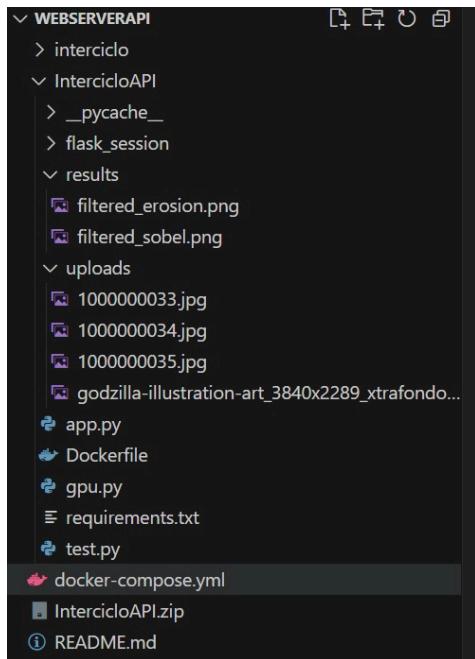


 <p>UNIVERSIDAD POLÍTÉCNICA SALESIANA ECUADOR</p>	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

		PRÁCTICA DE LABORATORIO
CARRERA: Computación		ASIGNATURA: Computación Paralela
NRO. PRÁCTICA:	TÍTULO PRÁCTICA:	UPSGlam una Plataforma Social de Imágenes con Convolución y Dockerización
OBJETIVO ALCANZADO: <ul style="list-style-type: none"> • Experimenta con soluciones basadas en: computación en la nube, cómputo de alto rendimiento, clústeres 		
ACTIVIDADES DESARROLLADAS		
<p>Para el presente proyecto integrador, se implementó Flutter para el desarrollo de la aplicación móvil y para la parte del backend se utilizó flask. La misma que se enlaza o vincula con la base de datos desarrollada en PgAdmin.</p> <p>La estructura de la aplicación móvil se establece de la siguiente manera:</p> <pre style="background-color: #2e3436; color: white; padding: 10px; font-family: monospace;"> < lib < pages buscar.dart edicion.dart Historias.dart home.dart inicioSesion.dart notificaciones.dart perfil.dart publicar.dart registro.dart < Utils constant.dart dio_client.dart main.dart </pre>		

En cuanto a la parte de la API del backend, la estructuramos de la siguiente manera:

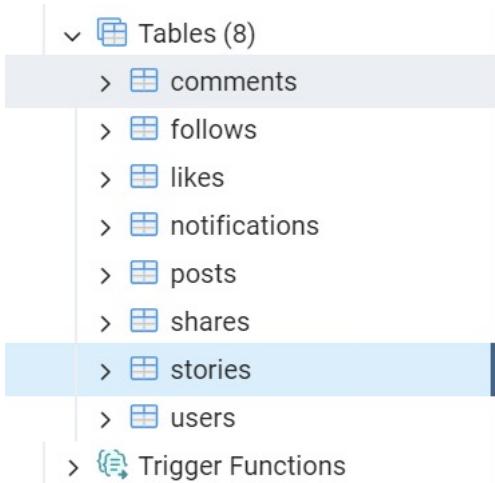


Es aquí donde se implementa la dockerización de la API para que sea consumida posteriormente por la aplicación móvil.

1. Desarrollo de Algoritmos de Convolución

- Creación de la Base de Datos.

Se creó como primera instancia una base de datos para más adelante trabajar con la API el cual se otorga todos los permisos necesarios para trabajar dentro de la aplicación móvil.



- Implementación de los algoritmos de convolución básicos vistos en clase.

