University Attendance System

MAP Innovative Assignment

By

17BIT034 (Kandarp Kakkad)
17BIT046 (Siddharth Marvania)
17BIT065 (Parth Maheshwari)
17BIT075 (Parth Patel)
17BIT085 (Himanshu Prajapati)

Department of Computer Science & Engineering
Ahmedabad 382481

University Attendance System

MAP Innovative Assignment

Submitted in partial fulfillment of the requirements
For the degree of
Bachelor of Technology in Information Technology

By
17BIT034 (Kandarp Kakkad)
17BIT046 (Siddharth Marvania)
17BIT065 (Parth Maheshwari)
17BIT075 (Parth Patel)
17BIT085 (Himanshu Prajapati)

Guided By
Prof. Gaurang Raval
&
Prof. Vivek Prasad
Department of Computer Science & Engineering

NIRMA UNIVERSITY
INSTITUTE OF TECHNOLOGY
NAAC ACCREDITED 'A' GRADE

Department of Computer Science & Engineering
Ahmedabad 382481

Contents

1 Introduction

We have made a University Attendance System. It is manual attendance system which directly adds to database where the percentage update is automatic. University Attendance System is a difficult task and providing facilities in each and every field is a burdensome task. Problems like paperwork, updating attendance and modifying the attendance. Giving an effective method to communicate among teachers and students isn't a simple task.
The main objective of the project is to provide an efficient way of managing university processes. Records include student data, teacher data, attendance data etc. The goal is to provide an online system to perform tasks like inserting, fetching and manipulating the data, various online services. In this system we will be reducing the paperwork and also some offline evaluation will be done online.
As many universities have adopted university attendance systems, we are trying to make this change in Nirma University as well. This will ensure real-time attendance uploading thereby removing the teacher/professor interference. University Attendance System removes redundancy and saves time. It also reduces mistakes committed during attendance.
Our project will work as a standalone system along with automatically recording attendance, it will also provide the user interface for students, professors and administration. After completion, the project can be extended to provide functionalities like detecting prohibited activities, requesting leave applications from students with sub-par attendance and also reduce the manual efforts of creating the attendance review reports and other paperwork.

1.1 Actors

This project contains mainly 4 actors:

1. Professor
2. Student

1.1.1 Professor

Right now the problems that professors are facing is the manual attendance taking and uploading it later. In this way they have to waste their time and the university wastes its resources like paper. Also there is a chance of mistake while taking attendance or uploading it. The uploading means to re-enter the manual attendance. Also there are chances of proxy.

1.1.2 Student

The problems that students face is untimely updates of attendance, mistakes in attendance taking. Student does not know how many bunks of the class is still left before safe zone.

1.2 Functionality

1.2.1 Professor

The following are the functionalities of professor:

- The professor can take attendance.
- The professor can modify attendance.
- The professor can update their details and inform other professor if they are not available for the lecture.
- The professor can update password, achievements and authority.

1.2.2 Student

The following are the functionalities of student:

- The student can view attendance.
- The student can view attendance of a particular subject date wise.
- The student can also check if their attendance is above 85% or not.

2 API

API means “Application Programming Interface”. It is used as a medium of communication by Software Components. An API can contain any logic, object class, variables or Data Structure. Software Program can use services and resources provided by other Software by following set of rules and protocols defined by that services provider Software using APIs. APIs works similar way as UI acts as medium for Human – Computer Interaction but for Software – Software Interaction.

APIs enables content and services to be accessible on any device and platform. As these services are available to all one can use these APIs and build up something new using these so that one can concentrate on innovation rather than building same thing or services which are already available.

Example:

Like
Like
API
The Like button was at first intended to be used only on Facebook to like photos, status, comments and fan pages.
By embedding a line of code in their website, now anyone can benefit from the Like button functionality and enable their visitors to Like any content while browsing.

```
<iframe src="Some Facebook URL"          g="no" frameborder="0"
allowTransparency="true" style=          one; overflow:hidden; width:450px;
height:px">
```

Figure 1: Like Button of Facebook

Our task was to make “University Attendance System”, So as one can thought there is mainly 3 primary tasks.

1. Take Attendance
2. Modify Attendance
3. View Attendance

We have Created these three APIs in python using Flask Framework and Restful as communication protocols.

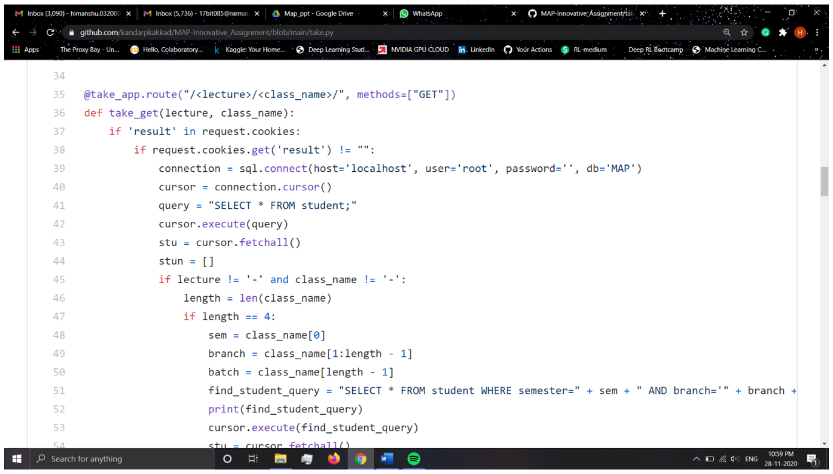
2.1 Take Attendance

This API has two services as shown in below images GET and POST.

