Seminario de Aplicación Profesional

Materia: Seminario de Aplicación Profesional

Docentes: Alejandro Roberto Santorio, Matias Pablo Banega

Alumno: Cristian David Galarza

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Índice:

1 Aspectos descriptivos de la solución tecnológica.....4
1.1 Requerimientos:.....4
1.2 Requerimientos no funcionales (RNF).....5
1.3 Requerimientos técnicos (RT).....6
1.4 Requerimientos de entorno (RE).....7
1.5.1 Tecnologías principales utilizadas en el proyecto:.....8
1.6.1 Prototipos:.....9
Formulario de login:.....9
Formulario de olvide contraseña:.....10
Formulario de Crear Cuenta:.....11
Formulario de crear garage:.....12
Formulario de agregar imagen:.....13
Vista del administrador de mis garages:.....14
1.7 Diagramas UML.....15
1.7.1 Diagrama de casos de uso.....15
1.7.2 Especificación de casos de uso.....16
Caso de uso 1: Registrarse.....16
Caso de uso 2 Enviar mail de registro.....17
Caso de uso 3 Confirmar cuenta.....18
Caso de uso 4 Iniciar Sesión.....19
Caso de uso 5 Validar credenciales.....20
Caso de uso 6: Olvidé contraseña.....21
Caso de uso 7: Enviar mail de reseteo de contraseña.....22
Caso de uso 8 Restablecer contraseña.....23
Caso de uso 9: Ver mis garages.....24
Caso de uso 10: Crear garage.....25
Caso de uso 11: Editar garage.....27
Caso de uso 12: Eliminar garage.....28
1.7.3 Diagrama de clases:.....29
1.7.4 Diagramas de secuencia:.....32
1.7.4.1 Diagrama de secuencia core 1:.....32
Iniciar sesión:.....32
Olvide Contraseña:.....33
Restablecer contraseña:.....33
1.7.4.2 Diagrama de de secuencia core 2 crear propiedad y agregar imagen:....34
Crear propiedad.....34

Agregar imagen..... 35
1.7.5 Patrones de diseño utilizados en el proyecto y su explicación..... 36
Patrón MVC: Modelo-Vista-Controlador..... 36
Patron singleton en el proyecto..... 47
Patrón Observer en el proyecto:..... 51
Patrón Chain of Responsibility - Middleware de Express..... 59
Anexos..... 63
Bibliografía consultada..... 63

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

SAP - Parte Tecnológica

1 Aspectos descriptivos de la solución tecnológica

1.1 Requerimientos:

Requerimientos funcionales (RF)

RF1 – Registro de usuario
Formulario de alta que recibe nombre, email y contraseña; las valida y almacena en la tabla users de MySQL, con la contraseña hasheada usando bcrypt.

RF2 – Inicio de sesión
Login que verifica email/contraseña, genera un JWT

RF3 – Recuperación de contraseña
Flujo “Olvidé mi contraseña” que crea un token, lo envía por email con nodemailer, lo valida al recibir la petición y permite resetear la contraseña.

RF4 – CRUD de propiedades (“garages”)
Páginas server-side para crear, editar y eliminar propiedades con campos como título, descripción, flags (techado, alarma, expensas), precio, ubicación y URL de imagen.

RF5 – Geolocalización
En la vista de creación/edición se incorpora un mapa interactivo (Leaflet o Google Maps) donde el usuario marca la ubicación (latitud/longitud), que luego se guardan en MySQL.

RF6 – Subida de imágenes
El sistema debe permitir adjuntar una imagen a cada propiedad - garage, validando formato y tamaño; y guardarla para poder mostrarla en la ficha de la propiedad.

RF7 – Protecciones de acceso
Middleware protegerRuta comprueba la existencia y validez del JWT y la existencia del usuario en BD antes de permitir acceso a rutas de perfil o de propiedades.

RF8 – Logging y notificaciones internas
Patrón Observer tras eventos críticos (registro, login, creación/borrado) se emiten señales que los listeners capturan y registran en logs

1.2 Requerimientos no funcionales (RNF)

RNF1 – Seguridad
- Hashing de contraseñas con bcrypt.
- Protección CSRF en todo el sitio con csurf + cookie-parser.
- HTTPS obligatorio en producción.

RNF2 – Rendimiento

- Tiempos de respuesta de vistas < 200 ms en endpoints críticos.

- Uso de caché a nivel de servidor para listados frecuentes si es necesario.

RNF3 – Escalabilidad
Arquitectura modular en capas (MVC) para facilitar un futuro split en microservicios si crece la carga.

RNF4 – Disponibilidad
Backups diarios de la base MySQL

RNF5 – Mantenibilidad
Tests unitarios e integración

RNF6 – Usabilidad
- Vistas responsivas con Tailwind CSS.
- Validaciones en cliente y servidor, y mensajes de feedback claros.

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

1.3 Requerimientos técnicos (RT)

RT1 – Stack principal

- Node.js (ES Modules) + Express.js como servidor.
- Pug para renderizado server-side de plantillas.
- Secualize como ORM
- JavaScript en cliente para interacción con el mapa y validaciones básicas.

RT2 – Base de datos

- MySQL mediante Sequelize.
- Pool de conexiones configurado como Singleton en config/db.js (única instancia durante toda la ejecución).

RT3 – Seguridad y sesiones

- jsonwebtoken para emitir/verificar JWT.
- csurf para CSRF y cookie-parser para leer cookies.

RT4 – Gestión de ficheros

- Multer para multipart/form-data(manejos de archivos) y control de tamaño/formato.
- Almacenamiento local de imágenes en public/uploads/ con identificadores únicos.

RT5 – Emails

Nodemailer configurado vía SMTP (.env) para envíos de token de recuperación y notificaciones, en la etapa de testing, luego en produccion elegir un servicio como Gmail u otro

RT6 – Validación

express-validator en rutas críticas para validar y sanitizar inputs.

RT7 – Front-end build

- Tailwind CSS
- PugJS
- Webpack para empaquetar src/js/ y estilos.

RT8 – Logging y eventos
Módulo nativo EventEmitter para desacoplar la lógica de logs (observer) y escribir en fichero.

RT9 – Herramientas de desarrollo

- nodemon para autoreload en dev.
- dotenv para cargar variables de entorno.

1.4 Requerimientos de entorno (RE)

RE1 – Entornos diferenciados

- Desarrollo: npm run dev (nodemon) con MySQL local de prueba.
- Producción: contenedores Docker Compose (app y MySQL).

RE2 – Variables de entorno
archivo .env con DB_HOST, DB_USER, DB_PASS, DB_NAME, JWT_SECRET, EMAIL_HOST, EMAIL_USER, EMAIL_PASS, claves de Map API. Recordar incluir en el .gitignore antes de subir el proyecto a github

RE3 – Contenerización
Dockerfile y docker-compose que definen la aplicación, la base de datos y, opcionalmente, un servicio de caché/cola (Redis).

RE4 – Control de versiones

- GitHub.

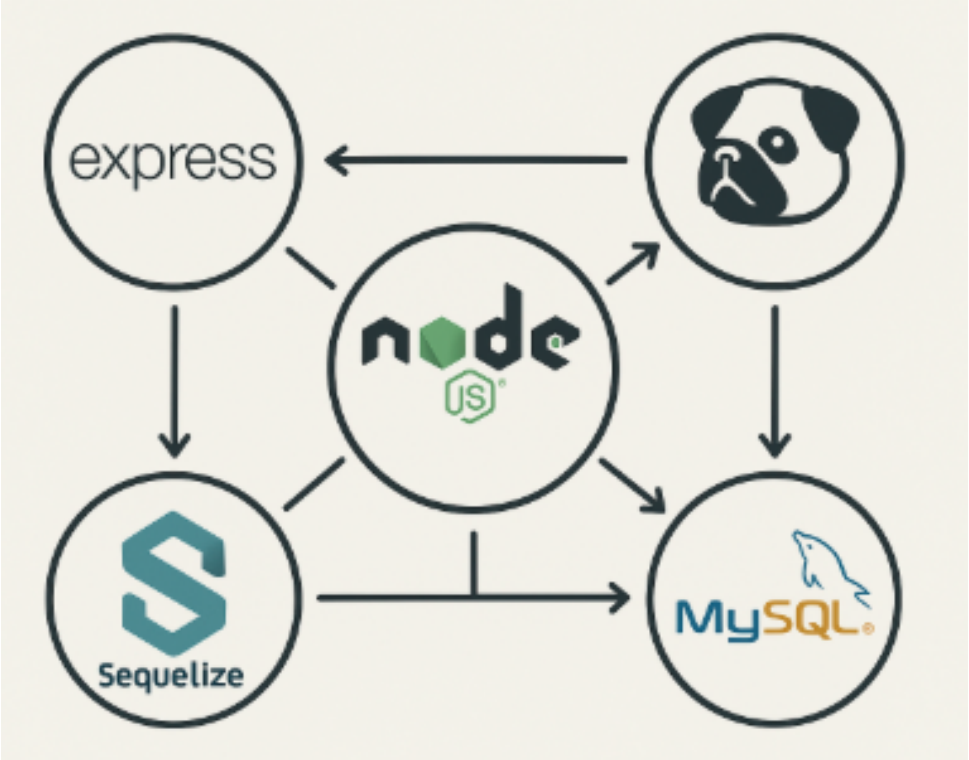
RE5 – Documentación
README.md actualizado con pasos de instalación, variables de entorno, scripts npm y guía de despliegue.

RE6 – Backup y recuperación

Scripts/cron jobs diarios de MySQL, retención mínima de 7 días y alertas en caso de fallo.

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

1.5.1 Tecnologías principales utilizadas en el proyecto:



Esta imagen fue generada con IA

Describiendo las tecnologías principales utilizadas:

Node.js:

- Runtime JavaScript en servidor.
- Ejecuta toda la lógica backend y gestiona el flujo de peticiones.

Express.js:

- Framework ligero para crear rutas y middleware HTTP.
- Define endpoints (registro, login, garages…) y aplica autenticación/CSRF.

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Pug.js:

- Motor de plantillas que genera HTML a partir de sintaxis sencilla.
- Renderiza las vistas server-side (formularios, listados y detalles de propiedades).

Sequelize:

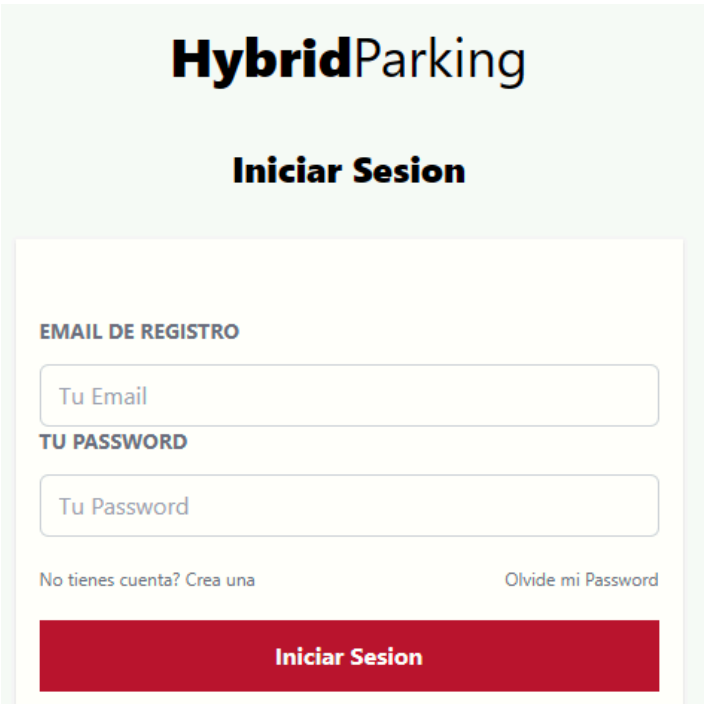
- ORM que traduce operaciones JS en consultas SQL.
- Modela tablas (User, Property), hace CRUD y gestiona el pool de MySQL.

MySQL:

- Base de datos relacional.
- Almacena usuarios, contraseñas hasheadas, propiedades, ubicaciones e imágenes.

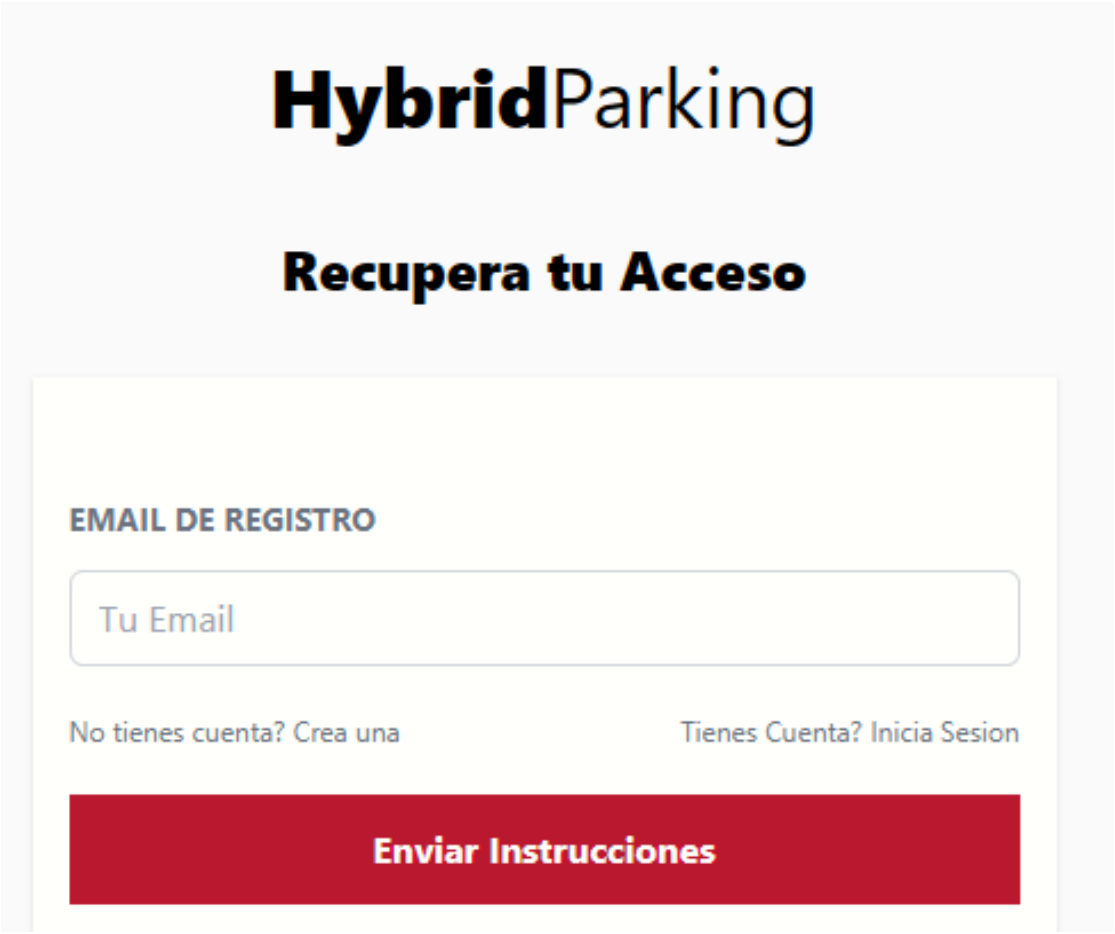
1.6.1 Prototipos:

Formulario de login:



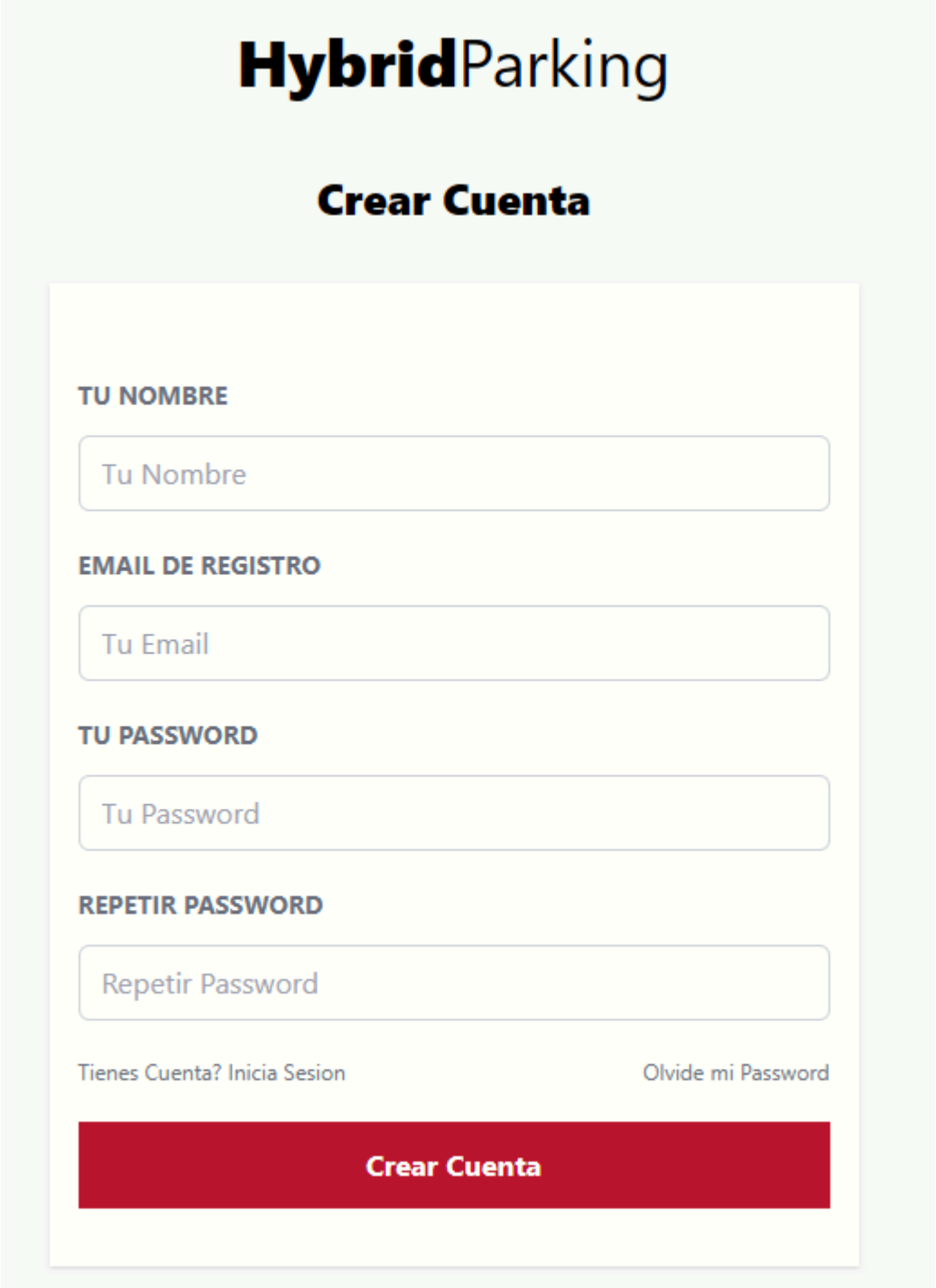
	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Formulario de olvide contraseña:



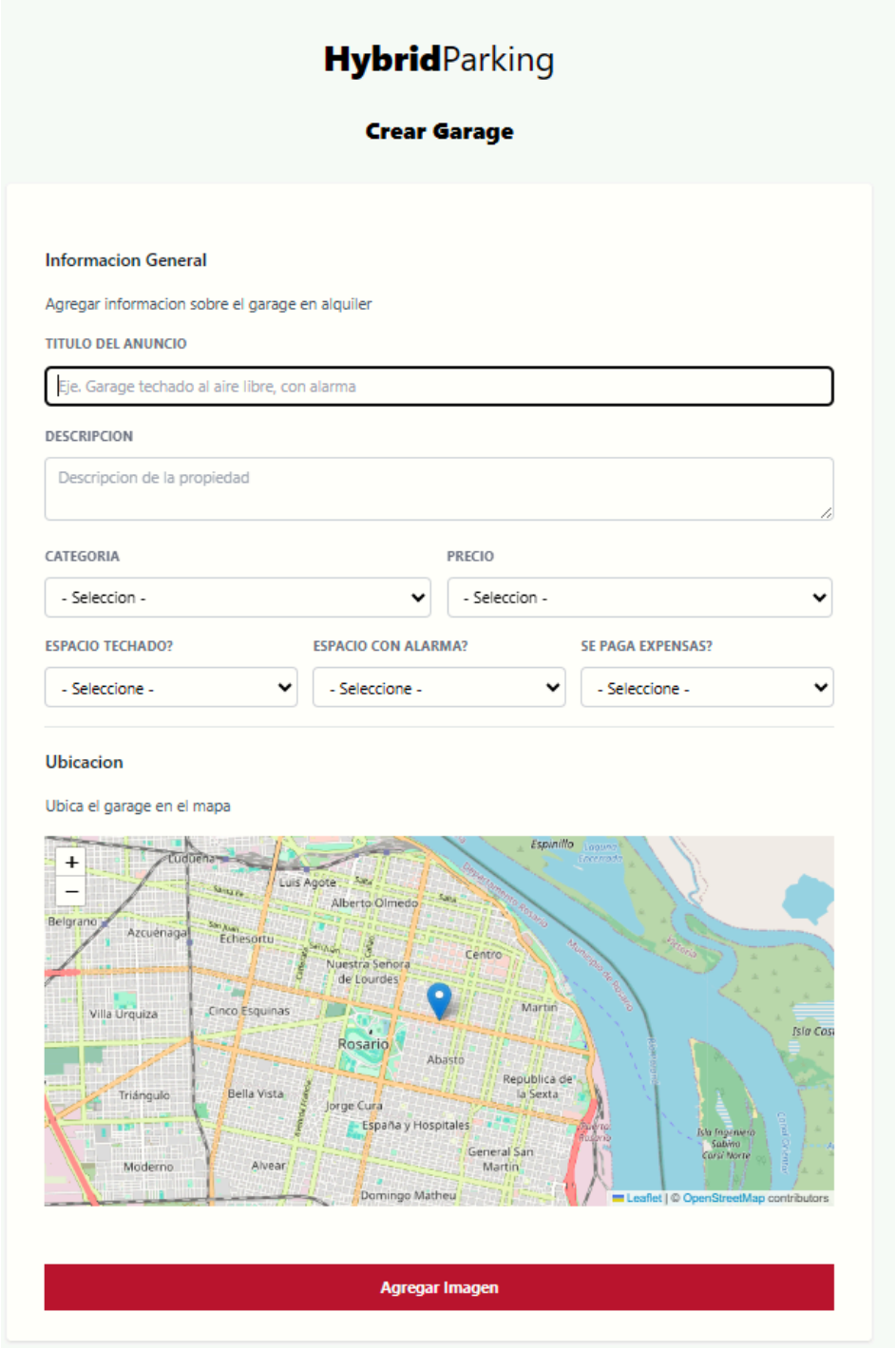
	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Formulario de Crear Cuenta:



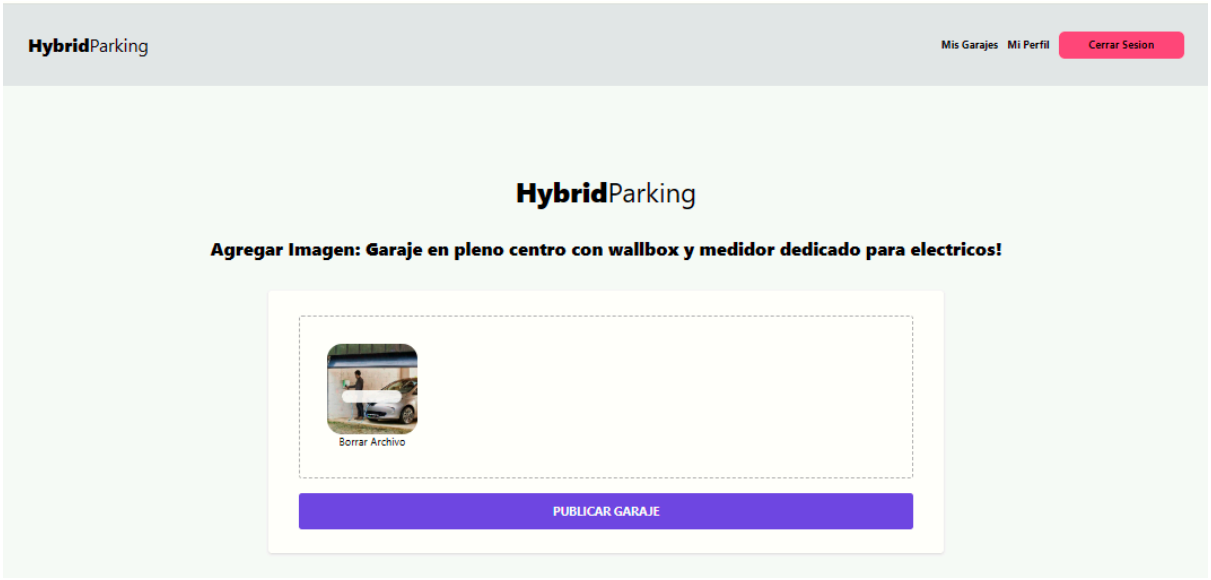
	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Formulario de crear garage:



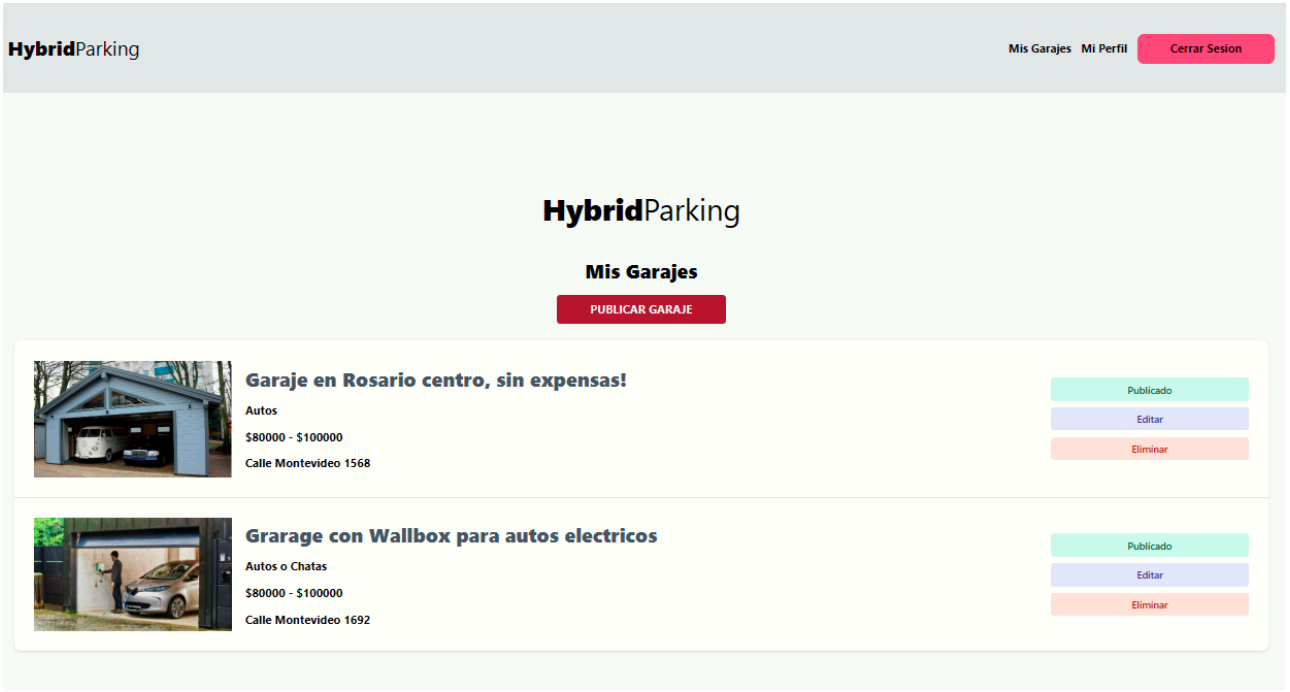
	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Formulario de agregar imagen:



	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Vista del administrador de mis garages:

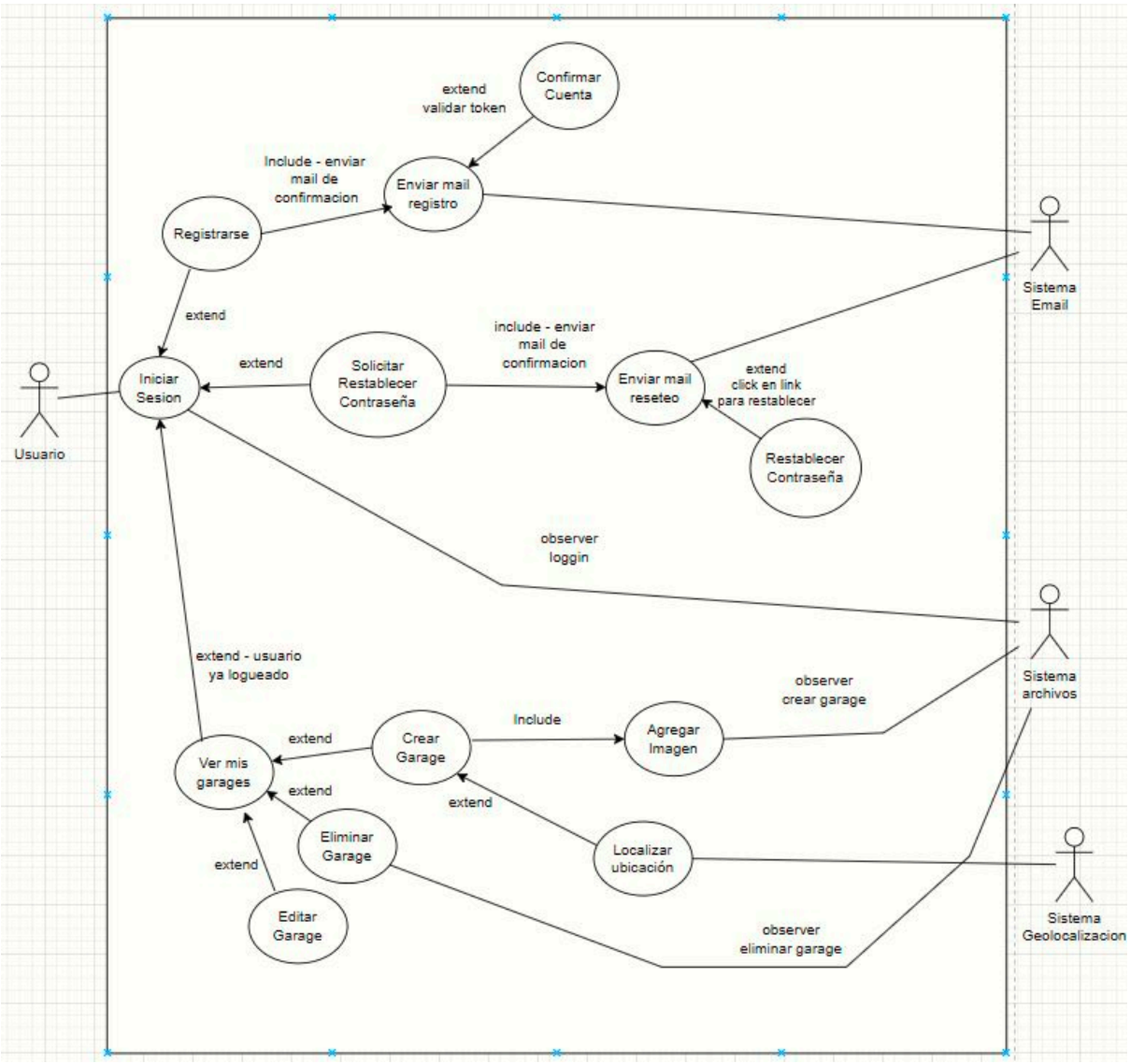


	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

1.7 Diagramas UML

Diagramas para los casos core de sesión anti Cross Site Request Forgery (CSRF) + patron singleton y el caso ver mis garajes con el patrón Observer que también se emplea al iniciar sesión un usuario

1.7.1 Diagrama de casos de uso



	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

1.7.2 Especificación de casos de uso

Caso de uso 1: Registrarse

	CASO DE USO
Código	CU1
Nombre	Registrarse
Referencias	
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Crear una nueva cuenta en la plataforma.
Actor/es	Usuario sin cuenta
Precondición	El usuario no tiene una cuenta activa ni sesión iniciada
CU-Extensión	
Curso Básico	
1	El Usuario accede al formulario de registro y proporciona nombre, e-mail y contraseña.
2	El sistema valida los datos
3	El usuario hace click en el botón aceptar
4	El sistema envía mail de registro
5	El sistema muestra un mensaje para que el usuario revise su bandeja de entrada
Curso Alternativo	
1	El usuario cierra la ventana de registro
Post.Condición	El usuario recibe un mail de confirmación

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 2 Enviar mail de registro

	CASO DE USO
Código	CU2
Nombre	Enviar mail de registro
Referencias	
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Enviar al nuevo usuario un correo con enlace de confirmación.
Actor/es	Sistema de Mails
Precondición	El usuario no tiene una cuenta activa ni sesión iniciada
CU-Extensión	CU1
Curso Básico	
1	El Usuario accede al formulario de registro y proporciona nombre, e-mail y contraseña.
2	El sistema valida los datos
3	El usuario hace click en el botón aceptar
4	El sistema envía mail de registro
5	El sistema muestra un mensaje para que el usuario revise su bandeja de entrada
Curso Alternativo	
1	El usuario cierra la ventana de registro
Post.Condición	El usuario recibe un mail de confirmación

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 3 Confirmar cuenta

	CASO DE USO
Código	CU3
Nombre	Confirmar cuenta
Referencias	
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Activar la cuenta validando el token recibido por mail
Actor/es	Usuario no logueado
Precondición	Usuario ha recibido el e-mail de confirmación; tiene token válido.
CU-Extensión	CU4
Curso Básico	
1	Usuario hace clic en el enlace que le llegó por email
2	El sistema extrae y valida el token.
3	Si es correcto, el sistema cambia el estado de la cuenta a “confirmada”.
Curso Alternativo	
1	El usuario no valida la cuenta, no ve el mail
Post.Condición	El usuario es redirigido a la vista de iniciar sesión para ingresar sus credenciales creadas y poder loguearse

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 4 Iniciar Sesión

	CASO DE USO
Código	CU4
Nombre	Iniciar Sesión
Referencias	CU3 (redirecciona a este CU)
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Autenticar al usuario para acceder a funciones protegidas
Actor/es	Usuario
Precondición	Cuenta confirmada; no hay sesión iniciada
CU-Extensión	CU5
Curso Básico	
1	Usuario introduce e-mail y contraseña.
2	El sistema valida credenciales (compara con DB),si son correctas, genera JWT y lo envía en cookie.
3	Si es correcto, el sistema cambia el estado de la cuenta a “confirmada”.
4	El sistema registra en log la sesión iniciada (Observer “userLoggedIn”)
Curso Alternativo	
1	El usuario no recuerda su password o no está registrado
Post.Condición	Sesión activa; el usuario puede acceder a su panel de garajes cargados

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 5 Validar credenciales

	CASO DE USO
Código	CU5
Nombre	Validar credenciales
Referencias	CU4
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Comprobar que e-mail y password coinciden con un usuario confirmado
Actor/es	Usuario
Precondición	Invocado desde el CU Iniciar sesión
CU-Extensión	
Curso Básico	
1	El sistema busca al usuario por e-mail en la base de datos
2	El sistema compara hash de contraseña
3	Si es correcto, el sistema cambia el estado de la cuenta a “confirmada”.
4	El sistema devuelve “válido” o “inválido”.
Curso Alternativo	
1	El usuario invalido
Post.Condición	Usuario validado exitosamente

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 6: Olvidé contraseña

	CASO DE USO
Código	CU6
Nombre	Olvide contraseña
Referencias	
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Iniciar proceso de recuperación de contraseña.
Actor/es	Usuario
Precondición	Usuario no ha iniciado sesión.
CU-Extensión	CU7
Curso Básico	
1	El usuario ingresa su e-mail.
2	El sistema enviar mail reseteo de pass por email
3	El sistema iindica “Revisa tu correo para restablecer contraseña”.
Curso Alternativo	
1	No se envía en token
Post.Condición	Token de reseteo generado y enviado por e-mail.

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 7: Enviar mail de reseteo de contraseña

	CASO DE USO
Código	CU7
Nombre	Enviar mail de reseteo de contraseña
Referencias	CU6
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Enviar enlace de recuperación con token.
Actor/es	Sistema de Email
Precondición	El usuario había olvidado su contraseña
CU-Extensión	CU8
Curso Básico	
1	El sistema recibe email y token.
2	El sistema envía correo con enlace que incluye token de reseteo.
Curso Alternativo	
1	No se envia el email
Post.Condición	El sistema envía el correo con enlace que incluye token de reseteo.

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 8 Restablecer contraseña

	CASO DE USO
Código	CU8
Nombre	Restablecer contraseña
Referencias	CU7
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Definir una nueva contraseña usando el token
Actor/es	Usuario
Precondición	El usuario hizo clic en enlace de reseteo; token aún es válido.
CU-Extensión	
Curso Básico	
1	El usuario accede al formulario de nueva contraseña.
2	El usuario ingresa y confirma la nueva clave.
3	El sistema valida el token y actualiza el hash en la base de datos.
Curso Alternativo	
1	El usuario recuerda su contraseña y cierra el formulario
Post.Condición	Contraseña cambiada, el usuario puede iniciar sesión con la nueva clave

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 9: Ver mis garages

	CASO DE USO
Código	CU9
Nombre	Ver mis garages
Referencias	CU5
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Mostrar al usuario el listado de sus propiedades (garages) publicadas.
Actor/es	Usuario
Precondición	Sesión iniciada
CU-Extensión	CU10
Curso Básico	
1	El sistema consulta las Propiedades asociadas al usuario..
2	El sistema renderiza la vista con la lista y botones para crear, editar o eliminar un garage con estado publicado.
Curso Alternativo	
1	No se envia el email
Post.Condición	El usuario ve su panel de garajes cargados

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 10: Crear garage

	CASO DE USO
Código	CU10
Nombre	Crear garage
Referencias	CU9
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Dar de alta un nuevo garage.
Actor/es	Usuario
Precondición	Sesión iniciada
CU-Extensión	CU11 Y CU12
Curso Básico	
1	Usuario abre formulario “Crear garage”.
2	El sistema lista categorías, precios
3	El usuario interactúa con Sistema Geolocalización para mostrar mapa y coordenadas de su garage
4	El usuario completa los campos faltantes (título, descripción, opciones techado/alarma/expensas, dirección)
5	El usuario hace click en el botón crear garage
6	El sistema valida campos
7	El sistema persiste el garage en base de datos
8	El sistema redirige a la vista de subir imagen del garage utilizando un

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

	sistema de archivos
9	El usuario sube una imagen, arrastrandola desde una carpeta o directamente haciendo click en el espacio de subir imagen
10	El sistema muestra una miniatura de que la imagen se cargo correctamente
11	El usuario hace click en el botón de subir imagen
12	El sistema persiste la imagen en el servidor de archivos
13	El sistema persiste en base de datos el nombre de la imagen,no el archivo, usando el id del usuario
14	El sistema genera un log de la nueva propiedad creada(Observer)
15	El sistema redirecciona a la vista de mis garajes mostrando el garage creado junto con la imagen asociada
Curso Alternativo	
1	El usuario cierra las ventanas, no quiere cargar imagen o crear garage
Post.Condición	Garage creado con sus parámetros e imagen

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Caso de uso 11: Editar garage

	CASO DE USO
Código	CU11
Nombre	Editar garage
Referencias	CU10
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Modificar los datos de un garage existente.
Actor/es	Usuario
Precondición	Sesión iniciada, en la vista de mis garages
CU-Extensión	
Curso Básico	
1	El usuario selecciona botón “Editar garage”.
2	El sistema busca en base de datos el garage y lista todos los valores en cada campo de la vista
3	El usuario modifica los campos necesarios o posición en el mapa
4	El usuario hace clic en guardar cambios
6	El sistema valida campos
7	El sistema persiste cambios en bases de datos
7	El sistema redirige a la vista de mis garajes mostrando las propiedades creadas y modificadas
Curso Alternativo	
1	No editan garajes

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

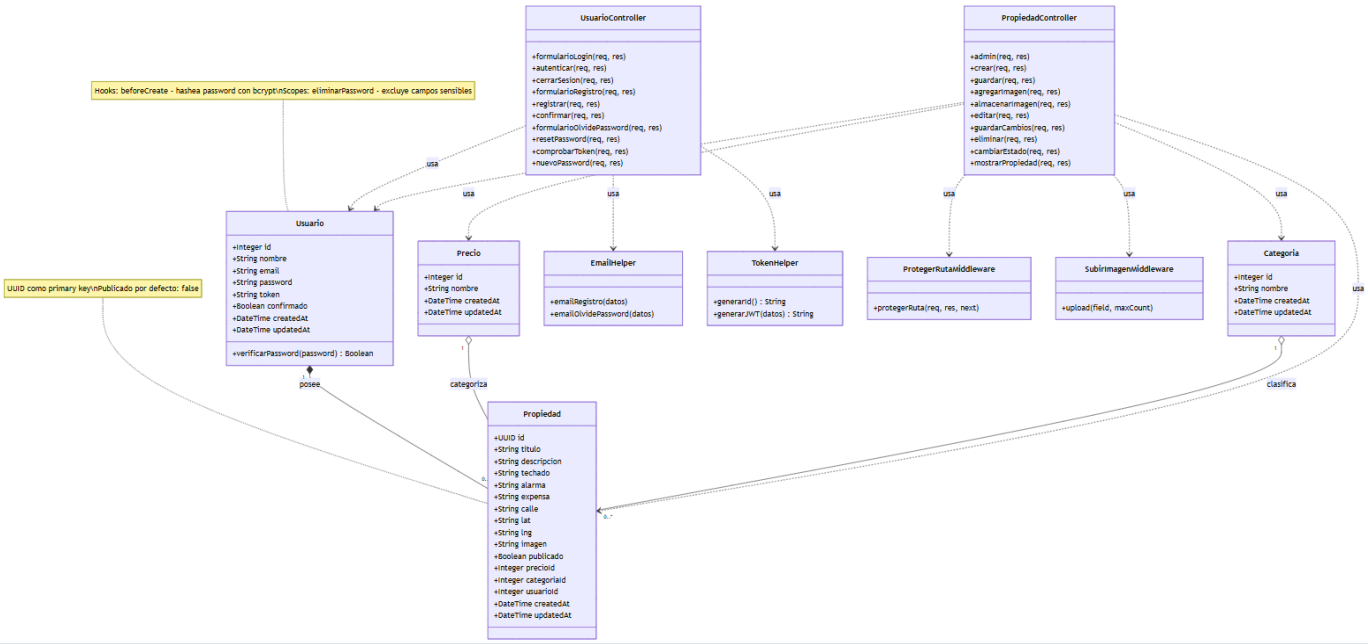
Post.Condición	El usuario edita su garage.

Caso de uso 12: Eliminar garage

	CASO DE USO
Código	CU12
Nombre	Eliminar garage
Referencias	CU10
Autor/es	Galarza
Revisor/es	Banega, Matías - Sartorio, Alejandro
Versión	1.0
Estado	Pendiente
Descripción	Borrar una propiedad y su imagen asociada.
Actor/es	Usuario
Precondición	Sesión iniciada, en la vista de mis garages
CU-Extensión	
Curso Básico	
1	El usuario hace clic en el botón de eliminar imagen
2	El sistema solicita confirmación
3	Si el usuario confirma, elimina el registro y la imagen del servidor.
4	El sistema registra un log de la propiedad eliminada (Observer)
Curso Alternativo	
3	El usuario no confirma eliminar el garage
Post.Condición	El usuario elimina su garage.

