Tilskuddsbasen

Sytemdokumentasjon

1 - Functional View

En overordnet beskrivelse av de mest sentrale aspekter i flod applikasjonene finner man her.

Vesentlige funksjoner

Beskrivelsen under gir en ide om hva Tilskuddsbasen GUI tilbyr av funksjonalitet. Legg merke til at deler av funksjonaliteten er utviklet basert på allerede eksisterende tjenester (f.eks. flod booking tjenester).

- Saksbehandling av søknader om tilskudd. Applikasjonen leverer funksjonalitet for håndtering av søknadene fra opprettelse av kladd til utbetaling av innvilget beløp.
- Støtte for opplasting av vedlegg på søknad, rapport og for opplasting av saksvedlegg (kun synlige for interne brukere).
- Funksjonalitet for å kunne eksportere informasjon til fil for viderebehandling utenfor Tilskuddsbasen (f.eks. eksport av søknader).
- Generering av forskjellige dokumenttyper som for eksempel vedtaksbrev, utbetalingsfil (ERV fil) eller diverse epost.
- Utsending av epost og batch jobber (f.eks. purringer ved utløpt frist)
- Kodeverk og funksjonalitet for opprettelse og vedlikehold av tilskuddsordninger.
- Kodeverk og funksjonalitet for opprettelse og vedlikehold av standardtekster (kan brukes ved generering av vedtaksbrevet)
- Oppslag i aktørbasen: Bruker Brønnøysundregistrene for informasjonen relatert til organisasjoner og personer.
- Opprettelse av nye organisasjoner og vedlikehold av informasjon knyttet til organisasjoner etter at de er hentet inn i Tilskuddsbasen. Brukes for å kunne dekke for eventuelle mangler i Brønnøysundregistrene.
- Vedlikehold av profilinformasjon for søkere.
- Automatisk arkivering i Trondheim kommunes saksarkiv, ved gitte hendelser på søknaden.

UML aktører

- Eksterne brukere
    - *Søker*: Bruker med organisasjon i Norge som bruker Tilskuddsbasen for å søke om tilskudd og følge opp sine søknader.
- Interne brukere
    - *Saksbehandler*: Brukere ansatte i Trondheim kommune, ansvarlige for saksbehandling av søknadene.
    - *Godkjenner*: Brukere ansatte i Trondheim kommune, ansvarlige for godkjenning av de vedtakene saksbehandlerne oppretter.
- System bruker
    - *Superuser*: Bruker som brukes av selve systemet for å initiere batch jobber (for eksempel ved purring). Det er ikke mulig å logge seg inn i applikasjonen med denne brukeren

Pålogging Tilskuddsbasen

Påloggingsmekanismene

Påloggingen gjøres basert på to mekanismer avhengige av om brukeren er eksterne bruker (f.eks. søkere) eller interne bruker (f.eks. saksbehandlere, godkjennere og administratorer).

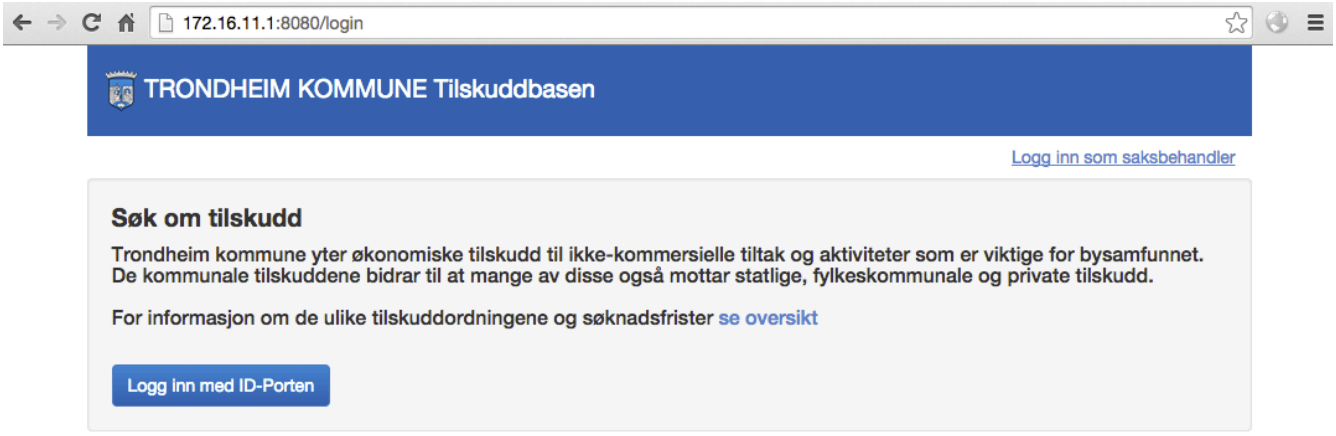
Påloggingsmekanismene som støttes er:

- "Logg inn med ID-Porten": ID-porten, benyttes av eksterne brukere.
- "Logg inn som saksbehandler": ADFS, benyttes av interne brukere.

Selve påloggingsmekanismene er tredjeparts systemer (ikke utviklet under Tilskuddsbasen-prosjektet).

Landingsside pålogging

Tilskuddsbasen tilbyr en side som gjør det mulig for brukeren å velge påloggingsmekanisme:



Hver av knappene sender brukeren videre til det aktuelle påloggingsmekanisme. "Logg inn med ID-Porten"-knappen logger man på med ID-porten for eksterne brukere, mens lenka oppe i høyre hjørne "Logg inn som saksbehandler" logger man inn via ADFS.

Direkte lenker til påloggingsmekanismene

Landingsside for pålogging er ikke en side man *må* bruke for å logge seg inn i Tilskuddsbasen. De url-ene som knappene "Logg inn med ID-Porten" og lenken "Logg inn som saksbehandler" bruker (ref bildet over) kan bli kopierte andre steder slik at brukeren går rett inn i påloggingsmekanismens påloggingsside i stedet for å gå via landingssiden. Det betyr at man kan legge til en "Logg inn i Tilskuddsbasen" knapp på for eksempel. https://www.trondheim.kommune.no/tilskudd/ slik at søkere slipper å gå gjennom påloggingssiden.

Pålogging som intern bruker

Pålogging som intern bruker/saksbehandler skjer via ADFS, og er kun støttet på Trondheim Kommune sitt nettverk, enten internt eller via VPN. For at saksbehandlerne skal slippe å se påloggingsbildet hver gang de skal inn i Tilskuddsbasen har vi laget en løsning som "husker" at de er saksbehandlere. Det vil si at når man velger "Logg inn som saksbehandler" så settes en cookie i nettleseren, som gjør at man neste gang også logger inn som saksbehandler (via ADFS). Løsningen er cookie basert og fungerer så sant cookies tillates og ikke slettes.

URL-er i Tilskuddsbasen

I Tilskuddsbasen er URL-ene utviklet slik at de lett kan deles på tvers av brukerne. Her er noen av prinsippene (se 9 - GUI View for mer detaljer om hva de forskjellige GUI-elementene nevnt under er):

- Valg av element i venstre meny reflekteres i URL-en.
- Valg av aksjon i aksjon meny reflekteres i URL-en.
- "Se" bilde for et objekt viser stien til objektet avsluttet av dens id.
    - for eksempel `http://<hostname>:<port>/soknad/23/` er url til "Se" bilde for søknad med id 3
- "Rediger" bilde for et objekt er samme url som "Se" bilde bortsett fra at det står `edit` etter id-en.
    - for eksempel `http://<hostname>:<port>/soknad/23/edit`
- Vi bruker ren tekst i url så mye som mulig i url slik at de er lesbare. Eneste avvik er
    - Spesiell tegn må url encodes før de settes i url'en, så meny valg "Trekk soknad" blir "Trekk%20søknad". Dette er ikke noe vi kan gjøre noe med.
    - Strengen som bygges ved "Filtrer" på landingsside, over søknadene kan bli veldig lang så vi må dessverre enkode (komprimere) til et annet egnet format der filtreringsinformasjonen fortsatt er intakt.

2 - Non-functional View

Alle ikke funksjonelle askpekter i applikasjonen beskrives her.

Tilgjengelighet og skalerbarhet

Tilskuddsbasen er utviklet slik at den skal kunne skalere (mikro tjenester, ingen tilstand lagret på serveren). Lastbalansering er også noe som skal kunne tas i bruk ved eventuell behov for bedre tilgjengelighet.

Det er ikke blitt identifisert behov for å kjøre flere enn en instanse av applikasjonen så skalering og lastbalansering er dog ikke i bruk per skrivende tidspunkt.

Caching

Caching er ikke tatt i bruk i Tilskuddsbasen

3 - Logical View

Logical view dokumentere de viktige aspektene relatert til måten komponentene i applikasjonen fungerer med hverandre

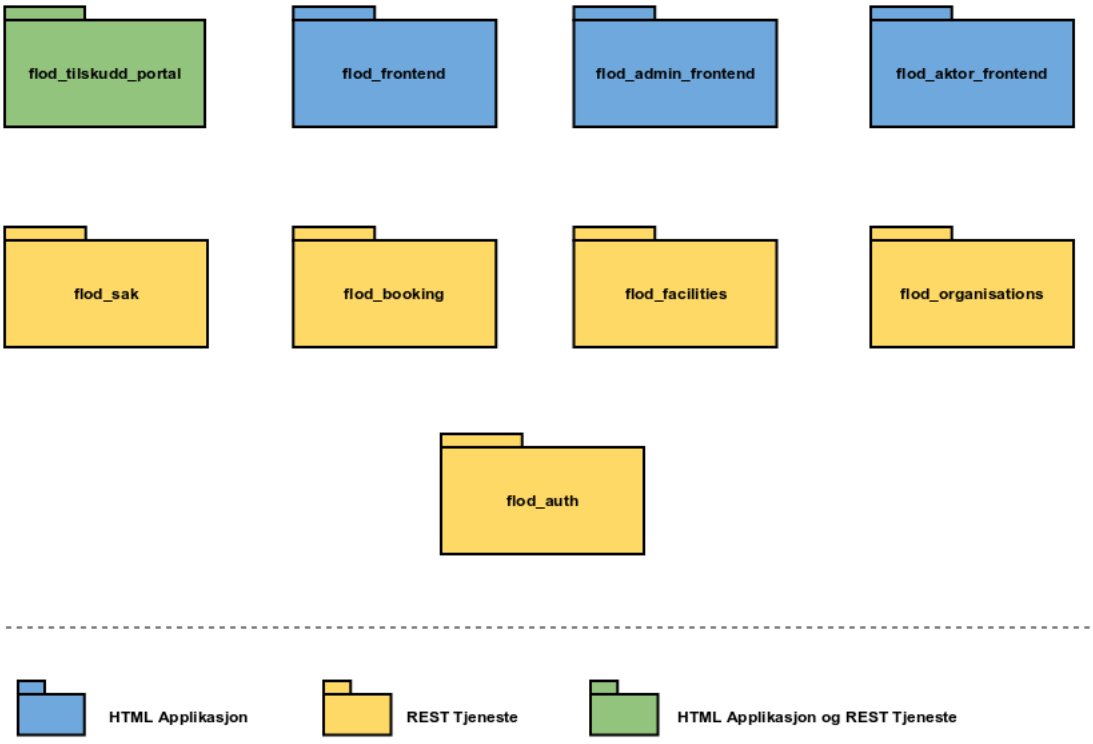
Applikasjoner og moduler

Mikroapplikasjoner og webapplikasjoner

Det mest sentrale prinsippet under utviklingen av flod booking og Tilskuddsbasen er at systemet er bygd basert på mikrotjenester. Hver mikrotjeneste har ansvar for en avgrenset del av funksjonaliteten i systemet, og den funksjonaliteten leveres via et REST API. Funksjonaliteten i en mikrotjeneste er som regel knyttet til et spesielt type sentral objekt i systemet som for eksempel *organisasjoner* i `flod_organisations`, *lokaler* i `flod_facilities` eller *søknader om tilskudd* i `flod_sak`.

Mikroapplikasjoner danner grunnlag for god gjenbrukbarhet av funksjonalitet på tvers av applikasjoner samt gode muligheter for skalerbarhet eller tilgjengelighet (man kan f.eks. kjøre flere instanser av samme mikroservice).

Diagrammet under viser hvilke sentrale mikroapplikasjoner finnes nå som prosjektene booking og tilskudd er utviklet.



Hver mikroapplikasjon er under versjonkontroll under eget `git` repository, og skal kunne eksekveres.

