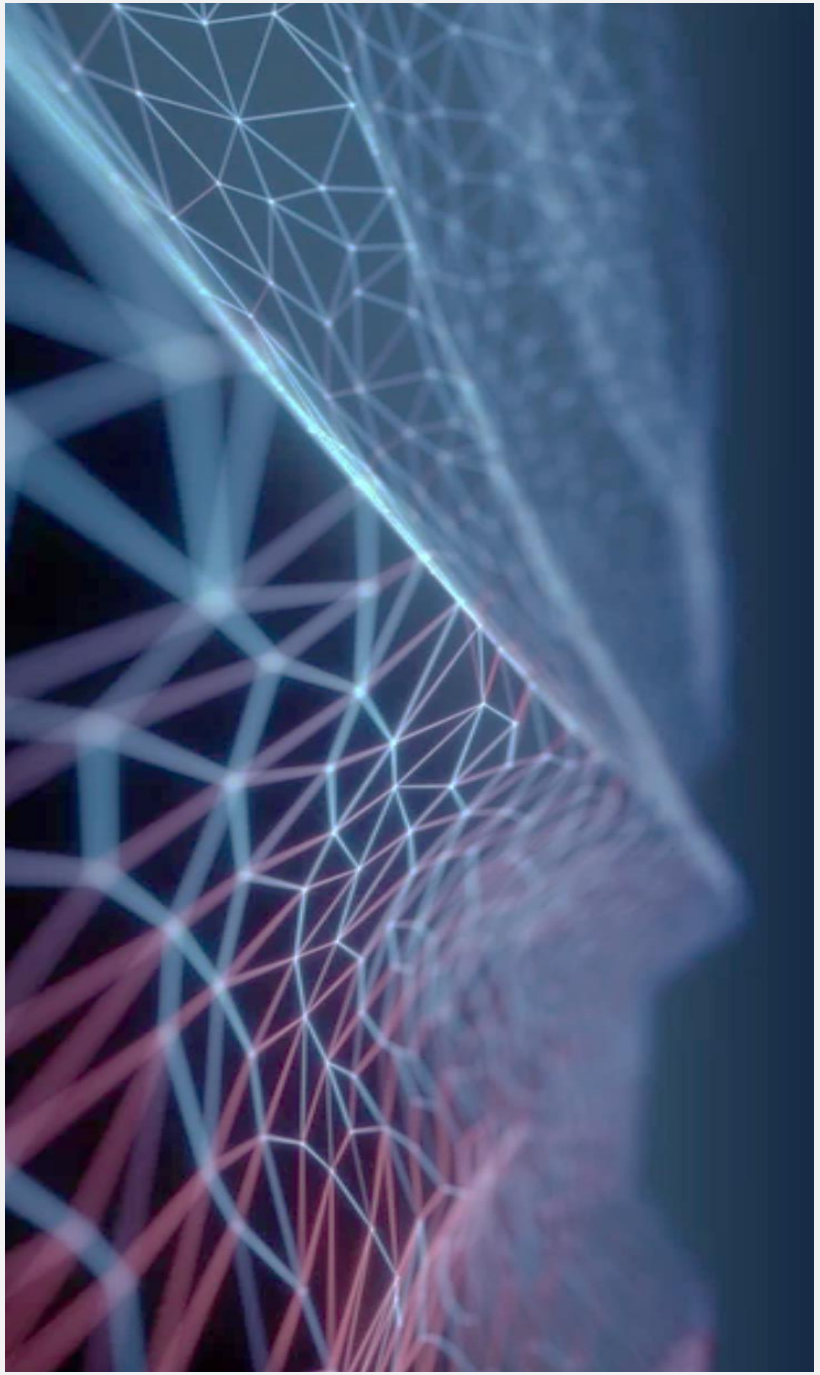
FORGE AI

4th June Report

Agenda

- Motivation and Goals
- Architecture
- Service Structure
- Development Plan
- Showcase of Prototype



Introduction to the Problem

AI Platforms tend to offer a single product type. Either text, image, audio, or video related... At least until Chat-GPT 4o came up!

This requires multiple accounts and subscriptions in multiple platforms, making the whole process inefficient and bothersome.

That goes without saying the need to learn a whole new User Interface!

Solution!

FORGE AI

Goal

An all-in-one SaaS platform that allows users to do the basics of each area: Text, Audio, Image, Video.

For better accessibility, a frontend should be implemented that deals with the backend interaction for the user.

The user should be able to upload files, get results, and then be able to retrieve their previous queries (persistence)

ForgeAI - Requirements of MVP

- Users can upload audio files and receive a transcription of the contents.
- Users can ask questions and receive answers (text generation).
- Users can upload movies/videos and receive a text summary (video captioning)
- Users can upload images and receive an explanation of the content (image captioning)
- Users interact through a web application, offering all services.
- Users are able to retrieve their past queries and responses from Forge AI.

