Machine Learning Powered Automated Facial Attendanc Tracking System

Mid Term Status and Progress Evaluation

April 24, 2024

Content

Status Overview

Frontend Technologies

Frontend Drafts

Frontend Work Remaining

Backend Technologies

Backend Drafts

Backend Work Done

Face Recognition Libraries Used

Training Data

Preliminary Results

Backend Work Remaining

Status Overview

Frontend

1. App Implementation is 50 % complete.
2. Integration with Backend is pending.

Backend

1. Backend Implementation of APIs is 80 % complete.
2. Core functions are working.
3. Integration with Frontend is pending.
4. Improvement in Face Recognition is pending.

Frontend Technologies

Libraries Used for Creating Cross Platform App

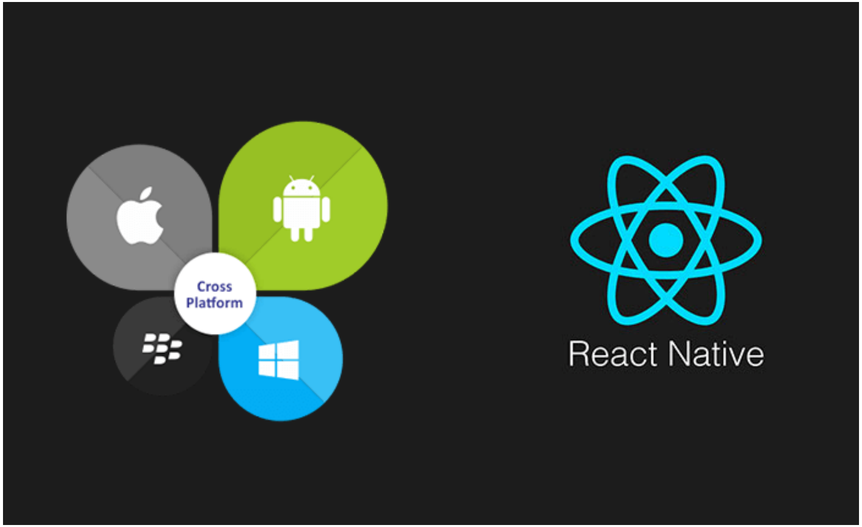
1. React Native
2. Axios
3. React Navigation

React Native

1. React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android.
2. It's based on React, Facebook's JavaScript library for building user interfaces, but instead of targeting the browser, it targets mobile platforms.
3. Its Cross Platform, meaning it will work on both iOS and Android, *which are the primary Operating systems our teachers use.*

React Native

Cross Platform App Development



Frontend Drafts

These are the initial drafts of the app

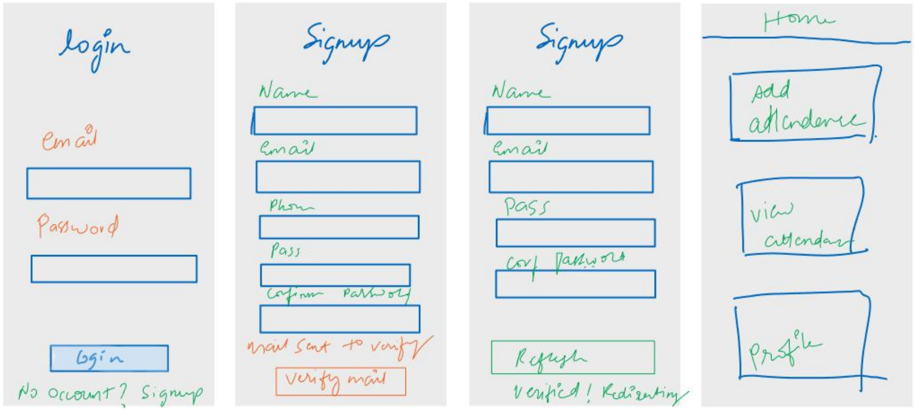
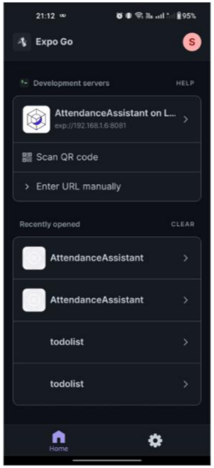


