PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

Unidad Académica de Formación Técnica y Tecnológica – PUCE TEC

SISTEMA DE AGENDAMIENTO DE CANCHAS DEPORTIVAS Y PARQUEADEROS PARA CLUB SPARTANS DE LA EMPRESA SG CONSULTING GROUP

Proyecto de titulación previo a la obtención del título de:

Tecnología Superior en Desarrollo de Software

Autores: Gabriel Sebastián Márquez Muñoz

Luis Felipe Cepeda Peralta

Tutor: Ing. Billy Garzón

Quito, Ecuador

2025

Dedicatoria

Dedicamos este trabajo a nuestras familias, quienes han sido el pilar fundamental en este proceso académico y personal. A nuestros padres, madres y seres queridos, que con paciencia, amor y sacrificio han estado siempre a nuestro lado, motivándonos en los momentos de cansancio y celebrando con alegría cada uno de nuestros pequeños logros. Su apoyo incondicional ha sido la fuerza que nos impulsó a continuar, aun cuando el camino parecía difícil.

A nuestros docentes y mentores, quienes, con dedicación y vocación, han compartido con nosotros no solo su conocimiento técnico, sino también valores y principios que trascienden las aulas. Gracias por guiarnos, por exigirnos dar lo mejor de nosotros mismos y por mostrarnos que la tecnología es un puente que conecta la creatividad humana con la innovación y el progreso de la sociedad.

Dedicamos también este trabajo a nosotros mismos, estudiantes de la carrera de Desarrollo de Software, por la disciplina, el esfuerzo y la perseverancia invertidos en cada línea de código, en cada investigación y en noches de desvelo. Este logro refleja nuestro compromiso con la excelencia académica y la pasión por la tecnología como herramienta de transformación.

Finalmente, dedicamos este trabajo a todas aquellas personas que, de una u otra manera, han creído en nuestro potencial. Que este trabajo de titulación sea testimonio de que, con esfuerzo, dedicación y trabajo en equipo, es posible alcanzar metas que parecían lejanas. Este no es el final, sino el inicio de un camino de retos y aprendizajes que nos llevará a crecer como profesionales y como seres humanos.

Tabla de contenidos

Contenido

Lista de tablas

Se presentan las tablas que componen este trabajo de titulación.

1. Diccionario de Datos

Área					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdArea	Identificador de cada área insertado	NO	INT	Sin longitud
	Nombre	Nombre que tendrá cada área creada	NO	VARCHAR	100
FK	IdTipoArea	Referencia a la tabla TipoArea, especifica el tipo de área al que pertenecerá cada objeto.	NO	INT	Sin longitud
	Capacidad	Describirá la capacidad que tendrá cada área.	SI	INT	Sin longitud
	Estado	Describirá el estado de cada área.	SI	BIT	Sin longitud

AuditoriaSP					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**

PK	IdAuditoriaSP	Identificador de cada dato de auditoria insertado.	NO	INT	Sin longitud
	NombreSP	Nombre del Stored Procedure ejecutado.	SI	VARCHAR	128
	Usuario	Usuario de la Base de Datos donde se ejecutó ese Stored Procedure.	SI	VARCHAR	128
	FechaHora	Fecha y hora de la ejecución del Stored Procedure.	SI	DATETIME	Sin longitud
	Parametros	Todos los datos recibidos para la ejecución del Stored Procedure	SI	VARCHAR	512
	Estado	Estado del Stored Procedure	SI	VARCHAR	50

Cancha					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdCancha	Identificador de cada cancha insertada.	NO	INT	Sin longitud
FK	IdArea	Referencia a la tabla Área, especifica el área al que pertenece la cancha.	SI	INT	Sin longitud

FK	IdTipoCancha	Referencia a la tabla TipoCancha, especifica el tipo de cancha.	SI	INT	Sin longitud
FK	IdEstadoCancha	Referencia a la tabla EstadoCancha, determina el estado que tendrá cada cancha.	SI	INT	Sin longitud
	Nombre	Nombre que tendrá cada cancha	SI	VARCHAR	128
	Capacidad	Determina la capacidad que tiene cada cancha.	SI	INT	Sin longitud
	PrecioHora	Determina el precio que será cobrado por hora.	NO	DECIMAL	10,2

Cliente					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdCliente	Identificador de cada cliente insertada.	NO	INT	Sin longitud
FK	IdTipoDocumento	Referencia a la tabla TipoDocumento, determina el tipo de documento que registrara el usuario.	SI	INT	Sin longitud
	Nombre	Nombre del Cliente	NO	VARCHAR	128

	Apellido	Apellido del Cliente	NO	VARCHAR	128
	NumeroDocumento	Identificación del cliente, como pasaporte, cedula, RUC, etc.	NO	VARCHAR	32
	Teléfono	Número telefónico del cliente	SI	VARCHAR	32
	Email	Correo electrónico que utiliza el cliente	SI	VARCHAR	128
	FechaRegistro	Fecha y hora exacta del registro del cliente.	SI	DATETIME	Sin longitud
	Estado	Describe el estado en el que se encuentra el cliente.	SI	BIT	Sin longitud
	Contraseña	Contraseña con la que el usuario podrá ingresar al sistema.	NO	VARCHAR	300
	UidFirebase	En caso de autenticación mediante Firebase, se guardará uid enviada durante el registro.	SI	VARCHAR	128

ComplejoDeportivo					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**

PK	IdComplejo	Identificador de cada cliente insertada.	NO	INT	Sin longitud
	Nombre	Nombre del Complejo Deportivo.	NO	VARCHAR	100
	Dirección	Describe la dirección donde se encuentra ubicado el Complejo Deportivo.	NO	VARCHAR	200
	Ciudad	Describe la ciudad donde se encuentra el Complejo Deportivo	NO	VARCHAR	50
	Teléfono	Teléfono de contacto del Complejo Deportivo.	SI	VARCHAR	20
	Email	Correo Electrónico de Contacto del Complejo Deportivo.	SI	VARCHAR	100
	HorarioApertura	Horario de Apertura que tendrá el Complejo Deportivo.	SI	TIME	Sin longitud
	HorarioCierre	Horario de Cierre que tendrá el Complejo Deportivo.	SI	TIME	Sin longitud
	Estado	Describe el estado en el que se encuentra el Complejo Deportivo.	SI	BIT	Sin longitud

	FechaRegistro	Muestra la fecha en la que fue creado en la tabla el Complejo Deportivo.	SI	DATETIME	Sin longitud

ConfiguracionComplejo					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdConfiguración	Identificador de la configuración del complejo.	NO	INT	Sin longitud
FK	IdComplejo	Referencia a la tabla de ComplejoDeportivo, determina la configuración para el complejo asignado.	SI	INT	Sin longitud
	TiempoMinimo Reserva	Describe el tiempo mínimo de reservación del complejo.	SI	INT	Sin longitud
	TiempoMáximo Reserva	Describe el tiempo máximo de reservación del complejo.	SI	INT	Sin longitud
	AnticipaciónReserva	Se guardará los días de anticipación con los que el cliente hizo la reservación.	SI	INT	Sin longitud

	PoliticaCancelación	Detalla las normativas en caso de que un cliente cálcelo la reservación.	SI	VARCHAR	500

EstadoCancha					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdEstadoCancha	Identificador del Estado de la Cancha.	NO	INT	Sin longitud
	EstadoCanchaNombre	Describe los estados que puede tener la cancha.	SI	VARCHAR	64

EstadoPago					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdEstadoPago	Identificador del Estado de Pago.	NO	INT	Sin longitud
	EstadoPagoNombre	Describe los estados que puede tener al momento de realizar los pagos.	SI	VARCHAR	64

LogExcepcion					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	Id	Identificador del registro insertando del Log.	NO	INT	Sin longitud
	GuidMensaje	Código único de cada error suscitado en la ejecución de Stored Procedures.	SI	UNIQUEID ENTIFIER	Sin longitud
	Fecha	Guarda la fecha en la que ocurrió el error del Stored Procedure.	SI	DATETIME	Sin longitud
	Nivel	Indica el nivel que tiene el error al ser detectado.	SI	VARCHAR	64
	Mensaje	Guarda el mensaje del error ocurrido en la ejecución de los Stored Procedures.	SI	VARCHAR	max
	StrackTrace	Guarda el nombre del Stored Procedure donde ocurrió el error.	SI	VARCHAR	max
	Usuario	Guarda el usuario que ejecuto el Stored Procedure.	SI	VARCHAR	64
	Origen	Guarda el origen del error suscitado.	SI	VARCHAR	64

Pago					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdPago	Identificador de cada pago de la reserva de cancha registrada.	NO	INT	Sin longitud
FK	IdReserva	Referencia a la tabla de Reserva. Guarda las reservas de canchas realizadas del cliente.	SI	INT	Sin longitud
FK	IdCliente	Referencia a la tabla de Cliente. Guarda el cliente que realizo el pago.	NO	INT	Sin longitud
FK	IdTipoPago	Referencia a la tabla de TipoPago. Guarda el tipo de pago que selección el cliente.	NO	INT	Sin longitud
	FechaPago	Registra la fecha en la que el cliente realizo el pago.	SI	DATETIME	Sin longitud
	Monto	Guarda el monto total a pagar de la reserva de la cancha.	NO	DECIMAL	(10,2)
FK	IdEstadoPago	Referencia a la tabla EstadoPago, selecciona el	SI	INT	Sin longitud

		estado en el que se encuentra el pago.			
	Referencia	Guarda la referencia del pago.	SI	VARCHAR	64

PagoParqueadero					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdPagoParqueadero	Identificador de cada pago de la reserva de parqueadero registrado.	NO	INT	Sin longitud
FK	IdReservaParqueadero	Referencia a la tabla de ReservaParqueadero. Guarda las reservas de parqueadero realizadas por el cliente.	SI	INT	Sin longitud
FK	IdCliente	Referencia a la tabla de Cliente. Guarda el cliente que realizo el pago.	NO	INT	Sin longitud
FK	IdTipoPago	Referencia a la tabla de TipoPago. Guarda	NO	INT	Sin longitud

		el tipo de pago que selección el cliente.			
	FechaPagoParqueadero	Registra la fecha en la que el cliente realizo el pago.	SI	DATETIME	Sin longitud
	Monto	Guarda el monto total a pagar de la reserva del parqueadero.	NO	DECIMAL	(10,2)
FK	IdEstadoPago	Referencia a la tabla EstadoPago, selecciona el estado en el que se encuentra el pago.	SI	INT	Sin longitud
	Referencia	Guarda la referencia del pago.	SI	VARCHAR	64

Parqueadero					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdParqueadero	Identificador de cada parqueadero registrado.	NO	INT	Sin longitud

FK	IdÁrea	Referencia a la tabla de Area. Guarda el área donde está el parqueadero.	SI	INT	Sin longitud
	NúmeroEspacio	Guarda el nombre del parqueadero.	SI	VARCHAR	16
	Estado	Guarda el estado del parqueadero.	SI	VARCHAR	32
	IdTipoVehiculo	Referencia a la tabla de TipoVehiculo. Guarda el tipo de vehículo para el cual fue diseñado el parqueadero.	SI	INT	Sin longitud

Reserva					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdReserva	Identificador de cada reserva registrada.	NO	INT	Sin longitud
FK	IdCliente	Referencia a la tabla de Cliente. Guarda el cliente que realizo la reservación.	SI	INT	Sin longitud
FK	IdCancha	Referencia a la tabla de Cancha. Guarda la cancha reservada por el cliente.	SI	INT	Sin longitud

FK	IdComplejo	Referencia a la tabla de ComplejoDeportivo. Guarda el complejo deportivo al que pertenece la cancha reservada.	SI	INT	Sin longitud
	FechaHoraInicio	Guarda la fecha y la hora del inicio de la reserva.	NO	DATETIME	Sin longitud
	FechaHoraFin	Guarda la fecha y la hora del fin de la reserva.	NO	DATETIME	Sin longitud
	Observaciones	Guarda observaciones que puede hacer el cliente al momento de realizar la reservación.	SI	VARCHAR	526

Reserva Parqueadero					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdReservaParqueadero	Identificador de cada reserva de parqueadero registrada.	NO	INT	Sin longitud

FK	IdCliente	Referencia a la tabla de Cliente. Guarda el cliente que realizo la reservación del parqueadero.	SI	INT	Sin longitud
FK	IdParqueadero	Referencia a la tabla de Parqueadero. Guarda el parqueadero reservado por el cliente.	SI	INT	Sin longitud
FK	IdComplejo	Referencia a la tabla de ComplejoDeportivo. Guarda el complejo deportivo al que pertenece el parqueadero reservado.	SI	INT	Sin longitud
	FechaHoraInicio	Guarda la fecha y la hora del inicio de la reserva del parqueadero.	NO	DATETIME	Sin longitud
	FechaHoraFin	Guarda la fecha y la hora del fin de la reserva del parqueadero.	NO	DATETIME	Sin longitud
	Observaciones	Guarda observaciones que puede hacer el cliente al momento de realizar la reservación del parqueadero.	SI	VARCHAR	526

TipoArea					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdTipoArea	Identificador de cada Tipo de Área registrada.	NO	INT	Sin longitud
	Nombre	Guarda el nombre del Tipo de Área que se asignara.	NO	VARCHAR	50
	Descripción	Guarda una pequeña descripción de cada tipo de área.	SI	VARCHAR	200

TipoCancha					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdTipoCancha	Identificador de cada Tipo de Cancha registrada.	NO	INT	Sin longitud

	TipoCanchaNombre	Guarda el nombre del Tipo de Cancha que se asignara.	NO	VARCHAR	64
	Descripcion	Guarda una pequeña descripción de cada tipo de cancha.	SI	VARCHAR	256

TipoDocumento					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdTipoDocumento	Identificador de cada Tipo de Documento registrado.	NO	INT	Sin longitud
	TipoDocumentoNombre	Guarda el nombre del Tipo de Documento que se asignara a cada cliente en su registro.	NO	VARCHAR	32

TipoPago					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdTipoPago	Identificador de cada Tipo de Pago registrado.	NO	INT	Sin longitud

	TipoPagoNombre	Guarda el nombre del Tipo de Pago que se asignara a la tabla Pago.	NO	VARCHAR	64
	Descripcion	Guarda una pequeña descripción de cada tipo de pago.	SI	VARCHAR	200

TipoVehiculo					
Llave	**Nombre del Campo**	**Descripción**	**Es nulo**	**Tipo de dato**	**Longitud**
PK	IdTipoVehiculo	Identificador de cada Tipo de Vehículo registrado.	NO	INT	Sin longitud
	TipoVehiculoNombre	Guarda el nombre del Tipo de Vehículo que se asignara a la tabla Parqueadero.	NO	VARCHAR	32
	PrecioHora	Guarda el Precio por Hora que se cobrara dentro de la tabla PagoParqueadero.	NO	DECIMAL	(10,2)

Lista de figuras

DECLARACIÓN y AUTORIZACIÓN

Yo, GABRIEL SEBASTIÁN MÁRQUEZ MUÑOZ con C.I. 1755855747 autor(a) del trabajo de titulación intitulado: **"SISTEMA DE AGENDAMIENTO DE CANCHAS DEPORTIVAS Y PARQUEADEROS PARA CLUB SPARTANS DE LA EMPRESA SG CONSULTING GROUP"**, previa a la obtención del título de TECNÓLOGIA 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, agosto 2025

Gabriel Sebastián Márquez Muñoz

C.I. 1755855747

Yo, LUIS FELIPE CEPEDA PERALTA con C.I. 1726892324 autor(a) del trabajo de titulación: **"SISTEMA DE AGENDAMIENTO DE CANCHAS DEPORTIVAS Y PARQUEADEROS PARA CLUB SPARTANS DE LA EMPRESA SG CONSULTING GROUP"**, previa a la obtención del título de TECNÓLOGIA 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, agosto 2025

Luis Felipe Cepeda Peralta

C.I. 1726892324

Agradecimientos

Como autores de este proyecto, Sebastián Márquez y Felipe Cepeda, expresamos nuestro agradecimiento a las personas y organizaciones que aportaron conocimiento, tiempo y recursos para hacer posible el desarrollo y la documentación de este trabajo de titulación.

En primer lugar, agradecemos a SG Consulting Group, que nos brindó la oportunidad de ejecutar el proyecto para el equipo de baloncesto “Spartans”. Valoramos el acceso a información, la coordinación con sus áreas funcionales y la apertura para validar necesidades reales del usuario final. Este entorno profesional nos permitió alinear los entregables con expectativas de negocio y con buenas prácticas de la industria.

De manera particular, reconocemos al equipo de TI de SG Consulting Group por su acompañamiento en la arquitectura y la seguridad de la información. Sus revisiones técnicas, la provisión de ambientes de prueba y el soporte en integración continua fueron decisivos para asegurar la calidad del software, la trazabilidad de cambios y la estabilidad del proyecto.

Extendemos nuestro reconocimiento a nuestros docentes y tutores académicos, por el rigor metodológico y la guía en la estructuración del trabajo. Sus observaciones sobre redacción científica, citación de fuentes, diseño experimental y evaluación de resultados fortalecieron la solidez académica del documento y la reproducibilidad de los procedimientos.

A la facultad y sus unidades de apoyo (biblioteca, laboratorios, mesa de servicios y coordinación académica), gracias por facilitar acceso a infraestructura computacional, software con licencia y calendarios de revisión. Este soporte institucional fue clave para mantener cronogramas y estándares de calidad.

A nuestros compañeros, gracias por las revisiones por pares, los casos de prueba y las sesiones de retroalimentación que ayudaron a detectar escenarios límite y a mejorar el rendimiento del sistema.

Reconocemos, además, a la comunidad de software libre y de código abierto que mantiene los frameworks, librerías y herramientas que utilizamos. Su trabajo continuo de documentación, corrección de errores y publicación de mejoras hizo posible construir sobre una base tecnológica confiable.

Finalmente, agradecemos a nuestras familias y amistades por la flexibilidad y el respaldo logístico que nos permitieron concentrarnos en hitos críticos del proyecto. Su apoyo práctico y discreto resultó determinante para cumplir el plan de trabajo.

Cualquier omisión involuntaria es responsabilidad nuestra. Los aciertos que aquí se presentan son el resultado del esfuerzo compartido; los errores que permanezcan son exclusivamente responsabilidad de los autores.

Introducción

El presente trabajo de titulación desarrolla una versión beta de un sistema web para el agendamiento de canchas deportivas y parqueaderos del Club Spartans, perteneciente a SG Consulting Group. La propuesta responde a un historial de gestión manual (archivos locales, agendas y cuadernos) que ha generado falta de trazabilidad, riesgo de doble reserva y pérdida de información. Adicionalmente, Spartans aún no cuenta con un complejo operativo y su habilitación se encuentra en un proceso prolongado; por ello no existe inventario definitivo de canchas ni de plazas de estacionamiento. Esta entrega sienta la base funcional para que, cuando el complejo entre en operación, el sistema pueda parametrizarse y evolucionar hacia producción.

AVISO INSTITUCIONAL: Esta es una versión preliminar (beta) no productiva, desplegada únicamente en un entorno de desarrollo y destinada a fines académicos y de validación técnica. Por directriz expresa de la Dirección de TI de SG Consulting Group, la solución no está autorizada para uso en producción ni para exposición a usuarios finales. La disponibilidad de canchas/parqueaderos se gestiona con datos semilla y la funcionalidad de pagos opera en modo simulación; en consecuencia, no se procesan transacciones reales ni se generan reservas efectivas.

El sistema organiza su funcionalidad en dos ejes: reservas de canchas y asignación de parqueaderos (vinculada a la reserva de cancha o independiente). Se contemplan dos roles de acceso: cliente y administrador, permitiendo a este último registrar complejos y gestionar la operación futura. Como apoyo a la experiencia de uso, se integra un asistente conversacional en el sitio web del club, orientado a consultas de disponibilidad y recomendaciones básicas a partir de la información disponible.

El alcance de esta fase se limita al canal web y excluye: aplicación móvil nativa, control de acceso por códigos QR, facturación y módulos de mantenimiento. Tampoco se incluyen políticas institucionales de reserva/cancelación, que serán definidas por la administración cuando el complejo esté activo. Para asegurar la consistencia del proceso de agendamiento, se prevé la realización de pruebas de carga y concurrencia que verifiquen la ausencia de solapamientos lógicos en las reservas.

En síntesis, este trabajo entrega una plataforma funcional que resuelve los problemas base de registro y trazabilidad, incorpora un canal conversacional para la interacción y establece la línea tecnológica sobre la cual Spartans y SG Consulting Group podrán avanzar hacia un despliegue productivo cuando existan las condiciones organizacionales y técnicas requeridas.

Antecedentes

Históricamente, el Club Spartans gestionó la reserva de canchas y cupos de parqueadero utilizando infraestructura facilitada por un patrocinador privado. La administración de turnos se realizaba de forma manual, mediante archivos locales (hojas de cálculo en equipos personales), agendas físicas y cuadernos, sin un repositorio central ni mecanismos de control de cambios. Este esquema originó pérdida de información, dificultades para reconstruir el historial de reservas, riesgo de doble asignación y la ausencia de reportes consolidados de ocupación e ingresos diarios.

En la actualidad, Spartans se encuentra en un proceso de independencia institucional, ya no utiliza los complejos del patrocinador y avanza en la habilitación de su propio complejo deportivo, un proceso de mediano plazo. Por ello, no existe inventario definitivo de canchas ni plazas de estacionamiento que permita operar un sistema en producción.

Del diagnóstico histórico se desprenden los siguientes hallazgos:

- **Dispersión y fragilidad de datos:** registros en papel y archivos locales sin respaldo ni trazabilidad.
- **Riesgo operativo**: solapamientos de horarios y asignaciones informales de parqueaderos.
- **Limitada auditoría:** inexistencia de bitácoras para determinar quién reservó, modificó o canceló y cuándo.
- **Reportería insuficiente**: falta de indicadores de ocupación, ingresos por día, cancelaciones.
- **Dependencia de personas clave:** el estado de las reservas dependía del criterio y disponibilidad de quien llevaba la agenda.

En este contexto de transición de uso de instalaciones de un tercero hacia un complejo propio, surge la necesidad de sistematizar el proceso de agendamiento y dejar preparada una base tecnológica parametrizable, lista para adoptar el inventario real cuando el complejo entre en operación y conforme a las directrices institucionales.

Planteamiento del Problema

El Club Spartans de SG Consulting Group transita de utilizar instalaciones de un patrocinador privado a habilitar un complejo propio que aún no está operativo. En este escenario, la gestión de reservas de canchas y asignación de parqueaderos se ha realizado de forma manual, mediante archivos locales, agendas y cuadernos, sin un sistema centralizado ni inventario definitivo. Esta situación ha provocado dispersión y fragilidad de los datos, pérdida o sobreescritura de información, dificultad para reconstruir el historial de operaciones y ausencia de reportes confiables de ocupación e ingresos. Operativamente, se presentan riesgos de solapamiento de horarios y asignaciones inconsistentes de parqueaderos; para los usuarios, se traduce en demoras, incertidumbre y una experiencia limitada.

A corto plazo, y por directriz expresa de la Dirección de TI de SG Consulting Group, cualquier solución debe permanecer en un entorno de desarrollo no productivo hasta contar con inventario real, pasarela de pago habilitada y lineamientos operativos formales. En consecuencia, no existen políticas institucionales consolidadas de reserva y cancelación, la pasarela de pagos debe simularse y los datos disponibles corresponden a configuraciones semilla. La operación manual no escala, carece de trazabilidad (quién reservó, modificó o canceló y cuándo) y no permite validar con seguridad reglas básicas como el bloqueo de solapamientos ni asegurar la consistencia entre la reserva de cancha y la asignación de parqueaderos, ya sea vinculada o independiente.

En este contexto, el problema central se define como la inexistencia de un sistema de agendamiento que centralice la información de canchas y parqueaderos, prevenga colisiones de horarios, registre con trazabilidad las operaciones y provea reportes mínimos de gestión, incorporando además un canal conversacional que facilite consultas de disponibilidad y recomendaciones; todo ello bajo la restricción institucional de operar únicamente en versión beta no productiva, con pagos simulados y sin inventario definitivo. En términos de pregunta orientadora: ¿cómo diseñar e implementar, en un entorno de desarrollo, un sistema web para el agendamiento de canchas y parqueaderos del Club Spartans que garantice centralización, no solapamiento y trazabilidad, apoyado por un asistente conversacional, respetando la directriz de no despliegue productivo y la ausencia temporal de inventario y pasarela de pagos reales?

Justificación

El "Complejo Deportivo de Spartans" no dispone actualmente de un sistema digital para gestionar reservas de canchas y parqueaderos. La administración manual mediante archivos locales, agendas y cuadernos ha provocado confusiones operativas, pérdida o sobreescritura de datos, dificultad para verificar el historial de movimientos y errores al momento de registrar y conciliar pagos por uso de canchas o parqueaderos por hora. A esto se suma que el club atraviesa una transición hacia la independencia de su propio complejo, periodo en el cual no existe inventario definitivo y, por directriz de la Dirección de TI, cualquier solución debe mantenerse fuera de producción hasta contar con lineamientos y pasarela de pago reales. En este contexto, la sistematización no puede postergarse: es necesaria para ordenar el proceso, aun cuando su operación inicial se limite a un entorno de desarrollo.

El proyecto justifica su realización porque centraliza la información crítica en una sola plataforma, incorpora trazabilidad (quién hizo qué y cuándo), previene solapamientos de

horarios y estandariza la asignación de parqueaderos (vinculada a la reserva de cancha o de forma independiente). Además, habilita un panel administrativo para control y monitoreo de la operación, genera reportes de ocupación e ingresos, y deja definidos los flujos que la institución podrá activar cuando el complejo entre en operación. Mientras la pasarela oficial se habilita, el módulo de pagos opera en modo simulación, permitiendo validar el proceso end-to-end sin ejecutar transacciones reales.

La inclusión de un asistente conversacional aporta valor desde el inicio: facilita consultas de disponibilidad y recomendaciones básicas en tiempo real, reduce fricción en la búsqueda de horarios y orienta al usuario dentro del sitio. Aunque su alcance es acotado en esta fase, sienta las bases para incorporar capacidades más avanzadas a futuro (recordatorios, cancelaciones guiadas, políticas de uso), una vez que existan reglas institucionales definitivas.

Desde el punto de vista institucional y académico, la solución entrega una base tecnológica comprobable (modelos, procesos, reglas y evidencias de prueba) sobre la cual la organización puede: definir políticas de reserva/cancelación con datos reales, medir capacidad, ocupación y demanda por franja y mitigar riesgos de errores humanos antes de pasar a producción. En síntesis, aun en condición beta no productiva, el sistema reduce incertidumbre operativa, mejora el control administrativo y acelera la futura transición a un servicio digital robusto, alineado con la transformación organizacional de Spartans.

Objetivos Generales

Desarrollar un sistema de agendamiento de canchas deportivas y parqueaderos para el Club Spartans de la empresa SG Consulting Group, comprendiendo la construcción de una solución web funcional que centralice la información de reservas, evite solapamientos,

permita asignar cupos de parqueadero asociados o independientes a la reserva de cancha, y ofrezca al personal un panel de administración para la gestión operativa.

Objetivos Específicos

- Diseñar interfaces graficas que sean amigables e intuitivas, para que el usuario pueda navegar de forma eficiente y segura por la aplicación, abordando el diseño centrado en el usuario, priorizando flujos claros y rotulación comprensible. El resultado esperado es una experiencia simple y predecible que reduzca tiempos de aprendizaje y la necesidad de asistencia externa.
- Implementa la función de Reserva de Canchas, que le facilite al usuario tener una comprensión rápida de su uso, para que pueda realizar reservaciones de canchas de manera exitosa.
- Implementar la función de Reserva de Parqueaderos, que facilite al usuario encontrar parqueaderos disponibles dentro del complejo, para ahorrar tiempo en la búsqueda al usuario cuando se encuentre en el complejo.
- Implementar un chatbot interactivo con inteligencia artificial que realizará recomendaciones de horarios de disponibilidad de canchas y espacios disponibles en el parqueadero, dependiendo del análisis de información del usuario.

Metodología de Desarrollo de Software

Se adoptó un marco ágil basado en Scrum, soportado por un tablero Kanban en Azure DevOps Boards para visualizar el flujo de trabajo. Este enfoque (a menudo llamado Scrumban) nos permite planificar por sprints e, internamente, gestionar el flujo continuo con límites de trabajo en progreso, manteniendo alta visibilidad y capacidad de ajuste.

Roles y responsabilidades

Product Owner (PO): representante funcional de Spartans.

Scrum Master: facilita eventos, remueve impedimentos.

Equipo de Desarrollo: Sebastián Márquez y Felipe Cepeda, responsables de análisis, diseño, codificación, pruebas, despliegue en DEV y documentación.

Stakeholders clave: Dirección de TI de SG Consulting Group (define directrices y el no despliegue productivo), usuarios de negocio.

Estructura de los capítulos

- **Capítulo I** - Levantamiento de requisitos y Diseño del Sistema:

Se explicará de forma detalla el problema detectado dentro del Complejo Deportivo de Spartans basadas en entrevistas realizadas con encargados del Complejo y con trabajadores departamento de TI de la empresa SQ Consulting Group , para la posterior redacción de historias de usuario, esto facilitando el levantamiento de requerimiento funcionales y no funcionales, en conjunto a ello, también se establecieron los objetivos generales y específicos del proyecto que son la base para dar forma a la estructura del proyecto para su futuro desarrollo. Al igual se explicará él porqué del uso de la Metodología Ágil SCRUM, el manejo de tareas, épicas y la construcción de Sprints necesarios para la culminación del proyecto de manera organizada.

- **Capitulo II -** Construcción del Sistema:

Se explicará la manera en la que fue estructurado el proyecto desde la elección de la base de datos para el alojamiento de información, la construcción del BackEnd que llevará la lógica de la aplicación y el FrontEnd que será la interfaz donde el usuario podrá interactuar, junto a las herramientas y lenguajes de programación que se utilizaron para su desarrollo y despliegue de este.

- **Capitulo III -** Pruebas y Estabilización:

Se explicará las pruebas realizadas para probar la aplicación web. Como lo son pruebas unitarias de métodos de la API desarrollada y pruebas de estrés para conocer a qué punto la API puede llegar a tener un fallo en sus respuestas.

Capítulo I

Levantamiento de Requisitos y Diseño del Sistema

1. Marco Teórico

1.1. Club Spartans / SG Consulting Group

El Club Spartans forma parte del ecosistema de SG Consulting Group. Históricamente utilizó instalaciones facilitadas por un patrocinador privado; actualmente transita hacia la independencia con la habilitación de un complejo propio. En esta fase, no existe inventario definitivo de canchas ni de plazas de estacionamiento. Adicionalmente, por directriz expresa de la Dirección de TI de SGCG, toda solución tecnológica vinculada al club debe operar en entorno de desarrollo (DEV) como versión preliminar (beta) no productiva, con datos semilla y pagos simulados, sin exposición a usuarios finales ni despliegue en producción hasta nueva autorización institucional.

Conceptos y definiciones operativas

- **Agendamiento de canchas:** proceso de asignar franjas horarias a canchas específicas, evitando solapamientos y registrando trazabilidad (quién/qué/cuándo).
- **Gestión de parqueaderos:** asignación de cupos de estacionamiento, vinculados a una reserva de cancha o independientes.
- **Trazabilidad y auditoría:** registro verificable de eventos (creación, modificación, cancelación y pago), con sello temporal y usuario responsable.
- **No-show:** ausencia del usuario a una reserva confirmada (políticas aún no definidas en esta fase).

- **Asistente conversacional (chatbot):** componente de soporte para consultas y recomendaciones básicas de disponibilidad; en esta etapa se integra vía Copilot Studio y opera sobre información en Azure SQL.

- **Entorno DEV / datos semilla:** ambiente de desarrollo con información no productiva utilizada para pruebas, demostraciones y validaciones sin efectos reales.

- **Pagos simulados:** reproducción del flujo de pago sin transacción monetaria efectiva; integración futura prevista con PayPhone.

1.2. Sistemas de información y digitalización organizacional

Los sistemas de información son herramientas tecnológicas diseñadas para recolectar, almacenar, procesar y distribuir datos de soporte a la toma de decisiones y coordinación organizacional. Según Laudon & Laudon, un sistema de información bien alineado con las metas de la organización puede optimizar procesos, mejorar la colaboración entre departamentos y permitir mayor rapidez a nivel estratégico y operativo (Laudon & Laudon, 2020).

En el caso de Club Spartans, la implantación de un SI y su integración con módulos de reserva, pago y atención conversacional favorece la reducción de errores operacionales, agiliza procedimientos y configura un nuevo modelo de atención al socio con disponibilidad 24/7.

1.3. Reserva de canchas deportivas: fundamentos y casos previos

La literatura sobre sistemas corporativos de reserva de espacios deportivos evidencia un uso funcional del SI con módulos de registro de usuarios, visualización del calendario, selección de slot, pago y perfil de usuario (Can, Junjie, Hongxiang, & Zhan). En el sistema diseñado por Zhan. para una universidad china, se implementaron funcionalidades clave:

1. Autenticación de usuarios (estudiantes/visitantes)

2. Reserva en línea con planificación diaria o semanal

3. Pago virtual vinculado al perfil

4. Panel de administración con gestión de salones y horarios (Can, Junjie, Hongxiang, & Zhan).

Este modelo constituye un referente técnico cercano al Club Spartans, que requiere reserva de canchas y parqueaderos, con disponibilidad en franjas temporales y pagos automatizados.

1.4. Estacionamientos inteligentes con IoT y pagos en línea

Los sistemas "smart parking" combinan sensores (magnéticos, ultrasonidos, visión), comunicación IoT y arquitectura distribuida de capas fog + cloud para detectar ocupación, ofrecer reserva de plaza y reducir tiempo de búsqueda (Balfaqih, Waheb, Mashael, & Rosilah, 2021). En la propuesta de Balfaqih, el uso de IoT y fog computing se logro:

- Guía en tiempo real al usuario
- Menores tiempos de llegada (16–46 %)
- Ingresos incrementales para el gestor (10–15 %)
- Estimación predictiva de plazas libres (Balfaqih, Waheb, Mashael, & Rosilah, 2021).

Un sistema de Smart Parking como el del Club puede integrarse vía APIs a un sistema que actualice disponibilidad, valide reservas y controle acceso físico como barrera de autos, QR o RFID.

Sistema que servida de base para hacer la primera implementación del sistema de parqueaderos donde se registrará la reserva y agendamiento del parqueadero, mas no se

incluirá la implementación con sensores o cámaras para la detección del vehículo. Esto se pensará para una futura versión del sistema de agendamiento.

1.5. Integración de pagos digitales con PayPhone

PayPhone es una pasarela de pagos usada a nivel mundial que ofrece una API REST de conexión que genera transacciones desde aplicaciones externas. Incluye autenticación OAuth2, creación de transacciones, verificación de estado y webhook de notificación (Payphone, 2024). Usando esta API, el sistema de reserva del Club Spartans puede:

- Enviar información necesaria como el monto, la descripción y referencias del pago.
- Recibir un token para la sesión de pago
- Confirmar estados mediante “approved” o “rejected”.
- Garantizar consistencia transaccional entre reserva y cargo bancario.

De esta forma, el precio de la reserva queda cobrado antes de confirmar automáticamente la plaza.

1.6. Chatbot y experiencia conversacional

El uso de chatbots orientados al usuario mejora la accesibilidad de reservas mediante interfaces conversacionales accesibles vía WhatsApp, web o app. (Følstad & Brandtzaeg, 2017) indica que el nivel de satisfacción del usuario con chatbots depende en gran medida de:

1. Diseño claro del flujo conversacional (intenciones, respuestas rápidas)
2. Manejo imperfecto de errores y reintentos
3. Transparencia y personalización del diálogo (mostrar opciones, confirmar datos) (Følstad & Brandtzaeg, 2017).

En el contexto del Club Spartans, un chatbot integrado puede guiar al usuario desde la consulta de disponibilidad, paso por el pago y confirmación del turno, con fallback a agente humano si no se entiende la intención.

1.7. Tendencias de digitalización en clubes deportivos

Estudios recientes muestran que los principales clubes deportivos europeos estructuran sus portales web como el centro de operaciones digitales, sin explotar todas las posibilidades de interacción disponible (Tejedor, Cervi, & Gordon, 2019). Las funciones más usuales son marketing institucional, venta de entradas y merchandising, pero pocos ofrecen servicio de reserva de instalaciones ni chat de soporte en línea.

Dado que el Club Spartans desea además reservas automatizadas, cobros y atención personalizada, está adelantándose a muchas prácticas tradicionales en clubes deportivos, posicionándose como digitalmente competitivo y centrado en la experiencia del usuario (Tejedor, Cervi, & Gordon, 2019).

1.8. Flujo general del sistema propuesto

1. Usuario interactúa con el chatbot o FrontEnd web/react.
2. Consulta disponibilidad de cancha o parqueadero.
3. Sistema consulta base SQL Server para plaza libre.
4. Reserva se crea con estado “pendiente pago”.
5. Redirección al Hall de Pagos; si es exitoso, cambia estado a “pagado”; si falla, “rechazado”, con notificación al usuario.
6. Gestión central desde Azure DevOps: control de versiones, despliegue, tickets de incidencia.

Este diseño utiliza los módulos tecnológicos declarados: SQL Server (base de datos), ASP.NET Core + React (frontend/backend), C# (.NET 7), pipeline CI/CD en Azure DevOps. El chatbot puede implementarse con Bot Framework + Channels como WhatsApp o Messenger.

2. Desarrollo de la Metodología SCRUM

Durante el desarrollo del proyecto se asignó los roles necesarios como lo son:

Product Owner: Paul Márquez.

Scrum Master: Ing. Kevin Santillan.

Equipo de Desarrollo: Sebastián Márquez y Felipe Cepeda.

2.1. Sprints Realizados

Los Sprints fueron trabajados con un tiempo de duración de 2 semanas cada uno, con reuniones de 10 minutos con el Scrum Master quien nos daba las métricas para el desarrollo. Los Sprint se dividieron en los siguientes:

2.1.1. Sprint 1 Diagrama Entidad-Relación y Modelación de la Base de datos:

Fechas del Sprint: 03/03/2025 al 16/03/2025

Se realizo una reunión con el Product Owner para establecer las tablas de la base de datos necesarias, que se utilizaran para que funcione correctamente el sistema de reservas de canchas y parqueaderos. Se analizo un total de 18 tablas más una tabla de “LogException” donde principalmente se utilizará para llevar auditoria de los procedimientos almacenados ejecutados o distintos errores que puedes ir surgiendo en la ejecución de estos dentro de la Base de Datos

Ilustración 1 Sprint1

2.1.2. Sprint 2 Creación de la Base de Datos y Establecer modelos en el Backend:

Fechas del Sprint: 17/03/2025 al 30/03/2025

Se generaron los scripts de creación de la base de datos, en los que se plasmaron las tablas definidas en el Sprint 1, incluyendo la tabla LogException para la auditoría de procedimientos almacenados y el registro de errores.

