



Advertisement

[🏠](#) > [Code](#) > [Coding Fundamentals](#) > [Databases & SQL](#)

# Creando una Aplicación Web Desde Cero Usando Python Flask y MySQL

**Jay**

Feb 26, 2022 • 11 min read

**Español** ▾

Coding Fundamentals

Databases &amp; SQL

Python

This post is part of a series called [Creating a Web App From Scratch Using Python Flask and MySQL](#).

▶▶ [Creating a Web App From Scratch Using Python Flask and MySQL: Part 2](#)

(<sup>i</sup>) translation by (you can also [view the original English article](#))

# Creando una Aplicación Web Desde Cero Usando



# WEB DESDE CERO USANDO Python Flask y MySQL

En estas series, vamos a estar usando [Python](#), [Flask](#) y [MySQL](#) para crear una web aplicación simple desde cero. Será un simple aplicación bucket list donde los usuarios pueden registrarse, iniciar sesión y crear su bucket list.

Este tutorial asume que tienes conocimiento básico del lenguaje de programación `Python`. Estaremos usando `Flask`, un framework de aplicación para Python para crear nuestra aplicación, con `MySQL` como back-end.

## Introducción a Python Flask

Flask es un framework de Python para crear aplicaciones web. Desde el sitio oficial,

Flask es un microframework para Python basado en Werkzeug, Jinja 2 y en buenas intenciones.

Cuando pensamos en Python, el framework de facto que viene a nuestra mente es el framework [Django](#). Pero desde una perspectiva de principiantes de Python, Flask es mas fácil para comenzar que en comparación con Django.

## Instalando Flask



Instalar Flask es bastante simple y rápido. Con el administrador de paquetes `pip` todo lo que necesitamos hacer es:

```
1 | pip install flask
```

Una vez que hayas terminado de instalar Flask, crea una carpeta llamada `FlaskApp`. Navega a la carpeta `FlaskApp` y crea un archivo llamado `app.py`. Importa el módulo `flask` y crea una aplicación usando Flask como se muestra:

```
1 | from flask import Flask
2 | app = Flask(__name__)
```

Ahora define la ruta básica `/` y su correspondiente manejador de solicitud.

```
1 | @app.route("/")
2 | def main():
3 |     return "Welcome!"
```

Enseguida, revisa si el archivo ejecutado es el programa principal y ejecuta la aplicación:

```
1 | if __name__ == "__main__":
2 |     app.run()
```

Guarda los cambios y ejecuta `app.py`:

```
1 | python app.py
```

Apunta tu navegador a <http://localhost:5000/> y deberías de tener el mensaje de bienvenida.



## Creando una Página Principal

## Creando una Página Principal

Primero, cuando la aplicación se ejecuta deberíamos de mostrar una página principal con los últimos elementos de la bucket list. Así que vamos a agregar nuestra página principal a la carpeta de nuestra aplicación.

Flask busca archivos de plantilla dentro de la carpeta `templates`. Entonces navega a la carpeta `PythonApp` y crea una carpeta llamada `templates`. Dentro de `templates`, crea un archivo llamado `index.html`. Abre `index.html` y agrega el siguiente HTML:

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <title>Python Flask Bucket List App</title>
6
7
8      <link href="http://getbootstrap.com/dist/css/bootstrap.min.css"
9
10     <link href="http://getbootstrap.com/examples/jumbotron-narrow/ju
11
12
13 </head>
14
15 <body>
16
17     <div class="container">
18         <div class="header">
19             <nav>
20                 <ul class="nav nav-pills pull-right">
21                     <li role="presentation" class="active"><a href=";"
22                     </li>
23                     <li role="presentation"><a href="#">Sign In</a>
24                     </li>
25                     <li role="presentation"><a href="showSignUp">Sig
26                     </li>
27                 </ul>
28             </nav>
29             <h3 class="text-muted">Python Flask App</h3>
30         </div>
31
32         <div class="jumbotron">
33             <h1>Bucket List App</h1>
34             <p class="lead"></p>
35             <p><a class="btn btn-lg btn-success" href="showSignUp"
36             </p>
37         </div>
38
```



