

# Documento de Pruebas de la Plataforma de Acceso Remoto en el Laboratorio

Brayan Stiven López Méndez

16 de Septiembre de 2024



# Contents

<b>1</b>	<b>Introducción</b>	<b>3</b>
1.1	Propósito . . . . .	3
1.2	Alcance . . . . .	3
<b>2</b>	<b>Estrategia de Pruebas</b>	<b>3</b>
2.1	Tipos de Pruebas . . . . .	3
2.2	Herramientas de Pruebas . . . . .	3
<b>3</b>	<b>Plan de Pruebas</b>	<b>4</b>
3.1	Pruebas del Frontend . . . . .	4
3.1.1	Funcionalidad de Control de Experimentos . . . . .	4
3.1.2	Transmisión de Video . . . . .	4
3.2	Pruebas del Backend . . . . .	4
3.2.1	API RESTful . . . . .	4
3.2.2	Integración con ROS . . . . .	4
3.3	Pruebas de Seguridad . . . . .	5
3.3.1	Autenticación . . . . .	5
<b>4</b>	<b>Resultados de las Pruebas</b>	<b>5</b>
4.1	Pruebas del Frontend . . . . .	6
4.1.1	Funcionalidad de Control de Experimentos . . . . .	6
4.1.2	Transmisión de Video . . . . .	6
4.2	Pruebas del Backend . . . . .	7
4.2.1	API RESTful . . . . .	7
4.2.2	Integración con ROS . . . . .	10
4.3	Pruebas de Seguridad . . . . .	11
4.3.1	Autenticación . . . . .	11

# 1 Introducción

## 1.1 Propósito

Este documento tiene como objetivo proporcionar un enfoque detallado para las pruebas realizadas en la plataforma software de acceso remoto. El propósito es asegurar que todos los componentes del sistema funcionen según lo esperado, identificando posibles defectos y verificando que el sistema cumpla con los requisitos definidos.

## 1.2 Alcance

Las pruebas cubrirán los siguientes componentes de la plataforma:

- **Frontend:** Interfaz de usuario construida con Vue.js, incluyendo la gestión de scripts, control de experimentos y visualización de video.
- **Backend:** API RESTful desarrollada con FastAPI, incluyendo la gestión de usuarios, experimentos y robots.
- **Integración con ROS:** Comunicación y control de los robots, así como la transmisión de video.
- **Seguridad y Autenticación:** Mecanismos de login y autorización.

# 2 Estrategia de Pruebas

## 2.1 Tipos de Pruebas

- **Pruebas Funcionales:** Verificar que cada funcionalidad de la plataforma se comporte de acuerdo con los requisitos.
- **Pruebas de Integración:** Asegurar que los distintos módulos del sistema interactúan correctamente entre sí.
- **Pruebas de Rendimiento:** Evaluar la capacidad de la plataforma para manejar la carga esperada y medir el tiempo de respuesta.
- **Pruebas de Seguridad:** Validar que los mecanismos de autenticación y autorización sean efectivos y que la plataforma sea resistente a ataques comunes.

## 2.2 Herramientas de Pruebas

- **Para el Frontend:** Herramientas como Cypress para pruebas end-to-end, y Jest para pruebas unitarias.
- **Para el Backend:** Herramientas como Postman para pruebas de API, y pytest para pruebas unitarias.
- **Para la Integración con ROS:** Pruebas manuales y automatizadas utilizando herramientas ROS como *roslaunch* y *rosservice*.
- **Para Seguridad:** Herramientas de análisis de seguridad como OWASP ZAP.

## 3 Plan de Pruebas

### 3.1 Pruebas del Frontend

#### 3.1.1 Funcionalidad de Control de Experimentos

**Objetivo:** Verificar que los usuarios puedan controlar los experimentos a través de la interfaz.

**Casos de prueba:**

1. **Inicio de un Experimento:** El usuario debe poder seleccionar un experimento e iniciar la ejecución desde la interfaz.
2. **Detención de un Experimento:** El usuario debe poder detener un experimento en curso.
3. **Visualización de Resultados:** Los resultados del experimento deben ser visibles en la interfaz en tiempo real.

#### 3.1.2 Transmisión de Video

**Objetivo:** Asegurar que la transmisión de video en vivo funcione correctamente.

**Casos de prueba:**

1. **Visualización del Video:** El video transmitido desde la cámara USB debe mostrarse sin interrupciones.
2. **Sincronización de Video:** El video debe estar sincronizado con las acciones del experimento.

