Proxy - Attendance using Face Recognition

Vinod Reddy *CS14B013* | Hemanth Reddy *CS14B024* | Srinidhi Prabhu *CS14B028* | Sai Pavan *CS14B041* | Satish Kumar *CS14B042*

I. Abstract

Easy Attendance *is a face recognition tool intended for use in classrooms. In essence, this software tool allows the course instructor to upload pictures of the class to take attendance effortlessly. This report presents the main aspects of the project's requirements specification, use cases, scenarios, specifications in Z-language, system models and architectural views.*

Keywords: attendance, software project, requirements specification, use cases, scenarios, class diagram.

II. Introduction

This project is intended for use by professors to simplify the process of taking attendance of a class and check for proxies. This project can be used to take attendance in universities and schools with about 60 students in class. The software allows instructors to upload photos of the class from which the attendance will be automatically marked, thus reducing the instructor's tedious task of marking attendance.Though there are many applications that can perform face recognition and identification tasks, there are very few which are specifically used for taking attendance in universities or schools.

III. General Description

This project is intended to provide a means for marking attendance of students by matching faces in new photos to the students' faces in the database. The professor shall collect the pictures of individual students, preferably in different angles. The underlying learning algorithm in the software gives the professor a daily attendance report once he uploads the photos of the class. The professor shall have an option to view history, i.e., pictures and attendance. Students can make a query in case their attendance was not marked, in which case the professor can override the system to approve that the student has indeed attended.

IV. Requirements Specification

The requirements for the project are as detailed below.

A. Functional Requirements

The main functional requirements of the project are:

1) The students enrolled in a course shall be taken from the existing database and the professor shall upload pictures of students, if necessary.
2) The professor shall be able to upload images, along with tags, to the database. These images shall be used as the training set on the learning algorithm to make predictions on future images.
3) The training data shall have images of students with exactly one face in every picture. The image shall be taken under decent lighting with good image resolution.
4) Every professor and student shall have an account.
5) In the case of professors, the account shall have the following options:
    a) Upload images of the class for a particular day and get the attendance for that day.
    b) Look up the images uploaded on a particular date and get details of the students who were present or absent.
    c) A section for the queries raised by students, in case the student is present but marked absent. These queries shall be approved only by the professor.
6) In the case of students, the account shall have the following options:
    a) Show the number of classes attended by the student and the total number of classes.
    b) Warn the student, in case his attendance is less than the threshold.
    c) Provide an option to raise a query, in case he is present in the photo but not marked present.
7) The software shall authenticate students and professors during sign-up and provide required access to each account.
8) The software shall maintain a log of all activities.

B. Non-functional Requirements

The most important non-functional requirements for the project are:

1) Proxy shall be written in HTML, CSS and Python.
2) Face++ APIs shall be used for face detection and recognition.
3) Django framework shall be used for the backend.
4) The project shall be platform independent.
5) Memory requirements - Assuming an image to have a size of 2MB, training data requires about 500MB for a class of size 50. Also, to store images throughout the semester, we need another 500MB assuming 5 photos a day and 50 classes in a semester.
6) Since the project shall use API calls, the server is expected to be connected to the Internet.
7) The application shall be built over a span of three months.

V. ING

The use case diagram shown in the figure depicts the interactions between the actors (the professors and the students) and the software system. First, the professors upload individual students' photos. This is typically done at the beginning of each semester. Once this is done, the professor uploads class photos every day. The software would generate daily attendance reports that can be seen by the professors and the students. Since face recognition may not be correct always, the software allows for students to raise queries. The queries can later be approved by the professor, if needed. It shall also be ensured that a professor can only approve queries for the courses he teaches.

VI. Detailed Use Cases & Scenarios

We present four(two each from professor and student) examples of use cases. Figure 2 depicts the Upload individual students photos use case, in which a professor uploads individual pictures of students for training, while Figure 3 shows the Uploads class photos for attendance use case where faces of students in the class are recognised, and matching is done.

