

Desarrollo de Software Multiplataforma

App Web
Orientadas a Servicios
Saber

Ap Am Nombre

MAYL. David Belmares

Unidad # 1

Introducción al Desarrollo Web
Orientado a Servicios

Fecha de Entrega:

jueves, 22 de enero del 2026

UTC
5 DSM A

Enero – Abril del 2026

Proyecto: Portafolio de Evidencias (Aplicación Web)

Objetivo:

Diseñar y desarrollar una aplicación web que funcione como portafolio de evidencias, donde se presenten los contenidos vistos en la materia de Desarrollo Web Orientado a Servicios, aplicando buenas prácticas de diseño y desarrollo web.

Requisitos generales

- El proyecto deberá desarrollarse como una **Aplicación Web**.
- Para el diseño del **index (página principal)** se permite el uso de:
 - **CSS**
 - **Frameworks Frontend** (Bootstrap, Tailwind, etc.)
 - **Herramientas de Inteligencia Artificial** como apoyo al desarrollo.
- La aplicación deberá estar **alojada en un servicio de hosting** y se deberá proporcionar la **URL pública**.

Estructura de la aplicación

La **página principal (index)** deberá de incluir los siguientes elementos:

1 - Encabezado (Header)

Debe aparecer en la parte superior e incluir:

- **Nombre del proyecto**
- **Nombre del alumno**
- **Carrera, grupo y materia**
- **Logotipo** (opcional)
- Menú de navegación principal

2 - Menú de navegación

Debe permitir acceder fácilmente a las secciones del portafolio:

- Introducción al Desarrollo Web Orientado a Servicios
- Implementación de API
- Desarrollo de API
- Implementación
- (Opcional) Contacto o Acerca de

☞ Puede ser un **navbar** hecho con CSS o algún **framework frontend** (Bootstrap, Tailwind, etc.).

3 - Sección de bienvenida

Breve introducción que explique:

- El propósito del portafolio
- Qué aprenderá o encontrará el usuario
- Objetivo del proyecto

Ejemplo:

“Este portafolio presenta las evidencias del desarrollo de una aplicación web orientada a servicios, aplicando conceptos de arquitectura SOA y uso de APIs.”

4 - Secciones o tarjetas de contenido

Mostrar **4 opciones principales** usando:

- Tarjetas (cards)
- Botones
- Iconos
- Imágenes representativas

Cada tarjeta debe incluir:

- Título de la unidad
- Breve descripción
- Botón o enlace para acceder

Las secciones son (**Unidades**):

1. Introducción al Desarrollo Web Orientado a Servicios
2. Implementación de Interfaz de Programación de Aplicaciones (API)
3. Desarrollo de una Interfaz de Programación de Aplicaciones (API)
4. Implementación

Cada opción deberá dirigir a una sección o página específica dentro del portafolio.

5 - Pie de página (Footer)

Ubicado al final del index, debe incluir:

- Nombre del alumno
- Universidad
- Materia
- Año o cuatrimestre
- Derechos reservados (opcional)
- Enlace al repositorio o hosting (si aplica)

Contenido de la Primera Unidad

La opción “**Introducción al Desarrollo Web Orientado a Servicios**” deberá mostrar información clara y organizada sobre los siguientes temas:

1. *Paradigma del desarrollo de aplicaciones orientadas a servicios*

- **Distinguir los servicios que se ofrecen en la nube.**

Servicios que se Ofrecen en la Nube

La **computación en la nube** proporciona recursos tecnológicos a través de Internet, permitiendo acceder a servicios sin necesidad de infraestructura física propia.

◆ **Tipos de Servicios en la Nube**

Los servicios en la nube se distinguen principalmente en **tres modelos**, según el nivel de control y responsabilidad del usuario.

1 IaaS – Infraestructura como Servicio

(Infrastructure as a Service)

¿Qué ofrece?

- Servidores virtuales
- Almacenamiento
- Redes
- Sistemas operativos

Características:

- El proveedor gestiona el hardware
- El usuario administra el sistema operativo y las aplicaciones
- Alta flexibilidad y escalabilidad

Ejemplos:

- Amazon EC2
- Google Compute Engine
- Microsoft Azure Virtual Machines

¿Para quién es ideal?

- Administradores de sistemas
- Proyectos que requieren control total del entorno

2 PaaS – Plataforma como Servicio

(Platform as a Service)

¿Qué ofrece?

- Entorno completo para desarrollar aplicaciones
- Herramientas de desarrollo
- Bases de datos
- Middleware

Características:

- No se gestiona hardware ni sistema operativo
- El desarrollador se enfoca solo en el código
- Reduce tiempos de desarrollo

Ejemplos:

- Google App Engine
- Heroku
- Microsoft Azure App Services

¿Para quién es ideal?

- Desarrolladores de software
- Equipos que trabajan con aplicaciones web

3 SaaS – Software como Servicio

(Software as a Service)

¿Qué ofrece?

