**WIS Architecture**

## **Grupo C1.048**



## **Integrantes del grupo**

* Basallote Braza, David: [davbasbra@alum.us.es](mailto:davbasbra@alum.us.es)
* Chaves Cumbreras, David: [davchacum@alum.us.es](mailto:davchacum@alum.us.es)
* González Vázquez, Guillermo: [guigonvaz@alum.us.es](mailto:guigonvaz@alum.us.es)
* Herrera Luján, Marco Antonio (manager): [marherluj@alum.us.es](mailto:marherluj@alum.us.es)
* Rodríguez Muñoz, Rafael: [rafrodmunn@alum.us.es](mailto:rafrodmunn@alum.us.es)

## **Repositorio**

<https://github.com/DP2-IIS-C1048/Acme-ANS-D>

04/03/2025

**ÍNDICE**

[Grupo C1.048 1](#_Toc1509476631)

[Integrantes del grupo 1](#_Toc753843756)

[Repositorio 1](#_Toc503343683)

[Resumen 2](#_Toc1162676071)

[Tabla de versiones 2](#_Toc543419420)

[Introducción 2](#_Toc1369659904)

[Arquitectura del Software 3](#_Toc747524268)

[Ventajas 3](#_Toc107170609)

[Diseño de la Arquitectura 3](#_Toc1838234900)

[Elementos del Diseño 3](#_Toc1802421784)

[Elementos que influyen en el diseño 4](#_Toc1169885556)

[Notaciones 4](#_Toc1225685180)

[Escalabilidad 4](#_Toc1192181093)

[Estilos y Patrones Arquitectónicos 5](#_Toc1569289179)

[Estilos Arquitectónicos 5](#_Toc444589900)

[Patrones arquitectónicos 7](#_Toc1017953194)

[Arquitectura de Microservicios (Tema 5 AISS) 8](#_Toc362933527)

[Arquitectura de Capas 9](#_Toc17473293)

[División de la arquitectura de capas 9](#_Toc1409275091)

[Capa de aplicación 10](#_Toc1858686686)

[Capa de la lógica de negocio 10](#_Toc825288310)

[Capa de recursos 10](#_Toc857798324)

[Organización de la capa de presentación 10](#_Toc1629077281)

[Organización de la capa de negocio 10](#_Toc580585471)

[Organización de la capa de recursos 10](#_Toc987076957)

[Integración Software 11](#_Toc89681724)

[Definición de integración de aplicaciones 11](#_Toc527985543)

[Mecanismo de integración Web 11](#_Toc1074085938)

[Mashups 11](#_Toc553412538)

[Sindicación de contenido 12](#_Toc1575910484)

[Servicios web 12](#_Toc1123324772)

[APIs 12](#_Toc1630023597)

[Conclusión 12](#_Toc157425802)

[Bibliografía 13](#_Toc52422628)

# Resumen

Este documento recopilará toda la información relevante sobre el conocimiento adquirido en asignaturas previas a DP2, como AISS, IISSI2 y DP1. En primer lugar, se abordarán la arquitectura y el diseño de software, así como los estilos arquitectónicos, centrándose principalmente en la arquitectura en capas utilizada durante el proyecto de Diseño y Pruebas I. Finalmente, se tratará la integración web y los distintos mecanismos que esta emplea.

# Tabla de versiones

|  |  |  |
| --- | --- | --- |
| **Versión** | **Cambio** | **Fecha** |
| 1.0.0 | Inicio del documento | 4/03/2025 |
| 1.1.0 | Revisión de todos los apartados por cada miembro y preparación para primera entrega. | DD/03/2025 |

# 

# Introducción

En el presente documento se describe la arquitectura WIS previamente aprendida en asignaturas previas a Diseño y Pruebas II. Concretamente en la asignatura Introducción a la Ingeniería del Software y los Sistemas de Información 2 (IISSI 2), Arquitectura e Integración de Sistemas Software (AISS) y en Diseño y Pruebas I.

Para comenzar, se presentará una introducción a la arquitectura del software, definiéndola como el conjunto de estructuras necesarias para razonar sobre un sistema, incluyendo sus elementos software, relaciones y propiedades. Destacaremos sus ventajas, como la gestión de la complejidad y la reutilización de componentes.

