MANUAL TECNICO

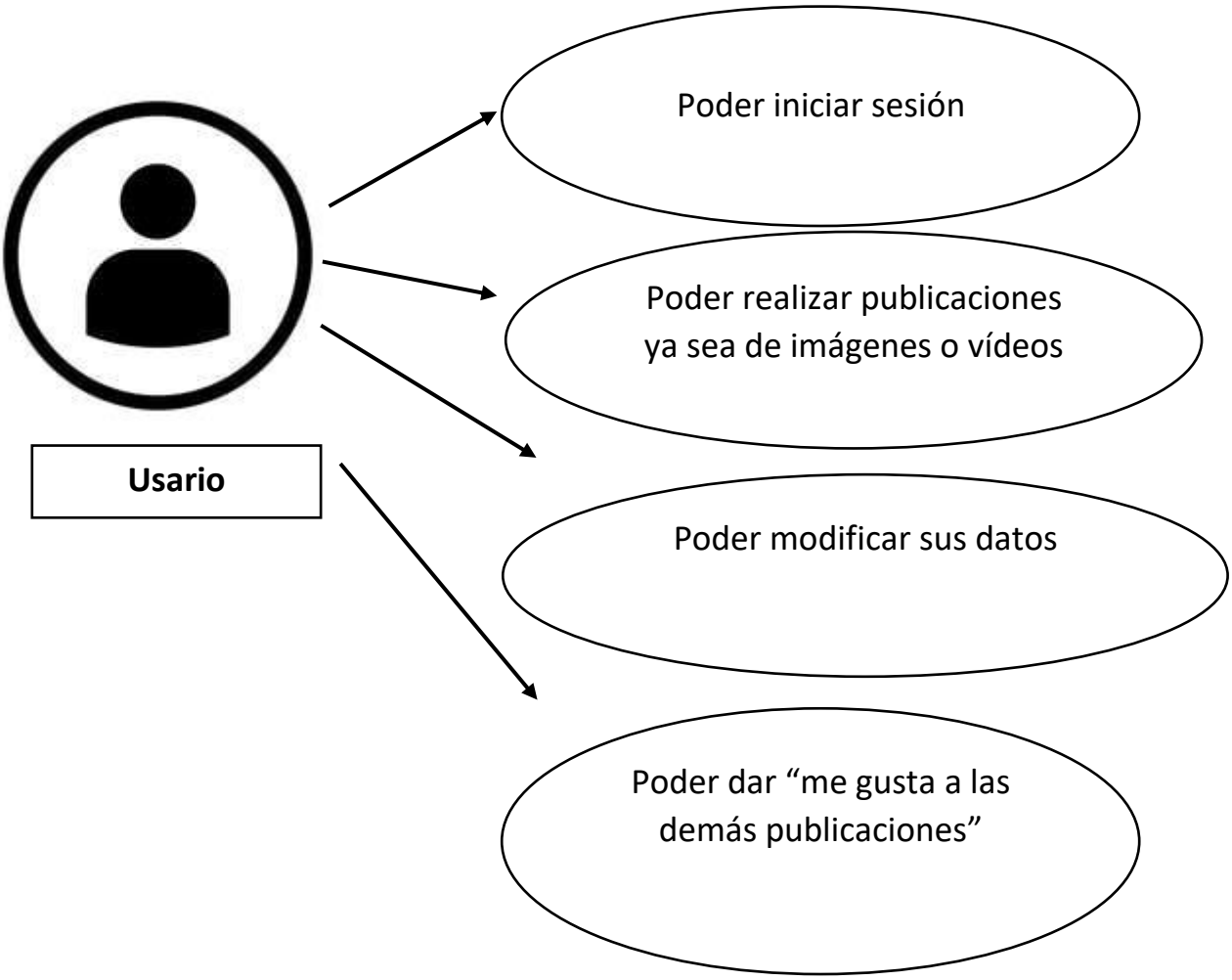
APLICACIÓN UBLOG

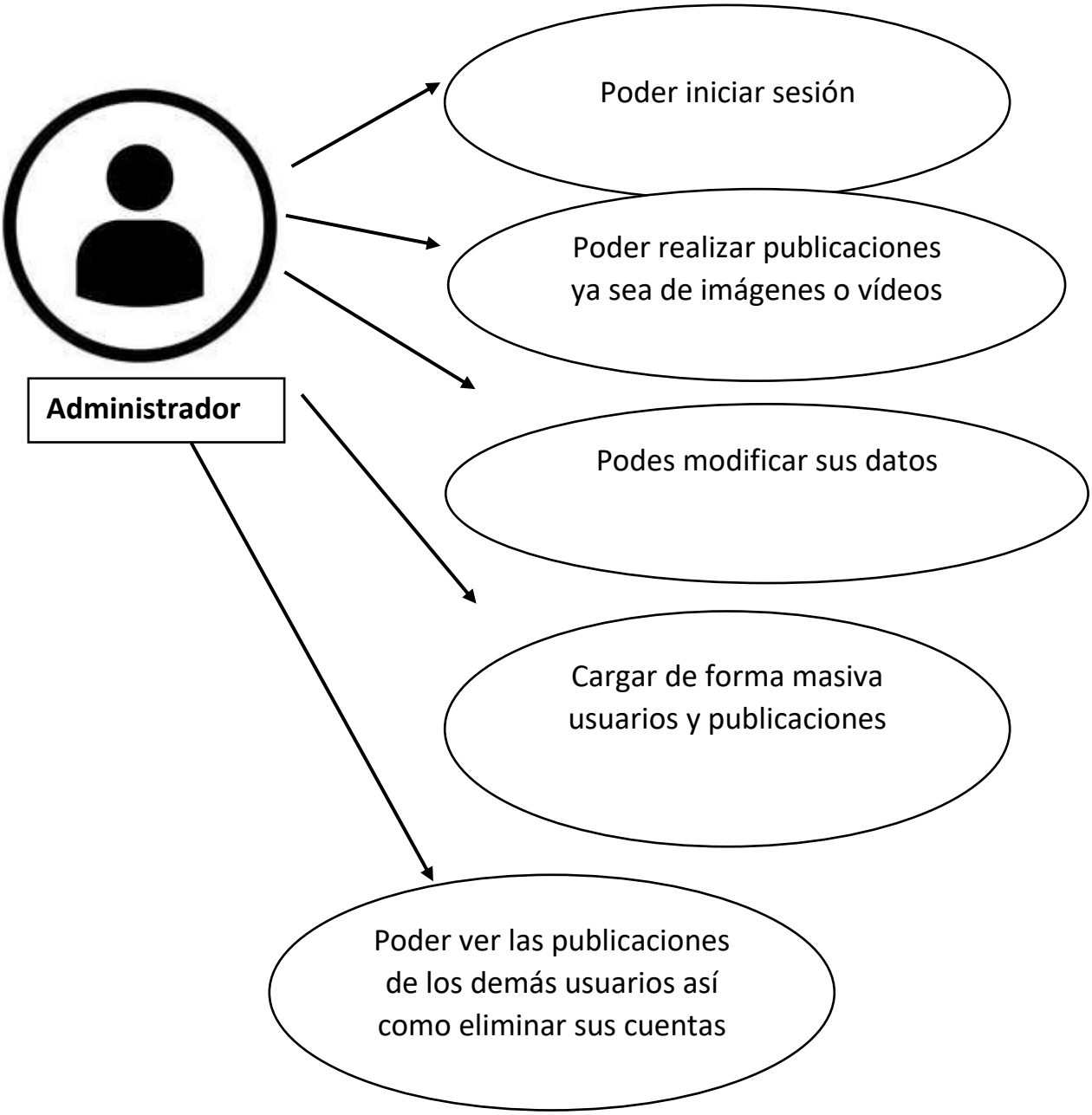
Elaborado por: Cristian Alexander Mejía Cahuec
Carné: 201807085
Fecha: 07/10/2021
Lugar: Guatemala

Objetivos del sistema y alcances del sistema

El objetivo primordial de este manual es ayudar a guiar al técnico a informarse y utilizar las distintas funciones de la aplación hecha para un blog en Internet llamado UBlog", para que de esta manera poder hacer uso de la información deseada para poder despejar todas las dudas respecto al funcionamiento del mismo, el sistema fue realizado para poder crear publicaciones ya sea de imágenes o vídeos de Internet, por medio de la implementación y manipulación de una API hecha en Python utilizando Flask.