Architecture of Forge AI

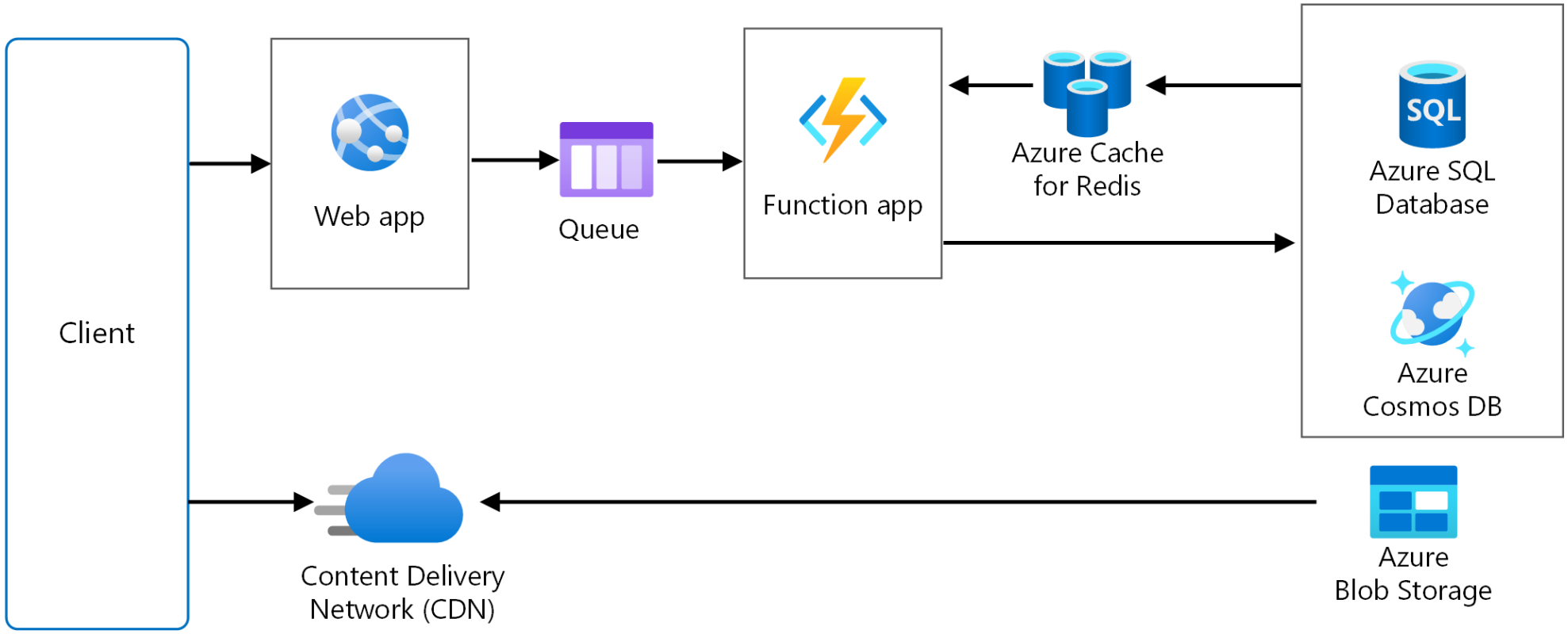
We use the web-worker and serverless architecture.

This allows the asynchronous handling of requests, allowing the main application to remain responsive while processing things in the background

It also separates the core logic from the frontend, making the system easier to maintain and update.

