KODE OG UDVIKLINGSMILJØ

Dato: 02-02-2026

Status: Final

Version: 1.2

Formål

At give en samlet oversigt over den kode og de tekniske komponenter, der er udviklet i projektet – herunder hjælpeværktøjer, scripts, pre-processering og opsætning. Dokumentet skal understøtte genbrug, transparens og samarbejde med udviklermiljøer.

Indholdsfortegnelse

Formål 2
1. Introduktion 4
2. Udviklingsværktøjer, miljøstyring og kodehåndtering 5
2.1 Kodesprog samt centrale frameworks og biblioteker 5
2.1.1 Afprøvningsklienter 5
2.1.2 Valideringsklient 5
2.1.3 Data science 5
2.1.4 Database 6
2.2. Udviklingsmiljøer, afhængigheder og projektversionering 6
2.3. Deployment 6
3. Repositories 7
3.1 TALT-UI 7
3.2 TALT-Forretning 8
3.3 Sygeplejefaglig udredning 8
3.4 UC2 8
3.5 Speechcomponent 8
3.6 SharedComponents 8
3.7 test-server-deployment og prod-server-deployment 9
4. Opsætning og brug kode 10
4.1 TALT Afprøvningsklient 10
4.2 Valideringsklient 10
4.3. Data generering 10

1. Introduktion

Vi har i projektet arbejdet med 3 områder for værdiskabelse i sygeplejen, kaldet use cases, og har til hver af dem udviklet en løsning baseret på talegenkendelse, tekstklassificering og/eller generative tekst modeller. Løsningerne har det til fælles at de forsøger at afhjælpe byrden forbundet med at generere eller orientere sig i journaldata. Disse løsninger har vi gennem afprøvninger vist frem til sygeplejefagligt personale fra et bredt udsnit af kommuner.

Projektet har arbejdet med forskellige muligheder for værdiskabelse i syge- og hjemmeplejen samlet i 3 use cases (UC). Følgende er use cases overskrift, efterfulgt af en kort beskrivelse:

- **UC1: Sygeplejefaglig udredning**

Støtte til oprettelse af sygeplejetilstandsnotater på baggrund af en sygeplejefaglig udredningssamtale (SFU-samtale) mellem borger og sygeplejerske. Støtten ydes gennem udkast til notater baseret på samtalens indhold.

- **UC2: Opsummering af borgerjournal**

Opsummering af borgerjournal til støtte ved borgerbesøg udført af hhv. Sygeplejersker (SPL) og sundheds- og omsorgsassistenter og -hjælpere (SSA/SSH).

- **UC3: Indtaling af besøgsbeskrivelse**

Støtte til oprettelse og opdatering af besøgsbeskrivelser til hjælp med hjemmebesøg udført af SSA/SSH. Besøgsbeskrivelserne oprettelse på baggrund af en indtaling.

For mere information om arbejdet med at definere, afgrænse og afprøve projektets use cases se dokument “Use cases og afprøvning”

Dette dokument beskriver de vigtigste udviklingsværktøjer, frameworks og kodebiblioteker anvendt i projektet i forbindelse med den tekniske udvikling af løsninger. Der vil desuden blive præsenteret et overblik over, hvilke kodebaser der er oprettet i projektet, med overordnede beskrivelser af hvert enkelt samt beskrivelse af, hvordan koden opsættes og bruges.

For en mere detaljeret beskrivelse af mappe- og kodestrukturen og funktionaliteten for hvert kode repository, henviser vi til README-filerne i de forskellige kodebaser, der er udgivet på GitHub.

2. Udviklingsværktøjer, miljøstyring og kodehåndtering

I dette afsnit vil vi kort gennemgå de centrale kodesprog, frameworks og biblioteker anvendt i projektet.

