

# UNIVERSIDAD POLITÉCNICA SALESIANA

## Carrera de Computación

**Integrantes:** Bryam Guachum, Sebastian Andrade, Kaar Mashingashi

**Asignatura:** Computación paralela

**Tema:** Desarrollar una Plataforma Social de Imágenes con Convolución y Dockerización



# Introducción

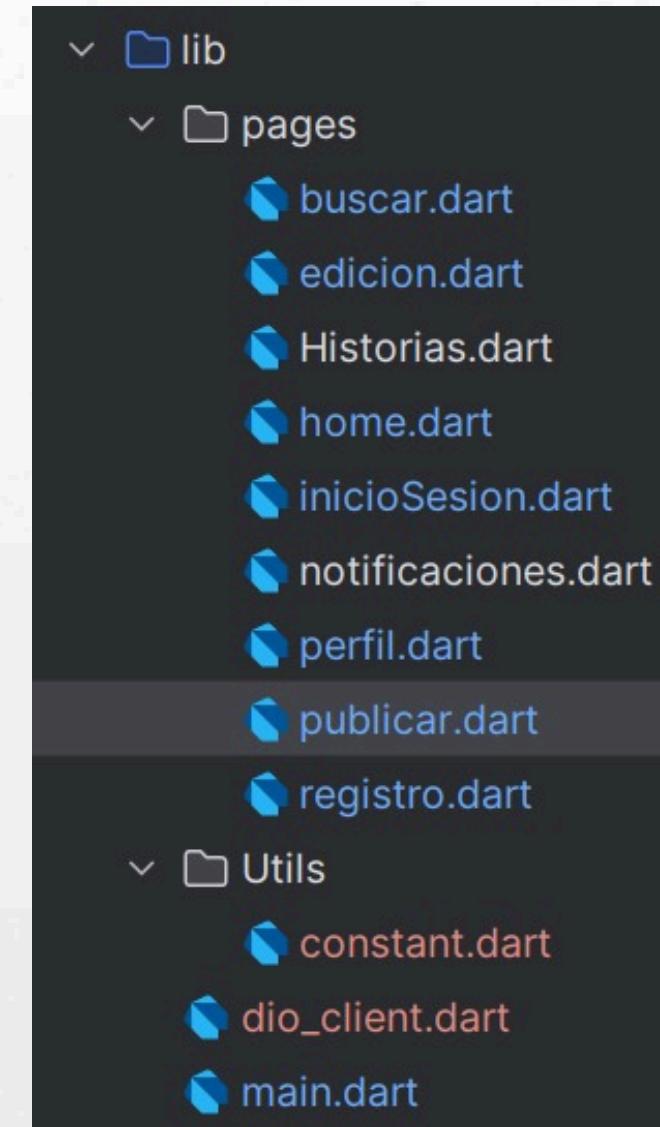
Este proyecto es una aplicación móvil que permite la publicación e interacción a través de imágenes, incorporando algoritmos avanzados de procesamiento con PyCUDA y una API eficiente. Todo esto está respaldado por un entorno dockerizado que asegura flexibilidad y escalabilidad. La combinación de diseño intuitivo y funcionalidad avanzada busca ofrecer una experiencia atractiva y práctica para los usuarios.

# Objetivos del Proyecto

- **Desarrollar una aplicación móvil interactiva:** Crear una plataforma similar a Instagram que permita a los usuarios registrarse, iniciar sesión, publicar imágenes, interactuar mediante "likes" y comentarios, y aplicar filtros de convolución personalizados.
- **Implementar algoritmos avanzados de procesamiento de imágenes:** Diseñar y programar algoritmos de convolución utilizando PyCUDA para aprovechar la potencia de la GPU, incluyendo filtros personalizados que representen a la UPS.
- **Diseñar una API eficiente:** Crear una API para gestionar el procesamiento de imágenes, garantizando un flujo eficiente entre la aplicación móvil y los servicios en la nube.
- **Dockerizar los servicios del proyecto:** Encapsular la API y la base de datos en contenedores Docker para garantizar portabilidad, escalabilidad y facilidad de implementación en diferentes entornos.
- **Generar un APK funcional:** Entregar una aplicación lista para instalación en dispositivos Android, con una interfaz de usuario atractiva e intuitiva.
- **Fomentar la creatividad e innovación:** Desarrollar filtros de imágenes únicos y un diseño visual que combine funcionalidad con estética, destacando la originalidad en la aplicación móvil.

