Acknowledgment

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to our mentor, **Dr. Sharmishta Desai**, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

Furthermore, I would also like to acknowledge with much appreciation the crucial role of the staff of MIT WPU, who gave the permission to use all required equipment and the necessary materials to complete the task. A special thanks goes to my team mates,who helped me enormously to assemble the parts and gave suggestion about the task of using the techniques of measurements.

I have to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advices.

I would also like to thank my parents for their wise counsel and sympathetic ear. You are always there for me. Finally, I wish to thank my friends for their support and encouragement throughout my study.

Name of Students

Krishnaraj Thadesar, 1032210888
Parth Zarekar, 1032210846
Sourab Karad, 1032211150
Saubhagya Singh, 1032211144

Abstract

The Attendance-Assistant project introduces an innovative, cross-platform mobile application designed to automate attendance tracking in educational institutions using advanced facial recognition technology. The system leverages Flutter for the frontend, ensuring a seamless and intuitive user experience, while the backend is powered by FastAPI and MongoDB, providing robust, secure, and scalable data management. This comprehensive solution streamlines the entire attendance workflow, encompassing user authentication, face-encoding management, real-time attendance marking, and detailed reporting.

This report documents the systematic approach undertaken to evaluate and compare various face recognition algorithms, focusing on their precision metrics, computational efficiency, and real-world applicability. After rigorous testing, the ResNet-based implementation in Python, utilizing the *face_recognition* library, was selected for its superior accuracy and reliability. The report further elaborates on the system architecture, database schema, and core functionalities, highlighting how the Attendance-Assistant enhances operational efficiency, accuracy, and user convenience compared to traditional attendance methods.

The project also explores the integration of IoT devices, such as Raspberry Pi, to enable real-time facial recognition in classrooms and other institutional settings. By combining cutting-edge technologies with a user-centric design, the Attendance-Assistant aims to revolutionize attendance management, offering a scalable and efficient solution for modern educational institutions.

Keywords

Facial Recognition, Attendance Automation, Flutter, FastAPI, MongoDB, ResNet, Educational Institutions, IoT, Raspberry Pi, Backend Development, Frontend Development, Data Integrity, Firebase, Precision Metrics, Attendance Management System

List of Figures

4.1 App Design 20
4.2 App Design 21

5.1 Block Diagram 23
5.2 Activity Diagram 25
5.3 Activity Diagram Continued 26

6.1 Project Timeline 27

7.1 Cropped Faces using OpenCV Haar Cascades 34
7.2 Cropped Faces using OpenCV Haar Cascades 34
7.3 Cropped Faces using OpenCV Haar Cascades 35
7.4 API Endpoints for Attendance details 35
7.5 API Endpoints for Student details 36
7.6 API Endpoints for Schools and Specialization details 36
7.7 API Endpoints for Panel details 36
7.8 API Endpoints for Rooms and Buildings details 37
7.9 API Endpoints for Subjects and Semesters details 37
7.10 API Endpoints for Teacher details 37
7.11 API Endpoints for Lecture details 38

8.1 Krishnaraj's Segregated Images 43
8.2 Saubhagya's Segregated Images 43
8.3 Parth's Segregated Images 44
8.4 Sourab's Segregated Images 44
8.5 Abhijeet's Segregated Images 45

10.1 Confusion Matrix of LBPH Face Recognizer 48
10.2 Accuracy per Person (Class) - LBPH Face Recognizer 48
10.3 Accuracy per Person (Class) - LBPH Face Recognizer 49
10.4 Confusion Matrix of Facenet Face Recognizer 50
10.5 Accuracy per Person (Class) - Facenet Face Recognizer 50
10.6 Accuracy per Person (Class) - Facenet Face Recognizer 51
10.7 Confusion Matrix of Resnet Face Recognizer 52
10.8 Accuracy per Person (Class) - Resnet Face Recognizer 52
10.9 Accuracy per Person (Class) - Resnet Face Recognizer 53
10.10 Final Model Comparison 53

14.1 Block Diagram highlighting the modules supported by Parth Zarekar. 60
14.2 Swagger UI for API documentation (Krishnaraj Thadesar's contribution). 61
14.3 Swagger UI for API documentation (Krishnaraj Thadesar's contribution). 62
14.4 Swagger UI for API documentation (Krishnaraj Thadesar's contribution). 62

15.1 MongoDB Collections (Parth Zarekar's area) 65
15.2 MongoDB Collections (Parth Zarekar's area) 65
15.3 MongoDB Collections (Parth Zarekar's area) 66
15.4 MongoDB Collections (Parth Zarekar's area) 66

16.1 Accuracy per Person (Class) - LBPH Face Recognizer 69
16.2 Accuracy per Person (Class) - Facenet Face Recognizer 70
16.3 Accuracy per Person (Class) - Resnet Face Recognizer 70
16.4 Final Model Comparison (Sourab Karad's results) 71

17.1 Frontend (Saubhagya Singh's contribution). 73

List of Tables

4.1 List of Hardware Requirements 15

7.1 Training Data Images 32
7.2 Training Data Images (Continued) 33

Contents

1 Introduction **9**
- 1.0.1 Problem Statement . . . 9
- 1.0.2 Need of the Project . . . 9

2 Literature Survey **10**
- 2.1 Paper 1 . . . 10
  - 2.1.1 Positives and Learnings from this Paper . . . 10
  - 2.1.2 Identified Research Gaps . . . 10
- 2.2 Paper 2 . . . 10
  - 2.2.1 Positives and Learnings from this Paper . . . 10
  - 2.2.2 Identified Research Gaps . . . 11
- 2.3 Paper 3 . . . 11
  - 2.3.1 Positives and Learnings from this Paper . . . 11
  - 2.3.2 Identified Research Gaps . . . 11
- 2.4 Paper 4 . . . 11
  - 2.4.1 Positives and Learnings from this Paper . . . 11
  - 2.4.2 Identified Research Gaps . . . 12
- 2.5 Paper 5 . . . 12
  - 2.5.1 Positives and Learnings from this Paper . . . 12
  - 2.5.2 Identified Research Gaps . . . 12

3 Problem Statement **13**
- 3.1 Project Scope . . . 13
- 3.2 Project Assumptions . . . 13
- 3.3 Project Limitations . . . 14
- 3.4 Project Objectives . . . 14

4 Project Requirements **15**
- 4.1 Requirements . . . 15
  - 4.1.1 Hardware Requirements . . . 15
  - 4.1.2 Software Requirements . . . 15
- 4.2 Risk Management . . . 18
- 4.3 Functional Specifications . . . 18
  - 4.3.1 Interfaces . . . 18
  - 4.3.2 Interactions . . . 21

5 System Analysis Proposed Architecture **23**
- 5.1 Block Diagram . . . 23
- 5.2 Activity Diagram . . . 25

6 Project Plan - Timeline **27**
6.1 Phase 1: Research . . . 27
6.2 Phase 2: Requirements Gathering . . . 27
6.3 Phase 3: System and Architecture Design . . . 28
6.4 Phase 4: Development . . . 28
6.5 Phase 5: Testing and Quality Assurance . . . 28
6.6 Phase 6: Deployment and Training . . . 28
6.7 Phase 7: Closure and Documentation . . . 29

7 Implementation **30**
7.1 Methodology . . . 30
7.2 Advantages . . . 30
7.3 Implementations . . . 31
7.3.1 Platform . . . 31
7.3.2 Training Data . . . 31
7.3.3 Creation of Dataset . . . 33
7.4 API Endpoints . . . 35
7.4.1 Attendance API Endpoints . . . 35
7.4.2 Student API Endpoints . . . 36
7.4.3 Schools and Specialization API Endpoints . . . 36
7.4.4 Panel API Endpoints . . . 36
7.4.5 Rooms and Buildings API Endpoints . . . 37
7.4.6 Subjects and Semesters API Endpoints . . . 37
7.4.7 Teacher API Endpoints . . . 37
7.4.8 Lecture API Endpoints . . . 38
7.5 Model Training . . . 38
7.5.1 Code Snippet for face recognition using LBPH . . . 38
7.5.2 Code Snippet for face recognition using face_net . . . 39
7.5.3 Code Snippet for face recognition using res_net . . . 40

8 Performance Evaluation and Testing **42**
8.1 Evaluation Metrics . . . 42
8.2 Testing Methodology . . . 42
8.3 Testing and Training Dataset . . . 43

9 Deployment Strategies **46**
9.1 Deployment Environment . . . 46
9.2 Deployment Steps . . . 46
9.3 Deployment Challenges . . . 46
9.4 Security Concerns . . . 47

10 Result and Analysis **48**
10.1 Model 1: LBPH Face Recognizer . . . 48
10.1.1 Confusion Metrics . . . 48
10.1.2 Per Person Results . . . 48
10.2 Model 2: Facenet Face Recognizer . . . 50
10.2.1 Confusion Metrics . . . 50
10.2.2 Per Person Results . . . 50
10.3 Model 3: Resnet Face Recognizer . . . 52
10.3.1 Confusion Metrics . . . 52
10.3.2 Per Person Results . . . 52
10.4 Final Comparison . . . 53

11 Applications **54**
11.1 Educational Institutions . . . 54
11.2 Corporate Environments . . . 54
11.3 Healthcare Facilities . . . 54
11.4 Government and Public Sector . . . 54
11.5 Retail and Hospitality . . . 55
11.6 Event Management . . . 55
11.7 Smart Cities and IoT Integration . . . 55
11.8 Research and Development . . . 55

12 Conclusion **56**

13 Future Prospects for the Attendance Assistant **57**
13.1 Integration of advanced Deep-Learning Models . . . 57
13.2 Dataset Expansion and Synthetic Augmentation . . . 57
13.3 Edge Deployment and Resource Optimization . . . 57
13.4 Privacy, Security, and Ethical Considerations . . . 58
13.5 Mutlti-Modal and Context-Aware Fusion: . . . 58
13.6 Broader Applications and Commercialization . . . 58

Bibliography **59**

14 Individual Contribution **60**
14.1 Problem Statement . . . 61
14.2 Student Details . . . 61
14.3 Module Title . . . 61
14.4 Project's Module Scope (Individual Perspective) . . . 61

15 Individual Contribution **64**
15.1 Problem Statement . . . 64
15.2 Student Details . . . 64
15.3 Module Title . . . 64
15.4 Project Module Scope . . . 64
15.5 Project Modules – Individual Contribution . . . 66

16 Individual Contribution **68**
16.1 Problem Statement . . . 68
16.2 Student Details . . . 68
16.3 Module Title . . . 68
16.4 Project Module Scope . . . 68
16.5 Project Modules – Individual Contribution . . . 71

17 Individual Contribution **72**
17.1 Problem Statement . . . 72
17.2 Student Details . . . 72
17.3 Module Title . . . 72
17.4 Project Module Scope . . . 72
17.5 Project Modules – Individual Contribution . . . 73

A Publication Details **74**

B Base Paper **75**

C Plagiarism Report **76**

Chapter 1

Introduction

Face recognition is a biometric technology that utilizes distinctive features of the face to identify individuals. Widely employed in security systems, it serves various applications such as access control, attendance tracking, and surveillance. While face recognition has existed for decades, recent advancements in machine learning and computer vision have significantly enhanced its accuracy and reliability. Numerous algorithms and techniques are available, each with unique strengths and weaknesses. In this seminar, we aim to compare popular face recognition algorithms and assess their performance using a standardized dataset.

1.0.1 Problem Statement

We need to compare the various face recognition algorithms and techniques to determine which one is the most accurate and efficient. We also need to discuss the implementation of these algorithms in real-world applications.

1.0.2 Need of the Project

- The motivation for this topic came from impending research for a Project titled "Machine Learning Powered Automated Facial Attendance Tracking System".
- The project aims to develop a system that can automatically track attendance using facial recognition technology.
- To achieve this goal, it is essential to understand the different face recognition algorithms and techniques available and evaluate their performance to identify the most suitable approach for the project.
- By comparing the performance of different face recognition algorithms and techniques, we can gain insights into their strengths and weaknesses and make informed decisions about which approach to use for the project.
- This seminar will provide a comprehensive overview of the most popular face recognition algorithms and techniques and evaluate their performance on a common dataset to help guide the development of the attendance tracking system.

This project will help in understanding the various face recognition algorithms and techniques and how they can be implemented in real-world applications. It will also help in understanding the challenges faced in face recognition and how these challenges can be overcome.

To use the correct method and library in finding attendance, so as to reduce time and cost, while also maintaining high levels of accuracy, it was necessary to compare the various face recognition algorithms and techniques.

Chapter 2

Literature Survey

2.1 Paper 1

Title: *"A Comparative Study of Facial Recognition Techniques: With focus on low computational power."* Author: *Schenkel, T., Ringhage, O. and Branding, N.* [7]

2.1.1 Positives and Learnings from this Paper

1. The publication compares five performance metrics, including recall and F-score, providing a comprehensive evaluation of facial recognition techniques.
2. It addresses the importance of balancing low computational time and prediction ability for security systems, offering practical guidelines for implementation.
3. The research questions are clearly defined, focusing on significant differences in performance, training time, and prediction time among different facial recognition techniques and classifiers.

2.1.2 Identified Research Gaps

1. The document lacks detailed information on the specific facial recognition techniques and classifiers used in the experiments.
2. It does not provide a detailed breakdown of the dataset used for training and testing the facial recognition models.
3. While the document mentions the comparison of results, it does not delve into the specific findings or implications of these comparisons.

2.2 Paper 2

Title: *"A Comparative Study on Facial Recognition Algorithms"* Author: *Sanmoy Paul and Sameer Acharya* [8]

2.2.1 Positives and Learnings from this Paper

1. Comparative Analysis: The study provides a comparative analysis of different facial recognition algorithms, allowing developers to make informed choices based on recognition accuracies.
2. Algorithm Selection: By studying the advantages and disadvantages of various algorithms, developers can select the best facial recognition algorithm for their specific implementation needs.

3. Future Improvements: The research suggests future efforts to test on a larger set of images to enhance the accuracy of CNN and explore combining multiple machine learning classification algorithms for increased recognition accuracy and handling large datasets.

2.2.2 Identified Research Gaps

1. The document lacks detailed discussion on the specific methodologies used for training and testing the algorithms, which could provide more clarity on the experimental setup.
2. There is no mention of the computational resources or hardware specifications used for running the experiments, which could impact the reproducibility and scalability of the results.
3. The publication does not delve into the potential biases or limitations in the dataset used for training and testing the facial recognition models, which could affect the generalizability of the findings.

2.3 Paper 3

Title: *"A comparison of facial recognition algorithms."* Author: *Delbiaggio, Nicolas.* [9]

2.3.1 Positives and Learnings from this Paper

1. Thesis covers a comprehensive comparison of facial recognition algorithms like Eigenfaces, Fisherfaces, LBPH, and OpenFace.
2. The study includes a detailed explanation of each algorithm, their strengths, weaknesses, and performance in a test case scenario.
3. The findings highlight OpenFace as the most accurate algorithm for facial recognition, providing valuable insights for further research in the field.

2.3.2 Identified Research Gaps

1. Lack of Exploration of Real-World Applications: The paper focuses on comparing facial recognition algorithms in a controlled setting. However, it does not delve into the practical applications of these algorithms in real-world scenarios.
2. Limited Discussion on Algorithm Limitations: While the strengths of the algorithms are discussed, there is a lack of emphasis on the limitations of each algorithm.
3. Absence of Future Research Directions: The paper concludes with the identification of the most accurate algorithm but fails to suggest potential future research directions in the field of facial recognition.

2.4 Paper 4

Title: *"Evaluating impact of race in facial recognition across machine learning and deep learning algorithms."* Author: *Coe, James, and Mustafa Atay.* [10]

2.4.1 Positives and Learnings from this Paper

1. The paper provides a detailed comparison of various facial recognition algorithms, including Eigenfaces, Fisherfaces, Local Binary Pattern Histogram, deep convolutional neural network algorithm, and OpenFace.
2. It highlights the efficiency and accuracy of these algorithms in real-life settings, with OpenFace being identified as the algorithm with the highest accuracy in identifying faces.

3. The study's findings offer valuable insights for practitioners in selecting the most suitable algorithm for facial recognition applications and suggest ways for academicians to enhance the current algorithms' accuracy further.

2.4.2 Identified Research Gaps

1. The paper focuses on a few specific facial recognition algorithms like Eigenfaces, Fisherfaces, and Local Binary Pattern Histograms. It lacks exploration of a wider range of algorithms available in the field, potentially missing out on newer, more accurate models.
2. While the study evaluates the algorithms' accuracy, it does not delve into their performance in real-life settings or practical applications. This gap could impact the algorithms' effectiveness when deployed in scenarios beyond controlled test environments.
3. The paper mentions the use of a custom dataset for testing the algorithms but does not elaborate on the dataset's diversity or size.

2.5 Paper 5

Title: *"A Comparative Study of Facial Recognition Techniques: With focus on low computational power."* Author: *Schenkel, T., Ringhage, O. and Branding, N.* [11]

2.5.1 Positives and Learnings from this Paper

1. Efficiency Evaluation: The paper provides a detailed comparison of popular open source facial recognition algorithms, highlighting the efficiency and accuracy of each in real-life settings.
2. Practical Implications: The findings of the study offer valuable insights for practitioners in selecting the most suitable algorithm for facial recognition applications, enhancing decision-making processes.
3. Academic Contribution: The research contributes to the academic field by emphasizing the importance of improving the accuracy of existing algorithms, paving the way for further advancements in facial recognition technology.

2.5.2 Identified Research Gaps

1. The paper focuses on comparing a few facial recognition algorithms like Eigenfaces, Fisherfaces, and Local Binary Pattern Histogram. However, it lacks a comparison with a wider range of algorithms to provide a more comprehensive analysis.
2. While the paper evaluates the algorithms' performance in a controlled environment using test datasets, it doesn't discuss the practical implementation challenges or results in real-life scenarios, which could be a crucial research gap.
3. The paper does not delve into the scalability and efficiency aspects of the facial recognition algorithms studied. Understanding how these algorithms perform with larger datasets or in real-time applications could be a significant research gap to address.

Chapter 3

Problem Statement

Manual attendance tracking at MITWPU Campus is a time-consuming process prone to inefficiencies and potential malpractices. The current system lacks automation, leading to increased per-class time spent on attendance management. To address these challenges, there is a need for the development of an Automated

Attendance Tracking System. The solution should leverage computer vision for accurate data capture, utilize cloud infrastructure for efficient storage and processing, and incorporate advanced data science techniques for analytics.