Luego, abordaremos el diseño de software, su rol como primer paso en el desarrollo y su función clave en la identificación y organización de los componentes del sistema. También veremos cómo factores como el contexto, la experiencia y los requisitos influyen en el diseño arquitectónico.

A continuación, hablaremos sobre los estilos arquitectónicos, en donde hablaremos de los estilos Tuberías y Filtros, Pizarra, Publicar-Suscribir y Capas, siendo este último desarrollado con mayor profundidad. Además, hablaremos también el patrón Modelo-Vista-Controlador (MVC) y de la arquitectura de microservicios, ampliamente utilizado en la industria del software.

Finalmente, abordaremos la integración de aplicaciones, un aspecto clave para la comunicación entre sistemas independientes manteniendo un acoplamiento bajo. Exploraremos diversos mecanismos de integración web, como mashups, que combinan datos y funcionalidades de múltiples fuentes, y la sindicación de contenido, que permite reutilizar información mediante formatos estándar como RSS y Atom. Además, analizaremos el rol de los servicios web, tanto los tradicionales basados en SOAP como los más utilizados RESTful, y la importancia de las APIs, fundamentales para la interoperabilidad y reutilización en entornos digitales.

# Arquitectura del Software

Por definición la arquitectura del software es el conjunto de estructuras necesarias para razonar sobre el sistema, lo cual, comprende elementos de software, relaciones entre ellos y las propiedades de ambos.

### Ventajas

La arquitectura del software aporta una presentación del sistema, ayuda a gestionar la complejidad, aumenta la precisión del coste y tiempo, permite analizar el impacto de los cambios, permite identificar partes reutilizables y sirve como referencia para formar a nuevos participantes dentro de un sistema.

## Diseño de la Arquitectura

El diseño es la primera etapa del proceso de diseño software. Es el vínculo crítico entre el diseño y la ingeniería de requisitos, identifica los principales componentes estructurales de un sistema y su relación entre ellos.

El resultado del proceso de diseño arquitectónico es un modelo arquitectónico que describe cómo se organiza el sistema como un conjunto de comunicación de componentes que se comunican entre sí.

### Elementos del Diseño

El diseño de la arquitectura puede incluir:

* Vistas.
* Puntos de variabilidad y extensión.
* Aspectos de seguridad.
* Documentación de decisiones de diseño.
* Patrones y tácticas de diseño utilizadas.
* Análisis de costes de despliegue en la nube.

El elemento más destacado de todos estos son las vistas. Éstas representan un aspecto parcial de una arquitectura software. Cada participante en el desarrollo estará interesado en una o varias vistas.

Un ejemplo de modelo de vistas sería el “4+1” de Kruchten, el cuál propone 5 vistas:

* Vista de desarrollo: Muestra el sistema desde la perspectiva del desarrollador mostrando sus principales componentes y paquetes.
* Vista lógica: Describe como se implementará la funcionalidad del sistema.
* Vista de proceso: Describe la interacción entre los distintos elementos de un sistema en un tiempo de ejecución.
* Vista física: Describe como el sistema será desplegado.
* Vista de escenarios: Describe escenarios de uso de la aplicación.

### Elementos que influyen en el diseño

Los elementos que influyen en el diseño son: El contexto, los requisitos y la experiencia del arquitecto.

* Requisitos: podemos distinguir entre funcionales y no funcionales (atributos de calidad).
* Contexto: Influyen los aspectos del negocio, las tendencias actuales y la tecnología disponible.
* Experiencia del arquitecto: El diseño final de la arquitectura dependerá en gran medida de la experiencia previa del arquitecto y sus conocimientos de diseño.

### Notaciones

Existen distintas notaciones para describir las vistas de la arquitectura, se pueden dividir en:

* Informales:

Son notaciones visuales realizadas con herramientas genéricas. No usan un lenguaje estándar, la vista debe estar explicada en lenguaje natural. Suele ser el primer paso antes de un diseño más detallado (Diagramas de bloque).

* Semi-formales:

Son notaciones estándar con elementos visuales y reglas para la descripción de sistema software. Permiten realizar algunas operaciones de análisis básicas. La notación más conocida es UML.

* Formales:

Son notaciones precisas y bien definidas conocidas como lenguaje de descripción de arquitecturas. Permiten realizar análisis de la arquitectura y generar código a partir de la misma. Se usan poco en la práctica.