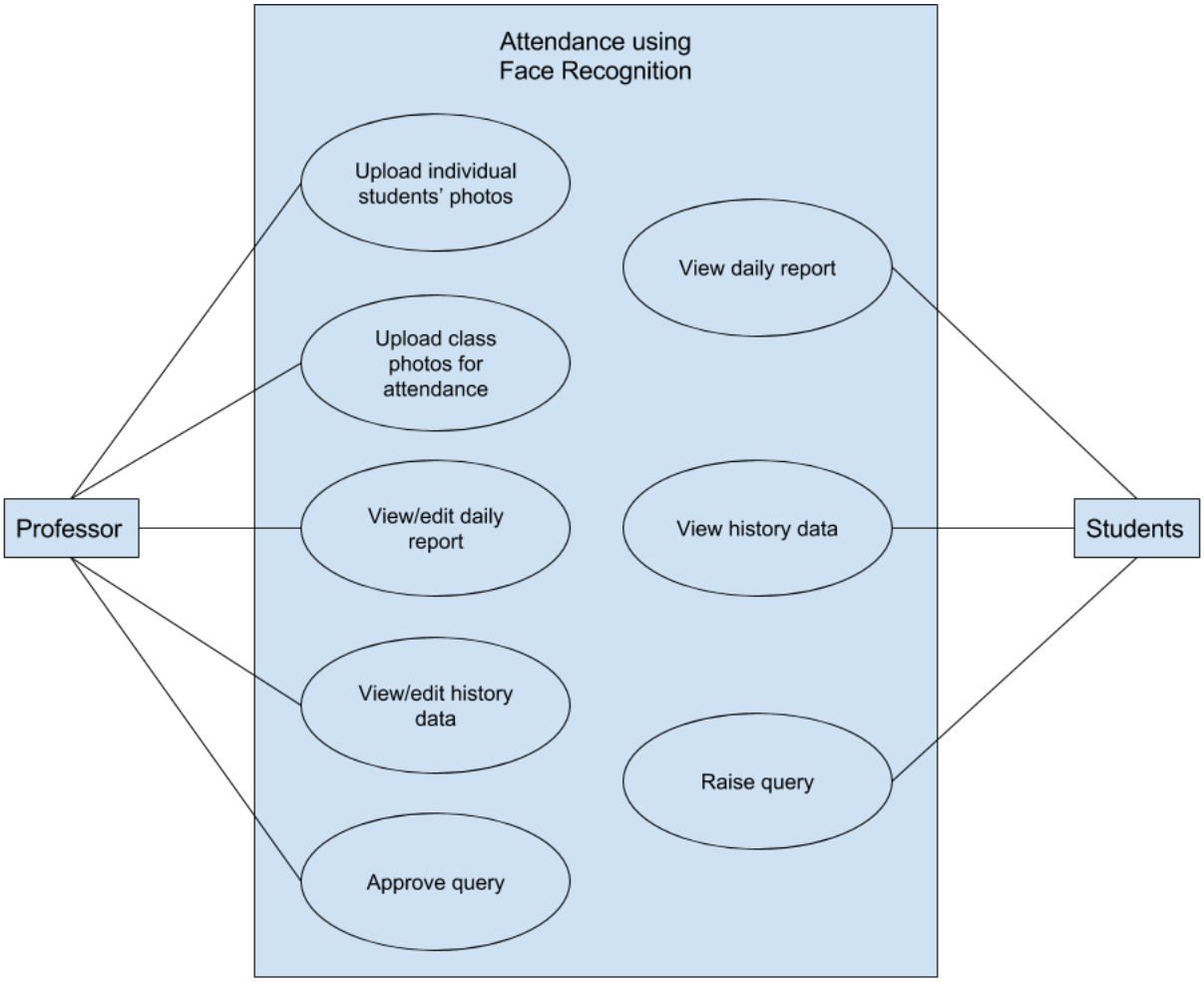


Fig. 1: Use Case Diagram

Use Case: Upload individual students' photos

ID: UC1

Actor: Professor

Precondition:

1. Professor has individual students' photos.

Flow of events:

1. The use case starts right after the professor signs up.
2. The system prompts the user to upload the students' photos.
3. The user uploads the photos.
4. The system saves the photos to the database.

Secondary scenarios:

1. Upload failed due to connectivity error.
2. Saving to database failed due to insufficient memory.
3. In the case of duplicate photos, display a warning message.

Postcondition:

1. The photos are saved in the database.

Fig. 2: Use case 1

Use Case: Uploads class photos for attendance

ID: UC2

Actor: Professor

Precondition:

1. Professor has high-quality images of the class.
2. Training data is present.

Flow of events:

1. The use case starts when the professor wants to take attendance.
2. The system prompts the user to upload the class photos.
3. The user uploads the class photos.
4. The system saves the images in the database and matches all the faces recognised in the class photo with those in the training data.
5. The system creates a daily report.

Secondary scenarios:

1. Matching failed due to connectivity error.
2. Unable to recognize face(s) due to poor quality.
3. Recognized face(s) couldn't be matched with those in the training data.

Postcondition:

1. The attendance data is saved in the database.

Fig. 3: Use case 2

Use Case: View daily report
ID: UC6
Actor: Student
Precondition: 1. Attendance data is saved in the database.
Flow of events: 1. The use case starts when the student wants to view the daily report. 2. The student views the daily report which is obtained from the database.
Secondary scenarios: 1. Fetching the daily report is failed due to connectivity error.
Postcondition: 1. The student views his/her attendance.

Fig. 4: Use case 2

Use Case: Raise Query
ID: UC8
Actor: Student
Precondition: 1. Attendance data is saved in the database. 2. Student viewed the daily report/history data. 3. There should be some error concerning the student.
Flow of events: 1. The use case starts when the student wants to raise a query. 2. The system prompts the user to enter the query details. 3. The user enters the details. 4. The system alerts the professor regarding the query.
Secondary scenarios: 1. Failed due to connectivity error.
Postcondition: 1. The professor is notified about the query.

Fig. 5: Use case 2

Scenario for Use Case: Upload class photos for attendance.
Primary scenario
Scenario ID: S2.1
Actor: Professor
Precondition: 1. Professor has high quality images of the class. 2. Training data is present.
Flow of events: 1. When the user chooses to upload the class photos, a dialog box appears asking to select all the image files to be uploaded. 2. The user clicks the “Upload” button. If the user clicks the “Cancel” button, the system returns to the previous state. 3. The system saves the images to the database and performs face recognition. 4. The system notifies the user saying that face identification has been successful.
Postcondition: 1. The attendance data is saved in the database.

Fig. 6: Use case 2

Scenario for Use Case: Upload class photos for attendance.
Secondary scenario: Recognized face(s) couldn’t be matched with those in the training data.
Scenario ID: S2.4
Actor: Professor
Precondition: 1. Professor has high quality images of the class. 2. Training data is present.
Flow of events: 1. This scenario starts following step 3 of S2.1. 2. The system is unable to match few faces with those in the training data. 3. The system then warns the user regarding the same. 4. The system also displays the faces which couldn’t be identified.
Postcondition: 1. The attendance data is saved in the database.