The system should be user-friendly, providing both teachers and students with a seamless experience while ensuring security and tamper resistance to mitigate the risk of malpractices in attendance tracking. The goal is to enhance overall efficiency and significantly reduce the manual effort involved in attendance monitoring.

3.1 Project Scope

The scope of the project includes the following key components:

1. Development of a mobile application using Flutter for user interaction and attendance management.
2. Implementation of a backend system using FastAPI for handling API requests and data processing.
3. Integration of a cloud database (MongoDB) for secure and scalable data storage.
4. Utilization of computer vision techniques for real-time face recognition and attendance marking.

3.2 Project Assumptions

1. The system will be used in a controlled environment, such as a classroom or lecture hall, where the lighting and camera angles are optimal for face recognition.
2. Users will have access to the necessary hardware and software components required for the system to function effectively.
3. The system will be used primarily for attendance tracking purposes and not for any other applications.
4. Users will have basic knowledge of using mobile applications and web interfaces.
5. The system will comply with relevant data privacy regulations and guidelines.

3.3 Project Limitations

1. Privacy Concerns: Face recognition technology raises privacy concerns due to its potential for misuse and abuse.
2. Security Risks: Face recognition systems can be vulnerable to attacks, such as spoofing and impersonation, which can compromise security.
3. Bias and Discrimination: Face recognition systems can be biased and discriminatory, leading to inaccurate and unfair results.
4. Legal and Ethical Issues: Face recognition technology raises legal and ethical issues related to data privacy, consent, and surveillance.
5. Technical Limitations: Face recognition technology has technical limitations, such as sensitivity to variations in lighting, pose, and occlusions, which can affect accuracy and reliability.
6.

3.4 Project Objectives

1. Develop an automated attendance tracking system using computer vision for any organization
2. Utilize cloud infrastructure for efficient storage and processing of attendance data.
3. Create a user-friendly application for both teachers and students to simplify attendance management.
4. Implement advanced data science techniques for processing large attendance datasets and performing analytics.
5. Significantly reduce per-class time spent on attendance tracking through automated processes, enhancing overall efficiency.
6. Mitigate the risk of malpractices in attendance tracking by implementing secure and tamper-resistant mechanisms.

Chapter 4

Project Requirements

4.1 Requirements

4.1.1 Hardware Requirements

Name	Purpose	Cost (INR)
Raspberry Pi Hi Quality Camera	Official camera from Raspberry Pi, more expensive for high resolution.	8000
Raspberry Pi Camera Module 3	Official camera from Raspberry Pi, cheaper and highest resolution for cheapest cost.	3000
Raspberry Pi 4 Model B with 2 GB RAM	To send image from camera to server.	5000

Table 4.1: List of Hardware Requirements

4.1.2 Software Requirements

1. Amazon S3
2. Amazon EC2
3. Amazon DynamoDB
4. Raspian OS
5. Python Libraries

```
annotated-types==0.6.0
anyio==4.2.0
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
arrow==1.3.0
asttokens==2.4.1
async-lru==2.0.4
attrs==23.2.0
Babel==2.14.0
beautifulsoup4==4.12.3
bleach==6.1.0
CacheControl==0.14.0
cachetools==5.3.2
certifi==2024.2.2
cffi==1.16.0
charset-normalizer==3.3.2
click==8.1.7
```

```
colorama==0.4.6
comm==0.2.1
contourpy==1.2.0
cryptography==42.0.5
cycler==0.12.1
debugpy==1.8.1
decorator==5.1.1
defusedxml==0.7.1
dlib==19.24.2
dnspython==2.6.1
exceptiongroup==1.2.0
executing==2.0.1
face-recognition==1.3.0
face_recognition_models==0.3.0
fastapi==0.109.2
fastjsonschema==2.19.1
firebase-admin==6.4.0
fonttools==4.49.0
fqdn==1.5.1
gcloud==0.18.3
google-api-core==2.17.1
google-api-python-client==2.119.0
google-auth==2.28.1
google-auth-httplib2==0.2.0
google-cloud-core==2.4.1
google-cloud-firestore==2.15.0
google-cloud-storage==2.14.0
google-crc32c==1.5.0
google-resumable-media==2.7.0
googleapis-common-protos==1.62.0
grpcio==1.62.0
grpcio-status==1.62.0
h11==0.14.0
httpcore==1.0.4
httplib2==0.22.0
httpx==0.27.0
idna==3.6
ipykernel==6.29.2
ipython==8.21.0
ipywidgets==8.1.2
isoduration==20.11.0
jedi==0.19.1
Jinja2==3.1.3
json5==0.9.17
jsonpointer==2.4
jsonschema==4.21.1
jsonschema-specifications==2023.12.1
jupyter==1.0.0
jupyter-console==6.6.3
jupyter-events==0.9.0
jupyter-lsp==2.2.2
jupyter_client==8.6.0
jupyter_core==5.7.1
jupyter_server==2.12.5
jupyter_server_terminals==0.5.2
jupyterlab==4.1.2
jupyterlab_pygments==0.3.0
jupyterlab_server==2.25.3
jupyterlab_widgets==3.0.10
jwcrypto==1.5.4
kiwisolver==1.4.5
MarkupSafe==2.1.5
matplotlib==3.8.3
matplotlib-inline==0.1.6
mistune==3.0.2
msgpack==1.0.7
nbclient==0.9.0
nbconvert==7.16.1
```

```
nbformat==5.9.2
nest-asyncio==1.6.0
notebook==7.1.0
notebook_shim==0.2.4
numpy==1.26.4
oauth2client==4.1.3
opencv-python==4.9.0.80
overrides==7.7.0
packaging==23.2
pandocfilters==1.5.1
parso==0.8.3
pillow==10.2.0
pipdeptree==2.16.0
platformdirs==4.2.0
prometheus_client==0.20.0
prompt-toolkit==3.0.43
proto-plus==1.23.0
protobuf==4.25.3
psutil==5.9.8
pure-eval==0.2.2
pyasn1==0.5.1
pyasn1-modules==0.3.0
pycparser==2.21
pycryptodome==3.20.0
pydantic==2.5.3
pydantic_core==2.14.6
Pygments==2.17.2
PyJWT==2.8.0
pymongo==4.6.2
pyparsing==3.1.1
python-dateutil==2.8.2
python-dotenv==1.0.1
python-json-logger==2.0.7
python-jwt==4.1.0
python-multipart==0.0.9
pywin32==306
pywinpty==2.0.12
PyYAML==6.0.1
pyzmq==25.1.2
qtconsole==5.5.1
QtPy==2.4.1
referencing==0.33.0
requests==2.29.0
requests-toolbelt==0.10.1
rfc3339-validator==0.1.4
rfc3986-validator==0.1.1
rpds-py==0.18.0
rsa==4.9
Send2Trash==1.8.2
services==0.1.1
six==1.16.0
sniffio==1.3.0
soupsieve==2.5
stack-data==0.6.3
starlette==0.37.1
terminado==0.18.0
tinycss2==1.2.1
tomli==2.0.1
tornado==6.4
traitlets==5.14.1
types-python-dateutil==2.8.19.20240106
typing_extensions==4.9.0
uri-template==1.3.0
uritemplate==4.1.1
urllib3==1.26.18
uvicorn==0.27.0
wcwidth==0.2.13
webcolors==1.13
```

```
webencodings==0.5.1
websocket-client==1.7.0
widgetsnbextension==4.0.10
```

4.2 Risk Management

1. **Technical Risks**: The project may face technical challenges related to the integration of various components, such as the mobile application, backend, and cloud services. To mitigate this risk, thorough testing and validation will be conducted at each stage of development.

2. **Data Privacy Risks**: The use of face recognition technology raises concerns about data privacy and security. To address this, the system will comply with relevant data protection regulations and implement secure data storage and transmission practices.

3. **User Acceptance Risks**: Users may be resistant to adopting the new system due to concerns about privacy or usability. To mitigate this risk, user training and support will be provided, along with clear communication about the benefits of the system.

4. **Performance Risks**: The system may not perform as expected under high load or in real-world conditions. To address this, performance testing will be conducted to ensure the system can handle the expected number of users and data volume.

5. **Regulatory Risks**: Changes in regulations related to data privacy and face recognition technology may impact the project. To mitigate this risk, the project team will stay informed about relevant regulations and adapt the system as needed to ensure compliance.

4.3 Functional Specifications

4.3.1 Interfaces

External Interfaces Required

1. **User Interface**: The mobile application will have a user-friendly interface for students and teachers to interact with the system. It will include features for face capture, attendance marking, and viewing attendance history.

2. **Camera Interface**: The system will utilize the camera on the mobile device or an external camera (e.g., Raspberry Pi camera) for face capture. The camera will be integrated with the mobile application to capture images for face recognition.

3. **Cloud Storage**: The system will use cloud storage (e.g., AWS S3) for storing images and other media files. The backend will handle the upload and retrieval of these files.

4. **Database Interface**: The system will connect to a cloud database (e.g., MongoDB Atlas) for storing user data, face encodings, and attendance records. The backend will interact with the database to perform CRUD operations.

5. **IoT Device Interface**: If IoT devices (e.g., Raspberry Pi) are used for real-time face recognition, they will communicate with the backend via HTTP requests to send captured images and receive attendance updates.

Internal Interfaces Required

1. **Frontend**: The mobile application will be developed using Flutter, which will communicate with the backend via RESTful APIs. The frontend will handle user authentication, face capture, and attendance display.

2. **Backend**: The backend will be developed using FastAPI, which will handle API requests from the frontend. It will also manage face encoding and attendance logging. The backend will connect to the MongoDB database for data storage and retrieval.

3. **Database**: The MongoDB database will store user data, face encodings, and attendance records. The backend will interact with the database to perform CRUD operations.

Communication Interfaces

1. **REST API**: The system will use RESTful APIs for communication between the frontend and backend components. The APIs will handle requests for user authentication, face encoding, and attendance logging.

2. **Database Connection**: The backend will connect to the MongoDB database using a secure connection string. The database will store user data, face encodings, and attendance records.

3. **Cloud Storage**: The system will use cloud storage (e.g., AWS S3) for storing images and other media files. The backend will handle the upload and retrieval of these files.

4. **IoT Device Communication**: If IoT devices (e.g., Raspberry Pi) are used for real-time face recognition, they will communicate with the backend via HTTP requests to send captured images and receive attendance updates.

5. **User Authentication**: The system will implement user authentication using JWT (JSON Web Tokens) for secure access to the APIs. Users will need to log in to access the attendance features.

Graphical User Interfaces

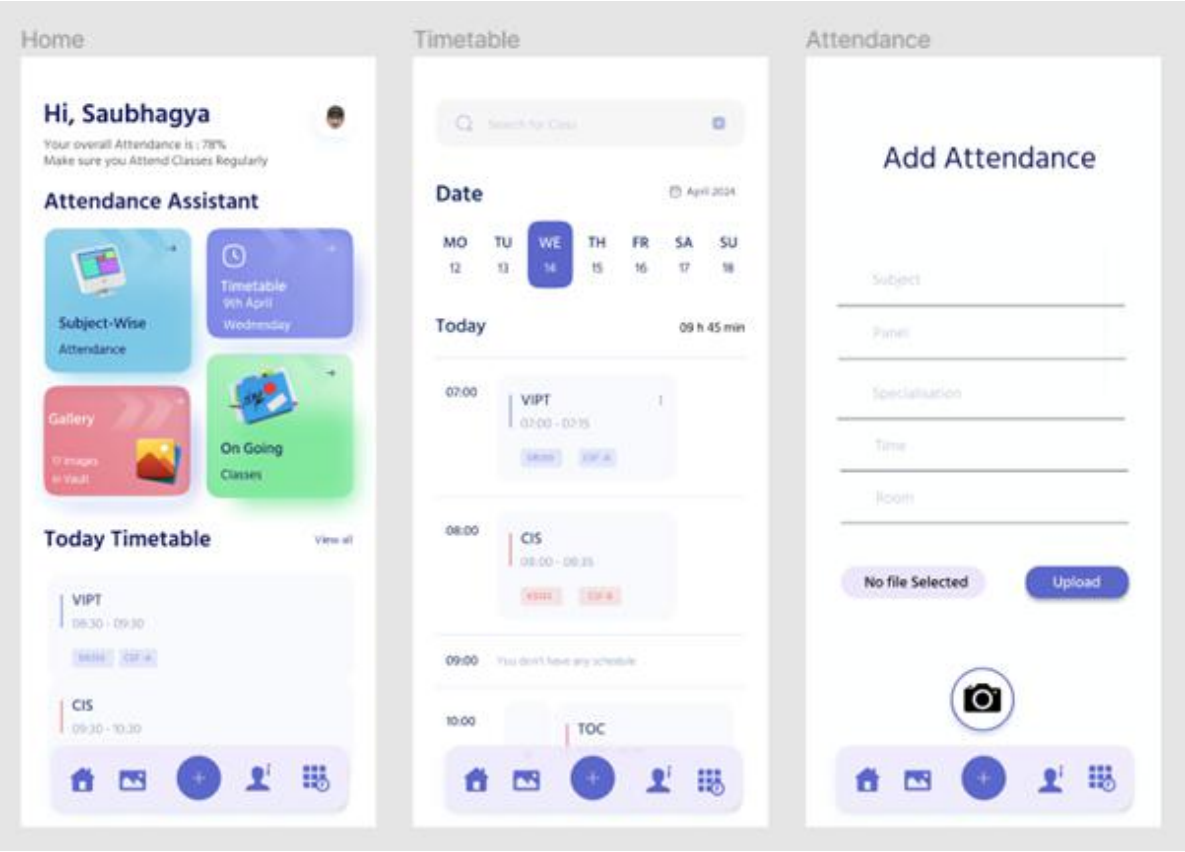


Figure 4.1: App Design

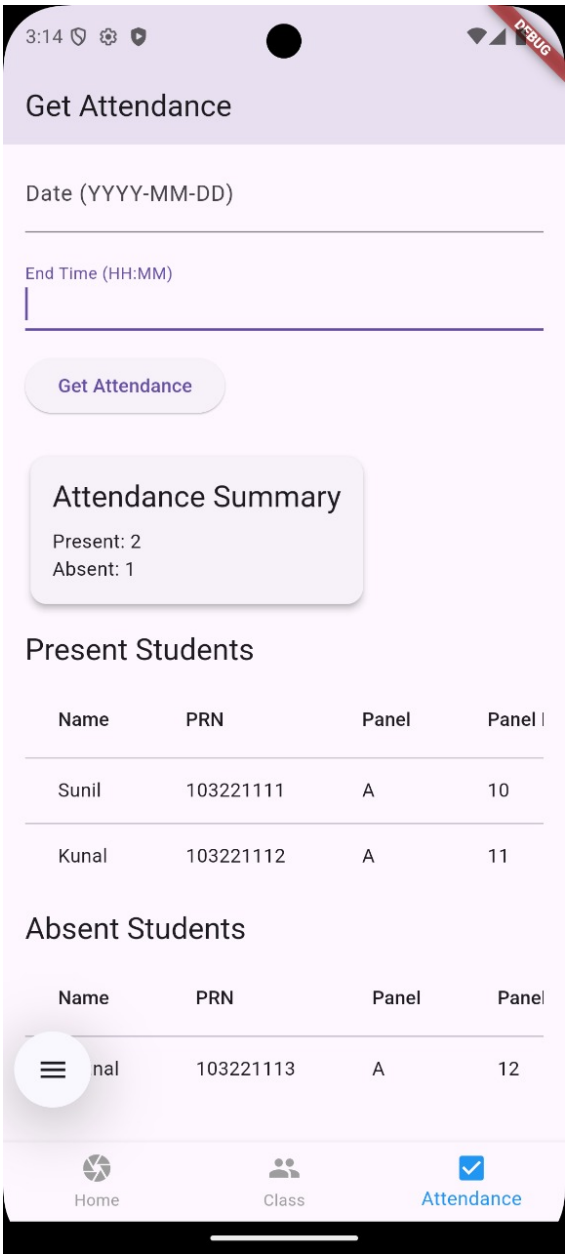


Figure 4.2: App Design

4.3.2 Interactions

Sustainability

1. The system will be designed to be scalable, allowing for easy addition of new features and functionalities in the future.
2. The use of cloud services will ensure that the system can handle increased data storage and processing requirements as the user base grows.
3. The system will be developed using modular architecture, making it easier to maintain and update individual components without affecting the entire system.

4. The project will follow best practices in software development, including version control, code reviews, and documentation, to ensure long-term maintainability.

Quality Management

1. The system will undergo rigorous testing, including unit tests, integration tests, and user acceptance tests, to ensure high quality and reliability.
2. The project will follow coding standards and best practices to maintain code quality and readability.
3. The system will implement error handling and logging mechanisms to identify and resolve issues quickly.
4. The project will include documentation for both developers and users, providing clear instructions on how to use and maintain the system.

Security

1. The system will implement secure user authentication using JWT (JSON Web Tokens) to protect user data and prevent unauthorized access.
2. All sensitive data, including user credentials and face encodings, will be encrypted before being stored in the database.
3. The system will use HTTPS for secure communication between the frontend and backend components, ensuring that data transmitted over the network is encrypted.
4. The project will follow best practices for data privacy and compliance with relevant regulations, such as GDPR, to protect user information.
5. The system will implement access controls to restrict access to sensitive data and functionalities based on user roles (e.g., admin, teacher, student).
6. The project will include regular security audits and vulnerability assessments to identify and address potential security risks.
7. The system will implement logging and monitoring mechanisms to track user activities and detect any suspicious behavior.

