

# Tutorial Flask – Lección 4: Login de usuarios en Flask

📌 Categoría: Flask (<https://j2logo.com/category/blog/flask/>)

🔗 flask (<https://j2logo.com/tag/flask/>), medio (<https://j2logo.com/tag/medio/>), python (<https://j2logo.com/tag/python/>), tutorial flask (<https://j2logo.com/tag/tutorial-flask/>)

## ¿Quieres ser expert@ en Python?

He ayudado a miles de programadores/as como tú a ser mejores Pythonistas

Recibe de vez en cuando **trucos, scripts y tutoriales Python** en español

**Accede a nuestra comunidad privada de Slack: Pythonistas-es**

Nombre

(Requerido) Email

Quiero ser Pythonista

Conviértete en maestr@ Pythonista

Al enviar el formulario confirmas que aceptas la POLÍTICA DE PRIVACIDAD (<https://j2logo.com/politica-de-privacidad/>)


adecuadamente, ya que, en gran medida, de él depende parte de la seguridad de nuestra aplicación. Pero no te preocupes, como verás, hacer el login de usuarios en Flask es muy sencillo.

En esta lección crearemos nuestro modelo que representa a los usuarios de la aplicación y el formulario de login. También veremos cómo hacer login en Flask y cómo proteger ciertas vistas de aquellos usuarios que no se han autenticado.

Continuaremos por donde lo dejamos en la lección anterior, en la que te expliqué cómo usar formularios en Flask (<https://j2logo.com/tutorial-flask-leccion-3-formularios-wtforms/>). De hecho repasaremos parte de esa lección ya que tendremos que **añadir el formulario de login** al blog.

En fin, ya no te entretengo más y doy paso a la acción 😊

## !! ATENCIÓN !!

 Puedes descargar el código correspondiente a la Lección 3 desde el siguiente repositorio de Github:

```
git clone https://github.com/j2logo/tutorial-flask.git  
git checkout tags/leccion3 -b leccion3
```

# Índice

A continuación te muestro el índice de esta lección:

- La función para cargar el modelo
- Login de usuarios
- Logout
- Personalizando el login
- Protegiendo las vistas
- Mostrando la información del usuario logueado en las plantillas

## Introducción a Flask-login

Para implementar el login de usuarios en Flask haremos uso de una conocida extensión llamada *Flask-login* (<https://flask-login.readthedocs.io>). Siempre que se pueda, no hay que reinventar la rueda y esta extensión nos facilitará mucho la vida. ¿Qué nos ofrece Flask-login? Entre otras cosas:

- Almacenar el ID del usuario en la sesión y mecanismos para hacer *login* y *logout*.
- Restringir el acceso a ciertas vistas únicamente a los usuarios autenticados.
- Gestionar la funcionalidad *Recuérdame* para mantener la sesión incluso después de que el usuario cierre el navegador.
- Proteger el acceso a las cookies de sesión frente a terceros.

El primer paso para usar *Flask-login* en nuestra aplicación será instalarla. Para ello, ejecutaremos en la consola lo siguiente:

```
1. pip install flask-login
```

Dado que *Flask-login* hace uso de la sesión para la autenticación, debemos establecer la variable de configuración `SECRET_KEY`. Como recordarás, esto ya lo hicimos en la lección anterior (<https://j2logo.com/tutorial-flask-leccion-3-formularios-wtforms/>).

Vamos a crear un objeto de la clase `LoginManager` que llamaremos `login_manager`. Abre el fichero `run.py` y justo después de instanciar la `app` añade lo siguiente:

```
1. ...
2.
3. from flask_login import LoginManager
4.
5. ...
6.
7. app = Flask(__name__)
8. app.config['SECRET_KEY'] = '7110c8ae51a4b5af97be6534caef90e4bb9bdcb3380af008f90b23a5d1616bf319bc298105da20fe'
9.
10. login_manager = LoginManager(app)
11.
12. ...
```

Con esto habremos dado el primer paso para que los usuarios puedan hacer login en nuestra aplicación Flask.

