# Subletic arc42-Architekturüberblick

## Inhaltsverzeichnis

# 1. Einführung und Ziele

Dieser Abschnitt führt in die Aufgabenstellung ein und skizziert die Ziele, die Subletic verfolgt.

## 1.1. Aufgabenstellung

Das Ziel unserer Software ist die teil-automatisierte Transkriptionen für Untertitel in Live-Kontexten, mithilfe von modernen KI-Modellen. Diese sollen dann von geschultem Personal, wie auch Laien zusätzlich überprüft und wenn notwendig korrigiert werden. Die nachträgliche manuelle Korrektur wird besonders bei Kunden notwendig die einen besonderen Wert auf die Richtigkeit der Untertitel legen, wie zum Beispiel bei der Untertitlung von Nachrichtensendungen oder in politischen Kontexten. Mit der Verschmelzung von fortschrittlicher KI und dem Menschen als Korrektur-Instanz, soll eine hohe Qualität der Untertitel gewährleistet werden, bei gleichzeitig minimalem Aufwand für den Menschen.

### 1.1.1. Features

- **Automatische Transkription** von Sprache zu Text mithilfe eines Services, der durch einen WebSocket aufgerufen wird
- **Web-Editor** für die Korrektur von Untertiteln
- **Audio-Player** zum Abspielen des Audio-Streams und Navigieren in diesem
- Verwendbarkeit von **Stenografen-Equipment**, wie Fuß- und Hand-Schalter
- **Soundslike-Dictionary** zum beschleunigen der Korrektur

## 1.2. Qualitätsziele

| Qualitätsziel | Motivation und Erläuterung |
|---|---|
| Benutzerfreundliche Oberfläche | Die Oberfläche für die Korrektur soll schlicht, modern und intuitiv bedienbar sein, um dem Benutzer die bestmögliche Orientierung zu bieten und das schnellst mögliche Korrigieren zu ermöglichen. |
| Schnelle Antwortzeiten | Die Antwortzeiten des Systems sollen möglichst gering sein, damit das System in *Echtzeit-Systemen* integrierbar ist. |
| DSGVO-Konformität | Die Software soll die DSGVO einhalten, damit die Software in Europa betrieben werden darf. |
| Codequalität | Da das Entwicklungsteam aus unerfahrenen Entwicklern besteht, soll die Codequalität durch die Verwendung verschiedener Qualitätssicherungs-Systeme und Prozesse sichergestellt werden. |

## 1.3. Stakeholder

| Stakeholder | Verantwortlichkeiten |
|---|---|
| **Software Architekt**<br>Benedikt Beigang | • Hat die Softwarearchitektur im Blick und steht für technische Fragestellungen als erstes zur Verfügung<br>• Hilft beim Entwurf der Schnittstellenbeschreibung, sowie der Komponenten der Software<br>• Hilft komplizierte Merge-Requests aufzulösen, Git-Problemen, und technischen Details<br>• Kümmert sich um Deployment bzw. Auslieferung unserer Software<br>• Erstellt Codereviews für die Entwickler mit der MoSCoW-Methode<br>• Behält die CI/CD-Pipeline und deren Einhaltung im Blick<br>• Verantwortlich für Durchsetzung eines Styleguides |
| **Product Owner**<br>Luca Noack | • Hält Kontakt mit dem Kunden<br>• Vertritt den Kunden im Team und ist bei Fragen an den Kunden erster Ansprechpartner<br>• Befüllt und priorisiert das Product-Backlog<br>• Grobe Voraus-Planung der Sprints und deren Ziele (Genauer Planung erfolgt durch Team)<br>• Führen der Projektleitdokumentation<br>• Leiten der Kunden Meetings<br>• Schreibt und verwaltet die Dokumentation, samt Issues (Board), Projektleitdokumentation, Protokolle, Wiki, Übersichten, etc. |

| Stakeholder | Verantwortlichkeiten |
| --- | --- |
| **Team-Manager**<br>Benedikt Beigang<br>Luca Noack | • Hat die Stimmung im Team im Blick und räumt Schwierigkeiten der Entwickler aus dem Weg (Team-Coach)<br>• Gewährleistung der Einhaltung des agilen Manifestes (Timeboxing, SCRUM, etc.)<br>• Ansprechpartner bei allen Fragen die nicht technische Details oder Fragen an den Kunden sind<br>• Vorbereiten und Leiten der Retrospektive um Team weiterzuentwickeln<br>• Leiten aller Meetings außer Kundenmeeting<br>• Projekt-Planung und Organisation<br>• Verantwortlich für schwerwiegende Projektentscheidungen<br>• Hat Veto-Recht<br>• Hat die Einhaltung der Anforderungen im Blick |
| **Developer**<br>Chantal Bley<br>Pascal Fabian<br>Amine Jegani<br>Christoph Neidahl<br>Luca Niklas<br>Finn Romeis | • Teilnahme an offline/online-*Team-Meetings*<br>• Programmiertechnische Umsetzung, infolge von *Tickets/Issues*<br>• *Selbstständige* Entwicklungsarbeit und Aufgabenzuteilung innerhalb des Team<br>• *Informieren des Teams* zu Projektfortschritt, bei Problemen und Entscheidungen die getroffen werden müssen in Issue-Kommentaren, Discord oder im Weekly<br>• Einhalten der vom Team festgelegten *Definition of Done's*<br>• Präsentieren der entwickelten Features spätestens im *Review*<br>• *Dokumentieren* des geschriebenen Codes<br>• Schreiben von *Unit-Tests* zu den implementierten Funktionalitäten<br>• Erstellung von *Protokollen*<br>• Schreiben von Code-Reviews falls ein anderer Developer dies wünscht |
| **Philipp Platis** | • Mitarbeiter der gbs und Ansprechpartner für Fragen bezüglich der Anforderungen |

| Stakeholder | Verantwortlichkeiten |
| --- | --- |
| **Grundig Business GmbH & Co. KG (gbs)** | • Kunde (Unternehmen) der unsere Software nutzen und nach Abschluss des Projekts weiterentwickeln möchte<br>• Spezialisiert auf die Entwicklung von KI-gestützter Stenografie-Hardware und Software |
| **Karsten Weicker** | • Zuständiger Professor für das Modul<br>• Steht bei Fragen bezüglich der Organisation des Moduls und dessen Abgaben und Prüfungsleistungen zur Verfügung |

# 2. Randbedingungen

Beim Lösungsentwurf waren zu Beginn verschiedene Randbedingungen zu beachten, sie wirken in der Lösung fort. Dieser Abschnitt stellt sie dar und erklärt auch – wo nötig – deren Motivation.

## 2.1. Technisch

| Randbedingung | Erläuterungen, Hintergrund |
|---|---|
| Moderate Hardwareausstattung | Betrieb der Lösung auf einem marktüblichen Standard-Notebook, damit sie von Profis wie Laien genutzt werden kann. |
| Betriebssystemunabhängigkeit für korrigierende Person | Die korrigierende Person soll die Lösung auf ihrem eigenen Rechner betreiben können, unabhängig vom Betriebssystem. |
| Docker | Die Lösung soll in einem Docker-Container laufen, um die Installation und den Betrieb zu vereinfachen. |
| Chromium-Browser und externe Geräte | Fuß- und Handschalter sind auf Chromium-Browsern angewiesen. Sollen diese also genutzt werden, ist die Software nur mit Browsern wie zum Beispiel Chrome oder Edge möglich. |
| Angular und ASP.NET | Der Kunde entwickelt bereits mit Angular und ASP.NET, sodass die Lösung in diesen Technologien entwickelt werden soll. Außerdem sind diese Technologien sehr gut dokumentiert und es existieren viele Tutorials, Bibliotheken und Beispiele. |

## 2.2. Organisatorisch

| Randbedingung | Erläuterungen, Hintergrund |
|---|---|
| Zeitplan | Die Lösung soll in einem Zeitraum von zwei Semestern entwickelt werden. Effektiv beginnt die Entwicklungszeit Anfang Mai 2023 und endet Anfang Februar 2024, unterbrochen von den Semesterferien. |
| Vorgehensmodell | Grundlegend wird nach dem agilen Manifest gearbeitet, wobei das Scrum-Framework als Leitplanken dient. |

| Randbedingung | Erläuterungen, Hintergrund |
|---|---|
| Entwurfswerkzeuge | Entwürfe entstehen zunächst auf als einfache Zeichnungen auf Papier, Tafel oder Tablet. Später werden ausgearbeitete Wireframes welche mit Figma erstellt wurden dem Kunden präsentiert und mit ihm besprochen. |
| Konfigurations- und Versionsverwaltung | GitLab mit CI/CD Pipeline für die Versionsverwaltung und das automatisierte Testen und Bauen der Software. |
| Testwerkzeuge und -prozesse | Es wird nUnit und Jasmine für die Unittests verwendet. |
| Dokumentationswerkzeuge | Die Dokumentation befindet sich größtenteils in unserem Wiki der GitLab-Gruppe. Für versionierte Dokumentation existiert ein zusätzliches *Documentation*-GitLab-Repository. |
| Kommunikationswerkzeuge | Für die Kommunikation innerhalb des Teams nutzen wir Discord, unsere Meetings in der Hochschule oder die Kommentar-Bereiche im GitLab. Die Kommunikation mit dem Kunden erfolgt auf Microsoft Teams. |
| Veröffentlichung als Open Source | Es wird auf keine Technologien oder Bibliotheken zurückgegriffen, die nicht Open Source sind. Die Software wird deshalb nach Beendigung des Projekts unter einer Open-Source Lizenz veröffentlicht. |

## 2.3. Konventionen

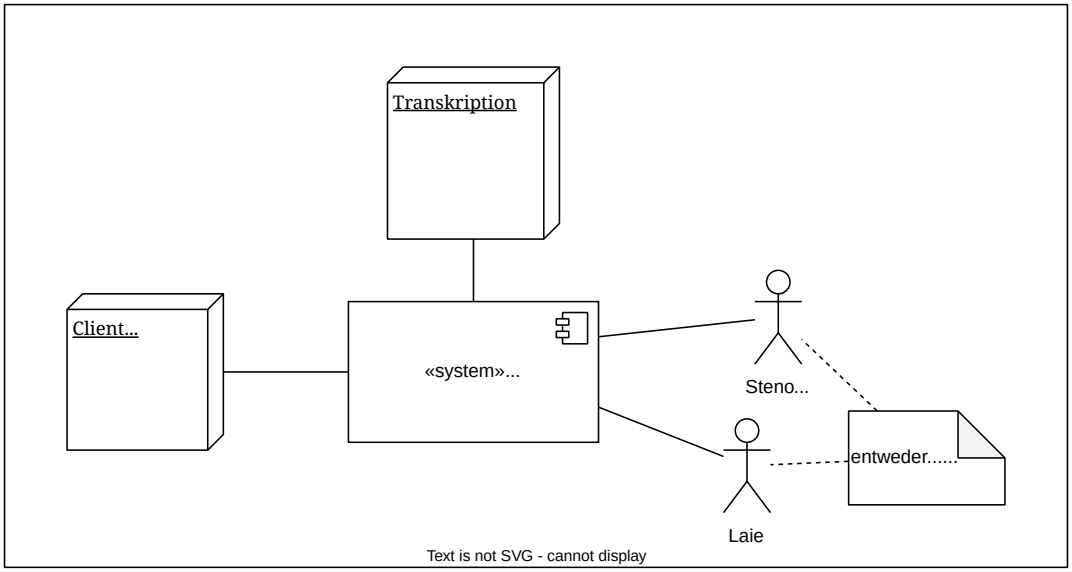
| Konvention | Erläuterungen, Hintergrund |
|---|---|
| Architekturdokumentation | Terminologie und Gliederung größtenteils nach dem arc42-Beispiel DokChess. |
| Code-Style Backend | Es wird der Standard C#-Code-Style verwendet, sowie getestet mit StyleCopAnalyzers, den Roslyn Analyzers und einer `.editorconfig`. Außerdem wird Inline-Kommentierung (`///` statt `/**/`) verwendet. |
| Code-Style Frontend | Es wird der Standard TypeScript-Code-Style verwendet. |
| Sprache | Die Software, sowie dessen Kommentare werden in englischer Sprache geschrieben. Die Dokumentation im GitLab wird in deutscher Sprache verfasst. |

| Konvention | Erläuterungen, Hintergrund |
| --- | --- |
| Code-Reviews | Die Code-Reviews werden nach der MoSCoW-Priorisierung durchgeführt. |
| Branches | Es wird nach dem Feature-Branch-Workflow gearbeitet. Neben dem `main`- und `dev`-Branch dürfen nur `feature`- und `bug`-Branches erstellt werden, welche nur von `dev` abzweigen dürfen. Ausnahme ist hier der `hotfix`-Branch, welcher von `main` abzweigen darf. Der Name des Branches muss folgendem R2-Regex entsprechen: `main\|dev\|((feature\|bug\|hotfix)\/{3,})` |
| Commit-Name | Die Commits müssen folgendem R2-Regex entsprechen: `(^#\d+\s+(feat\|docs\|fix\|test):\s+[[:ascii:]]{4,})\|(Merge (.)*)` |
| Email-Adresse | Die Email-Adressen mit denen commited wird, müssen folgendem R2-Regex entsprechen: `[[:ascii:]]+(@stud.htwk-leipzig.de){1}` |

# 3. Kontextabgrenzung

Dieser Abschnitt beschreibt das Umfeld von Subletic. Für welche Benutzer ist es da, und mit welchen Fremdsystemen interagiert es?

## 3.1. Fachlicher Kontext




### 3.1.1. Stenograph*in (Benutzer*in)

Das Korrigieren der generierten Transkriptionen erfolgt unter anderem durch Stenograph*innen. Diese sind durch Ihre besondere Schulung an besonderes Equipment gewöhnt, welches unterstützt werden müssen muss.

### 3.1.2. Laie (Benutzer*in)

Das Korrigieren der generierten Transkription erfolgt unter anderem durch Laien. Diese benötigen im Gegensatz zu Stenograph*innen eine besonders einfache Bedienung und Hilfen, sodass Sie in einem Live-Kontext mithalten können. Auch die Verwendung *Laien-nahen* Dateiformaten ist wichtig, wie CSV, XLS oder XLSX, sodass keine zusätzlichen Hürden entstehen.

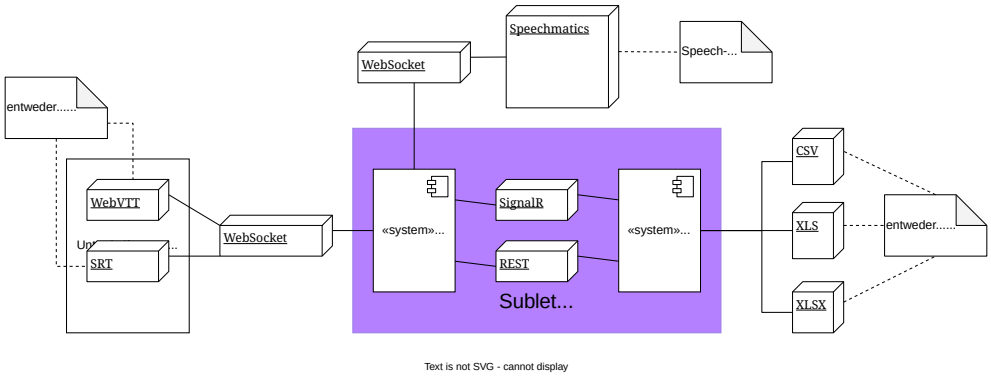
### 3.1.3. Externer Videostream (Fremdsystem)

Ein Videostream ohne Untertitlung kann durch das Nutzen unseres Services, schnell und einfach mit Untertiteln versehen werden. Dazu werden einfache/gebräuchliche Schnittstellen benötigen, sodass die Einbindung in unser System einfach möglich ist.

### 3.1.4. Soundslike-Dictionary (Fremdsystem)

Bestimmte Namen und Begriffe sind von KI-Modellen schwer zu erkennen. Diese werden in einem *Soundslike-Dictionary* gespeichert und können zum Start der Software bereitgestellt werden.

# 3.2. Technischer Kontext




## 3.2.1. Angular (Fremdsystem)

Für die Entwicklung der Korrektur-Oberfläche, wird das Framework *Angular* verwendet. Dieses Framework ist weit verbreitet und bietet deshalb genügend Dokumentation und Tutorials, sodass ein schneller Einstieg in die Technologie möglich ist. Da Angular Teil des Node.js Frameworks ist, kann auf viele JavaScript- und TypeScript-Bibliotheken zurückgegriffen werden, welche die Entwicklung vereinfachen.

## 3.2.2. ASP.NET Core (Fremdsystem)

Für die Entwicklung des Backends, welcher als Schnittstelle zwischen der Korrektur-Oberfläche und den Fremdsystemen dient, wird das Framework *ASP.NET Core* verwendet. Dieses Framework ist weit verbreitet und bietet deshalb genügend Dokumentation und Tutorials, sodass ein schneller Einstieg in die Technologie möglich ist. Außerdem bieten die zahlreichen nuGet-Packages eine einfache Möglichkeit, um auf Bibliotheken zurückzugreifen, welche die Entwicklung vereinfachen.

## 3.2.3. REST (Fremdsystem)

Um eine einfache Kommunikation zwischen der Korrektur-Oberfläche zum Backend zu ermöglichen, wird das *REST*-Paradigma verwendet. Es handelt sich um ein weit verbreitetes Paradigma und stellt in ASP.NET den quasi-Standard dar.

## 3.2.4. SignalR (Fremdsystem)

Um eine Echtzeit-Kommunikation zwischen der Korrektur-Oberfläche und dem Backend zu ermöglichen, wird das nuGet-Package *SignalR* verwendet. Somit lassen sich Daten vom Backend aus, in das Frontend streamen. Mit *SignalR* werden Fallstricke wie die Skalierbarkeit umgangen, die bei klassischen Ansätzen (zB WebSockets) auftreten.

## 3.2.5. WebSocket (Fremdsystem)

Damit unser Service nach außen möglichst breit genutzt werden kann, wird das *WebSocket* -Protokoll verwendet. Dieses Protokoll ist weit verbreitet und wird von vielen Systemen unterstützt. Es ermöglicht eine bidirektionale Kommunikation zwischen Client und Server, welche für die Echtzeit-Kommunikation benötigt wird.

### 3.2.6. Speechmatics (Fremdsystem)

Um eine KI-gestützte Transkription zu ermöglichen, wird die das Tool *Speechmatics* verwendet. Über einen WebSocket wird die Audiospur an *Speechmatics* gesendet, welche dann die Transkription zurücksendet. Zusätzlich kann beim Verbindungsaufbau ein *Soundslike-Dictionary* übergeben werden, welches die Transkription verbessern kann.

### 3.2.7. Untertitelformat WebVTT/vtt und SubRip/srt (Fremdsystem)

Unterschiedliche Videostream-Systeme benötigen, abhängig von Anwendungsfall unterschiedliche Untertitelformate. Die zwei gängigsten Formate sind *WebVTT* und *SRT*. *WebVTT* ist das modernere Format, welche zum Beispiel Features wie Textformatierung unterstützt. *SRT* ist das ältere Format, welches von vielen Systemen noch verwendet wird. Beide Formate müssen von unserem System unterstützt werden.

### 3.2.8. CSV/XLS/XLSX (Fremdsystem)

Um das *Soundslike-Dictionary* Importieren und Exportieren zu können, werden gängige Tabellenformate benötigt. Die verbreiteten sind *CSV*, *XLS* und *XLSX*. Diese Formate müssen von unserem System unterstützt werden.

# 4. Lösungsstrategie

Dieser Abschnitt enthält einen stark verdichteten Architekturüberblick. Eine Gegenüberstellung der wichtigsten Ziele und Lösungsansätze.

## 4.1. Einstieg

| Qualitätsziel | Dem zuträgliche Ansätze in der Architektur |
|---|---|
| Benutzerfreundliche Oberfläche | • Nutzen von schon existierenden UI-Komponenten<br>• Im Blick halten des Mental-Loads, schon während des Designprozesses<br>• Anfertigung von Wireframes/Mockups<br>• Orientierung an der 60:30:10 Faustregel |
| Schnelle Antwortzeiten | • Verwendung von Echtzeit-Technologien wie zum Beispiel WebSockets<br>• Beschränkung auf Audio-Stream, da Videostream, sowieso nicht genutzt wird<br>• Ständige End2End-Tests durch die Entwickler |
| DSGVO-Konformität | • Verwendung von frei verfügbaren Bibliotheken<br>• Kein Deployment außerhalb der EU<br>• Keine Verwendung von Cookies oder Speichern anderer Daten |
| Codequalität | • Code-Styleguides: C#-Code-Style<br>• Analyse-Tools: StyleCopAnalyzers, Roslyn Analyzers, IDE mit Hilfe einer `.editorconfig`<br>• Unit-Tests mit nUnit und Jasmine<br>• CI/CD-Pipeline mit Linting und automatisiertem Tests |

## 4.2. Aufbau

## 4.3. Anbindung

# 5. Bausteinsicht

## 5.1. Kontextabgrenzung

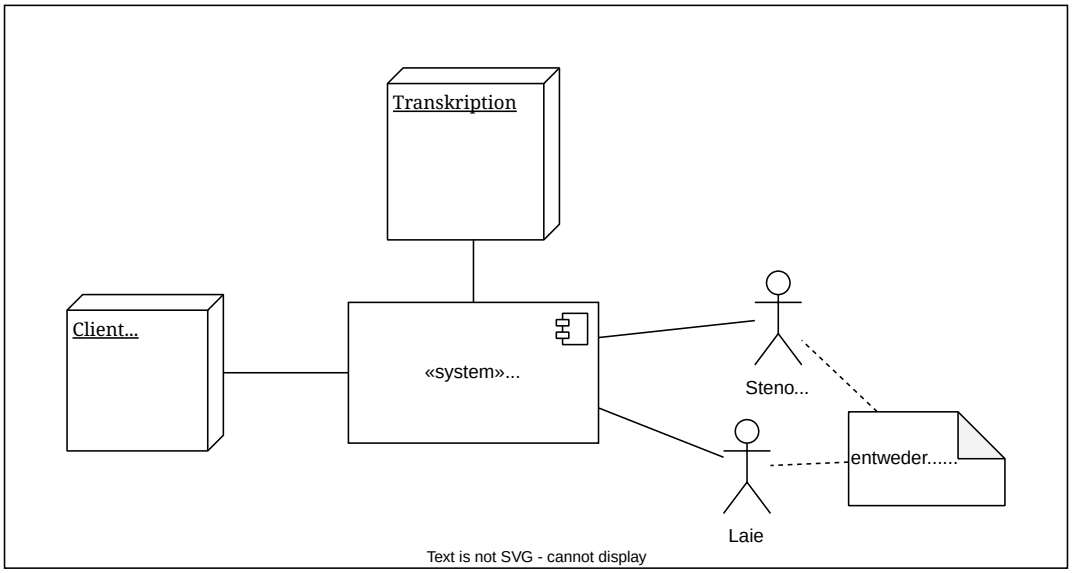



*Abbildung 1. Kontextabgrenzung von Subletic zu anderen Systemen*

Subletic interagiert wie im Kapitel Kontextabgrenzung beschrieben, mit zwei Verschiedenen Fremdsystemen und einem Nutzer:

1. Der Client möchte für einen Videostream eine Untertitelung erhalten und sendet dazu einen Audiostream an Subletic.
2. Eine Speech-Engine wird benötigt, welche den Audiostream in Text umwandelt.
3. Es wird eine Schnittstelle zur korrigierenden Person benötigt.

Für die Interaktion mit den beiden Fremdsystemen wird ein Backend benötigt, welches die Kommunikation mit diesen übernimmt. Für die Interaktion mit der korrigierenden Person wird ein Frontend benötigt, welches die Schnittstelle zum Nutzer darstellt. Das Frontend soll dem Nutzer die Möglichkeit geben, die Transkription zu korrigeren, den Audiostream zu hören und Unterstützt die Korrektur auf verschiedene Arten.

## 5.2. Ebene 1 - Komponenten-Sicht

Mit der Komponenten-Sicht werden die Teilbereiche beider Systeme transparent gemacht. Neben den API-Bereichen die für die Kommunikation mit den Fremdsystemen benötigt werden, tauchen die Domänen: Audio, SpeechBubble und Configuration in Backend und Frontend auf. Sie stellen die Grundpfeiler unserer Software-Architektur dar.

### 5.2.1. Backend

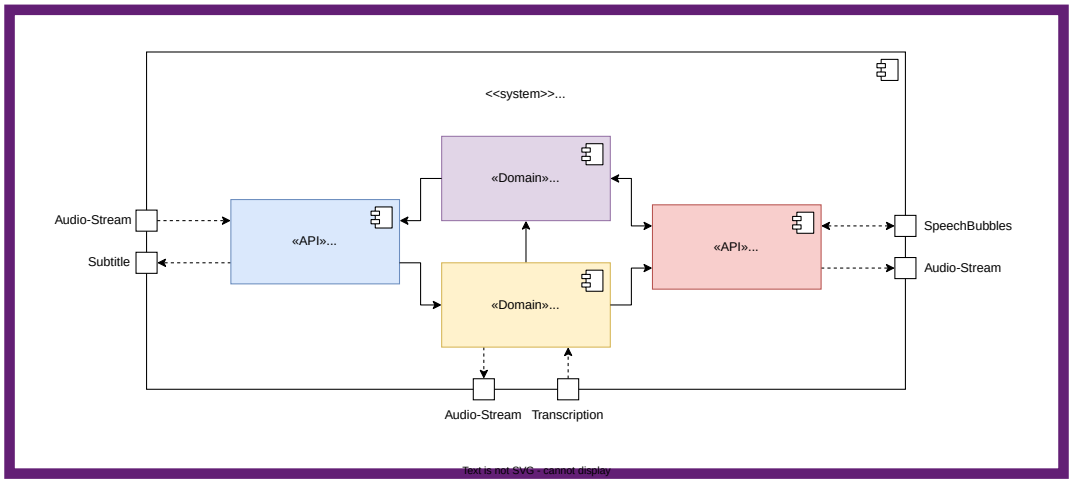



*Abbildung 2. Komponenten-Sicht des Backends*

Unser ASP.NET-Backend besteht aus vier Komponenten. Die *Client-Communication*-API kapselt die Kommunikation mit unserem Clienten. Sie empfängt den eingehenden Audiostream und überträgt den fertigen Untertitel zurück. Die *Audio*-Domäne erhält von der *Client-Communication*-API den Audio-Stream und leitet diesen direkt weiter an unsere Speech-Engine. Diese antwortet mit einer Transkription, welche später als Untertitel dient. Die rohe Transkription wird anschließend an die *SpeechBubble*-Domäne weitergeleitet, wo sie zur internen *SpeechBubble*-Datenstruktur übersetzt wird. Diese stellt eine früher Version der Untertitel dar, welche benötigt wird um Teile des Untertitels zu korrigieren. Nach Ablauf einer bestimmten, einstellbaren Zeit, werden einzelne *SpeechBubbles* zu einem echten Untertitel umgewandelt und mit Hilfe der *Client-Communication* -API an den Clienten ausgeliefert.

Dieser Kreislauf läuft autonom ab, kann jedoch durch die korrigierende Person bei Bedarf ergänzt werden. Dazu wird die *Frontend-Communication*-API benötigt, welche die Kommunikation mit dem Frontend übernimmt. Zunächst wird diese durch die *Audio*-Domäne dafür genutzt den Audio-Stream an das Frontend weiter zu leiten, sodass dieser gehört werden kann. Außerdem wird die *Frontend-Communication*-API genutzt um neue SpeechBubbles an das Frontend zu senden, um korrigierte SpeechBubbles zu empfangen und über das Ableben von SpeechBubbles zu informieren.

### 5.2.2. Frontend

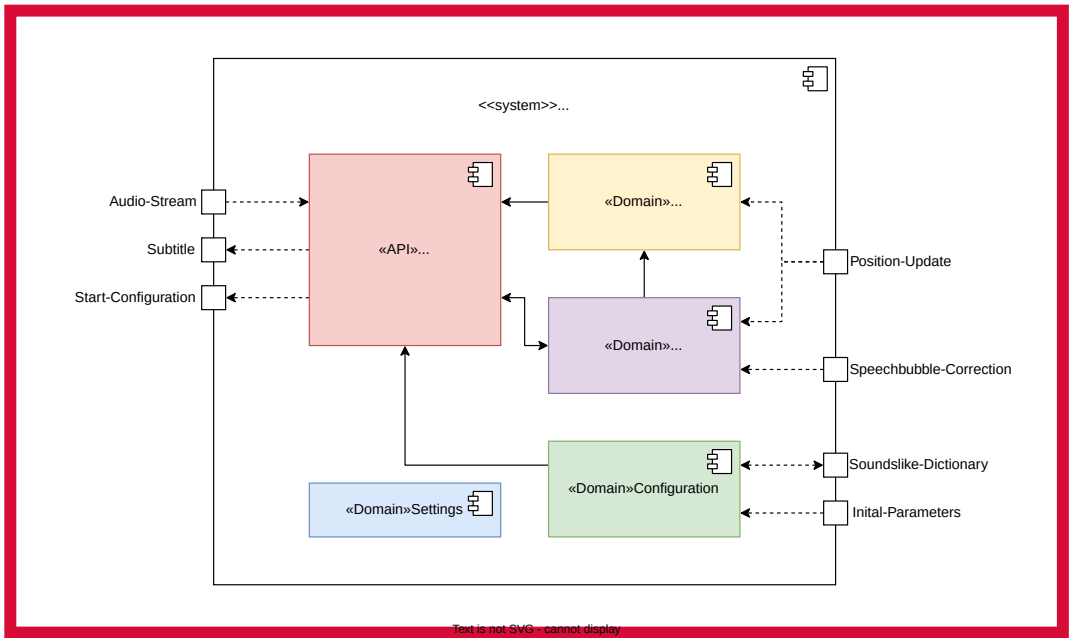



*Abbildung 3. Komponenten-Sicht des Frontends*

Unser Angular-Frontend besteht aus vier Komponenten. Analog zum Backend, wird die *Backend-Communication*-API genutzt um Daten zum Backend zu Senden oder entgegenzunehmen. Der empfangene Audio-Stream wird innerhalb der *Audio*-Domäne verarbeitet, modifiziert und für die korrigierenden Person abgespielt. Gleiches gilt für die empfangenen SpeechBubbles. Diese werden entgegengenommen und an die *SpeechBubble*-Domäne weitergeleitet. Stößt die korrigierende Person die Bearbeitung einer SpeechBubbles an, wird sie über den selben Weg an das Backend zurückgesendet. Wird ein Sprung in der Position des Audio-Streams (und daraus resultierend die Position des Cursors) ausgelöst, wird dies an die *Audio*-Domäne und die *SpeechBubble*-Domäne kommuniziert.

Isoliert davon wird die *Configuration*-Domäne genutzt, die Konfiguration der Software beim Start des Korrektur-Prozesses an alle betreffenden Komponenten und Fremdsysteme zu kommunizieren. So kann Initial ein *Soundslike-Dictionary* als Tabellen-Datei übergeben und bearbeitet werden, um der *SpeechEngine* bei der Erkennung schwieriger Wörter zu helfen. Außerdem werden *Initiale Parameter*, wie die Länge des Zeitintervalls in der eine Sprechblase existiert, hier übergeben. Die gebündelte Start-Konfiguration wird mit Hilfe der *Backend-Communication*-API an das Backend gesendet und damit die Software, beziehungsweise der Korrektur-Prozess gestartet.

## 5.3. Ebene 2 - Modul-Sicht

### 5.3.1. Backend

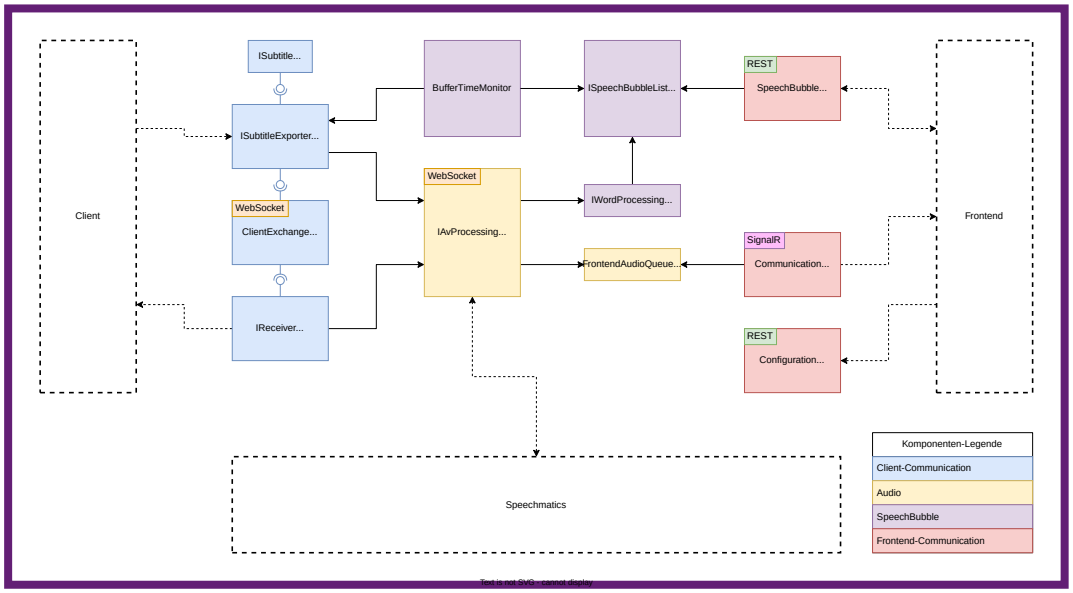



*Abbildung 4. Modul-Sicht des Backends*

## 5.3.2. Frontend

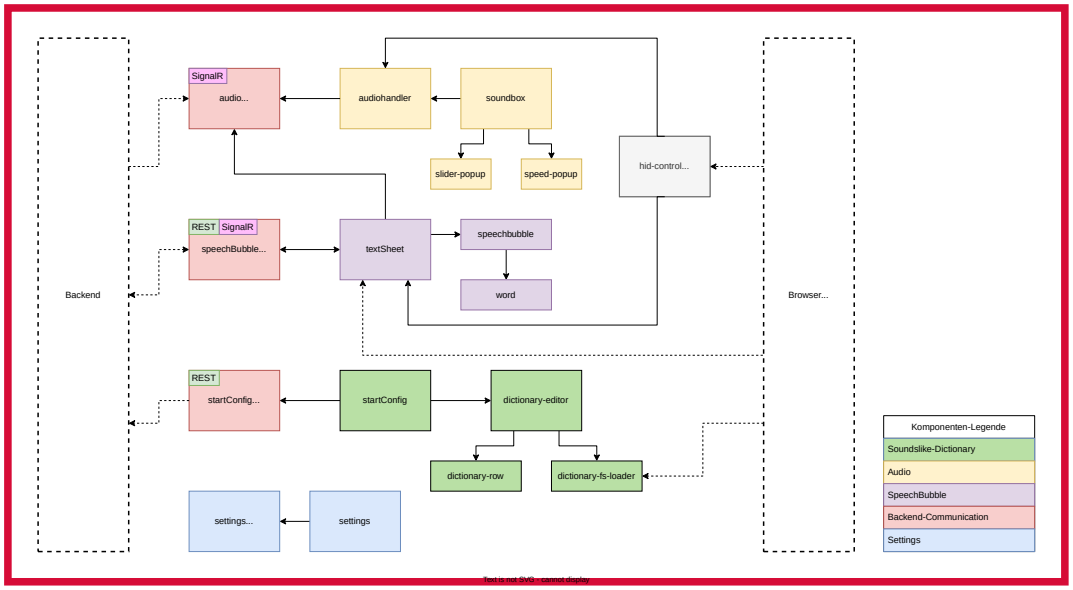



*Abbildung 5. Modul-Sicht des Frontends*

# 6. Laufzeitsicht

# 7. Verteilungssicht

# 8. Querschnittliche Konzepte

# 9. Entscheidungen

# 10. Qualitätsanforderungen

## 10.1. Qualitätsbaum

# 11. Risiken

# 12. Glossar