Figure: Drafts of the App

Frontend Progress

Current Progress in App Development (Mid Term Stage)

Current Development going on on Expo (React Native support framework for Application Development)



21:12 95%

Expo Go

Development servers HELP

AttendanceAssistant on L...
exp://192.168.1.6:8081

Scan QR code

Enter URL manually

Recently opened CLEAR

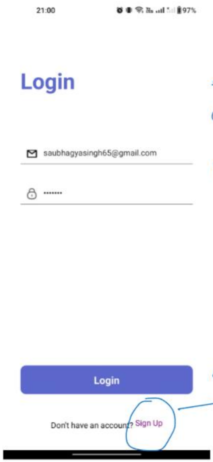
AttendanceAssistant

AttendanceAssistant

todolist

todolist

Home



21:00 97%

Login

saubhagyasingh65@gmail.com

•••••••

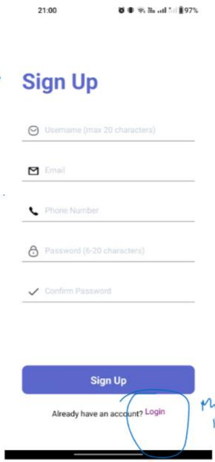
Login

Don't have an account? Sign Up

$\longleftrightarrow$

Connection to Backend Pending.

Minor Bug $\rightarrow$



21:00 97%

Sign Up

Username (max 20 characters)

Email

Phone Number

Password (6-20 characters)

Confirm Password

Sign Up

Already have an account? Login

Minor Bug

Frontend Progress

Current Progress in App Development (Mid Term Stage) Continued

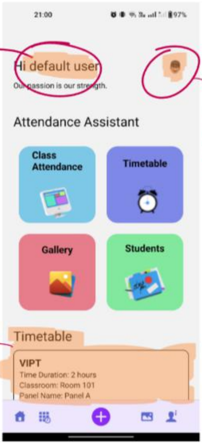
Current Development going on on Expo (React Native support framework for Application Development)

Username update pending

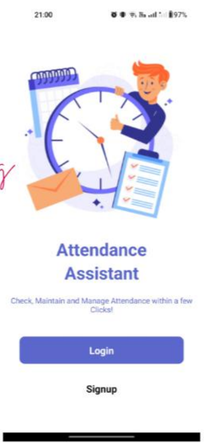
P f P Pending

(More pages pending)

UI changes pending



VIPT
Time Duration: 2 hours
Classroom: Room 101
Panel Name: Panel A



Frontend Work Remaining

1. Integration with Backend.
2. Improving User Interface.
3. Adding Core Features of taking photos and uploading Attendance

Backend Technologies

1. FastAPI for creating APIs.
2. MongoDB for Database.
3. Multiple Face Recognition Libraries
4. Python for Backend Development
5. Docker for Containerization
6. Swagger for API Documentation

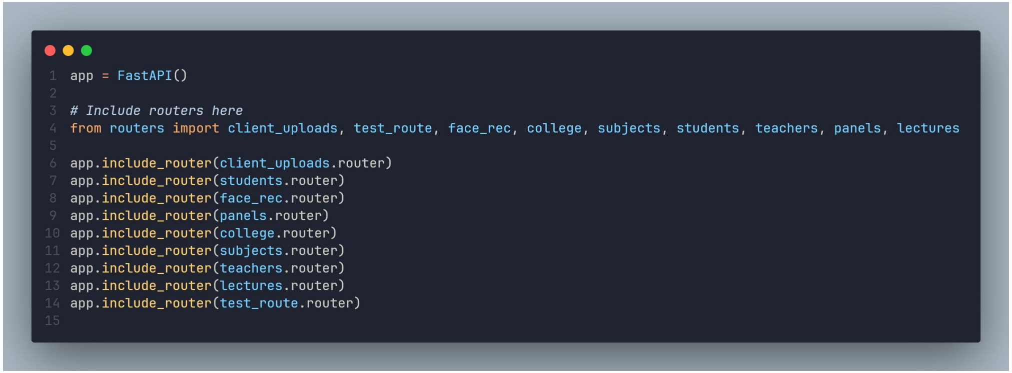
FastAPI

Overview

1. FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.
2. It is based on standard Python type hints, which makes it easy to use and understand.
3. It is one of the fastest Python frameworks available.

