Manual Técnico

Aplicación Web para la Gestión de PQRSD

Versión 1.1 04 de noviembre 2025

Alcaldía de Girardot

SECRETARÍA DE TRÁNSITO Y TRANSPORTE

Tabla de Contenido

Lista de Figuras

Lista de Tablas

1. PRESENTACIÓN

El presente Manual Técnico documenta el sistema web PQRSD (Peticiones, Quejas, Reclamos, Sugerencias y Denuncias) desarrollado para la Secretaría de Tránsito y Transporte de Girardot, Cundinamarca.

El sistema tiene como propósito optimizar la gestión interna de solicitudes ciudadanas, permitiendo su recepción, seguimiento, respuesta y cierre de manera centralizada, transparente y trazable.

La plataforma fue diseñada para ser accesible desde cualquier navegador web y cuenta con una arquitectura moderna que garantiza seguridad, escalabilidad y facilidad de mantenimiento.

Está desarrollada bajo un enfoque modular y basado en contenedores Docker, lo que permite su despliegue rápido y confiable tanto en entornos locales como en servidores de producción.

Este documento contiene la información técnica necesaria para instalar, configurar y desplegar el sistema, así como las especificaciones de hardware, software, estructura arquitectónica y procedimientos de verificación.

El manual está dirigido principalmente al personal técnico de la Secretaría de Tránsito y Transporte, así como a los funcionarios encargados de la administración del sistema, brindando las herramientas necesarias para comprender su funcionamiento y resolver posibles incidencias.

El objetivo principal es ofrecer una guía completa y práctica que permita garantizar el correcto funcionamiento del sistema y la continuidad del servicio de atención ciudadana.

2. FUNCIONAMIENTO DEL SISTEMA

El sistema PQRSD está diseñado bajo una arquitectura por capas e implementado utilizando contenedores Docker, lo que permite una correcta separación de responsabilidades y un despliegue eficiente en distintos entornos.

2.1 Capa de Presentación (Frontend)

Corresponde a la interfaz web utilizada por los ciudadanos y los funcionarios de la Secretaría. Está desarrollada con React 18+ y Vite, utilizando HTML, TAILWIND Y TYPESCRIPT. Desde esta capa se permite a los usuarios registrar PQRSDF, consultar el estado de sus solicitudes y recibir notificaciones.

La interfaz también incluye el panel administrativo, desde el cual los funcionarios gestionan las solicitudes, generan reportes y realizan seguimiento de tiempos de respuesta. El contenedor del frontend se ejecuta en el puerto externo 25000, accediendo al backend mediante una API REST.

2.2 Capa de Lógica de Negocio (Backend)

Es el núcleo del sistema, desarrollado en Java 21 con el framework Spring Boot 3.x, estructurado bajo el patrón RESTful. Esta capa gestiona la lógica de negocio, el control de roles y permisos, el manejo de autenticación mediante JWT, la generación de reportes y la comunicación con la base de datos.

La API está documentada con Swagger UI, accesible desde http://localhost:27845/swagger-ui/index.html.
El backend corre dentro de un contenedor Docker y se comunica con la base de datos PostgreSQL a través de una red interna definida en el archivo Docker-compose.yml.

2.3 Capa de Datos (Base de Datos)

El sistema utiliza PostgreSQL 15.5 como motor de base de datos relacional. Esta capa almacena de manera segura toda la información relacionada con usuarios, roles, PQRSDF, adjuntos y registros de auditoría.

El contenedor de la base de datos se inicializa automáticamente con los scripts definidos en pq.sql, asegurando que las tablas y esquemas requeridos estén disponibles desde el primer despliegue.

3. RESUMEN EJECUTIVO

3.1 Descripción general del sistema

El sistema PQRSD es una aplicación web orientada a fortalecer la gestión administrativa y la atención al ciudadano dentro de la Secretaría de Tránsito y Transporte de Girardot.

Su objetivo principal es centralizar el proceso de recepción, clasificación, seguimiento y respuesta de solicitudes, permitiendo una trazabilidad completa de cada caso desde su registro hasta su cierre.

Además, el sistema contribuye al cumplimiento de la normatividad sobre atención ciudadana y transparencia institucional, al proporcionar reportes automáticos y una base de datos histórica consultable.

3.2 Estructura del manual

El manual técnico del sistema PQRSD está compuesto por doce capítulos, organizados de manera secuencial para guiar al lector desde la comprensión general del sistema hasta su instalación, despliegue y mantenimiento.

En los primeros capítulos se presentan los requisitos técnicos necesarios para la correcta implementación del sistema. Posteriormente, se describen en detalle la arquitectura de la aplicación, las tecnologías utilizadas, los diagramas de casos de uso y el modelo de datos que sustentan su funcionamiento.

Los capítulos intermedios explican de forma práctica el proceso de configuración y despliegue del sistema en entornos Docker, incluyendo la estructura de los archivos de configuración (Dockerfile, docker-compose.yml y .env), así como los pasos para la verificación del funcionamiento de cada contenedor (base de datos, backend y frontend).

Finalmente, el manual concluye con orientaciones sobre el mantenimiento preventivo y correctivo del sistema, además de recomendaciones técnicas para garantizar su disponibilidad, integridad y rendimiento a largo plazo.

3.2.1 Finalidad del manual

La finalidad de este manual es proporcionar una guía técnica detallada y estructurada que permita al personal de la Secretaría de Tránsito y Transporte:

- Instalar y desplegar correctamente el sistema PQRSD.
- Comprender su arquitectura y componentes internos.
- Identificar, mantener y solucionar posibles incidencias técnicas.

Asimismo, busca garantizar que el sistema cumpla con los requisitos de disponibilidad, integridad y rendimiento, asegurando una atención eficiente a los ciudadanos y un soporte técnico sostenible en el tiempo.

4. REQUISITOS DEL SISTEMA

4.1 Características del Sistema

Antes de continuar con la instalación del proyecto en cualquier máquina, es fundamental verificar que el servidor cuente con los siguientes requisitos y configuración previa.

Características	Especificaciones recomendadas
CPU asignada	2 núcleo virtual (vCPU)
Memoria RAM	4 GB (dependiendo del plan)
Almacenamiento	Mínimo 40 GB
Sistema de Contenedores	Docker (versión 24 o superior) y Docker Compose (versión 2 o superior) instalados y en ejecución
Ancho de Banda	Escalable según la demanda y límites del plan contratado

Tabla 1. *Requisitos Mínimos de Hardware para la Instalación del Proyecto*

Fuente. *Autoría Propia*

4.2 Requisitos de Software y Configuración Previa

Requisito	Descripción
Sistema de contenedores	Docker versión 24 o superior, utilizado para la creación y ejecución de los contenedores del proyecto
Orquestador de servicios	Docker Compose versión 2 o superior, empleado para gestionar múltiples servicios (Frontend, Backend y base de datos)
Imagen base del Backend	Eclipse Temurin 21 (OpenJDK 21 sobre Alpine Linux), utilizada dentro del contenedor para ejecutar la aplicación Spring Boot.
Framework Backend	Spring Boot 3.x, compilado y ejecutado dentro del contenedor de backend mediante Maven.

Framework Frontend	React 18+, compilado y desplegado dentro del contenedor de Frontend con Vite y Node.js 18+.
Base de datos	PostgreSQL 15.5, instalada localmente
Cliente de base de datos (opcional)	pgAdmin 4 o herramienta psql (pueden instalarse localmente si se requiere acceso manual a la base de datos).
Navegador compatible	Google Chrome, Microsoft Edge o Firefox (últimas versiones)
Git	Para clonado y versionado del proyecto

Tabla 2. *Requisitos Técnicos del Entorno*

Fuente. *Autoría Propia*

5. REQUISITOS DE HARDWARE Y SOFTWARE DEL EQUIPO DEL CLIENTE

5.1 Ciudadanos

Los ciudadanos del sistema no necesitan realizar instalaciones locales ni configuraciones avanzadas, ya que el acceso se efectúa mediante navegadores web. A continuación, se especifican los requisitos mínimos recomendados para un uso óptimo:

Sistema operativo: Compatible con cualquier sistema operativo que cuente con un navegador web, incluyendo Windows 11, Windows 10, Windows 8.1, Windows 8, Linux y Android.

Navegador web: Compatible con los principales navegadores, como Google Chrome, Microsoft Edge y Mozilla Firefox. Se recomienda utilizar versiones recientes para garantizar un mejor rendimiento y compatibilidad.

Resolución de pantalla: El diseño del sistema es completamente adaptable (responsive), permitiendo una visualización óptima en diferentes dispositivos, incluyendo computadoras, tabletas y teléfonos celulares. La interfaz se ajusta automáticamente al tamaño de pantalla, asegurando una experiencia de usuario fluida y cómoda.

Conectividad: Es necesario contar con una conexión estable a Internet para el correcto funcionamiento del sistema. Se recomienda una conexión de banda ancha con una velocidad mínima de 5 Mbps para garantizar un acceso rápido y sin interrupciones a todas las funciones disponibles.

5.2 Asignadores, Funcionarios y Administrador

Los usuarios con roles administrativos o de gestión, como asignadores, funcionarios y administradores, tampoco requieren instalaciones locales ni configuraciones avanzadas. El acceso se realiza mediante navegadores web desde equipos de escritorio o portátiles. A continuación, se detallan los requisitos mínimos recomendados para un funcionamiento óptimo:

Sistema operativo: Compatible con sistemas operativos de escritorio que cuenten con un navegador web, incluyendo Windows 11, Windows 10, Windows 8.1, Windows 8 y distribuciones Linux.

Navegador web: Compatible con los principales navegadores web, como Google Chrome, Microsoft Edge y Mozilla Firefox. Se recomienda utilizar las versiones más recientes para asegurar el correcto funcionamiento y la máxima compatibilidad con todas las funciones del sistema.

Resolución de pantalla: La interfaz para estos usuarios está optimizada exclusivamente para computadoras, garantizando una visualización clara, ordenada y eficiente en pantallas de mayor tamaño. No se recomienda el uso desde dispositivos móviles, ya que ciertas funcionalidades pueden no mostrarse correctamente.

Conectividad: Es indispensable contar con una conexión a Internet estable y de buena calidad. Se recomienda una velocidad mínima de 10 Mbps para asegurar la carga rápida de datos, el acceso simultáneo a múltiples módulos y un desempeño fluido en operaciones administrativas.

6. PORTABILIDAD DEL SISTEMA

El sistema presenta una alta portabilidad al estar desarrollado bajo una arquitectura cliente-servidor, utilizando tecnologías web modernas y de código abierto. Esta arquitectura permite una clara separación entre el frontend y el backend, facilitando el mantenimiento, la escalabilidad y la implementación en distintos entornos sin necesidad de herramientas adicionales como Docker u orquestadores externos.

Sus principales ventajas incluyen:

Arquitectura cliente-servidor: El sistema está dividido en dos capas principales: un cliente desarrollado en React, encargado de la interfaz de usuario y la interacción con el usuario final, y un servidor implementado con Spring Boot, responsable de la lógica de negocio y la gestión de datos. Esta separación permite mantener y actualizar cada componente de forma independiente, mejorando la escalabilidad y la estabilidad del sistema.

Backend en Spring Boot (Java): El servidor está desarrollado con Spring Boot, lo que permite un despliegue nativo directo en el servidor sin requerir contenedores ni configuraciones complejas. Puede ejecutarse fácilmente mediante un archivo .jar o integrarse como un servicio del sistema operativo. Su estructura modular y la gestión automática de dependencias proporcionan un entorno robusto y flexible.

Frontend en React: El cliente, desarrollado con React, ofrece una interfaz moderna, dinámica y eficiente. Su diseño basado en componentes facilita la actualización y personalización de la interfaz sin afectar la lógica del backend. Además, el diseño es responsive, asegurando compatibilidad con diferentes resoluciones de pantalla y dispositivos.

Base de datos PostgreSQL: El sistema utiliza PostgreSQL como gestor de base de datos, reconocido por su estabilidad, seguridad y rendimiento. Está configurado para integrarse directamente con el backend sin depender de servicios externos, y puede migrarse fácilmente a otras instancias o servidores manteniendo la integridad y consistencia de los datos

7. ADAPTABILIDAD A CAMBIOS DEL ENTORNO

El sistema ha sido diseñado bajo una arquitectura cliente-servidor modular y flexible, lo que permite una alta capacidad de adaptación a nuevas necesidades funcionales y técnicas. Su estructura favorece el mantenimiento, la escalabilidad y la integración con otros servicios o aplicaciones externas.