GET service used when the faculty will click “Take Attendance” button. We can fetch subject name (lecture) and class name using current time and faculty’s Time Table. According to lecture name or code we can identify whether it is lecture, tutorial or lab (7ITB, 7ITB1, 7ITBT1) and respective query will fetch that roll numbers from database and display it.

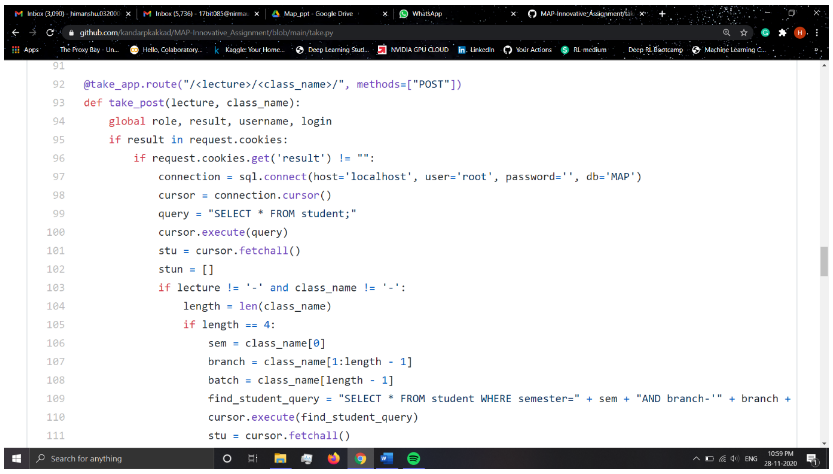
POST service come to picture when faculty is done with taking attendance and submit it. Again we already have or can derive lecture name or code and subject name and respective query will be performed and marked attendance will be reflected to database.

In case of proxy lecture faculty will be asked lecture name or code and subject name.



```python
@take_app.route("/<lecture>/<class_name>/", methods=["GET"])
def take_get(lecture, class_name):
    if 'result' in request.cookies:
        if request.cookies.get('result') != "":
            connection = sql.connect(host='localhost', user='root', password='', db='MAP')
            cursor = connection.cursor()
            query = "SELECT * FROM student;"
            cursor.execute(query)
            stu = cursor.fetchall()
            stun = []
            if lecture != '-' and class_name != '-':
                length = len(class_name)
                if length == 4:
                    sem = class_name[0]
                    branch = class_name[1:length - 1]
                    batch = class_name[length - 1]
                    find_student_query = "SELECT * FROM student WHERE semester=" + sem + " AND branch='" + branch +
                    print(find_student_query)
                    cursor.execute(find_student_query)
                    stu = cursor.fetchall()
```

Figure 2: Take API - GET



```python
@take_app.route("/<lecture>/<class_name>/", methods=["POST"])
def take_post(lecture, class_name):
    global role, result, username, login
    if result in request.cookies:
        if request.cookies.get('result') != "":
            connection = sql.connect(host='localhost', user='root', password='', db='MAP')
            cursor = connection.cursor()
            query = "SELECT * FROM student;"
            cursor.execute(query)
            stu = cursor.fetchall()
            stun = []
            if lecture != '-' and class_name != '-':
                length = len(class_name)
                if length == 4:
                    sem = class_name[0]
                    branch = class_name[1:length - 1]
                    batch = class_name[length - 1]
                    find_student_query = "SELECT * FROM student WHERE semester=" + sem + "AND branch-'" + branch +
                    cursor.execute(find_student_query)
                    stu = cursor.fetchall()
```

Figure 3: Take API - POST

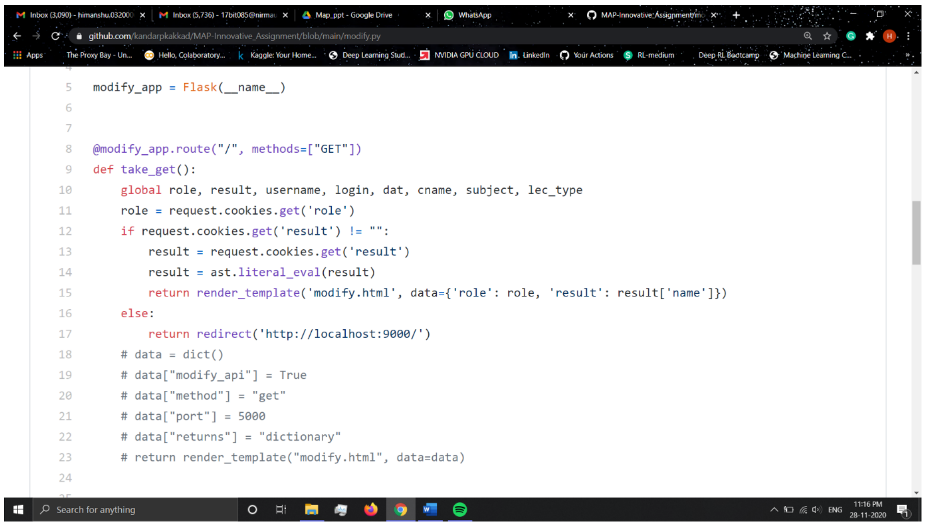
2.2 Modify Attendance

It also contains 2 services GET and POST.

