Capa de Servicio - Fundamentos

Pablo Sánchez

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria
Santander (Cantabria, España)
p.sanchez@unican.es

Advertencia

Todo el material contenido en este documento no constituye en modo alguno una obra de referencia o apuntes oficiales mediante el cual se puedan preparar las pruebas evaluables necesarias para superar la asignatura.

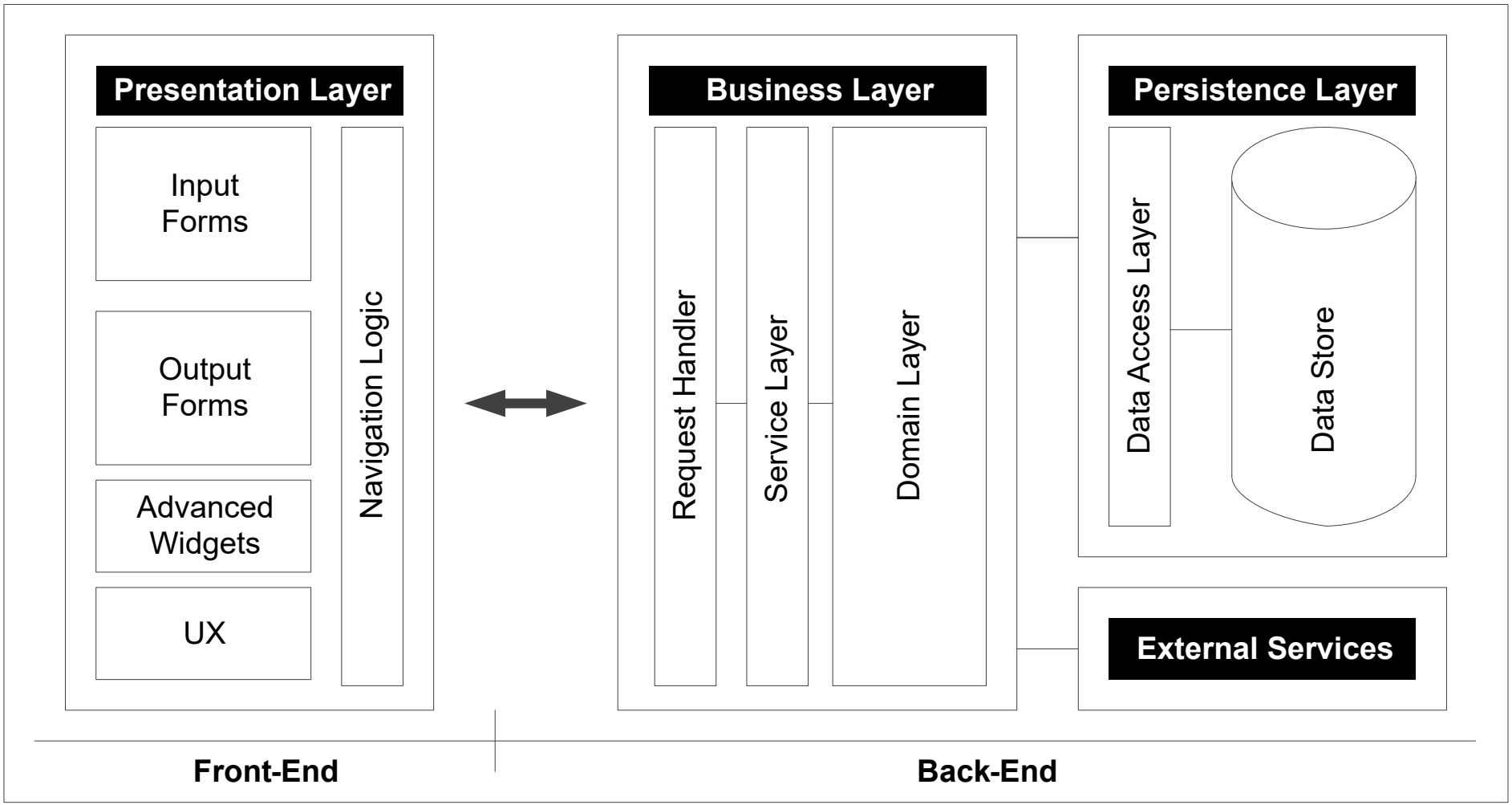
Este documento contiene exclusivamente una serie de diapositivas cuyo objetivo es servir de complemento visual a las actividades realizadas en el aula para la transmisión del contenido sobre el cual versarán las mencionadas pruebas evaluables.

Dicho de forma más clara, estas transparencias no son apuntes y su objetivo no es servir para que el alumno pueda preparar la asignatura.

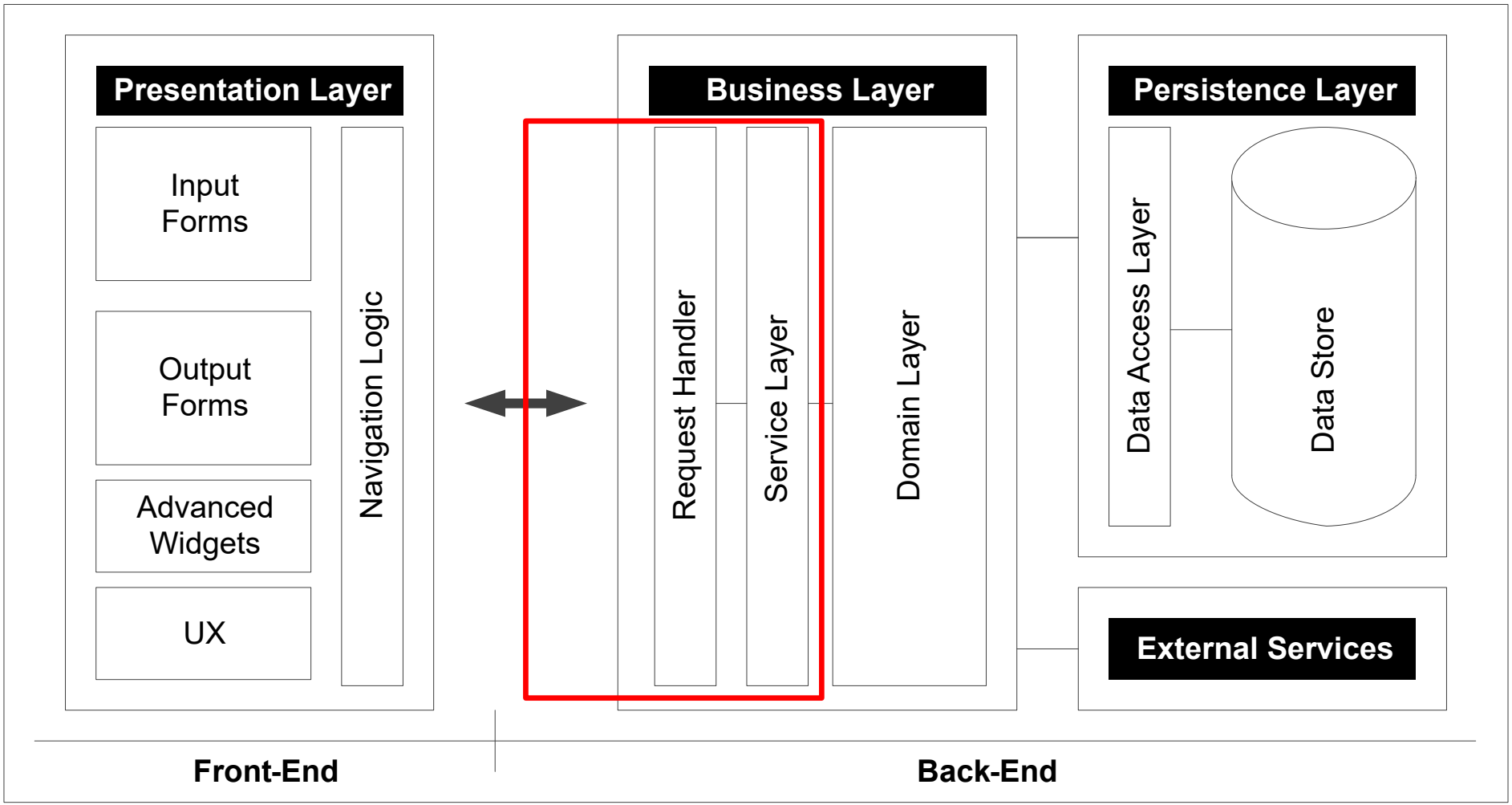
Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Capa de Servicio



Capa de Servicio



Objetivos del Tema

1. Comprender en profundidad cuáles son las responsabilidades de la capa de servicio.
2. Comprender cómo funciona el protocolo HTTP.
3. Comprender los principios de las arquitecturas REST.
4. Saber diseñar una API REST.
5. Ser capaz de usar los patrones de diseño propios del diseño de una capa de servicio.

Objetivos del Tema

1. Comprender en profundidad cuáles son las responsabilidades de la capa de servicio.
2. Comprender cómo funciona el protocolo HTTP.
3. Comprender los principios de las arquitecturas REST.
4. Saber diseñar una API REST.
5. Ser capaz de usar los patrones de diseño propios del diseño de una capa de servicio.

Objetivos del Tema

1. Comprender en profundidad cuáles son las responsabilidades de la capa de servicio.
2. Comprender cómo funciona el protocolo HTTP.
3. Comprender los principios de las arquitecturas REST.
4. Saber diseñar una API REST.
5. Ser capaz de usar los patrones de diseño propios del diseño de una capa de servicio.

Objetivos del Tema

1. Comprender en profundidad cuáles son las responsabilidades de la capa de servicio.
2. Comprender cómo funciona el protocolo HTTP.
3. Comprender los principios de las arquitecturas REST.
4. Saber diseñar una API REST.
5. Ser capaz de usar los patrones de diseño propios del diseño de una capa de servicio.

Objetivos del Tema

1. Comprender en profundidad cuáles son las responsabilidades de la capa de servicio.
2. Comprender cómo funciona el protocolo HTTP.
3. Comprender los principios de las arquitecturas REST.
4. Saber diseñar una API REST.
5. Ser capaz de usar los patrones de diseño propios del diseño de una capa de servicio.

Bibliografía

Fowler, M. (2002).
Patterns of Enterprise Application Architecture.
Addison-Wesley.

Esposito, D. y Saltarello, A. (2014).
Microsoft .NET - Architecting Applications for the Enterprise. 2ª Ed..
Microsoft Press

Richardson, L. and Ruby, S.
RESTful Web Services.
O'Reilly, 2007.

Responsabilidades de la Capa de Servicio

1. Atender las peticiones HTTP de los clientes, delegando en el modelo de dominio.
2. Asegurar la transaccionalidad de las operaciones de negocio.
3. Validar las peticiones de los clientes.
4. Recuperar y almacenar datos del almacén o almacenes persistentes.
5. Facilitar la eficiencia del sistema.
6. Controlar el acceso a los datos.
7. Gestionar la comunicación con los servicios externos.
8. Gestionar de manera adecuada casos excepcionales.
9. Ayudar a satisfacer los requisitos no funcionales.

Responsabilidades de la Capa de Servicio

1. Atender las peticiones HTTP de los clientes, delegando en el modelo de dominio.
2. Asegurar la transaccionalidad de las operaciones de negocio.
3. Validar las peticiones de los clientes.
4. Recuperar y almacenar datos del almacén o almacenes persistentes.
5. Facilitar la eficiencia del sistema.
6. Controlar el acceso a los datos.
7. Gestionar la comunicación con los servicios externos.
8. Gestionar de manera adecuada casos excepcionales.
9. Ayudar a satisfacer los requisitos no funcionales.

Responsabilidades de la Capa de Servicio

1. Atender las peticiones HTTP de los clientes, delegando en el modelo de dominio.
2. Asegurar la transaccionalidad de las operaciones de negocio.
3. Validar las peticiones de los clientes.
4. Recuperar y almacenar datos del almacén o almacenes persistentes.
5. Facilitar la eficiencia del sistema.
6. Controlar el acceso a los datos.
7. Gestionar la comunicación con los servicios externos.
8. Gestionar de manera adecuada casos excepcionales.
9. Ayudar a satisfacer los requisitos no funcionales.