## 3.2 Pruebas del Backend

### 3.2.1 API RESTful

**Objetivo:** Verificar que todas las rutas de la API funcionen correctamente y que los datos sean manejados adecuadamente.

**Casos de prueba:**

1. **Crear Usuario:** Se debe poder crear un nuevo usuario a través de la API.
2. **Leer Usuario:** La API debe devolver la información del usuario correctamente.
3. **Actualizar Usuario:** Se deben aplicar cambios en la información del usuario a través de la API.
4. **Eliminar Usuario:** La API debe permitir la eliminación de usuarios.

### 3.2.2 Integración con ROS

**Objetivo:** Verificar la comunicación entre la API y el sistema ROS.

**Casos de prueba:**

1. **Envío de Comandos a los Robots:** La API debe enviar comandos correctos a los robots a través de ROS.

2. **Recepción de Datos de Sensores:** Los datos de los sensores deben ser recibidos y procesados correctamente por la API.

### 3.3 Pruebas de Seguridad

#### 3.3.1 Autenticación

**Objetivo:** Asegurar que el sistema de login sea seguro y funcional.

**Casos de prueba:**

1. **Inicio de Sesión Exitoso:** Los usuarios deben poder iniciar sesión con credenciales válidas.
2. **Inicio de Sesión Fallido:** Los usuarios no deben poder iniciar sesión con credenciales incorrectas.
3. **Acceso No Autorizado:** Los usuarios no autenticados no deben acceder a secciones protegidas de la plataforma.

## 4 Resultados de las Pruebas

En esta sección se presenta un resumen de los resultados obtenidos durante las pruebas de la plataforma software de acceso remoto. Se describe el estado actual de cada componente probado y se proporciona un resumen de los hallazgos más significativos.

## 4.1 Pruebas del Frontend

### 4.1.1 Funcionalidad de Control de Experimentos

**Resultado: Exitoso.**

En la Figura 1. se observa los botones asignados para iniciar, detener y reiniciar experimentos, además en el apartado de resultados de experimentos se observa lo que devuelve uno de los prototipos al interactuar con estos botones.

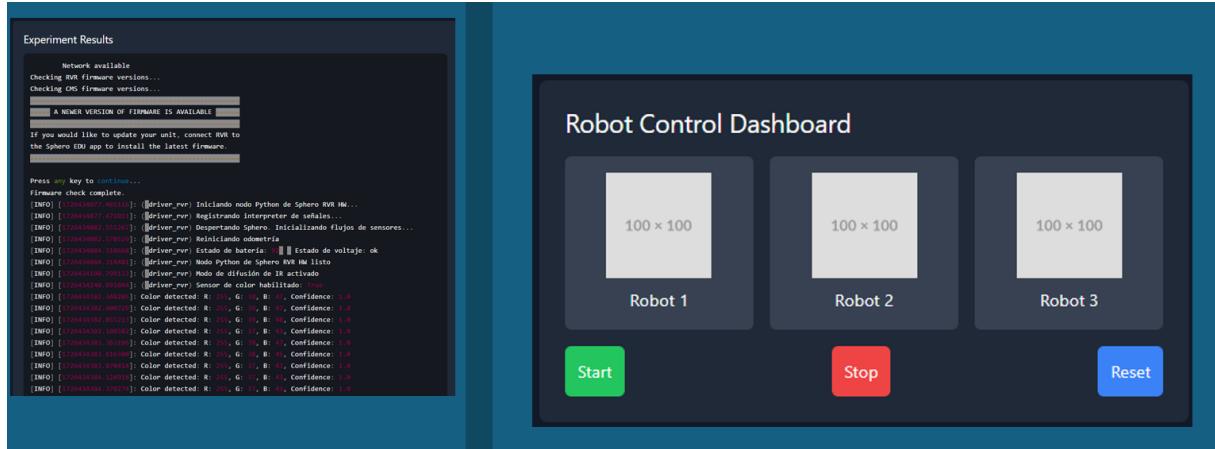


Figure 1: Interfaz de control de experimentos.

### 4.1.2 Transmisión de Video

**Resultado: Exitoso.**

En la Figura 2. se observa la interfaz con la transmisión de video, en pruebas no se detecta delay en esto, los movimientos frente a lo ejecutado en tiempo real no tienen diferencia significativa.

