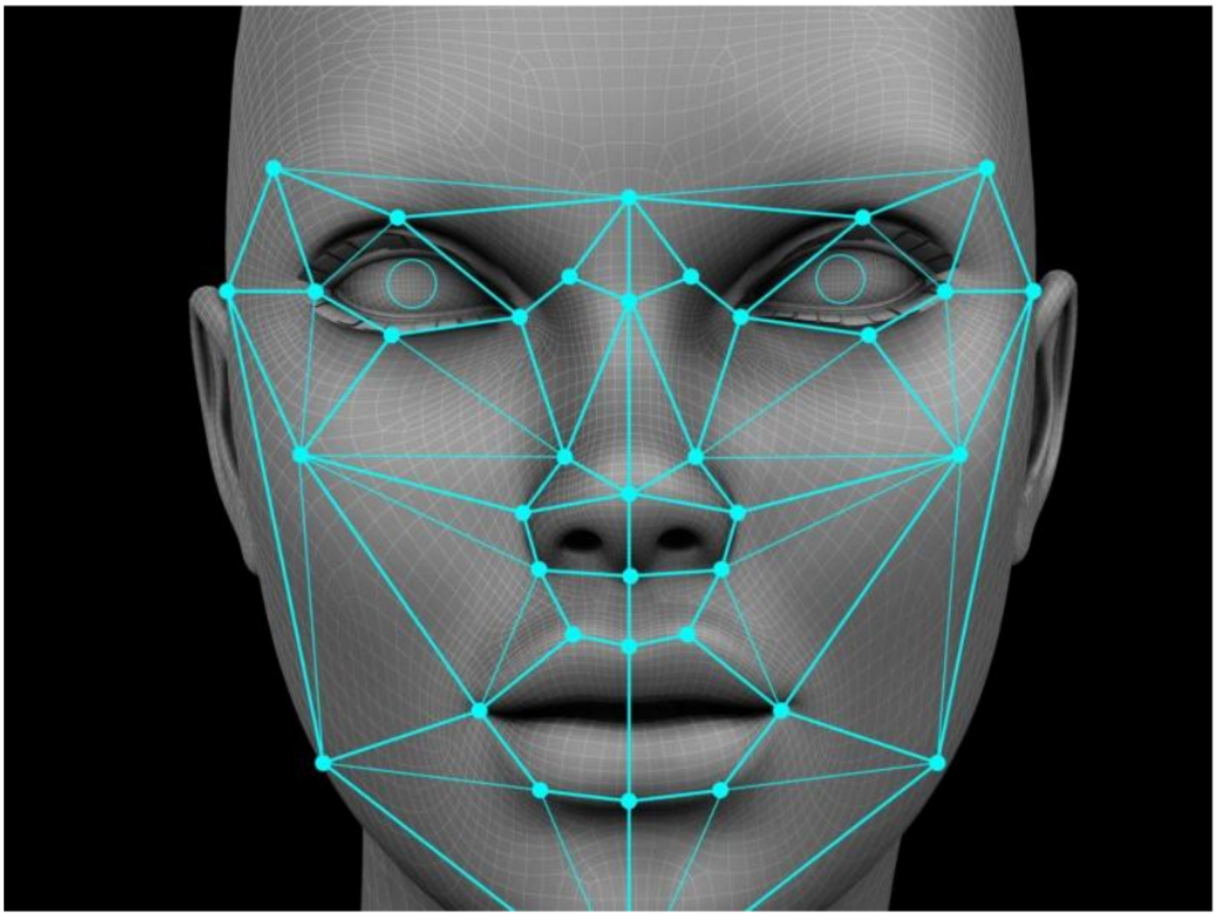
WhoWhenWhere



Introduction:

Authentication has always been a challenge. In case of passwords hashes are used for securing, but there were cases where hashes collide. When finger print based authentication methods were introduced there were cases where the security was compromised by forging ways to manipulate fingerprints,

also the accuracy was 1 in 50,000 (i.e: there could be 1 failure in 50,000 attempts). A unique feature is our face. Building a system to authenticate using the facial data will ensure that the security is maintained in a system.

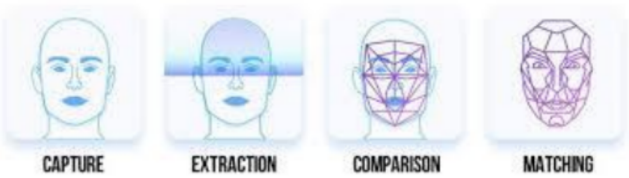
Existing System:

The existing system uses fingerprints which can be forged and the security is compromised. To authenticate using fingerprints special hardware is used. Also there are limitations on how many finger prints can be stored in the fingerprint reader.