Responsabilidades de la Capa de Servicio

1. Atender las peticiones HTTP de los clientes, delegando en el modelo de dominio.
2. Asegurar la transaccionalidad de las operaciones de negocio.
3. Validar las peticiones de los clientes.
4. Recuperar y almacenar datos del almacén o almacenes persistentes.
5. Facilitar la eficiencia del sistema.
6. Controlar el acceso a los datos.
7. Gestionar la comunicación con los servicios externos.
8. Gestionar de manera adecuada casos excepcionales.
9. Ayudar a satisfacer los requisitos no funcionales.

Responsabilidades de la Capa de Servicio

1. Atender las peticiones HTTP de los clientes, delegando en el modelo de dominio.
2. Asegurar la transaccionalidad de las operaciones de negocio.
3. Validar las peticiones de los clientes.
4. Recuperar y almacenar datos del almacén o almacenes persistentes.
5. Facilitar la eficiencia del sistema.
6. Controlar el acceso a los datos.
7. Gestionar la comunicación con los servicios externos.
8. Gestionar de manera adecuada casos excepcionales.
9. Ayudar a satisfacer los requisitos no funcionales.

Responsabilidades de la Capa de Servicio

1. Atender las peticiones HTTP de los clientes, delegando en el modelo de dominio.
2. Asegurar la transaccionalidad de las operaciones de negocio.
3. Validar las peticiones de los clientes.
4. Recuperar y almacenar datos del almacén o almacenes persistentes.
5. Facilitar la eficiencia del sistema.
6. Controlar el acceso a los datos.
7. Gestionar la comunicación con los servicios externos.
8. Gestionar de manera adecuada casos excepcionales.
9. Ayudar a satisfacer los requisitos no funcionales.

Responsabilidades de la Capa de Servicio

1. Atender las peticiones HTTP de los clientes, delegando en el modelo de dominio.
2. Asegurar la transaccionalidad de las operaciones de negocio.
3. Validar las peticiones de los clientes.
4. Recuperar y almacenar datos del almacén o almacenes persistentes.
5. Facilitar la eficiencia del sistema.
6. Controlar el acceso a los datos.
7. Gestionar la comunicación con los servicios externos.
8. Gestionar de manera adecuada casos excepcionales.
9. Ayudar a satisfacer los requisitos no funcionales.

Responsabilidades de la Capa de Servicio

1. Atender las peticiones HTTP de los clientes, delegando en el modelo de dominio.
2. Asegurar la transaccionalidad de las operaciones de negocio.
3. Validar las peticiones de los clientes.
4. Recuperar y almacenar datos del almacén o almacenes persistentes.
5. Facilitar la eficiencia del sistema.
6. Controlar el acceso a los datos.
7. Gestionar la comunicación con los servicios externos.
8. Gestionar de manera adecuada casos excepcionales.
9. Ayudar a satisfacer los requisitos no funcionales.

Responsabilidades de la Capa de Servicio

1. Atender las peticiones HTTP de los clientes, delegando en el modelo de dominio.
2. Asegurar la transaccionalidad de las operaciones de negocio.
3. Validar las peticiones de los clientes.
4. Recuperar y almacenar datos del almacén o almacenes persistentes.
5. Facilitar la eficiencia del sistema.
6. Controlar el acceso a los datos.
7. Gestionar la comunicación con los servicios externos.
8. Gestionar de manera adecuada casos excepcionales.
9. Ayudar a satisfacer los requisitos no funcionales.

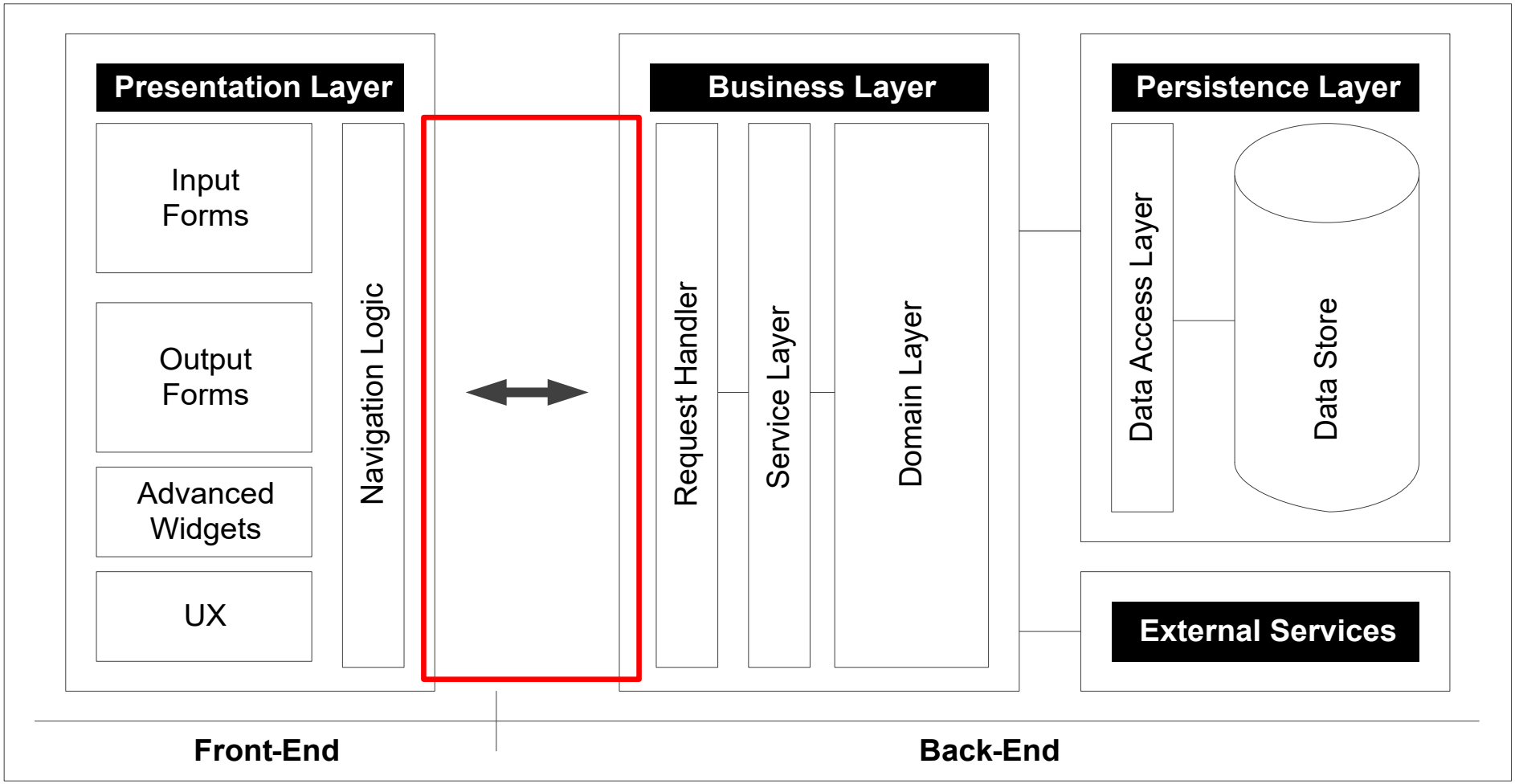
Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Índice

1. Introducción
2. Arquitecturas REST
   - Arquitecturas REST
   - Recursos REST
   - Componentes y Conectores REST
   - HATEOAS
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Arquitecturas REST



Arquitectura REST

REST *(REpresentational State Transfer)*

1. REST es un estilo arquitectónico diseñado para satisfacer las necesidades de los sistemas de hipermedia a escala de internet.
2. Un *sistema hipermedia* se concibe como una máquina de estados donde el usuario progresa mediante la selección de enlaces y el envío de formularios.
3. La representación de cada estado se transfiere al usuario.

Sistema Hipermedia

Un *sistema hipermedia* es un sistema que fusiona información de control de la aplicación con la presentación de su información. La información se puede presentar utilizando diversos formatos, tales como texto, gráficos, audio o vídeo.

Arquitectura REST

REST *(REpresentational State Transfer)*

1. REST es un estilo arquitectónico diseñado para satisfacer las necesidades de los sistemas de hipermedia a escala de internet.
2. Un *sistema hipermedia* se concibe como una máquina de estados donde el usuario progresa mediante la selección de enlaces y el envío de formularios.
3. La representación de cada estado se transfiere al usuario.

Sistema Hipermedia

Un *sistema hipermedia* es un sistema que fusiona información de control de la aplicación con la presentación de su información. La información se puede presentar utilizando diversos formatos, tales como texto, gráficos, audio o vídeo.

Arquitectura REST

REST *(REpresentational State Transfer)*

1. REST es un estilo arquitectónico diseñado para satisfacer las necesidades de los sistemas de hipermedia a escala de internet.
2. Un *sistema hipermedia* se concibe como una máquina de estados donde el usuario progresa mediante la selección de enlaces y el envío de formularios.
3. La representación de cada estado se transfiere al usuario.

Sistema Hipermedia

Un *sistema hipermedia* es un sistema que fusiona información de control de la aplicación con la presentación de su información. La información se puede presentar utilizando diversos formatos, tales como texto, gráficos, audio o vídeo.

Arquitectura REST

REST *(REpresentational State Transfer)*

1. REST es un estilo arquitectónico diseñado para satisfacer las necesidades de los sistemas de hipermedia a escala de internet.
2. Un *sistema hipermedia* se concibe como una máquina de estados donde el usuario progresa mediante la selección de enlaces y el envío de formularios.
3. La representación de cada estado se transfiere al usuario.

Sistema Hipermedia

Un *sistema hipermedia* es un sistema que fusiona información de control de la aplicación con la presentación de su información. La información se puede presentar utilizando diversos formatos, tales como texto, gráficos, audio o vídeo.

Objetivos Arquitectura REST

1. Minimizar latencias.
2. Minimizar comunicaciones a nivel de red.
3. Maximizar la escalabilidad de los componentes.
4. Maximizar la independencia de los componentes.
5. Permitir almacenar en cachés resultados e interacciones.
6. Permitir la sustitución dinámica de componentes.
7. Permitir el procesamiento de acciones por intermediarios.

Objetivos Arquitectura REST

1. Minimizar latencias.
2. Minimizar comunicaciones a nivel de red.
3. Maximizar la escalabilidad de los componentes.
4. Maximizar la independencia de los componentes.
5. Permitir almacenar en cachés resultados e interacciones.
6. Permitir la sustitución dinámica de componentes.
7. Permitir el procesamiento de acciones por intermediarios.

Objetivos Arquitectura REST

1. Minimizar latencias.
2. Minimizar comunicaciones a nivel de red.
3. Maximizar la escalabilidad de los componentes.
4. Maximizar la independencia de los componentes.
5. Permitir almacenar en cachés resultados e interacciones.
6. Permitir la sustitución dinámica de componentes.
7. Permitir el procesamiento de acciones por intermediarios.

Objetivos Arquitectura REST

1. Minimizar latencias.
2. Minimizar comunicaciones a nivel de red.
3. Maximizar la escalabilidad de los componentes.
4. Maximizar la independencia de los componentes.
5. Permitir almacenar en cachés resultados e interacciones.
6. Permitir la sustitución dinámica de componentes.
7. Permitir el procesamiento de acciones por intermediarios.

Objetivos Arquitectura REST

1. Minimizar latencias.
2. Minimizar comunicaciones a nivel de red.
3. Maximizar la escalabilidad de los componentes.
4. Maximizar la independencia de los componentes.
5. Permitir almacenar en cachés resultados e interacciones.
6. Permitir la sustitución dinámica de componentes.
7. Permitir el procesamiento de acciones por intermediarios.

Objetivos Arquitectura REST

1. Minimizar latencias.
2. Minimizar comunicaciones a nivel de red.
3. Maximizar la escalabilidad de los componentes.
4. Maximizar la independencia de los componentes.
5. Permitir almacenar en cachés resultados e interacciones.
6. Permitir la sustitución dinámica de componentes.
7. Permitir el procesamiento de acciones por intermediarios.

Objetivos Arquitectura REST

1. Minimizar latencias.
2. Minimizar comunicaciones a nivel de red.
3. Maximizar la escalabilidad de los componentes.
4. Maximizar la independencia de los componentes.
5. Permitir almacenar en cachés resultados e interacciones.
6. Permitir la sustitución dinámica de componentes.
7. Permitir el procesamiento de acciones por intermediarios.

