7th March 2020

Python and Flask Bootcamp: Create a website using Flask. Udemy Jose Portilla

|  |  |
| --- | --- |
| [Flask logo.svg](https://en.wikipedia.org/wiki/File:Flask_logo.svg) | |
| [**Developer(s)**](https://en.wikipedia.org/wiki/Software_developer) | [Armin Ronacher](https://en.wikipedia.org/wiki/Armin_Ronacher) |
| **Initial release** | April 1, 2010; 9 years ago |
| [**Stable release**](https://en.wikipedia.org/wiki/Software_release_life_cycle) | 1.1.1 / July 8, 2019; 7 months ago[[1]](https://en.wikipedia.org/wiki/Flask_(web_framework)#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Flask_(web_framework)#cite_note-2) |
| [**Repository**](https://en.wikipedia.org/wiki/Repository_(version_control)) | [github.com/pallets/flask](https://github.com/pallets/flask) |
| **Written in** | [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) |
| [**Type**](https://en.wikipedia.org/wiki/Software_categories#Categorization_approaches) | [Web framework](https://en.wikipedia.org/wiki/Web_framework) |
| [**License**](https://en.wikipedia.org/wiki/Software_license) | [BSD](https://en.wikipedia.org/wiki/BSD_licenses) |
| **Website** | [palletsprojects.com/p/flask/](https://palletsprojects.com/p/flask/) |

‘myflaskenviro’

Front end design: html, css, bootstrap 4

Templates for displaying webpages in Flask, Forms for user input, databases for storing user input.

Get user input, store it, report it back.

Blueprints for larger applications.

OAuth (outside authorization) like login thru google, facebook, twitter

‘conda’ is the package manager for anaconda. Atom.io

Create virtual environment: conda create –n <enviornName> python=3.7

Activate the virtual enviro: conda activate <enviro name> // deactivate

Download the libraries from file(to the virtual environment): pip install –r requirements.txt

Bootstrap will provide automatic styling and components through CSS and JS

Flask is the web framework that allows to connect python code to web

Flask can do everything that Django can with the use of tonnes of 3rd party libs

Workflow: user > front-end>flask>database.CRUD create, read, update, delete

Forms use WTfoms library. Jinja2 grabs information from python and flask and sends as html

Environment name: myflaskenviro

Html: strong for bold and ‘em’ for emphasis

https://developer.mozilla.org/en-US/ for html tag learning.

Self closing tags in html. <img src=”” alt=””> // label ‘for=’ tag is for ‘id’ of a ‘form input’. // placeholder // dropdown - <select name=”fruits”> <option value=> banana </option> option .. </select> // <textarea rows= cols = >

Stylesheet: define property-value pair for elements. In Html: link to stylesheet using: <link rel=”stylesheet” href=”/file.css”>

CSS file:

h1 {

color: blue;

}

p {

color: red

}

label {

color: rgba(255, 0, 0, 0.5);

}

title {

color: #eab06c

}

HTML file:

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<link rel="stylesheet" href="style.css" >

<title></title>

</head>

<body>

Hi this is my fist comment!

<h1> Hello, my comment </h1>

<form>

<label for = 'email\_id'> EMAIL </label>

<input type="text" width="20" >

<input type='email' id='email\_id'>

</form>

</body>

</html>

<div>Border : color, width, style

For linking javascript, <script src=”<file path> “

Every html element can accept a ‘class’ or ‘id’ attribute. It can accessed by ‘.’ And ‘#’ operator.

Font-family: cursive, monospace, font-size, font-weight, font-style, text-align,

Bootstrap has a very large list of easy to use components. [www.getbootstrap.com](http://www.getbootstrap.com)

Bootstrap has components and classes. ‘select multiple’ allows to select multiple options from dropdown list.

Bootstrap:

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<!-- <link rel="stylesheet" href="style.css" > --> #comment

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

<title></title>

</head>

<body>

Hi this is my fist comment!