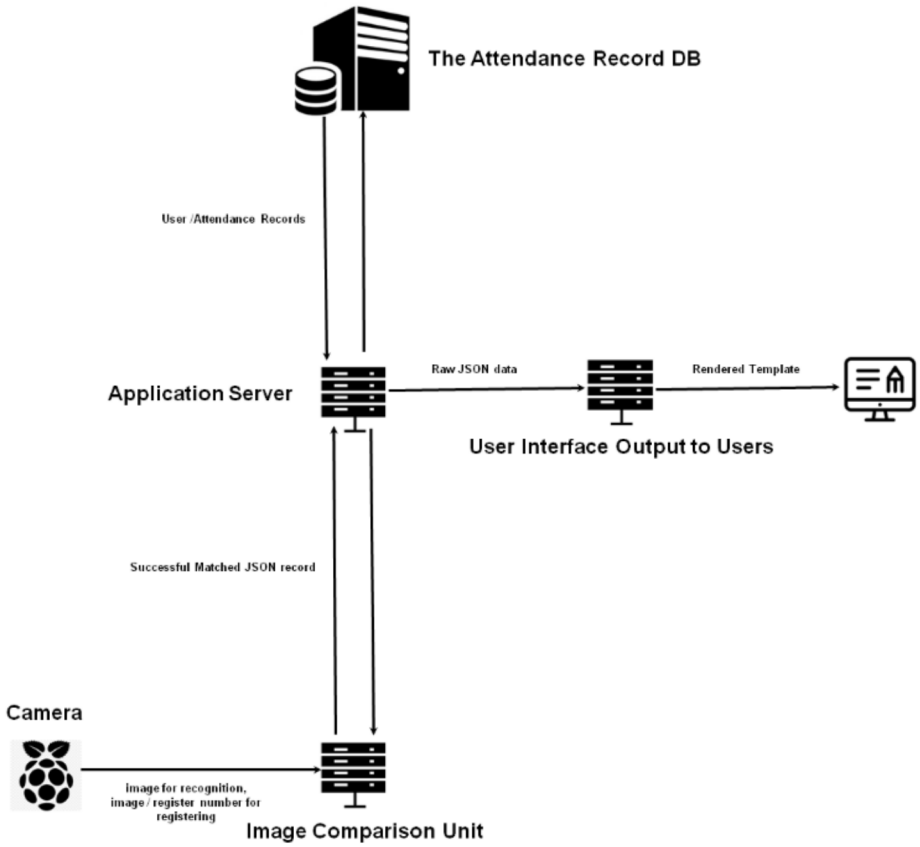
Iris recognition is state of the art but it is too costly to be used.

Advantages of using Facial Recognition:

Unique identity for each individual. The space required to store image data is very less (100KB at most per person). When 300 faces are registered the space is just 30000KB.



Architecture for Face Recognition System for Hostel Attendance :



About the project:

This project uses facial recognition to recognize students and mark attendance in Hostels. The project has it's core Image processing exposed through a Flask API. This Flask API is used to register a face and authenticate a face. The image processing API has two routes:

1. '/register' route (it is a HTTP POST method) is responsible for taking in an image with the face and along with the students register number and stores the image in the image processing unit's memory.

2.'/enter_batch' route (it is a HTTP POST method) is responsible for taking in an image and adding it to a target set of images which will be processed by a script which periodically compares these target images with known images and generates attendance records. This route stores the image with a name of the format ('YY-MM-DD-HH-MM-SS.jpg').

The Images for use by the Image processing API are captured using a camera which is interfaced with a Single Board Computer such as a RaspberryPi. A client script gets the input from a user and grabs an image from the connected camera and sends it to the Image Processing API in the proper route for further processing.

The face recognition library extracts 128 facial features and creates an array of 128 elements. If any other image is converted and if the Euclidean distance between two face arrays is less than a tolerance value then it is considered a match.

The batch process which periodically generates a dictionary from known images and the dictionary is organized as "roll-no : face_encoding" and processes the unknown images for matching with the known images. Then

for each unknown image the known face is matched with and a record is generated for marking the attendance and the matched key is removed from the dictionary and the matched unknown image is also deleted. The record is generated by splitting the file name of the unknown image to get time data and the register number is the matched key in the known encodings dictionary.

