Institutional Management System

A Data Management in Python project

Chandra Shekar Srinivasaiah ***Abhishek Mathur***

Contents:

Introduction:

The goal of this project would be to create a management system for an Institution where the necessary data including students’ data (undergraduate and postgraduate), faculty data and employee data are stored and managed efficiently using microframework Flask, for this project. Flask is one of the most popular web application frameworks written in Python. It is a microframework designed for an easy and quick start.

Using Flask, we would be able to store the data (SQLite) in a database, clean the data (using pandas) and to manage the data (Create, Retrieve, Update, Delete) using security features which ensures that administrator alone and few authorized personnels are allowed to have complete control over the data. Remaining users will be only able to view the data.

Technologies Used:

- Python
- Flask framework
- SQlite3
- html5
- CSS
- jinja2
- Pandas

Installation:

Before we can use the code, the packages have to be installed via requirements.txt:

```
pandas~=1.2.4
numpy~=1.20.1
flask~=0.11.1
wtforms~=2.1
werkzeug~=2.0.2
sqlalchemy~=1.1.4
alembic~=0.8.9
```

In case of linux, follow the below steps:

```
sudo apt install python3-venv ## incase if not installed already
mkdir DMP_CRUD_APP
cd DMP_CRUD_APP
python3 -m venv venv
pip install -r requirements.txt
source venv/bin/activate
export FLASK_APP=app.py
flask run
(venv) deactivate ## to deactivate virtual env
```

In case of windows, follow the below steps:

```
pip install python3-venv
mkdir DMP_CRUD_APP
cd DMP_CRUD_APP
python3 -m venv venv
pip install -r requirements.txt
venv/bin/activate.bat
export FLASK_APP=app.py
flask run
(venv) deactivate.bat ## to deactivate virtual env
```

The user shall be able to run the flask application in case of above steps and will be able to access the contents on *http://localhost:5000/*

About Data and its processing:

Flask has support for several relational database management systems (RDBMS), including SQLite, MySQL, and PostgreSQL. For this project, we will be using SQLite3. It's popular and therefore has a support, in addition to being scalable, secure, and rich in features.

We have chosen SQLite database, one can apply the same knowledge with any other database also like Mysql or others.

The data used for the purpose of this project has demographic data.

- Undergraduate and postgraduate student level data has Demographic data including age, sex, address, highest qualification, etc.
- Faculty level data has Experience, name, joining date, address, etc.
- Administration level data has login and sign-in features to maintain the student, faculty and employee data.

These qualitative and quantitative data are structured and stored in the database under the name ***data.db***. Processing the data including cleaning, extraction, access follows further in this document.

```
import sys, os
from os import abort
from flask_login import login_required, current_user, login_user, logout_user
from flask import Flask, render_template, request, redirect, url_for
from models import db, EmployeeModel, login, UserModel, UGStudentsModel, PGStudentsModel,
FacultyModel

import pandas as pd
import numpy as np
import sqlite3

app = Flask(__name__)
app.secret_key = 'xyz'

# Upload folder
UPLOAD_FOLDER = 'static/files'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///data.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db.init_app(app)
login.init_app(app)
login.login_view = 'login'


conn = sqlite3.connect('data.db')
print("Opened database successfully", file=sys.stderr)
cur = conn.cursor()
conn.execute('''SELECT * from UGStudents''')
rows = cur.fetchall()
print(rows, "abcd", file=sys.stderr)
conn.close()


@app.before_first_request
def create_table():
    db.create_all()
```

A model is **a Python class that inherits from the Model class**. The model class defines a new Kind of datastore entity and the properties the Kind is expected to take. The name is defined by the instantiated class name that inherits from db. Model.

We use Models for UserData, UGStudents, PGStudents, FacultyData and Employee for out project.

- UserModel for storing users data (with email and hashed passwords)

- Students model (separate model for UGStudents and PGStudents)
- Faculty model for storing faculty data
- Employee model for storing employee data

Once the above models are created, they are initialised and can be used to execute CRUD operations.

Security features:

Register, signup and login:

We will need model for Security related feature for secure login.

We make use of some of security helper methods, generate_password_hash, which allows us to hash passwords, and check_password_hash. This allows us to ensure the hashed password matches the password. To enhance the security, we have a password method that ensures that the password can never be accessed and instead, an error will be raised.