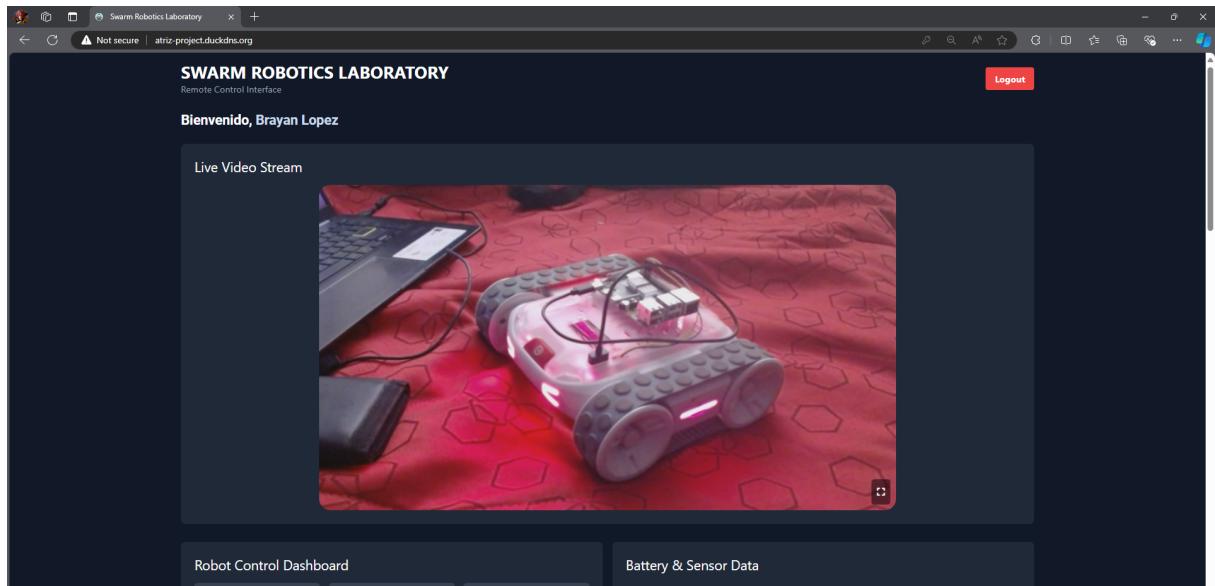


Figure 2: Transmisión de Video a través de la plataforma de acceso remoto.

## 4.2 Pruebas del Backend

### 4.2.1 API RESTful

**Resultado: Exitoso.**

Esta prueba debe ser muy detallada, por lo que se llevara a través de las siguientes imágenes con su respectiva explicación.

The screenshot shows a terminal window at the top displaying a PostgreSQL session. The user has connected to the 'swarm\_lab' database as 'postgres' and run a query to select all users from the 'users' table. The result shows one row: 'bura-admin' with hashed password '\$2b\$12\$mqb0vwDhafV/8MmkDJFBDe9Uo0IxCE05mnflCFOZ2T3TKQYz76eC' and full name 'Brayan Lopez'. Below the terminal is a FastAPI documentation page for the 'Users' endpoint. It lists several methods: GET /api/users/{user\_id} (Read User), PUT /api/users/{user\_id} (Update Existing User), DELETE /api/users/{user\_id} (Delete Existing User), POST /api/users/ (Create New User), and POST /api/login (Login For Access Token). Each method is shown with its HTTP verb, URL, and a brief description.

Figure 3: Verificación de usuarios en PosgreSQL

En la Figura 3. se observa inicialmente una consulta a la base de datos *swarm.lab* exactamente a la tabla *users* que es la que guarda el listado de usuarios registrados en la plataforma, en este caso solo existe un usuario. Verificado el listado de usuarios, se ingresa a interactuar desde FastAPI a la sección de *Users* donde se puede observar los todos métodos CRUD de esta sección.

**Creación de Usuario** En la Figura 4. se puede observar la estructura del método POST para la creación de usuario, donde se registran datos de *username*, *full\_name* y *password*.

