# imVEnv

## install

pip install virtualenv

## Create a virtual environment

virtualenv env

## activate

# Install Flask

pip intall flask

Check version flask –version

# First pgm

Create python file “market.py”

Check website

<https://flask.palletsprojects.com/en/2.1.x/quickstart/#a-minimal-application>

clear terminal with “clear” and add following commands

1. set FLASK\_APP=market.py
2. $env:FLASK\_APP = "market.py"
3. flask run

Text

Description automatically generated

127.0.0.1.5000 is home address

Should see the web page with hello world

Hit Ctrl+C to quit

Type flask run to run program again.

## Turn on debug mode to make easy

$env:FLASK\_ENV = "development"

Now no need to run and restart web server. Change python and refresh page.

# Adding another page

Add following code

@app.route("/about")  
def about\_page():  
 return "<h1>About page</h1>"

In browse add “/about”

Graphical user interface, text, application

Description automatically generated

# Cuastom about jpages

Change code as following

@app.route("/about/<username>")  
def about\_page(username):  
 return f"<h1>This is the about page of {username}</h1>"

Add about/Ranju to get the about page of user

Graphical user interface, text, application, Word

Description automatically generated

# Styling and templates

Create new directory called templates (name is important)

Graphical user interface, text, application

Description automatically generated

Create an html file in the template folder and call it home

Add h1 tag in body as following

<h1>Home Page</h1>

In market.py Import render template as following

from flask import Flask, render\_template

change market.py as following

from flask import Flask, render\_template  
  
app = Flask(\_\_name\_\_)  
  
@app.route("/")  
def home\_page():  
 return render\_template("home.html")

It will run the home page.

# Styling with bootstrap

Copy code from the following page

<https://getbootstrap.com/docs/4.5/getting-started/introduction/>

copy code to html file

Web page should be updated

Add @app.route(“/home”) so that when adderess has /home it takes to home jpage

@app.route("/")  
@app.route("/home")  
def home\_page():  
 return render\_template("home.html")

Copy following code at start in body tag

<nav class = "navbar navbar-expand-md navbar-dark bg-dark">  
 <a class = "navbar-brand" href="#">Flask Market</a>  
 <button class = "navbar-toggler" type="button" data-toggle="collapse" data-target = "#navbarNav"></button>  
 <div class = "collapse navbar-collapse" id = "navbarNav">  
  
 </div>  
</nav>

Add following code inside the div tag of nav bar

<ul class="navbar-nav mr-auto">  
 <li class="nav-item active">  
 <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link" href="#">Market</a>  
 </li>

</ul>

<ul class="navbar-nav">  
 <li class="nav-item">  
 <a class="nav-link" href="#">Login</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link" href="#">Register</a>  
 </li>  
</ul>

We should get a nav bar on top

Graphical user interface, text, application

Description automatically generated

Below body tag add the following for grey bg

</body>  
 <style>  
 body {  
 background-color: #212121;  
 color: white;  
 }  
 </style>  
</html>

# Sending date to template

Create new template as following in py file

@app.route("/market")  
def market\_page():  
 return render\_template("market.html")

Create new html file called market in template folder

Graphical user interface, text, application, chat or text message

Description automatically generated

For ease copy code from home.html

Change title and h1 appropriately

In address bar <http://127.0.0.1:5000/market> add this to get new page.

You should get new jpage

Add phone to new route as following

@app.route("/market")  
def market\_page():  
 return render\_template("market.html", item\_name="Phone")

To access the new item in webpage add jinja code

In market.html add the following under list and h1

Text

Description automatically generated

Refresh market page to see the item.

A screenshot of a computer

Description automatically generated

Add following dictionary to the market route. Change return item\_name to items as following

@app.route("/market")  
def market\_page():  
 items = [  
 {'id': 1, 'name': 'Phone', 'barcode': '893212299897', 'price': 500},  
 {'id': 2, 'name': 'Laptop', 'barcode': '123985473165', 'price': 900},  
 {'id': 3, 'name': 'Keyboard', 'barcode': '231985128446', 'price': 150}  
 ]  
 return render\_template("market.html", items=items)

Change item name in market.html as following

Text

Description automatically generated

In market.html it looks like this

Text

Description automatically generated

We will add them in to a bootstrap table

Add the following code to market.html replacing <p> {{ items }} </p>

<table class="table table-hover table-dark">  
 <thead>  
 <tr>  
 <!-- Your Columns HERE -->  
 <th scope="col">ID</th>  
 </tr>  
 </thead>  
 <tbody>  
 <!-- Your rows inside the table HERE: -->  
 <tr>  
 <td>Value for Id</td>  
 </tr>  
 </tbody>  
</table>