Estilos Arquitectura REST

1. Cliente-Servidor.
2. Sin estado.
3. Respuestas cacheables.
4. Interfaces uniformes entre componentes.
5. Código bajo demanda (opcional).

Estilos Arquitectura REST

1. Cliente-Servidor.
2. Sin estado.
3. Respuestas cacheables.
4. Interfaces uniformes entre componentes.
5. Código bajo demanda (opcional).

Estilos Arquitectura REST

1. Cliente-Servidor.
2. Sin estado.
3. Respuestas cacheables.
4. Interfaces uniformes entre componentes.
5. Código bajo demanda (opcional).

Estilos Arquitectura REST

1. Cliente-Servidor.
2. Sin estado.
3. Respuestas cacheables.
4. Interfaces uniformes entre componentes.
5. Código bajo demanda (opcional).

Estilos Arquitectura REST

1. Cliente-Servidor.
2. Sin estado.
3. Respuestas cacheables.
4. Interfaces uniformes entre componentes.
5. Código bajo demanda (opcional).

Índice

1. Introducción
2. Arquitecturas REST
   - Arquitecturas REST
   - Recursos REST
   - Componentes y Conectores REST
   - HATEOAS
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Recursos REST

Recurso REST

Un *recurso REST* es cualquier pieza de información a la que sea posible darle un nombre, que será el *identificador del recurso*.

Representación de un Recurso REST

Una *representación de recurso* es un conjunto de información que contiene datos, metadatos y, opcionalmente, metadatos sobre los metadatos. La representación de un recurso REST puede ser variable.

Recursos REST

Recurso REST

Un *recurso REST* es cualquier pieza de información a la que sea posible darle un nombre, que será el *identificador del recurso*.

Representación de un Recurso REST

Una *representación de recurso* es un conjunto de información que contiene datos, metadatos y, opcionalmente, metadatos sobre los metadatos. La representación de un recurso REST puede ser variable.

Tipos de Recursos REST

Documentos Instancias individuales de un concepto (e.g., un viaje concreto).

Colecciones Repositorios de recursos manejados por el servidor (e.g., todos los viajes).

Store Repositorio de recursos manejado por el cliente.

Controller Abstrae una función o procedimiento.

Tipos de Recursos REST

Documentos Instancias individuales de un concepto (e.g., un viaje concreto).

Colecciones Repositorios de recursos manejados por el servidor (e.g., todos los viajes).

Store Repositorio de recursos manejado por el cliente.

Controller Abstrae una función o procedimiento.

Tipos de Recursos REST

Documentos Instancias individuales de un concepto (e.g., un viaje concreto).

Colecciones Repositorios de recursos manejados por el servidor (e.g., todos los viajes).

Store Repositorio de recursos manejado por el cliente.

Controller Abstrae una función o procedimiento.

Tipos de Recursos REST

Documentos Instancias individuales de un concepto (e.g., un viaje concreto).

Colecciones Repositorios de recursos manejados por el servidor (e.g., todos los viajes).

Store Repositorio de recursos manejado por el cliente.

Controller Abstrae una función o procedimiento.

Índice

1. Introducción
2. Arquitecturas REST
   - Arquitecturas REST
   - Recursos REST
   - Componentes y Conectores REST
   - HATEOAS
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Componentes REST

1. User agent
2. Origin-server
3. Proxy
4. Gateway

Componentes REST

1. User agent
2. Origin-server
3. Proxy
4. Gateway

Componentes REST

1. User agent
2. Origin-server
3. Proxy
4. Gateway

Componentes REST

1. User agent
2. Origin-server
3. Proxy
4. Gateway

Conectores REST

1. Client connector.
2. Server connector.
3. Cache.
4. Resolver
5. Tunnel

Conectores REST

1. Client connector.
2. Server connector.
3. Cache.
4. Resolver
5. Tunnel

Conectores REST

1. Client connector.
2. Server connector.
3. Cache.
4. Resolver
5. Tunnel

Conectores REST

1. Client connector.
2. Server connector.
3. Cache.
4. Resolver
5. Tunnel

Conectores REST

1. Client connector.
2. Server connector.
3. Cache.
4. Resolver
5. Tunnel

Índice

1. Introducción
2. Arquitecturas REST
   - Arquitecturas REST
   - Recursos REST
   - Componentes y Conectores REST
   - HATEOAS
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Niveles de Adopción REST

HATEOAS

HATEOAS (Hypermedia As The Engine Of Application State) es una técnica utilizada por sistemas hipermedias en la que los clientes reciben como parte de la respuesta enlaces a recursos que le permiten ejecutar diversas acciones sobre las mismas.

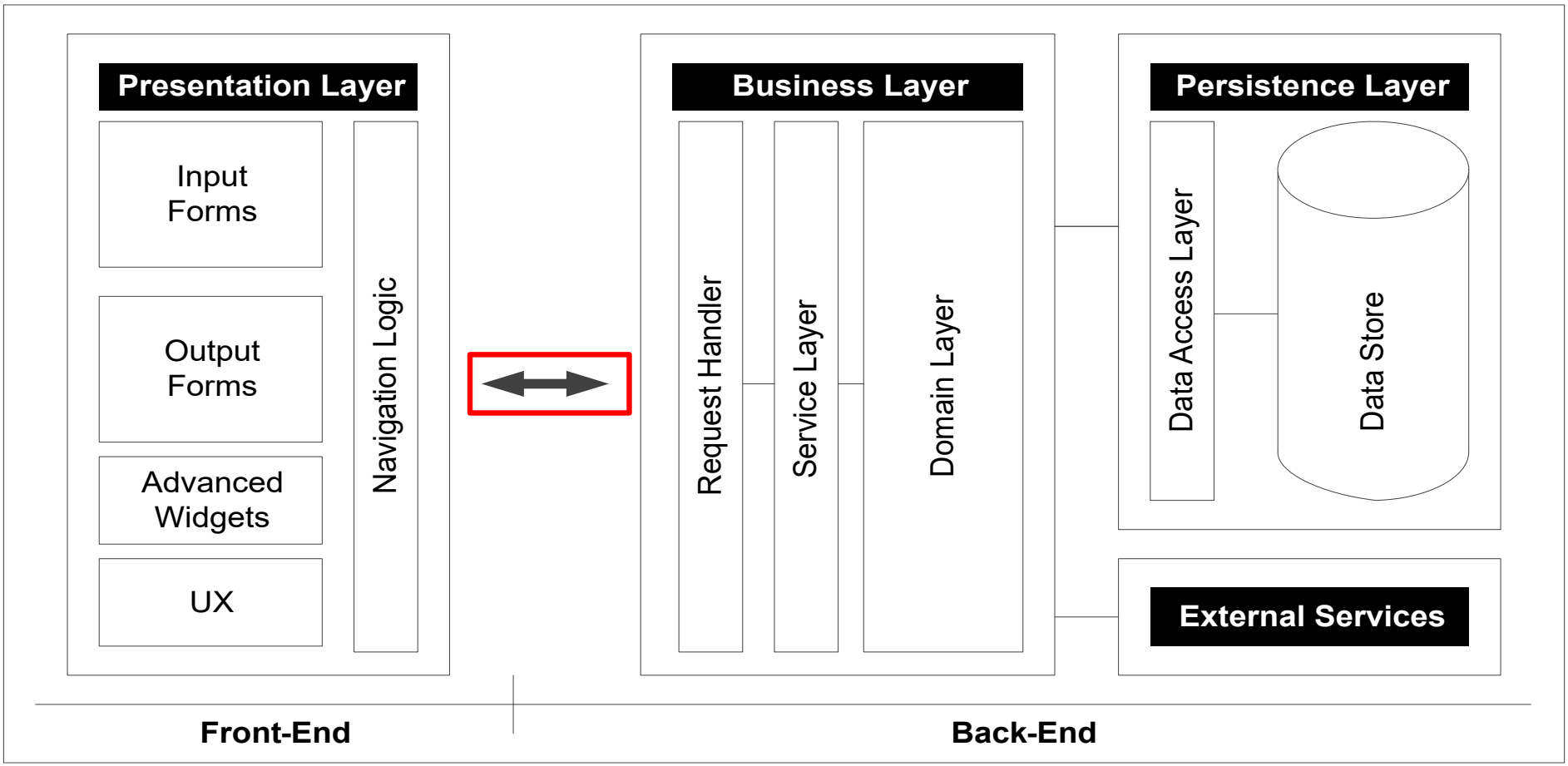
Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
   - Definición
   - Paquetes HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Protocolo HTTP



Protocolo HTTP

Protocolo HTTP

HTTP (Hypertext Transfer Protocol) es un protocolo a nivel de aplicación diseñado para su utilización en *sistemas de información hipermedia* distribuidos y colaborativos.

Características HTTP

1. Modelo cliente-servidor, petición-respuesta.
2. Los mensajes HTTP contienen códigos de control, metadatos y datos.
3. Facilita la creación de cachés intermedias.
4. Identifica recursos mediante URLs
5. Soporta varios mecanismos de autenticación.

Características HTTP

1. Modelo cliente-servidor, petición-respuesta.
2. Los mensajes HTTP contienen códigos de control, metadatos y datos.
3. Facilita la creación de cachés intermedias.
4. Identifica recursos mediante URLs
5. Soporta varios mecanismos de autenticación.

Características HTTP

1. Modelo cliente-servidor, petición-respuesta.
2. Los mensajes HTTP contienen códigos de control, metadatos y datos.
3. Facilita la creación de cachés intermedias.
4. Identifica recursos mediante URLs
5. Soporta varios mecanismos de autenticación.

Características HTTP

1. Modelo cliente-servidor, petición-respuesta.
2. Los mensajes HTTP contienen códigos de control, metadatos y datos.
3. Facilita la creación de cachés intermedias.
4. Identifica recursos mediante URLs
5. Soporta varios mecanismos de autenticación.

Características HTTP

1. Modelo cliente-servidor, petición-respuesta.
2. Los mensajes HTTP contienen códigos de control, metadatos y datos.
3. Facilita la creación de cachés intermedias.
4. Identifica recursos mediante URLs
5. Soporta varios mecanismos de autenticación.

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
   - Definición
   - Paquetes HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

HTTP Request

```
POST /usuarios HTTP/1.1                          | Línea inicial
Host: localhost:8080                             | Cabeceras
Content-Type: application/json                   |
cache-control: no-cache                          |
{                                                | Cuerpo
   "username": "TaxiDriver",                     |
   "nombre": "Travis",                           |
   "apellido": "Bickle",                         |
   "email": "travisBickle@nyc-taxi.us"           |
}                                                |
```