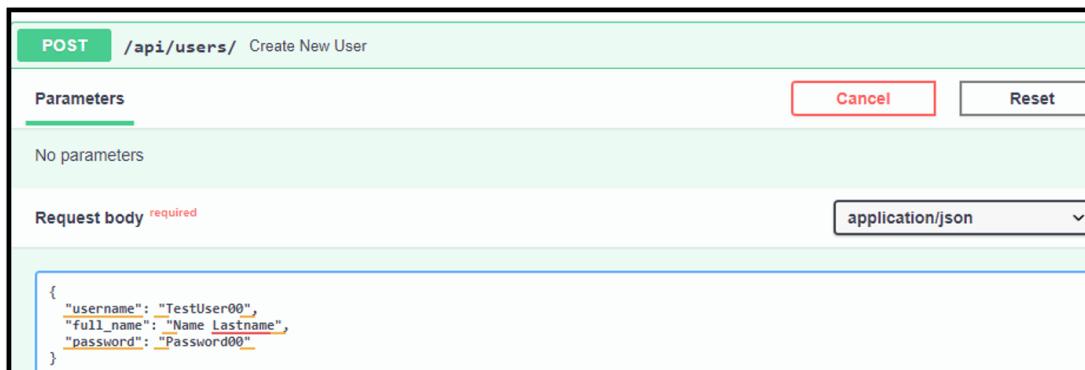


Figure 4: Creación de usuario nuevo

Finalmente en la Figura 5. se observa la respuesta de la base de datos al método ejecutado y finalmente se observa en la base de datos el efecto de la publicación del método POST llamando a la lista de todos los usuarios.

```
Curl
curl -X 'POST' \
  'http://localhost:5000/api/users/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "username": "TestUser00",
    "full_name": "Name Lastname",
    "password": "Password00"
}'
```

Request URL

id	username	hashed_password	full_name
1	bura-admin	\$2b\$12\$mqb0vwDhafV/8MmkDJFBDeD9Uo0Ixcel05mnFLCF0ZT3TKQYz76eC	Brayan Lopez
2	TestUser00	\$2b\$12\$yD41YfyQ/4QWzqnRzlZqfufGPPTfq.8YikGTPNfqj7FeaYnupHGLC	Name Lastname

(2 rows)

Figure 5: Respuesta desde la API y verificacion en posgreSQL

**Leer usuario** En la Figura 6. se puede observar la estructura de obtención con el método GET para la lectura de usuario, la entrada debe ser el ID único generado al momento de registro.

Users

GET /api/users/{user\_id} Read User

Parameters

Name	Description
user_id * required	integer (path)

Code Details

200 Response body

```
{
  "username": "TestUser00",
  "full_name": "Name Lastname",
  "id": 2
}
```

Response headers

```
content-length: 60
content-type: application/json
date: Sun,15 Sep 2024 23:35:43 GMT
servers: Unicorn
```

Figure 6: Lectura de usuario metodo GET.

**Actualizar usuario** En la Figura 7. se puede observar la estructura de actualización con el método PUT, la entrada debe ser el ID único generado al momento de registro, para identificar el usuario a actualizar, para seguidamente cambiar los datos del usuario con la misma estructura de la creación de usuario.

Finalmente, en la Figura 8. se observa la respuesta de la base de datos al método ejecutado para posteriormente verificar en la base de datos el efecto de la actualización de datos llamando a la lista de todos los usuarios.

PUT /api/users/{user\_id} Update Existing User

Parameters

Name	Description
<code>user_id</code> * required	integer (path)

Request body required

```
{
  "username": "TestUser11",
  "full_name": "Update Name",
  "id": 2
}
```

application/json

Figure 7: Actualización de datos de usuario método PUT.

Code Details

200 Response body

```
{
  "username": "TestUser11",
  "full_name": "Update Name",
  "id": 2
}
```

Download

swarm\_lab=# SELECT \* FROM users;

id	username	hashed_password	full_name
1	bura-admin	\$2b\$12\$mqb0vwDhafV/8MmkDJFBDeD9Uo0IxcE05mnflCFOZ2T3TKQYz76eC	Brayan Lopez
2	TestUser11	\$2b\$12\$0YzMkIIu06051d3e/Q2PT.ywE0gtFa70twLBvpw77zMGE2AExdgG	Update Name

Figure 8: Respuesta desde la API y verificación en PosgreSQL.

**Eliminar usuario** En la Figura 9. se puede observar la estructura de actualización con el método DELETE, la entrada debe ser el ID único generado al momento de registro.

DELETE /api/users/{user\_id} Delete Existing User

Parameters

Name	Description
<code>user_id</code> * required	integer (path)

Code Details

200 Response body

```
{
  "username": "TestUser11",
  "full_name": "Update Name",
  "id": 2
}
```

Download

swarm\_lab=# SELECT \* FROM users;

id	username	hashed_password	full_name
1	bura-admin	\$2b\$12\$mqb0vwDhafV/8MmkDJFBDeD9Uo0IxcE05mnflCFOZ2T3TKQYz76eC	Brayan Lopez

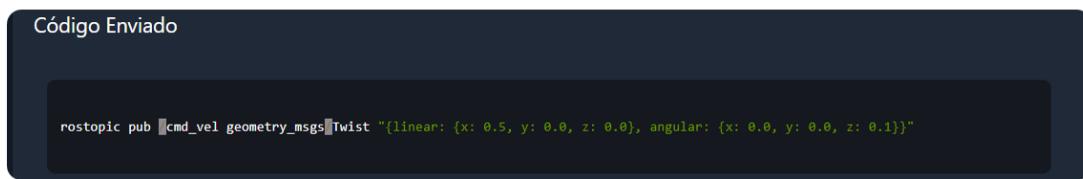
Figure 9: Eliminación de usuario método DELETE y su efecto en PosgreSQL.