```
39         <div class="row marketing">
40             <div class="col-lg-6">
41                 <h4>Bucket List</h4>
42                 <p>Donec id elit non mi porta gravida at eget metus.
43
44                 <h4>Bucket List</h4>
45                 <p>Morbi leo risus, porta ac consectetur ac, vestibulu
46
47                 <h4>Bucket List</h4>
48                 <p>Maecenas sed diam eget risus varius blandit sit a
49             </div>
50
51             <div class="col-lg-6">
52                 <h4>Bucket List</h4>
53                 <p>Donec id elit non mi porta gravida at eget metus.
54
55                 <h4>Bucket List</h4>
56                 <p>Morbi leo risus, porta ac consectetur ac, vestibulu
57
58                 <h4>Bucket List</h4>
59                 <p>Maecenas sed diam eget risus varius blandit sit a
60             </div>
61         </div>
62
63         <footer class="footer">
64             <p>&copy; Company 2015</p>
65         </footer>
66
67     </div>
68 </body>
69
70 </html>
```

Abre `app.py` e importa `render_template`, el cual usaremos para renderizar los archivos de plantilla.

```
1 | from flask import Flask, render_template
```

Modifica el método principal para devolver el archivo plantilla renderizado.

```
1 | def main():
2 |     return render_template('index.html')
```

Guarda los cambios y reinicia el servidor. Apunta tu navegador a <http://localhost:5000/> y deberías de tener la pantalla de abajo:



Python Flask App

Home

Sign In

## Bucket List App

Sign up today

### Bucket List

Donec id elit non mi porta gravida at eget metus. Maecenas faucibus mollis interdum.

### Bucket List

Morbi leo risus, porta ac consectetur ac, vestibulum at eros. Cras mattis consectetur purus sit amet fermentum.

## Creando una Página de Registro

Advertisement



## Paso 2: Instalando la Base de Datos

Estaremos usando `MySQL` como back-end. Así que inicia sesión a MySQL desde la línea de comandos, o si lo prefieres en una GUI como [MySQL work bench](#), puedes usar eso también. Primero, crea la base de datos llamada `BucketList`. Desde la línea de comandos:

```
1 | mysql -u <username> -p
```

Ingresa la contraseña requerida y cuando hayas iniciado sesión, ejecuta el siguiente comando y crea la base de datos:

```
1 | CREATE DATABASE BucketList;
```

Una vez que la base de datos ha sido creada, crea una tabla llamada `tbl_user` como se muestra:

```
1 | CREATE TABLE `BucketList`.`tbl_user` (  
2 |   `user_id` BIGINT NULL AUTO_INCREMENT,  
3 |   `user_name` VARCHAR(45) NULL,  
4 |   `user_username` VARCHAR(45) NULL,  
5 |   `user_password` VARCHAR(45) NULL,  
6 |   PRIMARY KEY (`user_id`));
```

Estaremos usando `Stored procedures` para nuestra aplicación de Python para interactuar con la base de datos MySQL. Así que, una vez que la tabla `tbl_user` ha sido creada, crea un procedimiento almacenado llamado `sp_createUser` para registrar a un usuario.

Cuando se esté creando un procedimiento almacenado para crear un usuario en la tabla `tbl_user`, primero necesitamos revisar si un usuario con el mismo `username` ya existe. Si existe necesitamos tirar un error al usuario, de lo contrario crearemos el usuario en la tabla de usuario. Aquí está como el procedimiento almacenado `sp_createUser` se vería:



```
1 DELIMITER $$
2 CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_createUser` (
3     IN p_name VARCHAR(20),
4     IN p_username VARCHAR(20),
5     IN p_password VARCHAR(20)
6 )
7 BEGIN
8     if ( select exists (select 1 from tbl_user where user_username = p_username) )
9     then
10         select 'Username Exists !!';
11     else
12         insert into tbl_user
13         (
14             user_name,
15             user_username,
16             user_password
17         )
18         values
19         (
20             p_name,
21             p_username,
22             p_password
23         );
24     end if;
25 END IF;
26 END$$
27 DELIMITER ;
```

## Paso 2: Crea una Interfaz de Registro

Navega al directorio `PythonApp/templates` y crea un archivo HTML llamado `signup.html`. Agrega el siguiente código HTML a `signup.html`:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title>Python Flask Bucket List App</title>
5
6     <link href="http://getbootstrap.com/dist/css/bootstrap.min.css"
7
8     <link href="http://getbootstrap.com/examples/jumbotron-narrow/jui
9     <link href="../static/signup.css" rel="stylesheet">
10
11   </head>
12
13   <body>
```





```

15
16     <div class="container">
17         <div class="header">
18             <nav>
19                 <ul class="nav nav-pills pull-right">
20                     <li role="presentation" ><a href="main">Home</a></li>
21                     <li role="presentation"><a href="#">Sign In</a></li>
22                     <li role="presentation" class="active"><a href="#">Sign I
23                 </ul>
24             </nav>
25             <h3 class="text-muted">Python Flask App</h3>
26         </div>
27
28         <div class="jumbotron">
29             <h1>Bucket List App</h1>
30             <form class="form-signin">
31                 <label for="inputName" class="sr-only">Name</label>
32                 <input type="name" name="inputName" id="inputName" class="fo
33                 <label for="inputEmail" class="sr-only">Email address</label>
34                 <input type="email" name="inputEmail" id="inputEmail" class=
35                 <label for="inputPassword" class="sr-only">Password</label>
36                 <input type="password" name="inputPassword" id="inputPasswor
37
38                 <button id="btnSignUp" class="btn btn-lg btn-primary btn-blo
39             </form>
40         </div>
41
42
43
44         <footer class="footer">
45             <p>&copy; Company 2015</p>
46         </footer>
47
48     </div>
49 </body>
50 </html>

```

También agrega el siguiente `CSS` como `signup.css` a la carpeta estática dentro de `PythonApp`.

```

1  body {
2      padding-top: 40px;
3      padding-bottom: 40px;
4  }
5
6  .form-signin {
7      max-width: 330px;
8      padding: 15px;
9      margin: 0 auto;
10 }
11 .form-signin .form-signin-heading,
12 .form-signin .checkbox {
13     margin-bottom: 10px;
14 }
15 .form-signin .checkbox {

