

# Sesión 02

## Inyección de Dependencias (DI)

Instructor:

**ERICK ARÓSTEGUI**

[earostegui@galaxy.edu.pe](mailto:earostegui@galaxy.edu.pe)



8  
NET  
FULL-STACK  
DEVELOPER

# ÍNDICE

- 01** Inyección de Dependencias

---
- 02** Desarrollo de casos y aplicaciones DI

---
- 03** Diseñando y Modelando Minimal APIs

---
- 04** Instalando y configurando entorno

---
- 05** Creación de la solución y proyectos de la arquitectura propuesta

---

01



# Inyección de Dependencias

# → Inyección de Dependencias

## ¿Qué es Inyección de Dependencias?



Inyección de Dependencias (en inglés **Dependency Injection**, DI), es un patrón de diseño de software usado en la Programación Orientada a Objetos.

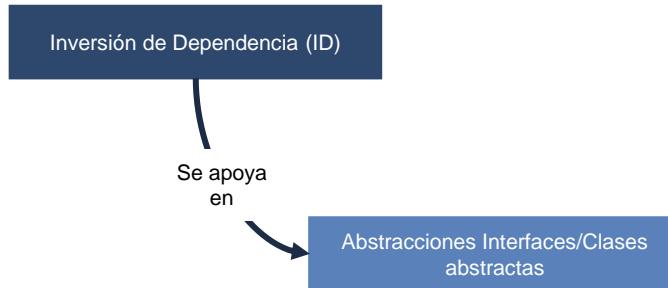
**Permite desarrollar componentes desacoplados** permitiendo una fácil gestión de cambios a futuro, implementación de pruebas unitarias, factoría para creación de instancias, prevención de fugas de memoria, entre otros.

# → Inyección de Dependencias

## Conceptos

# → Inyección de Dependencias

## Conceptos



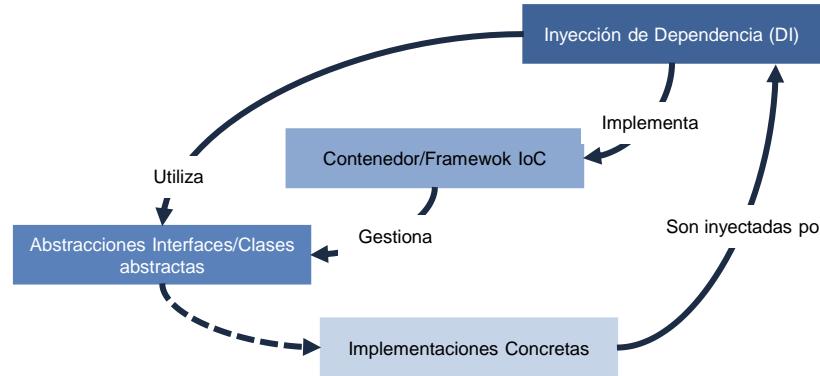
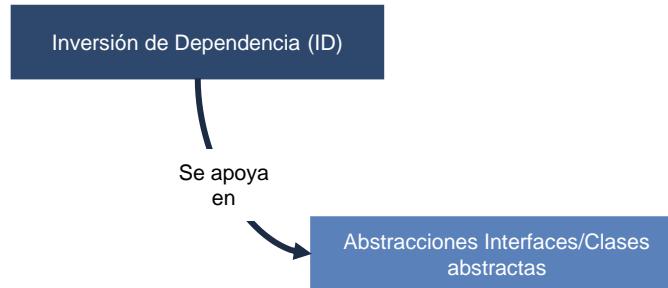
### INVERSIÓN DE DEPENDENCIA (ID)

**Principio de Diseño:** Las clases de alto nivel **no deberían depender de clases de bajo nivel**; ambas deberían **depender de abstracciones** (interfaces o clases abstractas).

**Objetivo:** Reducir el acoplamiento entre clases para mejorar la modularidad y facilitar la mantenibilidad y la extensibilidad del código.

# → Inyección de Dependencias

## Conceptos



### INVERSIÓN DE DEPENDENCIA (ID)

**Principio de Diseño:** Las clases de alto nivel **no deberían depender de clases de bajo nivel**; ambas deberían **depender de abstracciones** (interfaces o clases abstractas).

**Objetivo:** Reducir el acoplamiento entre clases para mejorar la modularidad y facilitar la mantenibilidad y la extensibilidad del código.

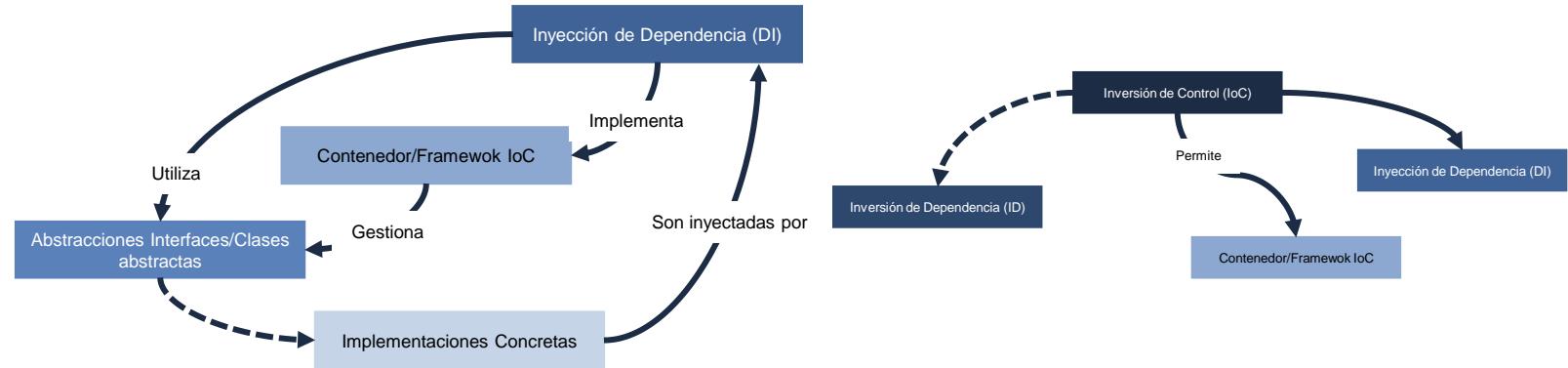
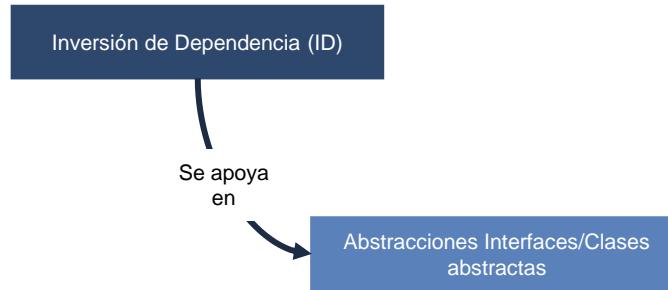
### INYECCIÓN DE DEPENDENCIA (DI)

**Patrón de Diseño:** Una **forma de implementar la inversión de control** donde las dependencias (objetos) son proporcionadas a una clase en lugar de ser creadas por la clase.

**Métodos:** Las dependencias pueden ser inyectadas a través de constructores, métodos setter o directamente en propiedades.

# Inyección de Dependencias

## Conceptos



### INVERSIÓN DE DEPENDENCIA (ID)

**Principio de Diseño:** Las clases de alto nivel **no deberían depender de clases de bajo nivel**; ambas deberían **depender de abstracciones** (interfaces o clases abstractas).

**Objetivo:** Reducir el acoplamiento entre clases para mejorar la modularidad y facilitar la mantenibilidad y la extensibilidad del código.

### INYECCIÓN DE DEPENDENCIA (DI)

**Patrón de Diseño:** Una **forma de implementar la inversión de control** donde las dependencias (objetos) son proporcionadas a una clase en lugar de ser creadas por la clase.

**Métodos:** Las dependencias pueden ser inyectadas a través de constructores, métodos setter o directamente en propiedades.

### INVERSIÓN DE CONTROL (IOC)

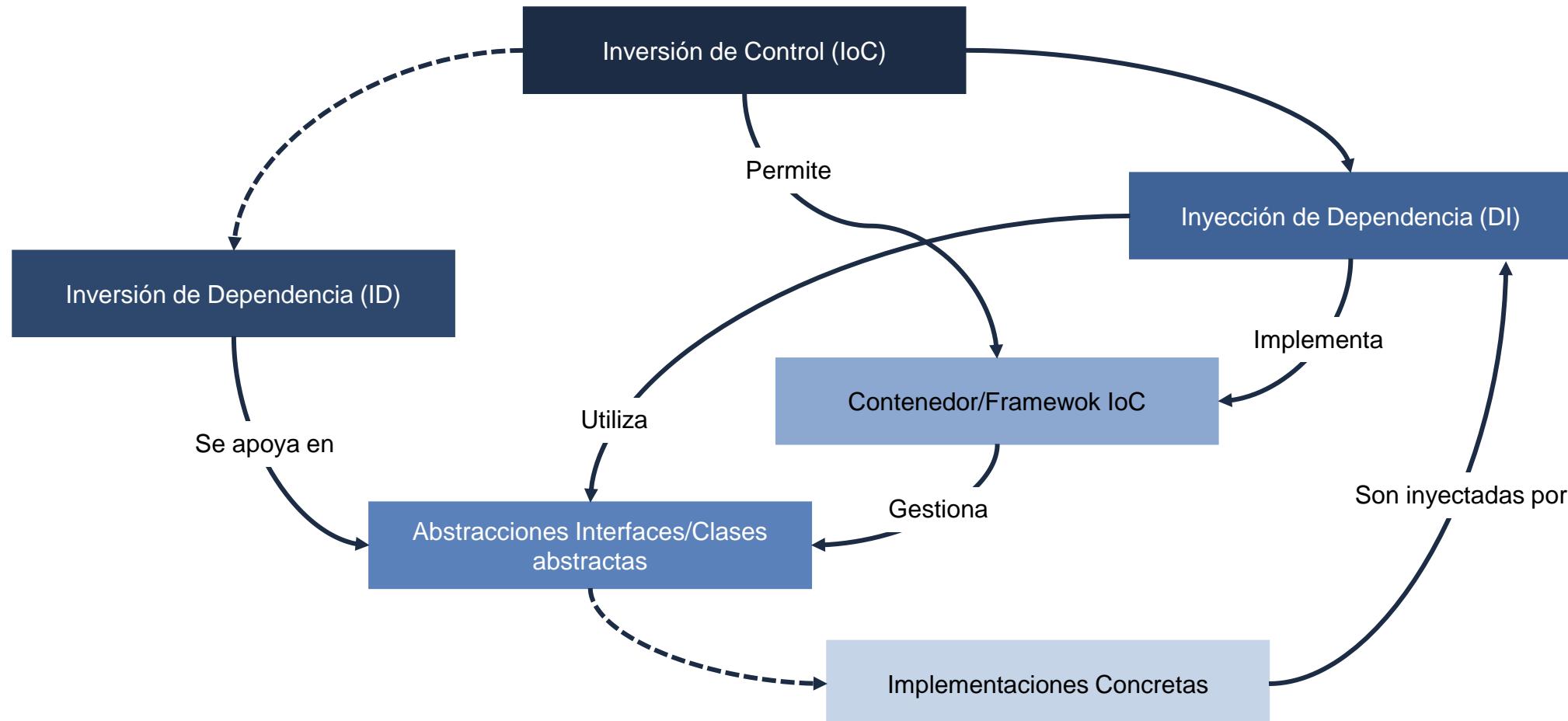
**Principio General:** Refiere a invertir el control del flujo del programa, **delegándolo a un contenedor o framework externo**.

