CHAPTER 3: METHODOLOGY

3.1 Introduction

The methodology is a key part in any key project, where we decide the ways and means, and then the technologies with which we want to build our projects. This chapter defines the system Architecture design process, development techniques, the functional and non-functional requirements, and a valid and vivid visual representation of the system and its use cases in the form of all sorts of diagrams. Kothari (2004) Emphasizes that methodology is "the systematic way to solve a problem," including the logic behind methods.

3.2 System Development Methodology

The earliest systematic studies on the development of face recognition abilities were conducted by Goldstein and Chance, who tested school children across different age groups (as cited in Johnston & Ellis, 1995). In these experiments, children were first shown unfamiliar faces and later asked to identify them from a larger set. This aligns with the common observation that young children often make category-inclusive errors in facial recognition, much to their parents' embarrassment.

Johnston and Ellis (1995) explain that the origin of the multidimensional face space lies in the central tendency of its dimensions, with facial features varying normally around this point. To quantify distinctiveness, correlation coefficients were calculated between each face's linear distinctiveness rating and its distance from the origin in the dimensional space. The authors argue that an exemplar-based model best explains the development of this face space.

This understanding of facial recognition development parallels modern deep learning approaches. So, in development we basically needed a facial recognition model to handle these computations and then later build our whole attendance system around that model.

The development step included several phases.

Phase1(System Design and Architecture): Here we planned out how the attendance system was going to function and interact with our users, primarily the flow of data from one point to another. Here we landed on an architecture that would log already registered users in and then redirect new users to register for our system and have their information stored in our database. Now for registered users, once their faces are detected, the attendance system performs a verification function on their input image and then logs the attendance for the user if the user exists in the database.

PHASE2(Dependencies and Tech stack): Once we had the system architecture down, we had to choose a stack to work with, install all our dependencies, and set up our files and folder structure key software dependencies to mention include python, OpenCV, ageitgey's deep learning facial recognition model, and dlib King (2009) Dlib's facial recognition. which is written in C++.

PHASE3: Importing the facial recognition model successfully into our workspace and setting key variables that we would work with in the future.

PHASE4: This phase mainly entailed the attendance logic, where we define our various functions in units like our log in function and our matching logic, we also investigated timestamps, and thresholding to avoid as many false positives as possible.

PHASE5(Testing): Now we had to validate our system's accuracy with fresh unknown faces. Here we encountered the confusion matrix. We would also find that the system encountered some issues with light sensitivity, and dataset size bias. We would also learn about ethical concerns when dealing with facial recognition

3.3 Crystallization of the Problem

Attendance has become a large part of our society, the school system uses it, the work system uses it, and it is used by everyone everywhere even in ways that may seem abstract. From a software Engineer logging all the active sessions in their application to the participant count on say a TikTok live feed, we use attendance in our everyday life.

For all the good that attendance has been able to accomplish, there are some issues that come to mind, for one it takes way longer than it should in taking attendance, especially in professional environments, How many times have we got to work only to join a long queue of people all in an attempt to sign our attendance sheets? And how many times have u gotten stuck after work or school just waiting for your turn so u could go home? Now we can't avoid attendance taking because it has its importance, that is why we decided to build a system with Kantanka Financial Co-operative Society who also happen to face these exact challenges in hopes to streamline the whole attendance process.

3.4 Requirements of the Proposed System

A system is only as good as its foundation, and for this facial recognition model, the requirements set the limits and goals for what it should achieve. These requirements make sure the system is not just working in theory but also useful in real-world situations. Since deep learning models can be complex and facial recognition needs to work in real-time, the system must meet certain functional and non-functional requirements to be effective.

The system is designed to process facial images, extract key features, compare them to a trained what is stored in the database, and identify faces accurately. But it's not just about recognition or verification, it also needs to be fast, scalable, and secure. The requirements are divided into two categories:

- **Functional Requirements** – What the system must do (like detecting and recognizing faces).
- **Non-Functional Requirements** – How well the system must perform (like speed, accuracy, and security).

3.4.1 Functional Requirements

Functional requirements outline the key tasks the system must perform to achieve accurate and reliable facial recognition. These tasks ensure the attendance system works as expected. The main functional aspects include:

Image Acquisition and Input Handling

The system should accept images from different sources, like live camera feeds and the device's native webcam
It must support common image formats (JPEG, PNG, BMP) and ensure they are compatible with processing.

Preprocessing and Feature Extraction

Input images should be resized, normalized, and, when needed, augmented to improve model performance.
The system must use Convolutional Neural Networks (CNNs) to extract facial features and convert them into numerical representations.

