

UIPR – Recinto de Arecibo
Programa de Ciencias de computadoras

Tabla de Artículos

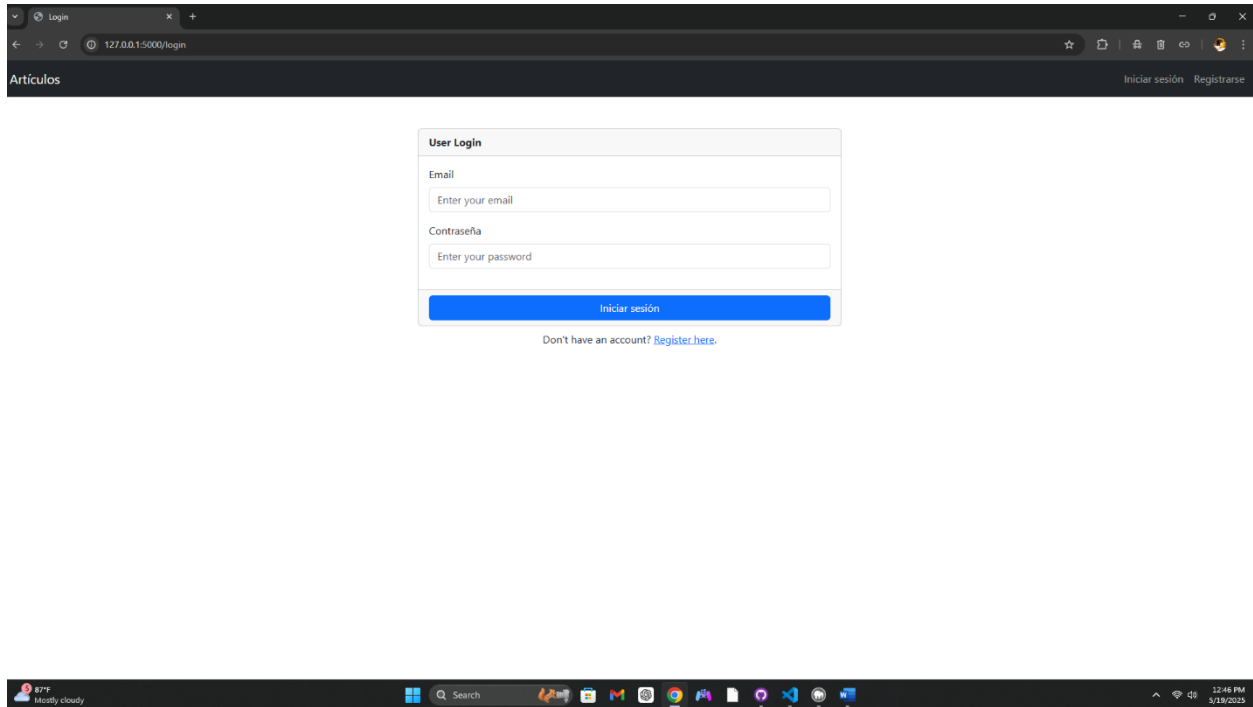
Web Development service – Side & Microservice

