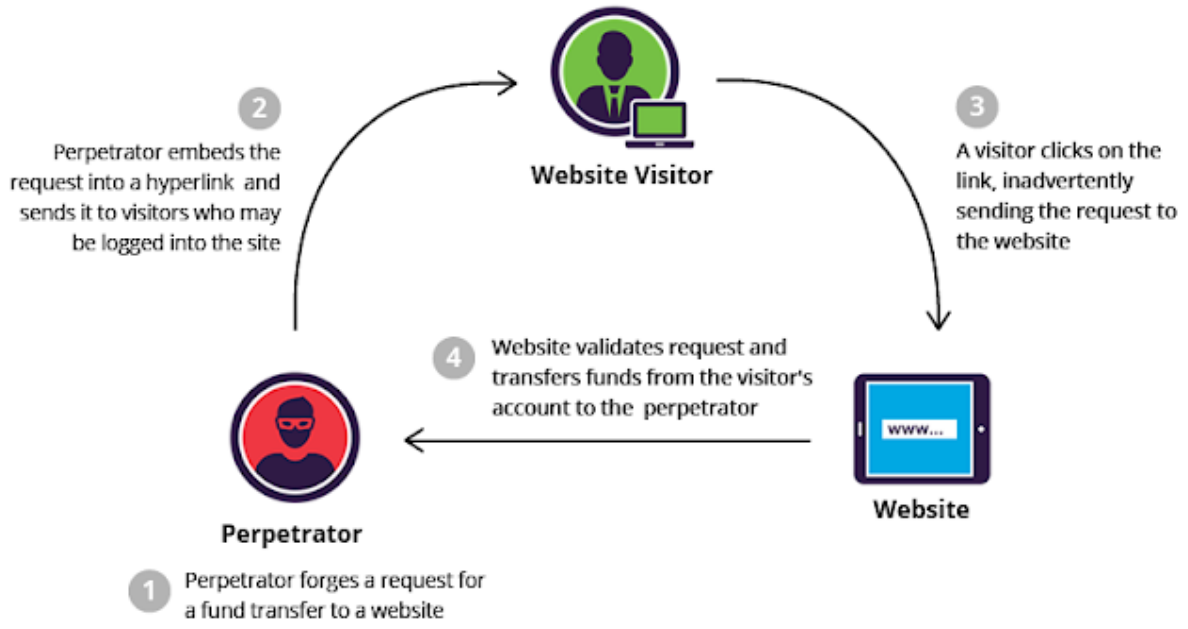
Flask_SQLAlchemy_Python_LOGIN_REGISTER
by Jan Vallve

Seguire la base creada del anterior proyecto

Primerament esos son los módulos de flask importados que usare para hacer el login:

```
P > app.py > ...
from flask import Flask, render_template, request, url_for, redirect, flash #jsonify
from flask_sqlalchemy import SQLAlchemy
#from flask_marshmallow import Marshmallow
from flask_bcrypt import Bcrypt
#--flask wtf
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, BooleanField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError

from flask_login import UserMixin, login_user, current_user, logout_user, login_required, LoginManager
#from flask_login import login_user, current_user, logout_user, login_required
```

Quiero destacar el FLASK_WTF, queria introducir un sistema para evitar los **"ataques csrf"**



Este esquema resume un ataque CSRF, para hacer peticiones desde el pc de un cliente infectado sin saberlo. Flask WTF por defecto te genera un token único cuando el cliente se loguea a nuestra página.

CSRF Protection

Any view using `FlaskForm` to process the request is already getting CSRF protection. If you have views that don't use `FlaskForm` or make AJAX requests, use the provided CSRF extension to protect those requests as well.

```python
app = Flask(__name__)
#app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root@localhost/flask_warships'
app.config['SECRET_KEY'] = '8456984at5b1ace7gfs578fhdjs129e8'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database/warship_jan.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

#ma = Marshmallow(app)
bcrypt =Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
login_manager.login_message_category = 'info'
```

Usare el módulo Bcrypt para poder encriptar y comprobar las contraseñas de una manera muy cómoda.

El login manager: https://flask-login.readthedocs.io/en/latest/

Tiene algunas funciones q me interesan como restringir views o hacer que el botón de “stay logged”.
Facil de seguir con la documentación.

Flask-Login

Flask-Login provides user session management for Flask. It handles the common tasks of logging in, logging out, and remembering your users’ sessions over extended periods of time.