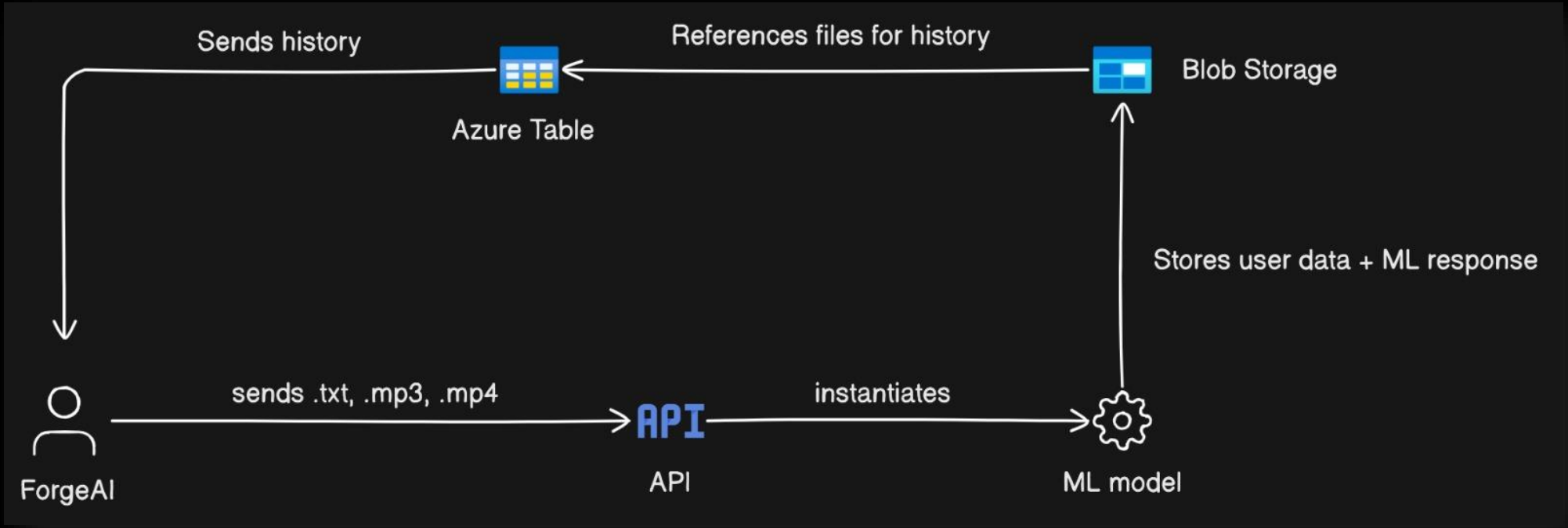
This also enhances security, as users do not have direct access to the backend or its data.

Web-Worker Architecture

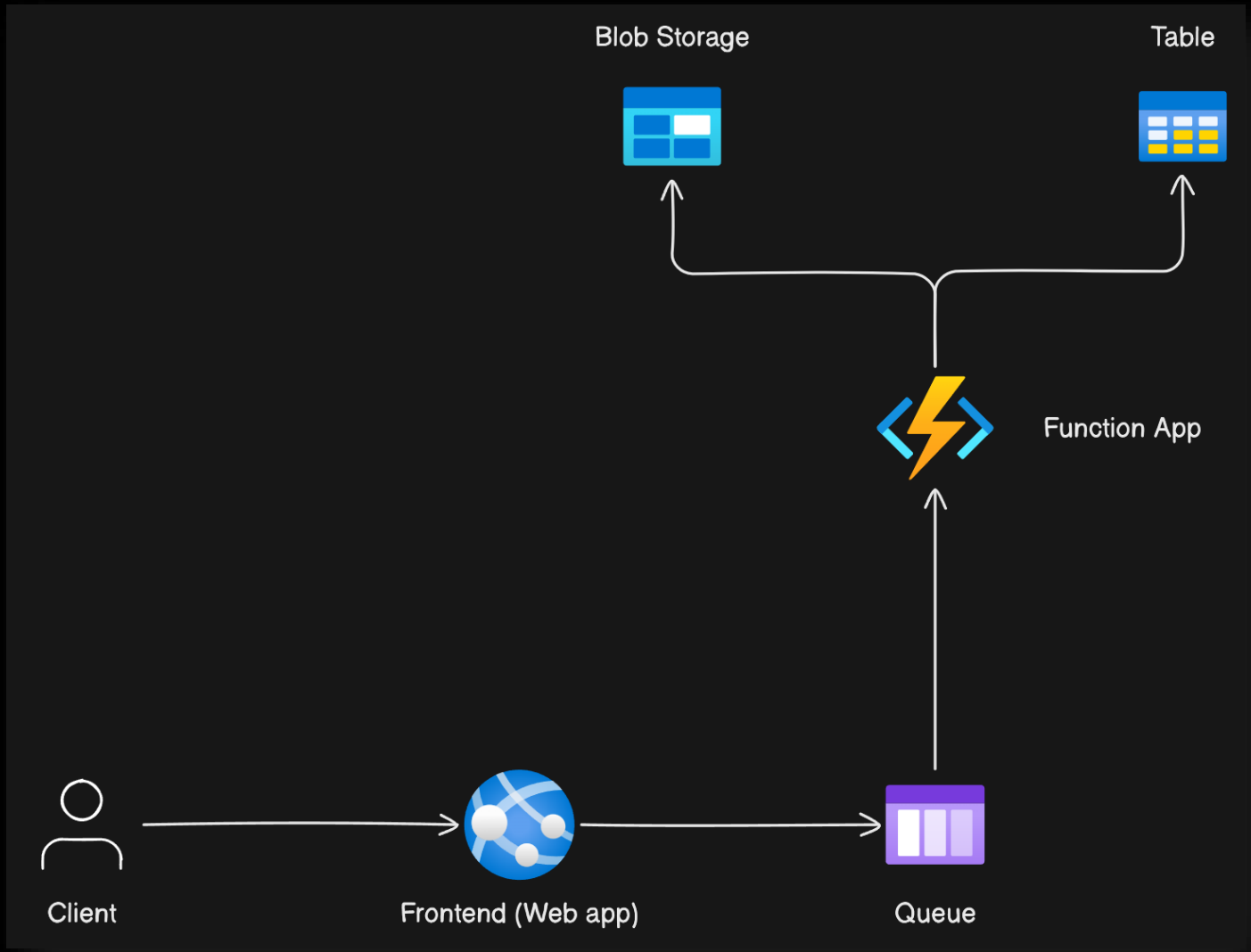


Microsoft
Azure

The previous architecture



The new architecture



Reasons for switching to **Function Apps**

Perfomance

Function apps are faster than FastAPI when sending data or fetching the history.