**Implementación:** Se logra mediante técnicas como la inyección de dependencia.

**Efecto:** Las dependencias entre componentes se gestionan externamente en lugar de que cada componente gestione sus propias dependencias.

# → Inyección de Dependencias

## Conceptos

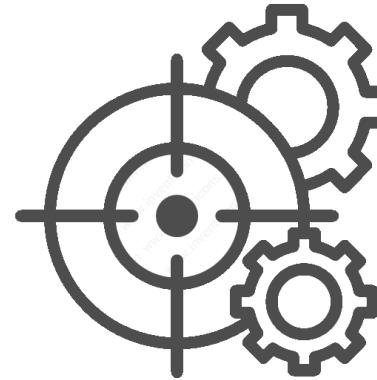


# → Inyección de Dependencias

## Tipos de Inyección de Dependencias en .NET



Transient



Scoped



Singleton

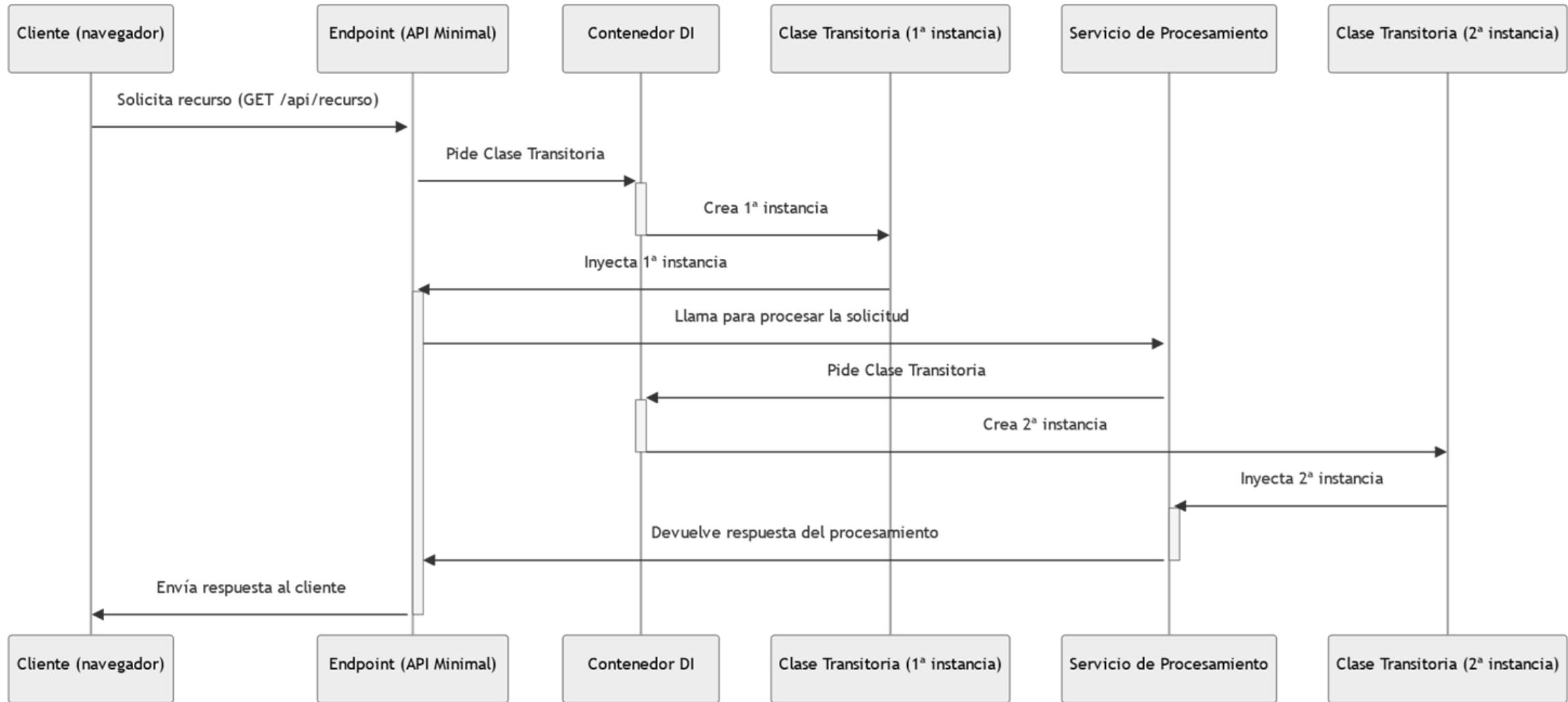
# → Inyección de Dependencias

## Uso de Transient



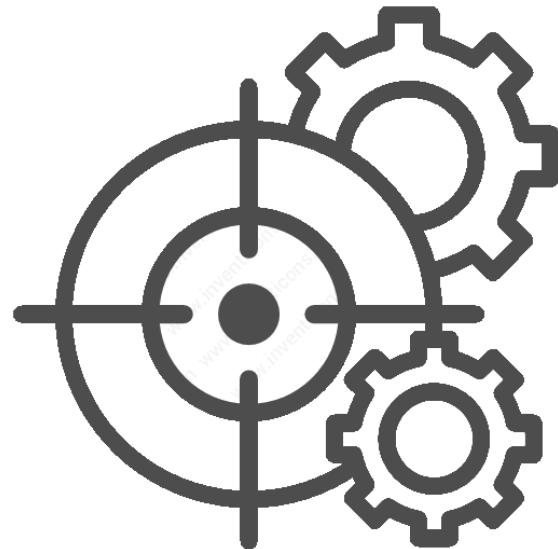
Crea y comparte una instancia del servicio cada vez que la aplicación lo solicite. **El componente se puede agregar como transitorio** utilizando el método **AddTransient()** para **IServiceCollection**

# → Inyección de Dependencias (Transient)



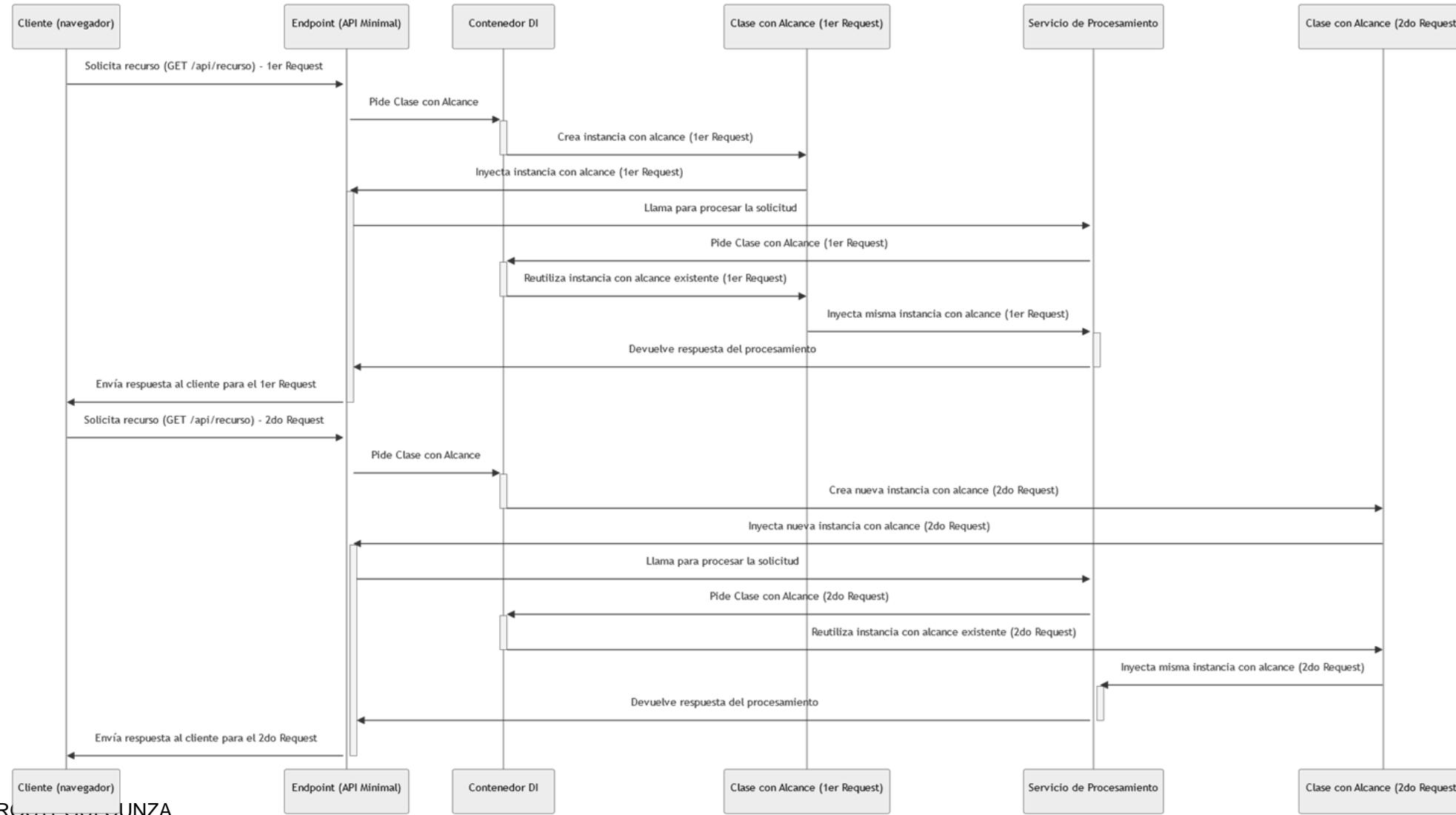
# → Inyección de Dependencias

## Uso de Scoped



Se crea una vez por alcance y se implementa utilizando el método **AddScoped()**. La mayoría de las veces, el alcance se refiere a la **duración de una solicitud http**.