It will:

- Store the active user’s ID in the session, and let you log them in and out easily.
- Let you restrict views to logged-in (or logged-out) users.
- Handle the normally-tricky “remember me” functionality.
- Help protect your users’ sessions from being stolen by cookie thieves.
- Possibly integrate with Flask-Principal or other authorization extensions later on.

```python
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20))
    email = db.Column(db.String(60))
    password = db.Column(db.String(50))
```

Para tenerlo claro he defino la clase USer que es la que usaremos para iniciar sesión desde la base de datos. NO ES NECESARIO, pero personalmente creo que es más claro.

Para hacer las página de login/register usare “forms” con ayuda del module “wtform”

Almacenaremos los datos puestos en variables automáticamente y será mucho más fácil trabajar con estos datos.
Al importar

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

podremos definir aquí mismo validadores muy fácilmente para no introducir valores no válidos y dar alertas a los users.

```
#-- Forms para login/register


#-Estas clases con variables las llamaremos en las paguinas que queramos hacer un formulario

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired("Introduce un nombre de usuario valido")])
    #email = StringField('Email', validators=[DataRequired("Introduce un mail valido"), Email()])
    password = PasswordField('Password', validators=[DataRequired("Contraseña")])
    submit = SubmitField('Login')

class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')

    #def validate_username(self, username):
    #    user = User.query.filter_by(username=username.data).first()
    #    if user:
    #        raise ValidationError('That username is taken. Please choose a different one.')

    #def validate_email(self, email):
    #    user = User.query.filter_by(email=email.data).first()
    #    if user:
    #        raise ValidationError('That email is taken. Please choose a different one.')
```

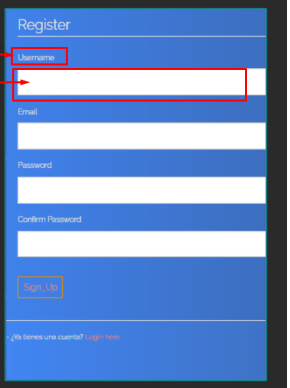
Para aplicar la “form” a nuestro html, primeramente tenemos q darle la info del form q queremos desde el arch .py el el @app.route:

```
@app.route("/register", methods=['GET', 'POST'])
def register():

    if current_user.is_authenticated:
        return redirect(url_for('index'))
    form = RegistrationForm()

    if form.validate_on_submit(): #esto actua como un condicional de cuando usamos el metodo POST
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        #print(hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8'))
        user = User(username=form.username.data, email=form.email.data, password=hashed_password)
        db.session.add(user)
        db.session.commit()

        #flash('Tu cuenta ha sido creada!', 'success')
        return redirect(url_for('login'))
        #return redirect(url_for('/tiers/'))
    print(form.errors)#imprimira los errores que me devuelve del form que he llamado antes!! Imporante ya que asi veremos
    desde la terminal si tenemos algo mal
    #la idea es poner condicionales para usar mensajes flash para dar feedback al usuario y que pueda correg el error sin
    ver la pantalla fea de errores
    #if form.is_submitted():
    #    print ("SE añade") #cada vez que le demos al boton submit entrara en este condicional
    return render_template('register.html', form=form)
```

Una vez lo tenemos importado, lo devolvemos con toda la info, nos acirdemos que

```
if form.validate_on_submit(): #esto actua como un condicional de cuando usamos el metodo POST
```

```
<!--Aqui empieza el body-->
{% block body %}
<div class="card-header about-contact">
    <form method="POST" action="">
        {{ form.hidden_tag() }}
        <fieldset class="form-group whitey">
            <legend class="border-bottom mb-4">Register</legend>
            <div class="form-group">
                <!--Esto es la "label" de la form (del modulo Flask_WTF) que hemos importado desde app.py, en este caso el
                RegisterForm -->
                <!--Los nombres de las form tienen q ser iguales a las variables que hemos definido-->
                {{ form.username.label(class="form-control-label") }}
                <!--Aqui ponemos el campo para escribir y que luego lo enviaremos como "username" al usar el boton que tmb
                hemos definido en el "Register form"-->
                {{ form.username(class="form-control form-control-lg") }}
            </div>
            <!--Este div lo repetiremos por cada campo que queremos poner en nuestro formulario, simplemente tendremos que
            cambiar el obj de la form "Registerform"-->
            <div class="form-group">
                {{ form.email.label(class="form-control-label") }}
                <!---->
                <!--Para tal de mostrar un error si el email se ha introducido incorrectamente pondremos un condicional q
                mostrara un error, en el caso de que no tengamos errores se mostrara el mismo cuadro q los otros bloques de
                texto -->
                <!--Usaremos el sietema de "flask wtf" para tal de mostrar los errores y que compruebe si hay problemas-->
                {% if form.email.errors %} {{ form.email(class="form-control form-control-lg is-invalid") }}
```