HTTP Response

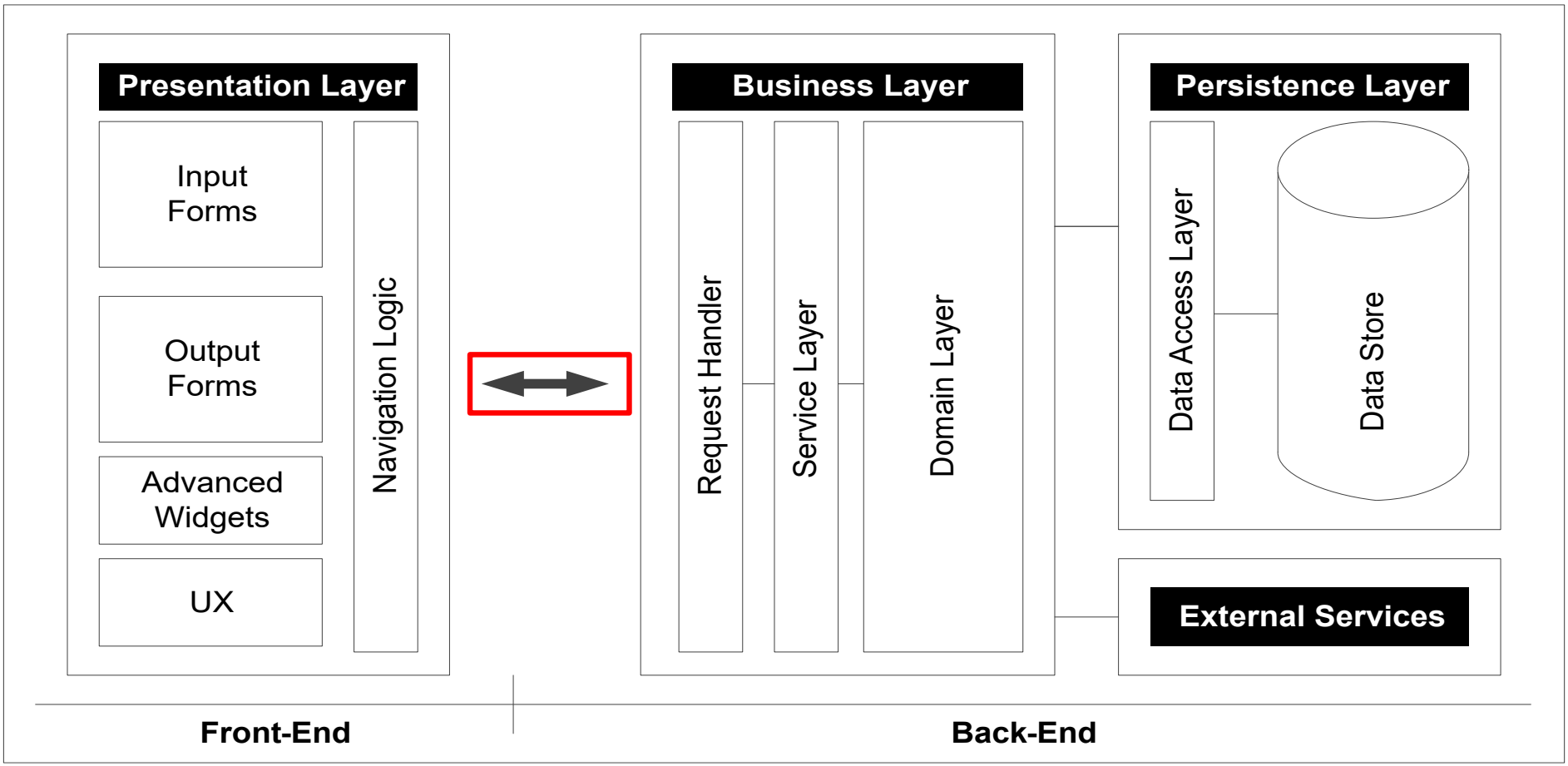
HTTP/1.1 200	Línea inicial
status: 200 Content-Type: application/json;charset=UTF-8 Transfer-Encoding: chunked Date: Thu, 21 Mar 2019 10:49:41 GMT	Cabeceras

<pre>{
"username": "TaxiDriver",
"nombre": "Travis",
"apellido": "Bickle",
"fechaAlta": "2019-03-21T10:47:58.367+0000",
"email": "travisBickle@nyc-taxi.us",
"telefonos": [],
"plazasSolicitadas": [],
"viajesAceptados": []
}</pre>
	Cuerpo

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Formato de intercambio JSON



Formato de intercambio JSON

JSON

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos fácilmente comprensible por humanos y fácilmente procesable por máquinas.

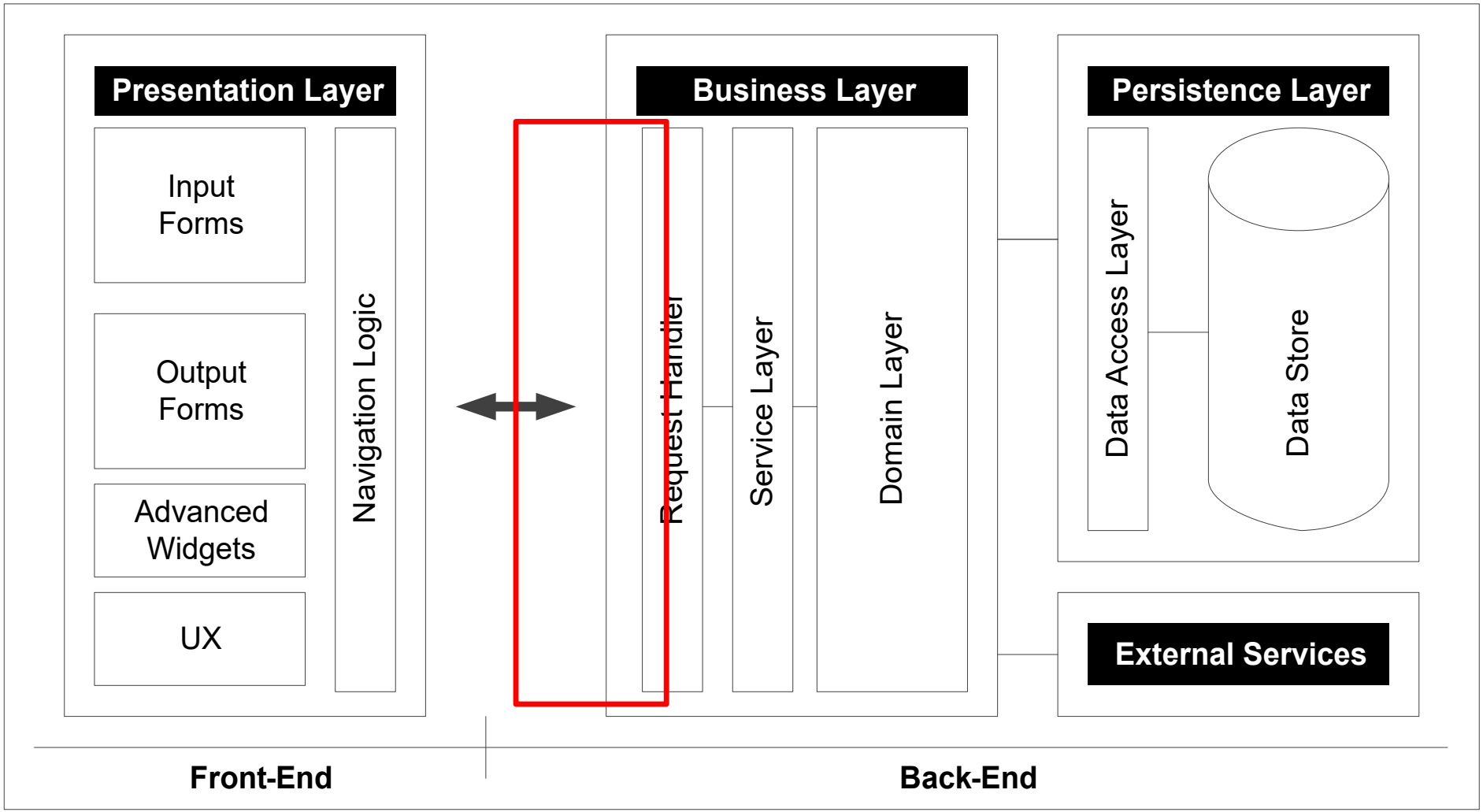
Gramática JSON

```
<json> ::= <object> | <array>
<object> ::= {} |
             {(<string>:<value>,)* <string>:<value>}
<value>  ::= <string> | <number> | <object> |
             array | true | false | null
```

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

API REST


Presentation Layer
Input Forms
Output Forms
Advanced Widgets
UX
Navigation Logic
Business Layer
Request Handler
Service Layer
Domain Layer
Persistence Layer
Data Access Layer
Data Store
External Services
Front-End
Back-End

Responsabilidades del Controlador HTTP

1. Atender las peticiones HTTP de los clientes.
2. Convertir datos de HTTP al lenguaje que corresponda (*unmarshalling*).
3. Validar las peticiones de los clientes.
4. Convertir las respuestas a una respuesta HTTP adecuada.
5. Convertir los resultados a datos HTTP (*marshalling*).

Responsabilidades del Controlador HTTP

1. Atender las peticiones HTTP de los clientes.
2. Convertir datos de HTTP al lenguaje que corresponda (*unmarshalling*).
3. Validar las peticiones de los clientes.
4. Convertir las respuestas a una respuesta HTTP adecuada.
5. Convertir los resultados a datos HTTP (*marshalling*).

Responsabilidades del Controlador HTTP

1. Atender las peticiones HTTP de los clientes.
2. Convertir datos de HTTP al lenguaje que corresponda (*unmarshalling*).
3. Validar las peticiones de los clientes.
4. Convertir las respuestas a una respuesta HTTP adecuada.
5. Convertir los resultados a datos HTTP (*marshalling*).

Responsabilidades del Controlador HTTP

1. Atender las peticiones HTTP de los clientes.
2. Convertir datos de HTTP al lenguaje que corresponda (*unmarshalling*).
3. Validar las peticiones de los clientes.
4. Convertir las respuestas a una respuesta HTTP adecuada.
5. Convertir los resultados a datos HTTP (*marshalling*).

Responsabilidades del Controlador HTTP

1. Atender las peticiones HTTP de los clientes.
2. Convertir datos de HTTP al lenguaje que corresponda (*unmarshalling*).
3. Validar las peticiones de los clientes.
4. Convertir las respuestas a una respuesta HTTP adecuada.
5. Convertir los resultados a datos HTTP (*marshalling*).

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
   - Identificación de recursos
   - Verbos HTTP y Recursos REST
   - Códigos de Respuestas HTTP
   - HATEOAS
   - Niveles de Adopción
6. Patrones de la Capa de Servicio
7. Sumario

Creación de una API REST

1. Decidir qué elementos del *domain model* se expondrán como *recursos*.
2. Exponer los *aggregate roots* como colecciones.
3. Exponer las partes internas de un *aggregate roots* como elementos subordinados a un aggregate root.
4. Exponer las operaciones de negocio que no corresponda a operaciones CRUDs básicas o parametrizadas como controladores, asociados al recurso que corresponda.
5. Asignar una URI para cada recurso.
6. Decidir qué verbos HTTP son aplicables a cada respuesta.
7. Decidir los posibles parámetros de entrada a cada petición.
8. Decidir las posibles respuestas a cada petición.

Creación de una API REST

1. Decidir qué elementos del *domain model* se expondrán como *recursos*.
2. Exponer los *aggregate roots* como colecciones.
3. Exponer las partes internas de un *aggregate roots* como elementos subordinados a un aggregate root.
4. Exponer las operaciones de negocio que no corresponda a operaciones CRUDs básicas o parametrizadas como controladores, asociados al recurso que corresponda.
5. Asignar una URI para cada recurso.
6. Decidir qué verbos HTTP son aplicables a cada respuesta.
7. Decidir los posibles parámetros de entrada a cada petición.
8. Decidir las posibles respuestas a cada petición.

Creación de una API REST

1. Decidir qué elementos del *domain model* se expondrán como *recursos*.
2. Exponer los *aggregate roots* como colecciones.
3. Exponer las partes internas de un *aggregate roots* como elementos subordinados a un aggregate root.
4. Exponer las operaciones de negocio que no corresponda a operaciones CRUDs básicas o parametrizadas como controladores, asociados al recurso que corresponda.
5. Asignar una URI para cada recurso.
6. Decidir qué verbos HTTP son aplicables a cada respuesta.
7. Decidir los posibles parámetros de entrada a cada petición.
8. Decidir las posibles respuestas a cada petición.

Creación de una API REST

1. Decidir qué elementos del *domain model* se expondrán como *recursos*.
2. Exponer los *aggregate roots* como colecciones.
3. Exponer las partes internas de un *aggregate roots* como elementos subordinados a un aggregate root.
4. Exponer las operaciones de negocio que no corresponda a operaciones CRUDs básicas o parametrizadas como controladores, asociados al recurso que corresponda.
5. Asignar una URI para cada recurso.
6. Decidir qué verbos HTTP son aplicables a cada respuesta.
7. Decidir los posibles parámetros de entrada a cada petición.
8. Decidir las posibles respuestas a cada petición.

Creación de una API REST

1. Decidir qué elementos del *domain model* se expondrán como *recursos*.
2. Exponer los *aggregate roots* como colecciones.
3. Exponer las partes internas de un *aggregate roots* como elementos subordinados a un aggregate root.
4. Exponer las operaciones de negocio que no corresponda a operaciones CRUDs básicas o parametrizadas como controladores, asociados al recurso que corresponda.
5. Asignar una URI para cada recurso.
6. Decidir qué verbos HTTP son aplicables a cada respuesta.
7. Decidir los posibles parámetros de entrada a cada petición.
8. Decidir las posibles respuestas a cada petición.