# → Inyección de Dependencias (Scoped)

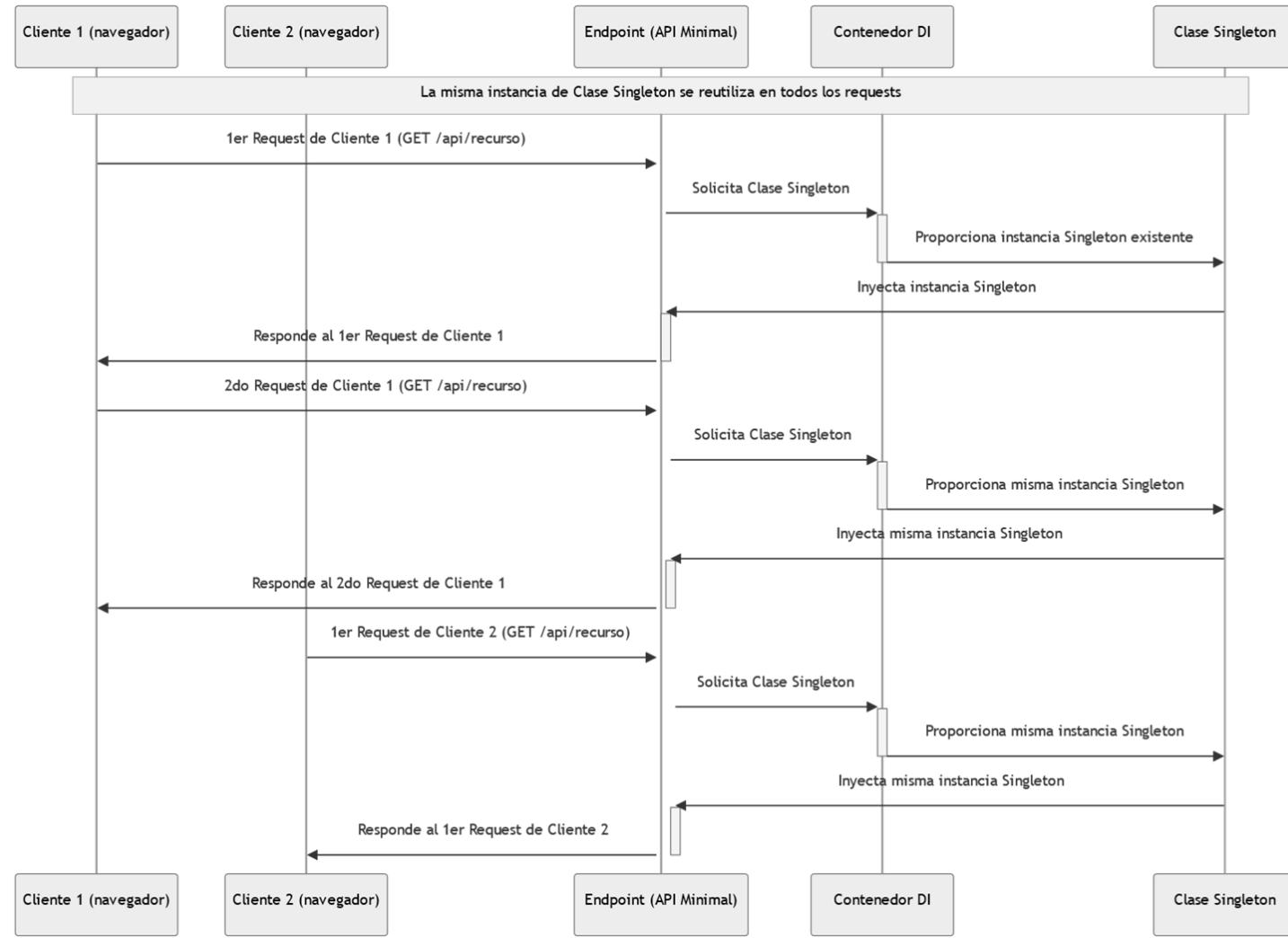


## Uso de Singleton



Crea y comparte una única instancia del **componenete** a lo largo de la vida de la **aplicación**. El componente se puede agregar como singleton utilizando el método **AddSingleton()** para **IServiceCollection**.

# → Inyección de Dependencias (Singleton)



# → Inyección de Dependencias

## Beneficios



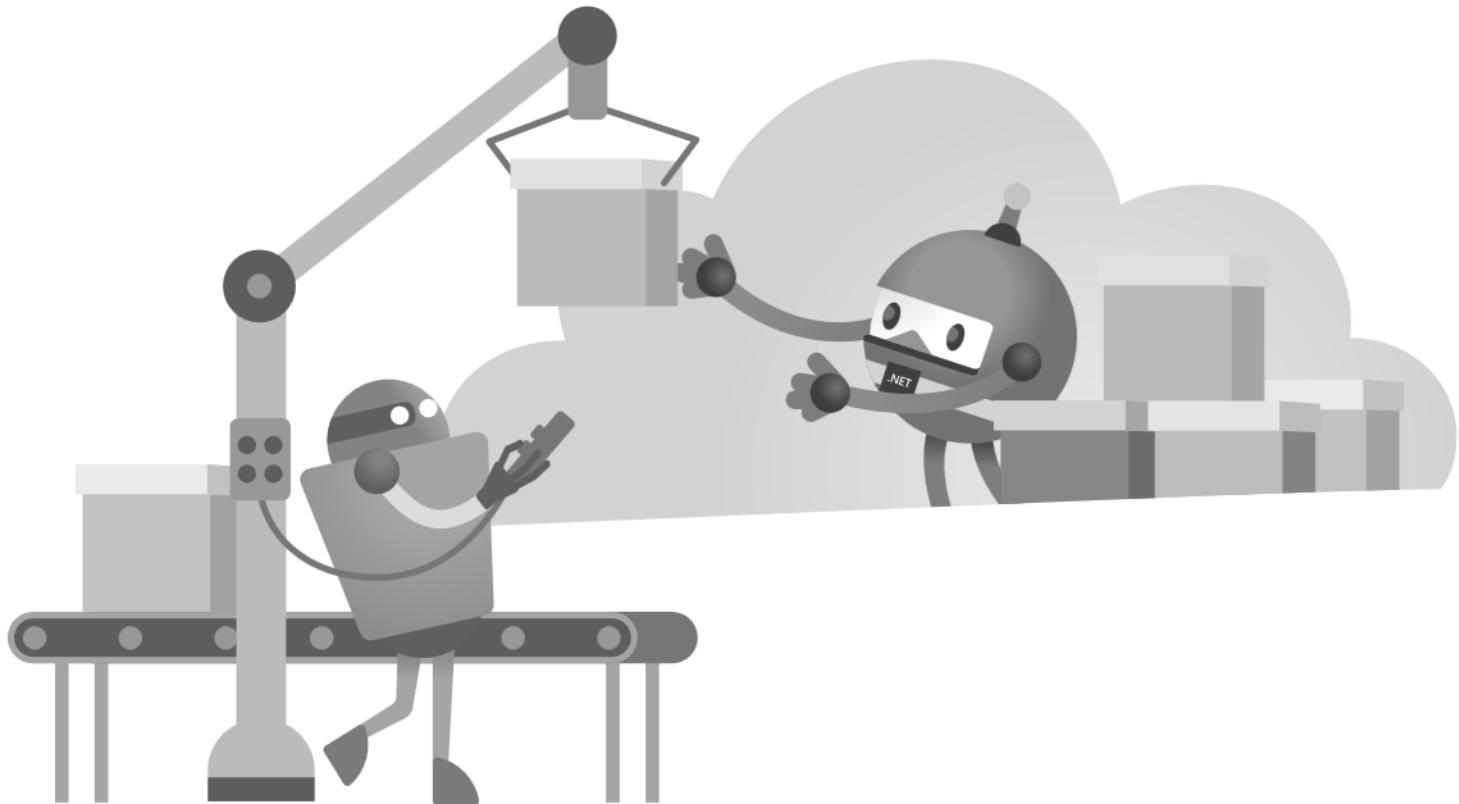
- Aplicación de **bajo acoplamiento**.
- En caso de cualquier cambio en la implementación del servicio, simplemente cambie el nombre del archivo servicio o Implementación en el archivo program.cs.
- No más cambios en ninguna clase del controlador.
- Facilita la vida del desarrollador

02



## Desarrollo de casos y aplicaciones DI

# Desarrollo de casos y aplicaciones DI



# 03



## Diseñando y Modelando Minimal APIs

# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación

# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación

Frontend

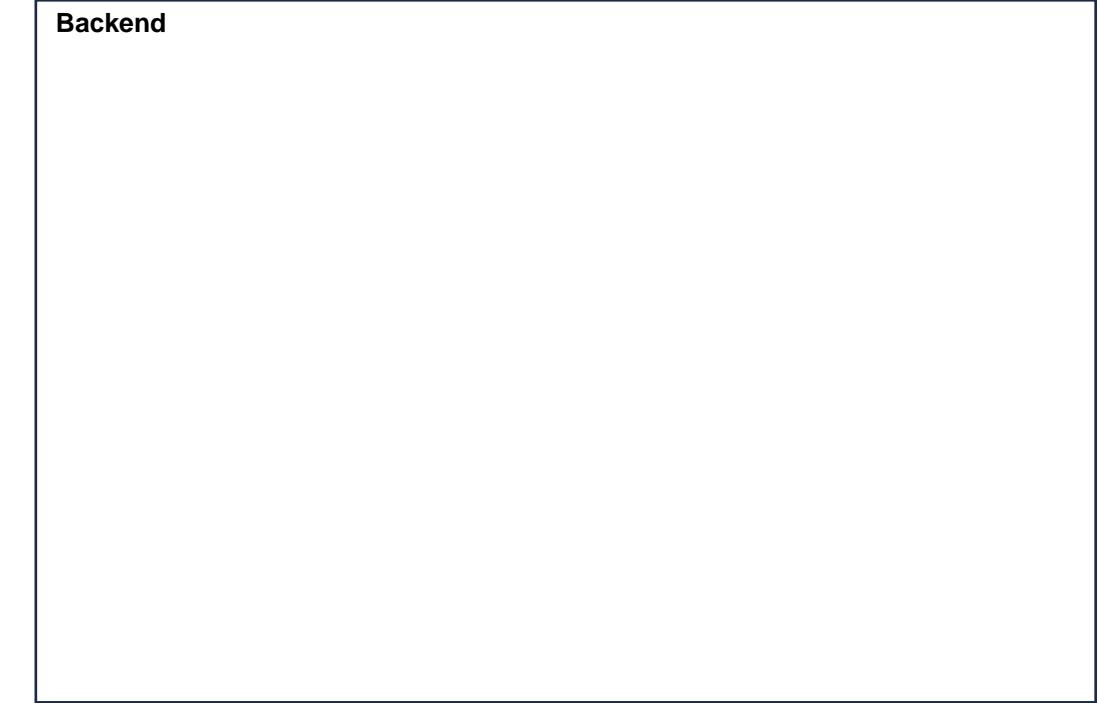
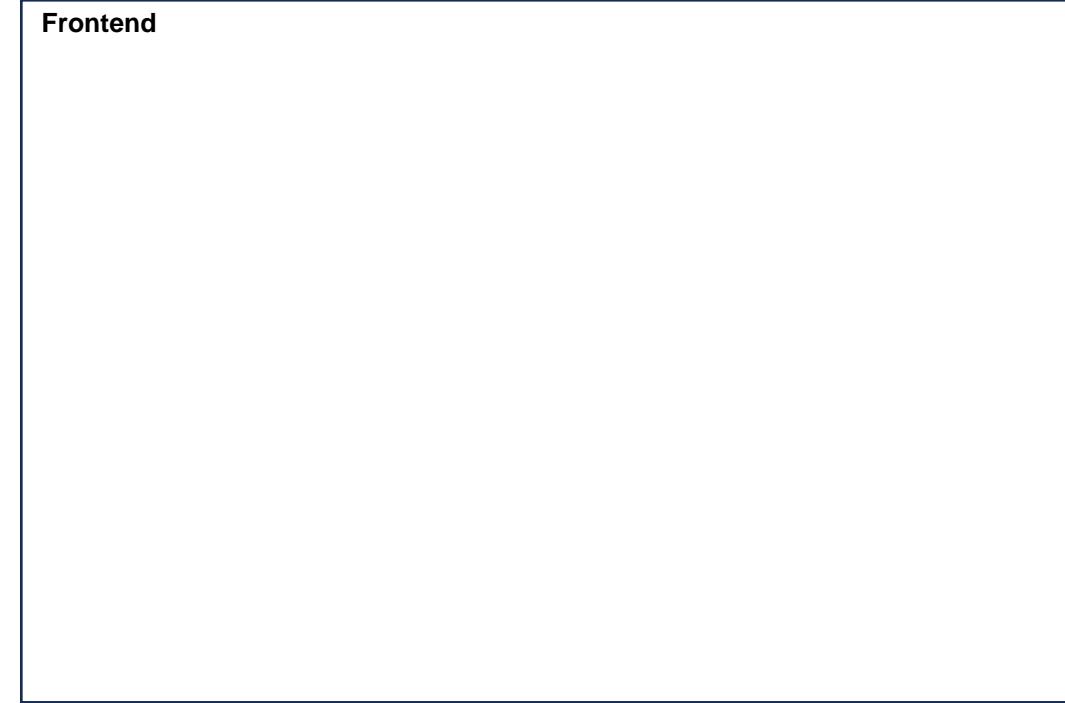


# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación

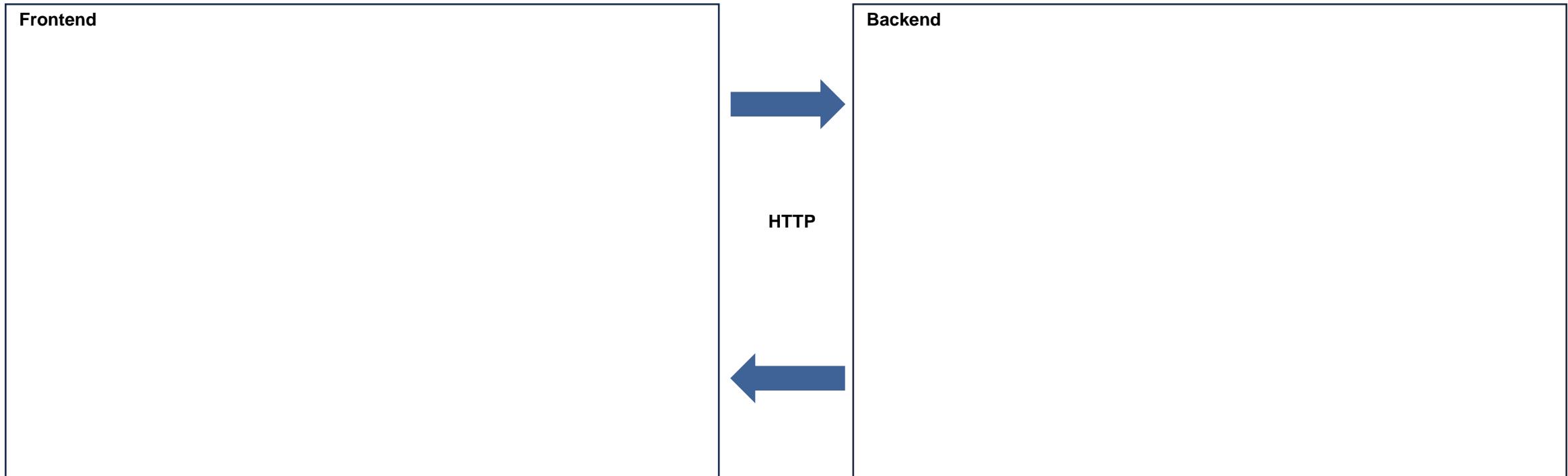
Frontend

Backend



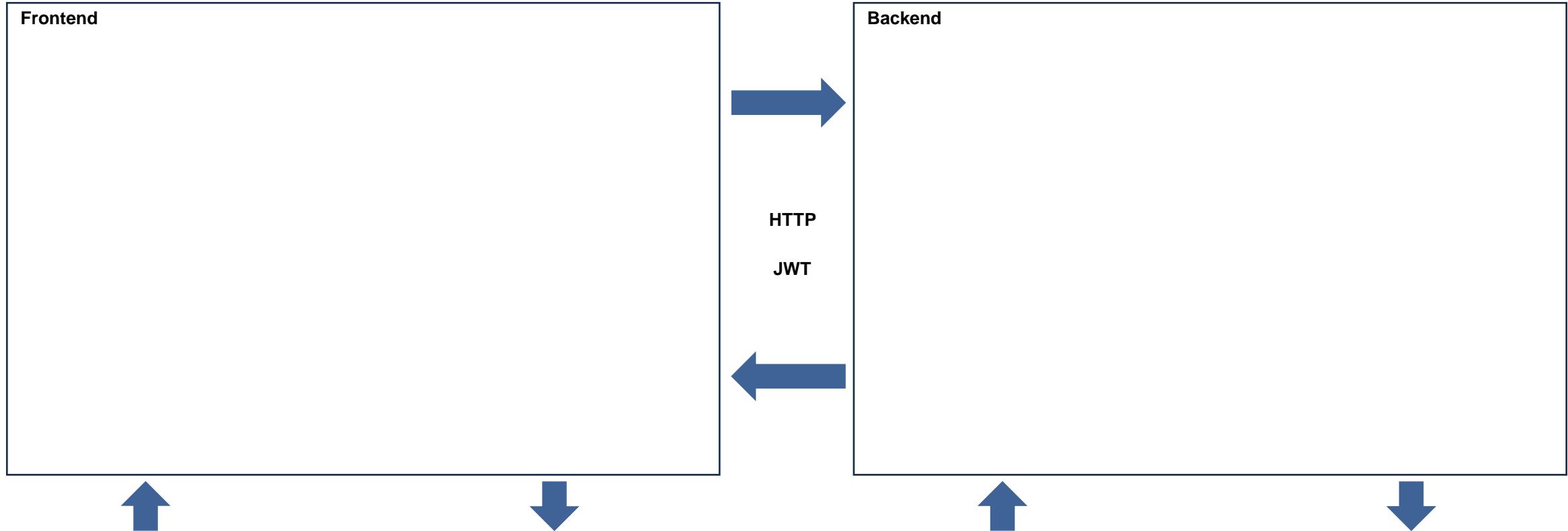
# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación



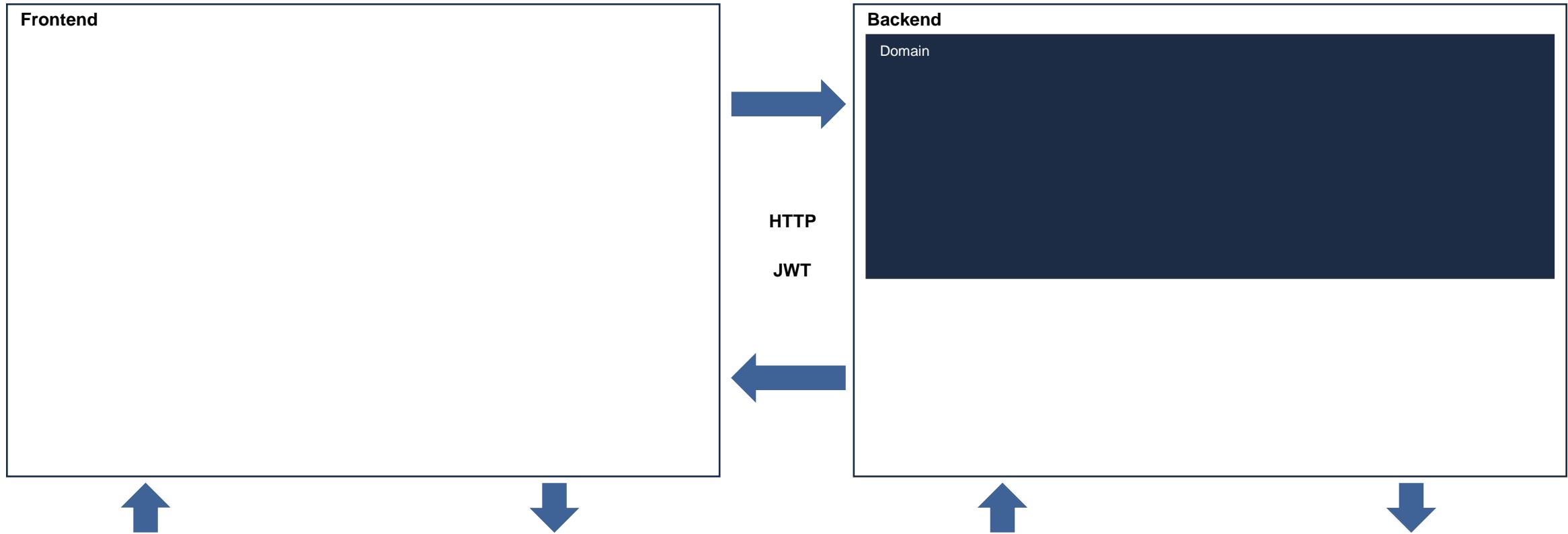
# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación



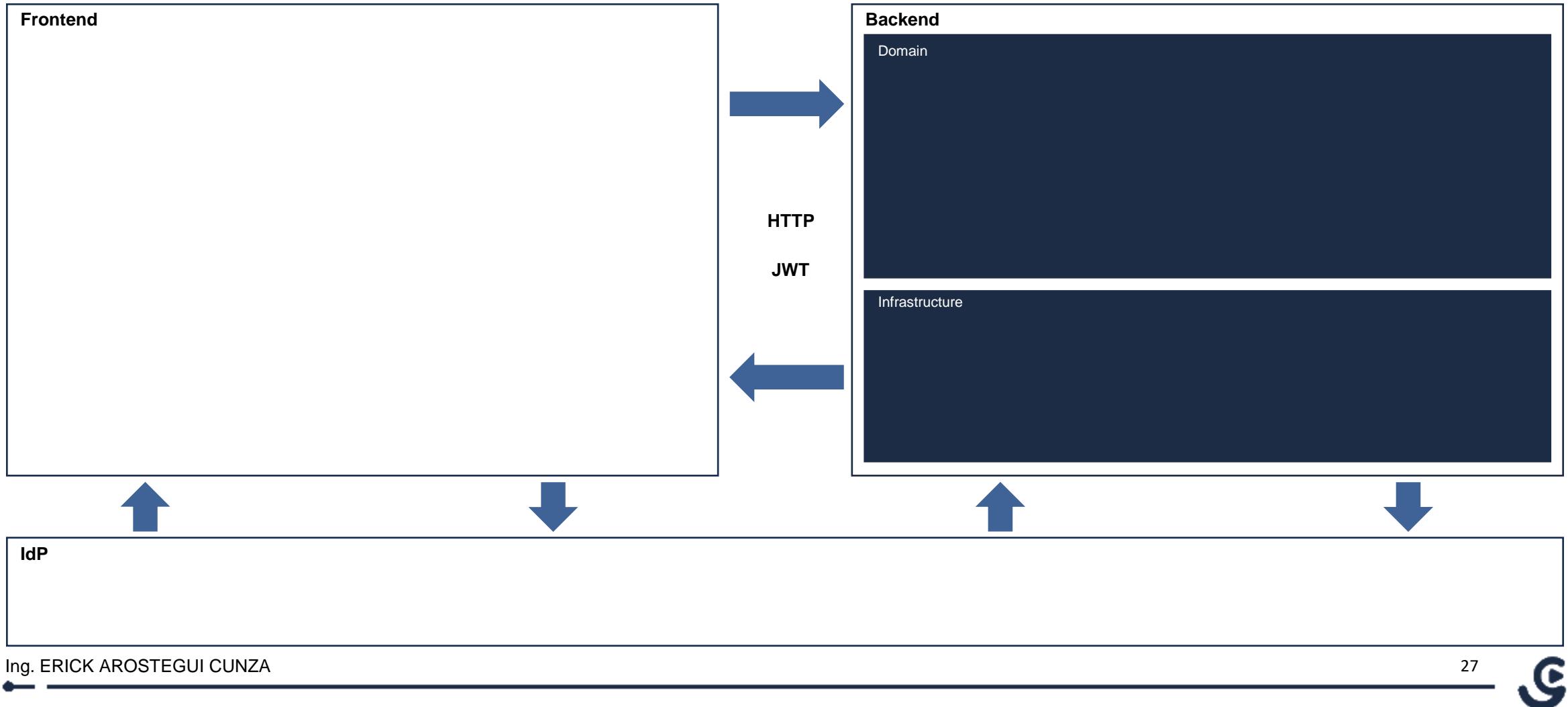
# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación



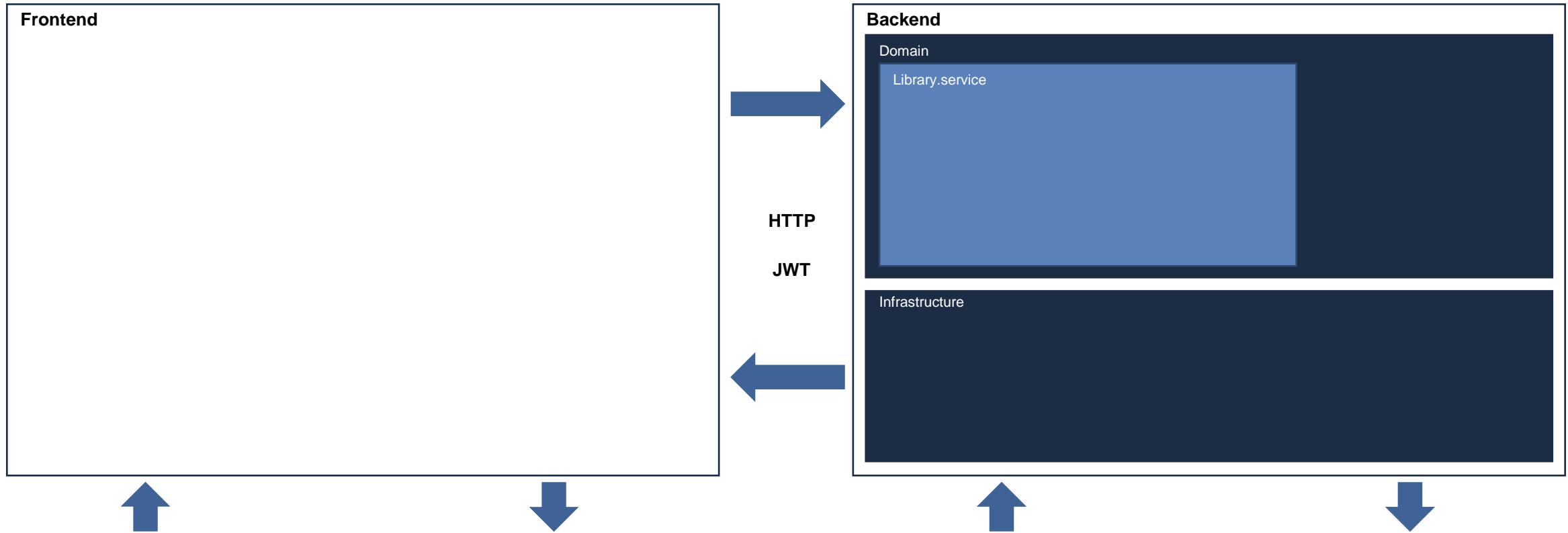
# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación



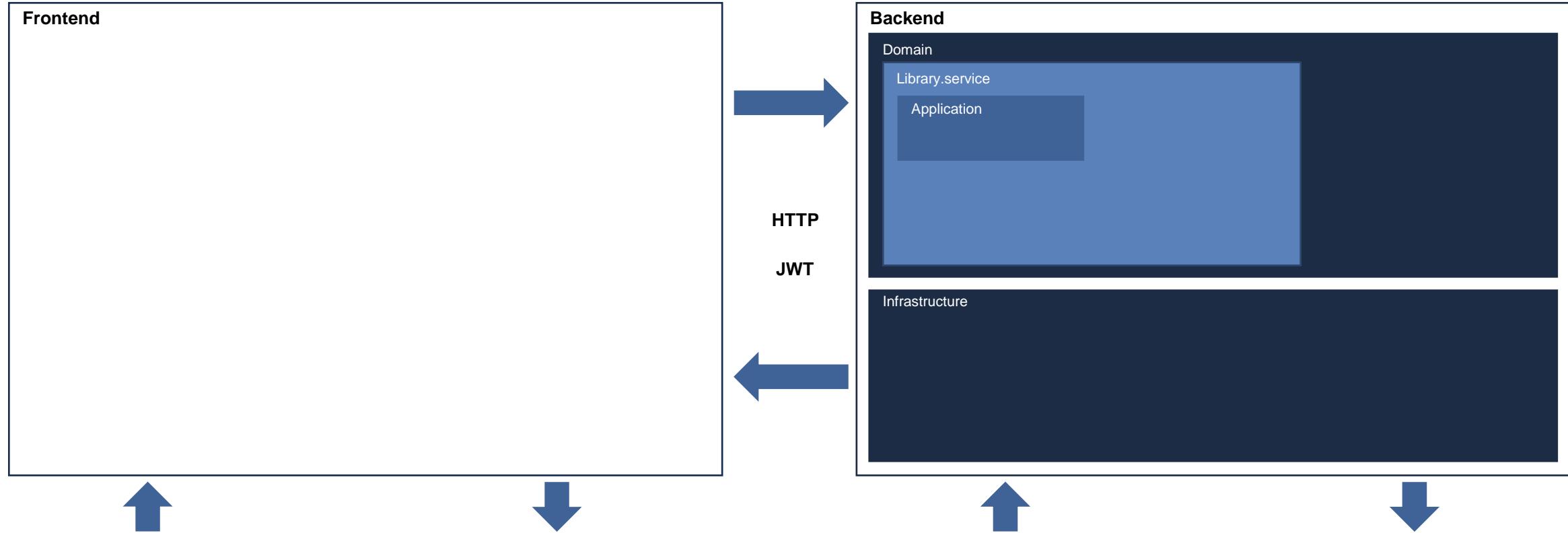
# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación



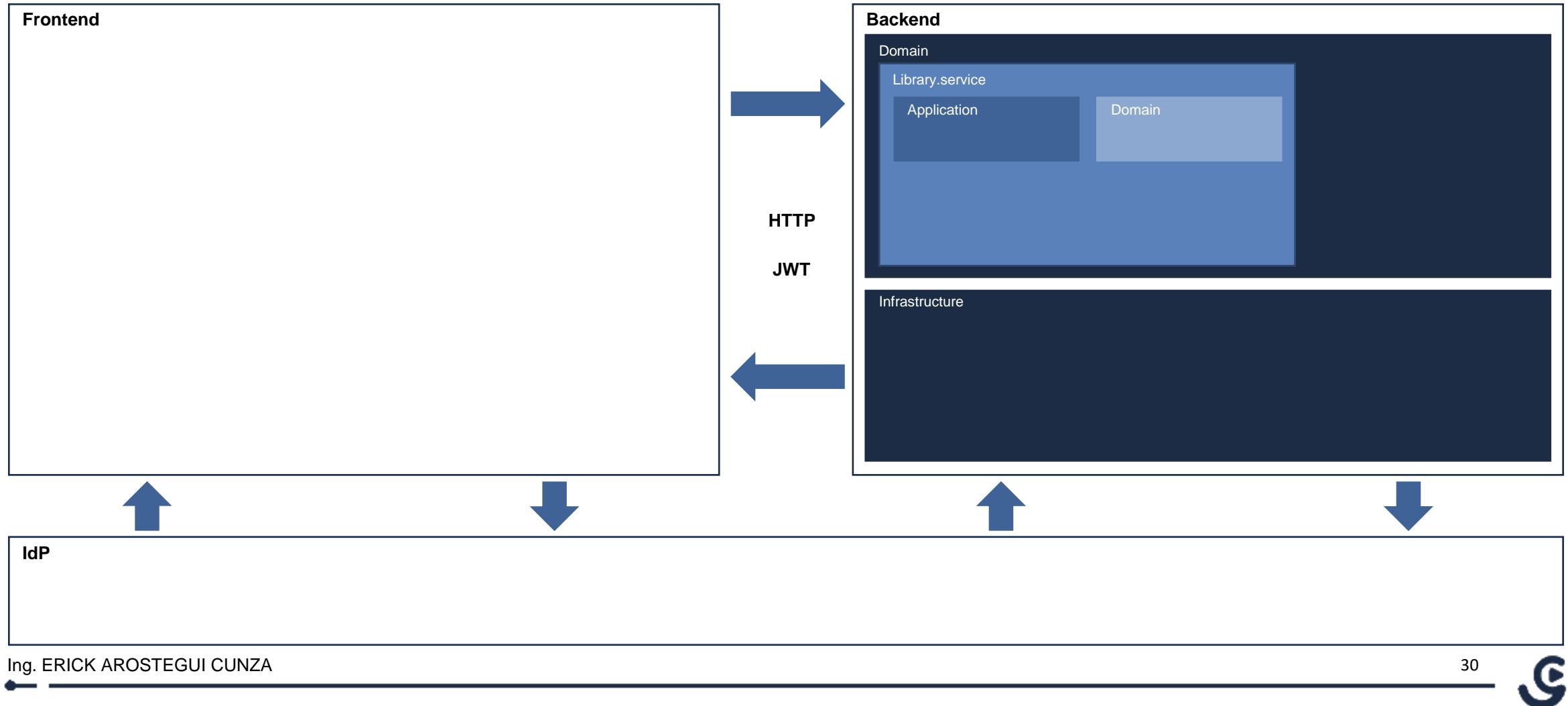
# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación



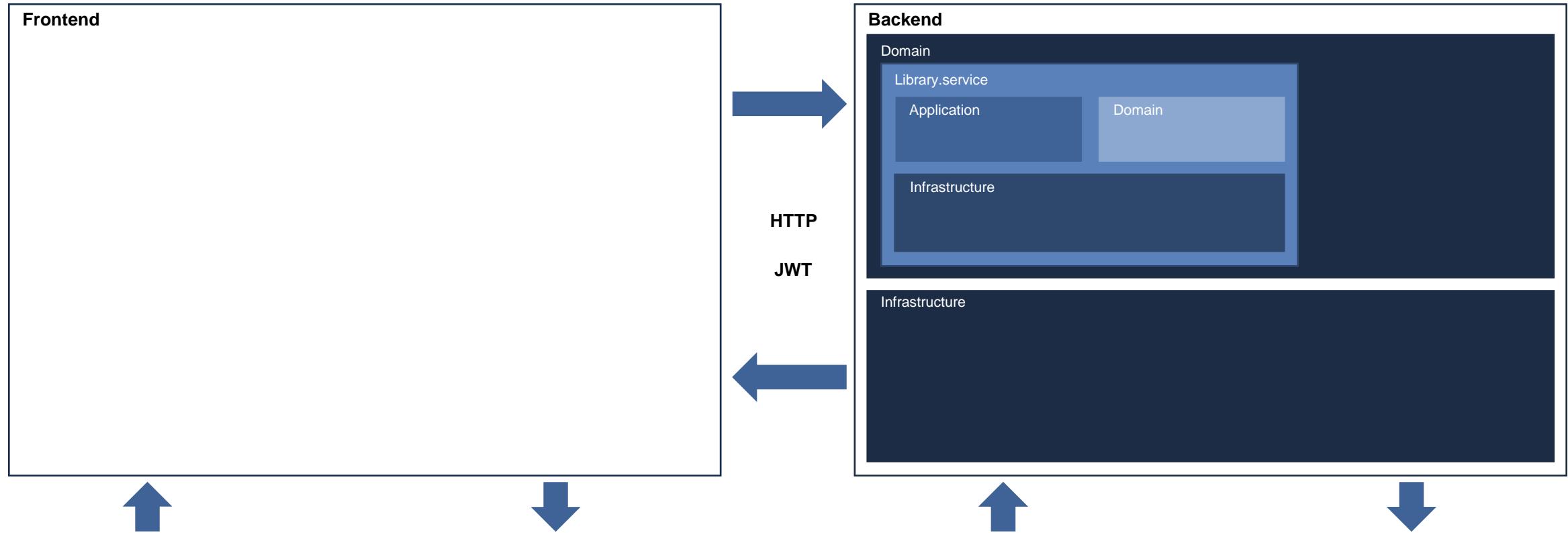
# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación



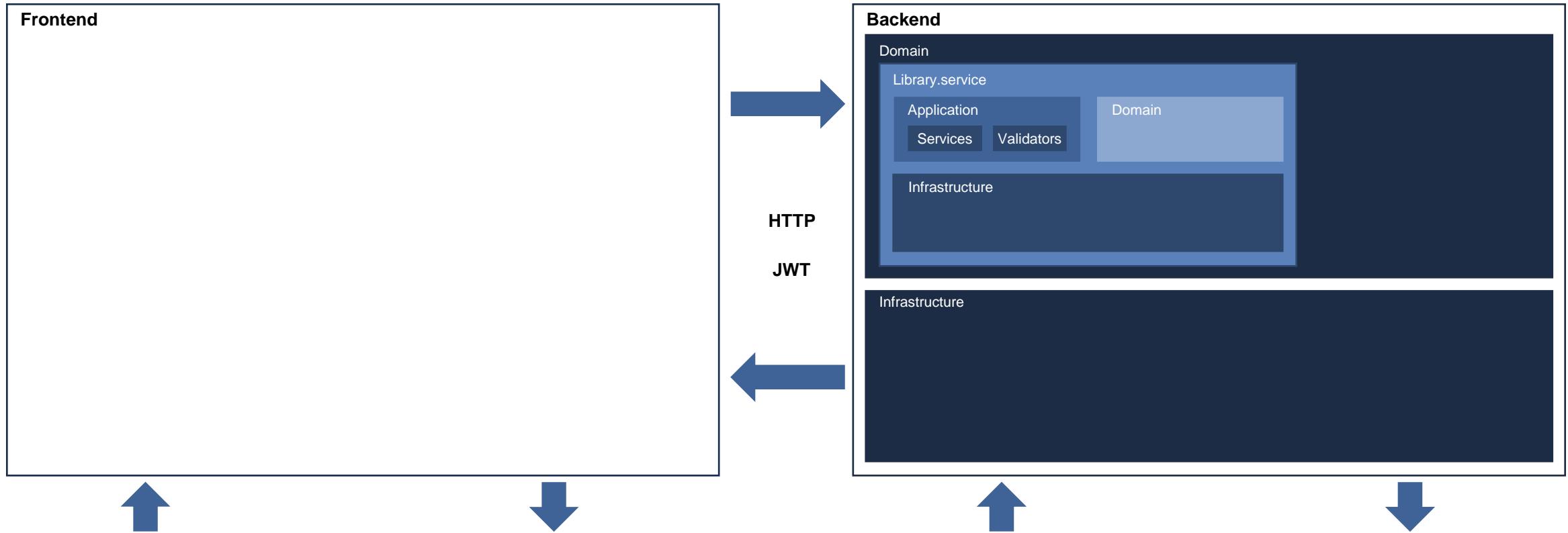
# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación

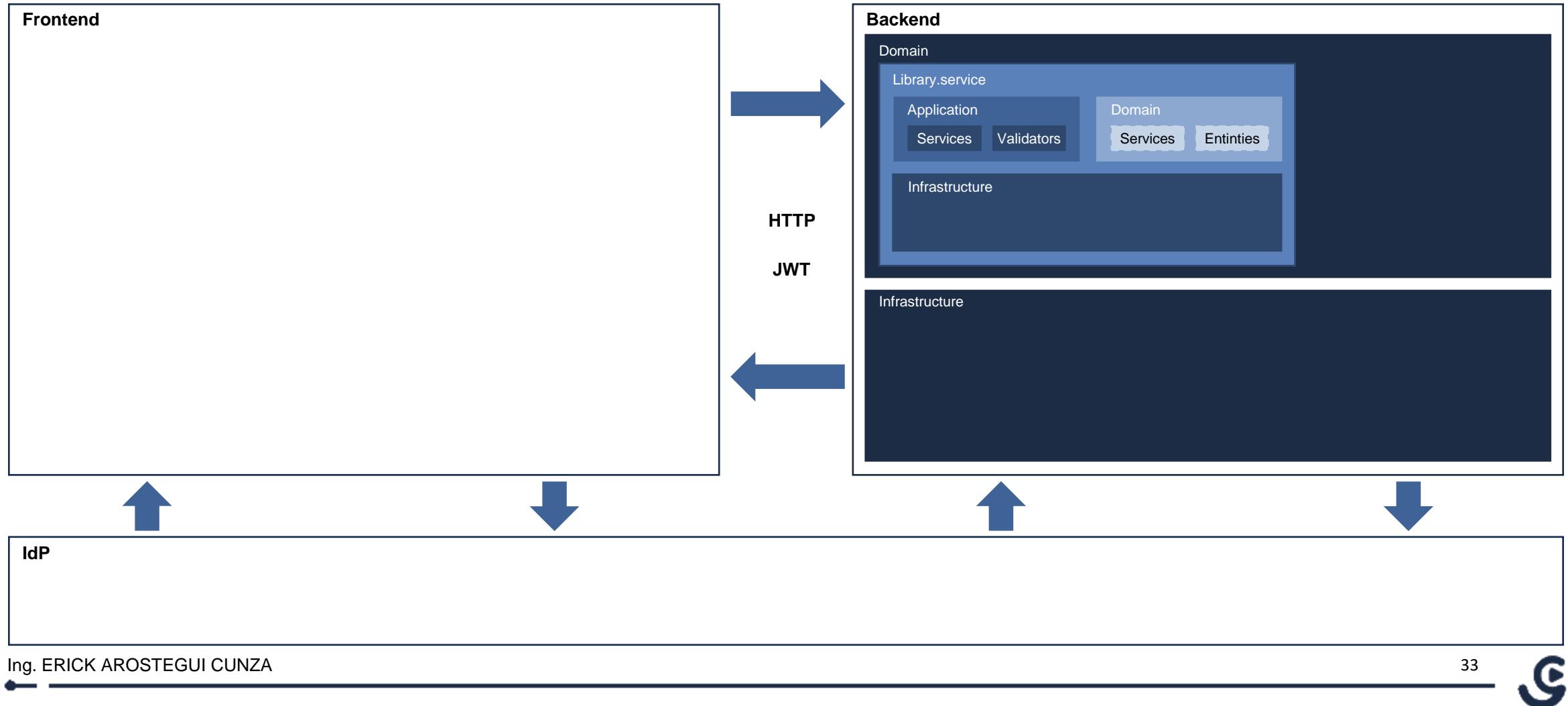


# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación

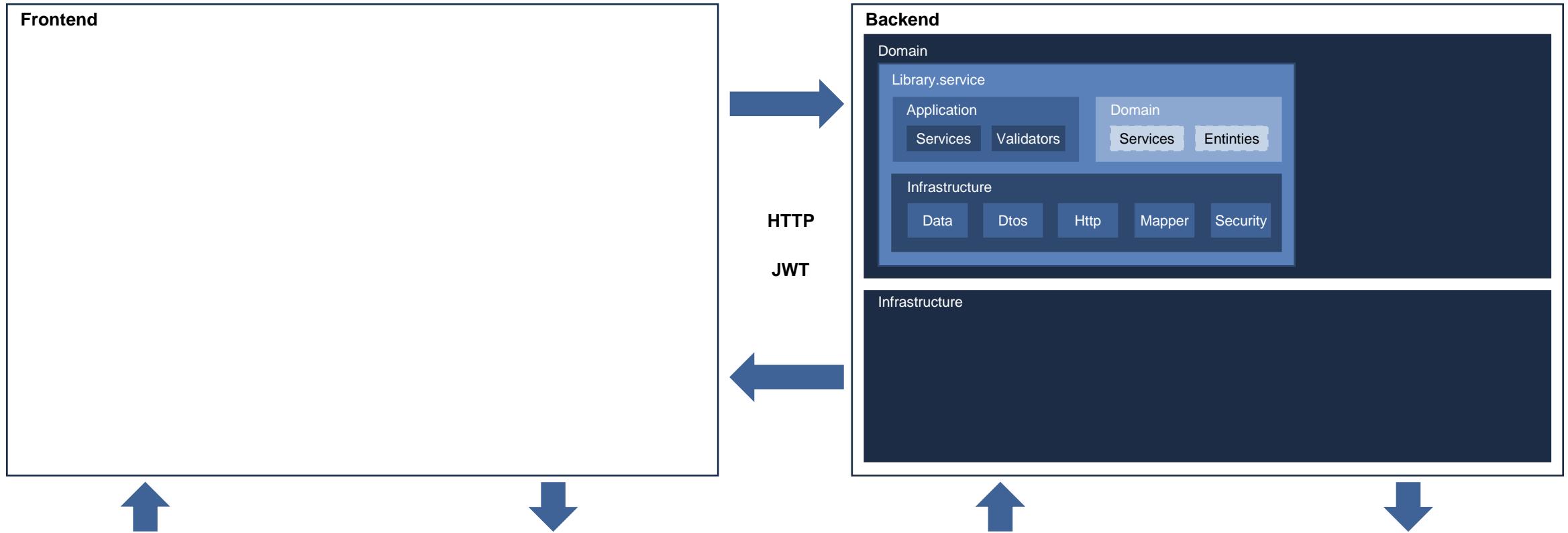


## Arquitectura de Aplicación



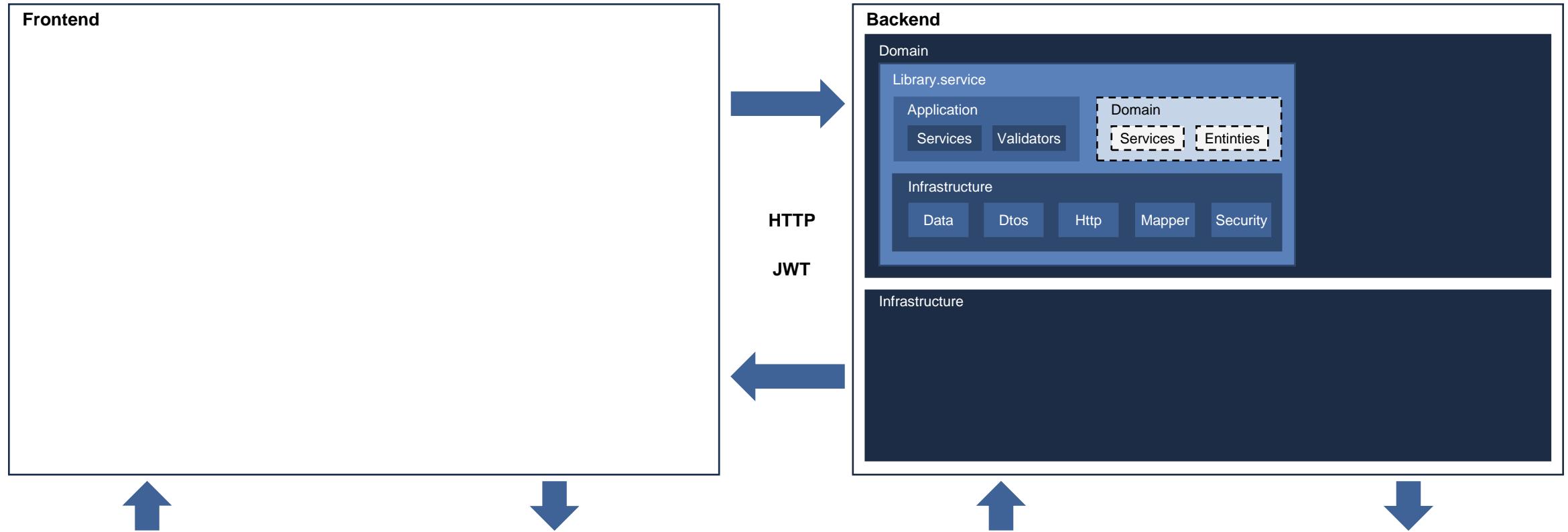
# → Diseñando y Modelando Minimal APIs

## Arquitectura de Aplicación

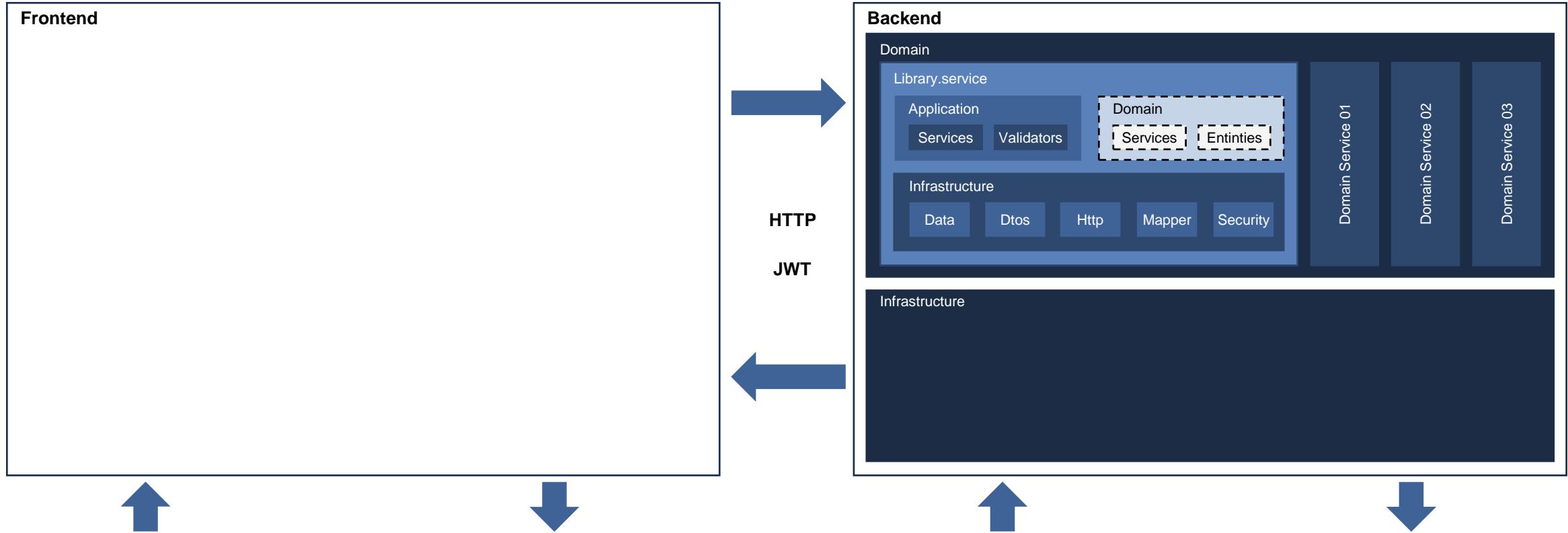


# → Diseñando y Modelando Minimal APIs

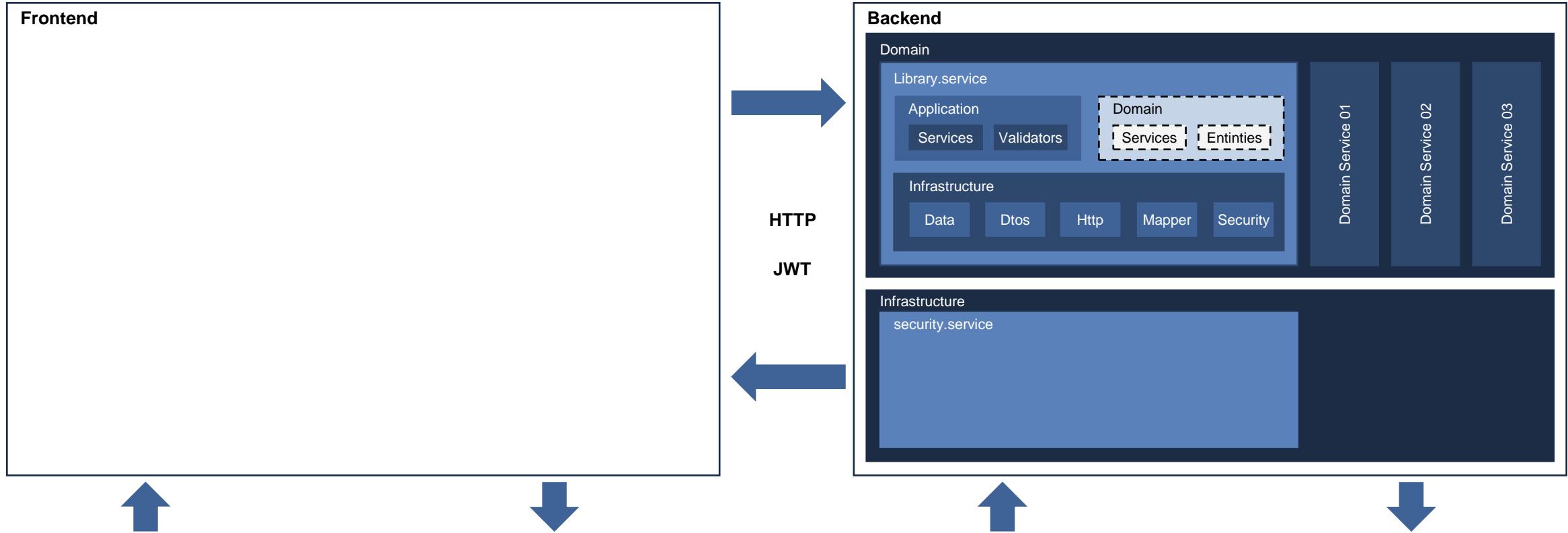
## Arquitectura de Aplicación



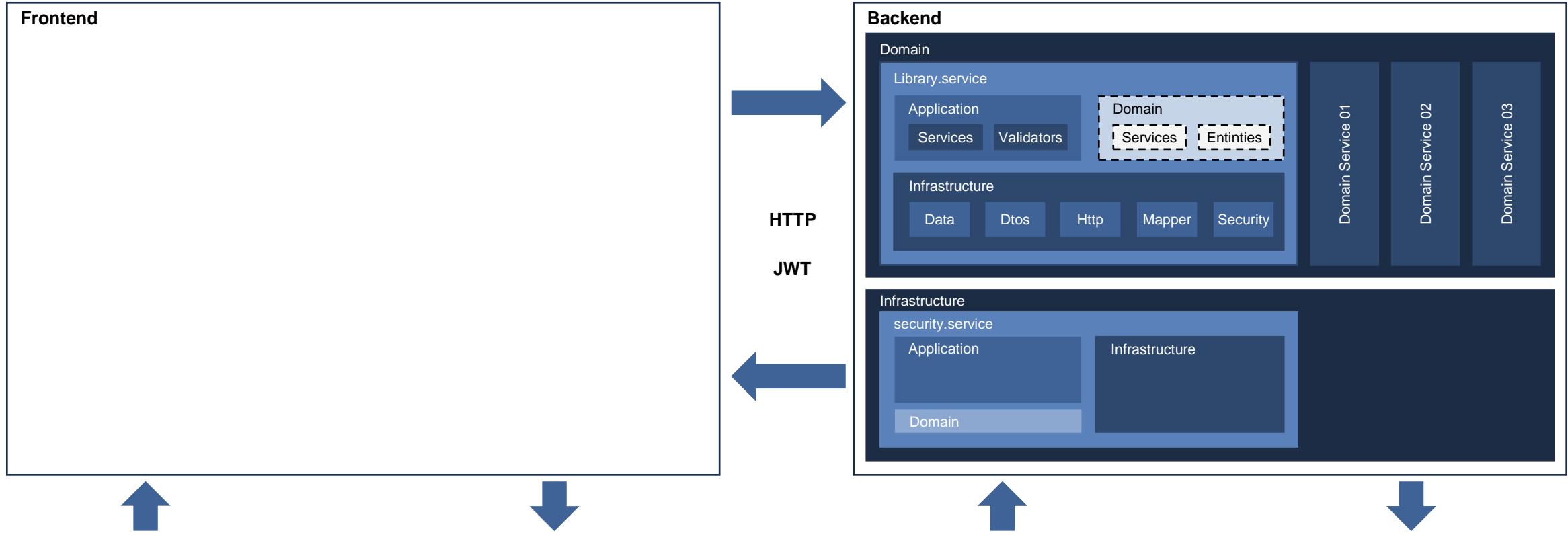
## Arquitectura de Aplicación



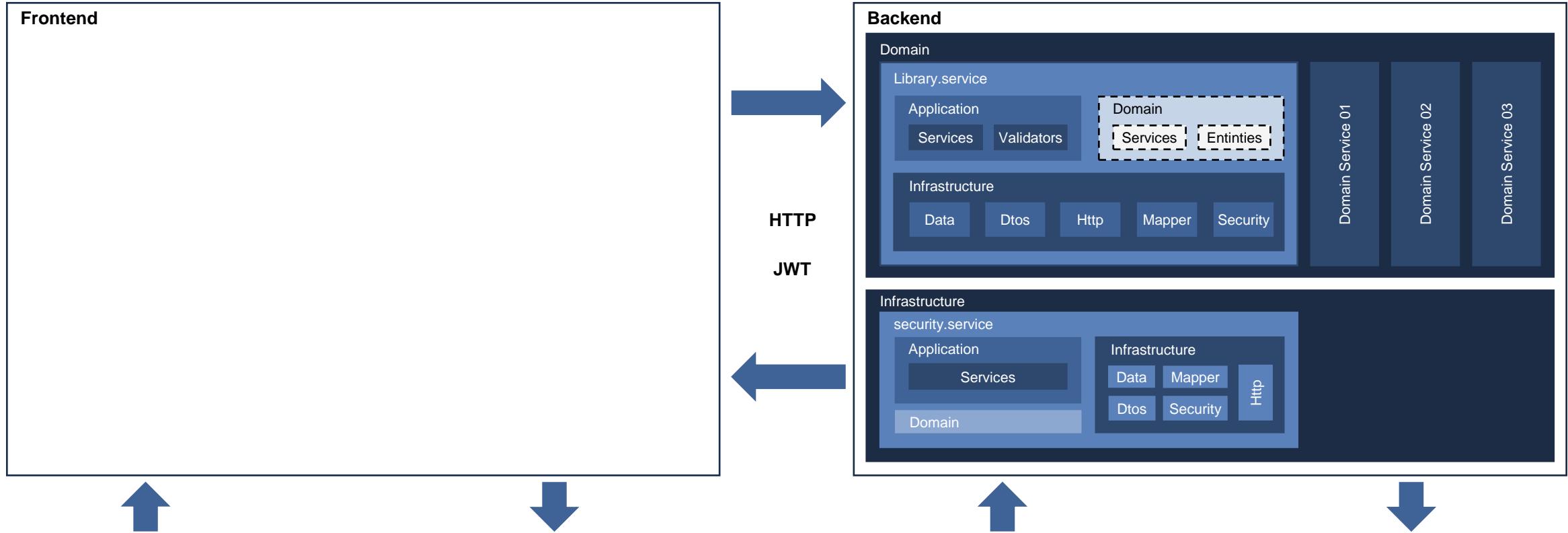
## Arquitectura de Aplicación



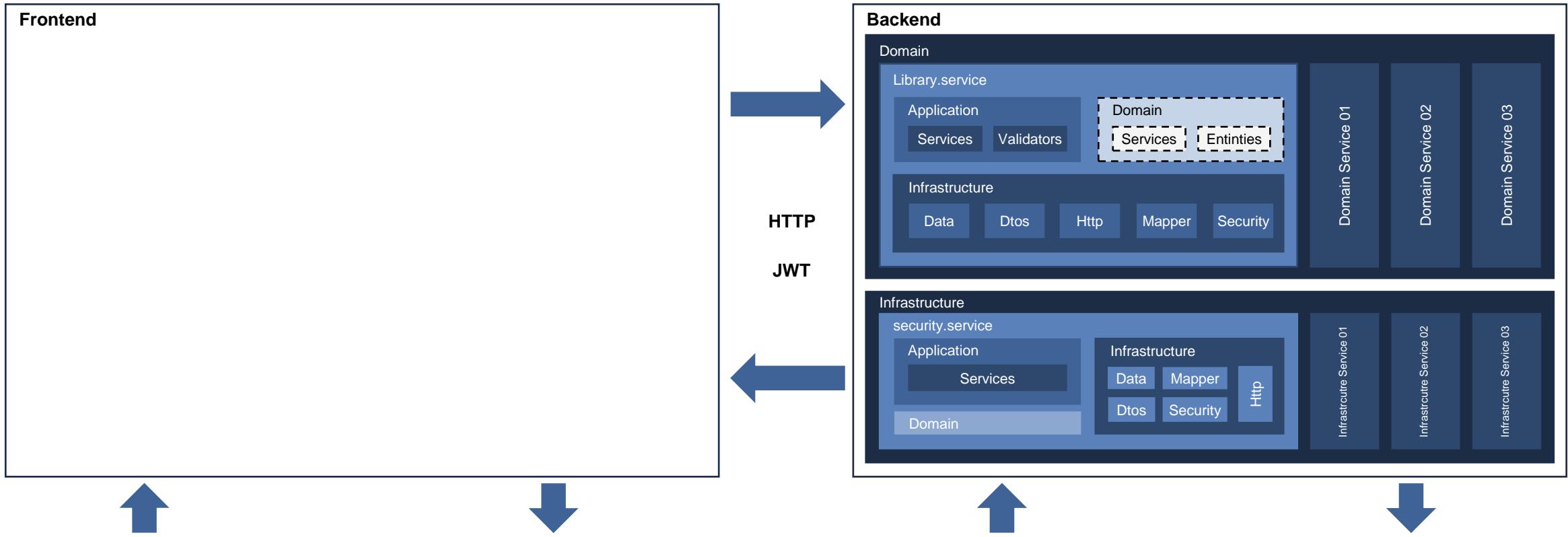
## Arquitectura de Aplicación



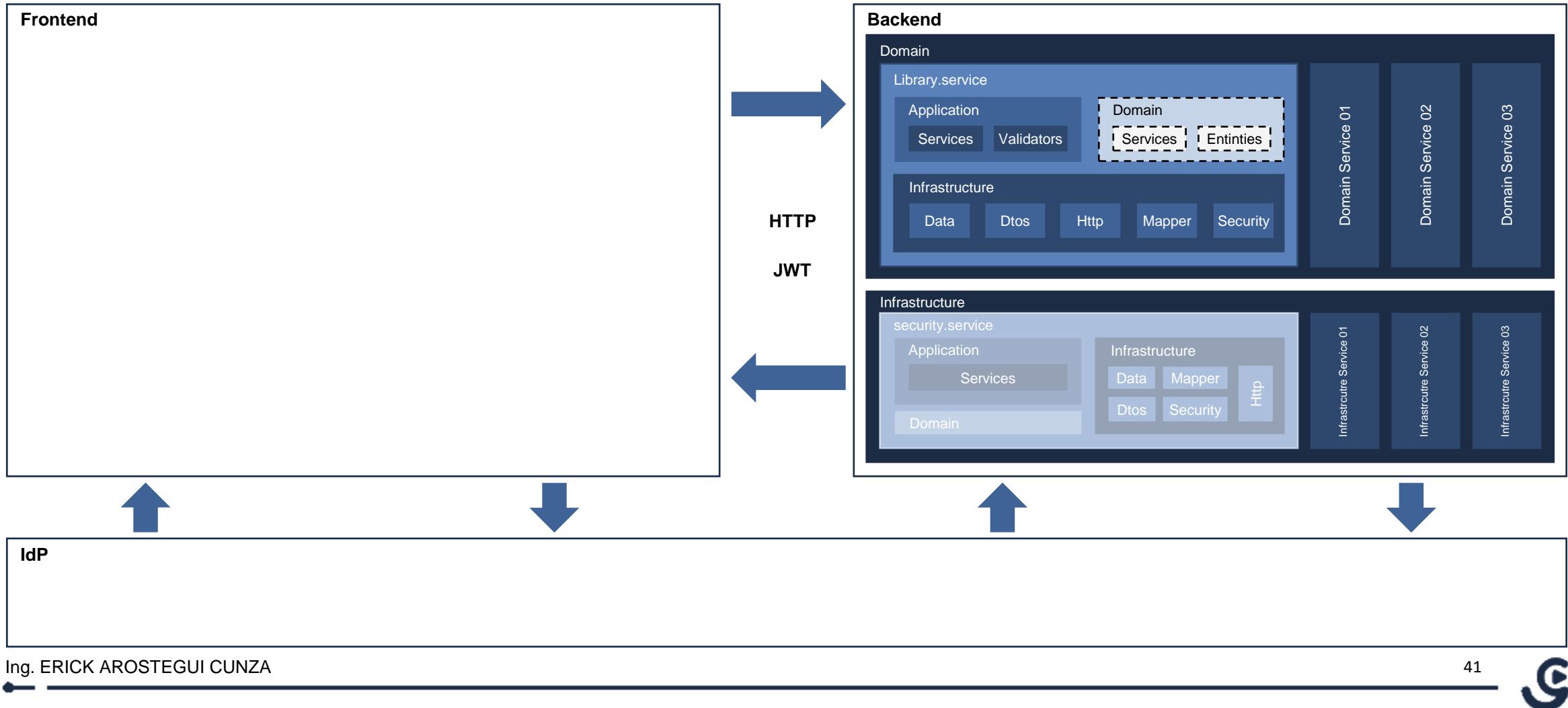