<div class="myClass">

<h1> Hello, my comrades </h1>

<h2 id="myId"> Dear bro's and siters </h2>

</div>

<form>

<label for = 'email\_id'> EMAIL </label>

<input type="text" width="20" >

<input type='email' id='email\_id'>

</form>

<p> Hello dear <span> friends! </span>

<div class="">

<h2> Hi all, come here for a trip! </h1>

<h1 id=""> Hi my brothers and sisters </h1>

</div>

<ul class="nav flex-column">

<li class="nav-item">

<a class="nav-link active" href="#">Active</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Link</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Link</a>

</li>

<li class="nav-item">

<a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>

</li>

</ul>

</body>

</html>

System design technologies: waterfall and agile development

Software Engineering | Classical Waterfall Model

Classical waterfall model is the basic **software development life cycle** model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.  
Classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure:  


Let us now learn about each of these phases in brief details:

1. **Feasibility Study**: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software.  
   The feasibility study involves understanding the problem and then determine the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.
2. **Requirements analysis and specification**: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

* + **Requirement gathering and analysis:** Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (inconsistent requirement is one in which some part of the requirement contradicts with some other part).
  + **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

1. **Design**: The aim of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.
2. **Coding and Unit testing**: In coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.
3. **Integration and System testing**: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.

System testing consists three different kinds of testing activities as described below :

* + **Alpha testing:** Alpha testing is the system testing performed by the development team.
  + **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
  + **Acceptance testing:** After the software has been delivered, the customer performed the acceptance testing to determine whether to accept the delivered software or to reject it.

1. **Maintainence:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is the 60% of the total effort spent to develop a full software. There are basically three types of maintenance :
   * **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
   * **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer’s request.
   * **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as work on a new computer platform or with a new operating system.

**Advantages of Classical Waterfall Model**

Classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered as the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model:

* This model is very simple and is easy to understand.
* Phases in this model are processed one at a time.
* Each stage in the model is clearly defined.
* This model has very clear and well undestood milestones.
* Process, actions and results are very well documented.
* Reinforces good habits: define-before- design,  
  design-before-code.
* This model works well for smaller projects and projects where requirements are well  
  understood.

**Drawbacks of Classical Waterfall Model**

Classical waterfall model suffers from various shortcomings, basically we can’t use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

* **No feedback path:** In classical waterfall model evolution of a software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phases. Therefore, it does not incorporate any mechanism for error correction.
* **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers’ requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
* **No overlapping of phases:** This model recommends that new phase can start only after the completion of the previous phase. But in real projects, this can’t be maintained. To increase the efficiency and reduce the cost, phases may overlap.

Agile: <https://www.guru99.com/agile-scrum-extreme-testing.html>

Python data types:

Strings are ordered sequence of characters.

Lists are ordered sequence of objects

Tuples are ordered sequence of immutable objects

Sets are unordrered collection of unique objects

Dictionaries are collection of key:value pairs

Integers, floats, Booleans

--

What is Full Stack?

Full Stack Web Developer

A full stack web developer is a person who can develop both client and server software.

In addition to mastering HTML and CSS, he/she also knows how to:

• Program a browser (like using JavaScript, jQuery, Angular, or Vue)

• Program a server (like using PHP, ASP, Python, or Node)

• Program a database (like using SQL, SQLite, or MongoDB)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Client Software

(Front End)

• HTML

• CSS

• Bootstrap

• W3.CSS

• JavaScript

• ES5

• HTML DOM

• JSON

• XML

• jQuery

• Angular

• React

• Backbone.js

• Express.js

• Ember.js

• Redux

• Storybook

• GraphQL

• Meteor.js

• Grunt

• Gulp

Server Software

(Back End)

• PHP

• ASP

• C++

• C#

• Java

• Python

• Node.js

• Ruby

• REST

• GO

• SQL

• MongoDB

• Firebase.com

• Sass

• Less

• Parse.com

• PaaS (Azure and Heroku)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Popular Stacks