In table head add the following

<th scope="col">ID</th>  
<th scope="col">Name</th>  
<th scope="col">Barcode</th>  
<th scope="col">Price</th>

In table body add the following

<td>Value for Id</td>  
<td>Value for Name</td>  
<td>Value for Barecode</td>  
<td>Value for Price</td>

When you refresh you should get as following

A screenshot of a computer

Description automatically generated

To display row in loop change ins side <tbody> as following and actual values

<tbody>  
 <!-- Your rows inside the table HERE: -->  
 {% for item in items %}  
 <tr>  
 <td>{{ item.id }}</td>  
 <td>{{ item.name }}</td>  
 <td>{{ item.barcode }}</td>  
 <td>${{ item.price }}</td>  
 </tr>  
 {% endfor %}  
</tbody>

Pjage will look like this

A screenshot of a computer

Description automatically generated

To add buttons as one more option

Add one more column in <thead>

<thead>  
 <tr>  
 <!-- Your Columns HERE -->  
 <th scope="col">ID</th>  
 <th scope="col">Name</th>  
 <th scope="col">Barcode</th>  
 <th scope="col">Price</th>  
 <th scope="col">Options</th>  
 </tr>  
</thead>

Add a column in <tbody> as following

<tbody>  
 <!-- Your rows inside the table HERE: -->  
 {% for item in items %}  
 <tr>  
 <td>{{ item.id }}</td>  
 <td>{{ item.name }}</td>  
 <td>{{ item.barcode }}</td>  
 <td>${{ item.price }}</td>  
 <td>  
 <button class="btn btn-outline btn-info">More Info</button>  
 <button class="btn btn-outline btn-success">Purchase This Item</button>  
 </td>  
 </tr>  
 {% endfor %}  
</tbody>

Web page should look like this

A screenshot of a computer

Description automatically generated

# Template inheritance

To create a base template to inherit from

Create a base.html file in template folder

A screenshot of a computer

Description automatically generated with medium confidence

Delete whats in it and add the following code to the base.html

<!doctype html>  
<html lang="en">  
 <head>  
 <!-- Required meta tags -->  
 <meta charset="utf-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
 <!-- Bootstrap CSS -->  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">  
 <title>Base Title</title>  
 </head>  
 <body>  
 <!-- Navbar here -->  
 <!-- Future Content here -->  
  
  
  
 <!-- Optional JavaScript -->  
 <!-- jQuery first, then Popper.js, then Bootstrap JS -->  
 <script src='https://kit.fontawesome.com/a076d05399.js'></script>  
 <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>  
 <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN" crossorigin="anonymous"></script>  
 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" integrity="sha384-B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV" crossorigin="anonymous"></script>  
 </body>  
 <style>  
 body {  
 background-color: #212121;  
 color: white  
 }  
 </style>  
</html>

Open home.html and delete everything

Add the following

{% extends "base.html" %}

Copy navigation bar from market.html to base in the same place.

Now home page will have navigation bar and grey bg colour

# To get page title.

Change title tag in base.html as following

<title>  
 {% block title %}  
  
 {% endblock %}  
</title>

In home.html add code as following

{% block title %}  
 Home Page  
{% endblock %}

This should give the title home page to home.html

Graphical user interface, text, application

Description automatically generated

## Changing contents

In base.html add the following under nav

{% block content %}  
  
{% endblock %}

In home.html add the following

{% block content %}  
 Home page contents to appear soon...  
{% endblock %}

Now home jpage will have that text.

In market.html add the following to the start of the file before all html tags

{% extends "base.html" %}  
{% block title %}  
 Home Page  
{% endblock %}  
  
{% block content %}  
  
{% endblock %}

Copy the table tag and all its contents into the **block content**

Delete all of the html

Final market.html file will look like this

{% extends "base.html" %}  
  
{% block title %}  
 Market Page  
{% endblock %}  
  
{% block content %}  
<table class="table table-hover table-dark">  
 <thead>  
 <tr>  
 <!-- Your Columns HERE -->  
 <th scope="col">ID</th>  
 <th scope="col">Name</th>  
 <th scope="col">Barcode</th>  
 <th scope="col">Price</th>  
 <th scope="col">Options</th>  
 </tr>  
 </thead>  
 <tbody>  
 <!-- Your rows inside the table HERE: -->  
 {% for item in items %}  
 <tr>  
 <td>{{ item.id }}</td>  
 <td>{{ item.name }}</td>  
 <td>{{ item.barcode }}</td>  
 <td>${{ item.price }}</td>  
 <td>  
 <button class="btn btn-outline btn-info">More Info</button>  
 <button class="btn btn-outline btn-success">Purchase This Item</button>  
 </td>  
 </tr>  
 {% endfor %}  
 </tbody>  
</table>  
{% endblock %}