## Crear el modelo User

Lo siguiente que haremos será crear la clase `User`. Esta clase representa a los usuarios de nuestra aplicación. Además, contiene toda la lógica para crear usuarios, guardar las contraseñas de modo seguro o verificar los passwords.

Un punto a favor de la extensión *Flask-login* es que te da libertad para definir tu clase para los usuarios. Esto hace posible que se pueda utilizar cualquier sistema de base de datos y que modifiquemos el modelo en función de las necesidades que vayan surgiendo. El único requisito indicado por *Flask-login* es que la clase usuario debe implementar las siguientes propiedades y

- `is_active`: una propiedad que indica si la cuenta del usuario está activa (`True`) o no (`False`). Es decisión tuya definir qué significa que una cuenta de usuario está activa. Por ejemplo, se ha verificado el email o no ha sido eliminada por un administrador. Por defecto, los usuarios de cuentas inactivas no pueden autenticarse.
- `is_anonymous`: una propiedad que vale `False` para los usuarios reales y `True` para los usuarios anónimos.
- `get_id()`: un método que devuelve un `string` (`unicode` en caso de Python 2) con el `ID` único del usuario. Si el `ID` del usuario fuera `int` o cualquier otro tipo, es tu responsabilidad convertirlo a `string`.

De nuevo, *Flask-login* ha pensado en nosotros ya que pone a nuestra disposición la clase `UserMixin` con una implementación por defecto para todas estas propiedades y métodos. Tan solo tenemos que heredar de ella en nuestra propia clase `User`.

Con esto en mente, crea un nuevo fichero llamado `models.py` en el directorio raíz del proyecto y añade la clase siguiente:

```
1. from flask_login import UserMixin
2. from werkzeug.security import generate_password_hash, check_password_hash
3.
4.
5. class User(UserMixin):
6.
7.     def __init__(self, id, name, email, password, is_admin=False):
8.         self.id = id
9.         self.name = name
10.        self.email = email
11.        self.password = generate_password_hash(password)
12.        self.is_admin = is_admin
13.
14.    def set_password(self, password):
15.        self.password = generate_password_hash(password)
16.
17.    def check_password(self, password):
18.        return check_password_hash(self.password, password)
19.
20.    def __repr__(self):
21.        return '<User {}>'.format(self.email)
```

seguridad. En su lugar, guardaremos un hash del password. Para ello, nos valdremos de la librería `werkzeug.security`, aunque puedes usar cualquier otra (siempre que sea segura).

Para verificar la contraseña, hemos definido el método `check_password` que comprueba si el hash del parámetro `password` coincide con el del usuario.

## Listado de usuarios

Como ha sucedido en otras lecciones anteriores, todavía no guardaremos los usuarios en base de datos (lo veremos por fin en la siguiente lección 🎉). En su lugar, haremos uso de una lista de usuarios almacenada en memoria que llamaremos `users` (esto no es útil ya que los usuarios que guardemos se borrarán al reiniciar el servidor pero nos servirá para esta lección).

Añade lo siguiente después de la clase `User`:

```
1. users = []
2.
3.
4. def get_user(email):
5.     for user in users:
6.         if user.email == email:
7.             return user
8.     return None
```

La función `get_user` la utilizaremos provisionalmente para buscar un usuario por su email dentro de la lista `users`.

## La función para cargar el modelo

¿Cómo podemos acceder en nuestro código al usuario cuyo ID se encuentra almacenado en sesión? Fácil, implementando un callback que será llamado por el método `user_loader` del objeto `login_manager`. ¿Recuerdas que definimos este objeto al principio?