Fig. 7: Use case 2

Scenario for Use Case: Raise Query
Primary scenario
Scenario ID: S8.1
Actor: Student
Precondition: 1. Attendance data is saved in the database. 2. Student viewed the daily report/history data. 3. There should be some error concerning the student.
Flow of events: 1. The user chooses to raise a query. 2. The systems displays a dialog box asking the user to enter the query details. 3. The user enters the details and clicks “OK” 4. The system alerts the professor regarding the query. 5. If the user clicks “Cancel”, the system returns to the previous state.
Postcondition: 1. Professor is notified about the query.

Fig. 8: Use case 2

Scenario for Use Case: Raise Query
Secondary scenario: Failed due to connectivity error.
Scenario ID: S8.2
Actor: Student
Precondition: 1. Attendance data is saved in the database. 2. Student viewed the daily report/history data. 3. There should be some error concerning the student.
Flow of events: 1. This scenario starts following step 3 of S8.1. 2. The system displays an error message saying lost network connectivity. 3. The system waits until the connectivity is restored and then returns to the previous state.
Postcondition: 1. The system returns to the previous state.

Fig. 9: Use case 2

VII. Specifications in Z Language

The software is initially on a login screen(LOGIN) or a signup screen(SIGNUP).

$$PROXY = LOGIN \lor SIGNUP$$

The possible states of the software in case of PROFESSOR_ACCOUNT and STUDENT_ACCOUNT are as follows:

$$PROFESSOR\_ACCOUNT = PROFESSOR\_HOME \lor PROFESSOR\_COURSE$$
$$PROFESSOR\_COURSE =$$
$$UPLOAD\_TRAINING\_DATA \lor UPLOAD\_ATTENDANCE \lor VIEW\_HISTORY \lor APPROVE\_QUERIES \lor EDIT\_HISTORY$$
$$STUDENT\_ACCOUNT = STUDENT\_HOME \lor STUDENT\_COURSE$$
$$STUDENT\_COURSE = VIEW\_ATTENDANCE \lor RAISE\_QUERY$$

The LOGIN schema defines the PROXY's state for authenticating the user. It is executed at the beginning.

LOGIN

$\Delta PROXY$
// *Inputs*
email? : *String of the form* $<$ *name* $>$@*iitm.ac.in for professors and* $<$ *roll_no* $>$@*smail.iitm.ac.in for students.*
password? : *String with at least* 8 *characters and at most* 15 *characters.*
// *Outputs*
$success! = (Yes, No)$
$type! = (Professor, Student)$
// *Actions to perform*
$InDatabase(email?) \land Row(email?).getPassword() == password? \Rightarrow success! = Yes$
$\lor$
$success! = No$
$InDatabase(email?) \land Row(email?).getPermission() == Professor \Rightarrow type! = Professor$
$\lor$
$type! = Student$
$type! = Professor \Rightarrow PROFESSOR\_HOME$
$\lor$
$type! = Student \Rightarrow STUDENT\_HOME$

The SIGNUP schema defines the state when a user wants to create an account.

SIGNUP

$\Delta PROXY$
// *Inputs*
email? : *String of the form* $<$ *name* $>$@*iitm.ac.in for professors and* $<$ *roll_no* $>$@*smail.iitm.ac.in for students.*
newPassword? : *String with at least* 8 *characters and at most* 15 *characters.This is taken as input after verifying the email.*
// *Outputs*
$success! = (Yes, No)$
$type! = (Professor, Student)$
// *Actions to perform*
$success! = No$
$type! = Student$
$email?\ matches\ regex * @iitm.ac.in \Rightarrow type! = Professor$
Send verification mail(*with link*) *to email*?.
User verifies $\Rightarrow$ $verified = Yes$
Input newPassword?
Add e $-$ *mail*? *and newPassword*? *into the database with type*! *Permissions.*
$success! = Yes$
$success! = Yes \Rightarrow LOGIN$

The PROFESSOR_COURSE schema defines the state when a professor is on his home page.

PROFESSOR_HOME

$\Delta LOGIN$
// *Input*
course? : *The course to view the attendance.*
// *Outputs*
email! : *Professor's email*
courses! : *Courses taught by the professor*
photo! : *Professor's photo*
// *Actions to perform*
onClick(*course*?) $\Rightarrow$ *PROFESSOR_COURSE with details of course*?.

The PROFESSOR_COURSE schema defines the state when a professor selects a particular course.

PROFESSOR_COURSE

$\Delta PROFESSOR\_HOME$
// *Input*
tab? : *Selects the tab to go to.*
// *Actions to perform*
tab? == *UploadTrainingData* $\Rightarrow$ *UPLOAD_TRAINING_DATA*
tab? == *UploadAttendance* $\Rightarrow$ *UPLOAD_ATTENDANCE*
tab? == *ViewHistory* $\Rightarrow$ *VIEW_HISTORY*
tab? == *ApproveQueries* $\Rightarrow$ *APPROVE_QUERIES*
tab? == *EditHistory* $\Rightarrow$ *EDIT_HISTORY*
tab? == *Back* $\Rightarrow$ *PROFESSOR_HOME*

The UPLOAD_TRAINING_DATA schema is the state when the professor chooses the option to upload images for training.

UPLOAD_TRAINING_DATA

$\Delta PROFESSOR\_COURSE$
// *Inputs*
image? : *Image with the face of exactly one student*
tag? : *Roll number of the student in the image*
// *Output*
success! : (*Yes*, *No*)
// *Actions to perform*
token = *findFaceToken*(*image*)
addtoDB(*token*, *tag*)

The UPLOAD_ATTENDANCE schema specifies the state when the professor uploads images of the class.