Se desarrollaron los filtros vistos en las prácticas anteriores, de igual manera se dialogó entre el equipo para establecer los dos nuevos filtros para el proyecto.

```
# CUDA Kernels
kernels = {
    "sobel": """
        __global__ void sobel_filter(unsigned char *image, unsigned char *output, int width, int height) {
            int x = blockIdx.x * blockDim.x + threadIdx.x;
            int y = blockIdx.y * blockDim.y + threadIdx.y;
            if (x > 0 && x < width - 1 && y > 0 && y < height - 1) {
                int gx = -image[(y - 1) * width + (x - 1)] - 2 * image[y * width + (x - 1)] - image[(y + 1) * width + (x - 1)]
                    + image[(y - 1) * width + (x + 1)] + 2 * image[y * width + (x + 1)] + image[(y + 1) * width + (x + 1)];
                int gy = -image[(y - 1) * width + (x - 1)] - 2 * image[(y - 1) * width + x] - image[(y - 1) * width + (x + 1)]
                    + image[(y + 1) * width + (x - 1)] + 2 * image[(y + 1) * width + x] + image[(y + 1) * width + (x + 1)];
                int magnitude = min(255, (int)sqrtf(gx * gx + gy * gy));
                output[y * width + x] = magnitude;
            }
        }
    """,
    "erosion": """
        __global__ void erosion_filter(unsigned char *image, unsigned char *output, int width, int height) {
            int x = blockIdx.x * blockDim.x + threadIdx.x;
            int y = blockIdx.y * blockDim.y + threadIdx.y;
            if (x > 0 && x < width - 1 && y > 0 && y < height - 1) {
                int min_val = 255;
                for (int dx = -1; dx <= 1; ++dx) {
                    for (int dy = -1; dy <= 1; ++dy) {
                        min_val = min(min_val, (int)image[(y + dy) * width + (x + dx)]);
                    }
                }
                output[y * width + x] = min_val;
            }
        }
    """
}
"""

```

```
"highpass": """
        __global__ void highpass_filter(unsigned char *image, unsigned char *output, int width, int height) {
            int x = blockIdx.x * blockDim.x + threadIdx.x;
            int y = blockIdx.y * blockDim.y + threadIdx.y;
            if (x > 0 && x < width - 1 && y > 0 && y < height - 1) {
                int center = image[y * width + x];
                int avg_neighbors = (image[(y - 1) * width + x] + image[(y + 1) * width + x] +
                    image[y * width + (x - 1)] + image[y * width + (x + 1)]) / 4;
                output[y * width + x] = max(0, min(255, center - avg_neighbors));
            }
        }
    """
,
    "gaussian": """
        __global__ void gaussian_filter(unsigned char *image, unsigned char *output, int width, int height) {
            int kernel[3][3] = {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}};
            int x = blockIdx.x * blockDim.x + threadIdx.x;
            int y = blockIdx.y * blockDim.y + threadIdx.y;
            if (x > 0 && x < width - 1 && y > 0 && y < height - 1) {
                int sum = 0;
                int weight = 0;
                for (int dx = -1; dx <= 1; ++dx) {
                    for (int dy = -1; dy <= 1; ++dy) {
                        int val = image[(y + dy) * width + (x + dx)];
                        int w = kernel[dy + 1][dx + 1];
                        sum += val * w;
                        weight += w;
                    }
                }
                output[y * width + x] = sum / weight;
            }
        }
    """
}
"""

```

```

"gaussian": """
    __global__ void gaussian_filter(unsigned char *image, unsigned char *output, int width, int height) {
        int kernel[3][3] = {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}};
        int x = blockIdx.x * blockDim.x + threadIdx.x;
        int y = blockIdx.y * blockDim.y + threadIdx.y;
        if (x > 0 && x < width - 1 && y > 0 && y < height - 1) {
            int sum = 0;
            int weight = 0;
            for (int dx = -1; dx <= 1; ++dx) {
                for (int dy = -1; dy <= 1; ++dy) {
                    int val = image[(y + dy) * width + (x + dx)];
                    int w = kernel[dy + 1][dx + 1];
                    sum += val * w;
                    weight += w;
                }
            }
            output[y * width + x] = sum / weight;
        }
    }
""",
"emboss": """
    __global__ void emboss_filter(unsigned char *image, unsigned char *output, int width, int height) {
        int x = blockIdx.x * blockDim.x + threadIdx.x;
        int y = blockIdx.y * blockDim.y + threadIdx.y;
        if (x > 0 && x < width - 1 && y > 0 && y < height - 1) {
            int val = image[y * width + x] - image[(y + 1) * width + (x + 1)] + 128;
            output[y * width + x] = max(0, min(255, val));
        }
    }
"""
}