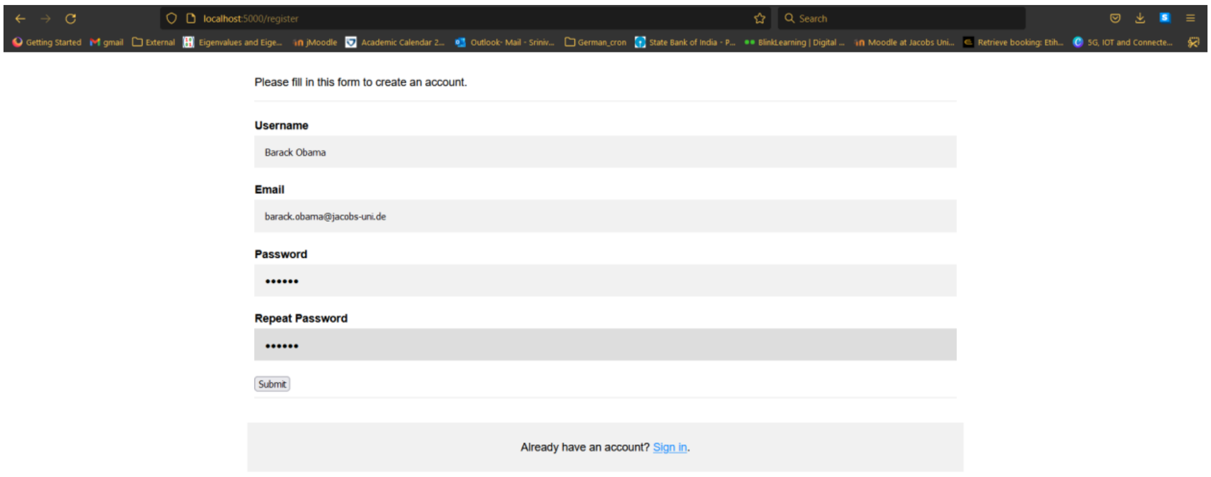
model.py

```python
class UserModel(UserMixin, db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(80), unique=True)
    username = db.Column(db.String(100))
    password_hash = db.Column(db.String())

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)
```

app.py

```python
@app.route('/register', methods=['POST', 'GET'])
def register():
    if current_user.is_authenticated:
        return redirect('/homepage')
    if request.method == 'POST':
        email = request.form['email']
        username = request.form['username']
        password = request.form['password']
        if UserModel.query.filter_by(email=email).first():
            return ('Email already Present')
        user = UserModel(email=email, username=username)
        user.set_password(password)
        db.session.add(user)
        db.session.commit()
        return redirect('/login')
    return render_template('register.html')


@app.route('/logout')
def logout():
    logout_user()
    return redirect('/homepage')
```

Implementing the above results in the following, for registering the users:


localhost:5000/register

Please fill in this form to create an account.

Username

Barack Obama

Email

barack.obama@jacobs-uni.de

Password

••••••

Repeat Password

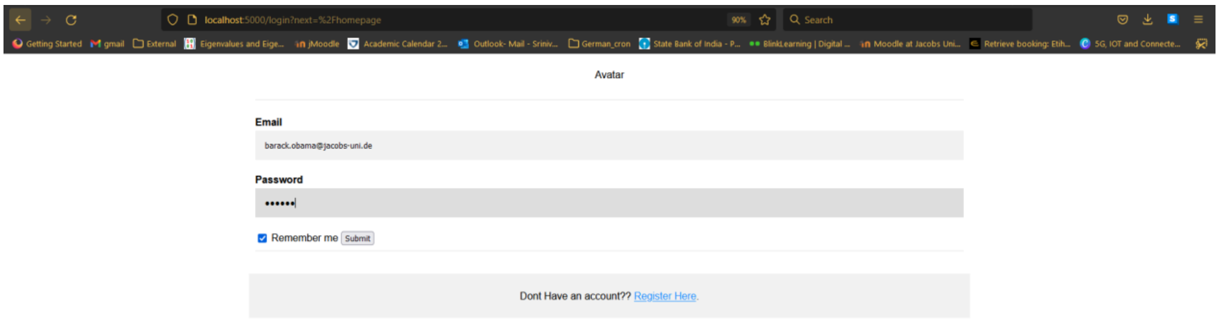
••••••

Submit

Already have an account? Sign in.

```
@app.route('/login', methods=['POST', 'GET'])
def login():
    if current_user.is_authenticated:
        return redirect('/homepage')
    if request.method == 'POST':
        email = request.form['email']
        user = UserModel.query.filter_by(email=email).first()
        if user is not None and user.check_password(request.form['password']):
            login_user(user)
            return redirect('/homepage')

    return render_template('login.html')
```

Implementing the above results in the following, the users to login:


localhost:5000/login?next=%2Fhomepage

Avatar

Email

barack.obama@jacobs-uni.de

Password

••••••

[✓] Remember me Submit

Dont Have an account?? Register Here.

