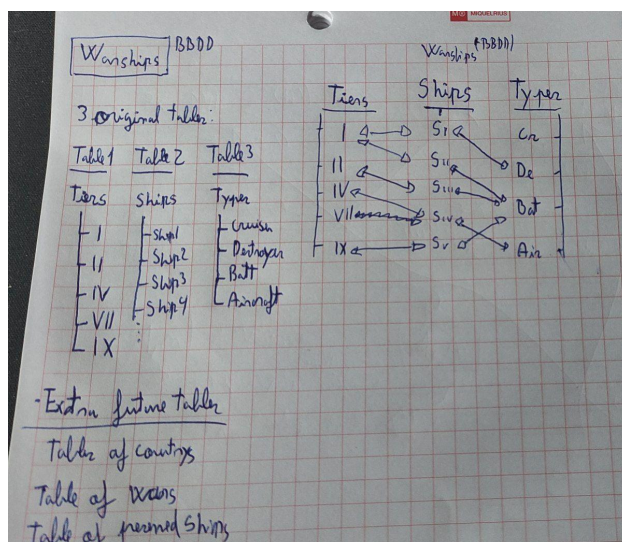
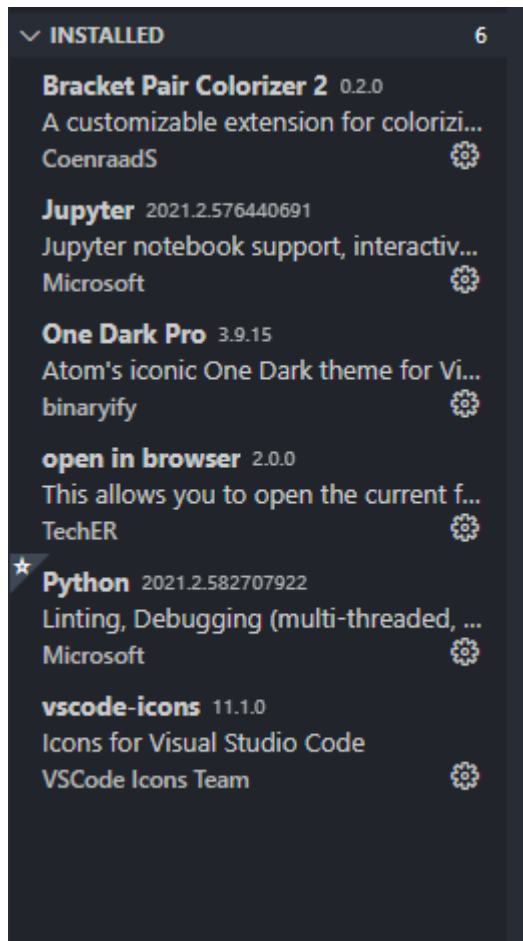


°Pasos seguidos para crear hacer el proyecto de **Flask\_SQLAlchemy\_Python**

uso visual studio code que me parece más cómodo y práctico.

suelo instalar esto para cambiar un poco el look y detalles para trabajar más cómodo.



lo primero que hago es actualizar pyth y seguidamente instalar el pip para poder trabajar con entornos virtuales.

```

PS E:\ASIX\m4\flask\ships_1> pip install pipenv
Collecting pipenv
  Downloading pipenv-2020.11.15-py2.py3-none-any.whl (3.9 MB)
    | 3.9 MB 3.2 MB/s
Collecting certifi
  Downloading certifi-2020.12.5-py2.py3-none-any.whl (147 kB)
    | 147 kB 3.3 MB/s
Collecting virtualenv
  Downloading virtualenv-20.4.2-py2.py3-none-any.whl (7.2 MB)
    | 2.0 MB 3.3 MB/s eta 0:00:02