2.1 Kodesprog samt centrale frameworks og biblioteker

2.1.1 Afprøvningsklienter

Til afprøvningsklienterne i projektet bruges Angular 17 som frontend framework til at rendere UI'en og styre app-logikken. Klienterne bygger derfor på Angular komponenter bygget i Typescript, med Node.js som miljø under runtime og npm som pakkemanager.

Backenden ligger I Talt-Forretning-repositoriet og består af et RESTful API udviklet i Python og bygget med FastAPI frameworket. Den bruger uvicorn som ASGI-web server og bearer token authentication, mens input- og output data bliver valideret og formateret med Pydantic objekter.

Klienterne er bygget med det formål at kunne foretage afprøvninger af AI funktionaliteten, men ikke til at kunne integrere ind i et eksisterende EOJ-system.

2.1.2 Valideringsklient

Frontenden i valideringsklienten er bygget med Dash-frameworket i Python. Den kører med Flask som webapplikation, uWSGI som applikationsserver og Nginx som reverse proxy med SSL-certifikat. Backenden i appen er, ligesom vores afprøvningsklient, vores API i TALT-Forretning, hvor der ligger en service lavet specifikt til at servicere valideringsklienten. Backenden bliver kaldt med HTTPX-biblioteket som HTTP-klient, mens navigationen mellem siderne bliver håndteret med 'Location'-komponenten i Dash Core Components-biblioteket. De indbyggede elementer i Dash-biblioteket er blevet kombineret med egne brugerdefinerede css-klasser mens også komponenter fra Dash Bootstrap Component biblioteket er blevet brugt til at få færdigt-designede komponenter heriblandt knapper, dialogbokse, spinnere og advarselsbeskeder.

2.1.3 Data science

Projektets data science-komponenter omfatter flere forskellige moduler udviklet i Python. Det gælder for talegenkendelses-, kategoriserings- og opsummeringskomponenten, at de alle bliver eksponeret som RESTful API'er bygget med FastAPI og Uvicorn som ASGI-webserver til understøtning af afprøvningssystemerne.

Talegenkendelseskomponenten anvender SYSTRANs re-implementering af OpenAI's Whisper-model, Faster Whisper, til transskribering af lyd, via Cloudflare Workers AI. Inden lyddata behandles af modellen, indlæses og gensamples de med Torchaudio for at sikre den korrekte frekvens-input til modellen. Selve transskriberingen sker

Klassifikationsmodulet er opbygget med SetFit-frameworket, som håndterer træning af modellen, mens SentenceTransformers anvendes til indlæsning af embedding-modeller og står for repræsentationen af tekstdata. Pre-processering foretages med Pandas, og Pydantic bruges til at validere og strukturere inputdata. Evalueringen af modellerne udføres med Scikit-learns metrics-modul.

Opsummeringskomponenten benytter Large Language Models deployeret i Microsoft Azure, hvor Azure AI-bibliotekerne anvendes til at generere tekstopsummeringer via Ollama-modellen og til at håndtere autentifikation mod Azure-servicen.

Den syntetiske datagenerering bygger på et system af prompt-funktioner, som kombineres til fleksible prompt-metoder. Nogle funktioner henter eksempeldata fra databasen via SQLAlchemy, mens samtaler og tekstgenerering udføres med OpenAI-modeller hostet i Azure, der indlæses med AzureOpenAI-pakken fra OpenAI-modulet.

Projektets UC2 indeholder en journal-parser, der konverterer journaldata fra Excel til JSON. Denne komponent benytter Pandas til dataindlæsning og -manipulation samt regular expressions (re-modulet) til at udtrække specifikke informationer fra tekstfelterne.

2.1.4 Database

Projektet bruger relationelle PostgreSQL-databaser til struktureret at organisere og håndtere data i projektet. Psycopg2-biblioteket bruges til at oprette tabeller, køre migreringer og sql-kommandoer og håndtere fejl. Med få undtagelser, er også SQLAlchemy anvendt til hurtig indlæsning af data til de eksperimentelle data science komponenter, der ikke bruges i produktion.