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

1.7.3 Diagrama de clases:



Descripción de Relaciones:

Relaciones entre Modelos (Entidades de Dominio):

1. Usuario *-- Propiedad (Composición 1:N)
   - Tipo: Composición (diamante negro relleno)
   - Cardinalidad: 1 Usuario posee 0 o muchas Propiedades
   - Justificación: Las propiedades PERTENECEN a un usuario y su ciclo de vida está controlado por él. Una propiedad sin usuario propietario no tiene sentido en el contexto del sistema.
   - Implementación: Propiedad.belongsTo(Usuario, { foreignKey: 'usuarioId' }) (models/index.js:10)
   - Foreign Key: usuarioId en tabla Propiedad

2. Precio o-- Propiedad (Agregación 1:N)
   - Tipo: Agregación (diamante blanco)
   - Cardinalidad: 1 Precio categoriza 0 o muchas Propiedades

- Justificación: Precio es un catálogo de rangos de valores (ej: "$0-100k", "$100k-500k"). Las propiedades referencian un precio pero pueden existir independientemente. Si se elimina un rango de precio del catálogo, las propiedades no se eliminan.
- Implementación: Propiedad.belongsTo(Precio, { foreignKey: 'precioId' }) (models/index.js:8)
- Foreign Key: precioId en tabla Propiedad

3. Categoría o-- Propiedad (Agregación 1:N)
- Tipo: Agregación (diamante blanco)
- Cardinalidad: 1 Categoría clasifica 0 o muchas Propiedades
- Justificación: Categoría es un catálogo de tipos (ej: "Casa", "Departamento", "Garage"). Similar a Precio, las propiedades solo referencian categorías del catálogo pero mantienen su existencia independiente.
- Implementación: Propiedad.belongsTo(Categoria, { foreignKey: 'categoriaId' }) (models/index.js:9)
- Foreign Key: categoriaId en tabla Propiedad

Relaciones de Dependencia (Controllers y Helpers):

4. Controllers ..> Modelos/Helpers (Dependencia)
- Tipo: Dependencia (línea punteada con flecha)
- Justificación: Los controladores USAN los modelos y helpers para realizar operaciones, pero no los contienen ni controlan su ciclo de vida.

Ejemplos:
UsuarioController ..> Usuario: El controller importa y usa el modelo Usuario
PropiedadController ..> Propiedad: El controller realiza operaciones CRUD sobre Propiedad
UsuarioController ..> EmailHelper: El controller usa el helper para enviar emails
PropiedadController ..> SubirImagenMiddleware: El controller usa el middleware Multer

<u>Componentes del Sistema:</u>

Modelos (Sequelize ORM):
Usuario: Gestión de usuarios con autenticación bcrypt
Propiedad: Propiedades inmobiliarias con geolocalización
Categoría: Clasificación de propiedades (casa, departamento, etc.)
Precio: Rangos de precios para filtrado

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Controladores:
UsuarioController: Maneja autenticación, registro y recuperación de contraseña
PropiedadController: CRUD de propiedades, gestión de imágenes y publicación

Helpers:
EmailHelper: Envío de correos de registro y recuperación
TokenHelper: Generación de tokens JWT y IDs únicos

Middleware:
ProtegerRutaMiddleware: Verifica autenticación JWT
SubirImagenMiddleware: Gestión de carga de imágenes con Multer

Características Especiales:
Seguridad:

- Contraseñas hasheadas con bcrypt
- Protección CSRF (Cross-Site Request Forgery, o Falsificación de Petición entre Sitios) por token entre el server y foormularios o pages
- Autenticación JWT
- Validación: Express-validator en controladores

Almacenamiento: Imágenes con Multer, carpeta public/

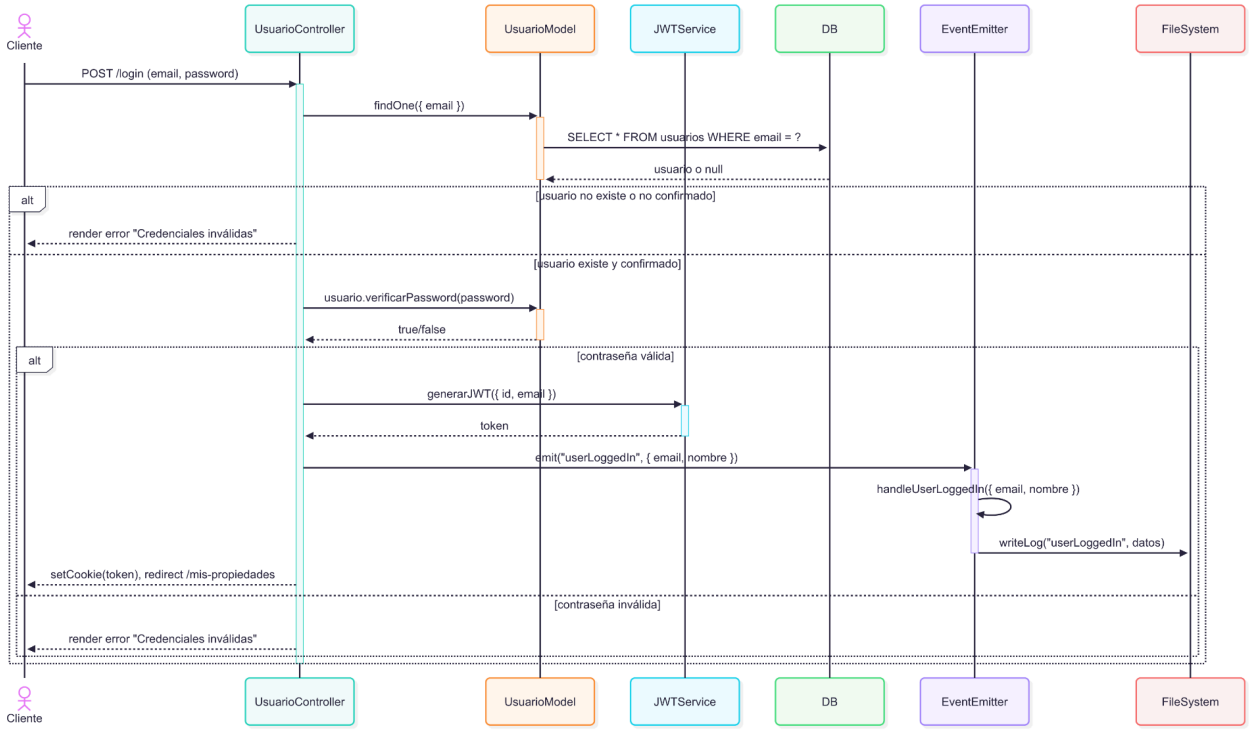
Base de datos: MySQL con Sequelize ORM

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN **Año**: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

1.7.4 Diagramas de secuencia:

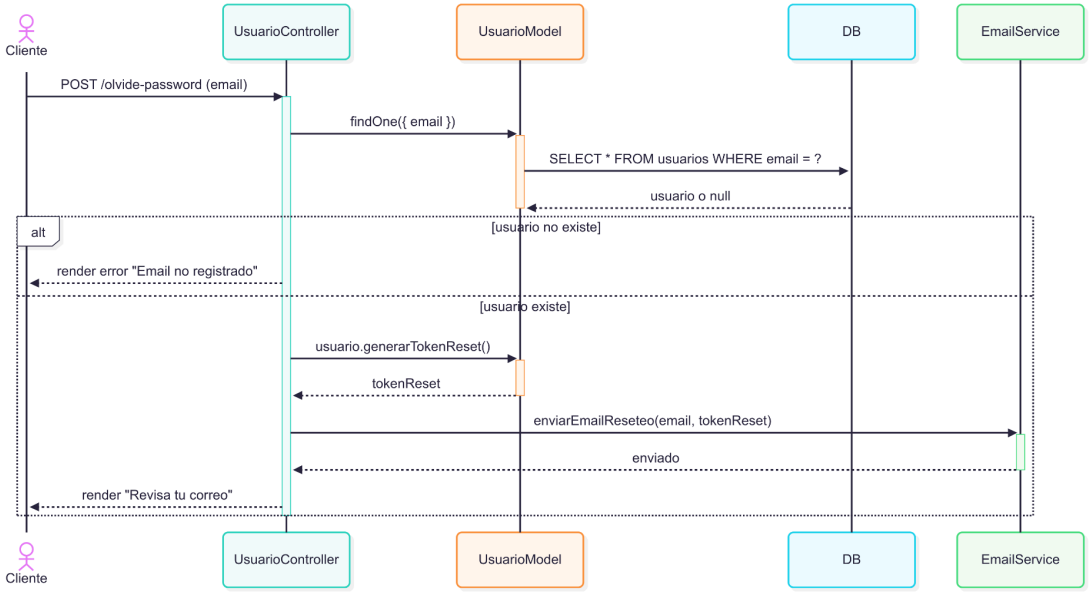
1.7.4.1 Diagrama de secuencia core 1:

Iniciar sesión:

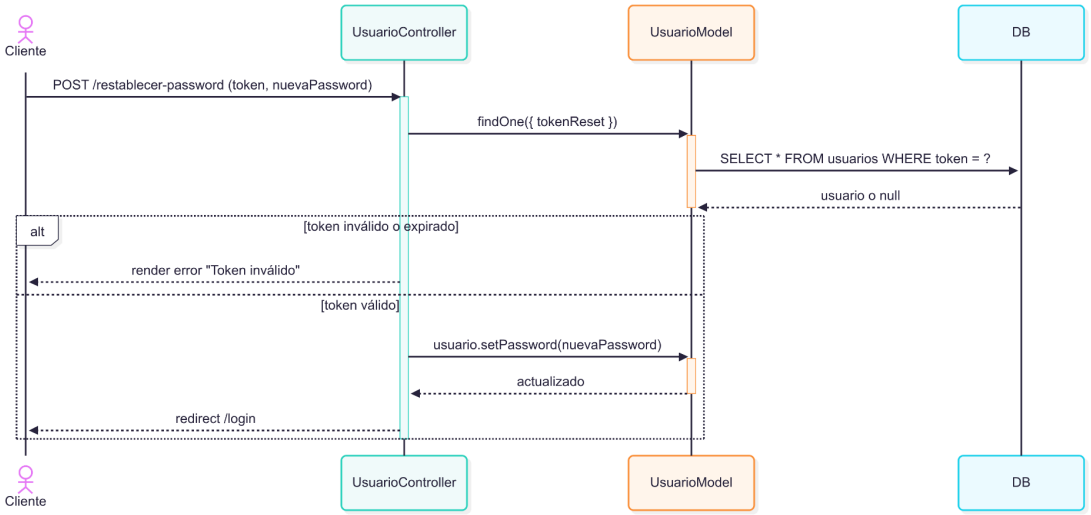


	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN **Año**: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Olvide Contraseña:



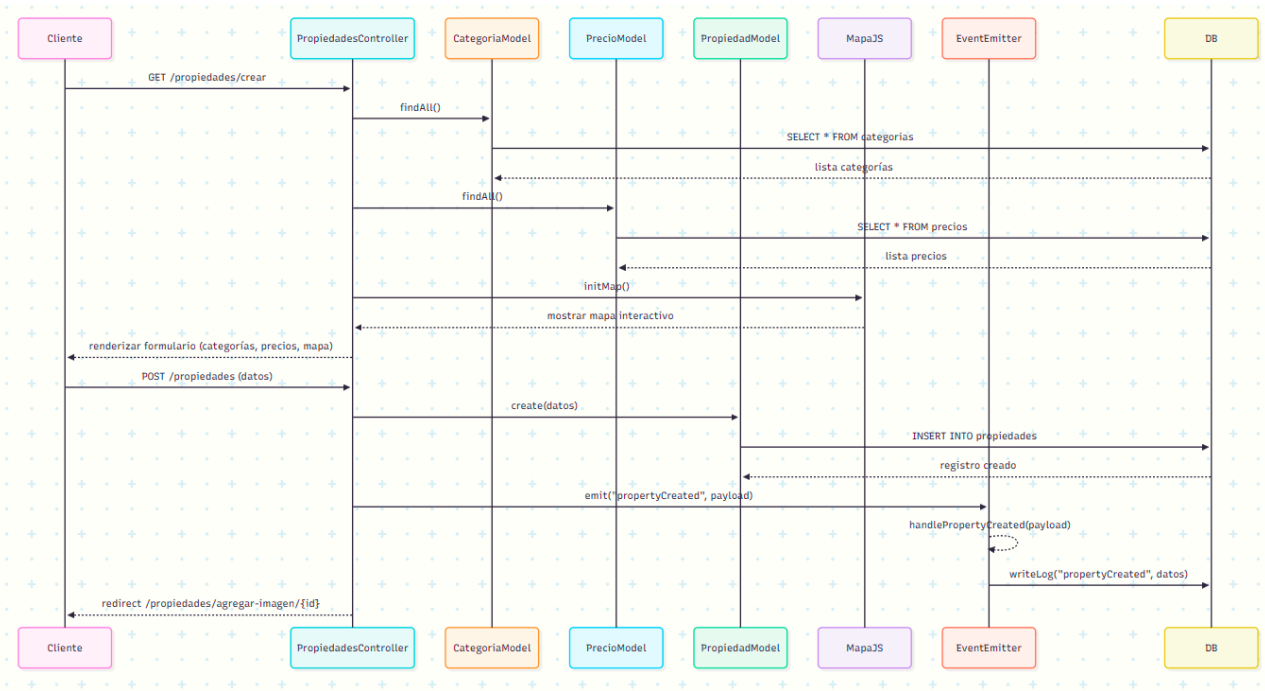
Restablecer contraseña:



	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

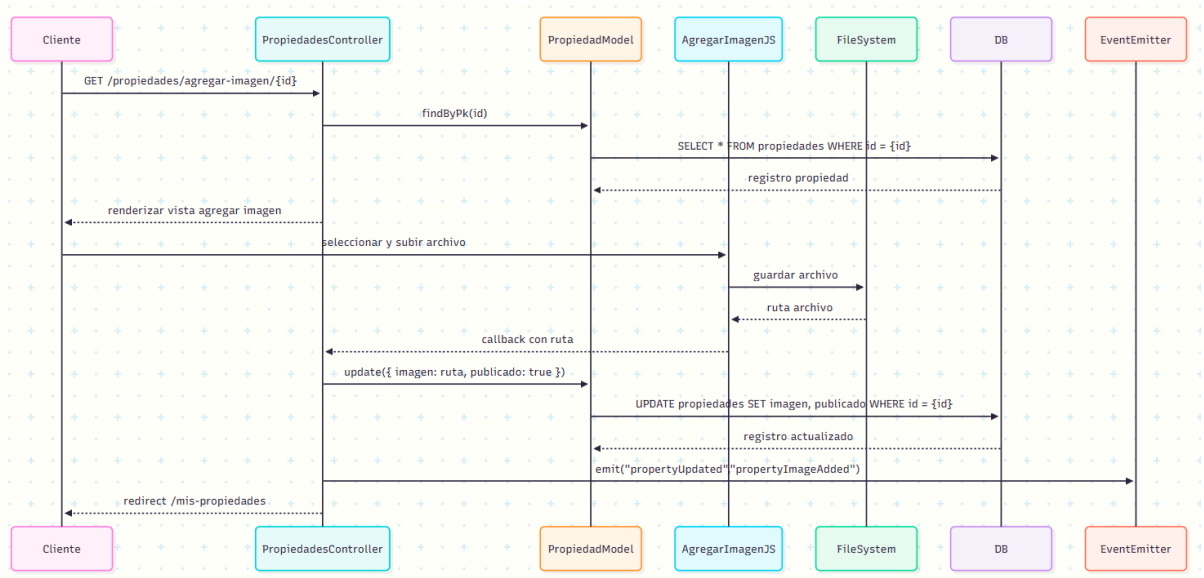
1.7.4.2 Diagrama de de secuencia core 2 crear propiedad y agregar imagen:

Crear propiedad



	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN **Año**: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Agregar imagen



	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

1.7.5 Patrones de diseño utilizados en el proyecto y su explicación

Se documentan cuatro patrones principales: Singleton (conexión a base de datos), MVC (arquitectura general), Observer (sistema de eventos y logging), y Chain of Responsibility (middleware de Express). Esta documentación sigue el estándar propuesto por Maximiliano Cristiá para la materia Ingeniería de Software de la FCEIA-UNR propuesto por la catedra.