Conviértete en maestro@Pythonista

Añadamos nuestro callback al final del fichero `run.py`:

```
1. @login_manager.user_loader
2. def load_user(user_id):
3.     for user in users:
4.         if user.id == int(user_id):
5.             return user
6.     return None
```

`users` hace referencia a la lista de usuarios que definimos en el módulo `models.py`. Recuerda importarla previamente antes de hacer uso de ella.

```
1. from models import users
```

## Login de usuarios

Bueno, ya lo tenemos todo preparado para poder hacer el login 🍻🤪

Ahora toca el turno de crear el formulario para que los usuarios de nuestro blog se puedan autenticar (para poder comentar los posts). Vamos a dividir este proceso en tres fases: crear la clase del formulario, crear la plantilla HTML e implementar la vista que realiza el login.

## Clase para el formulario de login

A estas alturas ya eres todo un expert@ en la materia, por tanto no voy a entrar mucho en detalle de cómo crear una clase que representa un formulario.

Abre el fichero `forms.py` y añade el código siguiente al final del mismo:

El campo `remember_me` es de tipo `BooleanField`. Deberás importarlo junto al resto de tipos que importamos en la lección anterior. Lo utilizaremos para dar la posibilidad al usuario de mantener la sesión incluso después de cerrar el navegador.

## Plantilla HTML para el formulario

Crea una nueva página HTML llamada `login_form.html` dentro de la carpeta `templates` del proyecto.

El contenido de la misma será el siguiente:

```
1. {% extends "base_template.html" %}
2.
3. {% block title %}Login{% endblock %}
4.
5. {% block content %}
6.     <div>
7.         <form action="" method="post" novalidate>
8.             {{ form.hidden_tag() }}
9.             <div>
10.                 {{ form.email.label }}
11.                 {{ form.email }}<br>
12.                 {% for error in form.email.errors %}
13.                 <span style="color: red;">{{ error }}</span>
14.                 {% endfor %}
15.             </div>
16.             <div>
17.                 {{ form.password.label }}
18.                 {{ form.password }}<br>
19.                 {% for error in form.password.errors %}
20.                 <span style="color: red;">{{ error }}</span>
21.                 {% endfor %}
22.             </div>
23.             <div>{{ form.remember_me() }} {{ form.remember_me.label }}</div>
24.             <div>
25.                 {{ form.submit() }}
26.             </div>
27.         </form>
28.     </div>
```



## La vista para realizar el login

Por último, debemos implementar la vista que muestre el formulario de login y compruebe si las credenciales proporcionadas por el usuario son válidas o no. Añade la siguiente función al final del fichero `run.py`:

```

1. from werkzeug.urls import url_parse
2.
3. @app.route('/login', methods=['GET', 'POST'])
4. def login():
5.     if current_user.is_authenticated:
6.         return redirect(url_for('index'))
7.     form = LoginForm()
8.     if form.validate_on_submit():
9.         user = get_user(form.email.data)
10.        if user is not None and user.check_password(form.password.data):
11.            login_user(user, remember=form.remember_me.data)
12.            next_page = request.args.get('next')
13.            if not next_page or url_parse(next_page).netloc != '':
14.                next_page = url_for('index')
15.            return redirect(next_page)
16.        return render_template('login_form.html', form=form)

```

Voy a ir desgranando poco a poco lo que hace esta vista:

- En primer lugar comprobamos si el usuario actual ya está autenticado. Para ello nos valemos de la instancia `current_user` de *Flask-login*. El valor de `current_user` será un objeto usuario si este está autenticado (el que se obtiene en el callback `user_loader`) o un usuario anónimo en caso contrario. Si el usuario ya está autenticado no tiene sentido que se vuelva a loguear, por lo que lo redirigimos a la página principal.

- A continuación comprobamos si los datos enviados en el formulario son válidos. En ese caso, intentamos recuperar el usuario a partir del email con `get_user()`.

página protegida pero no estaba autenticado. Por temas de seguridad, solo tendremos en cuenta dicho parámetro si la ruta es relativa. De este modo evitamos redirigir al usuario a un sitio fuera de nuestro dominio. Si no se recibe el parámetro `next` o este no contiene una ruta relativa, redirigimos al usuario a la página de inicio.

