PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

Unidad Académica de Formación Técnica y Tecnológica – PUCE TEC

DESARROLLO DE UNA APLICACIÓN WEB PARA EL REGISTRO Y CONTROL DE VISITANTES EN EL INSTITUTO DE INVESTIGACIÓN GEOLÓGICO ENERGÉTICO

Proyecto de titulación previo a la obtención del título de: tecnología Superior En Desarrollo De Software

Autor: José Francisco Muzo Tipuano

Autor: Dilan Jordan Segovia Posligua

Tutor: Ing. Christian Roberto Tapia Gaibor

Quito, Ecuador

2025

Tabla de Contenidos

Lista de tablas

Lista de Ilustraciones

DECLARACIÓN y AUTORIZACIÓN

Yo, **José Francisco Muzo Tipuano** con C.I. 1717706210 autor(a) del trabajo de "**Desarrollo de una Aplicación Web para el Registro y Control de Visitantes en el Instituto de Investigación Geológico Energético IIGE**", previa a la obtención del título de **Tecnología Superior En Desarrollo De Software** en la Unidad Académica de Formación Técnica y Tecnológica PUCE TEC:

1.- Declaro tener pleno conocimiento de la obligación que tiene la Pontificia Universidad Católica del Ecuador, de conformidad con el artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de graduación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la Pontificia Universidad Católica del Ecuador a difundir a través de sitio web de la Biblioteca de la PUCE el referido trabajo de titulación, respetando las políticas de propiedad intelectual de Universidad.

Quito, 15 de agosto 2025

José Francisco Muzo Tipuano

C.I. 1717706210

DECLARACIÓN y AUTORIZACIÓN

Yo, **Dilan Jordan Segovia Posligua** con C.I. 1753980042 autor(a) del trabajo de "**Desarrollo de una Aplicación Web para el Registro y Control de Visitantes en el Instituto de Investigación Geológico Energético IIGE**" previa a la obtención del título de **Tecnología Superior En Desarrollo De Software** en la Unidad Académica de Formación Técnica y Tecnológica PUCE TEC:

1.- Declaro tener pleno conocimiento de la obligación que tiene la Pontificia Universidad Católica del Ecuador, de conformidad con el artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de graduación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la Pontificia Universidad Católica del Ecuador a difundir a través de sitio web de la Biblioteca de la PUCE el referido trabajo de titulación, respetando las políticas de propiedad intelectual de Universidad.

Quito, 15 de agosto 2025

Dilan Jordan Segovia Posligua

C.I. 1753980042

Agradecimientos

Este proyecto a más de ser un proyecto que engloba los estudios por el cual se esfuerza uno todos los días es un proyecto de vida el cual mejorara nuestra aptitud ya sea de manera profesional o personal esto va dedicado tanto a mi familia como a mis amigos los cuales hemos compartido vivencias en cada día de asistencia, también como parte especial a mis padres, por su apoyo incansable, por creer en mí y por darme la fuerza para seguir adelante en todo momento. Su capacidad de un consejo hace que nuestro camino se vaya por el camino del bien.

Introducción

El Instituto de Investigación Geológico Energético (IIGE) enfrenta un problema significativo en la gestión del registro de visitantes debido a la utilización de métodos tradicionales, como registros en papel o sistemas manuales. Estos métodos generan ineficiencia, falta de trazabilidad y vulnerabilidades en la seguridad de los datos. En un entorno donde el control de acceso es esencial para la protección de información sensible, la ausencia de un sistema limita la capacidad del IIGE para supervisar en tiempo real el flujo de personas en sus instalaciones.

Este proyecto tiene como objetivo la creación de una aplicación web que proporcione optimizar el registro mediante un control registrado y validado para un, mejorando la eficiencia operativa y garantizando un monitoreo preciso. La solución se alinea con la tendencia global hacia la digitalización de procesos administrativos y la implementación de tecnologías de la información para la mejora de la gestión institucional.

El desarrollo de esta aplicación web no solo moderniza la administración del IIGE, sino que también minimiza errores humanos, reduce los tiempos de espera en el ingreso de visitantes y refuerza la seguridad mediante un registro confiable.

Problema Técnico/Tecnológico

El Instituto de Investigación Geológico Energético (IIGE) enfrenta actualmente deficiencias en la gestión del acceso de visitantes, derivadas del uso de métodos manuales y registros físicos. La documentación en formato físico está expuesta a extravíos, deterioro y errores en la recopilación de datos, lo que compromete la confiabilidad y disponibilidad de la información. Estos inconvenientes afectan el entorno de la institución para llevar un control preciso de las personas que ingresan a sus instalaciones. El proceso manual de verificación de identidad y autorización genera retrasos en el acceso, ocasionando demoras innecesarias y afectando la operatividad diaria de la entidad. La ausencia de un sistema de registro automatizado dificulta la trazabilidad de los visitantes, lo que impide la consulta ágil de historiales y la generación de reportes confiables. Además, la falta de digitalización en el control de accesos aumenta el riesgo de seguridad, ya que un sistema ineficiente puede facilitar el ingreso no autorizado de personas, comprometiendo la integridad de las instalaciones y la confidencialidad de la información manejada por la institución.

Marco Conceptual

El proyecto para el registro y control de visitantes en el Instituto de Investigación Geológico Energético (IIGE) se basa en la creación de una aplicación web que integra dos componentes principales: un frontend (cliente) y un backend (servidor), interconectados para garantizar una gestión digital eficiente, segura y escalable.

El backend constituye el núcleo funcional del proyecto, ya que gestiona la lógica de negocio, el almacenamiento seguro de datos y procesa todas las solicitudes provenientes del cliente. Mientras tanto, el frontend proporciona una interfaz amigable y responsiva que facilita la interacción del usuario con el sistema.

Tecnologías Empleadas Backend

Node.js.

Es el entorno de ejecución para JavaScript en el servidor, elegido para construir el backend. Permite usar el mismo lenguaje tanto en el cliente (React) como en el servidor, facilitando el mantenimiento y la coherencia del proyecto. Su arquitectura asincrónica y basada en eventos permite manejar múltiples solicitudes simultáneamente sin bloquear procesos, lo que es crucial para aplicaciones como el control de accesos en el IIGE.

Express.js.

Framework para Node.js que facilita la creación de aplicaciones web y APIs RESTful. Permite definir rutas HTTP, gestionar peticiones (GET, POST, PUT, DELETE) y aplicar middleware que estructuran y organizan la lógica del servidor. En este proyecto, Express maneja rutas para el registro de visitantes, login de usuarios y consultas a la base de datos.

Sequelize.

Es el ORM (Object-Relational Mapping) utilizado para interactuar con la base de datos relacional PostgreSQL. Permite definir modelos en JavaScript que representan tablas, manejar relaciones entre datos y realizar consultas avanzadas sin escribir SQL directamente. Esto hace que el backend sea más legible, seguro y fácil de mantener.

PostgreSQL

Sistema gestor de bases de datos relacional robusto y de código abierto, encargado de almacenar los registros de visitantes, usuarios y logs. Sus características ACID garantizan integridad y seguridad en las transacciones, fundamentales para mantener la confiabilidad del sistema.

Dotenv

Herramienta para manejar variables de entorno de forma segura. Permite almacenar credenciales, puertos y configuraciones críticas en archivos. env, evitando exponer información sensible en el código fuente. Así se facilita el despliegue en distintos entornos (desarrollo, pruebas o producción).

CORS

Cross-Origin Resource Sharing es un mecanismo que permite al frontend, que puede estar alojado en un dominio distinto, acceder a la API del backend sin violar políticas de seguridad del navegador. Garantiza que se puedan comunicar entre el cliente React y el servidor Express sea segura y controlada.

body-parser

Middleware para Express que analiza el cuerpo de las solicitudes entrantes y lo convierte en objetos accesibles desde req.body. Esto es esencial para procesar formularios y datos JSON enviados desde el frontend

Tecnologías Empleadas para el Frontend

El frontend se construyó con React, es una biblioteca de JavaScript para creación de interfaces de usuario dinámicas y reactivas. Gracias a React, se pudo estructurar el cliente en componentes reutilizables, cada uno encargado de una funcionalidad específica como el formulario de registro de visitantes, tablas filtrables o la barra de navegación.

Tailwind CSS se emplea para el diseño visual, proporcionando un framework utilitario que permite crear interfaces modernas y responsivas de forma eficiente. Esto facilita que el sistema sea accesible desde distintos dispositivos (PC, tabletas, móviles).

El cliente se comunica con el backend mediante llamadas HTTP (fetch o axios) para enviar datos de formularios, solicitar listas de visitantes, marcar salidas o generar reportes, garantizando una experiencia fluida sin recargas totales (SPA).

Integración backend + frontend

El backend expone una API RESTful que el frontend consume. Cada acción del usuario en la interfaz —como iniciar sesión, registrar un visitante o filtrar registros— dispara peticiones HTTP que el servidor procesa, consultando o actualizando la base de datos PostgreSQL según sea necesario. Gracias a CORS y a la estructura modular del proyecto, ambos componentes pueden mantenerse, desplegarse y escalarse de forma independiente, pero trabajando de forma totalmente integrada.

Capítulo I: Levantamiento de Requisitos y Diseño del Sistema

1.1 Antecedentes

El Instituto de Investigación Geológico Energético (IIGE) enfrenta retos importantes en el registro y control de sus visitantes debido a su dependencia de procesos manuales, como libros físicos y registros en papel. Esta situación genera errores frecuentes, pérdida de datos y ausencia de trazabilidad, lo que compromete la seguridad y dificulta el análisis de estadísticas relacionadas con las visitas. Para modernizar y fortalecer sus procesos, el IIGE impulsó el desarrollo de una aplicación web con frontend en React y Tailwind, y backend en Node.js, Express y PostgreSQL. que permita automatizar el registro, control y seguimiento de sus visitantes.