### Escalabilidad

La escalabilidad está íntimamente ligada al diseño del sistema, ya que ésta supone un factor crítico en el crecimiento del sistema. Se pueden distinguir 2 tipos de escalabilidad:

* Vertical: Añade más recursos a un solo nodo en particular dentro de un sistema (Añadir memoria o un disco duro a un ordenador).
* Horizontal: Agregar más nodos a un sistema (Añadir un ordenador nuevo a un programa de aplicación para espejo).

# Estilos y Patrones Arquitectónicos

Los patrones arquitectónicos son soluciones generales comúnmente aceptadas para un problema de diseño arquitectónico recurrente que expresa una relación entre un contexto, un problema y una solución.

Los estilos arquitectónicos establecen restricciones sobre la arquitectura de todo el sistema. Cada estilo describe una familia de sistemas que comparten un conjunto amplio de decisiones de diseño.

Estilos y patrones son similares ya que ambos permiten abordar un problema arquitectónico en un determinado contexto y proponen una solución general.

## Estilos Arquitectónicos

* Tuberías y Filtros:

Usado en aplicaciones de procesamiento da datos donde los datos deben ser procesados en varias etapas hasta producir el estado deseado.

El procesamiento de datos se organiza de forma que cada componente de procesamiento (filtro) realice un tipo de transformación y los datos fluyen (tubería) de un componente a otro para su procesamiento.

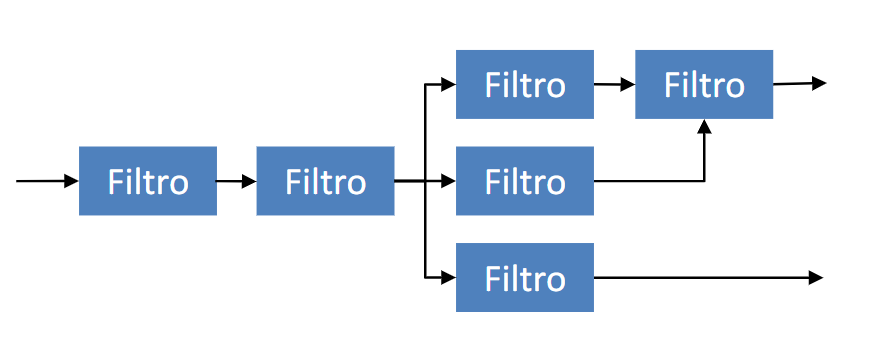


Figura 1: Diagrama de ejemplo del estilo arquitectónico de tuberías y filtros.

* Pizarra:

Cuando varios componentes necesitan compartir y manipular grandes cantidades de datos. Estos datos no pertenecen a ninguno de los componentes exclusivamente.

Todos los datos del sistema se gestionan en un repositorio denominado pizarra accesible a todos los componentes. Los componentes no interactúan directamente, sino tan sólo a través de la pizarra.

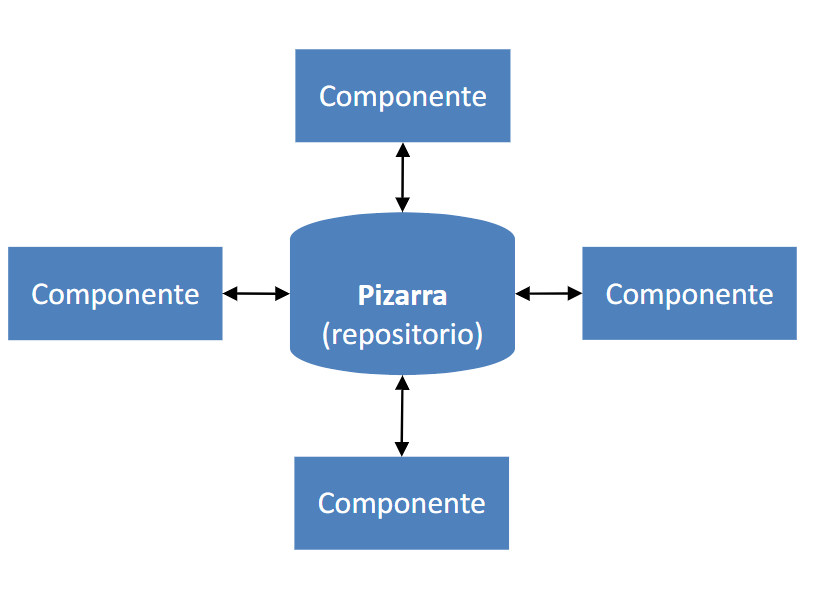


Figura 2: Diagrama de ejemplo del estilo arquitectónico de pizarra.