Model Training and Learning

The model should be trained using deep learning (TensorFlow/Keras) on a labeled dataset while reducing overfitting.
Data augmentation should be supported to help the model generalize better, especially when data is limited.

Face Detection and Recognition

The attendance system must detect and identify faces in images or video feeds by matching input feed with what it has stored in the database
It should do a good job to handle tricky cases like poor lighting, different head angles, and partially hidden faces.

Database and Storage Management

A structured database (SQL) should store facial data and related metadata for easy access.
The system should retrieve stored data quickly to support real-time or near-real-time recognition.

Integration with Front-End and Back-End

The system should provide APIs or direct integration for front-end applications.
A web or preferably desktop interface should allow users to upload images, see recognition results, and manage records.

Security and Access Control

Strict authentication and authorization must be in place to prevent any false attendance as it beats the whole purpose for an attendance system.
Stored facial data should follow ethical and legal standards, with encryption if needed for extra security.

3.4.2 Non-Functional Requirements

Aside from functionality, the system also needs to meet performance, usability, and scalability requirements to work well in real-world situations. These non-functional requirements determine how efficiently and effectively the system operates.

Performance and Accuracy

- The system should achieve at least **90% accuracy** on benchmark datasets.
- It should process images quickly, ideally within **500ms per recognition task**.
- False positives and false negatives should be minimized through proper model tuning.

Scalability and Extensibility

- The system should handle **larger datasets** and more users over time without slowing down or even crashing.
- It should be flexible enough to support **future improvements (version control)**, like combining facial recognition with fingerprint or voice authentication.

Usability and User Experience

- The **interface should be simple and easy to use**, even for first-time users.
- If errors occur (e.g., low-quality images), the system should give **clear feedback,** so users know how to fix them.

Security and Data Privacy

- Stored facial data should be **encrypted** to prevent unauthorized access.
- The system must follow **legal guidelines** when handling biometric data.

Reliability and Availability

- The system should be available **99.9% of the time** under normal conditions.
- It should have backup options, like **cloud storage**, to prevent data loss from hardware failures.

Compatibility and Integration

- It should work on different platforms, including **Windows, Linux, and cloud-based systems**.
- APIs should follow **standard RESTful formats** so they can easily connect with other applications.

Ethical Considerations and Bias Mitigation

- The model should be tested on **diverse datasets** to avoid racial, gender, or age bias.
- Ethical AI principles should be followed to **prevent misuse**, such as unethical surveillance.

These requirements help create a **reliable, secure, and fair** facial recognition system. By balancing strong functional features with well-defined non-functional goals, the system can perform efficiently and be ready for real-world use.

3.4.3 Software Requirements

- The effectiveness of a facial recognition system depends a lot on the software stack behind it. Every part from the machine learning framework to the back-end API needs to be carefully selected to ensure good performance, scalability, and easy maintenance. The software requirements outline the key technologies used to make the system work smoothly.

- ### 1. Programming Languages and Frameworks

- Python – Primary language for its rich ecosystem in deep learning and computer vision.
- TensorFlow/Keras – Powers neural network training/inference with GPU acceleration.
- OpenCV – Handles real-time image processing, video frame capture, and face detection.

- Dlib – Advanced facial landmark detection and feature extraction.
- face_recognition (built on dlib) – Simplifies face encoding and recognition tasks.
- Pillow – Assists in image preprocessing and manipulation.

2. Development and Execution Environments

- **Jupyter Notebook**: Primary environment for model experimentation, debugging, and visualization.
- **Google Colab**: Used for cloud-based training, leveraging free GPU resources to accelerate computation.
- **Local Development (VS Code)**: For integrating the trained model with the full system stack.

4. Version Control and Deployment Tools

- **GitHub / GitLab**: For version control, ensuring collaborative development and code backup.
- **Heroku / AWS** : Potential cloud deployment platforms for scaling the recognition service.

5. Security Considerations

- **SSL/TLS Encryption**: Ensures secure communication between the client, server, and database.
- **OAuth / JWT Authentication**: Provides role-based access control for managing user permissions.

By leveraging these technologies, the system remains **scalable, and efficient**, ensuring smooth development and deployment.

3.5 Design of the System

The design phase is where ideas are put together to build a working system. Everything, from how data moves to how users interact with the system, needs to be planned properly to avoid problems. A good design makes sure all parts of the system work well together for smooth facial recognition.

The system is built in separate parts, each handling a specific task but still working together as a whole.