No change to the pages as we are inheriting and showing the same layout and display

# Connecting Market and Home buttons

In base.html add /home inside href in the a tag of base.html will work

But its not efficient.

To work by calling the route in market.py add the following to base.html

<li class="nav-item active">  
 <a class="nav-link" href="{{ url\_for("home\_page") }} ">Home <span class="sr-only">(current)</span></a>  
</li>

Make same changes for market page

<li class="nav-item">  
 <a class="nav-link" href="{{ url\_for("market\_page")}}">Market</a>  
</li>

# Model and Database

Use database instead of passing values.

Store in a file.

Using SQLite to store data.

## Install flask tool

1. Hit Ctrl+C to stop website
2. Type - pip install flask-sqlalchemy

In market.py

from flask\_sqlalchemy import SQLAlchemy

add following line

Text

Description automatically generated

Add db columns as following just above the @app.route(“/”)

class Item(db.Model):  
 id = db.Column(db.Integer(), primary\_key=True)  
 name = db.Column(db.String(length=30), nullable=False, unique=True)  
 price = db.Column(db.Integer(), nullable=False)  
 barcode = db.Column(db.String(length=12), nullable=False, unique=True)  
 description = db.Column(db.String(length=1024),j nullable=False, unique=True)

Add following code just above db = SQLAlchemy(app)

app.config["SQLALCHEMY\_DATABASE\_URI"]= "sqlite:///market.db"

Now we can run the from program to create the db file

Type python in terminal to get the python command

Text

Description automatically generated

Type “from market import db”

Text

Description automatically generated

Now run “db.create\_all()” command

Text

Description automatically generated with medium confidence

This will create a db file in project directory

Graphical user interface, text, application, chat or text message

Description automatically generated

## Adding items to database

To add items add the following

Graphical user interface, text

Description automatically generated

To check if item is added add following command

A picture containing text

Description automatically generated

Add one more item as following and check if added as show below

Query in python shell

Graphical user interface, text, application

Description automatically generated

## Query in python shell

Add following in markey.py

def \_\_repr\_\_(self):  
 return f'Item {self.name}'

add following code in terminal

Text

Description automatically generated

To get all elements in the db add the following in terminal

Text

Description automatically generated

To clear screen

Text

Description automatically generated

To show filtered itemText

Description automatically generated

There might be a list of items so we need to loop

Text

Description automatically generated

Now we will change the dummy database in market.py to the python code

Chart, line chart

Description automatically generated

Add Item.query.all() there as below

@app.route("/market")  
def market\_page():  
 items = Item.query.all()  
 return render\_template("market.html", items=items)

Now we can run the server and get db in the web page

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

For a graphical representation of DB go to https://sqlitebrowser.org/

After installing or downloading you can file open the db and look at the tables

Graphical user interface, text, application, email

Description automatically generated

# Project Restructure

Need new file for model and routes

Create models.py file

Cut paste the following code to it from market.py

class Item(db.Model):  
 id = db.Column(db.Integer(), primary\_key=True)  
 name = db.Column(db.String(length=30), nullable=False, unique=True)  
 price = db.Column(db.Integer(), nullable=False)  
 barcode = db.Column(db.String(length=12), nullable=False, unique=True)  
 description = db.Column(db.String(length=1024), nullable=False, unique=True)  
  
 def \_\_repr\_\_(self):  
 return f'Item {self.name}'

create new routes.py file

Cut paste the following code from market.py to routes.py

@app.route("/")  
@app.route("/home")  
def home\_page():  
 return render\_template("home.html")  
  
@app.route("/market")  
def market\_page():  
 items = Item.query.all()  
 return render\_template("market.html", items=items)

To get all the imports we will create a package

Create run.py

Cut every thing from market.py to run.py

Run.py will look like this

from flask import Flask, render\_template  
from flask\_sqlalchemy import SQLAlchemy  
app = Flask(\_\_name\_\_)  
app.config["SQLALCHEMY\_DATABASE\_URI"] = "sqlite:///market.db"  
db = SQLAlchemy(app)

