Ministerio del Poder popular para la educación universitaria
Universidad nacional experimental de las telecomunicaciones e informática
Pnf: Informática

Programación 1 MOD 2 Sección 10

Profesor: David Segura

# Aplicación web con Python y Flask

Alumno:

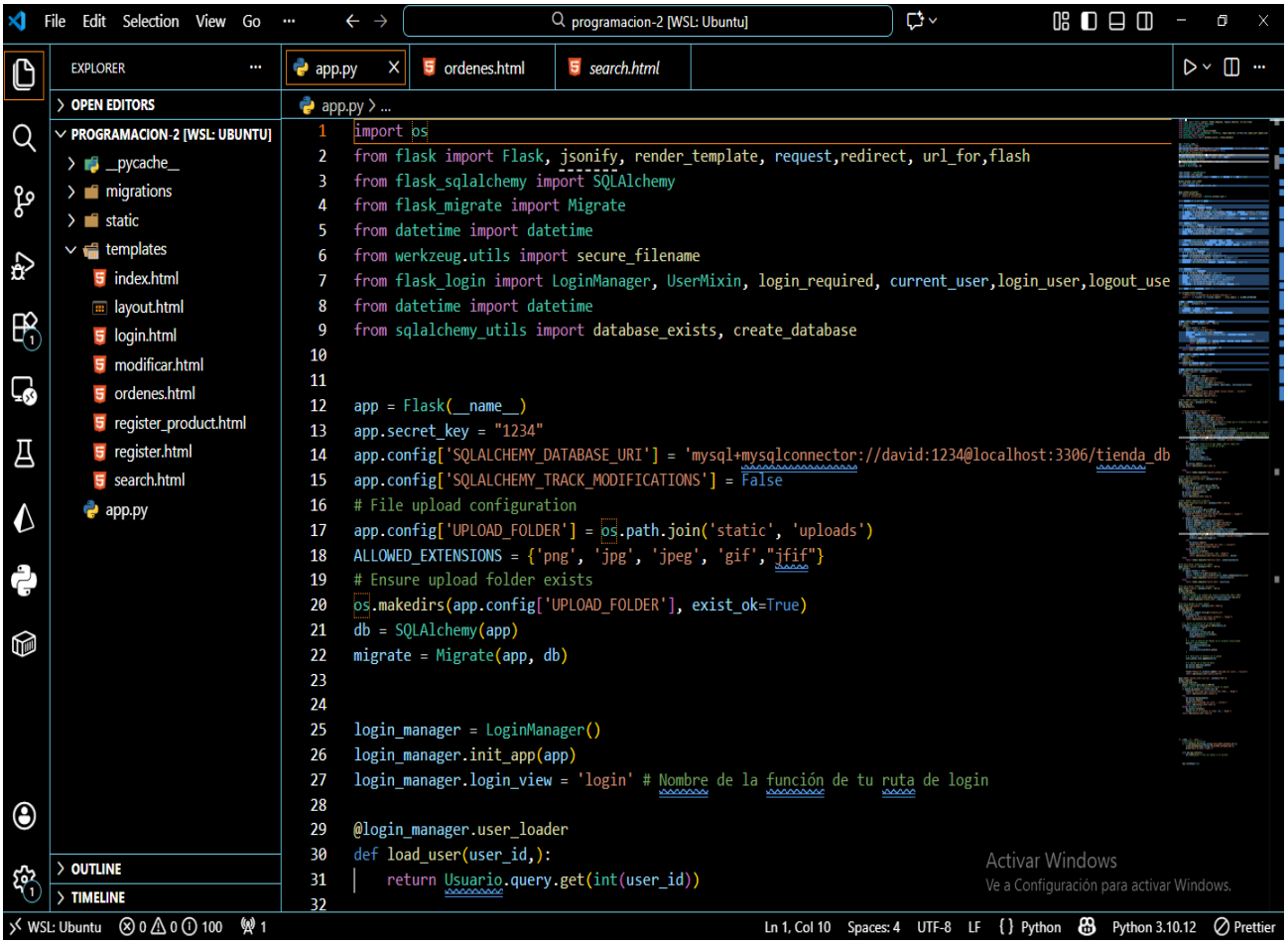
David Lovera V. 25.531.572

febrero, 2026

# Introducción

En el presente trabajo mostraremos el proceso de cómo creamos una aplicación web con Python como lenguaje de programación junto a Flask para el backend, MariaDB gestor de la base de datos y MySql para la base de datos y HTML, Bootstrap para  el frontend, el proyecto tiene 4 entidades Usuarios, Productos, Pedidos, PedidosDetalles la cuales  almacenan la información, además utilizamos Sqlalchemy que nos servirá para crear las estructuras de las tablas así como comunicarse con mariaDB para gestionar cuando queramos agregar, modificar o eliminar alguna columna, todo esto servirá para crear un CRUD sobre una tienda que vende repuestos para carros.