The records generated are JSON objects which contain the register number, date, hours and minutes. **For storing one attendance record in MongoDB the space required is 97Bytes (0.094 KB per record).**

The Attendance records are stored in a MongoDB Server. The DB operations are masked using a JavaScript API written on NodeJS runtime using ExpressJS. The Attendance records are set up as a MongooseODM schema. There are validations in the schema itself. This JavaScript API also is used by the UI to monitor the DB records at any time.

The JavaScript API is secured using JSON Web Token (JWT RFC 7519 standard). For any request to be processed by the backend, Token is first checked for validity.

The Authentication Process for a user:
1. A user provides a credentials pair.

2. On successful verification of the credentials a JWT is generated and returned as response to the user. This Token is mapped to this user.

3. From then on every request with this token will be valid.

4. On Logout the token is deleted from the memory and the mapping is also removed.

The Authentication for a device is given by hard coding a JWT into the client script. Only devices with a valid JWT will be able to interact with the DB.

The UI is served by another Flask API which uses the Jinja templating engine to render data fetched.

The above mentioned is only a prototype of this system. Currently we are developing an Algorithm to map faces as a graph based on the distance parameter so that the number of comparisons to identify a face gets reduced. Another approach which we have in mind is to make the process of comparing faces parallelized so that multiple images are compared at the same time.

DatabaseDesign:

Attendance Document:

_id :

4 bytes a 4-byte value representing the seconds since the Unix epoch
3 bytes a 3-byte machine identifier
2 bytes 2-byte process id
3 bytes 3-byte counter, starting with a random value

Number:

Register number of the student. It has to be 10 numbers.

Day:

Day on which the record was created

Month:

The month on which the record was created

Year:

The year on which the record was created

Hours and minutes:

The time at which the record was created

User Document:

_id:

same as above

Email:

It accepts an email and has a validation function built in.

Password:

This field has a hashed password stored.

Tokens:

This field is an Array of authentication tokens.

Role Based Access:

The access permissions can be modified to provide access to records based on their roles such as:

- RC (Resident Counselor): can get daily data updates.
- Block Warden: can get data for the blocks under their control.
- Executive Warden: will have the provision to access the whole database and download the data for analytics.

With Respect to Time how does the process occur?

For Example:

20:00 – Attendance starts

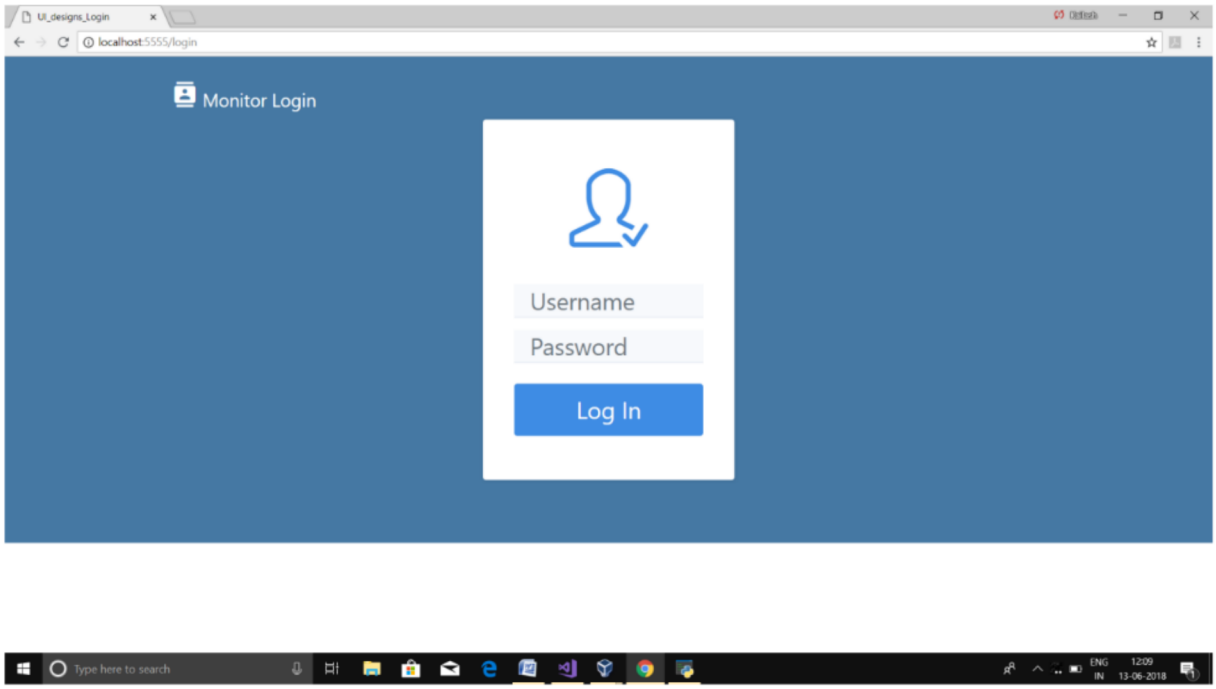
21:00 – Attendance ends

21:01- Batch Process is triggered

21:15 – Attendance Data is uploaded to the database

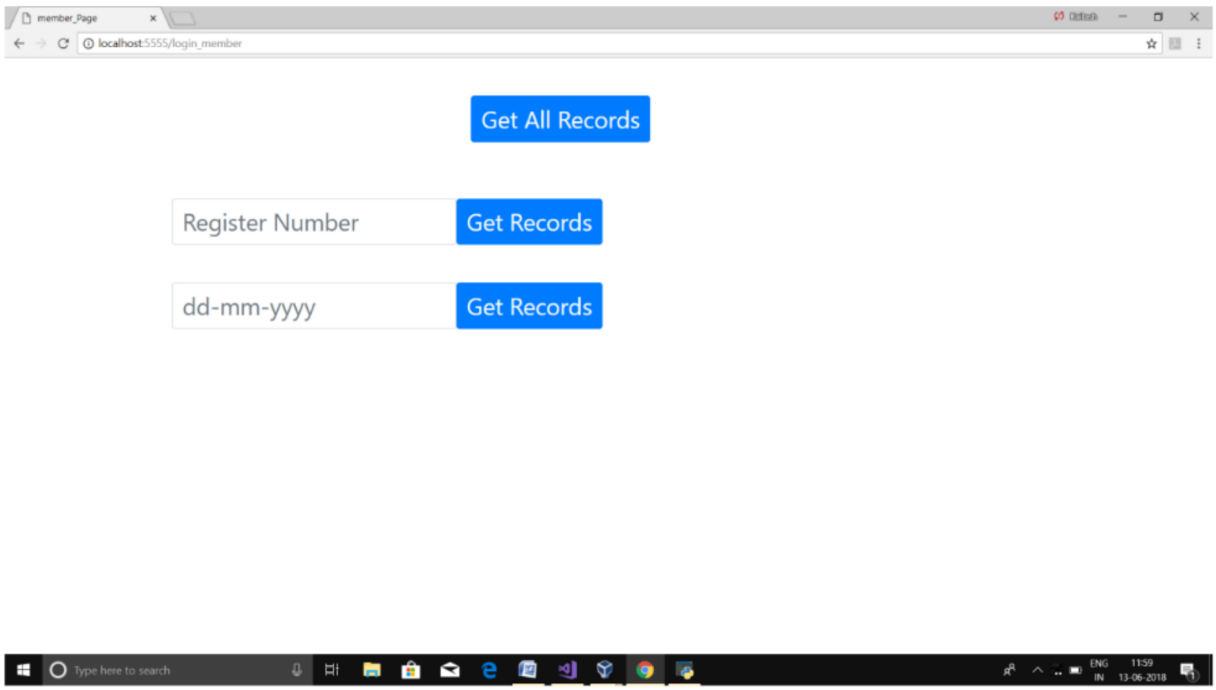
Application Workflow:

1: Login Screen



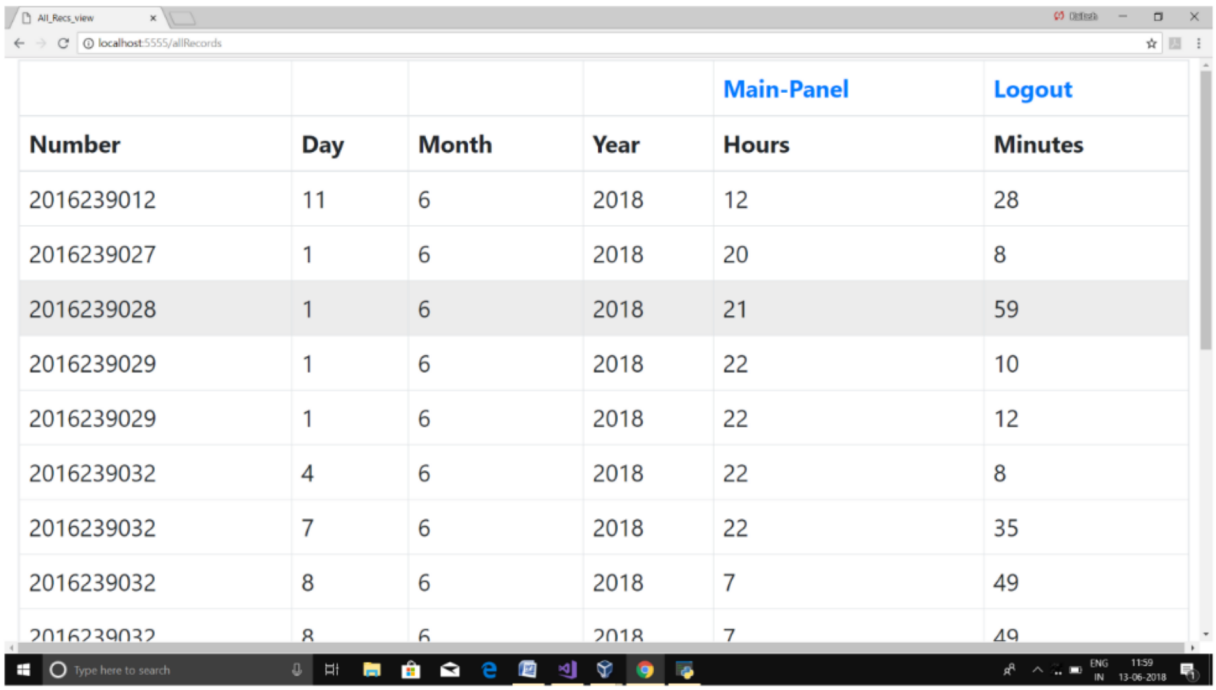
This is the login screen where the Wardens logs in.

2: Main Panel

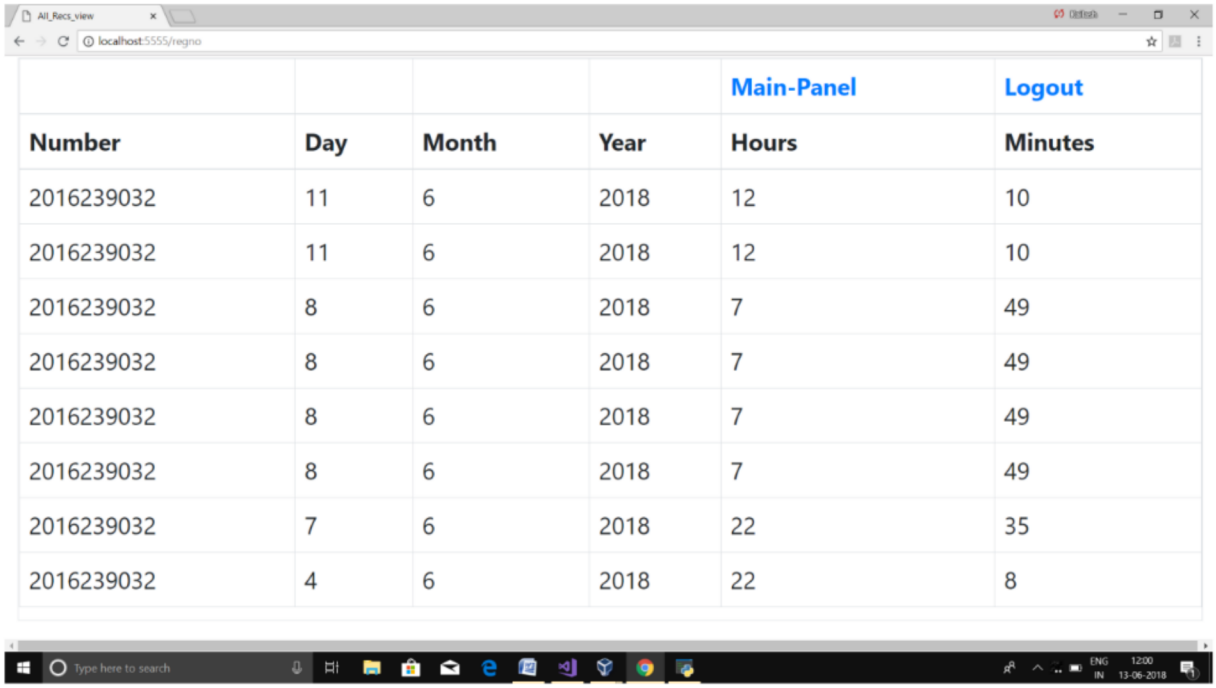


On a successful login this screen is used to filter data based on register number and date.