Chapter 5

System Analysis Proposed Architecture

5.1 Block Diagram

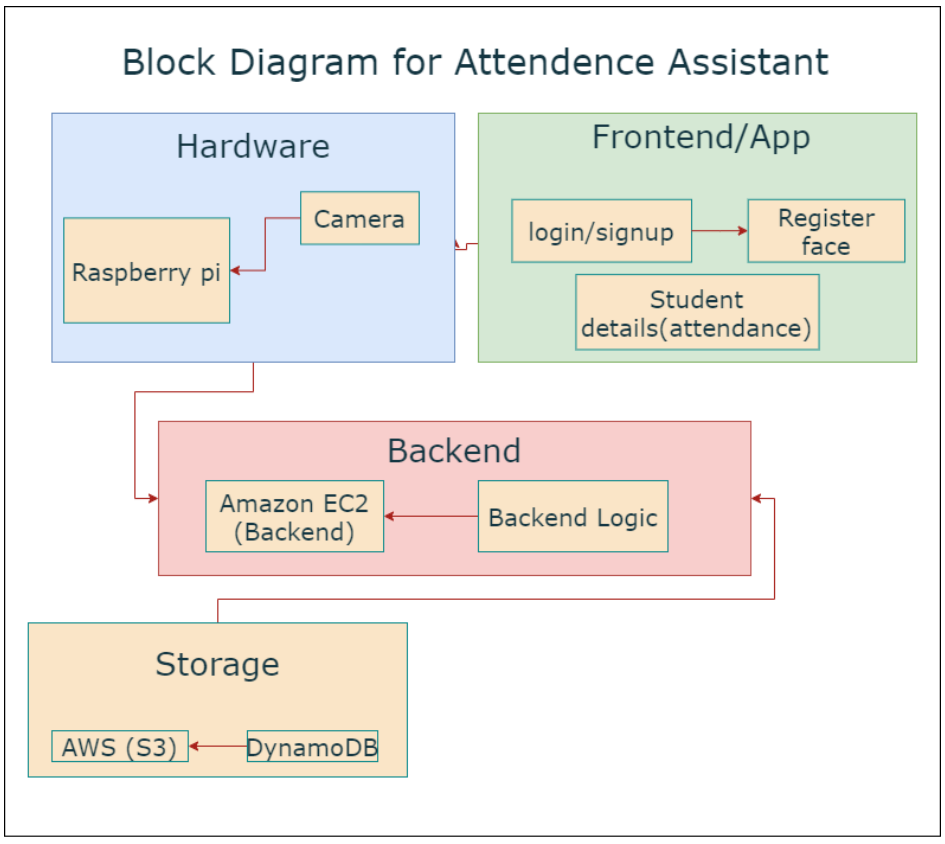


Figure 5.1: Block Diagram

5.2 Activity Diagram

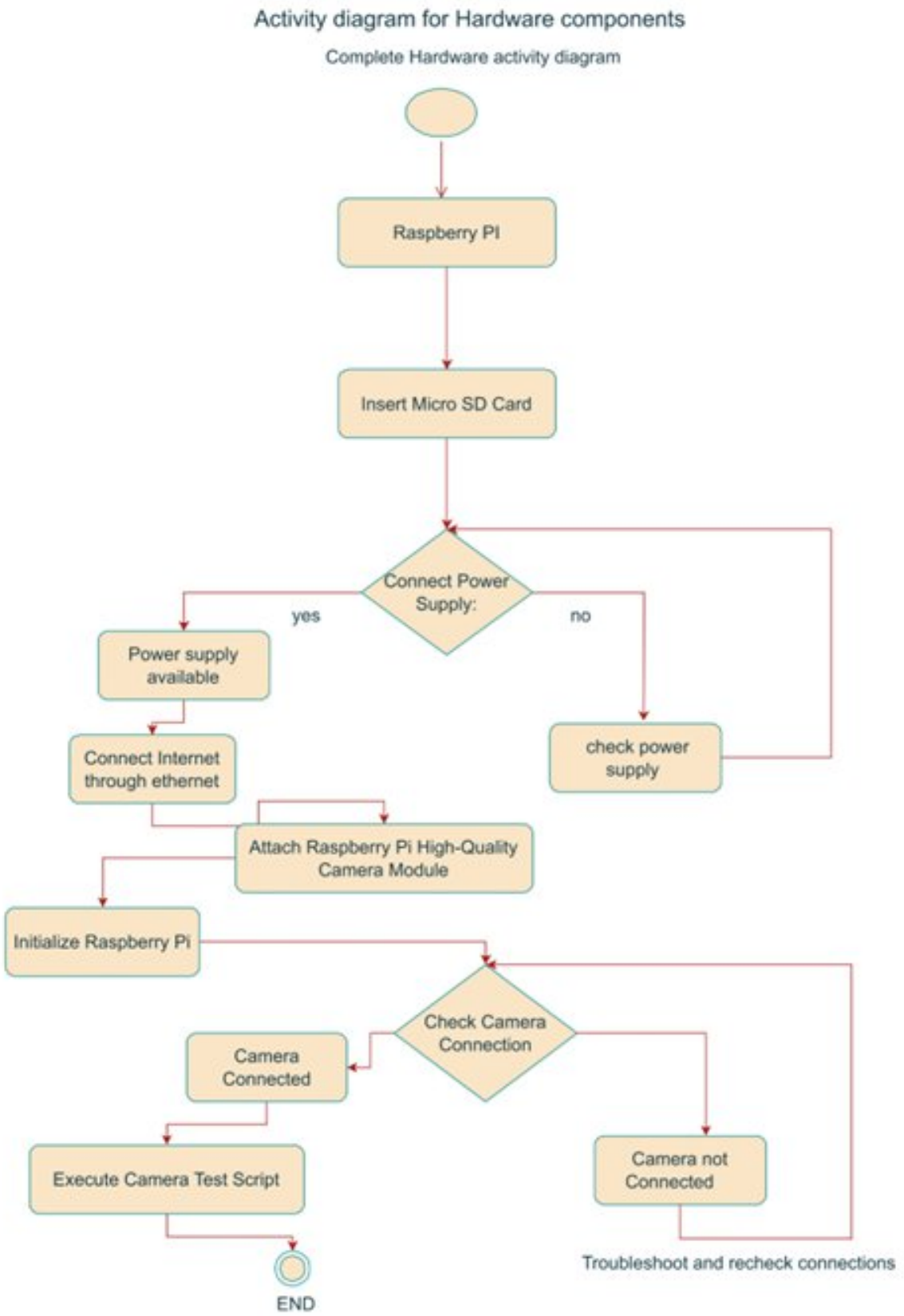


Figure 5.2: Activity Diagram

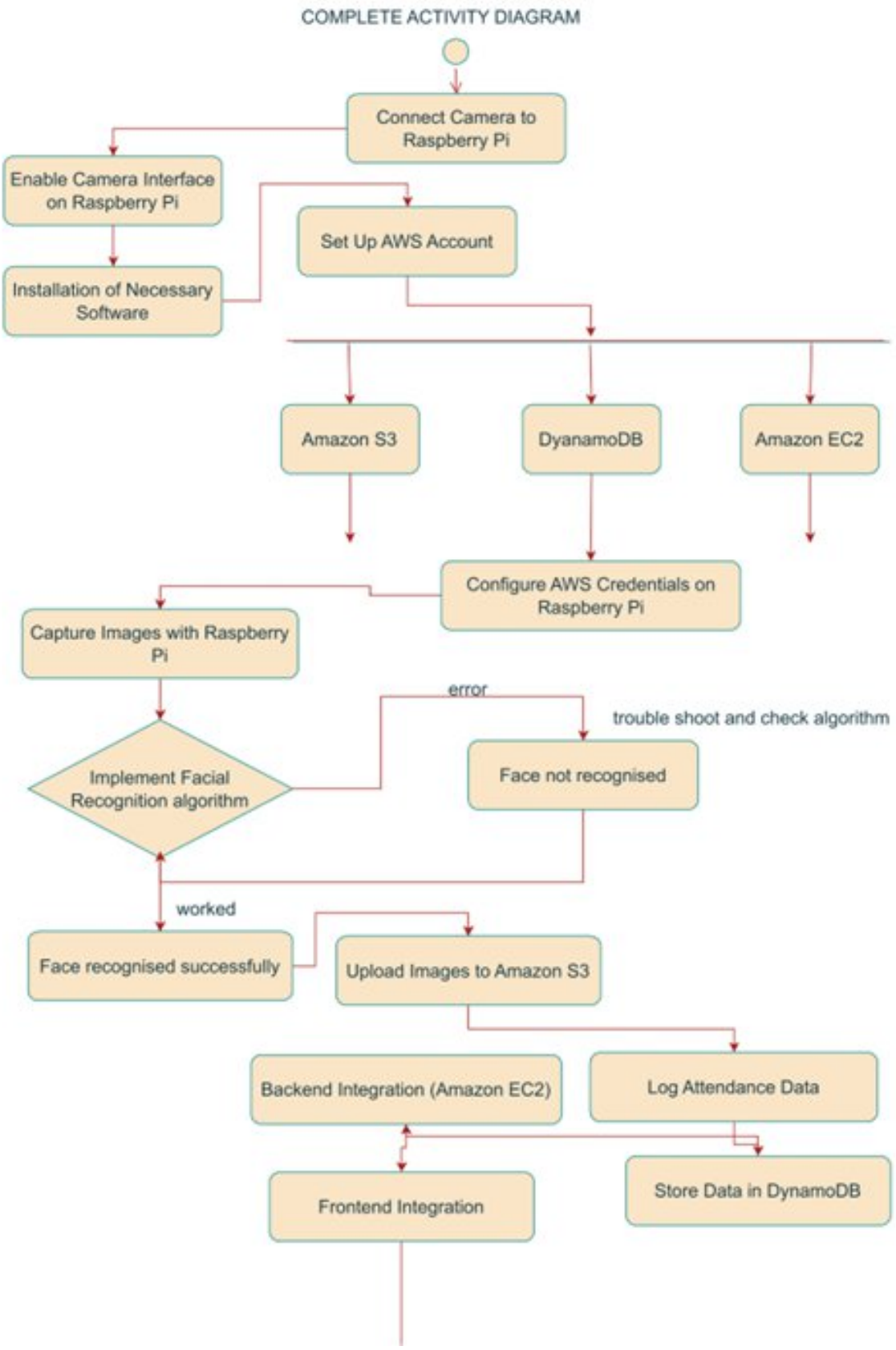


Figure 5.3: Activity Diagram Continued

Chapter 6

Project Plan - Timeline

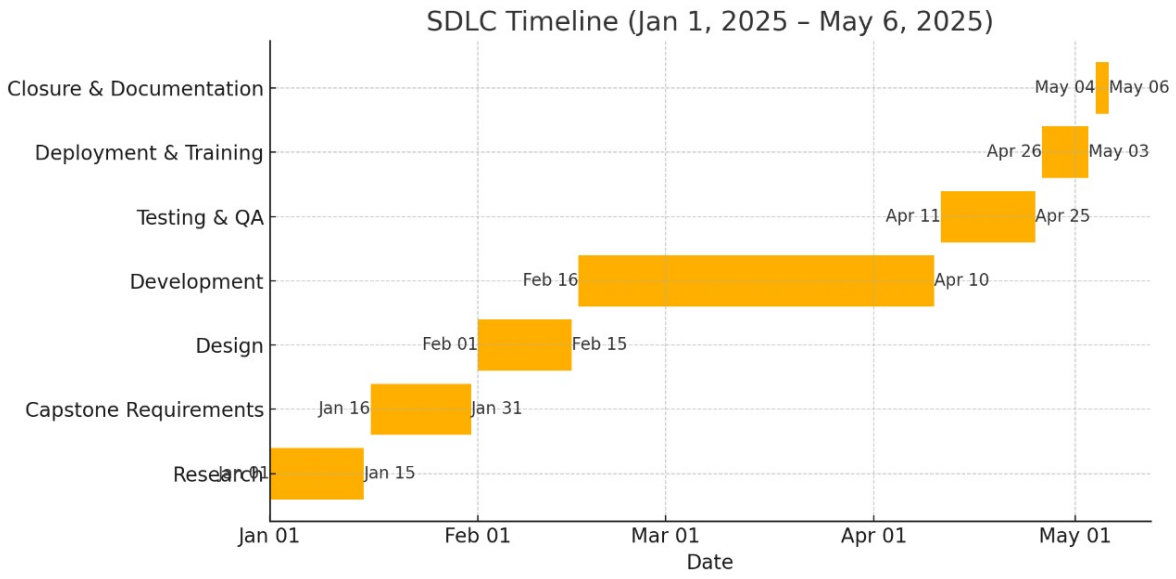


Figure 6.1: Project Timeline

6.1 Phase 1: Research

1. We started with a literature survey to understand the existing face recognition algorithms and their performance metrics.
2. We compared the performance of different algorithms, including Eigenfaces, Fisherfaces, LBPH, and OpenFace, to identify the most suitable one for our project.
3. We also explored the use of deep learning techniques, such as Convolutional Neural Networks (CNNs), for face recognition.
4.

6.2 Phase 2: Requirements Gathering

1. Conducted stakeholder interviews with professors and students to understand their pain points in attendance tracking.

2. Defined use cases such as "Mark attendance for a class of 50 students in under 5 minutes" and "Generate attendance reports for a semester."
3. Identified non-functional requirements like 95% accuracy, real-time processing, and GDPR compliance for data privacy.
4. Created a requirements traceability matrix to map user stories to functional and non-functional requirements.

6.3 Phase 3: System and Architecture Design

1. Designed a modular architecture with three main components: frontend (Flutter), backend (FastAPI), and database (MongoDB).
2. Created a system block diagram showing data flow between the mobile app, API, and database.
3. Defined REST API endpoints for user authentication, face encoding, and attendance logging.
4. Developed a database schema with collections for users, face encodings, and attendance logs.
5. Incorporated fault-tolerant mechanisms like retry logic for API calls and database backups.

6.4 Phase 4: Development

1. Implemented the backend using FastAPI, including endpoints for face encoding and attendance marking.
2. Developed the frontend using Flutter, with features like face capture, attendance history, and user authentication.
3. Integrated the face_recognition library for face detection and recognition in the backend.
4. Used Docker to containerize the application for consistent deployment across environments.
5. Followed Git flow for version control, with feature branches and pull requests for code reviews.

6.5 Phase 5: Testing and Quality Assurance

1. Conducted unit tests for backend API endpoints to ensure correct functionality.
2. Performed integration tests to validate end-to-end workflows, such as face capture to attendance logging.
3. Conducted user acceptance testing (UAT) with a small group of students and professors to gather feedback.
4. Measured system performance under load to ensure it can handle concurrent requests from multiple users.
5. Fixed bugs and optimized the application based on test results and user feedback.

6.6 Phase 6: Deployment and Training

1. Deployed the backend on Local system and MongoDB Atlas for database management.

6.7 Phase 7: Closure and Documentation

1. Prepared a final project report summarizing the implementation, testing, and deployment phases.
2. Delivered the source code, documentation, and deployment scripts to the project stakeholders.
3. Conducted a retrospective meeting to discuss lessons learned and areas for improvement.
4. Archived all project artifacts, including code, datasets, and test results, for future reference.
5. Presented the project outcomes to the academic panel and received feedback for further enhancements.

Chapter 7

Implementation

7.1 Methodology

Libraries Tested
These are the libraries that were used to train and test a model.

1. OpenCV
2. face_recognition
    - face_recognition is a Python library that provides a simple interface for face recognition tasks.
    - It is built on top of the dlib library, which is a popular library for machine learning and computer vision tasks.
    - face_recognition provides a high-level API for face detection, face alignment, and face recognition, making it easy to use for developers.
    - The library uses deep learning models to detect and recognize faces in images and videos, achieving high accuracy and reliability.
    - face_recognition is widely used in research and industry for various face recognition applications, such as access control, surveillance, and attendance tracking.

7.2 Advantages

1. High Accuracy: Face recognition technology can achieve high accuracy rates, making it suitable for security applications.
2. Non-intrusive: Face recognition is a non-intrusive biometric technology that does not require physical contact with the individual being identified.
3. Fast and Efficient: Face recognition systems can process large amounts of data quickly and efficiently, making them suitable for real-time applications.
4. Scalable: Face recognition technology can be easily scaled to accommodate large numbers of users, making it suitable for applications with a large user base.
5. Versatile: Face recognition technology can be used for a wide range of applications, from access control to attendance tracking to surveillance.

7.3 Implementations

7.3.1 Platform

Operating System: Windows 11 Pro
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters: Python 3.10.1

7.3.2 Training Data

- The dataset used for training and testing the face recognition algorithms is a collection of images of individuals, each labeled with their name.
- The dataset contains images of different individuals taken under various lighting conditions, angles, and expressions to ensure robustness in face recognition.
- The dataset is divided into two parts: a training set used to train the face recognition model and a testing set used to evaluate the model's performance.

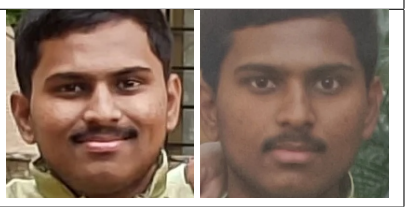
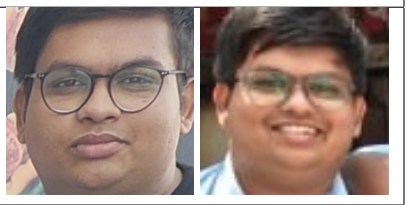
Serial Number	Name	Image
1	Saubhagya	
2	Avishkar	
3	Karad	
4	Krish	
5	Parth	
6	Abhijeet	
7	Naman	
8	Mayur	

Table 7.1: Training Data Images

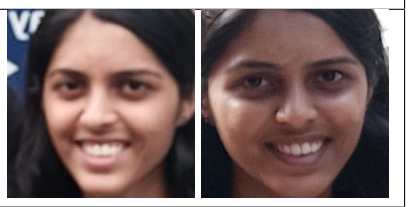
Serial Number	Name	Image
9	Sahaj	
10	Maitreyee	
11	Prathamesh	
12	Khare	

Table 7.2: Training Data Images (Continued)

7.3.3 Creation of Dataset

Cropped Faces using openCV-haar-cascades

The following code generated 18,832 possible faces (245x245px) each. The code uses the OpenCV library to detect faces in images and crop them. The cropped faces are then saved in a specified output folder. The code uses the Haar Cascade classifier for face detection, which is a popular method for real-time face detection.

```
import cv2
import os
import numpy as np

def detect_and_crop_faces(input_folder, output_folder, padding=10):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + '
    haarcascade_frontalface_default.xml')

    for filename in os.listdir(input_folder):
        if filename.lower().endswith(('png', 'jpg', 'jpeg', 'webp')):
            image_path = os.path.join(input_folder, filename)
            image = cv2.imread(image_path)

            if image is None:
                continue

            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
    minSize=(30, 30))

```

```
            for i, (x, y, w, h) in enumerate(faces):
                x1 = max(x - padding, 0)
                y1 = max(y - padding, 0)
                x2 = min(x + w + padding, image.shape[1])
                y2 = min(y + h + padding, image.shape[0])

                face_crop = image[y1:y2, x1:x2]
                output_path = os.path.join(output_folder, f"{os.path.splitext(filename)[0]}
    _face_{i}.jpg")
                cv2.imwrite(output_path, face_crop)
                print(f"Saved cropped face: {output_path}")

input_folder = os.path.join(os.getcwd(), "input_images")
output_folder = os.path.join(os.getcwd(), "output_images")

detect_and_crop_faces(input_folder, output_folder)
```

Results of the above code are shown in the figure below. The images are cropped and saved in the output folder.