- **Alcances:**


Poder iniciar sesión
Poder realizar publicaciones
ya sea de imágenes o vídeos
Usario
Poder modificar sus datos
Poder dar “me gusta a las
demás publicaciones”


Poder iniciar sesión
Poder realizar publicaciones
ya sea de imágenes o vídeos
Administrador
Podes modificar sus datos
Cargar de forma masiva
usuarios y publicaciones
Poder ver las publicaciones
de los demás usuarios así
como eliminar sus cuentas

Requisitos de Hardware

- Procesador Intel Core 2 Duo o superior
- Memoria RAM mínima recomendada 512 MB
- Espacio en disco duro 512 MB
- Poseer acceso a Internet

Requisitos de Software

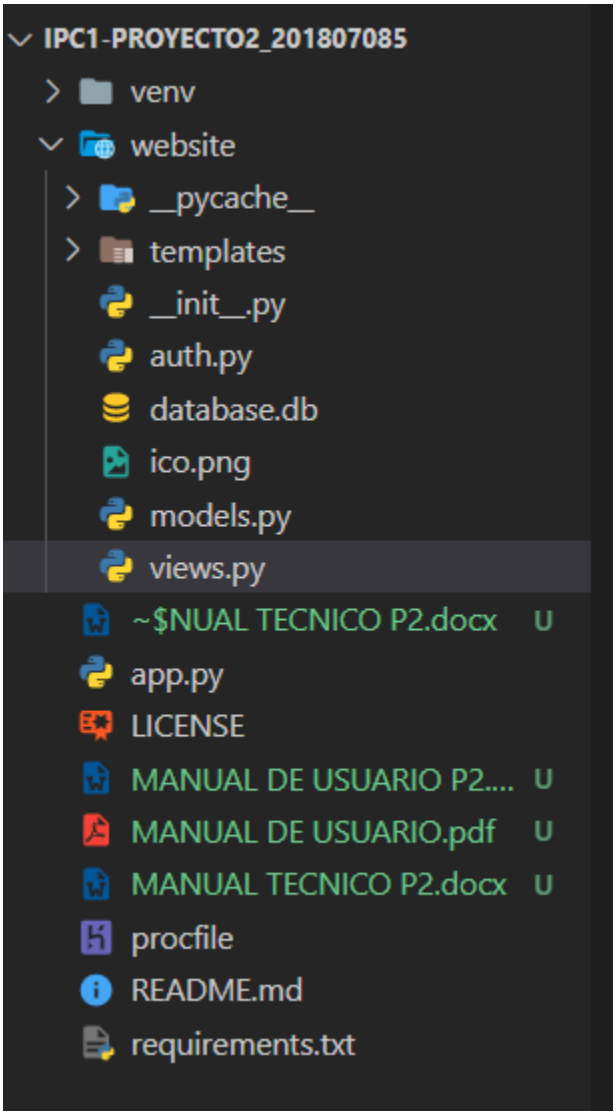
- Poseer sistema operativo Win7 o superior
- Poseer VS Code así como Python3 con flask instalado junto con las siguientes librerías:

- ***gunicorn==20.1.0***
- ***click==8.0.3***
- ***colorama==0.4.4***
- ***Flask==2.0.2***
- ***Flask_Login==0.5.0***
- ***Flask_SQLAlchemy==2.5.1***
- ***Flask_WTF==0.15.1***
- ***greenlet==1.1.2***
- ***itsdangerous==2.0.1***
- ***Jinja2==3.0.2***
- ***MarkupSafe==2.0.1***
- ***pip==21.2.3***
- ***setuptools==57.4.0***