Nos tenemos que acordar que el “texto” está definido en la clase form de register no en el nombre “username”

```
class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
```

```
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20))
    email = db.Column(db.String(60))
    password = db.Column(db.String(50))
```

```
user = User(username=form.username.data, email=form.email.data, password=hashed_pas
db.session.add(user)
db.session.commit()
```

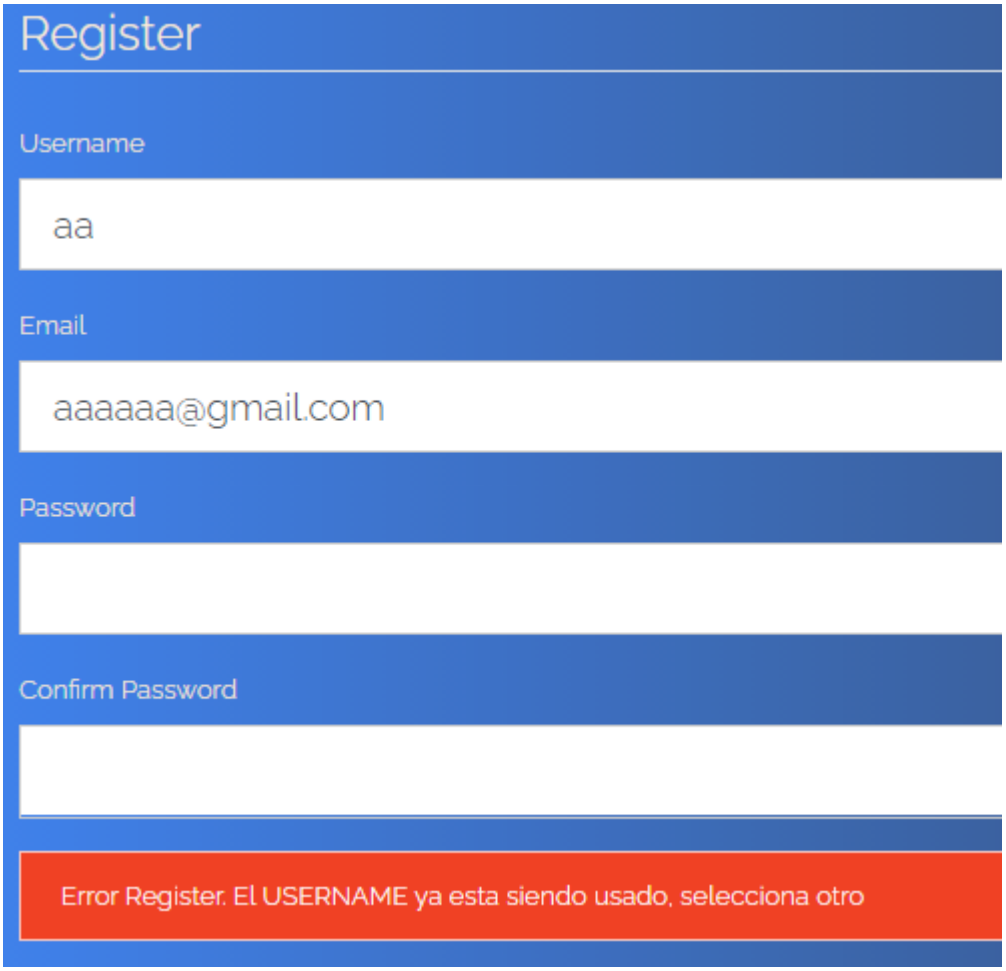
Seguidamente almacenemos los datos en la clase User y hacer un commit para guardar los datos.

```python
class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign_Up')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            flash('Error Register. El USERNAME ya esta siendo usado, selecciona otro', 'danger')#este msg sale en el html del cliente,, lo de danger es
            el tipo de error, en danger saldra de color rojo, info azul etc...
            raise ValidationError('El USERNAME ya esta siendo usado, selecciona otro')#este msg sale en la terminal

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            flash('Error Register. El EMAIL ya esta siendo usado, selecciona otro', 'danger')
            raise ValidationError('El EMAIL ya esta siendo usado, selecciona otro.')
```

lo que hacemos para validar es hacer una consulta en la bd si el usuario existe, si es null no entrara en el condicional y no hará nada.

```python
flash('Error Register. El USERNAME ya esta siendo usado, selecciona otro', 'danger')#este msg sale en el html del cliente,, lo de danger es
el tipo de error, en danger saldra de color rojo, info azul etc...
raise ValidationError('El USERNAME ya esta siendo usado, selecciona otro')#este msg sale en la terminal
```

```
{}
127.0.0.1 - - [03/Apr/2021 13:14:15] "GET /register HTTP/1.1" 200 -
{'username': ['El USERNAME ya esta siendo usado, selecciona otro']}
127.0.0.1 - - [03/Apr/2021 13:14:20] "POST /register HTTP/1.1" 200 -
```