Cristian Meléndez López

E00650174

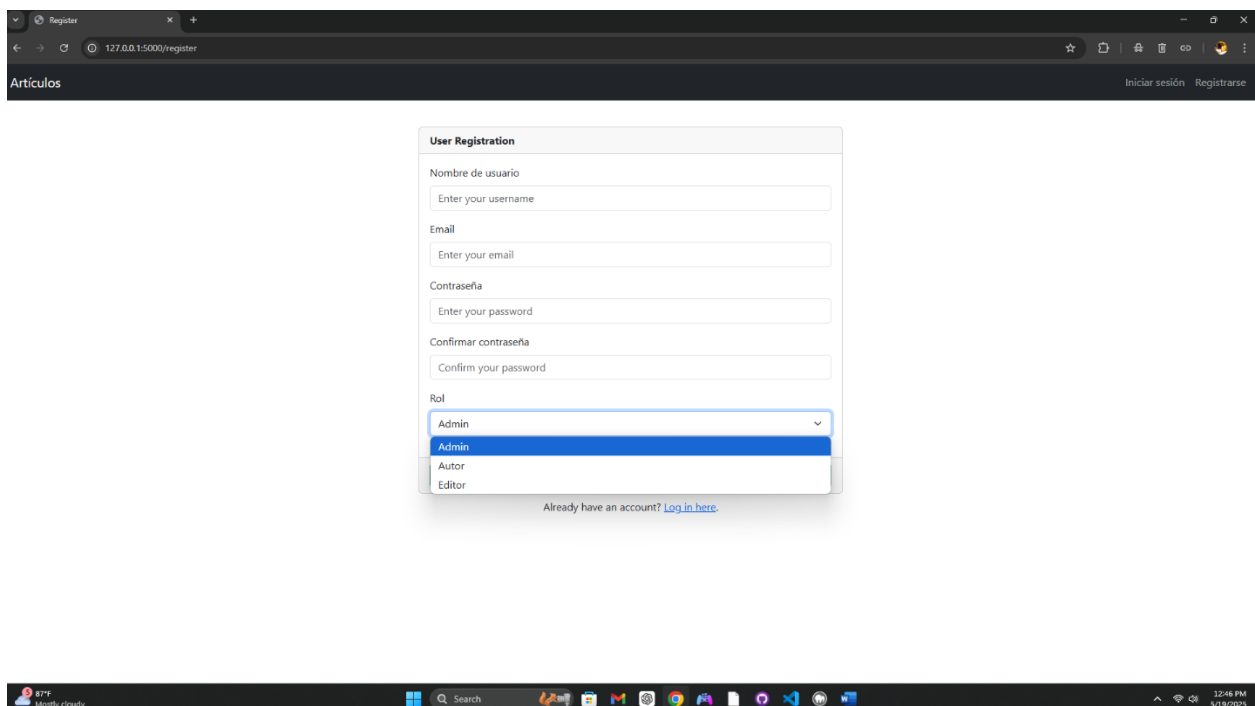
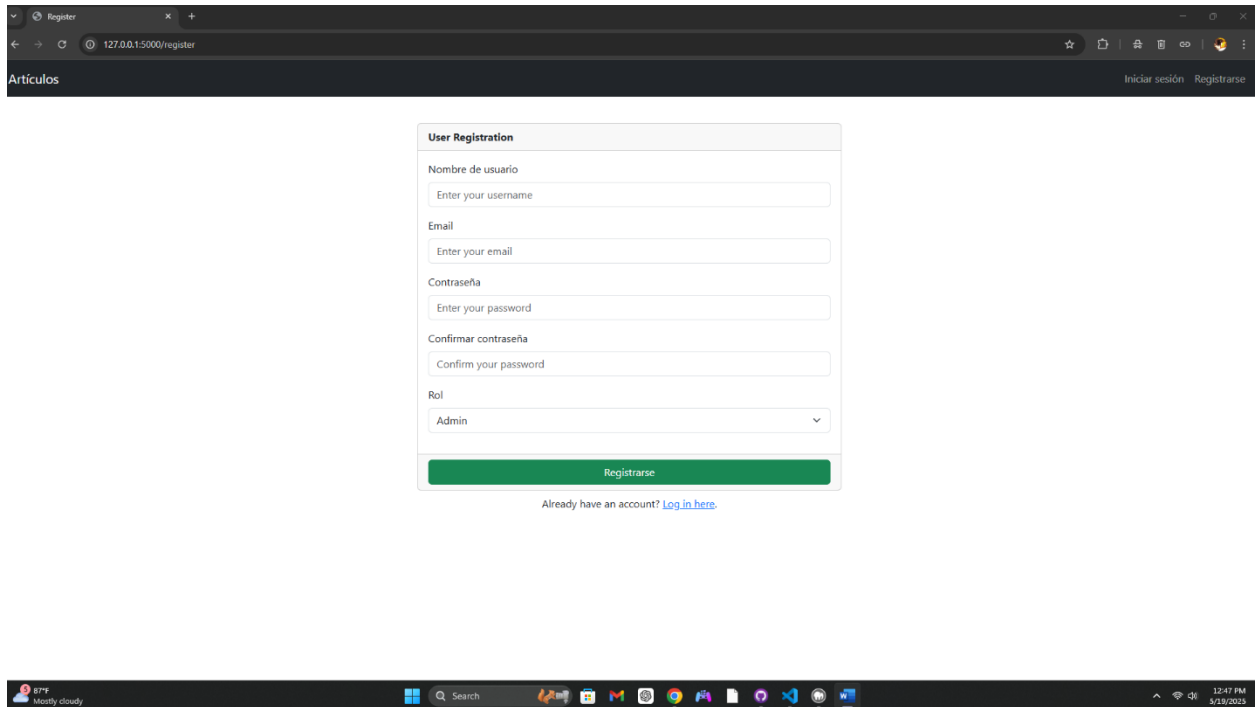
Tabla de Artículos