GET services invokes when faculty will click “Modify Attendance” button and render page that will be used to modify attendance.
POST service does two tasks,

1. Faculty will enter all necessary details like subject name and lecture code and date and submit it then it will fetch all the students with attendance status ‘A’ and display it so that faculty can modify their status.

2. Faculty will update the status of student whichever they want then they click second button which ensures that changes will reflect in database using proper query.

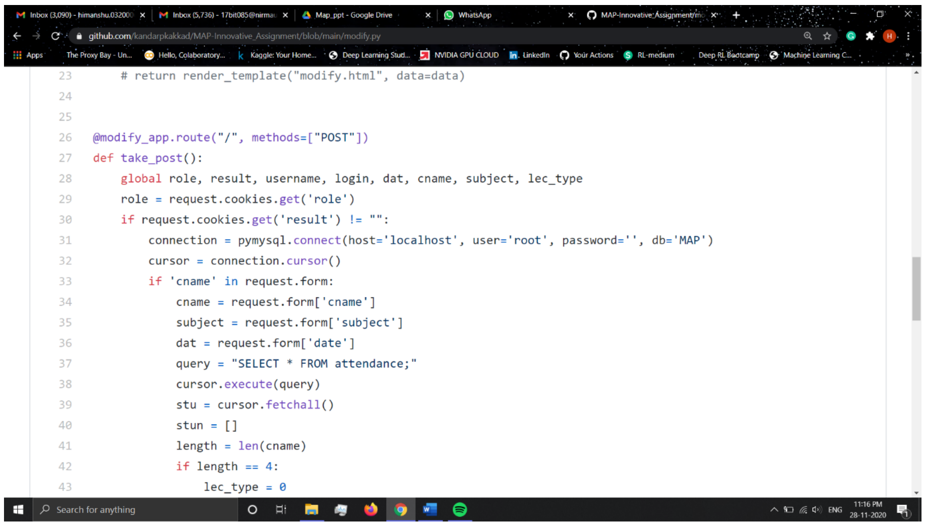


```
modify_app = Flask(__name__)


@modify_app.route("/", methods=["GET"])
def take_get():
    global role, result, username, login, dat, cname, subject, lec_type
    role = request.cookies.get('role')
    if request.cookies.get('result') != "":
        result = request.cookies.get('result')
        result = ast.literal_eval(result)
        return render_template('modify.html', data={'role': role, 'result': result['name']})
    else:
        return redirect('http://localhost:9000/')
    # data = dict()
    # data["modify_api"] = True
    # data["method"] = "get"
    # data["port"] = 5000
    # data["returns"] = "dictionary"
    # return render_template("modify.html", data=data)

```

Figure 4: Modify API - GET


github.com/kandarpkakkad/MAP-Innovative_Assignment/blob/main/modify.py

```python
    # return render_template("modify.html", data=data)


@modify_app.route("/", methods=["POST"])
def take_post():
    global role, result, username, login, dat, cname, subject, lec_type
    role = request.cookies.get('role')
    if request.cookies.get('result') != "":
        connection = pymysql.connect(host='localhost', user='root', password='', db='MAP')
        cursor = connection.cursor()
        if 'cname' in request.form:
            cname = request.form['cname']
            subject = request.form['subject']
            dat = request.form['date']
            query = "SELECT * FROM attendance;"
            cursor.execute(query)
            stu = cursor.fetchall()
            stun = []
            length = len(cname)
            if length == 4:
                lec_type = 0
```

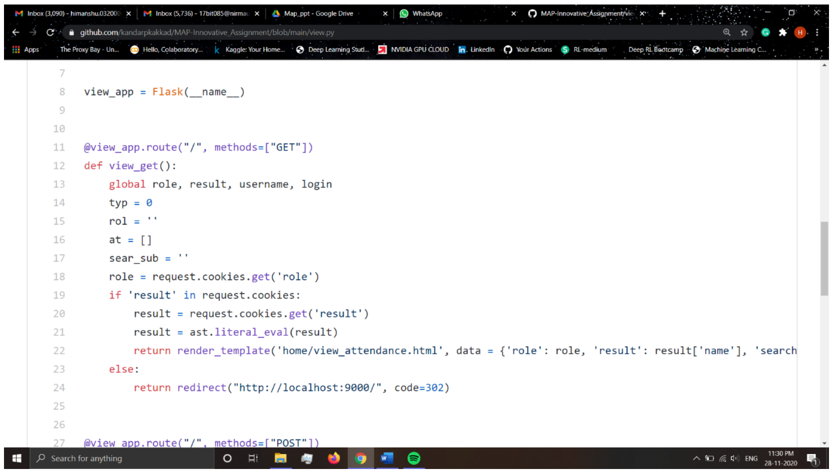
Figure 5: Modify API - POST

2.3 View Attendance

Again, it has 2 services GET and POST.