# Creación de la aplicación web




Lo primero que haremos para crear nuestra aplicación web será, crear la arquitectura de nuestro programa, crearemos las carpetas static, templates y app.py cada una tendra una funcion:

- Static: Es en donde se almacenarán las imágenes de los productos en caso de que el usuario ingrese una imagen
- Templates: Aquí es donde se almacenarán las distintas páginas, las cuales generamos con HTML, Bootstrap y Jinja 2 para poder transferir la lógica del backend hacia el frontend
- app.py: Es en donde se hará toda la lógica del Backend, ya sea el llamado a las librerías para que proyecto funcione, la creación de la app, la conexión a la base de datos de mysql a través de mariadb, la creación de las tablas a través sqlalchemy, así como toda la funcionalidad que tendrá el CRUD.

La carpeta migrations se iniciará una vez se termine el modelado de la base de datos una vez tengamos los esquemas creados de cómo deben relacionarse la tablas procederemos a ejecutar en el terminal el comando **flask db init**, esto creará las primeras instancias de las tablas, es decir, los contenedores donde

almacenaremos dichas tablas, luego con el comando flask db migrate -m “” con esto generamos el script basado en el contenido de nuestras tablas, aquí se pone un mensaje ya que con este comando podemos actualizar eliminar o agregar alguna columna a nuestras tablas el mensaje sería para llevar un registro de los cambios parecido a como funciona GIT, por último usaremos el comando flask db upgrade que aplica la migración a la base de datos y aplica los cambios definitivamente, el archivo __pyarche_ se inicia justo cuando ingresamos el comando flask run –port 8000 que lo que hará es utilizar nuestra Pc para que funcione como un servidor para ver nuestra página y se ingresara a esta página por el puerto indicado en este caso 8000.

Una vez creada la estructura procederemos a ir a app.py para importar todas las librerías que vayamos a usar, para instalarlas se usa la línea de comando y ponemos el comando pip3 seguido del nombre de la librería que queremos descargar, luego procederemos a indicar que librerías o herramientas utilizaremos para la aplicación web usaremos varias funciones de la librería Flask así como algunas funciones de sqlalchemy, luego crearemos una variable llamada app donde iniciaremos la app de Flask, indicaremos cómo se conectara la base de datos de mariadb, luego le indicaremos qué formato de imagen podrá ingresar el usuario, iniciaremos sqlalchemy en una variable db y le pasaremos toda la configuración que tiene ya la app y por último crearemos una variable llamada migrate que se encargará de hacer la migración de la base de datos así como una variable llamada login que detectara cuando el usuario esté conectado.

## Creación de la tablas

Crearemos 4 tablas para guardar los registros de la aplicación web las tablas serán:

- Productos
- Pedidos
- DetallePedidos
- Usuarios

Para crear las tablas crearemos una clase para cada una de las tablas y le indicaremos que es un db.model, es decir, creando el modelo de cómo se deben ver todas las columnas de la tabla.

### Tabla DetallesPedido

Lo primero que tendrá la tabla es un __tablename__ que es donde uno le asigna el nombre que tendrá la tabla siempre debe ir en plural ya que habrán varios datos almacenados, el nombre será detallespedidos,tendremos un ID que será una clave primaria, luego tendremos un precio unitario que será el precio total de pedido y será un numeric así como cantidad que será un integer y se encargará de almacenar cuántos productos están siendo pedidos, además tendrá 2 claves foráneas que serán una que hará referencia al ID del pedido su relación será de

muchos a uno ya que por pedido pueden haber varios detalles de dicho pedido y la otra que hará referencia al ID  de la tabla productos su relación es de uno a muchos ya que por detalle de pedido habrán varios productos, ambas claves foráneas tienen on delete esto lo que hará es que cuando se elimine algún producto tambien lo eliminará de la tabla detallepedidos, por ultimo tendrás un campo llamado producto que se encargara de hacer la relacion entre la tabla producto y la tabla detallespedidos, esto se hace para poder acceder a todos de los campos disponibles en la tabla producto.

```python
# --- MODELOS DE LA BASE DE DATOS ---

class DetallePedido(db.Model):
    __tablename__ = 'detalles_pedidos'
    id = db.Column(db.Integer, primary_key=True)
    id_pedido = db.Column(db.Integer, db.ForeignKey('pedidos.id', ondelete='CASCADE'), nullable=Fal
    id_producto = db.Column(db.Integer, db.ForeignKey('productos.id',ondelete='CASCADE'), nullable=
    cantidad = db.Column(db.Integer, nullable=False, default=1)
    precio_unitario = db.Column(db.Numeric(10, 2), nullable=False) # Precio al momento de la compra

class Pedido(db.Model):
    __tablename__ = 'pedidos'
    id = db.Column(db.Integer, primary_key=True)
    id_usuario = db.Column(db.Integer, db.ForeignKey('usuarios.id'), nullable=False)
    fecha_pedido = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    estado = db.Column(db.String(50), nullable=False, default='pendiente')

    # Relación para acceder a los productos desde el pedido
    items = db.relationship('DetallePedido', backref='pedidos', lazy=True, cascade="all, delete-orp
    usuario = db.relationship('Usuario', backref=db.backref('pedidos', lazy=True))
```