Kort om de forskjellige mikrotjenestene og applikasjonene:

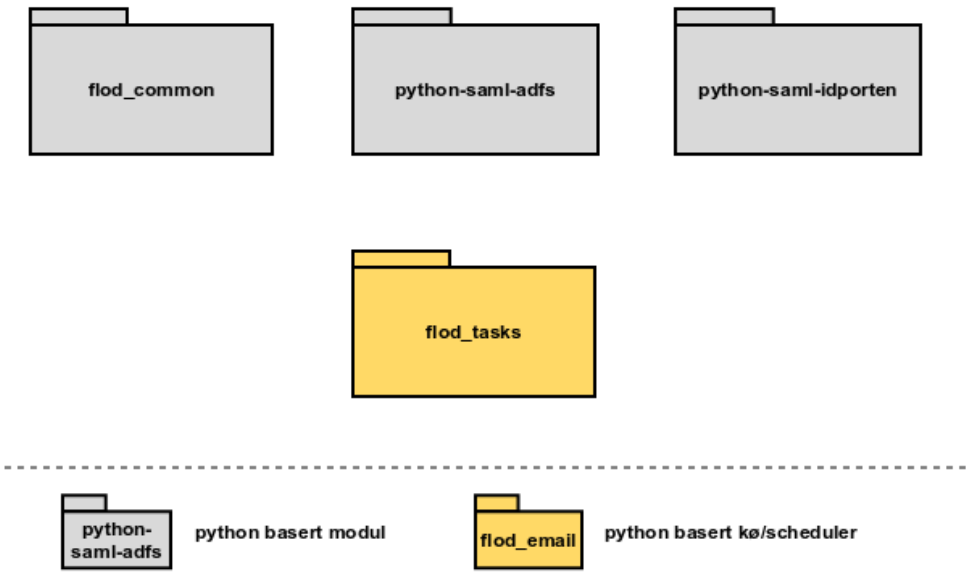
- HTML Applikasjon: I hovedsak HTML, Javascript og CSS klientkode. Koden leveres av en Python Flask applikasjon til sluttbrukeren, men applikasjonen inneholder svært lite logikk og persisterer ikke data. Den bruker de andre mikrotjenestene for å finne frem til informasjonen, enten direkte fra klientkoden (javascript) eller fra serversiden (python).
- REST tjeneste: En Python Flask applikasjon med tilhørende database. Applikasjonen tilbyr ikke noe GUI men et REST API..
- HTML Applikasjon og REST tjeneste: Applikasjoner av denne typen er i utgangspunktet applikasjoner som var opprinnelig kun HTML applikasjoner som som måtte utvides til å tilby et REST API.

Grunnen til at `flod_tilskudd_portal` er mer enn en HTML Applikasjon er følgende:

- Det dukket opp behov for å implementere funksjonalitet som hørte ikke til noen av de andre mikrotjenester.
- Funksjonaliteten var avansert og passet dårlig til en implementasjon i javascript på klient siden. Javascript koden på klient siden skal ta seg av å hente og vise data, ikke av noe avansert business logikk.
- Det ble derfor bestemt å tilby et REST API i denne modulen også.

Moduler

Det finnes en del kode som de forskjellige mikrotjenestene kan dele, vi har også noen slike moduler.

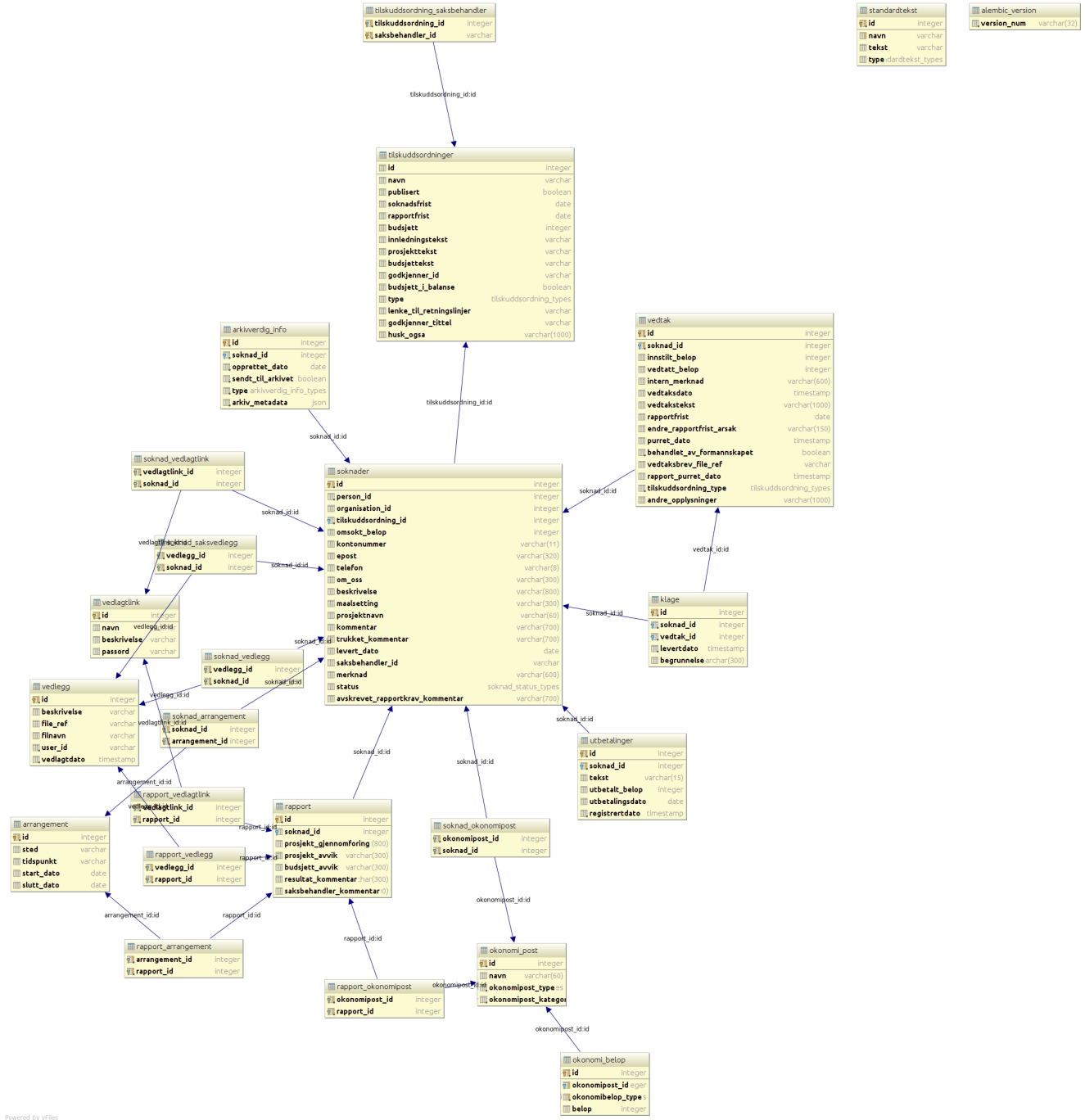


Kort om modulene:

- flod_common: utility og felles pythonkode som delest på tvers av mikrotjenestene
- python-saml-adfs: python bibliotek som brukes i forbindelse med adfs pålogging
- python-saml-idporten: python bibliotek som brukes i forbindelse med id-porten pålogging
- flod_tasks: En applikasjon som kan brukes for diverse schedulingsoppgaver, for eksempel arkivering og sende purringer etter x antall dager etter utløptfrist. Modulen er egentlig mer en felles tjeneste for alle mikrotjenestene, men siden den kun brukes av systemet og tilbys ikke som tjeneste til utsiden listes den i denne seksjonen.

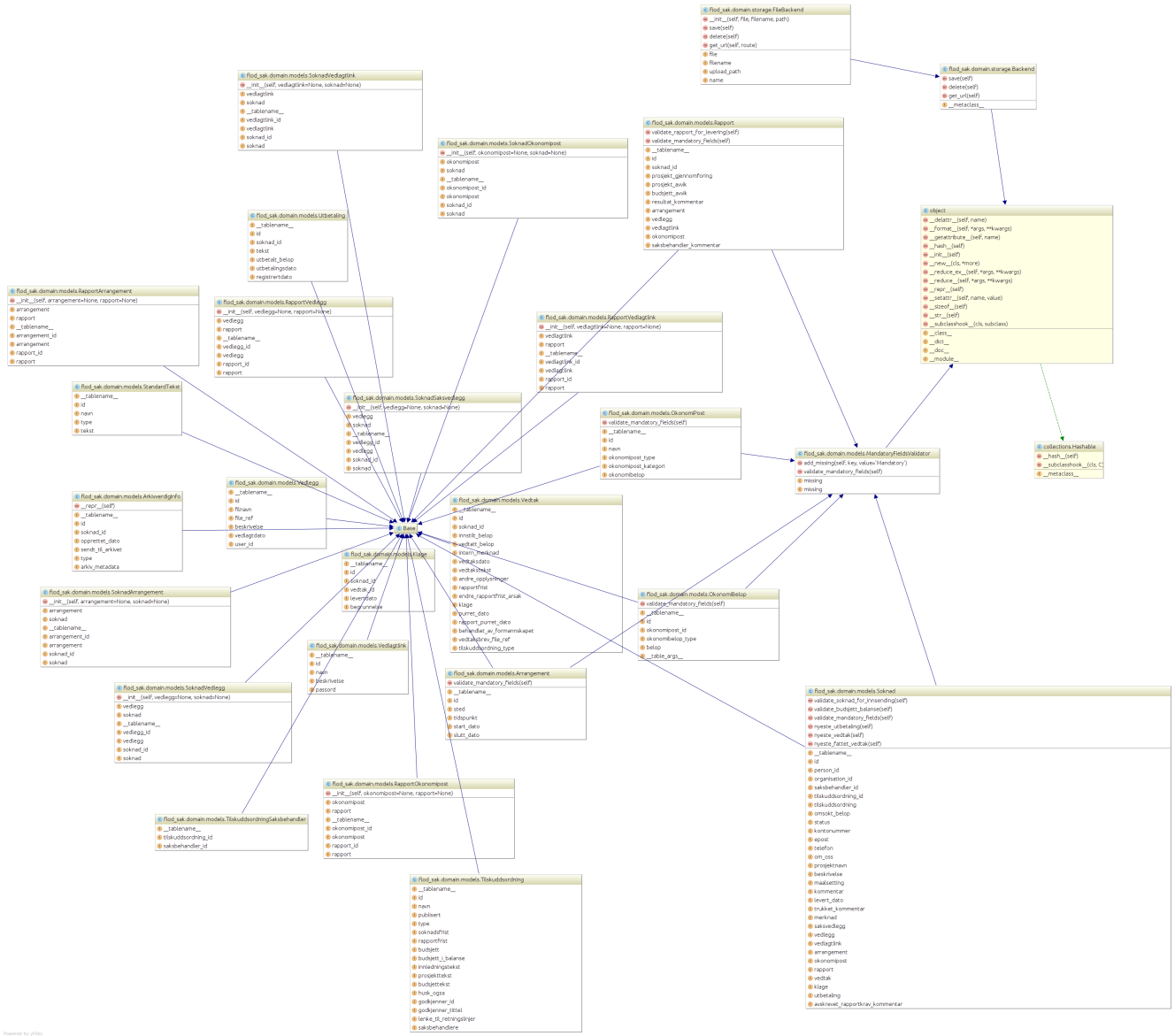
Diagrammer og modeller

Database modell



Domene modell (python)

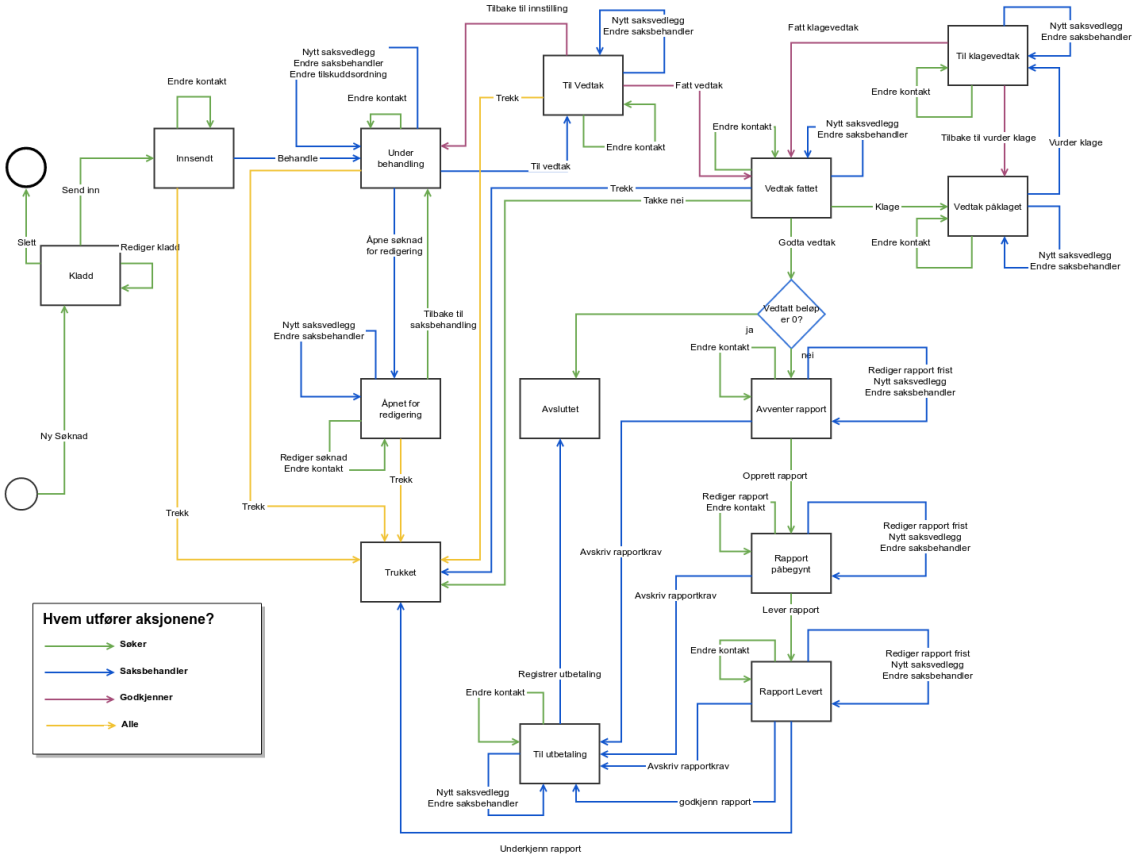
OBS: Base-klassen er en klasse alle de objektene som persisteres i databasen skal arve fra.