Patrón MVC: Modelo-Vista-Controlador

Modelo:

```
┌──────────────────────────────────────────────────────────────────────────┐
| Pattern ModelosDatos                                                     |
├──────────────────────────────────────────────────────────────────────────┤
| based on Model (del patrón MVC)                                          |
├──────────────────────────────────────────────────────────────────────────┤
| because                                                                  |
|                                                                          |
|   Cambios previstos:                                                     |
|   - Modificación de esquema de base de datos (agregar/quitar campos)     |
|   - Cambio de validaciones de datos                                      |
|   - Implementación de nuevas relaciones entre entidades                  |
|   - Agregar nuevos modelos (Mensaje, Favorito, Comentario, etc.)         |
|   - Cambio de ORM o migración a otro motor de base de datos              |
|                                                                          |
|   Funcionalidad:                                                         |
|   - Encapsular la lógica de acceso a datos y persistencia                |
|   - Definir la estructura de las entidades del dominio                   |
|   - Gestionar relaciones entre entidades (1:N entre Usuario-Propiedad)   |
|   - Implementar validaciones de negocio a nivel de datos                 |
|   - Proveer métodos para CRUD de entidades                               |
|                                                                          |
|   Restricciones de diseño:                                               |
|   - Los modelos no deben contener lógica de presentación                 |
|   - Deben ser independientes del framework de UI                         |
|   - Encapsular reglas de negocio relacionadas con datos                  |
└──────────────────────────────────────────────────────────────────────────┘
```

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
where

  Model           is Usuario (models/Usuario.js)
  Model           is Propiedad (models/Propiedad.js)
  Model           is Categoria (models/Categoria.js)
  Model           is Precio (models/Precio.js)
  attributes      is nombre, email, password, token, confirmado
                     (Usuario.js:6-19)
  attributes      is titulo, descripcion, techado, alarma, expensa,
                     calle, lat, lng, imagen, publicado
                     (Propiedad.js:11-51)
  operations      is Usuario.findByPk()
  operations      is Usuario.findAll()
  operations      is Propiedad.create()
  operations      is Propiedad.destroy()
  customMethod()  is Usuario.verificarPassword() (Usuario.js:40-42)
  hooks           is beforeCreate (Usuario.js:22-25)
  associations    is Propiedad.belongsTo(Usuario)
                     (models/index.js:10)
  associations    is Propiedad.belongsTo(Precio)
                     (models/index.js:8)
  associations    is Propiedad.belongsTo(Categoria)
                     (models/index.js:9)
```

```
comments

  Los modelos están implementados usando Sequelize ORM, que proporciona
  una abstracción sobre SQL y facilita el cambio de motor de BD.

  El modelo Usuario incluye un hook beforeCreate (línea 22) que hashea
  automáticamente la contraseña usando bcrypt antes de guardarla, y un
  método personalizado verificarPassword() (línea 40) para autenticación.

  El modelo Propiedad usa UUID como clave primaria (línea 6-9) para
  mayor seguridad y evitar enumeración de propiedades.

  Las relaciones se definen centralizadamente en models/index.js usando
  belongsTo, estableciendo que cada Propiedad pertenece a un Usuario,
  una Categoría y un rango de Precio.
```

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Vista:

```
┌──────────────────────────────────────────────────────────────────────────────┐
| Pattern VistasPresentacion                                                   |
├──────────────────────────────────────────────────────────────────────────────┤
| based on View (del patrón MVC)                                               |
├──────────────────────────────────────────────────────────────────────────────┤
| because                                                                      |
|                                                                              |
|   Cambios previstos:                                                         |
|   - Rediseño de interfaz de usuario (cambio de estilos, layouts)             |
|   - Adaptación responsive para diferentes dispositivos                       |
|   - Cambio de motor de templates (de Pug a otro template engine)             |
|   - Internacionalización (múltiples idiomas)                                 |
|   - Implementación de nuevas páginas o vistas                               |
|                                                                              |
|   Funcionalidad:                                                             |
|   - Renderizar la interfaz de usuario en formato HTML                        |
|   - Presentar datos recibidos de los controladores                           |
|   - Mostrar formularios para entrada de datos                                |
|   - Visualizar listas de propiedades y detalles                              |
|   - Presentar mensajes de error y validación al usuario                      |
|                                                                              |
|   Restricciones de diseño:                                                   |
|   - Las vistas no deben contener lógica de negocio                           |
|   - No deben acceder directamente a los modelos                              |
|   - Deben ser independientes de la fuente de datos                           |
|   - Renderizado server-side usando Pug template engine                       |
└──────────────────────────────────────────────────────────────────────────────┘
```

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
├──────────────────────────────────────────────────────────────────────────────┤
│ where                                                                        │
│                                                                              │
│   View              is views/auth/login.pug                                  │
│   View              is views/auth/registro.pug                               │
│   View              is views/auth/olvide-password.pug                        │
│   View              is views/propiedades/admin.pug                           │
│   View              is views/propiedades/crear.pug                           │
│   View              is views/propiedades/editar.pug                          │
│   View              is views/propiedades/agregar-imagen.pug                  │
│   Layout            is views/layout/index.pug                                │
│   TemplateEngine    is Pug (configurado en index.js:32-33)                   │
│   renderView()      is res.render('nombre-vista', datos)                     │
│   viewData          is { pagina, csrfToken, categorias, precios }            │
│   viewData          is { propiedades, errores, usuario }                     │
├──────────────────────────────────────────────────────────────────────────────┤
```

```
├──────────────────────────────────────────────────────────────────────────────┤
│ comments                                                                     │
│                                                                              │
│   Las vistas están organizadas por módulo funcional: 'auth/' para           │
│   autenticación y 'propiedades/' para gestión de propiedades.                │
│                                                                              │
│   Pug es configurado como motor de vistas en index.js (línea 32-33),         │
│   permitiendo sintaxis simplificada de HTML con indentación.                 │
│                                                                              │
│   Todas las vistas reciben csrfToken para protección CSRF, generado          │
│   por el middleware csurf y pasado desde los controladores.                  │
│                                                                              │
│   El layout principal (views/layout/index.pug) define la estructura          │
│   común HTML, incluyendo Tailwind CSS para estilos y secciones que           │
│   son reutilizadas por todas las vistas específicas.                         │
│                                                                              │
│   Los datos son pasados desde controladores mediante el segundo              │
│   parámetro de res.render(), ejemplo: propiedadesController.js:43-48         │
└──────────────────────────────────────────────────────────────────────────────┘
```

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Controlador

```
┌──────────────────────────────────────────────────────────────────────────┐
│ Pattern ControladorPropiedades                                           │
├──────────────────────────────────────────────────────────────────────────┤
│ based on Controller (del patrón MVC)                                     │
├──────────────────────────────────────────────────────────────────────────┤
│ because                                                                  │
│                                                                          │
│   Cambios previstos:                                                     │
│   - Modificación de flujo de navegación entre páginas                    │
│   - Cambio de reglas de validación de formularios                        │
│   - Implementación de nuevas acciones CRUD                               │
│   - Modificación de lógica de autorización                               │
│   - Integración con servicios externos (APIs, storage, email)            │
│                                                                          │
│   Funcionalidad:                                                         │
│   - Intermediar entre modelos (datos) y vistas (presentación)            │
│   - Procesar solicitudes HTTP (GET, POST) de propiedades                 │
│   - Validar datos de entrada de formularios                              │
│   - Coordinar operaciones de creación, edición y eliminación             │
│   - Gestionar subida de imágenes de propiedades                          │
│   - Manejar errores y redireccionamientos                                │
│                                                                          │
│   Restricciones de diseño:                                               │
│   - Los controladores no deben contener lógica de presentación           │
│   - No deben acceder directamente a la base de datos (usan modelos)      │
│   - Cada controlador maneja un conjunto coherente de funcionalidades     │
└──────────────────────────────────────────────────────────────────────────┘
```

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
where

   Controller          is propiedadesController (controllers/
                          propiedadesController.js)
   Controller          is usuarioController (controllers/
                          usuarioControllers.js)
   controllerAction()  is admin (propiedadesController.js:8-31)
   controllerAction()  is crear (propiedadesController.js:34-50)
   controllerAction()  is guardar (propiedadesController.js:52-106)
   controllerAction()  is agregarImagen (propiedadesController.js:108-135)
   controllerAction()  is almacenarImagen (propiedadesController.js:
                          137-172)
   controllerAction()  is editar (propiedadesController.js:174-205)
   controllerAction()  is guardarCambios (propiedadesController.js:
                          207-268)
   controllerAction()  is eliminar (propiedadesController.js:270-299)
   handleRequest()     is async (req, res) => { ... }
   updateModel()       is Propiedad.create(datos)
   updateModel()       is propiedad.save()
   updateModel()       is propiedad.destroy()
   renderView()        is res.render('propiedades/crear', datos)
   redirect()          is res.redirect('/mis-propiedades')
   validation          is validationResult(req) (línea 55)
```

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
comments

  PropiedadesController coordina las operaciones CRUD de propiedades.
  Cada método del controlador es una función async que recibe (req, res).

  El método 'admin' (línea 8) consulta propiedades del usuario usando
  Propiedad.findAll() con relaciones include (Categoria, Precio).

  El método 'guardar' (línea 52) valida datos con express-validator,
  crea la propiedad en BD, emite evento Observer (línea 96-99) y
  redirige a agregar imagen.

  La subida de imágenes se divide en dos métodos: 'agregarImagen'
  (muestra formulario) y 'almacenarImagen' (procesa archivo).

  UsuarioController (usuarioControllers.js) maneja autenticación,
  registro, recuperación de contraseña, usando helpers como EmailHelper
  y TokenHelper para funcionalidades auxiliares.

  Los controladores están conectados a rutas en routes/
  propiedadesRoutes.js y routes/usuarioRoutes.js mediante Router de
  Express.
```

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
Pattern ControladorUsuarios

based on Controller (del patrón MVC)

because

  Cambios previstos: modificación de flujo de autenticación, cambio de
  proveedor de email, implementación de OAuth, cambio de estrategia de
  tokens (JWT a sesiones), integración con servicios de autenticación
  externos (Google, Facebook).

  Funcionalidad: gestionar registro de usuarios, autenticación (login),
  cierre de sesión, recuperación de contraseña, confirmación de cuenta
  por email, generación y validación de tokens JWT.

  Restricciones de diseño: la contraseña debe hashearse antes de guardar,
  los tokens deben tener expiración, el email de confirmación debe
  enviarse automáticamente, la sesión se gestiona mediante JWT en
  cookies HTTP-only.
```

```
where

  Controller          is usuarioController (controllers/
                         usuarioControllers.js)
  controllerAction()  is formularioLogin
  controllerAction()  is autenticar
  controllerAction()  is cerrarSesion
  controllerAction()  is formularioRegistro
  controllerAction()  is registrar
  controllerAction()  is confirmar
  controllerAction()  is formularioOlvidePassword
  controllerAction()  is resetPassword
  controllerAction()  is comprobarToken
  controllerAction()  is nuevoPassword
  updateModel()       is Usuario.create()
  updateModel()       is usuario.save()
  authenticateUser()  is usuario.verificarPassword(password)
  generateToken()     is generarJWT({id: usuario.id})
  sendEmail()         is emailRegistro(datos)
  sendEmail()         is emailOlvidePassword(datos)
```

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

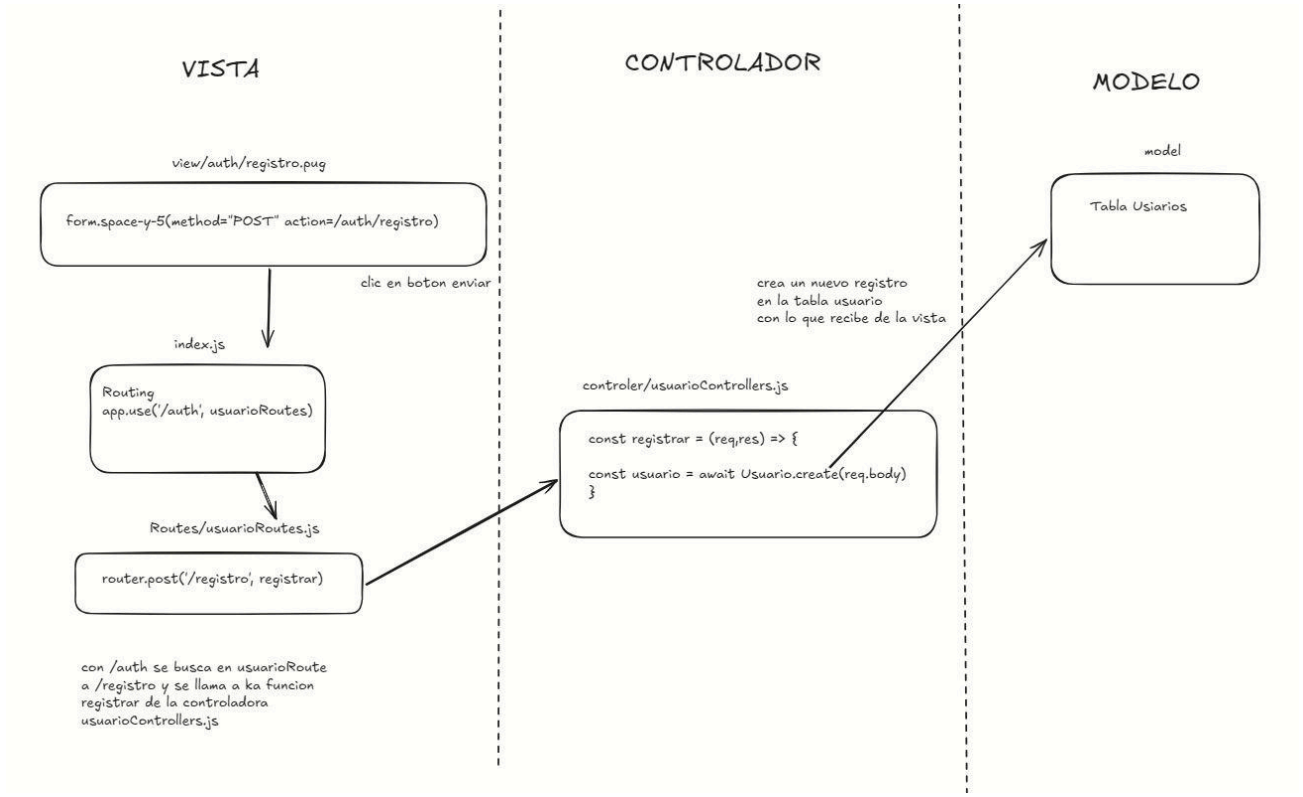
```
comments

  UsuarioController usa helpers externos (EmailHelper para envío de
  correos, TokenHelper para generación de tokens JWT y IDs únicos).

