# Group 1: Face Recognition System Documentation

**Ignatius Boadi**
AIMS AMMI
Mbour, Sénégal
iboadi@aimsammi.org

**Millicent Omondi**
AIMS AMMI
Mbour, Sénégal
momondi@aimsammi.org

**Leonard Sanya**
AIMS AMMI
Mbour, Sénégal
lsanya@aimsammi.org

## 1 Introduction

We are developing a face recognition system aimed at streamlining the enrollment process for individuals. This system can be utilized by clients of an organization to access designated services or products independently. Users can enrol with minimal supervision by implementing a self-service platform, providing their credentials and a facial photograph. Upon subsequent logins, the system will recognize their face, granting them access seamlessly. Find the GitHub repo here.

### 1.1 Goals

- Facilitate seamless enrollment for businesses and organizations.
- Enhance security through robust user verification.

### 1.2 Pains

- Many existing face recognition systems are tailored for specific applications. Our objective is to create a flexible, robust solution applicable across various organizations.
- Security is very important; only approved users should be able to access their accounts. Our face recognition system improves security, helping to build trust among users.

### 1.3 Value Proposition:

We aim to deliver a versatile and reliable system applicable across various organizations, empowering customers to enrol independently.

### 1.4 Solution

- **Technology:** Utilize advanced facial recognition algorithms such as MTCNN, known for their efficiency and accuracy.

Key Features:

- Real-time facial recognition
- Instant access upon successful identification.
- Automated security alerts for unauthorized access attempts.

- **Integration:** Compatible with existing systems, ensuring smooth deployment.

### 1.5 Feasibility and Resources

To implement this we need;

- Cameras: Webcam on laptops and phone cameras.
- Robust computational hardware for image processing.
- A comprehensive database of the enrolled users' facial images.

### 1.6 Potential Barriers

Challenges may include:

- Poor lighting conditions and low-quality cameras may cause poor recognition.
- Integration difficulties with existing users.

### 1.7 Data Sources

We will be using a pre-trained model, we will get our data as users enrol in the system.

### 1.8 Labeling Process

After a user's photo has been taken after enrollment, a folder for each user will be automatically created and stored in the database with the enrollment credentials.

### 1.9 Metrics

- **Primary Metrics:** Recognition accuracy, false acceptance rate (FAR), false rejection rate (FRR), system response time.
- **Tracking:** Continuously monitor these metrics from time to time to ensure the system meets performance standards.

### 1.10 Evaluation

- **Offline Evaluation:** Regular testing against a set default dataset to ensure consistent performance and adaptability to new data.
- **Online Evaluation:** Assessment of real-time performance using data captured from the operational environment to resolve any recognition issues quickly.

### 1.11 Modelling

- Use a pre-trained model for face recognition. We used MTCNN, which can detect and extract facial features
- **Development Phases:** Continuous updates and retraining as new data becomes available.

### 1.12 Inference

- **Mode:** Real-time processing on local edge devices to minimize latency and maximize responsiveness.
- **Deployment Strategy:** Integrate the facial recognition software into web applications for instant recognition on mobile phones and laptops.

### 1.13 Inference

- **Sources:** From users, system logs, and incident reports.
- **Utilisation:** Use the feedback to fine-tune recognition algorithms, improve system usability, and enhance security protocols.

### 1.14 Technological Considerations

We will be using the following tools in implementing the face recognition system:

- **GitHub:** For version control and collaboration.
- **VS Code:** As the integrated development environment.
- **Jupyter Notebook:** Data exploration and code testing.
- **DVC:** For data versioning and management.
- **Docker:** For containerization, ensuring consistent deployment.
- **MLflow:** Tracking, versioning, and management of the model used.
- **MySQL:** Database to store user data and metadata.
- **FasAPI:** Web framework for building APIs.
- **GCP:** Cloud platform for model deployment and to ensure scalability.
- **Streamlit:** Frontend interface for users.

### 1.15 Project

- **Team:** Our team consists of three MLOps engineers. We will security engineers, computer vision experts, and those who will train the organizations on how to set up the systems and their use.
- **Deliverables:** A fully functional self-service facial recognition system.
- **Timeline:** Estimated project completion within 4-6 months from initiation, including testing and adjustments.