Creación de una API REST

1. Decidir qué elementos del *domain model* se expondrán como *recursos*.
2. Exponer los *aggregate roots* como colecciones.
3. Exponer las partes internas de un *aggregate roots* como elementos subordinados a un aggregate root.
4. Exponer las operaciones de negocio que no corresponda a operaciones CRUDs básicas o parametrizadas como controladores, asociados al recurso que corresponda.
5. Asignar una URI para cada recurso.
6. Decidir qué verbos HTTP son aplicables a cada respuesta.
7. Decidir los posibles parámetros de entrada a cada petición.
8. Decidir las posibles respuestas a cada petición.

Creación de una API REST

1. Decidir qué elementos del *domain model* se expondrán como *recursos*.
2. Exponer los *aggregate roots* como colecciones.
3. Exponer las partes internas de un *aggregate roots* como elementos subordinados a un aggregate root.
4. Exponer las operaciones de negocio que no corresponda a operaciones CRUDs básicas o parametrizadas como controladores, asociados al recurso que corresponda.
5. Asignar una URI para cada recurso.
6. Decidir qué verbos HTTP son aplicables a cada respuesta.
7. Decidir los posibles parámetros de entrada a cada petición.
8. Decidir las posibles respuestas a cada petición.

Creación de una API REST

1. Decidir qué elementos del *domain model* se expondrán como *recursos*.
2. Exponer los *aggregate roots* como colecciones.
3. Exponer las partes internas de un *aggregate roots* como elementos subordinados a un aggregate root.
4. Exponer las operaciones de negocio que no corresponda a operaciones CRUDs básicas o parametrizadas como controladores, asociados al recurso que corresponda.
5. Asignar una URI para cada recurso.
6. Decidir qué verbos HTTP son aplicables a cada respuesta.
7. Decidir los posibles parámetros de entrada a cada petición.
8. Decidir las posibles respuestas a cada petición.

Identificación de Recursos REST

1. Las colecciones de *aggregates roots* se identifican con sustantivos en plural asociados a la raíz de la organización (`http://carsharing.es/viajes`).
2. Las instancias de un *aggregates roots* se identifican añadiendo el identificador de la instancia tras la URI asociado al *aggregate root* (`/viajes/{viajeId}`).
3. Los partes internas de un *aggregate* se exponen siguiendo las mismas reglas, utilizando la URI del *aggregate root* como prefijo (`/viajes/{viajeId}/conductor`).
4. La regla anterior se aplica recursivamente a las partes internas de las partes internas de un aggregate (`/viajes/{viajeId}/conductor/vehiculos/`).
5. Las operaciones no CRUD sobre un recurso se especifican como verbos tras la instancia del recurso al que se aplican (`/viajes/{viajeId}/cerrar`).

Identificación de Recursos REST

1. Las colecciones de *aggregates roots* se identifican con sustantivos en plural asociados a la raíz de la organización (`http://carsharing.es/viajes`).
2. Las instancias de un *aggregates roots* se identifican añadiendo el identificador de la instancia tras la URI asociado al *aggregate root* (`/viajes/{viajeId}`).
3. Los partes internas de un *aggregate* se exponen siguiendo las mismas reglas, utilizando la URI del *aggregate root* como prefijo (`/viajes/{viajeId}/conductor`).
4. La regla anterior se aplica recursivamente a las partes internas de las partes internas de un aggregate (`/viajes/{viajeId}/conductor/vehiculos/`).
5. Las operaciones no CRUD sobre un recurso se especifican como verbos tras la instancia del recurso al que se aplican (`/viajes/{viajeId}/cerrar`).

Identificación de Recursos REST

1. Las colecciones de *aggregates roots* se identifican con sustantivos en plural asociados a la raíz de la organización (`http://carsharing.es/viajes`).
2. Las instancias de un *aggregates roots* se identifican añadiendo el identificador de la instancia tras la URI asociado al *aggregate root* (`/viajes/{viajeId}`).
3. Los partes internas de un *aggregate* se exponen siguiendo las mismas reglas, utilizando la URI del *aggregate root* como prefijo (`/viajes/{viajeId}/conductor`).
4. La regla anterior se aplica recursivamente a las partes internas de las partes internas de un aggregate (`/viajes/{viajeId}/conductor/vehiculos/`).
5. Las operaciones no CRUD sobre un recurso se especifican como verbos tras la instancia del recurso al que se aplican (`/viajes/{viajeId}/cerrar`).

Identificación de Recursos REST

1. Las colecciones de *aggregates roots* se identifican con sustantivos en plural asociados a la raíz de la organización (`http://carsharing.es/viajes`).
2. Las instancias de un *aggregates roots* se identifican añadiendo el identificador de la instancia tras la URI asociado al *aggregate root* (`/viajes/{viajeId}`).
3. Los partes internas de un *aggregate* se exponen siguiendo las mismas reglas, utilizando la URI del *aggregate root* como prefijo (`/viajes/{viajeId}/conductor`).
4. La regla anterior se aplica recursivamente a las partes internas de las partes internas de un aggregate (`/viajes/{viajeId}/conductor/vehiculos/`).
5. Las operaciones no CRUD sobre un recurso se especifican como verbos tras la instancia del recurso al que se aplican (`/viajes/{viajeId}/cerrar`).

Identificación de Recursos REST

1. Las colecciones de *aggregates roots* se identifican con sustantivos en plural asociados a la raíz de la organización (`http://carsharing.es/viajes`).
2. Las instancias de un *aggregates roots* se identifican añadiendo el identificador de la instancia tras la URI asociado al *aggregate root* (`/viajes/{viajeId}`).
3. Los partes internas de un *aggregate* se exponen siguiendo las mismas reglas, utilizando la URI del *aggregate root* como prefijo (`/viajes/{viajeId}/conductor`).
4. La regla anterior se aplica recursivamente a las partes internas de las partes internas de un aggregate (`/viajes/{viajeId}/conductor/vehiculos/`).
5. Las operaciones no CRUD sobre un recurso se especifican como verbos tras la instancia del recurso al que se aplican (`/viajes/{viajeId}/cerrar`).

Guía de Estilo para URIs REST

1. Utilizar / para indicar jerarquía o navegación por el modelo de dominio.
2. No añadir una / final.
3. Usar guiones para separar palabras.
4. No usar guiones bajos.
5. Usar sólo minúsculas.
6. No referir ficheros, sino recursos.
7. No utilizar URIs para distinguir operaciones CRUD.
8. Utilizar parámetros para refinar operaciones CRUD (`/viajes?origen=Santander&destino=Bilbao`).

Guía de Estilo para URIs REST

1. Utilizar / para indicar jerarquía o navegación por el modelo de dominio.
2. No añadir una / final.
3. Usar guiones para separar palabras.
4. No usar guiones bajos.
5. Usar sólo minúsculas.
6. No referir ficheros, sino recursos.
7. No utilizar URIs para distinguir operaciones CRUD.
8. Utilizar parámetros para refinar operaciones CRUD (`/viajes?origen=Santander&destino=Bilbao`).

Guía de Estilo para URIs REST

1. Utilizar / para indicar jerarquía o navegación por el modelo de dominio.
2. No añadir una / final.
3. Usar guiones para separar palabras.
4. No usar guiones bajos.
5. Usar sólo minúsculas.
6. No referir ficheros, sino recursos.
7. No utilizar URIs para distinguir operaciones CRUD.
8. Utilizar parámetros para refinar operaciones CRUD (`/viajes?origen=Santander&destino=Bilbao`).

Guía de Estilo para URIs REST

1. Utilizar / para indicar jerarquía o navegación por el modelo de dominio.
2. No añadir una / final.
3. Usar guiones para separar palabras.
4. No usar guiones bajos.
5. Usar sólo minúsculas.
6. No referir ficheros, sino recursos.
7. No utilizar URIs para distinguir operaciones CRUD.
8. Utilizar parámetros para refinar operaciones CRUD (`/viajes?origen=Santander&destino=Bilbao`).

Guía de Estilo para URIs REST

1. Utilizar / para indicar jerarquía o navegación por el modelo de dominio.
2. No añadir una / final.
3. Usar guiones para separar palabras.
4. No usar guiones bajos.
5. Usar sólo minúsculas.
6. No referir ficheros, sino recursos.
7. No utilizar URIs para distinguir operaciones CRUD.
8. Utilizar parámetros para refinar operaciones CRUD (`/viajes?origen=Santander&destino=Bilbao`).

Guía de Estilo para URIs REST

1. Utilizar / para indicar jerarquía o navegación por el modelo de dominio.
2. No añadir una / final.
3. Usar guiones para separar palabras.
4. No usar guiones bajos.
5. Usar sólo minúsculas.
6. No referir ficheros, sino recursos.
7. No utilizar URIs para distinguir operaciones CRUD.
8. Utilizar parámetros para refinar operaciones CRUD (`/viajes?origen=Santander&destino=Bilbao`).

Guía de Estilo para URIs REST

1. Utilizar / para indicar jerarquía o navegación por el modelo de dominio.
2. No añadir una / final.
3. Usar guiones para separar palabras.
4. No usar guiones bajos.
5. Usar sólo minúsculas.
6. No referir ficheros, sino recursos.
7. No utilizar URIs para distinguir operaciones CRUD.
8. Utilizar parámetros para refinar operaciones CRUD (`/viajes?origen=Santander&destino=Bilbao`).

Guía de Estilo para URIs REST

1. Utilizar / para indicar jerarquía o navegación por el modelo de dominio.
2. No añadir una / final.
3. Usar guiones para separar palabras.
4. No usar guiones bajos.
5. Usar sólo minúsculas.
6. No referir ficheros, sino recursos.
7. No utilizar URIs para distinguir operaciones CRUD.
8. Utilizar parámetros para refinar operaciones CRUD (`/viajes?origen=Santander&destino=Bilbao`).

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
   - Identificación de recursos
   - Verbos HTTP y Recursos REST
   - Códigos de Respuestas HTTP
   - HATEOAS
   - Niveles de Adopción
6. Patrones de la Capa de Servicio
7. Sumario

Verbos HTTP y Recursos REST

GET	Recupera un recurso.
POST	Añade un recurso (no identificado) a una colección.
PUT	Actualiza completamente (o añade) un recurso (identificado).
PATCH	Actualizar parcialmente un recurso.
DELETE	Elimina un recurso.

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
   - Identificación de recursos
   - Verbos HTTP y Recursos REST
   - Códigos de Respuestas HTTP
   - HATEOAS
   - Niveles de Adopción
6. Patrones de la Capa de Servicio
7. Sumario

Códigos Respuesta HTTP

200	Ok
201	Created
202	Accepted
204	No content
400	Bad request
401	Unauthorized
404	Not Found
405	Method Not Allowed
412	Precondition Failed
500	Internal Server Error
501	Not Implemented

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
   - Identificación de recursos
   - Verbos HTTP y Recursos REST
   - Códigos de Respuestas HTTP
   - HATEOAS
   - Niveles de Adopción
6. Patrones de la Capa de Servicio
7. Sumario

HATEOAS sobre JSON


```
{
"_links": {
   "self": {"href": "http://example.com/api/book/hal-cookbook"},
   "next": {"href": "http://example.com/api/book/hal-case-study"},
   "prev": {"href": "http://example.com/api/book/json-and-beyond"},
   "first": {"href": "http://example.com/api/book/catalog"},
   "last": {"href": "http://example.com/api/book/upcoming-books"}
 },
 "_embedded": {
   "author": {
     "_links": {"self": { "href": "http://author-example.com"}},
     "id": "shahadat",
     "name": "Shahadat Hossain Khan"} },
 "id": "hal-cookbook",
 "name": "HAL Cookbook"
}
```

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
   - Identificación de recursos
   - Verbos HTTP y Recursos REST
   - Códigos de Respuestas HTTP
   - HATEOAS
   - Niveles de Adopción
6. Patrones de la Capa de Servicio
7. Sumario