• LAMP stack: JavaScript - Linux - Apache - MySQL - PHP

• LEMP stack: JavaScript - Linux - Nginx - MySQL - PHP

• MEAN stack: JavaScript - MongoDB - Express - AngularJS - Node.js

• Django stack: JavaScript - Python - Django - MySQL

• Ruby on Rails: JavaScript - Ruby - SQLite - Rails

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Advantages

The advantage of being a full stack web developer is:

• You can master all the techniques involved in a development project

• You can make a prototype very rapidly

• You can provide help to all the team members

• You can reduce the cost of the project

• You can reduce the time used for team communication

• You can switch between front and back end development based on requirements

• You can better understand all aspects of new and upcoming technologies

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Disadvantages

• The solution chosen can be wrong for the project

• The solution chosen can be dependent on developer skills

• The solution can generate a key person risk

• Being a full stack developer is increasingly complex

--

Armin Ronacher, "Flask for Fun and Profit", PyBay2016 // https://www.youtube.com/watch?v=1ByQhAM5c1I

--

Flask is having very fast iteration. Latency in high performance async IO.

from flask import Flask

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

@app.route('/')

def index():

return '<h1> Hello Puppies </h1>'

@app.route("/puppy/<name>")

def puppy(name):

return f'<h1> Hellow {name.upper()} puppy! '

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

--

Debugger pin is required to access the debugger console.

Make debug = to False only in the production environment.

‘Templates’ is a directory where Flask will automatically look for ‘html’ pages.

Flask ‘render\_template’ function will return an html page in ‘templates’ folder from main page view function. Jnja template engine.

Unsplash.com is a site for photos

Python logic can be implemented into variable, which can be injected into html using jinja to construct {{}} using Flask render\_template function.

Control flow statements use {%%} for loop and if conditions are used.

url\_for(): to connect to pages in templates. <a href= {{url\_for('puppy',name='Roku')}} > Puppies home </a>

--

@app.errorhandler(404)

def page\_not\_found(e):

return render\_template('error.html'), 404

--

Request.args.get(‘<name>’)

Flask\_wtf library; wtfroms

from flask import Flask, render\_template

from flask\_wtf import FlaskForm

from wtforms import StringField, SubmitField

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

app.config['SECRET\_KEY'] = 'mysecret\*key'

class MyForm(FlaskForm):

breed = StringField("What's the breed?")

submit = SubmitField('submit')

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

def index():

breed = False

form = MyForm()

if form.validate\_on\_submit():

breed = form.breed.data

form.breed.data=''

return render\_template('index.html', breed=breed, form=form)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

--

**Index.html**

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

</head>

<body>

{% if breed %}

<p> You selected the breeed {{breed}}

{% else %}

Select your breed </p>

{% endif %}

<form method='POST'>

{{form.hidden\_tag()}}

{{form.breed.label}} {{form.breed()}}

{{form.submit()}}

</form>

</body>

</html>

--

Flask’s ‘session’ object to grab the information provided in the form and pass it to another template.

BooleanField is for checkbox. SelectField is for dropdown menu. Choices are a list of tuple items as value and label

--

from flask import Flask, render\_template, session, redirect, url\_for

from flask\_wtf import FlaskForm

from wtforms import StringField, SubmitField, BooleanField, RadioField, TextAreaField, SelectField

from wtforms.validators import DataRequired

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

app.config['SECRET\_KEY'] = 'mysecretfuck\*key'

class MyForm(FlaskForm):

breed = StringField('Whats the breed?', validators=[DataRequired()])

neutered = BooleanField('Neutered?')

mood = RadioField('Whats your mood?', choices=[('Happy', 'Happy'), ('Excited', 'Excited')])

food = SelectField('Which food?', choices=[('chi', 'Chicken'), ('beef', 'Beef'), ('fish', 'Fish')])

feedback = TextAreaField('Any suggestion?')

submit = SubmitField('submit')

