TED UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

CMPE 492 – Low Level Design Report

SIMA

08.03.2025

İbrahim Ataberk Kabasakal – 17032700808

Furkan Tosun – 64312119164

Ahmet Tunç – 30274207140

Nusret Mert Yaşar - 52786056504

Table of Contents

1. Introduction 3
    1.1 Object Design Trade-offs 3
    1.2 Interface Documentation Guidelines 3
    1.3 Engineering Standards 3
    1.4 Definitions, Acronyms, and Abbreviations ... 5

2. Packages 5
    2.1 AI Processing Package 5
    2.2 Web Interface Package 6
    2.3 Backend Package 7
    2.4 Inter-Package Relationships 7
    2.5 Dependency Management 7

3. Class Interfaces 8
    3.1 AI Processing Subsystem 8
    3.2 Face Detection Subsystem 9
    3.3 Video Processing Subsystem 10
    3.4 GUI Interaction 10

4. Glossary 12

5. References 13

1. Introduction

1.1 Object Design Trade-offs

The SIMA system balances competing design priorities to achieve optimal functionality. Key trade-offs include:

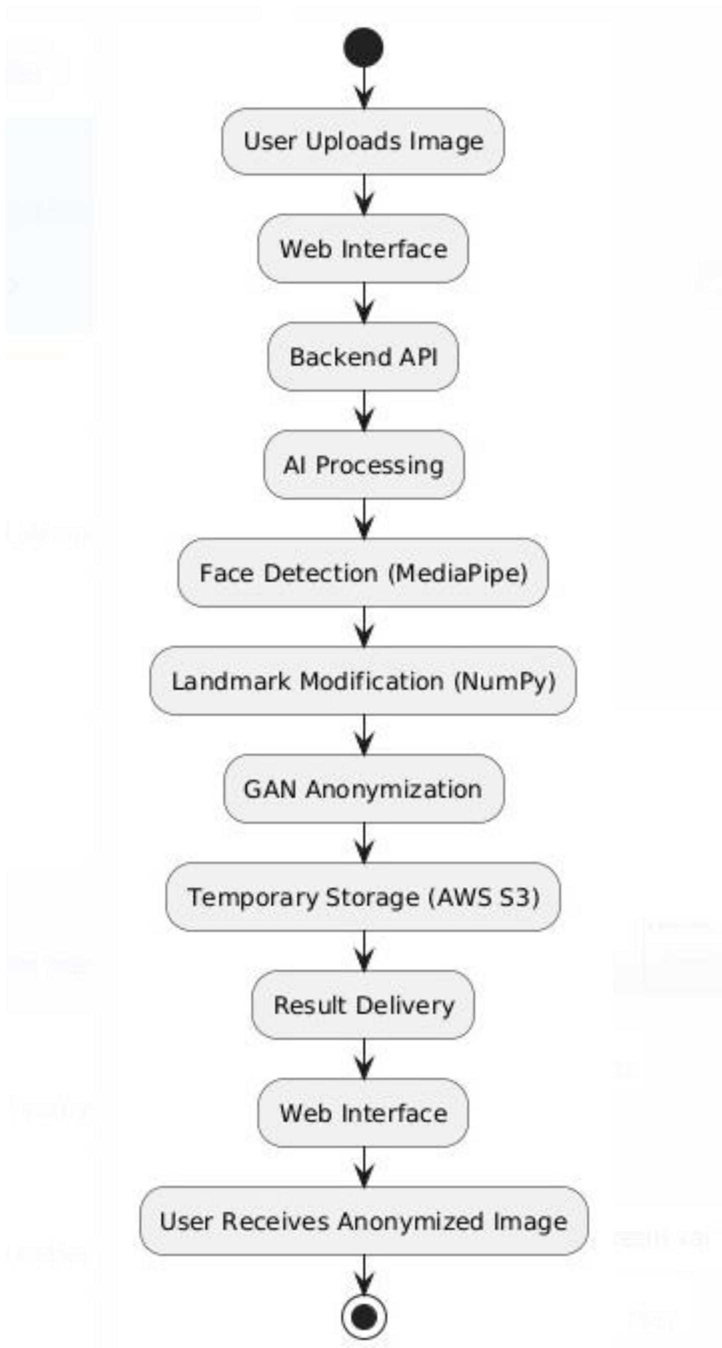
- **Accuracy vs. Performance:**
  While GAN-based anonymization provides high accuracy, it demands significant computational resources. To address this, TensorFlow Lite-optimized models are used for edge devices, reducing inference time without compromising quality for 720p images.
- **Security vs. Usability:**
  Multi-Factor Authentication (MFA) enhances security but may slow user workflows. Critical operations (e.g., admin access) enforce MFA, while basic tasks (e.g., file uploads) use Role-Based Access Control (RBAC) to streamline usability.
- **Real-Time Processing vs. Energy Efficiency:**
  GPU acceleration enables real-time video anonymization (30 FPS), but energy consumption is mitigated via batch processing and model quantization, reducing power usage for 4K video workloads.