Niveles de Adopción REST

1. Utiliza el protocolo HTTP.
2. Utiliza recursos REST.
3. Utiliza verbos HTTP.
4. Utiliza HATEOAS

Niveles de Adopción REST

1. Utiliza el protocolo HTTP.
2. Utiliza recursos REST.
3. Utiliza verbos HTTP.
4. Utiliza HATEOAS

Niveles de Adopción REST

1. Utiliza el protocolo HTTP.
2. Utiliza recursos REST.
3. Utiliza verbos HTTP.
4. Utiliza HATEOAS

Niveles de Adopción REST

1. Utiliza el protocolo HTTP.
2. Utiliza recursos REST.
3. Utiliza verbos HTTP.
4. Utiliza HATEOAS

Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

Índice

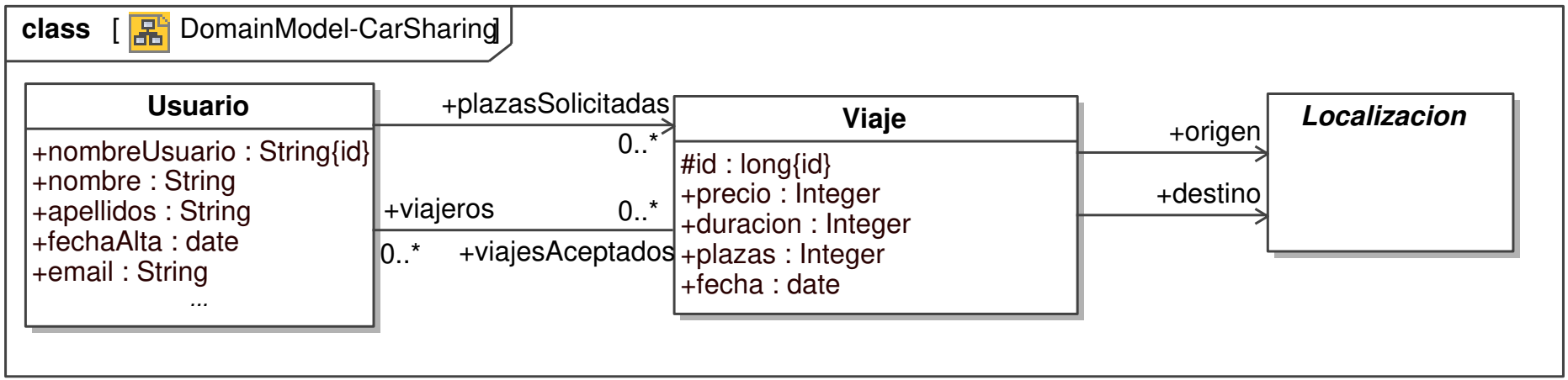
1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
   - Data Transfer Object
   - Front Controller
   - Procesamiento de Peticiones de Servicio
   - Unit of Work
7. Sumario

Data Transfer Object

Problema

En un sistema distribuido:

1. ¿Cómo consigo transferir un objeto entre computadoras sin tener que serializar en texto todos los objetos del sistema?
2. ¿Cómo consigo reducir el número de llamadas entre computadores para recuperar datos?

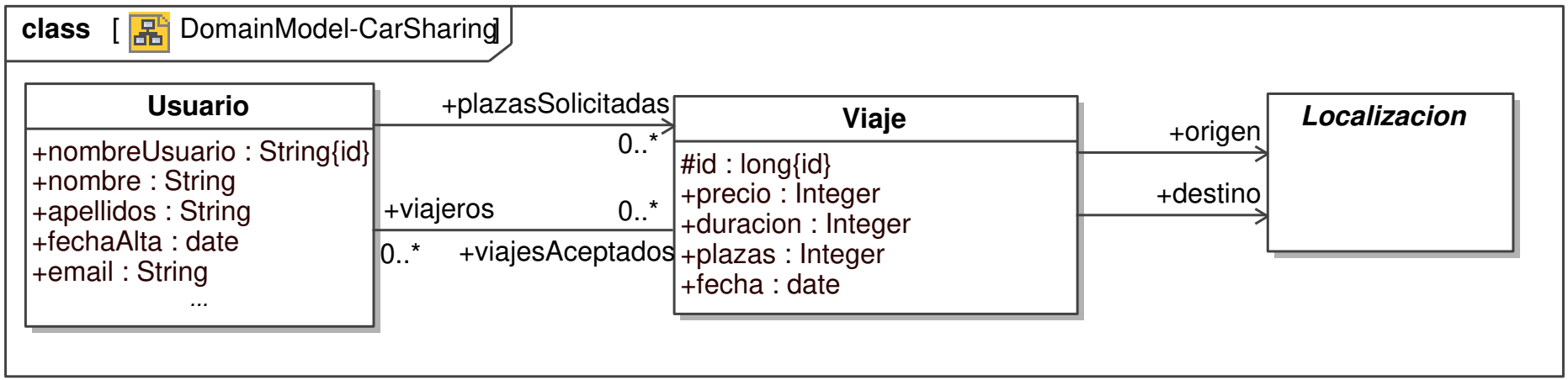


Data Transfer Object

Problema

En un sistema distribuido:

1. ¿Cómo consigo transferir un objeto entre computadoras sin tener que serializar en texto todos los objetos del sistema?
2. ¿Cómo consigo reducir el número de llamadas entre computadores para recuperar datos?

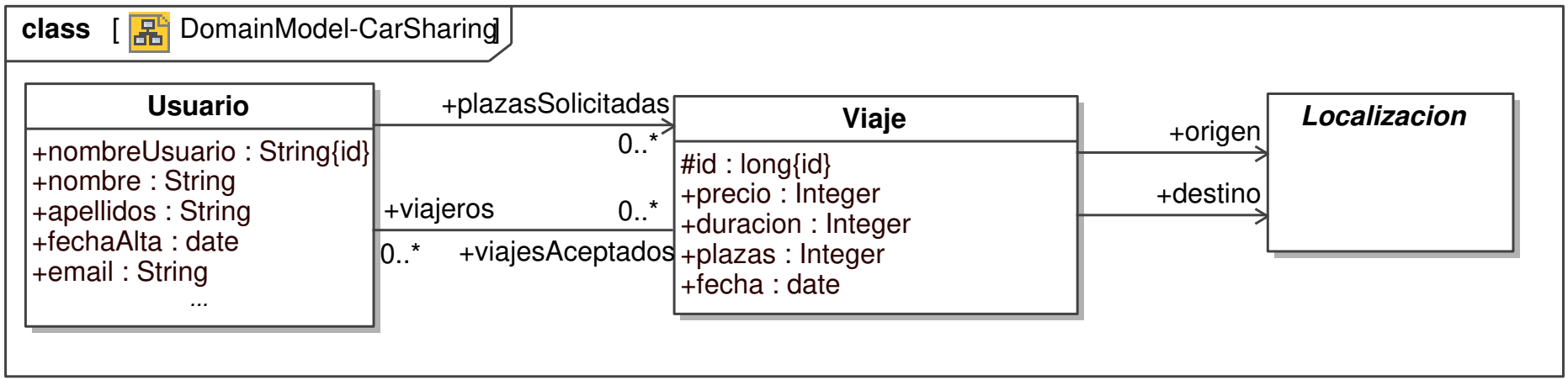


Data Transfer Object

Problema

En un sistema distribuido:

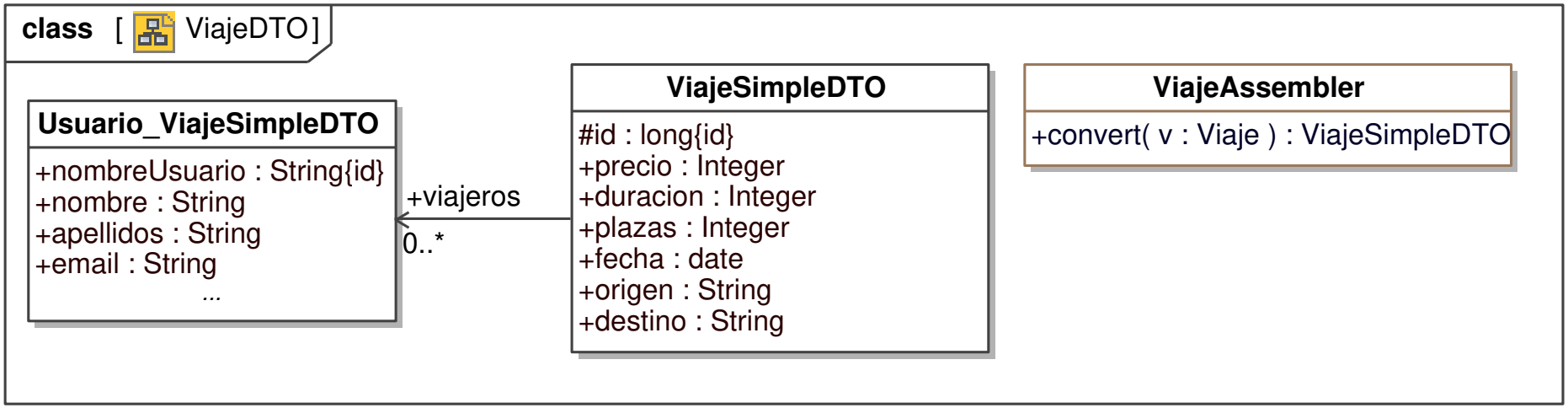
1. ¿Cómo consigo transferir un objeto entre computadoras sin tener que serializar en texto todos los objetos del sistema?
2. ¿Cómo consigo reducir el número de llamadas entre computadores para recuperar datos?



Data Transfer Object

Solución

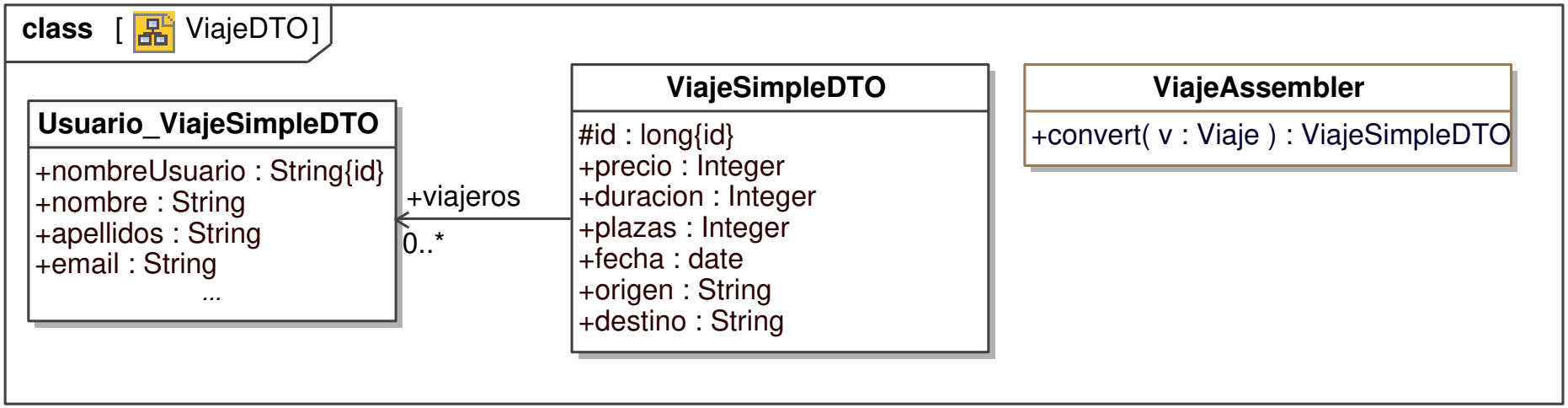
1. Crear un conjunto de objetos con la información necesaria para un propósito concreto y que sean *serializables*.
2. Crear una clase *Assembler* que recibe una entidad y genere uno de sus posibles DTOs.
3. Devolver DTOs en lugar de entidades de dominio.



