|  |
| --- |
|  |
| Prueba de Caja Blanca |
| ***“Detección de Blancos Biológicos en Rosales”*** |
| Versión 1.4 |
|  |
| **Integrantes:**  Nicolas Cedillo  Alisson Clavijo  Lizzette Zapata  **Fecha 2024-02-20** |

Contenido

[Ingreso del Usuario (LOGIN) 2](#_Toc159362299)

[1.1 CÓDIGO FUENTE 2](#_Toc159362300)

[1.2 Diagrama de Flujo 5](#_Toc159362301)

[1.3 GRAFO 1](#_Toc159362302)

[1.4 COMPLEJIDAD CICLOMÁTICA 1](#_Toc159362303)

[2. Carga de Imágenes 2](#_Toc159362304)

[2.1 CÓDIGO FUENTE 2](#_Toc159362305)

[2.2 DIAGRAMA DE FLUJO 5](#_Toc159362306)

[2.3 GRAFO 5](#_Toc159362307)

[2.4 COMPLEJIDAD CICLOMÁTICA 6](#_Toc159362308)

[3. Detectar el blanco biológico 6](#_Toc159362309)

[3.1 CÓDIGO FUENTE 6](#_Toc159362310)

[3.2 DIAGRAMA DE FLUJO 6](#_Toc159362311)

[3.3 GRAFO 6](#_Toc159362312)

[3.4 COMPLEJIDAD CICLOMÁTICA 6](#_Toc159362313)

[4. Identificar gravedad del blanco biológico 6](#_Toc159362314)

[4.1 CÓDIGO FUENTE 6](#_Toc159362315)

[4.2 DIAGRAMA DE FLUJO 6](#_Toc159362316)

[4.3 GRAFO 6](#_Toc159362317)

[4.4 COMPLEJIDAD CICLOMÁTICA 7](#_Toc159362318)

[5. Evaluar la confiabilidad de la detección 7](#_Toc159362319)

[5.1 CÓDIGO FUENTE 7](#_Toc159362320)

[5.2 DIAGRAMA DE FLUJO 7](#_Toc159362321)

[5.3 GRAFO 7](#_Toc159362322)

[5.4 COMPLEJIDAD CICLOMÁTICA 7](#_Toc159362323)

# Ingreso del Usuario (LOGIN)

## 1.1 CÓDIGO FUENTE

@api\_router.post("/login")

async def login(response: Response, form: OAuth2PasswordRequestForm = Depends()):

    """

    Endpoint para autenticar al usuario y generar un token de acceso.

    Args:

        response (Response): Objeto de respuesta HTTP.

        form (OAuth2PasswordRequestForm): Datos del formulario de inicio de sesión.

    Returns:

        Response: Objeto de respuesta HTTP con el token de acceso y tipo de token.

    Raises:

        HTTPException: Si la contraseña es incorrecta.

    """

    user: UserDB = search\_user(form.username)

    print(form.username)

    if not crypt.verify(form.password, user.user\_password):

        raise HTTPException(

            status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Incorrect password")

    expire = ((datetime.utcnow() + timedelta(minutes=ACCESS\_TOKEN\_DURATION)).timestamp())

    access\_token = {

        "sub": user.user\_name,

        "exp": expire

    }

    encoded\_token = jwt.encode(access\_token, SECRET, algorithm=ALGORITHM)

    response = JSONResponse(

        content={"access\_token": encoded\_token, "token\_type": "bearer"},

        status\_code=status.HTTP\_200\_OK

    )

    response.set\_cookie(

        key="access\_token",

        value=encoded\_token,

        expires=expire,

        httponly=True,

        samesite='lax',

    )

    return response

def search\_user(user\_name: str, db=True) -> Union[UserDB, User]:

    try:

        db\_connection = Database.get\_connection()

        user\_collection = db\_connection.user\_collection  # Reemplaza "user\_collection" con el nombre de tu colección

        print("ok")

        user\_document = user\_collection.find\_one({"user\_name": user\_name})

        if user\_document is None:

            raise HTTPException(

                status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User not found"

            )

        if db:

            return UserDB(id=str(user\_document["\_id"]), \*\*user\_document)

        else:

            return User(\*\*user\_document)

    except HTTPException:

        # Re-raise HTTPException to propagate the error with the correct status code

        raise

    except Exception as e:

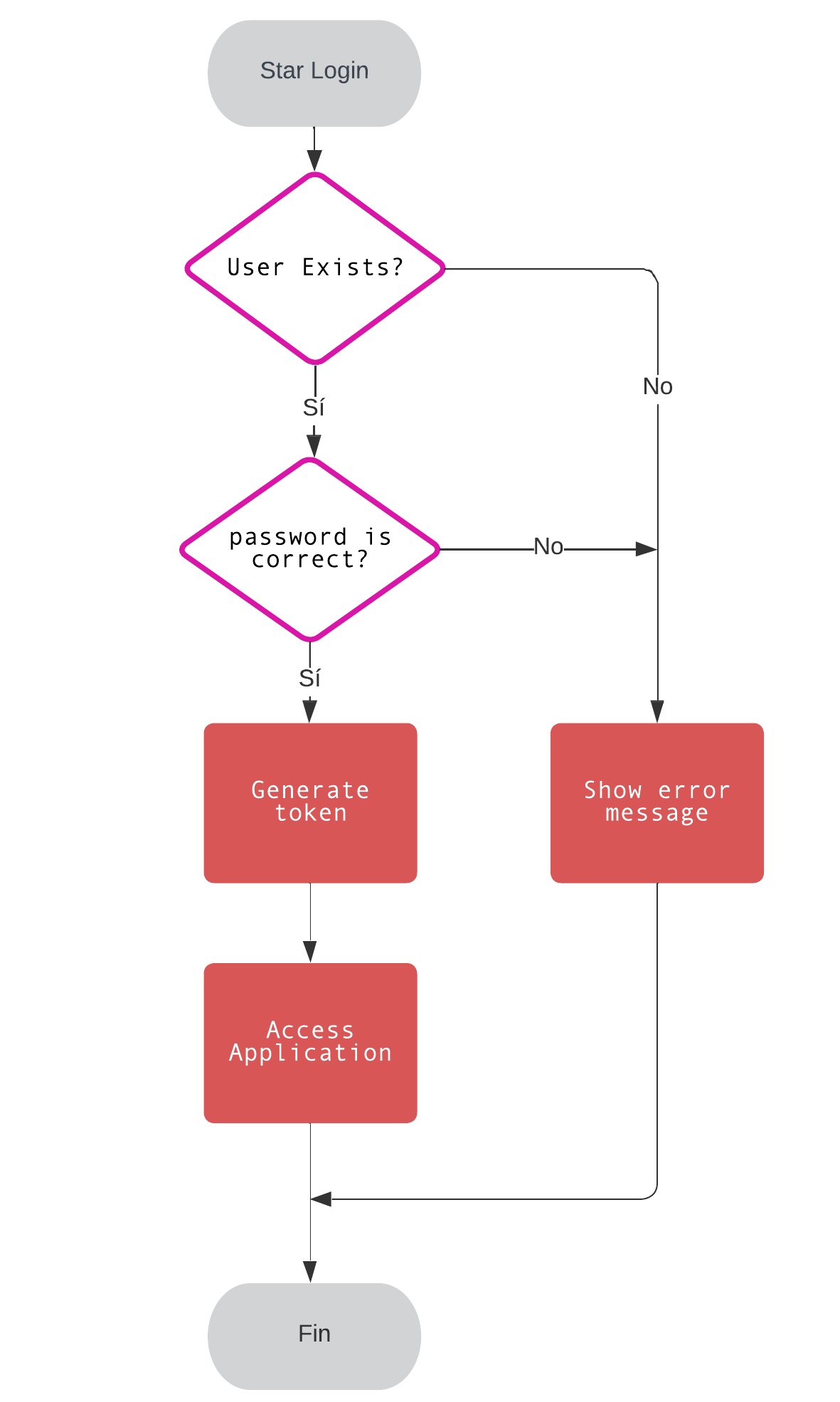
        print(e)

        raise HTTPException(

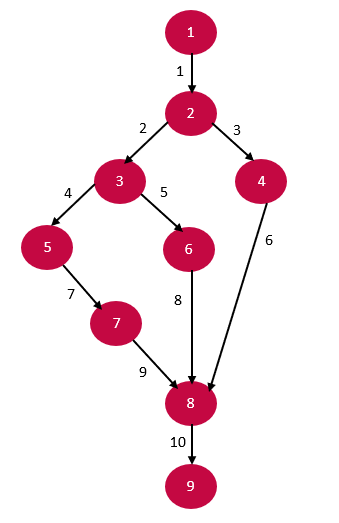
            status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR, detail="Server error " + e

        )

## 1.2 DIAGRAMA DE FLUJO

****

## 1.3 GRAFO

****

**RUTAS**

**R1:** 1**,**2,4,7,9,10

**R2:** 1,2,5,8,10

**R3:** 1,2,3,6,10

## 1.4 COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

* V(G) = A – N + 2 = 3
* V(G) = 10 – 9 + 2 = 3

DONDE:

**A:** Número de aristas

**N:** Número de nodos

# 2. Carga de Imágenes

## 2.1 CÓDIGO FUENTE

    async function classifyImage(result: string) {

        const imageBase64 = result.split(',')[1];

        // Configurar los parámetros para la llamada a la API con Axios

        const url = 'https://classify.roboflow.com/plagues-cwxwv/1';

        const apiKey = 'f1lBp5ksVVfqqMhci32v';

        const headers = {

            'Content-Type': 'application/x-www-form-urlencoded'

        };

        const params = {

            api\_key: apiKey

        };

        try {

            // Realizar la llamada a la API con Axios

            const response = await axios.post(url, imageBase64, {

                params: params,

                headers: headers

            });

            // Verificar si la respuesta es exitosa

            predicted\_classes = response.data;

            const confidence = encontrarCategoriaMaxima(response.data.predictions);

            return [predicted\_classes,confidence];

        } catch (error: any) {

            // Manejar errores

            console.error('Error en la llamada a la API:', error.message);

            return [];

        }

    }

    async function postHistoria(data: HistoriaData, selectedFile: File): Promise<void> {

        const url = 'http://localhost:8000/historias/';

        // Obtener el token de acceso de la cookie

        const access\_token = getCookie('access\_token');

        // Agregar el token al encabezado de la solicitud

        const headers = new Headers({

            Authorization: `Bearer ${access\_token}`, // Agregar el token al encabezado

            'Content-Type': 'application/json' // Establecer el tipo de contenido según tus necesidades

        });

        const requestOptions: RequestInit = {

            method: 'POST',

            headers: headers,

            body: JSON.stringify(data)

        };

        try {

            const response = await fetch(url, requestOptions);

            if (!response.ok) {

                // Si la respuesta no es exitosa, lanzar un error

                const errorText = await response.text();

                throw new Error(`Error en la solicitud: ${errorText}`);

            } else {

                const data = await response.json();

                historia = data;

                // console.log('Historia', historia);

                const formData = new FormData();

                formData.append('file', selectedFile);

                const resp = await fetch('http://localhost:8000/api/historia/image/' + historia.id, {

                    method: 'POST',

                    body: formData,

                    headers: {

                        Authorization: `Bearer ${access\_token}` // Agregar el token al encabezado

                    }

                });

                if (!resp.ok) {

                    const errorText = await resp.text();

                    throw new Error(`Error en la solicitud: ${errorText}`);

                }

                enviarMensajeAlPadre();

            }

            // console.log('Solicitud exitosa');

            // location.reload();

        } catch (error: any) {

            console.error(error.message);

            throw error; // Puedes manejar el error o relanzarlo según tus necesidades

        }

    }

    async function handleFileSelect(event: Event) {

        const input = event.target as HTMLInputElement;

        if (input.files && input.files.length > 0) {

            const selectedFile = input.files[0];

            // Leer el contenido del archivo como base64

            const reader = new FileReader();

            reader.onloadend = async function () {

                const result = reader.result as string | null;

                let blanco\_biologico: string;

                let intensidad: string;

                if (result) {

                    promise = classifyImage(result);

                    promise.then(async (res) => {

                        const data = res[0].predicted\_classes;

                        if (data.length <= 1) {

                            blanco\_biologico = 'healthy';

                            intensidad = '';

                        } else {

                            ({ blanco\_biologico, intensidad } = clasificarResultados(data));

                        }

                        // console.log(blanco\_biologico, intensidad);

                        const historiaData: HistoriaData = {

                            fecha: new Date().toISOString(),

                            intensidad: intensidad,

                            blanco\_biologico: blanco\_biologico,

                            id: ''

                        };

                        await postHistoria(historiaData, selectedFile);

                    });

                }

            };

