Taller 2 - Arquitectura

Que se va a realizar

El proyecto CarroCo consiste en el desarrollo de una plataforma cliente–servidor que permite a los usuarios visualizar un catálogo de automóviles y seleccionar aquellos de su preferencia.

El sistema busca demostrar la aplicación de principios de arquitectura de software moderna, empleando un stack tecnológico compuesto por Angular, Go, GraphQL y MongoDB, que garantiza una comunicación eficiente entre las capas de presentación y lógica de negocio.

Información de las tecnologías

Definición

El estilo Cliente–Servidor es un modelo de comunicación en el cual un cliente (navegador web o aplicación móvil) solicita servicios a un servidor, que responde procesando la petición y devolviendo la información.
Un Web Service es un conjunto de protocolos y estándares que permiten la interoperabilidad entre diferentes aplicaciones a través de la red.
La división Frontend/Backend complementa este estilo, separando la aplicación en dos capas:

- **Frontend:** se desarrolla en Angular (TypeScript), framework de Google para construir interfaces dinámicas.
- **Backend:** se implementa en Golang, un lenguaje compilado y altamente eficiente.

La comunicación entre cliente y servidor se establece mediante Web Services, usando GraphQL como protocolo de consultas flexibles y optimizadas.

Características

Cliente–Servidor / Backend–Frontend
- División clara de responsabilidades.
- Escalabilidad horizontal y vertical.

Angular (TypeScript)
- Framework basado en componentes.
- Data binding bidireccional y tipado fuerte.
- Soporte de herramientas CLI y ecosistema maduro.

Go (Golang)

- Concurrencia nativa con goroutines.
- Lenguaje compilado, portable y de bajo consumo.
- Seguridad de tipos y sintaxis simple.

GraphQL

- Consultas precisas, evitando sobrecarga de datos.
- Esquema definido y fuertemente tipado.
- Soporta suscripciones en tiempo real.

Historia y evolución

Cliente–Servidor

Años 80: primeras redes locales donde clientes ligeros accedían a servidores centrales.
Años 90: explosión de la web, modelo HTTP con navegadores como clientes.
2000 en adelante: transición hacia arquitecturas distribuidas y microservicios.

Angular

Surge como AngularJS (2010), enfocado en MVC en frontend.
Reescrito en TypeScript en 2016, se convierte en Angular moderno.

Go (Golang)

Creado en Google en 2007 y lanzado en 2009 para resolver problemas de escalabilidad en sistemas distribuidos.

GraphQL

Desarrollado por Facebook en 2012 para mejorar la eficiencia en la comunicación cliente–servidor.
Publicado en 2015 como estándar abierto.

Ventajas y desventajas

Tecnología / Estilo	Ventajas	Desventajas
Cliente–Servidor / Backend–Frontend	- Separación clara de responsabilidades (UI vs. lógica de negocio). - Escalabilidad horizontal (más clientes) y vertical (mejor servidor). - Centralización de datos y seguridad.	- Dependencia total del servidor (si falla, afecta a todos los clientes). - Puede generar sobrecarga en el servidor. - Requiere infraestructura de red estable.

	- Posibilidad de mantener y actualizar el servidor sin afectar al cliente.	- Mayor complejidad al distribuir múltiples servicios.
Angular (Frontend)	- Framework mantenido por Google.- Basado en componentes reutilizables. - Data binding bidireccional. - Soporte con TypeScript (tipado fuerte, más robustez). - Ideal para Single Page Applications (SPA).	- Curva de aprendizaje pronunciada. - Código más verboso comparado con otros frameworks (ej. React, Vue). - Actualizaciones frecuentes que pueden romper compatibilidad.
Go / Golang (Backend)	- Alto rendimiento y eficiencia en concurrencia. - Sintaxis simple y limpia. - Ejecución rápida al ser compilado. - Excelente para microservicios y sistemas distribuidos. - Gran portabilidad (binarios ligeros).	- Ecosistema de librerías más pequeño que Java o Python. - No tan orientado a aplicaciones de escritorio o frontend. - Manejo de errores menos flexible (no hay excepciones tradicionales).
GraphQL (Protocolo de datos)	- Consultas precisas: el cliente pide solo lo que necesita. - Disminuye llamadas redundantes. - Esquema fuertemente tipado. - Soporta suscripciones para tiempo real. - Ideal para aplicaciones con datos muy relacionados.	- Riesgo de consultas costosas que sobrecarguen el servidor. - Configuración más compleja que REST. - Necesita control de seguridad adicional (exposición de datos sensibles). - Curva de aprendizaje para diseñar esquemas eficientes.