```

```

PS E:\ASIX\m4\flask\ships_1> pipenv shell
Creating a virtualenv for this project...
Pipfile: E:\ASIX\m4\flask\ships_1\Pipfile
Using E:/Programas/python/python.exe (3.9.2) to create virtualenv...
[= ] Creating virtual environment...created virtual environment CPython3.9.2.final.0-64 in 3310ms
creator CPython3Windows(dest=C:\Users\janva\.virtualenvs\ships_1-wDEISKiU, clear=False, no_vcs_ignore=False,
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Use
added seed packages: pip==21.0.1, setuptools==52.0.0, wheel==0.36.2
activators BashActivator,BatchActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator

Successfully created virtual environment!
Virtualenv location: C:\Users\janva\.virtualenvs\ships_1-wDEISKiU

```

iniciamos el entorno virtual, nos crea un doc llamado pipfile.

```

ships_1 > Pipfile
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
flask = "*"

[dev-packages]

[requires]
python_version = "3.9"

```

ahora en este entorno virtual instalamos el flask.

```
PS E:\ASIX\m4\flask\ships_1> pipenv install flask
Installing flask...
Adding flask to Pipfile's [packages]...
Installation Succeeded
Pipfile.lock not found, creating...
Locking [dev-packages] dependencies...
```

```
Pipfile
Pipfile.lock
workspace.code-workspace
```


```
ships_1 > app.py > ...
from flask import Flask

app = Flask(__name__)
```

```
@app.route('/')
def index():
    return 'hey'
```


```
25
26
27 if __name__ == '__main__':
28     app.run(port = 8082, debug = True)
```

una vez q lo hemos iniciado y podemos abrir el msg en el navegador



HelpSponsorLog inRegister

# Flask-SQLAlchemy 2.4.4

`pip install Flask-SQLAlchemy`

✓ Latest version

Released: Jul 14, 2020

Adds SQLAlchemy support to your Flask application.

### Navigation

- Project description
- Release history
- Download files

### Project links

- Homepage
- Issue tracker
- Code
- Documentation

### Statistics

GitHub statistics:

★ Stars: 3,403

### Project description

Flask-SQLAlchemy is an extension for [Flask](#) that adds support for [SQLAlchemy](#) to your application. It aims to simplify using SQLAlchemy with Flask by providing useful defaults and extra helpers that make it easier to accomplish common tasks.

### Installing

Install and update using [pip](#):

```
$ pip install -U Flask-SQLAlchemy
```

### A Simple Example

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///example.sqlite"
db = SQLAlchemy(app)
```

ahi se puede ver como instalarlo tanto en la pag of de flask.

```
You should consider upgrading via the 'e:\programas\python\python.exe' command.
PS E:\ASIX\m4\flask\ships_1> pipenv install Flask-SQLAlchemy
Installing Flask-SQLAlchemy...
[ ===] Installing Flask-SQLAlchemy...
```

```
app.py  Pipfile  X
ships_1 > Pipfile
1  [[source]]
2  url = "https://pypi.org/simple"
3  verify_ssl = true
4  name = "pypi"
5
6  [packages]
7  flask = "*"
8  flask-sqlalchemy = "*"
9
10 [dev-packages]
11
12 [requires]
13 python_version = "3.9"
14
```

```
app = Flask(__name__)
dba = SQLAlchemy(app)
```

con “dba” usaremos el cursor de sqlalchemy

lo mejor es ir mirando la documentación oficial:

# Quickstart

Flask-SQLAlchemy is fun to use, incredibly easy for basic applications, and readily extends for larger applications. For the complete guide, checkout the API documentation on the [SQLAlchemy](#) class.

## A Minimal Application

For the common case of having one Flask application all you have to do is to create your 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:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///tmp/test.db'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self):
        return '<User %r>' % self.username
```

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root@localhost/flask_warships'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
```

en el app.config, ponemos q vamos usar y donde esta nuestra base de datos. En mi caso al usar xampp, tengo q poner el usuario y contraseña. Al no tener contraseña solo pongo root@localh...