También he realizado unas pequeñas modificaciones en la vista que realiza el registro para ir añadiendo a los usuarios registrados a la lista `users`. Puedes ver los cambios a continuación:

```
1. @app.route("/signup/", methods=["GET", "POST"])
2. def show_signup_form():
3.     if current_user.is_authenticated:
4.         return redirect(url_for('index'))
5.     form = SignupForm()
6.     if form.validate_on_submit():
7.         name = form.name.data
8.         email = form.email.data
9.         password = form.password.data
10.        # Creamos el usuario y lo guardamos
11.        user = User(len(users) + 1, name, email, password)
12.        users.append(user)
13.        # Dejamos al usuario logueado
14.        login_user(user, remember=True)
15.        next_page = request.args.get('next', None)
16.        if not next_page or url_parse(next_page).netloc != '':
17.            next_page = url_for('index')
18.        return redirect(next_page)
19.    return render_template("signup_form.html", form=form)
```

Si has seguido todos los pasos hasta aquí, ya tendrías implementado un sistema de login en Flask. Sin embargo, todavía podemos mejorar el proceso. Para saber cómo, sigue leyendo hasta el final 😊

## Logout

Conviértete en maestr@ Pythonista

```
2. def logout():  
3.     logout_user()  
4.     return redirect(url_for('index'))
```

## ◀Personalizando el login

Si no queremos que la aplicación muestre un error 401 cuando un usuario intenta acceder a una vista protegida, hay que personalizar el objeto `login_manager`. En este caso, lo que haremos será indicarle cuál es la vista para realizar el login.

Añade lo siguiente después de crear el objeto `login_manager`:

```
1. login_manager = LoginManager(app)  
2. login_manager.login_view = "login"  
3.  
4. ...
```

Ahora el usuario será redirigido a la página de login en lugar de ver el error 401.

## Protegiendo las vistas

Como indiqué al inicio del tutorial, solo los usuarios administradores pueden crear entradas en el blog. Por tanto, la vista `post_form` debe ser accesible solo por este tipo de usuarios. Todavía es pronto para ver cómo hacer esto pero realizaremos una primera aproximación.

Un usuario administrador debe estar autenticado para poder crear entradas. La manera en que *Flask-login* permite proteger el acceso a las vistas solo a los usuarios autenticados es a través del decorador `@login_required`.

```
3. @login_required
4. def post_form(post_id):
5.     form = PostForm()
6.     if form.validate_on_submit():
7.
8.     ...
```

Puedes comprobar que si no estás autenticado e intentas acceder, la aplicación te redirigirá a la página de login.

## Mostrando la información del usuario logueado en las plantillas

Por último nos queda jugar con la información del usuario, tanto si está registrado como si no. Vamos a crear un menú superior para el blog en el que se muestre el nombre del usuario en caso de estar autenticado o un enlace para loguearse en caso contrario.

Abre la plantilla `base_template.html` y añade lo siguiente:

```
1. ...
2.
3. <body>
4. <div>
5.     <ul class="user-info">
6.         <li><a href="{{ url_for('index') }}">Home</a></li>
7.         {% if current_user.is_anonymous %}
8.             <li><a href="{{ url_for('login') }}">Login</a></li>
9.             <li> | </li>
10.            <li><a href="{{ url_for('show_signup_form') }}">Registrar</a></li>
11.        {% else %}
12.
13.            <li>{{ current_user.name }}</li>
14.            <li> | </li>
15.            <li><a href="{{ url_for('logout') }}">Logout</a></li>
16.        {% endif %}
17.    </ul>
18. </div>
```

Los estilos de la clase `user-info` son:

```
1. .user-info {  
2.     list-style: none;  
3.     margin: 0;  
4.     padding: 0;  
5. }  
6.  
7. .user-info li {  
8.     display: inline-block;  
9.     margin: 0;  
10.    padding: 0 10px 0 0;  
11. }
```

Añádelos al final del fichero de estilos `base.css`.