Casos de uso

Situación / Problema	Rol del Cliente (Angular)	Rol del Servidor (Go + GraphQL)	Beneficio del Modelo Cliente–Servidor
Necesidad de mostrar grandes volúmenes de datos (ej. catálogo de información)	Interfaz interactiva que filtra y presenta los datos dinámicamente	Procesa las consultas de GraphQL y retorna solo lo necesario	Reduce carga en la red, mejor experiencia para el usuario final
Múltiples usuarios accediendo simultáneamente (alta concurrencia)	Navegador solicita operaciones concurrentes	Go gestiona múltiples peticiones con goroutines de forma eficiente	Escalabilidad y soporte de alta carga sin pérdida de rendimiento
Aplicaciones que requieren	Angular permite actualizar la	Backend en Go y APIs GraphQL se	Mantenimiento sencillo,

modularidad y evolución constante	interfaz sin alterar el backend	mantienen estables, aunque el frontend evolucione	desacoplamiento de capas
Evitar sobrecarga de datos en la comunicación cliente–servidor	Cliente pide datos específicos usando GraphQL	Servidor responde con datos exactos solicitados	Optimización en consumo de ancho de banda
Sistemas distribuidos en la nube	Acceso desde múltiples dispositivos	Backend desplegado en contenedores (Docker/Kubernetes en Go)	Escalabilidad en infraestructura cloud
Aplicaciones con necesidad de comunicación en tiempo real (ej. notificaciones)	Angular actualiza la vista de manera reactiva	GraphQL implementa suscripciones y Go maneja eventos concurrentes	Actualizaciones inmediatas sin recargar la página
Seguridad y centralización del manejo de datos	Cliente solo consume vistas, sin exponer lógica sensible	Servidor centraliza autenticación, lógica de negocio y validación	Mayor control de seguridad, integridad de datos

Casos de aplicación

Industria / Empresa	Cliente (Frontend) – Angular	Servidor (Backend) – Go/Golang	Protocolo / Servicio – GraphQL	Relación Cliente–Servidor
Google Ads / Gmail	Angular usado para construir SPA (Single Page Applications)	Backend robusto con microservicios escalables	APIs que optimizan la entrega de datos	Cliente ligero consume funciones distribuidas en servidores
Docker / Kubernetes	Paneles de administración con frameworks frontend	Core de la plataforma desarrollado en Go	Comunicación API para orquestación	Cliente gestiona contenedores mientras servidor maneja recursos distribuidos
GitHub (API v4)	Interfaz web moderna que consume datos dinámicos	Servidores procesan en Go y otros lenguajes	GraphQL para consultas personalizadas	Cliente pide solo datos necesarios (repos, issues, commits)
Shopify	Interfaz de administració	Backend escalable	GraphQL para	Cliente personaliza vistas de productos

	n con Angular y React	soportado por Go	manejar catálogos y ventas	con datos filtrados del servidor
Netflix	Aplicaciones cliente (web y móvil) con frameworks modernos	Backend en Go y Java para procesar streaming y recomendaciones	APIs que combinan REST y GraphQL	Cliente reproduce contenido según consultas al servidor
Uber	App móvil como cliente con interfaz modular	Backend en Go maneja miles de peticiones concurrentes	APIs optimizadas para geolocalización y viajes	Cliente (app) envía solicitudes que el servidor responde en milisegundos
Trello	Interfaz de tablero dinámico con frameworks JS (Angular-like)	Backend en Go para gestión de tareas	GraphQL implementado para sincronización de datos	Cliente muestra tareas en tiempo real gracias a servidor que gestiona estado y concurrencia
Twitter (infraestructura interna)	Interfaces con frameworks modernos	Backend de microservicios en Go	GraphQL para mejorar eficiencia de las consultas	Cliente muestra timeline personalizado según respuestas del servidor

Relación entre los temas asignados

Que tan común es el stack

El stack elegido: Angular / Go / GraphQL / MongoDB no es ampliamente común en la industria como combinación estándar, aunque cada una de sus tecnologías es popular por separado.

Su integración conjunta aparece principalmente en proyectos educativos, prototipos o implementaciones personalizadas, más que en productos empresariales consolidados.