Entre sus principales características se destacan:

Arquitectura modular y flexible: La separación entre el cliente (React) y el servidor (Spring Boot) permite realizar actualizaciones, mejoras o correcciones de manera independiente, sin afectar la operatividad global del sistema. Esta independencia facilita la evolución tecnológica y la incorporación de nuevas funcionalidades.

Escalabilidad adaptable: El sistema admite escalabilidad vertical, mediante la ampliación de los recursos del servidor (CPU, memoria o almacenamiento), y escalabilidad horizontal, al distribuir las solicitudes entre múltiples instancias del backend. Esto garantiza un rendimiento estable incluso con un alto número de usuarios concurrentes.

Backend estructurado en Spring Boot: La organización modular del backend, basada en controladores, servicios y repositorios, permite un crecimiento ordenado y facilita el mantenimiento del código. Además, el uso de controladores REST simplifica la comunicación con el frontend y con servicios externos.

Compatible con APIs RESTful: La arquitectura REST implementada en el backend facilita la interoperabilidad con otros sistemas o aplicaciones externas, permitiendo el intercambio de datos mediante peticiones HTTP seguras y estandarizadas.

Frontend en React: Las interfaces están desarrolladas en React, utilizando componentes reutilizables que aseguran consistencia visual y eficiencia en la renderización. El diseño es responsivo, adaptándose a distintas resoluciones y dispositivos, garantizando una experiencia de usuario óptima tanto en equipos de escritorio como en dispositivos móviles.

Base de datos PostgreSQL: La base de datos está implementada en PostgreSQL, ofreciendo integridad transaccional, seguridad y soporte para operaciones complejas. Su configuración permite una conexión directa y eficiente con el

backend, facilitando la administración y la portabilidad a otros entornos si es necesario.

8. PLAN DE CONTINGENCIA

El sistema, desplegado de forma nativa en un servidor local, cuenta con medidas preventivas y procedimientos de recuperación orientados a garantizar la continuidad del servicio ante fallos técnicos, errores humanos o incidentes de seguridad. Estas estrategias buscan minimizar el tiempo de inactividad y asegurar la integridad de la información.

Entre las acciones recomendadas se incluyen:

Recuperación ante desastres: En caso de pérdida de datos o fallas críticas en la aplicación, el sistema permite restaurar la información a partir de respaldos automáticos de la base de datos PostgreSQL, generados diariamente a las 2:00 a.m.

Adicionalmente, los administradores pueden generar copias de seguridad manualmente desde la interfaz de administración mediante un botón dedicado.

En caso de una pérdida total del servicio, el sistema puede ser reinstalado desde el repositorio de código fuente en GitHub, lo que permite restablecer el entorno rápidamente.

Escalabilidad y continuidad operativa: Aunque el sistema se ejecuta en un servidor local, se recomienda mantener un servidor de respaldo con una copia actualizada del sistema y de la base de datos para garantizar la recuperación inmediata en caso de fallas graves o mantenimiento no programado.

La estructura modular del sistema permite su migración a otro servidor o entorno de producción con configuraciones mínimas, sin alterar el funcionamiento general.

Medidas de seguridad preventiva: La seguridad del sistema depende principalmente de las configuraciones implementadas en el servidor local donde se aloje.

Se recomienda:

- Mantener el servidor protegido mediante firewall, acceso restringido por roles y protocolos seguros de conexión (HTTPS o SSH).
- Limitar el acceso a puertos o servicios innecesarios para reducir la superficie de ataque.

- Actualizar regularmente el sistema operativo, dependencias de Java, librerías de React y PostgreSQL.
- Restringir el acceso a la interfaz de administración solo a personal autorizado.

Gestión de registros (logs): El sistema genera registros internos de actividad, almacenando información sobre el usuario, ruta accedida, tipo de petición (GET, POST, PATCH, PUT o DELETE) y el estado de la respuesta.

Estos registros permiten realizar auditorías básicas y verificar la correcta interacción de los usuarios con el sistema, aunque no contemplan métricas de rendimiento ni alertas automáticas.

Código fuente y control de versiones: El proyecto se encuentra disponible de forma pública en GitHub, lo que facilita la recuperación del código, el seguimiento de cambios y la colaboración en futuras mejoras. Esta práctica contribuye a la transparencia, mantenibilidad y posibilidad de restauración en caso de pérdida del entorno local.

9. DICCIONARIO DE DATOS

Campo	Tipo de dato	Descripción	Clave
id	SERIAL	Identificador único de cada registro en la tabla. Se genera automáticamente.	PK
pq_id	INTEGER	Identificador del registro asociado en la tabla principal de PQRSD. Representa la relación entre el adjunto y la solicitud correspondiente.	FK → pqs(id)
nombre_archivo	VARCHAR	Nombre original del archivo adjunto cargado por el usuario.	
ruta_archivo	VARCHAR	Ruta o ubicación en el servidor donde se almacena físicamente el archivo.	
created_at	TIMESTAMP	Fecha y hora en que se realizó la carga del archivo. Permite llevar control temporal de los registros	
Respuesta	BOOLEAN	Indica si el archivo corresponde a una respuesta emitida por la entidad (TRUE) o si fue un archivo adjunto por el ciudadano (FALSE).	
deleted_at	TIMESTAMP	Fecha y hora en la que el registro fue eliminado lógicamente del sistema.	

Tabla 3. *Diccionario de Datos – Adjuntos_pq*
Fuente. Autoría Propia

Campo	Tipo de dato	Descripción	Clave
Id	SERIAL	Identificador único de cada área registrada. Se genera automáticamente por el sistema.	PK
codigo_dep	VARCHAR	Código identificador del departamento o dependencia dentro de la Secretaría de Tránsito y Transporte.	

Nombre	VARCHAR	Nombre del área o dependencia responsable (por ejemplo: Atención al Ciudadano, Jurídica, Técnica, etc.).	
deleted_at	TIMESTAMP	Fecha y hora en que el registro fue eliminado lógicamente del sistema. Si el valor es nulo, el área sigue activa.	

Tabla 4. *Diccionario de Datos – Areas_resp*
Fuente: *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
Id	SERIAL	Identificador único del registro de auditoría. Se genera automáticamente por el sistema.	PK
username	VARCHAR	Nombre de usuario que realizó la acción dentro del sistema. Permite rastrear quién ejecutó una operación.	
action	VARCHAR	Acción realizada por el usuario (por ejemplo: crear, actualizar, eliminar, iniciar sesión).	
method	VARCHAR	Método HTTP utilizado en la solicitud (por ejemplo: GET, POST, PUT, DELETE).	
endpoint	TEXT	Ruta o endpoint del sistema que fue accedido durante la acción.	
status_code	INTEGER	Código de estado HTTP devuelto por el servidor tras la ejecución de la acción (por ejemplo: 200, 404, 500).	
timestamp	TIMESTAMP	Fecha y hora exacta en la que ocurrió la acción registrada.	

Tabla 5. *Diccionario de Datos – Audit_log*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave

id	SERIAL	Identificador único del departamento. Se genera automáticamente por el sistema.	PK
nombre	VARCHAR	Nombre del departamento (por ejemplo: Cundinamarca, Tolima, Huila).	
codigo_dane	VARCHAR	Código asignado por el DANE que identifica oficialmente el departamento dentro del territorio nacional.	

Tabla 6. *Diccionario de Datos – Departamentos*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
id	SERIAL	Identificador único del estado de la PQRSD. Se genera automáticamente por el sistema.	PK
nombre	VARCHAR	Nombre del estado asignado a una PQRSD (por ejemplo: “Radicado”, “En proceso”, “Resuelto”, “Cerrado”).	
color	VARCHAR	Código de color (en formato hexadecimal o texto) que representa visualmente el estado dentro de la interfaz del sistema.	
descripcion	VARCHAR	Descripción breve del significado o propósito del estado.	

Tabla 7. *Diccionario de Datos – Estados_pq*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
id	SERIAL	Identificador único del género. Se genera automáticamente por el sistema.	PK
nombre	VARCHAR	Nombre del género registrado (por ejemplo: Masculino, Femenino, Otro).	

Tabla 8. *Diccionario de Datos – Géneros*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
id	SERIAL	Identificador único del municipio. Se genera automáticamente por el sistema.	PK
nombre	VARCHAR	Nombre oficial del municipio (por ejemplo: Girardot, Nilo, Tocaima).	
codigo_dane	VARCHAR	Código asignado por el DANE que identifica de forma única al municipio dentro del país.	
departamento_id	INTEGER	Identificador del departamento al que pertenece el municipio.	

Tabla 9. *Diccionario de Datos – Municipios*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
Id	SERIAL	Identificador único del tipo de documento. Se genera automáticamente por el sistema.	PK
nombre	VARCHAR	Nombre del tipo de documento (por ejemplo: Cédula de Ciudadanía, Tarjeta de Identidad, Pasaporte, NIT, etc.).	

Tabla 10. *Diccionario de Datos – Tipos_docs*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
id	SERIAL	Identificador único del tipo de persona. Se genera automáticamente por el sistema.	PK
nombre	VARCHAR	Nombre o categoría del tipo de persona dentro del sistema (por ejemplo: Ciudadano, funcionario STTG, Asignador STTG, Administrador).	

Tabla 11. *Diccionario de Datos – Tipos_personas*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
Id	SERIAL	Identificador único de la persona. Se genera automáticamente por el sistema.	PK
Nombre	VARCHAR	Nombres completos de la persona registrada.	
Apellido	VARCHAR	Apellidos de la persona registrada.	
tipo_doc	INTEGER	Identificador del tipo de documento (por ejemplo: cédula, tarjeta de identidad, pasaporte).	FK → tipos_docs(id)
Dni	VARCHAR	Número del documento de identificación personal.	
fecha_nacimiento	DATE	Fecha de nacimiento de la persona.	
tipo_persona	INTEGER	Define el tipo de persona (por ejemplo: ciudadano, funcionario, asignador, administrador).	FK → tipos_personas(id)
Telefono	VARCHAR	Número de teléfono de contacto.	
Direccion	VARCHAR	Dirección de residencia o contacto de la persona.	
tratamiento_datos	BOOLEAN	Indica si la persona autoriza el tratamiento de sus datos personales según la política de privacidad.	
municipio_id	INTEGER	Identificador del municipio al que pertenece la persona (relacionado con la tabla de municipios).	FK → municipios(id)

Genero	INTEGER	Identificador del género de la persona (masculino, femenino u otro).	FK → generos(id)
created_at	TIMESTAMP	Fecha y hora de creación del registro.	
updated_at	TIMESTAMP	Fecha y hora de la última actualización del registro.	
deleted_at	TIMESTAMP	Fecha y hora en que el registro fue eliminado lógicamente. Si es nulo, el registro sigue activo.	

Tabla 12. *Diccionario de Datos – Personas*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
id	SERIAL	Identificador único del tipo de PQRSD. Se genera automáticamente por el sistema.	PK
nombre	VARCHAR	Nombre del tipo de solicitud registrada (por ejemplo: Petición, Queja, Reclamo, Sugerencia, Denuncia).	
dias_habiles_respuesta	INTEGER	Número de días hábiles establecidos para dar respuesta según el tipo de PQRSD, conforme a la normativa institucional.	