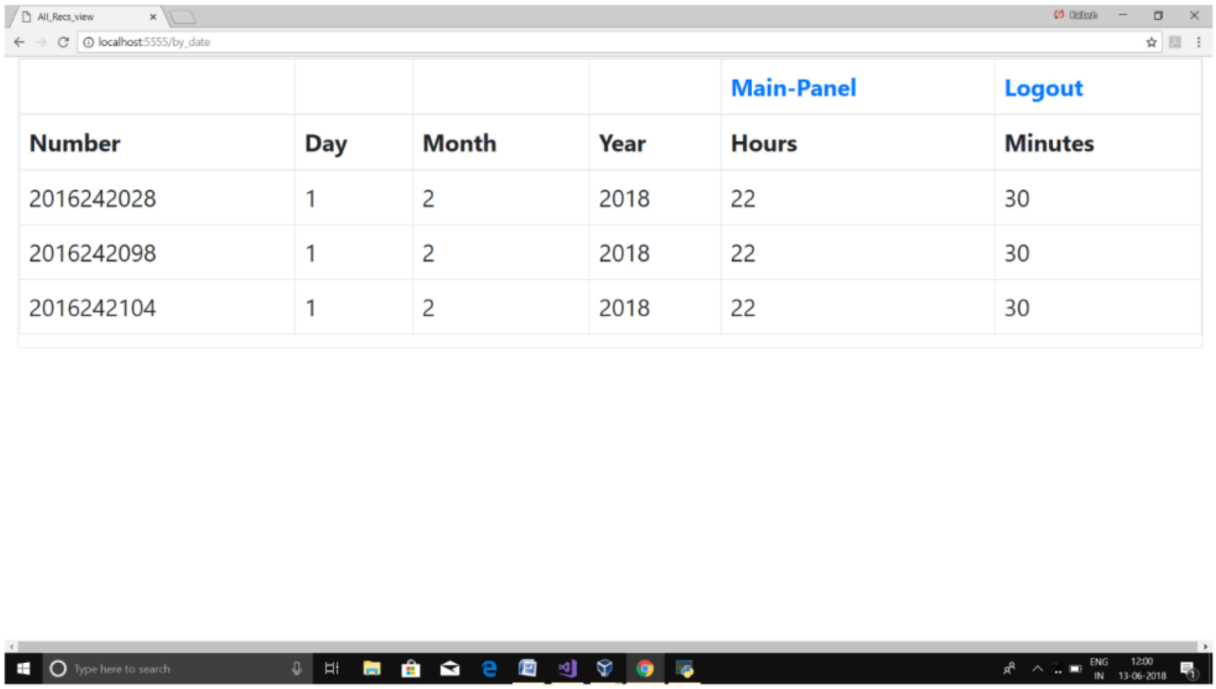
3: All Records Page:



4: Records for Particular Register Number



5: Records by Date



The Technology Stack Used:

Languages: Python, JavaScript (ES6)

Frameworks Used: Flask, ExpressJS

Database: MongoDB Community Edition

ODM: MongooseODM

Operating System: Ubuntu 16.04LTS

IDE: VisualStudio, WebStorm, PyCharm

DB Management Tools: Compass

API tool: Postman

Future Plans:

By positioning many such client cameras scattered at different places these client devices can be used to collect “whowhenwhere” data to monitor patterns on who spends how much time where. This data can be used to find the time spent at a given place by an individual (the client device locations are marked and the passage through these devices can be used). The same techniques can be used to find patterns in movements. So that in case of an intrusion this data can be used to predict the source of the intrusion and even mark unknown people in that area at that time. In a secure environment in addition to CCTV surveillance which is assisted by humans, this concept can be applied to detect and provide deeper analysis of what is going on in that location.

Mentor: Dr.SWAMYNATHAN.S

Team:

Kailash Suresh

Vignesh Kumar P

Saravanan N

Sanjay Kumar S M