Inicio de sesión



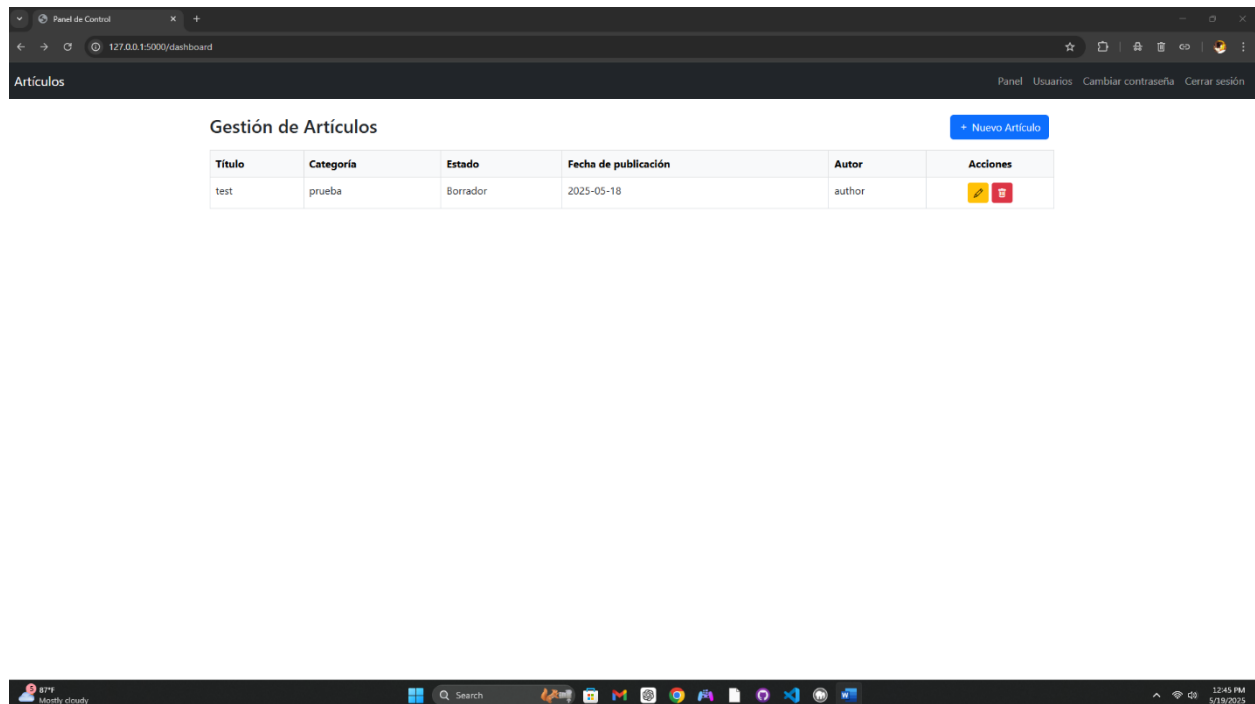
En la imagen se muestra la pantalla de inicio de sesión de la aplicación web "Artículos". El usuario ingresa su correo electrónico y contraseña en el formulario central para acceder al sistema. Al hacer clic en "Iniciar sesión", los datos se envían al backend, donde se validan usando el formulario LoginForm y el modelo User definidos en el código (app/forms.py, app/models.py). Si las credenciales son correctas, el usuario es autenticado mediante Flask-Login y redirigido al panel principal, como se implementa en la ruta /login del blueprint de autenticación (app/auth_routes.py). Si los datos son incorrectos, se muestra un mensaje de error y el usuario permanece en la misma página.

Registro de usuario



En las siguientes imágenes se muestra la pantalla de registro de usuario. El formulario permite al usuario ingresar su nombre, correo electrónico, contraseña, confirmar la contraseña y seleccionar un rol (Admin, Autor o Editor) antes de registrarse. Al hacer clic en "Registrarse", los datos se envían al backend, donde el formulario RegisterForm valida la información y el rol seleccionado (app/forms.py). Si la validación es exitosa, se crea un nuevo usuario en la base de datos con el rol elegido usando el modelo User y la relación con Role (app/models.py). Todo el proceso se gestiona con la ruta /register en el blueprint de autenticación (app/auth_routes.py), que muestra mensajes de éxito o error según corresponde.

Gestión de Artículos



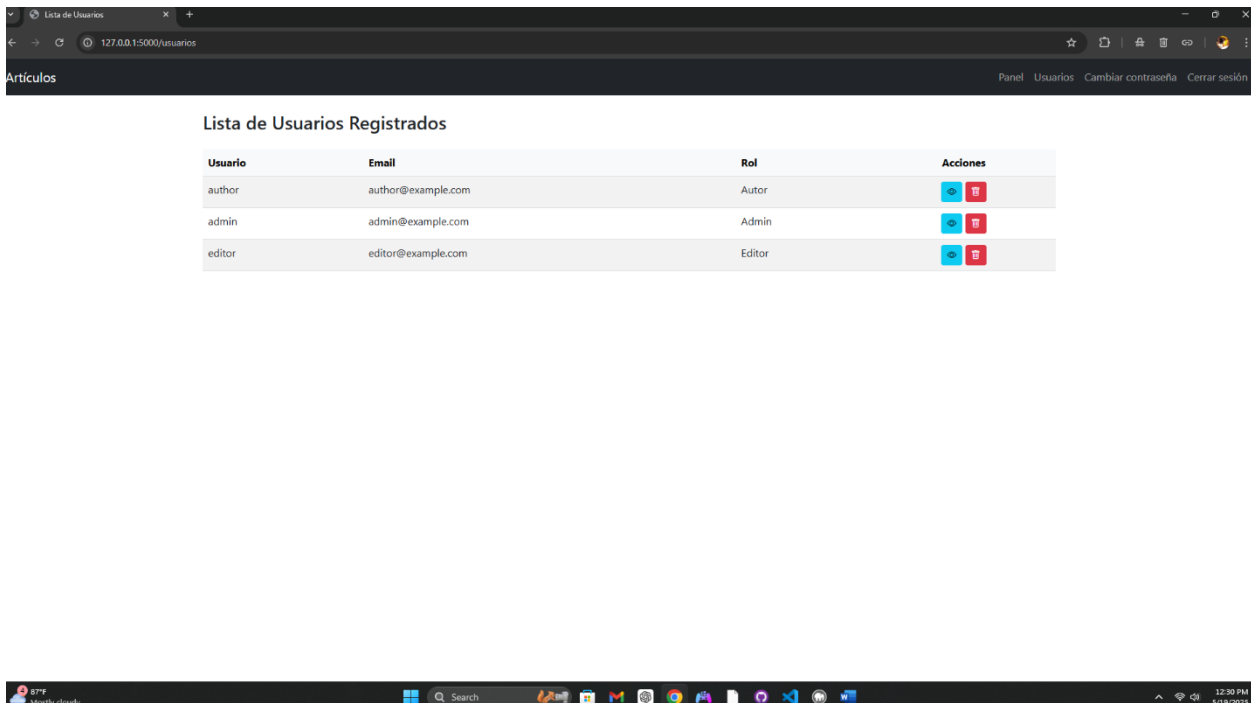
En la siguiente imagen se muestra el panel de control donde los usuarios pueden gestionar los artículos publicados. Se presenta una tabla con información relevante de cada artículo, como el título, categoría, estado, fecha de publicación y autor. Los botones de acciones permiten editar o eliminar artículos, funcionalidades disponibles solo para administradores o el propio autor, según las reglas del sistema. Esta vista se genera usando la ruta /dashboard definida en el blueprint principal (app/routes.py), que consulta los artículos desde la base de datos y los pasa a la plantilla dashboard.html. El código utiliza el modelo Artículo para obtener los datos y verifica el rol del usuario con métodos como "is_admin()" para mostrar las opciones adecuadas en la interfaz.