Se inicializó el backend con ASP.NET Core (C#), framework que facilita la construcción de la API y su futura integración con el frontend.

Se estructuró el backend bajo una arquitectura monolítica modular. Se creó la carpeta Models, donde se definieron las entidades que representan las tablas de la base de datos, favoreciendo la claridad, el mantenimiento y la reutilización del código.

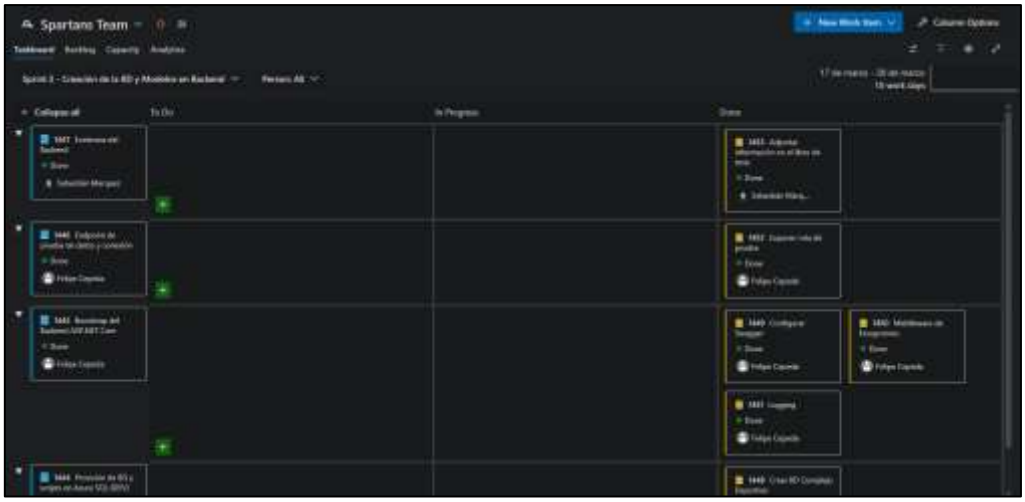


Ilustración 2 Sprint 2

2.1.3. Sprint 3 Programación de los Procesos Almacenados de Inserción:

Fechas del Sprint: 31/03/2025 al 13/04/2025

Una vez creada la base de datos, se procedió con el desarrollo de procedimientos almacenados destinados a facilitar la inserción de datos en cada tabla. Estos resultan fundamentales, ya que serán consumidos por los servicios del BackEnd. Además de gestionar las inserciones, dichos procedimientos encapsulan parte esencial de la lógica de la aplicación, garantizando consistencia, eficiencia y centralización en el manejo de la información.

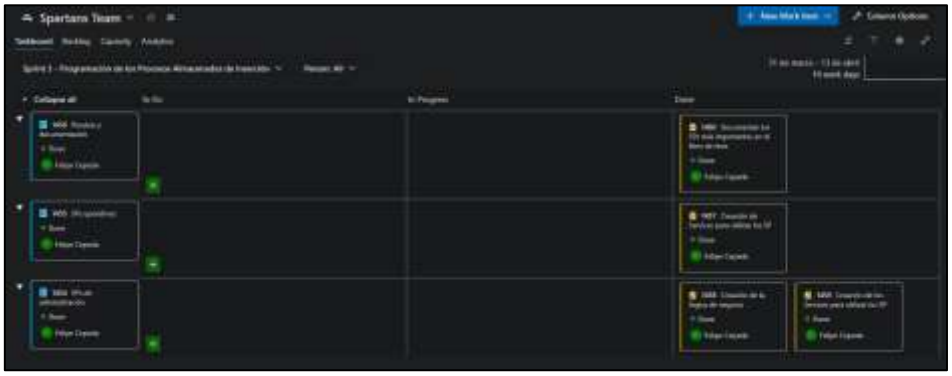


Ilustración 3 Sprint 3

2.1.4. Sprint 4 Pruebas y Correcciones de Funcionamiento de los Procesos Almacenados de Inserción:

Fechas del Sprint: 14/04/2025 al 27/04/2025

Se termino con la implementación y desarrollo de los Procedimientos Almacenados de Inserción de datos en las tablas, se procedió con su respectiva revisión y pruebas mediante el uso de Inserts de simulación para comprobar si todas las validaciones, sobre todo en los procedimientos almacenados de tablas como Pago y PagoParqueadero que calculan el monto de la reservación de las canchas, con el tiempo de reservación y el IVA del 15% que maneja la Republica del Ecuador.

Si ocurría algún fallo dentro de ellos, sea un fallo de datos o fallos de lógica, se sometió a una revisión profunda en el procedimiento almacenado para posteriormente resolver el problema.

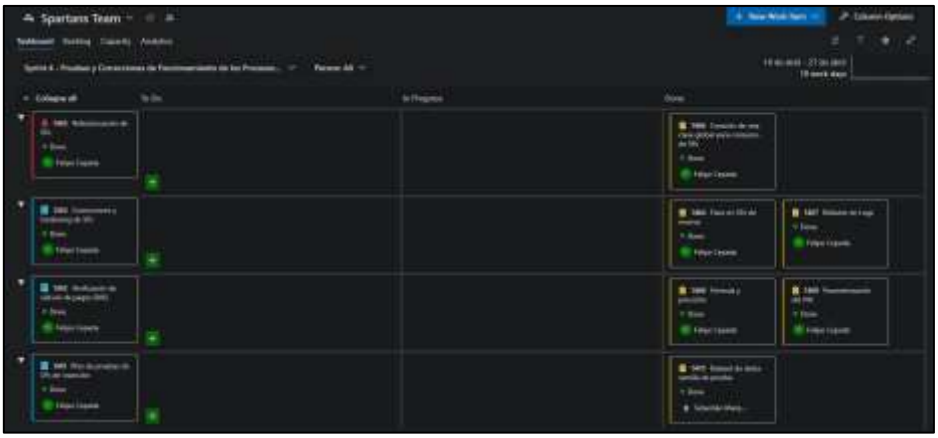


Ilustración 4 Sprint 4

2.1.5. Sprint 5 Programación de los Procesos Almacenados de Actualización:

Fechas del Sprint: 28/04/2025 al 11/05/2025

Se continuó con el desarrollo e implementación de los procedimientos almacenados orientados a la actualización de datos dentro de las tablas de la base de datos. Estos procedimientos siguen la misma lógica utilizada en los procesos de inserción, pero están enfocados en garantizar la modificación controlada y segura de la información ya existente.

La actualización de registros resulta fundamental, ya que permite mantener la integridad y consistencia de los datos a lo largo del ciclo de vida del sistema. Asimismo, este mecanismo es de gran relevancia tanto para los usuarios finales como para el personal encargado de la gestión del sistema, al asegurar que cualquier cambio realizado sobre la información refleje fielmente la realidad operativa y responda a las necesidades de la aplicación.

Ilustración 5 Sprint 5

2.1.6. **Sprint 6 Pruebas y Correcciones de Funcionamiento de los Procesos Almacenados de Actualización:**

Fechas del Sprint: 12/05/2025 al 25/05/2025

Se llevaron a cabo pruebas utilizando el envío de parámetros dentro del procedimiento almacenado, con el fin de verificar que la actualización de los datos en

las tablas se ejecutara de manera correcta y que las validaciones implementadas cumplieran adecuadamente su función.

Durante este proceso se evaluó la consistencia de los resultados y la capacidad del procedimiento para manejar distintos escenarios de entrada. En caso de detectarse errores o una compilación incorrecta de la lógica, se procedió con la revisión detallada del código y la posterior corrección de los fallos identificados. Este ciclo de prueba y ajuste permitió garantizar la confiabilidad del procedimiento, fortaleciendo tanto la robustez del sistema como la integridad de la información gestionada.

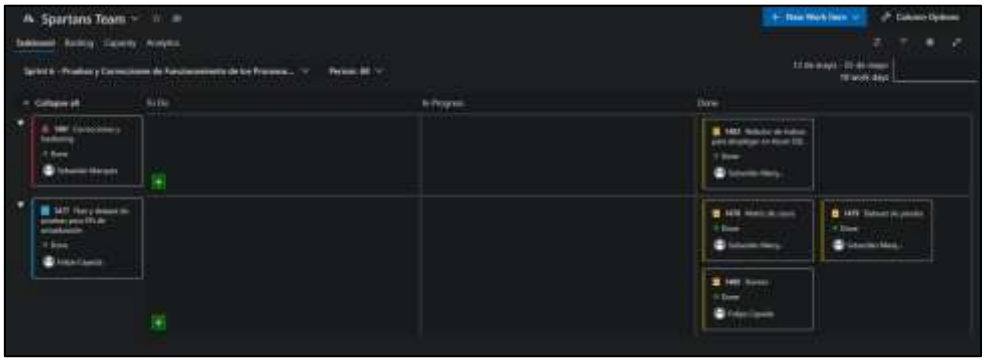


Ilustración 6 Sprint 6

2.1.7. Sprint 7 Programación de Vistas en la Base de Datos:

Fechas del Sprint: 26/05/2025 al 08/06/2025

Se desarrollaron e implementaron vistas en la base de datos para optimizar la ejecución de consultas (GET) del BackEnd, reduciendo la latencia y simplificando la lógica en la capa de Services. Estas vistas encapsulan los INNER JOIN necesarios y exponen proyecciones listas para consumo, evitando construir uniones en tiempo de ejecución y estandarizando la obtención de datos.

Ilustración 7 Sprint 7

2.1.8. Sprint 8 Desarrollo de los Servicios en Backend:

Fechas del Sprint: 09/06/2025 al 22/06/2025

En la capa de Services se implementó la lógica de orquestación para el consumo seguro de procedimientos almacenados y vistas definidos en la base de datos. Se estandarizó el envío de parámetros mediante contratos tipados y validaciones previas, y se centralizó la ejecución en métodos asíncronos que invocan los SP y consultan las vistas, reduciendo latencia y complejidad en la lógica de negocio. Los códigos de retorno de los procedimientos (éxito, reglas de negocio, errores controlados) se mapearon a respuestas consistentes de la aplicación, mientras que el manejo de errores se uniformó con bloques try/catch, registrando eventos en LogException para asegurar trazabilidad.

Además, los Services quedaron inyectados por dependencia y reutilizan un helper de acceso a datos para ejecutar comandos parametrizados, controlar timeouts y propagar tokens de cancelación cuando aplica. Las lecturas que alimentan consultas del frontend se realizan sobre vistas optimizadas, evitando construir joins en tiempo de ejecución. Todas las pruebas y evidencias se ejecutaron en entorno DEV, utilizando datos semilla y pagos simulados, en coherencia con la directriz institucional de no despliegue productivo.

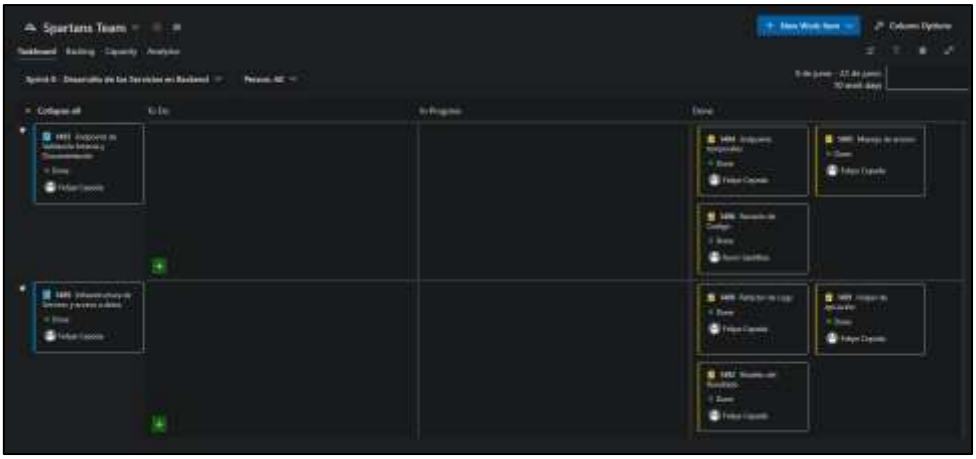


Ilustración 8 Sprint 8

2.1.9. Sprint 9 Desarrollo de los Controladores en BackEnd:

Fechas del Sprint: 23/06/2025 al 06/07/2025

Se desarrollaron los controladores de la API que orquestan la lógica implementada en la capa de Services y exponen endpoints REST para cada caso de uso. Cada controlador utiliza inyección de dependencias para invocar servicios, aplica model binding y validación de DTOs, y normaliza las respuestas mediante un helper que traduce los códigos de negocio a estados HTTP adecuados.

Se definió el uso semántico de los verbos GET/POST/PUT/PATCH/DELETE según el patrón CRUD. Además, se habilitó autenticación/autorización con [Authorize] y políticas por rol (administrador/cliente), manejo centralizado de errores y CORS para el FrontEnd.

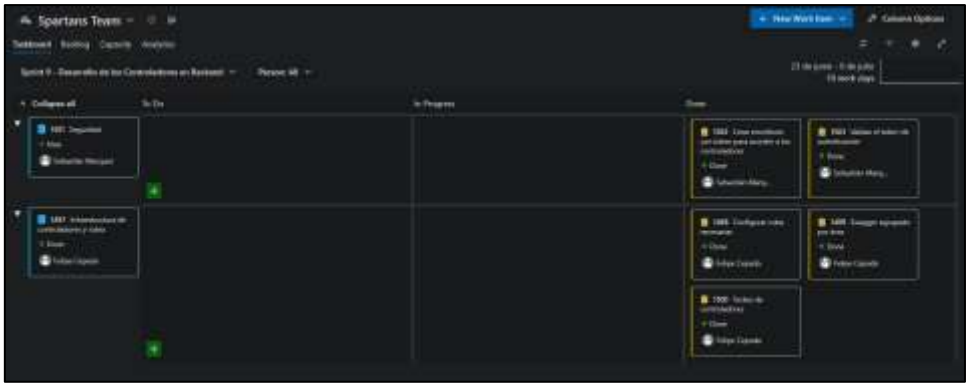


Ilustración 9 Sprint 9

2.1.10. **Sprint 10 Desarrollo del módulo de Autenticación y Seguridad con Firebase:**

Fechas del Sprint: 07/07/2025 al 20/07/2025

Se configura dentro del BackEnd los scripts necesarios para la implementación de Firebase, que puede proporcionar el uso de proveedores de inicio de sesión como Google, Facebook y Microsoft.

Se implemento el módulo de verificación mediante Tokens, con el uso de un BearerToken y Custom Clains añadiendo un rol administrador o cliente en el Token, dependiendo el rol que ocupe el token, el usuario accederá a distintas funcionalidades o pantallas dentro del sistema.

Con el rol administrador, tendrá el acceso a todos los endpoints de la API, mientras que con el rol cliente, tendrá un acceso limitado.

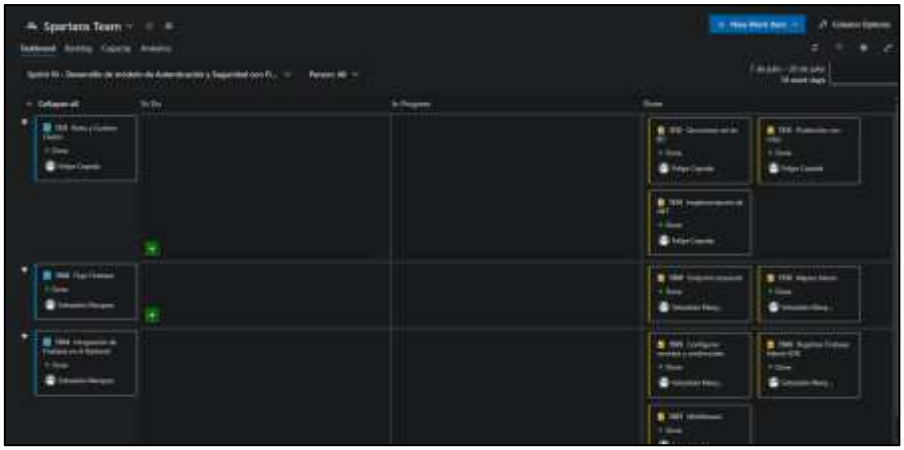


Ilustración 10 Sprint 10

2.1.11. Sprint 11: Desarrollo de los Componentes del Frontend:

Fechas del Sprint: 21/07/2025 al 03/08/2025

Se desarrollaron los componentes del Frontend a partir de una plantilla base implementada en React, diseñando las pantallas o vistas necesarias para consumir los métodos expuestos por el BackEnd y garantizar una interacción eficiente y fluida entre el usuario y la aplicación.

Asimismo, se implementó un sistema de control de acceso mediante vistas protegidas. De esta forma, únicamente los usuarios cuyo token contenga el rol de administrador podrán acceder a las secciones de gestión avanzadas, mientras que los usuarios con el rol de cliente tendrán acceso exclusivo a las funcionalidades de reservación de canchas, administración de parqueaderos y procesamiento de pagos asociados a sus reservaciones.

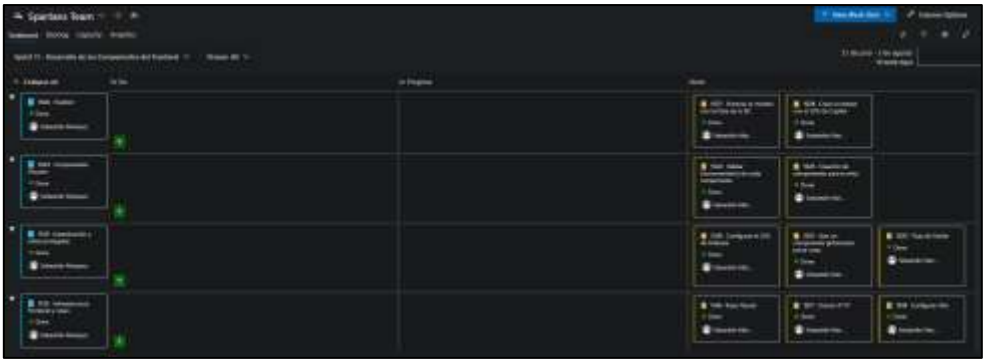


Ilustración 11 Sprint 11

2.1.12. Sprint 12 Desarrollo de los Pipelines y Artifacts para el Despliegue:

Fechas del Sprint: 04/08/2025 al 17/08/2025

Para el despliegue de la aplicación se implementaron Pipelines y Artifacts, en los cuales se configuraron las dependencias necesarias, así como la información correspondiente a las máquinas virtuales que alojarían la solución.

Adicionalmente, se hizo uso de Azure SQL Database como servicio de base de datos en la nube, donde finalmente quedó alojada la información. Es importante señalar que, durante la fase inicial, la base de datos se gestionó de manera local con el propósito de realizar todas las pruebas correspondientes en un entorno controlado. Una vez validadas dichas pruebas, se procedió a su migración hacia Azure mediante

un respaldo (backup) generado a partir de la base de datos local.

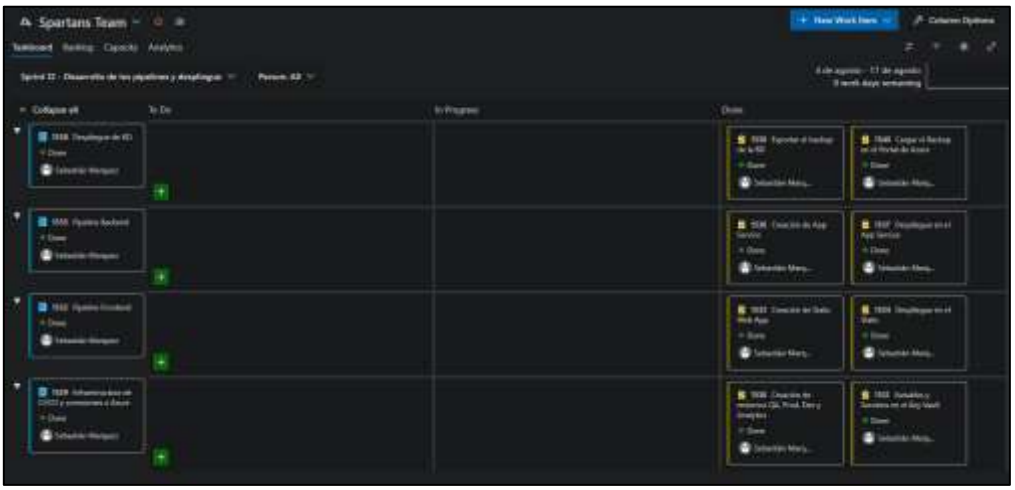


Ilustración 12 Sprint 12

3. Tecnologías Utilizadas

Frontend: React + ecosistema JavaScript

React. Librería de JavaScript para construir interfaces de usuario basadas en componentes y JSX.

Node.js. Runtime que ejecuta JavaScript fuera del navegador.

npm. Gestor de paquetes estándar del ecosistema Node.js para instalar y versionar dependencias del frontend.

Backend: ASP.NET Core (C#) + ORM

ASP.NET Core. Framework multiplataforma, de alto rendimiento y código abierto que se usa para la API del sistema.

Entity Framework Core (EF Core). Mapeo objeto-relacional (ORM) para .NET que soporta consultas.

Base de datos: SQL Server / Azure SQL

Microsoft SQL Server. Motor relacional usado para tablas, vistas y procedimientos.

Azure SQL Database. Servicio PaaS en la nube basado en la versión estable de SQL Server, con administración gestionada por Azure.

Asistente conversacional: Microsoft Copilot Studio

Copilot Studio. Plataforma low-code de Microsoft para crear agentes conversacionales y personalizar copilots.

En este proyecto, el copilot se integra por embed en la web del club y consulta la información disponible para consultas y recomendaciones (fase beta, DEV).

Infraestructura y despliegue en Azure

Azure Static Web Apps. Servicio para publicar aplicaciones web (como React) directamente desde repositorios.

Azure App Service. Plataforma PaaS para hospedar APIs y aplicaciones web sin administrar infraestructura.

Azure SQL Database. Instancia administrada donde se aloja la BD en entorno de desarrollo.

Autenticación y autorización

Firebase Authentication. Servicio gestionado que soporta proveedores federados como Google, se usa para emitir ID tokens y custom claims (rol admin/cliente) que el backend valida como Bearer.

4. Estado del Arte

Easy Cancha

Es una plataforma web y móvil para alquiler de canchas y clases por deporte y club, permite al usuario buscar por disciplina, ver opciones y reservar en pocos pasos. Además, ofrece una función “Match” para encontrar rivales de juego.

Promueve la reserva en línea, integración con calendario y disponibilidad de aplicaciones (iOS/Android), en algunos países habilita pagos en línea o pago en el club.

A nivel regional, opera en varios países de LATAM y tiene presencia en Ecuador con una red amplia de clubes afiliados.

Ilustración 13 Easy Cancha Web

Enlace: https://www.easycancha.com/es-EC/ecuador/alquiler

Ministerio del Deporte – Reserva de instalaciones y Centros Activos

El Ministerio del deporte dispone de un aplicativo web oficial para reservar escenarios/centros activos; la guía en GOB.EC detalla el flujo (solicitud, confirmación, pagos y tiempos límite). Es el referente estatal para canchas e instalaciones públicas.

Enlace: https://www.gob.ec/md/tramites/reserva-instalaciones-actividades-deportivas-recreativas

Ilustración 14 Ministerio del Deporte Web

Reserva de áreas en parques (SETED/Inmobiliar)

Para eventos y actividades recreativas en parques administrados por el Estado se tramita la reserva vía canal en línea o presencial, con pagos y confirmaciones definidos.

Complementa el acceso a espacios deportivos públicos.

Enlace: https://www.gob.ec/setegisp/tramites/reservacion-areas-eventos-deportes-recreacion-0

Ilustración 15 SETED Web

Guayaquil Tenis Club (GTC)

Comunicación oficial del club indica que las reservas se realizan desde su app móvil (Android/iOS), integrando la relación con socios y actividades. Modelo de operación "propietario".

Enlace: https://guayaquiltenisclub.ec/

Ilustración 16 Guayaquil Tenis Club Web

DK Sport Arena (Quito)

Permite la elección de cancha, horario y confirmación con abono. Representa la adopción de reserva online en recintos privados orientados a fútbol sintético.

Enlace: https://dksportarena.com/

Ilustración 17 DK Sport Arena Web

Círculo Militar (Quito)

Publica tarifas y horarios para canchas y establece la reservación previa como requisito; muestra la formalización de reglas y precios para usuarios/socios en un portal institucional.

Enlace: https://www.circulomilitar.ec/servicios/servicios-multiples/

Ilustración 18 Circulo Militar Web

Casos de Uso

Todos los casos se validan en entorno DEV, con datos semilla y pagos simulados; no hay operación productiva por directriz de TI.

Actores primarios:

- **Cliente** (usuario que reserva).
- **Administrador** (gestiona complejos, canchas, parqueaderos, parámetros, reportes).

Actores secundarios (sistemas):

- **Firebase Auth** (emite ID Token y custom claims rol).
- **Pasarela simulada** (registro de pago no real).
- **Chatbot Copilot Studio** (interfaz conversacional embebida).

Código	Nombre	Actor primario	Precondiciones	Flujo principal (resumen)	Excepciones	Postcondición
UC-01	Autenticarse con Google (Firebase)	Cliente / Administrador	Cuenta válida en Firebase	Inicia sesión, el BackEnd verifica el ID Token (Bearer), loguea al usuario el usuario, asigna rol (admin/cliente) y le da acceso	401 token inválido o expirado. 403 rol insuficiente.	Sesión activa con rol que tenga contexto de seguridad cargado
UC-02	Consultar disponibilidad de canchas	Cliente	Autenticación del usuario y se encuentren guardados en la base de datos los complejos, áreas y canchas.	Ingresa parámetros, luego la API consulta y en la interfaz muestra las canchas que se encuentran ocupadas, en mantenimiento y disponibles.	404 no encuentran canchas en la base de datos. 401 token inválido o expirado. **500** error controlado (log en **LogException**).	Usuario conoce las canchas que están disponibles en una franja horaria.

UC-03	Reservar cancha	Cliente	Autenticación del usuario, selección de la cancha que se encuentra disponible.	Enviar solicitud a la API, esta llama a sp_InsertarReserva, este valida los datos de disponibilidad y crea la reservación	**500** error controlado (log en **LogException**). 401 token inválido o expirado.	Reserva creada con exito.
UC-04	Reservar parqueadero	Cliente	Elegir el complejo donde se encuentra el parqueadero.	Verifica si el parqueadero está en mantenimiento, disponible u ocupado. Una vez verificado inserta la reserva del parqueadero con la ejecucion del sp_InsertarReservaParqueadero	**500** error controlado (log en **LogException**). 401 token inválido o expirado. 403 sin autorización.	Reserva de Parqueadero Registrado con éxito.
UC-05	Pagar una reserva del parqueadero	Cliente	Autenticación del usuario, haber reservado exitosamente un parqueadero.	Verifica si fue creado una reserva con la id del usuario. Llama a sp_InsertarPagoParqueadero creando un pago con estado “Pendiente”. Ingresa a una interfaz de pago con tarjeta de crédito donde se enviará el monto calculado con el IVA y las horas de uso desde la tabla de Parqueadero. Usuario realiza el pago y se actualiza su estado a “Pagado”.	**500** error controlado (log en **LogException**). 401 token inválido o expirado. 403 sin autorización.	Pago realizado por el Parqueadero reservado con éxito o mensaje de no se pudo realizar el pago.

UC-06	Pagar una reserva de cancha	Cliente	Autenticación del usuario, haber reservado exitosamente una cancha.	Verifica si fue creado una reserva de canchas con la id del usuario, Llama al sp_InsertaPago, creando un pago con estado “Pendiente”. Ingresa a un interfaz de pago con tarjeta de crédito, donde se enviará el monto calculado con el IVA y las horas de uso desde la tabla de Cancha. Usuario realiza el pago y se actualiza su estado a “Pagado”.	**500** error controlado (log en **LogException**). 401 token inválido o expirado. 403 sin autorización.	Pago realizado por el Cancha reservado con éxito o mensaje de no se pudo realizar el pago.
UC-07	Administración de catálogos (complejos, canchas, parqueaderos, parámetros)	Administrador	Sesión con rol administrador.	El administrador se dirige al panel de la vista protegida, se mostrará una tabla con todos los campos necesarios para ser modificados, eliminados o insertados.	**403** sin rol, **500** error controlado (log en **LogException**). 401 token inválido o expirado.	Elemento actualizado/eliminado con éxito o no se pudo actualizar/eliminar el elemento
UC-08	Consultar pagos	Cliente/ Administrador	Autenticación del usuario, haber realizado con éxito el pago.	Solicita a la vista vw_viewPago, envia la id del usuario, despliga todos los pagos realizados por el usuario en estado “Pendiente” o “Pagado”.	**500** error controlado (log en **LogException**). 401 token inválido o expirado.	Información de los pagos realizados por el usuario o no se encontraron pagos registrados.
UC-10	Chatbot: recomendaciones	Cliente	Chatbot embebido en el FrontEnd, datos	Usuario pregunta por información mediante un chat interactivo y el bot	**500** error controlado (log en	Respuestas del bot en base a datos

			guardados en la base de datos.	le contestara con la información solicitada por el usuario.	**LogException**).	guardados o, no se encontró esa respuesta.

5. Historias de Usuario

US-01 - Autenticación con Google (Firebase)

Como cliente/administrador quiero iniciar sesión con Google para acceder al sistema según mi rol.

Criterios de aceptación (CA):

Dado un ID Token válido cuando envío login entonces el backend verifica, crea usuario e inicia sesión con rol.

Dado token inválido/expirado cuando intento acceder entonces recibo 401 o 403.

US-02 - Autorización por roles (Policies)

Como administrador quiero permisos para gestionar catálogos para crear nuevos complejos, áreas, canchas; como cliente, tendré acceso limitado a la edición, creación y eliminación de elementos, podre hacer reservaciones y pagos.

Criterios de aceptación (CA):

Dado un token con role=admin cuando se accede a rutas admin entonces puedo ingresar; con role=cliente entonces error 403.

Dado ausencia de token cuando accedo entonces error 401.

US-03 - Consultar disponibilidad de canchas

Como cliente quiero ver canchas disponibles, ocupadas o en mantenimiento para elegir una cancha para asistir.

Criterios de aceptación (CA):

Dado la cancha, se consulta, entonces se ve los estados disponibles/ocupado/en mantenimiento.

Dado ausencia de canchas registradas cuando se consulta entonces error 404 (y mensaje).

Errores se registran en LogException (500 controlado).

US-04 - Reservar cancha

Como cliente quiero reservar una cancha para asegurar mi horario.

Criterios de aceptación (CA):

Dado el estado de la cancha, cuando se envía la reserva, entonces el proceso almacenado valida si la cancha no está ocupada y crea la reserva.

Si una cancha se encuentra ocupado o en mantenimiento, no permitirá la reservación.

US-05 - Reservar parqueadero

Como cliente quiero agregar parqueadero a mi reserva de cancha para tener estacionamiento asegurado.

Criterios de aceptación (CA):

Dado los parqueaderos registrados en el sistema, se mostrará los parqueaderos disponibles/en mantenimiento/ocupado.

Si un parqueadero se encuentra ocupado o en mantenimiento, no permitirá la reservación.

US-06 - Pago de reserva de cancha

Como cliente quiero registrar un pago de mi reserva de cancha para completar el flujo de reservación.

Criterios de aceptación (CA):

Dado una reserva cuando pago entonces se ejecuta el sp_InsertarPago registra Pendiente y al confirmar actualiza el pago a Pagado con IVA 15% correcto.

US-07 - Pago simulado de reserva de parqueadero

Como cliente quiero registrar un pago del parqueadero para cerrar mi proceso.

Criterios de aceptación (CA):

Dado reserva de parqueadero, se inicia el proceso de pago registrando un pago en estado “Pendiente”, cuando pago entonces se actualiza estado a Pagado con IVA 15%.

Dado intento sin autorización, pago con error 403.

US-08 - Administración de catálogos

Como administrador quiero crear/editar/eliminar complejos, canchas, parqueaderos y parámetros necesarios para la creación de los elementos mencionados.

Criterios de aceptación (CA):

CRUD exitoso actualiza catálogos, se crean nuevos complejos, áreas, parqueaderos.

US-9 - Chatbot: disponibilidad y recomendaciones

Como cliente quiero consultar al chatbot la disponibilidad de las canchas y parqueaderos, tambien opciones para recibir recomendaciones rápidas.

Criterios de aceptación (CA):

Dado una pregunta válida cuando se interactúa, entonces el bot devuelve disponibilidad o las recomendaciones básicas.

Dado falta de datos cuando se pregunta, entonces informa que no hay resultados.

6. Requisitos del Proyecto

Este sistema se entrega en versión beta no productiva, desplegada sólo en entorno DEV por directriz de TI. Todas las pruebas y evidencias usan datos semilla y pagos simulados, no existen transacciones ni reservas reales. Los requisitos siguientes definen lo mínimo necesario para operar el agendamiento cuando exista inventario definitivo.

6.1. Requisitos Funcionales:

- **RF-01 Autenticación y roles:** Iniciar sesión con Firebase (Google) y validar JWT en backend, los roles admin/cliente determinan permisos.
- **RF-02 Gestión de catálogos**: CRUD de Complejos, Canchas, Parqueaderos y Parámetros.
- **RF-03 Disponibilidad:** Consultar disponibilidad de la cancha, mostrando estados disponibles/ocupado/mantenimiento.
- **RF-04 Reserva de canchas:** Registrar reservas validando la disponibilidad.
- **RF-05 Parqueaderos:** Reservar parqueaderos en el área donde se encuentra cancha de manera independiente.
- **RF-06 Pagos:** Registrar pago de reservas (monto + IVA 15%, estado Pendiente/Pagado, referencia).
- **RF-07 Historial y auditoría:** Consultar historial por usuario y registrar eventos en LogException.
- **RF-08 Chatbot:** Embebido con Copilot Studio para consultas de disponibilidad y recomendaciones.
- **RF-09 Salud y documentación:** Tener un entorno de pruebas.

6.2. Requisitos no Funcionales:

- **RNF-01 Entorno y publicación:** Operación exclusiva en DEV, prohibido uso de producción hasta autorización de TI.
- **RNF-02 Seguridad:** Verificación estricta de JWT (Firebase), Policies por rol, y HTTPS obligatorio.
- **RNF-03 Rendimiento percibido:** Evitar problemas como la doble reserva bajo concurrencia.
- **RNF-04 Trazabilidad y registros:** Manejo uniforme de errores con TRY/CATCH y escritura en LogException.
- **RNF-05 Usabilidad:** Interfaz consistente (estados, mensajes claros, validaciones en cliente), flujos previsibles y accesibles.
- **RNF-06 Mantenibilidad:** Arquitectura monolítica modular y contratos REST versionados.
- **RNF-07 Datos y privacidad:** Uso de datos no productivos (semilla) con una mínima recolección de datos personales debido a que no se procesan pagos reales.
- **RNF-08 Disponibilidad/Despliegue (DEV):** Front en Azure Static Web Apps, API en Azure App Service, BD en Azure SQL y pipelines en Azure DevOps.
- **RNF-09 Documentación:** Libro de Tesis con información sobre la realización del proyecto.

Capítulo II

Construcción del Sistema

1. Base de Datos

1.1. Motor de Base de Datos

El motor de base de datos seleccionado para este proyecto es Microsoft SQL Server, desarrollado por Microsoft. La elección de este sistema gestor se debe principalmente a su versatilidad, estabilidad y escalabilidad, características que lo convierten en una herramienta ampliamente utilizada en entornos empresariales y académicos.

SQL Server ofrece un conjunto robusto de funcionalidades que permiten la gestión eficiente de datos, garantizando seguridad, integridad y disponibilidad de la información. Además, proporciona un lenguaje estructurado de consultas (T-SQL) que facilita la creación de procedimientos almacenados, triggers y funciones, lo que resulta esencial en la implementación de sistemas de información confiables (Microsoft, SQL Server Documentation , 2023).

Una de las razones fundamentales para su selección es su compatibilidad con la nube a través de Microsoft Azure, lo cual facilita la migración o implementación de bases de datos en entornos híbridos o totalmente en la nube. Esto asegura que los sistemas desarrollados puedan adaptarse a las demandas actuales de disponibilidad y escalabilidad que requieren las organizaciones modernas (Microsoft, Azure SQL Database documentation, 2025).

Adicionalmente, SQL Server cuenta con herramientas integradas como SQL Server Management Studio (SSMS), que permiten administrar, monitorear y optimizar el

rendimiento de las bases de datos de manera práctica, lo que contribuye al aprendizaje de los estudiantes y a la correcta implementación de proyectos académicos (Microsoft, 2023).