3.5.1 System Architecture

The architecture follows a **three-layered design**, ensuring separation of concerns:

1. **Data Processing Layer** (Model Training & Preprocessing)
   a. Handles image preprocessing, data augmentation, and transformation.
   b. Facilitates model training, evaluation, and storage.
2. **Application Layer** (Face Recognition & API)
   a. Receives image input, processes embeddings, and performs classification.
   b. Exposes endpoints for front-end interaction (e.g., API call for face matching).
3. **Presentation Layer** (Front-End & User Interface)
   a. Provides a UI for users to upload images, view results, and manage data.
   b. Displays system feedback, such as recognition success/failure messages.

3.5.2 Data Flow & Processing Pipeline

The system follows a structured pipeline to ensure efficient face recognition:

1. **User uploads an image (or video frame is captured in real-time).**
2. Preprocessing:
3. Resizing & normalization to uniform format.
4. Face detection using OpenCV/Dlib.
5. Feature extraction (converting faces to embeddings).
6. Trained deep learning model processes embeddings.
7. Model predicts identity (or returns "unknown").
8. Results sent via API and displayed on frontend.

This **structured data flow** ensures that processing remains efficient, whether handling a **single image, batch processing, or real-time detection.**

3.5.3 User Interaction & Interface Design

- The user interface (UI) is designed for **efficiency, clarity, and ease of use**. The front-end must:
  . Allow users to **upload an image** and receive instant recognition feedback.
  . Display **real-time webcam recognition**, showing matched identities.
  . Provide an **admin panel** for managing stored faces and dataset updates.
  . Offer a **responsive design**, adapting to both desktop and mobile screens.

3.5.4 Security & Performance Considerations

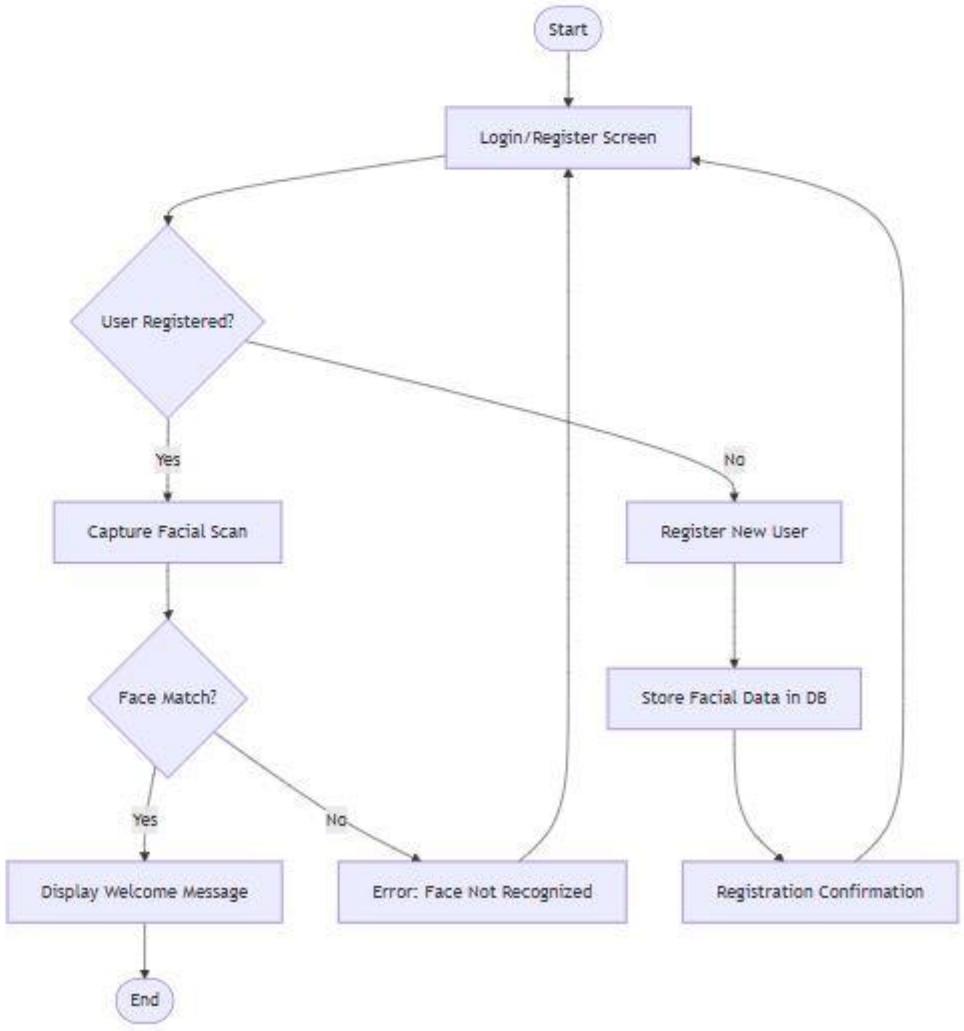
Data Privacy Measures:

- Facial embeddings are stored in an **encrypted format** to prevent unauthorized access.
- The system avoids **sending raw facial data over networks**, instead using hashed representations.