```

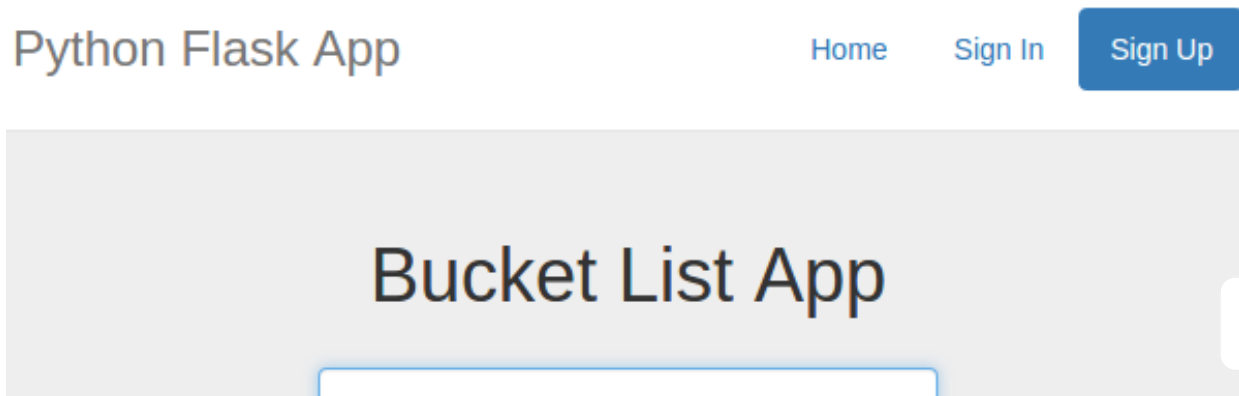


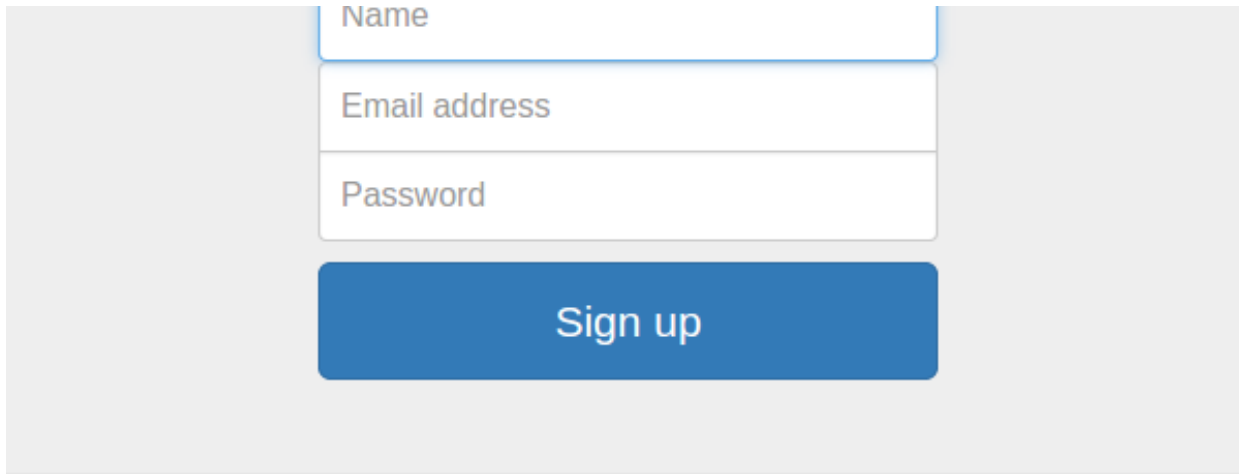
```
15 | .form-signin .form-control {
16 |     font-weight: normal;
17 | }
18 | .form-signin .form-control {
19 |     position: relative;
20 |     height: auto;
21 |     -webkit-box-sizing: border-box;
22 |     -moz-box-sizing: border-box;
23 |     box-sizing: border-box;
24 |     padding: 10px;
25 |     font-size: 16px;
26 | }
27 | .form-signin .form-control:focus {
28 |     z-index: 2;
29 | }
30 | .form-signin input[type="email"] {
31 |     margin-bottom: -1px;
32 |     border-bottom-right-radius: 0;
33 |     border-bottom-left-radius: 0;
34 | }
35 | .form-signin input[type="password"] {
36 |     margin-bottom: 10px;
37 |     border-top-left-radius: 0;
38 |     border-top-right-radius: 0;
39 | }
```

En `app.py` agrega otro método llamado `showSignUp` para renderizar la página de registro una vez que una solicitud llegue a `/showSignUp`:

```
1 | @app.route('/showSignUp')
2 | def showSignUp():
3 |     return render_template('signup.html')
```

Guarda los cambios y reinicia el servidor. Haz clic en el botón **Sign Up** en la página principal y deberías de tener la página de registro como se muestra:





A registration form with three input fields labeled "Name", "Email address", and "Password". Below the fields is a blue button labeled "Sign up".

© Company 2015

## Paso 3: Implementa un Método de Registro

Enseguida, necesitamos un método del lado del servidor para que la UI interactue con la base de datos MySQL. Así que navega a `PythonApp` y abre `app.py`. Crea un nuevo método llamado `signUp` y también agrega una ruta `/signUp`. Así es como se ve:

```
1 | @app.route('/signUp')
2 | def signUp():
3 |     # create user code will be here !!
```

Estaremos usando [jQuery AJAX](#) para enviar nuestros datos de registro al método `signUp`, así que vamos a especificar el método en la definición de la ruta.

```
1 | @app.route('/signUp', methods=['POST'])
2 | def signUp():
3 |     # create user code will be here !!
```

Para leer los valores enviados necesitamos importar `request` desde Flask.

```
1 | from flask import Flask, render_template, request
```



```
1 | from flask import Flask, render_template, request
```

Usando `request` podemos leer los valores publicados como se muestra bajo:

```
1 | @app.route('/signUp',methods=['POST'])
2 | def signUp():
3 |
4 |     # read the posted values from the UI
5 |     _name = request.form['inputName']
6 |     _email = request.form['inputEmail']
7 |     _password = request.form['inputPassword']
```