Y ahora sí, podemos dar por terminada esta lección del tutorial. Espero que te haya gustado 😊

## Conclusión

Como habrás podido comprobar, el login de usuarios es una cuestión delicada e importante y no hay que tomársela a la ligera. Por suerte, podemos hacer uso de la extensión Flask-login que nos facilita mucho las cosas, aunque eres libre de intentar hacer tu propio sistema de login siguiendo los principales aspectos comentados en esta lección.

En el siguiente tutorial veremos un tema importante que seguro lo estás deseando: Cómo integrar una base de datos relacional en una aplicación Flask.

¿Te ha gustado? Ayúdame a compartirlo 🙌



Facebook



Twitter



LinkedIn

¿Quieres ser expert@ en Python? Recibe trucos Python y accede a nuestro espacio privado de Slack

\* Al enviar el formulario confirmas que aceptas la POLITICA DE PRIVACIDAD (<https://j2logo.com/politica-de-privacidad/>)

¡Eyyy! Esto también te puede interesar 🙌



Conviértete en maestr@ Pythonista



en-flask/)

**Tutorial Flask – Lección 16: Procesar ficheros en Flask**  
(<https://j2logo.com/tutorial-flask-leccion-16-procesar-ficheros-en-flask/>)



(<https://j2logo.com/tutorial-flask-leccion-14-enviar-emails-con-flask/>)

**Tutorial Flask – Lección 14: Enviar emails con Flask**  
(<https://j2logo.com/tutorial-flask-leccion-14-enviar-emails-con-flask/>)

en-flask/)

**Tutorial Flask – Lección 15: Trabajar con Fechas en Flask**  
(<https://j2logo.com/tutorial-flask-leccion-15-trabajar-con-fechas-en-flask/>)



(<https://j2logo.com/tutorial-flask-leccion-13-pagar-las-consultas-de-base-de-datos/>)

**Tutorial Flask – Lección 13: Pagar las consultas de base de datos**  
(<https://j2logo.com/tutorial-flask-leccion-13-pagar-las-consultas-de-base-de-datos/>)



(<https://j2logo.com/tutorial-flask-leccion-12-tests-con-flask-unittest/>)

## Tutorial Flask – Lección 12: Tests con Flask y unittest

(<https://j2logo.com/tutorial-flask-leccion-12-tests-con-flask-unittest/>)

\* Te informo de que los datos de carácter personal que proporcionas al comentar serán tratados por **Juan José Lozano Gómez** como responsable de esta web. La **Finalidad** es moderar los comentarios. La **Legitimación** es gracias a tu consentimiento. **Destinatarios:** tus datos se encuentran alojados en Disqus (disqus.com), mi sistema de comentarios, que está acogido al acuerdo de seguridad EU-US Privacy. Podrás ejercer **Tus Derechos** de Acceso, Rectificación, Limitación o Suprimir tus datos enviando un email a [juanjo@j2logo.com](mailto:juanjo@j2logo.com) (<mailto:juanjo@j2logo.com>). Encontrarás más información en la POLÍTICA DE PRIVACIDAD ([politica-de-privacidad/](/politica-de-privacidad/)).

17 Comentarios [j2logo](#)  [Política de privacidad de Disqus](#)

 **Laura Dilcab** ▾

 [Recomendar](#)  [Tweet](#)  [Compartir](#)

[Ordenar por los mejores](#) ▾



Únete a la conversación...



**Eduardo Rodriguez Alviz** • hace 2 años

Hola,

Son excelentes los tutoriales, pero de casualidad no veo videos explicativos. Los tienes disponibles?

1 ^ | ▾ • [Responder](#) • [Compartir](#) >



**Juan José Lozano Gómez** Moderador ➔ Eduardo Rodriguez Alviz • hace 2 años

Hola Eduardo!

