Arquitectura:

Domain Driven Design (DDD)

Capas de DDD:

1. Capa de Dominio (Domain Layer)
2. Capa de Aplicación (Application Layer)
3. Capa de Infraestructura (Infrastructure Layer)
4. Capa de Interfaces o Presentación (User Interface Layer)

Relación entre las capas:

La capa de Interfaces interactúa con la capa de Aplicación a través de controladores o APIs.

La capa de Aplicación usa la **capa de Dominio** para implementar la lógica de negocio.

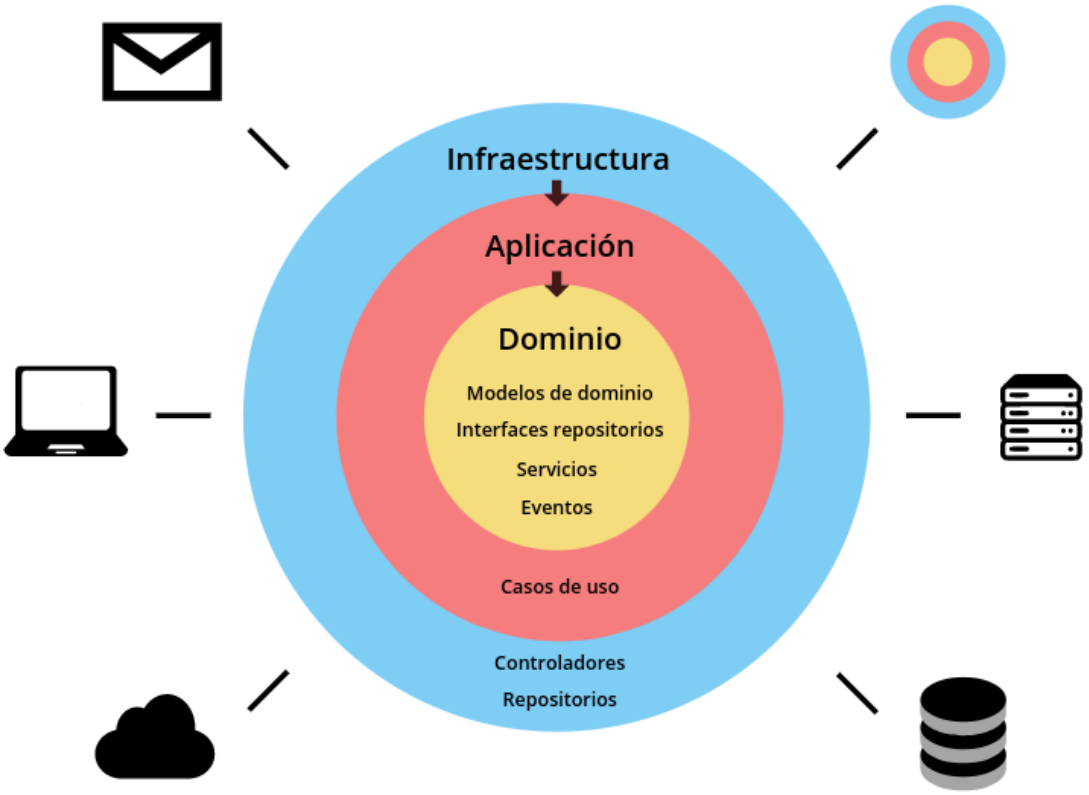
La capa de Infraestructura proporciona los servicios técnicos necesarios a las otras capas, especialmente la persistencia de los datos.

Ventajas de usar DDD:

Modularidad: DDD te ayuda a dividir la aplicación en áreas del dominio (bounded contexts) que pueden ser gestionadas y desarrolladas de forma independiente, lo que se alinea bien con tecnologías como Node.js y React.

Claridad en la lógica de negocio: Con DDD, permite organizar el código de manera que el dominio (las reglas del negocio) esté separado de la infraestructura (como bases de datos y APIs), lo que ayuda a mantener la aplicación limpia y escalable.

Escalabilidad: DDD te permite mantener las diferentes partes del sistema aisladas, haciendo que sea más fácil escalar o cambiar componentes sin afectar al sistema completo.



Tecnologías:

- Backend: Node.js con Express.js

Node.js es un entorno de ejecución para JavaScript que se ejecuta en el lado del servidor. Permite construir aplicaciones altamente escalables y eficientes en el manejo de solicitudes concurrentes.
Express.js es un framework minimalista y flexible que simplifica la creación de aplicaciones web y APIs en Node.js

- Frontend: React con Next.js

React es una biblioteca de JavaScript para construir interfaces de usuario interactivas y reactivas. **Next.js** es un framework que amplía React añadiendo renderizado del lado del servidor (SSR) y optimizaciones para mejorar el rendimiento de las aplicaciones web.

Ambos entornos están desarrollados en JavaScript.

El uso de tecnologías basadas en JavaScript, tanto en el frontend como en el backend, facilita el desarrollo de nuevas funcionalidades y el mantenimiento del código.

Base de datos: MySQL

MySQL es una base de datos **relacional**. Los datos se almacenan en tablas (filas y columnas) y las relaciones entre los datos se definen mediante claves foráneas. Es ideal para datos estructurados que siguen un formato definido.

MySQL es ideal para aplicaciones que necesitan manejar datos estructurados y bien definidos. En el sistema de control de stock de camiones, es fundamental tener tablas relacionadas para gestionar productos, camiones, clientes, entradas y salidas. MySQL permite crear un esquema relacional claro y escalable para estas necesidades.