Una vez que los valores son leídos, simplemente vamos a revisar si son válidos y mientras tanto vamos solo a devolver un mensaje simple:

```
1 | @app.route('/signUp',methods=['POST'])
2 | def signUp():
3 |
4 |     # read the posted values from the UI
5 |     _name = request.form['inputName']
6 |     _email = request.form['inputEmail']
7 |     _password = request.form['inputPassword']
8 |
9 |     # validate the received values
10 |    if _name and _email and _password:
11 |        return json.dumps({'html':'<span>All fields good !!</span>'})
12 |    else:
13 |        return json.dumps({'html':'<span>Enter the required fields</span>'})
```

También importa `json` desde Flask, ya que lo estamos usando en el código anterior para devolver datos `json`.

```
1 | from flask import Flask, render_template, json, request
```

Advertisement



## Paso 4: Crea una Solicitud de Registro

Estaremos usando [jQuery](#) AJAX para enviar la solicitud de registro al método de Python. [Descarga](#) y pon `jQuery` dentro de `PythonApp/static/js` y agrega un link a él desde la página de registro. Una vez que jQuery ha sido incluido, vamos a agregar una solicitud `POST` de jQuery cuando el usuario haga clic al botón `Sign Up`.

Vamos a adjuntar el evento del botón de registro como se muestra:

```
1 | $(function() {  
2 |     $('#btnSignUp').click(function() {  
3 |  
4 |         $.ajax({  
5 |             url: '/signUp',  
6 |             data: $('form').serialize(),  
7 |             type: 'POST',  
8 |             success: function(response) {  
9 |                 console.log(response);  
10 |             },  
11 |             error: function(error) {  
12 |                 console.log(error);  
13 |             }  
14 |         });  
15 |     });  
16 | });
```

Guarda todos los cambios y reinicia el servidor. Desde la página **Sign Up**, llena los detalles y haz clic en **Sign Up**. Revisa la consola del navegador y deberías de tener el mensaje de abajo:

```
1 | { "html": "<span>All fields good !!</span>" }
```

## Paso 5: Llama al Procedimiento Almacenado



## de MySQL

Una vez que tengamos `name`, `email address` y `password`, podemos simplemente llamar al procedimiento almacenado MySQL para crear el nuevo usuario.

Para conectarse con MySQL, vamos a estar usando [Flask-MySQL](#), el cual es una extensión de Flask. Para comenzar con `Flask-MySQL`, haz la instalación usando el administrador de paquetes `pip`:

```
1 | pip install flask-mysql
```

Importa MySQL dentro de `app.py`:

```
1 | from flask.ext.mysql import MySQL
```

Mas temprano definimos nuestra aplicación como se muestra:

```
1 | app = Flask(__name__)
```

Junto con eso incluye las siguientes configuraciones de MySQL:

```
1 | mysql = MySQL()  
2 |  
3 | # MySQL configurations  
4 | app.config['MYSQL_DATABASE_USER'] = 'jay'  
5 | app.config['MYSQL_DATABASE_PASSWORD'] = 'jay'  
6 | app.config['MYSQL_DATABASE_DB'] = 'BucketList'  
7 | app.config['MYSQL_DATABASE_HOST'] = 'localhost'  
8 | mysql.init_app(app)
```

Primero, vamos a crear la conexión de MySQL:

```
1 | conn = mysql.connect()
```

Una vez que la conexión es creada, vamos a requerir un `cursor` para solicitar nuestro procedimiento almacenado. Usando conexión `conn`

crear un cursor

crea un cursor.

```
1 | cursor = conn.cursor()
```

Antes de llamar al procedimiento almacenado para crear usuarios, vamos a salar nuestra contraseña usando una utilidad proveída por [Werkzeug](#). Importa el módulo dentro de `app.py`:

```
1 | from werkzeug import generate_password_hash, check_password_hash
```

Usa el modulo de salado para crear la contraseña en hash.

```
1 | _hashed_password = generate_password_hash(_password)
```

Ahora, vamos a llamar al procedimiento `sp_createUser`:

```
1 | cursor.callproc('sp_createUser',(_name,_email,_hashed_password))
```