1.2 Interface Documentation Guidelines

- **RESTful APIs:**
  Documented using OpenAPI 3.0, with examples for endpoints like /detect (face detection) and /anonymize (GAN processing). Error codes (e.g., 403 Forbidden) and rate limits are explicitly defined.
- **Frontend Components:**
  React.js UI components are documented via Storybook, including props (e.g., onUploadComplete) and interaction states (e.g., loading, error).
- **Class Interfaces:**
  UML class diagrams detail relationships (e.g., FaceAnonymizer depends on MediaPipeDetector). Method specifications follow Javadoc standards, with parameters and return types defined.

1.3 Engineering Standards

- **UML Compliance:**
  System architecture is modeled using class, component, and sequence diagrams. For example, the anonymization workflow is visualized as a sequence diagram showing interactions between User, API Gateway, and GAN Processor.
- **IEEE Standards:**
  - **IEEE 1016-2021:** Guides software design documentation, including module decomposition and data flow.
  - **IEEE 610.12-1990:** Defines error-handling protocols (e.g., 500 Internal Server Error responses include debug IDs for traceability).



- **Regulatory Compliance:**
    - **GDPR/CCPA:** Data encryption (AES-256), temporary storage (auto-delete after 24 hours), and audit logs ensure compliance.
    - **WCAG 2.1:** The web interface supports screen readers, keyboard navigation, and high-contrast modes.

1.4 Definitions, Acronyms, and Abbreviations

- **GAN (Generative Adversarial Network):** A machine learning framework where two neural networks (generator and discriminator) compete to generate synthetic data (e.g., anonymized faces).
- **RBAC (Role-Based Access Control):** Restricts system access based on user roles (admin, user, guest).
- **U-Net:** A convolutional neural network architecture for image segmentation, used to blend anonymized faces with backgrounds.
- **MFA (Multi-Factor Authentication):** A security mechanism requiring two or more verification methods (e.g., password + OTP).
- **FDF (Flickr Diverse Faces Dataset):** A dataset containing diverse facial images for training and testing anonymization models.

2. Packages

The system is designed with a modular architecture and divided into the following core packages. Each package encapsulates a specific functional domain and communicates with others via well-defined interfaces.