UPLOAD_ATTENDANCE

$\Delta PROFESSOR\_COURSE$
// *Inputs*
images? : *A set of images of the class on a particular day.*
date? : *Date on which the photos were taken.*
// *Outputs*
imageID! : *Numbers assigned to each image for the purpose of raising query by the student.*
attendance! : *A list of students who are present.*
// *Actions to perform*
tokens = *findFaceTokens*(*images*?)
attendance! = *getAttendance*(*tokens*)

VIEW_HISTORY

$\Delta PROFESSOR\_COURSE$
// *Input*
date? : *The date for which the professor wants to view the attendance.*
// *Outputs*
images! : *Shows the pictures uploaded by the professor on the entered date*
attendance! : *List of students present on the entered date.*
// *Actions to perform*
attendance! : = *getAttendance*(*date*?)
images! : = *getImages*(*date*?)

EDIT_HISTORY

$\Delta PROFESSOR\_COURSE$
// *Inputs*
date? : *The date for which the professor wants to edit the attendance.*
rollNum? : *The roll number of the student for which the professor wants to edit the attendance.*
change? : (*Add*, *Remove*) *To add or remove attendance for the student.*
// *Output*
success! : (*Yes*, *No*)
// *Actions to perform*
change? = *Add* $\Rightarrow$ *addRow*(*date*, *rollNum*)
$\vee$
change? = *Remove* $\Rightarrow$ *deleteRow*(*date*, *rollNum*)

APPROVE_QUERIES

$\Delta PROFESSOR\_COURSE$
// *Input*
approve? : (*Yes*, *No*)
// *Output*
success? : (*Yes*, *No*)
// *Actions to perform*
addRow(*getDate*(*Query*), *getRollNum*(*query*))

STUDENT_HOME

$\Delta LOGIN$
// *Input*
course? : *The course to view the attendance.*
// *Outputs*
email! : *Student's email*
courses! : *Courses taken by the student*
photo! : *Student's photo*
// *Actions to perform*
onClick(*course*?) $\Rightarrow$ *STUDENT_COURSE with details of course*?.

STUDENT_COURSE

$\Delta STUDENT\_HOME$
// *Input*
tab? : *Selects the tab to go to.*
// *Actions to perform*
tab? == *ViewAttendance* $\Rightarrow$ *VIEW_ATTENDANCE*
tab? == *RaiseQuery* $\Rightarrow$ *RAISE_QUERY*
tab? == *Back* $\Rightarrow$ *STUDENT_HOME*

VIEW_ATTENDANCE

Δ*STUDENT_COURSE*
// *Input*
date? : *The date for which the student wants to view his attendance.*
// *Outputs*
totalAttendance! : *Total number of days the student has attended the classes along with percentage.*
images! : *Shows the pictures uploaded by the professor on the entered date*
dateAttendance! : *Shows if the student was present or not on the entered date*
// *Actions to perform*
attendance! : = *getAttendance*(*date*?)
images! : = *getImages*(*date*?)

RAISE_QUERY

Δ*STUDENT_COURSE*
// *Inputs*
date? : *The date for which the student wants to raise the query.*
imageNum? : *The image number among the uploaded class images in which the student is present.*
// *Outputs*
success! = (*Yes*, *No*)
// *Actions to perform*
addRow(*Query*)

VIII. Description in natural language

Initially, the user sees a login or signup screen.

If he chooses to login, he enters the email and password. Based on the email, we decide if it is a professor or a student and take the user to the appropriate home page.

If he chooses to sign up, we take the email and send a verification mail with a link. When he clicks on the link, he is asked to enter a new password. After this, the account is created and he is taken to the login page.

When a professor logs in, he is shown his details and the courses he teaches. He can click on a course, and then perform actions like uploading class images, approving queries, etc.

When a student logs in, he is shown his details and the courses he has taken that semester. He can click on a course, and then perform actions like viewing attendance and raising queries.

IX. System Models

A. Context Models

The following figure is a simple context model that shows the Attendance system and the other systems in its environment. You can see that the Easy Attendance is connected to a student record system and a professor record system from which it gets data related to them. The system is also connected to a system for courses information. Finally, it makes use of a Face recognition system to mark attendance for a student.

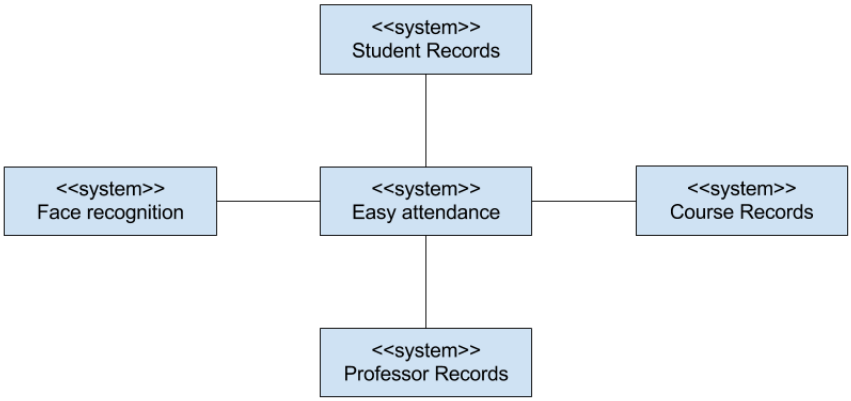


Fig. 10: The context of Easy Attendance System

The following figure is a model of an important system process that shows the processes in which the software is used. Once the professor uploads class photos,the Face Recognition system detects all the faces in the photo and tries matching them with the individual student pics in the database. If it finds a match, then the system updates the attendance record and informs the student. If no match is found, it warns the professor.