Edición de artículo

The screenshot displays a web browser window with the address bar showing '127.0.0.1:5000/articles/2/edit'. The page title is 'Artículos'. The main content area is titled 'Editar Artículo'. It contains three input fields: 'Título' (Title) with the value 'test', 'Categoría' (Category) with the value 'prueba', and 'Contenido' (Content) with the value 'prueba'. Below the fields are two buttons: 'Guardar artículo' (Save article) and 'Cancelar' (Cancel). The top navigation bar includes 'Panel', 'Usuarios', 'Cambiar contraseña', and 'Cerrar sesión'.

En la siguiente imagen se muestra la pantalla para editar un artículo existente. El formulario permite modificar el título, la categoría y el contenido del artículo, mostrando los valores actuales para que el usuario los actualice según sea necesario. Al hacer clic en "Guardar artículo", los datos se envían al backend, donde el formulario "ArticleForm" valida la información y, si es correcta, se actualizan los campos del artículo en la base de datos usando SQLAlchemy (app/routes.py). Esta funcionalidad está implementada en la ruta " /articles/<int:id>/edit ", la cual carga el artículo por su ID y actualiza sus datos si el usuario tiene permisos. Finalmente, tras guardar los cambios, el usuario es redirigido al panel de control y muestra un mensaje de éxito.

Lista de usuarios



En la siguiente imagen se muestra la página de administración donde esta la lista de todos los usuarios registrados en la aplicación "Artículos". Cada fila de la tabla presenta el nombre de usuario, correo electrónico y el rol asignado (Autor, Admin o Editor). En la columna de acciones, los botones permiten ver los detalles de cada usuario o eliminarlos, funcionalidades reservadas para administradores. El código verifica que solo los administradores puedan acceder a la página y realizar acciones de gestión, usando el método "is_admin() "

