

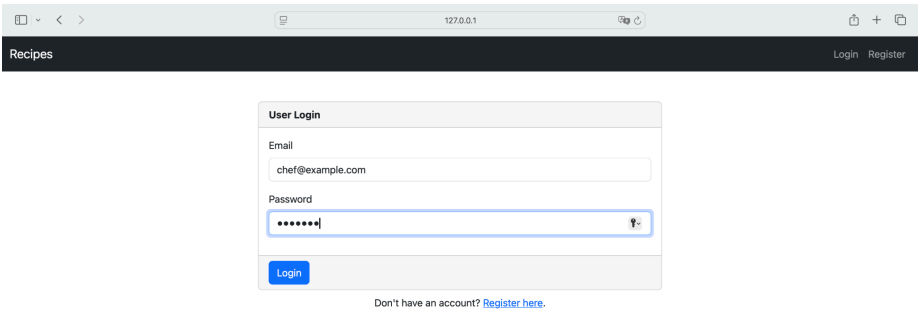
UIPR Arecibo Campus
Programa de Ciencias en Computadoras

Recetas Culinarias
COMP2052 microserv

Saúl Medina Vega
Yael Maldonado Díaz

Capturas de pantalla del lado visual

Página Login



Esta página es la página de hacer login. Cuenta con dos cajas de input en donde se ingresarán tanto el email como el password. Vemos que la página cuenta con un navbar, esto se encuentra también en todas las páginas de la aplicación de recetas. Pero vemos que no tiene todas las funcionalidades. Esto se debe a que aún no hay un usuario logged in. Cuando se haga login el usuario entonces más opciones pueden aparecer.

Página de manejo de recetas del chef

Dashboard

sopa de lentejas at DuckDuckGo

Recipes

DashboardChange PasswordLogout

Recipe Management

Now

Name	Ingredients	Instructions	Time	Portions	Image	Category	Creator	Actions
Pollo a la Plancha con Vegetales Salteados	Pechugas de pollo, brócoli, zanahorias, pimientos, aceite de oliva, sal, pimienta, ajo en polvo.	Cortar el pollo en filetes. Sazonar con sal, pimienta y ajo en polvo. Cocinar el pollo en una sartén con un poco de aceite hasta que esté dorado y cocido por dentro. Cortar los vegetales. En la misma sartén, saltear los vegetales con un poco más de aceite hasta que estén tiernos pero crujientes. Servir el pollo con los vegetales.	25 min	3		Saludable	John Doe	<div><div></div><div></div></div>
Sopa de Lentejas	Lentejas pardinas, cebolla, zanahoria, apio, ajo, tomate triturado, caldo de verduras, pimentón dulce, laurel, aceite de oliva, sal, pimienta.	Remojar las lentejas si es necesario (según el paquete). Picar finamente la cebolla, zanahoria y apio. Sofreír en una olla grande con aceite de oliva. Añadir el ajo picado y el pimentón. Incorporar el tomate triturado, las lentejas, el caldo de verduras y la hoja de laurel. Llevar a ebullición y luego reducir el fuego. Cocinar a fuego lento hasta que las lentejas estén tiernas (aproximadamente 30-40 minutos). Sazonar con sal y pimienta al gusto.	50 min	5		Sopa	John Doe	<div><div></div><div></div></div>

En la página de manejo de las recetas el chef puede añadir, editar o borrar recetas a su gusto con los botones respectivos. El chef puede añadirle un nombre, lista de ingredientes, una descripción, el tiempo de preparación, para cuantas personas es y finalmente una foto. Cuando se guarda una receta se le aplica el nombre del chef para que otro user en modo usuario pueda ver de quién es esa receta. Solo los chefs pueden añadir recetas, los usuarios no pueden. También vemos que el navbar también ahora tiene opciones adicionales como para cambiar su password o volver al dashboard, que es donde se encuentra ahora mismo.