## Tabla Pedidos

Lo primero que tendrá es un nombre que se le asignará con __tablename__ en este caso se llamará pedidos, luego tendrá un ID que identifica cada uno de los pedidos, además 3 columnas que serán estado que será un string que almacenará como se encuentra la entrega por default será pendiente, luego una fecha_pedido de cuando se hizo dicho pedido será un DateTime que almacenará la hora a la que se hizo el pedido y por último tendremos una variable llamada usuario que hará la relación a la tabla usuario, el nombre de esta relación que aparecerá en la tabla usuario será pedidos, esta columna tendrá la propiedad on delete esto quiere decir que si se borra el usuario se eliminará el pedido, haremos la misma relación con la tabla detallespedidos y de esta manera se creará una columna nueva en la tabla detallespedidos con el nombre pedidos.

## Tabla Usuario

Esta tabla tendrá UserMixin esta clase nos permitirá hacer los procesos necesarios que un usuario necesita para la autenticación y gestión de sesiones simplificando enormemente la creación de modelos de usuario personalizados sin tener que

definirlos manualmente,al igual que las otras 2 tablas se le asignará un nombre que sera usuarios, un ID que será una llave primaria para identificar a cada uno de los usuarios, tendra 3 columnas nombre, email y contraseña estas 3 columnas almacenarán strings y email que tendrá la propiedad unique=true que lo que quiere decir es que no pueden repetirse los emails, por último tendrá una columna donde haremos la relación de la tabla Usuario con la tabla Producto

```python
class Usuario(db.Model, UserMixin):
    __tablename__ = 'usuarios'
    id = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(80), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    contraseña = db.Column(db.String(120), nullable=False)
    productos = db.relationship('Producto', backref='propietario', lazy=True)


class Producto(db.Model):
    __tablename__ = 'productos'
    id = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(80), nullable=False)
    cantidad = db.Column(db.Integer, nullable=False, default=0)
    precio = db.Column(db.Float, nullable=False, default=0.0)
    imagen_url = db.Column(db.String(255), nullable=True)
    id_usuario = db.Column(db.Integer, db.ForeignKey('usuarios.id'))
```

## Tabla Producto

El nombre de esta tabla será productos, tendrá un ID que será una llave primaria que identificara cada uno de los productos, luego tendremos 2 columnas que almacenará strings nombre guardará el nombre del producto y imagen_url que guarda la dirección en la carpeta donde se almaceno dicha imagen, luego tendremos un integer que sera cantidad que nos indicará cuantos productos disponibles quedan, otra columna será precio que será un float, es decir, un numero con decimales y por último tendremo id_usuario que será una llave foránea de la tabla usuarios y su relación será 1 a muchos ya que un usuario puede tener crear varios productos.

# Lógica de la aplicación web

Ahora mostraremos todas las rutas que tendrá la aplicación web así como sus funcionalidades

## Función Index

En esta función lo que se hace es que cuando el usuario intente de ingresar a la página a través del localhost con el método GET, le mostrará la información de los productos, para que esto sea así crearemos una variable la cual tendrá de valor una búsqueda de la tabla productos y que nos muestre todos los productos en la base de datos, luego haremos un return y le pasaremos esta variable a nuestra página index.html que es donde se visualizará la data.