#### 4.2.2 Integración con ROS

##### Resultado: Exitoso.

Para estas secciones también hay apartados dentro de la plataforma de acceso en las que se puede ver visualmente con las secciones de envío de código y lectura de sensores. Es importante mencionar que existe un repositorio con videos de pruebas del movimiento de los robots a través de la plataforma en el siguiente enlace [https://drive.google.com/drive/folders/1b15\\_zxTqFxxL-Ejp7-rAT0bA579dx-Zf?usp=sharing](https://drive.google.com/drive/folders/1b15_zxTqFxxL-Ejp7-rAT0bA579dx-Zf?usp=sharing).

**Envío de Comandos a los Robots** En la figura 10 se observa el envío de código a través de la interfaz.



```
Código Enviado
rostopic pub /cmd_vel geometry_msgs/Twist "{linear: {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.1}}"
```

Figure 10: Envío de comandos a través de la plataforma.

**Recepción de Datos de Sensores** En la figura 11 se observa la recepción del código y la lectura de sensores a través de la interfaz.

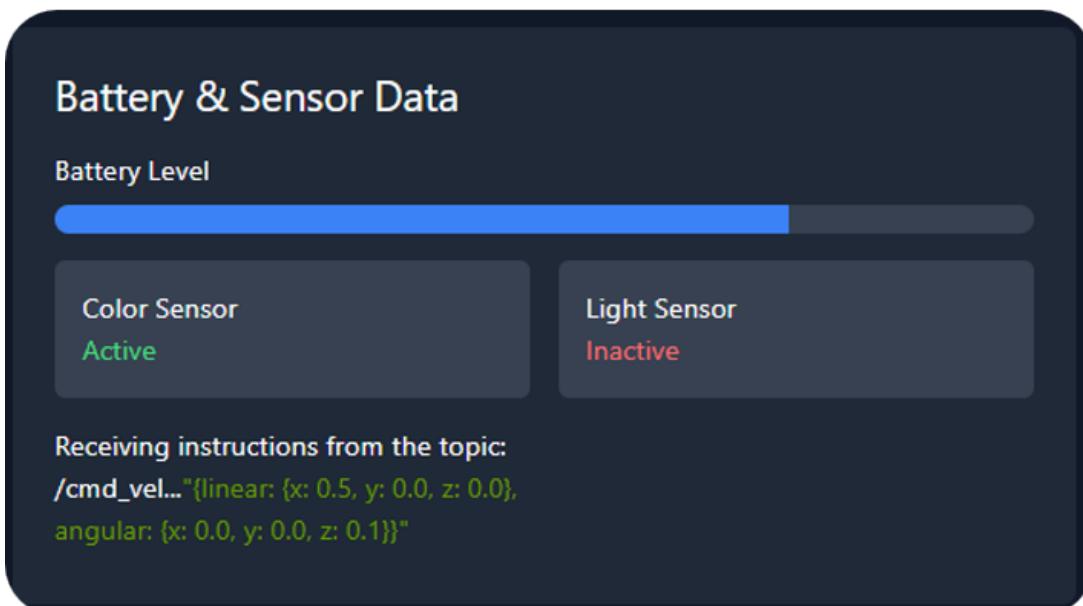


Figure 11: Recepción de comandos y lectura de sensores.

## 4.3 Pruebas de Seguridad

### 4.3.1 Autenticación

**Resultado: Exitoso.**

Para esta prueba es importante introducir el desarrollo de una ventana de *Login*, como se observa en la Figura 12. que estará siempre y cuando no exista un token válido asignado a algún usuario registrado, este desarrollado para asegurar el acceso de usuarios autorizados a la plataforma.