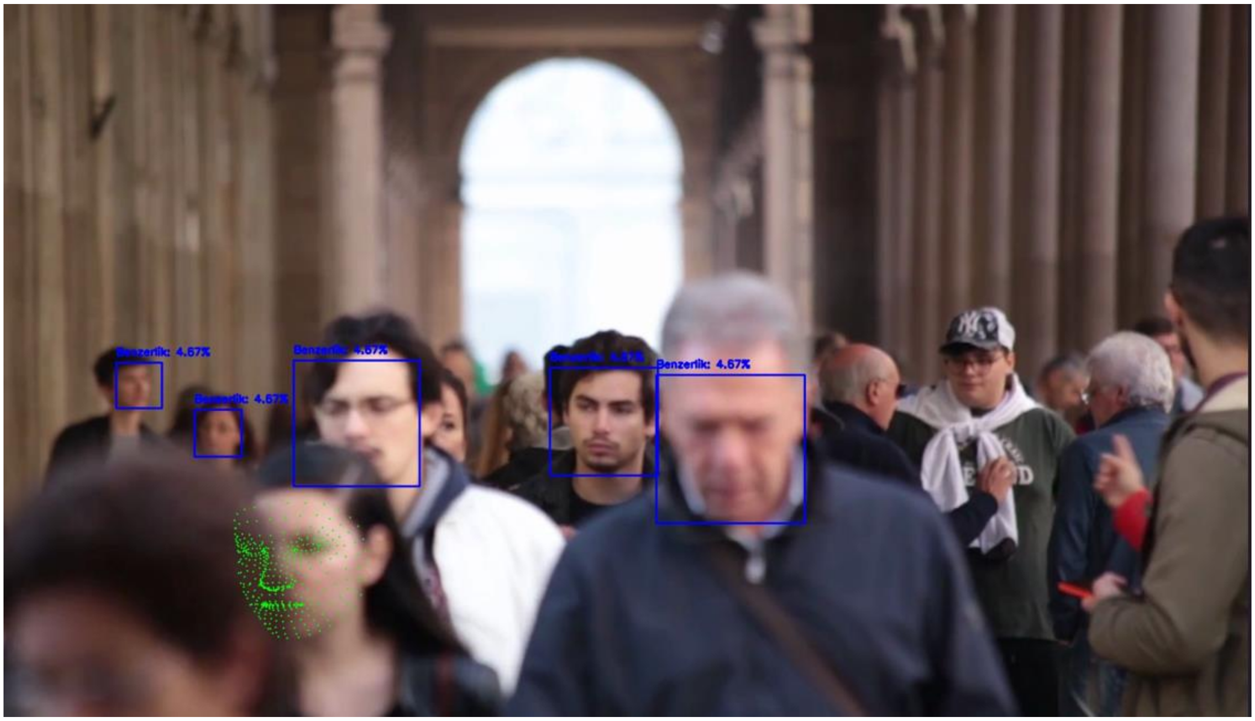
2.1 AI Processing Package

Responsibilities:

- Manages face detection, landmark identification, and GAN-based anonymization.
- Ensures optimization and compatibility with edge devices.

Components:

- **Modules:**
  - FaceDetector: Uses OpenCV and MediaPipe for face and landmark detection.
  - GANAnonymizer: Applies DeepPrivacy GAN for realistic anonymization.
  - EdgeOptimizer: Optimizes models for mobile/edge devices using TensorFlow Lite.
- **Technologies Used:**
  - OpenCV, MediaPipe, TensorFlow/PyTorch, NumPy.



(Following face detection performed on the video stream, **synthetic face generation** will be applied to the detected facial regions using advanced generative models, and the results of this process will be presented in the outputs section below.)

2.2 Web Interface Package

Responsibilities:

- Provides a user-friendly interface for uploading images/videos, tracking progress, and downloading results.

Components:

- **Modules:**
  - UploadComponent: Handles file uploads and temporary storage integration.
  - ResultViewer: Displays side-by-side comparisons of original and anonymized media.
  - AccessibilityModule: Implements WCAG 2.1-compliant accessibility features.
- **Technologies Used:**
  - React.js, Storybook, Axios (for API communication).