No, de momento no hay vídeos, solo los tutoriales. Es posible que en un futuro haga algún videotutorial o un webinar.

Conviértete en [maestro@Pythonista](#)

1 ^ | ▾ • [Responder](#) • [Compartir](#) >



Algo que quisiera pedir, de ser posible, ¿podrías hacer este mismo ejemplo sin sqlalchemy y trabajado con sql directamente? Sería de gran ayuda.

^ | v • Responder • Compartir >



**Juan José Lozano Gómez** Moderador ➔ samsepiol • hace 2 meses

Hola! Es cierto que llamo al método `get_user()` directamente. A veces, en los ejemplos, no muestro todos los imports que son necesarios en el código. Para ver el código completo de cada lección puedes descargarlo desde github, como se indica en cada una de ellas.

Aun así, para este caso en particular, deberías importar lo siguiente:

```
from models import users, get_user
```

Y respecto a usar una consulta nativa directamente, pues depende del proyecto, pero para este caso en particular prefiero usar sqlalchemy porque te abstrae de bastantes cosas y te facilita mucho la vida.

^ | v • Responder • Compartir >



**Guido** • hace 3 meses

al fin una implementacion de la libreria flask\_login sin sqlalchemy, y encima en español! implementarlo con sqlalchemy es mucho mas facil, pero se pierde de vista toda la mecanica, en especial el uso de la clase "usermixin".

aprovecho para consultarte:

en la vista/ruta para el login estan estas dos lineas de codigo mediante las cuales cargas/logueas al usuario

```
user = get_user(form.email.data)
login_user(user, remember=form.remember_me.data)
```

el parametro "user" de login\_user conserva simplemente los datos del usuario? no hace falta llamar al objeto User (de `models.py`)? en qué momento hacés uso de la clase "User"?

Conviértete en maestr@Pythonista

imagino que la funcion "login\_user" tiene algo que ver con el decorador "@login\_manager.user\_loader"?

muchas gracias!! Se hace uso de la clase User precisamente cuando se llama a la función `get_user()`. Fijate que esta función recorre la lista `users` del módulo `models.py`, la cuál es una lista de elementos de tipo `User`.

^ | v • Responder • Compartir >



**Juan Anonimo** • hace 9 meses

Hola amigo muy bueno tu tuyo ya voy por el numero 8 pero me han surgido varias dudas si me puedes ayduar a resolverlas o guiar a donde puedo.

- 1 cuando creo un login de usuario como puedo hacer para crear una pagina de perfil de ese usuario que se dio de alta en mi block. Algo asi como fb pero simple solo usuario y datos basicos.
2. Depronto ni entendi bien pero si una persona abre la ruta about de mi block sera redijigida a el login de usuario y creacion de usuario o eso lo tengo que especificar en cada una de mis rutas.
3. El control de versiones es importante creo yo en esta etapa de desarrollo alguna app o donde guardarla que no sea git. Tengo entendido que al ser publico cualquiera tiene acceso a los datos y despues de que los borre estos igual nunca se pierden del sistema. Amenos de que sea la version oaga y ps en fase de desarrollo en mi caso el presupuesto es 0 ya que es mas por aprender que por generar dinero
4. Gracias por este tuto me ha servido mucho en mi proyecto.

^ | v • Responder • Compartir >



**Juan José Lozano Gómez** Moderador ➔ Juan Anonimo • hace 9 meses

Te respondo a las dudas.

- 1: Tienes que crear una nueva página con el formulario y en la vista del login, en lugar de redirigir a index, rediriges a tu nueva página con la instrucción `url_for`.
- 2: A las páginas que quieras proteger con usuario y contraseña, debes añadirle el decorador `@login_required`. Está explicado en la lección.
- 3: Puedes instalar git de forma local en tu equipo sin necesidad de utilizar Github. Puedes hacer una copia de seguridad del repositorio local en un disco externo para no perder los datos.