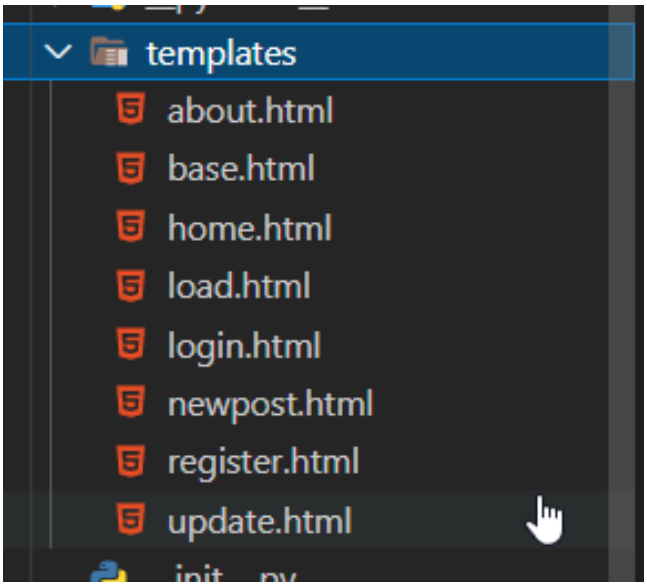
- ***SQLAlchemy==1.4.26***
- ***Werkzeug==2.0.2***
- ***WTForms==2.3.3***

Lógica del programa

- **Estructura raíz**

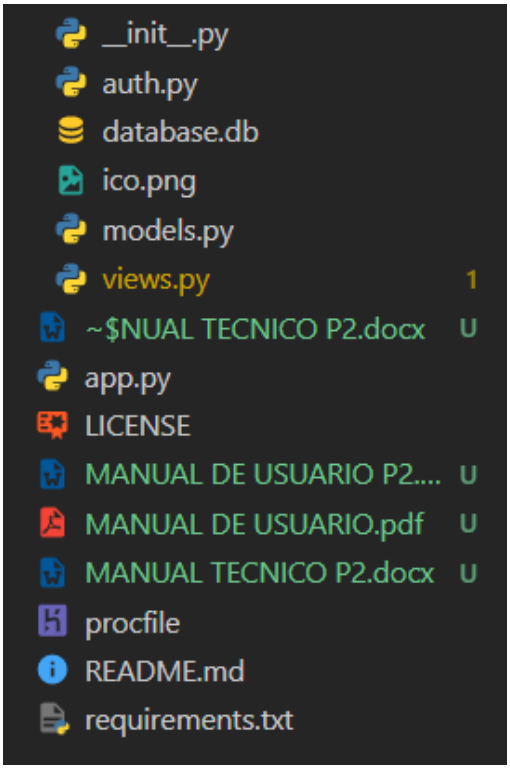


- **Archivos relacionados con el Fronted**



- **Archivos relacionados con el Backend**

-



- **App.py**
- Contiene la clase main de la aplicación

```python
# Importamos las configuraciones iniciales
from website import create_app

if __name__ == "__main__":
    app= create_app()
    app.run(debug=True)
```

- **Views.py**

```python
# Contiene todo lo relacionado con el blog como el home page, etc...
from flask import Blueprint, render_template, request, flash, redirect, url_for
from flask_login import login_required, current_user
#Importamos la clase Post
from .models import Post
from . import db

# Importamos las librerías que nos serviran para mostrar las páginas que solamente aparecen si hay una sesión iniciada


views = Blueprint("views",__name__)

@views.route("/")
@views.route("/home")
@login_required
def home():
    posts = Post.query.all()
    return render_template("home.html", user=current_user, posts=posts)
```

```python
@views.route("/new-post", methods=['GET', 'POST'])
@login_required
def new_post():
    if request.method == "POST":

        tipo = request.form.get('tipo')
        url = request.form.get('url')
        category = request.form.get('category')
        post = Post(tipo=tipo,url=url,category=category,author=current_user.id)
        #Añadimos el nuevo post relacionado con el user id a la tabla
        db.session.add(post)
        db.session.commit()
        flash('Publicación agregada con éxito!', category='success')
        return redirect(url_for('views.home'))

    return render_template('newpost.html', user=current_user)
```