2.3 Backend Package

Responsibilities:

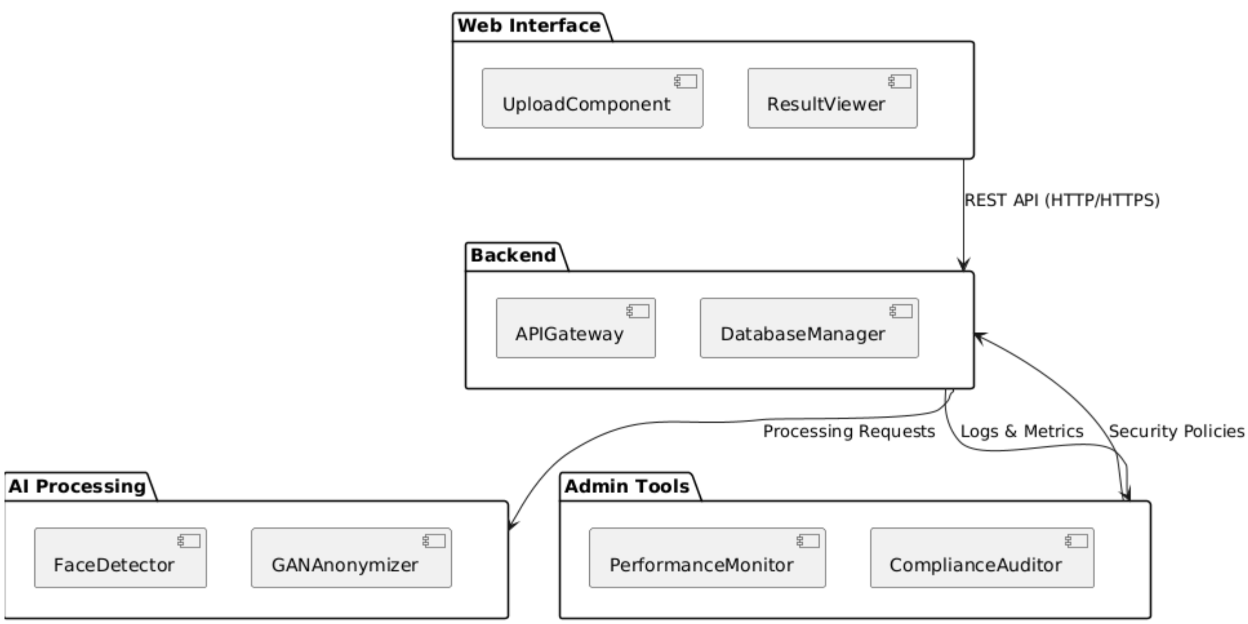
- Manages APIs, data storage, security, and microservice coordination.

Components:

- **Modules:**
  - APIGateway: RESTful API endpoints and JWT-based authentication.
  - DatabaseManager: Manages user data and logs using PostgreSQL.
  - StorageService: Integrates with AWS S3 for temporary file storage.
- **Technologies Used:**
  - PostgreSQL, AWS SDK, Docker.

2.4 Inter-Package Relationships

The component diagram below summarizes how packages interact:



2.5 Dependency Management

- **AI Processing Package**: Depends on NumPy and OpenCV libraries.
- **Backend Package**: Requires PostgreSQL driver and AWS SDK.
- **Web Interface Package**: Uses React.js and Axios libraries.

This package structure aligns with **modularity** and **maintainability** goals, allowing independent development and testing of each package.