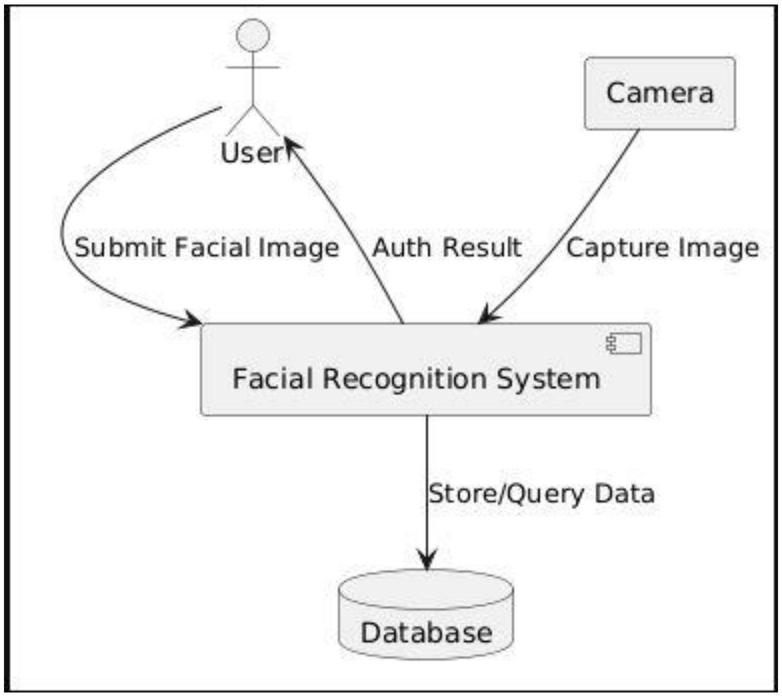
Performance Optimization:

- Model inference is optimized for **low latency (<500ms per recognition request)**.
- **GPU acceleration (TensorFlow/Keras)** ensures that the recognition system remains **scalable and efficient**.

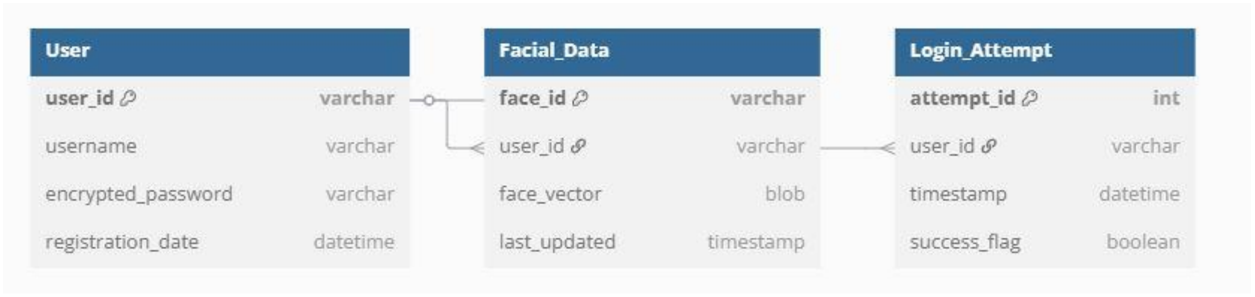
3.5.5 Flowchart



3.5.6 Context Diagram



3.5.7 Entity Relationship Diagram.

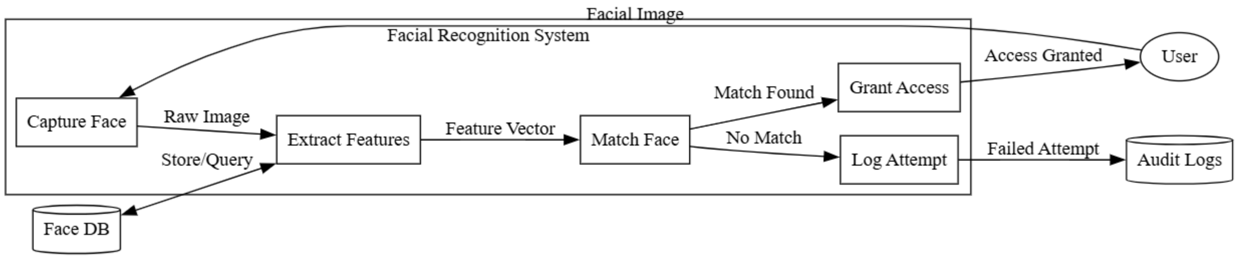


User	
user_id	varchar
username	varchar
encrypted_password	varchar
registration_date	datetime