2.2. Udviklingsmiljøer, afhængigheder og projektversionering

Projektet benytter Poetry til at håndtere udviklingsmiljøer, afhængigheder og projektversionering. Poetry sikrer et ensartet setup på tværs af udviklere ved automatisk at oprette virtuelle miljøer, styre pakkernes versioner og holder repositoriernes version opdateret via pyproject.toml og poetry.lock. UC3-komponenten administreres i øjeblikket via et manuelt venv uden pyproject.toml; dokumentér derfor, hvordan miljøet aktiveres, inden man kører uvicorn.

2.3. Deployment

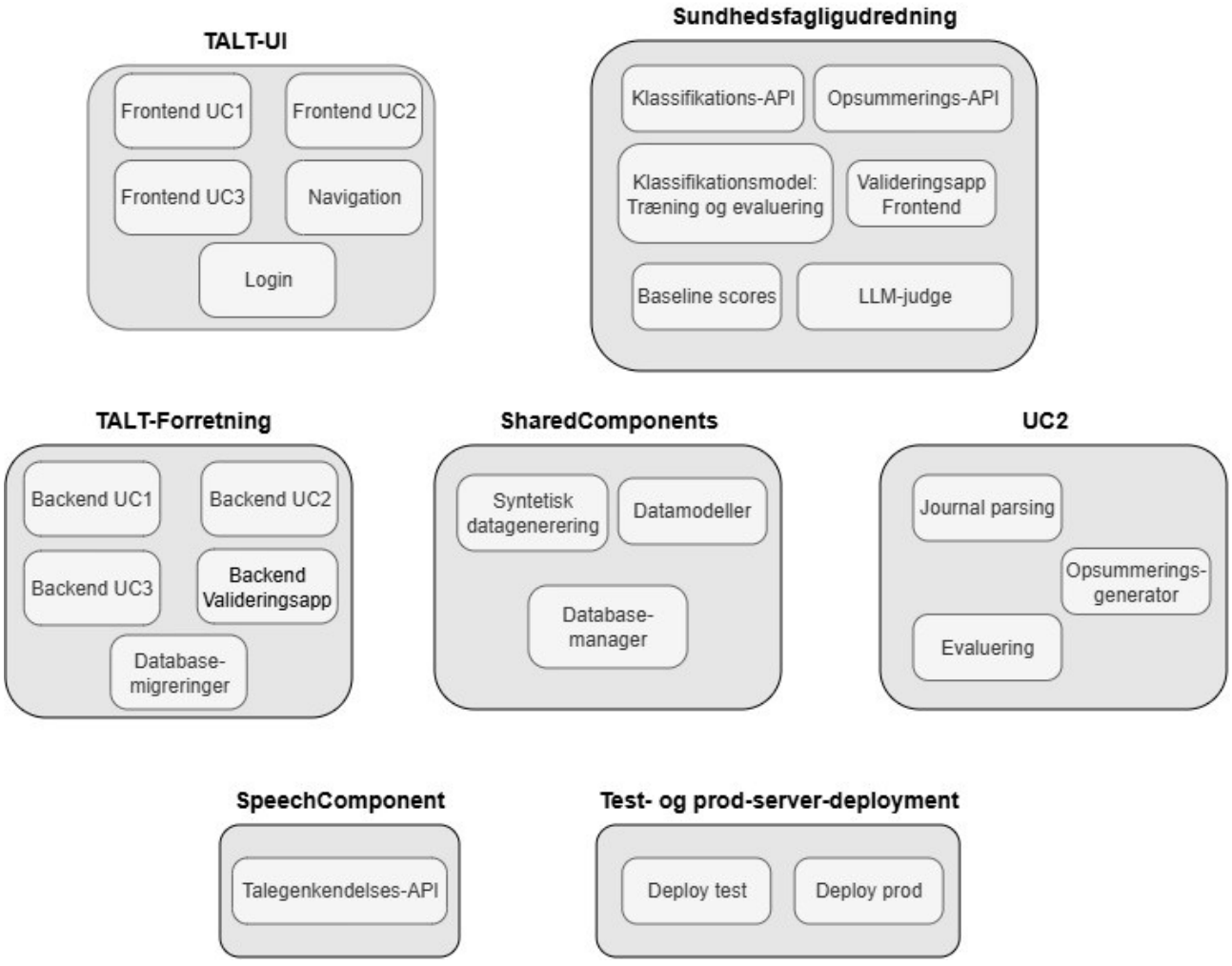
Projektet bruger Docker til containerisering af applikationerne og deres afhængigheder. Færdige Docker-images bygges og skubbes til et Sonatype Nexus Registry, hvor de versioneres. Til deployment anvendes Docker Compose, som henter de versionerede images fra Nexus og starter de valgte services på serveren.