Lista de Usuarios



Username	Email	Role
Administrator	admin@example.com	Admin
John Doe	chef@example.com	Chef
Saul Medina	sauljmedina@icloud.com	Chef
Steve Jobs	user@example.com	Usuario

Finalmente tenemos la pantalla de la lista de usuarios registrados. Esto es un panel accesible solo por el admin del sistema por razones de seguridad. Aquí el admin puede ver que usuario hay registrados en el sistema, tanto usuarios regulares y chefs como el mismo admin

Código CRUD principal

Leer y mostrar las recetas almacenadas de la base de datos

```
@main.route('/dashboard')
@login_required
def dashboard():
    if current_user.role.name == 'Usuario':
        recetas = Receta.query.all()
    else:
        recetas =
Receta.query.filter_by(usuario_id=current_user.id).all()

    return render_template('dashboard.html',
recetas=recetas)
```

Muestra el panel principal (dashboard) solo a usuarios autenticados. Si el usuario tiene el rol de “Usuario”, puede ver todas las recetas disponibles. En cambio, si tiene otro rol (como “Chef” o “Admin”), verá solo las recetas que él mismo ha creado. Al final, se renderiza la plantilla dashboard.html con la lista de recetas correspondientes.

Crear una receta nueva

```
@main.route('/recetas', methods=['GET', 'POST'])
@login_required
def recetas():
    form = RecetaForm()
    if form.validate_on_submit():
        receta = Receta(
            nombre=form.nombre.data,
            ingredientes=form.ingredientes.data,
            instrucciones=form.instrucciones.data,
```

```

tiempo_preparacion=form.tiempo_preparacion.data,
        porciones=form.porciones.data,
        imagen_url=form.imagen_url.data,
        categoria=form.categoria.data,
        usuario_id=current_user.id
    )
    db.session.add(receta)
    db.session.commit()
    flash("Recipe created successfully.") # 🔄
Traducido
    return redirect(url_for('main.dashboard'))

    return render_template('receta_form.html', form=form)

```

Este endpoint permite a un usuario autenticado crear una nueva receta. Si el formulario (RecetaForm) es enviado correctamente (POST) y pasa la validación, se crea una instancia del modelo Receta con los datos ingresados y se guarda en la base de datos, asociándola al usuario actual. Luego, redirige al dashboard. Si el formulario aún no ha sido enviado o contiene errores, se renderiza el formulario en la plantilla receta_form.html.

Editar o actualizar recetas ya existentes

```

@main.route('/recetas/<int:id>/editar', methods=['GET',
'POST'])
@login_required
def editar_receta(id):
    receta = Receta.query.get_or_404(id)

    # Validación de permisos
    if current_user.role.name not in ['Admin', 'Chef'] or
(

```

```

        receta.usuario_id != current_user.id and
current_user.role.name != 'Admin'):
        flash('You do not have permission to edit this
recipe.') # 🔄 Traducido
        return redirect(url_for('main.dashboard'))

form = RecetaForm(obj=receta)

if form.validate_on_submit():
    receta.nombre = form.nombre.data
    receta.ingredientes = form.ingredientes.data
    receta.instrucciones = form.instrucciones.data
    receta.tiempo_preparacion =
form.tiempo_preparacion.data
    receta.porciones = form.porciones.data
    receta.imagen_url = form.imagen_url.data
    receta.categoria = form.categoria.data
    db.session.commit()
    flash("Recipe updated successfully.") # 🔄
Traducido
    return redirect(url_for('main.dashboard'))

return render_template('receta_form.html', form=form,
editar=True)

```

Permite a un usuario autenticado editar una receta existente cuyo ID se pasa en la URL. Primero busca la receta en la base de datos. Luego verifica si el usuario tiene permisos para editarla: solo los admins o Chefs pueden editar, y los Chefs solo pueden hacerlo si la receta les pertenece. Si el formulario es enviado y válido, se actualizan los campos de la receta con los nuevos datos del formulario y se guardan en la base de datos. Finalmente, redirige al dashboard. Si la solicitud es GET o el formulario no es

válido, se muestra el formulario de edición precargado con los datos actuales de la receta.