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

def index():

form = MyForm()

if form.validate\_on\_submit():

session['breed'] = form.breed.data

session['neutered'] = form.neutered.data

session['mood'] = form.mood.data

session['food'] = form.food.data

session['feedback'] = form.feedback.data

return redirect(url\_for('thankyou'))

return render\_template('nxt\_index.html', form=form)

@app.route('/thankyou')

def thankyou():

return render\_template('thankyou.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

--------

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

</head>

<body>

<h1> You have to fill :)' </h1>

<form method='POST'>

{{form.hidden\_tag()}}

{{form.breed.label}} {{form.breed()}}<br>

{{form.neutered.label}} {{form.neutered()}}<br>

{{form.mood.label}} {{form.mood()}}<br>

{{form.food.label}} {{form.food()}}<br>

{{form.feedback.label}} {{form.feedback()}}<br>

{{form.submit()}}

</form>

</body>

</html>

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

</head>

<body>

<p> You have chosen: </p>

<p> Breed {{session['breed']}} <br>

Neutered {{session['neutered']}} <br>

Mood {{session['mood']}} <br>

Food {{session['food']}} <br>

Suggestion {{session['feedback']}} <br>

</p>

</body>

</html>

Flash() // get\_flashed\_messages()

--

from flask import Flask, render\_template, session, flash, redirect, url\_for

from flask\_wtf import FlaskForm

from wtforms import SubmitField

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

app.config['SECRET\_KEY']= 'appkey'

class Alert\_Flash(FlaskForm):

sub = SubmitField('Submit')

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

def index():

form = Alert\_Flash()

if form.validate\_on\_submit():

flash('You just clicked the button')

return redirect(url\_for('index'))

return render\_template('alerts.html', form=form)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

alerts.html

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

</head>

<body>

{% for mess in get\_flashed\_messages() %}

<div class="alert alert-warning alert-dismissible fade show" role="alert">

<strong>Holy guacamole!</strong> You should check in on some of those fields below.

<button type="button" class="close" data-dismiss="alert" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

{{mess}}

</div>

{% endfor %}

<form method="POST">

{{form.hidden\_tag()}}

{{form.sub()}}

</form>

</body>

</html>

--

ORM- Object relational mapper converts python code into sql commands. The most common Python ORM is SQLAlchemy.

Flask-SQLAlchemy // pip install Flask-SQLAlchemy

1. Setup SQLite in Flask. B) Create a model in Flask. C) Perform basic CRUD in the model

Inherit from db.Model // from flask\_sqlalchemcy import SQLAlchemy

App.config[‘SQLALCHEMY\_DATABASE\_URI’] = ‘sqlite:///’+os.path.join(basedir,’data.sqlite’)

App.config[‘SQLALCHEMY\_TRACK\_MODIFICATIONS’] = False

Db = SQLAlchemy(app) // db.create\_all() –creates database table from the model class // db.session.add\_all([<instance objects>]) // db.session.commit()

Basic.py

import os

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

#create app

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

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///'+os.path.join(basedir,'data.sqlite')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

#create database

db = SQLAlchemy(app)

#migrate the app with the database

Migrate(app,db)

#create model

class Puppy(db.Model):

\_\_tablename\_\_ = 'puppies'

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

name = db.Column(db.Text)

age = db.Column(db.Integer)

breed = db.Column(db.Text)

def \_\_init\_\_(self,name,age,breed):

self.name = name

self.age = age

self.breed = breed

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

return f"Puppy name is {self.name} and is {self.age} year old"setupdatabase.py

from basic import db, Puppy

#converts model class in to database table

db.create\_all()

sam = Puppy()

roku = Puppy()

sam.name = 'Sami'

sam.age = 3

roku.name = 'Roku'

roku.age = 2

#create

janko = Puppy(name='Edan',age=5)

db.session.add\_all([sam,roku])

db.session.commit()

print(sam.id)

print(roku.id)

crud.py

from basic import db, Puppy

#create

dinka = Puppy(name='Dinkan', age=10)

lut = Puppy('Luttappi',7)

db.session.add\_all([dinka,lut])

db.session.commit()

all\_puppies = Puppy.query.all()

print(all\_puppies)

puppy\_one = Puppy.query.get(3)

print(puppy\_one.name, puppy\_one.age)