Info del usuario

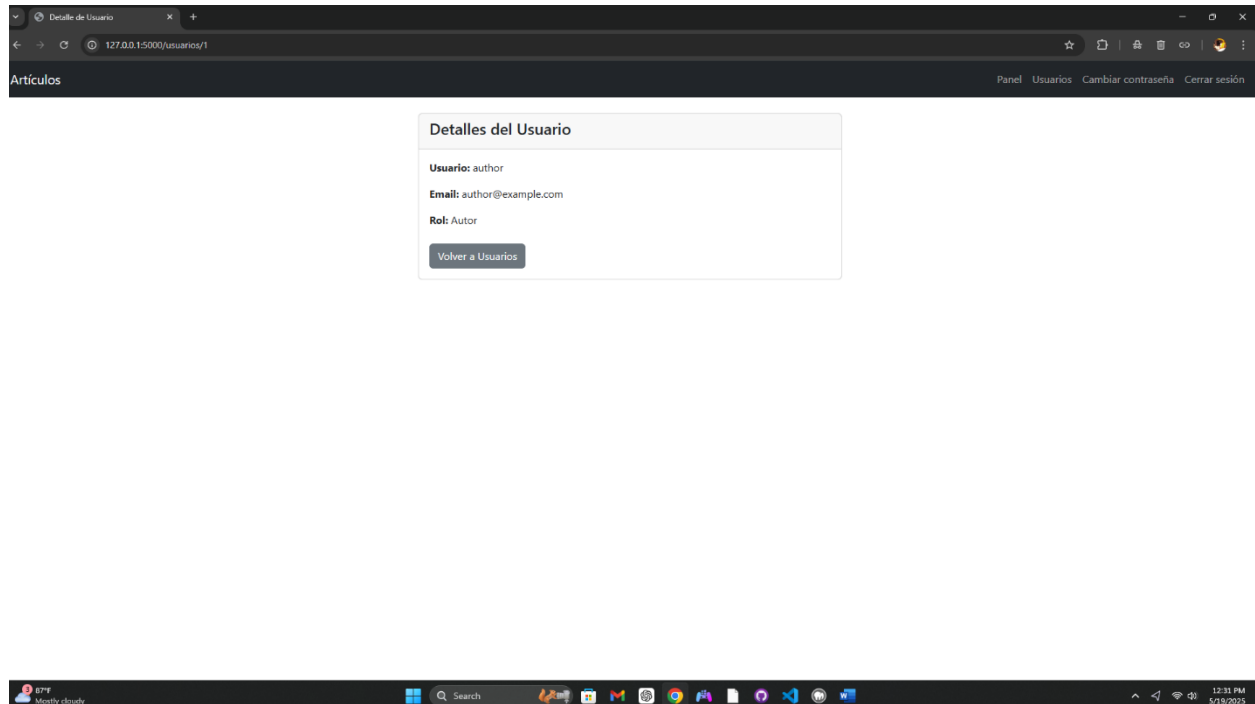
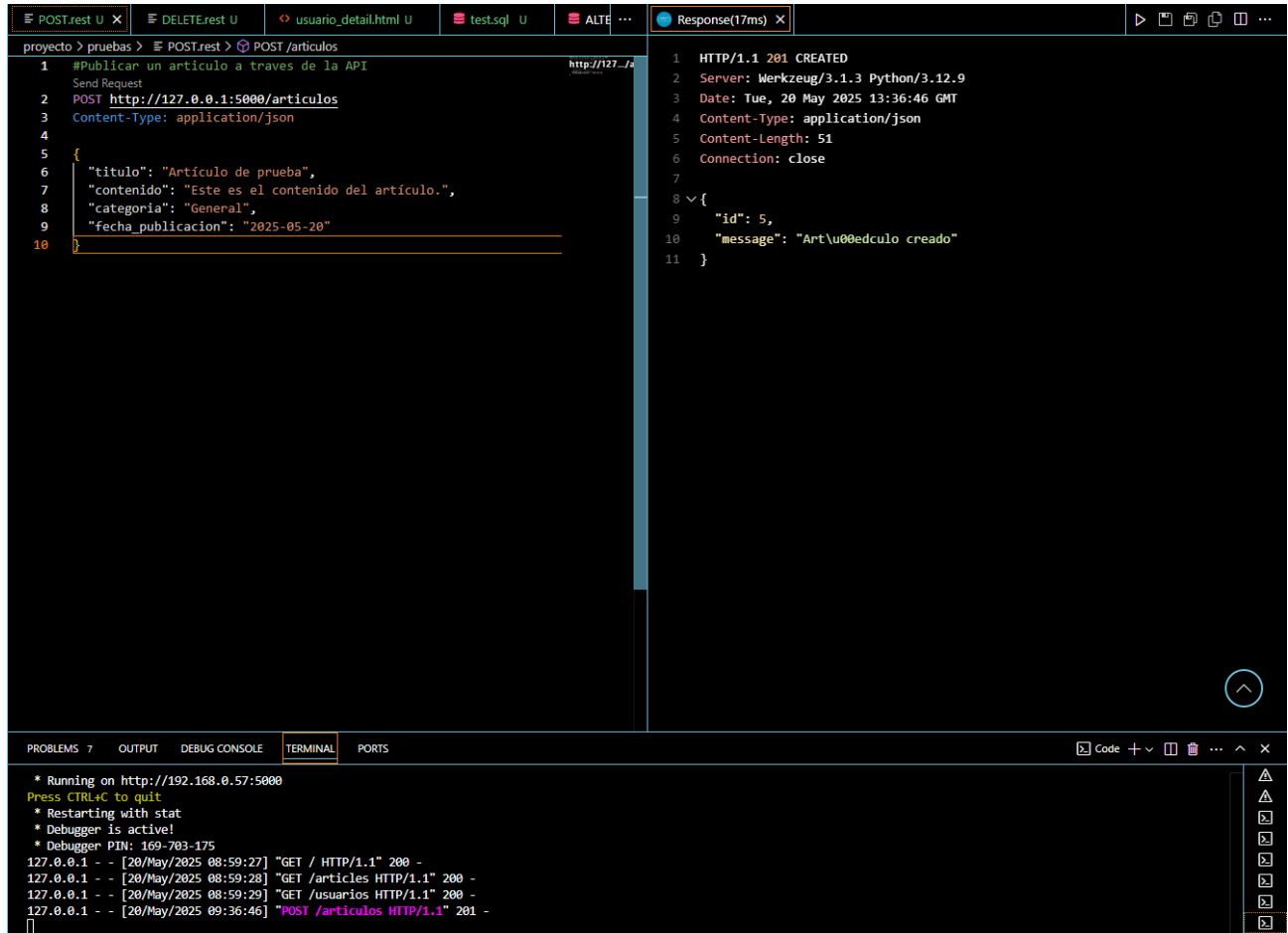


Imagen donde se muestra la página de detalles de un usuario específico. Se presenta la información básica del usuario seleccionado, incluyendo su nombre de usuario, correo electrónico y el rol que tiene asignado en el sistema.