FastAPI

In Use



```python
app = FastAPI()

# Include routers here
from routers import client_uploads, test_route, face_rec, college, subjects, students, teachers, panels, lectures

app.include_router(client_uploads.router)
app.include_router(students.router)
app.include_router(face_rec.router)
app.include_router(panels.router)
app.include_router(college.router)
app.include_router(subjects.router)
app.include_router(teachers.router)
app.include_router(lectures.router)
app.include_router(test_route.router)

```

Swagger for API Documentation

Overview

1. Swagger is a set of open-source tools built around the OpenAPI Specification that can help you design, build, document and consume REST APIs.
2. It is used to document the APIs in a user friendly way.
3. It is used to test the APIs.
4. It is used to generate client libraries for the APIs.
5. It is used to generate server stubs for the APIs.
6. It is default with FastAPI.

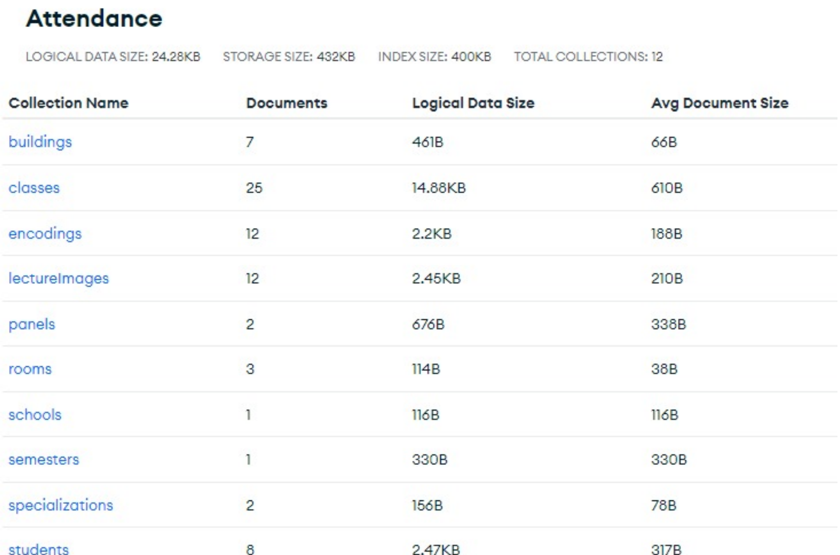
MongoDB for Database

Overview

1. MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era.
2. It is a NoSQL database, which means it stores data in JSON-like documents.
3. It is used to store the data of the students and their faces.

MongoDB for Database

In Use



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

Backend Drafts

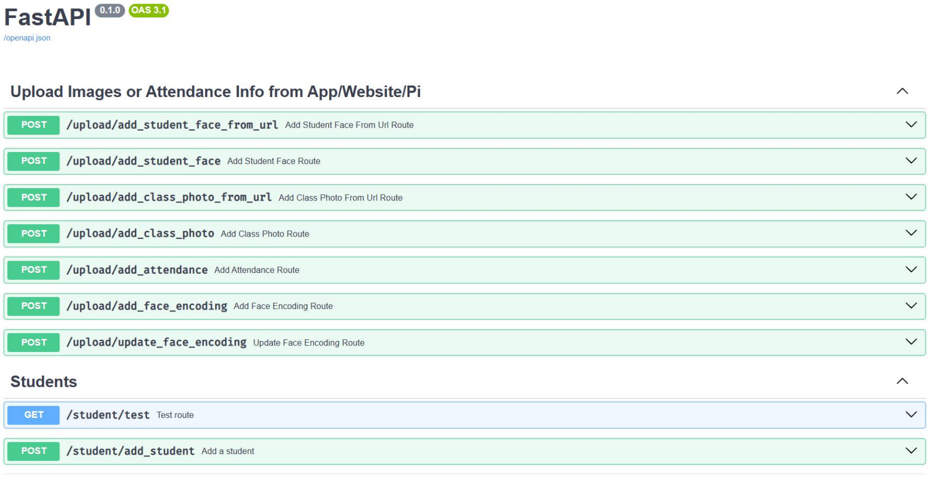
the readme file from github etc

Backend Work Done

1. Core APIs are working.
2. Face Recognition is working.
3. Database is working.
4. Swagger Documentation is complete.
5. Docker Containerization is complete.