1.2. Diagrama Entidad Relación

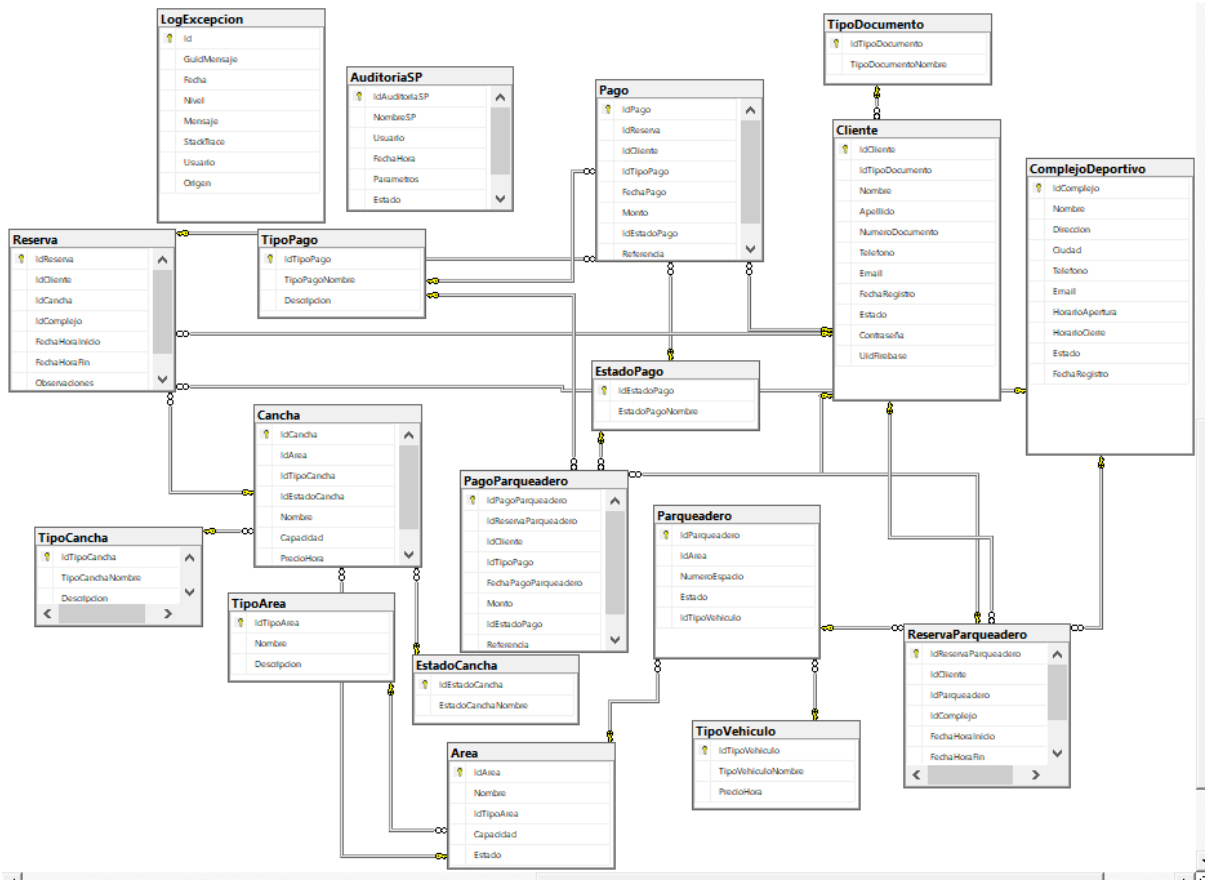


Ilustración 19 Diagrama Entidad Relación Complejo Deportivo Spartans

1.3. Procedimientos Almacenados

En el desarrollo del sistema para la gestión del complejo deportivo, se crearon las tablas necesarias en la base de datos y se implementaron “Procedimientos almacenados”. Estos procedimientos permiten centralizar la lógica de negocio directamente en el motor de base de datos, asegurando mayor eficiencia, seguridad e integridad en las operaciones.

De esta forma, la comunicación entre el BackEnd y la Base de Datos se agiliza, pues cada operación compleja como la inserción, actualización o validaciones se gestiona de manera encapsulada dentro de los procedimientos almacenados. Además, se incluye manejo

de excepciones, registros de auditoría y validaciones de entrada, lo que aporta robustez al sistema.

1.3.1. Procedimientos Almacenados de Inserción

1.3.1.1. sp_InsertarComplejoDeportivo

Este procedimiento permite registrar un nuevo complejo deportivo en el sistema, incluyendo datos como nombre, dirección, ciudad, teléfono, correo electrónico y horarios de apertura/cierre.

- Validaciones: Verifica que los campos obligatorios no estén vacíos, controla la longitud máxima de cadenas y asegura que los datos cumplan las restricciones.
- Auditoría: Registra en la tabla AuditoriaSP los parámetros utilizados y el estado de la operación.
- Errores: Si ocurre una excepción, se guarda en la tabla LogExcepcion.

1.3.1.2. sp_InsertarArea

Inserta una nueva área dentro de un complejo deportivo. Requiere el nombre del área, el tipo, su capacidad y el identificador del complejo asociado.

- Validaciones: Comprueba que no falten campos obligatorios y que la longitud de cadenas no exceda el límite.
- Lógica adicional: Establece el estado de un área como activo (1) por defecto.

1.3.1.3. sp_InsertarCancha

Registra una nueva cancha vinculada a un área y a un tipo de cancha. Se incluyen atributos como capacidad y precio por hora.

- Validaciones: Revisa campos obligatorios, longitudes y consistencia de datos.
- Auditoría y errores: Maneja excepciones y registra los parámetros de la operación en AuditoriaSP y LogExcepcion.

1.3.1.4. sp_InsertarParqueadero

Inserta un espacio de parqueadero asociado a un área, indicando número de espacio, tipo de vehículo y estado (por defecto “Disponible”).

- Validaciones: Controla campos obligatorios y la longitud del número de espacio.
- Auditoría: Igual que en otros SP, registra la operación y posibles errores.

1.3.1.5. sp_InsertarReserva

Registra la reserva de una cancha para un cliente en un intervalo de tiempo.

- Validaciones: Comprueba que las fechas sean válidas, evita solapamientos con reservas existentes en la misma cancha y controla el tamaño de observaciones.
- Lógica clave: Asegura que una cancha no pueda reservarse por dos clientes al mismo tiempo.

1.3.1.6. sp_InsertarReservaParqueadero

Registra la reserva de un espacio de parqueadero en un horario determinado.

- Validaciones: Verifica obligatoriedad de campos, control de fechas y límite de longitud en observaciones.

1.3.1.7. sp_InsertarPago

Gestiona los pagos relacionados con reservas de canchas.

- Cálculo del monto: Obtiene el precio por hora de la cancha y calcula el número de horas de uso, aplicando un IVA del 15%.
- Auditoría: Registra los valores base, IVA y monto total.
- Estados: Inserta el pago con estado inicial “Pendiente”.

1.3.1.8. sp_InsertarPagoParqueadero

Se encarga de calcular y registrar los pagos asociados a reservas de parqueadero.

- Cálculo del monto: Toma el precio base del tipo de vehículo y calcula el monto con IVA del 15%.
- Validaciones: Verifica existencia de datos obligatorios y controla longitud de la referencia.
- Estados: Inserta el pago como “Pendiente” hasta que se confirme.

1.3.1.9. sp_InsertarCliente

Registra un nuevo cliente en el sistema y devuelve el identificador generado. El procedimiento recibe los datos personales, contacto, la contraseña y opcionalmente el UidFirebase.

Validaciones.

- Obligatoriedad: tipo de documento, nombre, apellido, número de documento, email y contraseña. Si faltan, retorna @IdCliente = -2 y lanza error.
- Longitudes máximas: nombre (≤128), apellido (≤128), número de documento (≤32), teléfono (≤32), email (≤128). Si exceden, retorna @IdCliente = -3.

Auditoría y manejo de errores.

- Registra NombreSP, Usuario, FechaHora y los parámetros en AuditoriaSP.
- En caso de excepción, escribe en LogExcepcion dentro de un bloque TRY/CATCH y propaga error.

1.3.2. Procesos Almacenados de Actualización

1.3.2.1. sp_ActualizarCancha

Actualiza área, tipo, estado, nombre, capacidad y precio/hora de la cancha.

- Validaciones. Obligatoriedad de campos y longitud de Nombre.
- Reglas. UPDATE Cancha y normalización a mayúsculas; auditoría y manejo de errores.

1.3.2.2. sp_ActualizarArea

Modifica nombre, tipo, capacidad, estado y complejo del área.

- Validaciones: Revisa obligatorios y longitud de Nombre.
- Reglas: UPDATE Area con Nombre en mayúsculas; auditoría y el LogException.

1.3.2.3. sp_ActualizarCliente

Actualiza datos personales y de contacto del cliente (incluye contraseña).

- Validaciones: Campos obligatorios y límites de longitud por campo.
- Reglas: UPDATE Cliente; texto en mayúsculas (contraseña sin transformación); auditoría y manejo de errores.

1.3.2.4. sp_ActualizarComplejoDeportivo

Actualiza datos generales, horarios y estado del complejo.

- Validaciones: Campos obligatorios y longitudes; verifica existencia del ID.
- Reglas: UPDATE ComplejoDeportivo, normaliza textos, registra auditoría.

1.3.2.5. sp_ActualizarPago

Actualiza tipo de pago y referencia del pago de reservas.

- Validaciones: IdTipoPago obligatorio y Referencia ≤ 64.
- Reglas: UPDATE Pago con Referencia en mayúsculas; auditoría y manejo de errores.

1.3.2.6. sp_ActualizarPagoParqueadero

Actualiza tipo de pago y referencia del pago de parqueadero.

- Validaciones: IdTipoPago obligatorio y Referencia ≤ 64.
- Reglas: UPDATE PagoParqueadero con Referencia en mayúsculas; auditoría y logging.

1.3.2.7. sp_ActualizarParqueadero

Actualiza área, número de espacio, estado y tipo de vehículo de un parqueadero.

- Validaciones: Campos obligatorios; longitud de NumeroEspacio y Estado.
- Reglas: UPDATE Parqueadero con normalización a mayúsculas; auditoría y logging.

1.3.2.8. sp_ActualizarReserva

Actualiza cliente, cancha, complejo, horarios y observaciones de la reserva.

- Validaciones: Obligatoriedad de IDs y fechas; límite de Observaciones.
- Reglas: UPDATE Reserva con observaciones en mayúsculas; auditoría y manejo de errores.

1.3.2.9. sp_ActualizarReservaParqueadero

Actualiza cliente, parqueadero, complejo, horarios y observaciones de la reserva de parqueadero.

- Validaciones: Obligatoriedad de IDs y fechas; límite de Observaciones.
- Reglas: UPDATE ReservaParqueadero con observaciones en mayúsculas; auditoría y errores.

1.3.3. Vistas de las Tablas

Las vistas se implementan para ofrecer un modelo de lectura estable y seguro que abstrae la complejidad de las tablas y sus uniones. Presentan exactamente los campos que

consume la capa de presentación, evitando exponer información sensible y reduciendo la lógica de ensamblaje en el BackEnd.

- Aporte al Front (Método GET)

Para los listados de tablas o tarjetas, las vistas entregan datos ya unidos y normalizados, con nombres de columnas coherentes y listas para renderizar. Esto simplifica los controladores GET porque solo requieren un SELECT sobre la vista y devolver el resultado al Front.

- Aporte al Front (Método GET by id)

Para pantallas de detalle devuelve toda la información enriquecida de una entidad específica mediante un filtro por identificador. El BackEnd resuelve el endpoint GET /{entidad}/{id} con una única consulta simple, sin rearmar múltiples JOIN ni cálculos repetidos.

Beneficios.

- Abstracción y seguridad: exponen solo lo necesario para la visualización del usuario.
- Rendimiento y simplicidad: Menos lógica en controladores, las vistas concentran JOINS, alias y campos calculados.
- Mantenibilidad: cambios en tablas base no rompen el Front si la vista conserva el mismo contrato, esto quiere decir que tiene que respetar los nombres y variables declaradas en la Base de Datos.

1.4. Auditoria y Seguridad

Centralizar la trazabilidad de cada ejecución de los Procedimientos Almacenados que involucra quién, cuándo y qué parámetros fueron enviados. Registra de forma consistente los

errores ocurridos durante la ejecución. Esto agiliza el diagnóstico, soporta el cumplimiento y mejora la observabilidad del sistema.

Tablas involucradas

1. AuditoriaSP

- Qué guarda: nombre del Procedimientos Almacenados, usuario de base de datos que lo ejecuta, fecha/hora de la zona local, parámetros enviados al ejecutarlo.
- Cuándo se escribe: al finalizar con éxito la operación del Procedimientos Almacenados tras la ejecución de INSERT/UPDATE.

Notas clave

- Se usa SYSTEM_USER y GETDATE() AT TIME ZONE 'UTC' AT TIME ZONE 'SA Pacific Standard Time' para estandarizar la zona horaria.
- Los Procedimientos Almacenados construyen una cadena @Parametros con los valores recibidos para el diagnóstico posterior.

2. LogExcepcion

- Qué guarda: GuidMensaje, fecha, nivel, el mensaje de error, el nombre del procedimiento donde se dio este error, el usuario desde donde se ejecutó y la línea de origen.
- Cuando se escribe: dentro del bloque CATCH ante cualquier excepción; se detecta el error utilizando RAISERROR.

Buenas prácticas de uso del GUID

- El GuidMensaje impreso en el CATCH permite correlacionar el error en logs de aplicación con el registro en LogExcepcion.

Flujo típico

1. Ejecución exitosa

   - Validaciones → operación de negocio (INSERT/UPDATE) → registro en AuditoriaSP con nombre del SP, usuario, fecha y parámetros.

2. Ejecución con error

   - TRY/CATCH captura la excepción → inserta en LogExcepcion con detalles técnicos → RAISERROR para notificar a la capa llamadora.

Cifrado o Hasheo de contraseñas en la tabla Cliente

Por seguridad, las contraseñas no deben almacenarse en texto plano. En la versión actual de los Procedimientos Almacenados, el valor recibido en @Contraseña maneja un cifrado en la tabla Cliente, tanto al insertar como al actualizar ante el riesgo de exposición a datos sensibles del mismo.

El Método de "Cifrado de Contraseñas" se desarrolló dentro de la Lógica del BackEnd, quien maneja además de este método, otros métodos de Seguridad que se describirán más Adelante.

2. BackEnd

2.1. Servicios

Los Services encapsulan el acceso a datos con ADO.NET y con (SqlConnection/SqlCommand) y exponen operaciones GET / GET by id sobre vistas para lectura, y CREATE/UPDATE/DELETE/PATCH delegando la lógica principal a Procedimientos Almacenados. Todas las consultas y Procedimientos Almacenados usan parámetros (AddWithValue) para prevenir Inyecciones SQL en las formulario o vistas del FrontEnd. Las eliminaciones se resuelven como soft delete cuando aplica, en este caso hay

Services que manejan una eliminación completa de la base de datos y en otros casos hace una actualización del Estado, este caso principalmente se manejo en entidades importantes para el funcionamiento de la aplicación.

2.1.1. Autenticación y Seguridad

2.1.1.1. AuthService (Firebase + custom claims)

El Login Verifica el ID token de Firebase, extrae uid y email, asegura el custom claims que añade un rol al usuario logueado y vuelve a emitir un nuevo ID token aplicando un custom token contra la API de Identity Toolkit que requiere de ApiKey proporcionada por Firebase. Devuelve AuthLoginResult con Uid, Email, Role, IdToken, RefreshToken, ExpiresIn y ClientId.

El Logout permite revocar todos los refresh tokens del usuario.

2.1.1.2. Autorización por roles (policies + handler)

RoleRequirement: Define un requisito con la lista de roles permitidos, valida que exista al menos uno y conserva comparación case-insensitive.

RoleAuthorizationHandler: Evalúa el requisito contra los claims de rol presentes tanto en ClaimTypes.Role como en el claim crudo "role". Si alguno coincide, autoriza la operación.

2.1.1.3. Program.cs (bootstrap de seguridad y DI)

- Firebase Admin: Inicializa con Service Account.
- DI de servicios: Registra los services del dominio.
- Policies: Define AdminOnly, ClienteOnly y AdminOrCliente usando RoleRequirement.

- JWT Bearer: Configura autenticación contra securetoken.google.com/{projectId} y en el evento OnTokenValidated vuelve a verificar revocación con Firebase.
- CORS: Habilita la policy AllowFrontend con orígenes tomados de la configuración.
- Chequeo de BD. Al iniciar, realiza un smoke test de conexión a SQL Server.

Orden de middlewares. UseAuthentication() → UseAuthorization() y aplicación de CORS a controladores.

2.2. Controladores

Son la puerta de entrada del sistema. Reciben las peticiones del FrontEnd, y devuelven respuestas claras en formato JSON. No hacen la lógica compleja, solo coordinan y ordenan el trabajo.

2.2.1. Modelo de Operación de los Controladores

1. Escuchan una ruta declarada en el FrontEnd.
2. Revisan quién está intentando usarla con las reglas de roles, principalmente los roles de admin (Administrador), cliente (Usuario).
3. Validan que los parámetros enviados por el usuario son correctos para para ejecutar las acciones declaradas en Services.
4. Services analiza y procesa la petición enviada desde los Controllers.
5. Services responden al FrontEnd con la operación delegada.

2.2.2. Cómo ayuda al Front.

- Entrega datos listos para mostrar en las pantallas.
- Mantiene un contrato estable: Los nombres y estructuras que ve el Front no cambian, aunque por dentro se ajusten vistas o Procedimientos Almacenados.
- Unifica mensajes de éxito y error para una mejor experiencia de usuario.

2.2.3. Tipos de acciones.

- GET: listas o detalle insertados en las Tablas de la Base de Datos.
- POST: crear nuevos ítems dentro de la Base de Datos
- PUT: actualizar los ítems creados previamente.
- PATCH: cambios puntuales de estado. Se manejo Principalmente en PagoController, ParqueaderoController, ReservaController y ReservaParqueaderoController.
- DELETE: eliminación de datos en la Base de Datos o cambio de estado cuando corresponde, esto depende con la tabla que tendrá la interacción.

Seguridad

- Permite el acceso a la interacción con Services a los usuarios que contienen un rol asignado en su token. Al manejar 2 tipos de roles, hay ciertas acciones que el usuario no podrá concretarlas, mientras que el Administrador tendrá acceso a todas las funcionalidades del sistema.

- Al manejar un sistema de Tokens, la información nunca quedará expuesta y podrá el usuario interactuar de manera segura con el sistema.

- Si algo sale mal al ejecutar el Controlador, se explicará el problema de manera clara y para la facilidad de lectura del usuario.

2.2.4. Beneficio General

Ordenan el flujo "Front → Controlador → Service → Base de datos (SPs)", manteniendo el sistema limpio, seguro y fácil de mantener.

3. FrontEnd

3.1. Componentes

3.1.1. Vistas del Usuario

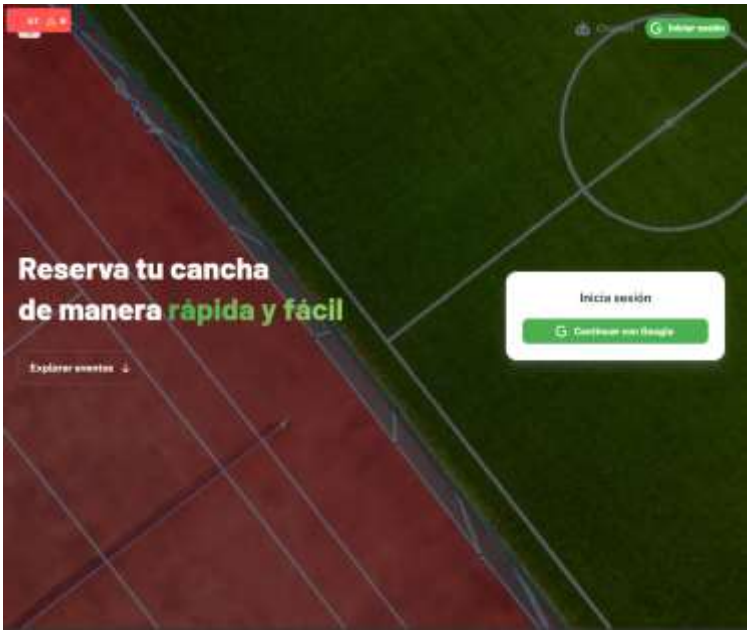


Ilustración 20 Landing Page Web

Ilustración 21 Footer Web

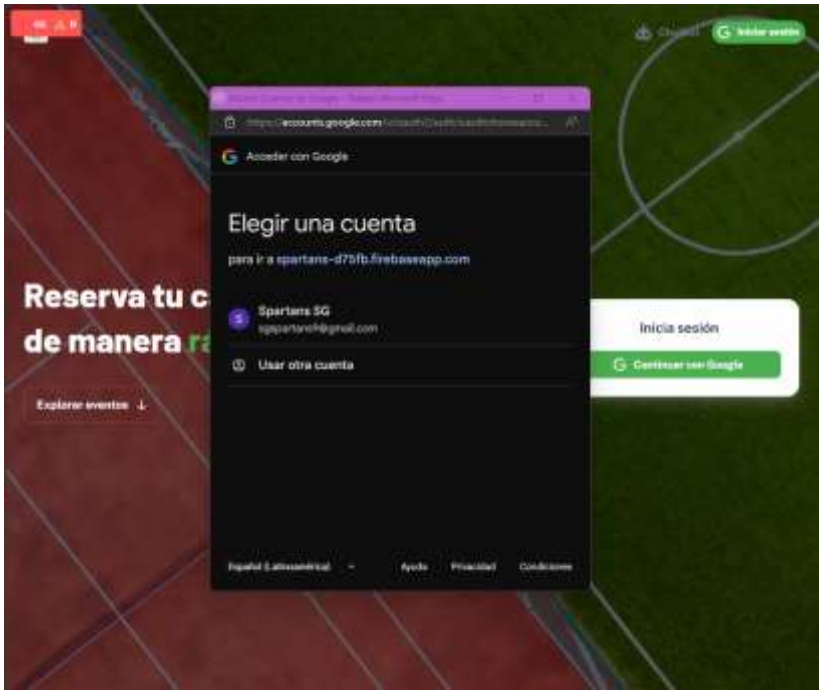


Ilustración 22 Login Google

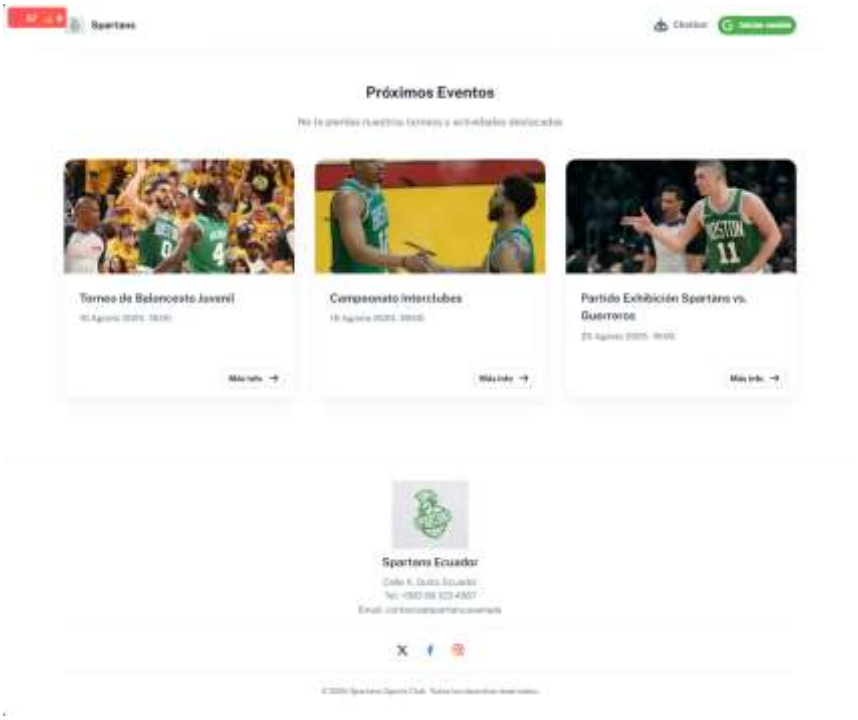


Ilustración 23 Información de Próximos Eventos

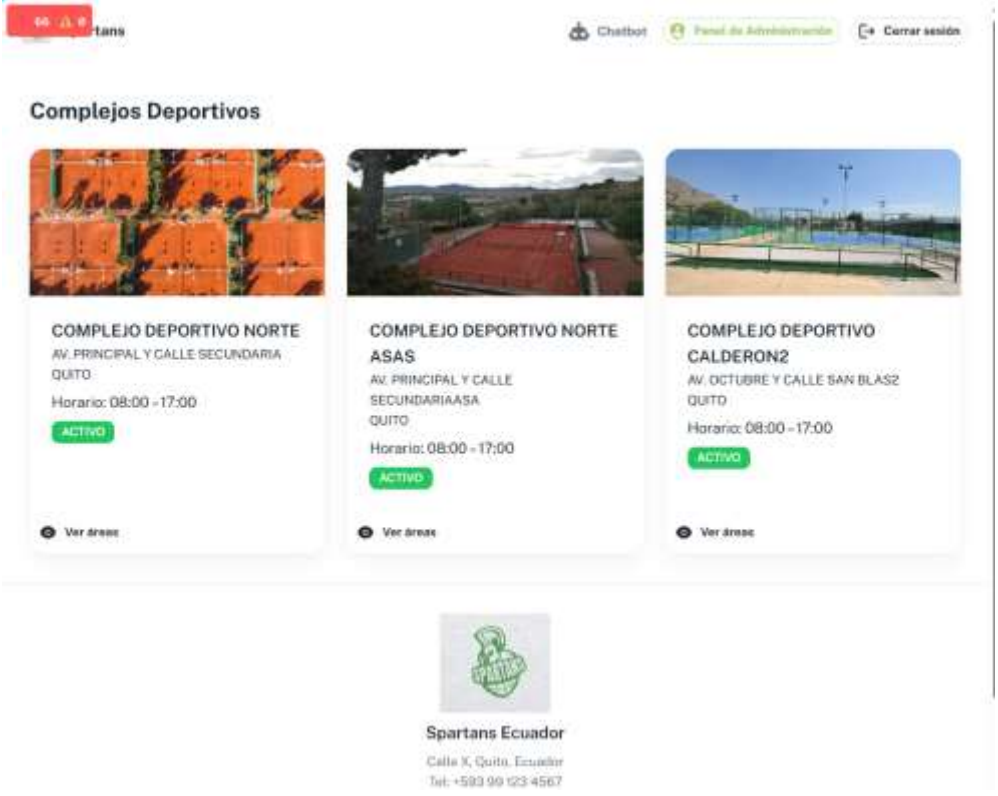


Ilustración 24 Index Web

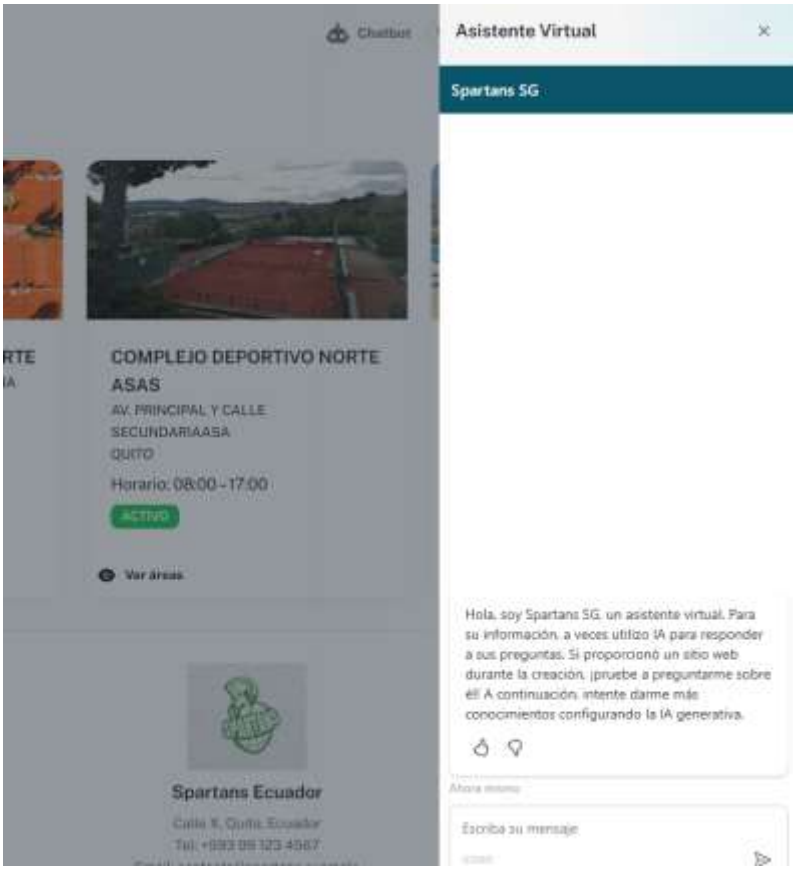


Ilustración 25 Interfaz de ChatBot

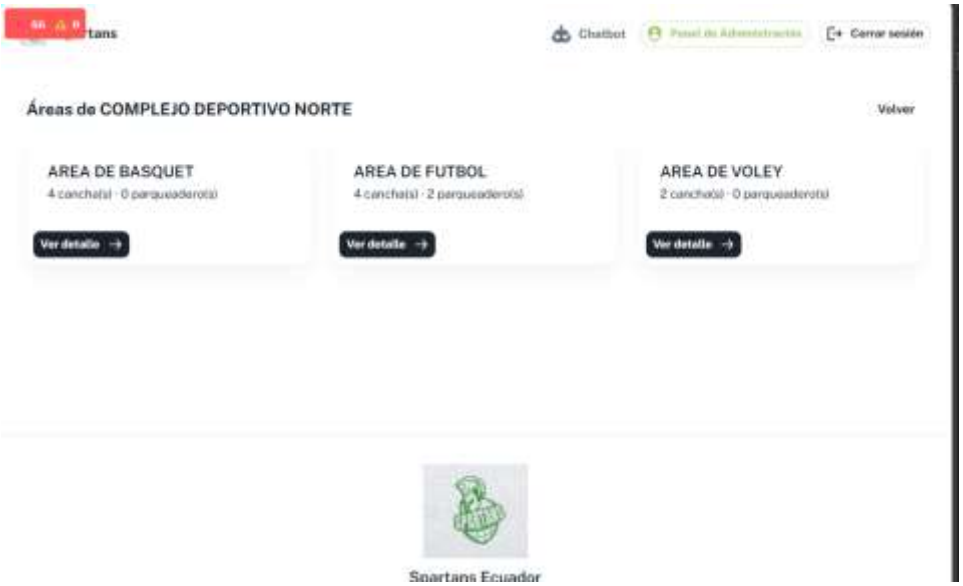


Ilustración 26 Area index

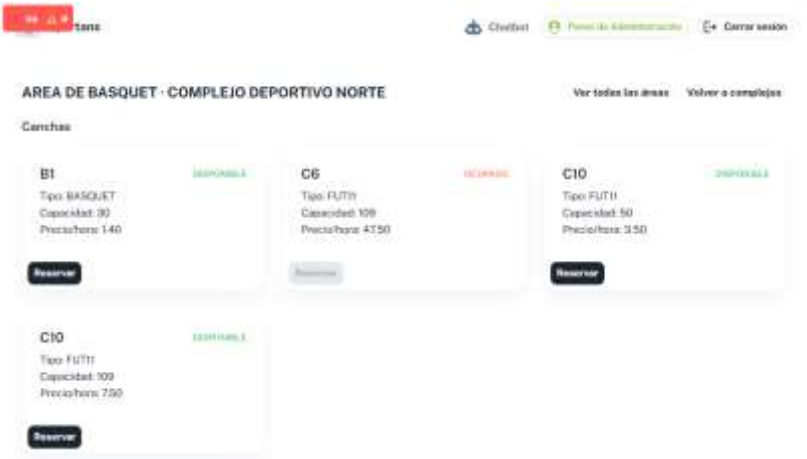


Ilustración 27 Canchas Index

Ilustración 28 Reserva

Ilustración 29 Pago Reserva

Ilustración 30 Select Tipo de Pago

Ilustración 31Hall de Pagos index

Ilustración 32 Splash Pago Realizado

3.1.2. Vistas Protegidas

Ilustración 33 Vista Protegida Panel de Administración

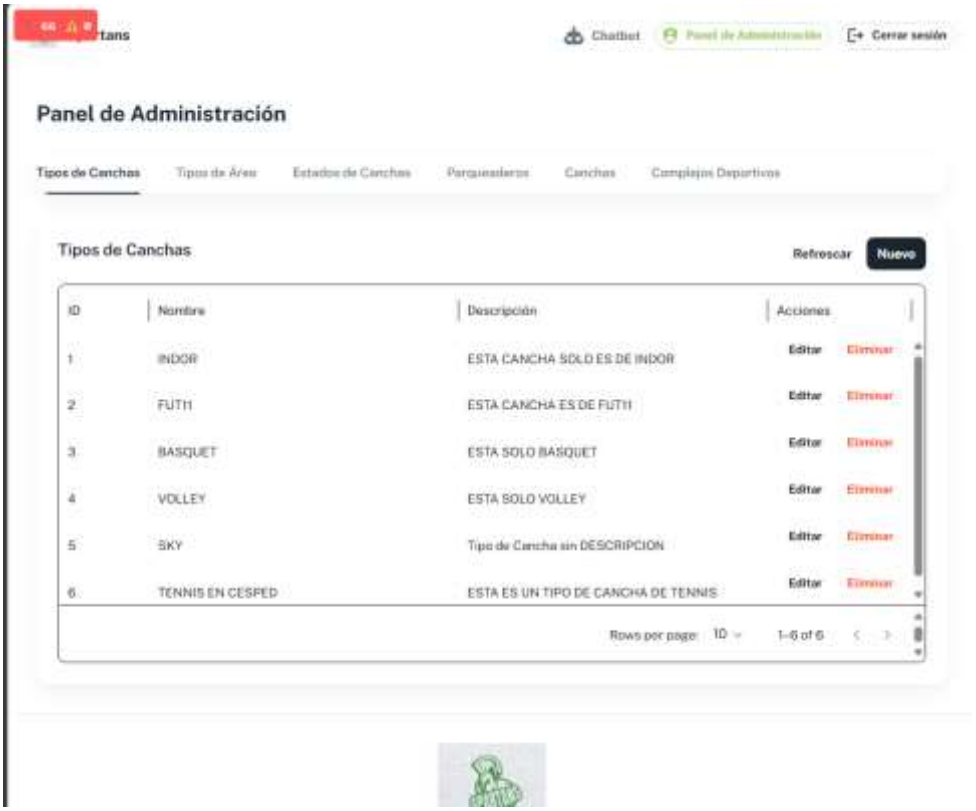


Ilustración 34 Vista Protegida Tipo de Canchas Edit Panel

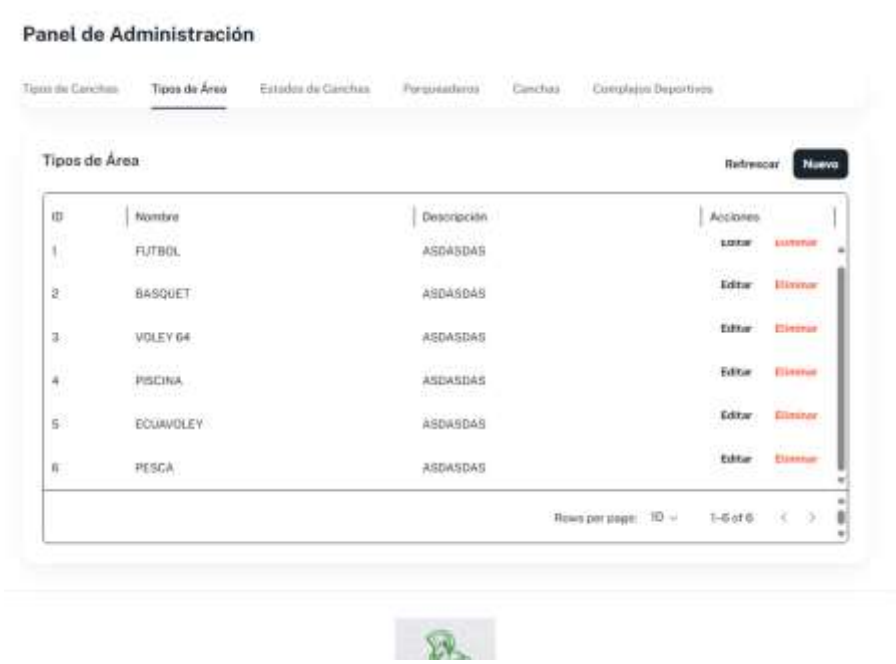


Ilustración 35 Vista Protegida Tipo de Áreas Edit Panel

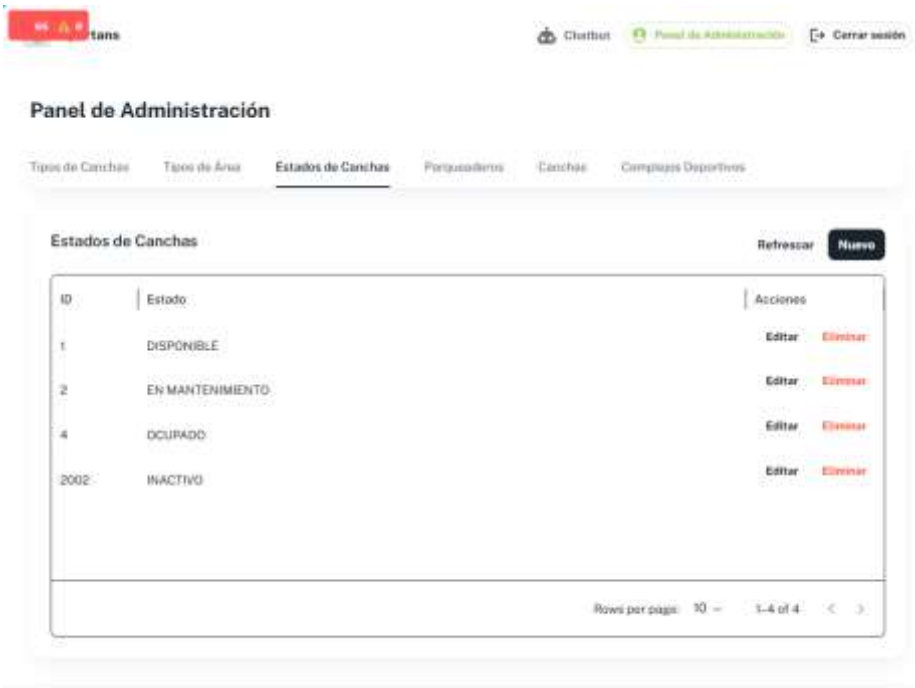


Ilustración 36 Vista Protegida Estado de Canchas Edit Panel

Ilustración 37 Vista Protegida Parqueadero Edit Panel

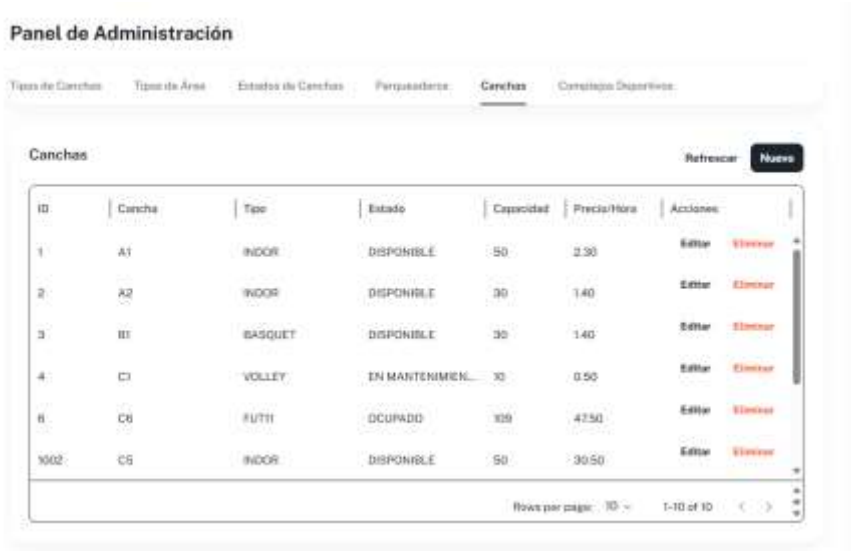


Ilustración 38 Vista Protegida Canchas Edit Panel

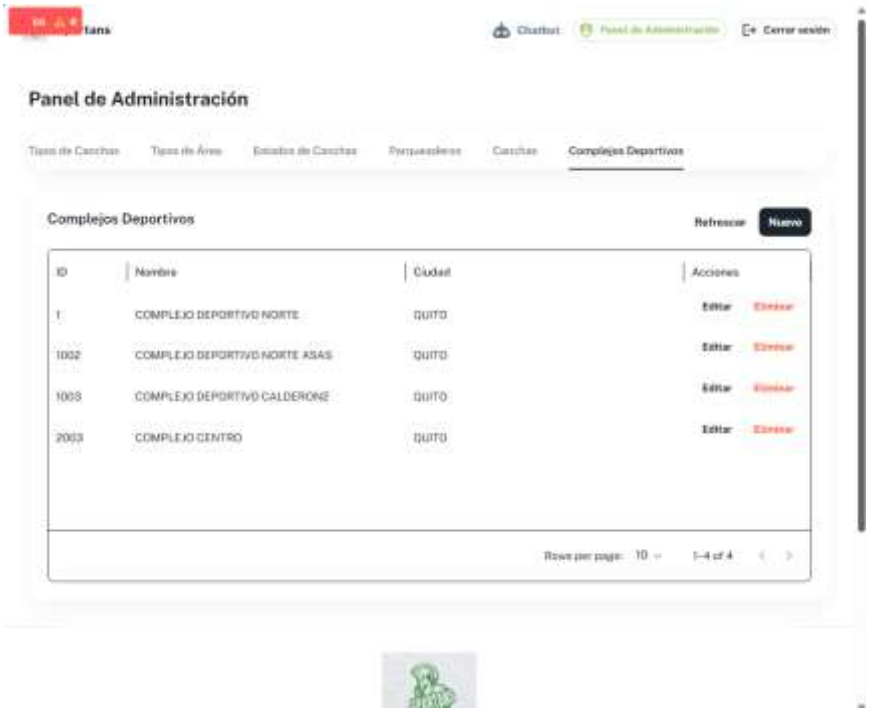


Ilustración 39 Vista Protegida Complejos Deportivos Edit Panel

4. Alojamiento de la Aplicación

4.1. Azure DevOps

Azure DevOps se utilizó como plataforma integral para la gestión del código y la automatización del despliegue. En un solo entorno se centralizo los repositorios, los flujos de trabajo de integración y entrega continua, y las variables y secretos necesarios para publicar la aplicacion Web.