## Función login

Lo que hace la función login primero nos indicara en que ruta funcionará ficha función dependiendo del método al que se accede la página pasarán cosas distintas si entramos a través del método GET nos mostrará 2 inputs un botón y un link para registrarse, los inputs son para escribir el correo electrónico y la contraseña que tiene dicho usuario, si se presiona el botón enviar se estará usando el método POST, esto lo que hará es que la función login accedera a la información en los inputs a través de un request.form.get, una vez accedemos a la función buscaremos en nuestra db si existe dicho correo con esa contraseña de ser cierto, para esto crearemos una variable llamada usuario y en ella accederemos a la clase usuario y haremos usaremos un .query para acceder al objeto asociado a la tabla en este caso Usuario y usaremos una función llamada filter_by que buscará la información que coincida con el email y contraseña ingresada y queremos que nos de el primero para eso usamos .first(),sí existe el usuario utilizaremos la función login_user de flask para loguear a la persona en su usuario, por último haremos un return redirect(url_for(“index”)) esto funciona para redireccionar a la persona y que pueda ver su usuario en la página index de la manera tradicional lo que haría es redireccionar si actualizar que ya el usuario ingreso.

```python
# RUTA: LOGIN (Iniciar sesión de usuario)
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        email = request.form.get("email")
        contraseña = request.form.get("contraseña")
        usuario = Usuario.query.filter_by(email=email, contraseña=contraseña).first()
        if usuario:
            login_user(usuario)
            flash(f'¡Bienvenido! Has {usuario.nombre} iniciado sesión exitosamente.', 'success')
            return redirect(url_for('index'))
        else:
            return "Credenciales inválidas", 401
    return render_template("login.html")

# RUTA: LOGOUT (Cerrar sesión de usuario)
@app.route('/logout',)
def logout():
    logout_user()
    flash('Has cerrado sesión.', 'info')
    return redirect(url_for('index'))
```

## Función Logout

La función logout primero nos indicara en que ruta se ejecutará la función logout, lo que hará esta función es utilizar la función logout_user que que hara es cerrar la sesión que tenga el usuario abierta, una vez salga de la sesión le aparecerá un mensaje en la pantalla gracias al comando flash, por último haremos return redirect(url_for(index)), esto es para actualizar la página cerrar la sesión del usuario.

## Función Register

```python
# RUTA: REGISTER (Registrar nuevo usuario)
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        nombre = request.form.get('nombre')
        email = request.form.get('email')
        contraseña = request.form.get('contraseña')
        nuevo_usuario = Usuario(nombre=nombre, email=email, contraseña=contraseña)
        db.session.add(nuevo_usuario)
        db.session.commit()
        flash('¡Registro exitoso! Ahora puedes iniciar sesión.', 'success')
        return redirect(url_for('login'))
    return render_template('register.html')
```

La función registrar funcionara en la ruta /register, lo que hará es que si ingresa a esta ruta a través de un método POST se crearán 3 variables  que serán nombre, email y contraseña que almacenarán la información de los inputs de la página register.html, para acceder a estos inputs utilizaremos request.form.get() entre

paréntesis escribiremos el nombre del input que queramos almacenar, una vez tenemos guardados los datos de los inputs en la variable procederemos a crear una nueva variable llamado nuevo_usuario y esta variable lo que hará el almacenar el registro en la tabla usuario luego accederemos a nuestra variable db, en esta variable es en donde queda iniciada la db con sqlachemy y se le pasa de argumento la app para que sepa a qué aplicación le corresponde, con esta conexión mariaDB podrá hacer los query de sql, luego lo que hacemos es llamar a nuestra variable db y accederemos a los métodos .add para agregar dicha información del nuevo usuario a la base de datos y luego .commit para confirmar y guardar la información en la base de datos, luego le aparecerá un mensaje notificando que se registró exitosamente y por último haremos un redirect(url_for(login)) con esto ya el usuario podrá entrar con el usuario que recién registro y si se accede a esta ruta a través de un método GET lo que haremos es enviar al usuario a la página register.html.

## Función Add

Lo primero que haremos será indicará en qué ruta se hará la lógica en este caso se hará en /add esta ruta podrás acceder por ambos métodos tanto POST como GET, luego indicaremos @login_required, esto lo que hará es que si el usuario no está conectado con un usuario ya registrado no podrá crear nuevo productos, ha dicho usuario se le redireccionará hacia la página login.html para que ingrese con su usuario, luego crearemos la función add_product que tendrá la lógica de dicha función, si el método que se ingresa es a través de un método POST, agarremos la información que se nos está pasando en el formulario para todos los inputs accederemos a través del request.form.get() primero tendremos la variable producto que almacenará el nombre del nuevo producto el tipo de dato será tipo string, luego una cantidad que será tipo integer, la cual nos indicará cuánto productos quedan disponibles, le pone int antes de hacer el request para asegurar que el dato que se pasará será un integer, luego tendremos un precio que será un float, esto quiere decir que tendrá números con decimales, tendremos una variable llamada uploaded_file que lo que hará el almacenar la ruta es donde se guardó la imagen, por último tendremos una variable usuario_actual que almacena el id del usuario actual para saber a quién le pertenece el producto, luego lo que haremos será procesar la imagen con una condición si ingresaron una uploaded_file y si esta imagen pasan por el allowed_file, quiere decir que el formato de la imagen es el adecuado, luego crearemos una variable llamada filename usaremos la función secure_filename para saber si el nombre trae algún tipo de virus si pasa todo esto haremos un print para saber donde se almaceno dicha imagen, luego haremos