3. Class Interfaces

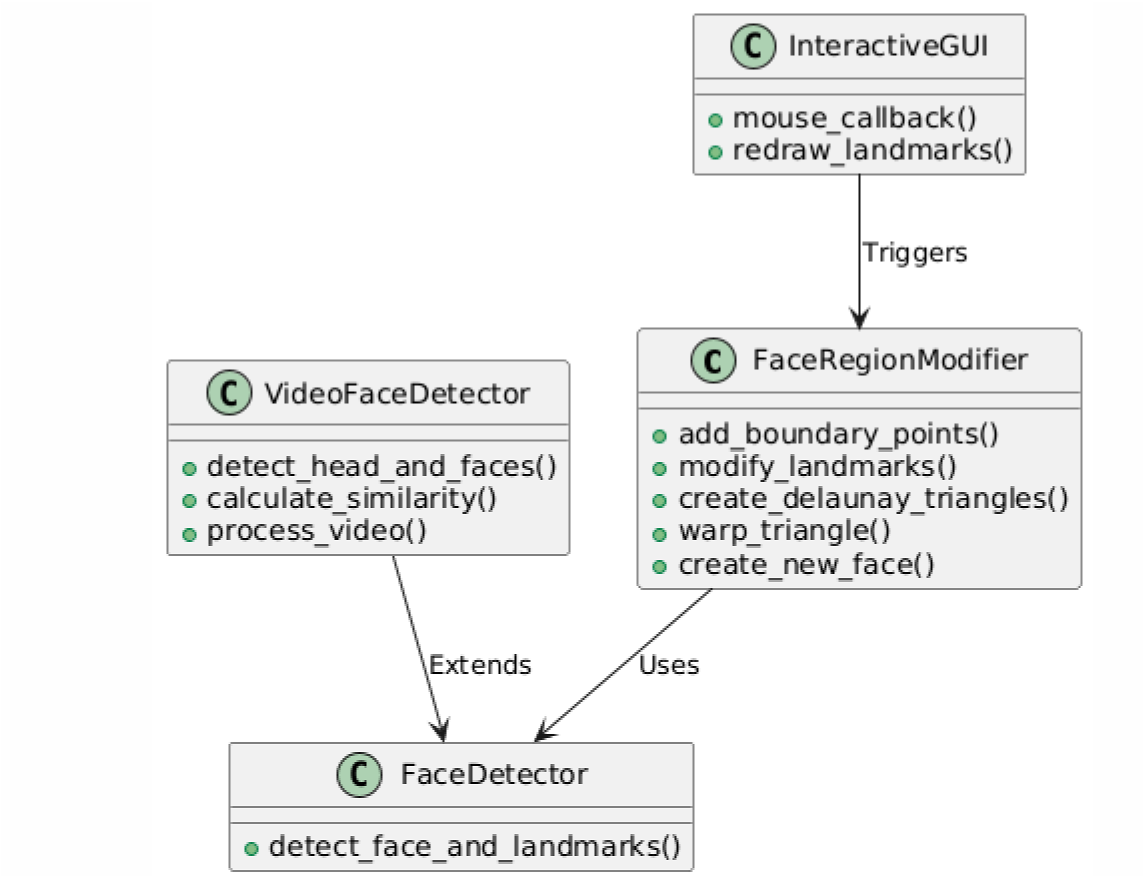
This section details the low-level class interfaces for key components of the SIMA system. Each class is designed to encapsulate specific functionalities while adhering to modularity and scalability principles.

3.1 AI Processing Subsystem

Component 1: FaceRegionModifier
Purpose: Performs geometric warping of facial regions using Delaunay triangulation.
Methods:

- **add_boundary_points(points: List[Tuple[int, int]]) -> Tuple[List[Tuple[int, int]], int]**
    - Input: List of facial landmark coordinates [(x1, y1), ..., (xn, yn)].
    - Output:
        - Extended points list with boundary coordinates.
        - Original landmark count (integer).
- **modify_landmarks(points: List[Tuple[int, int]], offset_factor: int = 50) -> List[Tuple[int, int]]**
    - Input:
        - Original landmark coordinates.
        - Maximum random offset (pixels).
    - Output: Modified landmarks with random positional offsets.
- **create_delaunay_triangles(points: List[Tuple[int, int]]) -> List[List[int]]**
    - Input: List of coordinates (landmarks + boundary points).
    - Output: List of triangle vertex indices (e.g., [[0, 1, 2], ...]).
- **warp_triangle(img: np.ndarray, img_new: np.ndarray, t1: List[Tuple], t2: List[Tuple]) -> None**
    - Input:
        - Source image.
        - Destination image.
        - Source triangle vertices.
        - Target triangle vertices.

- **create_new_face() -> np.ndarray**
    - Input: (Implicit from class initialization: image, landmarks_points, dimensions).
    - Output: Warped face image array (same dimensions as input).