Control de versiones con Git Flow.

El código se almacenó en Azure Repos aplicando el patrón Git Flow, lo que permitió trabajar de forma ordenada y colaborativa. Se utilizó las siguientes ramas.

- Ramas principales: main (producción) y develop (integración).

Trabajo diario:

- feature/* para nuevas funcionalidades, creadas desde develop y fusionadas por Pull Request con revisión de código.
- Correcciones urgentes: hotfix/* desde main, que luego se integran también en develop.

Este esquema facilitó la trazabilidad de cambios, las code reviews y la vinculación con las tareas desarrolladas en Boards siguiendo la estructura de la Metodología Agil Scrum.

4.1.1. Pipelines

Se definieron pipelines YAML que se ejecutan automáticamente al hacer push o abrir PR, con etapas separadas para compilar, probar, empaquetar y desplegar:

- Back-End
    - Build And Tests: Restauración de dependencias, compilación y pruebas automatizadas.
    - Artifact: Publicación del paquete listo para desplegar.
    - Deploy: Liberación a un servicio de hospedaje en Azure App Service en entornos Desarrollo y Producción, con aprobaciones manuales entre etapas cuando aplica.
- Front-End
    - Build: Instalación de dependencias y construcción del proyecto.
    - Artifact: Empaquetado de la carpeta de build.
    - Deploy: ublicación en un servicio web para alojar la Web para que el usuario pueda interactuar con ella.

Ilustración 40 Pipeline reference 1

Ilustración 41 Pipeline reference 2

Ilustración 42 Pipeline reference 3

Capítulo III
Pruebas y Estabilización

1. Pruebas Unitarias

Las pruebas unitarias se realizaron con el objetivo de validar el correcto funcionamiento de los métodos desarrollados en el BackEnd, verificando que cada componente responda adecuadamente a distintos escenarios de entrada y salida. Para ello se utilizó Postman, enviando parámetros de prueba y analizando la respuesta del sistema en cada caso.