Data Transfer Object

Solución

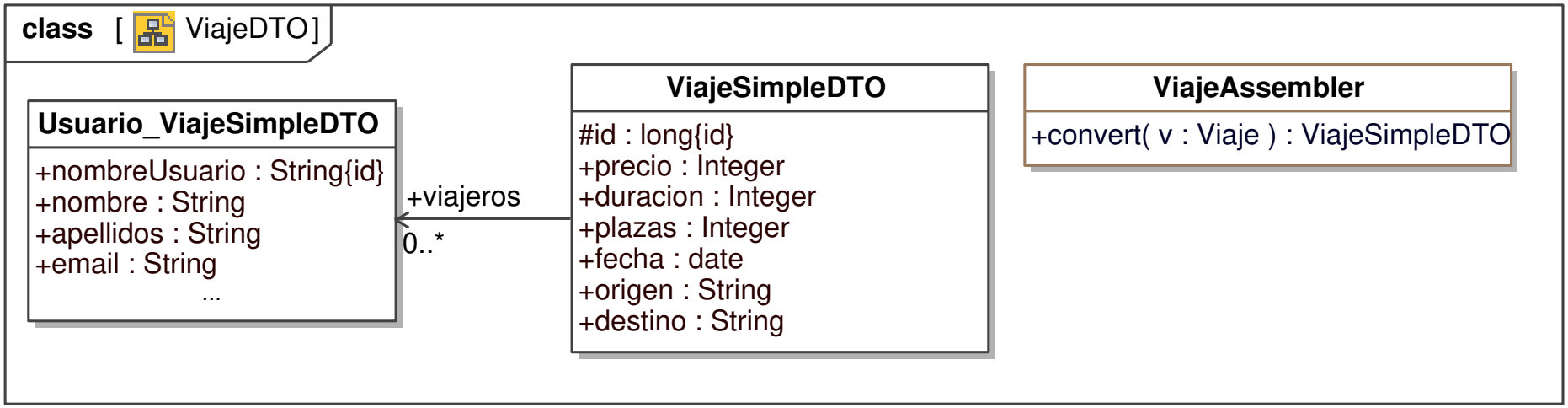
1. Crear un conjunto de objetos con la información necesaria para un propósito concreto y que sean *serializables*.
2. Crear una clase *Assembler* que recibe una entidad y genere uno de sus posibles DTOs.
3. Devolver DTOs en lugar de entidades de dominio.



Data Transfer Object

Solución

1. Crear un conjunto de objetos con la información necesaria para un propósito concreto y que sean *serializables*.
2. Crear una clase *Assembler* que recibe una entidad y genere uno de sus posibles DTOs.
3. Devolver DTOs en lugar de entidades de dominio.



Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
   - Data Transfer Object
   - Front Controller
   - Procesamiento de Peticiones de Servicio
   - Unit of Work
7. Sumario

Front Controller

Problema

1. ¿Cómo atender peticiones HTTP?
2. ¿Cómo encapsular la lógica común a la gestión de peticiones HTTP?

Solución

Crear una clase donde se deleguen todas las peticiones HTTP y que:

1. Parsea la URL solicitada.
2. Selecciona el controlador asociado a cada URL.
3. Extrae datos (si es necesario) de la petición HTTP.
4. Invoca al controlador.
5. Compone una respuesta con los valores devueltos por el controlador.

Front Controller

Problema

1. ¿Cómo atender peticiones HTTP?
2. ¿Cómo encapsular la lógica común a la gestión de peticiones HTTP?

Solución

Crear una clase donde se deleguen todas las peticiones HTTP y que:

1. Parsea la URL solicitada.
2. Selecciona el controlador asociado a cada URL.
3. Extrae datos (si es necesario) de la petición HTTP.
4. Invoca al controlador.
5. Compone una respuesta con los valores devueltos por el controlador.

Front Controller

Problema

1. ¿Cómo atender peticiones HTTP?
2. ¿Cómo encapsular la lógica común a la gestión de peticiones HTTP?

Solución

Crear una clase donde se deleguen todas las peticiones HTTP y que:

1. Parsea la URL solicitada.
2. Selecciona el controlador asociado a cada URL.
3. Extrae datos (si es necesario) de la petición HTTP.
4. Invoca al controlador.
5. Compone una respuesta con los valores devueltos por el controlador.

Front Controller

Problema

1. ¿Cómo atender peticiones HTTP?
2. ¿Cómo encapsular la lógica común a la gestión de peticiones HTTP?

Solución

Crear una clase donde se deleguen todas las peticiones HTTP y que:

1. Parsea la URL solicitada.
2. Selecciona el controlador asociado a cada URL.
3. Extrae datos (si es necesario) de la petición HTTP.
4. Invoca al controlador.
5. Compone una respuesta con los valores devueltos por el controlador.

Front Controller

Problema

1. ¿Cómo atender peticiones HTTP?
2. ¿Cómo encapsular la lógica común a la gestión de peticiones HTTP?

Solución

Crear una clase donde se deleguen todas las peticiones HTTP y que:

1. Parsea la URL solicitada.
2. Selecciona el controlador asociado a cada URL.
3. Extrae datos (si es necesario) de la petición HTTP.
4. Invoca al controlador.
5. Compone una respuesta con los valores devueltos por el controlador.

Front Controller

Problema

1. ¿Cómo atender peticiones HTTP?
2. ¿Cómo encapsular la lógica común a la gestión de peticiones HTTP?

Solución

Crear una clase donde se deleguen todas las peticiones HTTP y que:

1. Parsea la URL solicitada.
2. Selecciona el controlador asociado a cada URL.
3. Extrae datos (si es necesario) de la petición HTTP.
4. Invoca al controlador.
5. Compone una respuesta con los valores devueltos por el controlador.

Front Controller

Problema

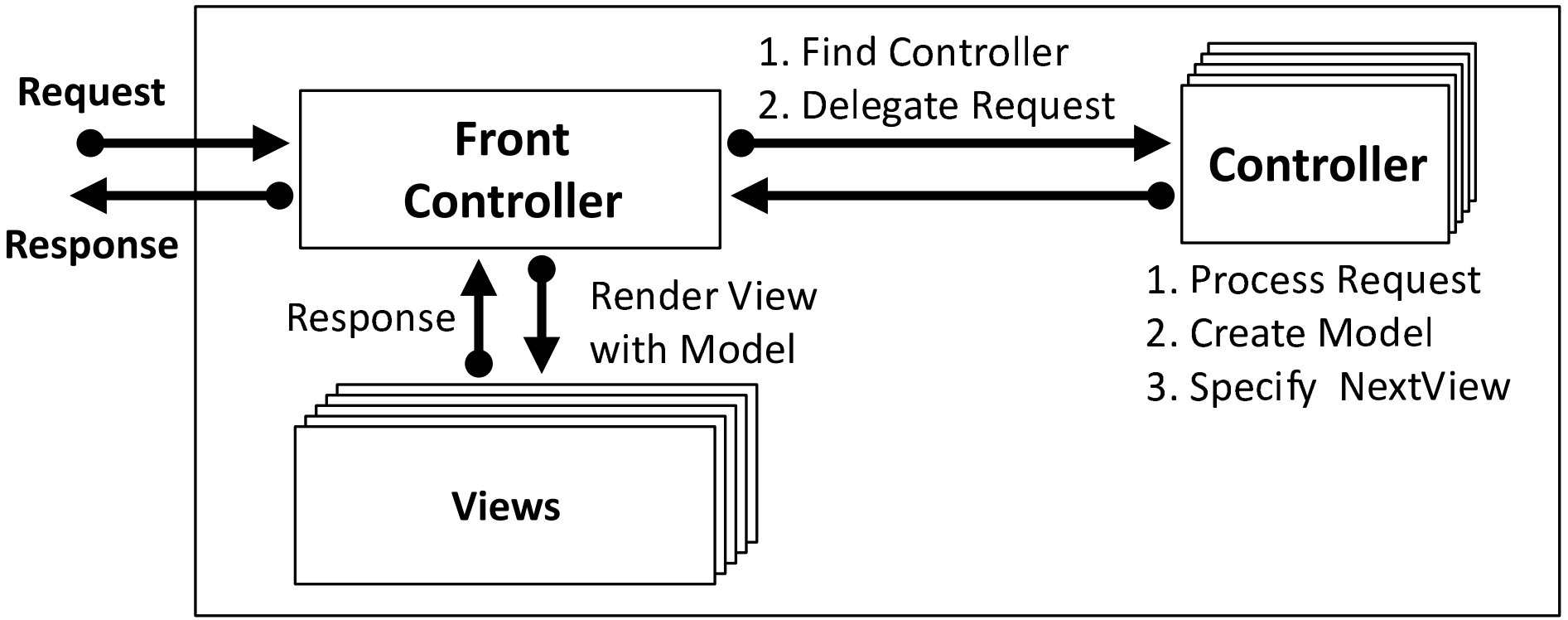
1. ¿Cómo atender peticiones HTTP?
2. ¿Cómo encapsular la lógica común a la gestión de peticiones HTTP?

Solución

Crear una clase donde se deleguen todas las peticiones HTTP y que:

1. Parsea la URL solicitada.
2. Selecciona el controlador asociado a cada URL.
3. Extrae datos (si es necesario) de la petición HTTP.
4. Invoca al controlador.
5. Compone una respuesta con los valores devueltos por el controlador.

Front Controller



Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
   - Data Transfer Object
   - Front Controller
   - Procesamiento de Peticiones de Servicio
   - Unit of Work
7. Sumario

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

Procesamiento de Peticiones de Servicio

1. El servidor web recibe la petición HTTP.
2. El servidor web delega en el *Front Controller (FC)*.
3. El *FC* extrae parámetros y delega el controlador adecuado.
4. El controlador valida los datos de entrada.
5. El controlador delega en la capa de servicio.
6. La capa de servicio recupera objetos de los repositorios.
7. La capa de servicio abre una transacción.
8. La capa de servicio delega en una entidad de dominio.
9. La capa de servicio recoge los resultados de la entidad de dominio.
10. La capa de servicio cierra (o aborta) la transacción.
11. La capa de servicio devuelve la respuesta al controlador.
12. El controlador compone una respuesta modelo.
13. El *Front Controller* convierte la respuesta modelo en respuesta HTTP.
14. El servidor web devuelve la respuesta HTTP al cliente.

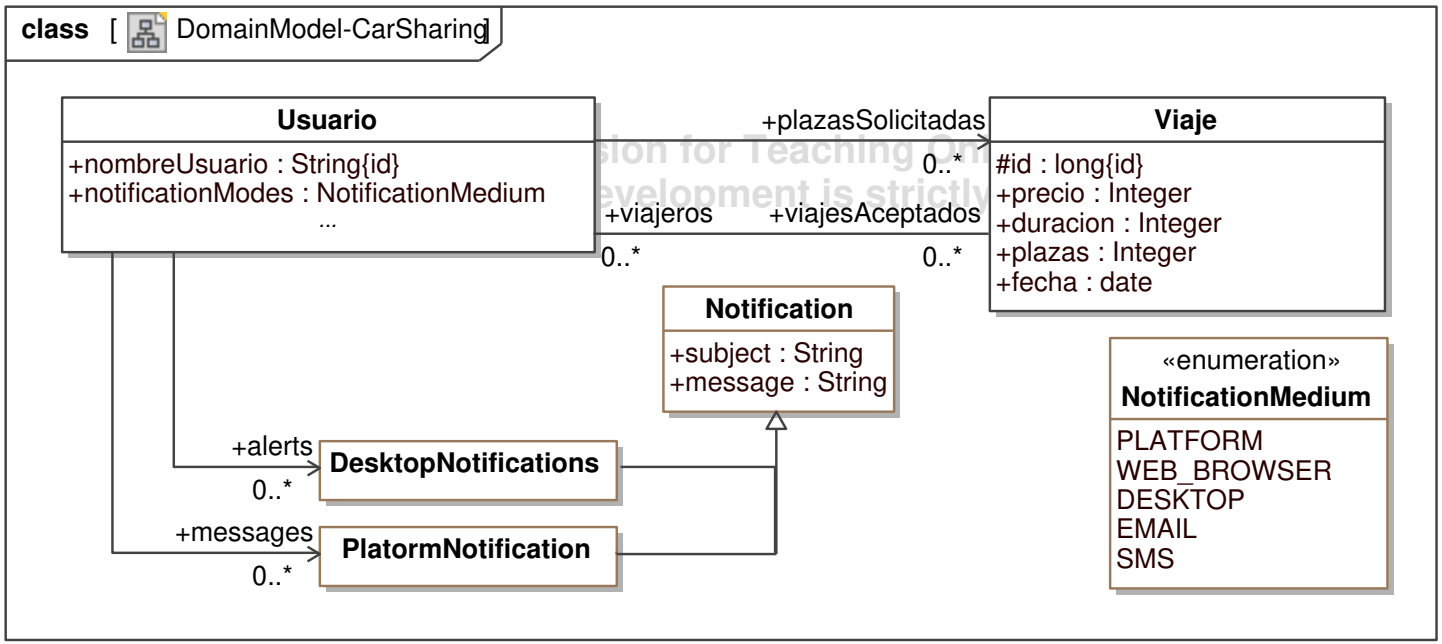
Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
   - Data Transfer Object
   - Front Controller
   - Procesamiento de Peticiones de Servicio
   - Unit of Work
7. Sumario