1.2 Metodología Scrum

En el proyecto se trabajó con la metodología Scrum adaptada a la dinámica del equipo. Se asignaron los siguientes roles: el Ing. Christian Roberto Tapia Gaibor como Scrum Master, encargado de coordinar reuniones y facilitar la resolución de inconvenientes; el Ing. Paul Jaramillo como Product Owner, responsable de priorizar y aprobar entregas; Dilan Jordan Segovia Posligua como desarrollador backend; y Jose Francisco Muzo Tipuano como desarrollador frontend.

Durante el desarrollo, se utilizó un product backlog con todas las funcionalidades solicitadas por el cliente, y para cada sprint se creó un sprint backlog con las tareas específicas. Cada ciclo concluyó con un incremento funcional del sistema. Se estableció un Definition of Done (DoD) que incluía la revisión del código, la validación de pruebas y la aprobación final del Product Owner.

Las actividades realizadas incluyeron reuniones de planificación al inicio de cada sprint, reuniones diarias para seguimiento, revisiones de producto al cierre y sesiones de retrospectiva para analizar aciertos y mejoras. Los sprints tuvieron una duración fija de dos semanas y al final de cada uno se entregó una versión funcional al cliente para su validación.

Ventajas específicas:

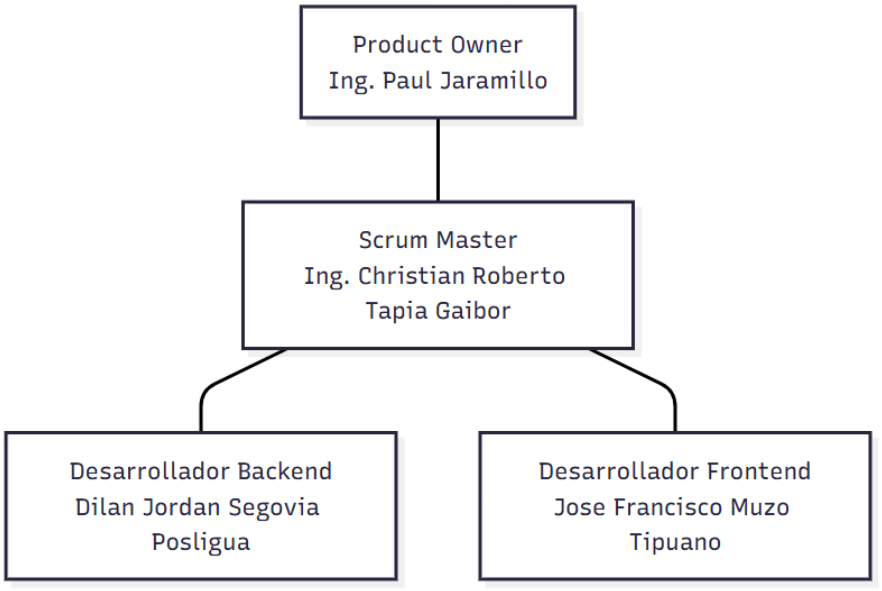
- Permite iterar con entregables funcionales.
- Identifica rápidamente errores o requisitos cambiantes.
- Fomenta la comunicación continua mediante reuniones diarias y retrospectivas.
- Maximiza el valor entregado en cada Sprint.

1.2.3 Actores principales

De acuerdo con la metodología ágil Scrum, los actores principales del proyecto se definieron de esta manera:

Figura 1

Figura 1 Organigrama de roles del equipo Scrum del proyecto



Nota .El diagrama muestra la estructura y comunicación del equipo Scrum al rpoyecto Owner , Scrum Master, desarrollador backend y desarrollador frontend.

- **Scrum Master :** Ing. Christian Roberto Tapia Gaibor. Responsable de facilitar el proceso Scrum, coordinar reuniones, eliminar impedimentos y supervisar el desarrollo general del proyecto como tutor de tesis.
- **Product Owner:** Ing. Paul Jaramillo. Encargado de priorizar las funcionalidades, validar entregas y asegurar que el producto final cumpla con las necesidades del IIGE.
- **Desarrollador Backend:** Dilan Jordan Segovia Posligua. Encargado de implementar la lógica de negocio y la conexión con la base de datos.

- **Desarrollador Frontend:** Jose Francisco Muzo Tipuano. Encargado de desarrollar la interfaz de usuario y la interacción con el backend.

1.3 Etapas del proceso

Se realizaron reuniones virtuales con el funcionario, en este caso el Product Owner (Ing. Paúl Jaramillo, Dpto. Sistemas), en las cuales se abordaron el objetivo principal y las funcionalidades del sistema.

Además, se llevaron a cabo visitas semanales para el análisis de procesos existentes con el guardia de seguridad y la recepcionista, donde se trató el tema del registro de información, verificando si, además del control físico a los visitantes, se realizaba un registro digital en el Instituto.

También se tomaron fotografías del registro manual que se estaba llevando a cabo, con el fin de documentar el proceso actual y contar con evidencia visual para el análisis funcional.

1.3.1 flujo de historias de usuario

A continuación, se detalla las historias de usuario identificadas durante el levantamiento de requerimientos requisitos, así también el flujo dentro del sistema.

HU01: Como personal de seguridad, quiero registrar a un visitante con sus datos personales y motivo de visita, para tener un control claro del acceso.

HU02: Como personal de seguridad, quiero marcar la salida del visitante, para mantener actualizado el estado en el sistema.

HU03: Como administrador, quiero gestionar los usuarios del sistema, para asegurar que solo personal autorizado acceda.

HU04: Como administrador, quiero visualizar un reporte diario de visitantes, para analizar el flujo de personas.

HU05: Como personal de seguridad, quiero buscar registros por nombre, cédula o fecha, para agilizar las consultas.

HU06: Como visitante recurrente, quiero que mis datos se autocompleten al ser identificado, para agilizar el proceso.

HU07: Como administrador de la aplicación quiero editar o eliminar usuarios, para corregir usuarios innecesarios si el personal es removido del puesto de trabajado.

HU08: Como administrador, quiero exportar los registros en archivo con formato PDF, para compartirlos con otras áreas del Instituto.

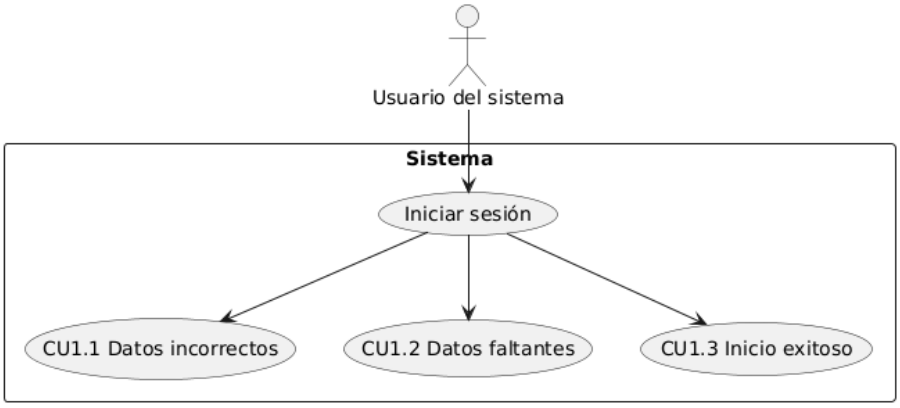
1.3.2 Casos de uso

Los casos de uso descritos a continuación fueron obtenidos a partir de las historias de usuario definidas en el product backlog y priorizadas por el Product Owner durante las reuniones de planificación (Sprint Planning). Cada caso fue validado y refinado por el equipo Scrum, con la supervisión del Scrum Master.

Caso se Usó CU1. Inicio sesión

Figura 2

Figura 2 Inicio de Sesión



Caso de Uso expandido

Tabla 1

Tabla 1 Casos de Usos Datos incorrectos

Campo	Detalle
Nombre del caso de uso	CU1.1 – Datos incorrectos
Actor principal	Usuario del sistema
Descripción	El sistema detecta que los datos de correo o contraseña son incorrectos y muestra un mensaje de error.
Precondiciones	El usuario ha ingresado datos en los campos de inicio de sesión.
Postcondiciones	El acceso es denegado y el usuario permanece en la página de login.
Requisitos no funcionales	RNF01: Seguridad en la transmisión RNF02: Tiempo de respuesta menor a 3s

Tabla 2

Tabla 2 Casos de Uso Datos Faltantes

Campo	Detalle
Nombre del caso de uso	CU1.2 – Datos faltantes
Actor principal	Usuario del sistema
Descripción	El sistema valida que los campos de correo y contraseña no estén vacíos y muestra un mensaje si falta alguno.
Precondiciones	El usuario intenta iniciar sesión sin completar todos los campos.
Postcondiciones	No se realiza la autenticación y el usuario recibe indicación de llenar los campos.
Requisitos no funcionales	RNF01: Seguridad en la transmisión RNF02: Tiempo de respuesta menor a 3s

Tabla 3

Tabla 3 Casos de Uso Inicio Exitoso

Campo	Detalle
Nombre del caso de uso	CU1.3 – Inicio exitoso
Actor principal	Usuario del sistema
Descripción	El sistema valida que los datos ingresados son correctos y permite el acceso al usuario al sistema.
Precondiciones	El usuario ha ingresado correo y contraseña válidos.