## 2 System Architecture

The face recognition system has been designed to provide a seamless user experience from enrolling to logging in. It ensures robust data management and security. The architecture 1 is divided into different key components, each serving its purpose for a smooth face recognition process.

In figure 1, we observe the following components of the system:

- **User Clients:** In the system, both web and mobile clients can interact with the system. They will be able to enrol and log in using their credentials.
- **Data Management:** The system uses DVC for data versioning, ensuring that all datasets are tracked. Uploaded images are processed through the data transformation pipeline before being stored in the pipeline.
- **Data Storage:** The images will be stored in the cloud on the GCP cloud bucket and queried using MySQL.
- **Model Registry:** It is utilized to manage the life-cycle of the machine learning model. The pre-trained model is registered and can be updated as needed, allowing easy retrieval and deployment.
- **Face Recognition Service:** It processes the uploaded images for the users and performs face detection and feature extraction. The extracted features are then matched against stored features in the database.
- **Performance monitoring:** For tracking prediction accuracy and data pipeline performance. It ensures that any degradation in service quality can be addressed.

### 2.1 Workflow

The workflow as shown in figure 2, starts with users inputting their credentials and uploading their face images from the photos taken during enrollment. The system then processes these images, extracts the relevant facial features like nose, eyes, and mouth, etc. These are then stored in the database for future recognition tasks. During authentication, user images are compared to the stored features, and access is granted based on a defined confidence level threshold.