Borrar recetas

```
@main.route('/recetas/<int:id>/eliminar',
methods=['POST'])
@login_required
def eliminar_receta(id):
    receta = Receta.query.get_or_404(id)

    if current_user.role.name not in ['Admin', 'Chef'] or
(
    receta.usuario_id != current_user.id and
current_user.role.name != 'Admin'):
        flash('You do not have permission to delete this
course.') # 🔄 Traducido
        return redirect(url_for('main.dashboard'))

    db.session.delete(receta)
    db.session.commit()

    flash("Recipe deleted successfully.") # 🔄 Traducido
    return redirect(url_for('main.dashboard'))
```

Permite a un usuario autenticado eliminar una receta específica mediante una solicitud POST. Primero busca la receta por su ID. Luego, verifica los permisos: solo los usuarios con rol Admin o Chef pueden eliminar recetas, y los Chefs solo si la receta las pertenece. Si no tiene permiso, se muestra un mensaje de error y se redirige al dashboard. Si tiene permiso, se elimina la receta de la base de datos, se confirma con un mensaje, y se redirige de nuevo al dashboard.

Tabla de archivos de prueba CRUD

Aquí tenemos una tabla de cómo deben funcionar las pruebas de los endpoints CRUD de nuestro proyecto. Tenemos el Endpoint, los valores que se le estarán enviando, los valores esperados y una breve descripción de la prueba


Endpoint	Valores enviados	Valores esperados	Descripción breve
/recetas GET	GET a esa dirección	200 con la lista de recetas	Para ver si las recetas se cargan correctamente
/recetas POST	POST con la info de la receta que se va añadir en json.	201, receta creada!	Si se crean recetas correctamente
/recetas/<int:id>/editar PUT	PUT con la info para actualizar la receta.	200 receta actualizada!	Para ver si se logra actualizar las recetas
/recetas/<int:id>/eliminar DELETE	Solo el DELETE	200 receta eliminada!	Probar la eliminación de las recetas

Pruebas de los end-points

read.rest

The screenshot displays a REST client interface with two tabs: 'Request' and 'Response'. The 'Request' tab shows a GET request to `http://localhost:5000/recetas` with the header `Content-Type: application/json`. The 'Response' tab shows a successful response with status `200 OK` and headers: `Server: Werkzeug/3.1.3 Python/3.13.3`, `Date: Thu, 15 May 2025 21:16:01 GMT`, `Content-Type: application/json`, `Content-Length: 1708`, and `Connection: close`. The response body is a JSON array containing two recipe objects. The first object is for 'Fried' wings, and the second is for 'humano' (human) wings. Both objects include an image URL, ingredients, instructions, name, portions, preparation time, and a user ID.