1.1. Resultados de las pruebas

Login Firebase

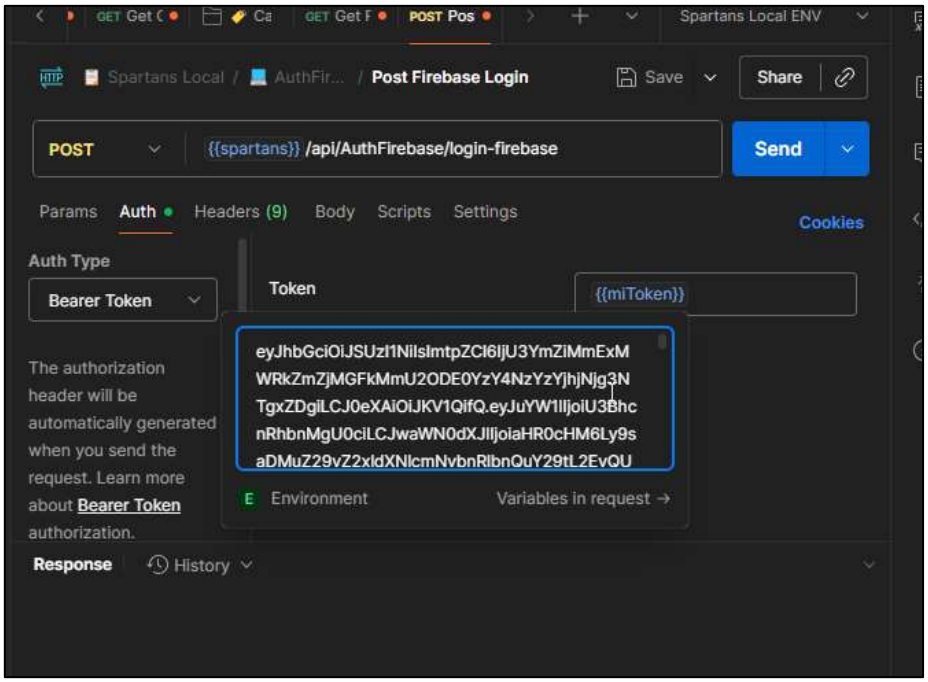


Ilustración 43 Prueba Unitaria 1

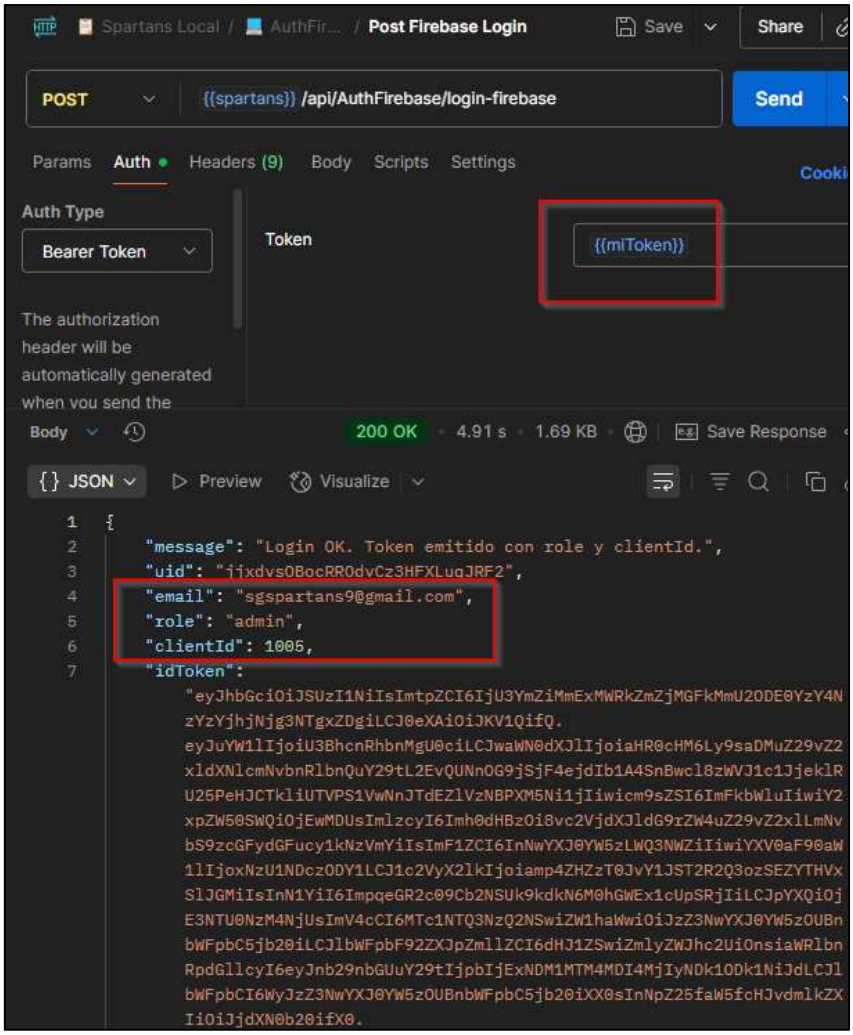


Ilustración 44 Prueba Unitaria 2

Register Firebase

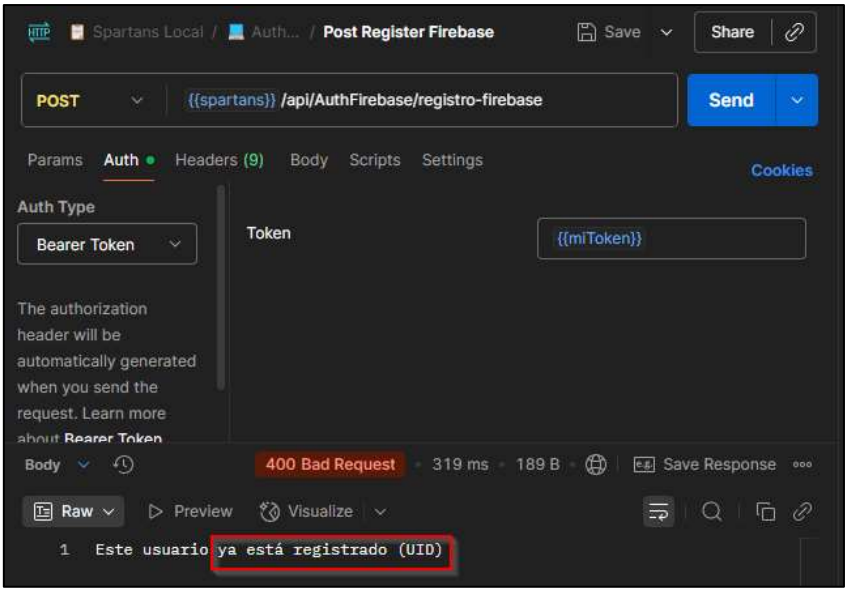


Ilustración 45 Prueba Unitaria 3

Admin Promote

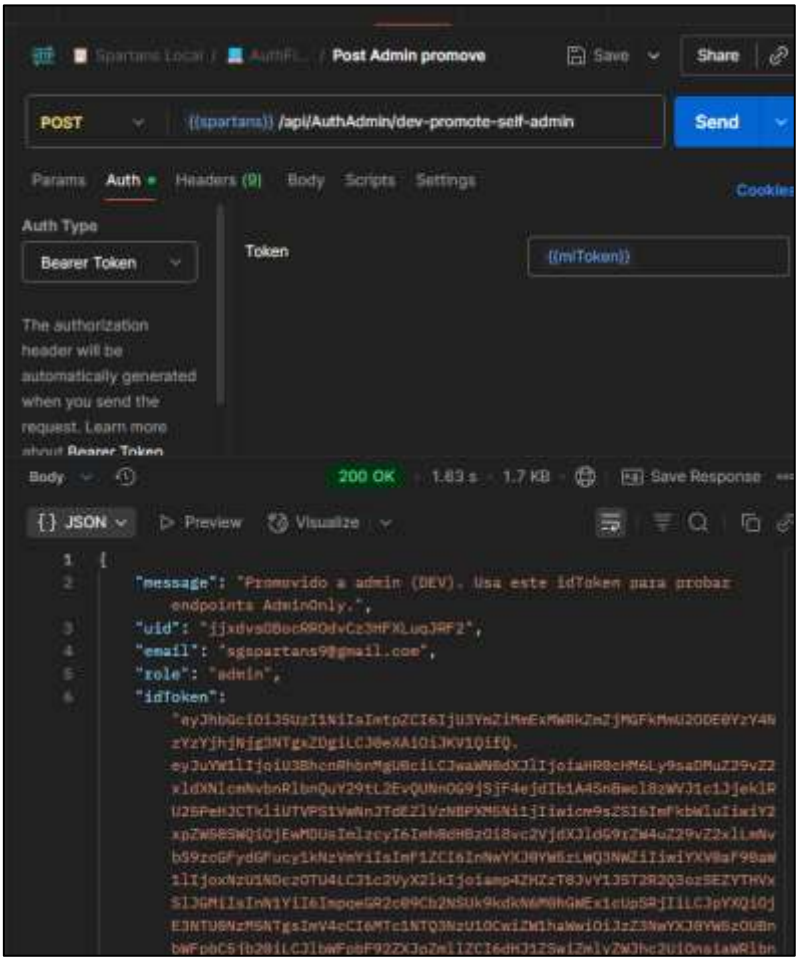


Ilustración 46 Prueba Unitaria 4

Áreas

- **Get Areas**

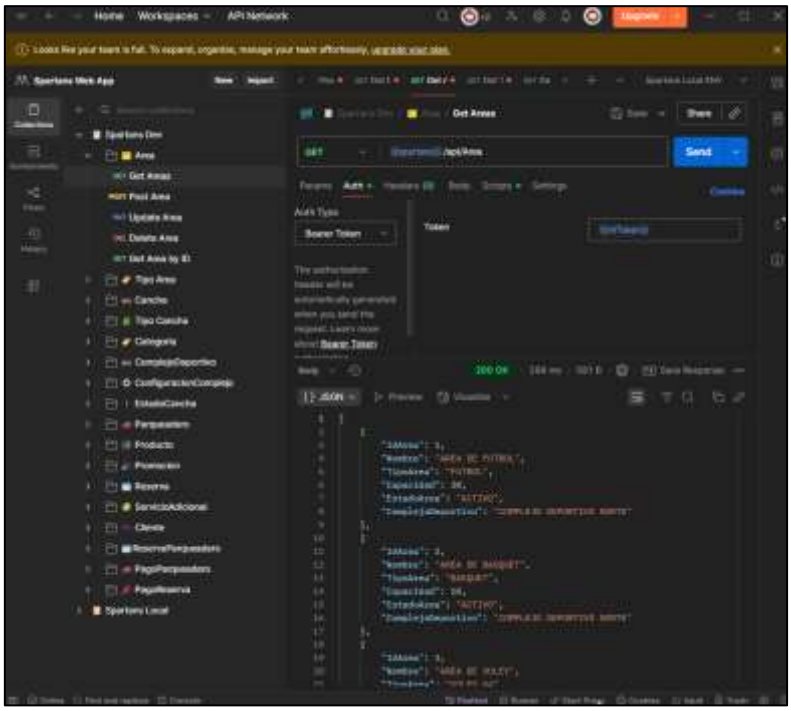


Ilustración 47 Prueba Unitaria 5

- **Post Area**

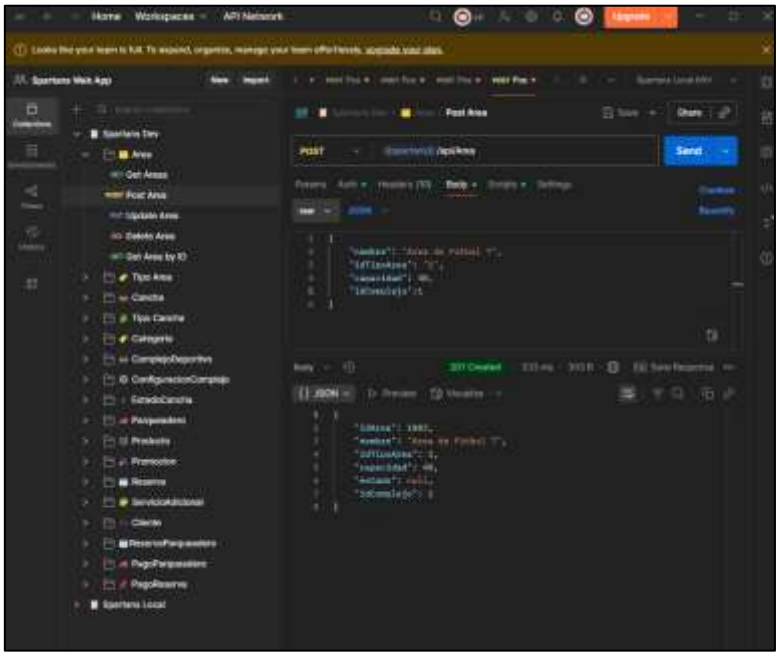


Ilustración 48 Prueba Unitaria 6

- **Update Area**

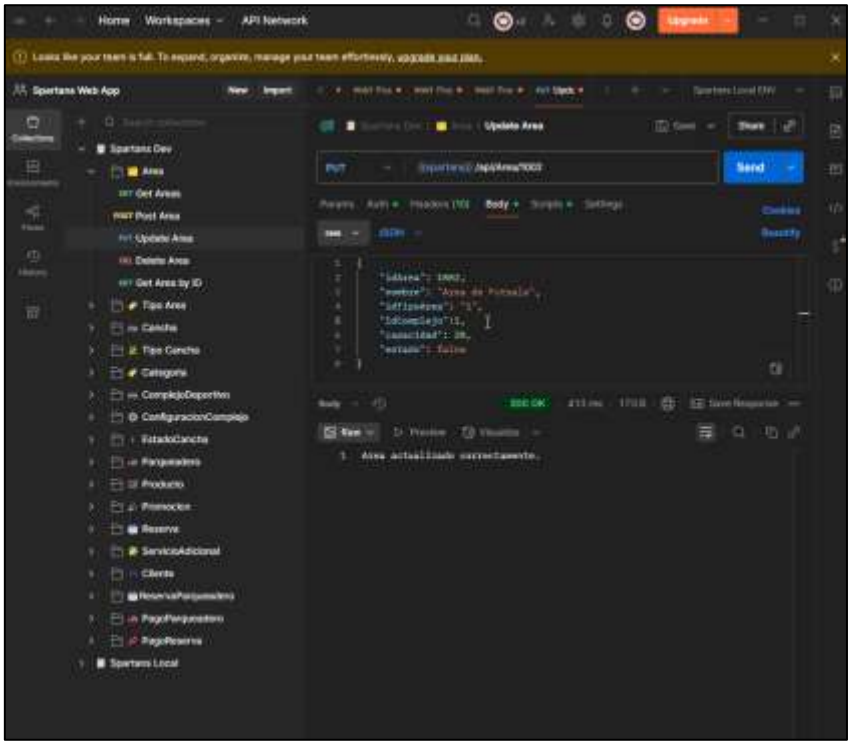


Ilustración 49 Prueba Unitaria 7

- **Get Area by ID**

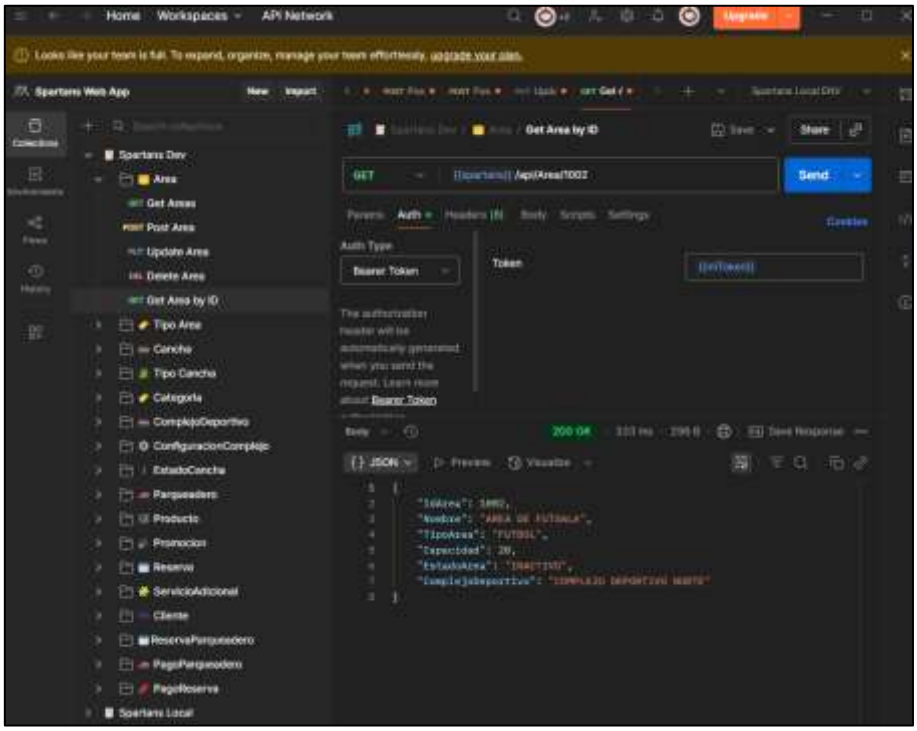


Ilustración 50 Prueba Unitaria 8

- **Get Tipo Area**

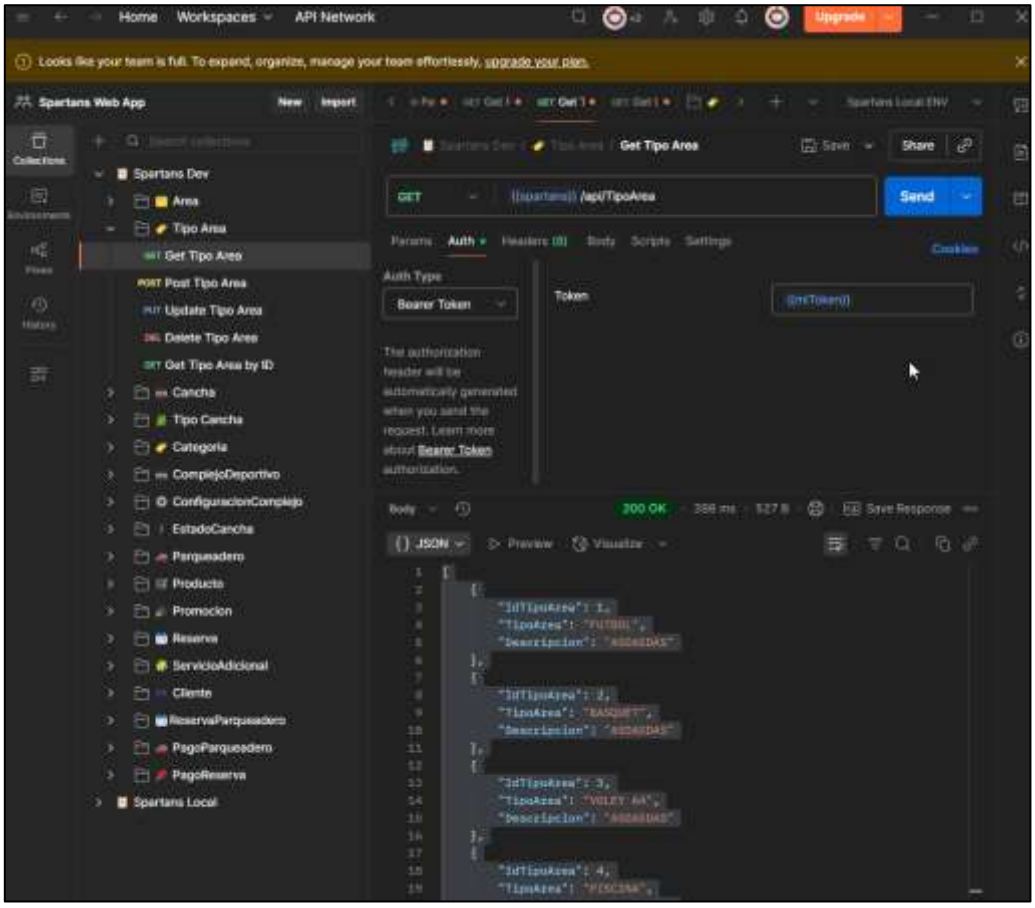


Ilustración 51 Prueba Unitaria 9

- **Post Tipo Area**

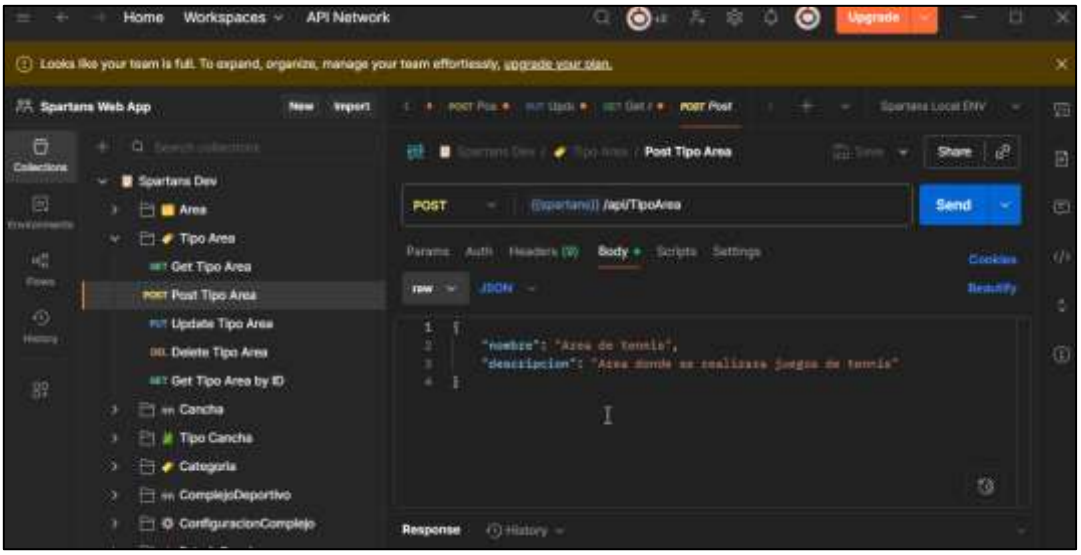


Ilustración 52 Prueba Unitaria 10

- **Update Tipo Area**

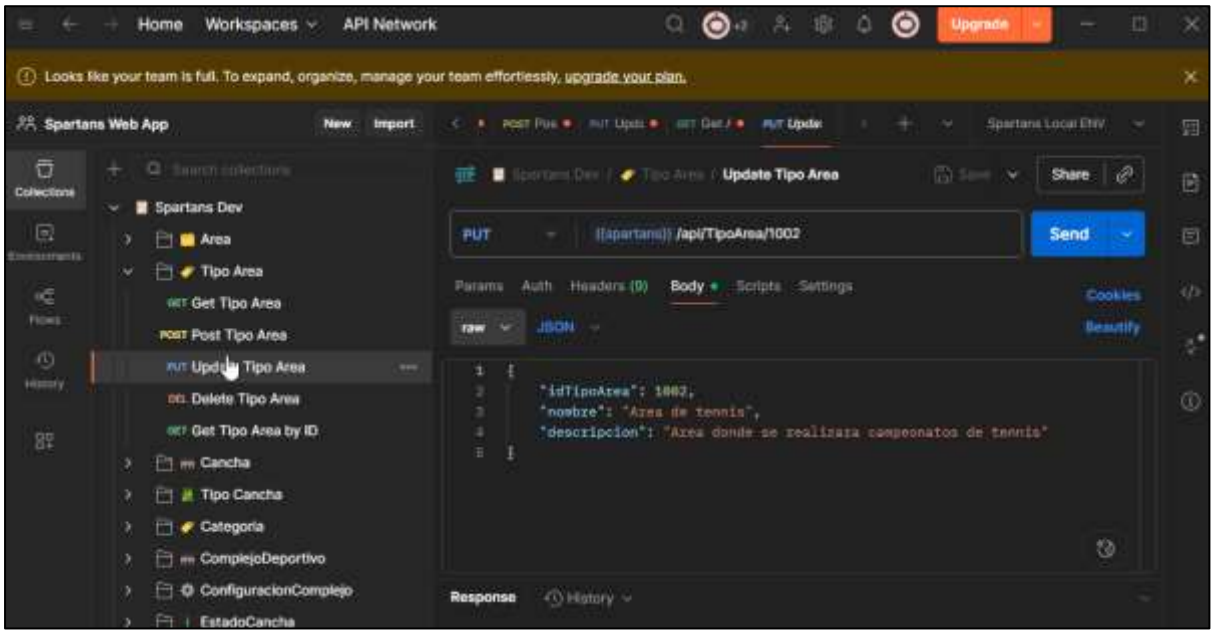


Ilustración 53 Prueba Unitaria 11

Canchas

- **Get Cancha**

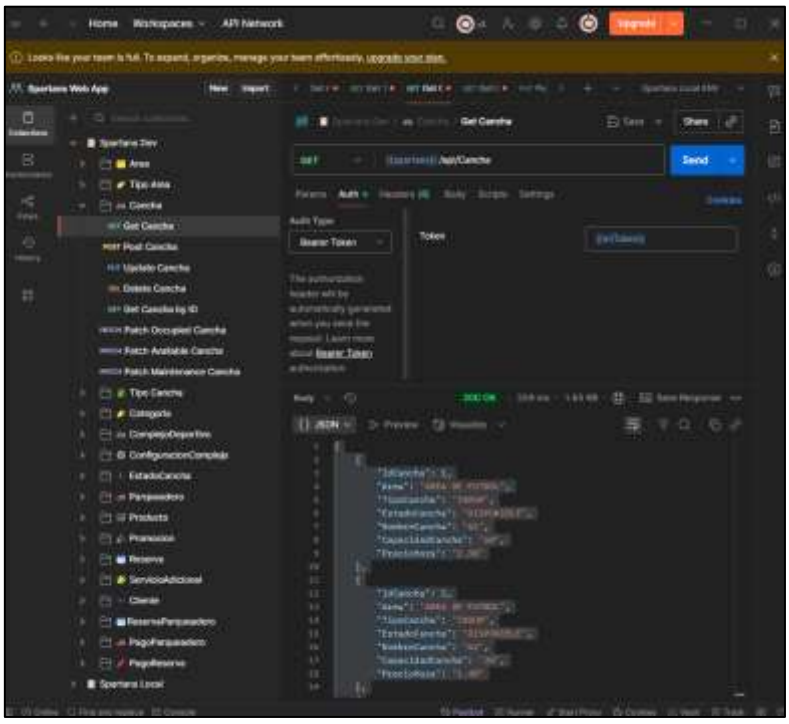


Ilustración 54 Prueba Unitaria 12

- **Post Cancha**

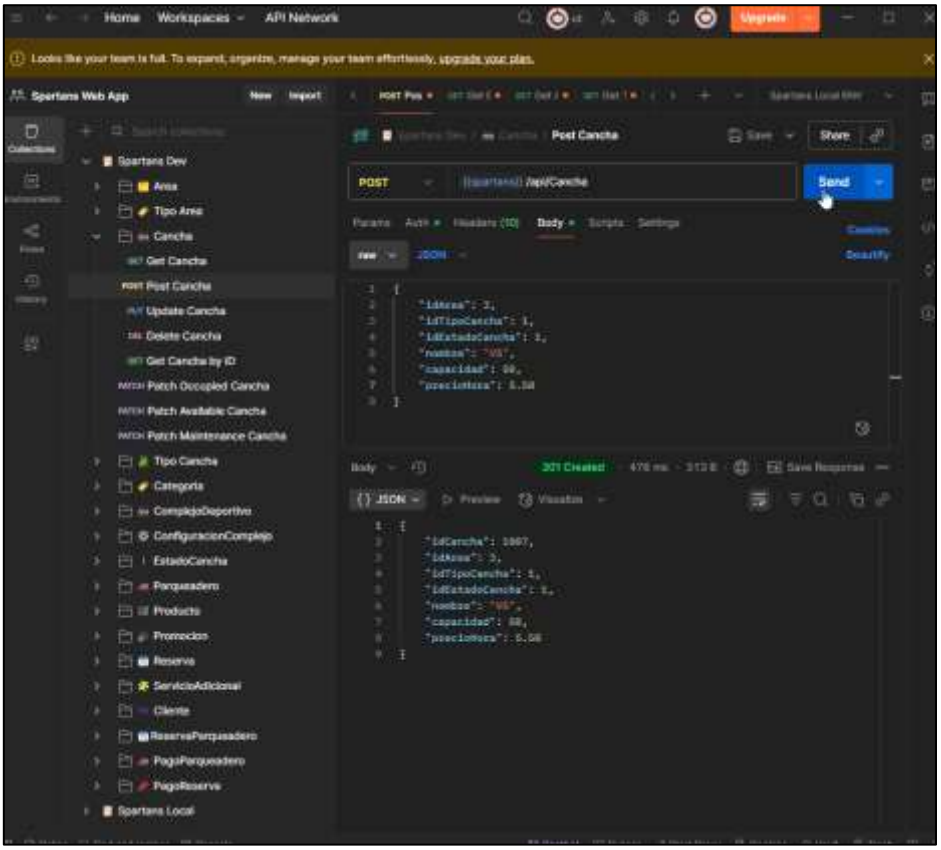


Ilustración 55 Prueba Unitaria 13

- **Update Cancha**

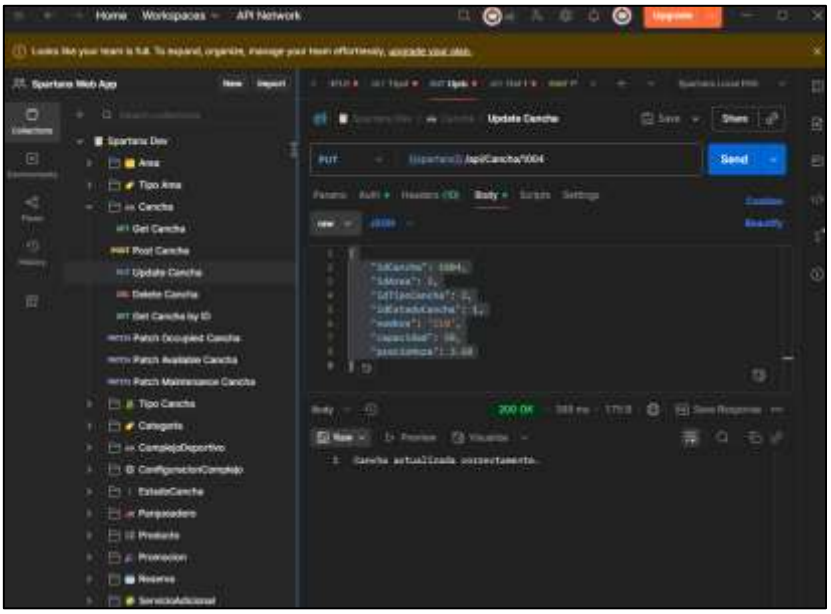


Ilustración 56 Prueba Unitaria 14

- **Get Cancha by ID**

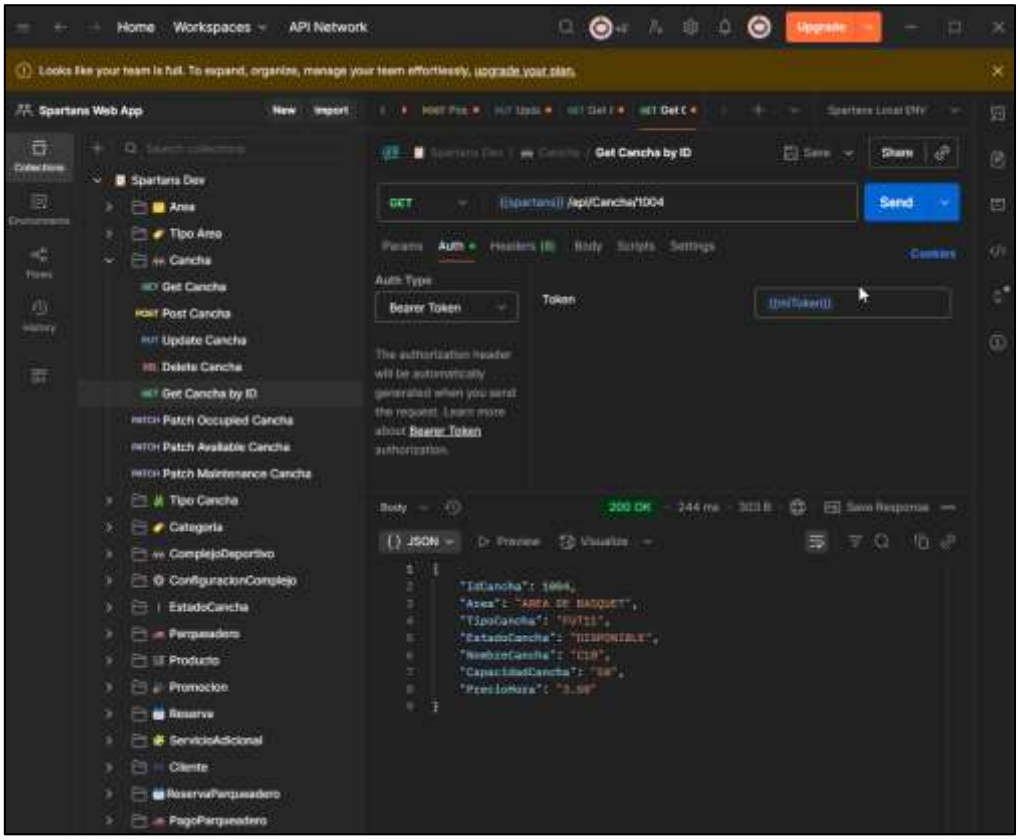


Ilustración 57 Prueba Unitaria 15

- **Patch Cancha**

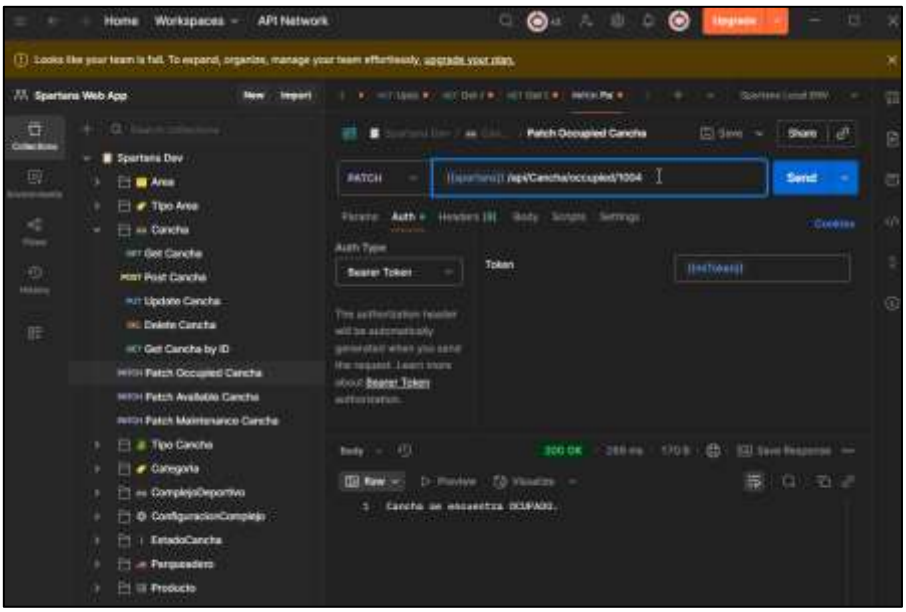


Ilustración 58 Prueba Unitaria 16

- **Patch Avaible Cancha**

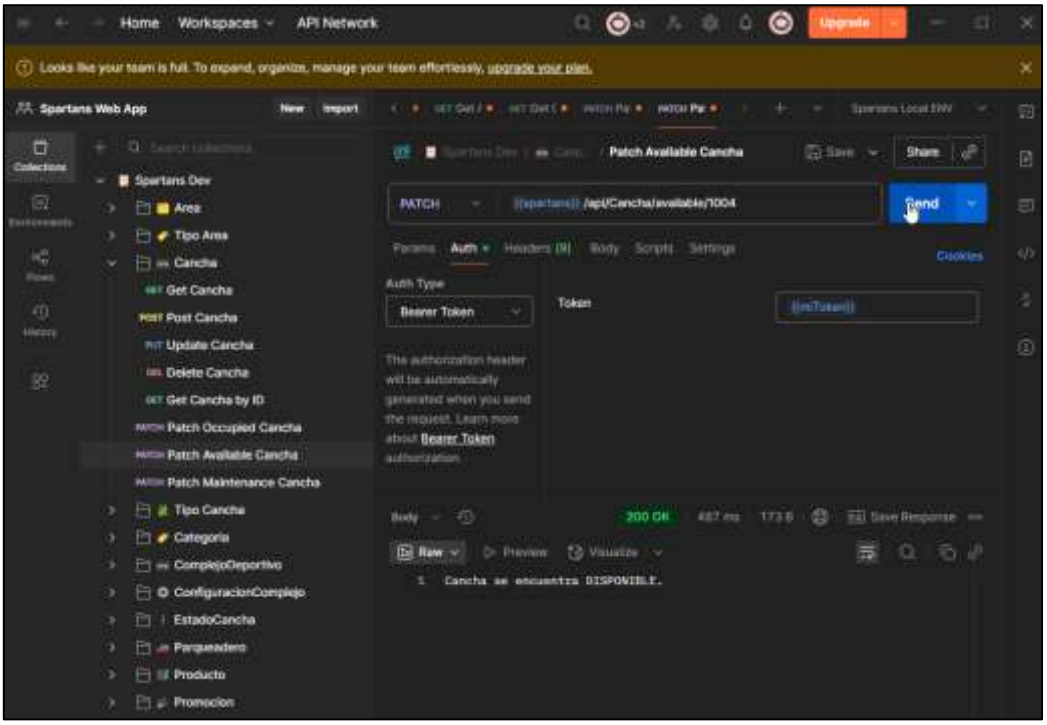


Ilustración 59 Prueba Unitaria 17

- **Patch Maintenance Cancha**

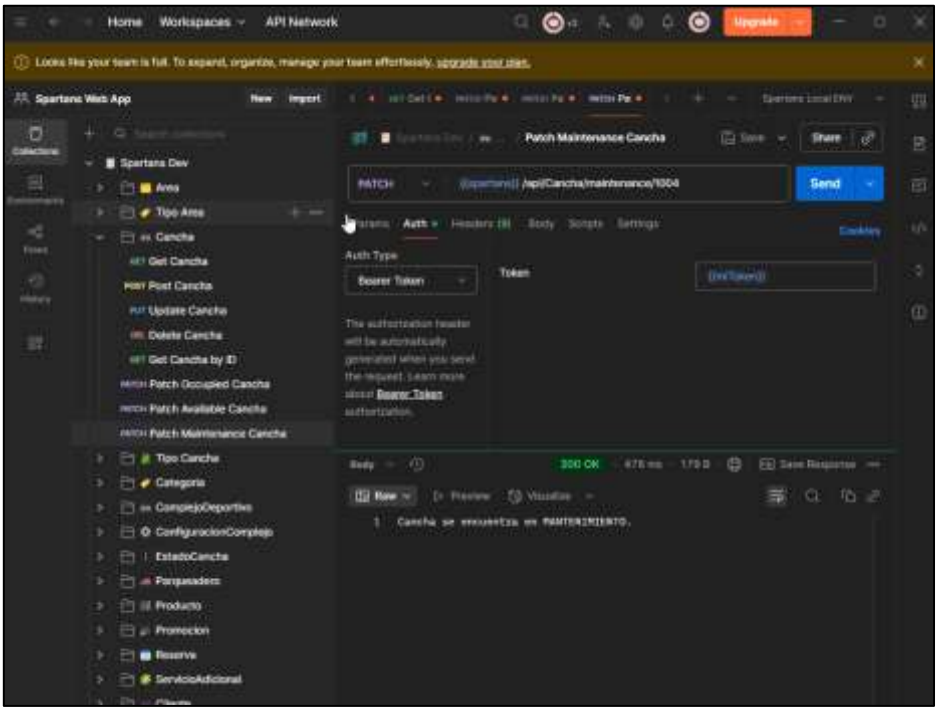


Ilustración 60 Prueba Unitaria 18

- **Get Tipo Cancha**

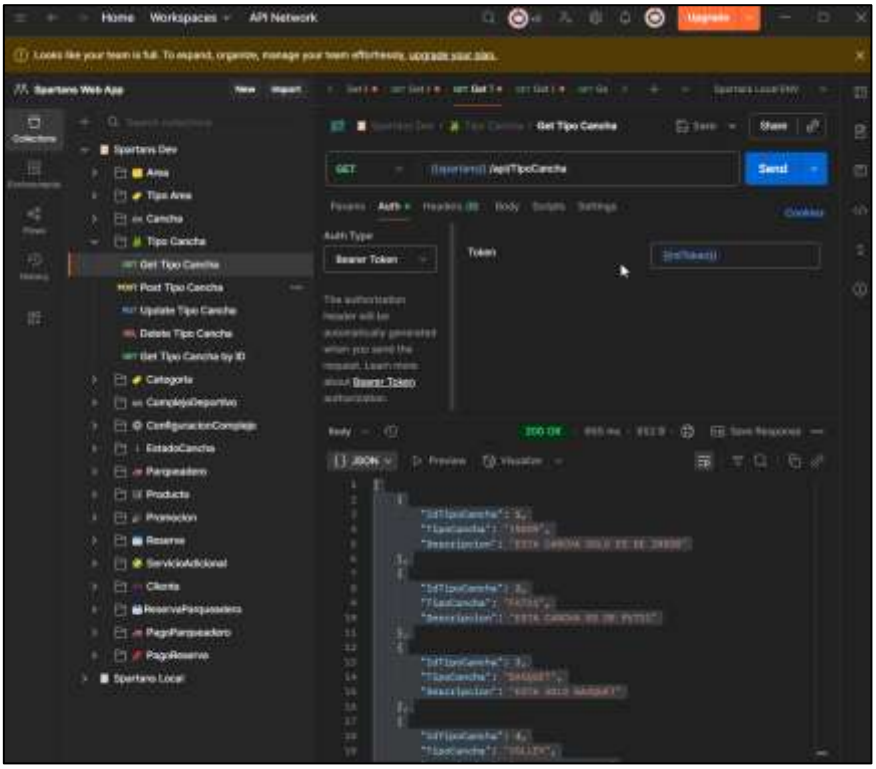


Ilustración 61 Prueba Unitaria 19

- **Post Tipo Cancha**

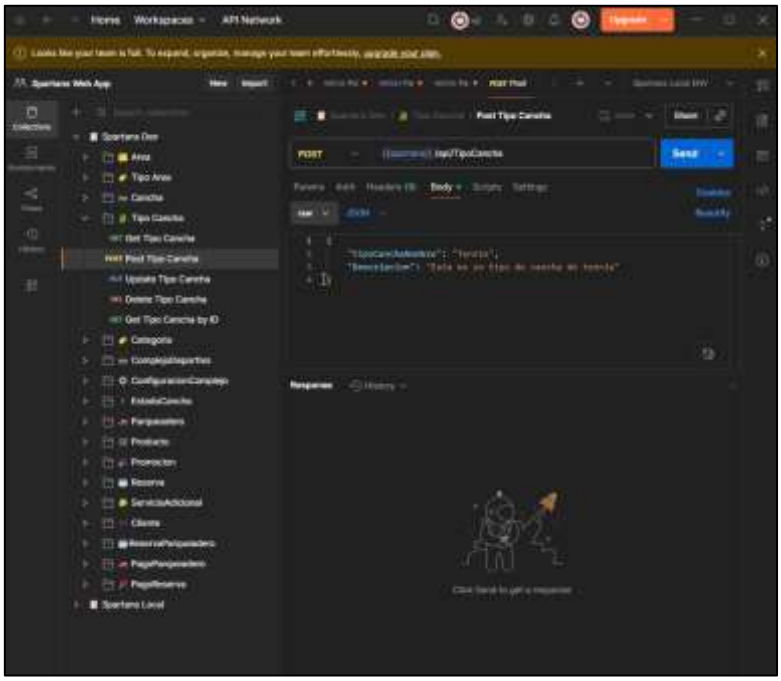


Ilustración 62 Prueba Unitaria 20

- **Update Tipo Cancha**

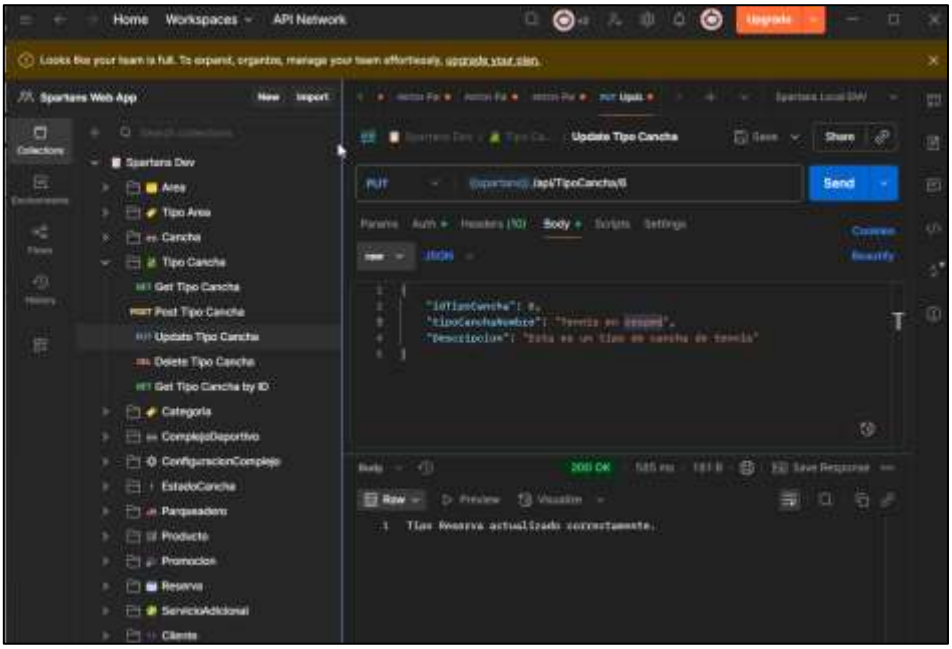


Ilustración 63 Prueba Unitaria 21

- **Get Tipo Cancha**

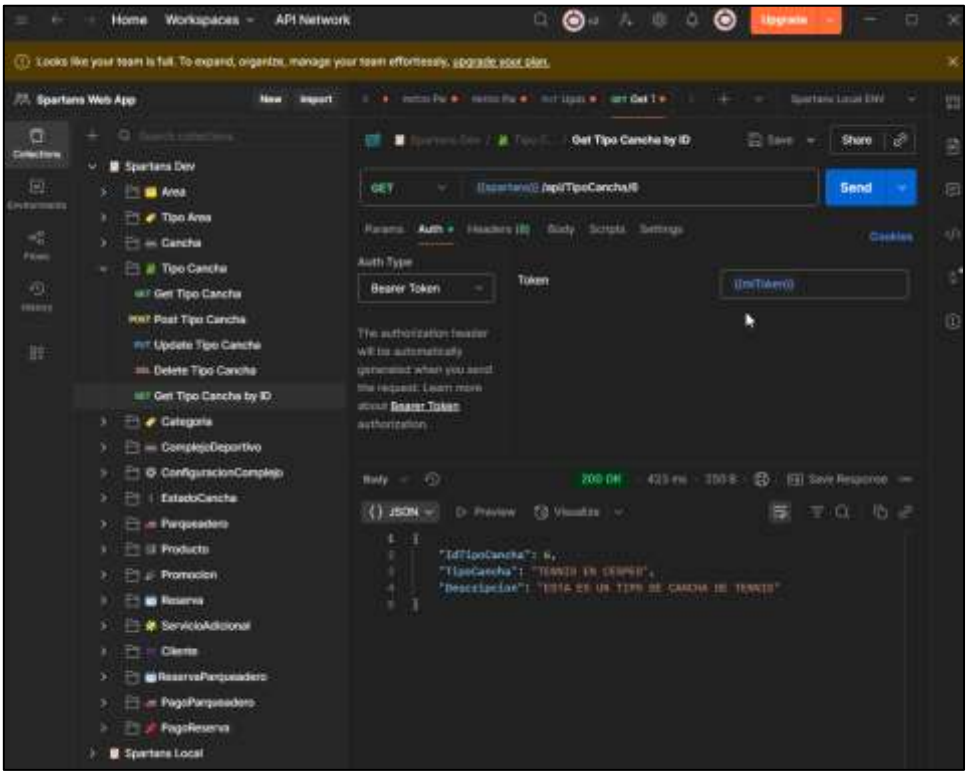


Ilustración 64 Prueba Unitaria 22

Complejos Deportivos

- **Get Complejo**

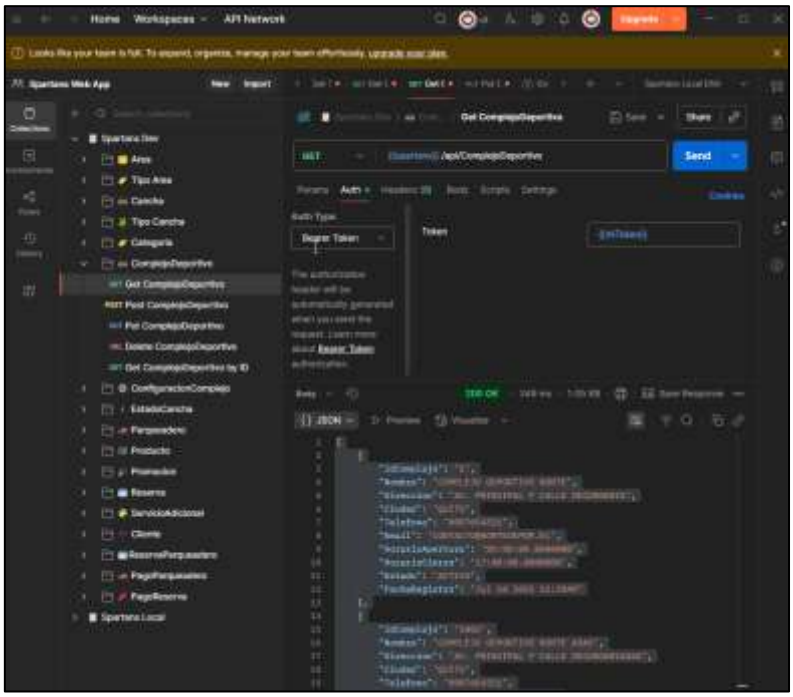


Ilustración 65 Prueba Unitaria 23

- **Post Complejo**

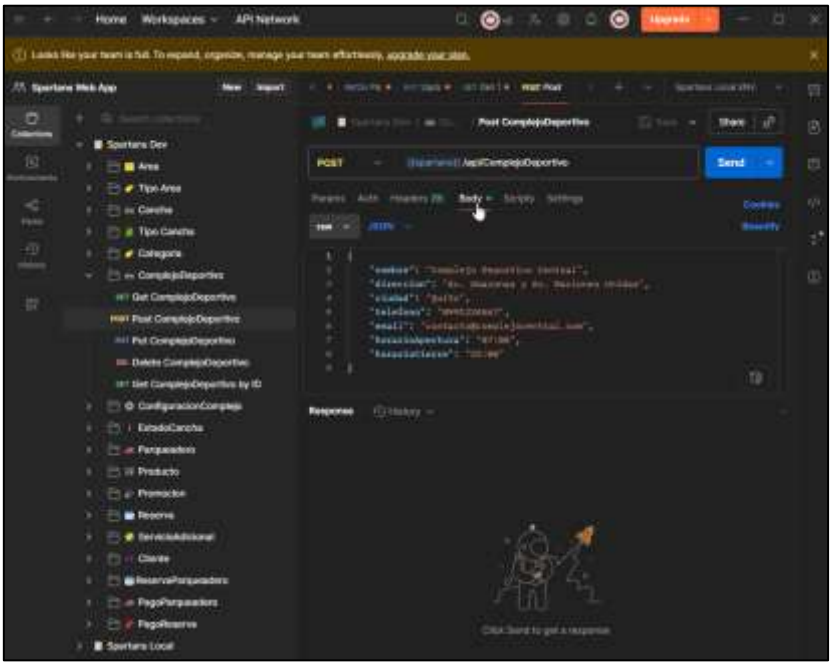


Ilustración 66 Prueba Unitaria 24

- **Put Complejo**

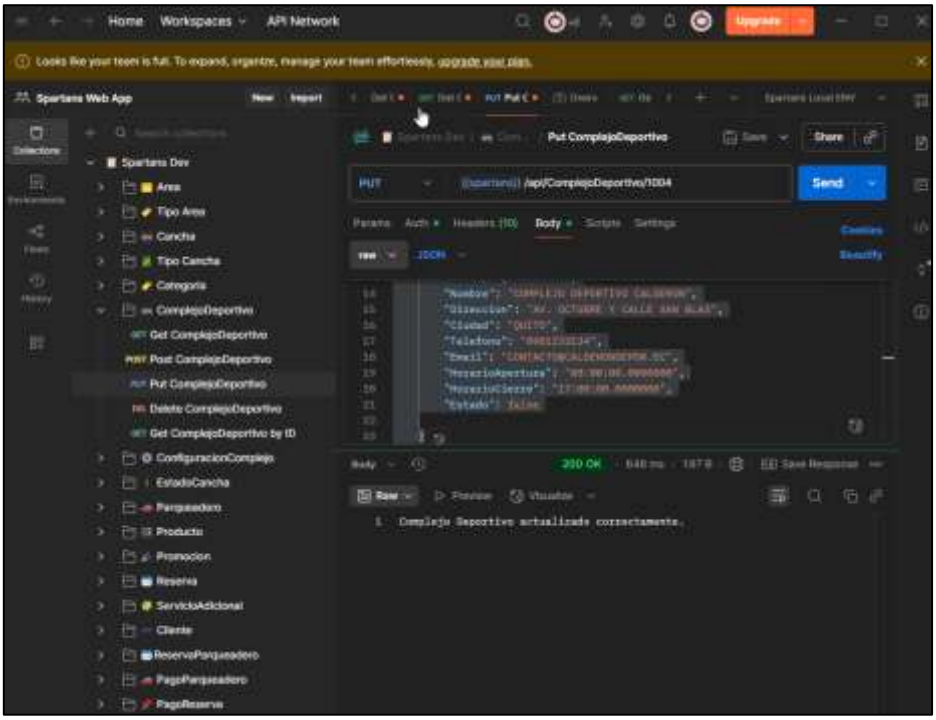


Ilustración 67 Prueba Unitaria 25

- **Get Complejo by ID**

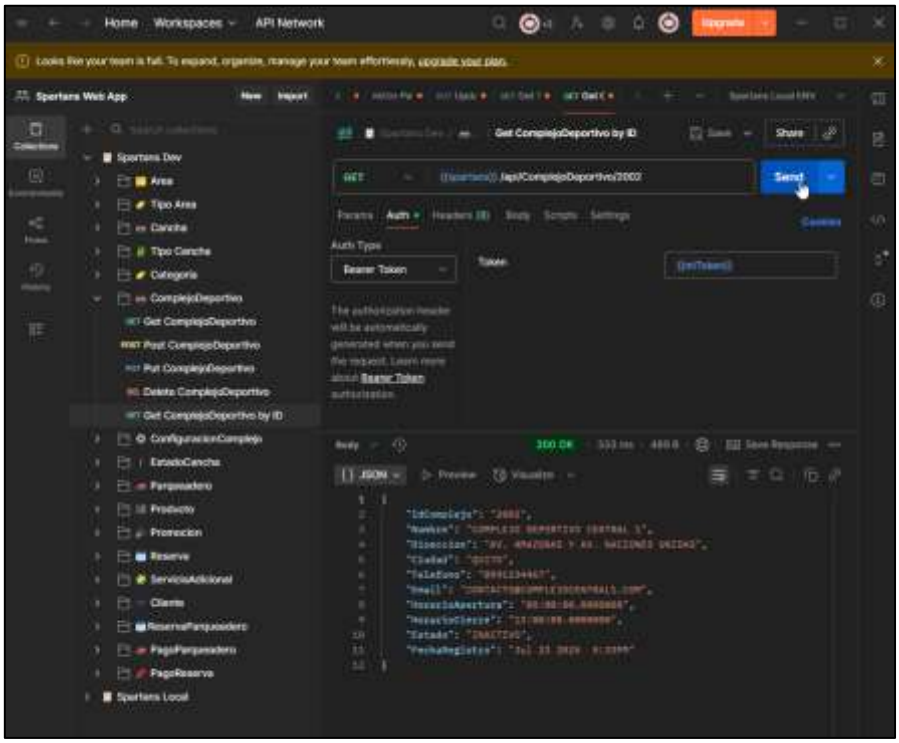


Ilustración 68 Prueba Unitaria 26

- **Get EstadoCancha**

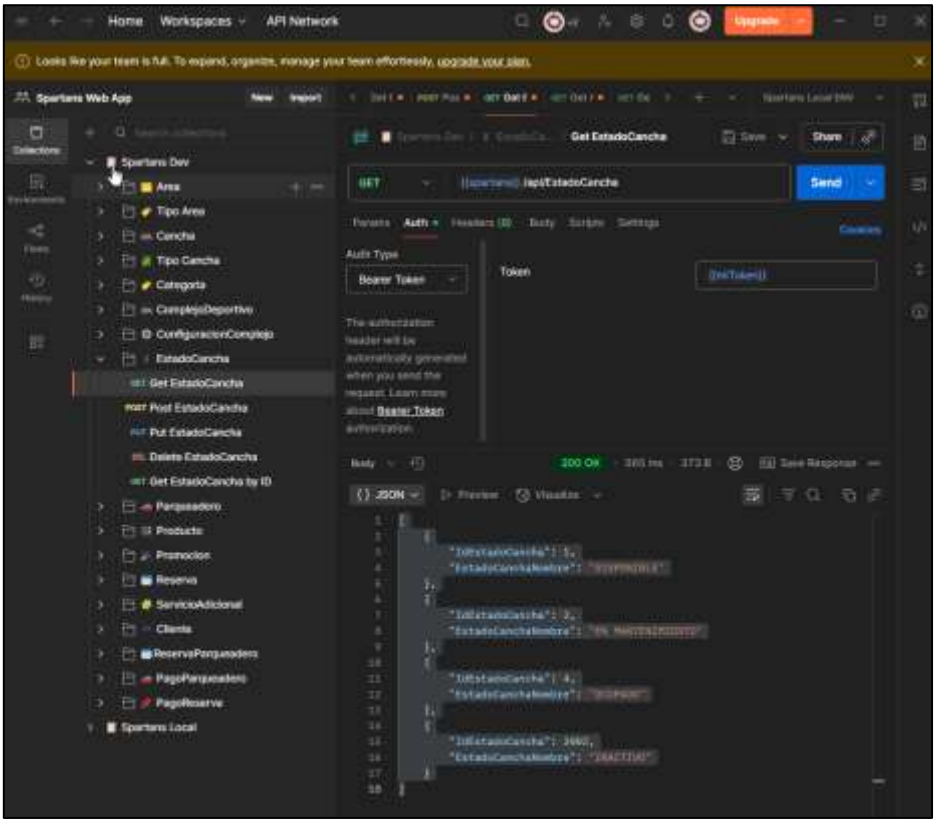


Ilustración 69 Prueba Unitaria 27

- **Post EstadoCancha**

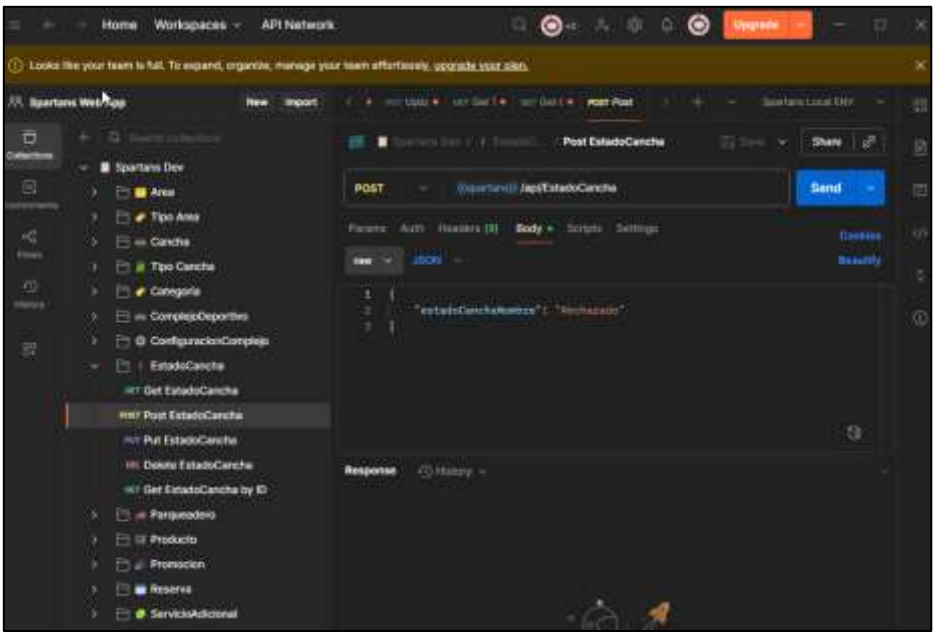


Ilustración 70 Prueba Unitaria 28

- **Put Estado Cancha**

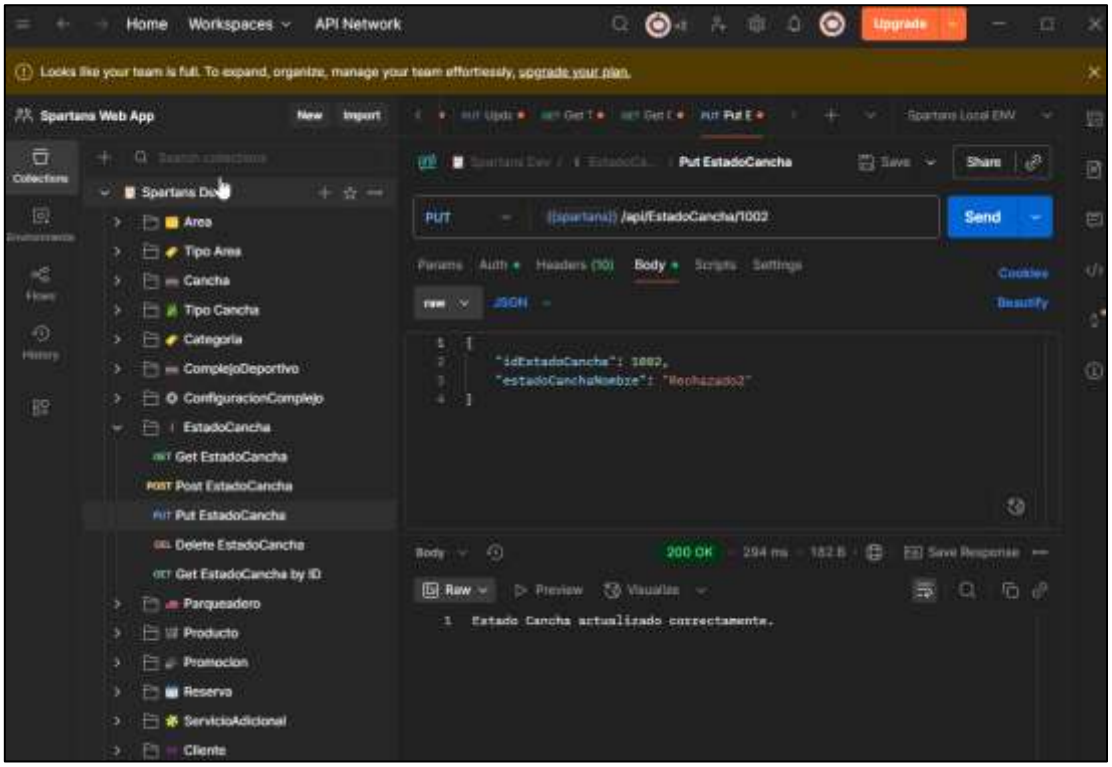


Ilustración 71 Prueba Unitaria 29

- **Get Estado Cancha by ID**

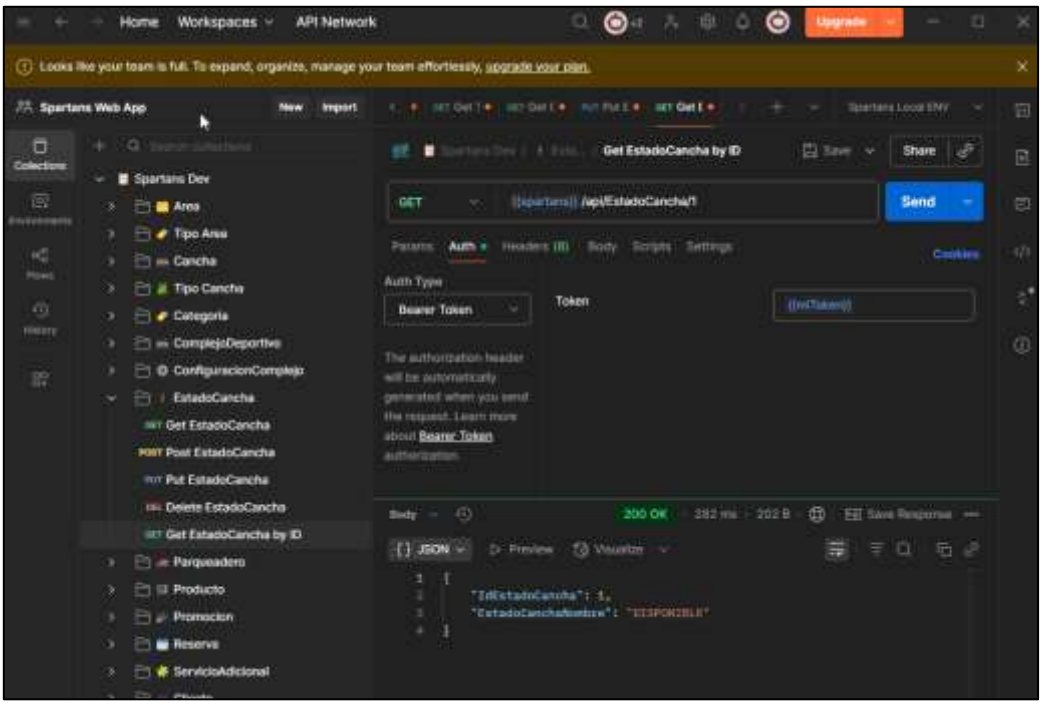


Ilustración 72 Prueba Unitaria 30

Parqueaderos

- **Get Parqueadero**

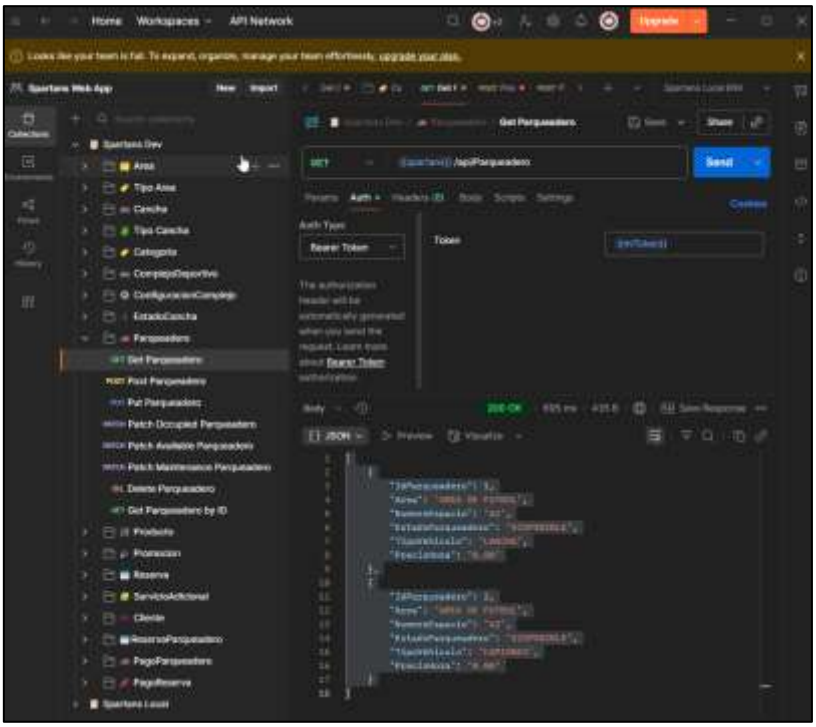


Ilustración 73 Prueba Unitaria 31

- **Post Parqueadero**

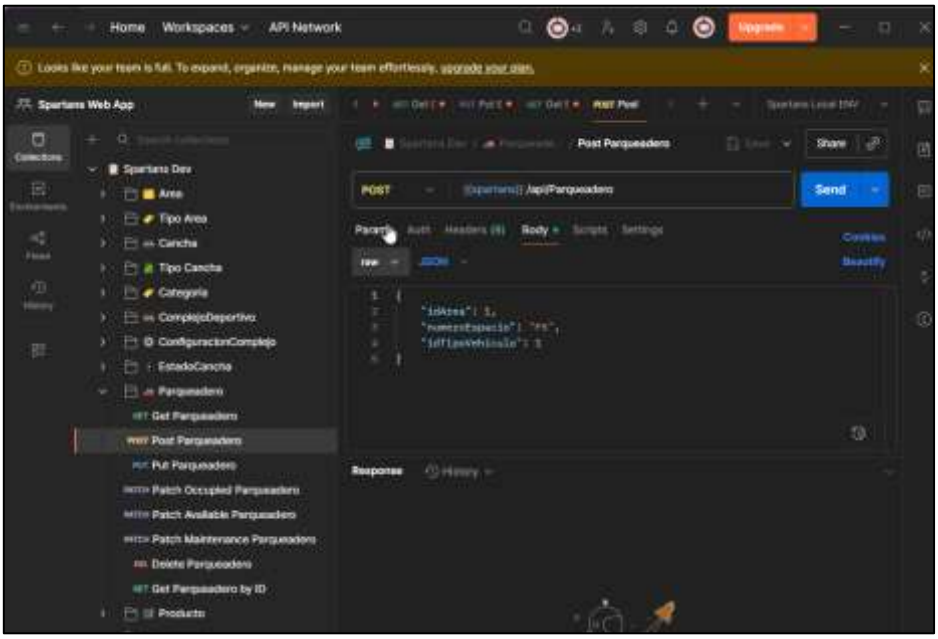


Ilustración 74 Prueba Unitaria 32

- **Put Parqueadero**

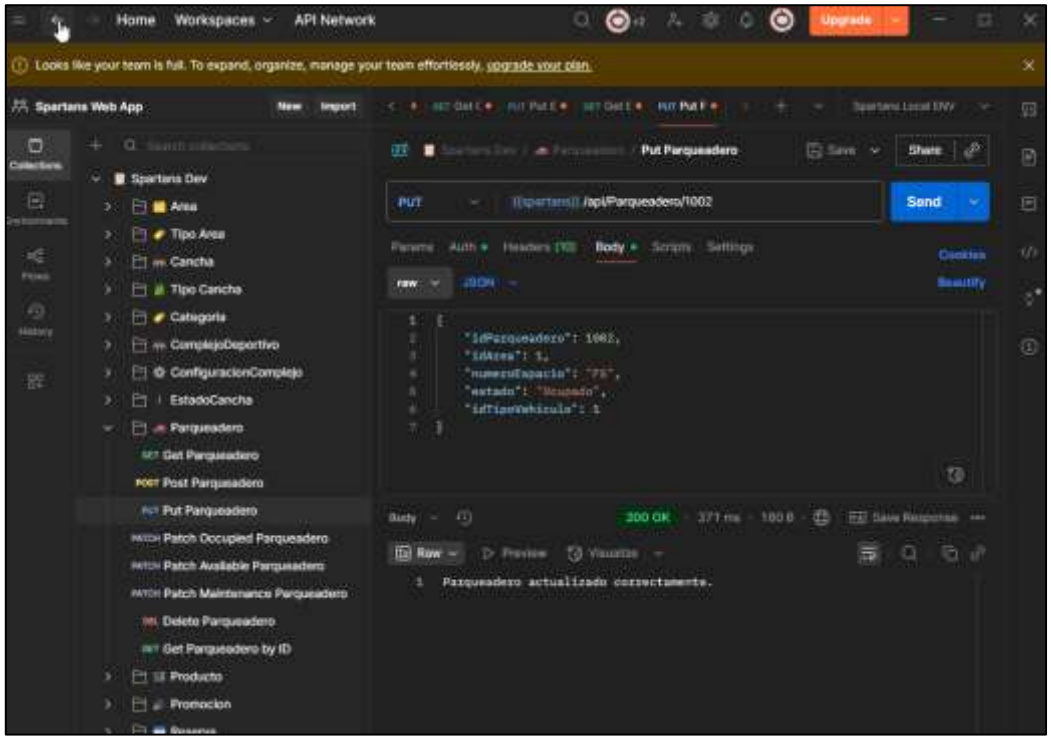


Ilustración 75 Prueba Unitaria 33

- **Patch Occupied Parqueadero**

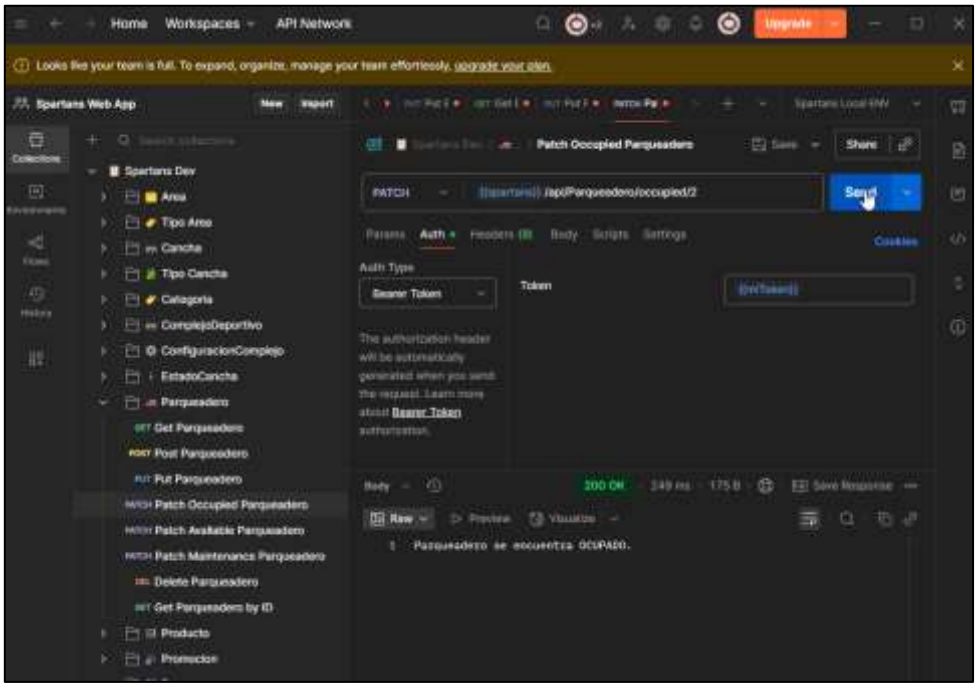


Ilustración 76 Prueba Unitaria 34

- **Patch Avaible Parqueadero**

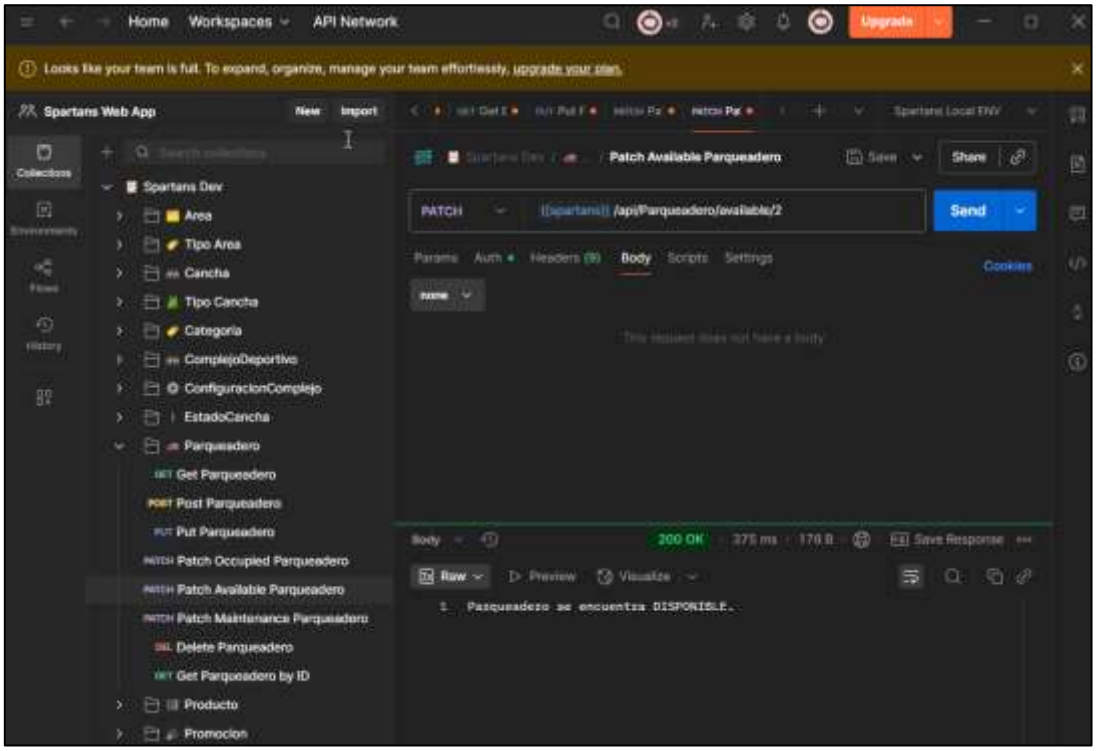


Ilustración 77 Prueba Unitaria 35

- **Patch Maintenance Parqueadero**

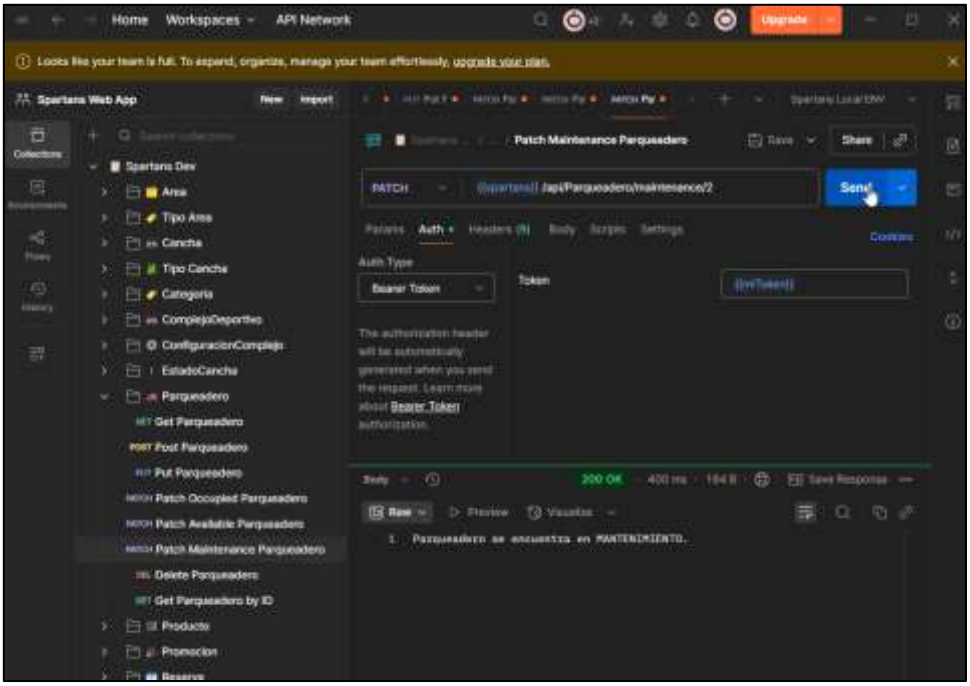


Ilustración 78 Prueba Unitaria 36

- **Get Parqueadero by ID**

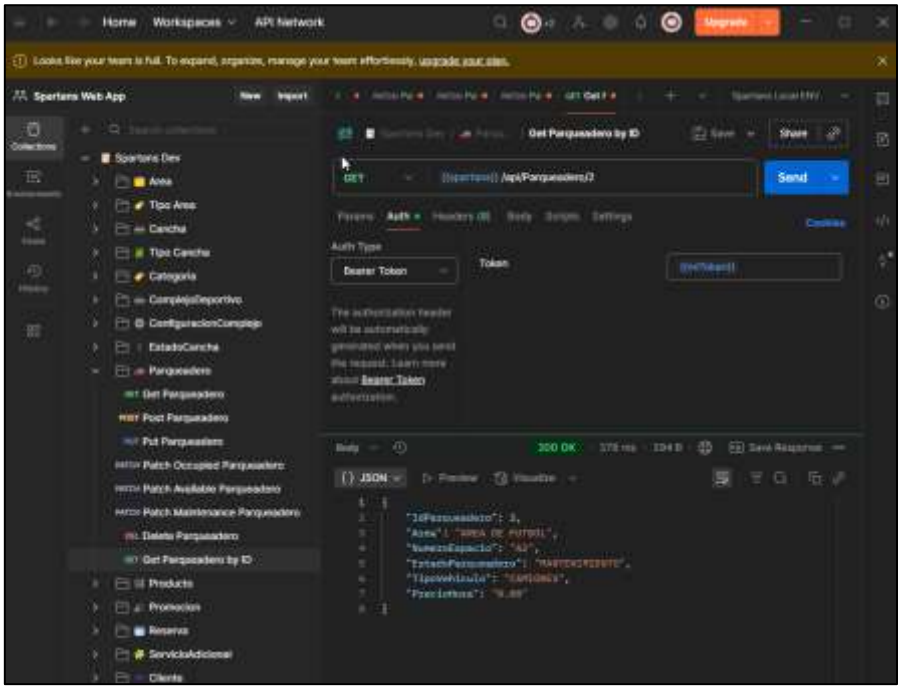


Ilustración 79 Prueba Unitaria 37

Reservas

- **Get Reserva**

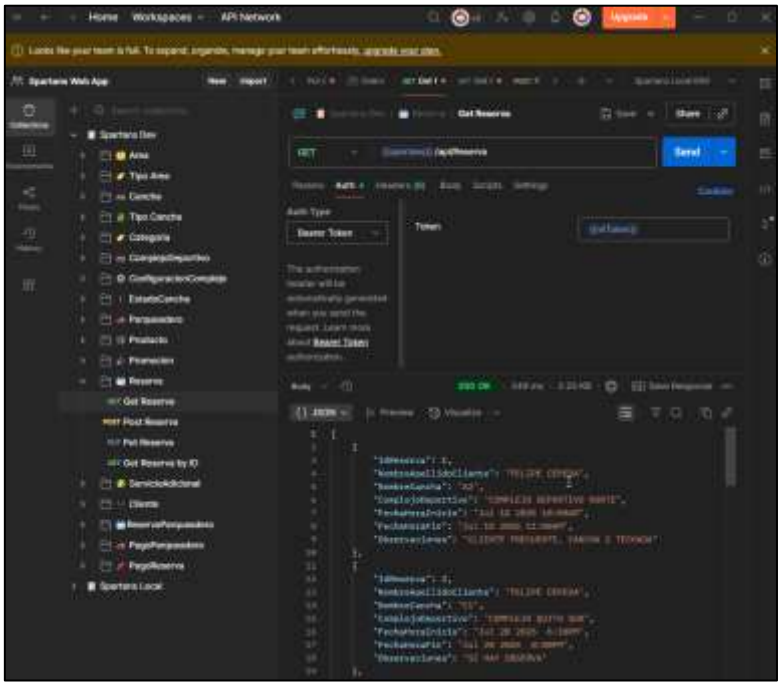


Ilustración 80 Prueba Unitaria 38

- **Post Reserva**

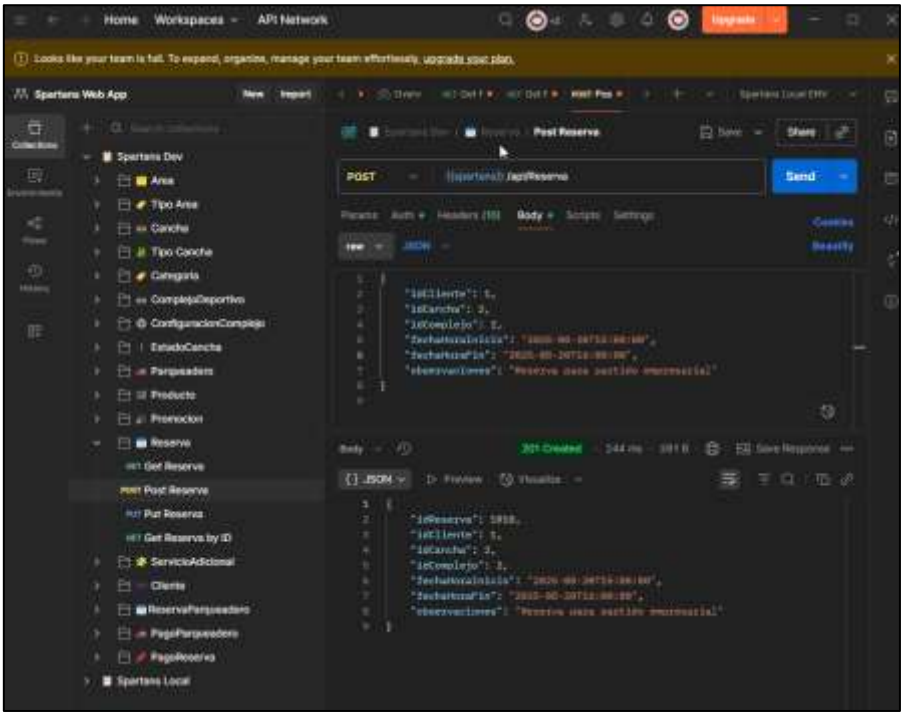


Ilustración 81 Prueba Unitaria 39

- **Put Reserva**

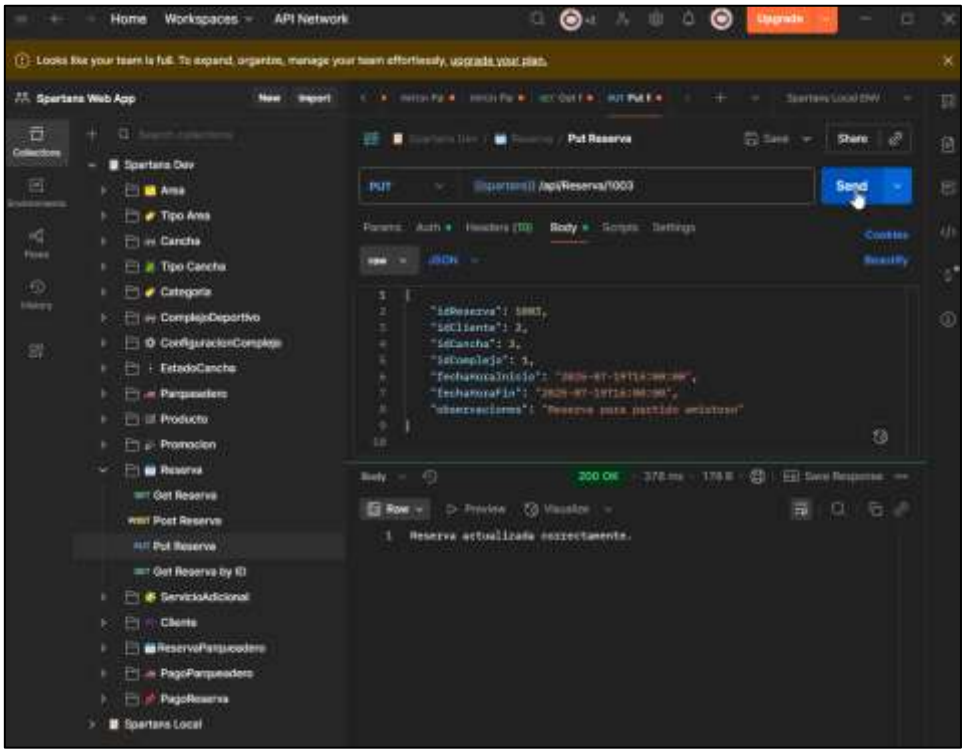


Ilustración 82 Prueba Unitaria 40

- **Get Reserva by ID**

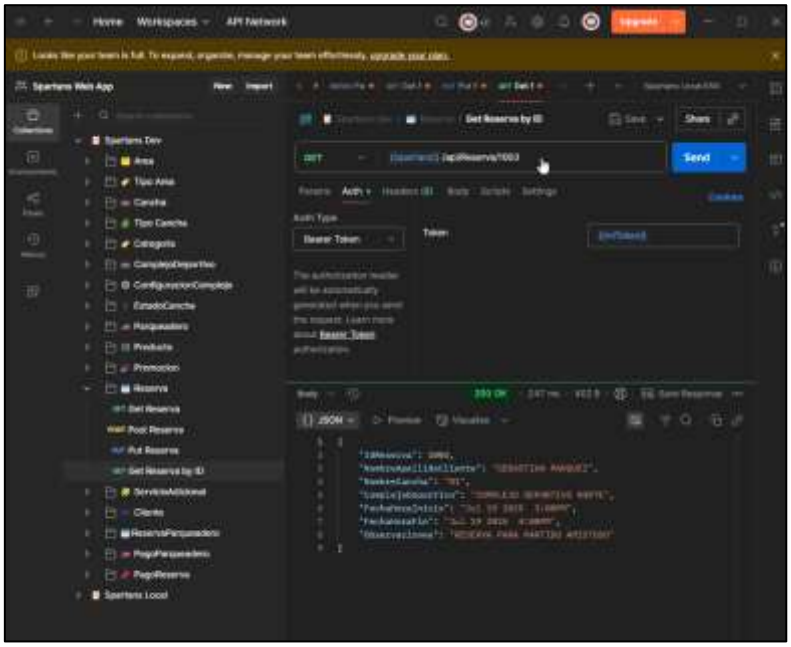


Ilustración 83 Prueba Unitaria 41

Clientes

- **Get Cliente**

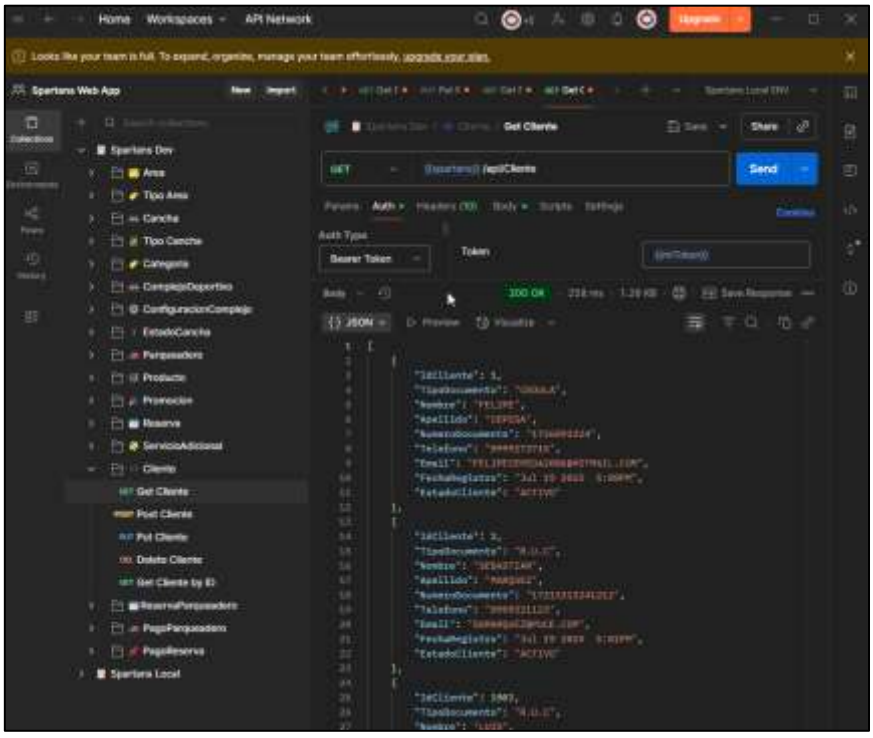


Ilustración 84 Prueba Unitaria 42

- **Post Cliente**

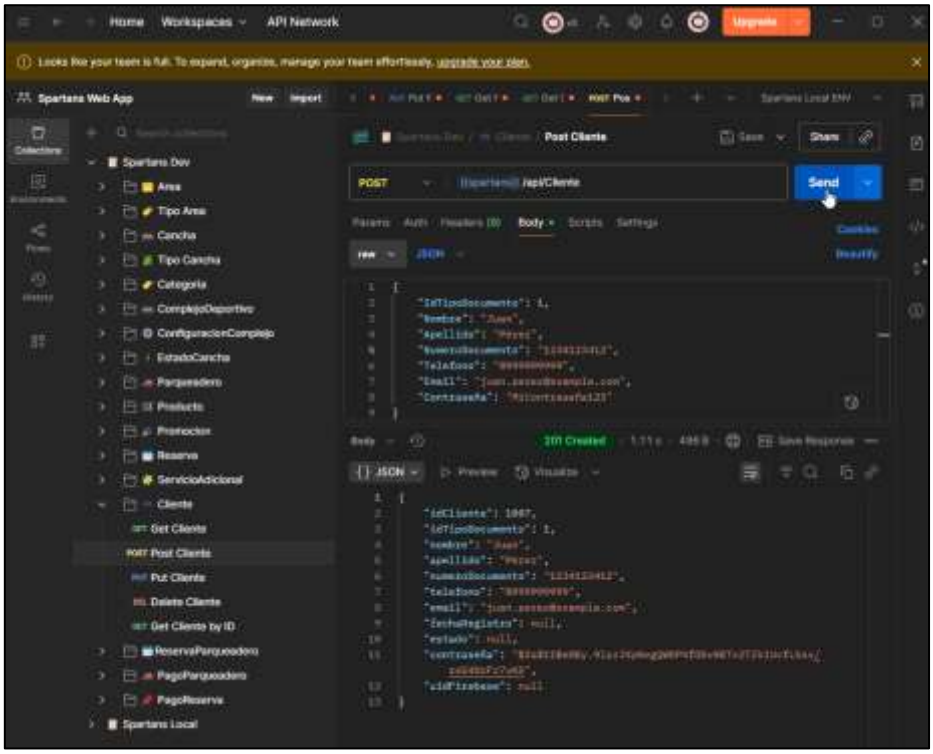


Ilustración 85 Prueba Unitaria 43

- **Put Cliente**

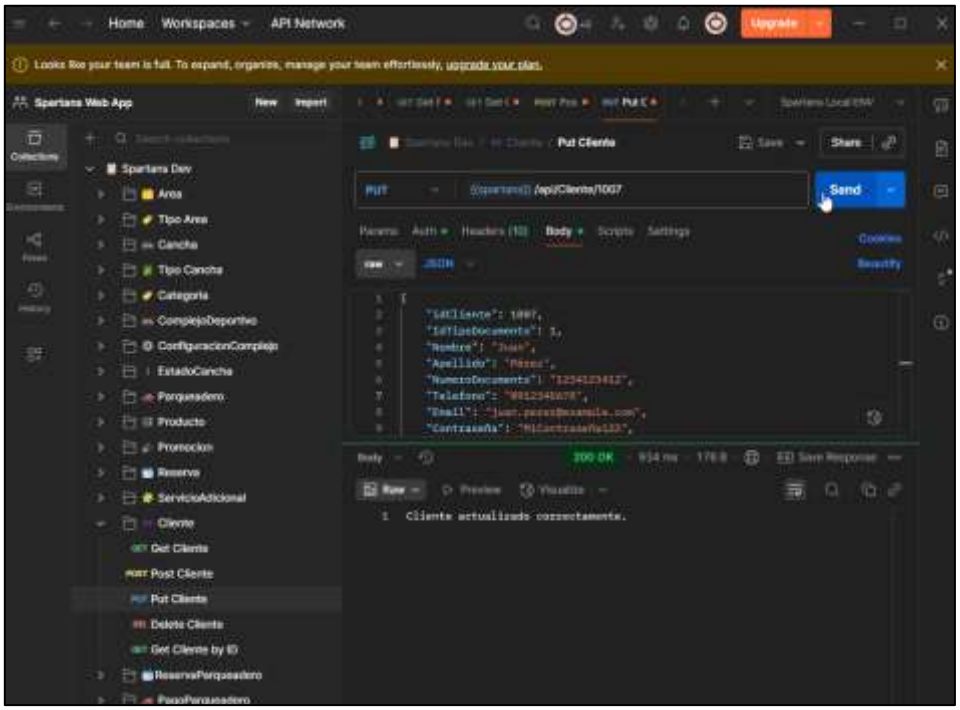


Ilustración 86 Prueba Unitaria 44

- **Get Cliente by ID**

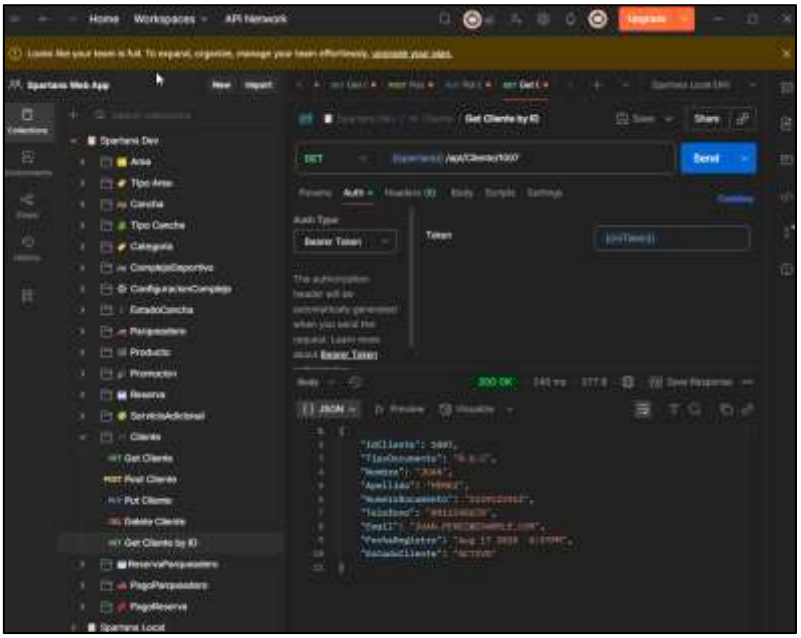


Ilustración 87 Prueba Unitaria 45

Reserva Parqueaderos

- **Get ReservaParqueaderos**

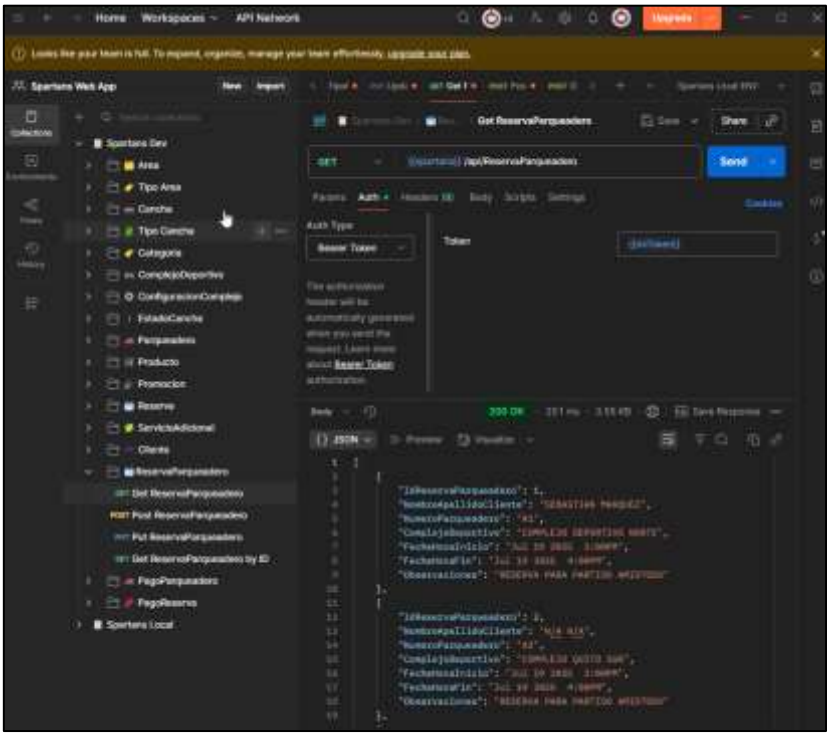


Ilustración 88 Prueba Unitaria 46

- **Post ReservaParqueaderos**

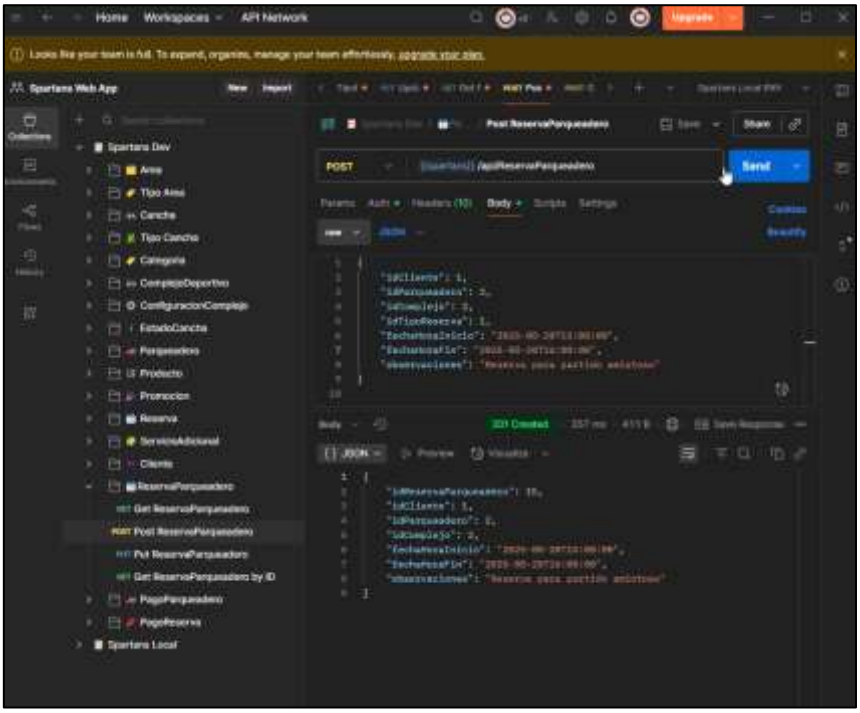


Ilustración 89 Prueba Unitaria 47

- **Put ReservaParqueaderos**

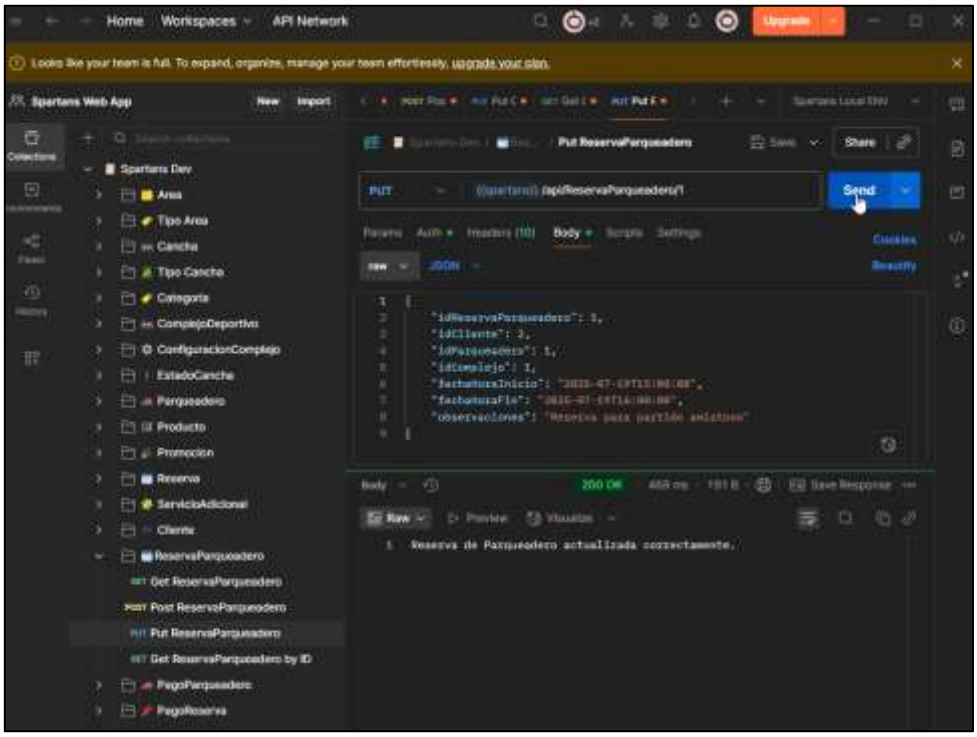


Ilustración 90 Prueba Unitaria 48

- **Get ReservaParqueaderos by ID**

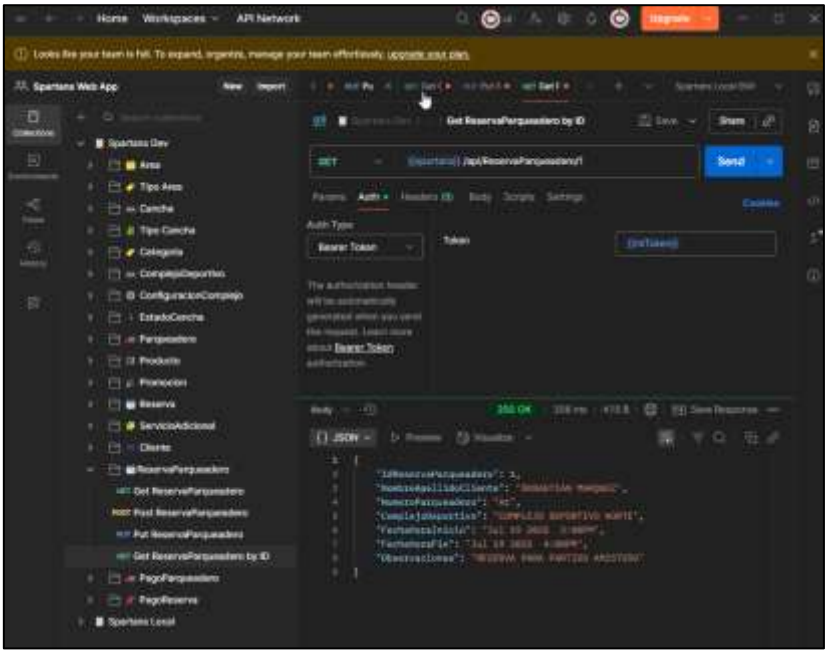


Ilustración 91 Prueba Unitaria 49

Pagos

- **Get PagoParqueadero**

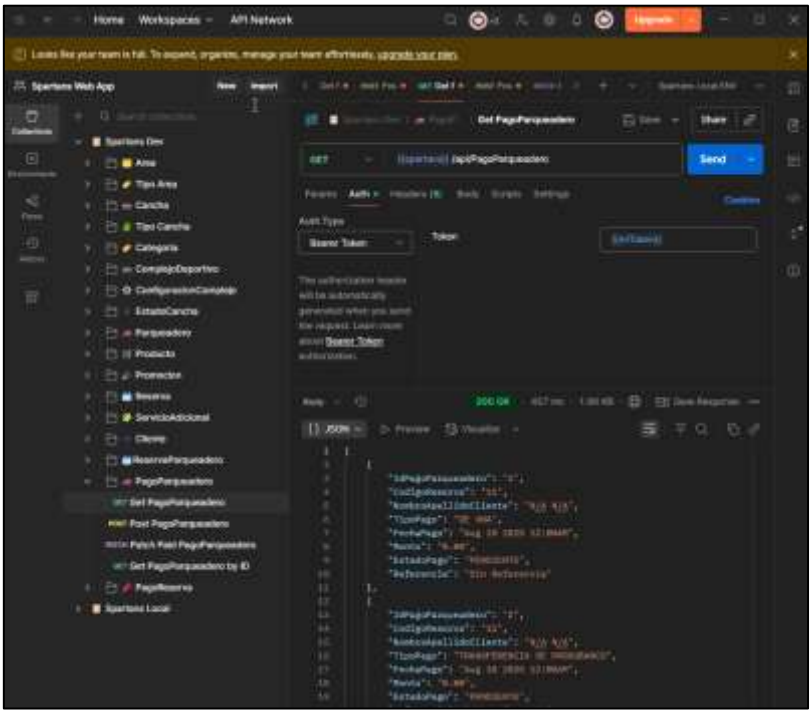


Ilustración 92 Prueba Unitaria 50

- **Post PagoParqueadero**

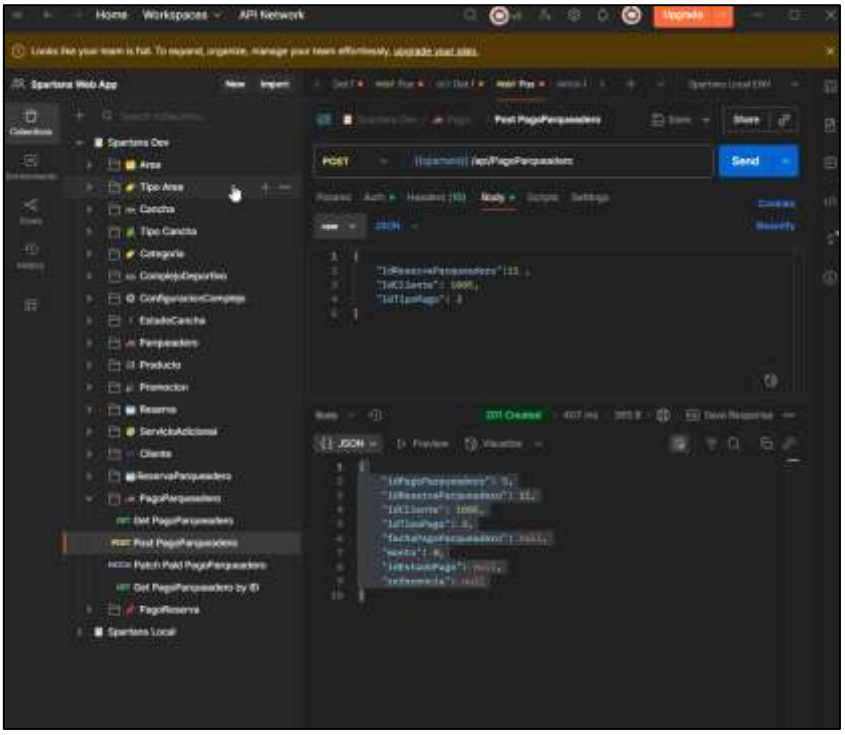


Ilustración 93 Prueba Unitaria 51

- **Patch Paid PagoParqueadero**

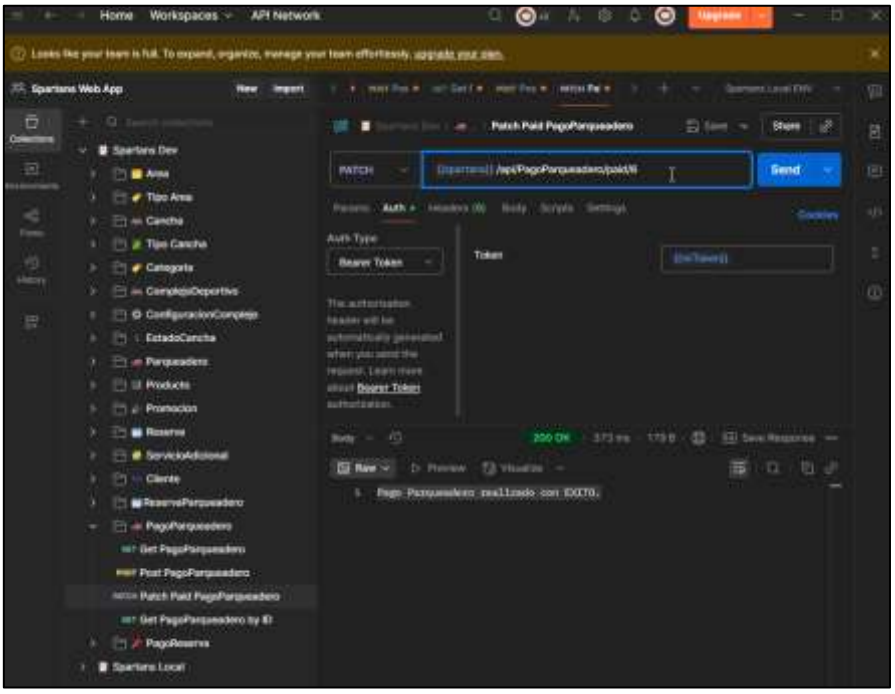


Ilustración 94 Prueba Unitaria 52

- **Get PagoParqueadero by ID**

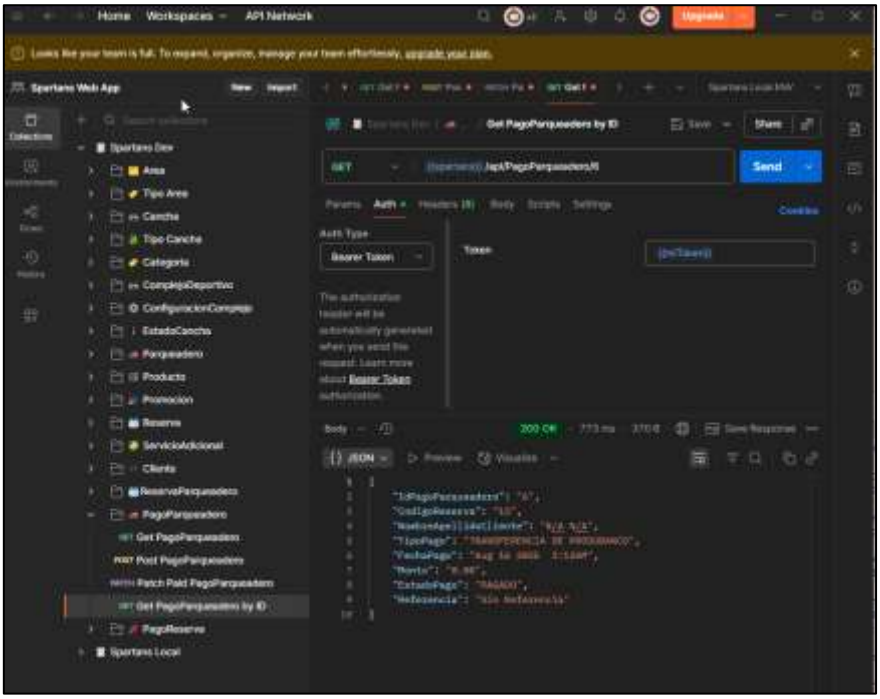


Ilustración 95 Prueba Unitaria 53

- **Get Pago**

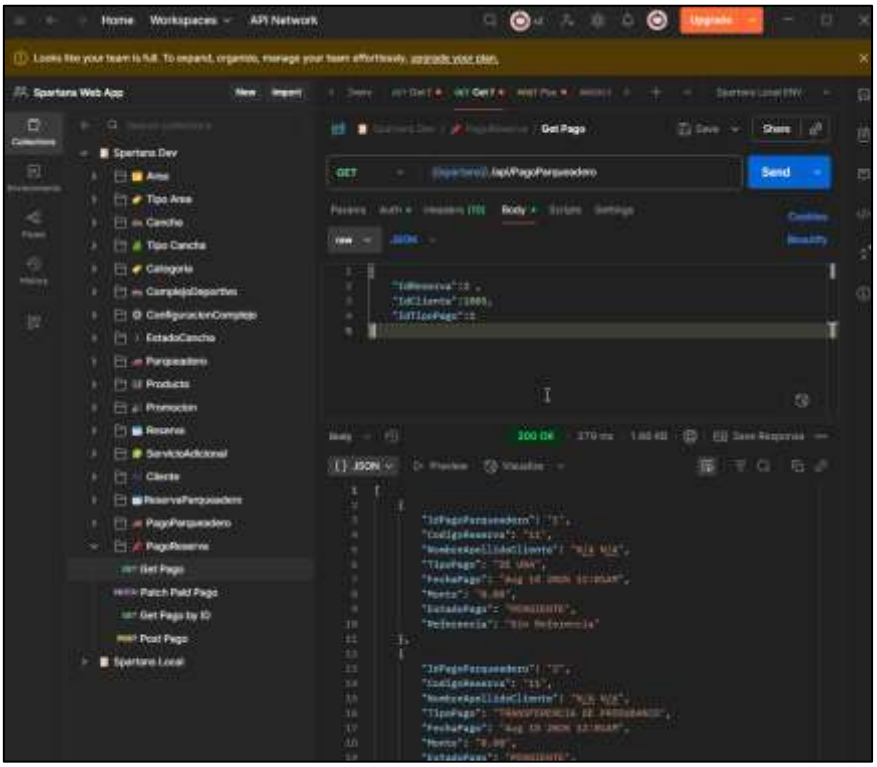


Ilustración 96 Prueba Unitaria 54

- **Patch Paid Pago**

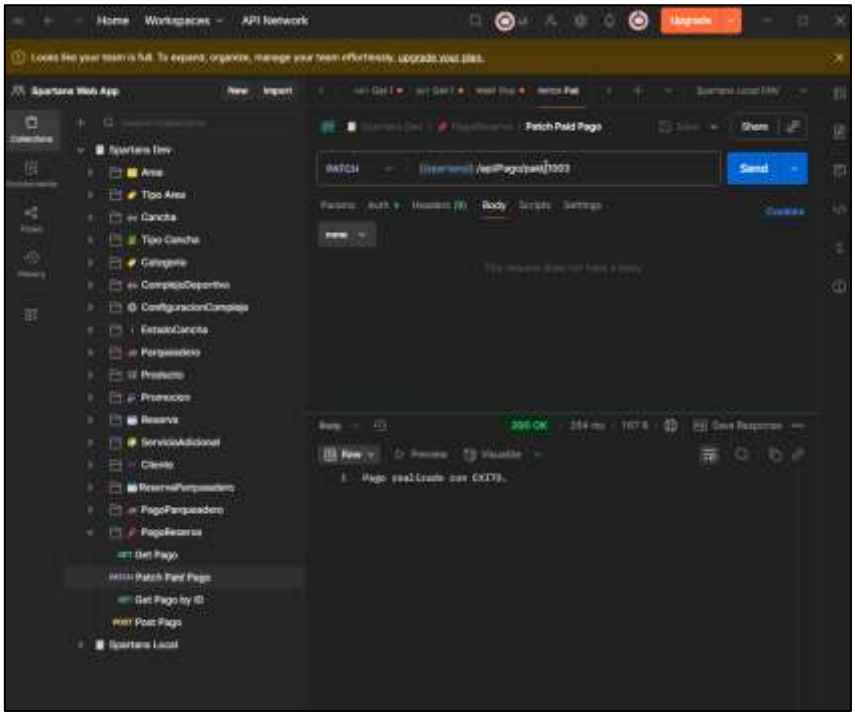


Ilustración 97 Prueba Unitaria 55

- **Get Pago by ID**

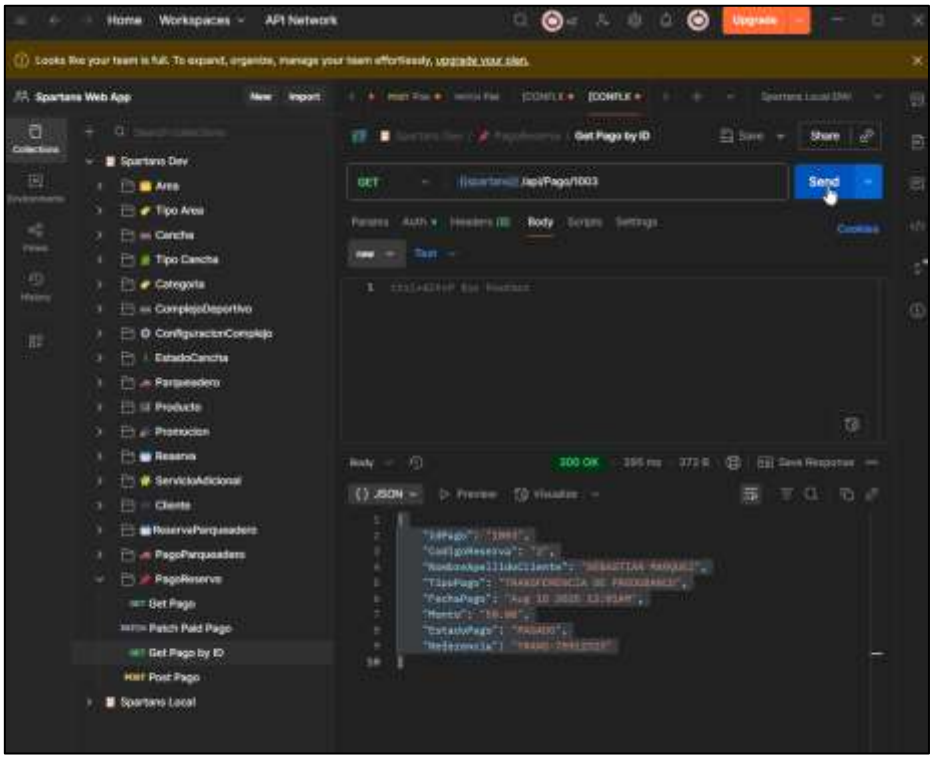


Ilustración 98 Prueba Unitaria 56

- **Post Pago**

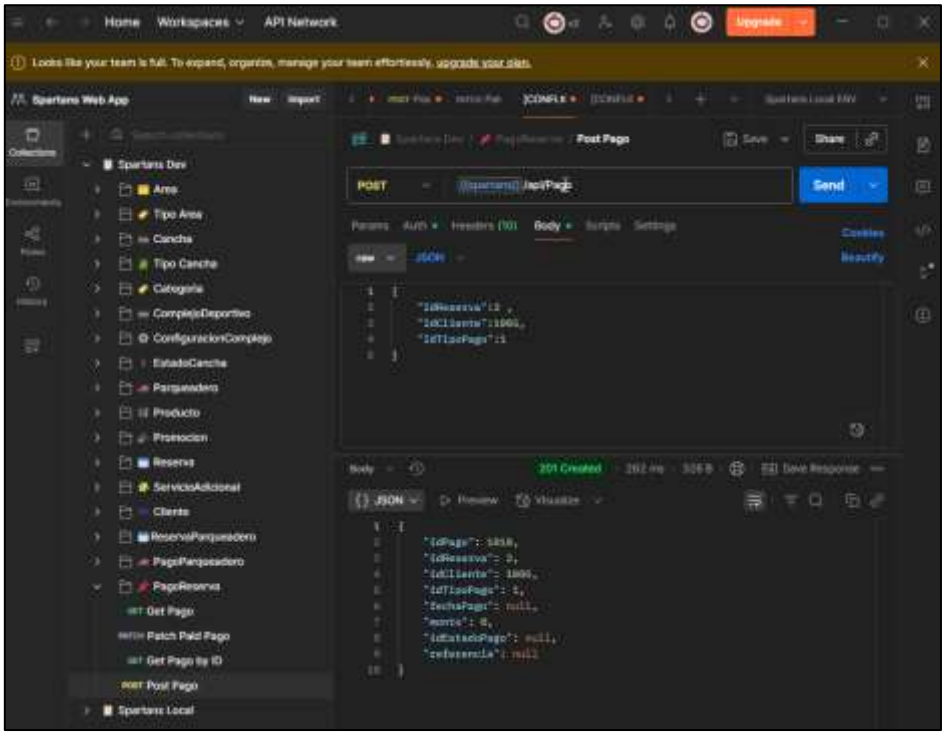


Ilustración 99 Prueba Unitaria 57

1.2. Análisis de las pruebas

1.2.1. Objetivo

Validar el correcto funcionamiento de los endpoints del BackEnd asegurando:

- Corrección funcional: Códigos de estado, payloads y reglas de negocio.
- Consistencia: Contratos de respuesta estables y mensajes de error claros.
- Seguridad básica: Uso de token y rechazo de solicitudes no autorizadas.

Metodología

1. Colecciones y entornos. Se creó una Collection llamado Spartans Dev donde se añadió las pruebas unitarias de cada Endpoint, facilitando ejecución local y en pipeline.

2. Datos de prueba. Se definieron datos mínimos válidos e inválidos para cubrir rutas felices y de error.

3. Secuencia CRUD. Cada suite ejecuta: POST, GET, PUT/PATCH, GET by id, DELETE.

4. Asserts en “Tests”. En cada request se evaluaron status, estructura y tiempos.

Criterios por método (qué se valida)

- GET (listado)
    - 200 OK, Content-Type: application/json, arreglo no nulo.
    - Tiempo de respuesta dentro del umbral acordado.
- GET by id
    - 200 OK con el id correcto y tipos de dato esperados.
    - 404 Not Found cuando el recurso no existe.
    - 400 Bad Request ante ids inválidos.
- POST (crear)
    - 201 Created y cuerpo con el recurso creado {id}.
    - Validaciones de negocio (400 si faltan obligatorios o formatos incorrectos).

- Opcional: Location en cabecera.

- PUT
  - 200/204 y persistencia de todos los campos.
  - 404 si el recurso no existe; 400 si el payload es inválido.

- PATCH
  - 200 y modificación solo de los campos enviados.
  - Validación de campos no permitidos; 400 si es inválido.

- DELETE
  - 204 no Content (o 200) y no disponibilidad posterior del recurso.

- Seguridad
  - 401 sin token, 403 sin rol adecuado; 200/201 con credenciales válidas.

Hallazgos típicos