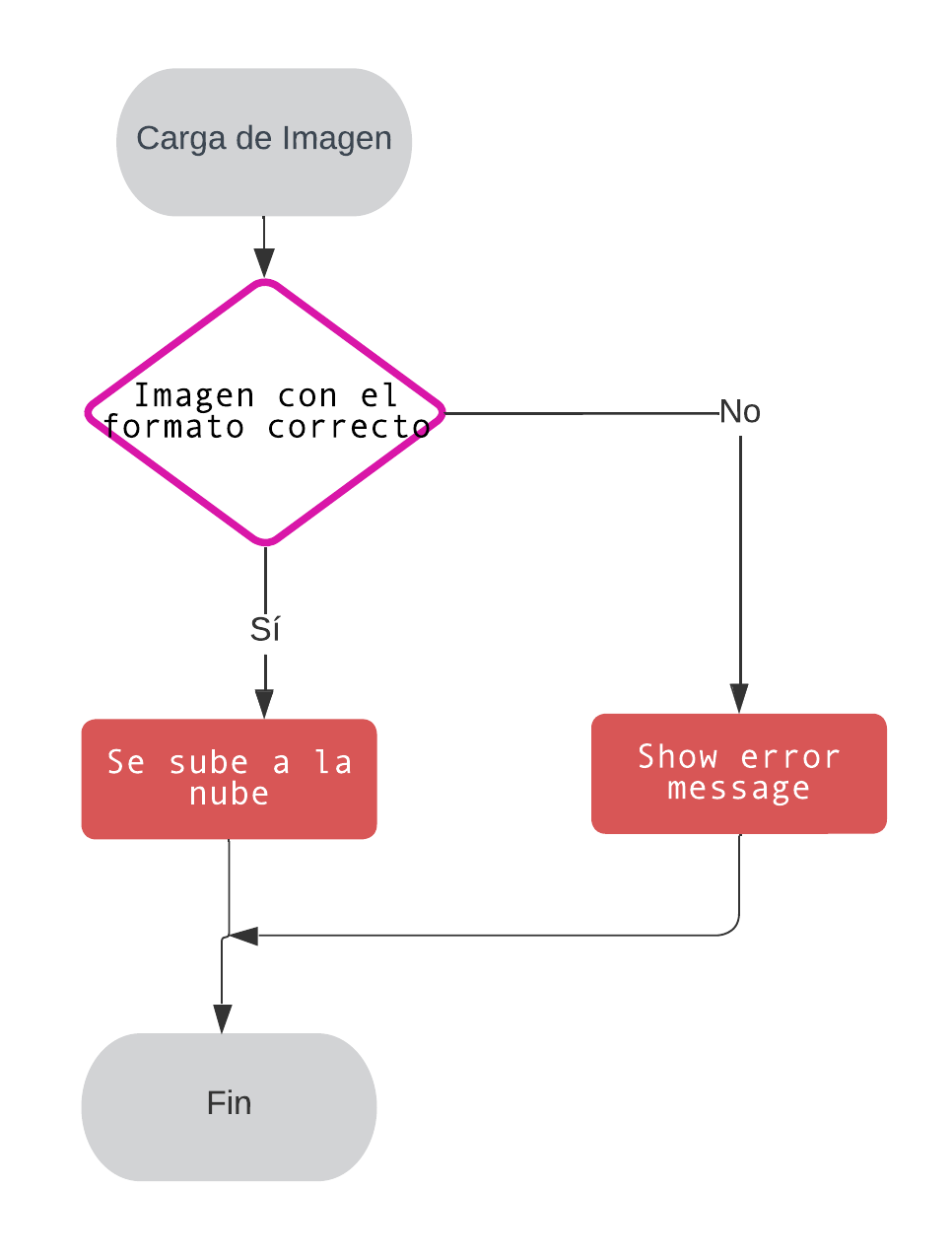
            // Leer el contenido del archivo como base64

            reader.readAsDataURL(selectedFile);

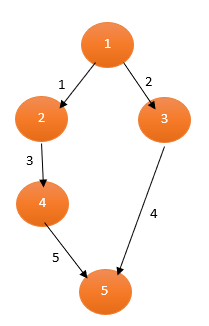
        }

    }

## 2.2 DIAGRAMA DE FLUJO



## 2.3 GRAFO



**RUTAS**

**R1:** 1**,**3,5

**R2:** 2,4

## 2.4 COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

* V(G) = A – N + 2 = 2
* V(G) = 5 – 5 + 2 = 2

DONDE:

**A:** Número de aristas

**N:** Número de nodos

# 3. Detectar el blanco biológico

## 3.1 CÓDIGO FUENTE

## 3.2 DIAGRAMA DE FLUJO

## 3.3 GRAFO

## 3.4 COMPLEJIDAD CICLOMÁTICA

# 4. Identificar gravedad del blanco biológico

## 4.1 CÓDIGO FUENTE

## 4.2 DIAGRAMA DE FLUJO

## 4.3 GRAFO

## 4.4 COMPLEJIDAD CICLOMÁTICA

# 5. Evaluar la confiabilidad de la detección

## 5.1 CÓDIGO FUENTE

## 5.2 DIAGRAMA DE FLUJO

## 5.3 GRAFO

## 5.4 COMPLEJIDAD CICLOMÁTICA