* Publicar y Suscribir

Cuando existe un número de productores y consumidores da datos que deben interaccionar. El número y la naturaleza de cada uno de ellos, así como de los datos no está predeterminado y puede variar.

Los componentes interactúan a través de mensajes o eventos. Los componentes puedes suscribirse a una serie de eventos. El trabajo de la infraestructura, típicamente un bus de eventos es asegurarse de que cada evento publicado sea entregado a todos los suscriptores de ese evento.

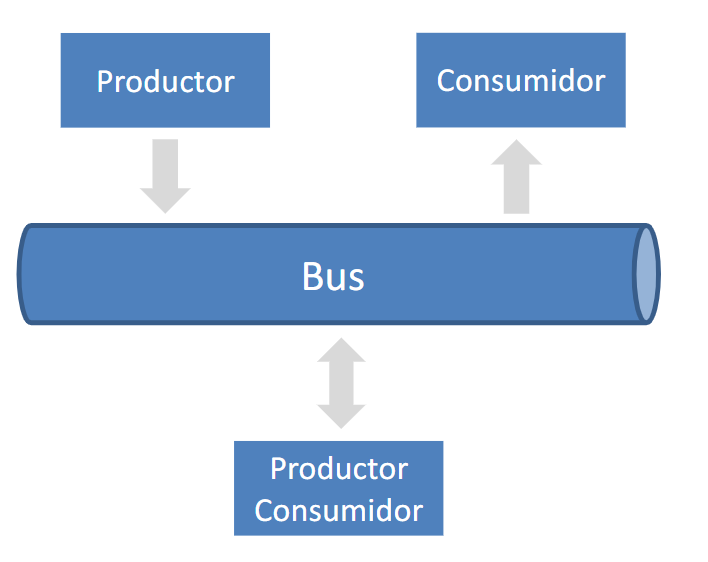


Figura 3: Diagrama de ejemplo del estilo arquitectónico de publicar y suscribir.

* Capas

Usado en aplicaciones donde sea necesario desarrollar y mantener partes del sistema de forma independiente.

La funcionalidad del sistema se organiza en capas. Cada capa agrupa componentes que ofrecen una funcionalidad común, y se apoya en la funcionalidad ofrecida por la capa inmediatamente debajo de ella.

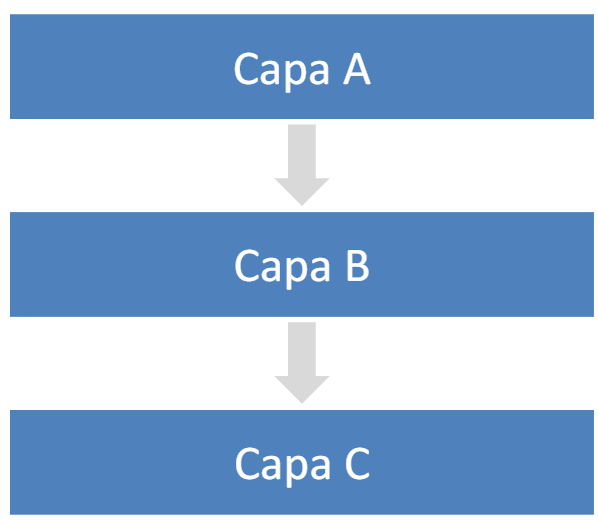


Figura 4: Diagrama de ejemplo del estilo arquitectónico de capas.

## Patrones arquitectónicos

* Modelo-Vista-Controlador:

Separa los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos:

* + Modelo: Es la representación específica de la información con la que se opera. Incluye datos y la lógica para operar con ellos.
  + Vista: Es la presentación del modelo de forma adecuada para interactuar con ella, normalmente a través de una interfaz de usuario.
  + Controlador: Responde a eventos de la interfaz de usuario e invoca cambios en el modelo y probablemente en la vista.