```

- **Pruebas de los filtros aplicados a imágenes de muestra.**

En el sistema por el lado del backend hace un guardado de las imágenes procesadas y guarda el path en la base de datos.



2. Implementación de la API

- Creación de los endpoints principales del proyecto.
 - Autenticación y Sesión

POST /register

Permite registrar nuevos usuarios en la plataforma mediante la creación de un perfil con nombre de usuario, correo electrónico y contraseña.

```
# Endpoints principales
@app.route('/register', methods=['POST'])
def register():
    data = request.json
    hashed_password = generate_password_hash(data['password'], method='pbkdf2:sha256')
    new_user = User(username=data['username'], email=data['email'], password=hashed_password)
    db.session.add(new_user)
    db.session.commit()
    return jsonify({"message": "Usuario registrado exitosamente"}), 201
```

POST /login

Valida las credenciales del usuario y establece una sesión activa.

```
@app.route('/login', methods=['POST'])
def login():
    data = request.json
    user = User.query.filter_by(email=data['email']).first()
    if user and check_password_hash(user.password, data['password']):
        session['user_id'] = user.id
        print(f"Cookies después de establecer sesión: {request.cookies}") # Depuración
        return jsonify({
            "message": "Inicio de sesión exitoso",
            "user_id": user.id}), 200
    return jsonify({"error": "Credenciales inválidas"}), 401
```

POST /logout

Cierra la sesión del usuario eliminando su ID de la sesión.

```
@app.route('/logout', methods=['POST'])
def logout():
    session.pop('user_id', None) # Elimina el user_id de la sesión
    return jsonify({"message": "Sesión cerrada exitosamente"}), 200
```

- Gestión de publicaciones

POST /posts

Permite a los usuarios crear nuevas publicaciones con imágenes y descripciones.

```
@app.route('/posts', methods=['POST', 'GET'])
def manage_posts():
    if request.method == 'GET':
        posts = Post.query.all()
        return jsonify([
            {
                "id": post.id,
                "user_id": post.user_id,
                "description": post.description,
                "image_path": f"http://{request.host}/{post.image_path.replace('\\', '/')}", # Corrige \ a /
            }
            for post in posts
        ]), 200
```

GET /posts

Devuelve una lista de todas las publicaciones disponibles en la base de datos.

```
@app.route('/uploads/<path:filename>', methods=['GET'])
def serve_uploaded_file(filename):
    return send_from_directory(UPLOAD_FOLDER, filename)
```

POST /posts/<post_id>/filter

Aplica un filtro de imagen a una publicación específica utilizando PyCUDA.

```

288     @app.route('/posts/<int:post_id>/filter', methods=['POST'])
289     def apply_post_filter(post_id):
290         try:
291             if 'user_id' not in session:
292                 return jsonify({"error": "Debe iniciar sesión"}), 401
293
294             post = Post.query.get(post_id)
295             if not post or post.user_id != session['user_id']:
296                 return jsonify({"error": "Publicación no encontrada o no autorizada"}), 404
297
298             filter_name = request.json.get('filter_name')
299             threads = int(request.json.get('threads', 4))
300
301             if filter_name not in kernels:
302                 return jsonify({"error": "Filtro no válido"}), 400
303
304             filtered_image_path = apply_filter(post.image_path, filter_name, threads)
305             if not filtered_image_path:
306                 return jsonify({"error": "Error al aplicar filtro"}), 500
307
308             post.processed_image_path = filtered_image_path
309             db.session.commit()
310
311             return jsonify({
312                 "message": "Filtro aplicado exitosamente",
313                 "filtered_image_path": filtered_image_path
314             }), 200
315
316         except Exception as e:
317             print(f"Error en apply_post_filter: {e}")
318             return jsonify({"error": "Error interno del servidor"}), 500
319
320         except Exception as e:
321             # Manejar errores inesperados
322             print(f"Error en apply_post_filter: {e}")
323             return jsonify({"error": "Error interno del servidor"}), 500
324