Deployment håndteres via de Docker Compose-filer, der følger med i service-repositorierne (fx talt-forretning/Docker/docker-compose.yml, talt-ui/docker-compose.yml og speechcomponent/docker/docker-compose.smoketest.yml). Når nye versioner bygges og pushes til Nexus, opdateres tags direkte i disse filer, og stakken startes lokalt eller på serveren uden brug af særskilte test-/prod-repositorier.

3. Repositories

Kodebasen er opgjort af en række forskellige repositorier, se figur 1 for et overblik. I dette afsnit vil vi gennemgå hvert repositorie og kort beskrive formålet med dets kode.

Figur 1: Oversigt over projektets repositorier



Hvert repositorie understøtter forskellige behov. De ydre kasser indikerer repositorier og de indre kasser indikerer komponenter.

3.1 TALT-UI

Repositoriet indeholder alle komponenter og services tilhørende frontenden af vores afprøvningssystem. Det er en samlet frontend applikation, men med en mappestruktur, hvor komponenterne tilhørende hver use-case er separeret i egne mapper, ligesom navigationskomponenten og loginkomponenten har fået egne mapper.

3.2 TALT-Forretning

TALT-Forretning er projektets forretningslag, der står for at håndtere kommunikationen til og fra TALT-forretnings database samt opsætning og opdatering af databasen. Det er et RESTAPI, hvor endpoint ruterne er opdelt i separate filer efter funktionalitet mens services, funktioner, datamodeller og dataadgangslag er samlet i undermapper ud fra komponenten de understøtter; hhv. UC1, UC2, UC3 og valideringsappen.

3.3 Sygeplejefaglig udredning

Her er samlet komponenter, som primært er knyttet til den sygeplejefaglig udredning (UC1). Der er i repositoriet placeret følgende komponenter med tilhørende beskrivelse:

- **Kategoriseringskomponenten**

Kategoriseringskomponenten består af klassifikations-API'et, og udviklingskoden, der står for at træne og evaluere kategoriseringsmodellerne.

- **Opsummeringskomponenten (forretningslag)**

Her er API'et til opsummering placeret sammen med alle nødvendige funktionaliteter for at kunne generere opsummeringer under afprøvning, heriblandt modelprompts.

- **Valideringsapp**

Frontenden til valideringsappen.

- **Baseline**

Evaluering af komponenterne for talegenkendelse, kategorisering og opsummering under test og afprøvning.

- **LLM-judge**

Indeholder LLM-judge brugt til at evaluere kvaliteten af UC1-opsummeringer.

3.4 UC2

UC2-repositoriet indeholder koden til klargøring af data og genereringen af opsummeringer til afprøvning og evalueringen af afprøvningsresultaterne. Der er derfor placeret journal-parsing og databasekode til at indlæse journaler fra Excel til databasen og selvstændige notebooks til generering af opsummeringer og evalueringen af afprøvningen.

3.5 Speechcomponent

Indeholder Fast-API'et, der kan transskribere samtaler og tilhørende services, der understøtter processeringen og transskriberingen af lydfilerne.