uploaded_file.save() esto lo que hará es guardar la imagen, para indicarle donde le pasaremos de argumento os.path.join(app.config([“UPLOAD_FOLDER],filename)), para terminar crearemos la variable imagen_url que almacena la url en donde se guardó la la imagen en caso de no haber no asignaremos ninguna imagen, una vez ya tenemos la imagen haremos un llamada a nuestra base de datos con db.session.add(Producto(nombre=producto, cantidad = cantidad, precio = precio, imagen_url=imagen_url,id_usuario=usuario_actual) una vez le pasamos toda la informacion que queremos agregar a la tabla Producto haremos un db.session.commit() para confirmar los cambios y redireccionaremos al usuario a la página index y se le mostrará dicho producto, si se llega a esta ruta a traves del metodo GET lo que haremos será mostrarle la información que se encuentra en register_product.html.

```python
# RUTA: CREATE (Añadir nuevo producto)
@app.route('/add', methods=["GET",'POST'])
@login_required
def add_product():
    if request.method == 'POST':
        producto = request.form.get('producto')
        cantidad = int(request.form.get('cantidad'))
        precio = float(request.form.get('precio'))
        uploaded_file = request.files.get('imagen')
        print(f"Archivos recibidos: {request.files}")
        usuario_actual = current_user.id
        # Guardar el archivo en el sistema de archivos y obtener la URL
        if uploaded_file and allowed_file(uploaded_file.filename):
            print(f"Nombre del archivo: {uploaded_file.filename}")
            filename = secure_filename(uploaded_file.filename)
            uploaded_file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            imagen_url = url_for('static', filename=f'uploads/{filename}')
        else:
            imagen_url = None
        db.session.add(Producto(
            nombre=producto,
            cantidad=cantidad,
            precio=precio,
            imagen_url=imagen_url,
            id_usuario=usuario_actual
        ))
        db.session.commit()
        return redirect(url_for('index'))

    else:
        return render_template('register_product.html')
```

## Función delete_product

Esta función se ejecutará en la ruta /delete/<int:id> y sólo admitirá el método POST, esta ruta es así ya que vamos a obtener el id del producto a través del frontend ya

que lo pasaremos como un atributo id: producto.id, con este id crearemos una variable llamada producto que lo que hara sera almacenar el resultado de la búsqueda en la base de datos del producto en base a su id, esto es así ya que esta función se ejecutara a los productos para eliminarlos si eres el dueño de dicho producto para esto comprobaremos si el id del usuario actual coincide con el id de la persona que creó el producto si no coincide lo que haremos será hacer un return no autorizado y mostrar un error 403 para indicarle al usuario que no se tiene el permiso, esta variable producto la obtenemos en la función index ya que ahí hacemos un query para obtener todos los productos y pasamos la variable para poder usarla en el frontend y poder mostrar la información que queramos mostrar del producto, en cambio si los id coinciden haremos un llamado a nuestra db con db.session.delete(producto) para eliminar dicho producto, luego haremos db.session.commit para  confirmar la eliminación de dicho producto por último haremos un return redirect(url_for("index)) para recargar la ruta index pero con las modificaciones en la db, para realizar cualquier acción en esta ruta tendrás que estar logueado con un usuario registrado en la db.

```
160  # RUTA: DELETE (Eliminar producto)
161  @app.route("/delete/<int:id>", methods=["POST"])
162  @login_required
163  def delete_product(id):
164      producto = Producto.query.get_or_404(id)
165      if producto.id_usuario != current_user.id:
166          return "No autorizado", 403
167      db.session.delete(producto)
168      db.session.commit()
169      return redirect(url_for('index'))
170
```

## Función modificar_producto

La ruta donde se ejecutara esta funcion sera en /modificar/<int:id> queremos el id del producto ya que necesitamos saber qué producto el usuario quiere modificar, se podrá acceder a esta función tanto GET como POST, el usuario primero tendrá que haber iniciado sesión luego generamos una variable producto para buscar el producto en nuestra tabla Productos esto lo haremos con el id que conseguimos a través de la URL la cual se le pasara a la función para acceder a dicho producto, crearemos una condición si el id del usuario que creó el producto no coincide con el id del usuario actual, le lanzara un mensaje indicando que no puede realizar esa acción y redireccionará a la página login.html, si el usuario llega a través del método POST, utilizaremos la variable producto para acceder a la información almacenada en la tabla Producto y le asignaremos estos valores a la variable a cada uno de los inputs de la modificar_producto.html, para acceder a la imagen lo que haremos es hacer un request.file.get("imagen"), luego esta imagen la guardaremos y haremos el