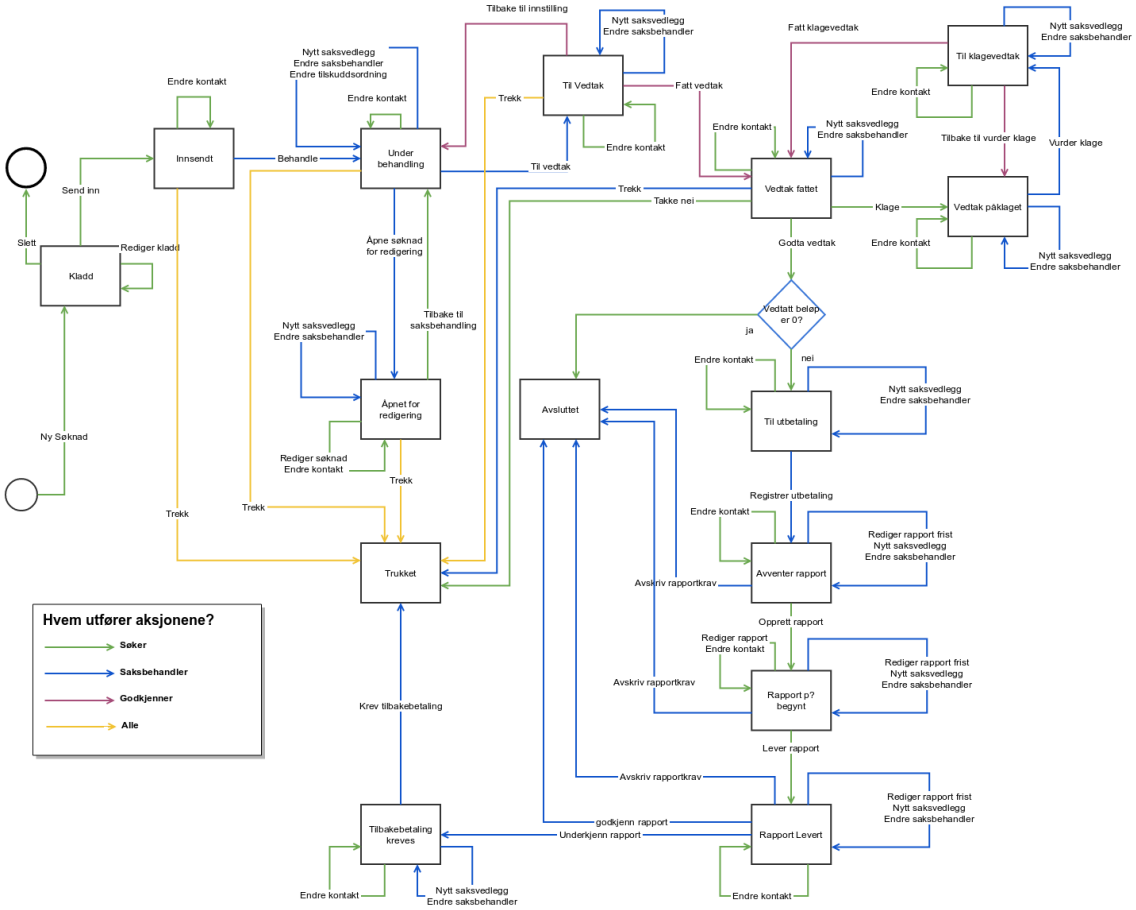
4 - Process view

Process view gir oversikt over dokumentasjon relatert til prosesser og tilstandsmaskiner i applikasjonen.

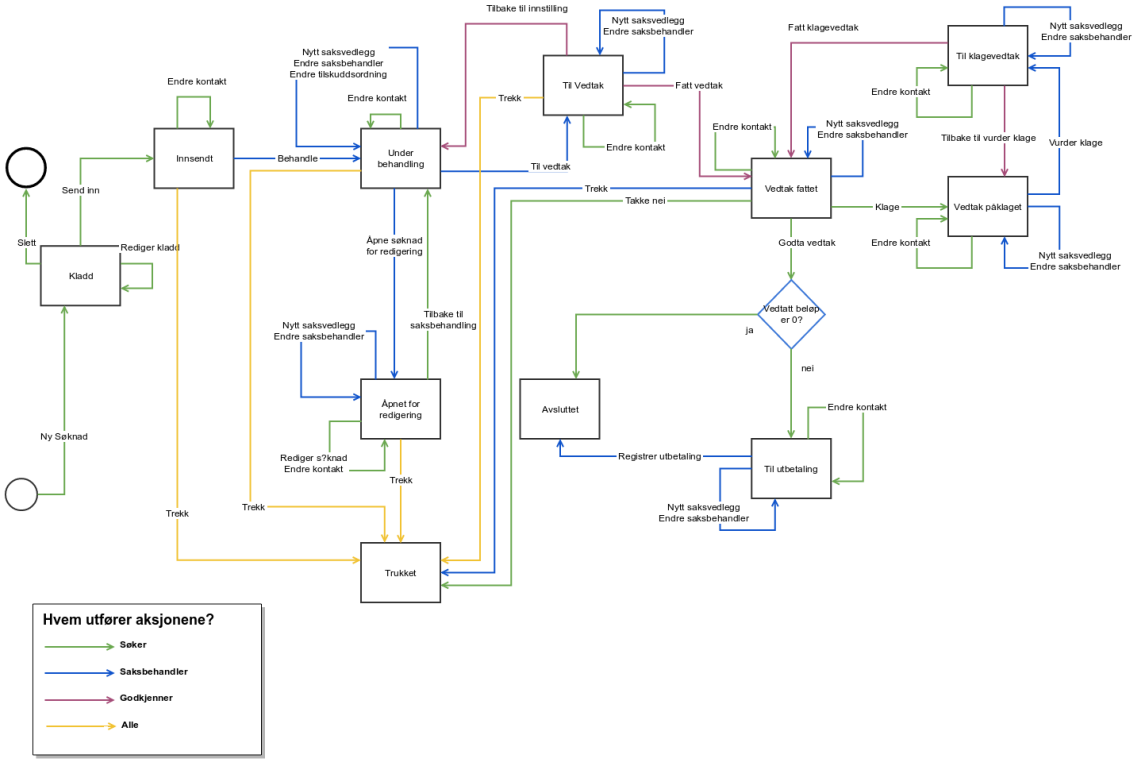
1 - Statemachine - Tilskuddsordning med Etterskuddsutbetaling



2 - Statemachine - Tilskuddsordning med Forskuddsutbetaling



3 - Statemachine - Tilskuddsordning uten rapportkrav



4 - Tilstandsendringer som fører til utsendelse av epost

Aksjon	Til	Kommentar
Søknad sendt inn	Søker*	
	Aktør	Begge epost-adressene som er registrert i Aktørbasen
	Saksbehandlere for tilskuddsordningen	Gjelder bare for tilskuddsordninger som ***ikke*** har søknadsfrist
Søknad sendt tilbake til saksbehandler	Søker*	
	Saksbehandlere for tilskuddsordningen	Gjelder bare for tilskuddsordninger som ***ikke*** har søknadsfrist
Vedtak fattet	Søker*	
Klagevedtak fattet	Søker*	
Rapport levert	Søker*	
	Saksbehandler for søknaden	
Rapport underkjent	Søker*	

*Epost-adressen oppgitt på søknaden

5 - Purring per epost

Disse purringene er konfigurerebare i systemkonfigurasjonen, både når det gjelder antall og tidspunkt. Per i dag er disse satt opp:

Type	Når	Til	Kommentar
Purring på svar på vedtak	14 dager *etter* vedtaket er fattet	Søker*	
Purring på svar på vedtak	30 dager *etter* vedtaket er fattet	Søker*	
Purring på rapport	14 dager *før* rapportfrist	Søker*	
Purring på rapport	1 dager *etter* rapportfrist	Søker*	
Purring på rapport	14 dager *etter* rapportfrist	Søker*	

*Epost-adressen oppgitt på søknaden

6 - Tilstandsendringer som fører til arkivering

Her er oversikt over de endringene som fører til automatisk arkivering:

Aksjon	Utført av	
Behandle søknad	saksbehandler	oppretter saken og legger til en journalpost som inneholder søknaden (pdf) og alle vedlegg
Send tilbake til saksbehandling	søker	legger til ny journalpost som inneholder søknaden (pdf) og alle vedlegg
Send rapport	søker	legger til ny journalpost som inneholder rapporten (pdf)
Klage på vedtak	søker	legger til ny journalpost som inneholder klage (txt)
Opprettholde klage på vedtak	søker	legger til ny journalpost som inneholder klage (txt)
Fatt vedtak	godkjenner	legger til ny journalpost som inneholder vedtak (pdf)
Fatt klagevedtak	godkjenner	legger til ny journalpost som inneholder vedtak (pdf)
Nytt saksvedlegg	saksbehandler	legger til ny journalpost som inneholder vedlegget

Verdt å merke seg:

- søknaden blir ***ikke*** arkivert før en saksbehandler velger å behandle søkaden. Dette er fordi arkivsystemet krever en saksbehandler ved registering av sak.
- utbetalinger blir ***aldri*** arkivert automatisk fra Tilskuddsbasen, dette kommer av at dokumentet må redigeres manuelt av saksbehandler.

5 - Design View

Design view gir oversikt over de bærende arkitektoniske prinsippene.

Bruk av mikrotjenester fra webapplikasjonene

Applikasjonstyper i flod

Her er noen av de applikasjonene som finnes i flod og deres type (tilskudd og booking):

- flod_users: mikrotjeneste
- flod_organisations: mikrotjeneste
- flod_booking: mikrotjeneste
- flod_auth: mikrotjeneste
- flod_sak: mikrotjeneste
- flod_frontend: webapplikasjon
- flod_admin_frontend: webapplikasjon
- flod_aktor_frontend: webapplikasjon
- flod_tilskudd_portal: webapplikasjon

Mikrotjenestene er applikasjoner som tilbyr en REST/HTTP API, mens webapplikasjonene er applikasjoner som tilbyr WEB/HTTP API.

Avhengigheter mellom applikasjonene i flod

Multiple page application

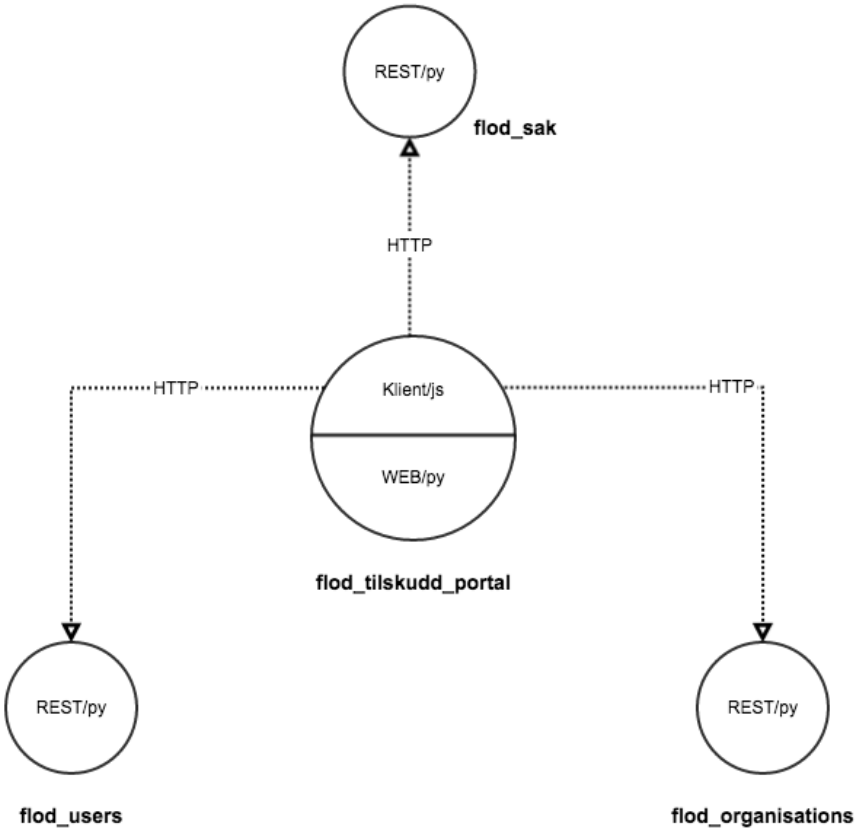
Webapplikasjonene er delt opp i flere sider (Multiple Page Application) og ved oppslag på predefinerte url får man tilbake en gitt side (HTML+js+css++).