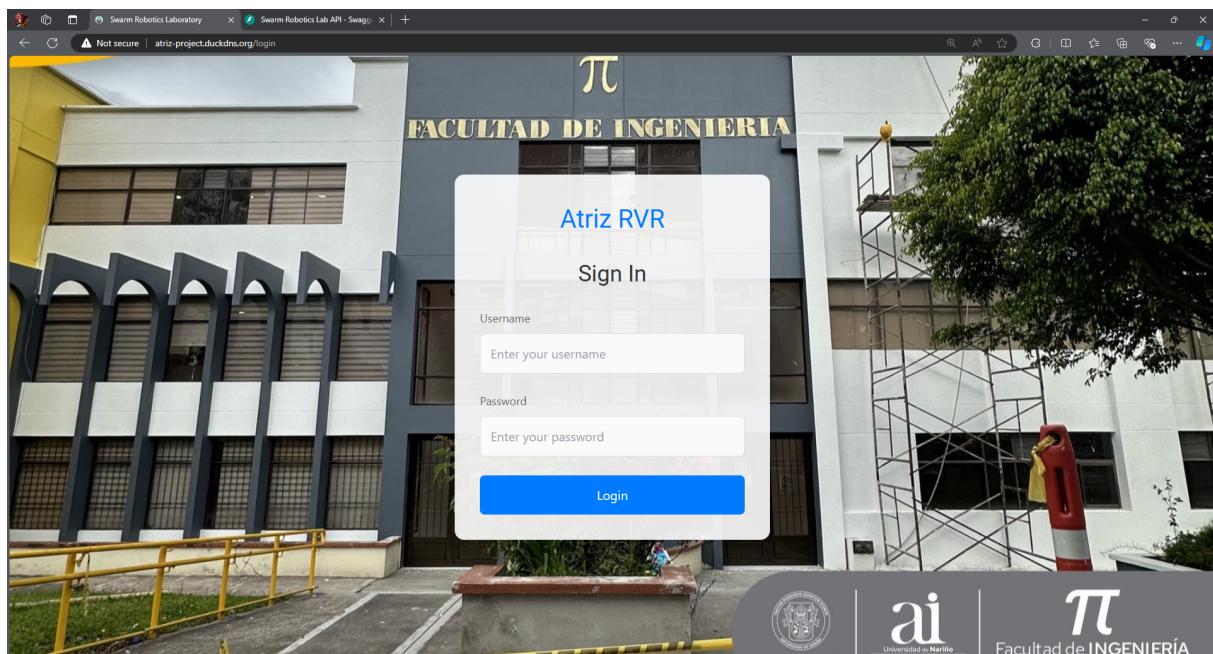


Figure 12: Login para control de acceso

**Inicio de Sesión Exitoso** Si las credenciales de usuario están la base de datos, la plataforma abre su interfaz principal, desde la que se hace un Callback al nombre del usuario, dando un mensaje de Bienvenida y completo acceso a todas las secciones, esto se observa en la Figura 13.

**Inicio de Sesión Fallido** Si las credenciales de usuario NO están la base de datos, la plataforma verifica y devuelve un error de autenticación, esto se observa en la Figura 14.

**Acceso No Autorizado** Si el usuario intenta modificar o acceder a la base de datos, se deniega su acceso desde la URL de publicación, esto se observa en la Figura 15.

**Diseño responsive** La plataforma tiene un diseño responsive, o sea el contenido se puede adaptar a diferentes resoluciones y dispositivos. Lo que hace al final una plataforma amigable para cualquier usuario. Esto se puede observar en la Figura 16

