Factum — Entregable 1: Web Services

Nicolas José Machado Martinez

Docente

Edwar Alfonso Villamizar Vallejo

Unidades tegnologicas de Santander

Web Services

2025

Resumen

Este documento presenta el desarrollo del proyecto Factum, una aplicación web de facturación diseñada para pequeñas y medianas empresas. El trabajo sigue la estructura solicitada en el curso de Web Services, abordando desde la definición del tema hasta la implementación conceptual de arquitectura en capas, diseño de entidades de dominio, repositorios, servicios, exposición de APIs REST y estandarización en formato JSON. Además, se justifica el uso de REST frente a SOAP y se plantean conclusiones sobre el aprendizaje logrado al integrar cada capa. La documentación busca demostrar comprensión de los principios SOLID, buenas prácticas de diseño de servicios web y la importancia de la separación de responsabilidades en arquitecturas modernas.

Palabras clave: Facturación, Web Services, REST, JSON, Arquitectura en Capas.

Introducción

La facturación es un proceso esencial en toda empresa, ya que permite registrar operaciones comerciales, controlar inventarios y cumplir obligaciones fiscales. Sin embargo, en muchos negocios pequeños y medianos, este proceso sigue siendo manual, lo que genera errores, retrasos y dificultad para centralizar la información.

El proyecto Factum surge como una propuesta académica y práctica para implementar un sistema de facturación basado en servicios web, aplicando una arquitectura en capas que permita escalabilidad, mantenibilidad y facilidad de integración con aplicaciones externas. A lo largo de este documento se desarrolla el análisis del dominio, la definición de la persistencia, la lógica de negocio, la exposición de APIs con JSON estandarizado y la justificación de la arquitectura REST frente a SOAP. Finalmente, se presentan conclusiones sobre el aprendizaje y la importancia de estas prácticas en el desarrollo de software moderno.

Justificación

La elección de REST como arquitectura de servicios web para Factum se fundamenta en su simplicidad, compatibilidad con JSON y amplia adopción en el desarrollo moderno. REST permite usar los métodos estándar de HTTP (GET, POST, PUT, DELETE), lo que facilita la integración con navegadores, aplicaciones móviles y sistemas contables externos. Además, reduce la sobrecarga en comparación con SOAP, ofreciendo mejor rendimiento y menor consumo de ancho de banda.

Su naturaleza ligera y flexible asegura que Factum pueda escalar e interoperar fácilmente con servicios de terceros, como pasarelas de pago o bancos. Mientras que SOAP proporciona contratos más estrictos, REST responde mejor a la necesidad de agilidad, rapidez y facilidad de uso que requieren las pequeñas y medianas empresas que usarán Factum.

En conclusión, REST es la opción más adecuada porque garantiza un desarrollo sencillo, seguro y compatible con las tendencias actuales en servicios web.

Objetivos

Objetivo General

Diseñar y documentar un sistema de facturación web llamado **Factum**, aplicando arquitectura en capas y servicios web REST, que permita gestionar clientes, productos y facturas de forma ágil, segura e interoperable.

Objetivos Específicos

- Identificar el contexto, problema y entidades principales involucradas en el proceso de facturación.
- Definir la arquitectura del sistema bajo los principios de separación en capas (Domain, Repository, Service y API).
- Diseñar un esquema estandarizado de intercambio de información en formato JSON, incluyendo manejo de errores, fechas y números.
- Justificar la elección de REST como arquitectura de servicios web frente a SOAP.
- Documentar el sistema integrando diagramas, ejemplos de APIs y conclusiones.

Definición del Tema

Contexto:
Factum es una aplicación web de facturación pensada para pequeñas y medianas empresas que necesitan gestionar clientes, productos/servicios y emitir facturas electrónicas o en PDF. El sistema busca automatizar el proceso de facturación, llevar control de inventario mínimo y ofrecer endpoints estandarizados para ser consumidos por una interfaz web o clientes móviles.

Problema que resuelve:

- Evita la gestión manual de facturas (hojas de cálculo, papeles).
- Centraliza información de clientes y productos.
- Calcula impuestos y totales automáticamente.
- Permite integraciones mediante APIs REST para automatizar procesos contables.