Tabla 13. *Diccionario de Datos – Tipos_pq*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
Id	SERIAL	Identificador único de la PQRSD. Se	PK

		genera automáticamente por el sistema.	
numero_radicado	VARCHAR	Código o número de radicado asignado automáticamente a la solicitud.	
detalle_asunto	TEXT	Descripción principal del asunto o motivo de la petición, queja, reclamo, sugerencia o denuncia.	
tipo_pq_id	INTEGER	Identificador del tipo de solicitud (Petición, Queja, Reclamo, Sugerencia o Denuncia).	FK → tipos_pq(id)
solicitante_id	INTEGER	Identificador del ciudadano o persona que radica la solicitud.	FK → personas(id)
fecha_radicacion	DATE	Fecha en la que la solicitud fue registrada en el sistema.	
hora_radicacion	TIME	Hora exacta en la que se realizó la radicación.	
fecha_resolucion_estimada	DATE	Fecha estimada para la resolución de la solicitud, de acuerdo con los tiempos legales.	
fecha_resolucion	DATE	Fecha real en la que se resolvió la solicitud.	
radicador_id	INTEGER	Identificador del funcionario que realizó la radicación en el sistema (en	FK → usuarios(id)

		caso de que no sea vía web).	
responsable_id	INTEGER	Identificador del funcionario asignado para atender y responder la solicitud.	FK → usuarios(id)
Web	BOOLEAN	Indica si la solicitud fue radicada directamente desde la página web (TRUE) o por un funcionario (FALSE).	
Respuesta	TEXT	Respuesta o resolución emitida por el funcionario responsable.	
detalle_descripcion	VARCHAR	Observaciones o detalles adicionales del proceso de atención de la solicitud.	
deleted_at	TIMESTAMP	Fecha y hora en que la solicitud fue eliminada lógicamente. Si es nula, la solicitud sigue activa.	

Tabla 14. *Diccionario de Datos – Pqs*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
Id	SERIAL	Identificador único del registro en el historial de cambios de estado de una PQRSD. Se genera automáticamente por el sistema.	PK

pq_id	INTEGER	Identificador de la PQRSD a la que pertenece el historial.	FK → pqs(id)
estado_id	INTEGER	Identificador del estado al que cambió la PQRSD (por ejemplo: Radicada, En trámite, Resuelta, Cerrada).	FK → estados_pq(id)
cambiado_por_id	INTEGER	Identificador del funcionario o usuario que realizó el cambio de estado.	FK → usuarios(id)
observacion	TEXT	Comentarios u observaciones relacionadas con el cambio de estado (por ejemplo, motivo del cambio o avance del caso).	
fecha_cambio	TIMESTAMP	Fecha y hora exacta en la que se realizó el cambio de estado.	
deleted_at	TIMESTAMP	Fecha y hora en que el registro fue eliminado lógicamente. Si es nulo, el registro sigue activo.	

Tabla 15. *Diccionario de Datos – Historial_estados_pq*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
id	SERIAL	Identificador único del permiso dentro del sistema. Se genera automáticamente por el sistema.	PK
tabla	VARCHAR	Nombre de la tabla o módulo del sistema al	

		que aplica el permiso (por ejemplo: usuarios, pqs, personas).	
accion	VARCHAR	Acción o tipo de operación que puede realizar el rol sobre la tabla (por ejemplo: crear, leer, actualizar, eliminar).	
descripcion	TEXT	Descripción detallada del permiso, indicando su propósito o alcance dentro del sistema.	

Tabla 16. *Diccionario de Datos – Permisos*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
Id	SERIAL	Identificador único del registro en la relación entre roles y permisos. Se genera automáticamente por el sistema.	PK
rol_id	INTEGER	Identificador del rol asociado al permiso, relaciona con la tabla roles.	FK → roles(id)
permiso_id	INTEGER	Identificador del permiso asignado al rol, relaciona con la tabla permisos.	FK → permisos(id)

Tabla 17. *Diccionario de Datos – Roles_permisos*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave

Id	SERIAL	Identificador único del rol dentro del sistema. Se genera automáticamente por el sistema.	PK
Nombre	VARCHAR	Nombre del rol asignado al usuario (por ejemplo: Administrador, Asignador STTG, Funcionario STTG, Ciudadano).	
created_at	DATE	Fecha en la que se creó el registro del rol en el sistema.	
updated_at	DATE	Fecha de la última actualización del registro del rol.	

Tabla 18. *Diccionario de Datos – Roles*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
id	SERIAL	Identificador único del usuario dentro del sistema. Se genera automáticamente por el sistema.	PK
correo	VARCHAR	Correo electrónico del usuario utilizado para el inicio de sesión y la comunicación oficial. Debe ser único.	
contrasena	VARCHAR	Contraseña cifrada del usuario para el acceso al sistema.	
is_enable	BOOLEAN	Indica si la cuenta del usuario está activa (TRUE) o deshabilitada (FALSE).	

reset_token	VARCHAR	Token temporal utilizado para restablecer la contraseña del usuario cuando lo solicita.	
reset_token_expiration	TIMESTAMP	Fecha y hora de expiración del token de restablecimiento de contraseña.	
persona_id	INTEGER	Identificador de la persona asociada al usuario, permitiendo vincular la información personal registrada.	FK → personas(id)
rol_id	INTEGER	Identificador del rol asignado al usuario, el cual determina los permisos y accesos dentro del sistema.	FK → roles(id)
created_at	TIMESTAMP	Fecha y hora de creación del registro del usuario.	
updated_at	TIMESTAMP	Fecha y hora de la última actualización del registro.	
deleted_at	TIMESTAMP	Fecha y hora en que el registro fue eliminado lógicamente. Si es nulo, el usuario sigue activo en el sistema.	

Tabla 19. *Diccionario de Datos – Usuarios*
Fuente. *Autoría Propia*

Campo	Tipo de dato	Descripción	Clave
Id	SERIAL	Identificador único del registro de asignación de responsables dentro del sistema. Se genera	PK

		automáticamente por el sistema.	
persona_responsable_id	INTEGER	Identificador de la persona designada como responsable de atender una o varias PQRSD.	FK → personas(id)
area_id	INTEGER	Identificador del área o dependencia a la que pertenece el responsable.	FK → areas_resp(id)
fecha_asignacion	TIMESTAMP	Fecha y hora en que se asignó al responsable dentro del sistema o para un caso específico.	
is_active	BOOLEAN	Indica si el responsable actualmente se encuentra activo dentro del sistema o área asignada.	
deleted_at	TIMESTAMP	Fecha y hora en que el registro fue eliminado lógicamente. Si es nulo, el registro sigue activo.	

Tabla 20. *Diccionario de Datos – Responsables_pq*
Fuente. *Autoría Propia*

10. DEFINICIÓN DE LOS CASOS DE USO

Para la profundización de los casos de usos (Tablas de especificación), se recomienda acceder al repositorio de la universidad piloto de Colombia **REPILO**, donde dispondrá del documento completo del proyecto. A continuación, se presentan los diagramas de casos de uso.

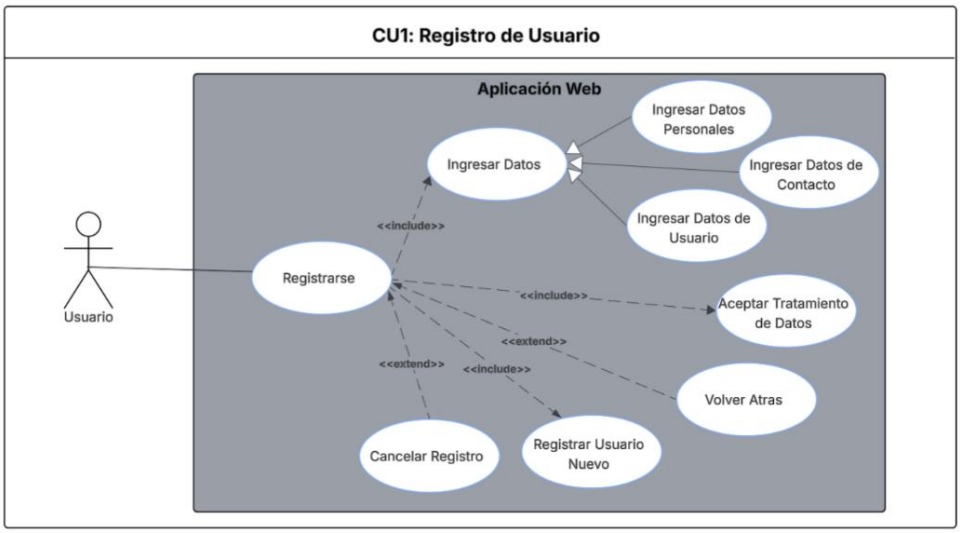


Figura 1. *Diagrama de Caso de Uso: Registro de Usuario*
Fuente. *Autoría Propia*

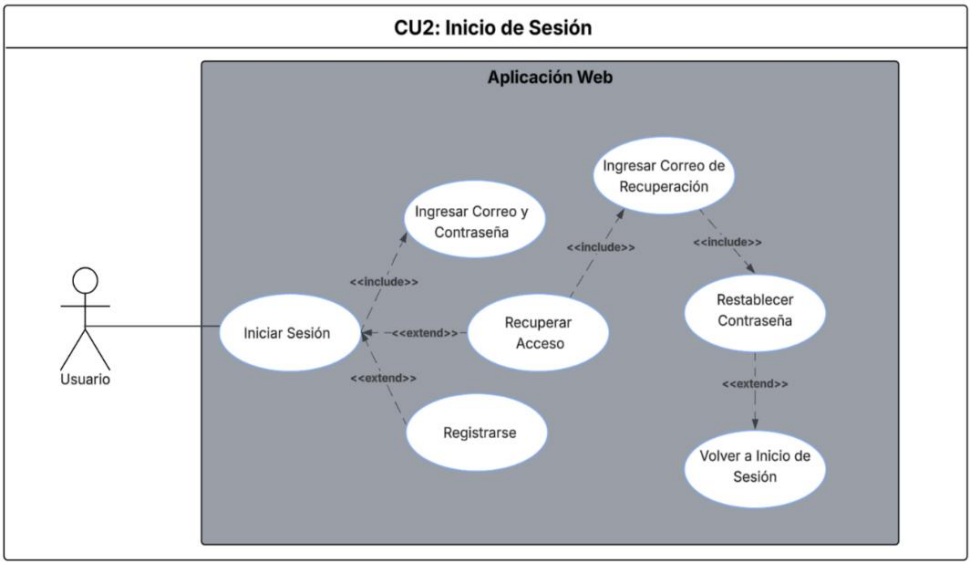


Figura 2. *Diagrama de Caso de Uso: Inicio de Sesión*

Fuente. *Autoría Propia*

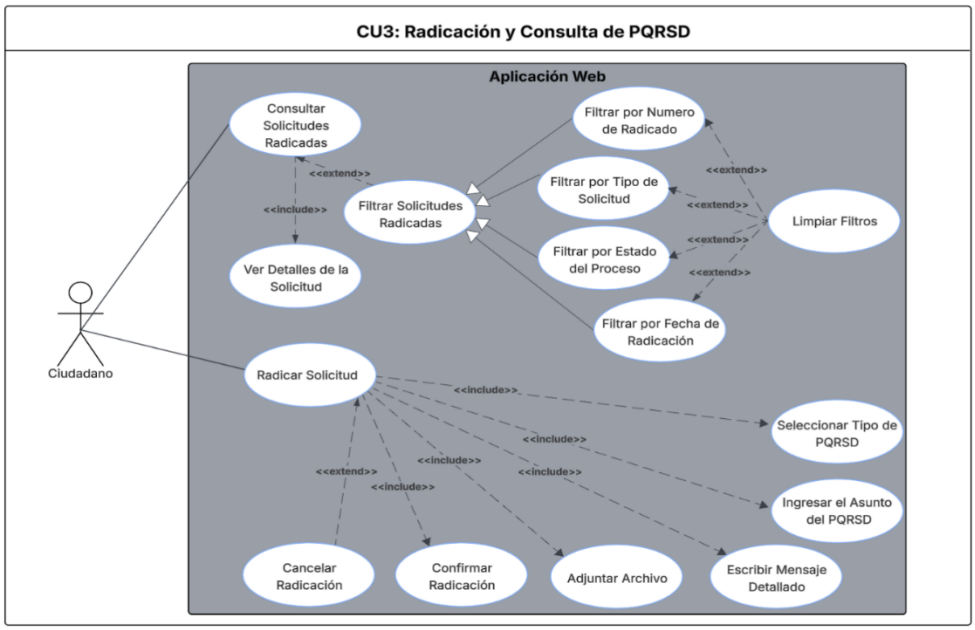


Figura 3. *Diagrama de Caso de Uso: Radicación y Consulta de PQRSD*
Fuente. *Autoría Propia*