API

Uploading Images from App or Pi or Website



FastAPI 0.1.0 OAS 3.1

/openapi.json

Upload Images or Attendance Info from App/Website/Pi

POST `/upload/add_student_face_from_url` Add Student Face From Url Route

POST `/upload/add_student_face` Add Student Face Route

POST `/upload/add_class_photo_from_url` Add Class Photo From Url Route

POST `/upload/add_class_photo` Add Class Photo Route

POST `/upload/add_attendance` Add Attendance Route

POST `/upload/add_face_encoding` Add Face Encoding Route

POST `/upload/update_face_encoding` Update Face Encoding Route

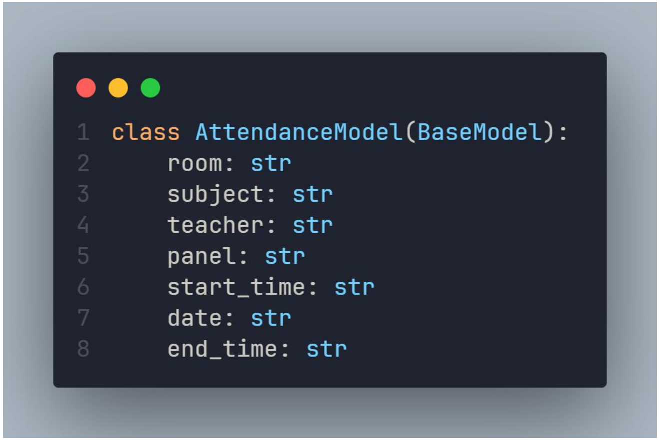
Students

GET `/student/test` Test route

POST `/student/add_student` Add a student

API

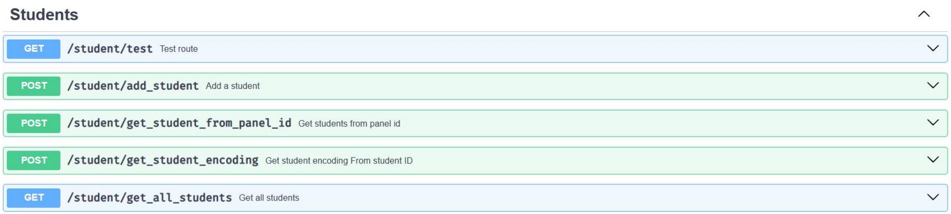
The Model to add attendance (from Teachers App only)



```python
class AttendanceModel(BaseModel):
    room: str
    subject: str
    teacher: str
    panel: str
    start_time: str
    date: str
    end_time: str
```

API

Students API



Students

GET /student/test Test route

POST /student/add_student Add a student

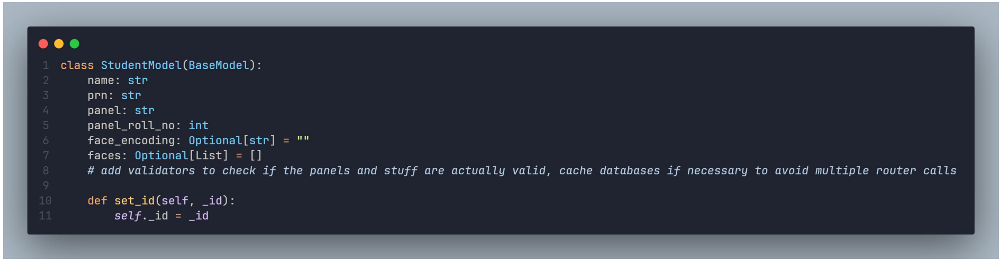
POST /student/get_student_from_panel_id Get students from panel id