  El método verificarPassword() es un método personalizado del modelo
  Usuario que usa bcrypt.compareSync() para comparar contraseñas.

  Los tokens JWT se almacenan en cookies HTTP-only configuradas en el
  método 'autenticar', proporcionando seguridad contra XSS.
```

Referencias al código MVC:

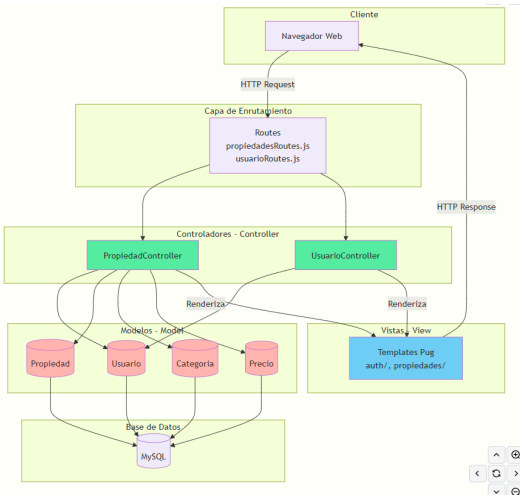
Modelos: models/Usuario.js, models/Propiedad.js, models/Categoria.js, models/Precio.js
Relaciones: models/index.js:8-10
Vistas: views/auth/, views/propiedades/
Controladores: controllers/propiedadesController.js, controllers/usuarioControllers.js
Configuración Pug: index.js:32-33
Rutas: routes/propiedadesRoutes.js, routes/usuarioRoutes.js

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

La siguiente imagen muestra cómo se relacionan la vista (pug.js) con la controladora (Node.js/Express.js) y la controladora con el modelo (Secualize(ORM) / base: Mysql)

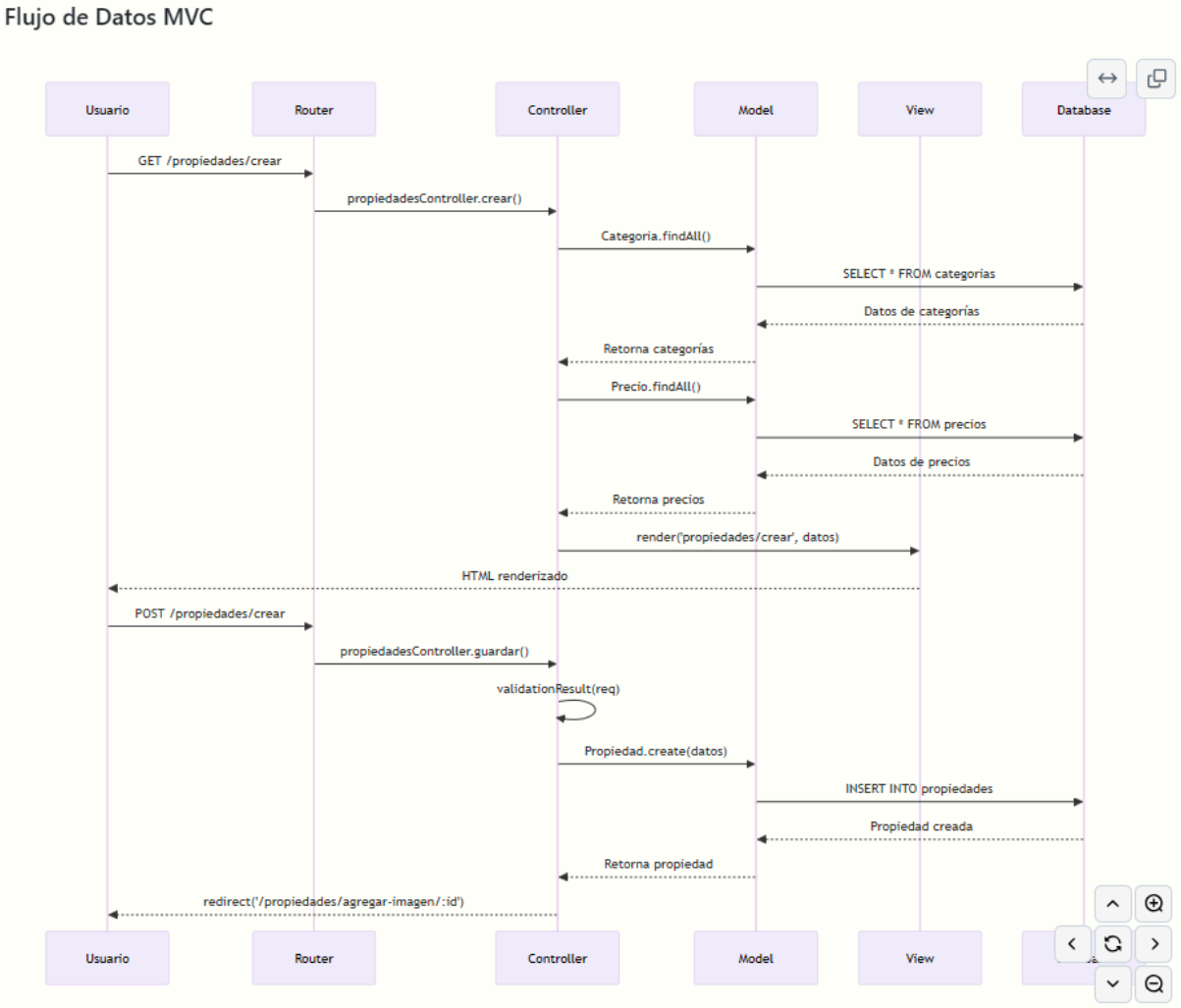


La siguiente imagen fue generada del Repositorio en GitHub del proyecto utilizando ClaudeCode, la cual grafica la estructura del patrón MVC en el proyecto



	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN **Año**: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

La siguiente imagen fue generada del Repositorio en GitHub del proyecto utilizando ClaudeCode, la cual grafica la secuencia del patrón MVC en el proyecto.



Patron singleton en el proyecto

El patrón Singleton es un patrón de diseño creacional que garantiza que una clase tenga una única instancia en toda la aplicación y proporciona un punto de acceso global a esa instancia.

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
┌──────────────────────────────────────────────────────────────────────────┐
| Pattern ConexionBaseDatos                                                |
├──────────────────────────────────────────────────────────────────────────┤
| based on Singleton                                                       |
├──────────────────────────────────────────────────────────────────────────┤
| because                                                                  |
|                                                                          |
|   Cambios previstos:                                                     |
|   - Cambio de proveedor de base de datos (MySQL, PostgreSQL, SQLite)     |
|   - Modificación de parámetros de conexión (pool size, timeouts)         |
|   - Migración entre ambientes (desarrollo, testing, producción)          |
|   - Implementación de réplicas o clustering de base de datos             |
|                                                                          |
|   Funcionalidad:                                                         |
|   - Proveer una única instancia de conexión a la base de datos MySQL     |
|   - Gestionar el pool de conexiones de forma centralizada                |
|   - Permitir que todos los modelos (Usuario, Propiedad, Categoria,       |
|     Precio) compartan la misma configuración de base de datos            |
|   - Sincronizar esquema de base de datos con los modelos Sequelize       |
|                                                                          |
|   Restricciones de diseño:                                               |
|   - Debe existir una única instancia de conexión en toda la aplicación   |
|   - La configuración debe ser externa (variables de entorno)             |
|   - Todos los modelos deben usar la misma instancia de Sequelize         |
|   - Optimización de recursos: evitar múltiples conexiones innecesarias   |
├──────────────────────────────────────────────────────────────────────────┤
| where                                                                    |
|                                                                          |
|   Singleton            is db (instancia de Sequelize en config/db.js)    |
|   getInstance()        is import db from './config/db.js'                |
|   uniqueInstance       is new Sequelize(DB_NOMBRE, DB_USER, DB_PASS)     |
|   singletonOperation() is db.authenticate()                              |
|   singletonOperation() is db.sync()                                      |
|   singletonOperation() is db.define()                                    |
|   Client               is Usuario (models/Usuario.js)                    |
|   Client               is Propiedad (models/Propiedad.js)                |
|   Client               is Categoria (models/Categoria.js)                |
|   Client               is Precio (models/Precio.js)                      |
└──────────────────────────────────────────────────────────────────────────┘
```

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

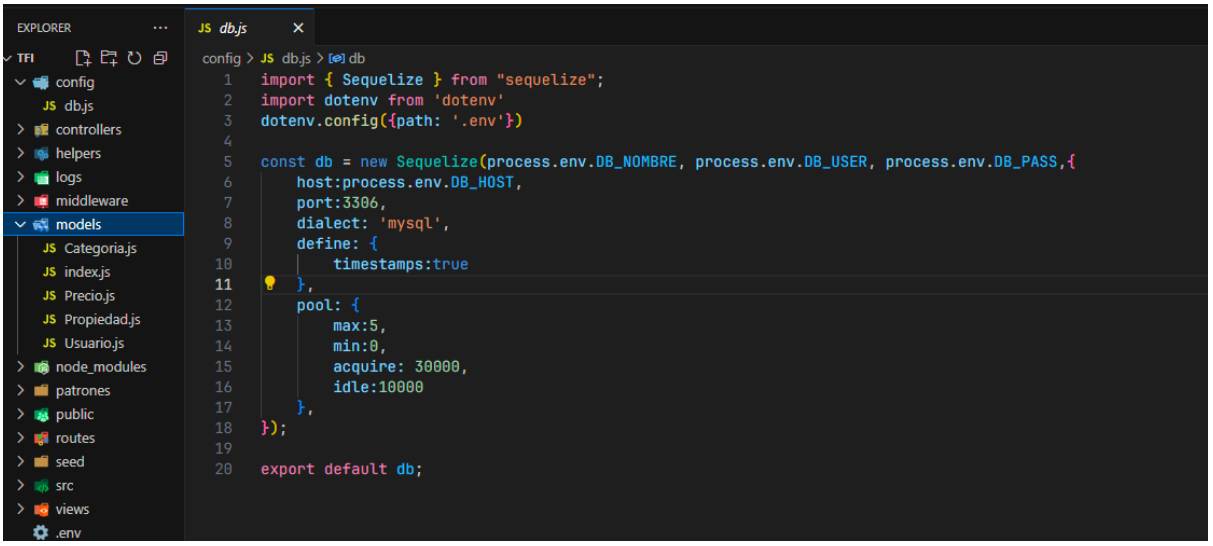
```
comments

  En JavaScript/ES6, el uso de 'export default db' combinado con el
  sistema de módulos de Node.js garantiza que la instancia de Sequelize
  sea única. Cuando múltiples módulos ejecutan 'import db from
  ./config/db.js', todos reciben la misma referencia al objeto.

  El archivo config/db.js (líneas 5-18) crea una única instancia de
  Sequelize con configuración de pool (max: 5, min: 0, acquire: 30000,
  idle: 10000) que es compartida por todos los modelos.

  Los cuatro modelos del sistema (Usuario, Propiedad, Categoria, Precio)
  importan 'db' y lo utilizan para definir sus esquemas mediante
  db.define(), garantizando que todos usen la misma conexión.

  En index.js (líneas 23-29) se realiza la autenticación y sincronización
  de la base de datos una única vez al iniciar la aplicación.
```

Referencias al código:

Singleton: config/db.js:5-18
Uso en Usuario: models/Usuario.js:5
Uso en Propiedad: models/Propiedad.js:2
Uso en Categoria: models/Categoria.js:2
Uso en Precio: models/Precio.js:2
Inicialización: index.js:23-29

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN **Año**: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Imagen de la aplicación del modelo singleton en el proyecto:



Descripción:

¿Por qué funciona como Singleton?

- Caché de módulos en Node.js
- Cuando importo (import db from './config/db.js') un archivo, Node lo ejecuta una sola vez y guarda en memoria el resultado de ese export. Cualquier otro import posterior obtiene exactamente la misma instancia.

Un solo pool de conexiones:

- Esa instancia de Sequelize mantiene internamente un pool de conexiones a MySQL. Si en cada controlador o modelo importara y creara una nueva conexión, saturaría la base de datos con múltiples pools. Con el Singleton, todos los modelos comparten el mismo pool.

Ventajas

- Eficiencia: menos sobrecarga de conexiones.
- Consistencia: configuración (timeout, logging, hooks) es única y centralizada.
- Facilidad de uso: sólo necesitas import db from './config/db.js' en cualquier parte de mi código para usar la conexión ya inicializada.