Det brukes server-side templates i webapplikasjonene; der har man blant annet mulighet til å få forhåndsgenererte data i sidene før de leveres tilbake til klienten.

Templates på server siden brukes dog lite til å populere sider med dynamisk data, men mest til å "pakketere" sidene.

Prinsipper innenfor en page application

Arkitekturen vår baseres på at webapplikasjonenes serverside kode (python) skal gjøre så lite som mulig. Prinsipper er at oppslag i de forskjellige mikrotjenester skal gjøres direkt fra klientkoden webapplikasjonene leverer (javascript) . HTTP kallene på tvers av applikasjonene ser sånn ut etter at en gitt side er blitt sendt til klient (eksempel):



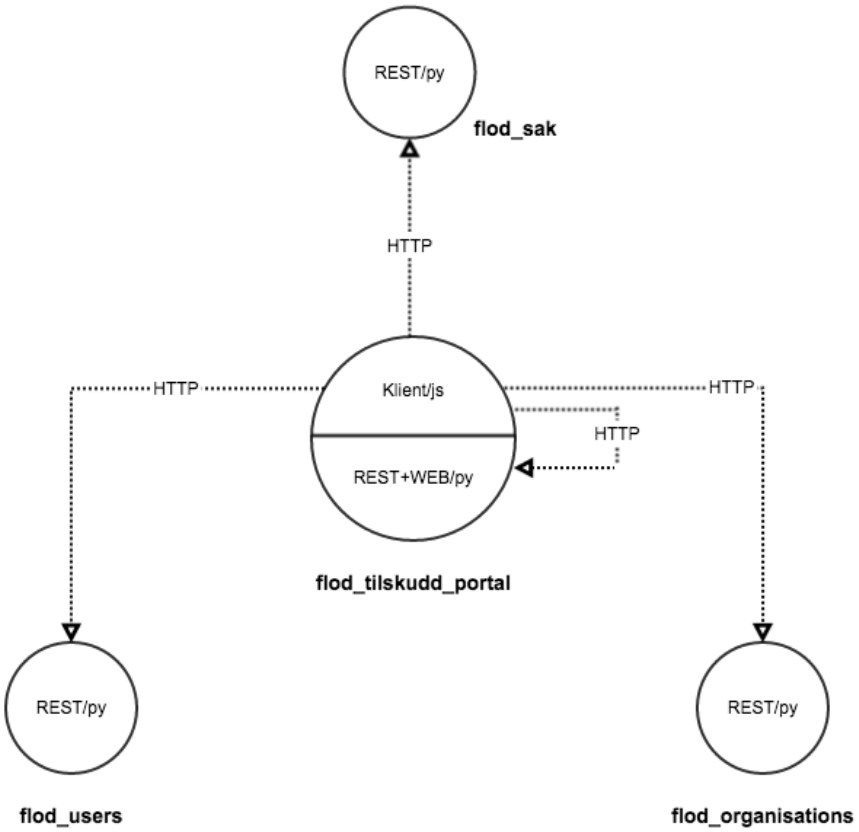
Siden mikrotjenester per definisjon kun har ansvar for en liten del av de systemene så må det av og til gjøres noe ekstra for å berike dataene. Hvis klient siden trenger å få tak i en kombinasjon av informasjon som ligger i forskjellige mikrotjenester så må informasjonen skapes et eller annet sted.

De tre alternativene er følgende:

1. mikrotjenestene kommuniserer seg i mellom og tilbyr tjenester som leverer mer enn den informasjon de har ansvar for
2. serversidekoden til webapplikasjonene utvides med tjenester som tilbyr den informasjonen (webapplikasjonen fungerer da også som REST/HTTP tjeneste, muligens kun med GET støtte)
3. klientkoden tar seg av å hente og agreggere informasjonen fra forskjellige tjenester

Alternativ 1 ødelegger for mikrotjeneste arkitekturen (de blir tett bundet til hverandre). Forskjellen på alternativ 2 og 3 er hvor vi legger logikken for å aggregere dataene. Vi har valg å gå for en blanding av strategi nummer 2 og 3.
For enkelt søknader har vi valgt å gå for alternativ 3, det vil si det gjøres et webservice-kall for å hente søknaden, og om *nødvendig* gjøres det et for hver av å hente saksbehandler, organisasjon, person, osv.
For søknadslisten har vi benyttet oss av alternativ 2, der er det mest effektiv å hente alle søknader (n) og deretter hente alle unike organisasjoner (o) og personer (p) tilhørende disse (her vet vi at vi alltid ønsker å ha med organisasjon og person). Dette gjøres dermed ved hjelp av kun tre webservice kall i stedet for 1 + (n*o) + (n*p).

Oppslag fra klientkoden kan derfor se sånn ut der behovet for beriket informasjonen dukker opp (basert på samme eksempel som over):



Webapplikasjonene i flod kan derfor i ny og ne også tilby tjenester.

Observasjon

Det finnes også noen få avhengigheter mellom mikroapplikasjonene, arvet fra Booking prosjektet.

Implisitte relasjoner på tvers av mikrotjenester

Problemstilling

Micro services skal ikke være avhengige av hverandre. Men det dukker allikevel opp behov for å kunne uttrykke relasjoner mellom de objektene forskjellige micro services er ansvarlig for.

En *søknad* i *flod_sak* er for eksempel semantisk sett relatert til en *organisasjon* i *flod_organisations*.

Løsning

Dette problemet ble løst i Tilskuddsbasen ved å lagre id-en til det eksterne objektet.

Relasjonen mellom en *soknad* (lagret i `flod_sak`) og dens *organisation* (lagret i `flod_organisations`)fanges for eksempel i `flod_sak` ved å lagre soknad.person_id.

Disksusjon

Problemstillingen er ikke like triviell som den ser ut, og det kan nevnes at det er blitt diskutert flere løsninger. Den valgte strategien løser ikke alt, men den løser nok for Tilskuddsbasen og er såpass triviell at det blir lett å videreutvikle den hvis det trenges.

Noen av de spørsmålene man må stille seg hvis man skal revurdere dette er listet under.

Hvor mye informasjon om den eksterne ressurs skal lagres?

Hva trenger man å vite for å slå opp en ekstern ressurs? Den mest naiv tilnærming er å lagre hele url-en til ressursen. Men den tilnærmingen gjør at man må garantere at den url-en endrer seg aldri. Hvis den gjør det så skal dataene måtte migreres i alle de stedene hvor de brukes. Vi ønsker ikke en slik løsning:

- å finne ut av hvor disse url-ene ligger på tvers av flere mikrotjenester kan bli en stor jobb
- slike migreringsscripter tar tid å skrive
- feil som oppstår når de kjøres er kritiske.

Vi ble derfor enige i at både host/port delen av urlen, samt stien til den entitettypen man refererer til skulle man ikke lagre i databasen. Det eneste vi mener er absolutt minimum som må lagres i basen er id-en til entiteten.

Bevis på at dette er nok

La oss si at vi har en *soknad* i `flod_sak` som er knyttet til en *person* lagreg i `flod_organisations`. Hvis vi også sier at *personen* slås opp med https://prod.trondheim.kommune.no:7778/person/41 hvordan kan det å lagre soknad.person_id = 41 være nok?

Det er det fordi resten av url-en blir skapt av applikasjonskoden på denne måtten:

- protocolen, applikasjonen og portnummeret til `flod_organisations` er konfigurasjonsparametre som `flod_sak` har tilgang til, de er tilgjengelige når applikasjonen kjører og vi trenger derfor ikke å lagre dem i basen.
- utvikleren *vet* under hvilken sti en person ligger i `flod_organisation` (i det tilfelle `/person`) så det trenger heller ikke å bli lagret.

Hele url-en til personen kan derfor skapes basert på den lagrede id-en.

6 - Infrastructure view

Infrastructure view er dokumentasjon av hvilken programmvare applikasjonen bygger på.

Rammeverk

Kun de mest sentrale rammeverk er listet opp her, for en presis oversikt vennligst referer til kodebasen.

Python

Navn	Homepage	Beskrivelse
Flask	http://flask.pocoo.org/	Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions. And before you ask: It's BSD licensed!
Flask-RESTful	https://flask-restful.readthedocs.org/en/0.3.0/	**Flask-RESTful** is an extension for Flask that adds support for quickly building REST APIs. It is a lightweight abstraction that works with your existing ORM/libraries. Flask-RESTful encourages best practices with minimal setup. If you are familiar with Flask, Flask-RESTful should be easy to pick up.
SQLAlchemy	http://www.sqlalchemy.org/	SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.
bouncer	https://github.com/jtushman/bouncer	Simple Declarative Authentication DSL inspired by Ryan Bates' excellent cancan library
Flask-bouncer	https://github.com/jtushman/flask-bouncer	Flask declarative authorization leveraging bouncer

celery	http://www.celeryproject.org/	Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well. The execution units, called tasks, are executed concurrently on a single or more worker servers using multiprocessing, Eventlet, or gevent. Tasks can execute asynchronously (in the background) or synchronously (wait until ready).
Jinja2	http://jinja.pocoo.org/docs/dev/	Jinja2 is a modern and designer-friendly templating language for Python, modelled after Django's templates. It is fast, widely used and secure with the optional sandboxed template execution environment
alembic	https://pypi.python.org/pypi/alembic	Alembic is a database migrations tool written by the author of SQLAlchemy.
suds-jurko	https://pypi.python.org/pypi/suds-jurko/0.6	Based on the original 'suds' project by Jeff Ortel (jortel at redhat dot com) hosted at 'http://fedorahosted.org/suds' 'Suds' is a lightweight SOAP-based web service client for Python licensed under LGPL (see the LICENSE.txt file included in the distribution).
xhtml2pdf	https://pypi.python.org/pypi/xhtml2pdf	xhtml2pdf is a html2pdf converter using the ReportLab Toolkit, the HTML5lib and pyPdf. It supports HTML 5 and CSS 2.1 (and some of CSS 3). It is completely written in pure Python so it is platform independent.

Javascript

Navn	Homepage	Beskrivelse
Backbone.js	http://backbonejs.org/	Backbone.js gives structure to web applications by providing **models** with key-value binding and custom events, **collections** with a rich API of enumerable functions, **views**with declarative event handling, and connects it all to your existing API over a RESTful JSON interface.
Marionette.js	http://marionettejs.com/	Backbone.Marionette is a composite application library for Backbone.js that aims to simplify the construction of large scale JavaScript applications. It is a collection of common design and implementation patterns found in the applications that we have been building with Backbone, and includes pieces inspired by composite application architectures, event-driven architectures, messaging architectures, and more.
jquery	http://jquery.com/	jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

underscore.js	http://underscorejs.org/	Underscore is a JavaScript library that provides a whole mess of useful functional programming helpers without extending any built-in objects. It's the answer to the question: "If I sit down in front of a blank HTML page, and want to start being productive immediately, what do I need?" … and the tie to go along with jQuery's tux and Backbone's suspenders.
backbone.stickit	https://github.com/NYTimes/backbone.stickit	Backbone's philosophy is for a View, the display of a Model's state, to re-render after any changes have been made to the Model. This works beautifully for simple apps, but rich apps often need to render, respond, and synchronize changes with finer granularity. Stickit is a Backbone data binding plugin that binds Model attributes to View elements with a myriad of options for fine-tuning a rich app experience. Unlike most model binding plugins, Stickit does not require any extra markup in your html; in fact, Stickit will clean up your templates, as you will need to interpolate fewer variables (if any at all) while rendering. In Backbone style, Stickit has a simple and flexible api which plugs in nicely to a View's lifecycle.

CSS

Navn	Homepage	Beskrivelse
Bootstrap.css	http://getbootstrap.com/css/	Global CSS settings, fundamental HTML elements styled and enhanced with extensible classes, and an advanced grid system.

Tredjeparts applikasjoner

Kun de mest sentrale applikasjonene er listet opp her, for en presis oversikt vennligst referer til kodebasen.

Navn	Homepage	Beskrivelse
postgresql	http://www.postgresql.org/	PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It includes most SQL:2008 data types, including INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. It also supports storage of binary large objects, including pictures, sounds, or video. It has native programming interfaces for C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, among others, and exceptional documentation.

nginx	http://nginx.org/	nginx [engine x] is an HTTP and reverse proxy server, as well as a mail proxy server, written by Igor Sysoev. For a long time, it has been running on many heavily loaded Russian sites including Yandex, Mail.Ru, VK, andRambler. According to Netcraft, nginx served or proxied 20.41% busiest sites in November 2014. Here are some of the success stories: Netflix, Wordpress.com, FastMail.FM.
ansible	http://www.ansible.com/home	Ansible is a powerful automation engine that makes systems and applications simple to deploy. ?No custom scripting or custom code. No agents. All using an automation language that's easy for anyone to understand and learn. Just get in, get it done, and make some time for other strategic projects.
postfix	http://www.postfix.org/	What is Postfix? It is Wietse Venema's mail server that started life at IBM research as an alternative to the widely-used Sendmail program. Postfix attempts to be fast, easy to administer, and secure. The outside has a definite Sendmail-ish flavor, but the inside is completely different.
libreoffice	http://www.libreoffice.org/download/libreoffice-fresh/	LibreOffice is a powerful office suite; Its clean interface and its powerful tools let you unleash your creativity and grow your productivity. LibreOffice embeds several applications that make it the most powerful Free & Open Source Office suite on the market
RabbitMQ	https://www.rabbitmq.com	Robust messaging for applications. Easy to use. Runs on all major operating systems. Supports a huge number of developer platforms. Open source and commercially supported.