Can delete market.py

Create new folder called market and move everything except run.py to the new folder ‘market’

A screenshot of a computer

Description automatically generated with medium confidence

Create a new file called \_\_init\_\_.py in the folder “market”

Graphical user interface, text, application, chat or text message

Description automatically generated

Move everything (following code) from run.py to \_\_init\_\_.py

from flask import Flask, render\_template  
from flask\_sqlalchemy import SQLAlchemy  
app = Flask(\_\_name\_\_)  
app.config["SQLALCHEMY\_DATABASE\_URI"] = "sqlite:///market.db"  
db = SQLAlchemy(app)

Since we have\_\_init\_\_.py in the folder market we can import market as a package.

Add the following code to run.py

from market import app  
  
# checks if the run.py file has executed directly and not imported  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

add the following code to \_\_init\_\_.py

from market import routes

add the following code to routes.py

from market import app  
from flask import render\_template  
from market.models import Item

in models.py add the following

from market import db

Now website should work fine

A screenshot of a computer

Description automatically generated with medium confidence

# Model Relationships

## Accepting users

## Use another model for users

Add the following model to models.py

class User(db.Model):  
 id = db.Column(db.Integer(), primary\_key=True)  
 username = db.Column(db.String(length=30), nullable=False, unique=True)  
 email\_address = db.Column(db.String(length=50), nullable=False, unique=True)  
 password\_hash = db.Column(db.String(length=60), nullable=False)  
 budget = db.Column(db.Integer(), nullable=False, default=1000)  
 items = db.relationship('Item', backref='owned\_user', lazy=True)

add the following line to the class Item to connect between user and item.

owner = db.Column(db.Integer(), db.ForeignKey('user.id'))

To delete the db

Text

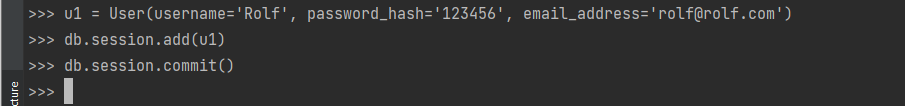
Description automatically generated

Create new db as below

A picture containing text

Description automatically generated

Add new user as below



Wwe can check if its added as below

Graphical user interface, application

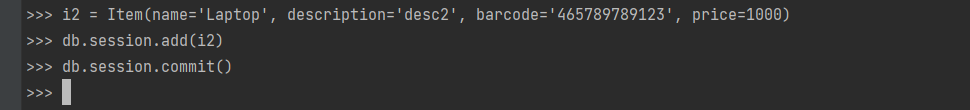
Description automatically generated

Add an item as below

Text

Description automatically generated

Add another item as below



We can check the items in db as below

Graphical user interface, text

Description automatically generated

Now lets try to assign ownership to iphone10

We can get the name if IPhone10 as below

Text

Description automatically generated

## Adding owner to an item

If you type item1.owner it returns nothing because no owner assigned

Shape

Description automatically generated with medium confidence

We can add owner as below

Text

Description automatically generated

To check if the user was added successfully type the following

Text

Description automatically generated

# Flask Forms

Using forms to enter data

## Install packages

Flask-wtforms



Wtforms should be installed with the earlier intall



Create a new file called forms.py

Add the following code

from flask\_wtf import FlaskForm  
from wtforms import StringField, PasswordField, SubmitField  
  
class RegisterForm(FlaskForm):  
 username = StringField(label='username')  
 email\_address = StringField(lable='email')  
 password1 = PasswordField(label='password1')  
 password2 = PasswordField(label='password2')  
 submit = SubmitField(label='submit')

In routes.py add the following import at start

from market.forms import RegisterForm

at the end add the following class

@app.route('/register')  
def register\_page():  
 form = RegisterForm()  
 return render\_template('register.html', form = form)

Create a new file in template folder called register.html

Add the following code

{% extends 'base.html' %}  
{% block title %}  
Register Page  
{% endblock %}  
  
{% block content %}  
<h1>Register Form</h1>  
{% endblock %}

We need to generate a secret key for the flask application to give one more security layer.