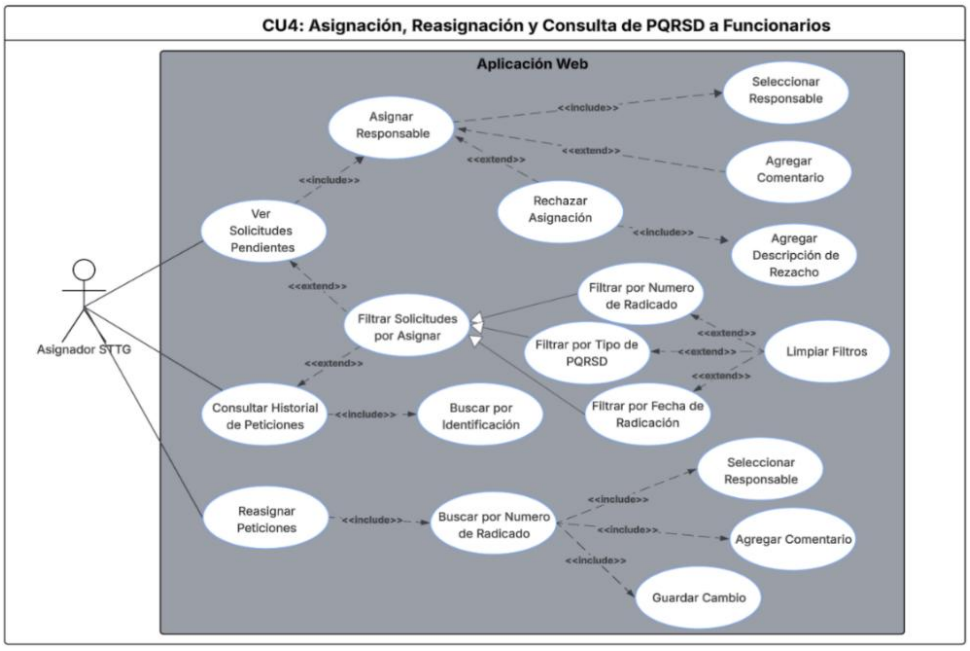


Figura 4. *Diagrama de Caso de Uso: Asignación, Reasignación y Consulta de PQRSD a Funcionarios*
Fuente. *Autoría propia*

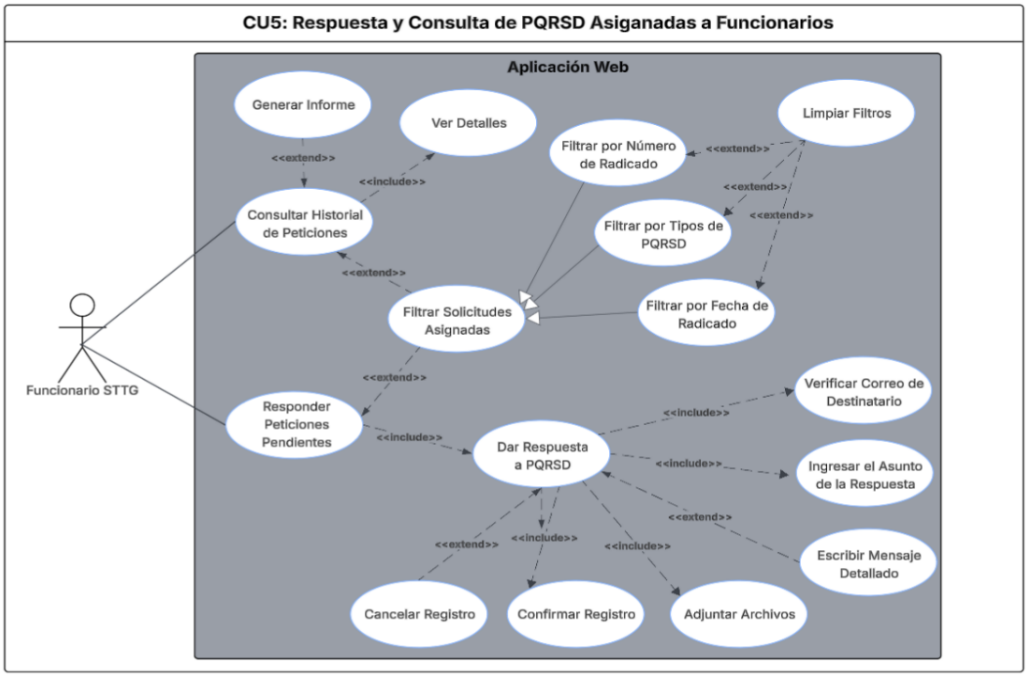


Figura 5. *Diagrama de Caso de Uso: Respuesta y Consulta de PQRSD Asignadas a Funcionarios*
Fuente. *Autoría Propia*

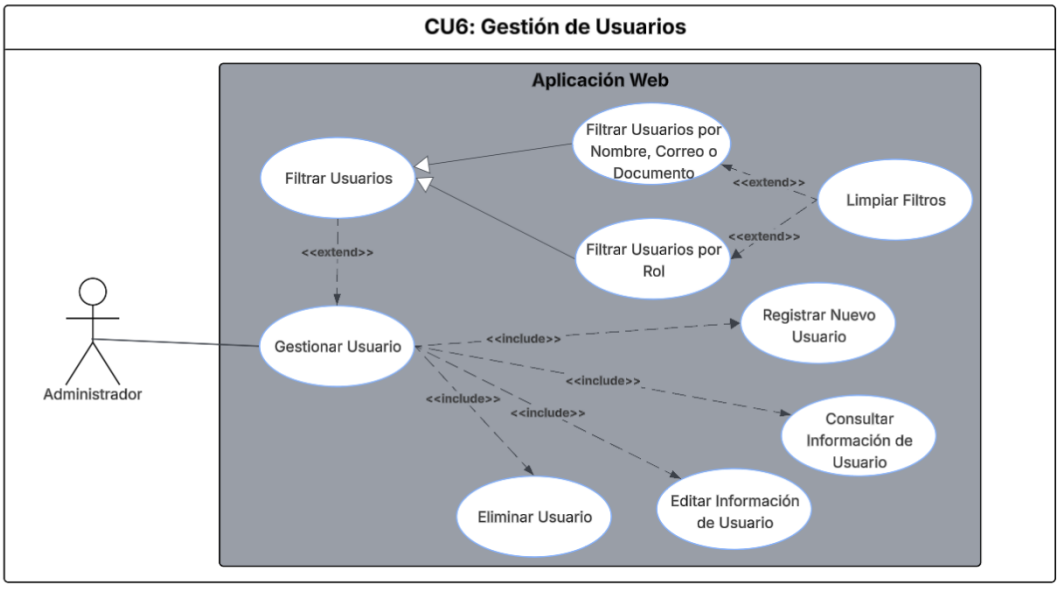


Figura 6. *Diagrama de Caso de Uso: Gestión de Usuarios*
Fuente. *Autoría Propia*

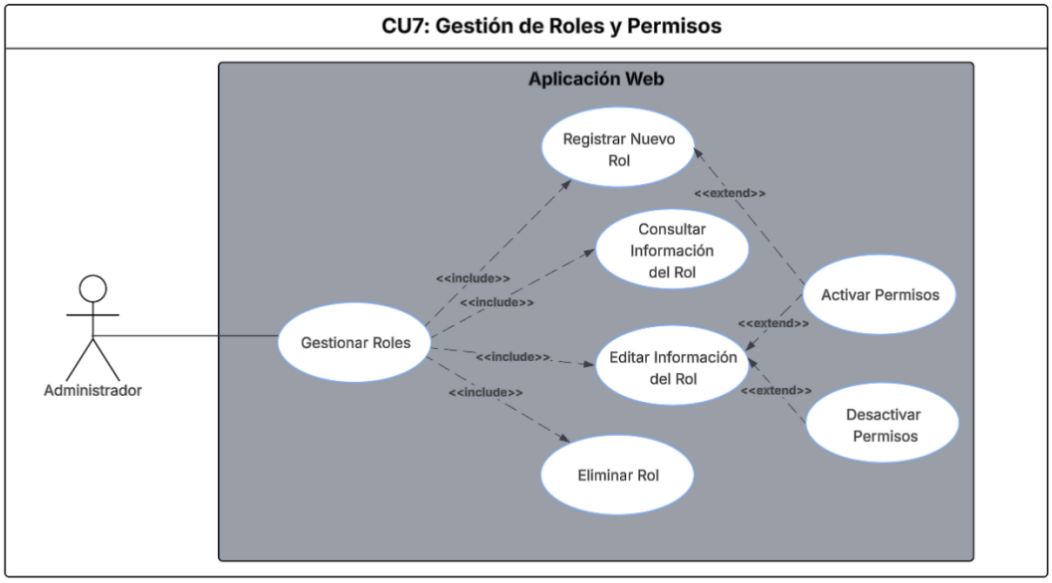


Figura 7. *Diagrama de Caso de Uso: Gestión de Roles y Permisos*
Fuente. *Autoría Propia*

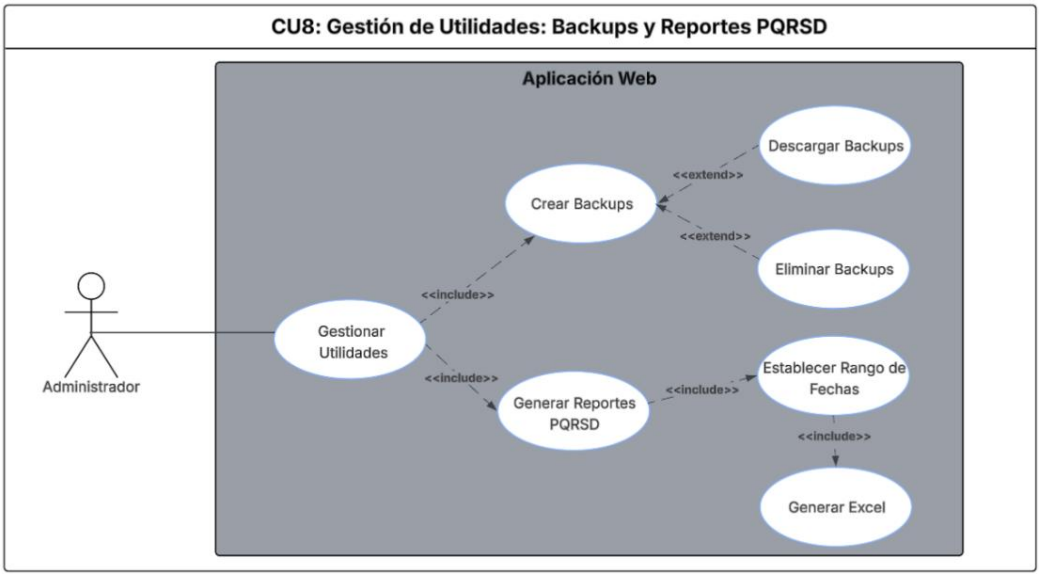


Figura 8. *Diagrama de Caso de Uso: Gestión de Utilidades: Backups y Reportes PQRSD*
Fuente. *Autoría Propia*

11. DESPLIEGUE DEL PROYECTO

En el presente apartado se detalla el proceso de despliegue del proyecto sobre un sistema operativo Debian 12, tomando en cuenta los requisitos previos, la configuración del entorno y la instalación de dependencias necesarias para su correcto funcionamiento. Se explicarán paso a paso las acciones realizadas para garantizar que la aplicación se ejecute de manera estable y segura, incluyendo la instalación de herramientas clave mencionadas anteriormente y otras variables de entorno. Este enfoque asegura que el proyecto sea reproducible en cualquier entorno basado en Debian 12, facilitando su mantenimiento y escalabilidad futura.