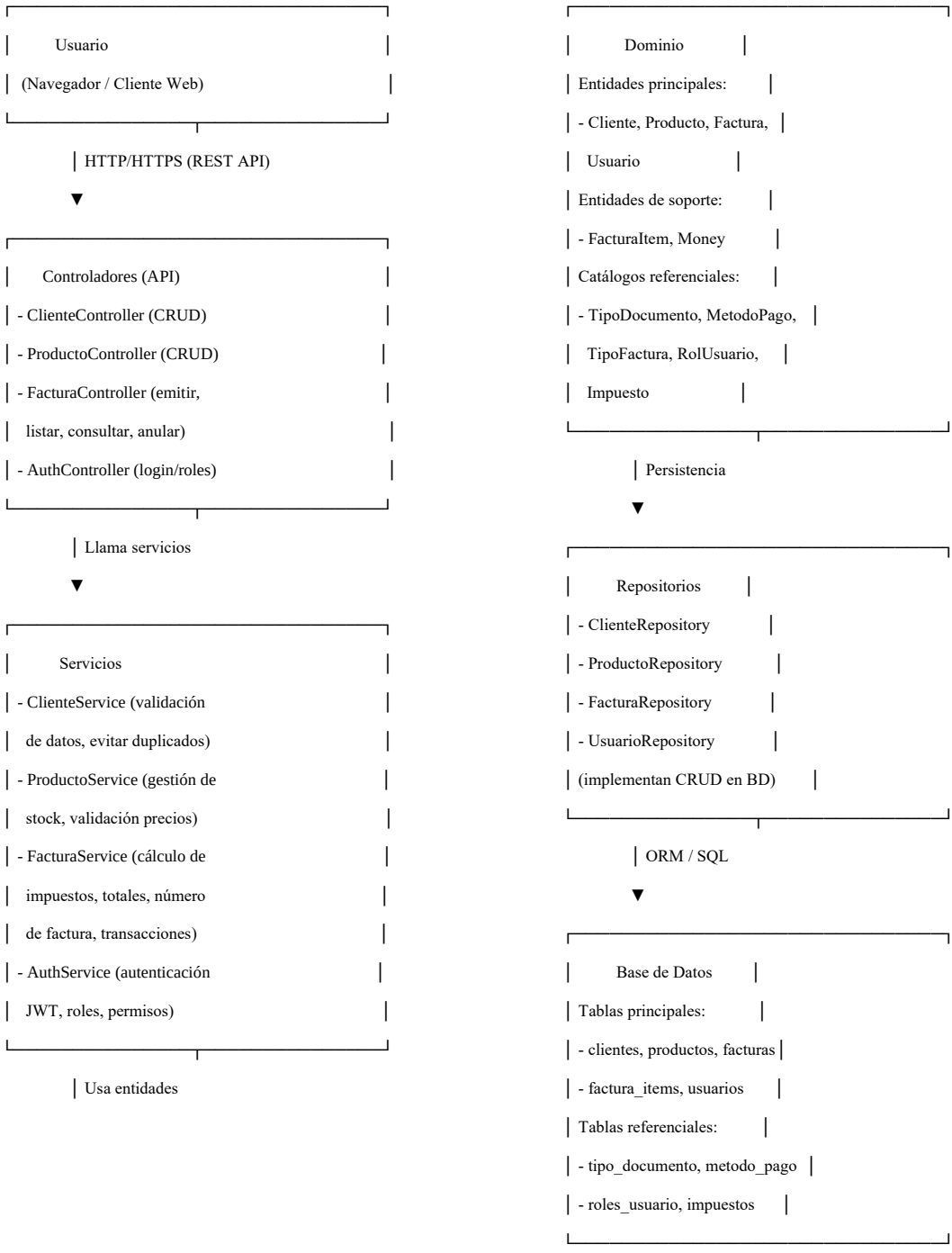
Entidades principales: Cliente, Producto, Factura, Usuario.

---

2. Arquitectura y Capas

Se propone una arquitectura en capas limpia que permita separación de responsabilidades y facilite pruebas, mantenimiento y escalabilidad.