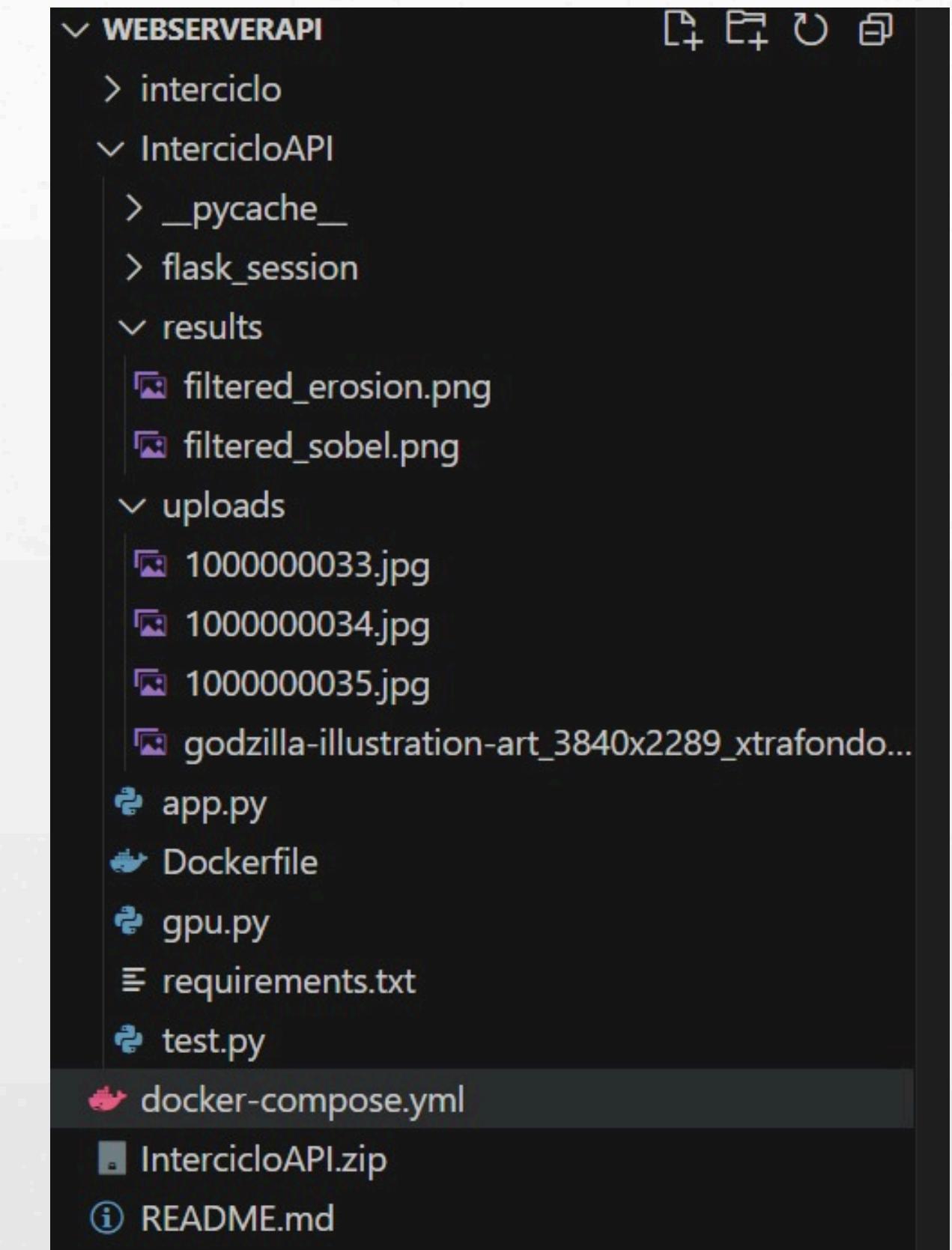
# Estructura del Proyecto

- **pages/**: Contiene las diferentes pantallas de la aplicación móvil.
- **buscar.dart**: Página para buscar contenido o usuarios.
- **edicion.dart**: Página para editar imágenes o publicaciones.
- **Historias.dart**: Página para visualizar historias de los usuarios.
- **home.dart**: Página principal que muestra el feed.
- **inicioSesion.dart**: Página de inicio de sesión para los usuarios.
- **notificaciones.dart**: Página para visualizar notificaciones.
- **perfil.dart**: Página del perfil del usuario.
- **publicar.dart**: Página para publicar imágenes o contenido.
- **registro.dart**: Página para registrar nuevos usuarios.
- **Utils/**: Contiene utilidades y configuraciones esenciales.
- **constant.dart**: Archivo con constantes utilizadas en toda la aplicación.
- **dio\_client.dart**: Cliente para manejar las solicitudes a la API.
- **main.dart**: Punto de entrada principal de la aplicación.