Ejemplos:

- **Angular+Go+MongoDB:**
  Aplicación de películas: https://github.com/lvthillo/angular-go-mongodb
- **Golang+GraphQL+MongoDB:**
  Aplicación de recetas: https://github.com/jonathanhecl/golang-graphql-mongodb?utm_source=chatgpt.com

De todas maneras, el stack representa una arquitectura moderna y escalable, principal porque la combinación GraphQL + MongoDB cuenta con soporte oficial de MongoDB y Apollo, y Go ofrece un backend eficiente para APIs.

Matriz de análisis de Principios SOLID vs Temas

Principio SOLID	Angular	Go (Golang)	GraphQL	MongoDB
S Responsabilidad Única	Cada componente, servicio o módulo tiene una única función (UI, lógica o datos).	Cada paquete o struct cumple una sola función	Cada resolver maneja una sola consulta o mutación.	Cada colección o modelo representa una sola entidad del dominio.
O Abierto/Cerrado	Componentes y servicios se extienden sin modificarse directamente.	Estructuras pueden ampliarse mediante interfaces o composición.	Nuevos resolvers o tipos se agregan sin alterar los existentes.	Se pueden agregar campos opcionales al esquema sin romper compatibilidad.
L Sustitución de Liskov	Componentes hijos pueden sustituir a los padres sin alterar el comportamiento.	Tipos que implementan una interfaz pueden reemplazarla sin romper funcionalidad.	Resolvers o tipos derivados mantienen el mismo comportamiento esperado.	Documentos derivados deben conservar compatibilidad estructural.
I Segregación de Interfaces	Interfaces pequeñas y específicas	Interfaces pequeñas y concretas; principio clave en Go	Interfaces de esquema dividen tipos grandes en partes manejables.	Operaciones separadas por contexto (lectura, escritura, agregación).
D Inversión de Dependencias	Inyección de dependencias mediante servicios	Dependencia de interfaces en lugar de implementaciones concretas.	Resolvers dependen de servicios o repositorios abstractos.	Acceso a datos a través de repositorios o capas intermedias, no directamente desde la lógica.

Matriz de análisis de Atributos de Calidad vs Temas

Atributo de calidad (ISO 25010)	Angular	Go (Golang)	GraphQL	MongoDB

1. Funcionalidad / Adecuación funcional	Permite construir interfaces ricas y accesibles que exponen correctamente las funciones del sistema.	Implementa lógica del negocio precisa y eficiente, asegurando el cumplimiento de los requisitos funcionales.	Proporciona un esquema fuertemente tipado que garantiza la exactitud de las consultas.	Modela estructuras flexibles que se adaptan fácilmente a los requisitos funcionales.
2. Fiabilidad	El manejo de errores en componentes y validaciones del frontend mejora la estabilidad percibida.	Lenguaje compilado y concurrente, con excelente manejo de memoria y goroutines seguras, lo que aumenta la confiabilidad del backend.	Mecanismo de validación de esquemas y tipado que evita respuestas inconsistentes.	Replica y recuperación automática (replica sets) para alta disponibilidad.
3. Usabilidad	Framework orientado a experiencia de usuario (UX) y diseño modular, con soporte para accesibilidad (a11y).		GraphQL permite que los clientes obtengan solo los datos que necesitan, simplificando la interfaz y reduciendo sobrecarga visual.	
4. Eficiencia de desempeño	Usa cambio por detección optimizado y lazy loading para mejorar el rendimiento del frontend.	Concurrencia nativa, bajo consumo de memoria y ejecución compilada: gran eficiencia en el servidor.	Minimiza la transferencia de datos al enviar solo lo solicitado.	Índices, sharding y escalabilidad horizontal optimizan rendimiento en grandes volúmenes de datos.
5. Mantenibilidad	Arquitectura modular (componentes, servicios, módulos) que facilita la extensión y el	Código estructurado por paquetes, fuerte tipado y enfoque minimalista reducen la deuda técnica.	Los esquemas y resolvers son fáciles de extender sin romper compatibilidad (principio abierto/cerrado).	Esquemas flexibles que permiten evolución del modelo sin migraciones costosas.

	mantenimiento.			
6. Portabilidad	Aplicaciones web ejecutables en cualquier navegador o entorno multiplataforma.	Compila binarios para múltiples sistemas operativos sin dependencia de intérpretes.	Protocolo estándar independiente del lenguaje, portable a cualquier entorno web.	Funciona en diferentes sistemas y nubes (Atlas, Docker, servidores locales).
7. Seguridad	Módulos integrados para autenticación, protección contra XSS, CSRF y sanitización de entradas.	Manejo de autenticación y cifrado con librerías seguras, control de acceso a nivel de API.	Permite definir autorización y control de acceso por campo o tipo de dato.	Soporte para cifrado en reposo y en tránsito, control de roles y usuarios.
8. Compatibilidad	Se comunica fácilmente con APIs REST o GraphQL mediante HTTP y JSON.	Exposición de APIs REST o GraphQL que permiten interoperar con múltiples frontends.	Diseñado para interoperar entre distintos lenguajes y plataformas.	Conectores para múltiples lenguajes y frameworks (Go, Node, Python, etc.).