7 - Deployment View

Informasjon relatert til deployment av applikasjonene dokumenteres her.

Ansible

Ansible brukes i forbindelse med ut-rulling og konfigurasjon av de forskjellige miljøene Tilskuddsbasen kjører i (brukes også i Bookingbasen). Det vil si at ut-rulling til utviklingsmiljøet, staging og produksjon bruker denne mekanismen.

For å forstå hvordan ansible opererer er det viktig å lese dokumentasjonen (http://www.ansible.com/how-ansible-works anbefalles som introduksjon) på og se på koden i Tilskuddsbasen.

Ansible i Tilskuddsbasen

Når man kloner repositoriet `flod_tilskudd_deploy` får man tilgang til de oppskriftene som benyttes for å deploye løsningen. Disse kalles *Ansible playbooks*. Oppskriftene er generelle, og kan brukes for å deploye løsningen på lokal maskin, på staging og på produksjon.

Oppskriftene er parameteriserte med variabler, og variablene er definerte i skript som ligger i `provisioning/host_vars`.

Kryptering av `host_vars` skriptene

De skriptene som inneholder variablene er i krypterte, dette for å kunne distribuere dem trygt selv om de inneholder sensitiv informasjon (som f.eks. navn til servere, eller brukernavn/passord til eksterne tjenester som Brønnøysundregistrene). Man trenger i utgangspunktet ikke å endre på disse.

Skape et nytt `host_vars` skript

Eksempel på variablene som må settes i produksjon ligger i provisioning/host_vars/production_example.

Først må man opprette et inventory som peker til maskin(ene) man ønsker å deploye på. F.eks slik:

```
# flod_deploy/inventory
[production]
1.2.3.4
```

Hvor 1.2.3.4 er IP-adressen til maskinen. Deretter må man kopiere provisioning/host_vars/production_example til provisioning/host_vars/1.2.3.4. Variablene i denne fila vil dermed gjelde ved deploy til 1.2.3.4.

For å deploye med Ansible, kjører man følgende kommando i flod_tilskudd_deploy repositoriet

```
ansible-playbook -K -i inventory provisioning/playbook.yml
```

Dersom det er satt opp flere miljø i inventory-fila og ikke ønsker å deploye alle, må man kommentere ut de man ikke vil deploye før man kjører deploy-kommandoen.

Merk at man på sin lokale linux-bruker må ha tilgang til repositoriene på git.bouvet.no og GitHub for å kunne deploye.

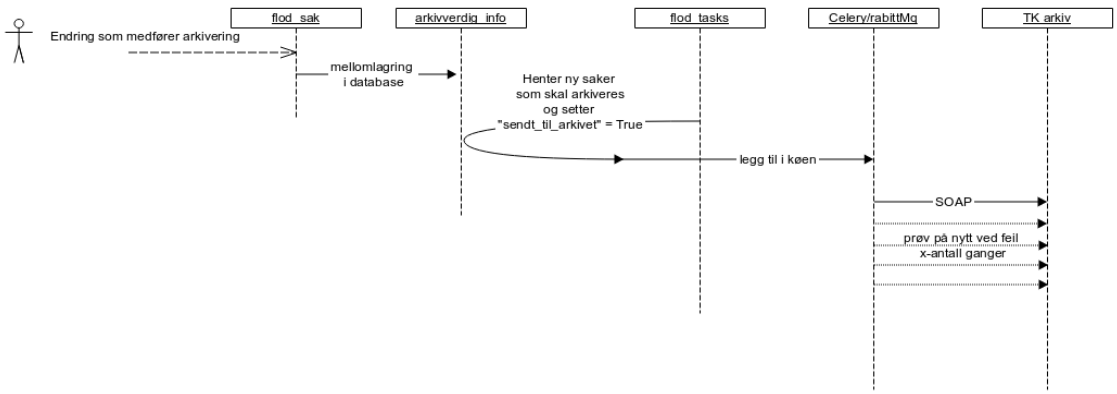
8 - Operational view

Operational view gir oversikt over dokumentasjonen relatert til drift av applikasjonen.

Arkiv

Alle søknader og vedlegg, vedtak osv som er knyttet til søknadene skal arkiveres i Trondheim Kommunes arkivsystem. For å slippe å gjøre dette manuelt, har vi laget en integrasjon for dette. Når en søknad skal arkiveres for første gang opprettes den en ny arkiv sak og **en** ny journalpost, denne inneholder all informasjon om søknaden så langt og knyttes opp mot den som er saksbehandler for søknaden. For alle påfølgende arkiveringer blir det opprettet nye journalposter enkeltvis.

All informasjon som skal arkiveres mellomlagres i en egen tabell før de opprettes som egne jobber i køsystemet.



Det hele starter med en hendelse i Tilskuddsbasen som skal arkiveres. Dette gjøres ved å lagre et nytt innslag i databasetabellen arkivverdig_info. Denne tabellen inneholder en kolonne som inneholder en type (sak eller journalpost) og metadata for det som skal arkiveres. Formatet på metadata er gitt av type, og er basert på tilsendt informasjon fra Trondheim Kommune. Denne ligger som vedlegg på https://jira.bouvet.no/browse/TIL-446 - det kan ha vært korrigeringer i forhold til disse i ettertid, uten at vi har mottatt oppdatert dokumentasjon fra arkiv, sjekk eventuelt implementasjonen for detaljer.

Regelmessig plukkes disse opp av flod_tasks og sendes som jobber til Celery/RabittMQ, hyppigheten er avhengig av konfigurasjon. Samtidig som de plukkes opp endres statusen på de mellomlagrede dataene, ved at attributtet "sendt_til_arkivet" settes til true.

Celery/RabbitMQ gjør så SOAP-kall mot Trondheim Kommunes arkiv system. Endpointene (ulike for sak og journalpost) er konfigurerbare. Dersom arkiveringen mot arkivsystemet feiler vil systemet forsøke å sende på nytt x antall ganger (konfigurerbart). Dette skjer ***uavhengig av årsak***, om det er på grunn av intern feil i arkivet eller om systemene ikke får kontakt med hverandre spiller ingen rolle. Dersom alle forsøkene blir brukt, uten suksess, vil systemet sende ut epost til tilskudd@trondheim.kommune.no (konfigurerbart).

Konfigurering

Adressen til arkiv, antall forsøk på arkivering og hvor lenge det er mellom hvert forsøk konfigureres i hosts_vars filen, mens konfigurering av epost-adresser (mottaker og avsender ved feil) og hvor ofte jobben skal kjøres, gjøres i celeryconfig.

For mer informasjon om konfigurering se 7 - Deployment View og kildekode (celeryconfig_prod.py).

Retrigging av arkivering

Det hender at man ønsker å trigge arkiveringen på nytt. For å gjøre dette trenger man

- ssh tilgang til serveren
- logge på databasen
- oppdatere arkivverdig_info tabellen og sette "sendt_til_arkivet" = False for den jobben/søknaden man ønsker å sende på nytt

fra linux, evt putty e.l på windows

```
ssh trk-web02 # logg på med egen bruker

# dersom sudo-tilgang (ellers les psql dokumentasjon for å logge på med
brukernavnet/passordet som står i systemconfig'en)
sudo su postgres
psql -d flod_sak
```

psql

```
-- for å oppdatere en jobb
update arkivverdig_info set sendt_til_arkivet = False where id = xyz;

-- for å oppdatere alt på en søknad
update arkivverdig_info set sendt_til_arkivet = False where soknad_id = xyz;

-- for å avslutte psql
\q
```

Logging

Logging av bruk og innsyn i Tilskuddsbasen

Fra krav:
”Løsningen skal ha funksjonalitet for å logge *hvem* som har vært inne og gjort endringer. Dette kan gjelde både saksbehandler og eksterne brukere”.

Bouvet skal logge følgende data:

- oppgavetype
- pålogget bruker
- tidspunkt for hendelse
- mulig informasjonsmelding (fritekst) som detaljerer hendelsen

Loggen skal kunne hentes av en administrator, men vi har ikke lagt opp til å lage et grafisk brukergrensesnitt for å vise den i applikasjonen.

Innsynslogg skal ikke være tilgjengelig for mange og det å lage et eventuelt brukergrensesnitt vurderes som en utviklingsoppgave som kan utsettes til en senere versjon.

Bouvet har valgt å dele inn i separate logger:

- Logging av HTTP Requests (med unntak av GET), med samtlige parametre, samt RESPONSE
- Logging av database queries (DML, og da ikke SELECT).

Loggene kan fåes tak i på følgende måte:

HTTP

1. Loggen blir skrive til {apps_etc}/logs på serveren. {apps_etc} settes default til /opt/tilskudd_etc. Loggen vil ha navnet http_requests.log
2. I utgangspunktet vil det bli laget en ny loggfil hver dag, omtrent ved midnatt, og gamle loggfiler vil få appendikset [YYYY-MM-DD], f.eks. http_requests.log.2014-09-17
3. Loggfiler vil for øyeblikket *IKKE* bli komprimert eller tatt spesifikt backup av utenom de vanlige backup-rutinene ved Alcom.
4. Radene i loggfila vil ha formatet:

DATE TIME - MODUL - TYPE REQUEST/RESPONSE PAIR ID UserID/Anonym Data

Data vil variere avhengig om det er en REQUEST eller RESPONSE som logges, og hvordan dataene sendes. REQUEST er dataene som kommer fra brukeren, mens RESPONSE er dataene som leveres til brukeren. REQUEST og RESPONSE er knyttet sammen med REQUEST/RESPONSE ID som er en UUID for å finne parene som henger sammen. Hver RESPONSE SKAL ha en REQUEST, men en REQUEST kan mangle en RESPONSE hvis det skjer en feil på serveren. Da skal det isf. komme en melding om feil, med mindre loggінga også feilet.

Loggen skal i utgangspunktet lagre entries i kronologisk rekkefølge, men man kan ikke anta at en RESPONSE etterfølger rett etter sin REQUEST. Derfor brukes Pair ID for å kunne identifisere disse.

Nedenfor er eksempel på REQUEST med tilhørende RESPONSE (samme PAIR ID).

Man finner hvilken bruker (eller anonym) ved å ta UserID som er en ID hash og slår opp i bruker-tabellen i flod_auth:

SQL spørring

```
select * from "user" where
id='755d94d88bb4da735db0e619006a6ebda892983bae3ade0855c3c866';
```

resultat

```
1 | 755d94d88bb4da735db0e619006a6ebda892983bae3ade0855c3c866 | id_porten
| 01039300601 |         1 | 2014-09-19 08:18:21.658468 |
887e8bcc-85e6-4100-8dbe-dfa8888a42c7 | 2014-09-19 12:13:16.656593 | \x80027d71012e
```

Eksempel REQUEST

logg utdrag

```
2014-09-19 12:13:20,584 - http_logger - SakAPI HTTP_AUDIT:REQUEST
ebf8663a-d13e-47af-9056-82fce3aaabe4
755d94d88bb4da735db0e619006a6ebda892983bae3ade0855c3c866 POST
http://localhost:8080/api/v1/soknad/ {"vedtatt_belop": 0, "person": {"uri":
"/persons/1"}, "omsokt_belop": 0, "tilskuddsordning_id": 1, "innstilt_belop": 0}
```

Eksempel RESPONSE

logg utdrag

```
2014-09-19 12:13:20,642 - SakAPI HTTP_AUDIT:RESPONSE
ebf8663a-d13e-47af-9056-82fce3aaabe4 201 CREATED "{\n    \"arrangement\": [], \n
\"beskrivelse\": null, \n    \"epost\": null, \n    \"id\": 7, \n    \"kommentar\":
null, \n    \"kontonummer\": null, \n    \"levert_dato\": null, \n
\"maalsetting\": null, \n    \"okonomipost\": [], \n    \"om_oss\": null, \n
\"omsokt_belop\": 0, \n    \"organisation\": {\n        \"id\": 0, \n
\"uri\": null\n    }, \n    \"person\": {\n        \"id\": 1, \n        \"uri\":
\"/persons/1\"\n    }, \n    \"prosjektnavn\": null, \n    \"status\": \"Kladd\",
\n    \"telefon\": null, \n    \"tilskuddsordning\": {\n        \"budsjett\":
1000000, \n        \"budsjettekst\": \"Frivillighetsmillionen 2014 -
budsjettekst\", \n        \"forhandsutbetaling\": true, \n        \"id\": 1, \n
\"innledningstekst\": \"Frivillighetsmillionen 2014 - innledningstekst\", \n
\"navn\": \"Frivillighetsmillionen 2014\", \n        \"prosjekttekst\":
\"Frivillighetsmillionen 2014 - prosjekttekst\", \n        \"publisert\": true, \n
\"rapportfrist\": null, \n        \"soknadsfrist\": \"2014-12-31\"\n    }, \n
\"trukket_kommentar\": null, \n    \"vedlagtlink\": [], \n    \"vedlegg\": [], \n
\"vedtatt_belop\": 0\n}\n"
```

En administrator skal kunne parse loggen med vanlige tekstsøksverktøy på Unix/Linux som f.eks. GREP, AWK, SED osv.