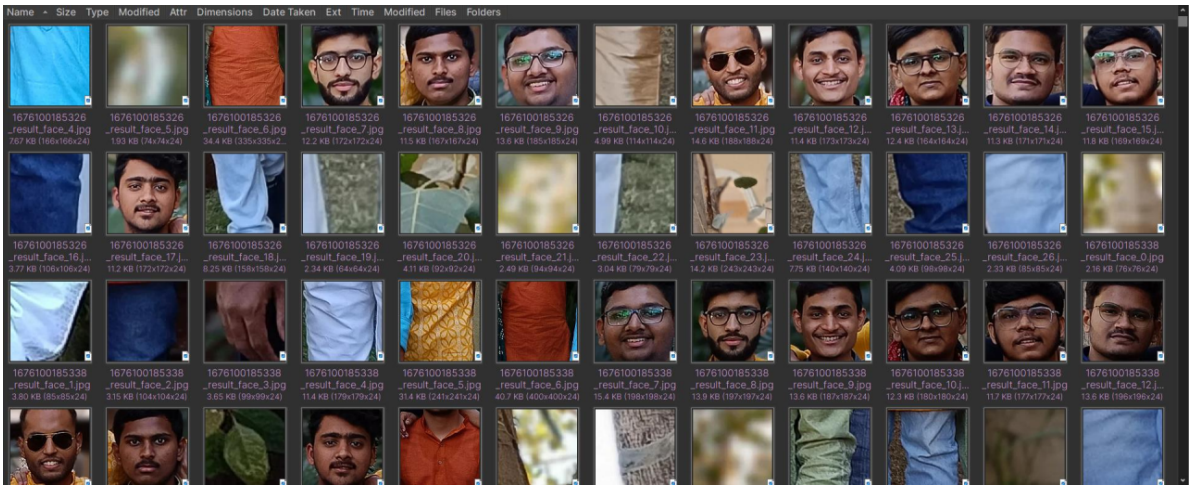


Figure 7.1: Cropped Faces using OpenCV Haar Cascades

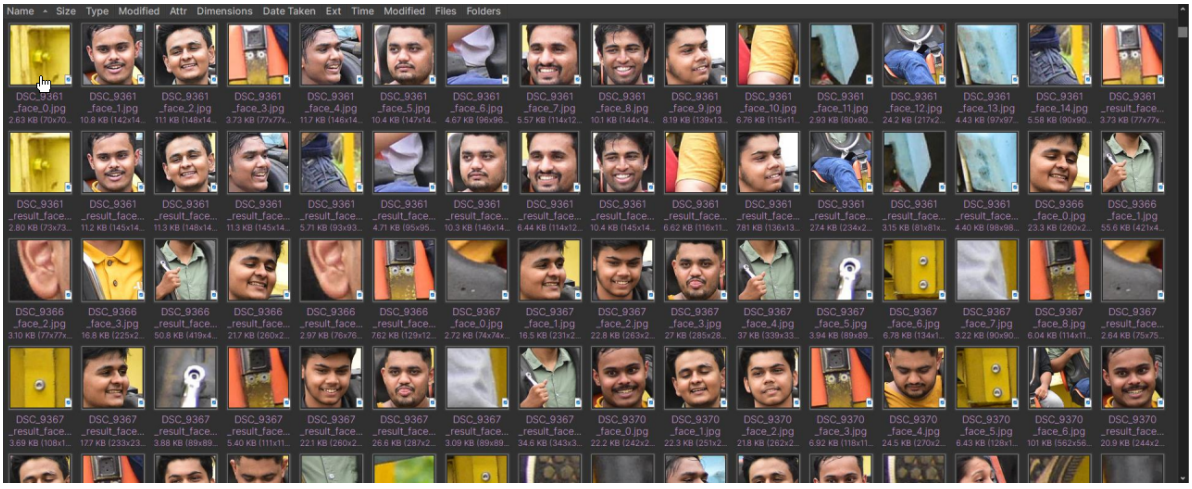


Figure 7.2: Cropped Faces using OpenCV Haar Cascades

Figure 7.3: Cropped Faces using OpenCV Haar Cascades

7.4 API Endpoints

7.4.1 Attendance API Endpoints

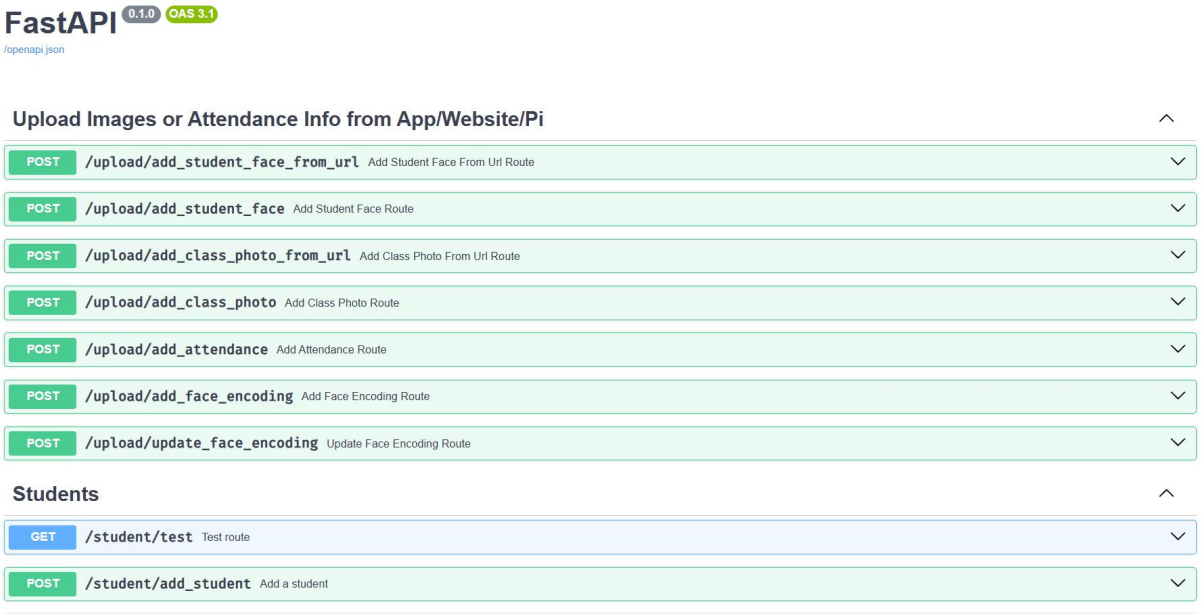


Figure 7.4: API Endpoints for Attendance details

7.4.2 Student API Endpoints

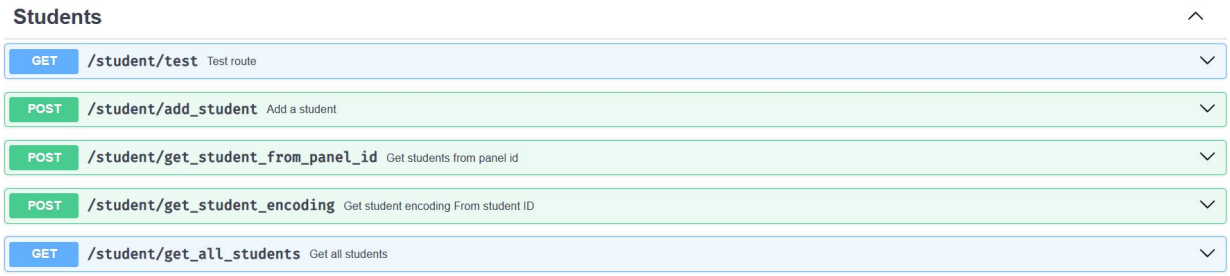


Figure 7.5: API Endpoints for Student details

7.4.3 Schools and Specialization API Endpoints

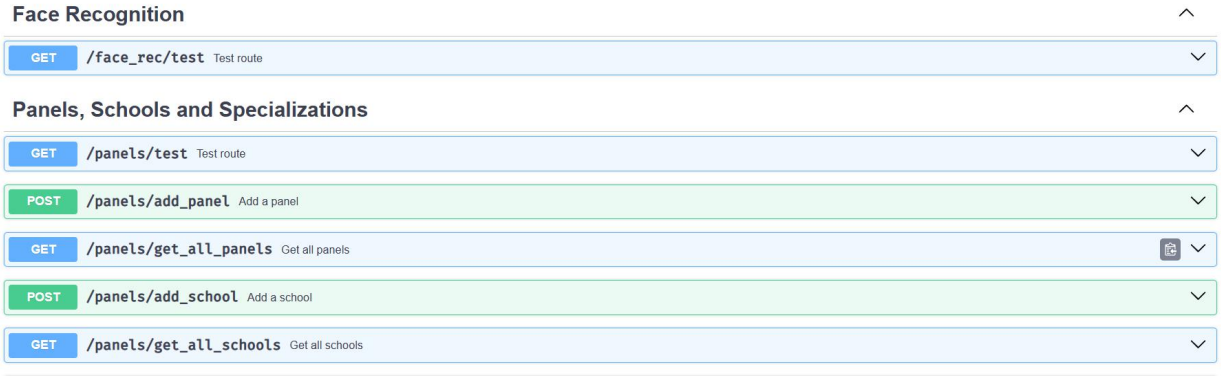


Figure 7.6: API Endpoints for Schools and Specialization details

7.4.4 Panel API Endpoints

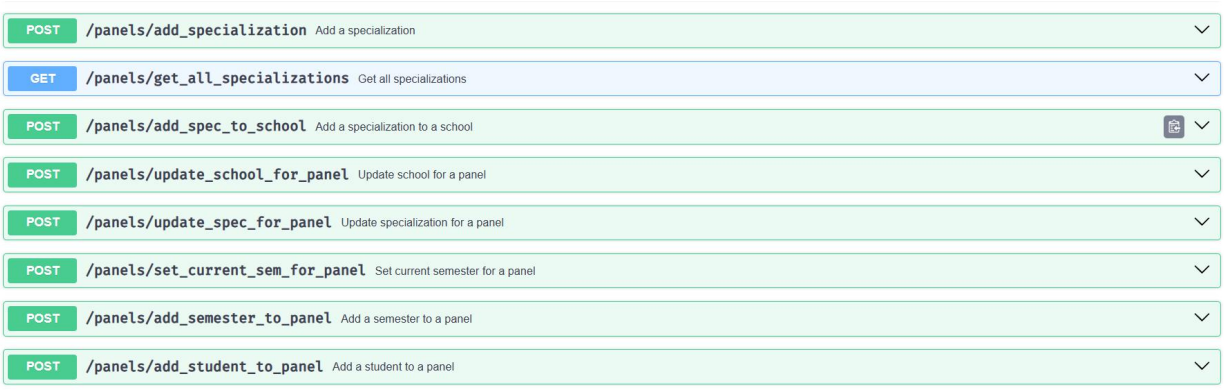


Figure 7.7: API Endpoints for Panel details

7.4.5 Rooms and Buildings API Endpoints

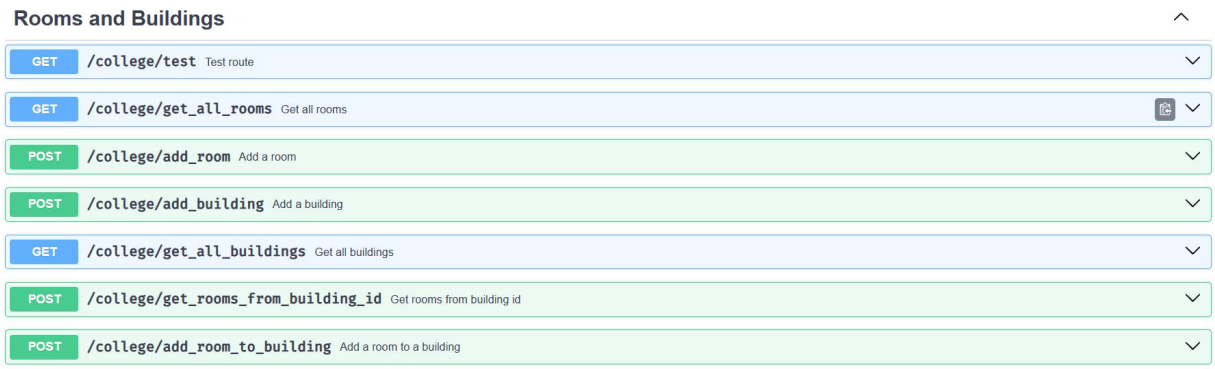


Figure 7.8: API Endpoints for Rooms and Buildings details

7.4.6 Subjects and Semesters API Endpoints

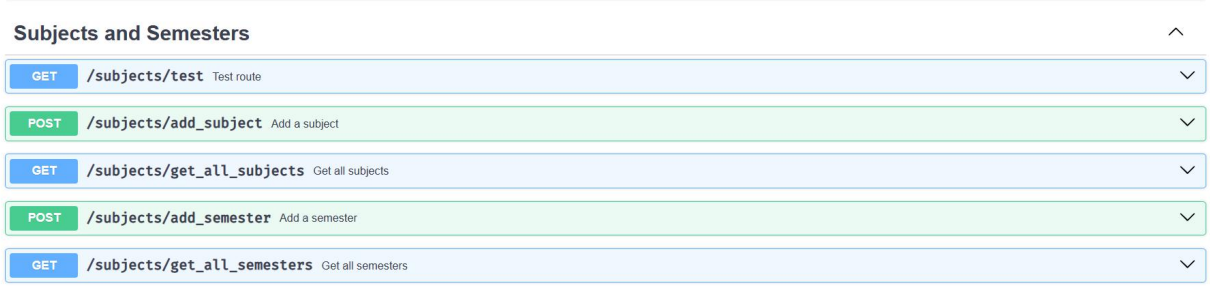


Figure 7.9: API Endpoints for Subjects and Semesters details

7.4.7 Teacher API Endpoints

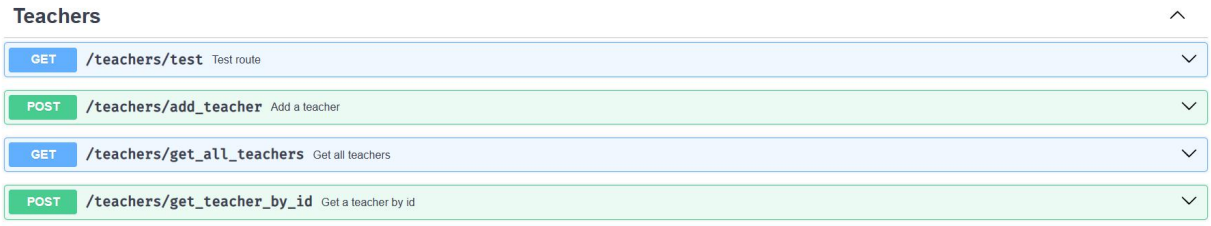


Figure 7.10: API Endpoints for Teacher details

7.4.8 Lecture API Endpoints

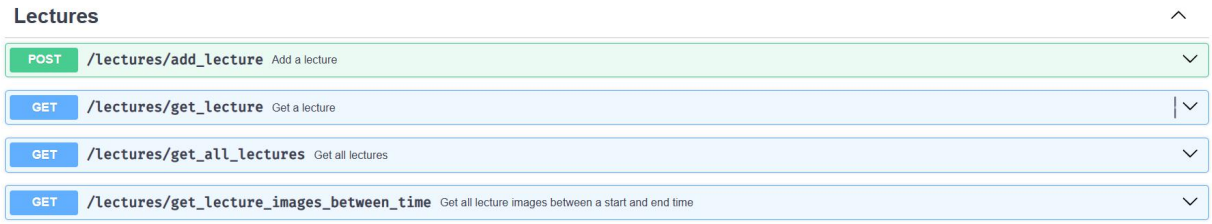


Figure 7.11: API Endpoints for Lecture details

7.5 Model Training

7.5.1 Code Snippet for face recognition using LBPH

```
import os
import cv2
import numpy as np
from sklearn.metrics import classification_report

base_path = "    "
image_size = (100, 100)

def load_images(folder, label, max_images=None):
    images = []
    labels = []
    for i, filename in enumerate(os.listdir(folder)):
        if max_images and i >= max_images:
            break
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        if img is not None:
            img = cv2.resize(img, image_size)
            images.append(img)
            labels.append(label)
    return images, labels

def train_individual_models(base_path):
    person_dirs = [d for d in os.listdir(base_path) if os.path.isdir(os.path.join(base_path,
     d))]
    for person in person_dirs:
        print(f"\nTraining model for: {person}")

        # Positive samples (label = 1)
        pos_train_dir = os.path.join(base_path, person, 'train')
        pos_test_dir = os.path.join(base_path, person, 'test')
        pos_train_imgs, pos_train_labels = load_images(pos_train_dir, label=1)
        pos_test_imgs, pos_test_labels = load_images(pos_test_dir, label=1)

        # Negative samples (label = 0) from other people's train dirs
        neg_train_imgs = []
        neg_train_labels = []
        neg_test_imgs = []
        neg_test_labels = []
        for other_person in person_dirs:
            if other_person == person:
                continue
            other_train_dir = os.path.join(base_path, other_person, 'train')
            other_test_dir = os.path.join(base_path, other_person, 'test')
            imgs, labels = load_images(other_train_dir, label=0, max_images=len(
    pos_train_imgs)//(len(person_dirs)-1))
```

```
            neg_train_imgs += imgs
            neg_train_labels += labels
            imgs, labels = load_images(other_test_dir, label=0, max_images=len(pos_test_imgs
    )//(len(person_dirs)-1))
            neg_test_imgs += imgs
            neg_test_labels += labels

        # Combine positives and negatives
        X_train = pos_train_imgs + neg_train_imgs
        y_train = pos_train_labels + neg_train_labels
        X_test = pos_test_imgs + neg_test_imgs
        y_test = pos_test_labels + neg_test_labels

        # Train LBPH model
        model = cv2.face.LBPHFaceRecognizer_create()
        model.train(X_train, np.array(y_train))

        # Evaluate
        predictions = []
        for face in X_test:
            label_pred, _ = model.predict(face)
            predictions.append(label_pred)

        print(classification_report(y_test, predictions, target_names=["Other", person]))

train_individual_models(base_path)
```