- **Models.py**

Contiene las plantillas para nuestros usuarios y nuestros Post

```python
from . import db
from flask_login import UserMixin
from sqlalchemy.sql import func

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    gender = db.Column(db.VARCHAR())
    username = db.Column(db.String(100), unique=True)
    email = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(100))
    #Acá hacemos la relación de los Post con los usuarios
    #Se hace referencia a todos los post que el usuario tiene, backref nos devuelve
    el usuario y hace referencia automáticamente a la clase usuario y nos permite
    hacer la adición directa de la publicación a éste, passive_delete hace referencia
    al CASCADE al momento de eliminar a un usuario se borran también las
    publicaciones
    post = db.relationship('Post', backref='user', passive_deletes=True )
```

```python
class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    tipo = db.Column(db.Text, nullable=False)
    url = db.Column(db.Text, nullable=False)
    category = db.Column(db.Text, nullable=False)
    fecha_creacion = db.Column(db.DateTime(timezone=True), default=func.now())
    #Le pasamos el id del usuario para saber quién hizo la publicación
    #Se utiliza como una llave alterna que liga el id de esta base de datos con la
    de los usuarios
    #CASCADE quiere decir que cuando se elimine el usuario eliminará también todas
    las publicaciones que este usuario ha hecho
    #nullable no puede ser nulo el valor para poder ser ingresado en la tabla
    author = db.Column(db.Integer, db.ForeignKey('user.id',ondelete="CASCADE"),
    nullable=False)
```

- **Auth.py**

Contiene todo lo relacionado con la autenticación y el login

```python
# Contiene todo lo relacionado con autenticación y login
from flask import Blueprint, render_template, redirect, url_for, request, flash
from . import db
from .models import User
from flask_login import login_user, logout_user, login_required, current_user


auth = Blueprint("auth",__name__)


@auth.route("/login", methods=['GET','POST'])
def login():
    if request.method == 'POST':
        username = request.form.get("username")
        password = request.form.get("password")

       # Buscamos al usuario si este existe y que nos devuelva el objeto
        user = User.query.filter_by(username=username).first()
        #Comparamos la contraseña para ver si es correcta
        if user:
          if user.password == password:
            flash("Sesión iniciada", category="success")
            #Verificamos si la sesión está iniciada
            login_user(user,remember=True)
            return redirect(url_for('views.home'))
```

```
            else:
                flash("contraseña incorrecta! ", category="error")
        else:
            flash("Usuario no existe!", category="error")

    return render_template("login.html", user=current_user)


@auth.route("/register", methods=['GET','POST'])
def register():
    if request.method == 'POST':
        name = request.form.get("name")
        gender = request.form.get("gender")
        username = request.form.get("username")
        email = request.form.get("email")
        password = request.form.get("password")

        #Buscamos que el username no exista
        #Si hay algún resultado no se puede agregar el usuario
        username_exists = User.query.filter_by(username=username).first()
        #Buscamos que el correo tampoco exista
        email_exists = User.query.filter_by(email=email).first()
        # Buscamos números en la cadena
        contiene_numero = any(chr.isdigit() for chr in password)
```

```
# Contiene todo lo relacionado con el blog como el home page, etc...
from flask import Blueprint, render_template, request, flash, redirect, url_for
from flask_login import login_required, current_user
#Importamos la clase Post
from .models import Post
from . import db

# Importamos las librerías que nos serviran para mostrar las páginas que solamente a


views = Blueprint("views",__name__)

@views.route("/")
@views.route("/home")
@login_required
def home():
    posts = Post.query.all()
    return render_template("home.html", user=current_user, posts=posts)

@views.route("/new-post", methods=['GET', 'POST'])
@login_required
def new_post():
    if request.method == "POST":
```

```
        if username_exists:
            #Si el usuario existe enviamos un mensaje de alerta
            flash('Este nombre de usuario ya existe!', category='error')
        elif email_exists:
            #Si el email ya existe enviamos un mensaje de alerta
             flash('Este correo ya está registrado!', category='error')
        elif not (contiene_numero):
            #Verificamos que el password contenga números
             flash('La contraseña debe contener un número! ', category='error')
        elif not(caracterEspecial(password)):
            #Verificamos que el password contenga un caracter especial
             flash('La contraseña debe contener un caracter especial!',
             category='error')
        else:
            new_user = User(name=name,gender=gender,username=username,email=email,
            password=password)
            #Se prepara, un paso previo a agregar
            db.session.add(new_user)
            # Se agrega a la base de datos
            db.session.commit()
            login_user(new_user,remember=True)
            flash('Usuario se ha creado con éxito! ')
            return redirect(url_for('views.home'))

    return render_template("register.html", user=current_user)
```

```
@auth.route("/logout")
#Este decorador nos permite que unicamente si hay una sesión iniciada podemos tener
acceso
@login_required
def logout():
    logout_user()
    # Hacemos referencia a una función dentro de view, que redirige a la url que está
    en la función
    return redirect(url_for("views.home"))

@auth.route("/update-profile", methods=['GET','POST'])
@login_required
def update_profile():
    user = User.query.filter_by(id=current_user.id).first()

    nombreCompleto = user.name
    username = user.username

    if request.method == 'POST':
        name = request.form.get("name")
        username = request.form.get("username")
        password = request.form.get("password")
        #Buscamos que el username no exista
        #Si hay algún resultado no se puede agregar el usuario
        username_exists = User.query.filter_by(username=username).first()
```

```
    # Buscamos números en la cadena
    contiene_numero = any(chr.isdigit() for chr in password)

    if username_exists:
        #Si el usuario existe enviamos un mensaje de alerta
        flash('Este nombre de usuario ya existe!', category='error')
    elif not (contiene_numero):
        #Verificamos que el password contenga números
         flash('La contraseña debe contener un número! ', category='error')
    elif not(caracterEspecial(password)):
        #Verificamos que el password contenga un caracter especial
         flash('La contraseña debe contener un caracter especial!',
         category='error')
    else:
        user.name = name
        user.username = username
        user.password = password
        # Se agrega a la base de datos
        db.session.commit()
        flash('Usuario se ha modificado con éxito!')
        return redirect(url_for('views.home'))

  return render_template("update.html", user=current_user,
  nombreCompleto=nombreCompleto,username=username)
```

```
      else:
          user.name = name
          user.username = username
          user.password = password
          # Se agrega a la base de datos
          db.session.commit()
          flash('Usuario se ha modificado con éxito!')
          return redirect(url_for('views.home'))

    return render_template("update.html", user=current_user,
    nombreCompleto=nombreCompleto,username=username)


@auth.route("/about")
def about():
    return render_template("about.html", user=current_user)
```

```python
@auth.route("/admin/load", methods=['GET','POST'])
@login_required
def load():
    if request.method == "POST":

        if request.files["publicaciones"]:
            publicaciones = request.files["publicaciones"]
            print(publicaciones)
            return redirect(request.url)

        elif request.files["usuarios"]:
            usuarios = request.files["usuarios"]
            print(usuarios)
            return redirect(request.url)


    return render_template("load.html", user=current_user)
```

```python
def caracterEspecial(test_str):
    import re
    #busca en la cadena si hay valores a-z o 0-9
    pattern = r'[^\.a-z0-9A-Z]'
    if re.search(pattern, test_str):
      return True
    else:
       return False
```