- Inconsistencias de status, por ejemplo, devolver 200 en errores de validación.
- Campos nulos o faltantes en respuestas.
- Respuestas sin Content-Type o con formatos no JSON.
- Errores no controlados con mensajes internos filtrados.

Conclusiones

El desarrollo de un sistema de agendamiento centralizado y trazable para canchas y parqueaderos permitirá consolidar la información, reducir la dispersión de datos y establecer la base tecnológica necesaria para su posterior paso a producción.

El diseño de interfaces gráficas amigables, consistentes e intuitivas contribuirá a agilizar la navegación, disminuir la curva de aprendizaje de los usuarios y reducir los errores en el registro de información.

La implementación de la función de reserva de canchas, con validaciones de no solapamiento y aplicación de tiempos de buffer, garantizará la confirmación de turnos sin colisiones y la integridad del calendario de reservas.

La función de reserva de parqueaderos, vinculada a la reserva de canchas o gestionada de manera independiente, permitirá administrar eficientemente los cupos por franja horaria y optimizar los tiempos de acceso al complejo deportivo.

Finalmente, la integración de un chatbot interactivo con inteligencia artificial, embebido en la aplicación web, ofrecerá orientación a los usuarios mediante consultas y recomendaciones de disponibilidad, mejorando significativamente la experiencia de uso.

Recomendaciones

El sistema requiere lineamientos técnicos que fortalezcan su despliegue, seguridad, calidad y operación. Entre los aspectos principales se destacan:

Despliegue y Dependencias (Azure DevOps + Azure)

Se recomienda congelar versiones para garantizar builds reproducibles, implementar pipelines con etapas de validación y gates de aprobación, asegurar paridad entre entornos mediante variables y configuración externa, y adoptar infraestructura como código (Bicep/Terraform). Además, se sugiere aplicar listas blancas estrictas en CORS, integrar

Application Insights para observabilidad, y establecer una estrategia de actualización controlada de paquetes.

Chatbot Embebido (Copilot Studio)

Es fundamental documentar la versión y dominio del embed, controlar fallbacks en caso de fallas, registrar telemetría básica de uso y definir claramente los intents soportados. Se recomienda habilitar feature flags para activar o desactivar el chatbot de manera controlada.

APIs y Servicios (ASP.NET Core)

Deben mantenerse contratos estables documentados en OpenAPI, validaciones robustas de entrada, y un manejo uniforme de errores mediante middleware. Se sugiere aplicar patrones de resiliencia (timeouts, reintentos, circuit breakers), realizar pruebas de integración sobre entornos reales y reforzar la seguridad con validación estricta de JWT y control de accesos por rol.

Base de Datos (SQL Server / Azure SQL)

Es necesario mantener migraciones y planes de rollback versionados, parametrizar reglas de negocio en tablas de configuración, garantizar integridad de reservas mediante validaciones en API y base de datos, y exponer datos al FrontEnd a través de vistas estables para evitar dependencias de esquema.

Calidad y Operación Diaria

Se recomienda establecer un checklist de liberación (build, migraciones, pruebas, métricas activas), mantener una bitácora operativa de incidentes y documentar un README vivo con configuraciones mínimas y problemas comunes.

Referencias Bibliográficas

Balfaqih, M., Waheb, J., Mashael, K., & Rosilah, H. (2021). *Design and Development of Smart Parking System Based on Fog Computing and Internet of Things.* Obtenido de https://www.mdpi.com/2079-9292/10/24/3184

Can, L., Junjie, L., Hongxiang, C., & Zhan, M. (s.f.). *Design and Implementation of the Online Booking System of University Sports Venues.* Obtenido de https://www.researchgate.net/publication/314715739_Design_and_Implementation_of_Online_Booking_System_of_University_Sports_Venues

Følstad, A., & Brandtzaeg, P. B. (Junio de 2017). *Chatbots and the new world of HCI.* Obtenido de https://www.researchgate.net/publication/317920872_Chatbots_and_the_new_world_of_HCI

Laudon, K. C., & Laudon, J. P. (2020). *Management Information Systems: Managing the Digital Firm.* Obtenido de https://repository.dinus.ac.id/docs/ajar/Kenneth_C.Laudon,Jane_P_.Laudon_-_Management_Information_Sysrem_13th_Edition_.pdf

Payphone. (2024). *Cobro a través de la app Payphone*. Obtenido de https://docs.payphone.app/api-implementacion

Tejedor, S., Cervi, L., & Gordon, G. (24 de Enero de 2019). *Analysis of the Structure and Use of Digital Resources on the Websites of the Main Football Clubs in Europe*. Obtenido de https://www.mdpi.com/1999-5903/11/5/104

Laudon, K. C., & Laudon, J. P. (2020). Management Information Systems: Managing the Digital Firm (16.ª ed.). Pearson. PagePlace

React. (s. f.). Documentación oficial de React. Meta / react.dev. React+1

Node.js. (2025). Node.js v24 – Documentación oficial. Node.js Foundation. Node.js

npm. (s. f.). npm Docs – Guías y CLI. GitHub / npm, Inc. Documentación de npm

Microsoft. (2025). ASP.NET Core – Documentación. Microsoft Learn. Microsoft Learn+1

Microsoft. (2024). Entity Framework Core – Overview. Microsoft Learn. Microsoft Learn

Microsoft. (s. f.). SQL Server – Documentación técnica. Microsoft Learn. Microsoft Learn

Microsoft. (2025). Azure SQL Database – Documentación / ¿Qué es Azure SQL Database?. Microsoft Learn. Microsoft Learn+1

Microsoft. (2024). Transact-SQL (T-SQL) – Lenguaje y tutorial. Microsoft Learn. Microsoft Learn+1

Microsoft. (2025). SQL Server Management Studio (SSMS) – Guía. Microsoft Learn. Microsoft Learn

Microsoft. (s. f.). Azure Static Web Apps – Documentación / ¿Qué es?. Microsoft Learn. Microsoft Learn+1

Microsoft. (2025). Azure App Service – Panorama y primeros pasos. Microsoft Learn. Microsoft Learn+1

Microsoft. (2025). Azure DevOps Pipelines – Esquema YAML / Stages. Microsoft Learn. Microsoft Learn+1

Microsoft. (2025). Azure Boards – Documentación / ¿Qué es Azure Boards?. Microsoft Learn. Microsoft Learn+1

Google Firebase. (2025). Firebase Authentication – Guía web e introducción. Google Developers. Firebase+1

Google Firebase. (2025). Custom Claims con Firebase Admin. Google Developers. Firebase

Google Firebase. (2018). Introducing the Firebase Admin SDK for .NET (entrada de blog). Google. The Firebase Blog

OpenAPI Initiative. (2021–2024). OpenAPI Specification v3.x. OAI / Swagger. swagger.ioOpenAPI Initiative Publications

Microsoft. (2024). ASP.NET Core Web API con Swagger / OpenAPI. Microsoft Learn. Microsoft Learn

Scrum Guides. (2020). The Scrum Guide. ScrumGuides.org / Scrum.org. scrumguides.orgScrum.org

Apache Software Foundation. (s. f.). Apache JMeter – User's Manual. ASF. jmeter.apache.org+1

easycancha. (2025). Arriendo de canchas y clases en Ecuador. easycancha. EasyCancha

easycancha. (2025). Portal Ecuador – Reserva tu cancha. easycancha. EasyCancha

Servicio de Rentas Internas (SRI). (2024). Circular NAC-DGECCGC24-00000002: aplicación de tarifa general del IVA 15% desde 01/04/2024. SRI. Sri

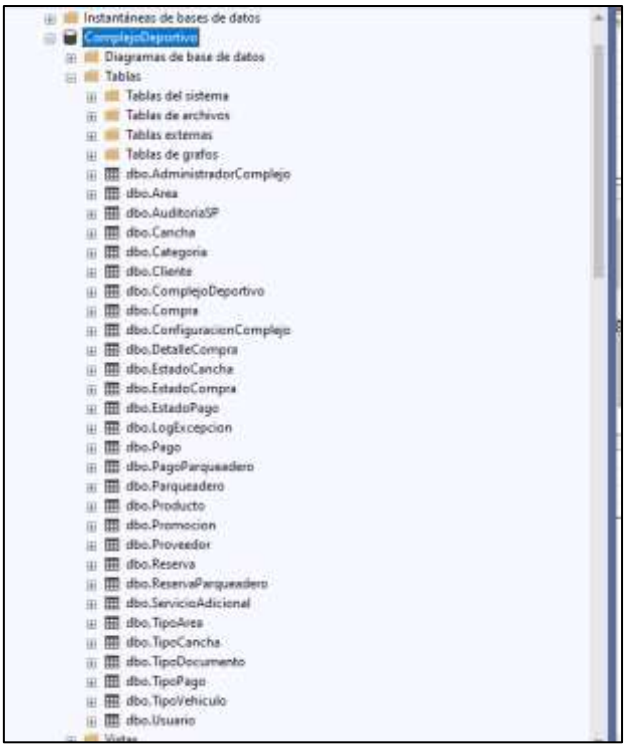
El Universo. (2024). IVA se mantendrá en 15% durante 2025 (Decreto 470). El Universo

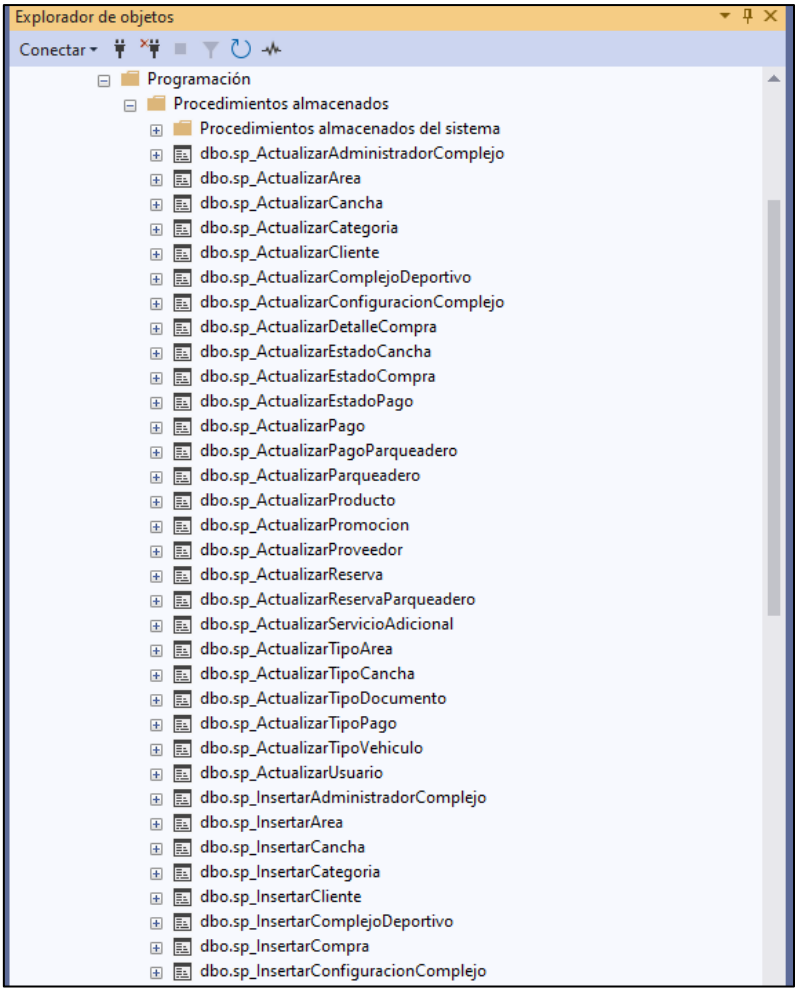
EY Ecuador. (2024). Tax Alert: Se mantiene IVA 15% para el 2025 (Decreto 470). EY

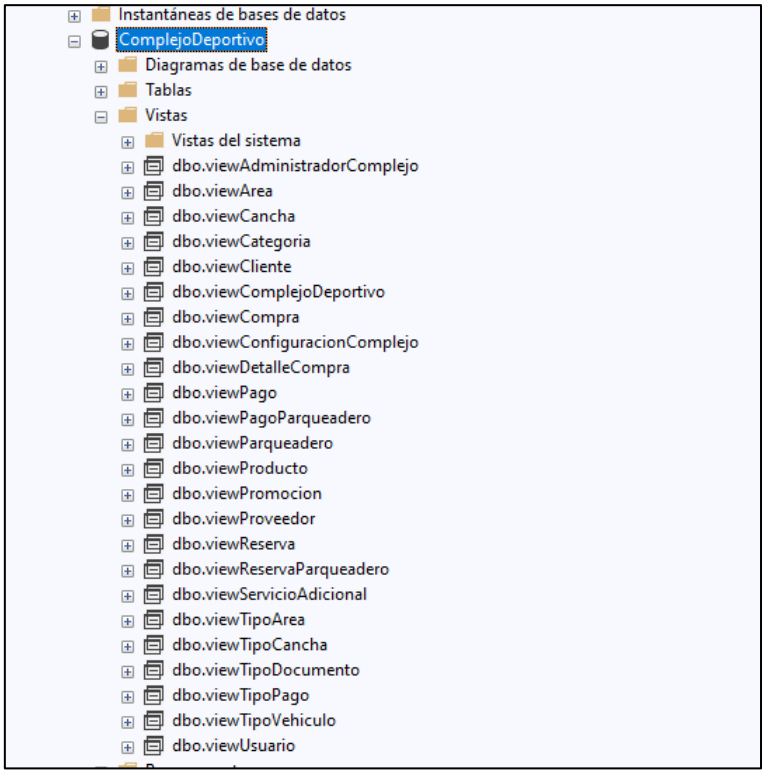
Microsoft. (2025). Microsoft Copilot Studio – Documentación / Publicar e integrar en web. Microsoft Learn. Microsoft Learn+2Microsoft Learn+2

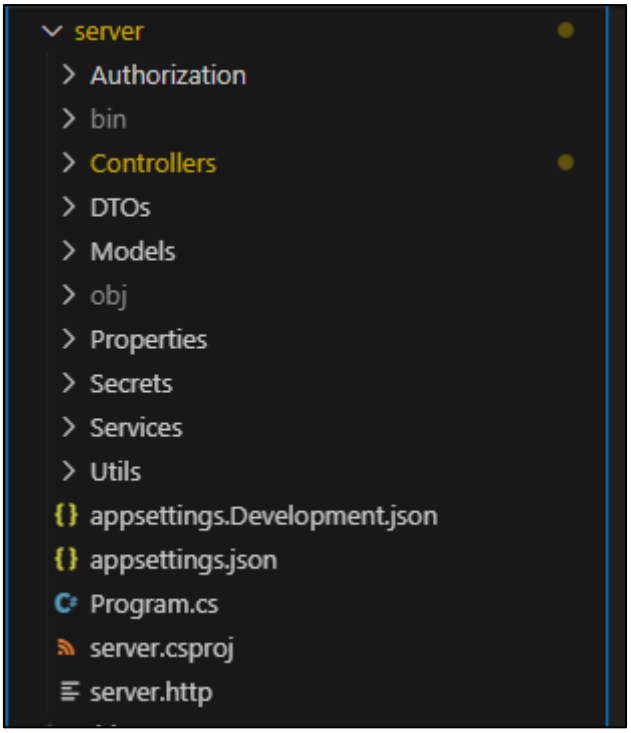
Microsoft. (2025). Copilot Studio – Fuentes de conocimiento y flujos. Microsoft Learn.

Anexos


