IITH-Forms

Saurabh Vankalas and GVV Sharma

I. Introduction

Its a software where we can display data onto a pdf forms of IITH where data is entered through html forms.

II. Installation

Step 1: sudo apt-get install python3.6
Step 2: sudo apt install python3-pip
Step 3: pip install -r requirements.txt
Step 4: sudo apt install git-all
Step 5: git clone
https://github.com/SaurabhVankalas/iithforms.git

III. Running the project on flask local server

A. *Type the below commands on console window*

1. cd iithforms
2. python3 createdb.py (sets all the database tables)
3. python3 app.py (starts the local flask server)
4. Type the url 0.0.0.0:8080 in your browser

IV. Flow

We are using SQLAlchemy as our database. We will be using the following libraries within our project:
1. SQLAlchemy (via Flask-SQLAlchemy)
2. WTForms (via Flask-WTF)
3. Flask
4. XlsxWriter (Managing xlsx files)
5. wtforms-html5 (Forms)
6. inflect (conversion of nomber to words)
7. flask-login (Login manager)
8. gunicorn (Creating server)
9. pyPDF2 (Reading PDFs)
10.Nginx (Creatong server)
11. reportlab (Creating and merging PDF)

A. *Structuring The Application Directory*

−−iithforms (folder)
  −−app.py
  −−createdb.py
  −−requirements.txt
  −−static (folder)
  −−flaskabc (folder)
    −−__init__.py
    −−models.py
    −−routes.py
    −−forms.py
    −−templates (folder)
    −−static (folder)

B. *Uses of all files*

1. __init__.py : Declares the flask app(variable name:app) and SQLAlchemy database(variable name:db).
2. models.py : Contains all the classes which will be converted into database tables.
3. forms.py : With the help of WTForms library its very easy to manage the data subitted by the browser, and the process is easy to manage.
4. routes.py : Here, all the objects from forms.py and models.py are imported here, and the actual functions are written here. Its the processing unit.
5. templates : All the templates are stored in this folder, which are required for the project.
6. createdb.py : Creates database and tables.
7. app.py : Runs the project on a flask local server.

C. *Understanding Code*

We will understand the code for Contingent form. The similar code is used for other forms.

1) __init__.py:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
app = Flask(__name__)
app.config['SECRET_KEY'] = 'aaf'
app.config['SQLALCHEMY_DATABASE_URI']
= 'sqlite:///site.db'
```

```
db = SQLAlchemy(app)
from flaskabc import routes
```

Here,our Flask app is set, and SQLAlchemy database is set up to use.

*2) **models.py**:*

```
class Contingent(db.Model):
    __tablename__ = 'contingent'
    id = db.Column(db.Integer,
primary_key=True)
    curr_date = db.Column(db.String(100),
nullable=False)
    station = db.Column(db.String(100),
nullable=False)
    name = db.Column(db.String(100),
nullable=False)
    address = db.Column(db.String(100),
nullable=False)
    bankbranch = db.Column(db.String(100),
nullable=False)
    acc_num = db.Column(db.String(100),
nullable=False)
    ifsc = db.Column(db.String(100),
nullable=False)
    pets = db.relationship('Contingent_a')
    def __repr__(self):
        return '¡Contigent
```

```
class Contingent_a(db.Model):
    id = db.Column(db.Integer,
primary_key=True)
    dt1 = db.Column(db.String(100),
nullable=False)
    des1 = db.Column(db.String(100),
nullable=False)
    amt1 = db.Column(db.Float, nullable=False)
    contingent_id =
db.Column(db.Integer,db.ForeignKey('contingent.id'))
```

For the common case of having one Flask application all we have to do is to create our Flask application, load the configuration of choice and then create the SQLAlchemy object by passing it the application.

Once created, that object then contains all the functions and helpers from both sqlalchemy and sqlalchemy.orm. Furthermore it provides a class called Model that is a declarative base which can be used to declare models:

The db.Column creates a column in the sqlalchemy database, and db.model creates a table with name contingent. We have two classes here, i.e. contingent and contingent_a. The contingent is the parent class and contingent_a is the child class. The contingent data, can contain several contingent_a data, therefore its a one to many relationship between two database tables.

We achieve this my declaring a column in the contingent_a table which will store its parent's id as a foreign key.

*3) **forms.py**:*

```
class ContingentForm(FlaskForm):
    curr_date =
DateField('Date',format='%Y%m%d')
    station = StringField('Station',
validators=[DataRequired()])
    name = StringField('Name',
validators=[DataRequired()])
    address = StringField('Address',
validators=[DataRequired()])
    bankbranch = StringField('Bank Name &
Branch', validators=[DataRequired()])
    acc_num = StringField('Bank Account
Number', validators=[DataRequired()])
    ifsc = StringField('IFSC Code',
validators=[DataRequired()])
    submit = SubmitField('Submit')
```

```
class Contingent_aForm(FlaskForm):
    dt1 = DateField('Date 1',format='%Y%m%d',
validators=[DataRequired()])
    des1 = StringField('Destination 1',
validators=[DataRequired()])
    amt1 = FloatField('Amount 1',
validators=[DataRequired()])
    submit = SubmitField('Submit')
```

*4) **contingentform.html (Template)**:* This html template consists of the form that browser submits the data to the database. WTForms library makes it easier to access and manage the forms.

Similar template form is made for contingent_a form. The sole purpose of these templates is to accept data from the browser and submit it to the database. After pressing the submit button the corresponding function in routes.py file is executed, and the code checks for errors. If no error, data is stored as an entry into the database table.

*5) **contingent.html (Template)**:* This html template consists of the form that browser submits the data to the database. WTForms library makes it easier to access and manage the forms.

*6) **routes.py**:* In this file, the functions are executed. Functions such as download the form, create

a new form, update and delete the forms are executed here.

First, we import all the forms and models. We check whether the user is trying the get or post data into the html. If the request method is GET, then the form is passed on to the template and the template is viewed.

If the user is trying to post information into database, we check whether its a post request, if it is, then we take the form data, and create a object for the corresponding class. We store that data into the object. Now, the object is saved in the database using the command db.session.add(data), db.session.commit(). Assuming data is our object. For delete, we use db.session.delete(data) followed by db.session.commit().

For the download function, we use reportlab library. We initialise a stream of string data and then create a pdf where the object data has to be displayed. We can specify the coordinates for each object data. Then we merge the blank contingent form with this new pdf and we have our object data displayed with in the contingent form. We save this new merged pdf and send it to the browser if user wishes to download it.

7) ***createdb.py****:* Here, we import db from our module and the command db.create_all() will create the database and the tables and corresponding coloumns.

8) ***app.py****:* app.run(host=0.0.0.0,port=8080) will run the project and start the server with the corresponding host and port number.

9) ***Deploy on Nginx****:*

Install Nginx on the machine:
sudo apt-get install nginx
Run the app locally through flask and then open another window of terminal. We need to make configuration file so as to make nginx listen to our flask app.
sudo nano /etc/nginx/conf.d/virtual.conf
Paste the following code onto the configuration file.

```
server {
    listen 80;
    server_name your_ip_address_here;
    location / {
        proxy_pass http://0.0.0.0:8080;
    }
}
```

Now, save the file as virtual.conf and execute the below two line on the terminal:
sudo nginx -t
sudo service nginx restart
Now, we can access the web app through the ip address and port number is not required to mention.