## Arquitectura de Aplicación



## Arquitectura de Aplicación



## Arquitectura de Aplicación



# 04



## Instalando y configurando entorno

# Instalando y configurando entorno

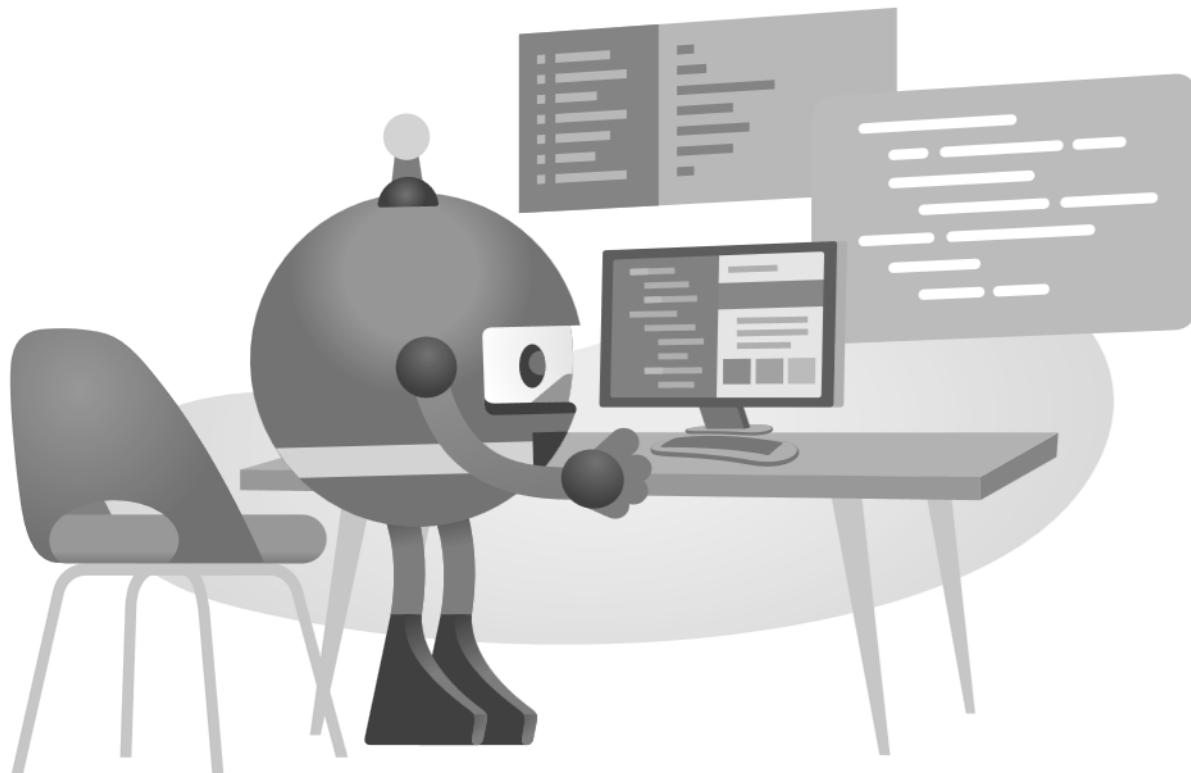


05



Creación de la solución y proyectos de  
la arquitectura propuesta  
(Template)

# Creación de la solución y proyectos de la arquitectura propuesta



A black and white photograph of a man in a suit and tie, sitting at a desk and working on a laptop. He is looking down at the screen with a thoughtful expression, his hand resting near his chin. A smartphone lies next to the laptop.

**GRACIAS  
POR SU PREFERENCIA**