Database

1. Hver tabell i **tilskudd/sak** APIet logges for hver **UPDATE, DELETE, INSERT, TRUNCATE**. ALTER, CREATE og SELECT etc. logges IKKE.
2. Det lages en ny rad i tabellen logged_actions i Audit-skjemaet for hver DML query. Denne lages med nok data og parametre til å finne ut hva som ble endret (fra hva, til hva), og når.
3. Det lagres IKKE hvem som gjorde endringen (vi har kun en felles bruker til databasen), så hvem som gjorde endringen må man finne fra HTTP loggen (ved å sammenligne tidspunkter og actions derfra).

Format

```
event_id | bigint | not null default
nextval('audit.logged_actions_event_id_seq'::regclass) | plain | Unique identifier
for each auditable event
schema_name | text | not null | extended | Database schema audited table for this
event is in
table_name | text | not null | extended | Non-schema-qualified table name of table
event occured in
relid | oid | not null | plain | Table OID. Changes with drop/create. Get with
'tablename'::regclass
session_user_name | text | | extended | Login / session user whose statement caused
the audited event
action_tstamp_tx | timestamp with time zone | not null | plain | Transaction start
timestamp for tx in which audited event occurred
action_tstamp_stm | timestamp with time zone | not null | plain | Statement start
timestamp for tx in which audited event occurred
action_tstamp_clk | timestamp with time zone | not null | plain | Wall clock time
at which audited event's trigger call occurred
transaction_id | bigint | | plain | Identifier of transaction that made the change.
May wrap, but unique paired with action_tstamp_tx.
application_name | text | | extended | Application name set when this audit event
occurred. Can be changed in-session by client.
client_addr | inet | | main | IP address of client that issued query. Null for unix
domain socket.
client_port | integer | | plain | Remote peer IP port address of client that issued
query. Undefined for unix socket.
client_query | text | | extended | Top-level query that caused this auditable
event. May be more than one statement.
action | text | not null | extended | Action type; I = insert, D = delete, U =
update, T = truncate
row_data | hstore | | extended | Record value. Null for statement-level trigger.
For INSERT this is the new tuple. For DELETE and UPDATE it is the old tuple.
changed_fields | hstore | | extended | New values of fields changed by UPDATE. Null
except for row-level UPDATE events.
changed by UPDATE. Null except for row-level UPDATE events.
statement_only | boolean | not null | plain | 't' if audit event is from an FOR
EACH STATEMENT trigger, 'f' for FOR EACH ROW
```

Eksempel entry

SQL spørring

```
select * from logged_actions;
```

resultat

```
5 | public | soknader | 243409 | flod | 2014-09-19 12:13:20.608968+02 | 2014-09-19
12:13:20.610772+02 | 2014-09-19 12:13:20.612452+02 | 30823 | | 127.0.0.1 | 37180 |
INSERT INTO soknader (person_id, organisation_id, tilskuddsordning_id,
omsokt_belop, innstilt_belop, vedtatt_belop, status, kontonummer, epost, telefon,
om_oss, prosjektnavn, beskrivelse, maalsetting, kommentar, levert_dato,
trukket_kommentar) VALUES (1, NULL, 1, 0, NULL, NULL, 'Kladd', NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL) RETURNING soknader.id | I | "id"=>"7",
"epost"=>NULL, "om_oss"=>NULL, "status"=>"Kladd", "telefon"=>NULL,
"kommentar"=>NULL, "person_id"=>"1", "beskrivelse"=>NULL, "kontonummer"=>NULL,
"levert_dato"=>NULL, "maalsetting"=>NULL, "omsokt_belop"=>"0",
"prosjektnavn"=>NULL, "vedtatt_belop"=>NULL, "innstilt_belop"=>NULL,
"organisation_id"=>NULL, "trukket_kommentar"=>NULL, "tilskuddsordning_id"=>"1" | |
f
```

Related articles

Logging

9 - GUI View

Det dokumentet oppsummerer viktige aspekter relatert til grafisk brukergrensesnitt i tilskuddsbasen.

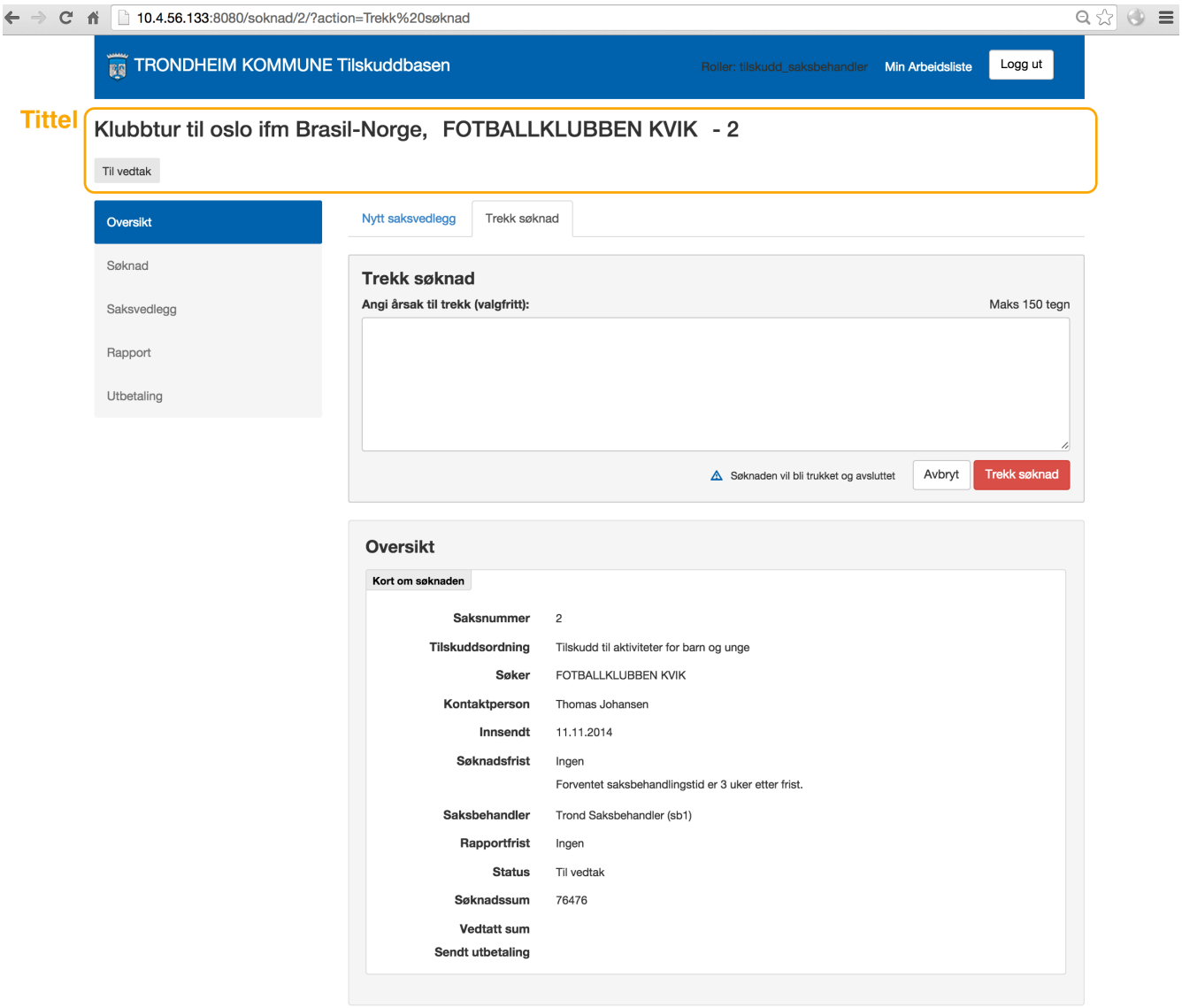
Homogen brukergrensesnitt

Områder i et typisk "se"/"rediger" skjermbilde

Det grafiske brukergrensesnittet (GUI) er bygget for å være så igjenkjennbart som mulig fra skjermbilde til skjermbilde. Det betyr av visse visuelle elementer alltid er plassert på samme sted og at vi tilbyr funksjonalitet på lik måtte på tvers av applikasjonen. Her oppsummerer vi hva et typisk skjermbilde i Tilskuddsbasen består av.

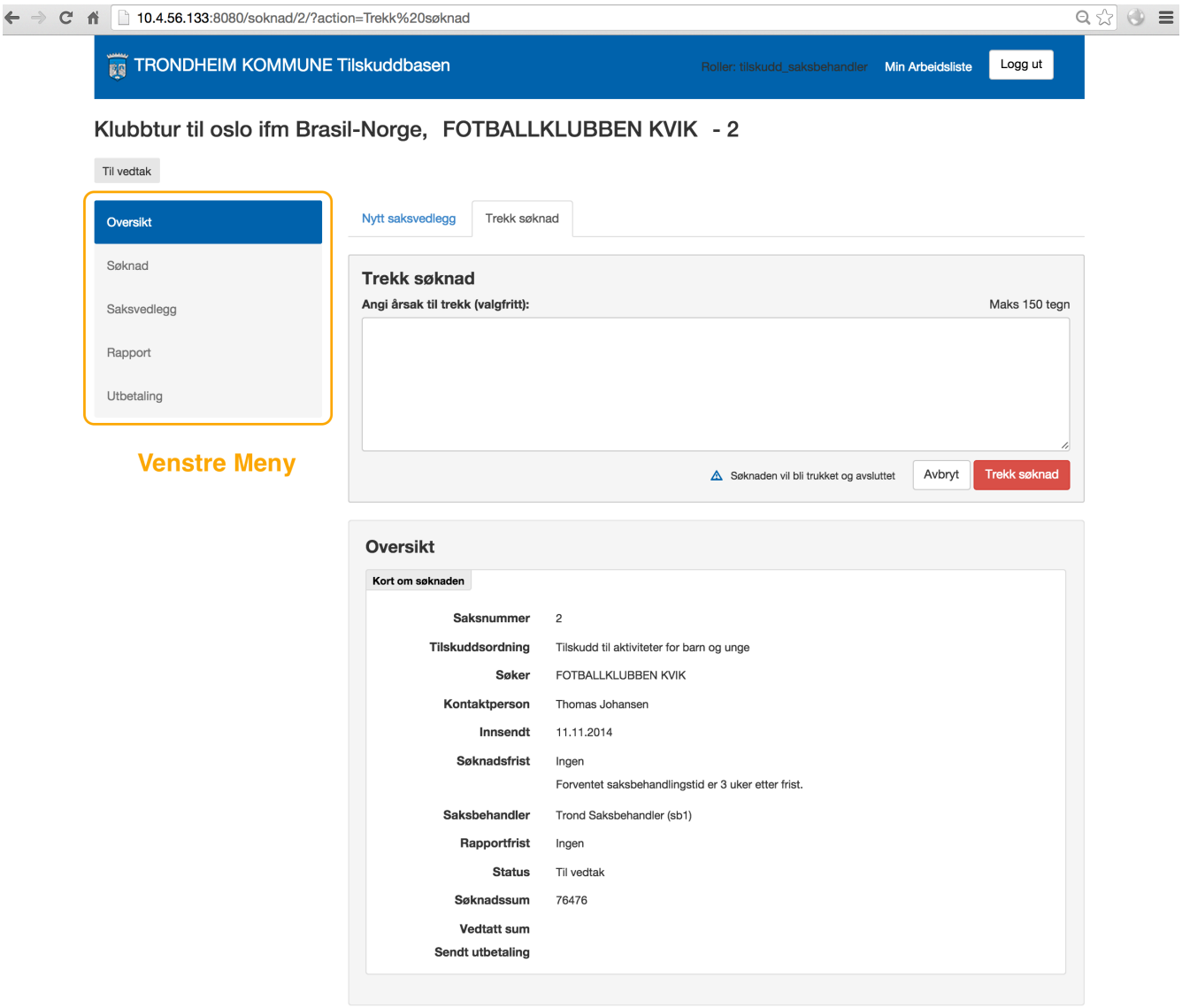
Tittel

Brukes for å beskrive kort hva de objektet som vises er. I eksemplet under kan man for eksempel se at status til søknaden, som er svært viktig informasjon, vises i tittelen.



