og prosessen har vært fruktbar.

På den tekniske siden tok vi ganske tidlig i prosessen noen valg som har påvirket arbeidet videre. De tre viktigste valgene var kanskje:

1. APIet skal være helt *stateless*. Med andre ord: et API-kall skal ikke være avhengig av et tidligere kall. Dette gjør at felleskomponenten ikke trenger å lagre data, at funksjonaliteten

blir enkel å enhetsteste og at den enkelt kan skaleres. Bakdelen med denne tilnærmingen er at den stiller noen større krav til klienten.
2. APIet skal ikke kommunisere direkte med NGIS Open API. Biblioteket skal fungere som et støttebibliotek og hjelpe klienter med å transformere data slik at de kan sendes til NGIS Open API.
3. Felleskomponentens datamodeller skal følge NGIS Open API sine datamodeller der det gir mening.

Vi oppdaget tidlig en utfordring med NetTopologySuite sin System.Text.JSON implementasjon av GeoJSON skriving. Denne håndterte ikke eksplisitt håndtering av orientering på flate-avgrensninger. Ble løst ved å bruke NTS sin Newtonsoft.Json-baserte GeoJSON skrivefunksjonalitet. Dette medførte litt mer jobb med integrering mot ASP.NET core API oppsett. 21. november 2022 ble problemet med System.Text.JSON løst i NTS (se[vi]), men vi har ikke tatt inn igjen dette. Felleskomponenten benytter i dag fortsatt NTS sin Newtonsoft.Json-baserte GeoJSON skrivefunksjonalitet.

En annen utfordring vi har støtt på relaterer seg til koordinattransformasjon og flyttallsproblematikk. Når man henter objekter fra NGIS Open API i et annet referansesystem enn originalen (for eksempel ved å hente geografiske koordinater for visning i en webklient) vil NGIS Open API transformere disse. Man kan da få grenselinjer som skal dele et nodepunkt, men som på grunn av flyttalsproblematikk ikke gjør det fordi de avviker med noen nanometer. Siden felleskomponenten bruker NTS til å lese innkommende GeoJSON-data vil dermed validering feile før koden i felleskomponenten kan gjøre noe med problemet. Løsningen her ble å avrunde transformerte koordinater fra NGIS Open API for å unngå flyttalsproblematikk.

Det parallelle utviklingsløpet med QGIS-plugin ble også rammet av en feil relatert til låsing av objekter i NGIS Open API. Dette medførte at arbeidet her stoppet opp og vi fikk ikke testet bruk av felleskomponenten fra QGIS så godt som vi hadde håpet. En god erfaring her var da at vi hadde flere klienter å teste med, samt at NGIS Open API-teamet var raskt på banen og bidro til feilsøking.

Forbedringer og veien videre

Som tidligere nevnt har utviklingen av felleskomponenten hatt en gitt kostnads- og tidsramme. På bakgrunn av dette har oppgaver blitt prioritert, og som med de fleste utviklingsprosjekter har ikke alle oppgaver blitt løst. På tross av dette er felleskomponenten en fullt ut brukbar komponent som skal kunne tas i bruk.

Veien videre for prosjektet ligger godt dokumentert i oppgavestyringssystemet vi har benyttet, GitHub issues [vii] , og kan kort sammenfattes slik:

- Støtte slette-operasjoner på flate
    - Kunne slette en flate og også automatisk slette unna dens avgrensningslinjer
- Støtte slette-operasjoner på linjer
    - Sletting av en linje vil kunne medføre at en eller flere flater ikke er gyldige. Dette må håndteres
- Håndtere splitting av geometrier
    - En flate skal kunne splittes ved å tegne en ny grenselinje. Da må sannsynligvis en rekke andre grenselinjer tilpasses
    - En grenselinje skal kunne splittes. Da må man generere nye linjer og rekonstruere eventuelle flater
- Ved redigering av en flate burde man ha mulighet til å kunne erstatte en eller flere av grenselinjene den refererer til

Dette er i hovedsak de tekniske utfordringene direkte relatert til delte grenselinjer teamet har identifisert, men ikke rukket å implementere løsninger på.

I tillegg til dette vil det alltid være en rekke andre ikke-funksjonelle krav som kunne vært oppfylt. De oppgavene teamet har identifisert inkluderer

- Gjennomføre mer storskala stresstesting av løsningen ved bruk av store datasett
- Oppsett av utrulling av komponenten som en NuGet-pakke til NuGet pakkeregister
- Oppsett av utrulling av APIet til driftslokasjon
    - Herunder oppsett av logging og overvåkning av tjenesten
- Innføre rate-limiting på APIet slik at det ikke kan overbelastes

Naturligvis vil også mer utstrakt bruk av felleskomponenten innebære at man finner feil, avdekker uklarheter eller mangler i dokumentasjon eller oppdager andre utfordringer. Disse burde registreres som issues i GitHUb, slik at man har dem samlet hvis eller når prosjektet skal gjenopptas.

I tillegg til dette er det en besnærende tanke å lage en referanseimplementasjon av en enkel webklient som benytter åpen-kildekode-komponenter og felleskomponenten for å jobbe mot NGIS Open API. På den annen side oppfyller QGIS-plugin-prosjektet mange av ønskene til et slikt prosjekt, så en enklere tilnærming kan kanskje være å bruke mer tid på å dokumentere og kommentere koden som er utviklet her.

Teamet legger ved innlevering av denne sluttrapporten ned sitt arbeid med felleskomponenten, men ser ingen hindringer for at prosjektet kan gjenopptas ved en senere anledning. Videre arbeid er godt dokumentert både i denne rapporten og i oppgavestyringssystemet og leverandørene har alle ansatte med kjennskap til både problemstilling og eksisterende løsning.

Et spørsmål er jo hva innføringen av FKB 5.0 med støtte for heleid geometri vil gjøre med behovet for en slik løsning. Men i FKB 5.0 og i Havnedata 3.0 vil de fremdeles finnes objekttyper med delt geometri, så behovet for topologi-felleskomponenten vil også være der i fremtiden.

---

[i] https://objektkatalog.geonorge.no/Pakke/Index/EAPK_3B1EA5AD_ED16_428f_B042_76E31F34499A
[ii] https://github.com/dotnet/aspnetcore
[iii] https://github.com/kartverket/NGIS-OpenAPI-felleskomponent-delt-geometri
[iv] https://github.com/NetTopologySuite/NetTopologySuite
[v] https://github.com/kartverket/NGIS-OpenAPI-felleskomponent-delt-geometri/issues
[vi] https://github.com/NetTopologySuite/NetTopologySuite.IO.GeoJSON/pull/112
[vii] https://github.com/kartverket/NGIS-OpenAPI-felleskomponent-delt-geometri/issues