```

PS E:\ASIX\m4\flask\ships_1> pipenv install marshmallow-sqlalchemy
Installing marshmallow-sqlalchemy...
Adding marshmallow-sqlalchemy to Pipfile's [packages]...
Installation Succeeded
Pipfile.lock (2b1467) out of date, updating to (2ad9ad)...
Locking [dev-packages] dependencies...
Locking [packages] dependencies...
  Locking...Building requirements...
Resolving dependencies...
[=== ] Locking...

```

instalamos el marshmallow



# Flask-Marshmallow

[changelog](#) // [github](#) // [pypi](#) // [issues](#)

## Flask + marshmallow for beautiful APIs

Flask-Marshmallow is a thin integration layer for [Flask](#) (a Python web framework) and [marshmallow](#) (an object serialization/deserialization library) that adds additional features to marshmallow, including URL and Hyperlinks fields for HATEOAS-ready APIs. It also (optionally) integrates with [Flask-SQLAlchemy](#).

Get it now

```
pip install flask-marshmallow
```

creamos una carpeta de templates y lo testamos

```

templates > index.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>start</title>
8  </head>
9  <body>
10     <p>hola JAN </p>
11 </body>
12 </html>

```

```

app.py > ...
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy
#from flask_marshmallow import Marshmallow

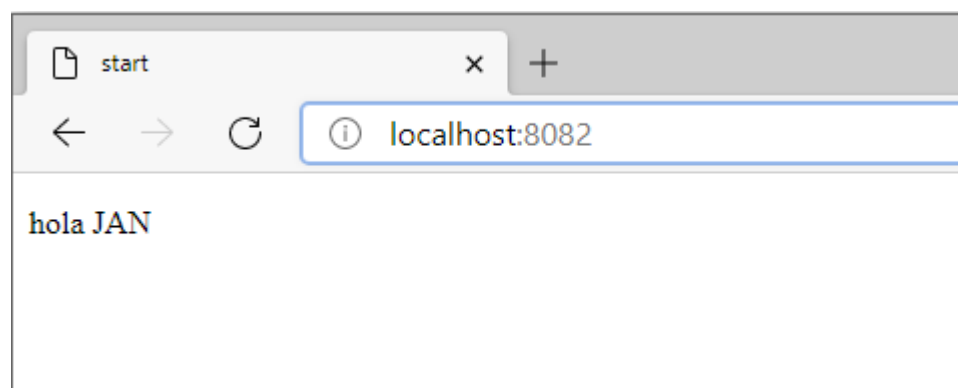
```

importamos el render\_template

```

@app.route('/')
def index():
    return render_template('index.html')

```



cremos un css para ver si funciona

```

app.py  index.html  main.css
main.css > ...
1  body {
2
3      background-color: powderblue;
4  }
5

```

para llamarlo no lo haremos de manera “manual” flask ya que usa jinja 2. Usaremos una **sentencia de motor de plantilla**. `{{}}` para los arch estáticos

```

<link rel="stylesheet" href="{{url_for('static', filename='main.css')}}">

```

esto lo ponemos en nuestro index



## Static Files

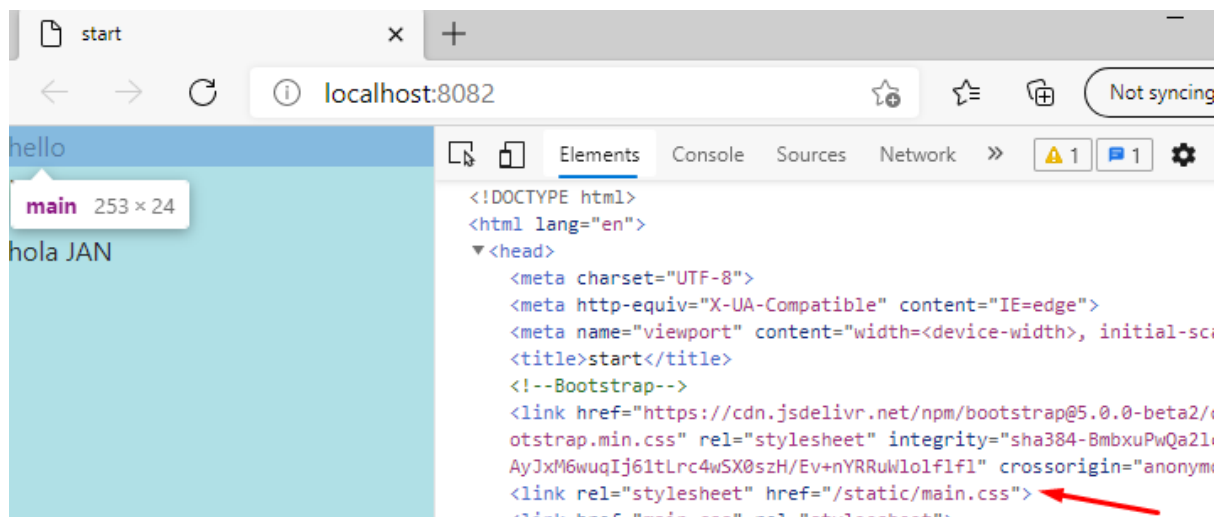
Dynamic web applications also need static files. That's usually where the CSS and JavaScript files are coming from. Ideally your web server is configured to serve them for you, but during development Flask can do that as well. Just create a folder called `static` in your package or next to your module and it will be available at `/static` on the application.

To generate URLs for static files, use the special `'static'` endpoint name:

```
url_for('static', filename='style.css')
```

The file has to be stored on the filesystem as `static/style.css`.

(me he pasado 1h con esto porq tenía escrito "stylesheet" mal, recordemos de usar el autocompletado de los programas...)



usaremos tmb el bootstrap

```
<!--Bootstrap-->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-BmbxuPwQa21c/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl" crossorigin="anonymous">
```

Ultimately this is a frustrating browser cache issue, which can be solved by forcing the browser to do a "hard refresh", which is going to be a browser/OS dependent keystroke, but generally this works:

- Windows: Ctrl+F5
- Mac: Cmd+Shift+R
- Linux: Ctrl+Shift+R

**HARD RESET DE SERVER amb control F5 al navegador**

ahora creamos un bloque para introducir texto y un botón para guardar esos datos que introducimos. como no estoy seguro de como hacerlo lo voy a simplificar y una vez funcione perfectamente lo voy a replicar y ponerlo bonito.

para el texto sera un input, podremos un atributo ""name='content'"" para que nuestro programa lo pueda recibir y a partir de ahí guarde los datos en nuestra base de datos. se puede llamar como queramos.

como usamos sqlalchemy vamos a modelar los datos antes de guardarlos.

```
<form action="/create-task" method="POST">
  <div class="form-group">
    <input type="text" name="content" placeholder="Task" class="form-control" autofocus>
  </div>
  <button type="submit" class="btn btn-primary btn-block">Save</button>
</form>
</div>
```

el objeto request que lo

una vez estamos en la ruta /add\_ships/ llamaremos a la función. una vez estamos dentro la variable ship es una instancia de la clase ships (un objeto)

una vez dentro diremos que el "name" es el de la clase de Ship el qual guardaremos la info q no llega por el html. entonces el name lo definimos en que sea igual a [content] que hemos definido en el html y que por defecto active sea false(es una booleana). para usar el request lo tenemos que importar de flask.

Una vez tenemos todo añadimos a el objeto ship a nuestra base de datos y le hacemos commit.

```
from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy
#from flask_marshmallow import Marshmallow
```

```
@app.route('/add_ship/', methods=['POST'])
def add_ship():
    ship = Ship(name=request.form['content'], active=False)
    #la variable "ship" es una instancia de la clase, un objeto
    db.session.add(ship)
    db.session.commit()
    return 'esta guardado'
```

```
<div class="card-header">
  <form action="/add_ship/" method="POST">
    <div class="form-group">
      <input type="text" name="content" placeholder="Task" class="form-control" autofocus>
    </div>
    <button type="submit" class="btn btn-primary btn-block">Save</button>
```

```
class Ship(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(200))
    active = db.Column(db.Boolean)
```

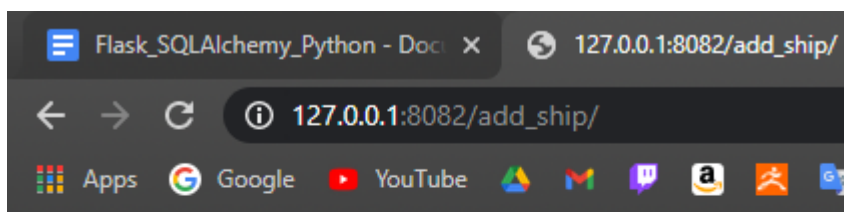
Como tengo varios problemas primero voy a hacerlo con la base de datos guardada de manera local y después lo cambio. En la terminal ejecutamos esto y con `.tables` nos mostrará qué tablas tenemos creadas, en nuestro caso ninguna.