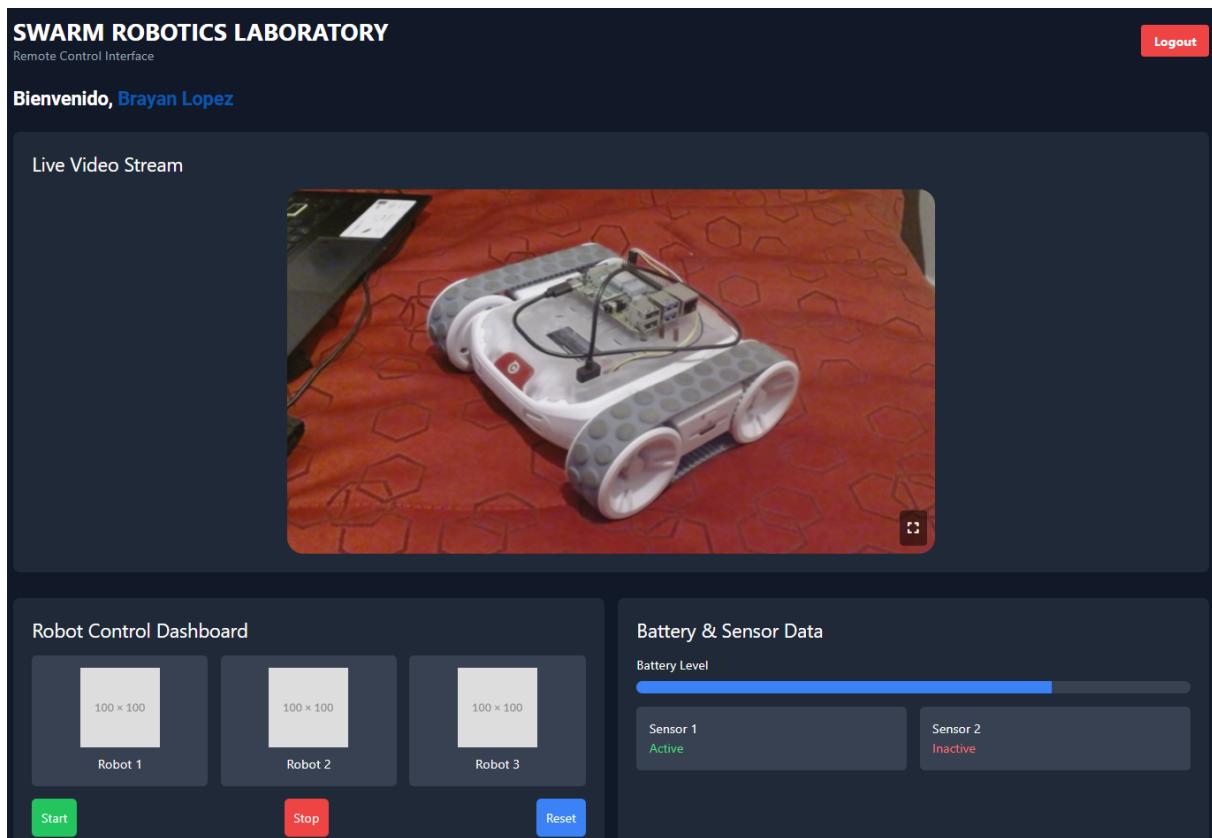


Figure 13: Acceso exitoso

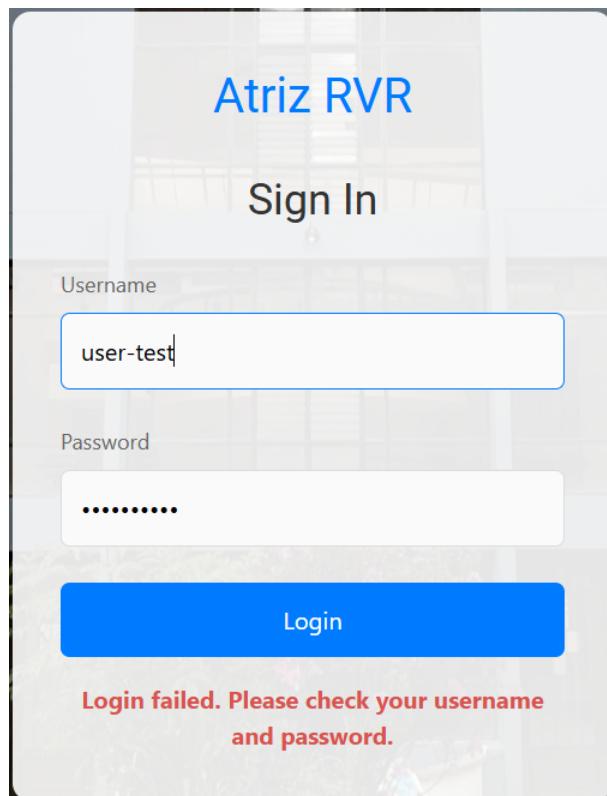


Figure 14: Acceso Fallido

