1. Create project folder
2. Pip3 install virtualenv (inside project folder)
3. Virtualenv env
4. Env/scripts/activate (enables virtualenv)
5. pip3 install flask flask-sqlalchemy
6. create new file app.py in in main project folder
7. fill with code:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return "Hello, World!"

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

1. Create two new folders inside main dir folder (naming based on purposed used for) – static, templates
2. Inside templates create new file called index.html
3. Press ! to generate boilerplate
4. Create base.html in templates folder with code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

{% block head %}{% endblock %}

</head>

<body>

{% block body %}{% endblock %}</body></html>

1. Fill index.html with code:

{% extends 'base.html' %}

{% block head %}

<h1>Template</h1>

{% endblock %}

{% block body %}

{% endblock %}

1. Create new folder inside static called css
2. Create a new file inside css fodler called main.css

body {

margin: 0;

font-family: sans-serif

}

1. Update base.html file with new code (inside head tag):

<meta http-equiv="X-UA-Compatible" content="ie-edge">

<link rel="stylesheet href=" href="{{ url\_for('static', filename='css/main.css')}}">

{% block head %}{% endblock %}

</head>

<body>

{% block body %}{% endblock %}

</body>

</html>

(linking the directory of css stylesheet file so flask will now where to look for it)

1. Add new line in app.py - From flask\_sqlalchemy import SQLAlchemy
2. Add new module url\_for in line from flask import Flask
3. Add this code to app.py (Creates a todo Model):

app.config['SQLALCHEMY\_DATABASE\_URI'] 'sqlite:///C:/Users/IT/FLASKINTRODUCTION/test.db'

db = SQLALCHEMY(app)

class todo(db.model):

id = db.Column(db.Integer, primary\_key=True)

content = db.Column(db.String(200), nullable=False)

completed = db.Column(db.Integer, default=0)

date\_created = db.Column(db.DateTime, default=datetime.utcnow)

1. Under this code, add new function:

def \_\_repr\_\_(self):

return '<Task %r> ' % self.id

**Which tells to return the task in a string. %r means task id**

1. Type env/scripts/activate
2. Type python to launch interactive py shell
3. From app import db
4. Then use this command:

: >>> from app import app, db >>> app.app\_context().push() >>> db.create\_all()

From App in this context is the app.py file

1. Database called test.db should be created
2. Add more code to index.html file.

{% extends 'base.html' %}

{% block head %}

{% endblock %}

{% block body %}

<div class="content">

<h1>Task Master</h1>

<table>

<tr>

<th>Task</th>

<th>Added</th>

<th>Actions</th>

</tr>

<tr>

<td></td>

<td></td>

<td>

<a href="">Delete</a>

<br>

<a href="">Update</a>

</td>

</tr>

</table>

</div>

{% endblock %}

1. Then run command to activate server: flask run
2. In app.py @app.route decorator add POST and GET method. End result should look like this:

@app.route("/", methods=['POST', 'GET'])

def index():

if request.method == 'POST':

task\_content = request.form['content']

else:

return render\_template('index.html')

1. Update index.html file now with functionality that is related to new methods addded:

<form action="/" method="POST">

<input type="text" name="content" id="content">

<input type="submit" value="Add task">

</form>

1. Add new module request in from flask import Flask line.
2. Now create a Todo object with this line of code:

new\_task = Todo(content=task\_content)

1. Add this new line of code which will push(add) the object to the database:

db.session.add(new\_task)

1. Commit it to database:

db.session.commit()

1. And then redirect back to index:

return redirect('/')

1. Import redirect module in from flask import Flask line
2. Add new line under else statement:

tasks = Todo.query.order\_by(Todo.date\_created).all()

Which orders ***all*** tasks that were created in the database and orders them by the date\_created

1. Now update index.html file with jinja2 syntax which references task\_content and date\_created models in app.py:

        <th>Actions</th>

  </tr>

  {% for task in tasks %}

  <tr>

      <td>{{ task.content }}</td>

      <td>{{ task.date\_created.date() }}

</td>

        <td>

            <a href="">Delete</a>

              <br>

              <a href="">Update</a>

          </td>

    </tr>

    {% endfor %}

1. Now to add functionality to the Delete button add this line:

@app.route('/delete/<int:id>')

**<int:id> finds the task id. Id references id variable in Todo model, which finds the task id by the primary key assigned to it in sql database.**

1. Then create a function with the id variable passed in as the argument:

def delete(id):

task\_to\_delete = Todo.query.get\_or\_404(id)

**This will try to get the task by id and if it fails it will throw a 404 error.**

1. Same logic goes through here as in steps 30-33 for pushing the task to a database:

  try:

        db.session.delete(task\_to\_delete)

      db.session.commit()

        return redirect('/')

    except:

        return "There was a problem deleting that task"

1. To complete functionality, add new line of code which references the task id in app.py to index.html:

<a href="/delete/{{task.id}}">Delete</a>

1. Now add new functionality to app. py after previous code, for the update button:

@app.route('/update/<int:id>', methods=['GET', 'POST'])

def update(id):

    return ''

1. To complete functionality, add new line of code which references the task id inapp.py to index.html:

<a href="/update/{{task.id}}">Update</a>

1. Create update.html file in templates folder
2. Copy and paste all code from index.html
3. Remove all table, tr, th code from it. End result shoud look like this:

{% extends 'base.html' %}

{% block head %}

<title> Task Master</title>

{% endblock %}

{% block body %}

<div class="content">

    <h1 style="text-align: center">Update Task</h1>

    <div class="form">

        <form action="/update/{{task.id}}"method="POST">

          <input type="text" name="content"id="content">

            <input type="submit" value="Add task">

        </form>

    </div>

</div>

{% endblock %}

1. Now update the update function again with task and now with update template:

task = Todo.query.get\_or\_404(id)

    if request.method == 'POST':

      pass

    else:

        return render\_template('update.html', task=task)

1. Update the update.html file referencing task.content object from app.py. End result should look like this, with value added at the end:

<input type="text" name="content" id="content" value="{{task.content}}">

1. Update 2nd input type value to update:

<input type="submit" value="Update">

1. Update app.py file with functionality that references task.content:

        task.content = request.form['content']

        try:

            db.session.commit()

            return redirect('/')

        except:

            return 'There was an issue updating your task'

    else:

        return render\_template('update.html', task=task)

1. In Index.html add new lines of code under h1:

    {% if tasks|length < 1 %}

  <h4 style="text-align: center"> There are no tasks. Create one below!</h4>

    {% else %}

1. Under closing table body tag, additional jinja2 code block needs to be added:

          </td>

      </tr>

      {% endfor %}

    </table>

    {% endif %}

1. **(FOR WINDOWS) - TO RUN THE PROJECT, SELECT CODE IN GITHUB AND GIT CLONE THE REPOSITORY IN TERMINAL. RUN COMMAND.**
2. **THEN CREATE A VIRTUAL ENVIRONMENT WITH virtualenv env**
3. **ENABLE VIRTUAL ENVIRONMENT WITH call env/scripts/activate**
4. **VIRTUAL ENV SHOULD BE ENABLED.**
5. **THEN RUN pip install flask flask\_sqlalchemy**
6. **NOW ENTER FLASK RUN AND THIS SHOULD LAUNCH THE SERVER WITH THE APP.**