Si el procedimiento es ejecutado exitosamente, entonces vamos a agregar los cambios y devolveremos el mensaje de éxito.

```
1 | data = cursor.fetchall()
2 |
3 | if len(data) is 0:
4 |     conn.commit()
5 |     return json.dumps({'message':'User created successfully !'})
6 | else:
7 |     return json.dumps({'error':str(data[0])})
```

Guarda los cambios y reinicia el servidor. Ve a la página de registro e ingresa el `name`, `email address` y `password` y haz clic en el botón **Sign Up**. Al ser exitosa la creación de usuario, serás capaz de ver un mensaje en la consola de tu navegador.

```
1 | {"message": "User created successfully !"}
```



# En Resumen

En este tutorial, vimos como comenzar con la creación de una aplicación usando `Python Flask`, `MySQL` y la extensión `Flask-MySQL`. Hemos creado y diseñado las tablas de la base de datos y el procedimiento almacenado, e implementado la funcionalidad de registro. En el siguiente tutorial, vamos a tomar esta serie al siguiente nivel implementando funcionalidad de inicio de sesión y algunas otras características.

El código fuente de este tutorial está disponible en [GitHub](#).

¡Déjanos saber lo que piensas en los comentarios de abajo!

Did you find this post useful?



Yes



No

## Want a weekly email summary?

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

**Sign up**





## Jay

Software engineer by profession and writer by choice. I'm a great fan of everything JavaScript.

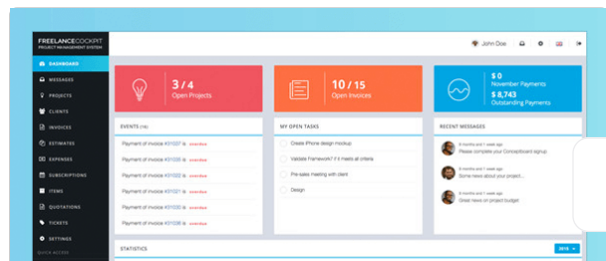



[View on GitHub](#)

Advertisement

**LOOKING FOR SOMETHING TO HELP KICK START YOUR  
NEXT PROJECT?**


**Envato Market** has a range of items  
for sale to help get you started.





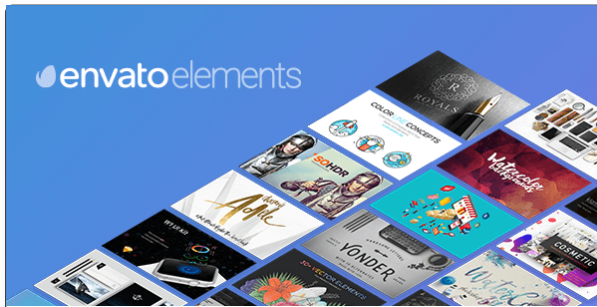
**WordPress Plugins**

From \$5



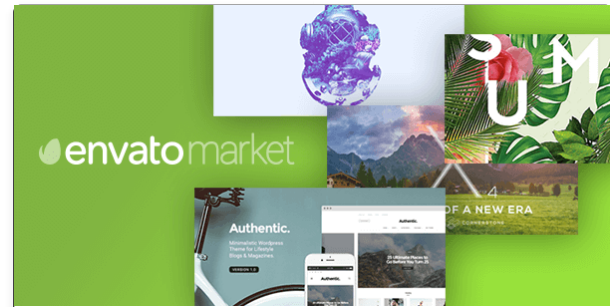
**PHP Scripts**

From \$5



**Unlimited Downloads**  
From \$16.50/month

Get access to over one million creative assets on Envato Elements.



**Over 9 Million Digital Assets**

Everything you need for your next creative project.

**QUICK LINKS** - Explore popular categories

## ENVATO TUTORIALS

About Envato Tuts+  
Terms of Use  
Advertise

## HELP

FAQ  
Help Center



tuts+

30,347  
Tutorials

553  
Courses

42,531  
Translations



---

[Envato](#) [Envato Elements](#) [Envato Market](#) [Placeit by Envato](#) [All](#)

[products](#) [Careers](#) [Sitemap](#)

© 2023 Envato Pty Ltd. Trademarks and brands are the property of their respective owners.