Matriz de análisis de Tácticas vs Temas

Atributo de calidad	Angular	Go (Golang)	GraphQL	MongoDB
Fiabilidad (Reliability)	Manejo de errores global (`ErrorHandler`) y reintentos automáticos con `RxJS`.	Uso de *panic recovery*, *retry policies* y manejo de concurrencia segura (canales, `sync`).	Validación de esquemas y control de errores en resolvers.	**Replicación** (Replica Sets), *failover automático* y respaldo distribuido.
Rendimiento (Performance Efficiency)	**Lazy loading**, *Ahead-of-Time (AOT)* y *change detection strategy* optimizada.	*Goroutines*, *worker pools*, *profiling* con `pprof`, y uso de cachés en memoria.	**Caching** de resultados de consultas (Apollo Cache / DataLoader).	**Índices**, *sharding* (fragmentación horizontal) y *aggregation pipelines* optimizados.

Seguridad (Security)	Protección contra XSS, CSRF y sanitización de entradas (`DomSanitizer`).	Uso de **JWT**, cifrado TLS/HTTPS, *middleware* de autenticación/ autorización.	**Control de acceso por campo o esquema**, validación de tokens y roles.	**Cifrado en reposo y en tránsito**, control de roles y usuarios, auditoría de accesos.
Mantenibilidad (Maintainability)	Arquitectura modular y patrones como *Dependency Injection* y *Facade*.	**Clean Architecture**, división en capas (`repository`, `service`, `handler`), *testing unitario*.	**Resolver modular**, separación de esquemas y *versionado de API*.	Modelos versionados y uso de **migraciones controladas** (por ejemplo, `migrate-mongo`).
Escalabilidad (Scalability)	Dividir la aplicación en módulos y cargar recursos bajo demanda.	**Microservicios**, balanceo de carga, y *horizontal scaling*.	**Federación de esquemas GraphQL** y *schema stitching* para múltiples APIs.	**Sharding**, *replica sets* y escalado horizontal con múltiples nodos.
Usabilidad (Usability)	Interfaces reactivas, accesibilidad (a11y), *lazy rendering* y validación visual.		**Consultas personalizables** que entregan solo los datos requeridos por el cliente.	
Portabilidad (Portability)	Compatible con cualquier navegador o plataforma web moderna.	**Compilación cruzada** para varios sistemas operativos.	API universal basada en HTTP y JSON.	Despliegue multiplataforma (Atlas, Docker, Kubernetes).
Compatibilidad / Interoperabilidad	Comunicación estandarizada con APIs REST y GraphQL (HTTP/JSON).	Creación de APIs REST/GraphQL interoperables con cualquier frontend.	Interfaz común de consulta compatible con diversos clientes.	Conectores oficiales para múltiples lenguajes y frameworks.

Matriz de análisis de Patrones vs Temas

Tecnologías	Patrones emergentes / integrados	Descripción / Ejemplo práctico
Angular + GraphQL	**BFF (Backend for Frontend)**	GraphQL actúa como capa intermedia que entrega al frontend solo los datos que necesita.
	Resolver Pattern	Cada query/mutation en GraphQL se traduce a funciones que Angular puede consumir.
	Cache Pattern (Apollo)	Angular usa Apollo Client con caché local para optimizar peticiones.
	Observer + Reactive Pattern	Angular recibe datos GraphQL como streams reactivos (Observable<QueryResult>).
Go + GraphQL	**Resolver + Repository Pattern**	Los resolvers de GraphQL llaman a repositorios Go para obtener datos.
	DataLoader Pattern	Optimiza múltiples consultas concurrentes a MongoDB.
	Dependency Injection / Service Pattern	Go inyecta servicios y controladores en los resolvers.
GraphQL + MongoDB	**Repository / Data Abstraction Pattern**	GraphQL oculta la estructura real de MongoDB tras resolvers y modelos.
	Aggregate Pattern	Las queries GraphQL usan agregaciones de MongoDB para cálculos complejos.
	Batch Loading Pattern	DataLoader combina consultas similares en una sola operación MongoDB.
	Schema Stitching / Federation	Permite unir múltiples fuentes MongoDB en una sola API GraphQL.
Go + MongoDB	**Repository Pattern**	Capa de datos con interfaces desacopladas.
	Unit of Work Pattern	Agrupa operaciones MongoDB en una misma transacción lógica.