7.5.2 Code Snippet for face recognition using face_net

```
import os
import torch
import numpy as np
from PIL import Image
from tqdm import tqdm
from torchvision import transforms
from facenet_pytorch import InceptionResnetV1, MTCNN
from sklearn.metrics.pairwise import cosine_similarity

# Device config
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Load FaceNet model
model = InceptionResnetV1(pretrained='vggface2').eval().to(device)

# MTCNN for face detection
mtcnn = MTCNN(image_size=160, margin=0, min_face_size=20, keep_all=False, device=device)

# Define dataset path
dataset_path = 'comprehensive_db\\comprehensive_db'

# Function to extract embeddings
def get_embedding(img_path):
    img = Image.open(img_path).convert('RGB')
    face = mtcnn(img)
    if face is not None:
        face_embedding = model(face.unsqueeze(0).to(device))
        return face_embedding.detach().cpu().numpy()[0]
    else:
        return None

# Step 1: Prepare reference embeddings from training data
reference_embeddings = {}  # person -> list of embeddings

print("\nCreating reference embeddings from train data...")
for person in os.listdir(dataset_path):
    person_train_path = os.path.join(dataset_path, person, 'train')
```

```
    embeddings = []
    for img_name in os.listdir(person_train_path):
        img_path = os.path.join(person_train_path, img_name)
        emb = get_embedding(img_path)
        if emb is not None:
            embeddings.append(emb)
    if embeddings:
        reference_embeddings[person] = np.mean(embeddings, axis=0)  # Mean embedding

# Step 2: Evaluate on test data
print("\nEvaluating on test data...")
correct = 0
total = 0

for person in os.listdir(dataset_path):
    person_test_path = os.path.join(dataset_path, person, 'test')
    for img_name in os.listdir(person_test_path):
        img_path = os.path.join(person_test_path, img_name)
        test_emb = get_embedding(img_path)
        if test_emb is None:
            continue

        # Compare with reference embeddings
        similarities = {}
        for ref_person, ref_emb in reference_embeddings.items():
            sim = cosine_similarity([test_emb], [ref_emb])[0][0]
            similarities[ref_person] = sim

        predicted_person = max(similarities, key=similarities.get)
        if predicted_person == person:
            correct += 1
        total += 1

accuracy = correct / total if total > 0 else 0
print(f"\n FaceNet Recognition Accuracy: {accuracy:.2f} ({correct}/{total})")
```

7.5.3 Code Snippet for face recognition using res_net

```

from torchvision import models
from torch.utils.data import DataLoader
import torch.nn as nn
import torch
import torch.optim as optim

# Paths
train_data_path = 'comprehensive_db\\comprehensive_db'

# Transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

# Load dataset
train_dataset = FaceTrainDataset(train_data_path, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# Model setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = models.resnet50(pretrained=True)

# Replace the classifier for num_classes
num_classes = len(train_dataset.class_to_idx)
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

```
model.to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    correct = 0
    total = 0
    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    acc = 100 * correct / total
    print(f"Epoch [{epoch+1}/{num_epochs}] - Loss: {total_loss:.4f}, Accuracy: {acc:.2f}%")