create.rest

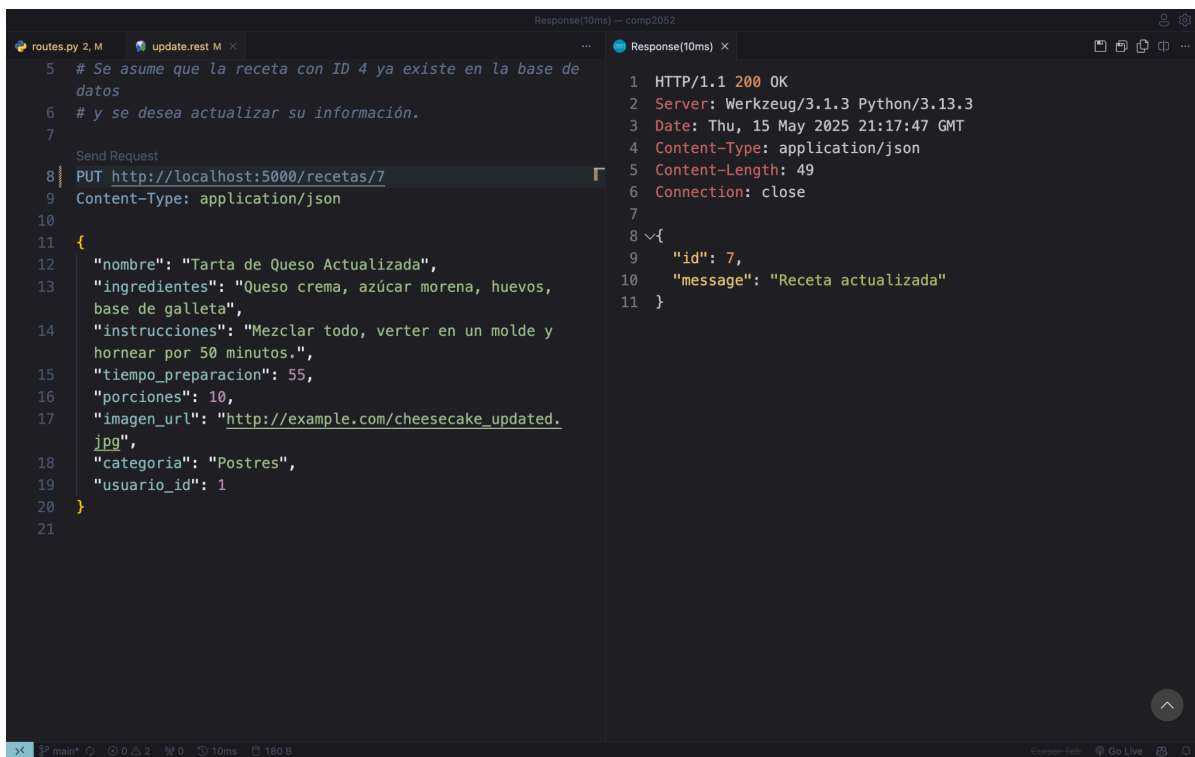


The screenshot shows a VS Code editor with two tabs: 'routes.py 2, M' and 'create.rest x'. The 'create.rest' tab is active and displays a REST client request. The request is a POST to 'http://localhost:5000/recetas' with a 'Content-Type: application/json' header. The body is a JSON object representing a recipe. To the right, a 'Response(11ms) x' tab shows the server's response, which is an HTTP 201 status code with headers indicating the server is Werkzeug/3.1.3 Python/3.13.3, the date is Thu, 15 May 2025 21:10:12 GMT, the content type is application/json, and the content length is 63. The response body is a JSON object with 'id': 8, 'message': 'Receta creada', and 'usuario_id': 1.

```
1  ### Crear una nueva receta (POST)
2
3  Send Request
4  POST http://localhost:5000/recetas
5  Content-Type: application/json
6  {
7    "nombre": "Tarta de Queso",
8    "ingredientes": "Queso crema, azúcar, huevos, base de
9    galleta",
10   "instrucciones": "Mezclar los ingredientes, verter en
11   un molde y hornear por 45 minutos.",
12   "tiempo_preparacion": 60,
13   "porciones": 8,
14   "imagen_url": "http://example.com/cheesecake.jpg",
15   "categoria": "Postres",
16   "usuario_id": 1
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
1  HTTP/1.1 201 CREATED
2  Server: Werkzeug/3.1.3 Python/3.13.3
3  Date: Thu, 15 May 2025 21:10:12 GMT
4  Content-Type: application/json
5  Content-Length: 63
6  Connection: close
7
8  {
9    "id": 8,
10   "message": "Receta creada",
11   "usuario_id": 1
12 }
```

update.rest



The screenshot shows a VS Code editor with two tabs: 'routes.py 2, M' and 'update.rest M'. The 'update.rest' tab is active, displaying a REST client request and its response.

