# High Level Architecture (Block Diagram)

## Components:

- \*\*Frontend:\*\*

- Technology: ReactJS

- Purpose: User interface for displaying questions and microphone icon.

- \*\*Backend:\*\*

- Technology: Python

- Purpose: Handle business logic, integrate AI verification, manage API interactions.

- \*\*Speech-to-Text Service:\*\*

- Technology: Whisper or Google TTS API

- Purpose: Convert speech to text in real-time.

- \*\*AI Verification Module:\*\*

- Technology: Python-based AI models

- Purpose: Verify the accuracy of candidate answers.

## Relationships:

- \*\*Frontend <--> Backend:\*\*

- Data Flow: Candidate interactions (question display, microphone activation).

- \*\*Backend <--> Speech-to-Text Service:\*\*

- Data Flow: Audio input from candidates, text output to backend.

- \*\*Backend <--> AI Verification Module:\*\*

- Data Flow: Transcribed text, verification results.

# Block Diagram:

## plaintext

+-------------+ +--------------+ +---------------------+ +-------------------+

| Frontend |<----->| Backend |<----->| Speech-to-Text API |<----->| AI Verification |

| (ReactJS) | | (Python) | | (Whisper/Google) | | (Python) |

+-------------+ +--------------+ +---------------------+ +-------------------+

```

# Workflow Design (Flowchart)

## plaintext

Start

|

Display Question ---> Candidate Speaks ---> Microphone Icon Clicked ---> Speech-to-Text API Called

| | |

v v v

AI Verification <--- Receive Transcribed Text <--- Speech Converted to Text

|

Verification Result Displayed

|

End

```

# Message Sequence Chart (MSC)

## Components:

- \*\*User (Candidate)\*\*

- \*\*Frontend (ReactJS)\*\*

- \*\*Backend (Python)\*\*

- \*\*Speech-to-Text API (Whisper/Google)\*\*

- \*\*AI Verification Module (Python)\*\*

## Sequence:

```plaintext

User Frontend Backend Speech-to-Text API AI Verification

| | | | |

|------------------> | | | |

| Click Microphone |------------------> | | |

| | Send Audio |------------------------>| |

| | | Convert Speech to Text| |

| |<------------------ |<------------------------| |

| | Receive Transcribed| | |

| | Text | | |

| |------------------->| | |

| | Send Transcribed | | |

| | Text | |----------------------->|

| | | | Verify Answer |

| |<------------------ | |<-----------------------|

| | Receive Verification Result | |

|<------------------ | | | |

Display Verification Result | | |

```

# UI Wireframing (Optional)

## Main Screen

- \*\*Elements:\*\*

* Question display area.
* Microphone icon/button.

Real-time text display of spoken answers.

- Verification result area.

# Wireframe:

## plaintext

+-----------------------------------+

| Question: |

| What is your name? |

+-----------------------------------+

| |

| [Microphone Icon] |

| |

+-----------------------------------+

| Real-time Transcription: |

| [Candidate's spoken answer here] |

+-----------------------------------+

| Verification Result: |

| [Correct/Incorrect] |

+-----------------------------------+

```

# Sample Code Snippet (Optional)

## Speech-to-Text Conversion (Python Example Using Google TTS API)

```python

import speech\_recognition as sr

from google.cloud import speech

# Initialize recognizer

recognizer = sr.Recognizer()

def record\_audio():

with sr.Microphone() as source:

print("Please say something...")

audio\_data = recognizer.listen(source)

print("Recording complete.")

return audio\_data

def speech\_to\_text(audio\_data):

client = speech.SpeechClient()

audio = speech.RecognitionAudio(content=audio\_data.get\_wav\_data())

config = speech.RecognitionConfig(

encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,

sample\_rate\_hertz=16000,

language\_code="en-US",

)

response = client.recognize(config=config, audio=audio)

for result in response.results:

print("Transcript: {}".format(result.alternatives[0].transcript))

if \_\_name\_\_ == "\_\_main\_\_":

audio\_data = record\_audio()

speech\_to\_text(audio\_data)

```

# Guidance

## Clarity and Consistency

- Use distinct shapes, line styles, and colors to differentiate elements within your diagrams.

- Maintain a clear visual hierarchy.

## SRS Alignment

- Clearly trace how your diagrams address the requirements outlined in the SRS.

## Collaboration

- Consider how your diagrams will facilitate understanding between different teams involved in the development process.

## Trade-offs

- Acknowledge design trade-offs and the reasoning behind them.

## Tools

- Utilize diagramming tools such as:

- draw.io: Free and intuitive

- Lucidchart: Collaborative with extensive templates

- Visio: Industry standard, suitable for complex diagrams

This SRS is designed to align with the specific goals and constraints of the PoC phase, ensuring a focused and effective demonstration of the AI interview tool's feasibility.