# Save model
torch.save(model.state_dict(), "resnet50_face_recognizer.pth")
```

Chapter 8

Performance Evaluation and Testing

8.1 Evaluation Metrics

1. Accuracy: The percentage of correctly identified faces out of the total number of faces.
2. Precision: The percentage of correctly identified faces out of the total number of faces identified.
3. Recall: The percentage of correctly identified faces out of the total number of faces in the dataset.
4. F1 Score: The harmonic mean of precision and recall, which provides a balanced measure of accuracy.

8.2 Testing Methodology

1. Dataset Preparation: The dataset was divided into training and testing sets, with 80% of the images used for training and 20% for testing.
2. Model Training: The face recognition model was trained using the training set, and the parameters were optimized for better accuracy.
3. Model Testing: The trained model was tested using the testing set, and the evaluation metrics were calculated based on the results.
4. Performance Comparison: The performance of different face recognition algorithms was compared based on the evaluation metrics.

8.3 Testing and Training Dataset

Manually Segregated photos for 5 people and labelled them

Figure 8.1: Krishnaraj's Segregated Images

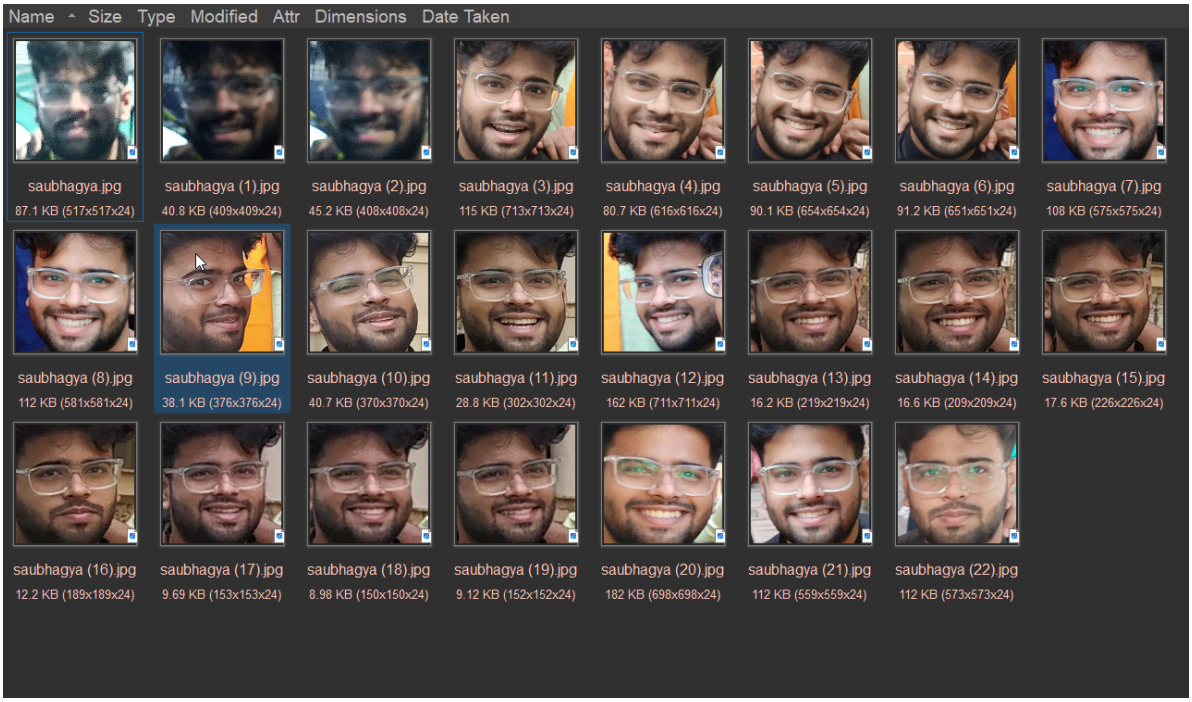


Figure 8.2: Saubhagya's Segregated Images

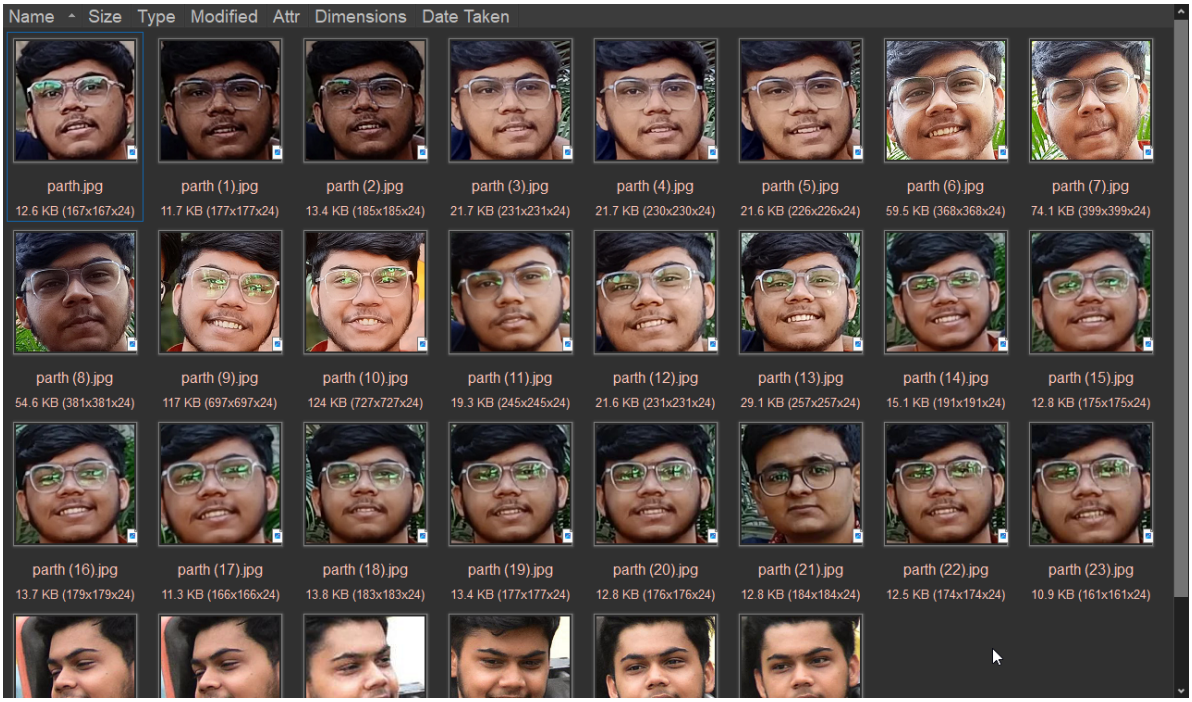


Figure 8.3: Parth's Segregated Images

Figure 8.4: Sourab's Segregated Images

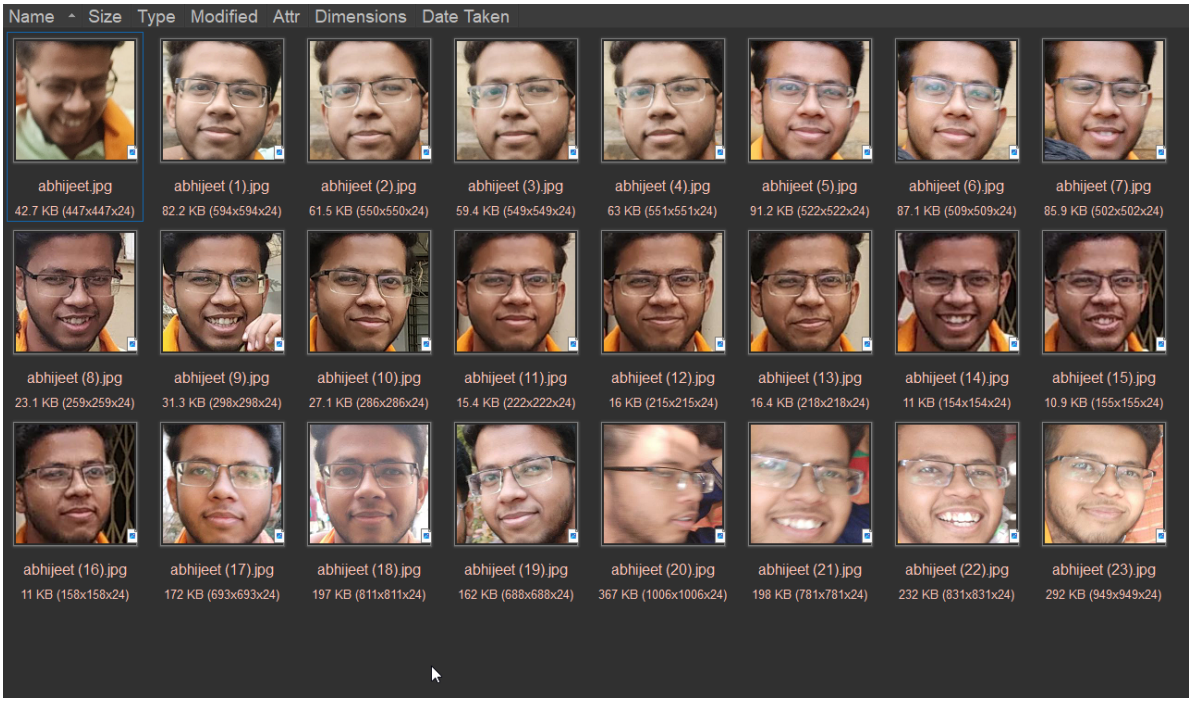


Figure 8.5: Abhijeet's Segregated Images

Chapter 9

Deployment Strategies

9.1 Deployment Environment

- The application is designed to be deployed on a cloud-based server, ensuring scalability and accessibility.
- The backend is built using FastAPI, which allows for efficient handling of requests and responses.
- The database is managed using MongoDB, providing a flexible and scalable solution for data storage.
- The frontend is developed using Flutter, enabling cross-platform compatibility for both Android and iOS devices.
- The application can be deployed on platforms like AWS, Google Cloud, or Azure, ensuring high availability and reliability.

9.2 Deployment Steps

1. Set up a cloud-based server with the required specifications (CPU, RAM, storage).
2. Install the necessary software dependencies (Python, FastAPI, MongoDB, Flutter).
3. Deploy the backend API using FastAPI and connect it to the MongoDB database.
4. Build and deploy the Flutter application for both Android and iOS platforms.
5. Configure domain and SSL certificates for secure access to the application.
6. Test the application in the deployed environment to ensure functionality and performance.
7. Monitor the application for any issues and optimize performance as needed.

9.3 Deployment Challenges

- Ensuring compatibility across different platforms and devices can be challenging, especially with Flutter.
- Managing database connections and ensuring data integrity during high traffic periods.
- Handling security concerns related to user data and ensuring compliance with data protection regulations.
- Optimizing the performance of the application to handle a large number of concurrent users.
- Ensuring proper documentation and support for users to facilitate smooth onboarding and usage of the application.

9.4 Security Concerns

- Implementing secure authentication mechanisms to protect user data and prevent unauthorized access.
- Encrypting sensitive data stored in the database to ensure data privacy and security.
- Regularly updating the application and its dependencies to patch any security vulnerabilities.
- Monitoring the application for any suspicious activities and implementing intrusion detection systems.
- Educating users about best practices for data security and privacy when using the application.

Chapter 10

Result and Analysis

10.1 Model 1: LBPH Face Recognizer

10.1.1 Confusion Metrics

Metric	Value
Accuracy	70%
Precision	85%
Recall	71%
F1 Score	73%

Figure 10.1: Confusion Matrix of LBPH Face Recognizer

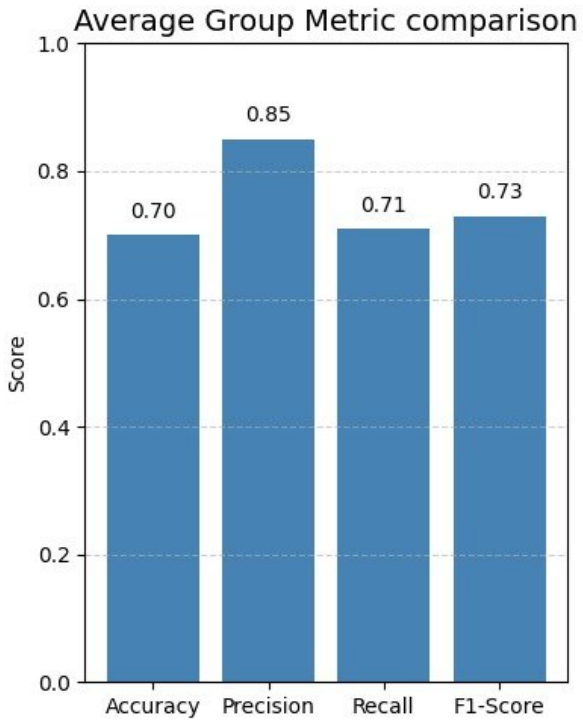


Figure 10.2: Accuracy per Person (Class) - LBPH Face Recognizer

10.1.2 Per Person Results

Graphs showing the accuracy of each person in the dataset. The results indicate that the model performs well for most individuals, with some variations in accuracy based on the number of images available for training.

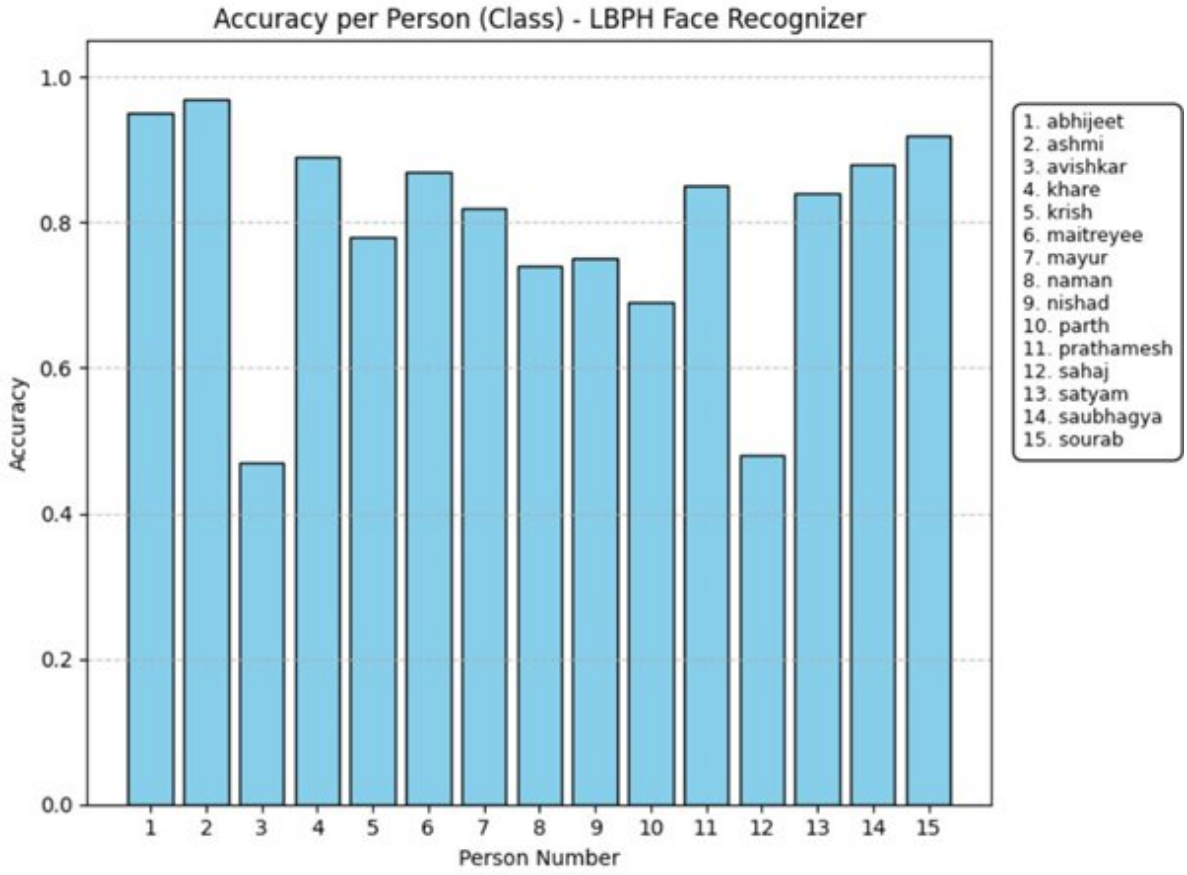


Figure 10.3: Accuracy per Person (Class) - LBPH Face Recognizer

10.2 Model 2: Facenet Face Recognizer

10.2.1 Confusion Metrics

Metric	Value
Accuracy	87.63%
Precision	81.63%
Recall	84.63%
F1 Score	82.63%

Figure 10.4: Confusion Matrix of Facenet Face Recognizer

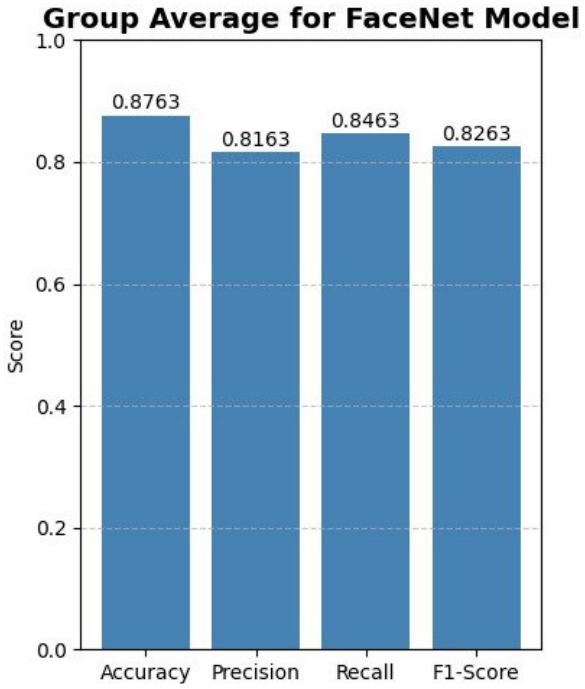


Figure 10.5: Accuracy per Person (Class) - Facenet Face Recognizer

10.2.2 Per Person Results

Graphs showing the accuracy of each person in the dataset. The results indicate that the model performs well for most individuals, with some variations in accuracy based on the number of images available for training.

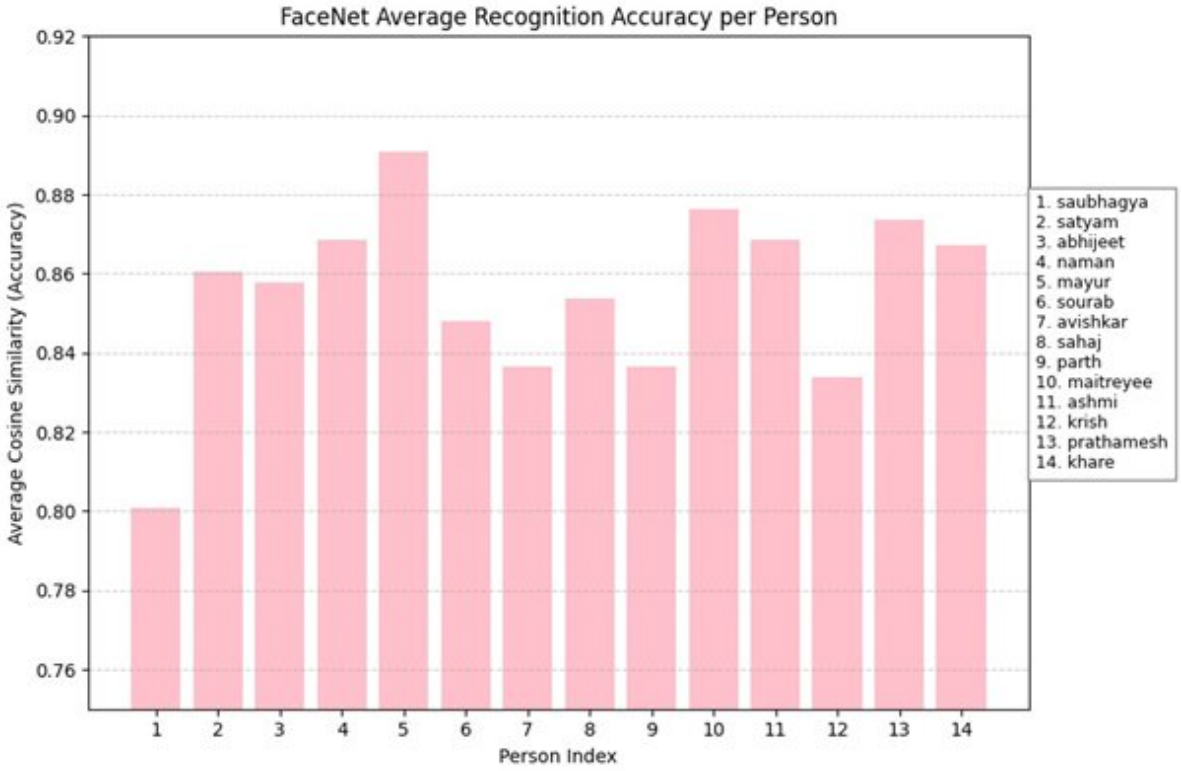


Figure 10.6: Accuracy per Person (Class) - Facenet Face Recognizer

10.3 Model 3: Resnet Face Recognizer

10.3.1 Confusion Metrics

Metric	Value
Accuracy	91.44%
Precision	88.73%
Recall	91.48%
F1 Score	89.32%

Figure 10.7: Confusion Matrix of Resnet Face Recognizer

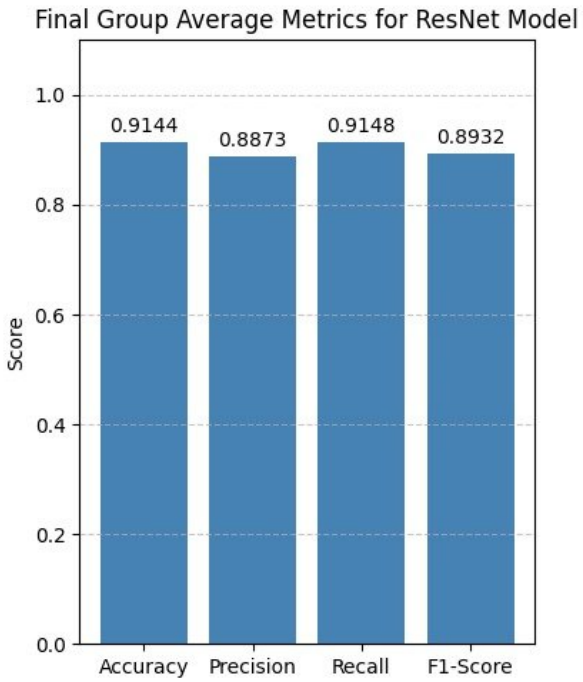


Figure 10.8: Accuracy per Person (Class) - Resnet Face Recognizer

10.3.2 Per Person Results

Graphs showing the accuracy of each person in the dataset. The results indicate that the model performs well for most individuals, with some variations in accuracy based on the number of images available for training.

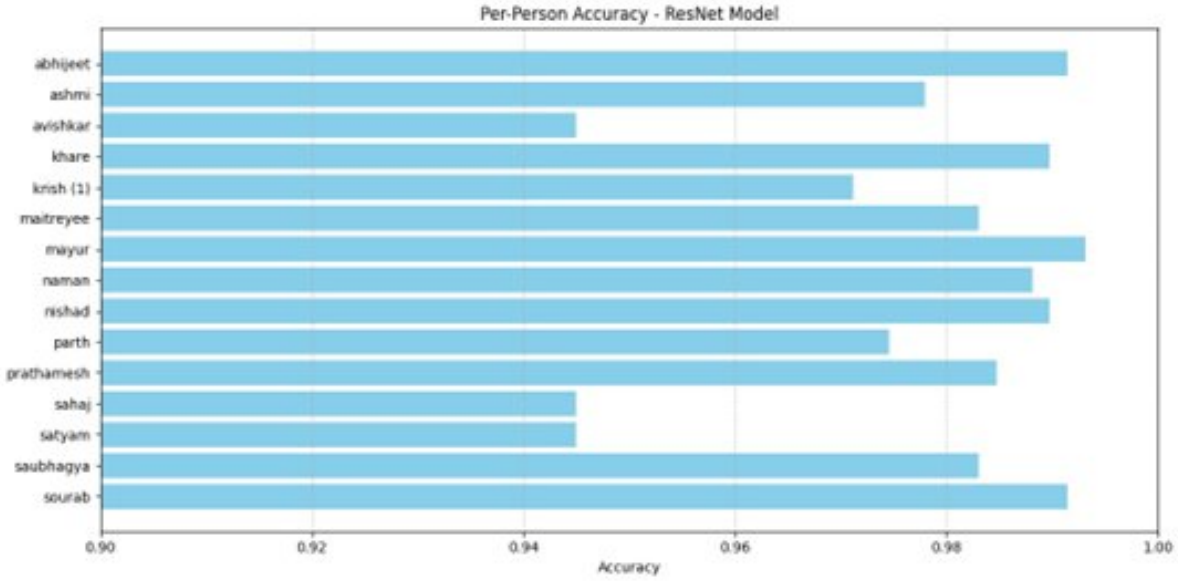


Figure 10.9: Accuracy per Person (Class) - Resnet Face Recognizer

10.4 Final Comparison

Resnet Face Recognizer outperformed the other two models in terms of accuracy, precision, recall, and F1 score. The results indicate that Resnet is the most effective model for face recognition tasks in this dataset.

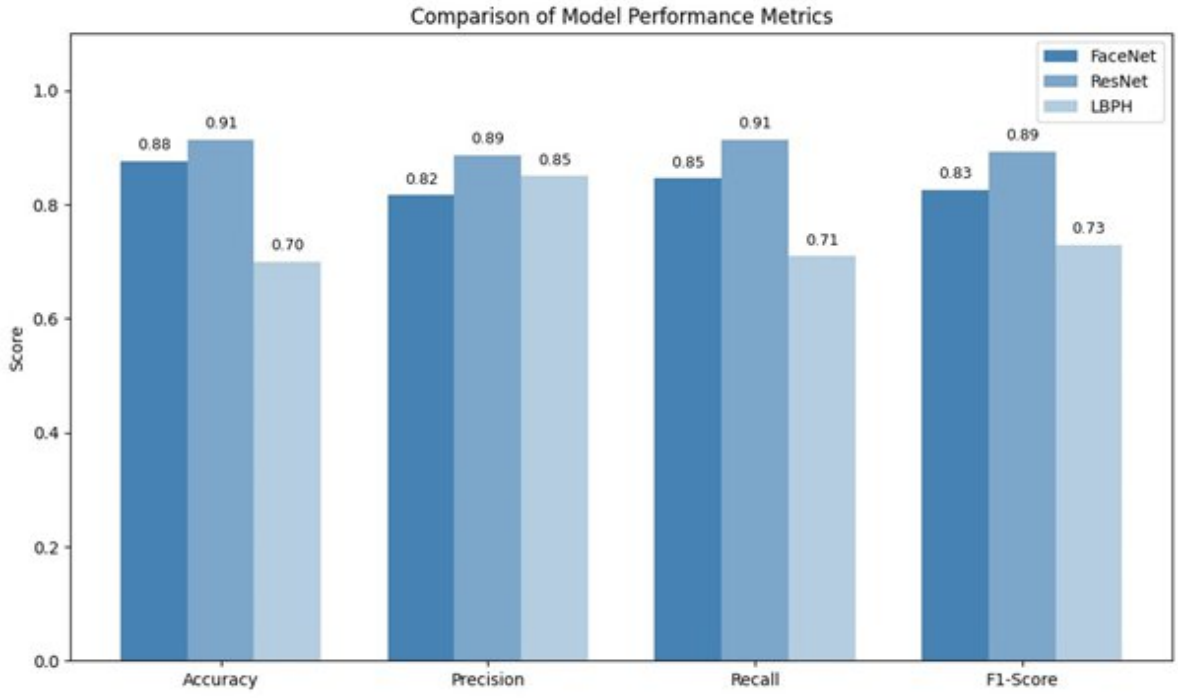


Figure 10.10: Final Model Comparison

Chapter 11

Applications

11.1 Educational Institutions

- **Automated Attendance Tracking:** The app can be used in schools, colleges, and universities to automate attendance marking, reducing manual errors and saving time.
- **Classroom Analytics:** Provides insights into Lectures attendance trends, helping educators identify patterns of absenteeism and take corrective actions.
- **Examination Monitoring:** Ensures only authorized students are present during exams by verifying their identity through facial recognition.

11.2 Corporate Environments

- **Employee Time Tracking:** Automates attendance logging for employees, integrating seamlessly with HR systems for payroll and performance evaluation.
- **Access Control:** Restricts access to secure areas within office premises by verifying employee identities.
- **Remote Work Monitoring:** Tracks attendance for remote employees during virtual meetings or work-from-home scenarios.

11.3 Healthcare Facilities

- **Patient Identification:** Ensures accurate identification of patients during check-ins, reducing errors in medical records.
- **Staff Attendance:** Tracks attendance of healthcare workers, ensuring adequate staffing levels at all times.
- **Visitor Management:** Monitors and records visitor entries for enhanced security in hospitals and clinics.

11.4 Government and Public Sector

- **Public Office Attendance:** Tracks attendance of government employees, ensuring accountability and transparency.
- **Voter Verification:** Can be used during elections to verify voter identities and prevent fraud.
- **Public Safety:** Enhances security in public spaces by identifying individuals in real-time.

11.5 Retail and Hospitality

- **Customer Personalization:** Identifies repeat customers and provides personalized services in retail stores and hotels.
- **Staff Management:** Tracks attendance and shift timings of employees in retail and hospitality sectors.
- **Security Monitoring:** Enhances security by identifying unauthorized individuals in restricted areas.

11.6 Event Management

- **Attendee Verification:** Verifies the identity of attendees at conferences, seminars, and other events.
- **Access Control:** Ensures only registered participants can access specific event areas.
- **Real-Time Analytics:** Provides insights into attendee participation and engagement during events.

11.7 Smart Cities and IoT Integration

- **Public Transport:** Tracks passenger attendance and ensures authorized access to public transport systems.