Instantáneas de bases de datos
ComplejoDeportivo
Diagramas de base de datos
Tablas
Tablas del sistema
Tablas de archivos
Tablas externas
Tablas de grafos
dbo.AdministradorComplejo
dbo.Area
dbo.AuditoriaSP
dbo.Cancha
dbo.Categoria
dbo.Cliente
dbo.ComplejoDeportivo
dbo.Compra
dbo.ConfiguracionComplejo
dbo.DetalleCompra
dbo.EstadoCancha
dbo.EstadoCompra
dbo.EstadoPago
dbo.LogExcepcion
dbo.Pago
dbo.PagoParqueadero
dbo.Parqueadero
dbo.Producto
dbo.Promocion
dbo.Proveedor
dbo.Reserva
dbo.ReservaParqueadero
dbo.ServicioAdicional
dbo.TipoArea
dbo.TipoCancha
dbo.TipoDocumento
dbo.TipoPago
dbo.TipoVehiculo
dbo.Usuario


Explorador de objetos
Conectar
Programación
Procedimientos almacenados
Procedimientos almacenados del sistema
dbo.sp_ActualizarAdministradorComplejo
dbo.sp_ActualizarArea
dbo.sp_ActualizarCancha
dbo.sp_ActualizarCategoria
dbo.sp_ActualizarCliente
dbo.sp_ActualizarComplejoDeportivo
dbo.sp_ActualizarConfiguracionComplejo
dbo.sp_ActualizarDetalleCompra
dbo.sp_ActualizarEstadoCancha
dbo.sp_ActualizarEstadoCompra
dbo.sp_ActualizarEstadoPago
dbo.sp_ActualizarPago
dbo.sp_ActualizarPagoParqueadero
dbo.sp_ActualizarParqueadero
dbo.sp_ActualizarProducto
dbo.sp_ActualizarPromocion
dbo.sp_ActualizarProveedor
dbo.sp_ActualizarReserva
dbo.sp_ActualizarReservaParqueadero
dbo.sp_ActualizarServicioAdicional
dbo.sp_ActualizarTipoArea
dbo.sp_ActualizarTipoCancha
dbo.sp_ActualizarTipoDocumento
dbo.sp_ActualizarTipoPago
dbo.sp_ActualizarTipoVehiculo
dbo.sp_ActualizarUsuario
dbo.sp_InsertarAdministradorComplejo
dbo.sp_InsertarArea
dbo.sp_InsertarCancha
dbo.sp_InsertarCategoria
dbo.sp_InsertarCliente
dbo.sp_InsertarComplejoDeportivo
dbo.sp_InsertarCompra
dbo.sp_InsertarConfiguracionComplejo


Instantáneas de bases de datos
ComplejoDeportivo
Diagramas de base de datos
Tablas
Vistas
Vistas del sistema
dbo.viewAdministradorComplejo
dbo.viewArea
dbo.viewCancha
dbo.viewCategoria
dbo.viewCliente
dbo.viewComplejoDeportivo
dbo.viewCompra
dbo.viewConfiguracionComplejo
dbo.viewDetalleCompra
dbo.viewPago
dbo.viewPagoParqueadero
dbo.viewParqueadero
dbo.viewProducto
dbo.viewPromocion
dbo.viewProveedor
dbo.viewReserva
dbo.viewReservaParqueadero
dbo.viewServicioAdicional
dbo.viewTipoArea
dbo.viewTipoCancha
dbo.viewTipoDocumento
dbo.viewTipoPago
dbo.viewTipoVehiculo
dbo.viewUsuario



server > Authorization > RoleAuthorizationHandler.cs > ...

```
using Microsoft.AspNetCore.Authorization;
using System.Linq;
using System.Security.Claims;
using System;
using System.Threading.Tasks;

1 reference
public class RoleAuthorizationHandler : AuthorizationHandler<RoleRequirement>
{
    0 references
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, RoleRequirement requirement)
    {
        // Opción A: roles mapeados por .NET -> ClaimTypes.Role
        var mapped = context.User.FindAll(ClaimTypes.Role).Select(c => c.Value);

        // Fallback por si el token llegara con "role" sin mapear (compatibilidad)
        var raw = context.User.FindAll("role").Select(c => c.Value);

        var userRoles = mapped.Concat(raw);

        if (userRoles.Any(r => requirement.AllowedRoles.Contains(r, StringComparer.OrdinalIgnoreCase)))
        {
            context.Succeed(requirement);
        }

        return Task.CompletedTask;
    }
}
```

Authorization > RoleRequirement.cs > ...

```
using Microsoft.AspNetCore.Authorization;
using System;
using System.Collections.Generic;

6 references
public class RoleRequirement : IAuthorizationRequirement
{
    2 references
    public IReadOnlyCollection<string> AllowedRoles { get; }

    3 references
    public RoleRequirement(params string[] roles)
    {
        if (roles == null || roles.Length == 0)
            throw new ArgumentException("Debes especificar al menos un rol.", nameof(roles));

        // Guardar con comparación case-insensitive
        AllowedRoles = Array.AsReadOnly(roles);
    }
}
```

server > Controllers > AuthFirebaseController.cs > ...

```
17      public class AuthFirebaseController : ControllerBase
31          [HttpPost("registro-firebase")]
32          [Authorize]
            0 references
33          public async Task<IActionResult> CrearDesdeFirebase()
34          {
35              try
36              {
37                  var uid   = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
38                  var email = User.FindFirst(ClaimTypes.Email)?.Value;
39
40                  if (uid == null || email == null)
41                      return Unauthorized("Token inválido o incompleto");
42
43                  // Evitar duplicados
44                  if (await _clienteService.ExistePorUidAsync(uid))
45                      return BadRequest("Este usuario ya está registrado (UID)");
46
47                  if (await _clienteService.ExistePorEmailAsync(email))
48                      return BadRequest("Este usuario ya está registrado (Email)");
49
50                  // Guardar en base local
51                  var cliente = new ClienteModel
52                  {
53                      Nombre          = "N/A",
54                      Apellido        = "N/A",
55                      Email           = email,
56                      NumeroDocumento = "N/A",
57                      Telefono        = "N/A",
58                      Contraseña      = "Firebase",
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTGRESQL QUERY RESULTS

```
[HttpPost("login-firebase")]
[AllowAnonymous]
0 references
public async Task<IActionResult> LoginFirebase([FromBody] LoginFirebaseInput? input)
{
    try
    {
        // ID token desde Authorization: Bearer o desde body
        var bearer = Request.Headers["Authorization"].FirstOrDefault();
        string idTokenIn = null;

        if (!string.IsNullOrEmpty(bearer) && bearer.StartsWith("Bearer ", System.StringComparison.OrdinalIgr
            idTokenIn = bearer.Substring("Bearer ".Length).Trim();

        if (string.IsNullOrWhiteSpace(idTokenIn))
            idTokenIn = input?.IdToken;

        if (string.IsNullOrWhiteSpace(idTokenIn))
            return Unauthorized("Falta ID token de Firebase (envíalo en Authorization: Bearer o en el body).

        // Decodificar/verificar para obtener UID (sin emitir aún)
        var decoded = await FirebaseAuth.DefaultInstance.VerifyIdTokenAsync(idTokenIn, /*checkRevoked*/ true
        var uid     = decoded.Uid;

        // Buscar IdCliente por UID
        var clientId = await _clienteService.GetIdByUidAsync(uid);
        if (!clientId.HasValue)
        {
            return NotFound(new
```

```
nController.cs
31      [HttpGet]
32      [Authorize(Policy = "AdminOrCliente")]
        0 references
33      public async Task<ActionResult<IEnumerable<Dictionary<string, object>>>> GetAll()
34      {
35          try
36          {
37              var canchas = await _service.GetAllAsync();
38              return Ok(canchas);
39          }
40          catch (Exception ex)
41          {
42              return StatusCode(500, $"Error interno del servidor: {ex.Message}");
43          }
44      }
45
46      [HttpGet("{id}")]
47      [Authorize(Policy = "AdminOrCliente")]
        1 reference
48      public async Task<ActionResult<Dictionary<string, object>>> GetById(int id)
49      {
50          try
51          {
52              var cancha = await _service.GetByIdAsync(id);
53              if (cancha == null)
54                  return NotFound($"Cancha con ID {id} no encontrado");
55              return Ok(cancha);
56          }
57          catch (Exception ex)

PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   POSTGRESQL QUERY RESULTS
```

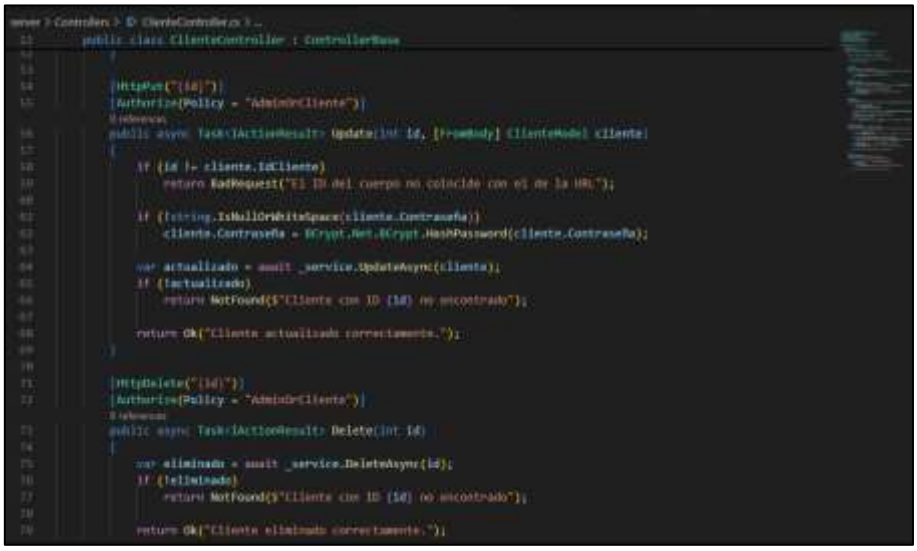
server > Controllers > CanchaController.cs > ...