Can be generated in python from terminal

In terminal add the following

Text

Description automatically generated

Copy the generated key

Add it to the \_\_init\_\_py file as below

Text

Description automatically generated

Quit from thee python terminal

Text

Description automatically generated

Run the server as below

Text

Description automatically generated

Click on link to open the web page

Type register at the end of address to go to register page and we can see the register headline there

A picture containing graphical user interface

Description automatically generated

Connect link to register button on nav bar

Add the following code in the base.html file

<a class="nav-link" href="{{ url\_for("register\_page")}}">Register</a>



If we run the server again with the command “python run.py” or if window is refreshed we should have the link working on register button

## Adding fields

Add the following code to register.html

{% block content %}  
 <form method="POST" class ="form-register" style="color:white">  
 {{ form.username.label() }}  
 {{ form.username(class="form-control", placeholder="User Name") }}  
 </form>  
{% endblock %}

Check the page to see a field

A picture containing graphical user interface

Description automatically generated

To make it look good make form inside a div as below

{% block content %}  
 <div class="container">  
 <form method="POST" class ="form-register" style="color:white">  
 {{ form.username.label() }}  
 {{ form.username(class="form-control", placeholder="User Name") }}  
 </form>  
 </div>  
{% endblock %}

Change the class RegisterForm in forms.py as following

Text

Description automatically generated

The result should be as following

Background pattern

Description automatically generated

Change the register.html as following

{% extends 'base.html' %}  
{% block title %}  
Register Page  
{% endblock %}  
  
{% block content %}

<body class="text-center">  
 <div class="container">  
 <form method="POST" class ="form-register" style="color:white">  
 <h1 class="h3 mb-3 font-weight-normal">  
 Please Create your Account  
 </h1>  
 <br>  
 {{ form.username.label() }}  
 {{ form.username(class="form-control", placeholder="User Name") }}  
  
 {{ form.email\_address.label() }}  
 {{ form.email\_address(class="form-control", placeholder="Email Address") }}  
  
 {{ form.password1.label() }}  
 {{ form.password1(class="form-control", placeholder="Password") }}  
  
 {{ form.password2.label() }}  
 {{ form.password2(class="form-control", placeholder="Confirm Password") }}  
  