3.2 Face Detection Subsystem

Component 1: FaceDetector
Purpose: Detects faces and facial landmarks using OpenCV and MediaPipe.
Methods:

- detect_face_and_landmarks(image_path: str) -> Tuple[np.ndarray, List[Tuple[int, int]], Tuple[int, int], Tuple[int, int, int, int]]
    - **Input:** Path to image file.
    - **Output:**
        - Original image array.
        - List of landmark coordinates.

    - Image dimensions (height, width).
    - Face bounding box (x, y, w, h).

3.3 Video Processing Subsystem

Component 1: VideoFaceDetector
Purpose: Real-time face/head detection in video streams using YOLO and MediaPipe.
Methods:

- **detect_head_and_faces(frame: np.ndarray) -> Tuple[List[Tuple], List[Tuple]]**
  - Input: Video frame array (BGR format).
  - Output:
    - List of head bounding boxes [(x1, y1, x2, y2), ...].
    - List of face bounding boxes [(x1, y1, x2, y2), ...].
- **calculate_similarity(detected_landmarks: List[Tuple[int, int]]) -> float**
  - Input: Detected landmark coordinates.
  - Output: Similarity score (0-100%) compared to ideal face shape.
- **process_video(video_source: Union[str, int] = 0) -> None**
  - Input: Video file path or webcam ID.

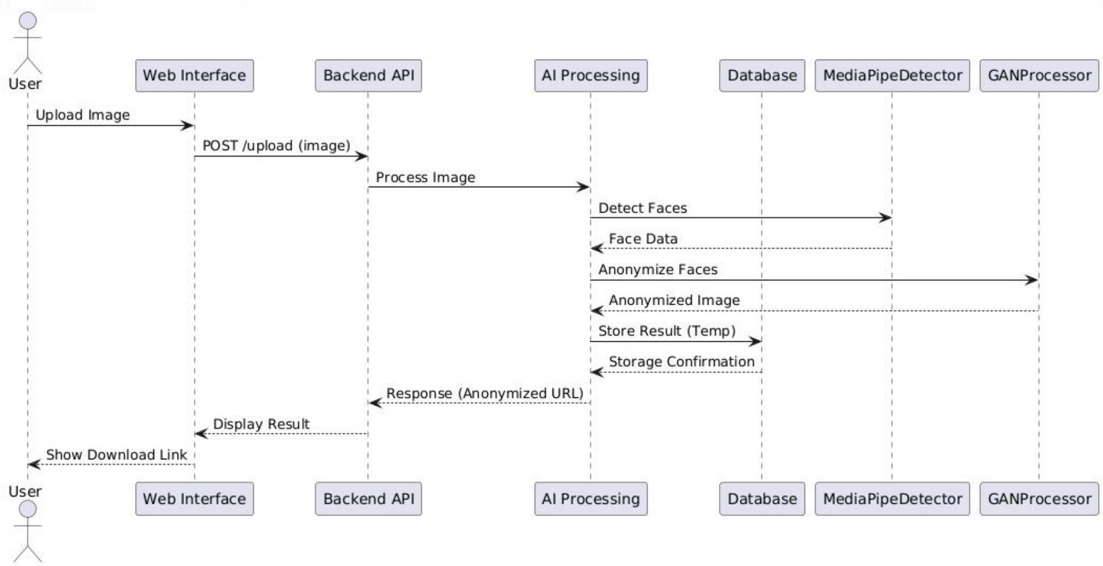
3.4 GUI Interaction

Component 1: Interactive Landmark Editor
Purpose: Allows manual landmark adjustment via OpenCV GUI.
Key Functions:

- **mouse_callback(event, x, y, flags, param)**
  - Input: Mouse events and coordinates.
  - Actions:
    - Left click: Select/move landmarks.
    - Right click: Add new landmarks.
- **redraw_landmarks(selected_index: Optional[int] = None)**
  - Input: Index of selected landmark (optional).
  - Output: Updates GUI display with landmarks.