POST /student/get_student_encoding Get student encoding From student ID

GET /student/get_all_students Get all students

API

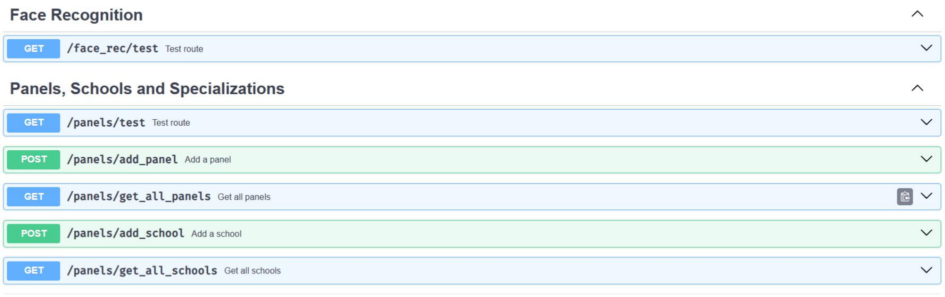
Students Models



```python
class StudentModel(BaseModel):
    name: str
    prn: str
    panel: str
    panel_roll_no: int
    face_encoding: Optional[str] = ""
    faces: Optional[List] = []
    # add validators to check if the panels and stuff are actually valid, cache databases if necessary to avoid multiple router calls

    def set_id(self, _id):
        self._id = _id
```

API

Face Recognition and Panels



Face Recognition

GET /face_rec/test Test route

Panels, Schools and Specializations

GET /panels/test Test route

POST /panels/add_panel Add a panel

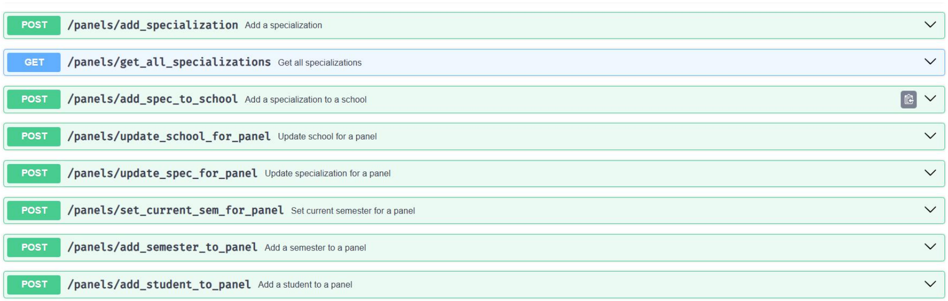
GET /panels/get_all_panels Get all panels

POST /panels/add_school Add a school

GET /panels/get_all_schools Get all schools

API

Panels Continued



POST /panels/add_specialization Add a specialization

GET /panels/get_all_specializations Get all specializations

POST /panels/add_spec_to_school Add a specialization to a school

POST /panels/update_school_for_panel Update school for a panel

POST /panels/update_spec_for_panel Update specialization for a panel

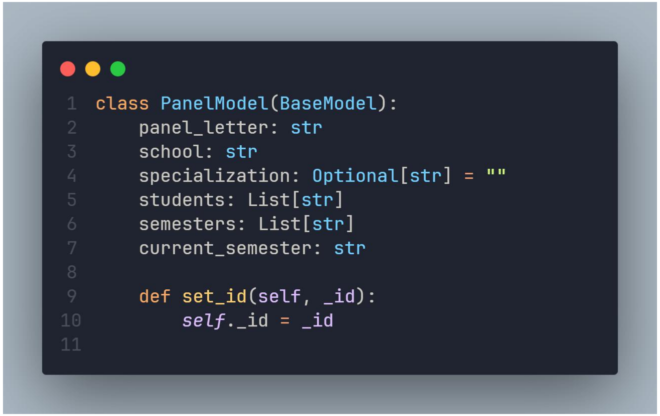
POST /panels/set_current_sem_for_panel Set current semester for a panel

POST /panels/add_semester_to_panel Add a semester to a panel

POST /panels/add_student_to_panel Add a student to a panel

API

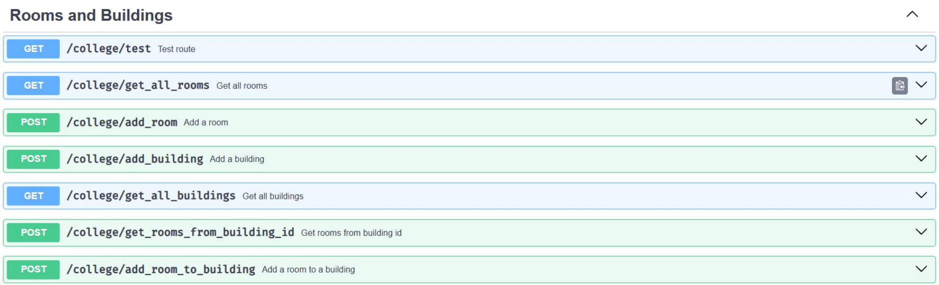
Panel Models



```python
class PanelModel(BaseModel):
    panel_letter: str
    school: str
    specialization: Optional[str] = ""
    students: List[str]
    semesters: List[str]
    current_semester: str

    def set_id(self, _id):
        self._id = _id