 <br>  
  
 {{ form.submit(class="btn btn-lg btn-block btn-primary") }}  
 </form>  
 </div>

</body>  
{% endblock %}

Result should be as below

A screenshot of a computer

Description automatically generated

# Validations

Add user at the end of import market.models as below

*from* market.models *import* Item, User

import redirect as below

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

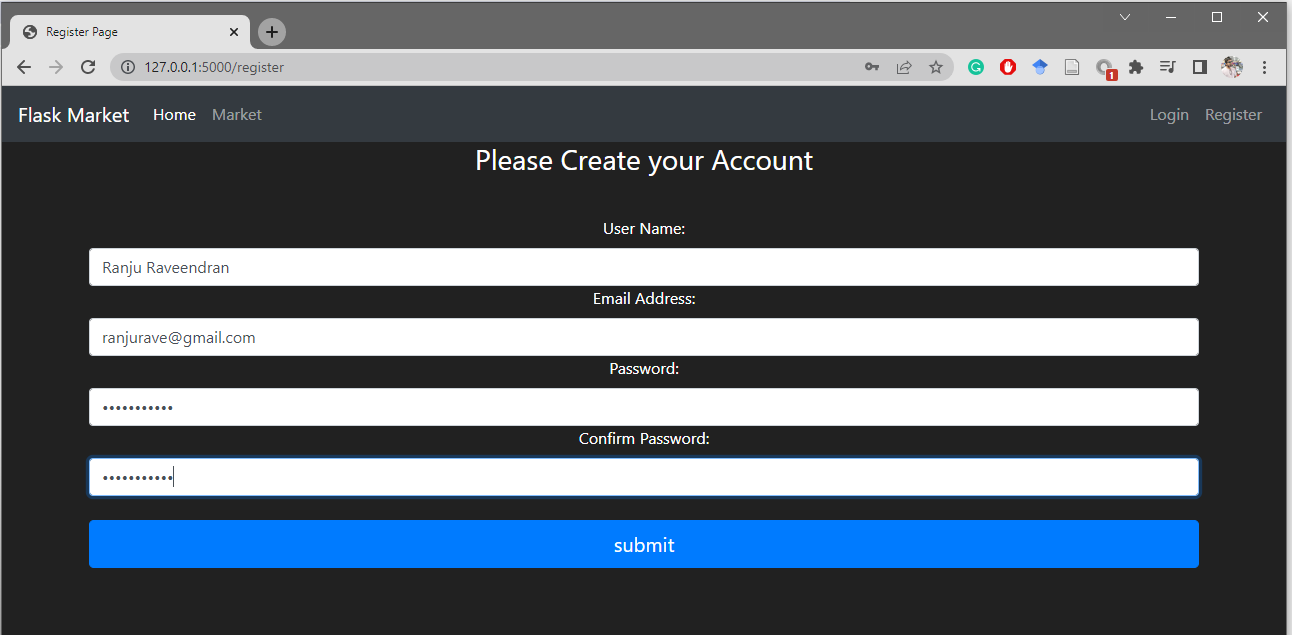
import db as below

*from* market *import* db

add following code to the register\_page method

@app.route('/register')  
*def* register\_page():  
 form = RegisterForm()  
 *if* form.validate\_on\_submit():  
 user\_to\_create = User(username=form.username.data, email\_address = form.email\_address.data, password\_hash= form.password1.data)  
 db.session.add(user\_to\_create)  
 db.session.commit()  
 *return* redirect(url\_for('market\_page'))  
 *return* render\_template('register.html', form=form)

Run the server and go to register page

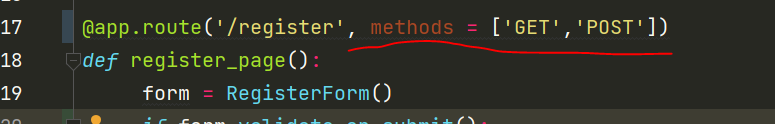


Add some data and hit submit button

There will be error.

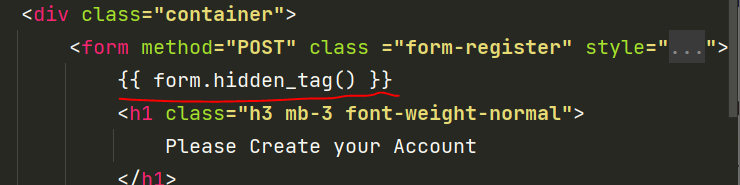
To fix this we need POST method

So change the method call in route.py as below



## Cross site Request Forgery (prevent this attack)

To prevent this add the following line in register.html



Now the account will be created and market page will be opened.

## Checking password

In forms.py add the following

*from* wtforms.validators *import* Length, EqualTo, Email

Now we can validation for all the forms

*class* RegisterForm(FlaskForm):  
 username = StringField(label='User Name: ', validators=Length(min=2,max=30))  
 email\_address = StringField(label='Email Address: ', validators=Email())  
 password1 = PasswordField(label='Password: ', validators=Lenght(min=6))  
 password2 = PasswordField(label='Confirm Password: ',validators=EqualTo('password1'))  
 submit = SubmitField(label='submit')

to add empty field validation, import one more class as below

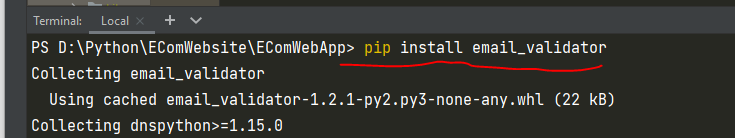
*from* wtforms.validators *import* Length, EqualTo, Email, DataRequired

change script as below

*class* RegisterForm(FlaskForm):  
 username = StringField(label='User Name: ', validators=[Length(min=2,max=30), DataRequired() ])  
 email\_address = StringField(label='Email Address: ', validators=[Email(), DataRequired()])  
 password1 = PasswordField(label='Password: ', validators=[Length(min=6), DataRequired()])  
 password2 = PasswordField(label='Confirm Password: ',validators=[EqualTo('password1'), DataRequired()])  
 submit = SubmitField(label='submit')

## Handling error from forms

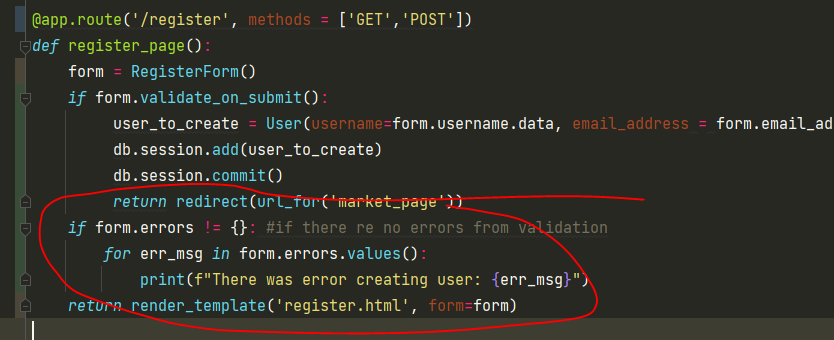
First we need to install email validator