GET service will render the page asking details for showing specific attendance but the user (student) has to login first.

POST service take all the mandatory data like roll number(already have that in login credential), subject name, lecture type (lecture/lab/tutorial) and will display that attendance by fetching that data using proper query from database.



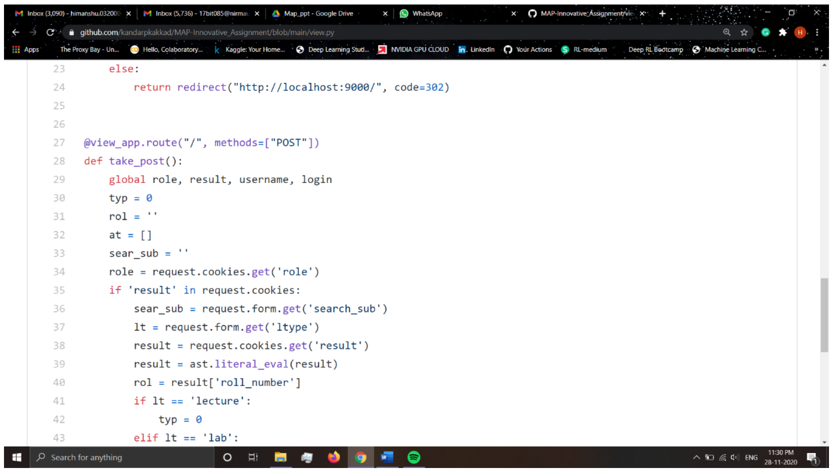
```

view_app = Flask(__name__)


@view_app.route("/", methods=["GET"])
def view_get():
    global role, result, username, login
    typ = 0
    rol = ''
    at = []
    sear_sub = ''
    role = request.cookies.get('role')
    if 'result' in request.cookies:
        result = request.cookies.get('result')
        result = ast.literal_eval(result)
        return render_template('home/view_attendance.html', data = {'role': role, 'result': result['name'], 'search
    else:
        return redirect("http://localhost:9000/", code=302)


@view_app.route("/", methods=["POST"])
```

Figure 6: View API - GET



```
    else:
        return redirect("http://localhost:9000/", code=302)


@view_app.route("/", methods=["POST"])
def take_post():
    global role, result, username, login
    typ = 0
    rol = ''
    at = []
    sear_sub = ''
    role = request.cookies.get('role')
    if 'result' in request.cookies:
        sear_sub = request.form.get('search_sub')
        lt = request.form.get('ltype')
        result = request.cookies.get('result')
        result = ast.literal_eval(result)
        rol = result['roll_number']
        if lt == 'lecture':
            typ = 0
        elif lt == 'lab':
```

Figure 7: View API - POST

3 API Gateway

An API gateway is an API management tool that sits between a client and a collection of backend services.

An API gateway acts as a reverse proxy to accept all application programming interface (API) calls, aggregate the various services required to fulfill them, and return the appropriate result.

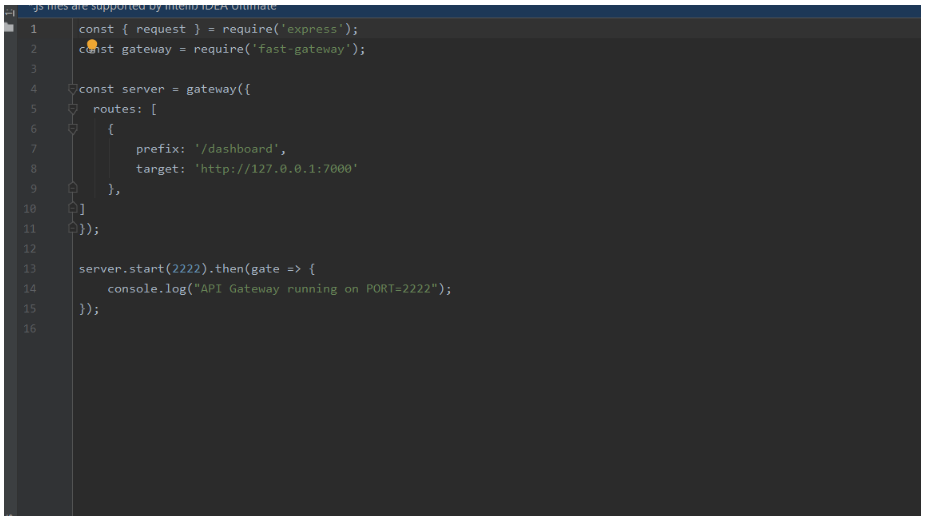
Most enterprise APIs are deployed via API gateways. It's common for API gateways to handle common tasks that are used across a system of API services, such as user authentication, rate limiting, and statistics.

For most microservices-based applications, it bodes well to execute an API gateway, since it goes about as a solitary section point into the system. The API gateway is liable for request routing and protocol translation, and can smooth out the system. With an API gateway, every one of the application's customers gets a custom API. The API gateway handles a few demands by basically steering them to the proper backend service, and handles others by summoning various backend services and aggregating the outcomes. On the off chance that there are failure in the backend service, the API gateway can veil them by returning cached or default information.