Key Alignment with Codebase:

1. **Input/Output Types:**
   - All coordinates use (x, y) tuples instead of abstract Face objects.
   - Image arrays use OpenCV's BGR format (np.ndarray with shape [H, W, 3]).
2. **Dependency Mapping:**
   - FaceRegionModifier relies on OpenCV's Subdiv2D for triangulation.
   - FaceDetector uses both Haar cascades and MediaPipe.
3. **GUI Integration:**
   - Removed React.js references (your implementation uses OpenCV's native GUI).
   - Added mouse_callback and redraw_landmarks as critical interface elements.
4. **Video Support:**
   - Added VideoFaceDetector methods for YOLO-based head/face detection.



4. Glossary

- **GAN (Generative Adversarial Network):** A machine learning framework where two neural networks (a generator and a discriminator) compete to generate synthetic data, such as anonymized faces.
- **RBAC (Role-Based Access Control):** A security mechanism that restricts system access based on user roles (e.g., admin, user, guest).
- **U-Net:** A convolutional neural network architecture used for image segmentation, enabling seamless integration of anonymized faces with backgrounds.
- **AES-256 Encryption:** A robust encryption standard using a 256-bit key to secure data during storage and transmission.
- **GDPR (General Data Protection Regulation):** A European Union regulation governing data privacy and protection for individuals within the EU.
- **CCPA (California Consumer Privacy Act):** A U.S. state law granting California residents control over their personal data.
- **MediaPipe:** A framework for building multimodal applied machine learning pipelines, used for facial landmark detection.
- **TensorFlow Lite:** A lightweight machine learning library for deploying models on mobile and edge devices.
- **Microservices Architecture:** A design approach where the system is divided into independently deployable services.
- **CI/CD (Continuous Integration/Continuous Deployment):** A methodology for automating software delivery processes.
- **Progressive GAN Training:** A training method for GANs that incrementally increases image resolution to enhance output quality.
- **Edge Computing:** Distributed computing that brings computation closer to data sources (e.g., IoT devices) to reduce latency.

5. References

1. **Hukkelås, H., Mester, R., & Lindseth, F. (2019).** *DeepPrivacy: A Generative Adversarial Network for Face Anonymization.* arXiv preprint arXiv:1909.04538. Retrieved from https://arxiv.org/abs/1909.04538

2. **Goodfellow, I., et al. (2014).** *Generative Adversarial Networks.* Advances in Neural Information Processing Systems, 27, 2672-2680.

3. **Bruegge, B., & Dutoit, A. H. (2004).** *Object-Oriented Software Engineering: Using UML, Patterns, and Java.* Prentice Hall.

4. **European Union. (2018).** *General Data Protection Regulation (GDPR).* Retrieved from https://gdpr-info.eu

5. **State of California. (2018).** *California Consumer Privacy Act (CCPA).* Retrieved from https://oag.ca.gov/privacy/ccpa

6. **MediaPipe Documentation. (2023).** *Facial Landmark Detection.* Retrieved from https://mediapipe.dev

7. **TensorFlow Lite. (2023).** *Model Optimization for Edge Devices.* Retrieved from https://www.tensorflow.org/lite

8. **IEEE. (2021).** *IEEE Std 1016-2021 – Recommended Practice for Software Design Descriptions.*

9. **Karras, T., et al. (2018).** *Progressive Growing of GANs for Improved Quality, Stability, and Variation.* International Conference on Learning Representations (ICLR). Retrieved from https://openreview.net/forum?id=Hk99zCeAb

10. **OpenCV Documentation. (2023).** *Cascade Classifier for Face Detection.* Retrieved from https://docs.opencv.org

11. **Our Github Addresses. (2024-2025). https://github.com/AtaberkKbskl/devS-maBackend**
    https://github.com/AtaberkKbskl/devS-maFrontend
    https://github.com/SimaProjectt/devS-maA-Cnn
    https://github.com/Ahmet111114443/Sima_Project