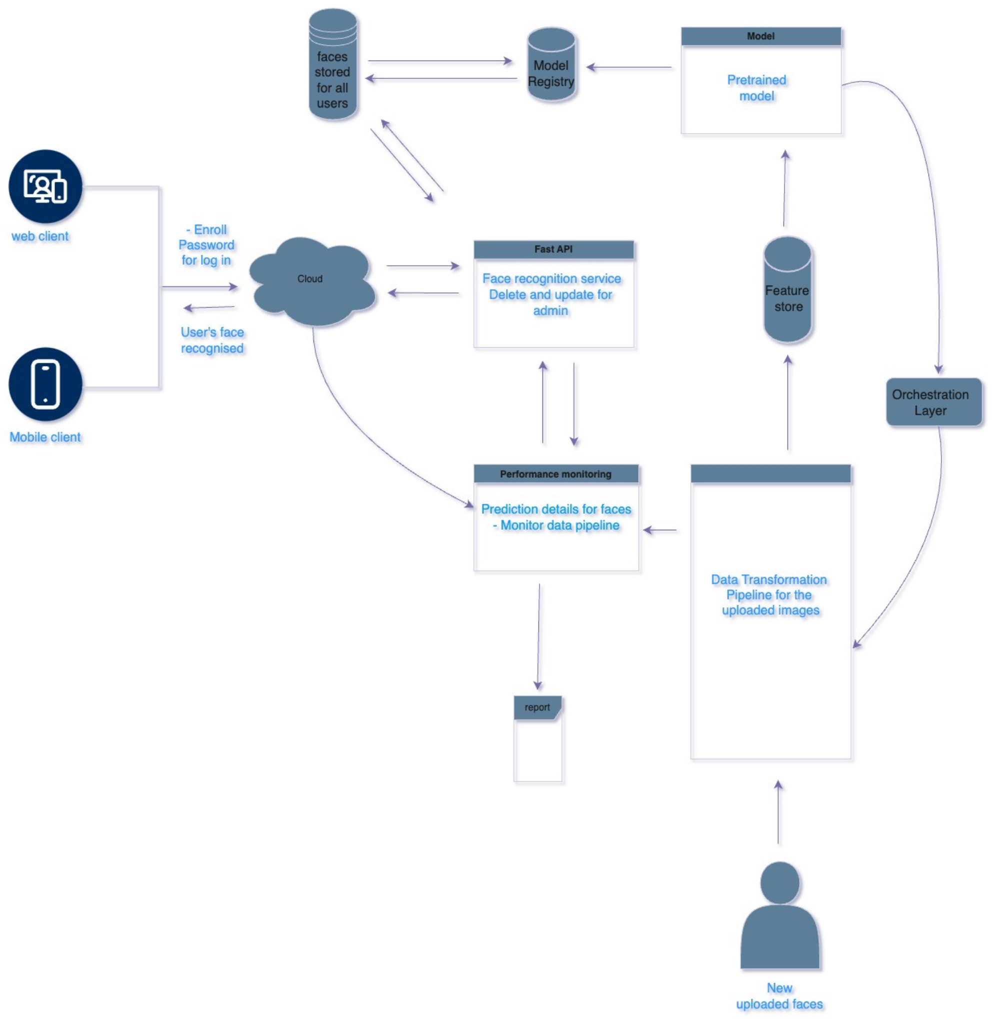

faces stored for all users
Model Registry
Model
Pretrained model
web client
- Enroll Password for log in
Cloud
Fast API
Face recognition service Delete and update for admin
Feature store
User's face recognised
Mobile client
Orchestration Layer
Performance monitoring
Prediction details for faces - Monitor data pipeline
Data Transformation Pipeline for the uploaded images
report
New uploaded faces

Figure 1: System Architecture Overview [source:neptune.ai]

## 3 Setup and installation

We will need to install the following tools and libraries before setting up our face recognition system:

**Tools**: Python (version 3.7 or higher), docker, git, MySQL, DVC, and Anaconda or virtual environment.

### 3.1 Environment Setup

#### 3.1.1 Local development

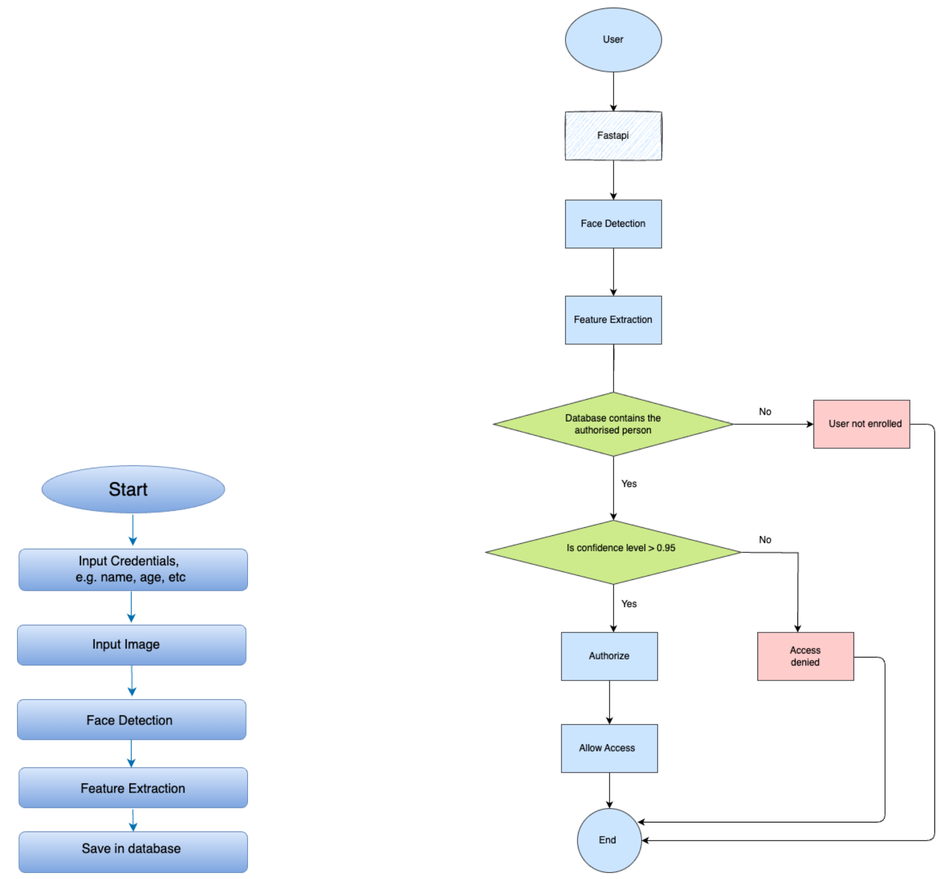
1. Clone our repository

```
git clone <git repo>
cd face_recognition_system
```