	Factory Pattern	Creación de conexiones y modelos de manera centralizada.
	Adapter Pattern	Go traduce estructuras MongoDB en entidades de dominio.
Angular + Go (API REST o GraphQL)	**Client–Server Pattern**	Angular envía peticiones HTTP o GraphQL a un servidor Go.
	DTO / Data Transfer Object Pattern	Los datos se trasladan entre capas en formatos bien definidos.
	Observer + Reactive Composition	Angular reacciona a respuestas de Go mediante observables.
Angular + GraphQL + MongoDB	**BFF + Repository + Cache Pattern**	Angular pide datos al servidor GraphQL, que usa repositorios MongoDB.
Go + GraphQL + MongoDB	**Clean Architecture / Hexagonal Pattern**	Go implementa resolvers que dependen de interfaces (repositorios) en lugar de la BD directamente.
	DataLoader + Aggregation	Eficiencia en la carga de datos combinando múltiples consultas.

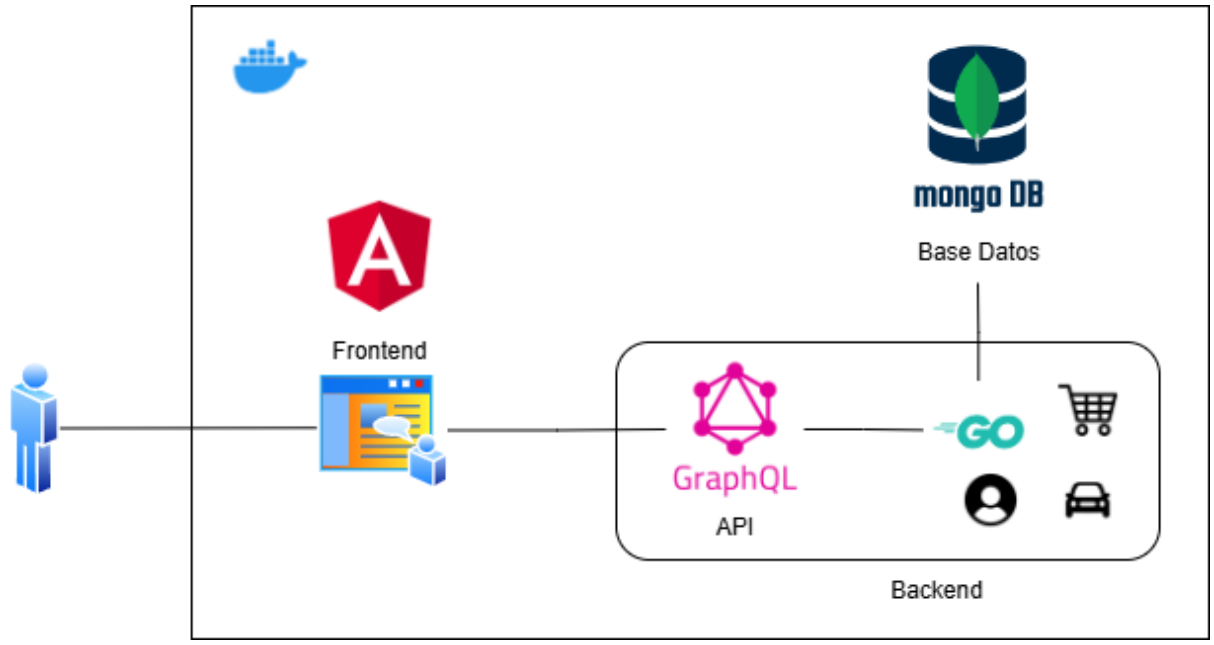
Mercado laboral Vs Temas

Criterio	Angular	Golang (Go)	ZeroMQ (ZMQ)	MongoDB
Rol en el Stack	Frontend (Interfaz de Usuario)	Backend (API de Alto Rendimiento)	Mensajería Asíncrona / Transporte de Datos	Base de Datos NoSQL (Almacenamiento de Documentos)
Demanda Laboral	Alta y Estable. Requisito fundamental en grandes empresas y	Creciente y de Alto Valor. Muy demandado en Microservicios y sistemas de infraestructura.	Media a Alta y en Auge. Buscado en proyectos que requieren optimización	Alta y Consolidada. Lidera el mercado NoSQL, un requisito común en la