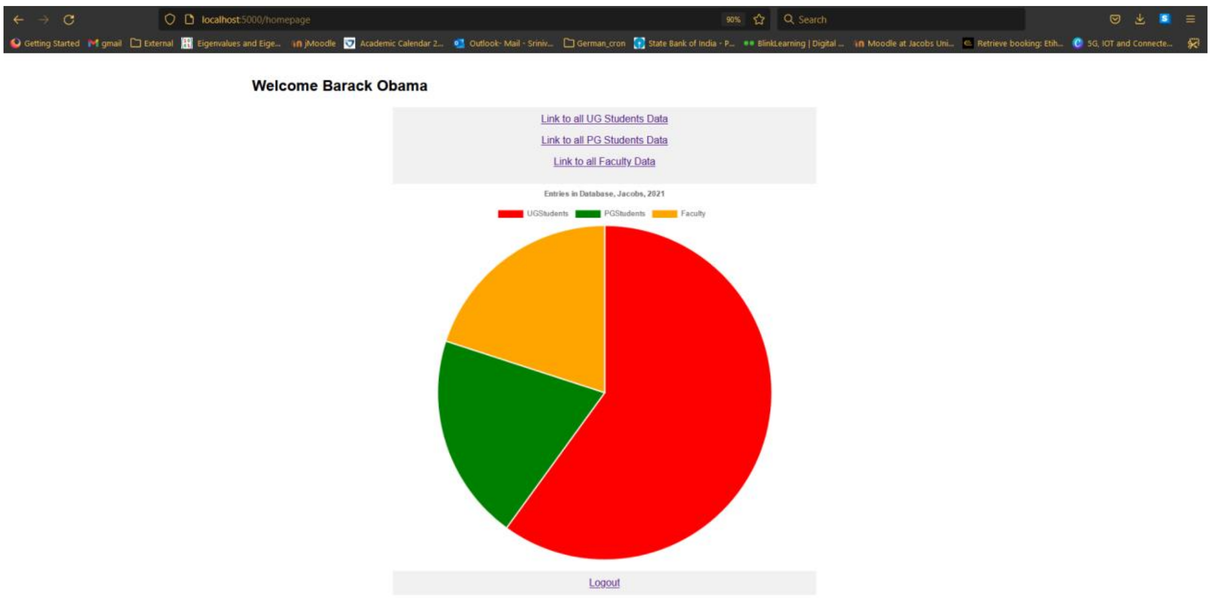
CRUD operations:

Homepage:

This would be the main page once the recognised user is registered and logged in. As soon as the user logs in, the analysis (in the form of a graph) for all the data in *data.db* (UGStudents, PGStudents and Faculty) is provided. A separate page shall be provided for employees working in the University.

```
@app.route('/homepage')
@login_required
def hpmepage():
    ug_len = UGStudentsModel.query.count()
    pg_len = PGStudentsModel.query.count()
    fac_len = FacultyModel.query.count()
    emp_len = EmployeeModel.query.count()
    data = [ug_len, pg_len, fac_len, emp_len]
    # print(data, file=sys.stderr)
    return render_template('homepage.html', data=data)
```

Homepage.html, provides the overview for all length of entries present in the database for UGStudents, PGStudents and Faculty.


localhost:5000/homepage

Welcome Barack Obama

Link to all UG Students Data

Link to all PG Students Data

Link to all Faculty Data

Logout

model.py for UGStudents model:

```
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import UserMixin,LoginManager

login = LoginManager()
db = SQLAlchemy()


class UGStudentsModel(db.Model):
    __tablename__ = "UGStudents"

    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer(), unique=True)
    primary_name = db.Column(db.String())
    middle_name = db.Column(db.String())
    last_name = db.Column(db.String())
    contact_num = db.Column(db.String())
    primary_email = db.Column(db.String())
    secondary_email = db.Column(db.String())
    uni_email = db.Column(db.String())
    age = db.Column(db.Integer())
    sex = db.Column(db.String())
    address = db.Column(db.String(80))
    admission_year = db.Column(db.Integer())
    highest_qual = db.Column(db.String())

    def __init__(self, student_id, primary_name, middle_name, last_name, age, contact_num,
                 primary_email, secondary_email, uni_email, sex, address, admission_year,
highest_qual):
        self.student_id = student_id
        self.primary_name = primary_name
        self.middle_name = middle_name
        self.last_name = last_name
        self.age = age
        self.contact_num = contact_num
        self.primary_email = primary_email
        self.secondary_email = secondary_email
        self.uni_email = uni_email
        self.sex = sex
        self.address = address
        self.admission_year = admission_year
        self.highest_qual = highest_qual

    def __repr__(self):
        return f"{self.last_name}:{self.student_id}"
```

Note: *A similar model for PG Students, Faculty and employees working for the university is present on* ***models.py***

Creating the data using models:

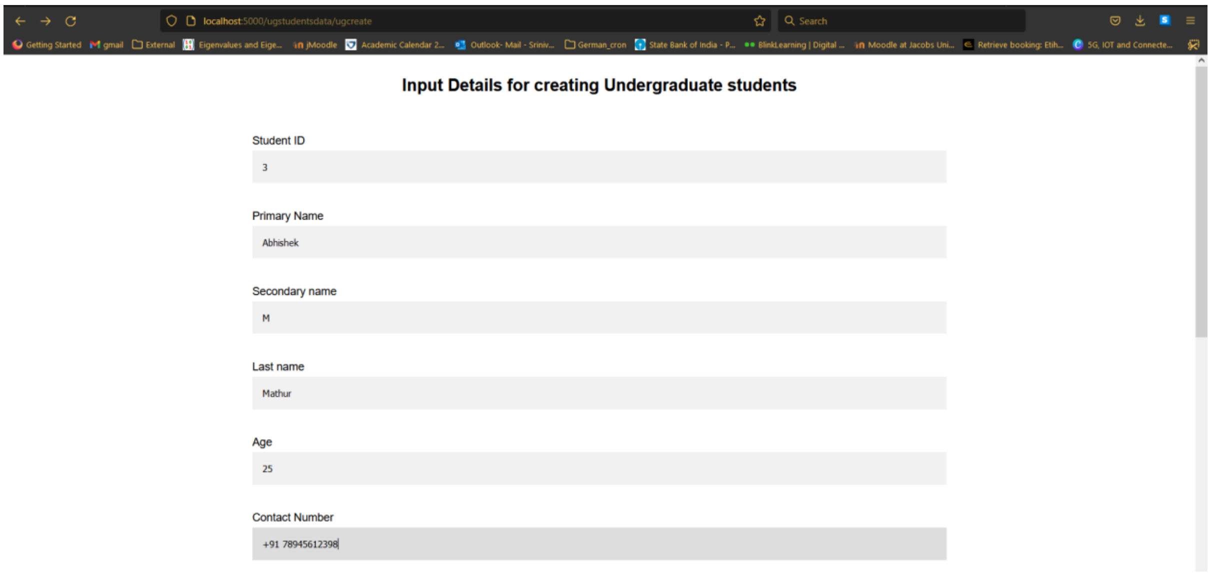
The Create view should be able to do the following:

- When the Client goes to this page (GET method), it should display a Form to get the Client's Data.
- On Submission (POST method), it should save the Client's data in the model Database.

app.py: contains the code to create the entry in the table "UGStudents" under "data.db". The model is initialised using the input form data from html and model for unique instance ID is created at a time as follows:

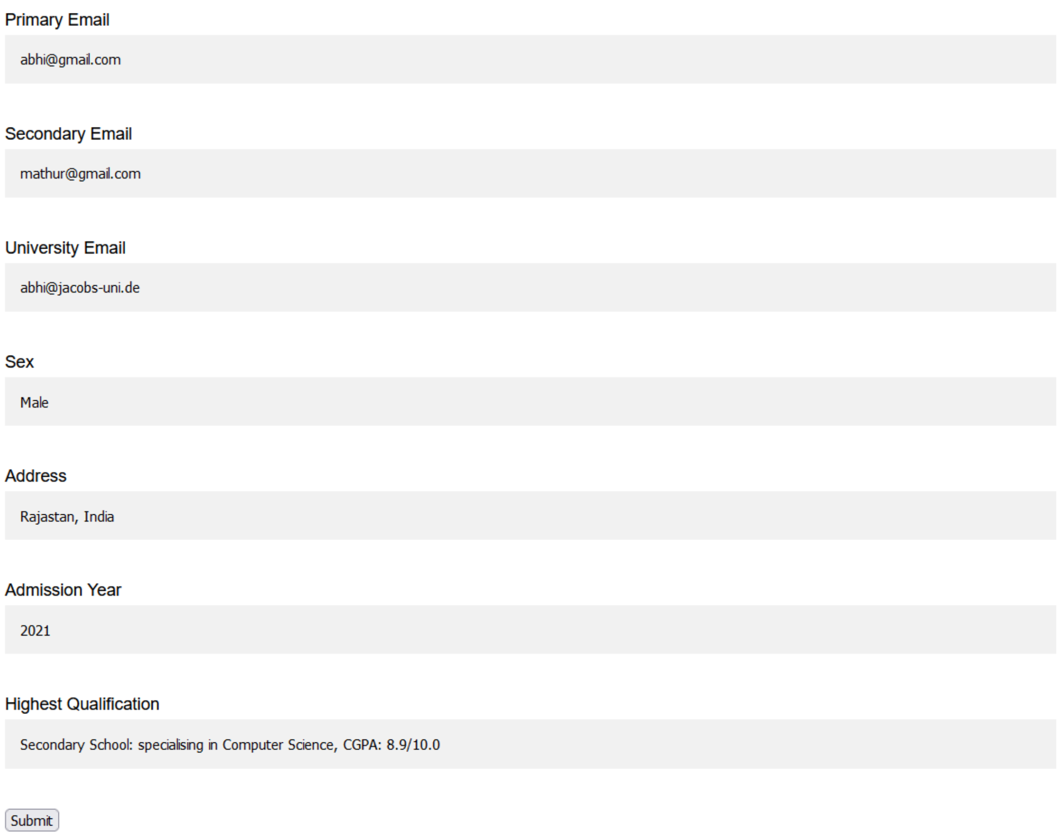
```
@app.route('/ugstudentsdata/ugcreate', methods=['GET', 'POST'])
def ugcreate():
    if request.method == 'GET':
        return render_template('ugCreatePage.html')

    if request.method == 'POST':
        student_id = request.form.get('student_id', '')
        primary_name = request.form.get('primary_name', '')
        middle_name = request.form.get('middle_name', '')
        last_name = request.form.get('last_name', '')
        age = request.form.get('age', '')
        contact_num = request.form.get('contact_num', '')
        primary_email = request.form.get('primary_email', '')
        secondary_email = request.form.get('secondary_email', '')
        uni_email = request.form.get('uni_email', '')
        sex = request.form.get('sex', '')
        address = request.form.get('address', '')
        admission_year = request.form.get('admission_year', '')
        highest_qual = request.form.get('highest_qual', '')
        ugstudents = UGStudentsModel(student_id=student_id, primary_name=primary_name,
middle_name=middle_name, last_name=last_name, age=age, contact_num=contact_num,
primary_email=primary_email, secondary_email=secondary_email, uni_email=uni_email, sex=sex,
address=address, admission_year=admission_year, highest_qual=highest_qual)
        db.session.add(ugstudents)
        db.session.commit()
        return redirect('/ugstudentsdata')
```