```

○ Interacciones con Publicaciones

POST /posts/<post_id>/like

Permite dar "like" o eliminar un "like" en una publicación.

```

@app.route('/posts/<int:post_id>/like', methods=['POST'])
def like_post(post_id):
    if 'user_id' not in session:
        return jsonify({"error": "Debe iniciar sesión"}), 401
    user_id = session['user_id']
    existing_like = Like.query.filter_by(user_id=user_id, post_id=post_id).first()
    if existing_like:
        db.session.delete(existing_like)
        db.session.commit()
        return jsonify({"message": "Like eliminado"}), 200
    new_like = Like(user_id=user_id, post_id=post_id)
    db.session.add(new_like)
    # Crear notificación
    new_notification = Notification(user_id=session['user_id'], post_id=post_id, action='like')
    db.session.add(new_notification)
    db.session.commit()
    return jsonify({"message": "Like agregado"}), 201

```

GET /posts/<post_id>/likes

Obtiene todos los usuarios que dieron "like" a una publicación.

```
@app.route('/posts/<int:post_id>/likes', methods=['GET'])
def get_post_likes(post_id):
    likes = Like.query.filter_by(post_id=post_id).all()
    if not likes:
        return jsonify({"message": "No hay likes para esta publicación"}), 200
    users = [{"user_id": like.user_id} for like in likes]
    return jsonify({"likes": users}), 200
```

POST /posts/<post_id>/comment

Agrega un comentario a una publicación.

```
@app.route('/posts/<int:post_id>/comment', methods=['POST'])
def comment_post(post_id):
    if 'user_id' not in session:
        return jsonify({"error": "Debe iniciar sesión"}), 401
    content = request.json.get('content')
    if not content:
        return jsonify({"error": "El comentario no puede estar vacío"}), 400
    new_comment = Comment(user_id=session['user_id'], post_id=post_id, content=content)
    db.session.add(new_comment)
    db.session.commit()
    # Crear notificación
    new_notification = Notification(user_id=session['user_id'], post_id=post_id, action='comment')
    db.session.add(new_notification)
    db.session.commit()
    return jsonify({"message": "Comentario agregado"}), 201
```

GET /posts/<post_id>/comments

Devuelve los comentarios de una publicación específica.

```
@app.route('/posts/<int:post_id>/comments', methods=['GET'])
def get_post_comments(post_id):
    comments = Comment.query.filter_by(post_id=post_id).all()
    if not comments:
        return jsonify({"message": "No hay comentarios para esta publicación"}), 200
    response = [{"user_id": comment.user_id, "content": comment.content, "created_at": comment.created_at} for comment in comments]
    return jsonify({"comments": response}), 200
```

POST /posts/<post_id>/share

Permite compartir una publicación con otros usuarios.

 UNIVERSIDAD POLÍTÉCNICA SALESIANA ECUADOR	VICERRECTORADO DOCENTE CONSEJO ACADÉMICO	Código: GUIA-PRL-001 Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