Postcondiciones	El usuario accede a la plataforma y puede utilizar las funcionalidades disponibles según su rol.
Requisitos no funcionales	RNF01: Seguridad en la transmisión RNF02: Tiempo de respuesta menor a 3s

Caso de Uso CU2: Registrar usuario

Figura 3

Figura 3 Diagrama de Registro de usuario

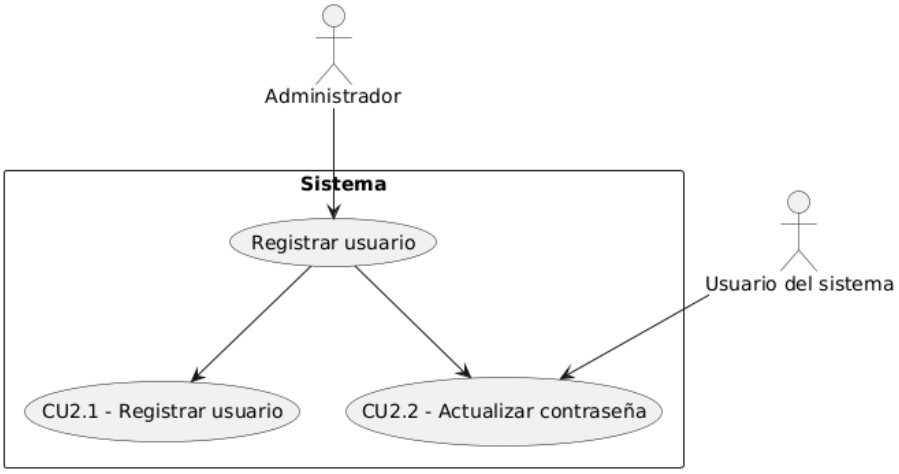


Tabla 4

Tabla 4 Caso de Uso Registrar Usuarios

Campo	**Detalle**
Nombre del caso de uso	CU2.1 – Registrar usuario
Actor principal	Administrador
Descripción	El sistema permite registrar un nuevo usuario con los datos requeridos.

Precondiciones	El administrador debe estar autenticado en el sistema.
Postcondiciones	El nuevo usuario queda registrado y puede iniciar sesión.
Requisitos no funcionales	RNF01: Arquitectura modular RNF02: API RESTful RNF03: Seguridad en transmisión

Tabla 5

Tabla 5 Caso de Uso Actualizar contraseña

Campo	**Detalle**
Nombre del caso de uso	CU2.2 – Actualizar contraseña
Actor principal	Usuario del sistema
Descripción	El usuario puede cambiar su contraseña proporcionando la actual y la nueva.
Precondiciones	El usuario debe estar autenticado y conocer su contraseña actual.
Postcondiciones	La contraseña se actualiza y queda vigente para futuros accesos.
Requisitos no funcionales	RNF01: Seguridad en la transmisión RNF02: Tiempo de respuesta RNF03: Integridad de datos

Caso de Uso CU2: Registrar visitante

Figura 4

Figura 4 Registro de Visitante

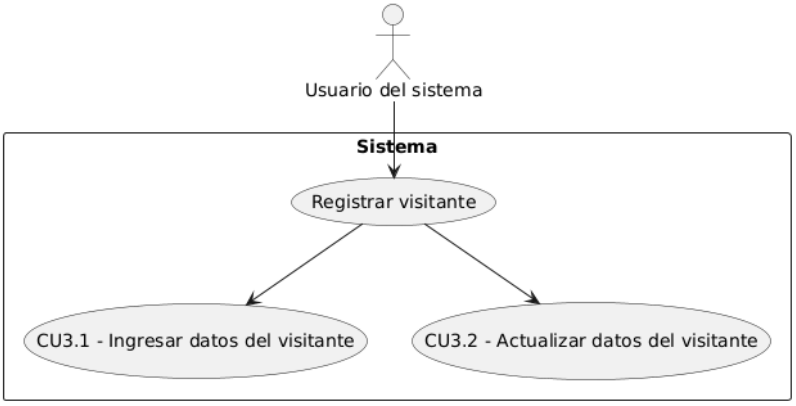


Tabla 6

Tabla 6 Caso de Uso Ingresar datos del visitante

Campo	**Detalle**
Nombre del caso de uso	CU3.1 – Ingresar datos del visitante
Actor principal	Usuario del sistema (seguridad o recepcionista)
Descripción	El usuario registra a un visitante ingresando nombre, identificación.
Precondiciones	El visitante debe estar presente en la entrada.
Postcondiciones	La visita queda registrada con fecha y hora de entrada.
Requisitos no funcionales	RNF01: Arquitectura modular RNF02: API RESTful RNF03: PostgreSQL + Sequelize

Tabla 7

Tabla 7 Caso de Uso Actualizar datos del visitante

Campo	Detalle
Nombre del caso de uso	CU3.2 – Actualizar datos del visitante
Actor principal	Usuario del sistema (seguridad o recepcionista)
Descripción	El usuario puede editar los datos de la visita siempre que esta no haya finalizado.
Precondiciones	La visita debe estar registrada y no haber sido finalizada.
Postcondiciones	Los datos de la visita son modificados correctamente.
Requisitos no funcionales	RNF01: Arquitectura modular RNF02: API RESTful RNF03: Validación de campos

1.3.2 Creación del backlog

El backlog del producto se construyó priorizando funcionalidades críticas para el IIGE:

Tabla 8

Tabla 8 Prioridades del sistema de control

ID	Funcionalidad	Prioridad
RF01	Autenticación de usuarios	Alta
RF02	Registro de ingreso de visitantes	Alta
RF03	Registro de salida de visitantes	Alta
RF04	Búsqueda avanzada	Media
RF05	Reportes y exportación	Media

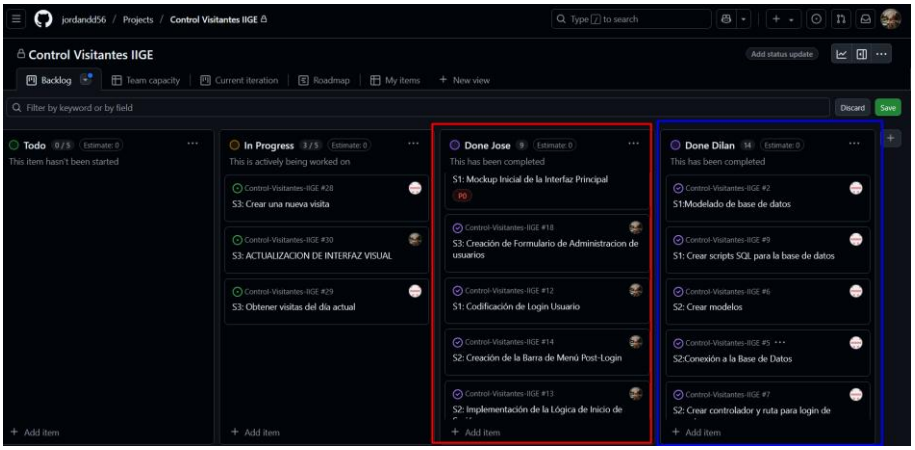
RF06	Gestión de usuarios	Media
RF07	Notificaciones y alertas	Baja
RF08	Registro de logs	Baja

1.3.3 Planificación de los Sprint

Para estructurar la construcción del sistema, se adaptó la metodología Scrum dividiendo el trabajo en sprints numerados como S1, S2, S3, S4, S5, tanto para backend como para el frontend lo que permitió organizar visual y conceptualmente cada fase del desarrollo.

Figura 5

Figura 5 Flujo de trabajo Frontend y Backend



Sprints 1

Tabla 9

Tabla 9 Desarrollo de los Sprint Backend

S1: Modelado de base de datos	**Creación de entidades y atributos de base de datos**	**1 hora**	**https://github.com/jordandd56/Control-Visitantes-IIGE/issues/2**

S1: Crear scripts SQL para la base de datos	Desarrollar y preparar los scripts SQL necesarios para la inicialización y poblamiento de la base de datos del sistema de Control de Visitantes IIGE.	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/9

Sprint 2

Tabla 10

Tabla 10 Desarrollo de los Sprint Backend 2

S2: Crear modelos	Modelos a crear: áreas, roles, usuarios, visitantes visitas	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/6
S2: Conexión a la Base de Datos	Tareas a realizar: Crear archivo de configuración config/db.js Crear Variable de entorno env. Crear archivo models/index.js para que : Inicie Sequelize, Importe los modelos y Sincronice las relaciones entre ellos.	4 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/5

S2: Crear controlador y ruta para login de usuarios	Implementar la funcionalidad de inicio de sesión que permita autenticar a los usuarios mediante su nombre de usuario y contraseña. Se debe verificar que las credenciales ingresadas coincidan con un usuario existente en la base de datos.	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/7
S2: Crear la funcionalidad para agregar usuarios al sistema	Implementar la funcionalidad para crear usuarios en el sistema mediante una petición HTTP. Esta funcionalidad permitirá registrar usuarios proporcionando su nombre completo, nombre de usuario, contraseña y rol asociado. Se debe verificar que el usuario no exista previamente y devolver una respuesta adecuada.	4 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/10
S2: Obtener listado de usuarios	Implementar la funcionalidad que permita recuperar todos los usuarios registrados en la base de datos. Esta funcionalidad será	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/15

	útil para la administración del sistema y la visualización general de usuarios.		
S2: Obtener listado de usuarios por nombre completo	Implementar una funcionalidad que permita recuperar usuarios de la base de datos filtrando por el campo nombre_completo. Esto será útil para búsquedas rápidas y filtrado dinámico por parte del personal administrativo.	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/16

Sprint 3

Tabla 11

Tabla 11 Desarrollo de los Sprint Backend 3

S3: Cambiar validación de campos cedula y pasaporte en Visitante	Motivo del cambio Anteriormente, el campo cedula era obligatorio. Sin embargo, se detectó que no todos los visitantes contaban con cédula, y algunos solo tenían pasaporte. Por ello:	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/22

	Se hizo cedula y pasaportes opcionales. Se aseguró que al menos uno esté presente mediante una restricción CHECK. Ambos campos se definieron como únicos (UNIQUE) para evitar duplicados.		
S3: Crear funcionalidad para agregar visitante	Implementar la funcionalidad que permita registrar nuevos visitantes en el sistema. El visitante puede ser identificado por cédula o pasaporte, pero debe tener al menos uno de los dos junto con su nombre completo. Esta funcionalidad será consumida desde el frontend mediante una petición POST.	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/23
S3: Obtener listado de áreas	Implementar el endpoint que permita obtener el listado de todas las áreas registradas en la base de datos del sistema de control de visitantes del IIGE.	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/26