#filter

puppy\_unni = Puppy.query.filter\_by(name='Unni')

print(puppy\_unni.all())

#update table

puppy\_add = Puppy.query.get(4)

puppy\_add.age = 8

db.session.add(puppy\_add)

print(Puppy.query.get(3).age)

pp = Puppy.query.get(3)

#delete item from table

db.session.delete(pp)

db.session.commit()

--

Flask-Migrate – to update changes to the database model

From flask-migrate import Migrate

Set FLASK\_APP = myapp.py

flast db init – sets the migration directory

flask db migrate –m ‘some message’ – sets the migration file

flask db upgrade – sets the database migration

(its like git updates)

Flask relationships

A primary key is a column, whose value is unique for each row.

A foreign key in a table is a primary key in another table.

One to many and one to one relationship (uselist=false)

Models.py

import os

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

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

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///'+os.path.join(basedir,'data.sqlite')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db = SQLAlchemy(app)

Migrate(app,db)

class Puppy(db.Model):

\_\_tablename\_\_ = 'puppies'

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

name = db.Column(db.Text)

#set 1 to many relationship (puppy can have many toys)

toys = db.relationship('Toy', backref='puppy', lazy='dynamic')

#set 1 to 1 relationship(puppy can have only one Owner)

owner = db.relationship('Owner', backref='puppy', uselist=False)

def \_\_init\_\_(self, name):

self.name = name

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

if self.owner:

return f"Puppy name is: {self.name}, owner name is: {self.owner.name}"

return f"Puppy {self.name} has no owner yet!"

def report\_toys(self):

print(f"Here is {self.name}'s toys")

a = 0

for toy in self.toys:

a += 1

print(a,':', toy.item\_name)

return 'finished'

class Toy(db.Model):

\_\_tablename\_\_ = 'toys'

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

item\_name = db.Column(db.Text)

puppy\_id = db.Column(db.Integer, db.ForeignKey('puppies.id'))

def \_\_init\_\_(self, item\_name, puppy\_id):

self.item\_name = item\_name

self.puppy\_id = puppy\_id

class Owner(db.Model):

\_\_tablename\_\_ = "owner"

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

name = db.Column(db.Text)

puppy\_id = db.Column(db.Integer, db.ForeignKey('puppies.id'))

def \_\_init\_\_(self, name, puppy\_id):

self.name = name

self.puppy\_id = puppy\_idbasic.py

from models import Puppy, Owner, Toy, db

db.create\_all()

roko = Puppy('Roko')

jno = Puppy('Joono')

db.session.add\_all([roko,jno])

db.session.commit()

ty1 = Toy('Doremon',roko.id) #puppy id of roko

ty2 = Toy('Shinjan',jno.id)

ty3 = Toy('Madumitha',roko.id) # puppy id of roko (1 to many relationship)

own1 = Owner('Jose',roko.id)

own2 = Owner('Maria',jno.id)

db.session.add\_all([ty1,ty2,ty3,own1,own2])

db.session.commit()

all = Puppy.query.all();

print(all)

print(roko.report\_toys())

output: [Puppy name is: Roko, owner name is: Jose, Puppy name is: Joono, owner name is: Maria]

Here is Roko's toys

1 : Doremon

2 : Madumitha

Finished

--

Adoption-site.py

# import flask modules

# create key for the forms

# configure the database

# create the app

# create the model

# create views and add forms

#=========================

import os

from flask import Flask, render\_template, redirect, url\_for

from flask\_migrate import Migrate

from flask\_sqlalchemy import SQLAlchemy

from forms import AddForm, DelForm

#=========================

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

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

app.config['SECRET\_KEY']='holahoola'

app.config['SQLALCHEMY\_DATABASE\_URI'] = "sqlite:///"+os.path.join(basedir,'data.sqlite')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db = SQLAlchemy(app)