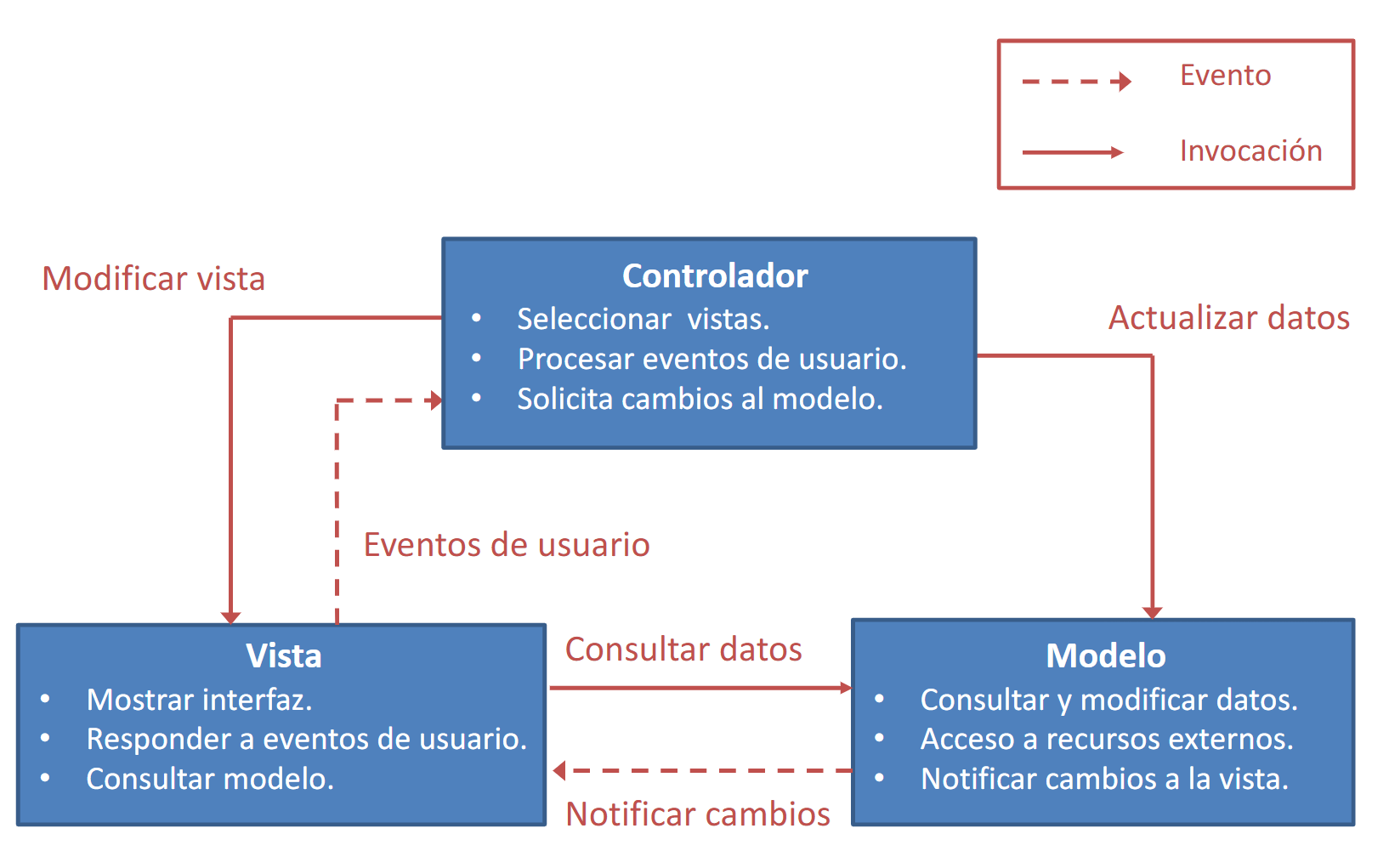


Figura 5: Diagrama de ejemplo del estilo arquitectónico de capas.

# Arquitectura de Microservicios (Tema 5 AISS)

A la hora de desarrollar aplicaciones web, cuando éstas se desarrollan como un sistema monolítico donde es necesario aumentar la escalabilidad del sistema y reducir el tiempo de despliegue se acude a la arquitectura de microservicios.

En este tipo de arquitectura se desarrolla el sistema como un conjunto de servicios pequeños y poco acoplados, corriendo en sus propios procesos y comunicándose a través de mecanismos ligeros, a menudo mensajes HTTP.

Los servicios se organizan de acuerdo con las funcionalidades del negocio y son abordados por equipos multidisciplinares.