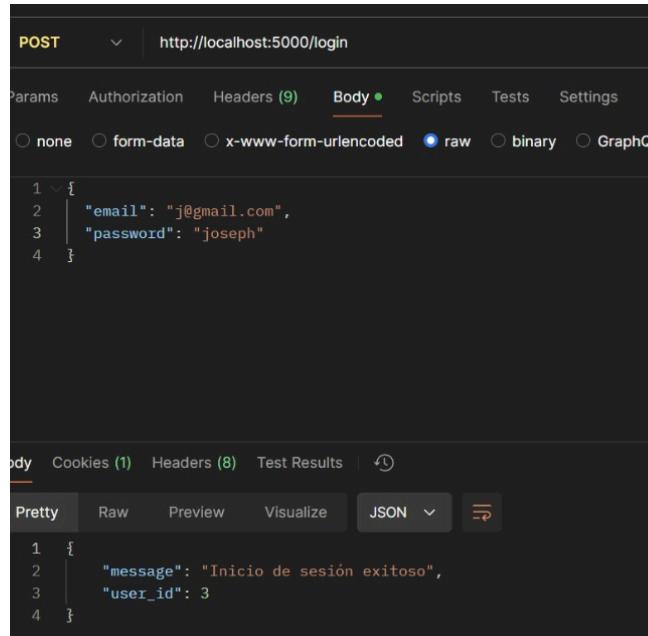
@app.route('/posts/<int:post_id>/share', methods=['POST'])
def share_post(post_id):
    if 'user_id' not in session:
        return jsonify({"error": "Debe iniciar sesión"}), 401
    new_share = Share(user_id=session['user_id'], post_id=post_id)
    db.session.add(new_share)
    db.session.commit()
    return jsonify({"message": "Publicación compartida"}), 201
  
```

- **Pruebas funcionales de la API para asegurar su correcto funcionamiento.**

Como se observa en la imagen, la aplicación y sus funciones funcionan correctamente mediante el uso del postman.

```

-----
(interciclo) PS C:\Users\Joseph\Documents\Computación Paralela\INTERCICLO\WebServerAPI\IntercicloAPI> python app.py
>>
* 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://172.16.214.105:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 670-203-464
  
```



The screenshot shows a Postman interface with a successful POST request to `http://localhost:5000/login`. The request body is set to `raw` JSON:

```

1 {
2   "email": "j@gmail.com",
3   "password": "joseph"
4 }
  
```

The response body is also in JSON format:

```

1 {
2   "message": "Inicio de sesión exitoso",
3   "user_id": 3
4 }
  
```

Como se observó en las imágenes anteriores, la estructura de la API se compone por un archivo `app.py` genérico para la ejecución de los procesos y subprocesos que son consumidos desde el frontend. Por lo tanto, la estructura de la API se conforma en primera instancia otorgando privilegios a la aplicación, como por ejemplo los permisos cors, seguido de la creación de las tablas que se pueden comprobar dentro de PgAdmin.

3. Dockerización del Sistema

- Creación del Dockerfile para la API y su entorno.

Se creó lo que es el dockerfile para utilizar la API y poder realizar exitosamente la dockerización.