11.1 Instalación de WSL (Subsistema de Windows para Linux)

Ingresamos a PowerShell como administrador y ejecutamos el comando `wsl –install`

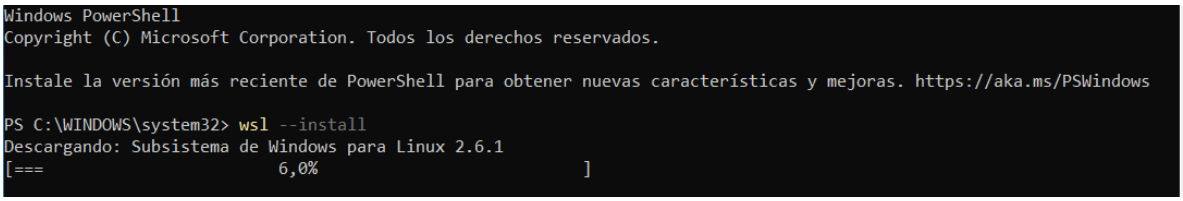


Figura 9. *Instalación de WSL*
Fuente. *Autoría Propia*

Comprobamos la instalación del WSL con el comando `wls –version`

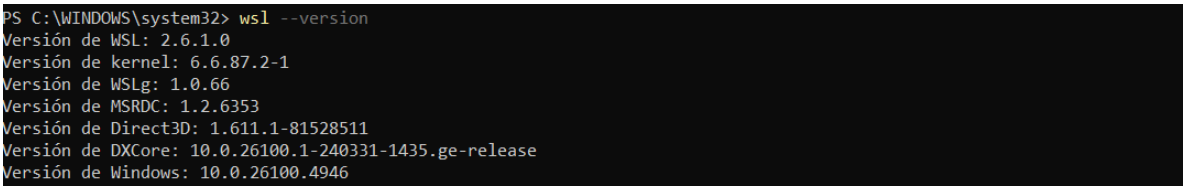


Figura 10. *Comprobación instalación WSL*
Fuente. *Autoría Propia*

11.2 Instalación de Docker Desktop

Para la instalación de Docker, es necesario entrar a la **Página Oficial de Docker**, con el fin de descargar la versión necesaria según la versión de Windows, una vez descargado empezamos con la instalación.

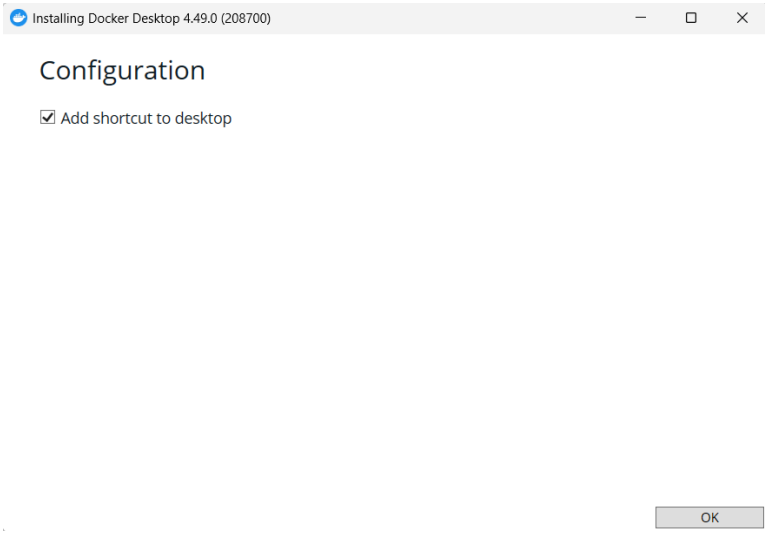


Figura 11. *Instalación de Docker Desktop*
Fuente. *Autoría Propia*

Por último, aceptamos y reiniciamos el dispositivo

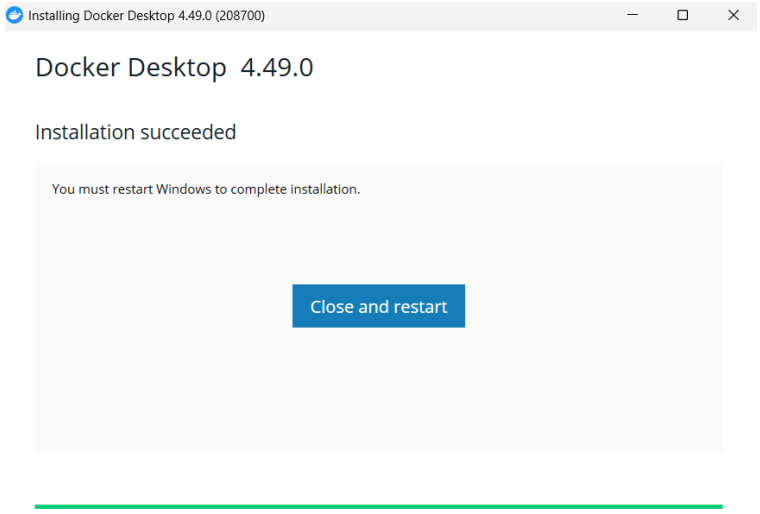


Figura 12. *Finalización de Instalación de Docker Desktop*
Fuente. *Autoría Propia*

Una vez iniciada la aplicación y aceptados los términos de uso, realizamos el registro, con alguno de los métodos de inicio de sesión disponible, se mostrará una ventana como la siguiente, confirmado el funcionamiento e instalación

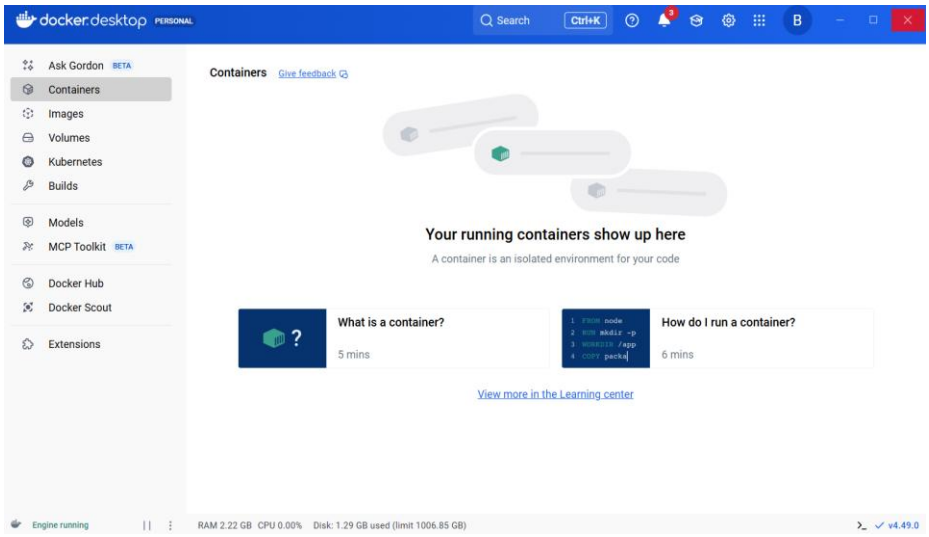


Figura 13. *Vista Inicial de Docker Desktop*
Fuente. *Autoría Propia*

11.3 Instalación de Git

Para la instalación de Git, es necesario entrar a la **Página Oficial,** con el fin de descargar el instalador, una vez descargado, será necesario configurar una identidad global con ayuda de los siguientes comandos

- `git config --global user.name "Tu Nombre"`
- `git config --global user.email “tunombre@ejemplo.com”`

```
Brayam Stiven Moreno@DESKTOP-EG90000 MINGW64 ~
$ git config --global user.name "Brayam Moreno"

Brayam Stiven Moreno@DESKTOP-EG90000 MINGW64 ~
$ git config --global user.email "brayanmoreno650@gmail.com"
```

Figura 14. *Configuración Global de Git*
Fuente. *Autoría Propia*

Y después verificar la configuración con

- `git config –list`

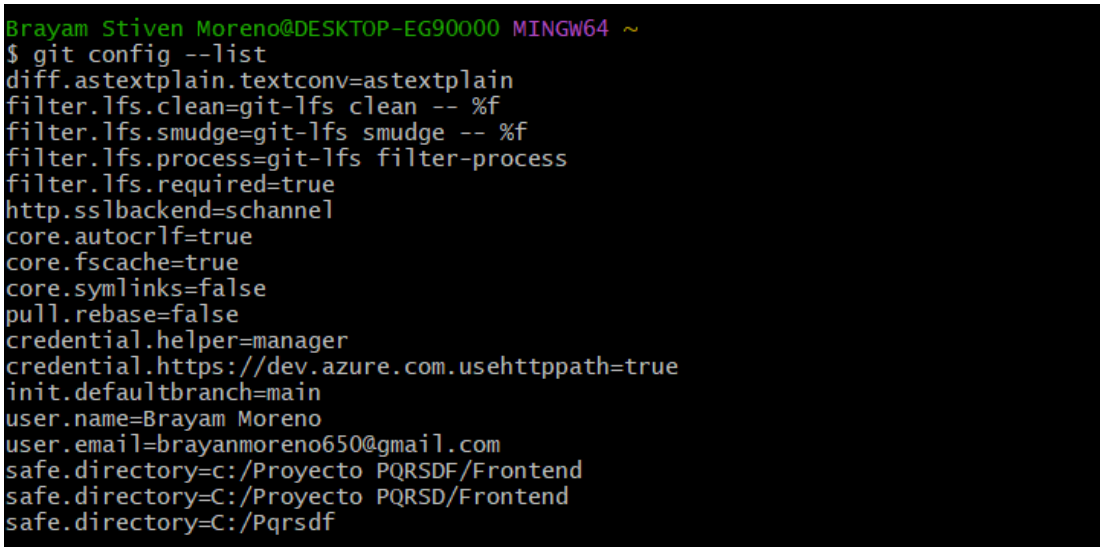


Figura 15. *Comprobación de Configuración Global de Git*
Fuente. *Autoría Propia*

11.4 Creación del Backend y Base de Datos del Proyecto

11.4.1 Clonación del Backend del proyecto

Para la clonación del Backend del proyecto, es necesario crear una carpeta nueva en la ruta raíz de almacenamiento del dispositivo, entramos a la carpeta damos clic derecho y abrir en el terminal. A continuación, nos dirigimos al repositorio del Backend del proyecto **https://github.com/BrayamMoreno/BackendPQRSD** y le damos al botón verde llamado code, y copiamos el enlace de HTTPS.

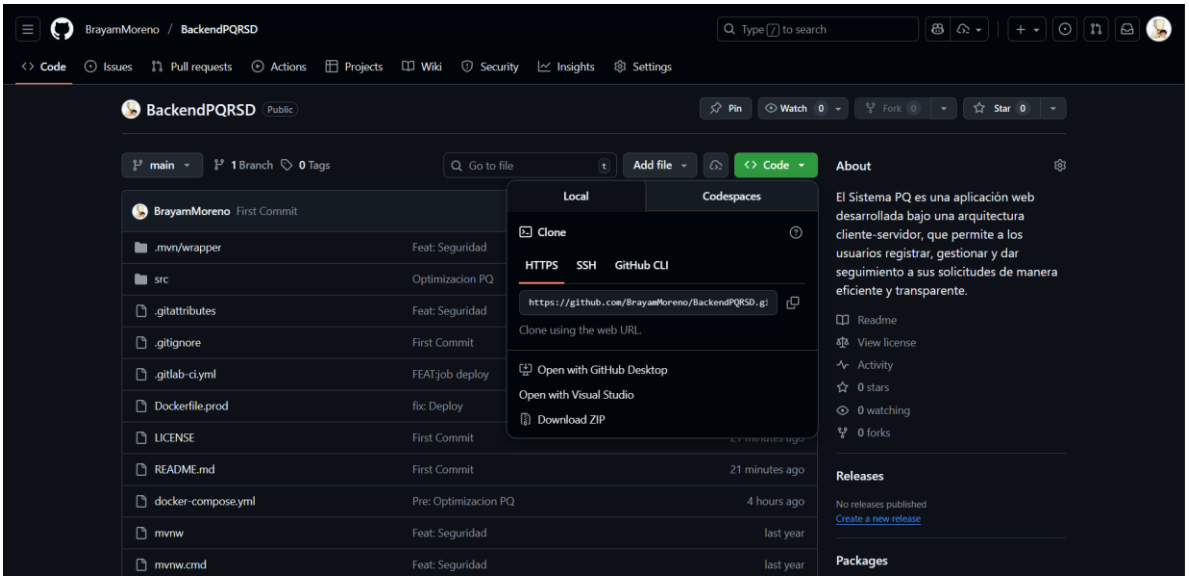


***Figura 16**. Repositorio del Backend GitHub*
***Fuente.** Autoría Propia*

Una vez copiado el enlace ejecutamos el siguiente comando en la terminal abierta anteriormente:

- `git clone` **`https://github.com/BrayamMoreno/BackendPQRSD.git`**

```
PS C:\BackendPQRSD> git clone https://github.com/BrayamMoreno/BackendPQRSD.git
Cloning into 'BackendPQRSD'...
remote: Enumerating objects: 2051, done.
remote: Counting objects: 100% (2051/2051), done.
remote: Compressing objects: 100% (593/593), done.
remote: Total 2051 (delta 1140), reused 2051 (delta 1140), pack-reused 0 (from 0)
Receiving objects: 100% (2051/2051), 3.63 MiB | 371.00 KiB/s, done.
Resolving deltas: 100% (1140/1140), done.
```

***Figura 17.** Clonación del Backend con Git*
***Fuente**. Autoría Propia*

11.4.2 Configuración de docker-compose.yml (Base de Datos)

Figura 18. *Configuración de docker-compose.yml (Base de Datos)*
Fuente. *Autoría Propia*

- **Imagen Base: postgres:15.5-alpine3.18** es PostgreSQL 15 sobre Alpine Linux, una base ligera y eficiente para producción.