CRUD

POST.rest



The screenshot shows the VS Code interface with a REST client file named `POST.rest` open. The editor displays a REST client request for a POST endpoint at `http://127.0.0.1:5000/articulos`. The request body is a JSON object with the following fields: `titulo`, `contenido`, `categoria`, and `fecha_publicacion`. The response is a 201 status code with a JSON body containing `id` and `message`.

```
1 #Publicar un articulo a traves de la API
2 Send Request
3 POST http://127.0.0.1:5000/articulos
4 Content-Type: application/json
5
6 {
7   "titulo": "Articulo de prueba",
8   "contenido": "Este es el contenido del articulo.",
9   "categoria": "General",
10  "fecha_publicacion": "2025-05-20"
11 }
```

```
1 HTTP/1.1 201 CREATED
2 Server: Werkzeug/3.1.3 Python/3.12.9
3 Date: Tue, 20 May 2025 13:36:46 GMT
4 Content-Type: application/json
5 Content-Length: 51
6 Connection: close
7
8 {
9   "id": 5,
10  "message": "Art\u00edculo creado"
11 }
```

The terminal at the bottom shows the output of the REST client, including the status of the request and the response body.

```
* Running on http://192.168.0.57:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 169-703-175
127.0.0.1 - - [20/May/2025 08:59:27] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/May/2025 08:59:28] "GET /articles HTTP/1.1" 200 -
127.0.0.1 - - [20/May/2025 08:59:29] "GET /usuarios HTTP/1.1" 200 -
127.0.0.1 - - [20/May/2025 09:36:46] "POST /articulos HTTP/1.1" 201 -
```

En el archivo “ post.rest “ se realiza una petición POST a la API para crear un nuevo artículo en la aplicación. A la izquierda, se envía una solicitud HTTP con los datos del artículo en formato JSON, incluyendo título, contenido, categoría y fecha de publicación. En la derecha, se observa la respuesta del servidor con un mensaje confirmando que el artículo fue creado exitosamente, devolviendo el ID asignado. El proceso funciona debido a la ruta /articulos definida en el blueprint de la api, donde el método crear_articulo recibe los datos, crea un objeto Artículo, lo guarda en la base de datos y responde con un mensaje en JSON.

Codigo

```
#Publicar un articulo a traves de la API
```



```

POST http://127.0.0.1:5000/articulos
Content-Type: application/json

{
  "titulo": "Artículo de prueba",
  "contenido": "Este es el contenido del artículo.",
  "categoria": "General",
  "fecha_publicacion": "2025-05-20"
}

```

GET.rest

The screenshot displays the GET.rest application interface. On the left, a list of requests is shown:

```

1 #Obtener lista de artículos
2
3 GET http://127.0.0.1:5000/articulos
4
5 #Obtener un artículo por id
6 GET http://127.0.0.1:5000/articulos/1

```

The right panel shows the details of the selected GET request (line 3):

```

1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.1.3 Python/3.12.9
3 Date: Tue, 20 May 2025 13:44:46 GMT
4 Content-Type: application/json
5 Content-Length: 574
6 Connection: close
7
8 [
9   {
10    "categoria": "prueba",
11    "contenido": "prueba",
12    "fecha_publicacion": "Sun, 18 May 2025 00:00:00 GMT",
13    "id": 2,
14    "titulo": "test"
15  },
16  {
17    "categoria": "General",
18    "contenido": "Este es el contenido del artículo.",
19    "fecha_publicacion": "Tue, 20 May 2025 00:00:00 GMT",
20    "id": 3,
21    "titulo": "Artículo de prueba"
22  },
23  {
24    "categoria": "General",
25    "contenido": "Este es el contenido del artículo.",
26    "fecha_publicacion": "Tue, 20 May 2025 00:00:00 GMT",
27    "id": 5,
28    "titulo": "Artículo de prueba"
29  }
30 ]

```

The bottom panel shows the terminal output:

```

Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 169-703-175
127.0.0.1 - - [20/May/2025 08:59:27] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/May/2025 08:59:28] "GET /articles HTTP/1.1" 200 -
127.0.0.1 - - [20/May/2025 08:59:29] "GET /usuarios HTTP/1.1" 200 -
127.0.0.1 - - [20/May/2025 09:36:46] "POST /articulos HTTP/1.1" 201 -
127.0.0.1 - - [20/May/2025 09:44:46] "GET /articulos HTTP/1.1" 200 -