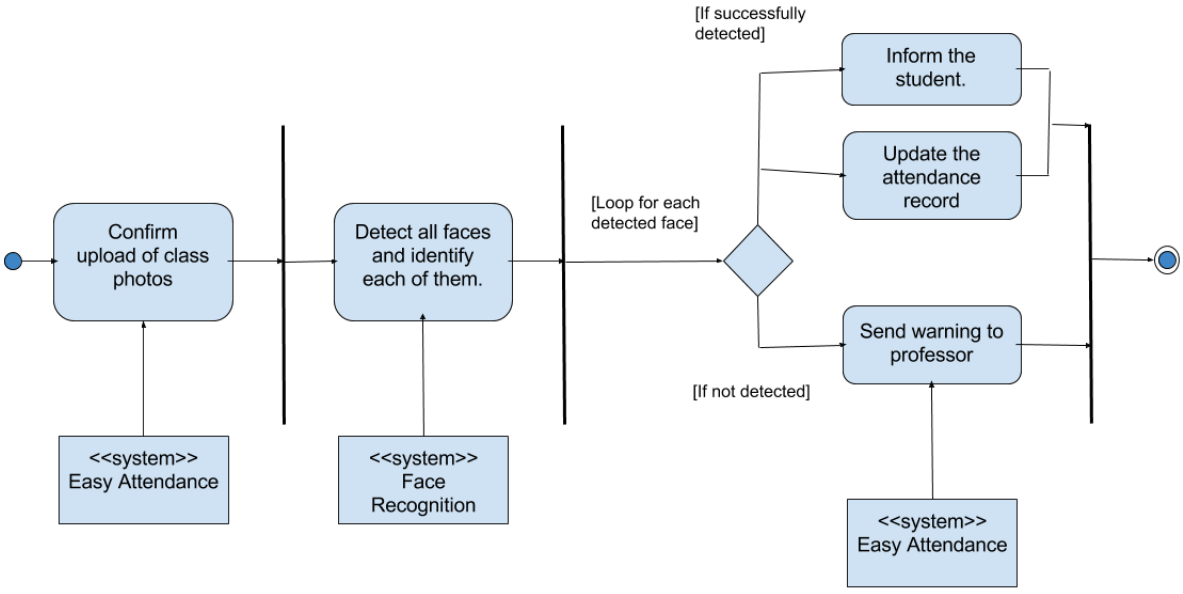


Fig. 11: Process model of uploading class photos

B. Interaction Models

1) Use Case Model: The following figure describes what each user expects from the system.

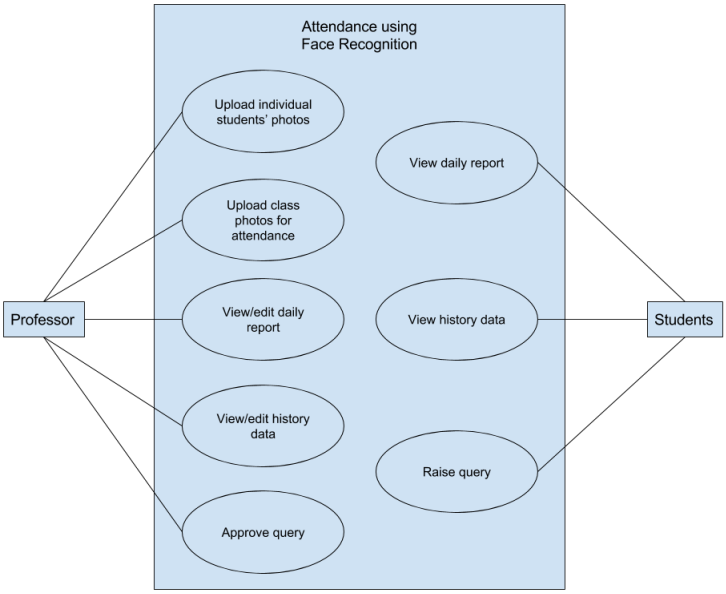
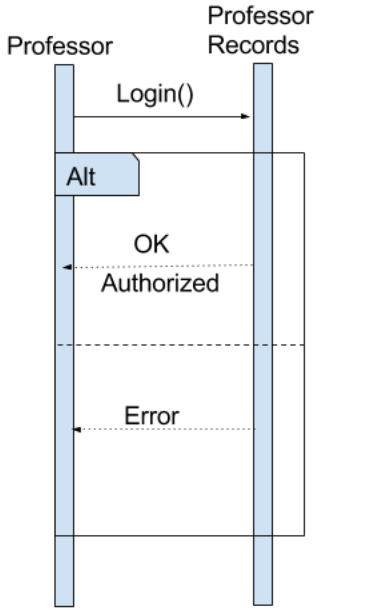


Fig. 12: Use cases involving professors and students

2) Sequence diagrams: The following diagram shows the sequence of actions that happen when a professor tries to login. First, the professor enters his login credentials. When the login button is pressed, the professor records are searched. If the professor with the ID and password are found, the professor is authenticated and he can view his homepage. Otherwise, he will be shown an error message.



In case professor is authorized, further operations will be allowed.