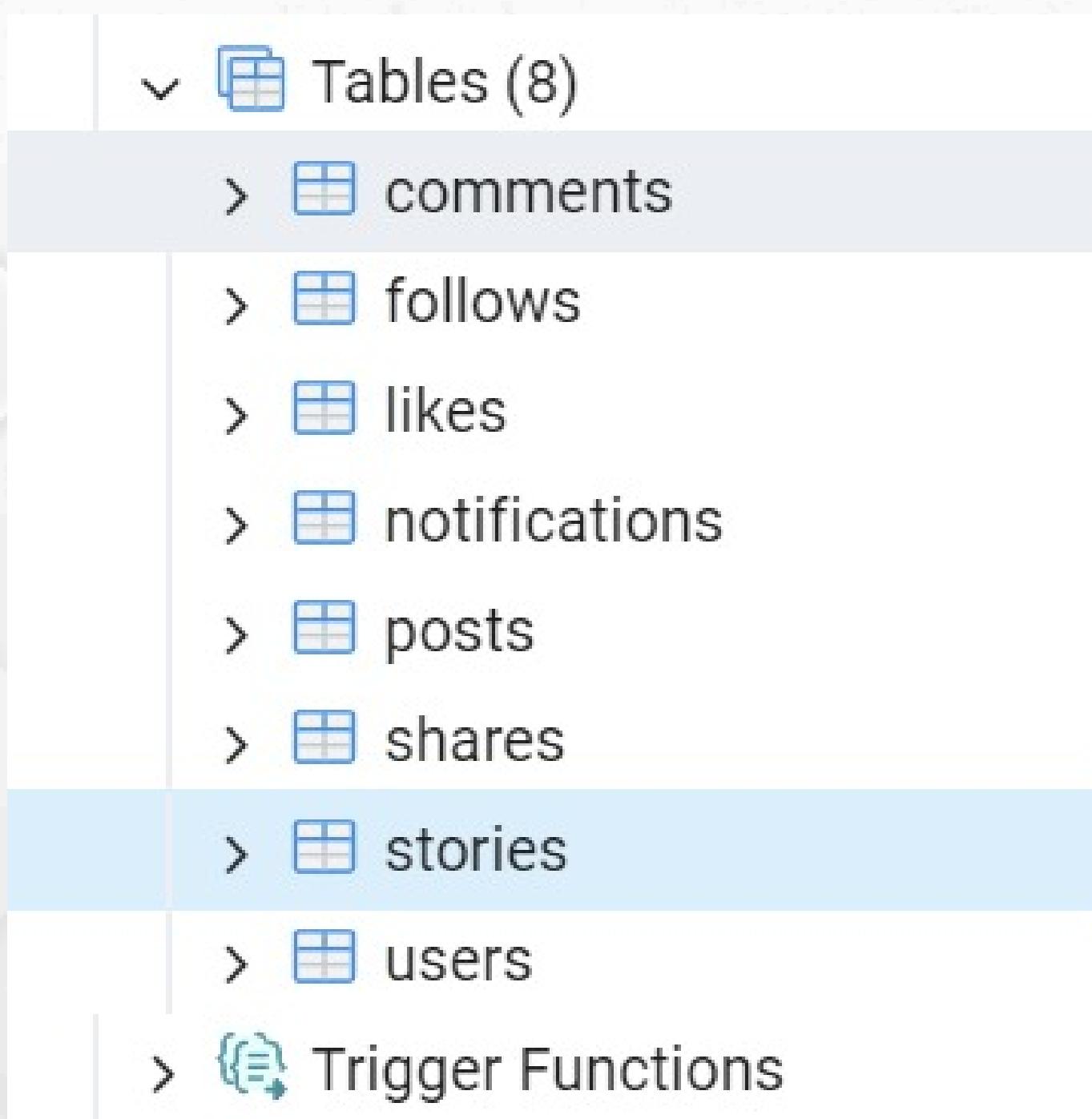


# Estructura del Proyecto

- **results/**: Carpeta que almacena imágenes procesadas con filtros, como Sobel y erosión.
- **uploads/**: Carpeta que contiene las imágenes subidas por los usuarios.
- **app.py**: Lógica principal de la API.
- **gpu.py**: Implementación de algoritmos de convolución con PyCUDA.
- **docker-compose.yml**: Configuración para desplegar la API y la base de datos con Docker.
- **requirements.txt**: Lista de dependencias necesarias para ejecutar el proyecto.



# Estructura del Proyecto



- **users:** Almacena información de los usuarios, como nombres, correos y contraseñas.
- **posts:** Registra las publicaciones realizadas por los usuarios, incluyendo descripciones y rutas de imágenes.
- **comments:** Guarda los comentarios realizados en las publicaciones.
- **likes:** Lleva el conteo de "me gusta" en las publicaciones.
- **stories:** Almacena las historias que los usuarios comparten.
- **follows:** Gestiona las relaciones entre usuarios que se siguen mutuamente.
- **notifications:** Registra notificaciones relacionadas con la interacción entre usuarios.
- **shares:** Controla las publicaciones compartidas.

# Estructura del Proyecto

```
"highpass": "",> apply
__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": "",> apply
__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": ""
```

```
# 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;
            }
        }
    """,
    "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));
            }
        }
    """
}
```

# Implementación de End Points

## Autenticación y Sesión

```
# 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
```

```
@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
```

```
@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
```

# Implementación de End Points

## Gestión de publicaciones

```
@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
```

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

```
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
```

# Implementación de End Points

## Interacciones con Publicaciones

```
@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
```

```
@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
```

```
@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
```