We have created two API gateway:

- Login gateway
- Home Gateway

Our Login Gateway runs on port number 2222 and is directed to the dashboard of the user. The code of the gateway is below:



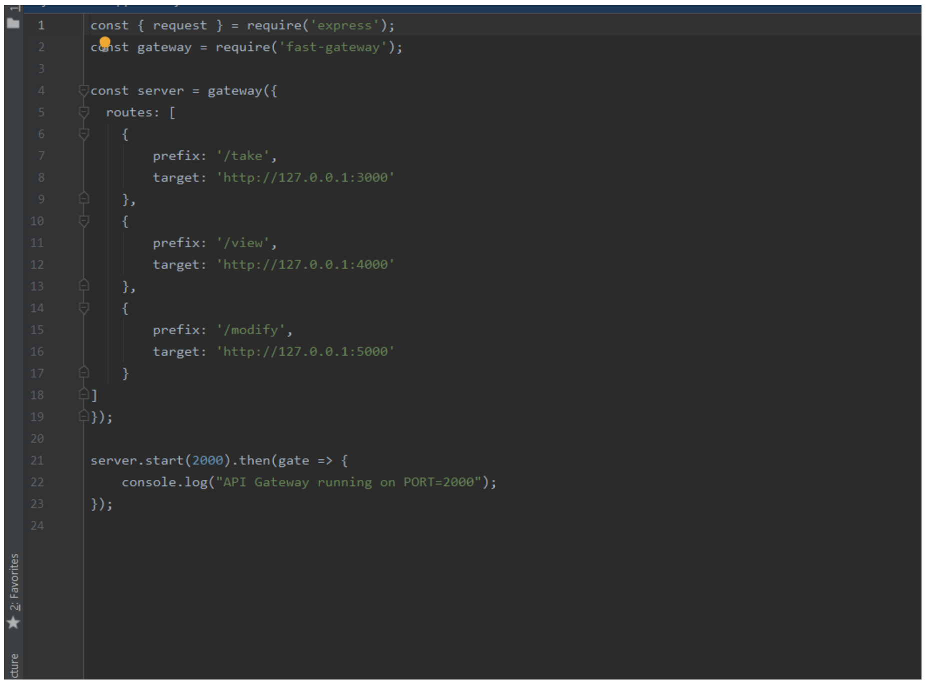
```
const { request } = require('express');
const gateway = require('fast-gateway');

const server = gateway({
  routes: [
    {
        prefix: '/dashboard',
        target: 'http://127.0.0.1:7000'
    },
]
});

server.start(2222).then(gate => {
    console.log("API Gateway running on PORT=2222");
});
```

Figure 8: Login Gateway

Home Gateway runs on port number 2000 and through which it would redirect it to the take attendance service if the user clicks on the take attendance button. If user clicks on the “Modify Attendance” then it would redirect it to the modify attendance microservice. If user clicks on the “View Attendance” the it would redirect the user to the View attendance micro service. The code for the the Home gateway is given below:



```
const { request } = require('express');
const gateway = require('fast-gateway');

const server = gateway({
  routes: [
    {
        prefix: '/take',
        target: 'http://127.0.0.1:3000'
    },
    {
        prefix: '/view',
        target: 'http://127.0.0.1:4000'
    },
    {
        prefix: '/modify',
        target: 'http://127.0.0.1:5000'
    }
]
});

server.start(2000).then(gate => {
    console.log("API Gateway running on PORT=2000");
});
```

Figure 9: Home Gateway

4 RabbitMQ

RabbitMQ is lightweight and simple to convey on-premises and in the cloud. It underpins various informing conventions. RabbitMQ can be conveyed in disseminated and united designs to meet high-scale, high-accessibility necessities.

RabbitMQ runs on many working frameworks and cloud conditions and gives a wide scope of designer instruments for most mainstream dialects.

We have used RabbitMQ in between server and database. Before updating the database the data goes of MQ and then it commits in the database. We don't have a messaging module, so we used RabbitMQ in this way.

5 Installation & Running Steps

5.1 Install Dependencies

In this project we have built an api gateway using node js and each api is built using flask/python.
$ *pip*3 *install -r requirements.txt* $ *npm install*

We also need to start XAMPP server and create a database named ”MAP” and import the database ”map.sql”.

5.2 Running Steps

First we have to run the api gateway in the terminal using the following command

```
$ node home_gateway.js
$ node login_gateway.js
```

Here we have two separate gateway for login and main logic.