mismo proceso que hicimos para almacenar la imagen cuando creamos el producto, una vez modifiquemos el campo que queramos cambiar, este actualizará el valor que se encuentra en la tabla Producto y se mostrará el nuevo valor actualizado, por último le mostraremos un mensaje al usuario que la modificación se hizo de forma exitosa y lo redireccionaremos a la página index para que vea los cambios, si se accede a esta ruta a través del método GET lo que haremos será enviar al usuario a la página modificar.html.

```python
# RUTA: UPDATE (Modificar producto)
@app.route("/modificar/<int:id>", methods=["POST","GET"])
@login_required
def modificar_producto(id):
    producto=Producto.query.get_or_404(id)
    if producto.id_usuario != current_user.id:
        flash("No tienes permiso para editar este producto.", "danger")
        return redirect(url_for('index'))
    if request.method=="POST":
        producto.nombre=request.form.get("nombre")
        producto.cantidad=int(request.form.get("cantidad"))
        producto.precio=float(request.form.get("precio"))
        uploaded_file = request.files.get('imagen')
        if uploaded_file and allowed_file(uploaded_file.filename):
            filename = secure_filename(uploaded_file.filename)
            uploaded_file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            imagen_url = url_for('static', filename=f'uploads/{filename}')
            producto.imagen_url=imagen_url

        db.session.commit()
        flash("Producto actualizado con éxito.", "success")
        return redirect(url_for('index'))

    else:
        return render_template("modificar.html", producto=producto)
```

## Función search

Crearemos la ruta donde ocurrirá la lógica de la función search, la ruta será /search y admitirá 2 métodos tanto GET como POST, si el usuario accede a esta ruta a traves del metodo POST lo que haremos, será identificar la información que el usuario está ingresando en el input de nombre búsqueda que se encuentra en index.html o en search.html que se accede a través del boton buscar, para agarrar la información almacenada en el input crearemos una variable llamada query que tendrá el valor de input, el cual obtendremos con un request.form.get(“búsqueda”),luego creamos una variable results que tendrá el valor de la búsqueda en la tabla producto, para conseguir dicho producto haremos Producto.query.filter(Producto.nombre.contains(query).all, acá lo que estamos haciendo es aplicar un filtro a nuestra query y el filtro que usaremos sera que vea en la tabla producto que producto coincide con la información guardada en la query y usaremos .all para que nos de todas las opciones de la busqueda por ultimo haremos un return render_template(search.html, results=results) eso lo que hara

recargarle la pagina al usuario para mostrarle el resultado de la busqueda, se pasa la variable results para mostrar el resultado de la busqueda en el frontend, por otra parte si se accede a traves del metodo GET lo que haremos enviar al usuario a search.html se le indica que tambien hara una variable results solo que sera un array vacio dado que aun no tenemos la información de que esta buscando el usuario.

```python
#ruta para buscar productos por nombre
@app.route("/search", methods=["POST", "GET"])
def search():
    if request.method == "POST":
        query = request.form.get("busqueda","")
        results = Producto.query.filter(Producto.nombre.contains(query)).all()
        return render_template("search.html", results=results)
    else:
        return render_template("search.html", results=[])
```

## Función search_order

Creamos la ruta donde se ejecutará la función en este caso será /orders, para acceder a esta ruta solo se podrá a través del método GET se debe haber iniciado sesión con algún usuario, luego creamos la función search_order, la cual tendrá una sola variable llamada orders, esta variable almacenará el resultado de la búsqueda en la tabla Pedido, usaremos el siguiente comando Pedido.query.filter_by(id_usuario=current_user.id).all() acá lo que hacemos en crear una query que lo que hara el filtrar la información en la base de datos para esto necesitaremos acceder al id del usuario y luego con el .all() le mostraremos todas las órdenes que tiene el usuario.

```python
209   #ruta para buscar ordenes por id_usuario
210   @app.route("/orders", methods=["GET"])
211   @login_required
212   def search_order():
213       
214       orders = Pedido.query.filter_by(id_usuario=current_user.id).all()
215       return render_template("ordenes.html", orders=orders)
216
```