En vez de que nos salga la pantalla de error :

podemos hacer que aparezca un mensaje de advertencia así el usuario no se asustara al ver una pantalla de error que no está familiarizado.

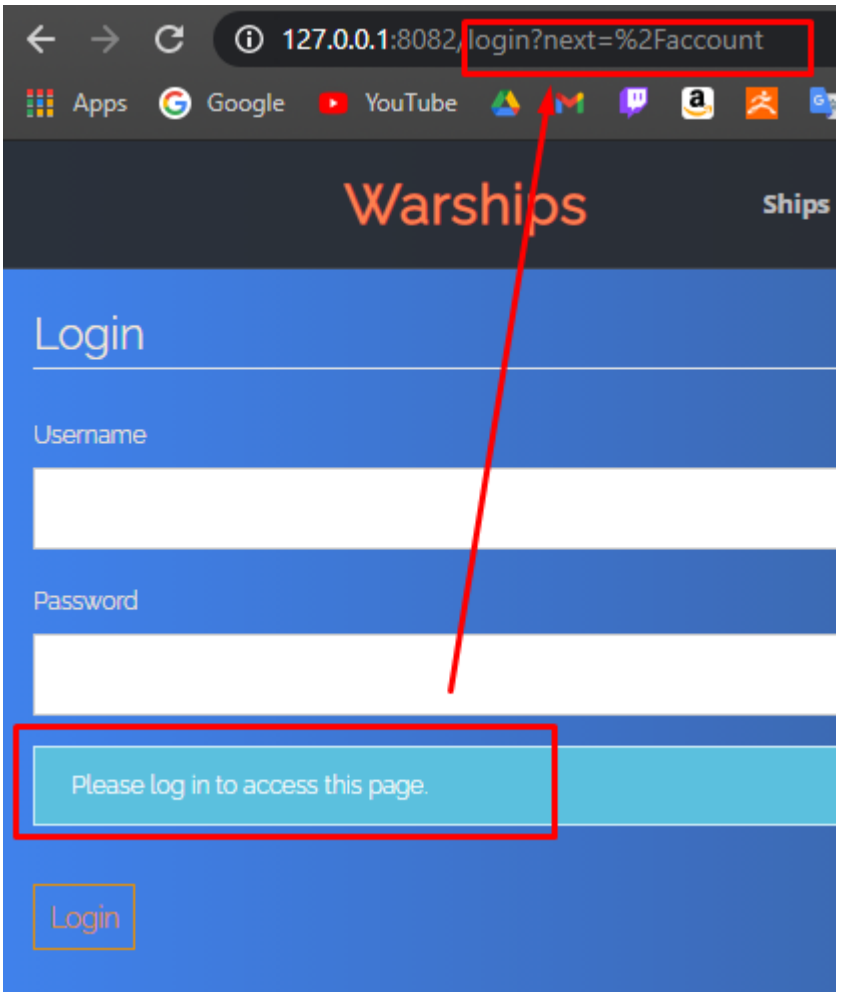
La idea es hacer unos errores similares a estos, usando flash.

```
    <!-- Flash (no flask, flashh)-->
    {% with messages =
get_flashed_messages(with_categories=true) %} {% if messages
%} {% for category, message in messages %}
    <div class="alert alert-{{ category }}">
        {{ message }}
    </div>
    {% endfor %} {% endif %}
    <!---->
    {% endwith %}

    <!--end msg flash-->
```

He encontrado unos condicionales que hacen que comprueben si hay un mensaje flash y lo enseñan con unas clases.

```python
#--Rutas

@app.route('/')
@login_required
def index():
    if current_user.is_authenticated:
        ships = Ship.query.all()
        return render_template('index.html', ships = ships)
    #return render_template('index.html', ships = ships)
```

Si queremos hacer que una pag sea obligatorio estar registrado podemos poner el login required, en el caso que no estemos loggeados nos saldra lo siguiente.