```

API

Rooms and Buildings



Rooms and Buildings

GET /college/test Test route

GET /college/get_all_rooms Get all rooms

POST /college/add_room Add a room

POST /college/add_building Add a building

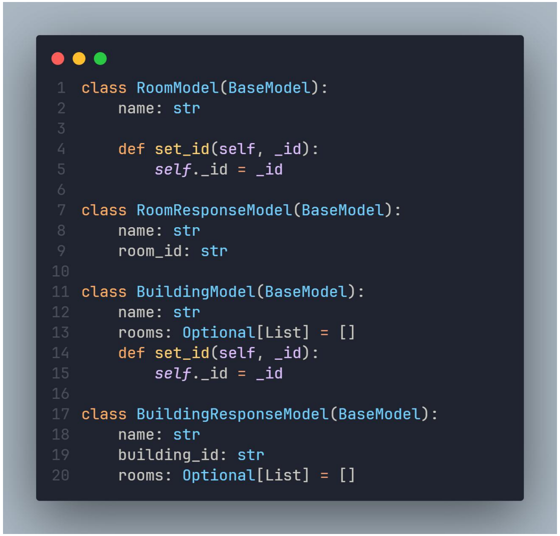
GET /college/get_all_buildings Get all buildings

POST /college/get_rooms_from_building_id Get rooms from building id

POST /college/add_room_to_building Add a room to a building

API

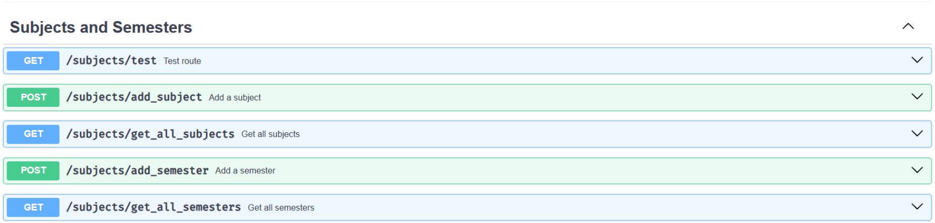
Models to Manage Rooms and Buildings



```python
class RoomModel(BaseModel):
    name: str

    def set_id(self, _id):
        self._id = _id

class RoomResponseModel(BaseModel):
    name: str
    room_id: str

class BuildingModel(BaseModel):
    name: str
    rooms: Optional[List] = []
    def set_id(self, _id):
        self._id = _id

class BuildingResponseModel(BaseModel):
    name: str
    building_id: str
    rooms: Optional[List] = []
```

API

Subjects and Semester



Subjects and Semesters

GET /subjects/test Test route

POST /subjects/add_subject Add a subject

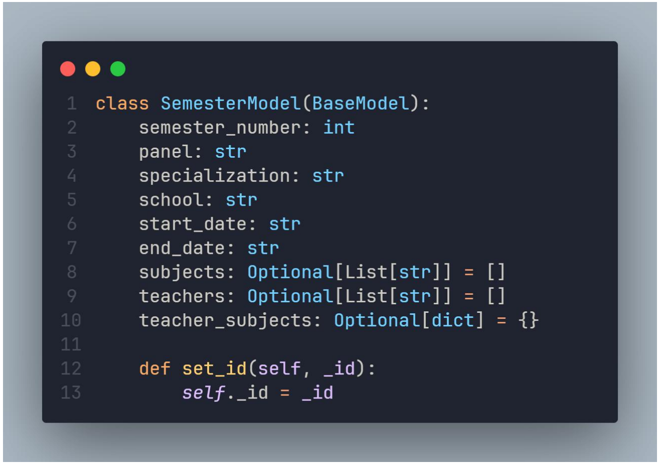
GET /subjects/get_all_subjects Get all subjects