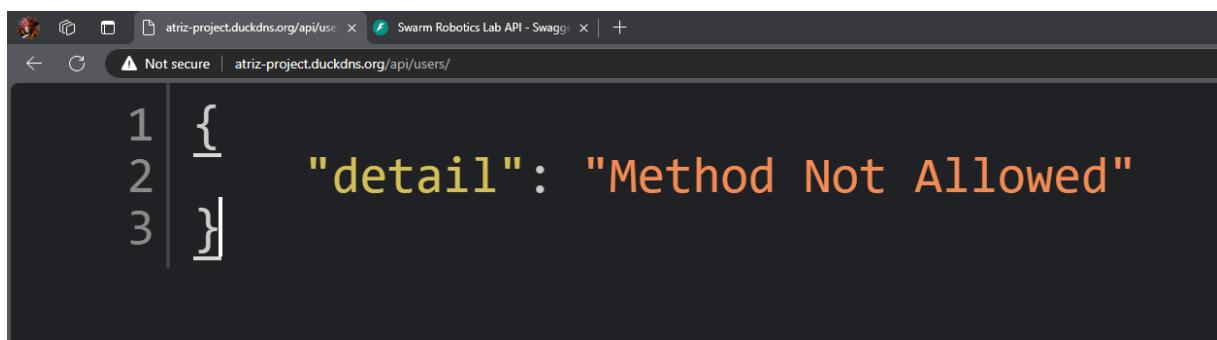


Figure 15: Acceso no autorizado API

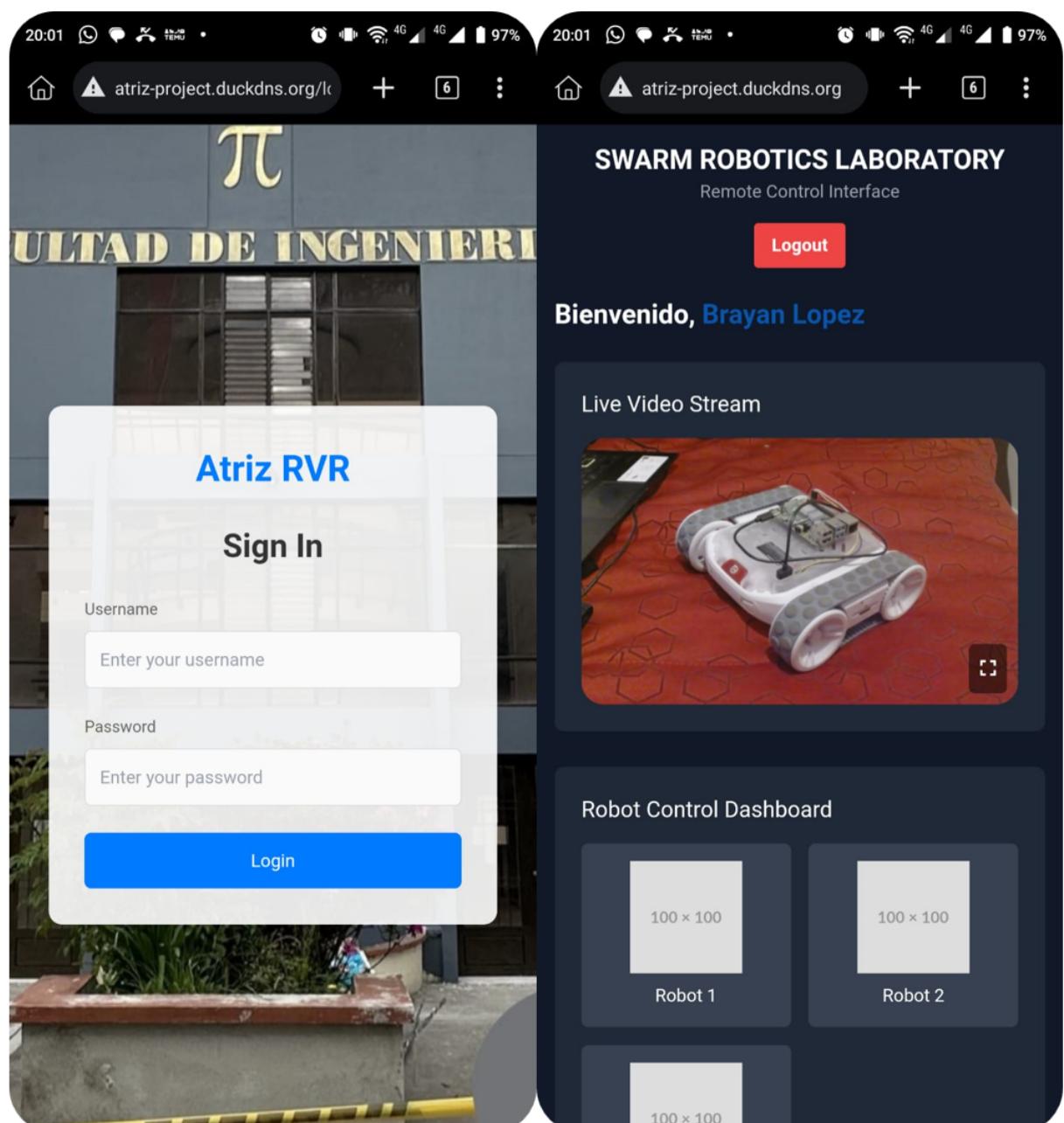


Figure 16: Login para control de acceso