S3: Crear una nueva área	Desarrollar el endpoint que permita registrar una nueva área dentro del sistema, asociada a la tabla áreas de la base de datos.	1 hora	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/27
S3: Crear una nueva visita	Implementar el endpoint que permita registrar una nueva visita en el sistema, almacenando todos los datos necesarios asociados a la tabla visitas.	1 hora	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/28
S3: Obtener visitas del día actual	Crear un endpoint que permita obtener todas las visitas registradas en el día actual, filtrando por la fecha del sistema.	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/29
S3: Actualización de la base de datos – Tablas visitantes y visitas	Actualizar la estructura de las tablas existentes en la base de datos para reflejar nueva información relevante para la gestión de visitantes.	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/32
S3: Visualización de visitas	Implementar funcionalidades para obtener:	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/34

activas y visitas del día	Listado de visitas activas (con estado "ingreso" y sin hora de salida registrada). Listado de todas las visitas registradas en el día actual.		
S3: Implementar cifrado de contraseñas con bcryp	Implementar la funcionalidad para que las contraseñas de los usuarios sean cifradas automáticamente antes de ser almacenadas en la base de datos. Esto mejora la seguridad del sistema al evitar guardar contraseñas en texto plano.	4 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/31
S3: Actualización de contraseña con cifrado bcrypt	Implementar la funcionalidad para que los usuarios puedan actualizar su contraseña, asegurando que la nueva contraseña sea almacenada de forma segura mediante cifrado con bcrypt.	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/35

Sprint 1

Tabla 12

Tabla 12 Desarrollo de los Sprint Frontend 1

S1: Mockup Inicial de la Interfaz Principal	Diseño visual inicial Crear un mockup de la pantalla de inicio, respetando la identidad visual del proyecto	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/11
S1: Diseño de Login, menú principal, formularios.	Maquetado de pantalla de inicio, formularios, diseño elección de colores.	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/39
S1: Codificación de Login Usuario	Maquetación del formulario Crear los campos necesarios (usuario y contraseña) con etiquetas y placeholders adecuados.	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/12

Sprint 2

Tabla 13

Tabla 13 Desarrollo de los Sprint Frontend 2

S2: Creación de la Barra de Menú Post-Login	Diseño y estructura del menú Definir las secciones y opciones del menú principal roles, reportes.	4 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/14

	Establecer una jerarquía clara si hay submenús o menús desplegables.		
S2: Implementación de la Lógica de Inicio de Sesión	Integración de lógica de login Implementar el consumo de la API de autenticación utilizando Fetch. Gestión de sesión Capturar y almacenar de sesión de manera local	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/13
S2: Creación de Formulario de Ingreso	Diseño y estructura del formulario Definir los campos necesarios: usuario, contraseña, validaciones. Establecer reglas de diseño y accesibilidad: tamaños, alineaciones, colores, mensajes de error claros. Incorporar elementos visuales como iconos del menú	4 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/17

Sprint 3

Tabla 14 *Tabla 14 Desarrollo de los Sprint Frontend 3*

S3: Creación de Formulario de Administración de usuarios	Desarrollar un módulo de administración de usuarios con un menú estructurado que permita gestionar roles, visualizar reportes y configurar accesos. La implementación debe ser intuitiva, responsiva y funcional dentro del sistema.	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/18
S3: Creación de rutas de usuario y administrador	Implementación de rutas diferenciadas en el frontend para permitir el ingreso de usuarios normales y administradores.	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/19
S3: Configuración de enrutamiento y peticiones	Carga inicial del componente AdminUser, obtenemos los datos de roles y usuarios usando fetch y mostrarlos dinámicamente en select y tabla, respectivamente.	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/21
S3: Configuración	Conexión el formulario de administración de usuarios con la API backend para enviar	4 horas	https://github.com/jordandd56/Co

para el Guardado en la base de datos	y guardar los datos ingresados en la base de datos mediante peticiones POST.		ntrol-Visitantes-IIGE/issues/20
S3: Actualización de interfaz de Frontend diseño, vista, formularios, menú	Reunión con el producto owner el cual aprueba el nuevo diseño de plantillas	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/37
S3: Actualización de interfaz de Frontend Lógica al nuevo diseño	Implementación de lógica en el nuevo diseño y formularios para el procesado de la información recibida	4 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/36
S3: Validación funcional del nuevo Frontend	Diseño del frontend, este Issue cubre la fase de revisión y prueba funcional, asegurando que los nuevos componentes respondan correctamente a la lógica del sistema y al consumo de datos desde el backend.	3 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/38

S3: Implementación de responsive en el diseño del Frontend	Adaptación completa del diseño del frontend para que sea totalmente responsive, asegurando una experiencia consistente desde computadoras de escritorio y teléfonos móviles.	2 horas	https://github.com/jordandd56/Control-Visitantes-IIGE/issues/40

1.3.4 Herramientas y software

Git & GitHub: Control de versiones con ramas feature, develop y main.

Postman: Para pruebas de las API REST.

Visual Studio Code & PgAdmin: Desarrollo y administración de la base de datos.

PowerDesigner: Modelado y diseño de la base de datos a nivel lógico y físico.

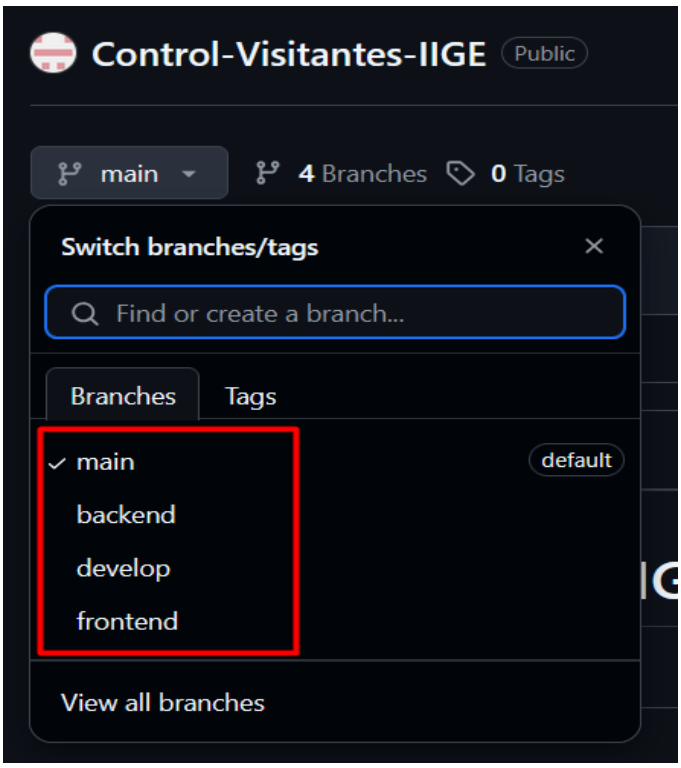
Azure: Plataforma en la nube utilizada para el despliegue de la solución:

- **Azure App Service:** Hospedaje del backend Node.js (mi-backend-nodejs).
- **Azure Static Web Apps:** Despliegue del frontend React (FrondTest).
- **Azure Database for PostgreSQL Flexible Server:** Base de datos (postgres-tesis).
- **Azure Resource Groups:** Organización de recursos

Flujo de trabajo con Git

Figura 6

Figura 6 Ramificación de github



Rama principal: main

- Contiene la versión estable y lista para producción del proyecto.
- Solo se fusiona (merge) desde la rama develop cuando se completa un ciclo de desarrollo.
- Protegida: no se debe trabajar directamente sobre ella.

Rama de desarrollo: develop

- Es la rama base donde se integran los avances del equipo antes de subir a producción.
- Sirve como base para nuevas funcionalidades en el frontend y backend.
- Recibe aportes desde las ramas frontend y backend.

Ramas funcionales

- frontend: Rama destinada al desarrollo de la interfaz de usuario.
- backend: Rama destinada a la lógica de negocio, controladores y base de datos.

Procedimiento de validación antes de pasar a producción

1. Cada nueva funcionalidad o corrección se desarrolla en una rama funcional (frontend o backend).
2. Una vez finalizada, se realiza *commit* con mensajes descriptivos y claros.
3. Se abre un Pull Request hacia develop para revisión de código (*code review*) y pruebas unitarias.
4. En la rama develop se realizan:
    - Pruebas locales con **Postman** (endpoints de la API).
    - Verificación en **Azure**.
    - Comprobación de conexión con la base de datos PostgreSQL en Azure.
5. Si todas las validaciones son correctas, se realiza el *merge* de develop a main.

1.3.5 Hardware

Requisitos mínimos del sistema

Tabla 15 *Tabla 15 Herramientas físicas para la implementación*

Cliente Web	**Servidor Backend**
Procesador i5 o Ryzen 5	Procesador i5 o superior
4GB RAM	8GB RAM
120 GB libres	SSD recomendado
Navegador actualizado	PostgreSQL y Node.js

1.3.6 Viabilidad del proyecto

La viabilidad técnica del proyecto es alta debido a la utilización de tecnologías modernas y probadas, tales como React, Node.js y PostgreSQL, complementadas con servicios en la nube de Microsoft Azure, incluyendo App Service, Static Web Apps y Azure Database for PostgreSQL. Estas herramientas permiten garantizar la escalabilidad, disponibilidad y

mantenimiento del sistema. Asimismo, se incorporaron herramientas de apoyo como Visual Studio Code, PgAdmin y PowerDesigner, que facilitaron el modelado, desarrollo y gestión de la base de datos.

En cuanto a la viabilidad económica, el uso de tecnologías de código abierto evitó la adquisición de licencias, reduciendo considerablemente la inversión inicial. Adicionalmente, la suscripción Azure for Students permitió el uso gratuito de recursos en la nube durante la etapa de desarrollo y pruebas. Esto, junto con la ejecución del proyecto por parte del equipo interno del Instituto Geográfico, minimizó costos relacionados con la contratación de personal externo.

Respecto a la viabilidad operativa, el personal del instituto participó de forma activa en las fases de pruebas, validación de funcionalidades y retroalimentación. Este grado de involucramiento garantizó que el sistema se adaptara a los procesos internos y que los usuarios se encontraran capacitados para operarlo. Finalmente, la arquitectura modular del software facilita futuras actualizaciones y mejoras, lo que asegura la continuidad y sostenibilidad del sistema a largo plazo.

Capítulo II: Construcción del Sistema

2.1 Alcance

El sistema de control de visitantes abarca el desarrollo completo de una aplicación web destinada al registro y control de visitantes del Instituto de Investigación Geológico Energético (IIGE). Incluye:

Frontend: Implementado con React y Tailwind CSS, encargado de proporcionar interfaces amigables, responsivas y adaptadas a diferentes dispositivos, permitiendo el registro, consulta y generación de reportes de visitantes.

Backend: Desarrollado en Node.js con Express, responsable de manejar la lógica de negocio, exponer APIs RESTful y gestionar la persistencia de datos en PostgreSQL.

Base de datos: Diseño e implementación en PostgreSQL, almacenando la información de usuarios, visitas y logs para auditoría.

Este alcance se limita a los procesos relacionados con la gestión de visitantes y usuarios, sin contemplar la integración con otros sistemas internos del IIGE, aunque preparado para futuras ampliaciones.

2.2 Análisis de diseño

Se derivan de los requisitos desde las historias de usuario.

A partir de las historias de usuario definidas en el levantamiento de requisitos, se concretaron los siguientes elementos:

Tabla 16 *Tabla 16 Historias de usuario*

CODIGO	DETALLE
HU01:	Como administrador, gestionar usuarios.
HU02:	Como personal de seguridad, registrar visitantes.
HU03:	Marcar salidas para actualizar estado de la visita.
HU04:	Buscar visitas filtrando por nombre, cédula o fecha.
HU05:	Generar reportes exportables.

2.3 Mapa de navegación

El sistema se organiza mediante una navegación estructurada por roles y permisos:

Pantalla de Login: Autenticación de usuarios autorizados (recepcionista o guardia)

Dashboard: Vista principal que presenta accesos rápidos a los distintos módulos.

Módulos específicos:

- Inicio pantalla de bienvenida
- Creación de Usuarios con permisos autorizados
- Generar reportes exportables

2.4 Lógica de negocio.

Figura 7

Figura 7 Arquitectura del Proyecto.

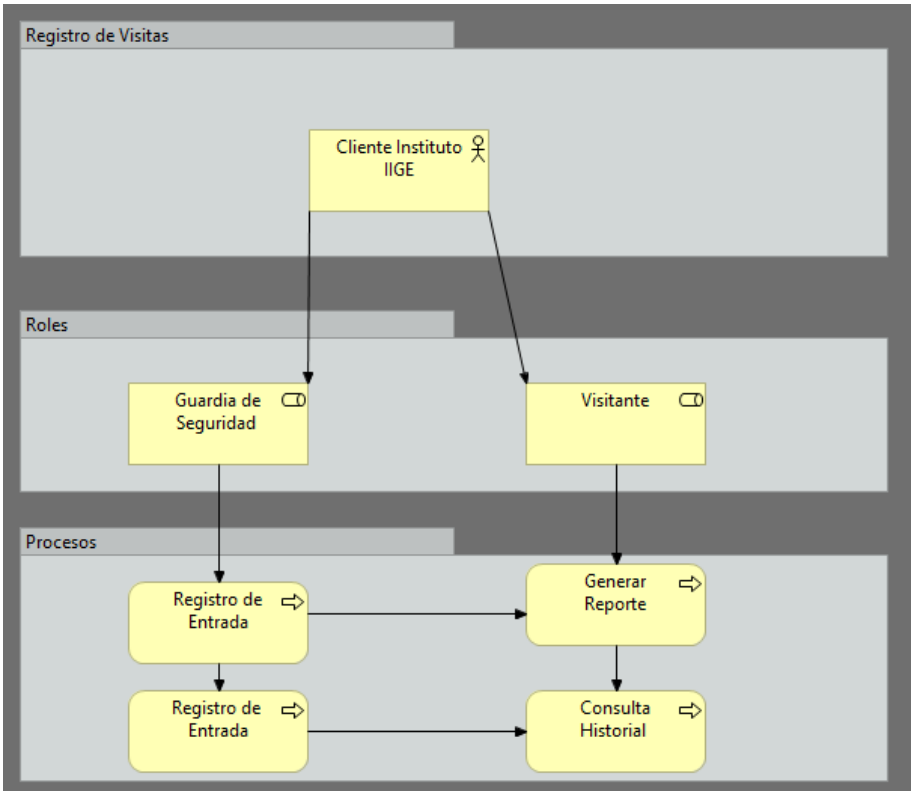
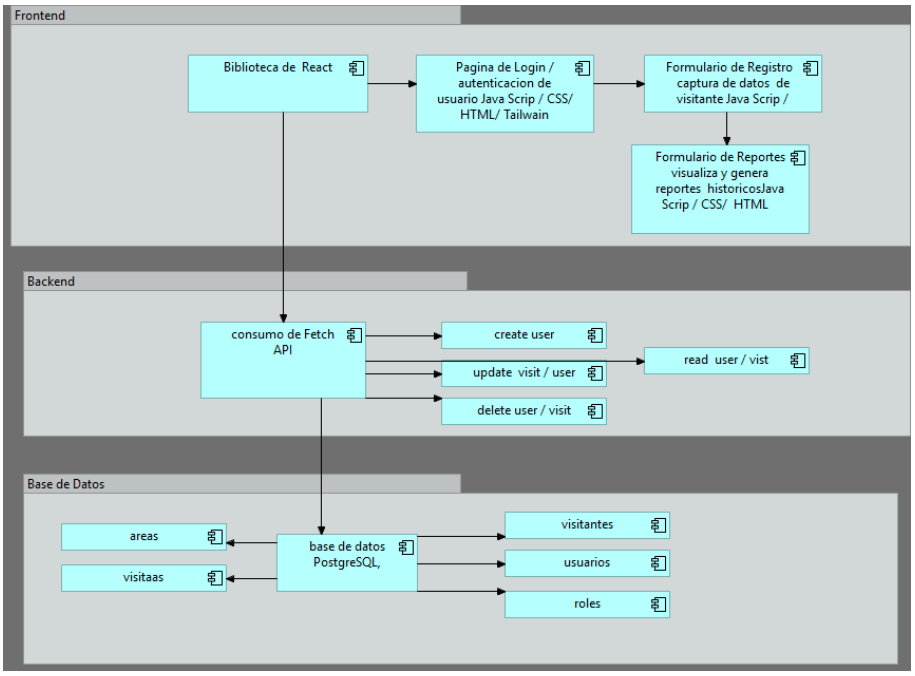


Figura 8

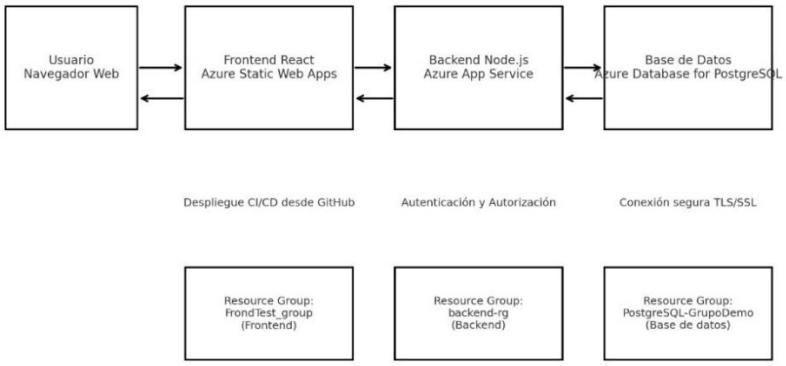
Figura 8 Arquitectura de Hardware



2.4 Diseño de la arquitectura de la aplicación

Figura 9

Figura 9 Arquitectura de la aplicación



La presente solución adopta una arquitectura en tres capas implementada sobre la plataforma Microsoft Azure. La capa de presentación está compuesta por una aplicación web desarrollada con React, publicada en el servicio *Azure Static Web Apps*, desde la cual los usuarios interactúan mediante su navegador y envían solicitudes a través de protocolos HTTP o HTTPS. La capa de negocio corresponde a una API de tipo REST desarrollada en Node.js y desplegada en *Azure App Service*, que concentra la lógica del sistema, administra las reglas operativas y expone los servicios necesarios para los distintos módulos. Por último, la capa de datos se gestiona mediante *Azure Database for PostgreSQL – Flexible Server*, donde se almacenan de manera estructurada las entidades principales, tales como visitantes, usuarios, áreas, roles y visitas. La interconexión entre capas se realiza a través de canales seguros y autenticados, lo que asegura la separación de responsabilidades y favorece la escalabilidad del sistema.