3.6 SharedComponents

SharedComponents er stedet hvor moduler, der bruges på tværs af respositorier, er placeret. Det drejer sig blandt andet om databasemanageren og datamodeller. Databasemanageren er et

postgreSQL-databasemodul, der sørger for connection, fetching af data og eksekvering af statements. Datamodellerne består af Pydantic dataklasser brugt generelt i data science-komponenterne.

Modulet til generering af syntetisk data er også placeret her, og er mappeopdelt efter typen af datagenerering.

3.7 test-server-deployment og prod-server-deployment

To repositorier, der hver indeholder en docker-compose-fil til at køre afprøvningsklienten og valideringsappen, og køres på hhv test- og prod-serveren. De tidligere test-server-deployment og prod-server-deployment repos er udfaset. Deployments udføres nu via de Docker Compose-filer, som ligger i de enkelte kodebaser (fx talt-ui/docker-compose.yml, talt-forretning/Docker/docker-compose.yml, sundhedsfagligudredning/validation_app/docker/docker-compose.smoketest.yml).

4. Opsætning og brug kode

4.1 TALT Afprøvningsklient

Appen køres lokalt ved at bygge og køre Docker-billederne for talt-ui, talt-forretning og databasen. Dockerbillederne kan køres samlet i et kald ved at køre docker-compose-filen på talt-forretning.

For deployment til test-/prod-serveren, bygges og pushes Dockerbillederne med det rette tag (der hentes fra versionscontrolleren i pyproject.toml på de respektive repositories), som herefter zippes og pushes til Nexus-serveren. Tags skal herefter opdateres i docker-compose filen til de nye versionstag, hvorefter der logges ind på test-/prod-serveren via SSH. Herfra hentes docker-billederne ned fra Nexus og køres med docker compose-filen lavet til det pågældende miljø. Der er lavet separate docker-compose filer til hhv. test og produktion, så der kan testes i testmiljøet før deploy til produktion, og vi kan spore os tilbage til tidligere versioner, hvis der opstår fejl.

TALT-afprøvningsklienten køres lokalt ved at bygge og køre dets Dockerbillede. Det køres via Angular CLI, installer afhængighederne med npm install, starter udviklingsserveren med npm start, og bruger npm run json-server hvis der er behov for mock-API-data. Docker/Compose-scripts kan fortsat bruges til integrerede miljøer, men er ikke påkrævet for lokal udvikling.

4.2 Valideringsklient

Valideringsklienten køres lokalt ved at bygge og køre dets Dockerbillede. Det deployes på test-/prod-serveren efter samme fremgangsmåde som afprøvningsklienten. Valideringsklienten (sundhedsfagligudredning/validation_app) startes lokalt via poetry install efterfulgt af make build start, eller via ./scripts/00-build-local.sh der bygger og kører Docker-containere. App'en kalder talt-forretning-API'et gennem httpx i src/app/utils/api_service.py.

4.3. Data generering

Datagenereringen er lavet som konfigurerbare Python-scripts, der ud fra konfigurationerne i config.toml, genererer det ønskede data og skriver det til databasen.

Datagenereringen styres af scripts i sharedcomponents/generering_af_syntetiske_samtaler. Prompt-metoder registreres via PROMPT_METHODS i prompt_methods.py, og hver metode kan vælges i config.toml. Samtaler genereres ved hjælp af Azure OpenAI (AzureOpenAI-klienten) med health condition-data hentet via DataManager.

Genereringerne bygger på promtps, der ligger samlet i en Python-klasse, og kombineret til forskellige prompt-metoder i en tilsvarende python-klasse. Disse metoder bliver registreret i et dictionary med en decorators, så de er tilgængelige under datagenereringen. Fra ens config-fil kan man derfra vælge den ønskede prompt-metode, men også andre konfigurationer skal vælges som fx længden af en ønsket syntetisk samtale og antallet af helbredstilstande. Når ens konfigurationer er valgt, kan Python-filen der genererer og skriver til databasen eksekveres.