- **Nombre del contenedor: db_pq** fija el nombre del contenedor para identificarlo fácilmente.

- **Puertos:** 48325:5432 → El puerto 48325 se usa desde la máquina host para conectarse al contenedor, mientras que 5432 es el puerto interno de PostgreSQL.

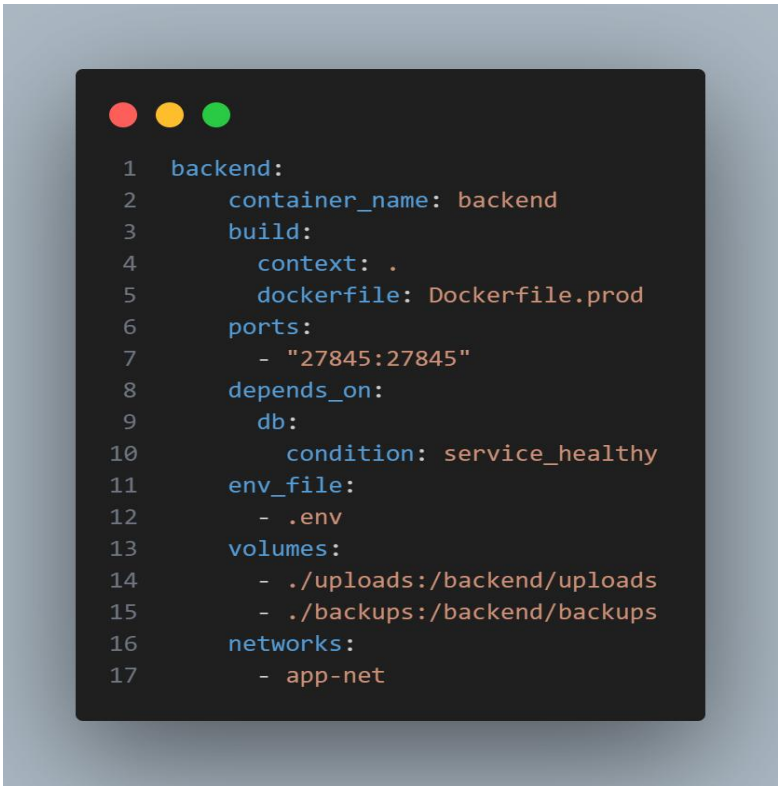
- **Volúmenes:**
  - ./pq.sql:/docker-entrypoint-initdb.d/init.sql monta un script SQL que se ejecuta automáticamente al iniciar el contenedor.
  - ./postgres/data:/var/lib/postgresql/data persiste los datos de la base de datos en la máquina host.

- **Variables de entorno:**

- POSTGRES_USER=pq define el usuario de la base de datos.
- POSTGRES_PASSWORD=pq_2025 define la contraseña del usuario.
- POSTGRES_DB=pq define el nombre de la base de datos.

- **Healthcheck:** Se asegura de que PostgreSQL esté listo antes de iniciar otros contenedores (pg_isready -U pq -d pq).
- **Red: app-net** conecta el contenedor a una red interna de Docker compartida con otros servicios.

11.4.3 Configuración de docker-compose.yml (Backend)



```yaml
backend:
    container_name: backend
    build:
      context: .
      dockerfile: Dockerfile.prod
    ports:
      - "27845:27845"
    depends_on:
      db:
        condition: service_healthy
    env_file:
      - .env
    volumes:
      - ./uploads:/backend/uploads
      - ./backups:/backend/backups
    networks:
      - app-net
```

***Figura 19**. Configuración de docker-compose.yml (Backend)*
***Fuente.** Autoría Propia*

- **Construcción del contenedor:**
  - context: . usa la carpeta actual como contexto de construcción.
  - dockerfile: Dockerfile.prod especifica el Dockerfile a usar.

- **Puertos:** 27845:27845 → Expone el puerto 27845 del contenedor al host, debe coincidir con el puerto del backend y .env.

- **Volúmenes:**
  - ./uploads:/backend/uploads sincroniza los archivos subidos por la aplicación.
  - ./backups:/backend/backups sincroniza los backups generados por la aplicación.

- **Dependencias: depends_on: db** → No inicia hasta que la base de datos esté lista.

- **Archivo de variables de entorno:** .env permite configurar la aplicación sin poner datos sensibles en el Dockerfile.

- **Red**: **app-net** conecta el contenedor a la red interna compartida con la base de datos.

11.4.4 Configuración de docker-compose.yml (Redes)

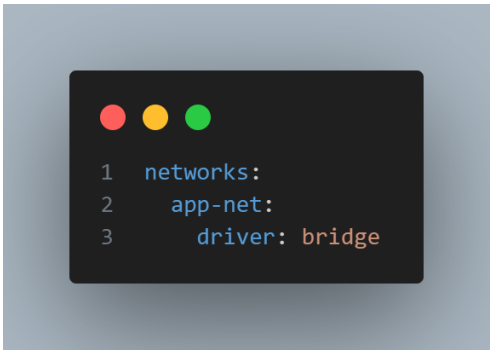
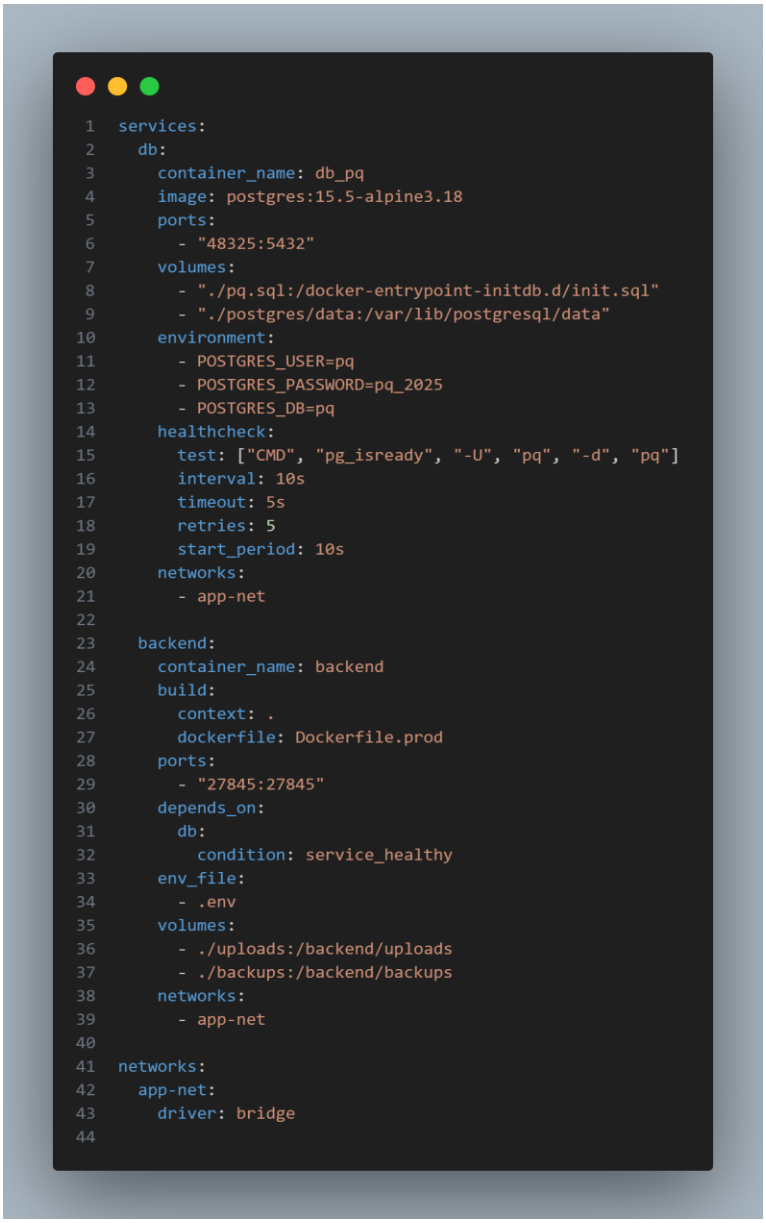


Figura 20. *Configuración de Docker-compose.yml (Redes)*
Fuente. *Autoría Propia*

- **Redes externas: app-net: driver: bridge** crea una red interna aislada donde los contenedores pueden comunicarse entre sí.

11.4.5 Archivo de docker-compose.yml (Backend)



```yaml
services:
  db:
    container_name: db_pq
    image: postgres:15.5-alpine3.18
    ports:
      - "48325:5432"
    volumes:
      - "./pq.sql:/docker-entrypoint-initdb.d/init.sql"
      - "./postgres/data:/var/lib/postgresql/data"
    environment:
      - POSTGRES_USER=pq
      - POSTGRES_PASSWORD=pq_2025
      - POSTGRES_DB=pq
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "pq", "-d", "pq"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 10s
    networks:
      - app-net

  backend:
    container_name: backend
    build:
      context: .
      dockerfile: Dockerfile.prod
    ports:
      - "27845:27845"
    depends_on:
      db:
        condition: service_healthy
    env_file:
      - .env
    volumes:
      - ./uploads:/backend/uploads
      - ./backups:/backend/backups
    networks:
      - app-net

networks:
  app-net:
    driver: bridge
```

Figura 21. *Archivo de docker-file.yml*
Fuente. *Autoría Propia*

11.4.6 Configuración del Archivo de variables (.env)

Una vez clonado el proyecto, localiza el archivo **.env.template** dentro del proyecto. Haz clic derecho sobre él y ábrelo con un editor de texto (por ejemplo, Bloc de notas o VS Code). Este archivo contiene todas las variables de entorno necesarias para configurar la aplicación. A continuación, se describen las principales variables:

Configuración general del servidor

- **SERVER_PORT**: Puerto en el que se ejecuta el backend.
- **FILE_DIR**: Directorio donde se almacenan los archivos generados o subidos por la aplicación.
- **BACKUPS_DIR**: Directorio donde se guardan las copias de seguridad de la base de datos.

Configuración de la base de datos

- **SPRING_DATASOURCE_URL**: URL completa de conexión a la base de datos.
- **SPRING_DATASOURCE_HOST**: Dirección IP o hostname del servidor de base de datos.
- **SPRING_DATASOURCE_PORT**: Puerto donde escucha la base de datos.
- **SPRING_DATASOURCE_USERNAME**: Nombre de usuario para conectarse a la base de datos.
- **SPRING_DATASOURCE_PASSWORD**: Contraseña del usuario de la base de datos.
- **SPRING_DATASOURCE_NAME**: Nombre de la base de datos utilizada por la aplicación.
- **Configuración del Frontend**
- **URL_RESET_PASSWORD**: URL del frontend que se utiliza para el endpoint de restablecimiento de contraseña.

Configuración de Seguridad

JWT_PRIVATE_KEY_GENERATOR: Clave privada utilizada para firmar los JSON Web Tokens (JWT), debe ser una cadena generada de forma aleatorio. Puedes generarla de la siguiente manera:

> **En PowerShell:** `[Convert]::ToBase64String((1..64 | ForEach-Object {Get-Random -Maximum 256}))`

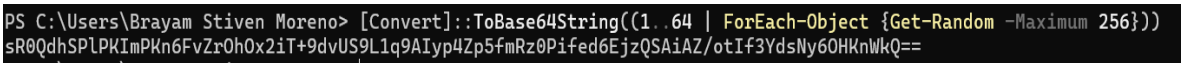


Figura 22. *Generación de Cadena de Forma Aleatoria*
Fuente*: Autoría Propia*

JWT_PRIVATE_USER_GENERATOR: Nombre o identificador del usuario o servicio que genera los tokens JWT

Configuración del Correo Electrónico

- **SMTP_HOST**: Dirección del servidor de correo electrónico.
- **SMTP_PORT**: Puerto del servidor de correo electrónico.
- **SMTP_USERNAME**: Dirección de correo utilizada para enviar emails desde la aplicación.
- **SMTP_PASSWORD**: Contraseña correspondiente al correo electrónico configurado, generalmente generada desde la plataforma del correo.