```
IntercicloAPI > 🐳 Dockerfile > ...
1  # Imagen base con CUDA y Python, sin cuDNN
2  FROM nvidia/cuda:12.5.1-devel-ubuntu22.04
3
4  # Instalar paquetes necesarios
5  RUN apt-get update -qq && \
6      apt-get install -y -qq python3-pip build-essential && \
7      pip3 install --no-cache-dir pycuda flask numpy pillow
8
9  # Copiar el código de la aplicación
10 COPY . /app
11 WORKDIR /app
12
13 # Exponer el puerto para Flask
14 EXPOSE 5000
15
16 # Comando para iniciar Flask
17 CMD ["python3", "app.py"]
```

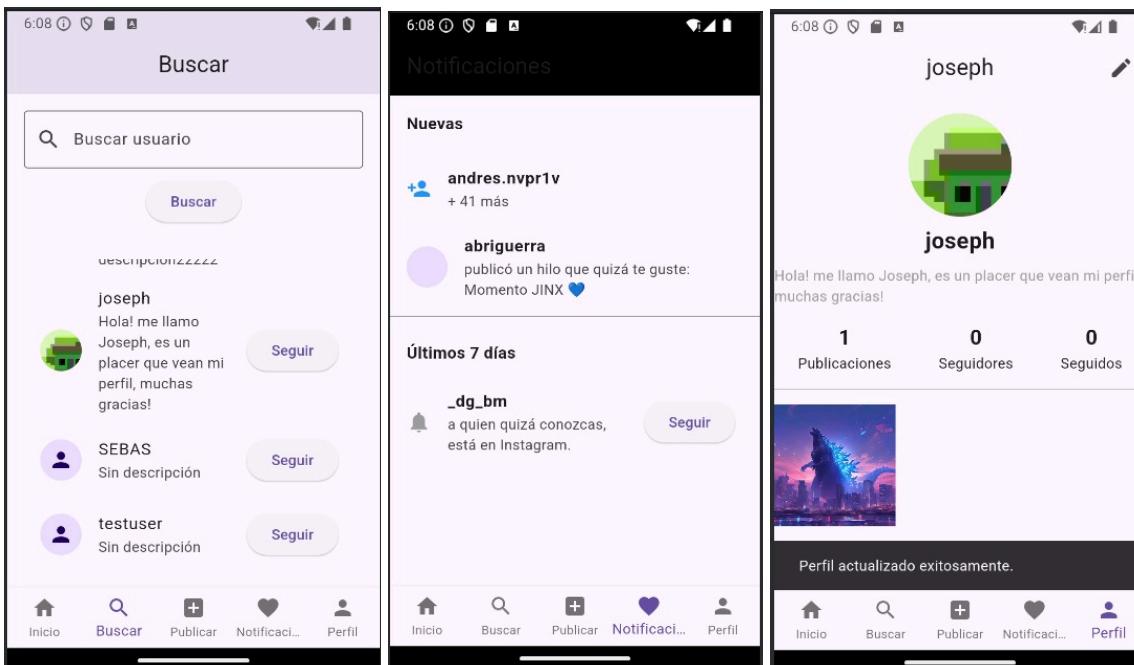
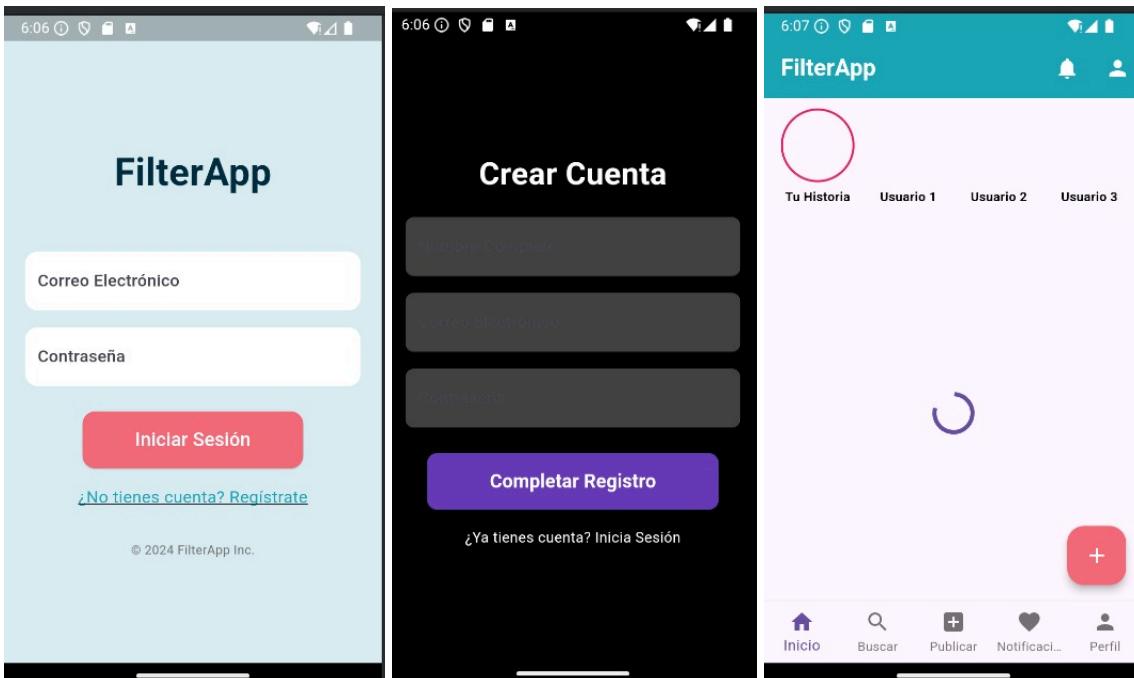
- Creación del archivo docker-compose.yml

Para esta parte de aquí, se creó el docker-compose, lo cual este nos sirve para definir y orquestar dos servicios principales del proyecto utilizando Docker: la API y la base de datos PostgreSQL. Permite levantar ambos servicios de manera conjunta en contenedores aislados, asegurando que puedan comunicarse entre sí correctamente.