El procesamiento de la información inicia cuando el usuario interactúa con la aplicación del frontend. Esta envía peticiones hacia la API hospedada en *App Service*, la cual procesa los datos, consulta o actualiza la base en *PostgreSQL* y devuelve una respuesta al cliente en un

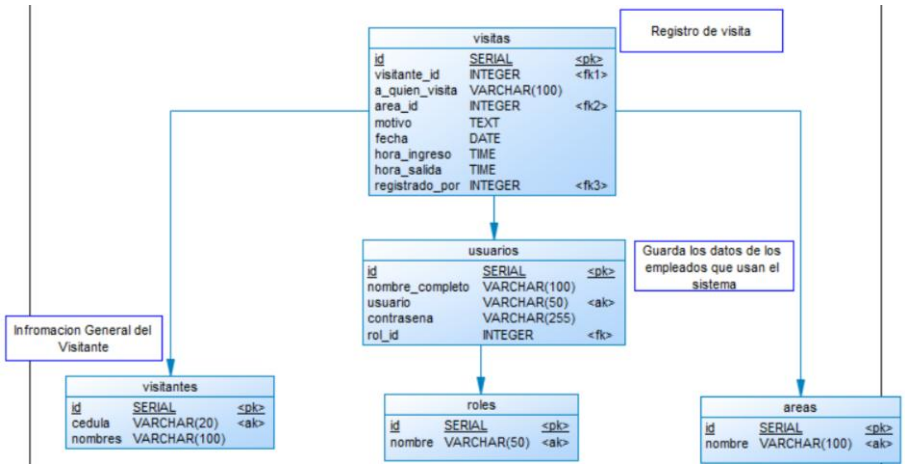
formato estructurado. La gestión de accesos se realiza mediante autenticación y asignación de permisos por rol, garantizando que las operaciones críticas —como el registro de ingresos, la finalización de visitas o la emisión de reportes— solo estén disponibles para usuarios autorizados. El despliegue del software está automatizado a través de un flujo de integración continua (CI/CD) conectado con GitHub; cada cambio aprobado en las ramas definidas genera compilaciones controladas y su correspondiente publicación tanto en *Static Web Apps* (interfaz) como en *App Service* (API). La infraestructura está organizada en grupos de recursos independientes —*FrondTest_group*, *backend-rg* y *PostgreSQL-GrupoDemo*— lo que facilita la administración, el control de costos y la supervisión del rendimiento.

Esta configuración ofrece alta disponibilidad y tolerancia a fallos, ya que *App Service* admite escalado automático y supervisión de estado (*health probes*), mientras que *PostgreSQL Flexible Server* proporciona copias de seguridad periódicas y la posibilidad de restaurar la información a un punto anterior. La segmentación por capas posibilita el mantenimiento evolutivo: el frontend puede modernizarse sin interrumpir la API, y los cambios en la lógica o en el modelo de datos pueden implementarse sin afectar la capa de presentación. Finalmente, el uso de tecnologías de código abierto, junto con servicios administrados de Azure, optimiza la relación entre costos, seguridad, monitoreo y escalabilidad, asegurando la viabilidad y sostenibilidad del sistema a largo plazo.

2.5 Modelo de Base de Datos (BDD)

Figura 10

Figura 10 Diagrama de la base de datos



El diseño del esquema de la base de datos en PostgreSQL contempla las siguientes tablas principales:

usuarios: almacena datos de autenticación y roles (administrador o seguridad).

visitantes: registra información de cada visita, incluyendo datos personales, motivo, área, fecha/hora de ingreso y salida, y estado de la visita en este caso se utilizó pendiente y finalizado.

Este modelo facilita futuras extensiones, como añadir módulos de notificaciones, integración con sistemas de control de acceso físico o reportes avanzados.

En el diagrama se muestran 5 tablas que se relacionan de la siguiente manera:

1. Relación entre visitas y visitantes

- Tipo: Muchos a uno (N:1)
- Clave foránea: visitante_id
- Descripción: Cada visita está vinculada a un visitante específico, mientras que un mismo visitante puede tener múltiples registros de visita.

2. Relación entre visitas y áreas

- Tipo: Muchos a uno (N:1)
- Clave foránea: area_id
- Descripción: Cada visita corresponde a un área determinada, pero un área puede estar asociada a varias visitas.

3. Relación entre visitas y usuarios

- Tipo: Muchos a uno (N:1)
- Clave foránea: registrado_por
- Descripción: Indica el usuario que registró la visita. Un mismo usuario puede registrar numerosas visitas.

4. Relación entre usuarios y roles

- Tipo: Muchos a uno (N:1)
- Clave foránea: rol_id
- Descripción: Cada usuario posee un rol específico, mientras que un rol puede asignarse a varios usuarios.

2.6 Requisitos Funcionales

Tabla 17 *Tabla 17 Requisitos funcionales del Backend*

CODIGO	DETALLE

RF-B01	Exponer API para autenticar usuarios según rol (administrador / seguridad).
RF-B02	Registrar visitas con datos como nombre, motivo, área, fecha/hora
RF-B03	Marcar salida de visitantes actualizando su estado y hora de salida.
RF-B04	Endpoint para búsqueda de visitas por nombre, cédula o fecha.
RF-B05	Generar reportes y enviarlos en formato exportable (PDF/CSV).
RF-B06	Gestionar creación y modificación de usuarios.
RF-B07	Registrar logs de operaciones críticas en la base de datos.

Tabla 18 *Tabla 18 Requisitos no funcionales del Backend.*

CODIGO	DETALLE
RNF-B01	Seguridad en APIs mediante validación.
RNF-B02	Arquitectura escalable para soporte de múltiples usuarios concurrentes.
RNF-B03	Persistencia confiable en PostgreSQL
RNF-B04	Registro automático de para auditoría y trazabilidad.
RNF-B05	Disponibilidad de endpoints bien documentado

Tabla 19 *Tabla 19 Requisitos funcionales del Frontend*

CODIGO	DETALLE
RF-F01	Interfaz de login con validación de credenciales.
RF-F02	Formulario responsivo para registrar visitantes.
RF-F03	Botón y flujo para marcar salida de visitante.

RF-F04	Tabla con listado de visitas para poder filtrar
RF-F05	Pantalla para generación y descarga de reportes.
RF-F06	Módulo visual para gestión de usuarios según permisos.
RF-F07	Navegación estructurada según roles (seguridad / administrador).

Tabla 20 *Tabla 20 Requisitos no funcionales del Frontend*

CODIGO	**DETALLE**
RNF-F01	Interfaz responsiva adaptada a distintos dispositivos.
RNF-F02	Carga rápida de componentes para una experiencia fluida.
RNF-F03	Validaciones en los formularios para evitar errores de entrada.
RNF-F04	Accesibilidad para usuarios con distintos roles Administrador y usuario estándar
RNF-F05	Uso coherente de estilos visuales con Tailwind CSS para mantener estética institucional.

Capítulo III: Pruebas y Estabilización

3.1 Reuniones con Product Owner

Es una pieza importa el Product Owner ya que es el puente que existe entre el equipo de desarrollo y el equipo técnico por parte del cliente por otro lado se enfoca se asegura que se contruya realmente y este cumpla con las necesidades del usuario y este acorde con los objetivos de control de la organización.

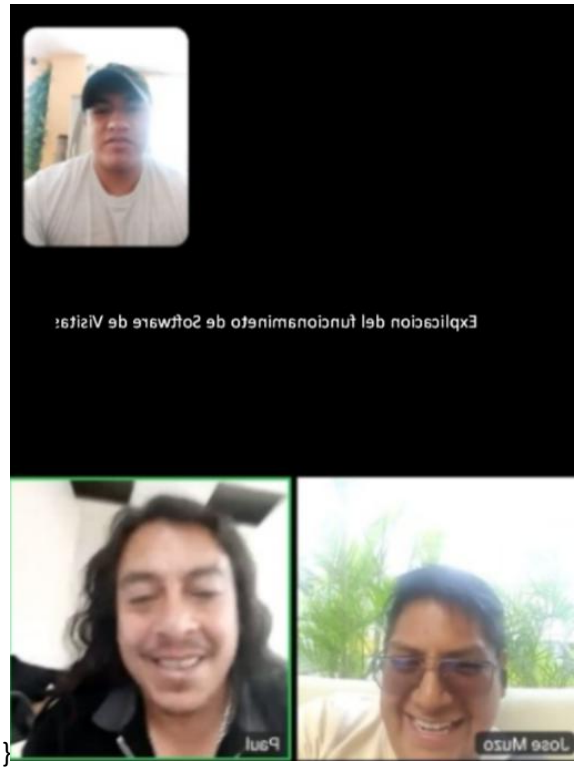
En este desarrollo tenemos la distribución de la siguiente manera:

- Product Owner -> Ing. Paul Jaramillo (Representante del IIGE)
- Equipo de desarrollo Backend (Dilan Segovia) Frontend (José Muzo)

Se realizaron reuniones virtuales y de manera presencial en el Instituto (IIGE)

Figura 11

Figura 11 Reunión Presentación del proyecto de Control de asistencia



En el transcurso del desarrollo web se presentan observaciones diseño, infraestructura, manejo del programa, títulos, recursos demás, que de acuerdo a las reuniones se han ido solventando y realizando en el desarrollo:

Figura 12

Figura 12 Pantall de Inicio (menú) principal

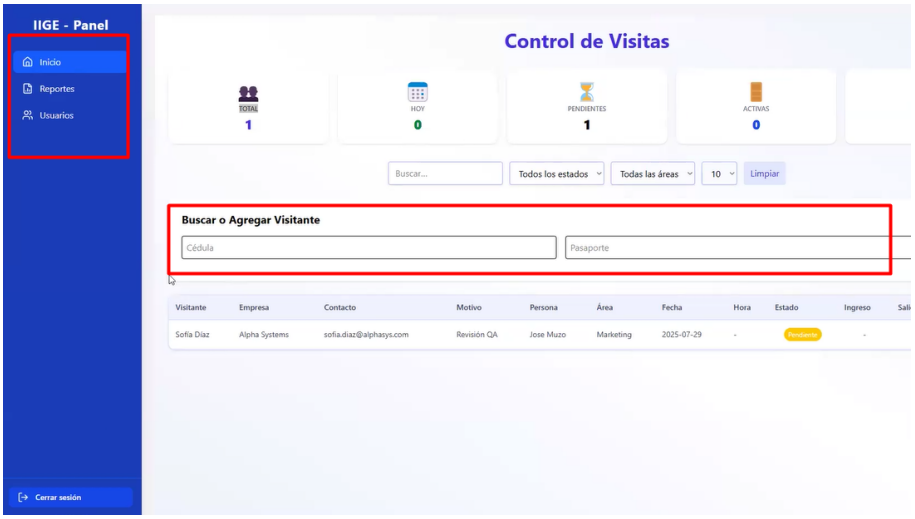


Figura 13

Figura 13 Ingreso de un visitante

Figura 14

Figura 14 Ingreso de formulario de registro

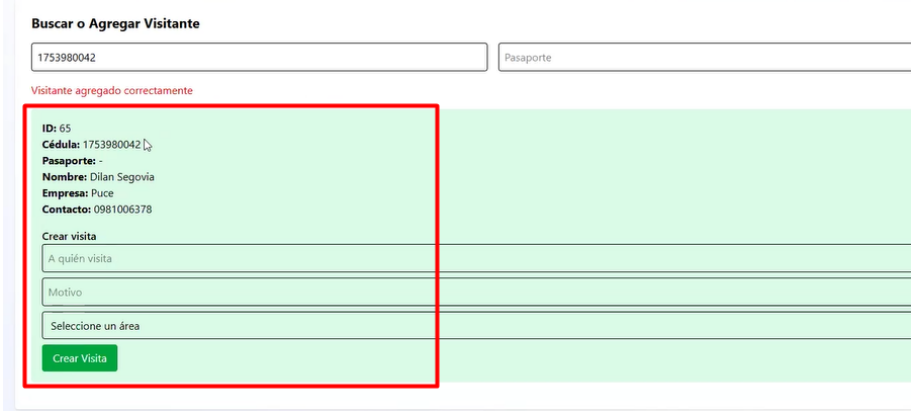


Figura 15 *Figura 15 Selección de roles IIGE*

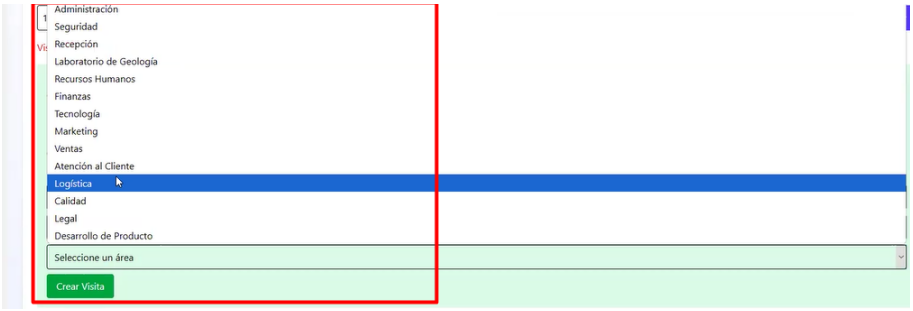


Figura 16 *Figura 16 Registro ingresado correctamente*

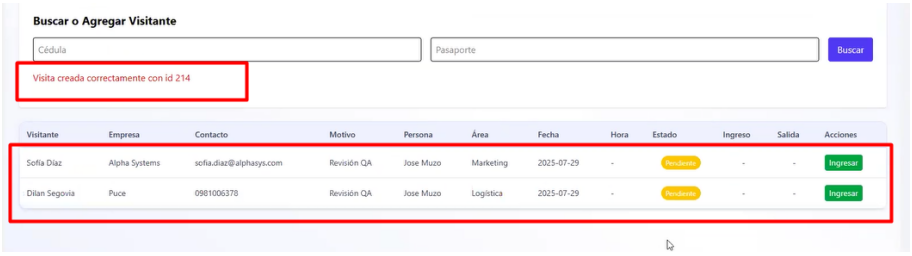


Figura 17 *Figura 17 Sección reportes diarios, mensuales.*

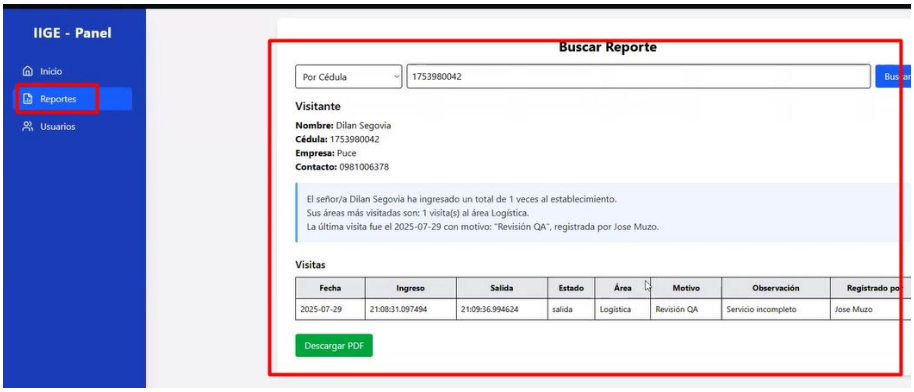


Figura 18 *Figura 18 Sección de registro de roles*

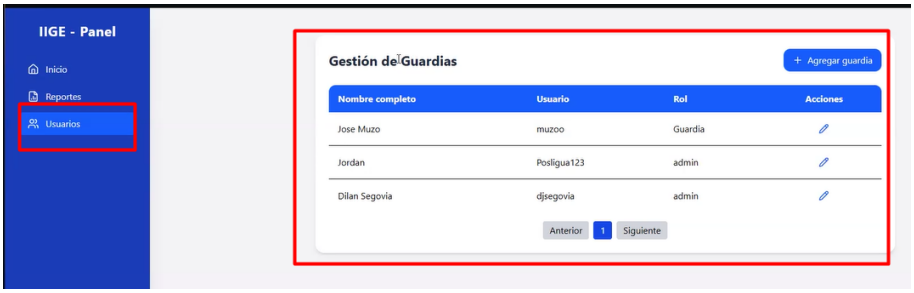
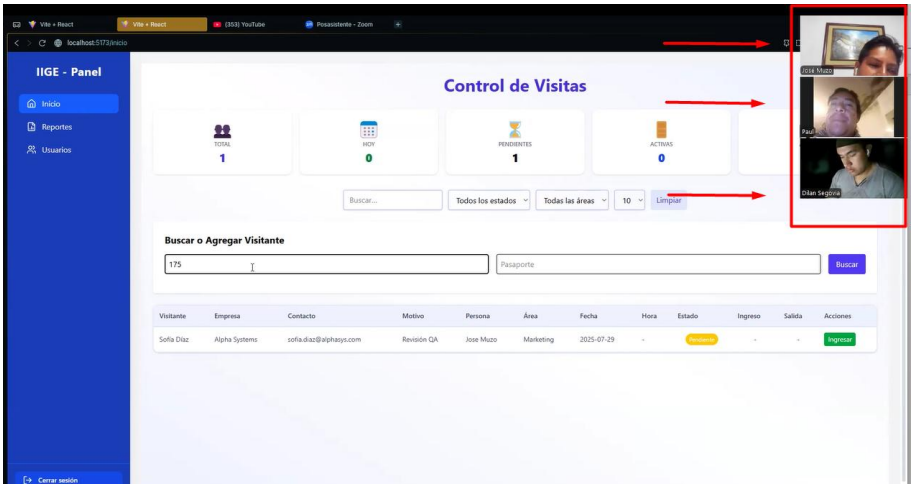


Figura 19 *Figura 19 Reunión con el Product Owner Revisión Final*



3.2 Pruebas y estabilización

Objetivo

Siguiendo la metodología Scrum, las pruebas y la estabilización del sistema se realizaron de forma iterativa en cada Sprint, garantizando que cada incremento del producto fuera funcional y estable antes de su integración. Esto permitió detectar y corregir errores de manera temprana, optimizar procesos y asegurar un flujo de trabajo ágil y controlado.

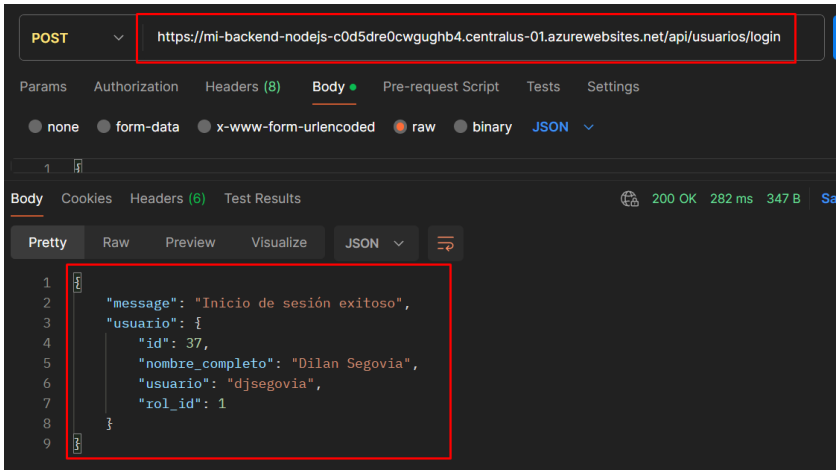
3.2.1 Proceso de pruebas en Scrum

En cada Sprint, una historia de usuario solo se consideró finalizada al cumplir con la Definición de Hecho (Definition of Done), que incluyó:

- **Ejecución de pruebas manuales y automáticas**

Ejemplo: Para verificar la funcionalidad de login, se usó Postman para enviar credenciales correctas e incorrectas y se comprobó que la API respondiera de manera adecuada.