^ | v • Responder • Compartir >



**Julio Barbagallo** • hace un año • edited

Conviértete en maestr@ Pythonista



(<https://j2logo.com>)

Hasta acá llegue perfecto, pero vengo hace un par de días dando vueltas con un error y no puedo salir. Serías tan amable de guiarme un poco en el sentido correcto?

Tanto cuando voy a signup o login, tengo el siguiente error:

NameError: name 'current\_user' is not defined

Y del lado del browser:

Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Gracias y saludos.

^ | v • Responder • Compartir >



**Juan José Lozano Gómez** Moderador ➔ Julio Barbagallo • hace un año



^ | v • Responder • Compartir >



**Juan Magallanes** • hace un año

Hola. Primero, darte las gracias por este excelente tutorial. Soy nuevo en programación y he aprendido mucho. Tengo una falla al correr el programa `run.py` me dice:

NameError: name 'login\_required' is not defined

^ | v • Responder • Compartir >



**Juan José Lozano Gómez** Moderador ➔ Juan Magallanes • hace un año

Debes importar el decorador al comienzo del fichero `run.py` así:

```
from flask_login import login_required
```

^ | v • Responder • Compartir >



**Isaías** • hace un año

Buenas tardes!

Me esta tirando error en la linea 35, "Archivo "C:\Users\Administrador\Desktop\tutorial-flask\env\run.py", línea 34

```
@ app.route ("/ signup /", method = ["GET", "POST"])
```

Conviértete en maestr@Pythonista



Hola Isaías. En Python debes tener cuidado con las tabulaciones y espacios ya que si no, el intérprete te mostrará errores como el que te está ocurriendo. En principio, la línea que indicas no debería tener ningún espacio ni tabulación a la izquierda. Aquí puedes ver un ejemplo completo del código <https://github.com/j2logo/t...>

^ | v • Responder • Compartir >



**h xia** • hace 2 años

Hola,

Si un usuario ha olvidado su contraseña, y queremos enviarle un link de tiempo limitado a su correo para restaurar la contraseña. Cómo podemos implemetar eso con Flask?

Gracias.

^ | v • Responder • Compartir >



**Juan José Lozano Gómez** Moderador ➔ h xia • hace 2 años • edited

La pregunta da para un post completo, pero intentaré resumirla.

Básicamente, la parte importante es generar el enlace con un token y, posteriormente, definir una view en la que recibas el token como parámetro y lo valides.

Esta es la parte que genera el token:

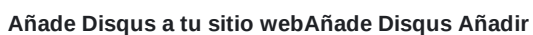
```
# Obtener el email de los parámetros de la petición (ya sea un formulario, JSON, etc.)
email = data['email'].lower()
user = User.get_by_email(email)

salt = current_app.config.get('EMAIL_RECOVERY_SALT', 'tu-salt-cambialo')
token = generate_token(user.email, salt)
recovery_link = url_for('auth.reset_password_with_token', token=token, _external=True)
body_txt = render_template('auth/mail/reset_password_txt.html', recovery_link=recovery_link)
body_html = render_template('auth/mail/reset_password.html', recovery_link=recovery_link)

# Tu función para enviar el email
send_email(subject=subject, sender=current_app.config['DONT_REPLY_FROM_EMAIL'],
```



^ | v • Responder • Compartir ›



[Aviso legal \(https://j2logo.com/aviso-legal/\)](https://j2logo.com/aviso-legal/) [Política de privacidad \(https://j2logo.com/politica-de-privacidad/\)](https://j2logo.com/politica-de-privacidad/) [Política de cookies](#)

(<https://j2logo.com/politicas-de-cookies/>) | [Sobre j2logo \(https://j2logo.com/about/\)](https://j2logo.com/about/)

Un saludo a todos (es pytho

Nunca dejes que nadie te diga que no puedes hacer algo

Copyright © 2018-2020 Juan José Lozano Gómez

Conviértete en maestr@ Pythonista