```
docker-compose.yml
1  version: '3.8'
2  services:
3    api:
4      build: .
5      ports:
6        - "5000:5000"
7      depends_on:
8        - db
9      environment:
10        - SQLALCHEMY_DATABASE_URI=postgresql://FilterApp:FilterApp@db/FilterApp
11    db:
12      image: postgres:15
13      environment:
14        POSTGRES_USER: FilterApp
15        POSTGRES_PASSWORD: FilterApp
16        POSTGRES_DB: FilterApp
17      ports:
18        - "5432:5432"
19
```

4. Desarrollo de la Aplicación Móvil

- Creación de la interfaz de usuario con pantallas funcionales: registro, inicio de sesión, feed, publicación y comentarios.



 UNIVERSIDAD POLÍTÉCNICA SALESIANA ECUADOR	VICERRECTORADO DOCENTE CONSEJO ACADÉMICO	Código: GUIA-PRL-001 Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

5. Resultados

POST http://localhost:5000/posts

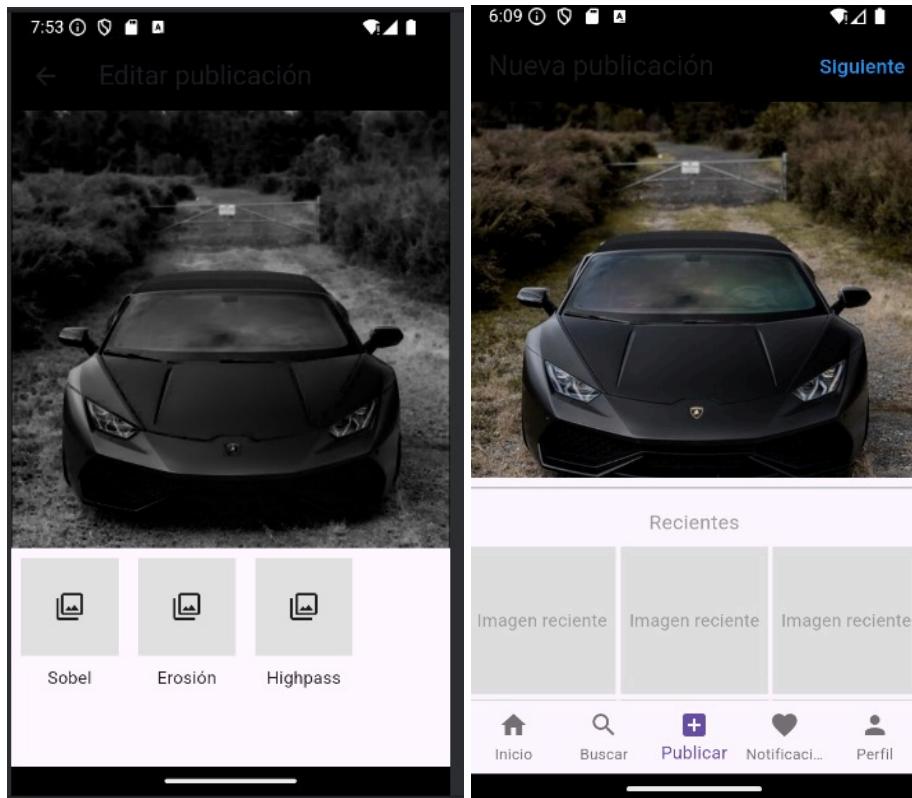
Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Body (1) Headers (7) Test Results ①

201 CREATED + 17 ms + 270 B ⌂ Save Response ⌂ ⌂ ⌂

Key	Value	Description	⋮ Bulk Edit
<input checked="" type="checkbox"/> image	File ▾ godzilla-illustration-art_3840x2289_xtrafondo...	<input type="button" value="Upload"/>	
<input checked="" type="checkbox"/> description	Text ▾ Primera publicación de prueba		
Key	Value	Description	