Unit Of Work

Problema

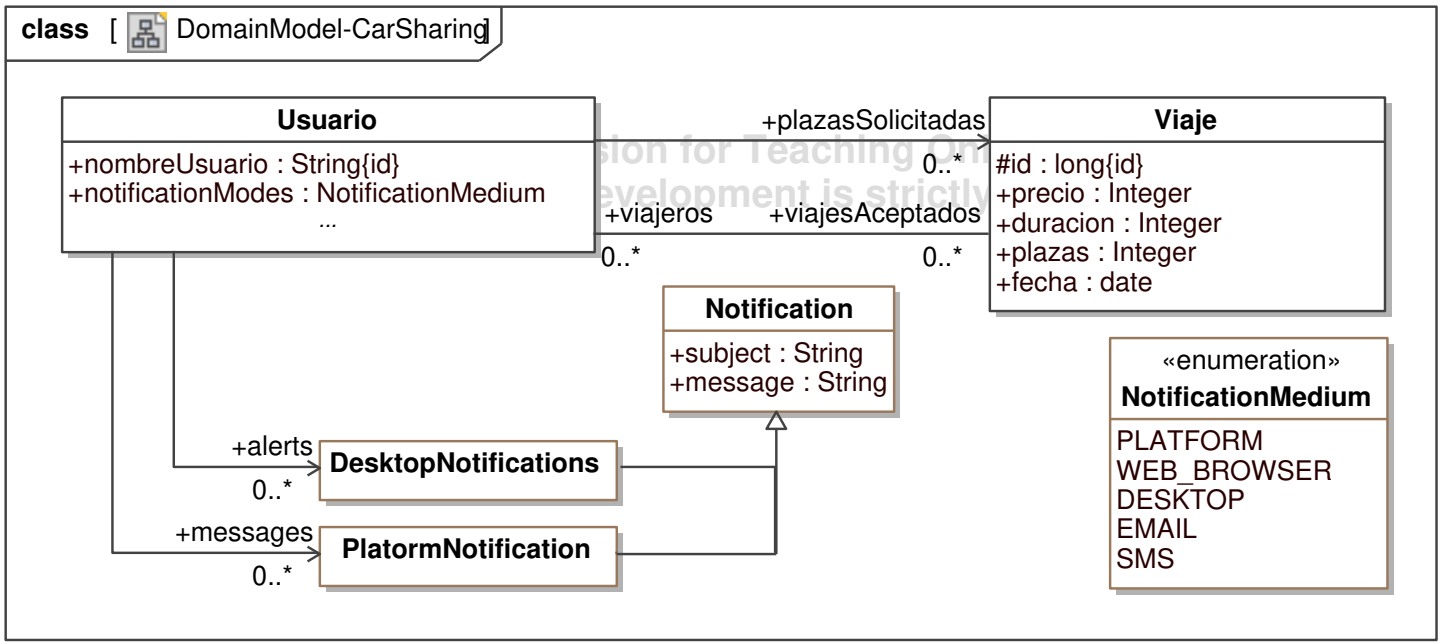
1. ¿Cómo saber qué elementos han cambiado tras delegar en una entidad de dominio?
2. ¿Cómo evitar tener abiertas transacciones demasiado largas en persistencia?



Unit Of Work

Problema

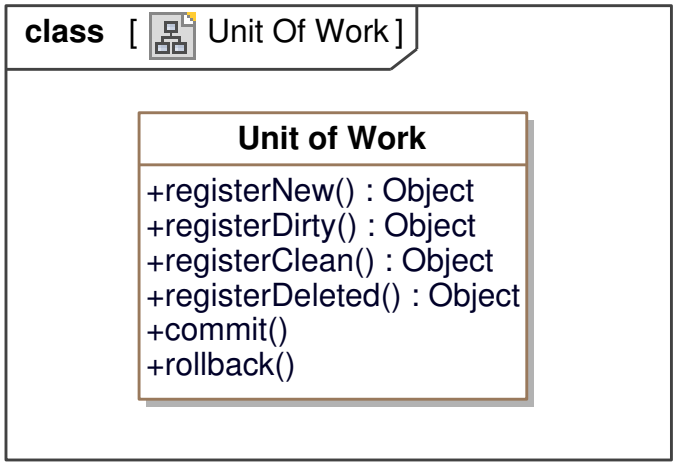
1. ¿Cómo saber qué elementos han cambiado tras delegar en una entidad de dominio?
2. ¿Cómo evitar tener abiertas transacciones demasiado largas en persistencia?



Unit Of Work

Solución

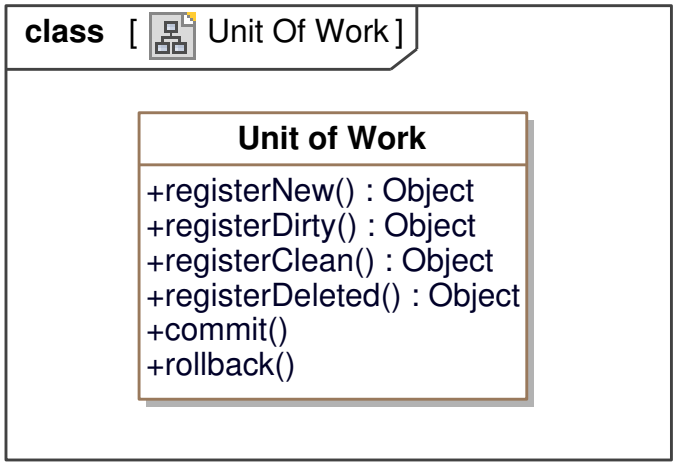
1. Crear una clase *Unidad de Trabajo*, para registrar los cambios realizadas en una *transacción de negocio*.
2. Anotar en la *Unidad de Trabajo* las modificaciones realizadas de manera transparente a la lógica de negocio.
3. Persistir todos los cambios anotados al cerrar la transacción de negocio.



Unit Of Work

Solución

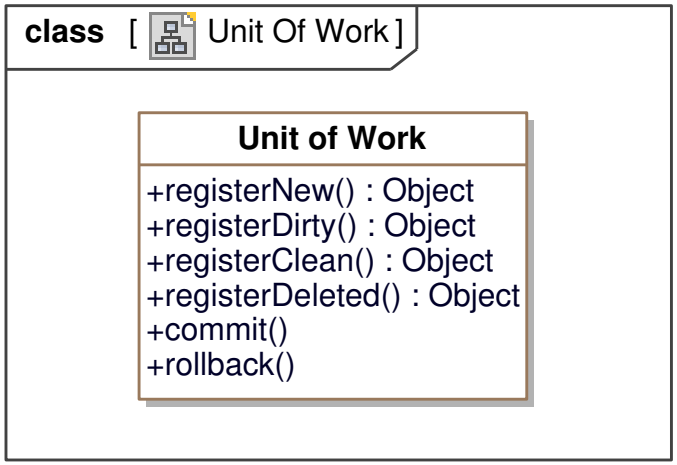
1. Crear una clase *Unidad de Trabajo*, para registrar los cambios realizadas en una *transacción de negocio*.
2. Anotar en la *Unidad de Trabajo* las modificaciones realizadas de manera transparente a la lógica de negocio.
3. Persistir todos los cambios anotados al cerrar la transacción de negocio.



Unit Of Work

Solución

1. Crear una clase *Unidad de Trabajo*, para registrar los cambios realizadas en una *transacción de negocio*.
2. Anotar en la *Unidad de Trabajo* las modificaciones realizadas de manera transparente a la lógica de negocio.
3. Persistir todos los cambios anotados al cerrar la transacción de negocio.



Índice

1. Introducción
2. Arquitecturas REST
3. Protocolo HTTP
4. JSON
5. Creación de una API REST sobre HTTP
6. Patrones de la Capa de Servicio
7. Sumario

¿Qué tengo que saber de todo ésto?

1. Comprender cuáles son las responsabilidades de la capa de negocio.
2. Comprender cuáles son los fundamentos de las arquitecturas REST.
3. Comprender el funcionamiento básico del protocolo HTTP.
4. Comprender la sintaxis básica de JSON.
5. Ser capaz de especificar una API REST para un modelo de dominio.
6. Ser capaz de definir DTOs.
7. Comprender el funcionamiento de los *Front Controller*.
8. Comprender el funcionamiento de las *Unit of Work*.

¿Qué tengo que saber de todo ésto?

1. Comprender cuáles son las responsabilidades de la capa de negocio.
2. Comprender cuáles son los fundamentos de las arquitecturas REST.
3. Comprender el funcionamiento básico del protocolo HTTP.
4. Comprender la sintaxis básica de JSON.
5. Ser capaz de especificar una API REST para un modelo de dominio.
6. Ser capaz de definir DTOs.
7. Comprender el funcionamiento de los *Front Controller*.
8. Comprender el funcionamiento de las *Unit of Work*.

¿Qué tengo que saber de todo ésto?

1. Comprender cuáles son las responsabilidades de la capa de negocio.
2. Comprender cuáles son los fundamentos de las arquitecturas REST.
3. Comprender el funcionamiento básico del protocolo HTTP.
4. Comprender la sintaxis básica de JSON.
5. Ser capaz de especificar una API REST para un modelo de dominio.
6. Ser capaz de definir DTOs.
7. Comprender el funcionamiento de los *Front Controller*.
8. Comprender el funcionamiento de las *Unit of Work*.

¿Qué tengo que saber de todo ésto?

1. Comprender cuáles son las responsabilidades de la capa de negocio.
2. Comprender cuáles son los fundamentos de las arquitecturas REST.
3. Comprender el funcionamiento básico del protocolo HTTP.
4. Comprender la sintaxis básica de JSON.
5. Ser capaz de especificar una API REST para un modelo de dominio.
6. Ser capaz de definir DTOs.
7. Comprender el funcionamiento de los *Front Controller*.
8. Comprender el funcionamiento de las *Unit of Work*.

¿Qué tengo que saber de todo ésto?

1. Comprender cuáles son las responsabilidades de la capa de negocio.
2. Comprender cuáles son los fundamentos de las arquitecturas REST.
3. Comprender el funcionamiento básico del protocolo HTTP.
4. Comprender la sintaxis básica de JSON.
5. Ser capaz de especificar una API REST para un modelo de dominio.
6. Ser capaz de definir DTOs.
7. Comprender el funcionamiento de los *Front Controller*.
8. Comprender el funcionamiento de las *Unit of Work*.

¿Qué tengo que saber de todo ésto?

1. Comprender cuáles son las responsabilidades de la capa de negocio.
2. Comprender cuáles son los fundamentos de las arquitecturas REST.
3. Comprender el funcionamiento básico del protocolo HTTP.
4. Comprender la sintaxis básica de JSON.
5. Ser capaz de especificar una API REST para un modelo de dominio.
6. Ser capaz de definir DTOs.
7. Comprender el funcionamiento de los *Front Controller*.
8. Comprender el funcionamiento de las *Unit of Work*.

¿Qué tengo que saber de todo ésto?

1. Comprender cuáles son las responsabilidades de la capa de negocio.
2. Comprender cuáles son los fundamentos de las arquitecturas REST.
3. Comprender el funcionamiento básico del protocolo HTTP.
4. Comprender la sintaxis básica de JSON.
5. Ser capaz de especificar una API REST para un modelo de dominio.
6. Ser capaz de definir DTOs.
7. Comprender el funcionamiento de los *Front Controller*.
8. Comprender el funcionamiento de las *Unit of Work*.

¿Qué tengo que saber de todo ésto?

1. Comprender cuáles son las responsabilidades de la capa de negocio.
2. Comprender cuáles son los fundamentos de las arquitecturas REST.
3. Comprender el funcionamiento básico del protocolo HTTP.
4. Comprender la sintaxis básica de JSON.
5. Ser capaz de especificar una API REST para un modelo de dominio.
6. Ser capaz de definir DTOs.
7. Comprender el funcionamiento de los *Front Controller*.
8. Comprender el funcionamiento de las *Unit of Work*.