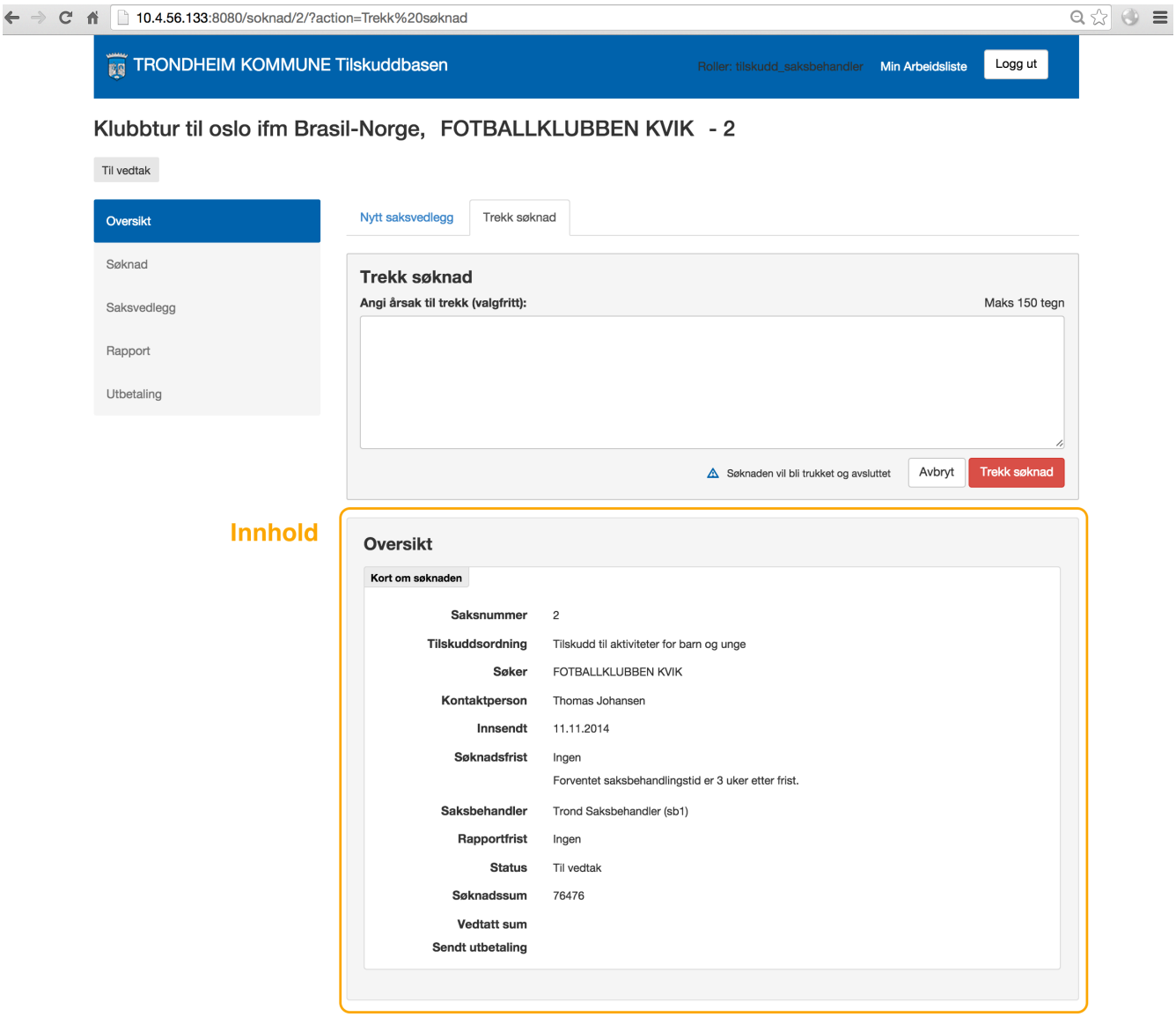
Venstre meny

Venstre meny brukes for å segmentere informasjonen. Når GUI viser informasjon uten at brukeren skal kunne redigere den ("se" modus) så blir inndelingen basert på hvilke aspekter det objektet som vises har. En søknad har blant sine viktige element "Rapport" og "Utbetaling" og de vises som elementer i menyen.



Meny innhold

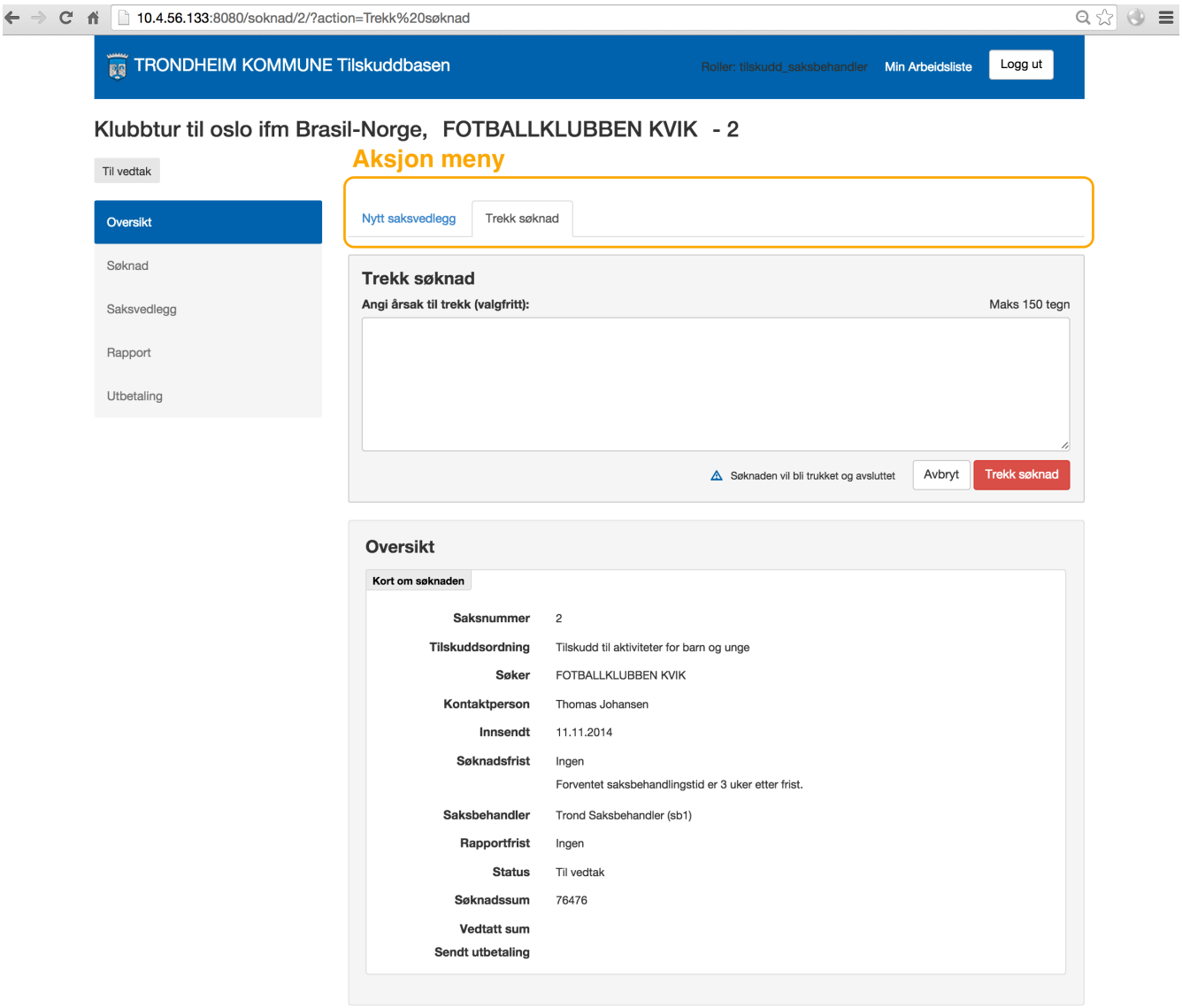
Når man velger noe i meny vises det i område merket som "innhold".



Aksjons meny

Aksjons meny brukes for å legge til hva man kan gjøre med et objekt. Her handler det ikke så mye om å redigere hva objektet består av, men fokuset er mer på det å styre hvilken tilstand objektet er i eller hva man kan gjøre derfra. I eksemplet over har saksbehandleren mulighet for å legge til "Nytt saksvedlegg" eller "Trekk søknad" (aksjonen er åpnet).

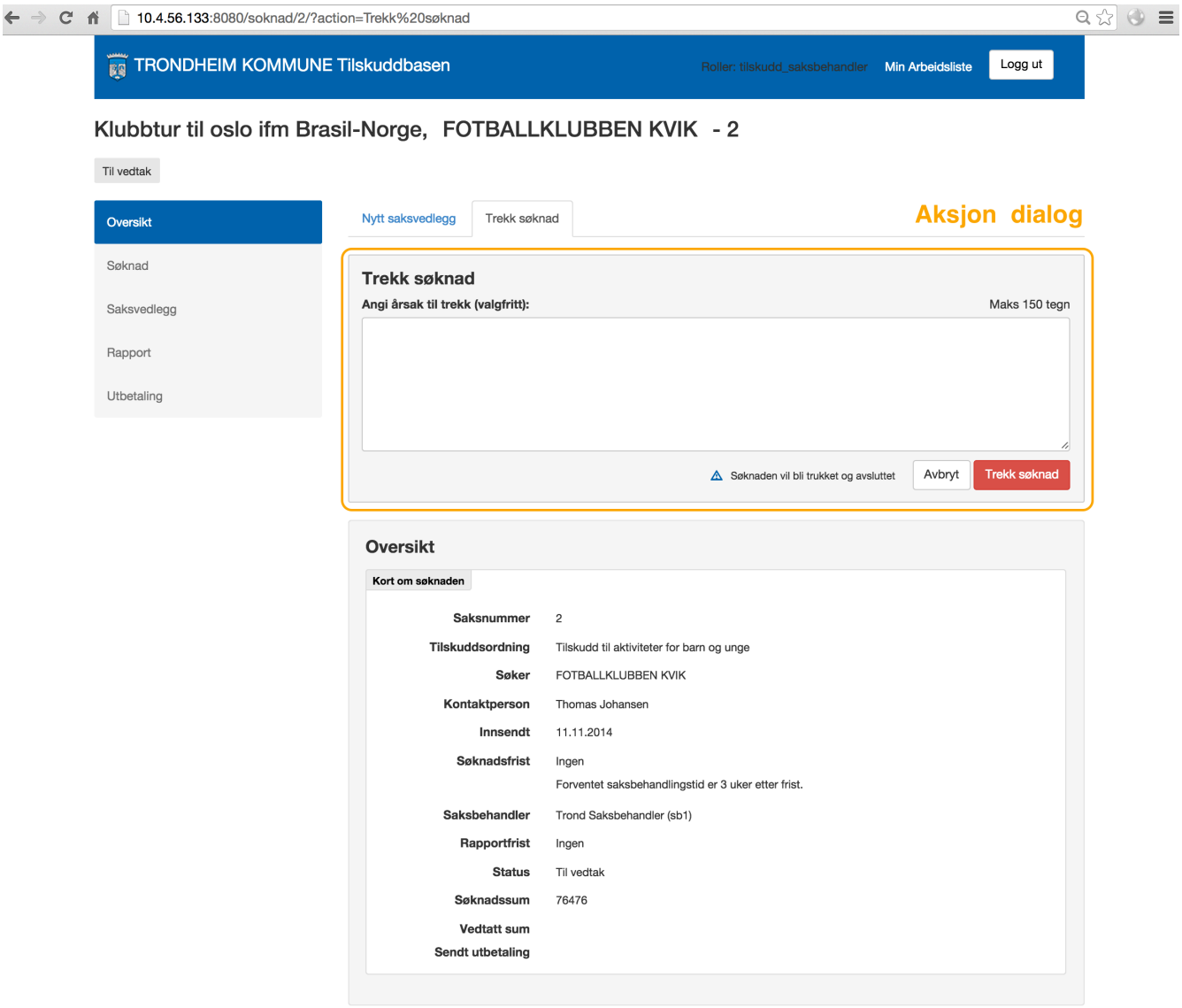
Hvilke aksjoner er tilgjengelig avhenger av forskjellige regler, og for en søknad er status til søknaden samt rollen til den påloggede brukeren avgjørende.



Aksjons dialog

Når man velger en aksjon blir den ikke utført med en gang, men en dialog åpner seg, og det er i denne dialogen at man finner den knappen som utfører aksjonen (style'et i rød/blått og alltid siste knapp til høyre). Det er ofte slik at man skal velge noe eller registrere informasjon når man skal utføre en aksjon så dialogene inneholder ofte felter.

Når en aksjon omfatter mye mer enn et par 2-3 felt så skal man istedet bare åpne en wizard (mer om det lengre ned). Wizard ser i utgangspunktet stort sett lik som bildet i "se" modus, så brukeren skal lett kunne forstå hvordan de fungerer.