```

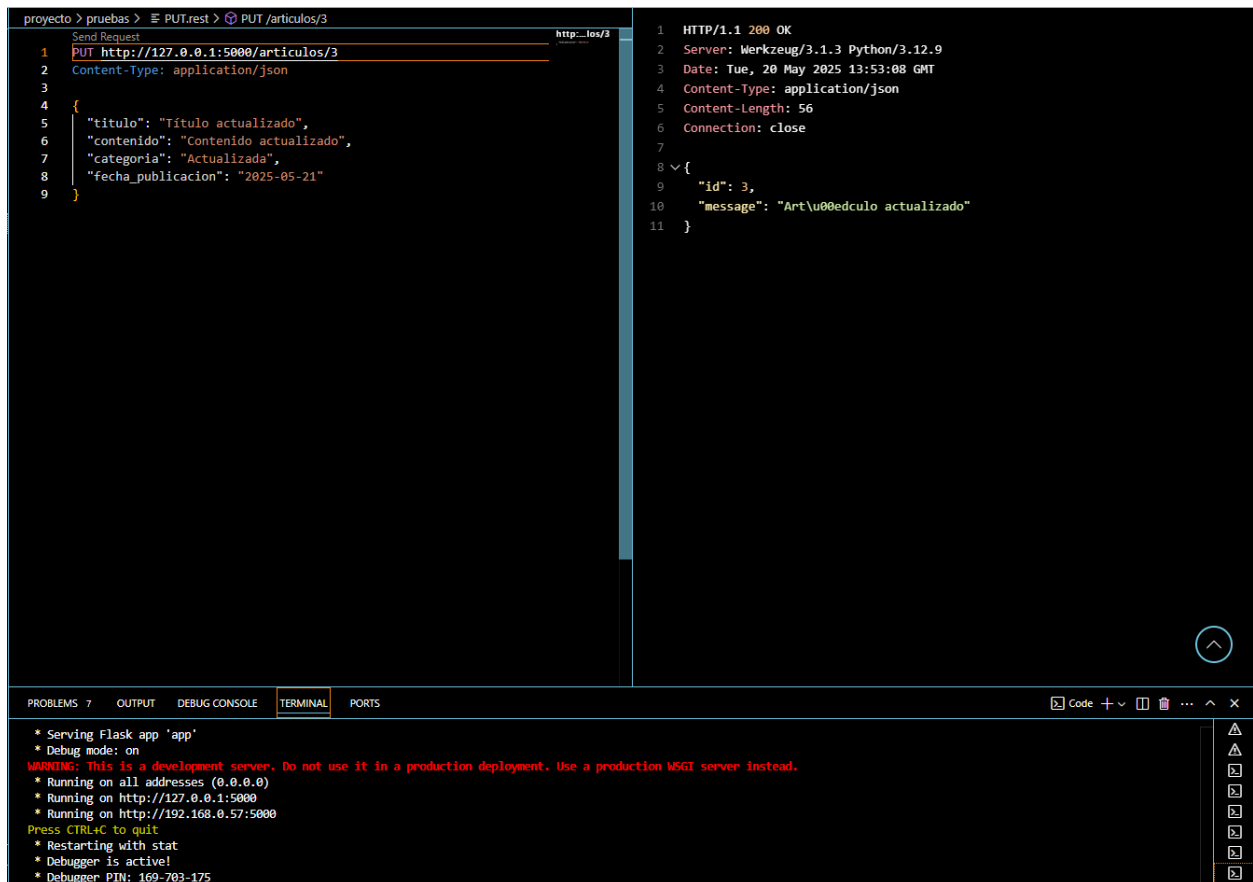
La petición GET a la API permite obtener la lista de artículos almacenados en la base de datos. El servidor devuelve un arreglo JSON con los datos de cada artículo, incluyendo id, título, contenido, categoría y fecha de publicación. La funcionalidad está implementada en la ruta “ /articulos ” del blueprint api, donde la función listar_articulos consulta todos los artículos usando el modelo Artículo y los retorna en formato JSON.

Codigo

```
#Obtener lista de articulos
GET http://127.0.0.1:5000/articulos

#Obtener un articulo por id
GET http://127.0.0.1:5000/articulos/1
```

PUT.rest



The screenshot shows a REST client interface with a dark theme. The top section is divided into two panes. The left pane shows the request details for a PUT request to `http://127.0.0.1:5000/articulos/3`. The request body is a JSON object: `{ "titulo": "Título actualizado", "contenido": "Contenido actualizado", "categoria": "Actualizada", "fecha_publicacion": "2025-05-21" }`. The right pane shows the response details for an `HTTP/1.1 200 OK` status. The response headers include `Server: Werkzeug/3.1.3 Python/3.12.9`, `Date: Tue, 20 May 2025 13:53:08 GMT`, `Content-Type: application/json`, and `Content-Length: 56`. The response body is a JSON object: `{ "id": 3, "message": "Art\u00e9culo actualizado" }`. The bottom section of the interface shows a terminal with the following output:

```
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.57:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 169-703-175
```