	proyectos complejos.		de red y flexibilidad de frontend.	mayoría de los stacks modernos.
Sector Principal	Software Empresarial, marketing digital, Sofware educativo, etc.	Startups, Fintech, Plataformas de Streaming, Infraestructura en la Nube.	E-commerce, Aplicaciones Móviles, Startups, Empresas con múltiples plataformas de frontend.	Comercio Electrónico, Desarrollo Full-Stack, Aplicaciones Móviles, Big Data.
Cargos Clave	Desarrollador Frontend, Desarrollador Full-Stack.	Desarrollador Backend Go, Ingeniero de Microservicios.	Desarrollador Full-Stack, Ingeniero de API Senior.	Desarrollador Full-Stack, DBA NoSQL, Ingeniero / científico de Datos.
Salario Promedio Estimado (COP Mensual)	Junior: $3.000.000 Senior: $9.000.000	Junior: $3.000.000 Senior: $13.000.000	Agregado a Go / Angular: $5.000.000 - $15.000.000	Junior: $3.000.000 Senior: $12.000.000

Diagramas de arquitectura

Alto nivel



Diagramas C4

Diagrama de Contexto:

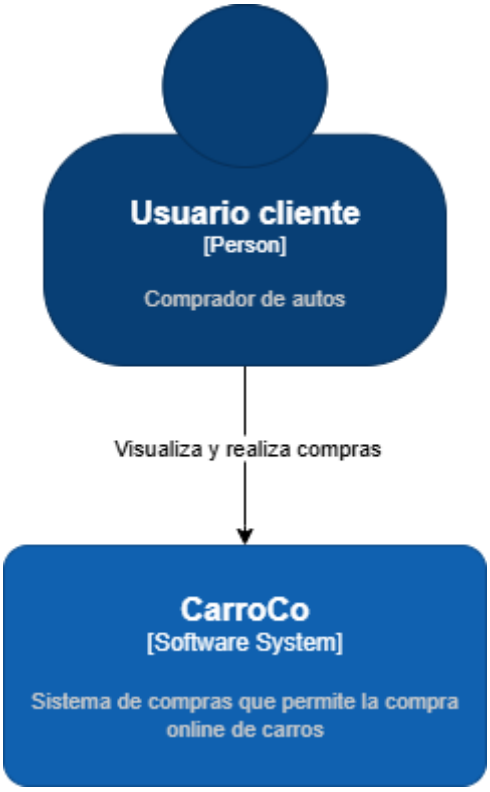


Diagrama de Contenedores:

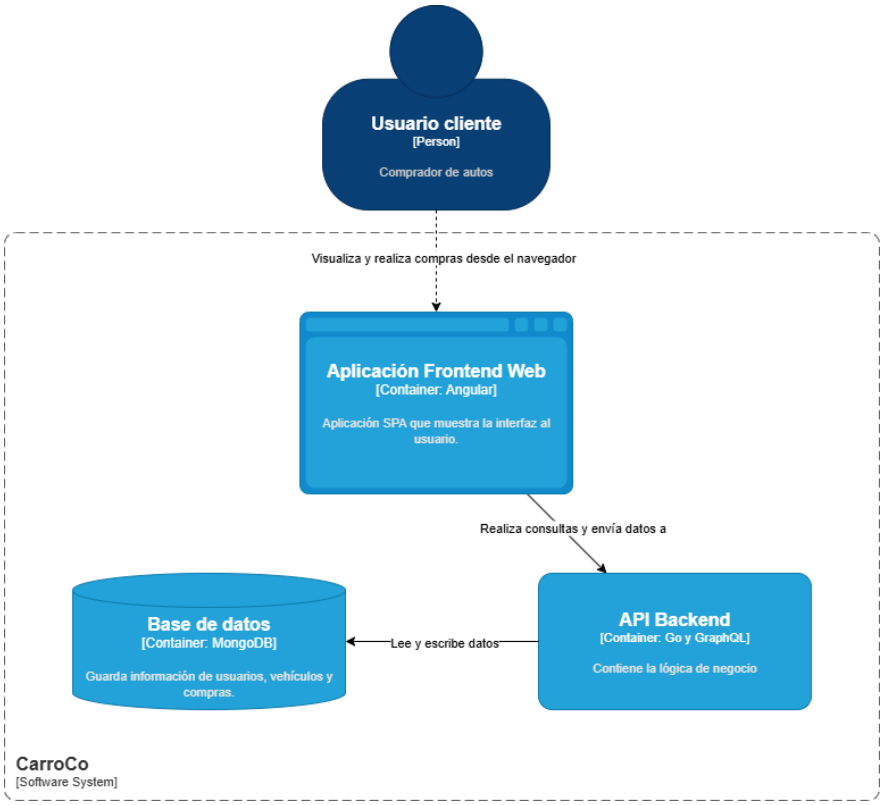


Diagrama de componentes:

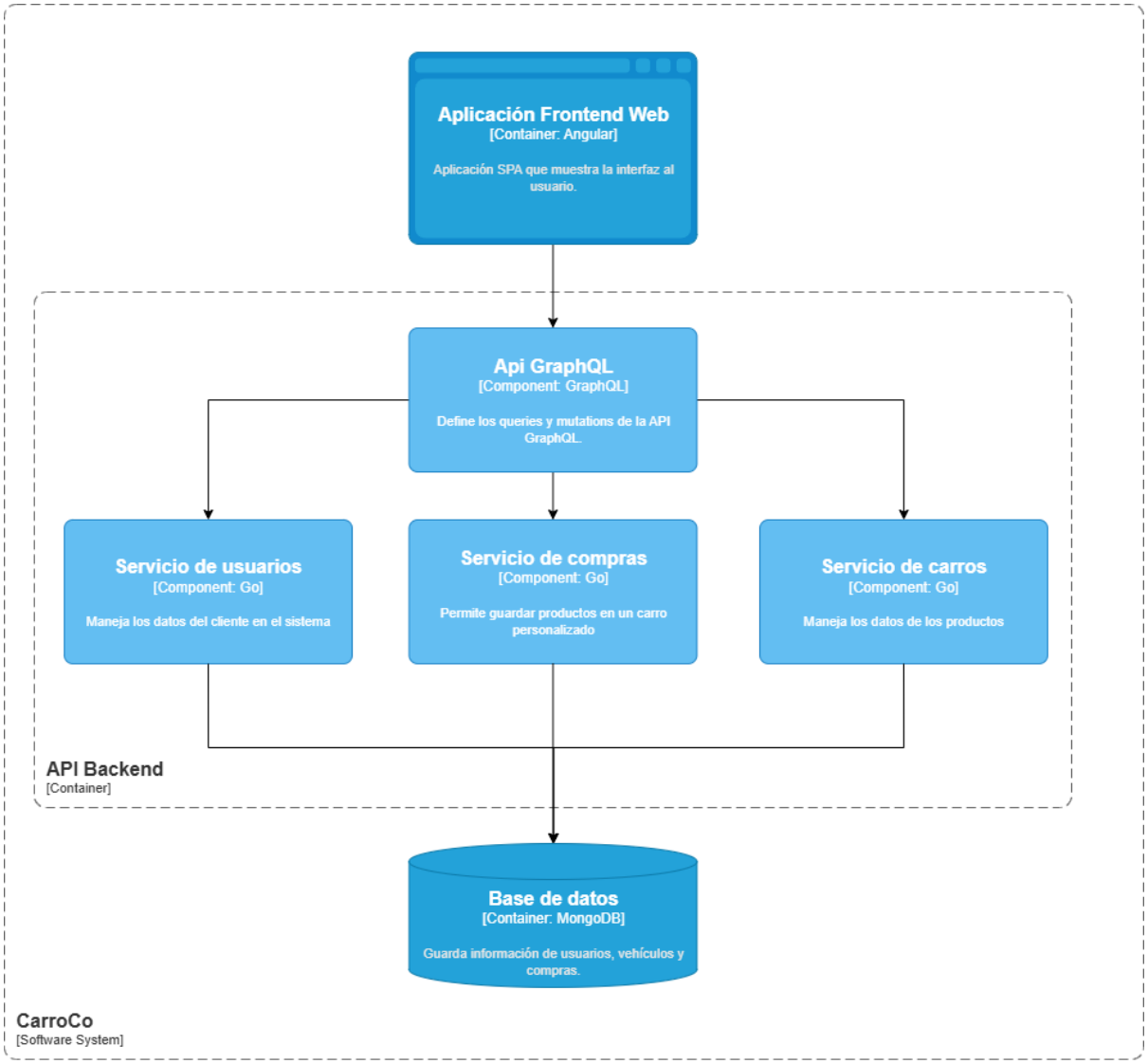

Aplicación Frontend Web
[Container: Angular]
Aplicación SPA que muestra la interfaz al usuario.
Api GraphQL
[Component: GraphQL]
Define los queries y mutations de la API GraphQL.
Servicio de usuarios
[Component: Go]
Maneja los datos del cliente en el sistema
Servicio de compras
[Component: Go]
Permite guardar productos en un carro personalizado
Servicio de carros
[Component: Go]
Maneja los datos de los productos
API Backend
[Container]
Base de datos
[Container: MongoDB]
Guarda información de usuarios, vehículos y compras.
CarroCo
[Software System]