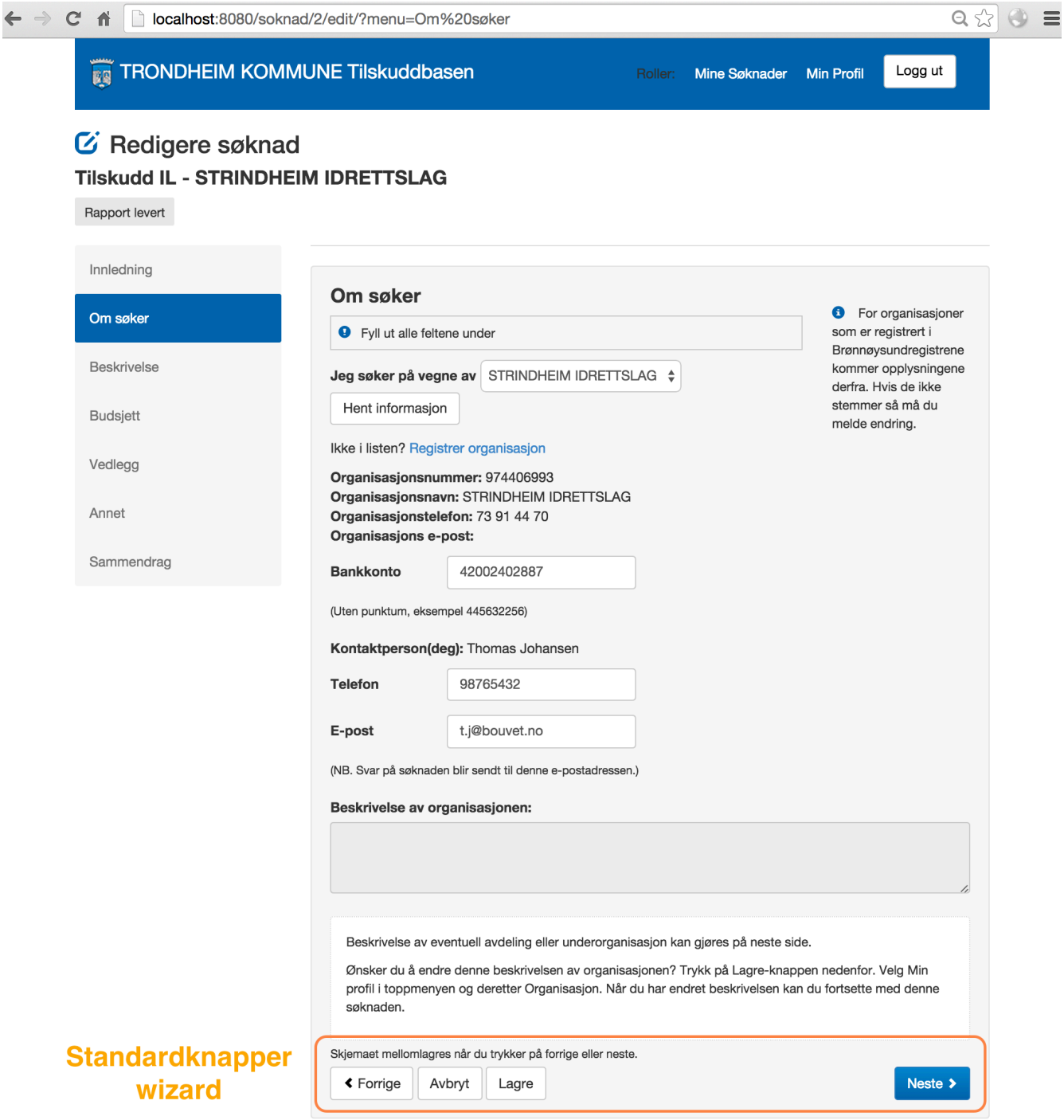
Områder i et typisk "Wizard"/ "Edit" bilde

De mest sentrale objektene i applikasjone har et "se" og er "edit" bilde. Edit bilde er lagt opp til å være wizards med steg slik at informasjonen som skal fylles presenteres i seksjoner/steg. Stegene synliggjøres i venstremenyen.

Det er også et sett av standardknapper på bunnen av hvert wizards bilde som skal tilby grunnfunksjonalitet.

- "Forrige"/"Neste" vises når det er aktuelt (når man er ikke på første side for "Forrige", og siste side for "Neste").
- "Lagre" lagrer innhold i det steget som vises. Det er viktig å få med seg at brukeren *må* trykke på "Neste", "Forrige" eller "Lagre" for at informasjonen skal lagres. Det er viktig at det som står i skjemaet sjekkes av brukeren slik at han ikke overskriver informasjon uten å mene det.
- "Avbryt" knappen lukker "wizard" og åpner "se" dialogen for det objektet "wizard" var brukt på. Grunnen til at knappen heter "Avbryt" og ikke "Lukk" er for å legge vekt på at de ulagrede endringene man kan ha gjort i dialogen *ikke* lagres hvis man trykker på knappen. Det gir mulighet til å angre på endringer man har potensielt gjort i dialogen.

Redigere søknad

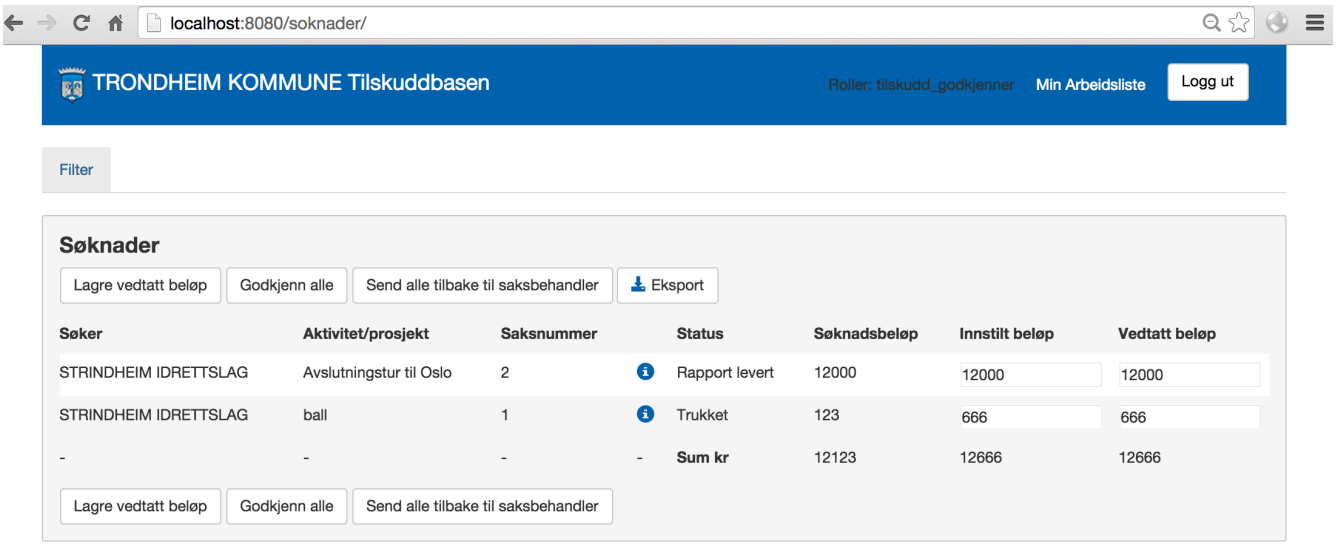


Områder i et "Se liste" bilde

En variant av "Se" bilder er "Se liste" bilder, som viser lister av objekter. Det er per skrivende tidspunkt kun ett slikt bilde, landingsside for brukere.

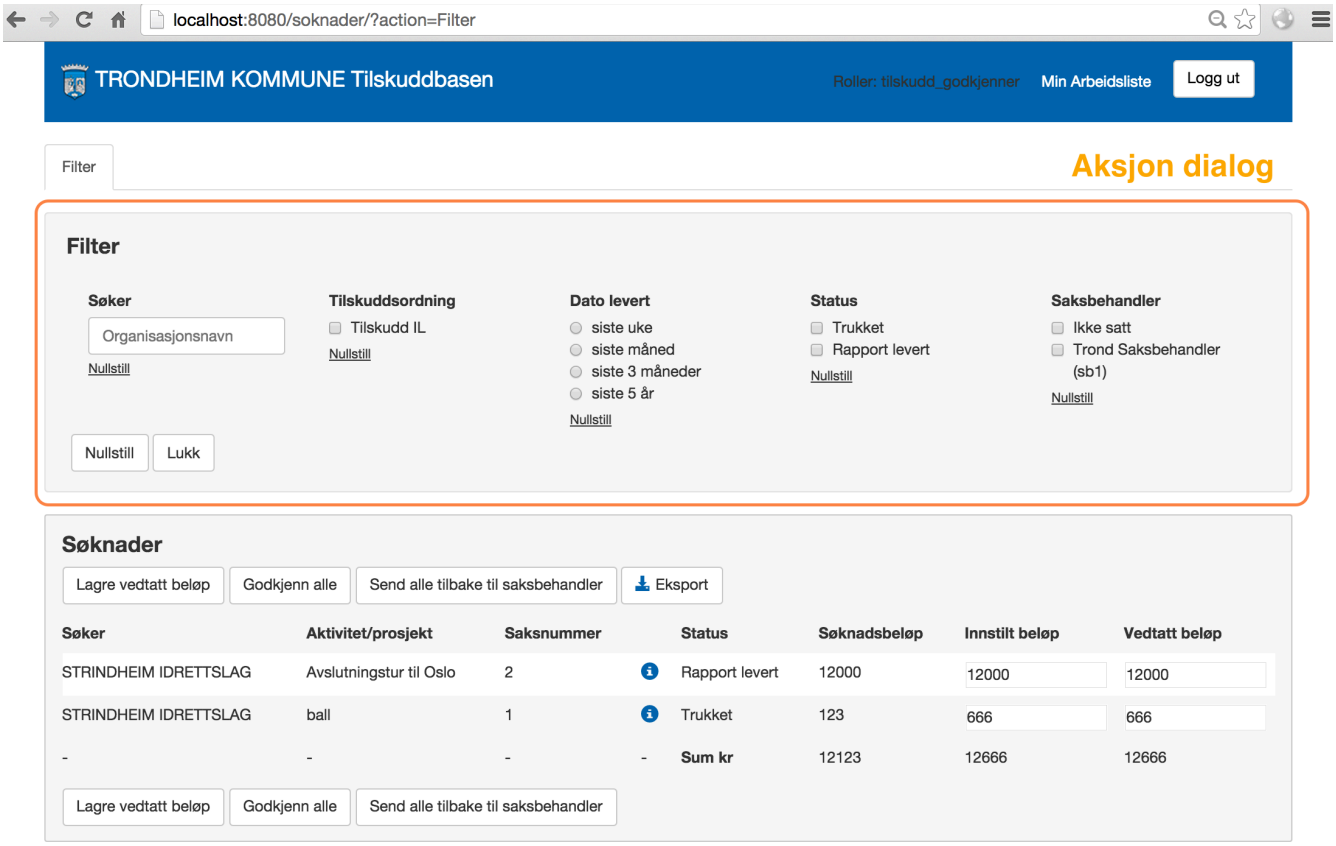
Siden det ble slik at man kuttet ut ting som "Varslinger", "Meldinger" og andre objekter som det hadde gitt mening å vise på landingsside da vi var i spesifiseringsfasen så ble det til slutt bare et element i meny, "Søknader", så vi har valgt å skjule hele menyen (var bare visuell støy).

Landingsside tilskuddsbasen



Aksjoner på landingsside

Aksjoner er også tilgjengelig på landingssiden, i det tilfelle kun filter (en søker hadde i tillegg kunne opprette "Ny Søknad"), og aksjonene ser ganske lik som de gjør andre steder (filter har ingen "Filtrer" knapp fordi man ønsker at hver filtervalg medfører til filtrering av listen uten å måtte be om det eksplisitt ved å trykke på en knapp).



Sikkerhet og integritet view

Sikkerhet

Autentisering

Autentisering av brukere er bygget på eksisterende teknologi: ADFS for interne brukere (Kommunens ansatte) og ID-porten for eksterne brukere (søkere). Det er med andre ord disse systemene som garanterer sikkerhet relatert til autentisering.

Autorisering

Autorisasjoner i systemet er rollebasert.

Hvilke rolle en intern bruker har (admin, saksbehandler og godkjenner) avgjøres av systemet basert på ADFS informasjonen. Dersom rollene endres i ADFS må brukeren logge inn på nytt for at ny tilgangsinformasjon skal kunne hentes fra ADFS.

Alle eksterne brukere har samme rolle (søker).

Ressursnøkler

Identifikasjon av personer

Aktørbasens rest API, flod_organisations (som benyttes av både Tilskuddsbasen og Bookingbasen), eksponerer unike id-er for personer. Id'ene er ikke personnummer, og alle personer, med eller uten personnummer, er garantert å ha en slik id. Id-en er unik og endrer seg aldri.

Integritet

Databaselåsing benyttes ikke, og heller ikke to-fase-commit på tvers av integrasjonspunktene.

Ved mange samtidige brukere som redigerer samme databaseentitet blir dataene lagret i den rekkefølge databasen prosesserer lagringskommandoene.

Backup

Sikkerhet mot tap og ødeleggelse av data håndteres på driftsnivå, med backup av hele maskinen ved jevne mellomrom.

Driftsavhengigheter

Systemet er avhengig av et fungerende internettilkobling og av at de integrasjonene som benyttes er oppe.