- Aplicaciones listas para usar vía Internet
- No requiere instalación local

Características:

- Acceso desde navegador
- Actualizaciones automáticas
- Pago por suscripción o uso

📌 Ejemplos:

- Google Drive
- Microsoft 365
- Gmail
- Dropbox

📌 ¿Para quién es ideal?

- Usuarios finales
- Empresas que necesitan soluciones rápidas

- **Identificar las** características de las aplicaciones orientadas a servicios.

◆ Características de las Aplicaciones Orientadas a Servicios (SOA)

Las **aplicaciones orientadas a servicios** se basan en la arquitectura **SOA (Service-Oriented Architecture)**, donde las funcionalidades del sistema se dividen en **servicios independientes** que se comunican entre sí a través de una red.

✳ Principales Características

1 Servicios independientes

- Cada servicio funciona de manera autónoma.
- Puede desarrollarse, modificarse o reemplazarse sin afectar a otros servicios.
- Facilita el mantenimiento y la evolución del sistema.

2 Interoperabilidad

- Los servicios pueden comunicarse aunque estén hechos en distintos lenguajes o plataformas.
- Usan estándares abiertos como **HTTP, XML, JSON, SOAP o REST**.

3 Reutilización de servicios

- Un mismo servicio puede ser usado por diferentes aplicaciones.
- Reduce duplicación de código y costos de desarrollo.

4 Bajo acoplamiento

- Los servicios no dependen directamente unos de otros.
- Se comunican mediante interfaces bien definidas.
- Permite mayor flexibilidad y escalabilidad.

5 Descubrimiento de servicios

- Los servicios pueden registrarse y ser localizados fácilmente.
- Facilita la integración de nuevos servicios al sistema.

6 Escalabilidad

- Los servicios pueden escalar de forma individual según la demanda.
- Optimiza el uso de recursos del sistema.

7 Seguridad

- Cada servicio puede tener mecanismos propios de seguridad.
- Uso de autenticación, autorización y cifrado.

8 Composición de servicios

- Varios servicios pueden combinarse para crear procesos más complejos.
 - Permite construir aplicaciones completas a partir de servicios pequeños.
-
- **Identificar el concepto y las características de las aplicaciones web híbridas (Mashup).**

Aplicaciones Web Híbridas (Mashup)

Concepto y Características

Concepto de Aplicaciones Web Híbridas (Mashup)

Las **aplicaciones web híbridas** o **Mashup** son aplicaciones que **combinan datos, servicios o funcionalidades de múltiples fuentes web** (internas y externas) para crear una **nueva aplicación con mayor valor funcional**.

Estas aplicaciones suelen integrar:

- **APIs**
- **Servicios web**
- **Datos de diferentes plataformas**
en una sola interfaz.

❖ *Ejemplo:*

Una aplicación que combina **Google Maps**, datos de tráfico en tiempo real y ubicaciones de restaurantes.

✳ Características de las Aplicaciones Web Híbridas (Mashup)

1 Integración de múltiples fuentes

- Unen información proveniente de diferentes servicios web.
- Pueden usar datos internos y externos.

2 Uso de APIs y servicios web

- Se apoyan en APIs REST o SOAP.
- Consumen servicios externos como mapas, redes sociales o bases de datos públicas.

3 Arquitectura flexible

- Permiten agregar o quitar servicios sin modificar toda la aplicación.
- Facilitan la evolución del sistema.

4 Reutilización de servicios

- Aprovechan servicios existentes.
- Reducen costos y tiempo de desarrollo.

5 Interfaz web unificada

- Presentan la información integrada en una sola interfaz para el usuario.
- Mejoran la experiencia de usuario.

Orientadas a la web

- Funcionan a través de navegadores web.
- Utilizan tecnologías como HTML, CSS y JavaScript.

Actualización dinámica de datos

- La información se actualiza en tiempo real o casi en tiempo real.
- Ideal para aplicaciones que dependen de datos externos.

Escalabilidad

- Pueden crecer agregando más servicios o fuentes de datos.
- Aprovechan infraestructuras en la nube.

2. Arquitectura Orientada a Servicios (SOA)

- Definir qué es la **Arquitectura Orientada a Servicios (SOA)**.

Arquitectura Orientada a Servicios (SOA)

Definición

La **Arquitectura Orientada a Servicios (SOA, por sus siglas en inglés: Service-Oriented Architecture)** es un **estilo de arquitectura de software** en el que las funcionalidades del sistema se organizan como **servicios independientes**, reutilizables y accesibles a través de una red, los cuales se comunican entre sí mediante **interfaces bien definidas y estándares abiertos**.

Cada servicio representa una **función específica del negocio** y puede ser utilizado por diferentes aplicaciones, sin importar el lenguaje de programación o la plataforma en la que esté implementado.