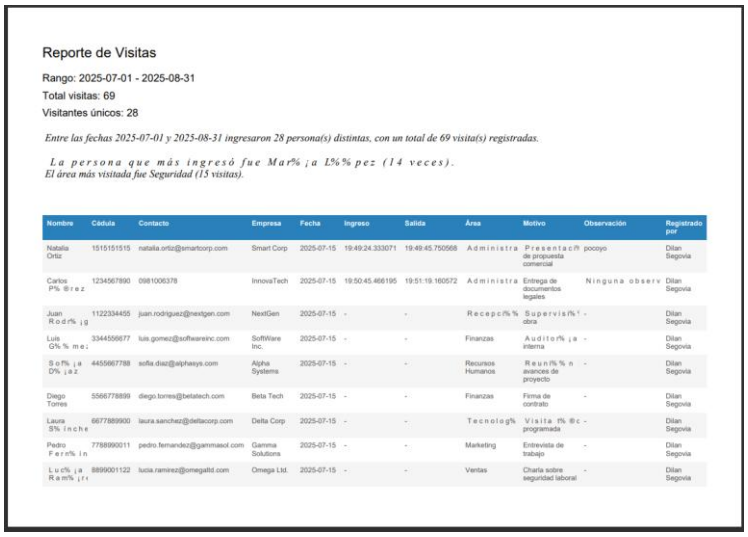
Figura 20 *Figura 20 Prueba realizada en Postman*



Validación visual y funcional de la interfaz

Ejemplo: En el frontend se comprobó que al presionar el botón "Generar Reporte" aparezca el PDF con los datos correctos.

Figura 21 *Figura 21 Ejecución de Pruebas de reportes*



Pruebas de comunicación entre frontend y API

Ejemplo: Se probó que, al buscar por cédula, el frontend enviara la petición y el backend devolviera los datos correctos en formato JSON.

Figura 22 *Figura 22 Pruebas de frontend y backend en la búsqueda de visitantes*

3.3 Tipos de pruebas realizadas

Frontend (React)

- Pruebas funcionales: validación de botones, formularios y navegación.
- Pruebas de usabilidad: comprobación de mensajes claros y flujos intuitivos.

Backend (API)

- Pruebas de base de datos: validación de inserción, actualización y consulta de datos en MySQL.
- Pruebas de endpoints: verificación de respuestas correctas en login, registro y listado de usuarios.
- Pruebas de integración: comprobación de interacción correcta entre modelos, controladores y rutas.

Herramientas utilizadas

- Postman: para pruebas de endpoints y validación de respuestas JSON.
- Consola y logs del servidor: para seguimiento de errores, depuración de código y monitoreo de ejecución.

3.1 Planilla de pruebas Sprint

Tabla 21 *Tabla 21 Tablas de Pruebas*

Sprint	Tarea	Descripción	Tipo de prueba	Responsable	Resultado esperado	Estado
S1	Modelado de base de datos	Diseño ER, tablas y relaciones necesarias para usuarios, visitas y reportes	Revisión / DB	jordandd56	Estructura coherente y aprobada	☑
S1	Crear scripts SQL para la base de datos	Scripts de creación de tablas y constraints + seeds básicos para pruebas	Ejecución scripts / DB	jordandd56	Scripts ejecutables sin errores; datos de prueba cargados	☑
S1	Crear migraciones y seeds	Migrations y seeders para entornos dev/staging	Ejecución / DB	jordandd56	Migraciones aplicadas correctamente	☑

S2	Crear modelos (ORM)	Modelos con relaciones y validaciones (Users, Reports, Visits, etc.)	Unit / Integración	jordandd56	Modelos con validaciones y relaciones correctas	☑
S2	Conexión a la Base de Datos	Configurar conexión, pool y variables de entorno	Integración (arranque)	jordandd56	Conexión estable sin errores en inicio	☑
S2	Crear controlador y ruta para login de usuarios	POST /login con validación de credenciales y respuesta clara	Endpoint (Postman)	jordandd56	200 con token para válidos / 401 para inválidos	☑
S2	Crear la funcionalidad para agregar usuarios	POST /users con validaciones	Endpoint (Postman)	jordandd56	Inserción y respuesta 201 con datos	☑
S2	Obtener listado de usuarios	GET /users (con posibilidad de filtros básicos)	Endpoint (Postman)	jordandd56	Lista JSON devolviendo usuarios	☑

S2	Validaciones y manejo de errores	Middlewares para validar inputs y manejar errores estándar (400/404/500)	Endpoint / Integración (Postman + logs)	jordandd56	Errores claros y códigos HTTP correctos	☑
S2	Pruebas unitarias básicas (modelos y controladores)	Tests automatizados para lógica crítica	Unit tests (local)	jordandd56	Cobertura de funciones críticas	☑
S3	Endpoint: Buscar por cédula	GET /users?cedula=xxxxxxxx — validar formato y resultado	Endpoint (Postman)	jordandd56	200 con datos si existe / 404 si no	☑
S3	Endpoint: Obtener reportes por fecha	GET /reports?from=YYYY-MM-DD&to=YYYY-MM-DD	Endpoint (Postman)	jordandd56	Devuelve registros dentro del rango	☑

S3	Endpoint: Combinar filtros (fecha + cédula)	GET /reports?cedula=&from=&to= para búsquedas compuestas	Endpoint (Postman)	jordandd56	Resultados consistentes y filtrados	☑
S3	Paginación y filtros avanzados	Implementar limit/offset o cursor en endpoints listados	Endpoint (Postman)	jordandd56	Paginación funcional y parámetros válidos	☑
S3	API para exportar datos para PDF	Endpoint que entregue JSON ya formateado para que frontend genere PDF (encabezado, filas, totales)	Endpoint (Postman)	jordandd56	JSON listo para jsPDF/autoTable	☑
S4	Documentación API y colección Postman	Generar OpenAPI/Swagger y exportar colección Postman para pruebas	Revisión / Postman	jordandd56	Documentación y colección compartible	☑
S4	Pruebas de integración	Flujos críticos: login → búsqueda →	Integración / E2E (manual	jordandd56	Flujos completos funcionando	☑

	con frontend	generación de PDF (simuladas)	+ Postman)			
S4	Preparación de Release Candidate y staging	Configuración de staging, logs y pruebas finales	Revisión / Logs	jordandd56	RC desplegado y logs activos para monitoreo	☑

3.2 Estabilización

En las últimas iteraciones antes de la entrega:

- Se corrigieron errores detectados en pruebas manuales y con Postman.
- Se depuró código Backend y se optimizó la conexión a base de datos.
- Se revisó y validó la correcta generación de reportes PDF en el Frontend.
- Se aseguraron mensajes claros en caso de errores o datos no encontrados.
- Se realizaron ajustes en el Fronted Roles (visualización de Administración y de Guardia de seguridad)
- Se realizaron cambio de Tonalidades de Color en el registro, estructura del Frontend.

Conclusiones

El desarrollo e implementación del sistema de control de visitas permitió modernizar el registro, seguimiento y gestión de ingresos a las instalaciones del instituto, sustituyendo procesos manuales por una solución digital escalable. La arquitectura en tres capas desplegada en Microsoft Azure, con separación de responsabilidades entre la interfaz de usuario, la lógica de negocio y la capa de datos, garantizó un rendimiento estable y seguro. El uso de tecnologías de código abierto como React, Node.js y PostgreSQL contribuyó a optimizar costos y favorecer la adaptabilidad de la solución a futuros requerimientos. La integración de herramientas como GitHub para el control de versiones, PowerDesigner para el modelado y Postman para las pruebas de API, junto con entornos de despliegue automatizados en Azure, consolidó un flujo de trabajo eficiente y colaborativo. Adicionalmente, se realizó la inducción al personal operativo, estandarizando el uso del sistema y facilitando su adopción institucional.

Durante la fase de implementación, el equipo de desarrollo brindó una inducción completa al personal del instituto, asegurando que comprendan el funcionamiento del sistema y sus principales procedimientos operativos. El involucramiento de los usuarios durante las pruebas y validaciones permitió ajustar la solución a las necesidades reales.

Recomendaciones

Se recomienda mantener un ciclo de actualización continua de las dependencias y librerías utilizadas en el frontend y backend, asegurando compatibilidad con nuevas versiones y parches de seguridad. Es conveniente documentar y versionar todo el código fuente en GitHub, siguiendo la estrategia de ramas definida, para garantizar trazabilidad y control en la evolución del sistema.

Asimismo, se sugiere implementar pruebas automatizadas, tanto unitarias como de integración, que permitan validar nuevas funcionalidades antes de desplegarlas en producción. También es recomendable configurar entornos de staging en Azure para validar cambios sin afectar la operación diaria.

En cuanto al rendimiento, se aconseja realizar revisiones periódicas de consultas y optimizar el modelo de datos si se detectan cuellos de botella. Igualmente, se puede considerar la implementación de caching para consultas recurrentes y la optimización de llamadas a la API para reducir latencia.

Finalmente, es fundamental que el instituto adopte un plan de mantenimiento preventivo para el equipo físico donde se ejecuta el sistema, junto con sesiones periódicas de capacitación técnica para los usuarios clave, a fin de garantizar la continuidad operativa y el aprovechamiento pleno de la solución desarrollada.

Referencias bibliográficas

Node.js. (s.f.). *Introduction to Node.js*. Recuperado el 28 de mayo de 2025 de https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

Express. (s.f.). *Express - Node.js web application framework*. Recuperado el 28 de mayo de 2025 de https://expressjs.com/

Sequelize. (s.f.). *Sequelize — Promises-based Node.js ORM for Postgres, MySQL, MariaDB, SQLite and Microsoft SQL Server*. Recuperado el 28 de mayo de 2025 de https://sequelize.org/docs/

PostgreSQL. (s.f.). *What is PostgreSQL?*. Recuperado el 28 de mayo de 2025 de https://www.postgresql.org/docs/current/intro-whatis.html

npm. (s.f.). *dotenv*. Recuperado el 28 de mayo de 2025 de https://www.npmjs.com/package/dotenv

Mozilla Developer Network. (s.f.). *CORS*. Recuperado el 28 de mayo de 2025 de https://developer.mozilla.org/es/docs/Web/HTTP/Guides/CORS