```
12      public class CanchaController : ControllerBase
53          [HttpPost]
54          [Authorize(Policy = "AdminOnly")]
            0 references
55          public async Task<ActionResult<CanchaModel>> Create([FromBody] CanchaModel cancha)
56          {
57              try
58              {
59                  if (!ModelState.IsValid)
60                      return BadRequest(ModelState);
61
62                  var newId = await _service.CreateAsync(cancha);
63                  cancha.IdCancha = newId;
64
65                  return CreatedAtAction(nameof(GetById), new { id = newId }, cancha);
66              }
67              catch (Exception ex)
68              {
69                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");
70              }
71          }
72
73          [HttpPut("{id}")]
74          [Authorize(Policy = "AdminOnly")]
            0 references
75          public async Task<IActionResult> Update(int id, CanchaModel cancha)
76          {
77              if (id != cancha.IdCancha)
78                  return BadRequest("El ID del cuerpo no coincide con el de la URL");
79
80              try
```

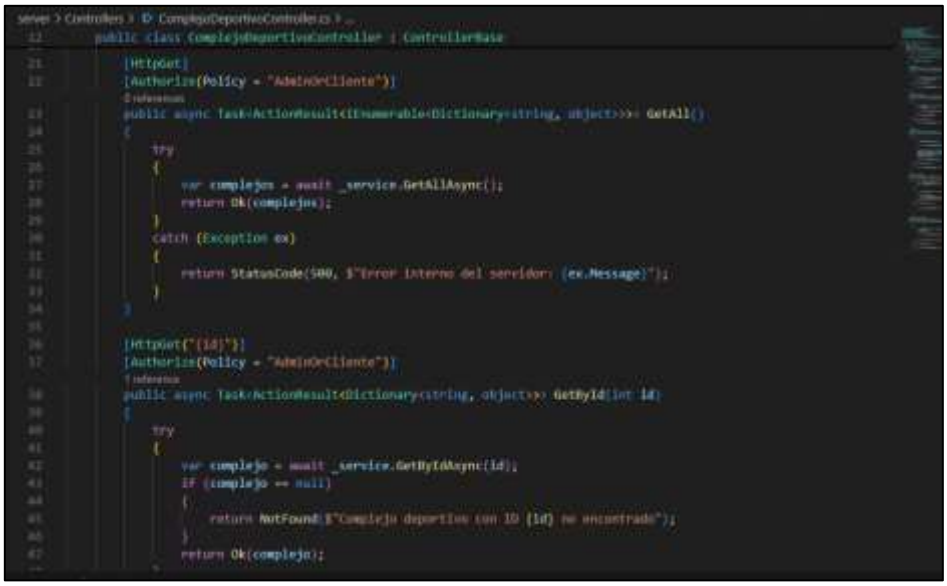
server > Controllers > AreaController.cs > ...

```
13      public class AreaController : ControllerBase
23          [Authorize(Policy = "AdminOrCliente")]
            0 references
24          public async Task<ActionResult<IEnumerable<Dictionary<string, object>>>> GetAll()
25          {
26              try
27              {
28                  var areas = await _service.GetAllAsync();
29                  return Ok(areas);
30              }
31              catch (Exception ex)
32              {
33                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");
34              }
35          }
36
37          [HttpGet("{id}")]
38          [Authorize(Policy = "AdminOrCliente")]
            1 reference
39          public async Task<ActionResult<Dictionary<string, object>>> GetById(int id)
40          {
41              try
42              {
43                  var area = await _service.GetByIdAsync(id);
44                  if (area == null)
45                      return NotFound($"Area con ID {id} no encontrado");
46                  return Ok(area);
47              }
48              catch (Exception ex)
49              {
50                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");
```

```
server > Controllers > © ClienteController.cs > ...
12      public class ClienteController : ControllerBase
13          public ClienteController(ClienteService service)
14          {
15          }
16
17          [HttpGet]
18          [Authorize(Policy = "AdminOnly")]
            0 references
19          public async Task<ActionResult<IEnumerable<Dictionary<string, object>>>> GetAll()
20          {
21              var clientes = await _service.GetAllAsync();
22              return Ok(clientes);
23          }
24
25          [HttpGet("{id}")]
26          [Authorize(Policy = "AdminOrCliente")]
            1 reference
27          public async Task<ActionResult<Dictionary<string, object>>> GetById(int id)
28          {
29              var cliente = await _service.GetByIdAsync(id);
30              if (cliente == null)
31                  return NotFound($"Cliente con ID {id} no encontrado");
32              return Ok(cliente);
33          }
34
35          [HttpPost]
36          [AllowAnonymous]
            0 references
37          public async Task<ActionResult<ClienteModel>> Create([FromBody] ClienteModel cliente)
38          {
39              if (!ModelState.IsValid)
40                  return BadRequest(ModelState);
```



```
server > Controllers > © ClienteController.cs > ...
12      public class ClienteController : ControllerBase
13          {
14          [HttpPut("{id}")]
15          [Authorize(Policy = "AdminOrCliente")]
            0 references
16          public async Task<IActionResult> Update(int id, [FromBody] ClienteModel cliente)
17          {
18              if (id != cliente.IdCliente)
19                  return BadRequest("El ID del cuerpo no coincide con el de la URL");
20
21              if (!string.IsNullOrWhiteSpace(cliente.Contraseña))
22                  cliente.Contraseña = BCrypt.Net.BCrypt.HashPassword(cliente.Contraseña);
23
24              var actualizado = await _service.UpdateAsync(cliente);
25              if (!actualizado)
26                  return NotFound($"Cliente con ID {id} no encontrado");
27
28              return Ok("Cliente actualizado correctamente.");
29          }
30
31          [HttpDelete("{id}")]
32          [Authorize(Policy = "AdminOrCliente")]
            0 references
33          public async Task<IActionResult> Delete(int id)
34          {
35              var eliminado = await _service.DeleteAsync(id);
36              if (!eliminado)
37                  return NotFound($"Cliente con ID {id} no encontrado");
38
39              return Ok("Cliente eliminado correctamente.");
```



```
server > Controllers > © ComplejoDeportivoController.cs > ...
12      public class ComplejoDeportivoController : ControllerBase
13          [HttpGet]
14          [Authorize(Policy = "AdminOrCliente")]
            0 references
15          public async Task<ActionResult<IEnumerable<Dictionary<string, object>>>> GetAll()
16          {
17              try
18              {
19                  var complejos = await _service.GetAllAsync();
20                  return Ok(complejos);
21              }
22              catch (Exception ex)
23              {
24                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");
25              }
26          }
27
28          [HttpGet("{id}")]
29          [Authorize(Policy = "AdminOrCliente")]
            1 reference
30          public async Task<ActionResult<Dictionary<string, object>>> GetById(int id)
31          {
32              try
33              {
34                  var complejo = await _service.GetByIdAsync(id);
35                  if (complejo == null)
36                  {
37                      return NotFound($"Complejo deportivo con ID {id} no encontrado");
38                  }
39                  return Ok(complejo);
```

```
server > Controllers > ComplejoDeportivoController.cs > ...
12      public class ComplejoDeportivoController : ControllerBase
57          public async Task<ActionResult<ComplejoDeportivoModel>> Create([FromBody] ComplejoDeportivoModel complejo)
64              }
65
66              var newId = await _service.CreateAsync(complejo);
67              complejo.IdComplejo = newId;
68              complejo.FechaRegistro = DateTime.Now;
69              complejo.Estado = true;
70
71              return CreatedAtAction(nameof(GetById), new { id = newId }, complejo);
72          }
73          catch (Exception ex)
74          {
75              return StatusCode(500, $"Error interno del servidor: {ex.Message}");
76          }
77      }
78
79      [HttpPut("{id}")]
80      [Authorize(Policy = "AdminOnly")]
        0 references
81      public async Task<IActionResult> Update(int id, ComplejoDeportivoModel modelo)
82      {
83          if (id != modelo.IdComplejo)
84              return BadRequest("El ID del cuerpo no coincide con el de la URL");
85
86          try
87          {
88              var actualizado = await _service.UpdateAsync(modelo);
89
```

```
server > Program.cs > Program > <top-level-statements-entry-point>
14  var builder = WebApplication.CreateBuilder(args);
 1  using FirebaseAdmin;
43
44  // Tus servicios
45  builder.Services.AddScoped<ComplejoDeportivoService>();
46  builder.Services.AddHttpClient<PayphoneClient>();
47  builder.Services.AddScoped<UsuarioService>();
48  builder.Services.AddScoped<TipoVehiculoService>();
49  builder.Services.AddScoped<TipoPagoService>();
50  builder.Services.AddScoped<TipoDocumentoService>();
51  builder.Services.AddScoped<TipoCanchaService>();
52  builder.Services.AddScoped<TipoAreaService>();
53  builder.Services.AddScoped<ServicioAdicionalService>();
54  builder.Services.AddScoped<ProveedorService>();
55  builder.Services.AddScoped<PromocionService>();
56  builder.Services.AddScoped<AreaService>();
57  builder.Services.AddScoped<AdministradorComplejoService>();
58  builder.Services.AddScoped<EstadoCanchaService>();
59  builder.Services.AddScoped<CanchaService>();
60  builder.Services.AddScoped<CategoriaService>();
61  builder.Services.AddScoped<ClienteService>();
62  builder.Services.AddScoped<ConfiguracionComplejoService>();
63  builder.Services.AddScoped<EstadoCompraService>();
64  builder.Services.AddScoped<ProductoService>();
65  builder.Services.AddScoped<ParqueaderoService>();
66  builder.Services.AddScoped<EstadoPagoService>();
67  builder.Services.AddScoped<PagoService>();
68  builder.Services.AddScoped<ReservaService>();
69  builder.Services.AddScoped<ReservaParqueaderoService>();
70  builder.Services.AddScoped<PagoParqueaderoService>();
```

```
// ============================
// Tu handler custom
builder.Services.AddSingleton<IAuthorizationHandler, RoleAuthorizationHandler>();

// Políticas usando RoleRequirement (para que ejecute TU handler)
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("AdminOnly", policy =>
        policy.RequireAuthenticatedUser()
              .AddRequirements(new RoleRequirement("admin")));

    options.AddPolicy("ClienteOnly", policy =>
        policy.RequireAuthenticatedUser()
              .AddRequirements(new RoleRequirement("cliente")));

    options.AddPolicy("AdminOrCliente", policy =>
        policy.RequireAuthenticatedUser()
              .AddRequirements(new RoleRequirement("admin", "cliente")));
});
```

server > Program.cs > Program > <top-level-statements-entry-point>

```
14  var builder = WebApplication.CreateBuilder(args);
1   using FirebaseAdmin;
...
164 builder.Services
165     .AddAuthentication(options =>
166     {
167         options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
168         options.DefaultChallengeScheme    = JwtBearerDefaults.AuthenticationScheme;
169     })
170     .AddJwtBearer(options =>
171     {
172         options.Authority            = $"https://securetoken.google.com/{projectId}";
173         options.Audience             = projectId;
174         options.RequireHttpsMetadata = true;
175         options.SaveToken            = true;
176         options.IncludeErrorDetails  = true;
177
178         // Opción A: trabajar con ClaimTypes (mapeo por defecto)
179         options.TokenValidationParameters = new TokenValidationParameters
180         {
181             ValidateIssuer           = true,
182             ValidIssuer              = $"https://securetoken.google.com/{projectId}",
183             ValidateAudience         = true,
184             ValidAudience            = projectId,
185             ValidateLifetime         = true,
186             RequireExpirationTime    = true,
187             ValidateIssuerSigningKey = true,
188
189             // "sub" -> ClaimTypes.NameIdentifier ; "role"/"roles" -> ClaimTypes.Role
190             NameClaimType = ClaimTypes.NameIdentifier,
191             RoleClaimType = ClaimTypes.Role,
```

server > Program.cs > Program > <top-level-statements-entry-point>

```
14  var builder = WebApplication.CreateBuilder(args);
1   using FirebaseAdmin;

264 // ============================
265 // 7) Middlewares de arranque
266 // ============================
267 if (app.Environment.IsDevelopment())
268 {
269     app.UseSwagger();
270     app.UseSwaggerUI();
271 }
272
273 // Prueba de conexión a la base de datos al iniciar
274 using (var scope = app.Services.CreateScope())
275 {
276     var dbFactory = scope.ServiceProvider.GetRequiredService<IDbConnectionFactory>();
277     try
278     {
279         using var conn = (SqlConnection)dbFactory.CreateConnection();
280         conn.Open();
281         Console.WriteLine("✅ Conectado a la base de datos");
282     }
283     catch (Exception ex)
284     {
285         Console.WriteLine("❌ Error al conectar a la base de datos:");
286         Console.WriteLine(ex.Message);
287     }
288 }
289
290 app.UseCors("AllowFrontend");
291
```

server > Secrets > HashPassword.cs > ...

```
1   using BCrypt.Net;
2
    2 references
3   public static class PasswordHelper
4   {
        2 references
5       public static string HashPassword(string password)
6       {
7           return BCrypt.Net.BCrypt.HashPassword(password);
8       }
9
        0 references
10      public static bool VerifyPassword(string plainPassword, string hashedPassword)
11      {
12          return BCrypt.Net.BCrypt.Verify(plainPassword, hashedPassword);
13      }
14  }
15
```

server > Services > AuthService.cs > ...

```
13      public class AuthService
            3 references
25          public async Task<AuthLoginResult> LoginWithIdTokenAsync(
26              string idTokenIn,
27              string defaultRole = "cliente",
28              int? clientId = null)
29          {
30              if (string.IsNullOrWhiteSpace(idTokenIn))
31                  throw new ArgumentException("Falta ID token de Firebase.", nameof(idTokenIn));
32
33              // 1) Verificar ID token recibido y revocación
34              var decoded = await FirebaseAuth.DefaultInstance.VerifyIdTokenAsync(idTokenIn, /*checkRevoked:*/ true);
35              var uid   = decoded.Uid;
36              var email = decoded.Claims.TryGetValue("email", out var emailObj) ? emailObj?.ToString() : null;
37
38              if (string.IsNullOrWhiteSpace(uid) || string.IsNullOrWhiteSpace(email))
39                  throw new InvalidOperationException("Token inválido o incompleto (sin uid/email).");
40
41              // 2) Asegurar custom claims (role + clientId)
42              var userRecord = await FirebaseAuth.DefaultInstance.GetUserAsync(uid);
43              var claims = userRecord.CustomClaims != null
44                  ? new Dictionary<string, object>(userRecord.CustomClaims)
45                  : new Dictionary<string, object>();
46
47              if (!claims.TryGetValue("role", out var roleObj) || string.IsNullOrWhiteSpace(roleObj?.ToString()))
48                  claims["role"] = defaultRole;
49
50              if (clientId.HasValue)
51                  claims["clientId"] = clientId.Value;
52
53              await FirebaseAuth.DefaultInstance.SetCustomUserClaimsAsync(uid, claims);
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTGRESQL QUERY RESULTS

server > Services > IDbConnectionFactory.cs > ...

```
1   using System.Data;
2
3   namespace Spartans.Services
4   {
        3 references
5       public interface IDbConnectionFactory
6       {
            2 references
7           IDbConnection CreateConnection();
8       }
9   }
```

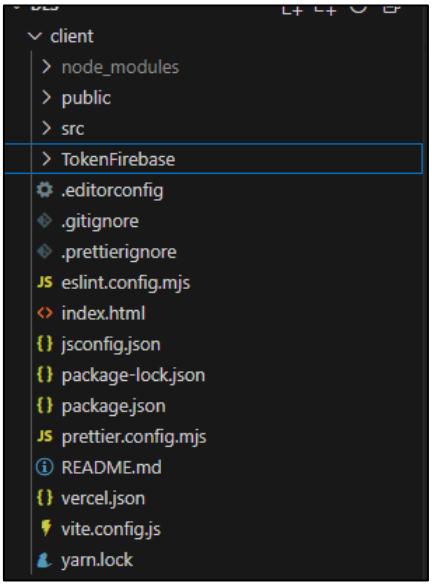
server > Services > SqlConnectionFactory.cs > ...

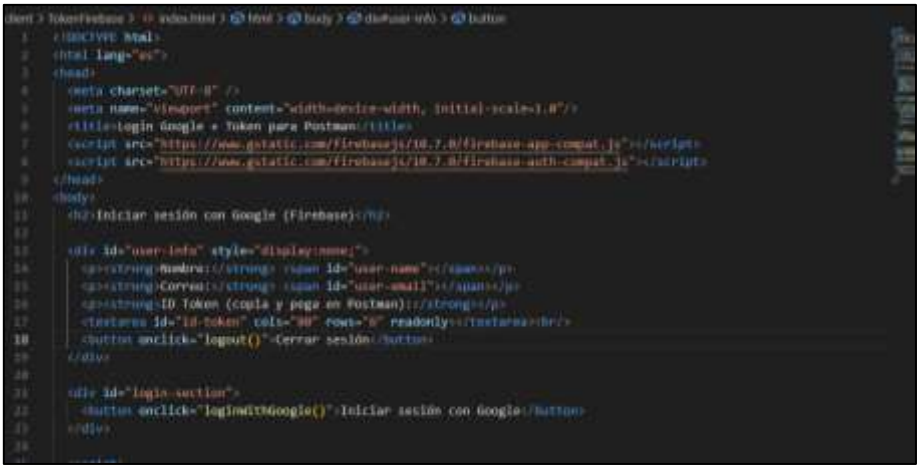
```cs
using Microsoft.Extensions.Configuration;
using Microsoft.Data.SqlClient;
using System.Data;

namespace Spartans.Services
{
    2 references
    public class SqlConnectionFactory : IDbConnectionFactory
    {
        2 references
        private readonly string _connectionString;

        1 reference
        public SqlConnectionFactory(string connectionString)
        {
            _connectionString = connectionString ?? throw new ArgumentNullException(nameof(connectionString));
        }

        2 references
        public IDbConnection CreateConnection()
        {
            return new SqlConnection(_connectionString);
        }
    }
}
```

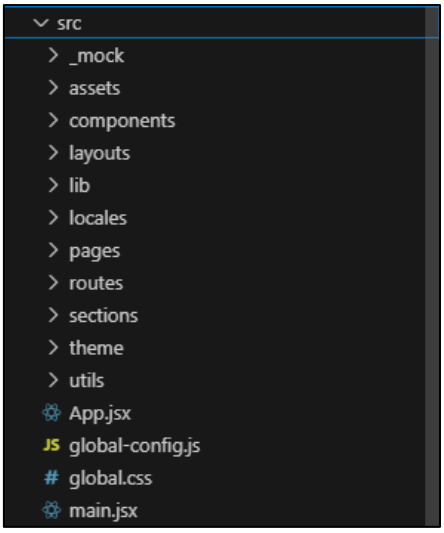


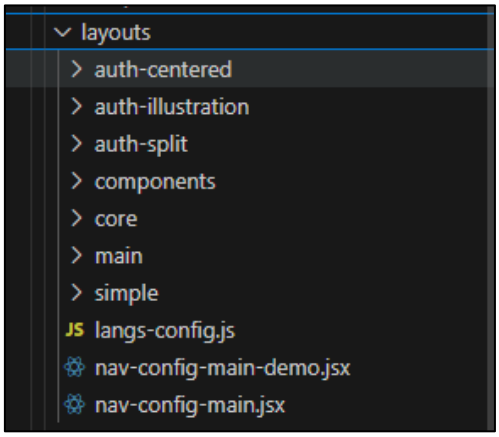

client > TokenFirebase > index.html > html > body > div#user-info > button
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Login Google + Token para Postman</title>
<script src="https://www.gstatic.com/firebasejs/10.7.0/firebase-app-compat.js"></script>
<script src="https://www.gstatic.com/firebasejs/10.7.0/firebase-auth-compat.js"></script>
</head>
<body>
<h2>Iniciar sesión con Google (Firebase)</h2>

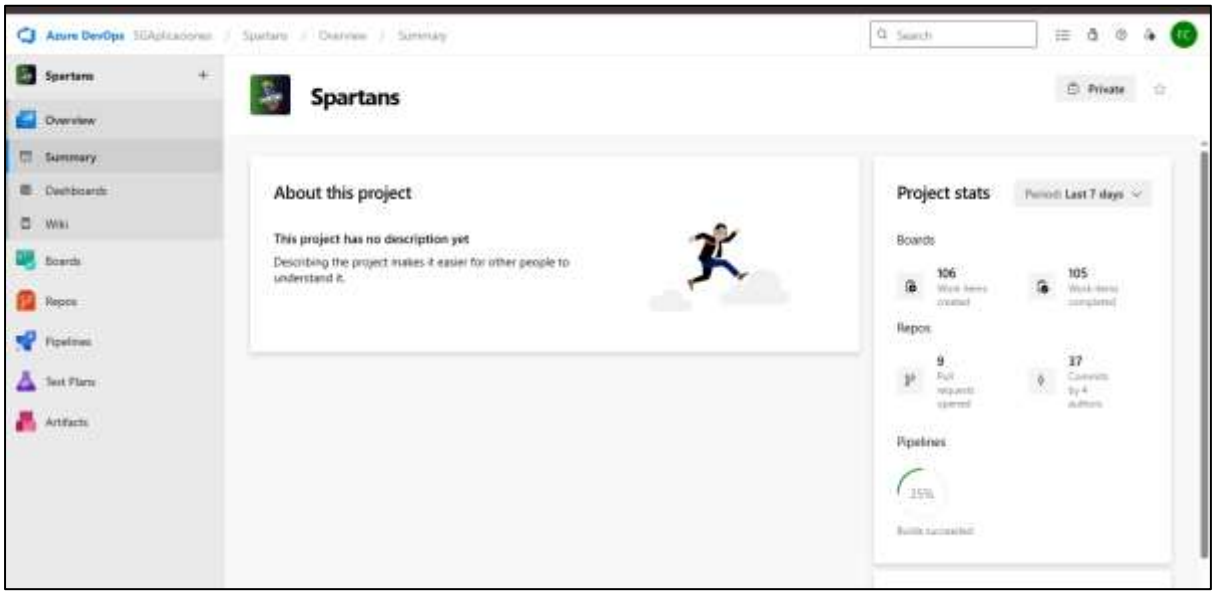
Nombre:
Correo:
ID Token (copia y pega en Postman):
<textarea id="id-token" cols="80" rows="5" readonly></textarea>
<button onclick="logout()">Cerrar sesión</button>

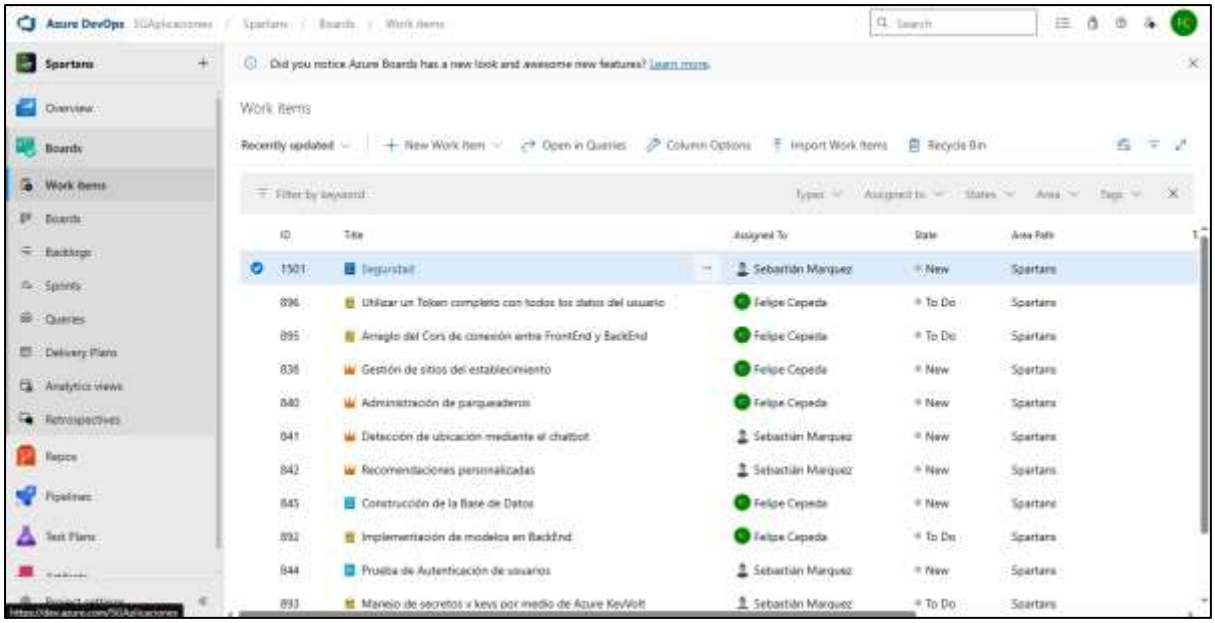
<button onclick="loginWithGoogle()">Iniciar sesión con Google</button>

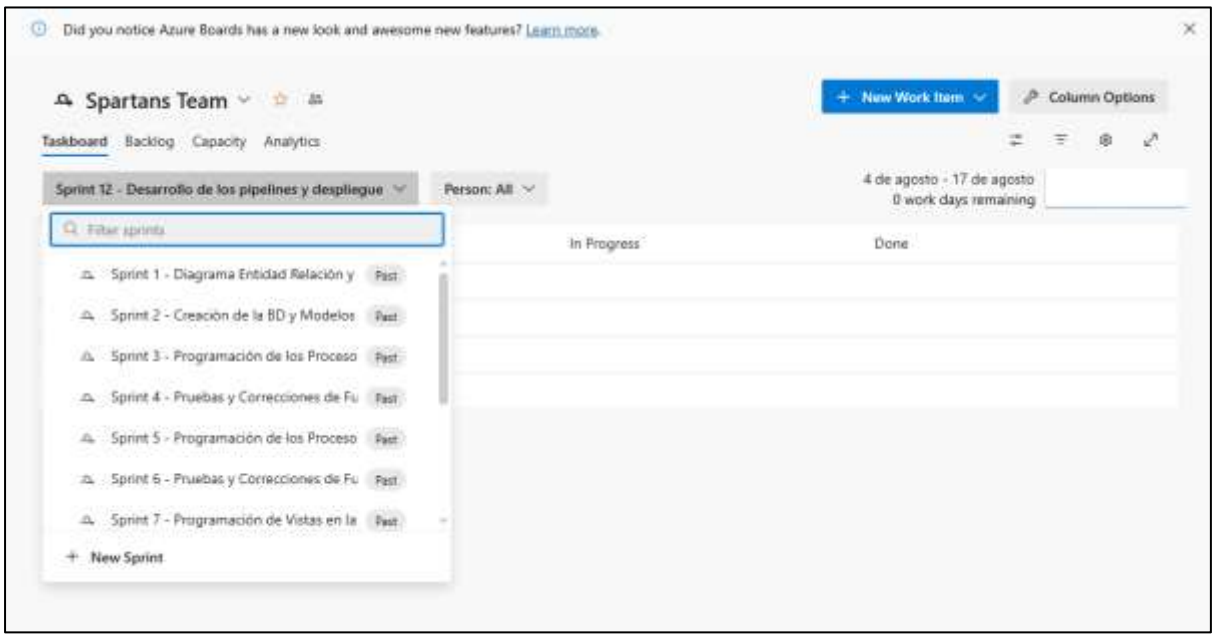
</body>
</html>
<script>
const firebaseConfig = {
apiKey: "AIzaSyDOCAbC123dEf456GhI789jKl01-mNoPqRs",
authDomain: "ejemplo-demo.firebaseapp.com",
projectId: "ejemplo-demo",
storageBucket: "ejemplo-demo.appspot.com",
messagingSenderId: "123456789012",
appId: "1:123456789012:web:abcdef1234567890abcdef"
};
firebase.initializeApp(firebaseConfig);
function loginWithGoogle() {
const provider = new firebase.auth.GoogleAuthProvider();
firebase.auth().signInWithPopup(provider)
.then((result) => {
const user = result.user;
document.getElementById("user-name").textContent = user.displayName;
document.getElementById("user-email").textContent = user.email;
user.getIdToken().then((idToken) => {
document.getElementById("id-token").value = idToken;
});
document.getElementById("user-info").style.display = "block";
document.getElementById("login-section").style.display = "none";
})
.catch((error) => {
alert("Error: " + error.message);
});
}
function logout() {
firebase.auth().signOut().then(() => {
document.getElementById("user-info").style.display = "none";
document.getElementById("login-section").style.display = "block";
document.getElementById("id-token").value = "";
});
}
</script>
```
```

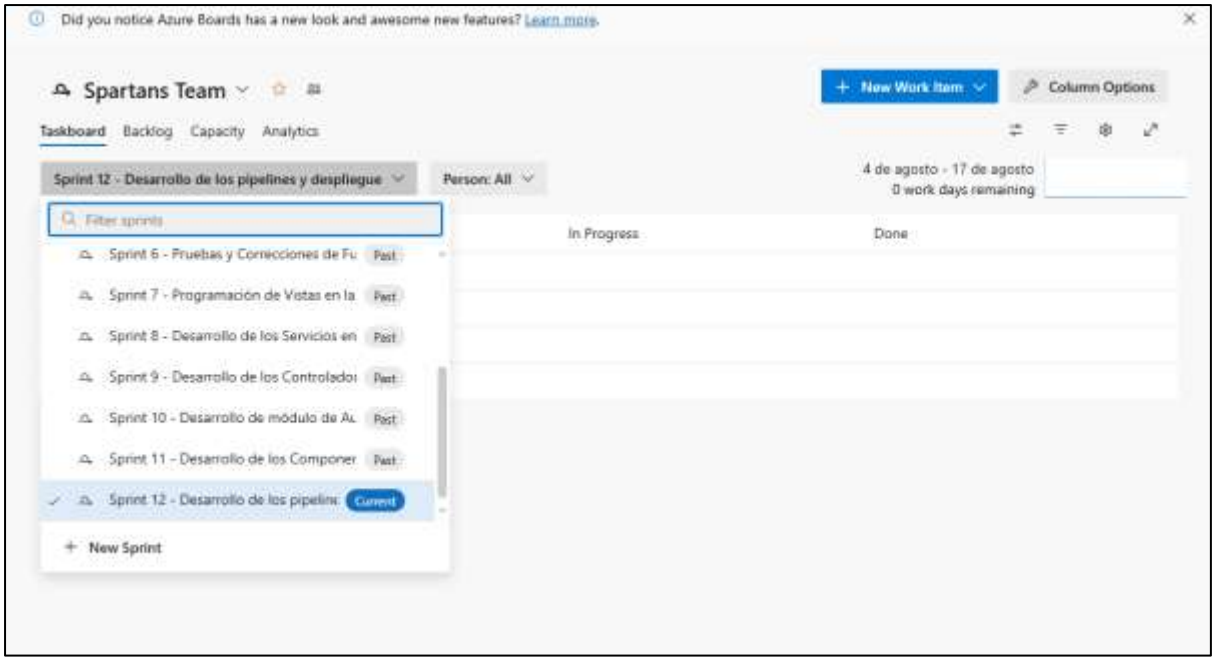

src
_mock
assets
components
layouts
lib
locales
pages
routes
sections
theme
utils
App.jsx
global-config.js
global.css
main.jsx

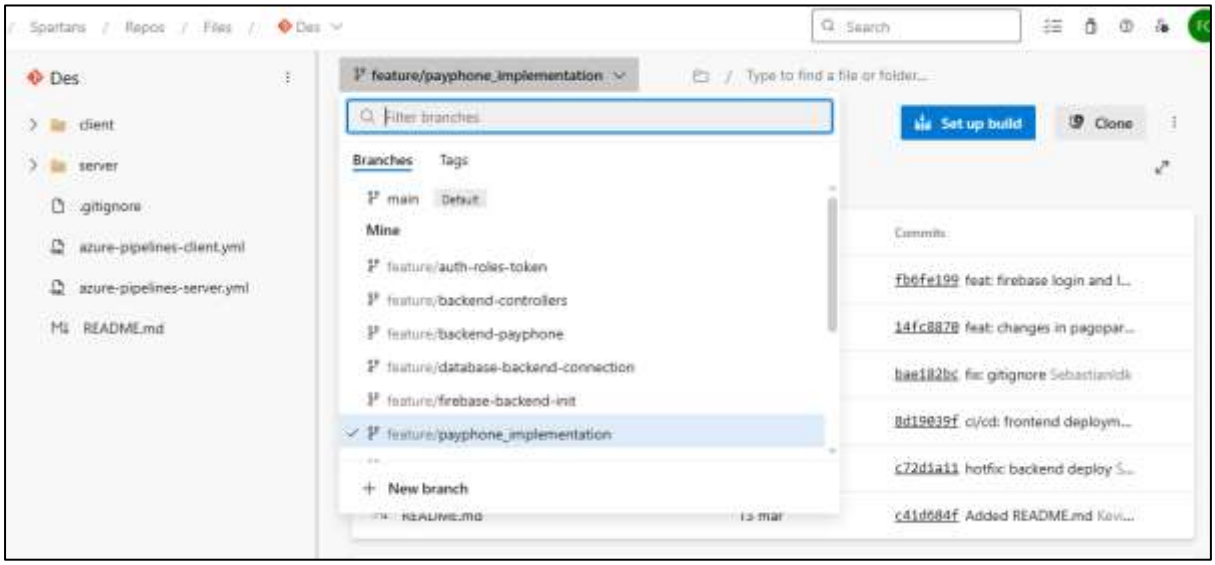

layouts
auth-centered
auth-illustration
auth-split
components
core
main
simple
langs-config.js
nav-config-main-demo.jsx
nav-config-main.jsx


Spartans
Private
About this project
This project has no description yet
Describing the project makes it easier for other people to understand it.
Project stats
Boards
Repos
Pipelines


Azure DevOps
Spartans
Boards
Work items
Did you notice Azure Boards has a new look and awesome new features? Learn more.
Work Items
Recently updated
New Work Item
Open in Queries
Column Options
Import Work Items
Recycle Bin
ID
Title
Assigned To
State
Area Path
1501 Seguridad Sebastián Marquez New Spartans
836 Utilizar un Token completo con todos los datos del usuario Felipe Cepeda To Do Spartans
895 Arreglo del Cors de conexión entre FrontEnd y BackEnd Felipe Cepeda To Do Spartans
838 Gestión de sitios del establecimiento Felipe Cepeda New Spartans
840 Administración de parqueaderos Felipe Cepeda New Spartans
841 Detección de ubicación mediante el chatbot Sebastián Marquez New Spartans
842 Recomendaciones personalizadas Sebastián Marquez New Spartans
845 Construcción de la Base de Datos Felipe Cepeda New Spartans
892 Implementación de modelos en BackEnd Felipe Cepeda To Do Spartans
844 Prueba de Autenticación de usuarios Sebastián Marquez New Spartans
893 Manejo de secretos y keys por medio de Azure KeyVolt Sebastián Marquez To Do Spartans


Did you notice Azure Boards has a new look and awesome new features? Learn more.
Spartans Team
New Work Item
Column Options
Taskboard
Backlog
Capacity
Analytics
Sprint 12 - Desarrollo de los pipelines y despliegue
Person: All
4 de agosto - 17 de agosto
0 work days remaining
Filter sprints
In Progress
Done
Sprint 1 - Diagrama Entidad Relación y Past
Sprint 2 - Creación de la BD y Modelos Past
Sprint 3 - Programación de los Proceso Past
Sprint 4 - Pruebas y Correcciones de Fu Past
Sprint 5 - Programación de los Proceso Past
Sprint 6 - Pruebas y Correcciones de Fu Past
Sprint 7 - Programación de Vistas en la Past
New Sprint


Did you notice Azure Boards has a new look and awesome new features? Learn more.
Spartans Team
New Work Item
Column Options
Taskboard
Backlog
Capacity
Analytics
Sprint 12 - Desarrollo de los pipelines y despliegue
Person: All
4 de agosto - 17 de agosto
0 work days remaining
Filter sprints
In Progress
Done
Sprint 6 - Pruebas y Correcciones de Fu
Past
Sprint 7 - Programación de Vistas en la
Past
Sprint 8 - Desarrollo de los Servicios en
Past
Sprint 9 - Desarrollo de los Controlador
Past
Sprint 10 - Desarrollo de módulo de Au
Past
Sprint 11 - Desarrollo de los Componer
Past
Sprint 12 - Desarrollo de los pipelines
Current
New Sprint


Spartans / Repos / Files / Des
Search
Des
feature/payphone_implementation
Type to find a file or folder...
client
server
.gitignore
azure-pipelines-client.yml
azure-pipelines-server.yml
README.md
Filter branches
Set up build
Clone
Branches
Tags
main
Default
Mine
feature/auth-roles-token
feature/backend-controllers
feature/backend-payphone
feature/database-backend-connection
feature/firebase-backend-init
feature/payphone_implementation
New branch
Commits
fb6fe199 feat: firebase login and l...
14fc8870 feat: changes in pagopar...
bae182bc fix: gitignore Sebastianldk
8d19039f ci/cd: frontend deploym...
c72d1a11 hotfix: backend deploy S...
c41d684f Added README.md Kevi...


Spartans / Pipelines
Search
Pipelines
New pipeline
Recent
All
Runs
Filter pipelines
Recently run pipelines
Pipeline
Last run
azure-pipelines-server
#20250817.1 • Updated ReservasPage.jsx
7h ago
2m 10s