Fig. 13: Sequence diagram for professor's login

The following diagram shows the sequence of actions that happen when a professor uploads photos for marking attendance. First, the professor uploads the photos. Then the face recognition module recognizes the students present in the photo. Then, the students records are updated to mark attendance for that particular day. In case all these steps are successful, a success message is shown. Otherwise, an error message is shown asking the professor to try again.

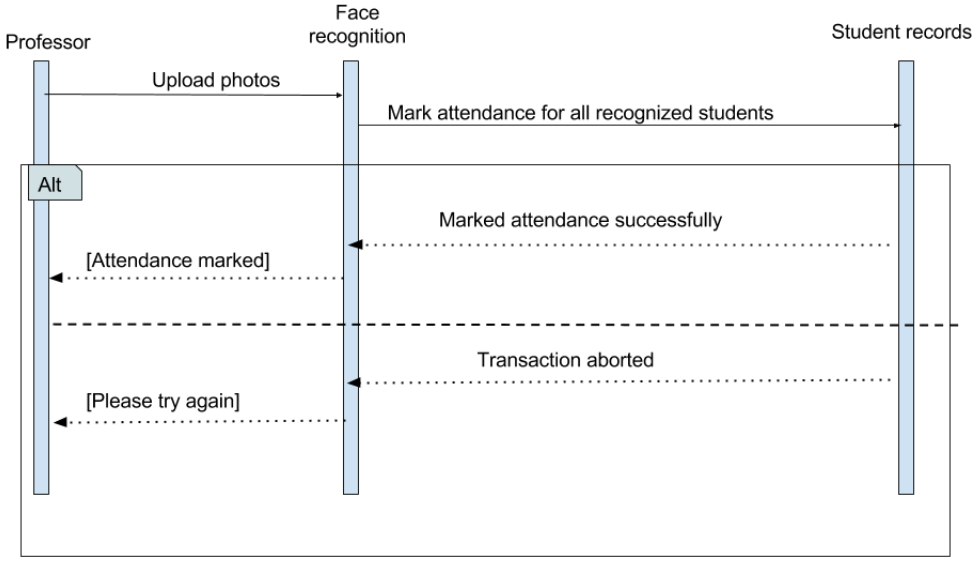


Fig. 14: Sequence diagram for professor to mark attendance

C. Structural Model

1) Class diagram: The following diagram shows the various classes to be used and how they are related to one another. It also specifies the relation between them quantitatively (one-to-one OR many-to-one OR one-to-many OR many-to-many). The different classes used are: Student, Professor, Course, Query, Attendance, Test Data, Training Data, Login Info and Enrolled data for each course. The attributes of each class are described previously in Entity Relationship(ER) diagrams.

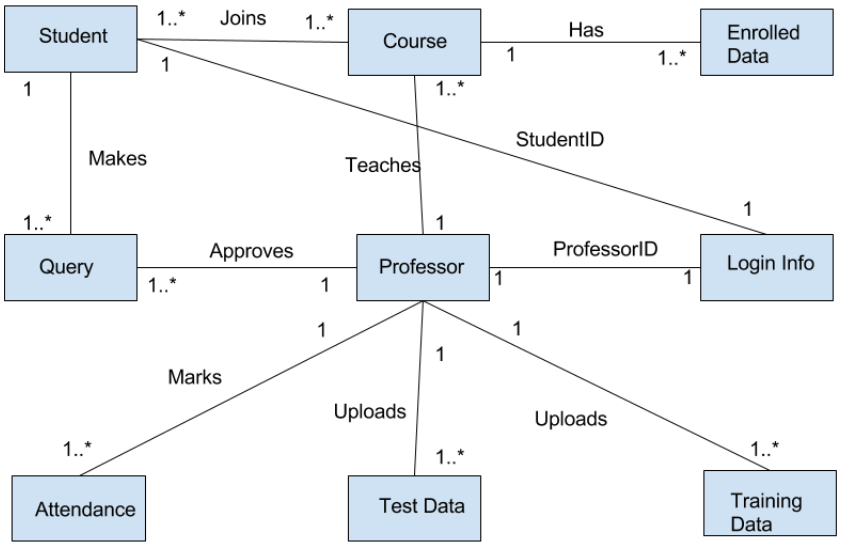
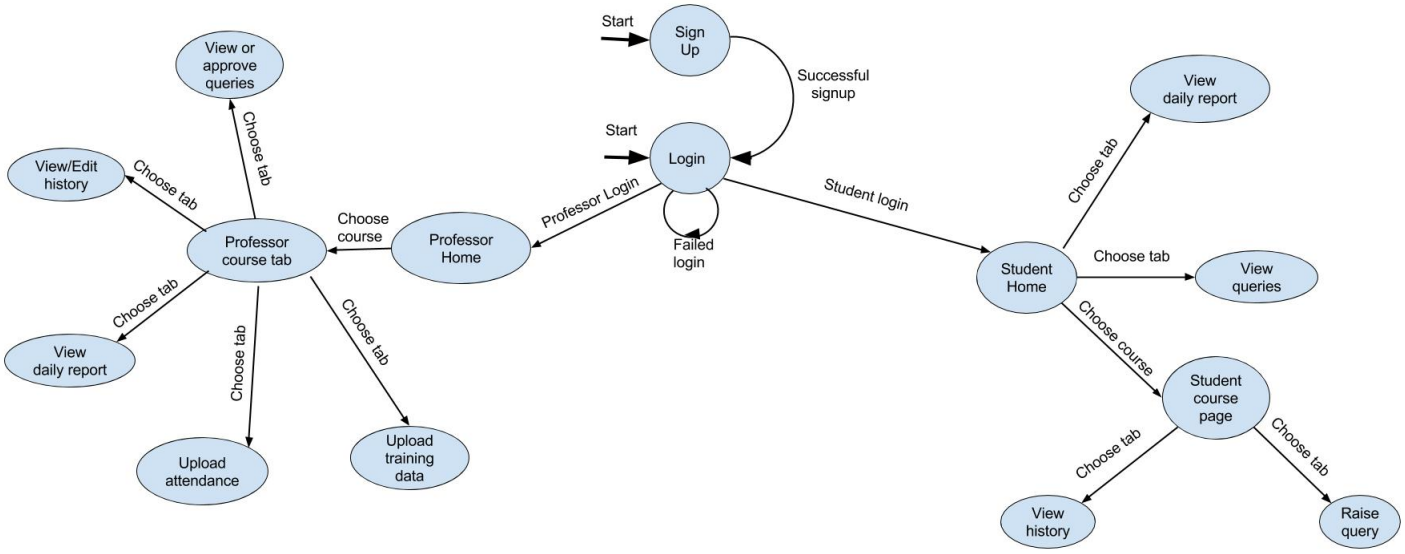