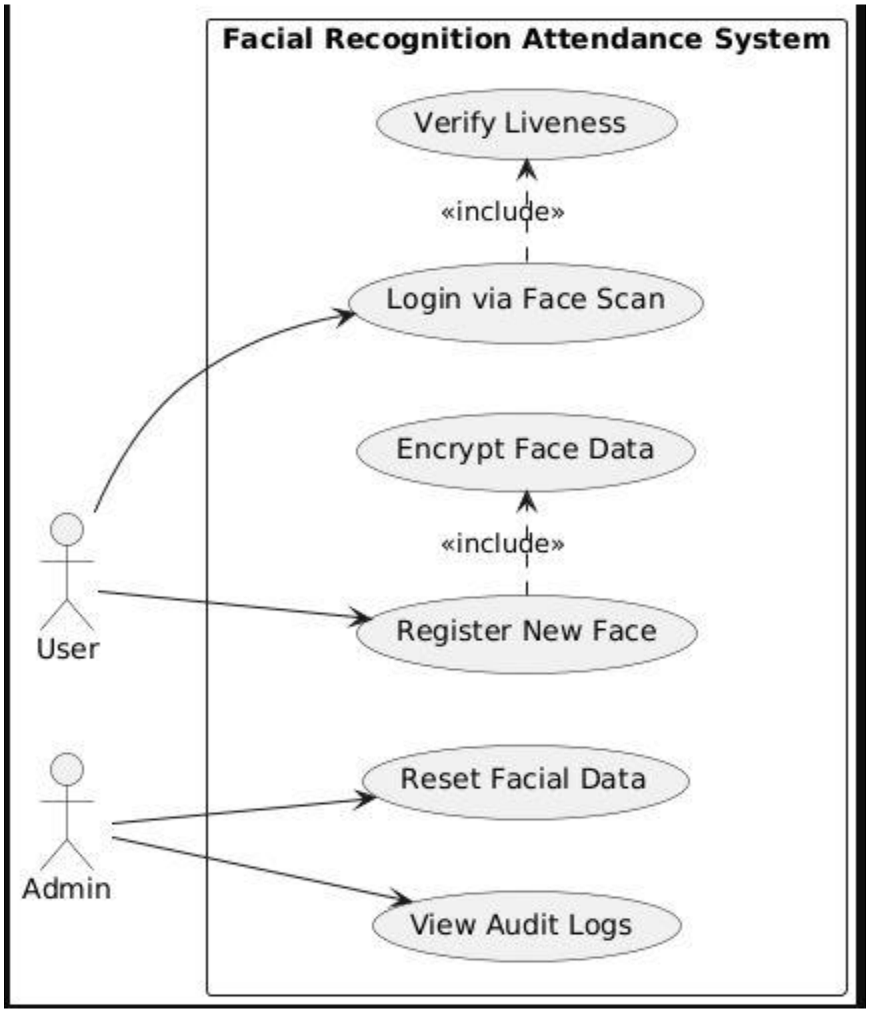
Facial_Data	
face_id	varchar
user_id	varchar
face_vector	blob
last_updated	timestamp

Login_Attempt	
attempt_id	int
user_id	varchar
timestamp	datetime
success_flag	boolean

3.5.8Data Flow



3..5.9 Use Case Diagram



Chapter Summary

This chapter outlined the systematic methodology for developing the facial recognition attendance system. The system development phases progressed from architectural design to model integration, leveraging Python and OpenCV for real-time face detection and verification. Functional requirements emphasized core capabilities like image preprocessing, CNN-based feature extraction, and secure database management, while non-functional requirements addressed accuracy (>90%), scalability, and ethical bias mitigation. The three-layer architecture (data processing, application, presentation) ensured modularity, with data flows optimized for low-latency recognition (<500ms). Also images were provided to help better visualize the system and its capabilities

Key challenges included lighting sensitivity and dataset bias, addressed through augmented training and threshold tuning. The proposed solution aligns with Kantanka Financial Co-operative Society's need for efficient attendance tracking, balancing technical robustness with usability. Subsequent chapters will detail implementation and validation of this methodology.