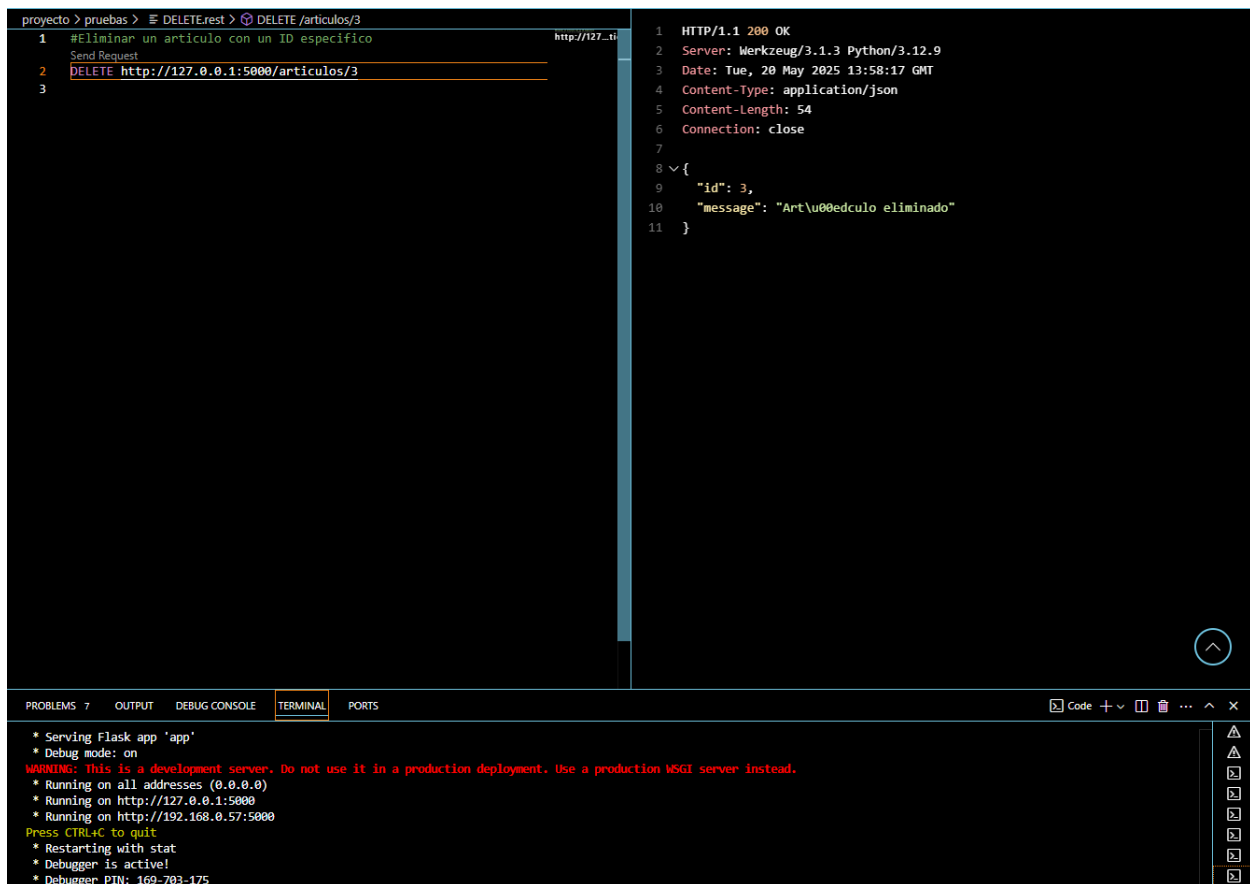
En la imagen se realiza la petición PUT a la API para actualizar un artículo existente con su respectivo ID. El commando envía una solicitud HTTP con los nuevos datos del artículo (título, contenido, categoría y fecha de publicación) al endpoint “ /articulos/3 ”, especificando el ID del artículo a modificar. A la derecha, se observa la respuesta del servidor, que confirma con un mensaje en formato JSON que el artículo fue actualizado correctamente y muestra el ID correspondiente.

Codigo

```
PUT http://127.0.0.1:5000/articulos/3
Content-Type: application/json

{
  "titulo": "Título actualizado",
  "contenido": "Contenido actualizado",
  "categoria": "Actualizada",
  "fecha_publicacion": "2025-05-21" }
```

DELETE.rest



The screenshot shows a VS Code editor with a REST client file named `DELETE.rest`. The file contains a DELETE request to `http://127.0.0.1:5000/articulos/3`. The response is an HTTP 200 OK with headers: `Server: Werkzeug/3.1.3 Python/3.12.9`, `Date: Tue, 20 May 2025 13:58:17 GMT`, `Content-Type: application/json`, and `Content-Length: 54`. The response body is a JSON object: `{ "id": 3, "message": "Art\u00edculo eliminado" }`. The bottom panel shows the terminal output of a Flask application running on `http://127.0.0.1:5000`.

```
proyecto > pruebas > DELETE.rest > DELETE /articulos/3
1 #Eliminar un artículo con un ID específico http://127...ti
  Send Request
2 DELETE http://127.0.0.1:5000/articulos/3
3

1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.1.3 Python/3.12.9
3 Date: Tue, 20 May 2025 13:58:17 GMT
4 Content-Type: application/json
5 Content-Length: 54
6 Connection: close
7
8 {
9   "id": 3,
10  "message": "Art\u00edculo eliminado"
11 }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.57:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 169-703-175
```

En el siguiente archivo, se envía una solicitud HTTP al endpoint “ /articulos/3 ”, indicando que se desea eliminar el artículo con el ID 3. A la derecha, se observa la respuesta del servidor, que confirma con un mensaje en formato JSON que el artículo fue eliminado correctamente y muestra el ID correspondiente.

Codigo

```
#Eliminar un articulo con un ID especifico  
DELETE http://127.0.0.1:5000/articulos/3
```