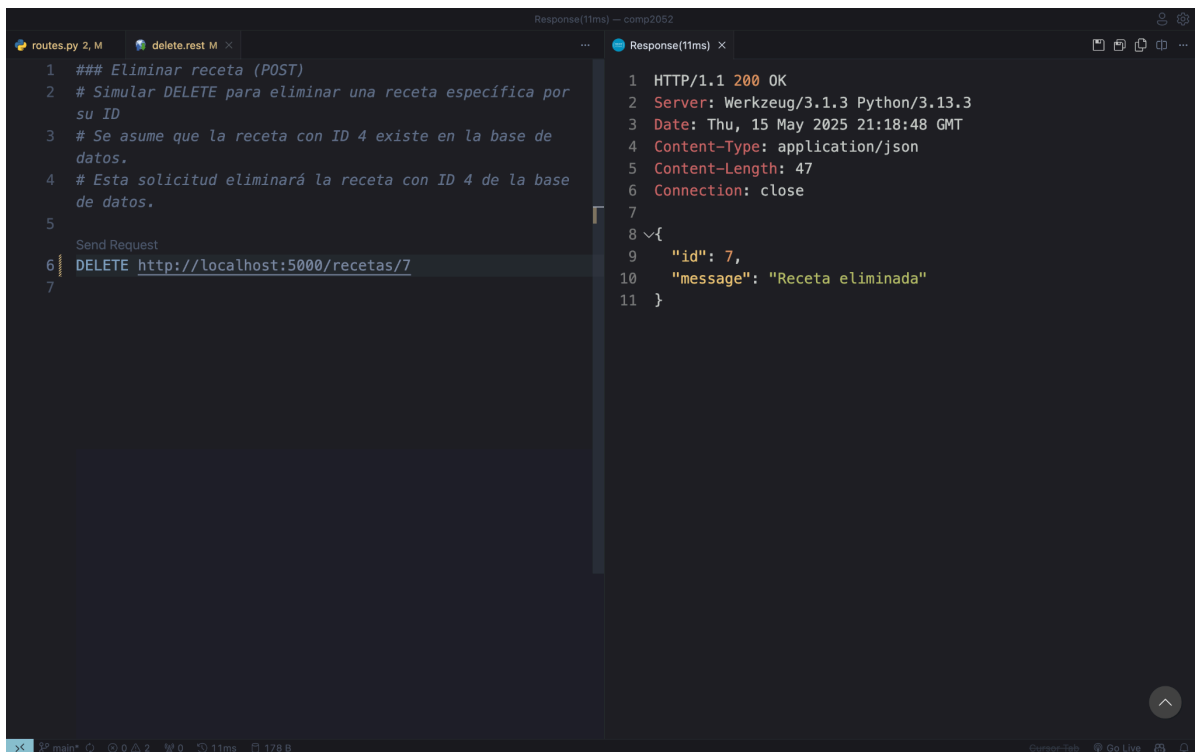
Request:

```
5 # Se asume que la receta con ID 4 ya existe en la base de
6 # y se desea actualizar su información.
7
8 PUT http://localhost:5000/recetas/7
9 Content-Type: application/json
10
11 {
12   "nombre": "Tarta de Queso Actualizada",
13   "ingredientes": "Queso crema, azúcar morena, huevos,
14     base de galleta",
15   "instrucciones": "Mezclar todo, verter en un molde y
16     hornear por 50 minutos.",
17   "tiempo_preparacion": 55,
18   "porciones": 10,
19   "imagen_url": "http://example.com/cheesecake_updated.
20     jpg",
21   "categoria": "Postres",
22   "usuario_id": 1
23 }
```

Response (10ms):

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.1.3 Python/3.13.3
3 Date: Thu, 15 May 2025 21:17:47 GMT
4 Content-Type: application/json
5 Content-Length: 49
6 Connection: close
7
8 {
9   "id": 7,
10  "message": "Receta actualizada"
11 }
```

delete.rest



The screenshot shows a VS Code editor with two tabs: 'routes.py 2, M' and 'delete.rest M'. The 'delete.rest' tab is active, displaying a REST client request and its response.

Request:

```
1 ## Eliminar receta (POST)
2 # Simular DELETE para eliminar una receta específica por
3 # su ID
4 # Se asume que la receta con ID 4 existe en la base de
5 # datos.
6 # Esta solicitud eliminará la receta con ID 4 de la base
7 # de datos.
8
9 Send Request
10 DELETE http://localhost:5000/recetas/7
11
```

Response (11ms):

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.1.3 Python/3.13.3
3 Date: Thu, 15 May 2025 21:18:48 GMT
4 Content-Type: application/json
5 Content-Length: 47
6 Connection: close
7
8 {
9   "id": 7,
10  "message": "Receta eliminada"
11 }
```

Repositorios de los integrantes

Saul Medina Vega

https://github.com/saulbruh/proyecto_final_microserv

Yael Maldonado Díaz

https://github.com/Yaelito77/MicroServ_Proyecto_Final