La generación de contraseña, se hace a través del **Sitio Oficial de Google**, simplemente ingresas el nombre de la aplicación y genera la contraseña de manera automática.

Nota 1: Para usar el archivo .env en el proyecto, copia .env.template y renómbralo a .env, luego completa los valores con los datos de tu entorno local o de producción.

Nota 2: Cuando configures la conexión del Backend a la base de datos dentro del archivo .env, asegúrate de que el host apunte al nombre interno del servicio (db) y no a localhost, y el puerto apunte al interno (5432) no al externo.

11.4.7 Archivo (.env) con Variables de Producción

Figura 23. *Archivo (.env) con Variables de Producción*
Fuente. *Autoría Propia*

11.4.8 Archivo de dockerfile.prod (Backend)

```
# Usa la imagen oficial de OpenJDK 21 en Alpine como base
FROM eclipse-temurin:21-jdk-alpine

# Establece el directorio de trabajo dentro del contenedor
WORKDIR /backend

# Copia solo los archivos necesarios primero para aprovechar el cache de Docker
COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY .env .env

# Descarga las dependencias del proyecto
RUN chmod +x ./mvnw && ./mvnw dependency:go-offline

# Instalar cliente de PostgreSQL (pg_dump, psql, etc.)
RUN apk add --no-cache postgresql15-client

# Copiar el código fuente
COPY src src

# Compila el proyecto sin ejecutar tests
RUN ./mvnw clean install -DskipTests

# Expone el puerto de la aplicación
EXPOSE 27845

# Comando por defecto para ejecutar la aplicación
CMD ["java", "-jar", "target/pqrsdf-0.0.1-SNAPSHOT.jar"]

```

Figura 24. *Archivo de dockerfile.prod (Backend)*
Fuente. *Autoría Propia*

- **Imagen base:** Se usa eclipse-temurin:21-jdk-alpine, que es OpenJDK 21 sobre Alpine Linux, una base ligera y eficiente para producción.

- **Directorio de trabajo:** `WORKDIR /backend` establece /backend como la carpeta donde se ejecutarán todos los comandos dentro del contenedor.

- **Copia de archivos esenciales:** Se copian mvnw, .mvn, pom.xml y .env primero, para aprovechar el cache de Docker y no volver a descargar dependencias si solo cambia el código.

- **Descarga de dependencias Maven:** `RUN chmod +x ./mvnw && ./mvnw dependency:go-offline` hace ejecutable el wrapper de Maven y descarga todas las dependencias offline, acelerando la construcción.

- **Instalación de cliente PostgreSQL:** `RUN apk add --no-cache postgresql15-client` instala herramientas como psql y pg_dump necesarias para interactuar con la base de datos.

- **Copia del código fuente:** `COPY src src` copia toda la carpeta de código fuente dentro del contenedor.

- **Compilación del proyecto:** `RUN ./mvnw clean install -DskipTests` compila el proyecto y genera el JAR sin ejecutar los tests, para ahorrar tiempo.

- **Exposición de puerto:** `EXPOSE 27845` indica que la aplicación escucha en el puerto 27845 dentro del contenedor.

- **Comando por defecto:** `CMD ["java", "-jar", "target/pqrsdf-0.0.1-SNAPSHOT.jar"]` define el comando que se ejecutará al iniciar el contenedor, lanzando la aplicación Java desde el JAR generado.

Nota 1: El puerto expuesto, debe ser el mismo que en el archivo docker-compose.yml y .env

11.4.9 Creación del Contenedor a través de la Terminal

Una vez configurados todos los archivos necesarios, abrimos una terminal dentro del directorio del proyecto y ejecutamos el siguiente comando: `docker-compose up --build -d`, este comando compila las imágenes y levanta los contenedores en segundo plano. (Es necesario ejecutar la aplicación de Docker desktop). Una vez ejecutado, debería obtener el siguiente resultado:

Figura 25. *Resultado de Ejecución del Comando*
Fuente. *Autoría Propia*

11.5 Creación del Frontend

11.5.1 Clonación del Frontend del Proyecto

Para la clonación del Frontend del proyecto, es necesario crear una carpeta nueva en la ruta raíz de almacenamiento del dispositivo, entramos a la carpeta damos clic derecho y abrir en el terminal. A continuación, nos dirigimos al repositorio del Backend del proyecto **https://github.com/BrayamMoreno/FrontendPQRSD** y le damos al botón verde llamado code, y copiamos el enlace de HTTPS.

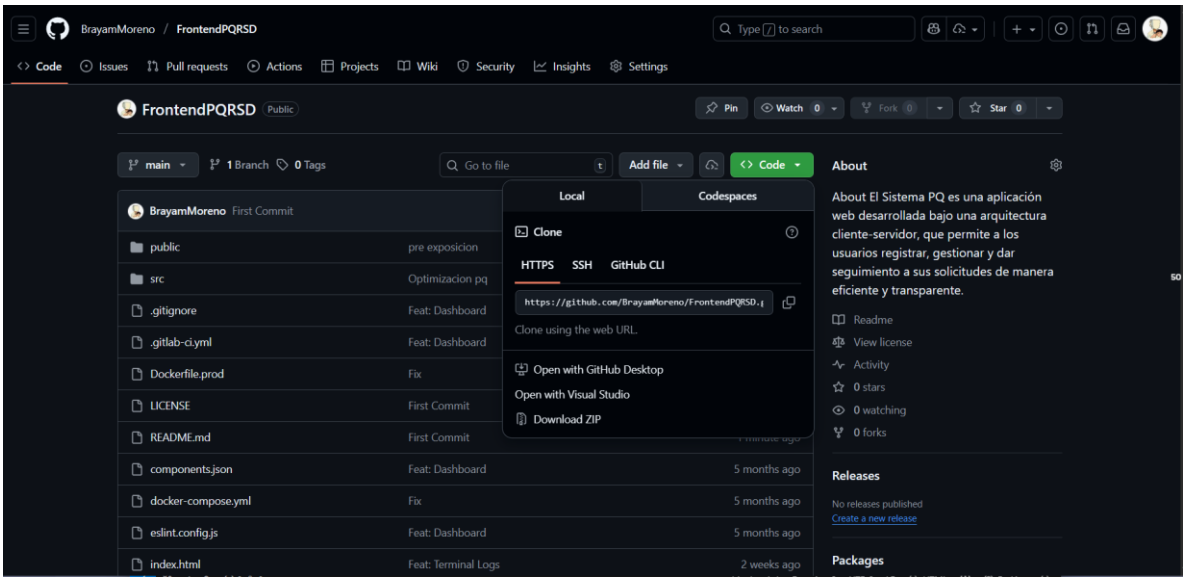


Figura 26. *Repositorio del Frontend en GitHub*
Fuente. *Autoría Propia*

Una vez copiado el enlace ejecutamos el siguiente comando en la terminal abierta anteriormente:

- `git clone https://github.com/BrayamMoreno/FrontendPQRSD.git`

```
PS C:\FrontendPQRSD> git clone https://github.com/BrayamMoreno/FrontendPQRSD.git
Cloning into 'FrontendPQRSD'...
remote: Enumerating objects: 1050, done.
remote: Counting objects: 100% (1050/1050), done.
remote: Compressing objects: 100% (340/340), done.
remote: Total 1050 (delta 671), reused 1050 (delta 671), pack-reused 0 (from 0)
Receiving objects: 100% (1050/1050), 1.74 MiB | 2.55 MiB/s, done.
Resolving deltas: 100% (671/671), done.
```

***Figura 27**. Clonación del Frontend con Git*
***Fuente.** Autoría Propia*

11.5.2 Archivo de dockerfile.prod (Frontend)

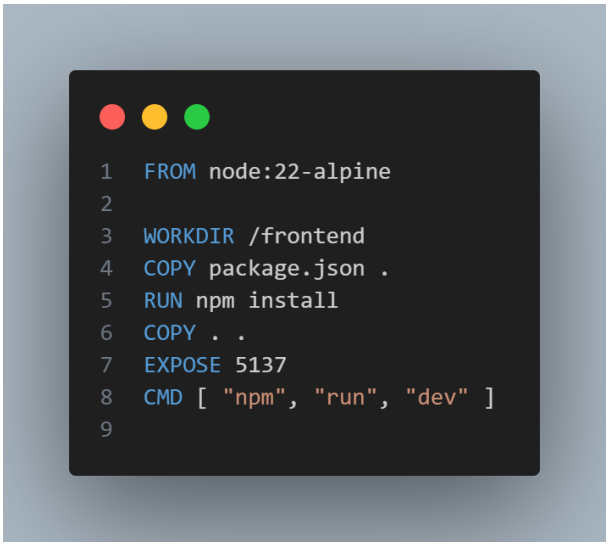


***Figura 28.** Archivo de dockerfile.prod (Frontend)*
***Fuente:** Autoría Propia*

- **Imagen base:** Se usa node:22-alpine, que es la versión 22 de Node.js sobre Alpine Linux, una distribución ligera, rápida y optimizada para producción.

  Proporciona un entorno eficiente para ejecutar aplicaciones basadas en JavaScript o frameworks como Angular, React o Vue.

- **Directorio de trabajo:** `WORKDIR /frontend` establece /frontend como la carpeta de trabajo dentro del contenedor, donde se ejecutarán todos los comandos posteriores.

Esto garantiza una estructura limpia y organizada para los archivos del proyecto.

- **Copia de dependencias principales:** `COPY package.json .` copia únicamente el archivo package.json dentro del contenedor.

  Este paso se realiza antes de copiar el resto del código para aprovechar la caché de Docker, de manera que, si el código fuente cambia, pero las dependencias no, no sea necesario volver a instalarlas.

- **Instalación de dependencias:** `RUN npm install` instala todas las dependencias listadas en el archivo package.json.

  Esto prepara el entorno del contenedor con todas las librerías necesarias para compilar y ejecutar el proyecto Frontend.

- **Copia del código fuente:** `COPY . .` copia todo el contenido del proyecto local dentro del contenedor, en la carpeta /frontend.

  Incluye el código fuente, configuraciones y recursos necesarios para ejecutar la aplicación.

- **Exposición de puerto:** `EXPOSE 5137` indica que la aplicación dentro del contenedor escuchará en el puerto 5137.
  Este puerto se vincula con el puerto externo configurado en el archivo docker-compose.yml (por ejemplo, 25000:5137), permitiendo acceder a la aplicación desde el navegador.

- **Comando por defecto:** `CMD ["npm", "run", "dev"]` define el comando que se ejecutará automáticamente al iniciar el contenedor.

  Ejecuta el script dev definido en el package.json, que normalmente inicia el servidor de desarrollo o despliegue del Frontend.

Nota 1: El puerto expuesto (5137) debe coincidir con el puerto interno configurado en el archivo docker-compose.yml y con el definido en la configuración del proyecto Frontend.

11.5.3 Archivo de docker-compose.yml (Frontend)

```yaml
services:
  frontend:
    container_name: frontend
    build:
      context: .
      dockerfile: Dockerfile.prod
    ports:
      - 25000:5137
    networks:
      - app-net

networks:
  app-net:
    driver: bridge
```

Figura 29. *Archivo de docker-compose.yml (Frontend)*
Fuente. *Autoría Propia*

El siguiente bloque define el servicio correspondiente al Frontend dentro del archivo docker-compose.yml. Este servicio se encarga de construir, ejecutar y conectar la aplicación web al entorno de red compartido con el resto de los servicios del proyecto.

- **Servicio del Frontend:** Crea y ejecuta un contenedor encargado de levantar la aplicación del Frontend, a partir de la configuración establecida en el archivo Dockerfile.prod.