HTML page for creating UG Students data with all the fields:



Input Details for creating Undergraduate students

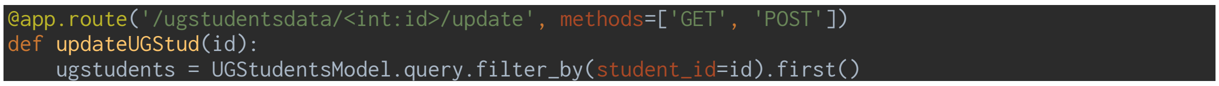
Student ID	3
Primary Name	Abhishek
Secondary name	M
Last name	Mathur
Age	25
Contact Number	+91 78945612398



Primary Email	abhi@gmail.com
Secondary Email	mathur@gmail.com
University Email	abhi@jacobs-uni.de
Sex	Male
Address	Rajastan, India
Admission Year	2021
Highest Qualification	Secondary School: specialising in Computer Science, CGPA: 8.9/10.0

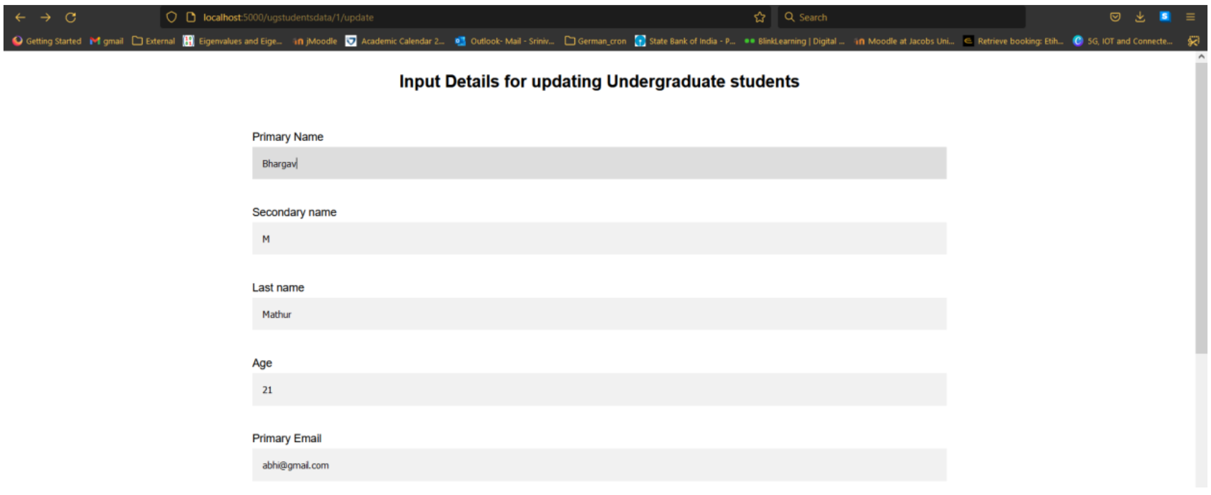
Updating the models:

This updates the existing database entry using ID identifier. The user will submit the new details via the Form. Here we first delete (if any) the old information present in the DB and then add the new information. The existing model is fetched using unique identifier and updated on the existing mode using the following:



```
@app.route('/ugstudentsdata/<int:id>/update', methods=['GET', 'POST'])
def updateUGStud(id):
    ugstudents = UGStudentsModel.query.filter_by(student_id=id).first()
    if request.method == 'POST':
        if ugstudents:
            db.session.delete(ugstudents)
            db.session.commit()
            primary_name = request.form.get('primary_name', '')
            middle_name = request.form.get('middle_name', '')
            last_name = request.form.get('last_name', '')
            age = request.form.get('age', '')
            contact_num = request.form.get('contact_num', '')
            primary_email = request.form.get('primary_email', '')
            secondary_email = request.form.get('secondary_email', '')
            uni_email = request.form.get('uni_email', '')
            sex = request.form.get('sex', '')
            address = request.form.get('address', '')
            admission_year = request.form.get('admission_year', '')
            highest_qual = request.form.get('highest_qual', '')
            ugstudents = UGStudentsModel(student_id=id, primary_name=primary_name,
middle_name=middle_name, last_name=last_name, age=age, contact_num=contact_num,
primary_email=primary_email, secondary_email=secondary_email, uni_email=uni_email, sex=sex,
address=address, admission_year=admission_year, highest_qual=highest_qual)
            db.session.add(ugstudents)
            db.session.commit()
            return redirect(f'/ugstudentsdata/{id}')
        return f"UG Student with id = {id} Does not exist"

    return render_template('updateUGData.html', ugstudents=ugstudents)
```

HTML for receiving the updated contents for UGStudents (for example id:1). Here all the fields are given the option to update except the unique ID and displayed as follows:



Input Details for updating Undergraduate students

Primary Name
Bhargav

Secondary name
M

Last name
Mathur

Age
21

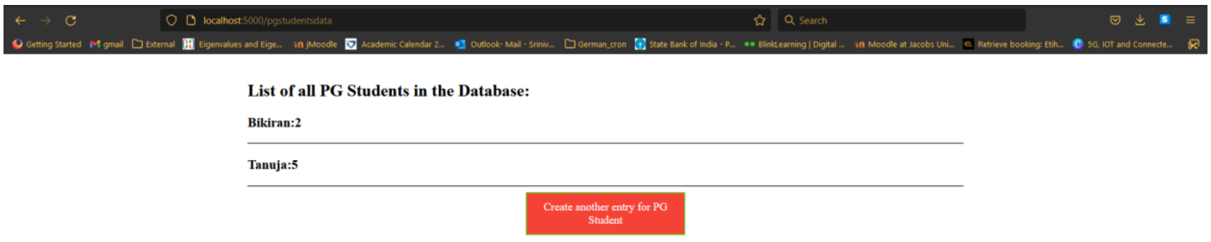
Primary Email
abhi@gmail.com

Retrieve Data from the Database:

This is an important operation because this belongs to Select query and has a lot more verity in fetching the records from the database. The following retrieves all the information for PGStudents:

```
@app.route('/pgstudentsdata')
def RetrievePGStudList():
    pgstudents = PGStudentsModel.query.all()
    return render_template('pgStudsDataList.html', pgstudents=pgstudents)
```

Retrieving the data for all postgraduates:


localhost:5000/pgstudentsdata

List of all PG Students in the Database:

Bikiran:2

Tanuja:5

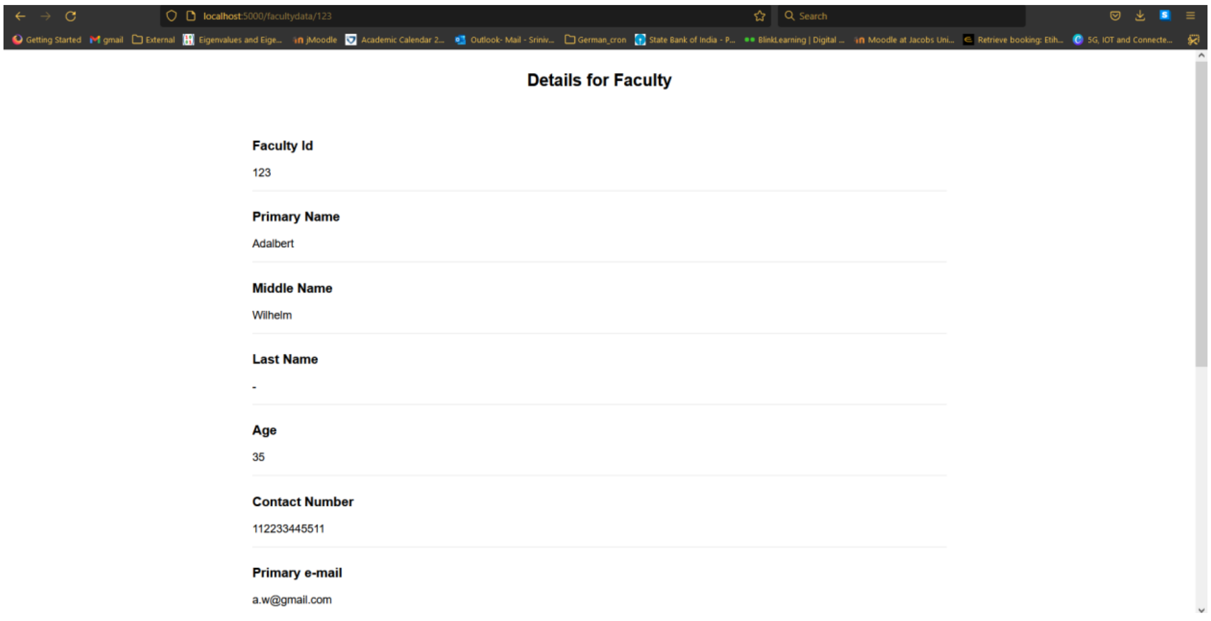
Create another entry for PG Student

Retrieving the data using ID (for specified individual):

Earlier we had used to retrieve all the entries for a specific model, but in order to read the complete entry for a specific user (or ID for that matter), we use the following:

```python
@app.route('/facultydata/<int:id>')
def RetrieveFaculty(id):
    faculty = FacultyModel.query.filter_by(faculty_id=id).first()
    if faculty:
        return render_template('facultyData.html', faculty=faculty)
    return f"Faculty with id ={id} Doesn't exist"
```

Retrieving the entry for faculty ID 123:



Details for Faculty

Faculty Id
123

Primary Name
Adalbert

Middle Name
Wilhelm

Last Name
-

Age
35

Contact Number
112233445511

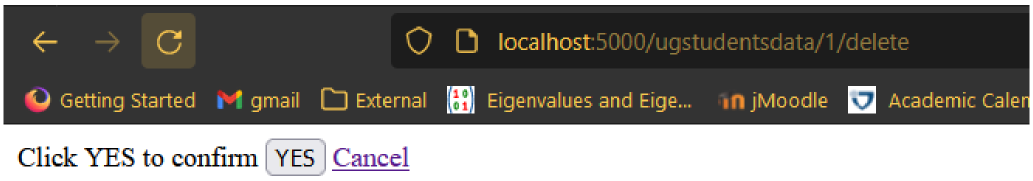
Primary e-mail
a.w@gmail.com

Deleting the entry from the Database:

Deletes the entry from the database in case if the administrator wants to delete the entry for various reasons:

```python
@app.route('/ugstudentsdata/<int:id>/delete', methods=['GET', 'POST'])
def deleteUGStudent(id):
    ugstudent = UGStudentsModel.query.filter_by(student_id=id).first()
    if request.method == 'POST':
        if ugstudent:
            db.session.delete(ugstudent)
            db.session.commit()
            return redirect('/ugstudentsdata')
        abort(404)

    return render_template('delete.html')
```

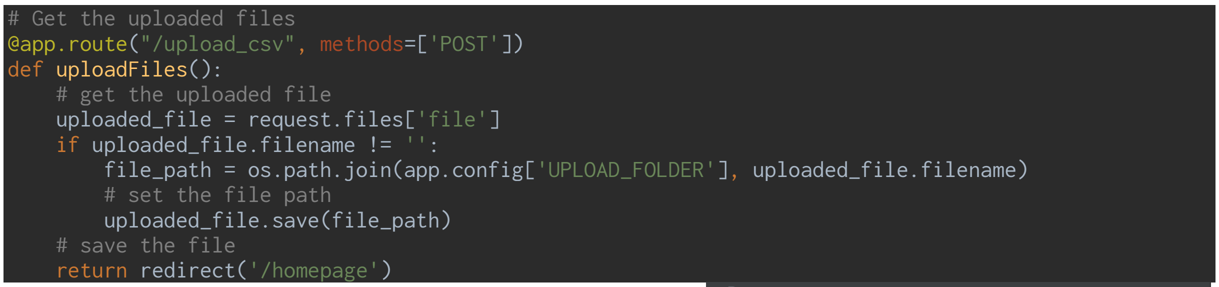
Deleting the data for student ID 1 for undergraduate:

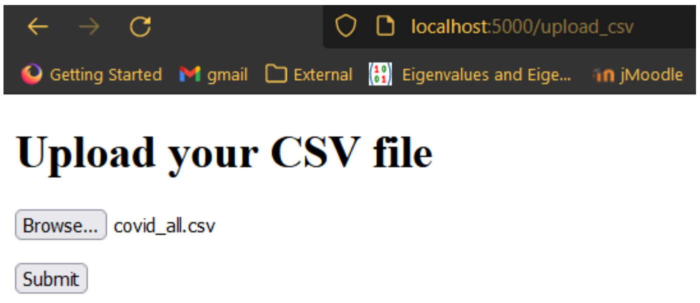

localhost:5000/ugstudentsdata/1/delete

Getting Started gmail External Eigenvalues and Eige... jMoodle Academic Calen

Click YES to confirm YES Cancel

Uploading the files (csv for this usecase) so as to store, save and read the data from the csv and then clean the data (if applicable)


```
# Get the uploaded files
@app.route("/upload_csv", methods=['POST'])
def uploadFiles():
    # get the uploaded file
    uploaded_file = request.files['file']
    if uploaded_file.filename != '':
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], uploaded_file.filename)
        # set the file path
        uploaded_file.save(file_path)
    # save the file
    return redirect('/homepage')
```

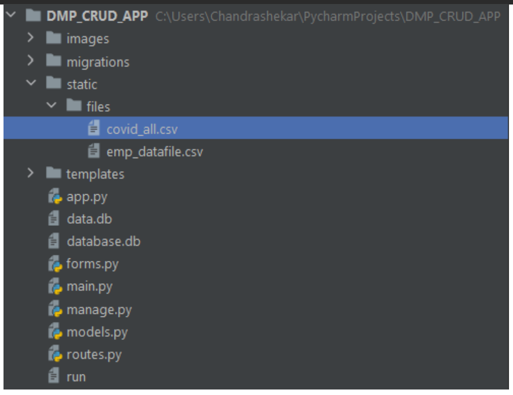

localhost:5000/upload_csv

Getting Started gmail External Eigenvalues and Eige... jMoodle

Upload your CSV file

Browse... covid_all.csv

Submit


```
DMP_CRUD_APP C:\Users\Chandrashekar\PycharmProjects\DMP_CRUD_APP
  images
  migrations
  static
    files
      covid_all.csv
      emp_datafile.csv
  templates
  app.py
  data.db
  database.db
  forms.py
  main.py
  manage.py
  models.py
  routes.py
  run
```

Data cleaning using python Pandas:

Note: This section purely focuses on employee data in the database. The similar data cleaning can be applied for all students data and faculty data but it is now limited to employee data to show the working model for cleaning data.

The records we read from the CSV or create using above defined methods may be incomplete due to missing attributes, they may have an incorrect spelling for user-entered text fields or they may have an incorrect value such as a date of birth in the future.

it's important that these data quality issues are recognised early during our exploration phase and cleansed prior to any analysis. By allowing uncleaned data through our analysis tools, we run the risk of incorrectly representing companies or users data by delivering poor quality findings based on incorrect data.

After checking for completely null columns it’s worth checking to see if there are any rows that do not contain enough usable elements. We can achieve this by making use of `.dropna(thresh=2)` to remove any rows that have less than two elements.