POST /subjects/add_semester Add a semester

GET /subjects/get_all_semesters Get all semesters

API

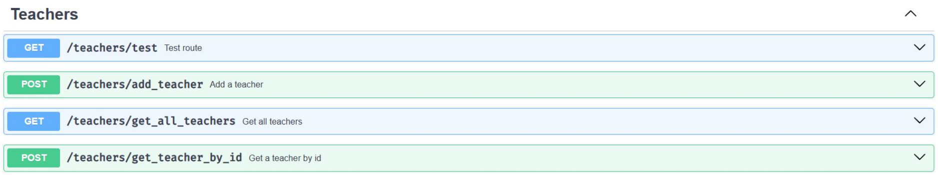
Semester Model to be added from Admin Page



```python
class SemesterModel(BaseModel):
    semester_number: int
    panel: str
    specialization: str
    school: str
    start_date: str
    end_date: str
    subjects: Optional[List[str]] = []
    teachers: Optional[List[str]] = []
    teacher_subjects: Optional[dict] = {}

    def set_id(self, _id):
        self._id = _id
```

API Documentation

Teachers API, used during Signup


Teachers

GET /teachers/test Test route

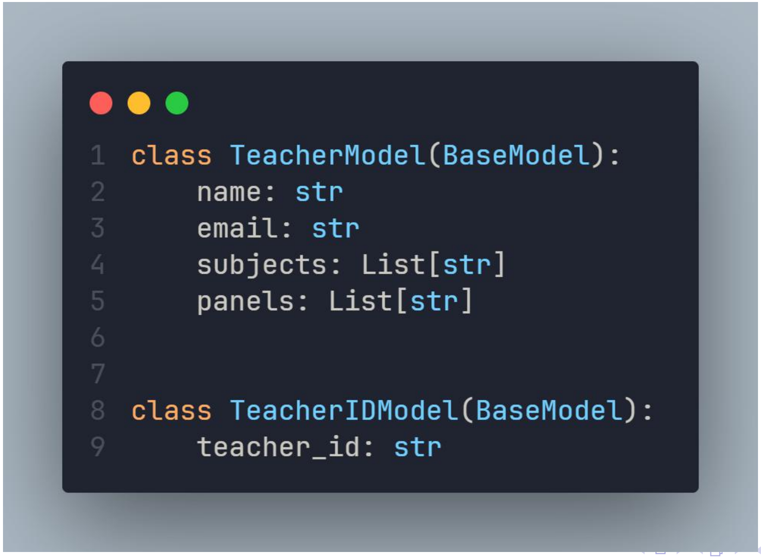
POST /teachers/add_teacher Add a teacher

GET /teachers/get_all_teachers Get all teachers

POST /teachers/get_teacher_by_id Get a teacher by id

API

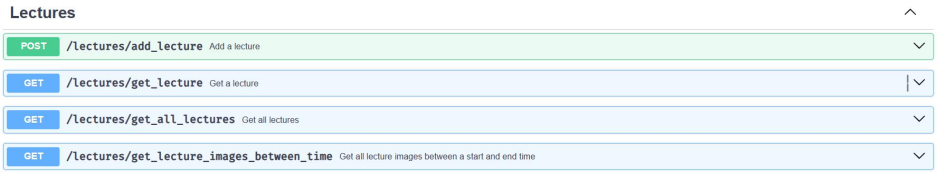
Teacher Model



```python
class TeacherModel(BaseModel):
    name: str
    email: str
    subjects: List[str]
    panels: List[str]


class TeacherIDModel(BaseModel):
    teacher_id: str
```

API

Lecture API, used when adding Attendance



Lectures

POST `/lectures/add_lecture` Add a lecture

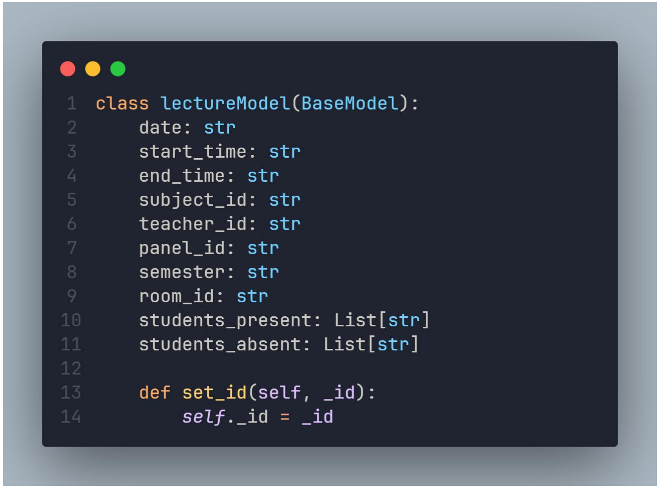
GET `/lectures/get_lecture` Get a lecture