Then first activate your python environment and run the following command in separate command prompt

```
$ python index.py
$ python dashboard.py
$ python take.py
$ python view.py
$ python modify.py
```

6 Output

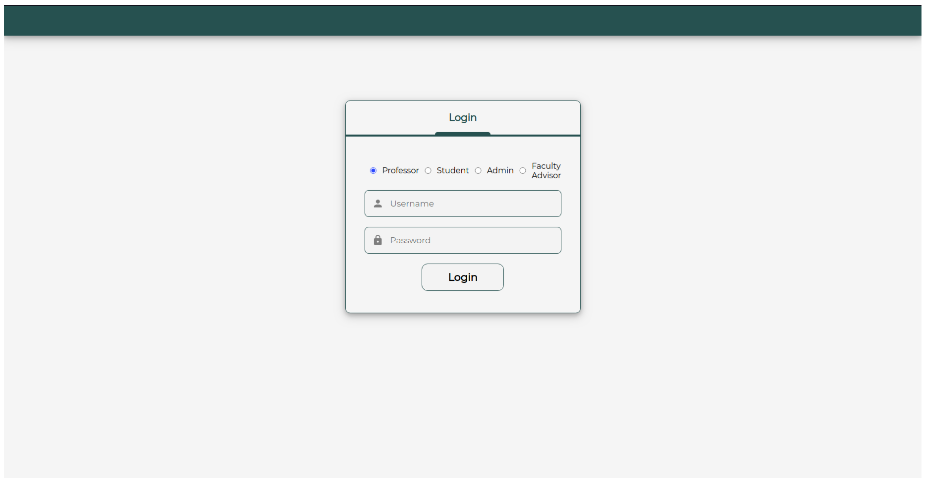
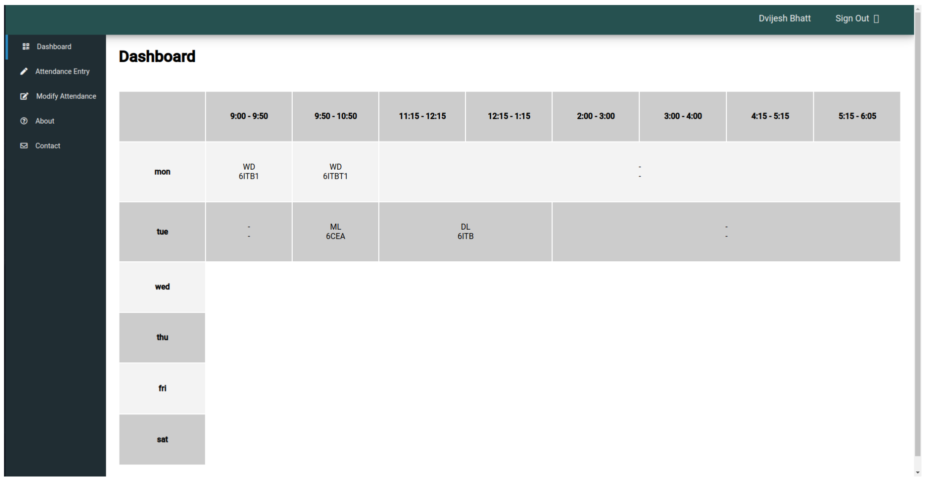


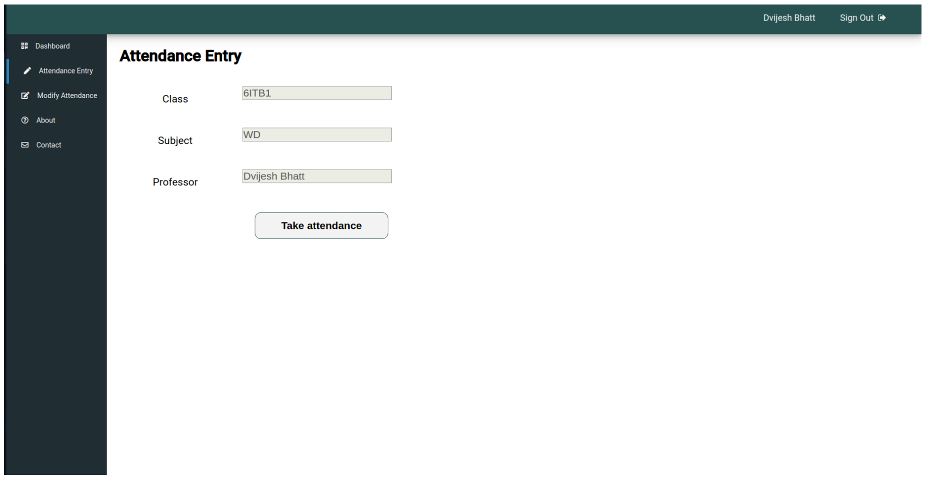
Figure 10: Login Page



Dashboard

	9:00 - 9:50	9:50 - 10:50	11:15 - 12:15	12:15 - 1:15	2:00 - 3:00	3:00 - 4:00	4:15 - 5:15	5:15 - 6:05
mon	WD 6ITB1	WD 6ITBT1	- -					
tue	- -	ML 6CEA	DL 6ITB		- -			
wed								
thu								
fri								
sat								