- **Smart Campus Integration:** Links attendance data with other smart campus systems like library access, cafeteria payments, and dormitory management.
- **Urban Security:** Enhances surveillance and security in urban areas by identifying individuals in real-time.

11.8 Research and Development

- **Algorithm Testing:** Provides a platform for testing and improving facial recognition algorithms.
- **Data Collection:** Facilitates the collection of diverse datasets for research purposes.
- **Prototype Development:** Serves as a base for developing new applications in facial recognition and attendance tracking.

Chapter 12

Conclusion

In this seminar, we have discussed the various face recognition algorithms and techniques used in the field of computer vision. We have compared the performance of these algorithms based on accuracy, efficiency, and scalability. We have also discussed the advantages and disadvantages of face recognition technology and its applications in real-world scenarios.

- Face recognition technology is a powerful biometric technology that can be used for a wide range of applications, from security to personalization.
- There are several face recognition algorithms and techniques available, each with its own strengths and weaknesses.
- By comparing the performance of different face recognition algorithms and techniques, we can gain insights into their suitability for different applications.
- The evaluation metrics provide a quantitative measure of the performance of face recognition algorithms and techniques, helping us identify the most suitable approach for a given application.
- Face recognition technology has the potential to revolutionize various industries and improve the quality of life for individuals by providing secure and personalized services.

Chapter 13

Future Prospects for the Attendance Assistant

Building on our implementation of an automated facial-recognition-based attendance assistant, several avenues can significantly enhance its performance, scalability, and real-world viability:

13.1 Integration of advanced Deep-Learning Models

- **Adopt Lightweight CNNs and Vision Transformers:** Replace or augment traditional feature-based methods (e.g., Eigenfaces, Fisherfaces) with compact convolutional neural networks (MobileNet, EfficientNet) or vision-transformer variants to boost accuracy under varied lighting and poses—while still enabling on-device inference.
- **Hybrid Pipeline Architecture:** Combine fast, classical face detection (e.g., OpenCV Haar cascades) with a secondary deep-learning re-identification stage to balance speed and precision in live classroom or office settings.

13.2 Dataset Expansion and Synthetic Augmentation

- **Larger, More Diverse Training Sets:** Scale beyond our initial 5-person dataset (=18k crops) by collecting images across multiple sessions, cameras, and demographics to reduce bias and improve generalization.
- **GAN-Based Augmentation:** Leverage generative adversarial networks to synthesize varied facial expressions, occlusions (masks, scarves), and lighting conditions—ensuring robust attendance capture even when subjects wear accessories or move dynamically.

13.3 Edge Deployment and Resource Optimization

- **Model Compression Techniques:** Apply quantization and pruning to shrink model size for deployment on low-power edge devices (e.g., Raspberry Pi, embedded IP cameras), minimizing latency in real-time roll-call scenarios.
- **Hardware Acceleration:**Utilize on-board NPUs or Coral TPUs to offload inference, enabling simultaneous multi-camera streams for large lecture halls or multiple office entrances.

13.4 Privacy, Security, and Ethical Considerations

- **Federated Learning and on-Device Training:**Implement a federated learning framework so endpoints (e.g., classroom tablets) collaboratively improve the recognition model without sharing raw images—protecting sensitive biometric data by sharing only encrypted weight updates.
- **Anti-Spoofing and liveness Detection:** Integrate texture analysis and micro-motion cues to distinguish live faces from photographs or video replays, safeguarding against presentation attacks
- **Bias Auditing:** Regularly evaluate performance metrics (accuracy, false positives/negatives) across gender, age, and skin-tone strata to identify and mitigate algorithmic bias.

13.5 Mutlti-Modal and Context-Aware Fusion:

- **Biometric Fusion:** Biometric Fusion Augment facial data with voice recognition or RFID badge readings to confirm identity when face confidence scores drop below a threshold (e.g., under poor lighting).
- **Environmental Sensing:** Dynamically adjust recognition thresholds based on scene context—bright vs. dim classrooms, seated vs. standing students—to maintain both high recall and precision.

13.6 Broader Applications and Commercialization

- **Enterprise Time-Tracking Systems:** Extend the attendance assistant to corporate environments, integrating with HR systems for automated timekeeping and employee verification at entrances.
- **Smart-campus and IoT Integration:** Link attendance data with campus access control, library entry logs, and canteen payments to create a unified student experience.
- **Analytics Dashboard:** Offer administrators real-time dashboards showing attendance trends, tardiness patterns, and automated alerts for absenteeism spikes.

By pursuing these directions—grounded in our core implementation and the comparative analyses documented earlier—our Attendance Assistant can evolve into a robust, privacy-preserving, and widely deployable solution for both educational institutions and enterprises alike.

Bibliography

[1] Paul, Sanmoy and Acharya, Sameer Kumar, A Comparative Study on Facial Recognition Algorithms (December 21, 2020). e-journal - First Pan IIT International Management Conference – 2018, Available at SSRN: https://ssrn.com/abstract=3753064 or http://dx.doi.org/10.2139/ssrn.3753064

[2] Kaur, P., Krishan, K., Sharma, S.K. and Kanchan, T., 2020. Facial-recognition algorithms: A literature review. Medicine, Science and the Law, 60(2), pp.131-139.

[3] Kukula EP, Elliott SJ. Evaluation of a facial recognition algorithm across three illumination conditions. IEEE Aerospace and Electronic Systems Magazine. 2004 Sep;19(9):19-23.

[4] Kukula EP, Elliott SJ. Evaluation of a facial recognition algorithm across three illumination conditions. IEEE Aerospace and Electronic Systems Magazine. 2004 Sep;19(9):19-23.

[5] Emami S, Suciu VP. Facial recognition using OpenCV. Journal of Mobile, Embedded and Distributed Systems. 2012 Mar 30;4(1):38-43.

[6] Chen J, Jenkins WK. Facial recognition with PCA and machine learning methods. In2017 IEEE 60th international Midwest symposium on circuits and systems (MWSCAS) 2017 Aug 6 (pp. 973-976). IEEE.

[7] Schenkel T, Ringhage O, Branding N. A Comparative Study of Facial Recognition Techniques: With focus on low computational power.

[8] Paul, S. and Acharya, S.K., 2020, December. A comparative study on facial recognition algorithms. In e-journal-First Pan IIT International Management Conference–2018.

[9] Delbiaggio, N., 2017. A comparison of facial recognition's algorithms.

[10] Coe, J. and Atay, M., 2021. Evaluating impact of race in facial recognition across machine learning and deep learning algorithms. Computers, 10(9), p.113.

[11] Dirin, Amir, Nicolas Delbiaggio, and Janne Kauttonen. "Comparisons of facial recognition algorithms through a case study application." (2020): 121-133.

Chapter 14

Individual Contribution

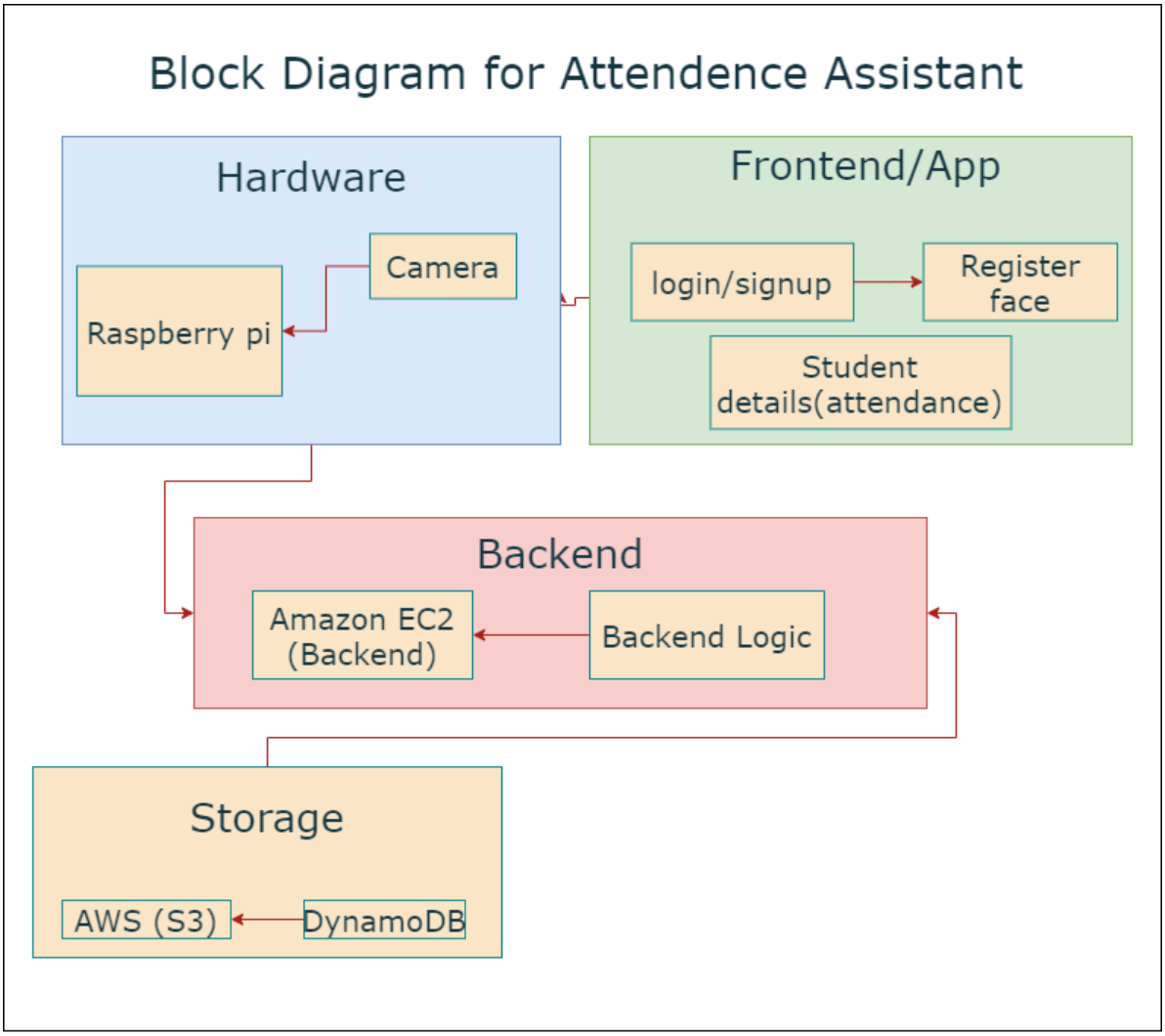


Figure 14.1: Block Diagram highlighting the modules supported by Parth Zarekar.

14.1 Problem Statement

Design and implement the backend API and face-recognition engine for the Attendance-Assistant system.

14.2 Student Details

Krishnaraj Thadesar
PRN: 1032210888
Roll Number: 15
Panel: A

14.3 Module Title

Backend & Face-Recognition Engine

14.4 Project's Module Scope (Individual Perspective)

End-to-end implementation of all backend services, face-encoding storage and lookup, handling concurrent API calls from clients, all hosted locally via Docker.

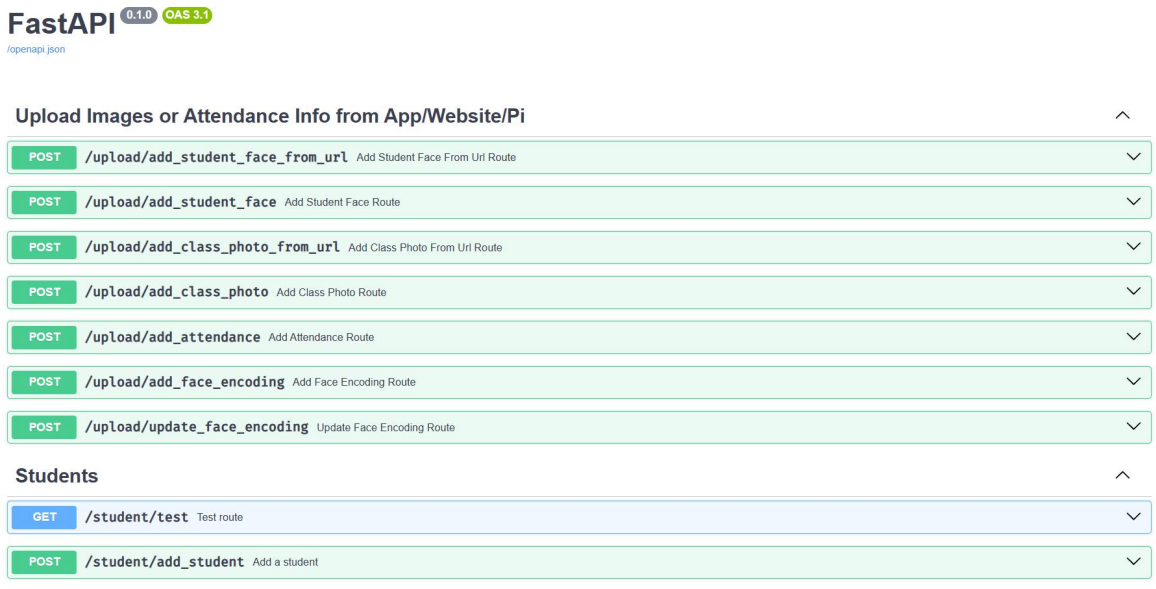


Figure 14.2: Swagger UI for API documentation (Krishnaraj Thadesar's contribution).

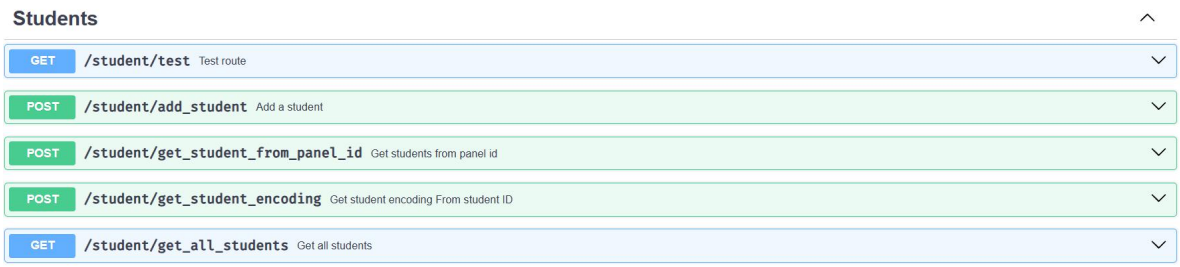


Figure 14.3: Swagger UI for API documentation (Krishnaraj Thadesar's contribution).

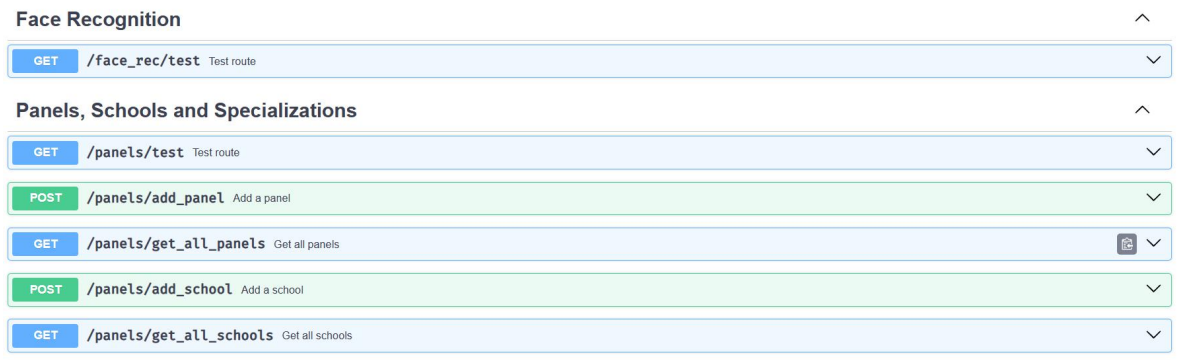


Figure 14.4: Swagger UI for API documentation (Krishnaraj Thadesar's contribution).

Module Interfaces

The FastAPI application exposes the following routes (defined in `main.py` and router files):

- **Add Attendance** `POST /api/v1/add_attendance` *Request body:*