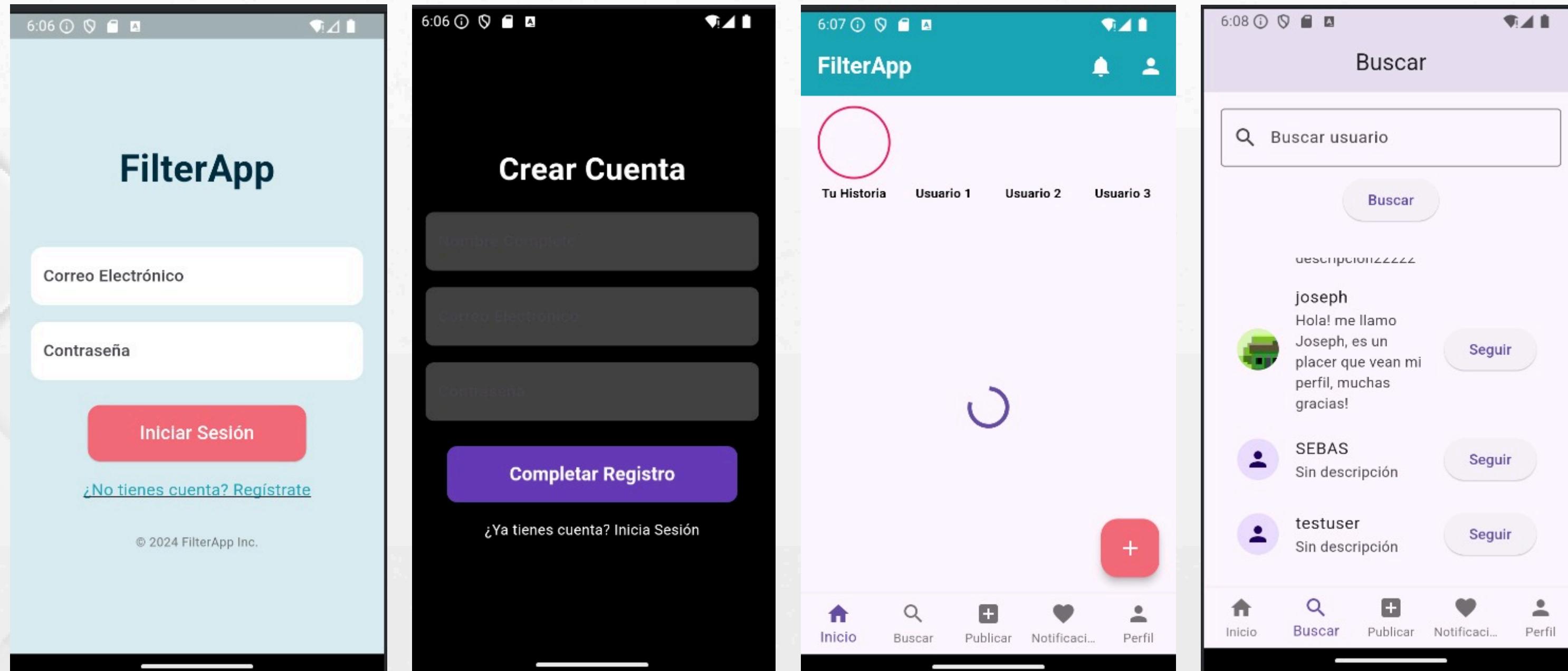
# Implementación de End Points

## Interacciones con Publicaciones

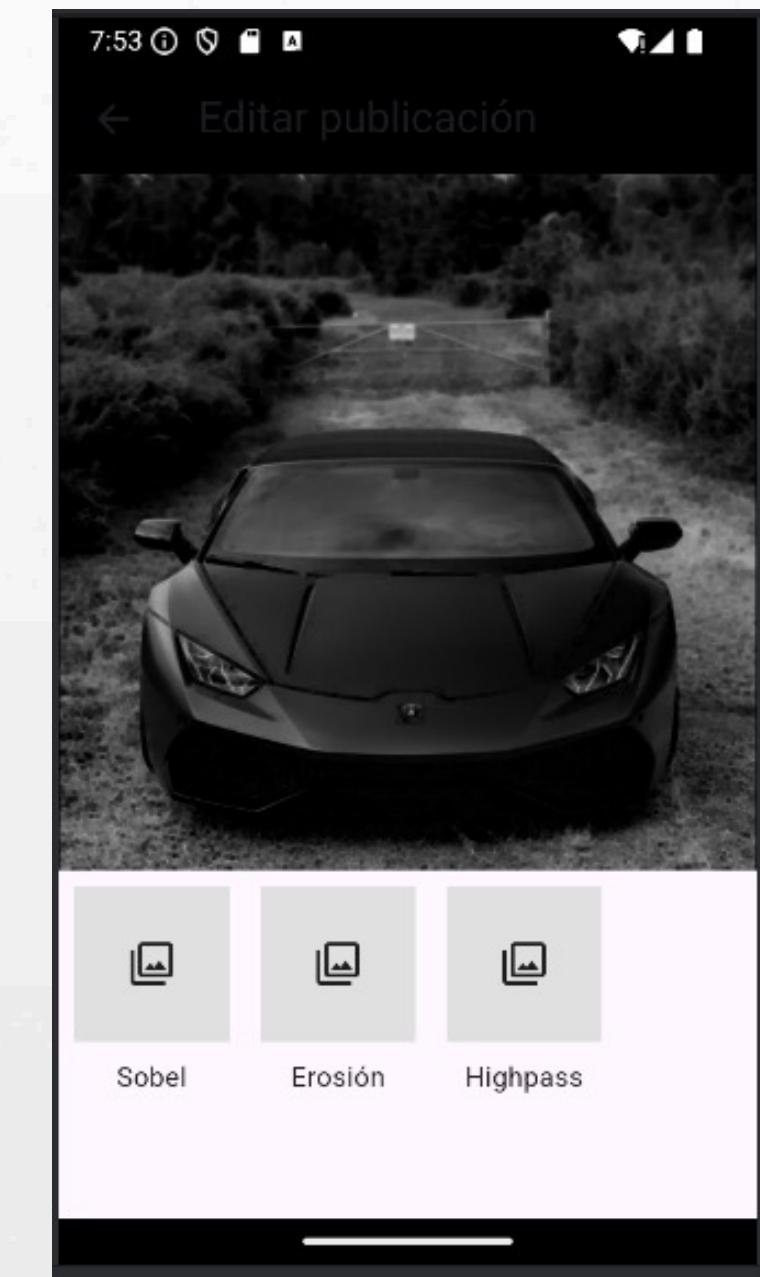