Figure 11: Professor Home Page



Attendance Entry

Class	6ITB1
Subject	WD
Professor	Dvijesh Bhatt

Figure 12: Take Attendance From Time Table

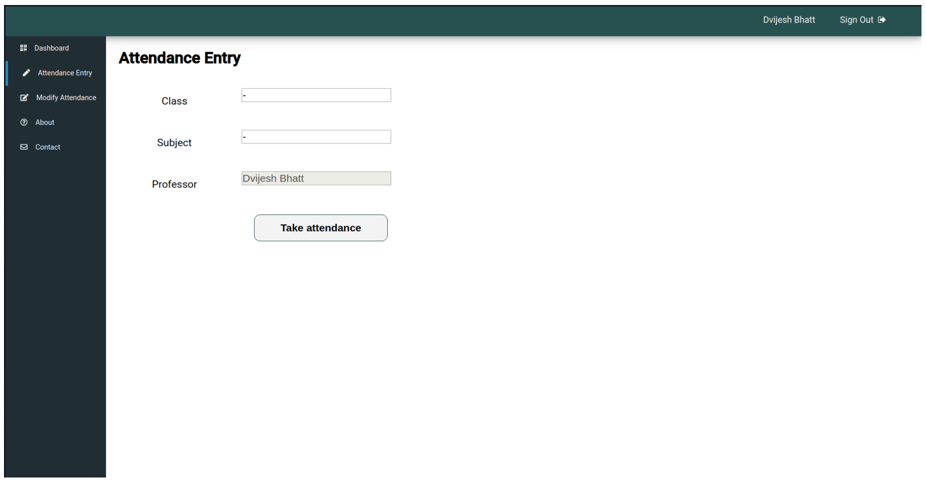


Figure 13: Take Attendance For Proxy

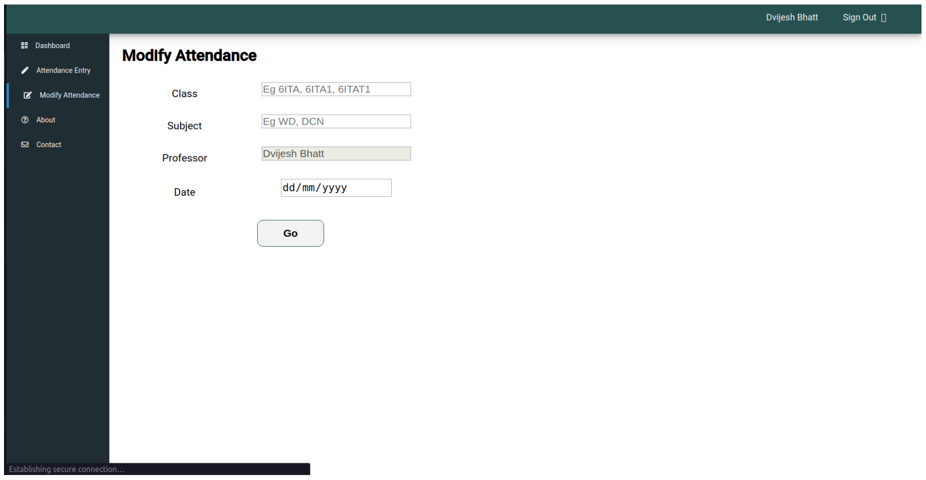
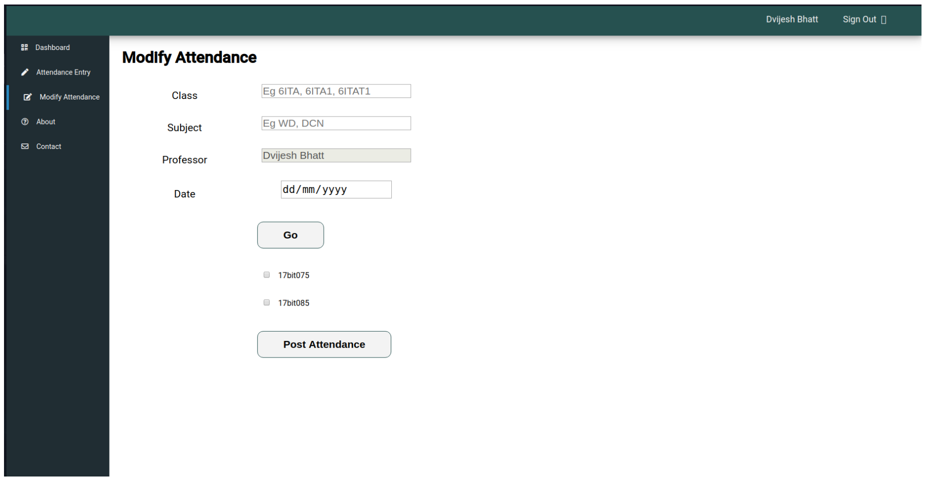