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Patrón Observer en el proyecto:

El patrón Observer es un patrón de diseño de comportamiento que define una relación de dependencia uno-a-muchos entre objetos, de manera que cuando un objeto (sujeto) cambia su estado, todos los objetos dependientes (observadores) son notificados automáticamente.

```
Pattern SistemaEventosLogging
based on Observer
because

  Cambios previstos:
  - Agregar nuevos observadores (envío de emails, notificaciones push,
    analytics, auditoría de seguridad)
  - Modificar destino de logs (archivo, base de datos, servicio externo)
  - Implementar nuevos eventos del sistema (userRegistered,
    propertyPublished, imageUploaded, etc.)
  - Cambiar formato de logs (JSON, CSV, syslog)
  - Remover observadores sin afectar la lógica de negocio

  Funcionalidad:
  - Registrar eventos importantes del sistema (creación de propiedad,
    eliminación, login de usuarios)
  - Escribir logs en archivo de texto (logs/log.txt)
  - Desacoplar la lógica de logging de los controladores
  - Permitir múltiples observadores para un mismo evento
  - Facilitar auditoría y debugging del sistema

  Restricciones de diseño:
  - Los observadores no deben bloquear la ejecución de los controladores
  - El sistema debe funcionar correctamente aunque fallen los observers
  - Los eventos deben ser síncronos pero no afectar el flujo principal
  - El emisor de eventos no debe conocer a los observadores concretos
```

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
├──────────────────────────────────────────────────────────────────────────────┤
│ where                                                                        │
│                                                                              │
│   Subject             is emitter (EventEmitter en helpers/eventEmitter.js)│
│   Observer            is writeLog function (eventEmitter.js:22-26)          │
│   ConcreteObserver    is propertyCreated listener (eventEmitter.js:31-37)  │
│   ConcreteObserver    is propertyDeleted listener (eventEmitter.js:40-45)  │
│   ConcreteObserver    is userLoggedIn listener (eventEmitter.js:48-53)     │
│   attach()            is emitter.on('eventName', callback)                  │
│   notify()            is emitter.emit('eventName', data)                    │
│   update()            is function(data) { writeLog(...) }                   │
│   observerState       is logFilePath (eventEmitter.js:13)                   │
│   subjectState        is payload de evento {id, titulo} o {email, nombre}  │
│   Publisher           is PropiedadController (propiedadesController.js:1)  │
│   Publisher           is UsuarioController (usuarioControllers.js)          │
│   notifyObservers()   is emitter.emit('propertyCreated', propiedad)         │
│                          (propiedadesController.js:96-99)                    │
│   notifyObservers()   is emitter.emit('propertyDeleted', propiedad)         │
│                          (propiedadesController.js:290-293)                  │
│   notifyObservers()   is emitter.emit('userLoggedIn', usuario)              │
├──────────────────────────────────────────────────────────────────────────────┤
```

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

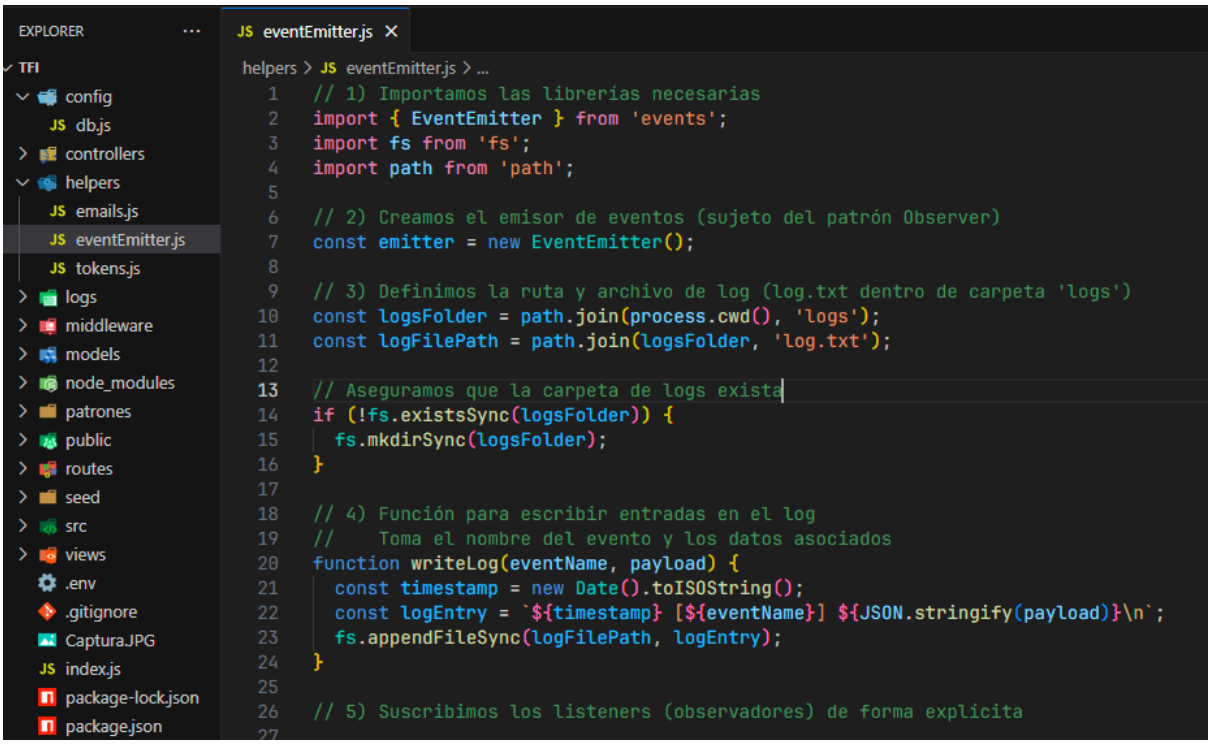
```
comments

  El patrón Observer está implementado usando la clase EventEmitter
  nativa de Node.js (helpers/eventEmitter.js línea 4), que proporciona
  los métodos on() para suscribir observadores y emit() para notificarlos.

  Los observadores se suscriben explícitamente en el archivo
  eventEmitter.js (líneas 31, 40, 48) usando emitter.on('evento',
  callback), estableciendo qué función se ejecutará cuando ocurra el
  evento específico.

  La función writeLog() (línea 22-26) es el comportamiento común de todos
  los observadores: recibe el nombre del evento y los datos, genera un
  timestamp ISO, formatea la entrada como JSON y la escribe en el archivo
  logs/log.txt usando fs.appendFileSync().

  Los controladores actúan como Publishers: PropiedadController emite
  'propertyCreated' después de crear una propiedad (línea 96-99) y
  'propertyDeleted' antes de destruirla (línea 290-293). Estos
  controladores importan el emitter y llaman emit() en momentos clave.

  El desacoplamiento es completo: los controladores solo conocen el
  emitter y los nombres de eventos, no saben qué observadores están
  suscritos. Se pueden agregar nuevos observers (envío de emails,
  notificaciones) sin modificar los controladores.

  Ejemplo de log generado: "2025-01-15T10:30:00.000Z [propertyCreated]
  {"id":"a1b2c3","titulo":"Garage en Centro"}"
```

Referencias al código:

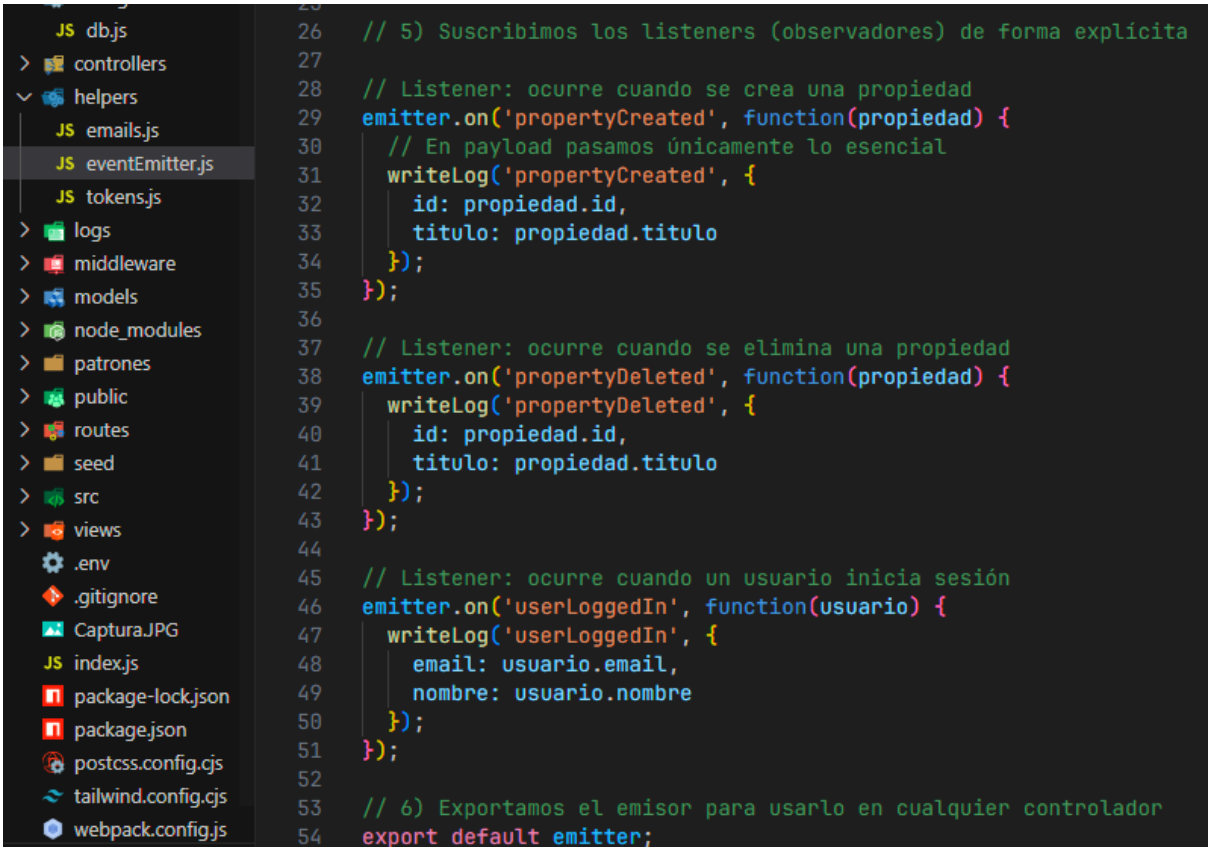
Subject (EventEmitter): helpers/eventEmitter.js:9
Observers (listeners): helpers/eventEmitter.js:31-53
Función writeLog: helpers/eventEmitter.js:22-26
Emisión propertyCreated: controllers/propiedadesController.js:96-99
Emisión propertyDeleted: controllers/propiedadesController.js:290-293
Import en controller: controllers/propiedadesController.js:2

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Imagen de la aplicación del patrón Observer en el proyecto

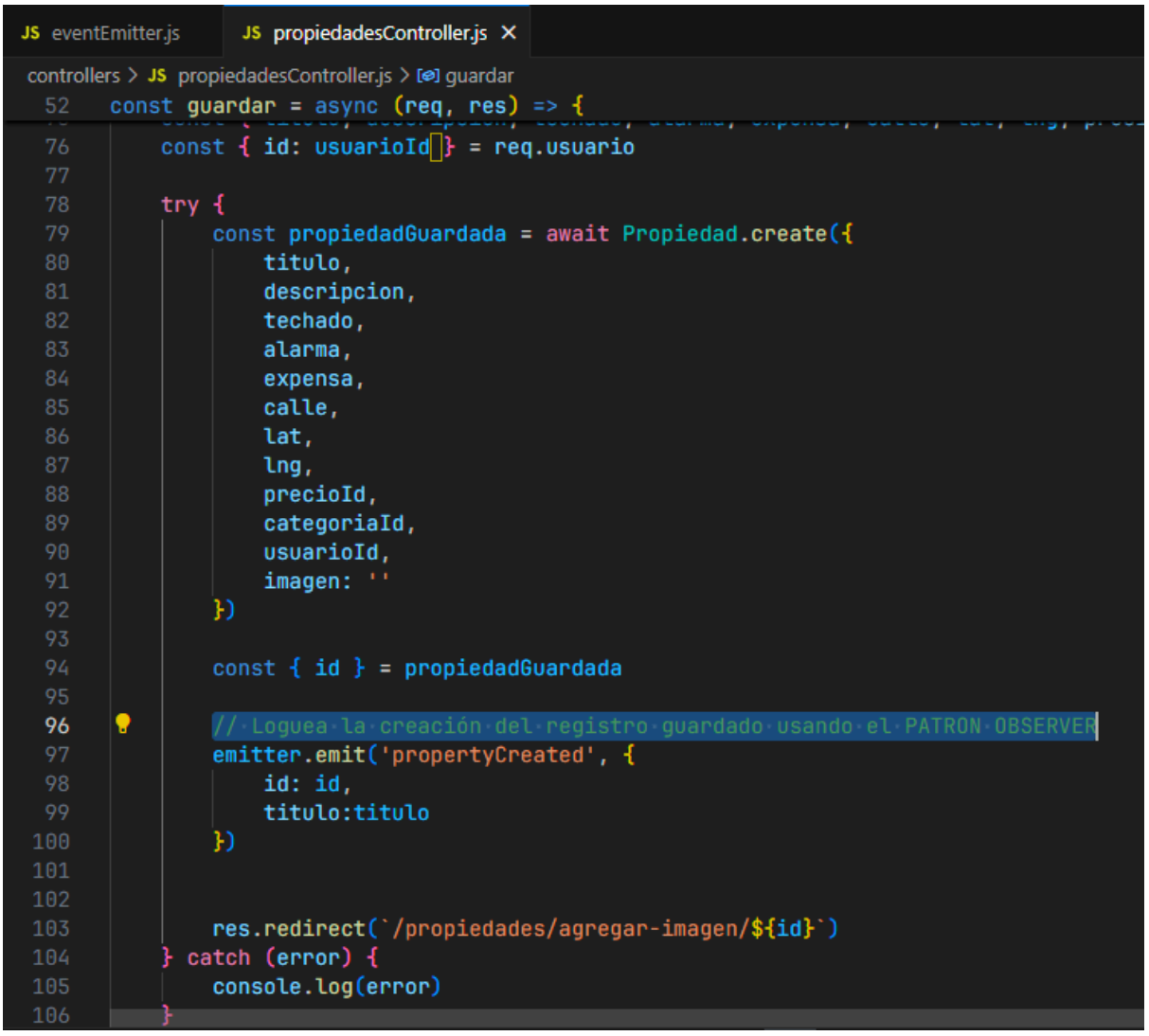


	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	



	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Utilización del patrón observer por parte de las controladoras



Descripción del patrón observer en el proyecto:

a) Existe un emisor único en helpers/eventEmitter.js (mi canal de avisos)

```
import { EventEmitter } from 'events';
const emitter = new EventEmitter();
export default emitter;
```

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

b) Suscribo el aviso dentro de helpers/eventEmitter.js