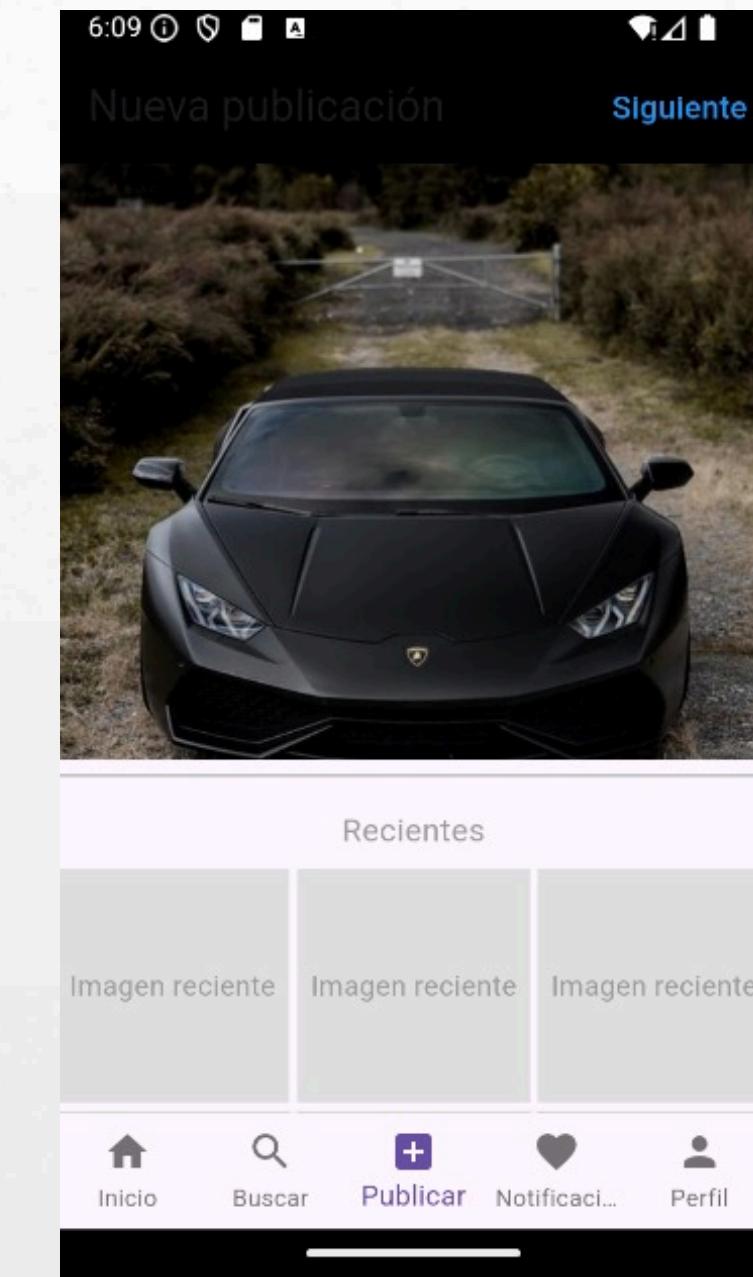
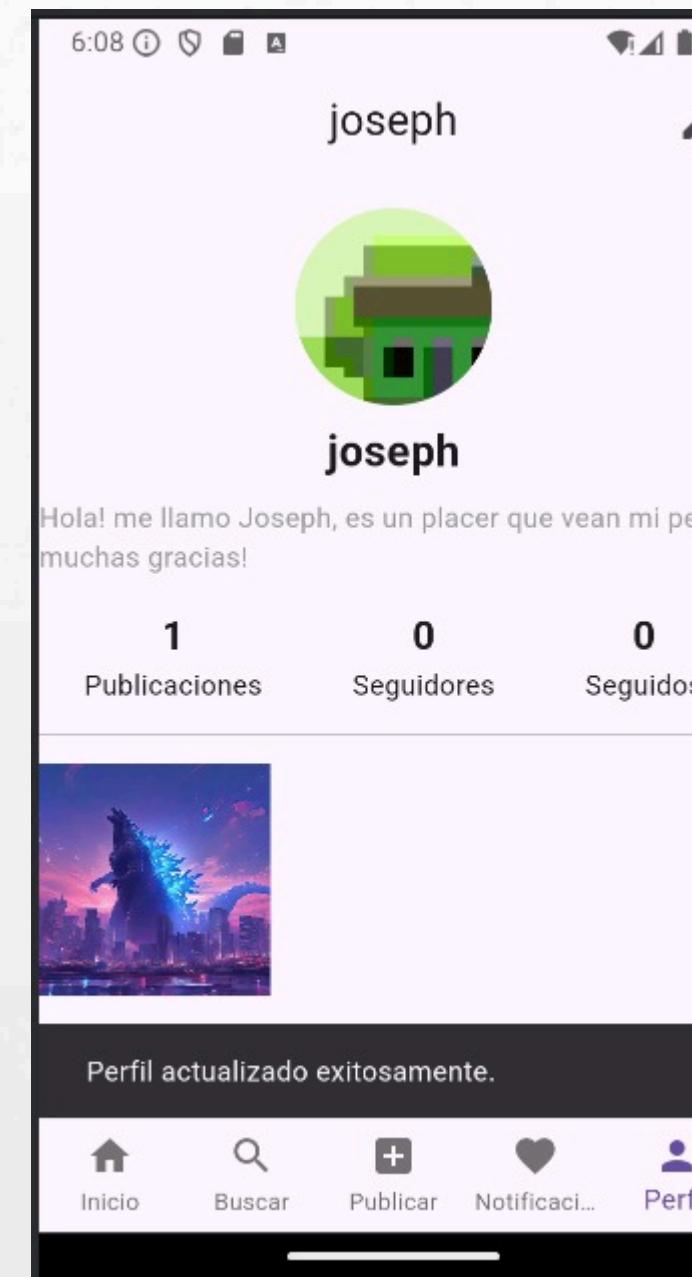
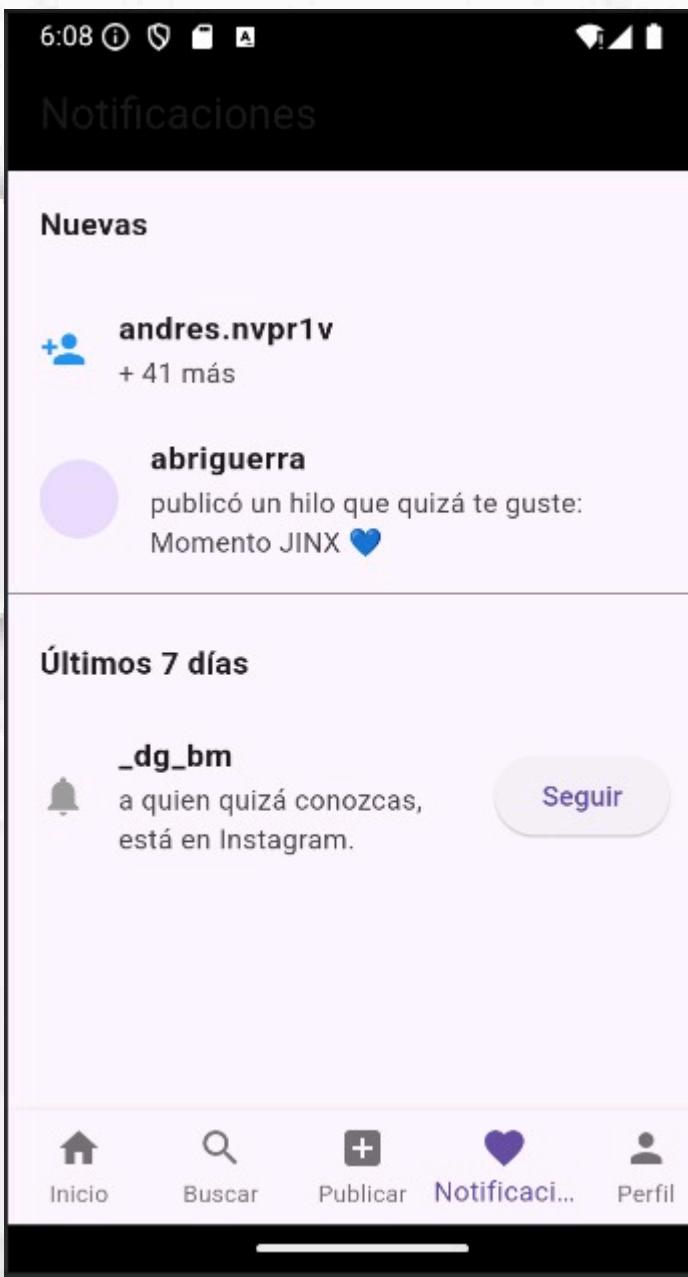
```
@app.route('/posts/<int:post_id>/comments', methods=['GET'])
def get_post_comments(post_id):
    comments = Comment.query.filter_by(post_id=post_id).all()
    if not comments:
        return jsonify({"message": "No hay comentarios para esta publicación"}), 200
    response = [{"user_id": comment.user_id,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