Migrate(app,db)

#=========================

class Puppy(db.Model):

\_\_tablename\_\_ = 'puppies'

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

name = db.Column(db.Text)

def \_\_init\_\_(self, name):

self.name = name

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

return f"Puppy name is: {self.name}"

#=======================

@app.route('/')

def index():

return render\_template('home.html')

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

def add\_pup():

form = AddForm()

if form.validate\_on\_submit():

name = form.name.data

new\_pup = Puppy(name)

db.session.add(new\_pup)

db.session.commit()

return redirect(url\_for('list\_pup'))

return render\_template('add.html', form=form)

@app.route('/list')

def list\_pup():

puppies = Puppy.query.all()

return render\_template('list.html',puppies=puppies)

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

def delete\_pup():

form = DelForm()

if form.validate\_on\_submit():

id = form.id.data

pup = Puppy.query.get(id)

db.session.delete(pup)

db.session.commit()

return redirect(url\_for('list\_pup'))

return render\_template('delete.html', form = form)

#======================

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

forms.py

from flask\_wtf import FlaskForm

from wtforms import StringField, IntegerField, SubmitField

class AddForm(FlaskForm):

name = StringField("Name of Puppy")

submit = SubmitField("Add Puppy")

class DelForm(FlaskForm):

id = IntegerField("Puppy id to remove")

submit = SubmitField("Remove puppy")

---

Base.html

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

</head>

<body>

<nav class="navbar navbar-light bg-light">

<div class="container-sm">

<a class="navbar-brand" href="{{url\_for('index')}}"> Home </a>

<a class="navbar-brand" href="{{url\_for('add\_pup')}}"> Add a Puppy </a>

<a class="navbar-brand" href="{{url\_for('delete\_pup')}}"> Remove a Puppy </a>

<a class="navbar-brand" href="{{url\_for('list\_pup')}}"> List all Puppy </a>

</div>

</nav>

{% block content %}

{% endblock %}

</body>

</html>

Home.html

{% extends "base.html" %}

{% block content %}

<h1> Welcome! </h1>

{% endblock %}

--

Add.html

{% extends "base.html" %}

{% block content %}

<div class="jumbotron">

<h1> Welcome! Add a Puppy </h1>

<form method='POST'>

{{form.hidden\_tag()}}

{{form.name.label}}{{form.name()}}

{{form.submit()}}

</div>

{% endblock %}

--

Delete.html

{% extends "base.html" %}

{% block content %}

<div class="jumbotron">

<h1> Adopted a puppy? </h1>

<form method='POST'>

{{form.hidden\_tag()}}

{{form.id.label}}{{form.id()}}

{{form.submit()}}

</div>

{% endblock %}

--

List.html

{% extends "base.html" %}

{% block content %}

<div class="jumbotron">

<h1> Here the cute puppies.. </h1>

<ul>

{% for pup in puppies %}

<li> {{pup}} / id:{{pup.id}} </li>

{% endfor %}

</ul>

</div>

{% endblock %}

Larger Applications: Models.py, Views.py, Forms.py

Flask has a built-in **blueprints** capability which will allow us to register modular components for our Flask App

Register blueprints in \_\_init\_\_.py

From flask import Blueprint

Owner\_blueprint = Blueprint(‘owners’,\_\_name\_\_,template\_folder=’templates/owner’)

In \_\_init\_\_.py, register as app.register\_blueprint(owner\_blueprint,url\_prefix=’/owners’)

Use hash function to implement for password security. Libraries for hashing security: Bcrypt, Werkzeug. Pip install flask-bcrypt

Bcrypt.generate\_password\_hash(password) // check\_password\_hash(password, ‘password\_string’)

Pip install Werkzeug // from werkzeug.security import generate\_password\_hash, check\_password\_hash // from flask\_login import LoginManager

From flask-login import user\_login, user\_logout, login\_required

We can use **flask\_dance** library to easily add in OAuth (Open Authorisation) backends for our application.