Figure 14: Modify Attendance Part 1



Modify Attendance

Class: Eg 6ITA, 6ITA1, 6ITAT1

Subject: Eg WD, DCN

Professor: Dvijesh Bhatt

Date: dd/mm/yyyy

17bit075

17bit085

Figure 15: Modify Attendance Part 2

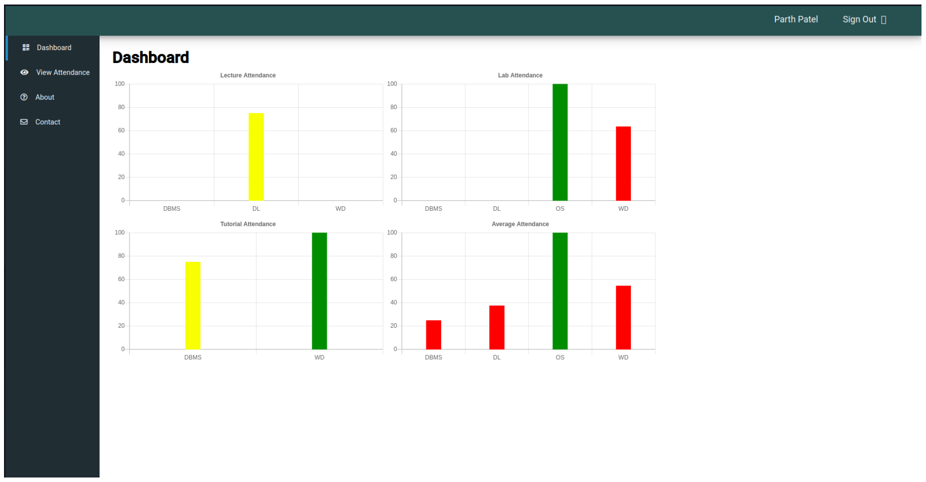


Figure 16: Student Dashboard

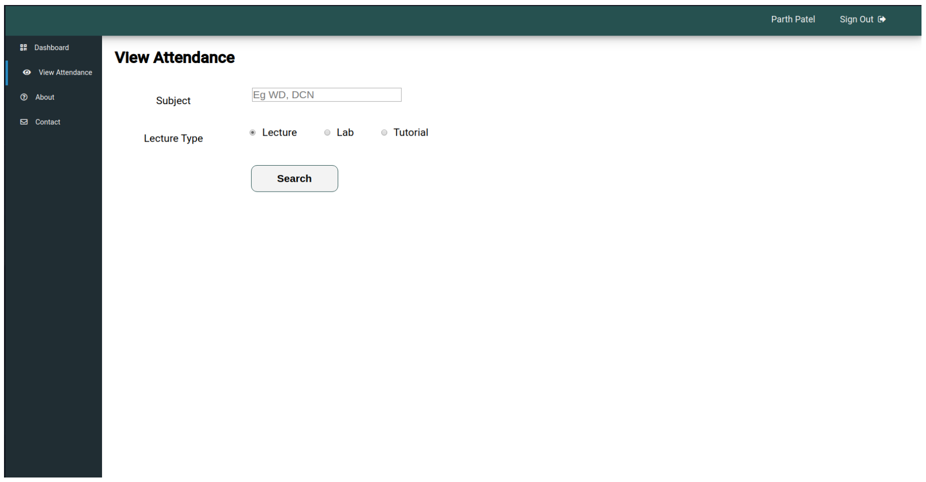
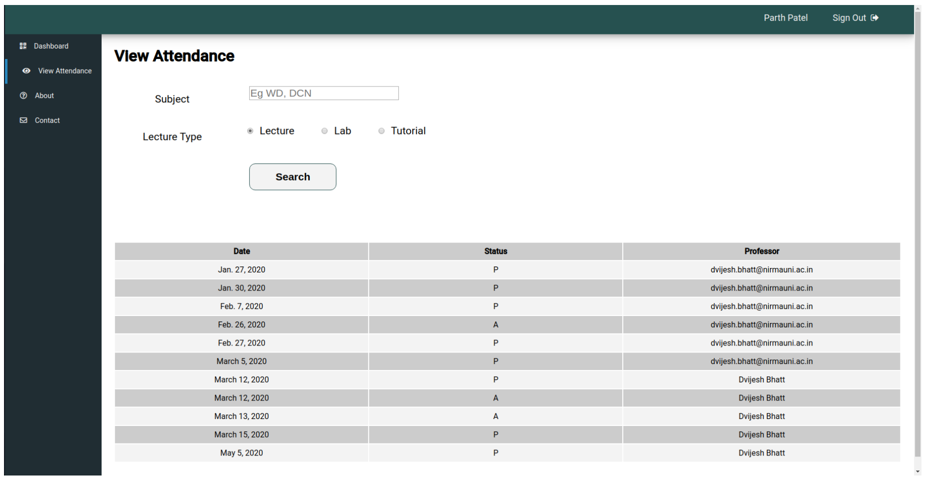


Figure 17: View Attendance Part 1



Date	Status	Professor
Jan. 27, 2020	P	dvijesh.bhatt@nirmauni.ac.in
Jan. 30, 2020	P	dvijesh.bhatt@nirmauni.ac.in
Feb. 7, 2020	P	dvijesh.bhatt@nirmauni.ac.in
Feb. 26, 2020	A	dvijesh.bhatt@nirmauni.ac.in
Feb. 27, 2020	P	dvijesh.bhatt@nirmauni.ac.in
March 5, 2020	P	dvijesh.bhatt@nirmauni.ac.in
March 12, 2020	P	Dvijesh Bhatt
March 12, 2020	A	Dvijesh Bhatt
March 13, 2020	A	Dvijesh Bhatt
March 15, 2020	P	Dvijesh Bhatt
May 5, 2020	P	Dvijesh Bhatt

Figure 18: View Attendance Part 2

Conclusion

We learnt about about different technologies and to make a final system. We have used APIs, API Gateway and RabbitMQ.