Fig. 15: Classes and associations in Easy Attendance

D. Behaviour Model

1) Event-driven model: The behavioral model describes the flow of data between various states and the events that cause this flow of data. The below model captures the event driven model and how the system moves from one state to another when it receives an event.



There will be a back button for every state. This is not mentioned for the sake of conciseness.

Fig. 16: Professor's activity

X. Architecture

A. Framework

Our team has decided to use the python based django framework for the web app. We have chosen it because of the following:

1) Ease of development: Django was designed to help developers take applications from concept to completion as quickly as possible. It is based on python, so coding becomes quite easy.

2) Inbuilt packages: Many important software components are already bundled with the framework. They include:

- Database system and database abstraction APIs.
- Authentication systems and admin templates.
- URL binders.
- Ability to design custom views and templates.
- Caching framework.

3) Highly secure: The framework prevents common attacks like SQL injection, cross site scripting, cross site request forgery and clickjacking. Its user authentication system provides a secure way to store usernames and passwords.

4) Highly scalable: Django is designed to take advantage of all the hardware available. This helps when the number of users change significantly.

5) High maintainability: Python code is readable and the framework has an emphasis on backward compatibility. This makes the code easy to maintain.

B. Architectural Views

1) Logical View: This shows the key classes in the system. They are :

- Professor
- Student
- Course
- Login Info
- Query
- Enrolled Data
- Test data
- Training data

– Attendance

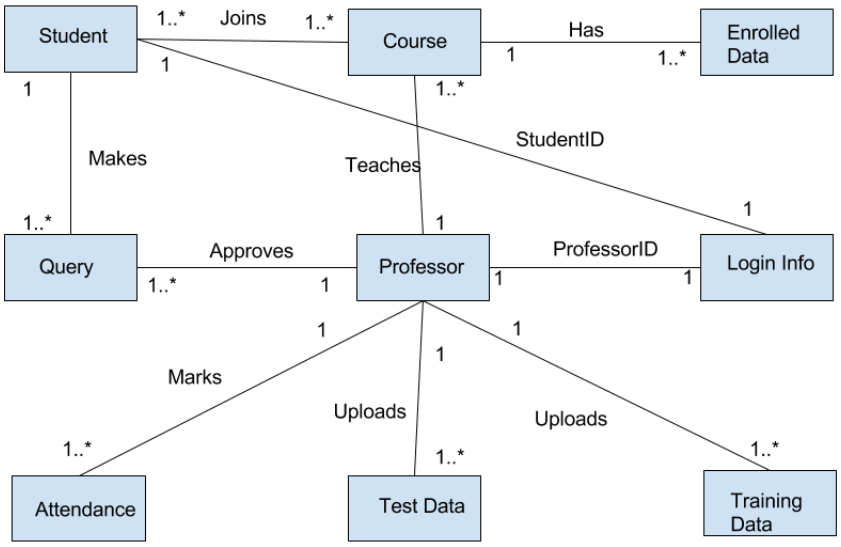


Fig. 17: Logical view, ie, key classes

2) Process View: Describes the interaction of processes at runtime. For example, consider the following process model for taking attendance:

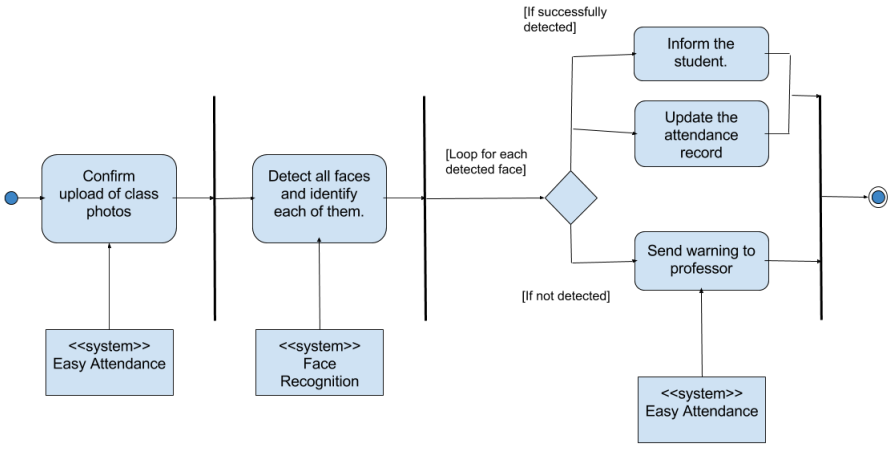


Fig. 18: Process view for taking attendance

3) Development View: We have assigned two members to take care of front-end, ie, view and some part of the controller. The remaining three members will take care of the backend, ie, model and remaining part of the controller. This includes the database setup, API wrappers, etc.