Si intentamos acceder a la pag account.

```python
@app.route("/register", methods=['GET', 'POST'])
def register():

    if current_user.is_authenticated:
        return redirect(url_for('index'))
    form = RegistrationForm()

    if form.validate_on_submit(): #esto actua como un condicional de cuando usamos el metodo POST
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        #print(hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8'))
        user = User(username=form.username.data, email=form.email.data, password=hashed_password)
        db.session.add(user)
        db.session.commit()

        #flash('Tu cuenta ha sido creada!', 'success')
        return redirect(url_for('login'))
        #return redirect(url_for('/tiers/'))
    print(form.errors)#imprimira los errores que me devuelve del form que he llamado antes!! Imporante ya que asi veremos
    desde la terminal si tenemos algo mal
    #la idea es poner condicionales para usar mensajes flash para dar feedback al usuario y que pueda correg el error sin
    ver la pantalla fea de errores
    #if form.is_submitted():
    #    print ("SE añade") #cada vez que le demos al boton submit entrara en este condicional
    return render_template('register.html', form=form)
```

para hacer el register, el condicional `validate_on_submit(): #esto actua como un condicional de cuando usamos el metodo POST`

el `print(form.errors)#imprimira los errores que me devuelve del form que he llamado antes!! Imporante ya que asi veremos desde la terminal si tenemos algo mal`

```
    #la idea es poner condicionales para usar mensajes flash para dar feedback al usuario y que pueda correg el error sin ver la pantalla fea de errores
```

lo errores se veran asi :

```
{}
127.0.0.1 - - [02/Apr/2021 12:15:30] "GET /register HTTP/1.1" 200 -
{'username': ['That username is taken. Please choose a different one.']}
127.0.0.1 - - [02/Apr/2021 12:15:38] "POST /register HTTP/1.1" 200 -
```

```python
def validate_username(self, username):
    user = User.query.filter_by(username=username.data).first()
    if user:
        raise ValidationError('That username is taken. Please choose a different one.')
```

ESTO ES SOLO PARA NOSOTROS, el cliente no vera ninguno de estos errores.

```python
def validate_username(self, username):
    user = User.query.filter_by(username=username.data).first()
    if user:
        flash('Error Register. El USERNAME ya esta siendo usado, selecciona otro', 'danger')
        raise ValidationError('That username is taken. Please choose a different one.')
```


```python
@app.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('/'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user)
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else redirect(url_for('index'))
            print(form.errors)

        else:
            flash('Error Login. El user/contrasenya no se han introducido correctamente', 'danger')
            print(form.errors)
    print(form.errors)#imprimira los errores que me devuelve del form que he llamado antes!! Imporante ya que asi veremos
    desde la terminal si tenemos algo mal
    #la idea es poner condicionales para usar mensajes flash para dar feedback al usuario y que pueda correg el error sin
    ver la pantalla fea de errores
    return render_template('login.html', form=form)
```

```python
@app.route("/logout")
def logout():
    logout_user()
    return redirect(url_for('login'))
```

```
registerform -->
<div class="form-group">
    {{ form.email.label(class="form-control-label") }}
    <!---->
    <!--Para tal de mostrar un error si el email se ha introducido incorrectamente pondremos un condicional q mostrara un error, en el caso
    de que no tengamos errores se mostrara el mismo cuadro q los otros bloques de texto -->
    <!--Usaremos el sietema de "flask wtf" para tal de mostrar los errores y que compruebe si hay problemas-->
    {% if form.email.errors %} {{ form.email(class="form-control form-control-lg is-invalid") }}
    <div class="invalid-feedback">
        {% for error in form.email.errors %}
        <span>{{ error }}</span> {% endfor %}
    </div>
    {% else %} {{ form.email(class="form-control form-control-lg") }} {% endif %}

</div>
```

para dar un poco de feedback al user, le pondremos msg de error si no pone el mail o la segunda contraseñ bn.

LogOut:

Para hacer el logout usare componentes del módulo, flask_login

con esto me ahorro lineas de código y queda mas claro.

En el caso de que este ya fuera de sesión, no encontrará una sesión activa y redirigirá a login, si pusiera “`@login_required`” no tendría mucho sentido ya que entraría en la sesión y saldría instantáneamente cosa que despistaria al cliente.

```
from flask_login import UserMixin, login_user, current_user, logout_user, login_required, LoginManager
```

```
@app.route("/logout/")
def logout():
    logout_user()
    return redirect(url_for('login'))
```

Delete User:


```python
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20))
    email = db.Column(db.String(60))
    password = db.Column(db.String(50))
```