Add the following code at the end of routes.py

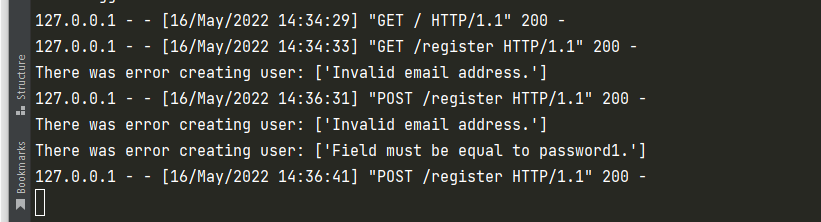
*if* form.errors != {}: #if there re no errors from validation  
 *for* err\_msg *in* form.errors.values():  
 print(f"There was error creating user: {err\_msg}")  
*return* render\_template('register.html', form=form)

like this



Open terminal and website side by side to see the error message

We can see the error message in the terminal



# Flash Messages and Adv validations

To get proper messages go to routes.py and make the changes

*from* flask *import* render\_template, redirect, url\_for, flash

change print to flash in the route.py

flash(f"There was error creating user: {err\_msg}")

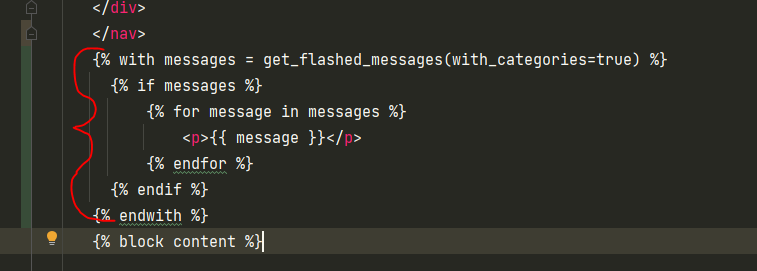
To make the flash message uniform in all forms in the application we create it in base.html

Open base.html

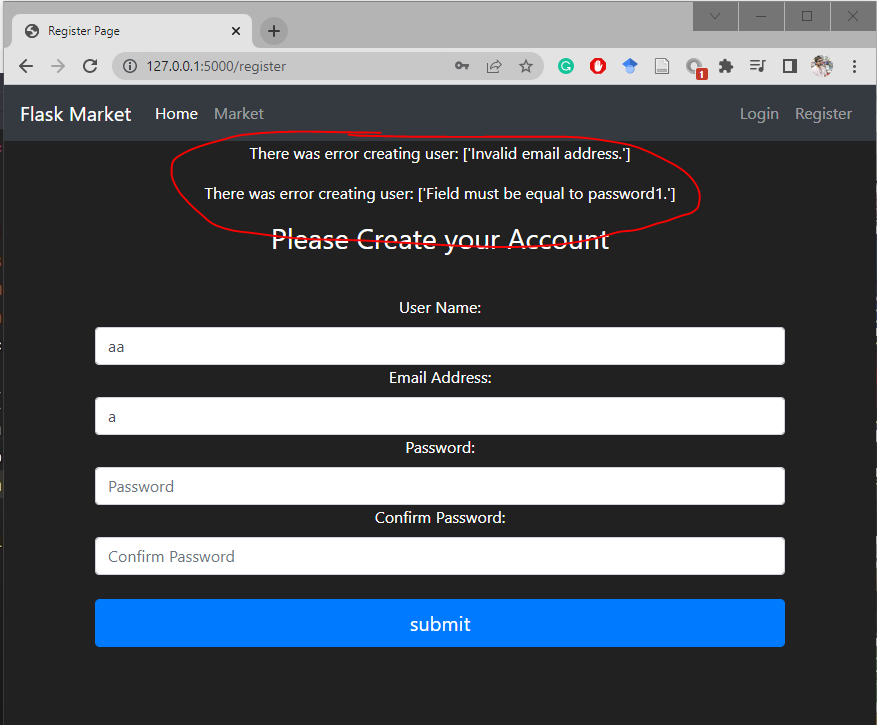
Add the following just under the closing nav tag

{% with messages = get\_flashed\_messages(with\_categories=true) %}  
 {% if messages %}  
 {% for message in messages %}  
 <p>{{ message }}</p>  
 {% endfor %}  
 {% endif %}  
{% endwith %}