Diagrama Dinámico C4

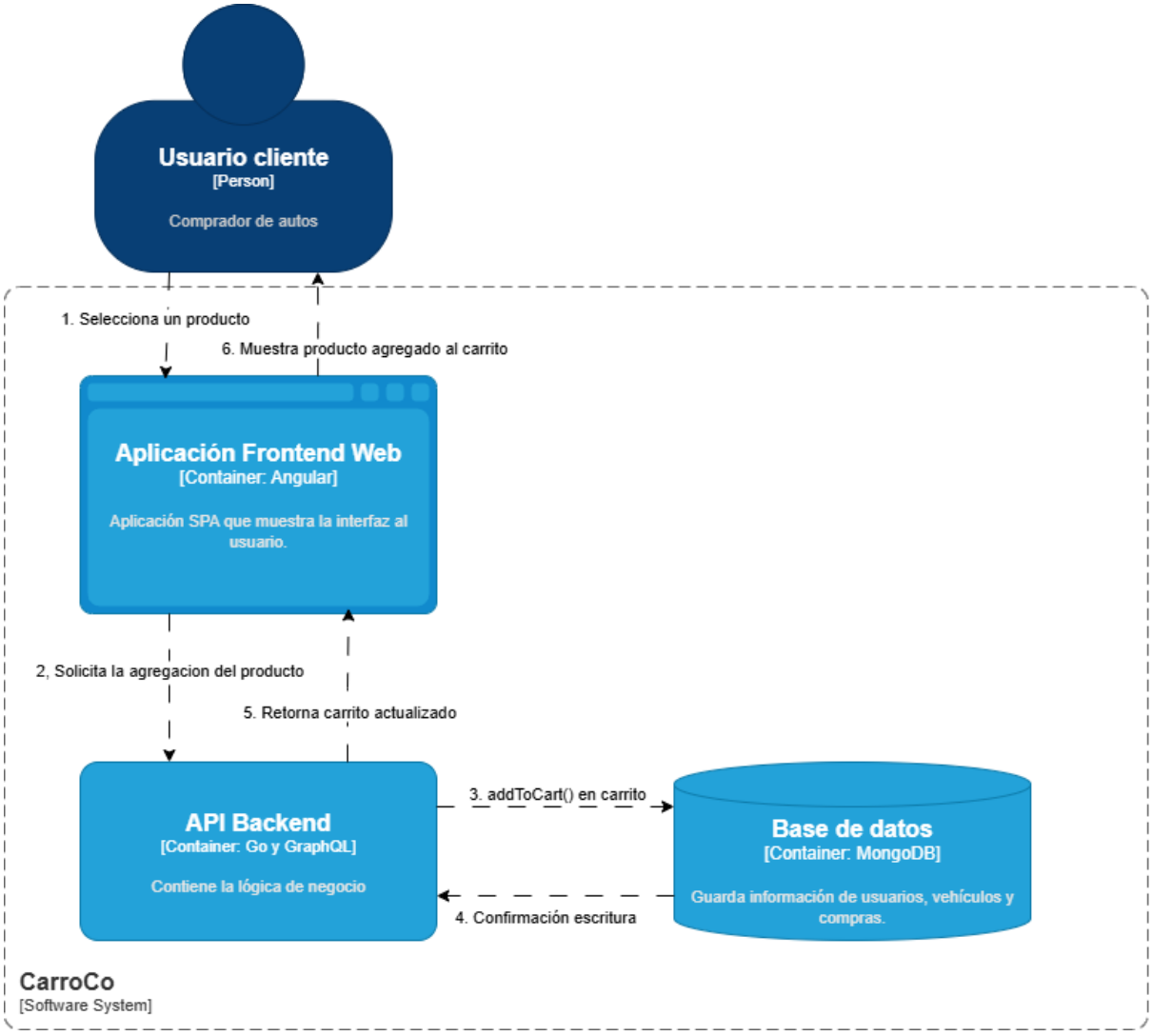


Diagrama de despliegue C4