GET `/lectures/get_all_lectures` Get all lectures

GET `/lectures/get_lecture_images_between_time` Get all lecture images between a start and end time

API

Lecture Models



```
class lectureModel(BaseModel):
    date: str
    start_time: str
    end_time: str
    subject_id: str
    teacher_id: str
    panel_id: str
    semester: str
    room_id: str
    students_present: List[str]
    students_absent: List[str]

    def set_id(self, _id):
        self._id = _id
```

Face Recognition Libraries Used

These are the libraries on which all images will be trained and tested.

1. ***face_recognition***: A simple face recognition library for Python. It is used to recognize the faces in the images, and to compare the faces.
2. ***OpenCV***: Open Source Computer Vision Library. It is used to detect the faces in the images. [1]
3. ***dlib***: A toolkit for making real world machine learning and data analysis applications in C++. It is used to detect the faces in the images.
4. ***DeepFace***: A lightweight face recognition and facial attribute analysis

[1] Marked in blue have been used so far.

Face Recognition Libraries To Use

Continued

1. ***imutils***: A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and Python.
2. ***MTCNN***: Multi-task Cascaded Convolutional Networks. It is used to detect the faces in the images.
3. ***FaceNet***: A face recognition library developed by Google. It is used to recognize the faces in the images.
4. ***InsightFace***: A face recognition library developed by the InsightFace team. It is used to recognize the faces in the images.

Training Data

These Images were Uploaded using the API along with student details.

Figure: Images Uploaded under Saubhagya's Name, and PRN.

Training Data

These Images were Uploaded using the API along with student details.

Figure: Images Uploaded under Avishkar's Name, and PRN.

Training Data

These Images were Uploaded using the API along with student details.

Training Data

These Images were Uploaded using the API along with student details.

Figure: These are the images uploaded under Krish's Name and PRN.

Training Data

These Images were Uploaded using the API along with student details.

Figure: These are the images uploaded under Parth's Name and PRN.

Preliminary Results

Recognized Faces

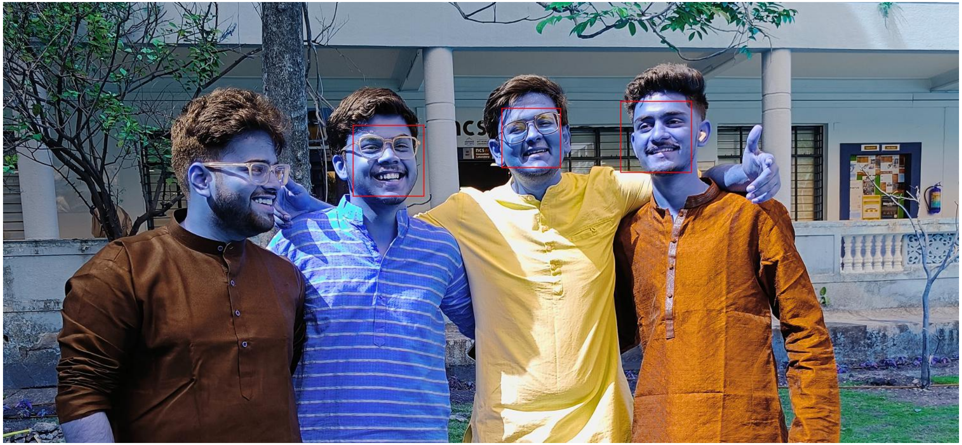


Figure: Results identifying 3 of the 4 faces. Empirical results show that the model is working, with accuracy of around 75 %

Backend Work Remaining

1. Complete the rest of the remaining APIs
2. Test all APIs
3. Integrate with Frontend
4. Try out all the other libraries.
5. Test with larger dataset
6. Documentation of Results