## Funcion add_pedido

La ruta donde se ejecutará esta función será en /add_pedido como métodos podrás acceder tanto a través de un POST como de un GET, para acceder a esta ruta necesitarás iniciar sesión con tu usuario, lo primero que hará esta función es crear una variable producto_id para saber el id de los productos, este se pasará a través de un input que está hidden y le asignaremos un value que será producto.id, de no haber producto_id le indicaremos al usuario que no se seleccionaron productos y lo redireccionaremos a la página principal que es index, luego crearemos la variable producto para hacer  una búsqueda en la tabla producto e identificar si dicho id existe, de no ser así devolveremos un error 404, si el usuario intenta acceder a esta ruta via metodo POST lo que haremos sera crear una variable en donde accederemos a la tabla pedido los datos del id del usuario, la fecha del pedido y el estado del pedido, luego crearemos aparte una variable detalle que lo que hará es acceder a la tabla detallePedido para de esta manera registrar el id del producto el precio y la cantidad, por último accederemos a la variable nuevo_pedido y accederemos a la columna ítems y aquí usaremos el método . append para agregar los detalles del pedido a nuestro pedido, agregaremos a las tablas haciendo un .session.add(nuevo_pedido) y haremos un db.session.commit() para confirmar los cambios y redireccionaremos hacia la pagina order.html.

```python
#ruta para agregar un nuevo pedido
@app.route("/add_pedido", methods=["GET","POST"])
@login_required
def add_pedido():
    producto_id = request.form.get('producto_id')
    if not producto_id:
        flash("No se seleccionó ningún producto.", "danger")
        return redirect(url_for('index'))

    producto = Producto.query.get_or_404(producto_id)
    if request.method == "POST":
        nuevo_pedido=Pedido(
            id_usuario= current_user.id,
            fecha_pedido=datetime.utcnow(),
            estado="pendiente",
        )

        detalle = DetallePedido(
            id_producto=producto.id,
            cantidad=1,
            precio_unitario=producto.precio
        )

        nuevo_pedido.items.append(detalle)

        db.session.add(nuevo_pedido)
        db.session.commit()

        flash(f"Pedido de {producto.nombre} realizado con éxito.", "success")
        return redirect(url_for("search_order"))
```

## Función agregando_orden

La ruta donde estará la función agregando_order será /agregando_order que aceptara tanto POST como GET si acceden a través del método GET, lo que haremos será primero acceder al id de la orden a través del URL ahi nos indicará el id de la orden, una vez tengamos el id de la orden haremos una variable para buscar dicho id en nuestra tabla Pedidos de no conseguirlo le mostrará un error 404, luego compararemos el id del usuario actual con el id del usuario que hizo el pedido, si no son iguales le mostraremos un mensaje que su usuario no está autorizado a hacer esa acción mostraremos un error 403 y lo redireccionaremos al index, se creará una variable que almacene todos los productos en la tabla Productos y luego redireccionaremos al usuario a la pagina add_order.html y le pasaremos las variables pedido=pedido y productos=productos para poder visualizarlo en el frontend a través de jinja2, si el usuario llega a esta ruta vía POST, creamos 2 variables una order_id y otra product_id y accederemos a los 2 inputs que están hidden en la página add_order.html, el value de estos inputs será order.id y producto.id y accederemos a él mediante el nombre que le asignamos, haremos un try para que en caso de que el usuario asignó la cantidad de productos que iba a querer convertiremos el dato en int ya que queremos un número, en caso de no indicar cuantos quiere comprar se asumirá que solo 1, si no hay order_id o product_id le indicaremos que faltan datos y lo redireccionaremos a la index

```python
@app.route('/add_order', methods=['GET','POST'])
@login_required
def agregando_orden():
    # GET: Mostrar productos para agregar a la orden indicada
    if request.method == 'GET':
        order_id = request.args.get('order_id')
        if not order_id:
            flash('Orden no especificada.', 'danger')
            return redirect(url_for('search_order'))
        pedido = Pedido.query.get_or_404(order_id)
        if pedido.id_usuario != current_user.id:
            flash('No autorizado para editar esta orden.', 'danger')
            return redirect(url_for('index'))
        productos = Producto.query.all()
        return render_template('add_order.html', pedido=pedido, productos=productos
    # POST: Añadir el producto seleccionado a la orden
    order_id = request.form.get('order_id')
    product_id = request.form.get('product_id')
    try:
        cantidad = int(request.form.get('cantidad', 1))
    except (TypeError, ValueError):
        cantidad = 1

    if not order_id or not product_id:
        flash('Faltan datos para agregar el producto.', 'danger')
        return redirect(url_for('index'))
```

Crearemos una variable producto que lo que hará es acceder a nuestra tabla Producto y buscar si existe algún producto con el id que le pasamos desde la url, de no conseguirlo lanzará un error 404, creamos otra variable detalle para acceder a la tabla DetallePedido y crearemos un nuevo pedido agregando el id del producto la cantidad y el precio del producto, buscaremos la variable pedido y accederemos a la columna .ítems y usamos .append para agregar todos los detalles de los productos, por último haremos un session.add(pedido) para agregar el pedido a nuestra tabla y .commit() para confirmar los cambios, se le mostrará un mensaje con el nombre del producto y el id del pedido y lo redireccionaremos a orders.html.

```
286     producto = Producto.query.get_or_404(product_id)
287     detalle = DetallePedido(
288         id_producto=producto.id,
289         cantidad=cantidad,
290         precio_unitario=producto.precio
291     )
292     pedido.items.append(detalle)
293     db.session.add(pedido)
294     db.session.commit()
295     flash(f'Producto {producto.nombre} agregado a la orden #{pedido.id}.', 'success')
296     return redirect(url_for('search_order'))
```