```
{
  "room_id": "Room ID",
  "subject_id": "Subject ID",
  "teacher_id": "Teacher ID",
  "panel_id": "Panel ID",
  "start_time": "10:00",
  "end_time": "11:00"
}
```

- **Add Image** `POST /api/v1/add_image` *Request body:*

```
{
  "room_id": "Room ID",
  "image": "Base64-encoded image"
}
```

Response:

```
{
  "status": "success",
  "message": "Image added successfully"
}
```

- **Add Specialization** `POST /api/v1/add_specialization`
- **Add School** `POST /api/v1/add_school`
- **Add Panel** `POST /api/v1/add_panel`
- **Add Student** `POST /api/v1/add_student`
- **Add Face Image** `POST /api/v1/add_face_image`
- **Add Face Encoding** `POST /api/v1/add_face_encoding`
- **Add Teacher** `POST /api/v1/add_teacher`
- **Add Semester** `POST /api/v1/add_semester`
- **Add Subject** `POST /api/v1/add_subject`
- **Get Students** `POST /api/v1/get_students`
- **Get Teachers** `POST /api/v1/get_teachers`

Module Dependencies

- `face_recognition` → `dlib`, `numpy`
- `FastAPI` → `uvicorn`, `pydantic`
- MongoDB driver (`motor`)

Module Design

Layered architecture: Controller → Service → Model → Persistence; singleton face-model loader; JWT authentication middleware.

Module Implementation

- Containerized services with Docker Compose.
- Approximately 1,200 lines of Python code.
- Integrated face_recognition pipeline with error handling.

Module Testing Strategies

- Unit tests via `pytest` (coverage >= 85%).
- Mocked face detection for CI.
- Postman end-to-end smoke tests.

Module Deployment

- Fully hosted on local Docker Compose setup.
- Single-command bring-up of all services (backend, database, model).
- Manual rollback by re-deploying previous Docker image versions.

Chapter 15

Individual Contribution

15.1 Problem Statement

Support the full-stack development cycle by contributing to UI design, API development, research, testing, and deployment for the Attendance-Assistant system.

15.2 Student Details

Parth Zarekar
PRN: 1032210846
Roll Number: 09
Panel: A

15.3 Module Title

Full-Stack Support & Research

15.4 Project Module Scope

Assisted across UI design, backend API development, model-training research, paper drafting, testing, and deployment.

Attendance

LOGICAL DATA SIZE: 24.28KB STORAGE SIZE: 432KB INDEX SIZE: 400KB TOTAL COLLECTIONS: 12

Collection Name	Documents	Logical Data Size	Avg Document Size
buildings	7	461B	66B
classes	25	14.88KB	610B
encodings	12	2.2KB	188B
lectureImages	12	2.45KB	210B
panels	2	676B	338B
rooms	3	114B	38B
schools	1	116B	116B
semesters	1	330B	330B
specializations	2	156B	78B
students	8	2.47KB	317B
subjects	2	124B	62B
teachers	2	353B	177B

Figure 15.1: MongoDB Collections (Parth Zarekar's area)

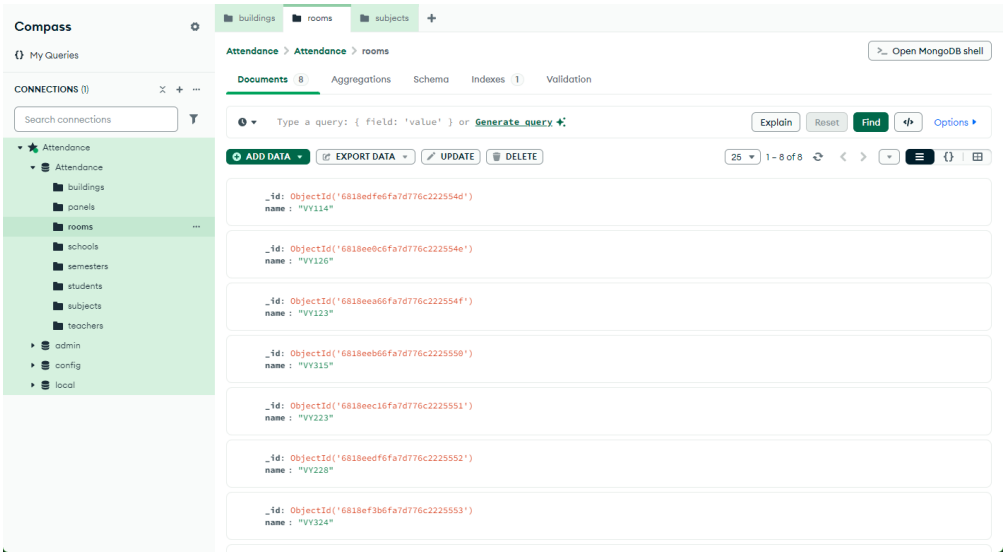


Figure 15.2: MongoDB Collections (Parth Zarekar's area)

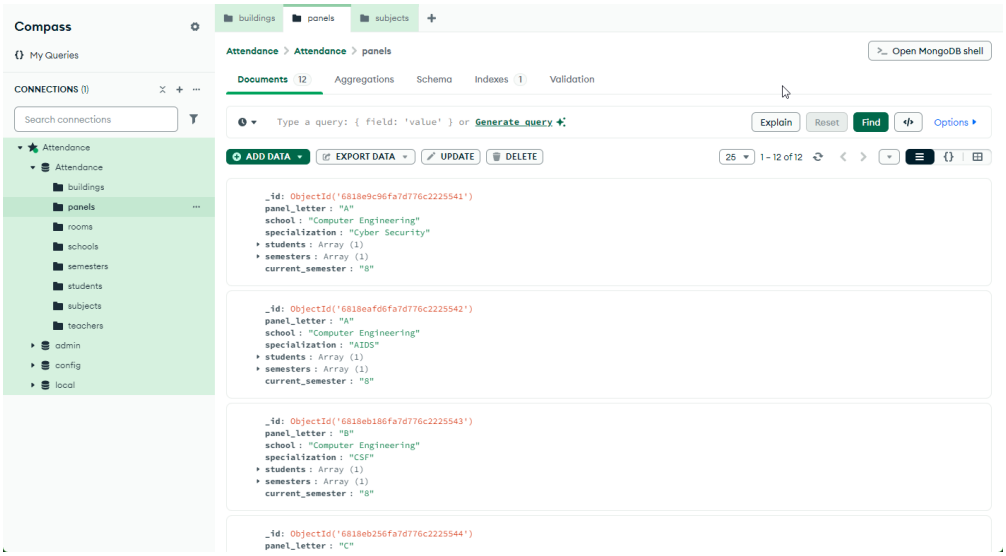


Figure 15.3: MongoDB Collections (Parth Zarekar's area)

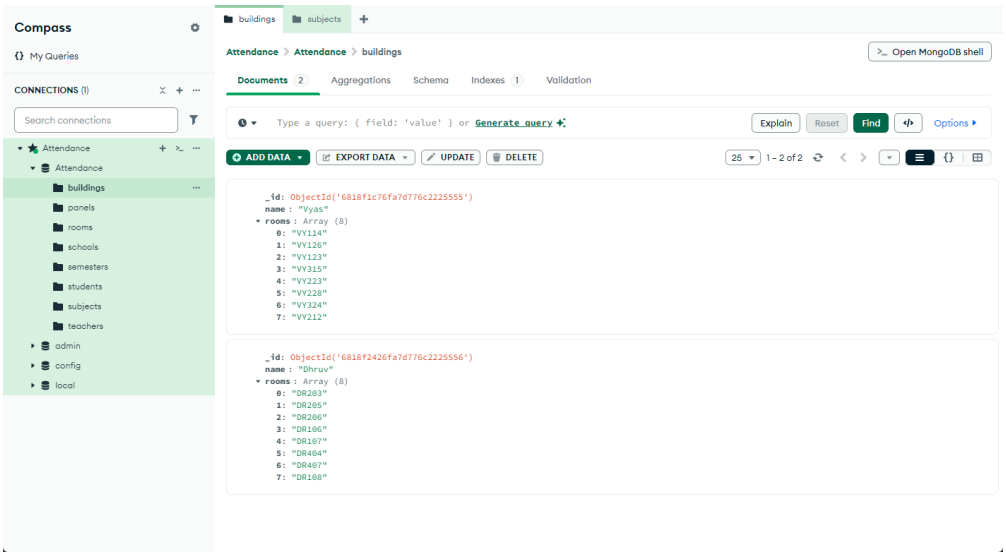


Figure 15.4: MongoDB Collections (Parth Zarekar's area)

15.5 Project Modules – Individual Contribution

1. **Frontend:** Provided feedback and enhancements on Figma wireframes and UI flows.
2. **Backend API:** Implemented core endpoints for image upload, face encoding, and attendance marking.
3. **Model Research:** Supported training experiments and benchmark comparisons for face-recognition models.
4. **Literature Research:** Drafted and edited sections of the project research paper on algorithm selection.
5. **Testing:** Created and executed end-to-end tests (API smoke tests, basic UI checks).

6. **Deployment:** Deployed Dockerized services to a basic AWS environment and configured DynamoDB storage.

Chapter 16

Individual Contribution

16.1 Problem Statement

Evaluate and benchmark multiple face-recognition algorithms; support model selection and integration.

16.2 Student Details

Sourab Karad
PRN: 1032211150
Roll Number: 40
Panel: A

16.3 Module Title

Algorithm Research & Model Integration

16.4 Project Module Scope

Implementation and evaluation of face-recognition methods; performance reporting and API stub delivery.

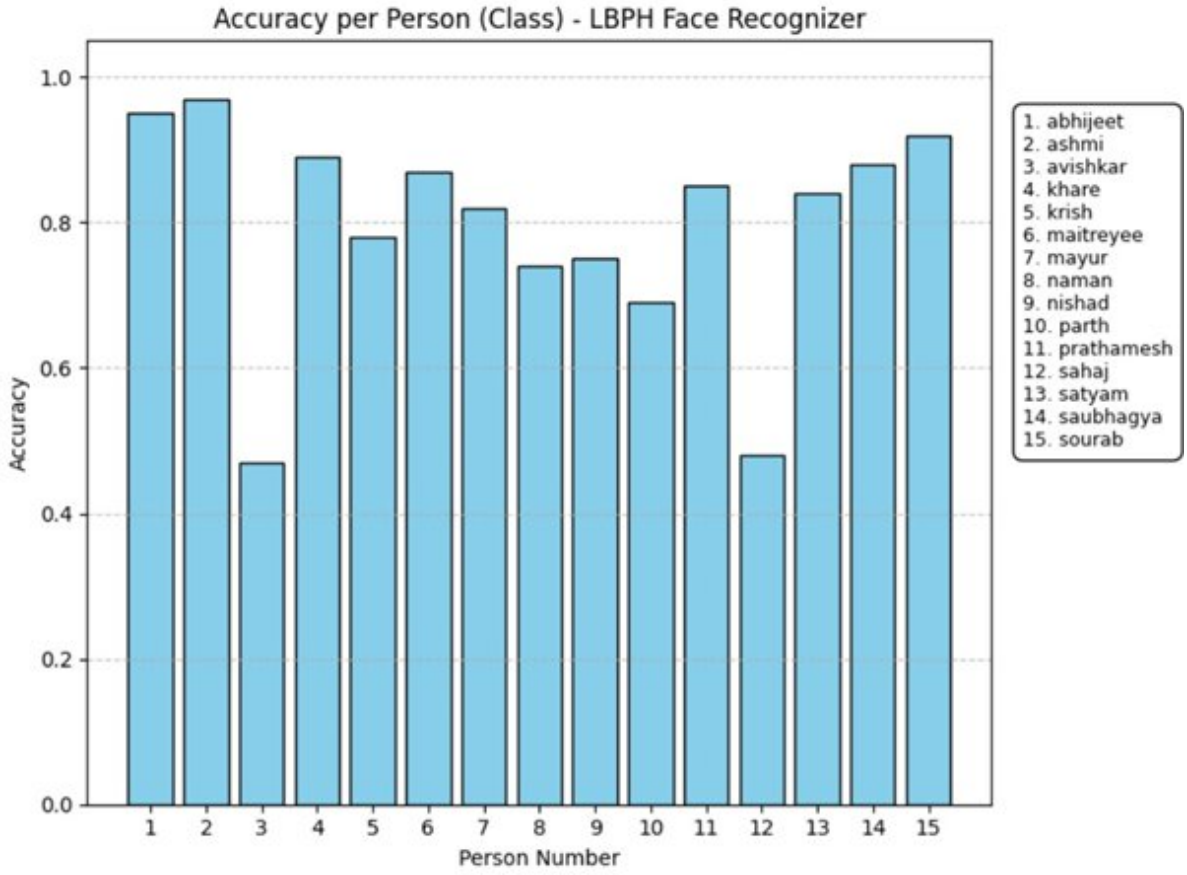


Figure 16.1: Accuracy per Person (Class) - LBPH Face Recognizer

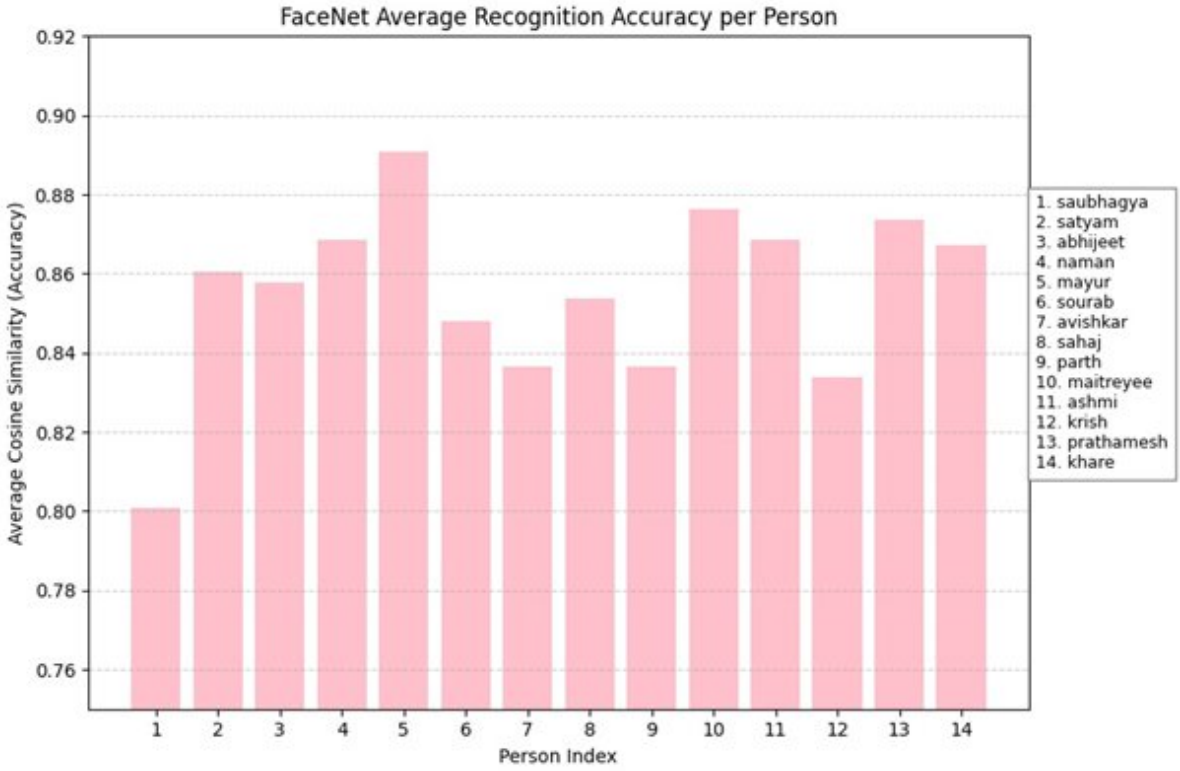


Figure 16.2: Accuracy per Person (Class) - Facenet Face Recognizer

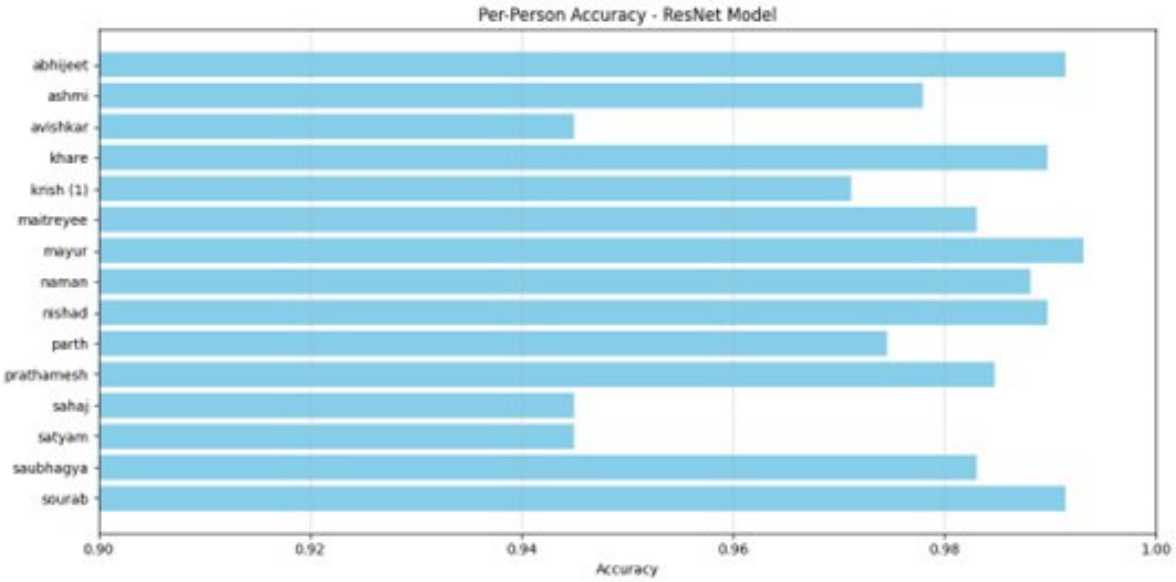


Figure 16.3: Accuracy per Person (Class) - Resnet Face Recognizer

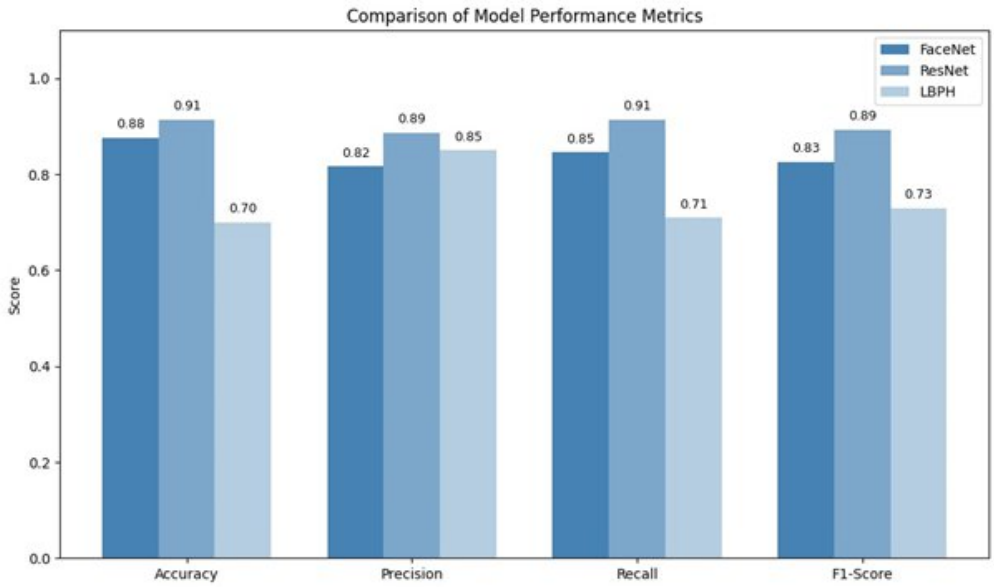


Figure 16.4: Final Model Comparison (Sourab Karad's results)

16.5 Project Modules – Individual Contribution

1. **Hardware & Software requirements:** GPU (RTX 2060), dlib, OpenCV, torch, scikit-learn, pandas.
2. **Module Interfaces:** `train_model.py`, `evaluate.py`; JSON output (`accuracy, precision, recall`).
3. **Module Dependencies:** torch→torchvision; face_recognition→dlib; numpy→pandas.
4. **Module Design:** Abstract base classes; modular trainer & evaluator.
5. **Module Implementation:** 800 LOC benchmarking harness; comparative plots in report.
6. **Testing Strategies:** 5-fold cross-validation; confusion matrices.
7. **Deployment:** Packaged ResNet model as pickle; provided Dockerfile snippet.

Chapter 17

Individual Contribution

17.1 Problem Statement

Design and build the cross-platform mobile app for attendance marking via facial capture.

17.2 Student Details

Saubhagya Singh
PRN: 1032211144
Roll Number: 38
Panel: A

17.3 Module Title

Flutter Front-End Application

17.4 Project Module Scope

Implement the Flutter-based UI for login, camera capture, attendance display, and offline support.

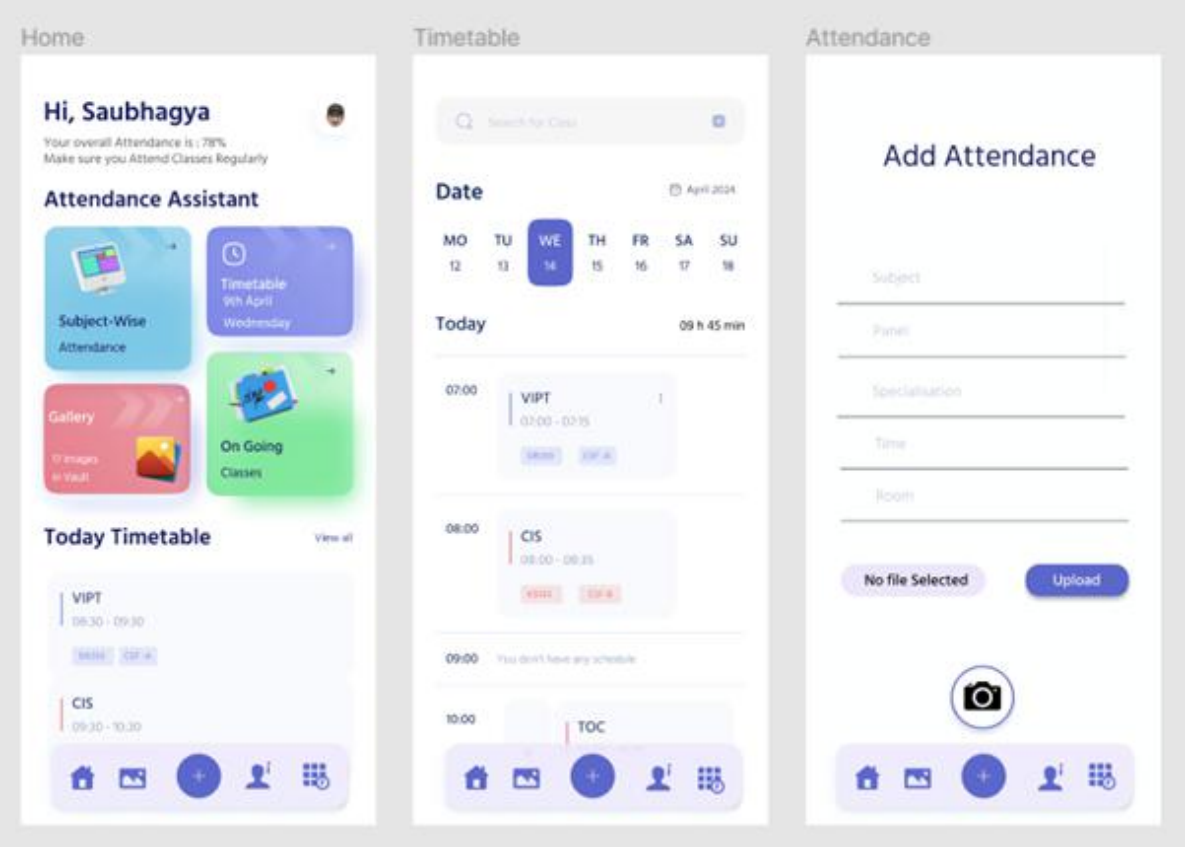


Figure 17.1: Frontend (Saubhagya Singh's contribution).

17.5 Project Modules – Individual Contribution

1. **UI Design:** Assisted in Figma wireframes and refined user flows.
2. **Flutter Development:** Built screens for login, camera preview, and attendance history.
3. **Camera Integration:** Integrated device camera plugin and handled image capture.
4. **Offline Support:** Added basic local caching to queue captures when offline.
5. **Testing:** Performed manual UI tests on both Android and iOS emulators.

Appendix A

Publication Details

Appendix B

Base Paper

Appendix C

Plagiarism Report