```
PS E:\ASIX\m4\flask\ships_02> sqlite3 .\database\warships.db
SQLite version 3.34.1 2021-01-20 14:10:07
Enter ".help" for usage hints.
sqlite> .tables
sqlite> █
```

tenemos una clase `ship` la cual sí a través de `py` la ejecutamos nos creará la tabla en nuestra base de datos.

```
PS E:\ASIX\m4\flask\ships_02> py
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> from app import db
>>> db.create_all()
>>>
KeyboardInterrupt
>>> exit()
PS E:\ASIX\m4\flask\ships_02> sqlite3 .\database\warships.db
SQLite version 3.34.1 2021-01-20 14:10:07
Enter ".help" for usage hints.
sqlite> .tables
ship
sqlite> █
```

cuando escribimos algo en el texto y le damos al botón, se ejecuta todo esto y si consultamos la base de datos podremos ver que se ha guardado.



esta guardado

```

PS E:\ASIX\m4\flask\ships_02> sqlite3 .\database\warships.db
SQLite version 3.34.1 2021-01-20 14:10:07
Enter ".help" for usage hints.
sqlite> select * from ship;
1|aaa|0
2|hola|0

```

si consultamos la base de datos desde la terminal podremos ver que lo q hemos introducido se ha guardado, vemos el id que es incremental, el name y si esta activo o no, que por defecto no lo esta.

ahora vamos a pedir datos:

lo primero que haremos sera asignar a una variable para que podremos consultar todos los datos que tengamos.

en la ruta tenemos que especificar la variable como tal. el cuadro verde podemos asignarle el valor que queremos.

```

@app.route('/')
def index():
    ships = Ship.query.all()
    return render_template('index.html', ships = ships)

```

ahora que tenemos todos los datos en el html podremos hacer consultas con el {}

tenemos que importar el redirect de flask antes. al inicio.

```

@app.route('/add_ship/', methods=['POST'])
def add_ship():
    ship = Ship(name=request.form['content'], types=request.form['content2'], active=False)
    #la variable "ship" es una instancia de la clase, un objeto

    db.session.add(ship)
    db.session.commit()
    return redirect(url_for('index'))

```

usare template inheritance para tal tener una estructura mejor definida y no repetir código porque si. En el caso de que tenga que modificar algún detalle solo lo tendré que hacer en el template y no en todos.

## TEMPLATE

```
index.html base_template.html x tiers.html
templates > base_template.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5
6 <meta charset="UTF-8">
7 <meta http-equiv="X-UA-Compatible" content="IE=edge">
8 <meta name="viewport" content="width=device-width, initial-scale=1.0">
9 <title>{% block title %}{% endblock %}</title>
10
11 <!-- Bootstrap YETI -->
12 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootswatch/4.5.2/yeti/
bootstrap.min.css" integrity="sha384-mLBxp+1RMvmQmXOjBzRjqqr0dP9VHU2tb3FK6VB0fJN/AOu7/y
+CAeYeWJZ4b3ii" crossorigin="anonymous">
13
14 <!-- CSS -->
15 <link rel="stylesheet" href="{{url_for('static', filename='main.css')}}">
16
17 <!-- Favicon -->
18 <link href="https://favicon-generator.org/favicon-generator/htdocs/favicons/2015-01-08/
d892392af0b9610eb1260840c73a0116.ico" rel="icon">
19
20 <!-- Google fonts -->
21 <link href="https://fonts.googleapis.com/css2?family=Raleway:wght@340&display=swap"
rel="stylesheet">
22
23 </head>
24
25 <body>
26 {% block body %}{% endblock %}
27 </body>
28
29 </html>
```

