# Architecture

* **Client-Side (Web Interface):**
  + HTML, CSS, and JavaScript for structure, styling, and basic interactions.
  + JavaScript library for image upload handling (e.g., basic validation and resizing).
* **Server-Side (Backend):**
  + Python (common choice for image processing and machine learning).
  + Web framework like Flask or Django to handle incoming requests and serve content.
  + OpenCV library for image processing.
  + Pre-trained models for facial landmark detection and emotion recognition (we'll address options below).
* **Cloud Integration (Optional):**
  + Consider cloud deployment (AWS, Google Cloud, Azure) for scalability and ease of testing. This can be simplified using containerization (e.g., Docker).

**Block Diagram Structure**

1. **Client (Web Browser):**
   * Box labeled "Web Browser"
   * Inside the box, you could include: "HTML, CSS, JavaScript"
2. **Server (Backend):**
   * Box labeled "Server"
   * Inside:
     + "Python (Flask/Django)"
     + "OpenCV"
     + "Facial Landmark Model"
     + "Emotion Recognition Model"
3. **Arrows:**
   * **Arrow 1:** From "Web Browser" to "Server" labeled "Image Upload (HTTP Request)"
   * **Arrow 2:** From "Server" to "Web Browser" labeled "Processed Image + Emotion (HTTP Response)"
4. **Optional (Cloud):**
   * If using cloud deployment, encapsulate the entire server-side with a larger box labeled "Cloud (AWS/GCP/Azure)"

**PowerPoint Presentation Notes**

Here's the kind of description you might use alongside the diagram in your slide:

* **Title:** Facial Analysis PoC - System Architecture
* **Speaker Notes:**
  + "This diagram illustrates the core components and data flow within our proof-of-concept system."
  + "The user interacts through a standard web browser, uploading an image for analysis."
  + "The server, powered by Python and specialized libraries, handles image processing, landmark detection, and emotion recognition using pre-trained models."
  + "The analysis results are returned to the web browser, where they are displayed visually."
  + "During the PoC phase, focus is on establishing this foundational architecture; more complex cloud integration or data handling would be part of later iterations."

**Tips for Making the Diagram**

* **Use PowerPoint Shapes:** Employ basic rectangles and arrows from the PowerPoint shape library.
* **Color Coding:** Consider subtle color differentiation between the client-side, server-side, and cloud elements (if applicable).
* **Keep it Clean:** Aim for visual clarity and a balanced layout.
* **Text Size:** Make sure labels are readable when presented.

# Workflow

1. **Image Upload:**
   * User selects an image through the web interface.
   * JavaScript performs basic validation (file type, size).
   * The image is sent to the server as a file.
2. **Preprocessing:**
   * Image is resized and standardized (if needed) using OpenCV.
   * Image may be converted to grayscale (depending on models used).
3. **Facial Landmark Detection:**
   * Pre-trained facial landmark detection model (see design considerations below) is applied to locate key points.
4. **Emotion Recognition:**
   * Pre-trained emotion recognition model analyzes facial features to classify emotion.
5. **Result Generation:**
   * Visual overlay representing detected landmarks is created (using OpenCV or a front-end drawing library).
   * Text label representing the classified emotion is generated.
6. **Result Display:**
   * The server sends the processed image with the overlay and the emotion label back to the client's web browser.
   * The web interface displays the results.

# Design Considerations

* **Facial Landmark Detection Model:**
  + dlib's shape predictor: A mature and well-regarded option.
  + MediaPipe Face Mesh: Google's solution providing very granular feature detection.
* **Emotion Recognition Model:**
  + Pre-trained on datasets like FER-2013, AffectNet
  + May require some fine-tuning on a smaller dataset more relevant to your use case.
* **Design Patterns:**
  + **Model View Controller (MVC):** To structure the web application (may be simplified, considering the PoC scope).
  + **API-driven interaction:** A RESTful API to facilitate communication between the front-end and back-end.

# UI Wireframing (Basic)

Here's a simple wireframe concept for the web interface:

* **Header:**
  + Project Title (e.g., "Facial Analysis Proof of Concept")
* **Main Area:**
  + **Image Upload Section:**
    - Large "Upload Image" button.
    - Display for the uploaded image filename.
  + **Results Section:**
    - Space to display the processed image with the overlay.
    - Text area to display the recognized emotion.
* **Footer:**
  + A simple disclaimer/project status note (e.g., "Proof of Concept - Development in Progress").

**Key Considerations**

* **Minimalism:** Focus on the absolute essentials for the PoC.
* **Clear Feedback:** Provide visual cues when an image is processing and when results are ready.
* **Error Messages:** Have placeholders for handling unsuccessful uploads or face/emotion detection failures.

# Message Sequence Chart

**Before You Start**

* **Tools:** Get a dedicated diagramming tool (e.g., WebSequenceDiagrams, Draw.io, yUML) or a general one (e.g., Lucidchart, Visio) that supports MSC notation.
* **Review Requirements:** Refresh your understanding of the feature expectations outlined in your SRS, as these drive the interactions.

**MSC Creation Steps**

1. **Identify Entities:** The main entities in our proof of concept are:
   * **User:** Interacts with the frontend.
   * **Frontend (Web Interface):** Handles image upload and result display.
   * **Backend (Server):** Houses the image processing and analysis logic.
2. **Establish the Timeline:** An MSC has a vertical timeline. Place your entities as columns along the top.
3. **Core Interaction: Image Upload**
   * **User → Frontend:** An arrow labeled "Upload Image" with the image data from the User to the Frontend.
   * **Frontend → Backend:** Arrow labeled "Image Data" going to the Backend to initiate processing.
4. **Image Processing**
   * **Backend (Internal):** You *could* represent steps like landmark detection and emotion classification with messages within the Backend component. However, for a PoC, it's often sufficient to represent this visually with a block or note to convey that processing takes place.
5. **Result Transmission**
   * **Backend → Frontend:** Arrow labeled "Processed Image, Emotion" signifying the analysis result returned to the frontend.
6. **Result Display**
   * **Frontend (Internal):** Again, a note or block can indicate the frontend updating the display.

**Example (Simplified)**

This assumes a tool like WebSequenceDiagrams:

msc {

hscale="1.5";

User,Frontend,Backend;

User->>Frontend [label="Upload Image"];

Frontend->>Backend [label="Image Data"];

Backend -->> Backend [label="Image Processing"];

Backend->>Frontend [label="Processed Image, Emotion"];

Frontend -->> Frontend [label="Display Result"];

}

**MSC Considerations**

* **Granularity:** Since this is a PoC, keep the chart fairly high-level. As the product evolves, you may create more detailed MSCs focusing on specific modules.
* **Alternative Interactions:** Explore representing potential error conditions (e.g., wrong file format) for a more thorough model.
* **Tool-Specific Notation:** The syntax varies across MSC tools. Adapt these instructions accordingly.

**Why Create an MSC?**

* **Visualization:** Clearly depicts the sequential flow of information and responsibilities between the system's parts.
* **Communication:** Promotes common understanding useful for discussion with stakeholders and development team alignment.
* **Evolution:** Can be refined as development progresses, detailing internal operations and interactions.

Let me know if you'd like me to help flesh out more complex scenarios within the message sequence chart – Happy diagramming!