4) Physical view: We are not really concerned about this. The django framework takes care of it provides us a layer of abstraction so that we can start working right away.

5) Conceptual view: These include the different contexts in the system. They are:

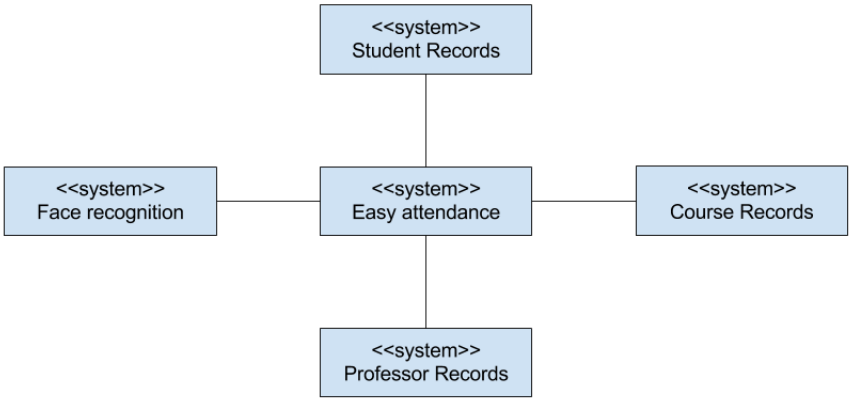


Fig. 19: Conceptual view, i.e, a high level view of the system

C. Architectural Pattern

We have decided to use the Model-View-Controller(MVC) architecture pattern because of the following reasons:

- It is inherently supported by django, thereby making the development process easy.
- There are multiple views, ie, there are two user roles: prof and student.
- Simultaneous development is possible because there is low coupling among the different parts.
- This pattern is quite flexible and can adapt to changing requirements.

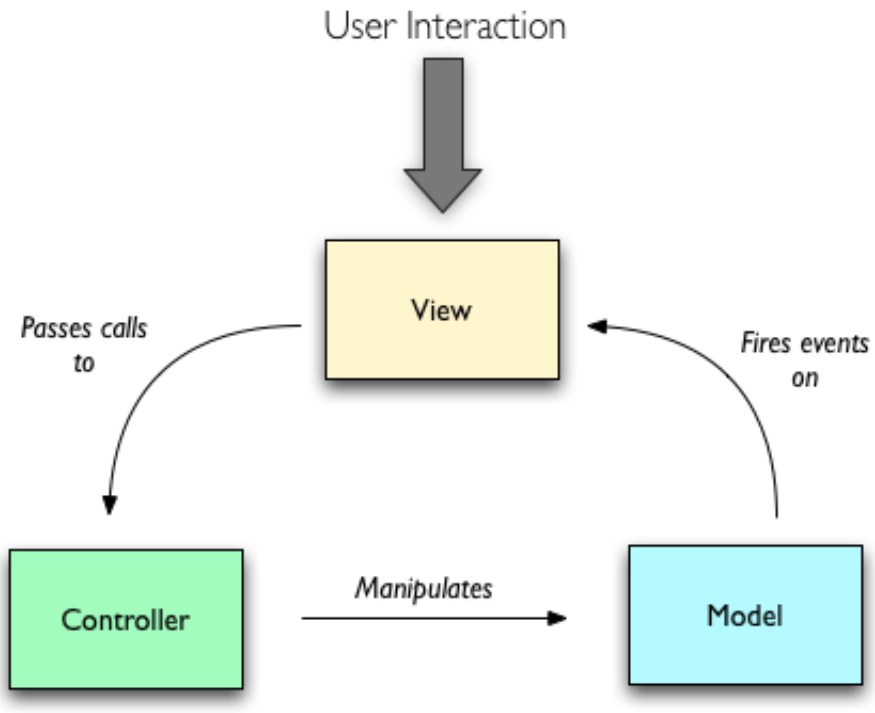


Fig. 20: The MVC pattern

XI. Glossary

- **CSS**: Cascading Style Sheets, which is a style sheet language used to style documents written in a markup language.
- **HTML**: HyperText Markup Language, a standardized system for tagging text files to achieve font, colour, graphic, and hyperlink effects on World Wide Web pages.
- **JavaScript**: A dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites.
- **Learning algorithm**: An algorithm that provides computers with the ability to learn without being explicitly programmed.
- **Log**: An official record of events.
- **Python**: An interpreter based programming language with emphasis on code readability.
- **Training data**: A set of data used to discover potentially predictive relationships.

XII. Appendices

The schema for the tables used in the database are as follows:

LoginInfo(UserID, Password, PermissionLevel)
SignUp(Email, URL, TimeStamp)
Student(StudentID, StudentName, Email)
Professor(ProfessorID, ProfessorName, Email)
Courses(CourseID, CourseName, Professor)
EnrolledData(EnrolledID, CourseID, UserID)
Attendance(AttendanceID, CourseID, UserID, Date, PresentOrNot(Yes/No))
Query(QueryID, UserID, CourseID, Description, Date, TimeStamp, Status(Pending/Accepted/Declined))
TrainingImages(ImageID, UserID)
TestImages(ImageID, CourseID, TimeStamp)
All IDs are default IDs assigned by the underlying database.
TimeStamp includes the date and time.
Other columns with enumerated variables are mentioned.

INDEX