Just like below

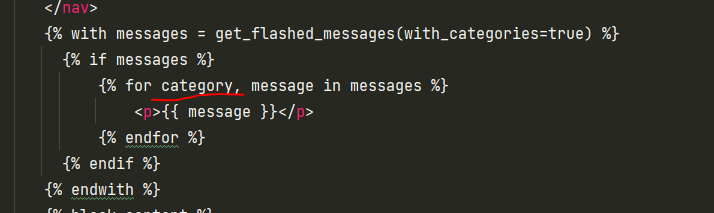


Now you should get message as below when entered wrong inputs



Messages are in tuples

We could add category as below in base.html to get rid of the brackets



To get a uniform better message

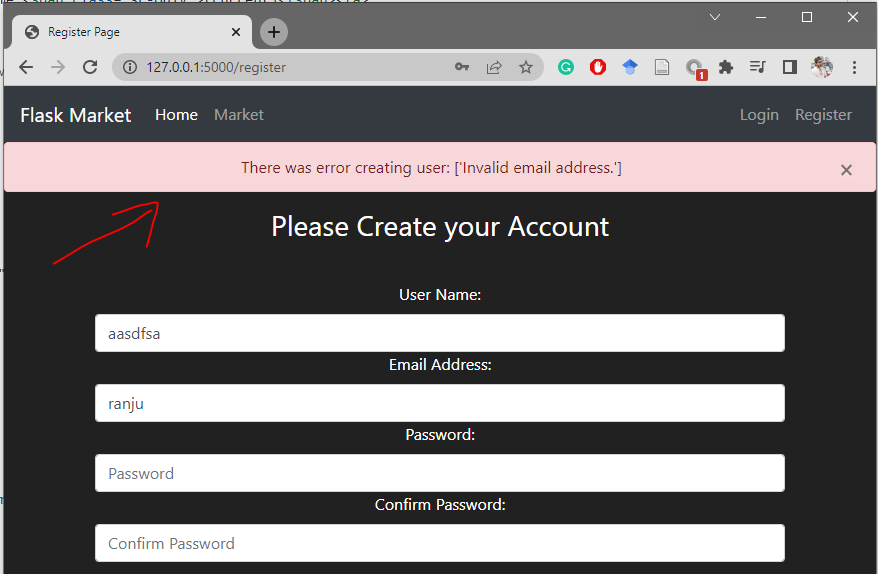
In route.py add the following

flash(f"There was error creating user: {err\_msg}", category='danger')

back in base.html add some html and bootstrap as below

{% with messages = get\_flashed\_messages(with\_categories=true) %}  
 {% if messages %}  
 {% for category, message in messages %}  
 <div class**="alert alert-{{ category }}"**>  
 <button type **= "button"** class **= "m1-2 mb-1 close"** date-dismiss**="alert"** aria-label**="Close"**>  
 <span aria-hidden**="true"**> **&times;**</span>  
 </button>  
 {{ message}}  
 </div>  
 {% endfor %}  
 {% endif %}  
{% endwith %}

We should get a message as below



## Preventing user register again

Do validation in forms.py

Add the following to forms.py

from market.models import User

add validationError to wtforms.validators as below

from wtforms.validators import Length, EqualTo, Email, DataRequired, ValidationError

add the following 2 methods to the start of the RegisterForm class in forms.py

def validate\_username(self, username\_to\_check):  
 user = User.query.filter\_by(username=username\_to\_check.data).first()  
 if user:  
 raise ValidationError("Username already exist!")  
  
def validate\_email\_address(self, email\_address\_to\_check):  
 email\_address = User.query.filter\_by(email\_address=email\_address\_to\_check.data).first()  
 if email\_address:  
 raise ValidationError("Email address already used")

## Hide password in DB

Will install a package called flask\_bcrypt

Text

Description automatically generated

In \_\_init\_\_.py file import the library as below

from flask\_bcrypt import Bcrypt

In \_\_init\_\_.py create an object at the end as below (above routes import)

bcrypt = Bcrypt(app)

in models.py import bcrypt as below

from market import bcrypt

In models.py add the following at the end of User class

@property  
def password(self):  
 return self.password  
  
@password.setter  
def password(self, plain\_text\_password):  
 self.password\_hash=bcrypt.generate\_password\_hash(plain\_text\_password).decode('utf-8')

Edit the password\_hash to password in routes.py.

user\_to\_create = User(username=form.username.data, email\_address = form.email\_address.data, password= form.password1.data)

new users will have password saved in coded format

Graphical user interface, application

Description automatically generated

# User Login