```
@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
```

# Resultado Interfaz



# Resultado Interfaz



# Conclusiones

El proyecto UPSGlam logró integrar tecnologías modernas como PyCUDA, Docker, Flutter, PgAdmin y Postman en el desarrollo de una plataforma funcional y escalable. Se implementaron algoritmos avanzados de convolución, una API robusta y una base de datos eficiente, gestionada con PgAdmin. Además, se utilizó Docker para la contenedorización, garantizando portabilidad y despliegue sencillo, y Flutter para una aplicación móvil intuitiva. El uso de Postman permitió pruebas exhaustivas, asegurando la calidad del sistema. Este proyecto no solo alcanzó los objetivos planteados, sino que demostró el valor de aplicar herramientas actuales para resolver problemas prácticos en entornos reales.

# Referencias Bibliográficas

- [1] R. C. González y R. E. Woods, Digital Image Processing, 3.<sup>a</sup> ed., Upper Saddle River, NJ, USA: Prentice-Hall, 2008.
- [2] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, y J. C. Phillips, "GPU Computing," Proceedings of the IEEE, vol. 96, no. 5, pp. 879-899, May 2008.
- [3] J. Serra, Image Analysis and Mathematical Morphology, New York, NY, USA: Academic Press, 1982.
- [4] B. Jahne, Digital Image Processing, 6.<sup>a</sup> ed., Berlín, Alemania: Springer-Verlag, 2005.
- [5] A. C. Gómez, "Métodos de procesamiento morfológico: Erosión y dilatación en imágenes," Revista Internacional de Tecnologías de la Información, vol. 5, no. 3, pp. 45-50, Sep. 2018.
- [6] S. López y A. Torres, "Evaluación de filtros de imágenes: un estudio comparativo," Revista Latinoamericana de Computación, vol. 15, no. 2, pp. 12-18, Jul. 2021.

**¡MUCHAS GRACIAS!**