Elementos clave de SOA

- **Servicios**: Unidades funcionales independientes.
- **Interfaces**: Definen cómo se accede al servicio.
- **Mensajes**: Medio de comunicación entre servicios.
- **Estándares**: HTTP, XML, JSON, SOAP, REST.

Objetivo principal

Permitir la **integración flexible de sistemas**, facilitando:

- Reutilización de servicios
- Escalabilidad
- Mantenimiento
- Interoperabilidad entre plataformas

Ejemplo sencillo

Un sistema de ventas donde:

- Un servicio gestiona clientes
- Otro servicio procesa pagos
- Otro servicio controla inventario

Todos funcionan de forma independiente pero colaboran entre sí.

- Identificar los **principios de diseño** aplicados a cada servicio modelado.

Principios de Diseño Aplicados a Cada Servicio Modelado (SOA)

Los **principios de diseño SOA** guían la creación de servicios bien estructurados, **reutilizables, escalables y mantenibles**, asegurando que cada servicio cumpla una función clara dentro del sistema.

Principales Principios de Diseño SOA

Contrato de Servicio Estándar

- Cada servicio define claramente cómo puede ser utilizado.
- El contrato especifica entradas, salidas, reglas y formatos de mensajes.
- Permite que otros sistemas consuman el servicio sin conocer su implementación.

Ejemplo:

Un servicio de pago expone un contrato con el método `procesarPago(monto, método)`.

2 Bajo Acoplamiento

- Los servicios dependen lo menos posible entre sí.
- Cambios internos no afectan a otros servicios.
- Mejora la flexibilidad del sistema.

Ejemplo:

Cambiar la base de datos de un servicio sin modificar los clientes que lo consumen.

3 Alta Cohesión

- Cada servicio realiza una función específica y bien definida.
- Evita servicios demasiado grandes o con múltiples responsabilidades.

Ejemplo:

Un servicio solo para autenticación y otro solo para facturación.

4 Reutilización del Servicio

- Un servicio puede ser utilizado por múltiples aplicaciones.
- Reduce duplicación de funcionalidades.

Ejemplo:

Un servicio de validación de usuarios usado por web y móvil.

5 Autonomía

- Cada servicio controla su propia lógica y datos.
- Puede evolucionar de manera independiente.

Ejemplo:

Un servicio de inventario que administra su propia base de datos.

6 Descubridabilidad

- Los servicios pueden ser fácilmente encontrados y entendidos.
- Se documentan y registran adecuadamente.

Ejemplo:

Servicios documentados con OpenAPI / Swagger.

7 Abstracción

- Se oculta la lógica interna del servicio.
- Solo se expone lo necesario mediante la interfaz.

💡 Ejemplo:

El consumidor no conoce cómo se calcula un descuento.

8 Interoperabilidad

- Los servicios pueden comunicarse entre plataformas distintas.
- Uso de estándares abiertos.

💡 Ejemplo:

Un servicio Java consumido por una aplicación en Python.

- Identificar los **estándares relacionados con los servicios**, tales como:

1 XML (eXtensible Markup Language)

💡 ¿Qué es?

Lenguaje de marcado utilizado para **estructurar, almacenar y transportar datos**.

💡 Características:

- Formato legible por humanos y máquinas
- Independiente de plataforma
- Permite definir etiquetas personalizadas

💡 Uso en servicios:

- Formato de intercambio de datos
- Base de otros estándares como SOAP y WSDL

2 SOAP (Simple Object Access Protocol)

¿Qué es?

Protocolo para el intercambio de información estructurada entre servicios web.

Características:

- Basado en XML
- Usa HTTP, SMTP u otros protocolos
- Comunicación estricta y formal

Uso en servicios:

- Integración de sistemas empresariales
- Servicios que requieren alta seguridad

3 WSDL (Web Services Description Language)

¿Qué es?

Lenguaje basado en XML que **describe un servicio web**.

Características:

- Define operaciones, mensajes y ubicación del servicio
- Permite a los clientes saber cómo consumir el servicio

Uso en servicios:

- Documentación automática de servicios SOAP
- Facilita la interoperabilidad

4 UDDI (Universal Description, Discovery and Integration)

📌 ¿Qué es?

Registro o directorio donde se **publican y descubren servicios web**.

📌 Características:

- Permite buscar servicios disponibles
- Almacena información técnica y de negocio

📌 Uso en servicios:

- Descubrimiento de servicios en SOA
- Integración entre organizaciones

5 REST (Representational State Transfer)

📌 ¿Qué es?

Estilo arquitectónico para el diseño de servicios web.

📌 Características:

- Usa HTTP (GET, POST, PUT, DELETE)
- Generalmente usa JSON o XML
- Ligero y flexible
- Sin estado (stateless)

📌 Uso en servicios:

- APIs modernas
- Aplicaciones web y móviles

Evidencias

En esta sección deberá de incluir capturas de pantallas de su portafolio , anexar entre 6 y 10 imágenes que muestren la evidencia de su App Web.