- **Nombre del contenedor:** `container_name: frontend` asigna el nombre fijo “frontend” al contenedor. Esto facilita su identificación dentro del entorno Docker y permite interactuar con él directamente mediante comandos como docker logs frontend o docker exec -it frontend sh.

- **Construcción del contenedor:** `context: .` indica que la construcción del contenedor se realizará usando la carpeta actual como contexto, lo que incluye todos los archivos necesarios del proyecto Frontend.

- **Dockerfile:** Dockerfile.prod especifica el archivo Dockerfile.prod como la guía de construcción del contenedor. Este archivo contiene todas las instrucciones necesarias para instalar dependencias, compilar el código y ejecutar la aplicación.

- **Puertos:** `25000:5137` define el mapeo de puertos entre la máquina anfitriona y el contenedor. El puerto 25000 corresponde al puerto externo de la máquina anfitriona, desde donde se accede al navegador. El puerto 5137 corresponde al puerto interno del contenedor donde corre el servidor del Frontend. Esto permite acceder a la aplicación desde un navegador web mediante la dirección **http://localhost:25000**.

- **Red del contenedor:** `networks: app-net` conecta el servicio del Frontend a la red interna app-net, compartida con otros servicios como el Backend o la base de datos. Esto permite la comunicación directa entre contenedores sin exponer sus puertos internos al exterior.

- **Configuración de red general:** Bajo la sección networks, se define la red interna app-net con el driver bridge. Esta configuración crea una red aislada dentro de Docker, permitiendo que los contenedores conectados a ella se comuniquen entre sí utilizando sus nombres de servicio (por ejemplo, backend, db o frontend).

Nota 1: El puerto expuesto en el archivo docker-compose.yml (25000:5137) debe coincidir con el puerto definido en el archivo Dockerfile.prod (EXPOSE 5137), para asegurar que la aplicación sea accesible correctamente desde el navegador.

Nota 2: Si el servicio Frontend necesita comunicarse con el Backend, debe hacerlo utilizando el nombre del contenedor (backend) y el puerto interno correspondiente, en lugar de usar localhost.

11.5.4 Archivo Config.ts

Figura 30. *Archivo Config.ts (Frontend)*
Fuente. *Autoria Propia*

Por ejemplo, si actualmente el archivo de configuración del Frontend (config.js) contiene la línea:

- `const config = { apiBaseUrl: "http://108.181.154.240:27845/api" };`

Durante la ejecución en contenedores Docker, esta dirección debe reemplazarse por:

- `const config = { apiBaseUrl: "http://backend:27845/api" };`

De esta forma, la aplicación Frontend se comunicará correctamente con el servicio del Backend a través de la red interna app-net, garantizando la conexión entre contenedores sin depender de direcciones externas o locales.

11.5.5 Archivo Vite.config.ts

Figura 31. *Archivo Vite.config.ts (Frontend)*
Fuente. *Autoría Propia*

Este archivo define los parámetros de configuración para el empaquetado y ejecución del proyecto React utilizando Vite, una herramienta moderna y eficiente diseñada para entornos de producción. A continuación, se detalla cada sección:

Servidor de despliegue:

El bloque server especifica la configuración del servidor que servirá la aplicación dentro del contenedor Docker:

- `port: 5137` define el puerto interno en el que el servicio del Frontend escuchará dentro del contenedor. Este valor debe coincidir con el definido en el archivo `Dockerfile (EXPOSE 5137)` y en el `docker-compose.yml (25000:5137).`

- `allowedHosts: ["dominio.ejemplo"]` permite definir el dominio público desde el cual se accederá a la aplicación una vez desplegada (por ejemplo, miapp.com o frontend.miempresa.com). Esto asegura que la aplicación esté

preparada para recibir solicitudes desde ese dominio específico al ser publicada en un servidor o VPS.

11.5.6 Creación del Contenedor a través de la Terminal

Una vez configurados todos los archivos necesarios, abrimos una terminal dentro del directorio del proyecto y ejecutamos el siguiente comando: `docker-compose up --build -d`, este comando compila las imágenes y levanta los contenedores en segundo plano. (Es necesario ejecutar la aplicación de Docker desktop). Una vez ejecutado, debería obtener el siguiente resultado:

Figura 32. *Resultado de Ejecución del Comando*
Fuente. *Autoría Propia*

11.6 Comprobación del Funcionamiento de cada Servicio

Una vez compilados y levantados los contenedores de los servicios del proyecto, es necesario comprobar que tanto los contenedores como las aplicaciones que ejecutan en su interior estén funcionando correctamente.

A continuación, se detallan los pasos recomendados para realizar esta verificación:

11.6.1 Verificación de Contenedores Activos

Para verificar que los contenedores están activos se debe ejecutar el siguiente comando `docker ps`. Este comando mostrará un listado de todos los contenedores actualmente en ejecución, incluyendo información como el nombre, la imagen, el estado y los puertos asignados.

```
PS C:\FrontendPQRSD\FrontendPQRSD> docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS                    PORTS                                          NAMES
5154bbdb4806   frontendpqrsd-frontend  "docker-entrypoint.s…"   36 minutes ago   Up 36 minutes             0.0.0.0:25000->5137/tcp, [::]:25000->5137/tcp  frontend
14323330716f   backendpqrsd-backend    "/__cacert_entrypoin…"   About an hour ago  Up About an hour        0.0.0.0:27845->27845/tcp, [::]:27845->27845/tcp backend
789f3aec106c   postgres:15.5-alpine3.18 "docker-entrypoint.s…"  About an hour ago  Up About an hour (healthy) 0.0.0.0:48325->5432/tcp, [::]:48325->5432/tcp db_pq
```

Figura 33. *Lista de Contenedores Activos*
Fuente. *Autoría Propia*

11.6.2 Revisión de Logs de cada Contenedor

Una vez comprobado que los contenedores están activos, es necesario revisar los registros (*logs*) de cada uno de ellos para confirmar que las aplicaciones se hayan iniciado correctamente y que no existan errores durante el arranque, esto lo hacemos con el siguiente comando `docker logs <nombre contenedor> -f.`

Base de datos

Para la comprobación de la base de datos, ejecutamos el comando `docker logs db_pq -f.` Este comando permite visualizar en tiempo real los registros (*logs*) generados por el contenedor db_pq, mostrando el proceso de inicialización y posibles mensajes de error o confirmación de que PostgreSQL se encuentra en ejecución.

El resultado debería ser similar al mostrado en la siguiente figura:

```
PS C:\FrontendPQRSD\FrontendPQRSD> docker logs db_pq -f

PostgreSQL Database directory appears to contain a database; Skipping initialization

2025-11-01 16:46:34.543 UTC [1] LOG:  starting PostgreSQL 15.5 on x86_64-pc-linux-musl, compiled by gcc (Alpine 12.2.1_git20220924-r10) 12.2.1 20220924, 64-bit
2025-11-01 16:46:34.543 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2025-11-01 16:46:34.543 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2025-11-01 16:46:34.559 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2025-11-01 16:46:34.643 UTC [24] LOG:  database system was shut down at 2025-11-01 16:46:24 UTC
2025-11-01 16:46:34.775 UTC [1] LOG:  database system is ready to accept connections
```

Figura 34*. Fragmento del Resultado del Comando docker logs db_pq -f*
Fuente. *Autoría Propia*

Una vez obtenidos los registros, se puede comprobar que la base de datos se ha iniciado correctamente y que se encuentra lista para aceptar conexiones.

Backend

Para la comprobación del backend, ejecutamos el comando `docker logs backend -f`. Este comando permite visualizar en tiempo real los registros (*logs*) generados por el contenedor backend, mostrando el proceso de inicialización y posibles mensajes de error o confirmación de que Springboot se encuentra en ejecución.

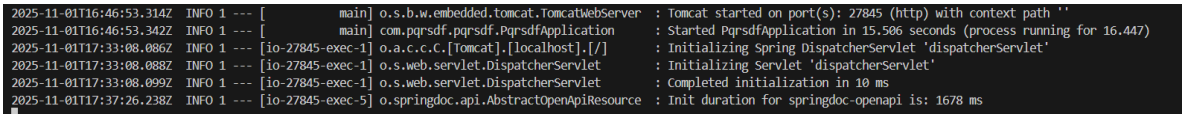


Figura 35. *Fragmento del Resultado del Comando docker logs backend -f*
Fuente. *Autoría Propia*

Una vez obtenidos los registros, se puede comprobar que el backend se ha iniciado correctamente y se esta ejecutando en el puerto 27845

Frontend

Para la comprobación del Frontend, ejecutamos el comando `docker logs frontend -f`. Este comando permite visualizar en tiempo real los registros (*logs*) generados por el contenedor frontend, mostrando el proceso de inicialización y posibles mensajes de error o confirmación de que vite se encuentra en ejecución.

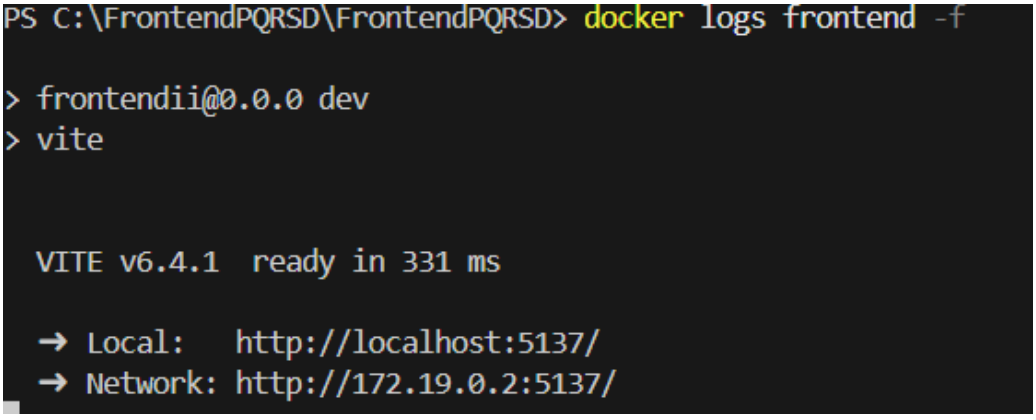


Figura 36. *Fragmento del Resultado del Comando docker logs frontend -f*
Fuente. *Autoría Propia*

11.6.3 Acceso por Consola a la Base de Datos

Ahora si queremos acceder a la base de datos mediante una terminal, debemos ejecutar los siguientes comandos:

- `docker exec -it db_pq bash:` Comando para acceder al contenedor y observar toda la información del servicio
- `psql -U pq:` comando para acceder a la interfaz de comandos de PostgreSQL

12. RECOMENDACIONES

Una vez finalizado el proceso de despliegue del sistema PQRSD en Docker Desktop para Windows, se recomienda realizar las siguientes comprobaciones y acciones para asegurar el correcto funcionamiento de la aplicación.

12.1 Inicialización de la Base de Datos

El sistema utiliza una base de datos PostgreSQL que se genera automáticamente a partir del archivo pq.sql, incluido en el proyecto. Este archivo contiene la estructura de tablas y datos iniciales necesarios para el funcionamiento del sistema PQRSDF.

En caso de que la base de datos no se cree durante el primer arranque, puede inicializarse manualmente ejecutando desde PowerShell el siguiente comando:

- `docker exec -i db_pq psql -U pq -d pq < pq.sql`

Con esto se asegura que la base de datos esté correctamente cargada dentro del contenedor db_pq.

12.2 Acceso a la Aplicación

Después de confirmar que todos los servicios están activos, se puede acceder a las interfaces del sistema desde el navegador web:

- **Backend (API y documentación Swagger):**http://localhost:27845/swagger-ui/index.html
- **Frontend (interfaz de usuario):**http://localhost:25000

Por defecto, el sistema crea un administrador inicial con las siguientes credenciales:

- **Correo:** admin@admin.com
- **Contraseña:** 123456

Este usuario debe emplearse únicamente para el primer acceso, con el fin de crear un nuevo usuario administrador con credenciales seguras y luego eliminar la cuenta inicial.

12.3 Confirmación del correcto funcionamiento

Al ingresar al sistema, se debe verificar que las principales funcionalidades, registro de solicitudes PQRSDF, consulta de casos y gestión de respuesta. Operen correctamente.
Si las interfaces responden adecuadamente y los datos se almacenan en la base de datos, el despliegue se habrá completado de forma exitosa.