```
emitter.on('propertyCreated', info => {
  // por ejemplo, anotar un log
  writeLog('Se creó propiedad', info);
});
```

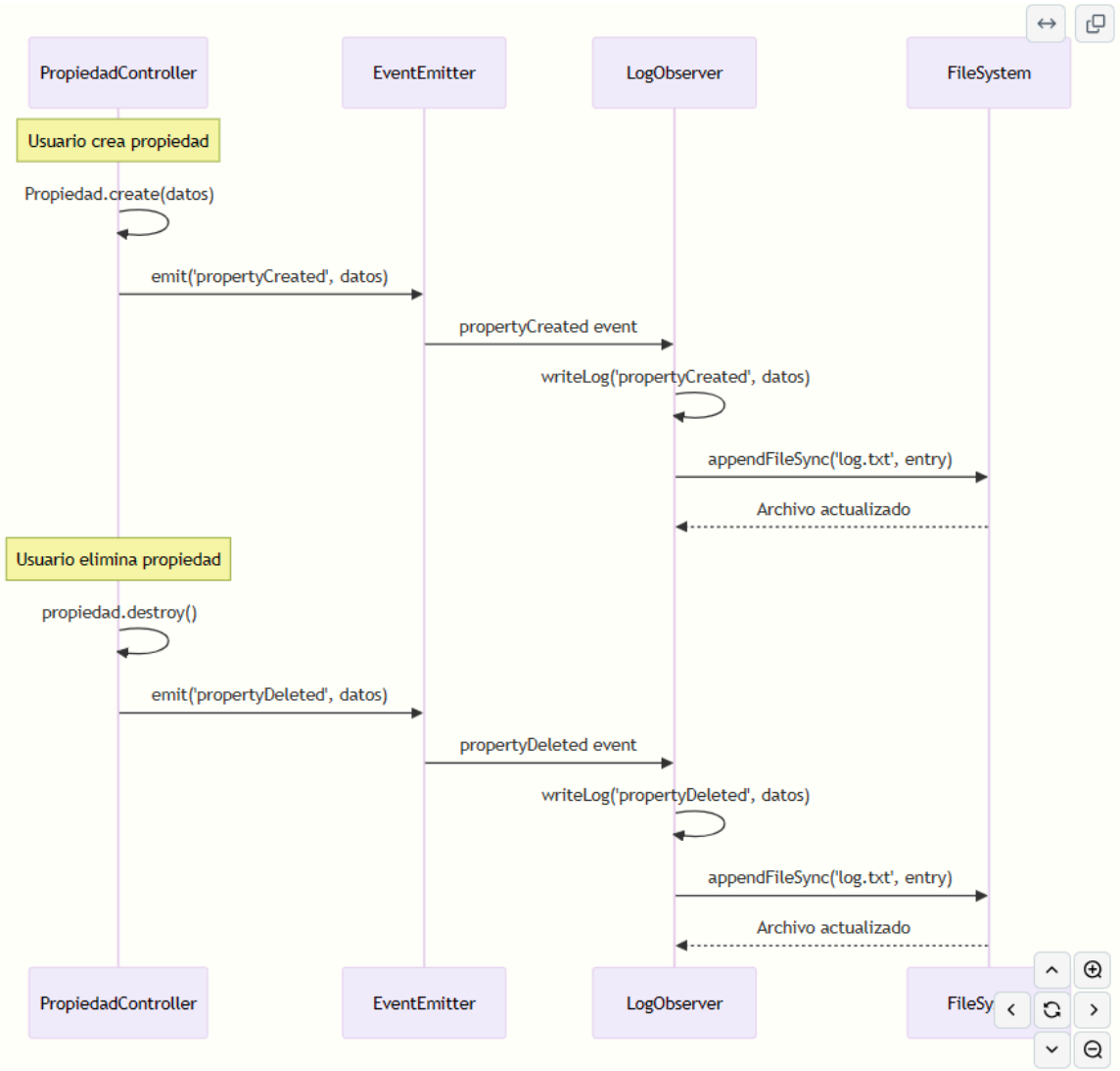
c) En las controladoras publicó el aviso, por ejemplo luego de crear un usuario en la controladora de usuarios usuarioControllers

```
emitter.emit('propertyCreated', { id: nuevoId, user: req.user.id });
```

Esto en mi caso va a registrar en un archivo txt el id del nuevo usuario registrado y la hora de creación, lo bueno es que en el emmiter.on uno puede agregar otros servicios fácilmente como alertar por un canal de slack o por mail desacoplando controladoras, En resumen, las controladores publican eventos con emit, y los listeners se suscriben con on, de forma totalmente independiente.

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN **Año**: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

La siguiente imagen fue generada del Repositorio en GitHub del proyecto utilizando ClaudeCode, la cual grafica la secuencia del patrón observer en el proyecto.



	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

Patrón Chain of Responsibility - Middleware de Express

```
┌──────────────────────────────────────────────────────────────────────────┐
│ Pattern CadenaMiddlewarePropiedades                                      │
├──────────────────────────────────────────────────────────────────────────┤
│ based on Chain of Responsibility                                         │
├──────────────────────────────────────────────────────────────────────────┤
│ because                                                                  │
│                                                                          │
│   Cambios previstos:                                                     │
│   - Agregar nuevos middlewares (rate limiting, compression, logging)     │
│   - Modificar orden de ejecución de middlewares                          │
│   - Cambiar estrategia de autenticación (JWT a OAuth)                    │
│   - Implementar middlewares de caché o transformación de respuestas      │
│   - Remover middlewares sin afectar otros en la cadena                   │
│   - Agregar validaciones personalizadas por ruta                         │
│                                                                          │
│   Funcionalidad:                                                         │
│   - Procesar solicitudes HTTP en múltiples etapas secuenciales           │
│   - Parsear cookies para autenticación (CookieParser)                    │
│   - Validar tokens CSRF para prevenir ataques (CSRF)                     │
│   - Verificar autenticación JWT del usuario (ProtegerRuta)               │
│   - Validar campos de formularios (Express-Validator)                    │
│   - Procesar subida de archivos/imágenes (Multer)                        │
│   - Ejecutar controlador final si todas las validaciones pasan           │
│                                                                          │
│   Restricciones de diseño:                                               │
│   - Cada middleware debe ser independiente y reutilizable                │
│   - Un middleware puede terminar la cadena (enviar respuesta) o          │
│     continuar al siguiente (llamar next())                               │
│   - El orden de middlewares es crítico (ej: cookieParser antes de CSRF)  │
│   - Los middlewares deben poder modificar req/res para siguientes        │
│   - Manejo de errores sin romper la cadena                               │
└──────────────────────────────────────────────────────────────────────────┘
```

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
where

  Handler              is Middleware (concepto general de Express)
  ConcreteHandler      is cookieParser (middleware, index.js:17)
  ConcreteHandler      is csurf (middleware, index.js:20)
  ConcreteHandler      is protegerRuta (middleware/protegerRuta.js)
  ConcreteHandler      is express-validator (body validations,
                          propiedadesRoutes.js:15-22)
  ConcreteHandler      is upload.single('imagen') (middleware/
                          subirimagen.js, propiedadesRoutes.js:29)
  ConcreteHandler      is PropiedadController.guardar (controlador
                          final, propiedadesController.js:52)
  handleRequest()      is (req, res, next) => { ... }
  setSuccessor()       is app.use(middleware) o router.METHOD(path, mw)
  successor            is next (función callback de Express)
  canHandle()          is condiciones if dentro de cada middleware
  passToSuccessor()    is next() o next(error)
  terminateChain()     is res.send(), res.redirect(), res.render()
```

```
where (ejemplo cadena específica: POST /propiedades/crear)

  Handler1             is cookieParser (parsea cookies del request)
  Handler2             is csurf (valida token CSRF de cookie)
  Handler3             is protegerRuta (verifica JWT, carga usuario)
  Handler4             is body('titulo').notEmpty() (valida título)
  Handler5             is body('descripcion').notEmpty() (valida desc)
  Handler6             is body('categoria').isNumeric() (valida cat)
  Handler7             is body('precio').isNumeric() (valida precio)
  Handler8             is body('techado').isIn() (valida techado)
  Handler9             is body('alarma').isIn() (valida alarma)
  Handler10            is body('expensa').isIn() (valida expensa)
  Handler11            is body('lat').isNumeric() (valida ubicación)
  Handler12            is guardar (controlador final que procesa todo)
```

	UNIVERSIDAD ABIERTA INTERAMERICANA **Facultad de Tecnología Informática**	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
comments

  Express implementa Chain of Responsibility mediante su sistema de
  middlewares. Cada middleware recibe (req, res, next) donde 'next' es
  la función que invoca al siguiente manejador en la cadena.

  MIDDLEWARE GLOBAL (index.js):
  Los middlewares globales (líneas 13, 17, 20) se aplican a todas las
  rutas: express.urlencoded() parsea body, cookieParser() parsea cookies,
  csurf() valida tokens CSRF. Estos forman la base de la cadena.

  MIDDLEWARE PROTEGERRUTA (middleware/protegerRuta.js):
  Implementa tres verificaciones secuenciales: 1) Verifica existencia del
  token _token en cookies (línea 7-10), si no existe termina la cadena
  con redirect. 2) Verifica validez del JWT usando jwt.verify (línea 14).
  3) Carga usuario desde BD (línea 15). Si todo pasa, adjunta req.usuario
  (línea 19) y llama next() (línea 23) continuando la cadena.

  MIDDLEWARE VALIDACIONES (propiedadesRoutes.js:15-22):
  Express-validator agrega múltiples middlewares de validación en cadena.
  Cada body('campo') agrega un middleware que valida ese campo. Todos se
  ejecutan y acumulan errores en req. El controlador los verifica con
  validationResult(req) (propiedadesController.js:55).

  MIDDLEWARE SUBIRIMAGEN (middleware/subirimagen.js):
  Usa Multer para procesar multipart/form-data. Configura storage con
  diskStorage definiendo destination (./public/uploads/) y filename
  (timestamp + nombre original). upload.single('imagen') procesa un
  archivo y lo adjunta a req.file, luego llama next().
```

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

```
MANEJO DE ERRORES:
Si un middleware lanza error o llama next(error), Express salta todos
los handlers normales y busca middleware de manejo de errores
(signature: (err, req, res, next) => {}). En protegerRuta, el catch
(línea 25-27) limpia la cookie y redirige en caso de JWT inválido.

EJEMPLO DE FLUJO COMPLETO:
POST /propiedades/crear → cookieParser → csurf → protegerRuta →
[si no autenticado: redirect /auth/login, termina cadena]
[si autenticado: req.usuario = {...}, continúa] →
11 validadores de express-validator → guardar (controlador) →
[si errores validación: renderiza crear.pug con errores, termina]
[si todo OK: Propiedad.create(), emit event, redirect, termina]
```

Referencias al código:

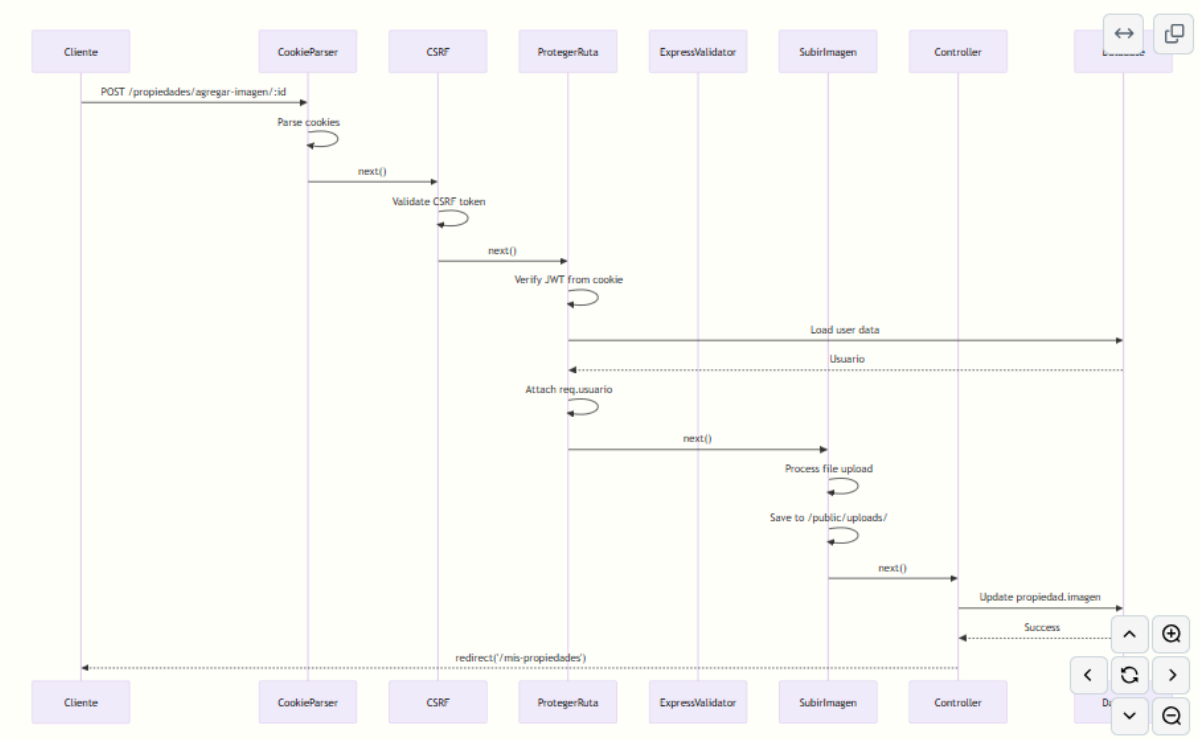
Middleware global cookieParser: index.js:17
Middleware global CSRF: index.js:20
Middleware protegerRuta: middleware/protegerRuta.js:4-32
Middleware subirImagen (Multer): middleware/subirimagen.js:3-13
Cadena de validadores: routes/propiedadesRoutes.js:15-23
Uso de next(): middleware/protegerRuta.js:23
Controlador final: controllers/propiedadesController.js:52-106

Beneficios del Patrón

- Separación de responsabilidades: Cada middleware tiene una única función
- Reutilización: Middlewares como protegerRuta se usan en múltiples rutas
- Orden flexible: Se pueden reorganizar según necesidades
- Fácil testing: Cada middleware se puede probar de forma aislada
- Extensibilidad: Agregar nuevo middleware sin modificar existentes

	UNIVERSIDAD ABIERTA INTERAMERICANA Facultad de Tecnología Informática	**Curso**: 5° TN
		Año: 2024
	Docentes: Alejandro Roberto Santorio, Matias Pablo Banaga	
	Alumno: Galarza Cristian	
	Proyecto: Hybrid Parking	

La siguiente imagen fue generada del Repositorio en GitHub del proyecto utilizando ClaudeCode, la cual grafica la secuencia del patrón chain of responsability en el proyecto.



Anexos

Bibliografía consultada

Shvets, A. (2019). Sumérgete en los patrones de diseño. Refactoring.Guru.

References

Cristiá, M. (2015). *Estándar para documentar el uso de patrones de diseño en un diseño de software*. FCEIA. Retrieved November 10, 2025, from https://www.fceia.unr.edu.ar/ingsoft/patrones-doc.pdf