OAuth 2.0, Flask-OAuth, Flask\_Dance // flask-dance.readthedocs.io

Os.environ[‘AUTHLIB\_INSECURE\_TRANSPORT’] = ‘1’

Os.environ[‘AUTHLIB\_RELAX\_TOKEN\_SCOPE] = ‘1’

From flask-dance.contrib.google import make\_google\_blueprint, google

Blueprint = make\_google\_blueprint(client\_id=’’,client\_secret=’’,offline=True,sope=[‘profile’,’email’])

App.regiser\_blueprint(blueprint,url\_prefix=’/login’)

Social company blog:

Bootstrap allows us to create popups.

If any module has to be imported from a folder, create a \_\_init\_\_.py file under it.

----

Project components: Core, Users, Blogs

Project views: Core – index, info // Users – Register, login, logout, Account, user posts // Blog Posts – create, update, delete, blog posts

Project Models: Users, Blog posts

Users: id, profile image, email, username, password, posts(relationship)

Blog posts: id, user-id, date, title, text

Create CRUD forms for Users and Blogs

Create templates that corresponds with views

Create error\_pages folder to store view file for error messages (when error happens, show our own html pages)

Create static folder to store js, css, image files

----

Components usually associate models with them.

PIL Pillow library allows to handle pictures in python

REST API’s

Flask-Restful library.

REST is a standard way of communication between computers in the web

Systems that support and are compliant with REST are often known as RESTFUL.

Automated systems will communicate with the web application

Postman testing tool, send json objects to rest, wrest commands

CRUD REST terms (POST, GET, PUT, DELETE)

Postman is a graphical interface for interactive with REST API

Pip install Flask-Resftul. From flask-restful import Resource, API

Resource is the ‘resource’ which is connecting, API is wrapper around the entire application, that allows to connect.

Create a collection in postman. Collection is a folder holding a bunch of requests.

#RESTful API using Postman GUI

#install flask-restful

#create a flask app

# instantiate an API class inheriting Flask app

#create a class inheriting Resource

#define REST method of either post, get, put, delete

#link the class with the api.add\_resource('class','url address')

#run the flask app

#create a collection folder in postman. choose the method to run.

#enter the url to connect wth the app.

#find the result in Postman

#=================================

from flask import Flask

from flask\_restful import Resource, Api

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

api = Api(app)

class HelloWorld(Resource):

def get(self):

return {'hello': 'world'}

api.add\_resource(HelloWorld, '/')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

All the API methods takes the same parameters.

from flask import Flask, request

from flask\_restful import Resource, Api

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

api = Api(app)

# Later on this will be a model call to our database!

# Right now its just a list of dictionaries

# puppies = [{'name':'Rufus'},{name:'Frankie'},......]

# Keep in mind, its in memory, it clears with every restart!

puppies = []

class PuppyNames(Resource):

def get(self,name):

print(puppies)

# Cycle through list for puppies

for pup in puppies:

if pup['name'] == name:

return pup

# If you request a puppy not yet in the puppies list

return {'name':None},404

def post(self, name):

# Add the dictionary to list

pup = {'name':name}

puppies.append(pup)

# Then return it back

print(puppies)

return pup

def delete(self,name):

# Cycle through list for puppies

for ind,pup in enumerate(puppies):

if pup['name'] == name:

# don't really need to save this

delted\_pup = puppies.pop(ind)

return {'note':'delete successful'},404

class AllNames(Resource):

def get(self):

# return all the puppies :)

return {'puppies': puppies}

api.add\_resource(PuppyNames, '/puppy/<string:name>')

api.add\_resource(AllNames,'/puppies')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

--

Flask-JWT (Json Web Token) library to require authorization before creating a REST API call

From flask\_jwt import JWT, jwt\_required

FlaskRestful documentation

from flask import Flask, request

from flask\_restful import Resource, Api

from secure\_check import authenticate,identity

from flask\_jwt import JWT ,jwt\_required

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

app.config['SECRET\_KEY'] = 'mysecretkey'

api = Api(app)

jwt = JWT(app, authenticate, identity)