2.1 Diagrama


Usuario
(Navegador / Cliente Web)
HTTP/HTTPS (REST API)
Controladores (API)
- ClienteController (CRUD)
- ProductoController (CRUD)
- FacturaController (emitir, listar, consultar, anular)
- AuthController (login/roles)
Llama servicios
Servicios
- ClienteService (validación de datos, evitar duplicados)
- ProductoService (gestión de stock, validación precios)
- FacturaService (cálculo de impuestos, totales, número de factura, transacciones)
- AuthService (autenticación JWT, roles, permisos)
Usa entidades
Dominio
Entidades principales:
- Cliente, Producto, Factura, Usuario
Entidades de soporte:
- FacturaItem, Money
Catálogos referenciales:
- TipoDocumento, MetodoPago, TipoFactura, RolUsuario, Impuesto
Persistencia
Repositorios
- ClienteRepository
- ProductoRepository
- FacturaRepository
- UsuarioRepository
(implementan CRUD en BD)
ORM / SQL
Base de Datos
Tablas principales:
- clientes, productos, facturas
- factura_items, usuarios
Tablas referenciales:
- tipo_documento, metodo_pago
- roles_usuario, impuestos

2.2 Capas detalladas

- **Domain (Modelo de negocio):** contiene las entidades con atributos, objetos de valor (por ejemplo `Money` para representar montos con moneda y precisión) y reglas de negocio (validaciones, invariantes). No conoce detalles de la persistencia.
- **Repository (Persistencia):** interfaz y/o implementación para guardar y recuperar entidades. Puede implementarse con ORM (Hibernate, SQLAlchemy) o con acceso directo SQL. Aquí se describe la estructura de tablas.
- **Service (Lógica de negocio):** orquesta operaciones, aplica reglas, valida y transforma datos entre controllers y repositorios.
- **API (Exposición del servicio):** capas de controladores que exponen endpoints REST. Se encarga del mapeo de DTOs, autenticación básica (JWT) y manejo de errores.

3. JSON Estandarizado

3.1 Esquema de respuesta

```
{

"success": true,

"message": "Descripción breve",

"data": { }

}
```

3.2 Formato de errores

```
{

"success": false,

"error_code": "CLIENT_NOT_FOUND",

"message": "Cliente no encontrado",
```

"details": {

"field": "cliente_id",

"value": "1234"

}

}

3.3 Formato de fechas y números

- **Fechas:** ISO 8601 (ej: `2025-09-09T22:00:00Z`).
- **Números:** usar decimales con dos cifras. Para monedas, representar como string (ej: "119000.00") para evitar problemas de precisión.

3.4 Ejemplos request/response

Crear cliente (POST /api/clientes)

Request:

{

"nombre": "Empresa X",

"tipo_documento_id": 1,

"documento": "900123456-7",

"direccion": "Cll 10 # 20-30",

"telefono": "3120000000",

"email": "contacto@empresa.com"

}

Response:

{

"success": true,

"message": "Cliente creado",

"data": { "id": "uuid" }

}

Crear factura (POST /api/facturas)

Request:

{

"cliente_id": "uuid-cliente",

"metodo_pago_id": 2,

"tipo_factura_id": 1,

"items": [

{ "producto_id": "uuid-prod-1", "cantidad": 2 },

{ "producto_id": "uuid-prod-2", "cantidad": 1 }

]

}

Response:

{

"success": true,

"message": "Factura creada",

"data": {

"id": "uuid-factura",

"numero": "202509-0001",

"total": "119000.00"

```
}

}
```

4. Elección de Arquitectura de Servicio Web

Elección: REST.

Justificación:

- **Simplicidad:** REST usa HTTP estándar y JSON, lo que facilita desarrollar clientes web y móviles.
- **Rendimiento:** JSON es más ligero que XML/SOAP; menor sobrecarga.
- **Interoperabilidad:** APIs REST son ampliamente soportadas y fáciles de consumir.
- **Necesidad del proyecto:** Factum requiere integraciones ágiles y consumo por frontend JS → REST es lo más adecuado.

SOAP solo se usaría si fueran obligatorios contratos WSDL muy estrictos o WS-Security.

6. Conclusiones

- Se logró definir un caso de uso práctico (Factum) aplicando arquitectura en capas.
- La separación clara de **Domain, Repository, Service y API** facilita mantenibilidad y escalabilidad.
- El uso de JSON estandarizado asegura interoperabilidad con distintos clientes.
- REST es la mejor opción por simplicidad, compatibilidad y rendimiento en este contexto.
- El proyecto permitió comprender cómo documentar un sistema completo desde el modelo de negocio hasta la exposición de servicios web.