## Función delete_order

Esta función se ejecutará en la ruta /delete_order/<int:id> y solo se podrá acceder a esta ruta a través del método POST, esta función requiere que el usuario inicie sesión, la función tendrá un argumento que será el id de la orden a eliminar, para esto creamos una variable llamada pedido que buscara en la tabla Pedido el id de dicho pedido, de no conseguirlo lanzará un error 404, luego creamos una condición si el id de usuario que creó el pedido no es igual que el id del usuario actual nos lanzará un error diciendo que no estamos autorizados para eliminar esta orden y lo redireccionaremos a la página orders.html, para eliminar dicha orden haremos un try y haremos el siguiente comando db.session.delete(pedido) y confirmaremos este cambio con .commit() y una vez eliminado redireccionamos al usuario a index.html y se le mostrará un mensaje indicando que se eliminó dicho pedido y si ocurre alguna excepción haremos un db.session.rollback() para cancelar los cambios, se redireccionará a index.html con un mensaje diciendo error al eliminar la orden.

```python
@app.route('/delete_order/<int:id>', methods=['POST'])
@login_required
def delete_order(id):
    pedido = Pedido.query.get_or_404(id)
    # Sólo permitir que el propietario elimine su pedido
    if pedido.id_usuario != current_user.id:
        flash('No autorizado para eliminar esta orden.', 'danger')
        return redirect(url_for('ordenes'))
    try:
        db.session.delete(pedido)
        db.session.commit()
        flash('Orden eliminada con éxito.', 'success')
        return redirect(url_for('index'))
    except Exception as e:
        db.session.rollback()
        flash(f'Error al eliminar la orden: {e}', 'danger')
    return redirect(url_for('index'))

```

Para terminar ejecutamos el código siguiente para cuando la app se inicie detecte si la base de datos ya está creada, de no ser asi creara dicha base de datos.

```python
if __name__ == '__main__':
    # Ejecutamos la aplicación
    if not database_exists(app.config['SQLALCHEMY_DATABASE_URI']):
        create_database(app.config['SQLALCHEMY_DATABASE_URI'])
        print("Base de datos creada.")

    with app.app_context():
        db.create_all() # Crea las tablas si no existen



    app.run(debug=True)
```

## Templates

**Index.html**: si inicio sesion mostraremos el nombre del usuario, así como los productos que estén almacenados en nuestra db, contará con 3 links que te enviaran hacia la ruta /search, /orders, /add, además de 3 formularios uno para acceder a la ruta /delete/{{producto.id}}, /modificar/{{producto.id}}, /add_pedido en esta última habrá un input hidden para pasarle el id del producto a dicho formulario para ejecutar la función.

**add_order.html**: tendremos un link que nos redireccionará a order.html, además de un formulario para acceder a la función agregando_orden, para que esta función pueda realizar su tarea tendremos 2 inputs hidden los cuales le pasaran el id de la orden así como el id del producto.

**layout.html**: Acá tendremos la estructura principal de las páginas html la cual se le transferirá el formato a través de jinja2, contará una barra de navegación con 5 links que serán: /login, /logout, /add, /orders, /search.

**login.html**: Tendrá un formulario y dentro de ese formulario habrán 2 inputs uno para el email y otro para la contraseña, además de un link para enviar al usuario a registrarse.

**modificar.html**: Tendrá un formulario y en ese formulario tendrá 3 inputs uno para el nombre del producto, otro para el precio, para la cantidad de productos que compraste y por último una imagen del producto, todos estos inputs tendrán la información del producto.

**ordenes.html**: Mostrará todas las características de los productos que están en la orden, en caso de ingresar a esta pagina y no hay orden se le mostrará un mensaje indicando que no hay productos en la orden, además habrá 2 botones uno que realizará la acción de eliminar los datos de la orden y otro para agregar cantidad de productos a la orden.

**register_product.html**: Tendrá un formulario el cual estará compuesto por 4 inputs, uno para el nombre del producto, otro para el precio del producto, uno para la cantidad de productos y uno para ingresar una imagen del producto, por último tendremos un boton que dira registrar y será tipo submit esto lo que hará es enviar la información a la función /add.

**register.html**: Tendrá un formulario el cual estará compuesto de 3 inputs uno para agregar el nombre del usuario, otro para su contraseña y uno para el email, además de un botón que mandará la información para poder almacenarla.

**search.html**: Es un formulario con un input que lo que hace, es buscar en la tabla productos que producto coincide con las palabras escritas, de haber alguno lo mostrará y podrás agregar el producto y si eres su creador poder modificarlo o eliminarlo.