2. create a virtual environment

```
    python -m venv venv
source venv/bin/activate  # On Windows use ‘venv\Scripts\activate‘
```

3. Install required packages. Create a requirements.txt file with the following dependencies:



(a) Data and Input storage (b) User authentication process

Figure 2: System design [Source: Authors]

```
tensorflow
pydantic
sqlalchemy
python-jose
opencv-python
numpy
python-multipart
bcrypt
fastapi[standard]
pydantic
sqlalchemy
face-recognition
opencv-python
numpy
Pillow
mtcnn
passlib
python-jose
uvicorn
mysql-connector-python
```

4. Install the required packages:

```
pip install -r requirements.txt
```

### 3.1.2 Using Docker

Docker allows seamless deployment of the application across different environments. Steps to set up the project using Docker:

1. Install Docker here.
2. Build the Docker image. Create a Docker file using "touch Dockerfile" in your terminal.

```
FROM python:3.11.5

RUN apt-get update -y && \
    apt-get install -y --no-install-recommends \
    fswebcam \
    v4l-utils

# Install system dependencies
RUN apt-get update && apt-get install -y \
    cmake \
    build-essential \
    libopenblas-dev \
    liblapack-dev \
    libx11-dev \
    libgtk-3-dev \
    libboost-python-dev \
    && rm -rf /var/lib/apt/lists/*

# Set the working directory
WORKDIR /code

# Copy the requirements to the working directory
COPY ./requirements.txt /code/requirements.txt

# Install dependencies
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

# Copy the app directory contents to the working directory
COPY ./main.py /code/main.py

# Expose port 80
EXPOSE 80

# Run the FastAPI application using Uvicorn
CMD ["sh", "-c", "uvicorn main:app --host 0.0.0.0 --port 80"]
```

3. Build the docker image and run the docker container.

```
    docker build -t face-recognition-system .
    docker run -d -p 8000:8000 face-recognition-system
```

4. Open `http://localhost:8000` on your browser to access the FastAPI application.

## 3.2 Data Management

### 3.2.1 Data Versioning with DVC:

Follow these steps to set up DVC for the project:

1. Install DVC, if not yet installed.

```
    pip install dvc
```

2. Initialize DVC

   ```
   dvc init
   ```

3. Track data files assuming your dataset is in a folder named data.

   ```
   dvc add data/
   ```

4. Push data to a remote storage

   ```
   dvc remote add -d myremote <remote_storage_url>
   dvc push
   ```

## 4 Model Development

1. We utilize a pre-trained model from the MTCNN library. Run the main.py file:

   ```
   python main.py
   ```

2. Log Metrics with MLflow

   ```
   import mlflow
   mlflow.start_run()
   mlflow.log_param("num_epochs", num_epochs)
   mlflow.log_metric("accuracy", accuracy)
   mlflow.end_run()
   ```

## 5 Deployment

### 5.1 API Development with FastAPI

### 5.2 Frontend with Plotly Dash

It provides an easy way to create a frontend interface for users to interact with the brain tumour segmentation service. We chose Plotly Dash as it is a powerful framework that is well-suited for data visualizations given that we will be using image data.

1. Create a plotly dash script, e.g app.py
2. Run the plotly dash application

   ```
   pip install dash
   python app.py
   ```

3. After running the command, you will see an output ( http://127.0.0.1:8050/ or http://localhost:8050/ ).
4. Open your browser and paste the URL to access the dash application.

**Database Integration:** The FastAPI application interacts with the MySQL database to manage user's data. It ensures that the database connection settings are configured correctly in the application.

## 6 Continuous Integration and Deployment (CI/CD)

We used Github actions to set up the workflow to ensure code quality and smooth deployment.

1. Set up GitHub Actions: Create a .github/workflows/ci.yml file with steps for building and testing the application.
2. Deploy to GCP

## 7 Monitoring and Maintenance

After a given period, we will use the images that were earlier collected and compare them to the recently uploaded ones for monitoring model performance. We will use "evidently" to detect data drift.

## References

`https://neptune.ai/blog/mlops-architecture-guide`

`Class notes: https://github.com/AMMI-2024/mlops-course.git`