Project Structure

Backend

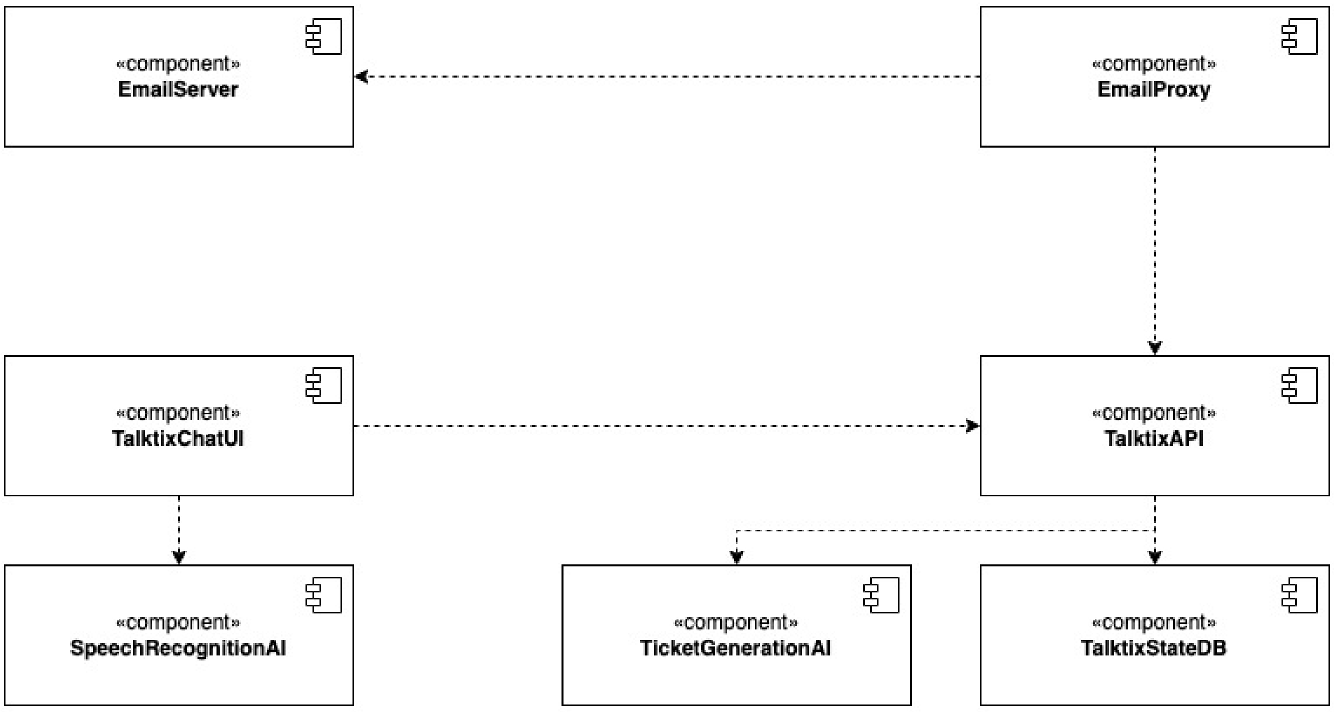
> - **app**
>
> > - **api**
> >
> > > - **v1** (API endpoints)
> > > - **dto** (data transfer objects, exchanged via API)
> >
> > - **model** (machine learning part)
> > - **email** (email proxy app)
> >
> > > - **main.py** (start EmailProxy)
> >
> > - **service** (logic layer)
> > - **repository** (persistence layer)
> >
> > > - **entity** (objects stored in the MongoDB)
> >
> > - **dependency** (FastAPI dependency injection functions)
> > - **enum** (global enums for entities and DTOs)
> > - **util** (useful functions)
> > - **main.py** (start API)
>
> - **test** (same structure as under app)

Frontend

> - **app**
>
> > - **service** (Angular services that support the components, e.g. http requests to the backend)
> > - **component** (Angular components)
>
> - **assets**
> - **environments**
> - **index.html**
> - **main.ts** (main app script)
> - **styles.css** (global styling)

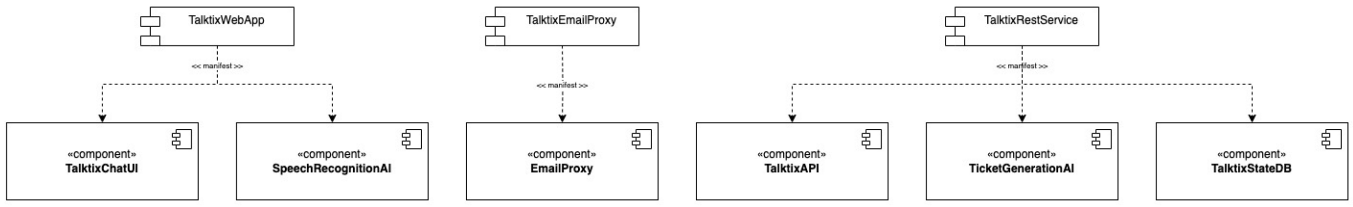
Software Architecture

Building Blocks View


«component» EmailServer
«component» EmailProxy
«component» TalktixChatUI
«component» TalktixAPI
«component» SpeechRecognitionAI
«component» TicketGenerationAI
«component» TalktixStateDB

- **TalktixChatUI**: presents the chat to the user
- **SpeechRecognitionAI**: translates audio input into text
- **TalktixAPI**: defines endpoints of the backend
- **TicketGenerationAI**: transforms the chat messages into a ticket
- **TalktixStateDB**: stores the tickets, users, categories, departments, services and locations in the MongoDB
- **EmailServer**: manages email communication (dev stage: Outlook email server)
- **EmailProxy**: opens a chat for received emails and replies via email

Realization View



Technology Stack

Frontend

- TypeScript
- Angular
- Angular Material
- TailwindCSS
- Karma

Backend

- Python

- FastAPI (OpenAPI & Swagger)
- PyMongo (MongoDB Community Server)
- Transformers by HuggingFace (PyTorch, NumPy)
- PyTest

Architecture Decisions

Microservices

The Microservice architecture of the backend has a low coupling between the single services and thus can be easily deployed and is highly reusable and able to experiment.

AI Ensemble

Initially, we experimented with generative models such as T5, which did not perform well enough with our limited data set due to their enormous complexity and size. Consequently, we switched to 8 separate models (some classifiers based on Bert/RoBerta) for the ticket fields "title", "affectedPerson", "keywords", "requestType", "category", "service", "customerPriority" and "priority".

Database

Chat and ticket data must be stored in a persistent database. We chose MongoDB, a document-based database which stores the data in an object-oriented way, because it's simple to maintain, easy to use and has a higher performance for simple CRUD operations than SQL databases.

Frameworks and Libraries

On the one hand, we decided to use Angular based on TypeScript as our framework for the frontend, because it is structured, modular, fast and delivers in-house solutions for common tasks. On the other hand, we chose FastAPI and PyTorch based on Python as our two frameworks for the implementation of the backend containing multiple APIs, data storage and artificial intelligence (Ai).