# Later on this will be a model call to our database!

# Right now its just a list of dictionaries

# puppies = [{'name':'Rufus'},{name:'Frankie'},......]

# Keep in mind, its in memory, it clears with every restart!

puppies = []

class PuppyNames(Resource):

def get(self,name):

print(puppies)

# Cycle through list for puppies

for pup in puppies:

if pup['name'] == name:

return pup

# If you request a puppy not yet in the puppies list

return {'name':None},404

def post(self, name):

# Add the dictionary to list

pup = {'name':name}

puppies.append(pup)

# Then return it back

print(puppies)

return pup

def delete(self,name):

# Cycle through list for puppies

for ind,pup in enumerate(puppies):

if pup['name'] == name:

# don't really need to save this

delted\_pup = puppies.pop(ind)

return {'note':'delete successful'}

class AllNames(Resource):

@jwt\_required()

def get(self):

# return all the puppies :)

return {'puppies': puppies}

api.add\_resource(PuppyNames, '/puppy/<string:name>')

api.add\_resource(AllNames,'/puppies')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

Database:

import os

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

from flask\_restful import Api,Resource

from secure\_check import authenticate,identity

from flask\_jwt import JWT ,jwt\_required

from flask\_migrate import Migrate

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

###################################################

################ CONFIGURATIONS ###################

##################################################

# Often people will also separate these into a separate config.py file

app.config['SECRET\_KEY'] = 'mysecretkey'

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///' + os.path.join(basedir, 'data.sqlite')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db = SQLAlchemy(app)

Migrate(app,db)

jwt = JWT(app, authenticate, identity)

api = Api(app)

###################################################

################ MODELS ###########################

##################################################

class Puppy(db.Model):

name = db.Column(db.String(80),primary\_key=True)

def \_\_init\_\_(self,name):

self.name=name

def json(self):

return {'name': self.name}

def \_\_str\_\_(self):

return f"{self.name} "

###################################################

################ RESOURCES ###########################

##################################################

class PuppyResource(Resource):

def get(self,name):

pup = Puppy.query.filter\_by(name=name).first()

if pup:

return pup.json()

else:

# If you request a puppy not yet in the puppies list

return {'name':'not found'}, 404

def post(self,name):

pup = Puppy(name=name)

db.session.add(pup)

db.session.commit()

return pup.json()

def delete(self,name):

pup = Puppy.query.filter\_by(name=name).first()

db.session.delete(pup)

db.session.commit()

return {'note':'delete successful'}

class AllPuppies(Resource):

#@jwt\_required()

def get(self):

# return all the puppies :)

puppies = Puppy.query.all()

# return json of (puppies)

return [pup.json() for pup in puppies]

api.add\_resource(PuppyResource, '/puppy/<string:name>')

api.add\_resource(AllPuppies,'/puppies')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

Heroku platform for Flask app deployment.

Steps: install a few libraries and command line tools to connect the local application and push to Heroku platform.

1. Install gunicorn
2. Install Heroku CLI
3. Create Flask app
4. Create requirements.txt file
5. Create a Procfile
6. Create a Heroku account
7. Push and launch the application

Gunicorn is a python wsgi for unix

Env flaskdeploy

Pip freeze > requirements.txt

Create an app in Heroku. From cmd, heroku login, Git init on the folder. Heroku git:remote –a <app name>

Git push heroku master. Copy the http address and run in the browser. <https://jj-my-first-flask-app.herokuapp.com/>

--

Online payments: Stripe is a company that allows to easily integrate their API to your web application to accept payments.

Shopify is one which allows to create online sore app quickly.

Pip install –upgrade stripe // <https://stripe.com/docs/legacy-checkout/flask>

[okstamps@gmail.com](mailto:okstamps@gmail.com) // 1@Puthinthara

xcross