```
1 {
2   "id": 7,
3   "message": "Publicación creada"
4 }
```



RESULTADO(S) OBTENIDO(S):

- Se implementaron con éxito algoritmos de convolución avanzados utilizando PyCUDA, logrando un procesamiento eficiente de imágenes con filtros personalizados.
- La API desarrollada permitió manejar autenticación, gestión de publicaciones, y aplicar filtros, cumpliendo con los requerimientos funcionales establecidos.
- La base de datos fue configurada correctamente utilizando PostgreSQL y administrada con PgAdmin, asegurando la integridad y la persistencia de los datos.
- La dockerización del sistema mediante Docker y Docker Compose facilitó la orquestación de la API y la base de datos, garantizando portabilidad y escalabilidad.
- Se desarrolló una aplicación móvil con Flutter, ofreciendo una interfaz intuitiva que incluye funcionalidades como registro, inicio de sesión, publicación de imágenes y comentarios.
- Las pruebas funcionales realizadas con Postman confirmaron el correcto funcionamiento de los endpoints y la interacción entre la API y la base de datos.

CONCLUSIONES:

- El proyecto logró integrar múltiples tecnologías modernas como PyCUDA, Docker, PgAdmin, Postman y Flutter, demostrando la capacidad de desarrollar un sistema completo y escalable.
- La implementación de algoritmos de convolución personalizados permitió profundizar en el uso de GPUs para mejorar el rendimiento en el procesamiento de imágenes.
- La dockerización fue clave para simplificar la implementación y garantizar que el proyecto pudiera desplegarse en distintos entornos sin problemas.
- La aplicación móvil no solo cumplió con los requisitos funcionales, sino que también ofreció una experiencia de usuario intuitiva y eficiente.
- La colaboración en equipo fue esencial para cumplir con los objetivos establecidos, permitiendo resolver desafíos técnicos y cumplir con los plazos de entrega.

RECOMENDACIONES:

- Implementar medidas de seguridad adicionales en la API, como validaciones avanzadas y cifrado, para proteger los datos de los usuarios.
- Explorar la integración de nuevas funcionalidades en la aplicación móvil, como recomendaciones basadas en inteligencia artificial o filtros más complejos.
- Optimizar los algoritmos de convolución para mejorar aún más el rendimiento, especialmente para imágenes de mayor resolución.
- Realizar pruebas adicionales en entornos de producción para garantizar la estabilidad y escalabilidad del sistema bajo diferentes cargas de trabajo.
- Documentar más ejemplos prácticos en Postman para demostrar cómo consumir los endpoints desarrollados.

Integrante 1:

Nombre de estudiante: Bryam Fernando Guachun Guaman



 <p>UNIVERSIDAD POLÍTÉCNICA SALESIANA ECUADOR</p>	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Firma de estudiante:

Integrante 2:

Nombre de estudiante: Juan Sebastian Andrade Alulema

Firma de estudiante:



Integrante 3:

Nombre de estudiante: Kaar Joseph Mashingashi Unkuch

Firma de estudiante:

