# 1. Introduction

## 1.1 Purpose

* **SRS Purpose:** The purpose of this document is to provide a comprehensive and detailed specification for the Fennec Transcription System. This Specification Requirements Document (SRS) defines both the functional and non-functional requirements for the system, which combines a Microsoft Office 365 Word Web Add-In with a Python-based backend. It aims to ensure that all stakeholders, including developers, testers, project managers, and end users, have a clear and consistent understanding of the system’s capabilities, constraints, and design objectives. In doing so, the document serves as a foundational reference throughout the development, implementation, and maintenance phases of the project, supporting the creation of a reliable, efficient, and secure transcription solution.
* **Context of Use:** This System is intended to replace the preexisting Dragon Transcription Software. As such it should enable Users to easily transcribe spoken words into their word documents.
* **Target Audience:** Employees at the Forensic Institute of the UZH.

## 1.2 Scope

* **Product Overview:** The System should use modern LLM’s to transcribe spoken text and automatically add it to a Word document. The Fennec Transcription System is indented to be used in conjunction with Microsoft Office 365.
* **System Boundaries:** Clarify integration points with Microsoft Office 365, any external services (e.g., speech-to-text APIs, cloud storage), and limited user operations (e.g., starting transcription, editing transcription results).

## 1.3 Definitions, Acronyms, and Abbreviations

* **Examples:**
  + **SRS:** System Requirements Specification
  + **FTS:** Fennec Transcription System
  + **IRM:** Institute for Forensic Medicine
  + **UZH:** University of Zürich
  + **O365:** Office 365
  + **UI:** User Interface
  + **UX:** User Experience

## 1.4 References

* **Documents and Resources:**
  + UML Diagram
  + User Manual
  + Developer Manual

# 2. Overall Description

## 2.1 Product Perspective

* A close-up of a computer screen

  AI-generated content may be incorrect.**System Overview Diagram:**
* **Modular Breakdown:**
* **Web AddIn Component:** Built on Office JavaScript API, handling UI interactions, file selection/dialogs, and asynchronous communication.
* **Backend Component:** Developed in Python, responsible for processing audio input, utilizing speech recognition libraries or external APIs, and returning transcription text.

## 2.2 User Needs and Use Cases

* **Primary Users:** Office 365 Word users who need to transcribe audio notes, meeting recordings, or dictations.
* **Typical Use Cases:**
* **Audio Upload or Live Recording:** User selects an audio file or initiates a live voice recording.
* **Initiating Transcription:** A user clicks a ribbon button in Word to launch the transcription window.
* **Displaying Transcription:** The backend processes the audio and the transcription result is rendered in a Word document.
* **Editing and Saving:** The user can further edit and save the transcribed text.

## 2.3 Constraints and Assumptions

* **Platform Constraints:** The AddIn must be compatible with Microsoft Office 365 on Windows and possibly Mac.
* **Assumptions:** Reliable network connectivity, modern web browsers for the AddIn, and sufficient backend resources for near-real-time processing.
* **Integration Constraints:** Adherence to Microsoft Office AddIn guidelines and Python’s compatibility with chosen libraries and services.

# 3. Functional Requirements

## 3.1 Web AddIn Requirements

* **User Interface:**
  + A dedicated Ribbon Command that opens the Transcription pane.
  + Responsive design conforming to Office 365 UI guidelines.
  + Clean One-Page Design.
* **Interaction:**
  + Ability to select between live, block and file transcription
  + Ability to initiate transcription session
  + Display of status of Transcription Backend
* **Communication:**
* Secure API calls (e.g., via HTTPS) to the Python backend.

## 3.2 Python Backend Requirements

* **Audio Processing:**
  + Accept audio file uploads or streaming input.
  + Process various common audio formats (e.g., MP3, WAV).
* **Transcription Engine:**
  + Integration with pre-built speech-to-text libraries or external APIs.
  + Capability to handle noise reduction, speaker diarization (if needed), and real-time or batch processing.
* **API Endpoints:**
* RESTful endpoints to manage transcription requests, status checks, and result retrieval.
* Error handling and clear error code returns for failures (e.g., file format issues, backend timeouts).

## 3.3 Data Flow and Security

* **Data Management:**
  + All audio data is stored until the session ends.
* **Security:**
* Encryption of data in transit (SSL/TLS).
* Authentication mechanisms for API requests (e.g., token-based authentication).
* Authorization levels ensuring that only permitted users can request and view transcriptions.

# 4. Non-Functional Requirements

## 4.1 Performance

* **Latency:** Transcriptions should be returned in near real-time (define acceptable thresholds).
* **Scalability:** The Python backend should scale horizontally to accommodate multiple transcription sessions concurrently.
* **Responsiveness:** The add-in should stay interactable during transcription

## 4.2 Reliability and Availability

* **Uptime:** Define expected system availability (e.g., 99.9% uptime).
* **Fault Tolerance:** The system should have a retry logic for transient network errors and fallbacks for failed transcriptions.

## 4.3 Usability

* **User Experience (UX):** The interface should be intuitive for non-technical users. Provide clear instructions and error messages.
* **Cross-Platform:** Ensure compatibility with different versions of Office 365 and various Windows/Mac environments.

## 4.4 Maintainability

* **Modular Architecture:** Clear separation between the frontend AddIn and the backend service.
* **Documentation:** Maintain comprehensive inline documentation for the Python codebase and user documentation for the AddIn.
* **Logging and Monitoring:** Robust logging and monitoring systems for troubleshooting both the web component and backend process.

## 4.5 Security and Compliance

* **Data Privacy:** Compliance with GDPR and other relevant data privacy regulations.
* **Secure Coding Practices:** Regularly scheduled security audits and penetration tests.

# 5. System Architecture and Design

## 5.1 Architecture Overview

* **Component Diagram:** Visualize how the Web AddIn communicates with the Python backend via RESTful APIs.
* **Technology Stack:**
  + **Frontend:** HTML, CSS, JavaScript (Office.js), potentially React or Angular for richer UI.
  + **Backend:** Python using the FastAPI-Framework speech-to-text libraries, external APIs.
* **Deployment Environment:**
* Specify cloud hosting or on-premise deployment details.
* Supported operating systems for the backend environment and browsers for the add-in.

## 5.2 Integration Points

* **Office 365 Integration:** Guidelines on how the add-in interacts with Word documents.
* **Third-Party APIs:** Define any third-party transcription services (if applicable) and their integration points.

# 6. Interface Requirements

## 6.1 User Interface (UI) Requirements

* **Office AddIn UI:**
  + Consistency with Microsoft Office Fluent UI.
  + Easily accessible buttons, status messages, and error displays.
* **Backend Admin UI:** For monitoring server health and logging transcription errors, as well as changing the Version of the Whisper model used.

## 6.2 API Interface Requirements

* **Endpoint Descriptions:** Document endpoints, parameters, sample requests/responses, and expected HTTP status codes.
* **Versioning:** API version details to ensure backward compatibility.

# 7. Data Requirements

## 7.1 Data Formats and Storage

* **Input Audio Files:** Supported audio file formats and maximum file size.
* **Transcription Data:** Format for storing and retrieving transcription results (e.g., plain text, JSON).
* **Temporary Storage Requirements:** Retention policies for audio and transcription data.

## 7.2 Data Handling and Privacy

* **Data Encryption:** Both in storage and during transmission.
* **User Consent:** Procedures to ensure users agree to data handling policies per privacy laws.

# 8. Quality Assurance and Testing

## 8.1 Testing Strategies

* **Unit Testing:** For both the Office AddIn (using JavaScript testing libraries) and Python backend (using frameworks like PyTest).
* **Integration Testing:** Ensure communication between the add-in and backend is robust and secure.
* **User Acceptance Testing (UAT):** Validate that the tool meets user accessibility and usability expectations.
* **Performance Testing:** Load and stress tests for the Python backend under various conditions.

## 8.2 Validation Criteria

* **Success Criteria:** Benchmark metrics (e.g., transcription accuracy rates, API response times).
* **Error Handling:** Test suite for common error cases like poor audio quality or network issues.

# 9. Project and Deployment Considerations

## 9.1 Deployment Environment

9.1.1 Setup Development Environment

## 9.2 Maintenance and Future Enhancements

### 9.2.1 Update Whisper Models.

To update the Whisper models, install the newest Version of the Whisper.net Library with NuGet. Then build the project in Debug mode. Make sure to set the models downloaded to the Models Directory to ...

# 10. Appendices

## 10.1 Glossary

* **Terms:** Provide detailed definitions for all technical and business terms used.

## 10.2 Document History

* **Versioning:** A table of updates, revisions, and change logs.

## 10.3 Supporting Diagrams

* **Diagrams:** Include any component diagrams, sequence diagrams, or workflow illustrations that further explain the system.