- **Init.py**

Contiene todo lo relacionado con las configuraciones principales que se cargan al inicio

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from os import path
from flask_login import LoginManager


db = SQLAlchemy()
DB_NAME = "database.db"

def create_app():
    app = Flask(__name__)
    app.config['SECRET_KEY'] = "LLAVESECRETA"
    app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{DB_NAME}'
    db.init_app(app)

# Importamos el blueprint de views y auth
    from .views import views
    from .auth import auth

    app.register_blueprint(views,url_prefix="/")
    app.register_blueprint(auth,url_prefix="/")

    #Importamos la clase User y Post
    from .models import User, Post

    create_database(app)
```

```python
# Importamos el blueprint de views y auth
    from .views import views
    from .auth import auth

    app.register_blueprint(views,url_prefix="/")
    app.register_blueprint(auth,url_prefix="/")

    #Importamos la clase User y Post
    from .models import User, Post

    create_database(app)

    login_manager = LoginManager()
    login_manager.login_view = "auth.login"
    login_manager.init_app(app)

    @login_manager.user_loader
    def load_user(id):
        return User.query.get(int(id))

    return app
```

```python
    #Importamos la clase User y Post
    from .models import User, Post

    create_database(app)

    login_manager = LoginManager()
    login_manager.login_view = "auth.login"
    login_manager.init_app(app)

    @login_manager.user_loader
    def load_user(id):
        return User.query.get(int(id))

    return app

def create_database(app):
    if not path.exists("website/" + DB_NAME):
        db.create_all(app=app)
        print("Base de datos creada")
```