Regularity

Might as well stick to Azure since we're already using most services such as Queue, Blob, Storage, and Video Indexer.

Safety

Setting things such as CORS and allowable file types are much easier within Function Apps.

How it works – Requesting Services

User selects task type and uploads a file (e.g image and image captioning).

This calls the azure function app.

```
callAzureFunction(URL, { action: 'text2text', data : 'data:text/txt;'+e.target[0].value }
```

The function app temporarily uploads user data into the blob storage, decodes the base64, and checks which machine learning model to run.

```
blob name = upload data to blob(CONTAINER NAME, content, extension)
base64.b64decode(request.split(sep=',')[1])
match action:
    case "text2text": response, dic = text2text(content)
    case "audio2text": response, dic = audio2text(content)
```

How it works – Requesting Services

After the ML model returns the result, this is also uploaded to the blob storage and a table stores a reference to the client's file and the result's file.

Timestamp	request	request_type	response	response_type
2024-05-31T19:15:19.28...	7ed73a3e-4e33-4d45-a...	audio	3a8acb28-e80d-496f-85...	text
2024-05-31T19:26:43.73...	469e04d6-1a2d-4176-9...	text	2a9bf591-280f-4899-9b...	text
2024-05-31T19:37:00.22...	16c1bbfb-8a1c-40e6-81...	audio	877f5dc3-7326-40c3-ba...	text
2024-06-01T09:20:36.20...	da1e1581-9448-458b-9f...	text	62b882e0-693d-4c5a-b...	text
2024-06-01T09:20:53.30...	0b37f5e3-6c32-42cc-82...	audio	5779ed6a-0715-4261-a...	text

The result of the ML model is then sent back to the frontend. If something went wrong, the function app returns 400

```
    return func.HttpResponse(f"{response}", status_code=200)
else:
    return func.HttpResponse("An error has occured during processing.",status_code=400)
```

How it works – Getting previous data

When the user refreshes the page, the frontend tells the function app to fetch previous data.

```
async function getHistory() {
    try {
        const response = await axios.get(HISTORY_URL);
```

The backend receives the request. Then it looks into the files referenced in the table, downloads them from the blob storage, and sends them to the user.

```
for entity in table_client.list_entities():
    blob_service_client = BlobServiceClient.from_connection_string(connect_str)
    request_data = blob_service_client.get_blob_client(container=CONTAINER_NAME, blob=entity['request']).download_blob().content_as_text()
    response_data = blob_service_client.get_blob_client(container=CONTAINER_NAME, blob=entity['response']).download_blob().content_as_text()

    history.append({
        "type": entity['request_type'],
        "user": request_data,
        "type_response": entity['response_type'],
        "response": response_data,
    })
```

Timestamp	request	request_type	response	response_type
2024-05-31T19:15:19.28...	7ed73a3e-4e33-4d45-a...	audio	3a8acb28-e80d-496f-85...	text
2024-05-31T19:26:43.73...	469e04d6-1a2d-4176-9...	text	2a9bf591-280f-4899-9b...	text
2024-05-31T19:37:00.22...	16c1bbfb-8a1c-40e6-81...	audio	877f5dc3-7326-40c3-ba...	text
2024-06-01T09:20:36.20...	da1e1581-9448-458b-9f...	text	62b882e0-693d-4c5a-b...	text
2024-06-01T09:20:53.30...	0b37f5e3-6c32-42cc-82...	audio	5779ed6a-0715-4261-a...	text

3a8acb28-e80d-496f-858d-dba50277995e.txt

7e3d071d-9e37-4335-be9e-1d0c77148cdc.mpeg

Development Plan

75%	Development of Frontend	Shaheryaar + Gabriel
100%	Development Blob / Table	Lulu
50%	Gitlab CI/CD	Victor
75%	Development of Function App	Victor
50%	Integration of Frontend to Backend	Gabriel
100%	Development of Video Captioning	Josue

Current Technologies Used

Azure Video Indexer

Azure Storage Account

Azure Function App

Open AI + g4f

React

Axios requests

Let’s try it out!

Showcase of Prototype

Let’s try it out!