Pandas provide a built-in function that can achieve this .fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None). Pandas .fillna() is an incredibly powerful function when cleaning data or manipulating a DataFrame. The value parameter can accept a dictionary which will allow you to specify values that will be used on specific columns to fill null values.

way of summarising this is calling `df.isnull().sum()` which sums the number of null elements in each column.

```
def parseCSV(filePath):
    # CVS Column Names
    col_names = ['employee_id', 'name', 'age', 'position', 'high_school']

    # Use Pandas to parse the CSV file
    csvData = pd.read_csv(filePath, names=col_names, header=None)

    # values from high_school dropped in the columns of the object.
    csvData.drop(columns=['high_school'], inplace=True)

    missing_values = csvData.isnull().sum()
    # print(“missing values are: {}”.format(missing_values))
    under_threshold_removed = csvData.dropna(axis='index', thresh=2, inplace=False)
    under_threshold_rows = csvData[~csvData.index.isin(under_threshold_removed.index)]
    # print(under_threshold_rows)

    # Set a default category for missing genders.
    csvData ['gender'].cat.add_categories(new_categories=['Male'], inplace=True)
    csvData.fillna(value={'gender': 'Male'}, inplace=True)
    # print(csvData.info())

    # Loop through the Rows
    for i, row in csvData.iterrows():
        sql = "INSERT INTO table (employee_id, name, age, position) VALUES (%s, %s, %s, %s)"
        value = (row['employee_id'], row['name'], row['age'], row['position'])
        conn.execute(sql, value, if_exists='append')
        conn.commit()
        print(i, row['first_name'], row['last_name'], row['address'], row['street'], row['state'],
row['zip'])
```

Analysis and Future implementation:

From the documentation, we see that a part of Institution Management System is implemented with following scope:

- Secure features including signup/register and login are implemented, post which the following operations can be done.
- Models created for UGStudents, PGStudents, Faculty and Employee.
- CRUD operations are implemented for UGStudents, PGStudents, Faculty and Employee.
- Data cleaning is applied for the csv read and it is applied on Employee model only.
- The homepage contains the statistics for Students and Faculty including links to view the list of all entries in respective tables in the database.

URLs which is working with the current implementation, tested and verified operations by running the app on Windows System (Win11, AMD Ryzen7, 8GB RAM)

Startpage:

In case if the user is not signed in, it is redirected to http://localhost:5000/login, which redirects to http://localhost:5000/homepage, by default once the user is signed in. Includes analysis from dbquery and displays the data in the form of a chart. (Pie chart used, bar chart tested (src: canva))

Signup: http://localhost:5000/register

Logout: http://localhost:5000/logout redirects to http://localhost:5000/login?next=%2Fhomepage

Create:

- UGStudents: http://localhost:5000/ugstudentsdata/ugcreate
- PGStudents: http://localhost:5000/pgstudentsdata/pgcreate
- Faculty: http://localhost:5000/facultydata/facultycreate
- Employee: http://localhost:5000/data/create

Retrieve all data from respective tables:

- UGStudents table: http://localhost:5000/ugstudentsdata
- PGStudents table: http://localhost:5000/pgstudentsdata
- Faculty table: http://localhost:5000/facultydata
- Employee table: http://localhost:5000/data

Retrieve specific (ROW) data from respective tables:

- UGStudents: http://localhost:5000/ugstudentsdata/<id>
- PGStudents: http://localhost:5000/pgstudentsdata/<id>
- Faculty: http://localhost:5000/facultydata/<id>
- Employee: http://localhost:5000/data/<id>

Delete:

- UGStudents: http://localhost:5000/ugstudentsdata/<id>/delete
- PGStudents: http://localhost:5000/pgstudentsdata/<id>/delete
- Faculty: http://localhost:5000/facultydata/<id>/delete
- Employee: http://localhost:5000/data/<id>/delete

Update:

- UGStudents: http://localhost:5000/ugstudentsdata/<id>/update
- PGStudents: http://localhost:5000/pgstudentsdata/<id>/update
- Faculty: http://localhost:5000/facultydata/<id>/update
- Employee: http://localhost:5000/data/<id>/update

Upload CSV:

- http://localhost:5000/upload_csv: The data files is accepted and stored under static/files for pasring, cleaning data

Future Implementations:

Using the Institutional Management system so far does the job of storing, cleaning the data and managing the data with login features. The future Implementations shall include:

- Develop more in-depth analysis on homepage (including personalised student data in each section)
- Including subjects and research field as a field linking faculty and students. Using the subject data, we will be able to manage to get Automated Time Table Generator using python.
- Dockerize the entire application so that there will be no dependencies on the platform the application runs on.
- Implementing the same on cloud-based server.
- Backing up the database periodically using python cron job once it is implemented in cloud.
- Automate cleaning the data for all the tables to improve the efficiency.

Conclusion:

The intended project Institutional Management System is implemented and runs as expected, based on the before-said documentation. Implementing these said features has resulted in managed Data Management System as part of learning from python and relational DBMSs alongside making use of libraries like Pandas for making most of the operations easier. Implementing the features as mentioned in the section “Future implementations” shall result in a very well management system.