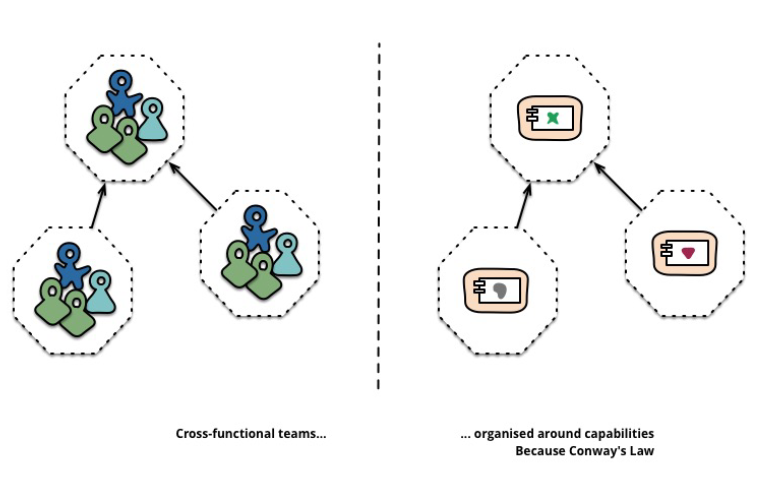


Figura 6: Diagrama de ejemplo de la arquitectura de Microservicios.

El desarrollo se basa en productos, el equipo de desarrollo es responsable de dar soporte al servicio durante todo el ciclo de vida.

Cada servicio es desarrollado con la tecnología más adecuada en cada caso.

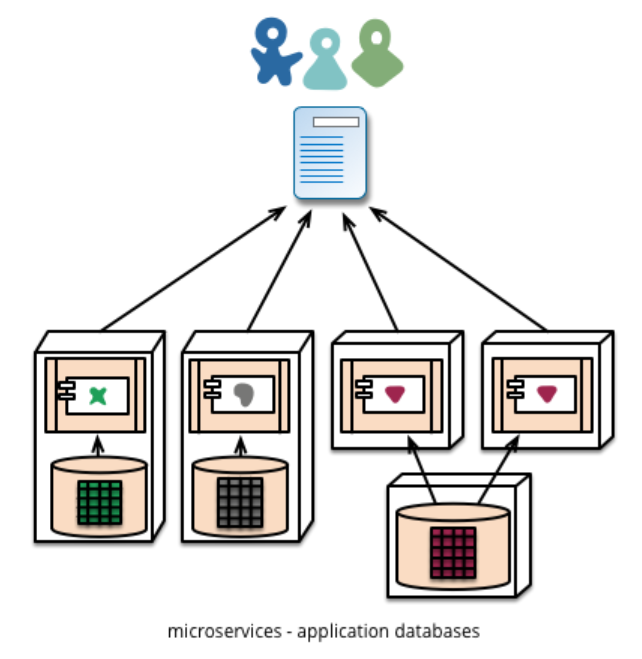


Figura 7: Diagrama de ejemplo de la arquitectura de Microservicios.

# Arquitectura de Capas

La arquitectura en capas es un estilo de diseño que organiza una aplicación en diferentes niveles, donde cada capa tiene una responsabilidad específica y se comunica con las capas adyacentes. Su principal ventaja es que permite la separación de responsabilidades, facilitando el mantenimiento y la escalabilidad del sistema.

## División de la arquitectura de capas

### Capa de aplicación

Responsable de presentar la información y de interactuar con el usuario a través de la interfaz gráfica de usuario.

### Capa de la lógica de negocio

Procesa las solicitudes de la capa de presentación, realiza cálculos y operaciones (por ejemplo, autenticación), y mueve datos entre las capas de presentación y recursos.

### Capa de recursos

Responsable de gestionar los datos: archivos planos, XML, bases de datos, servicios web, etc.

## Organización de la capa de presentación

A su vez, es común organizar la capa de presentación utilizando el Modelo-Vista-Controlador (MVC), combinándolo con la arquitectura en capas para mejorar la separación de responsabilidades.

Por un lado, el Modelo se encarga de recuperar y modificar los datos, además de acceder a recursos externos. Por otro lado, la Vista muestra la interfaz de usuario y responde a los eventos generados por el usuario. Finalmente, el Controlador selecciona qué vistas mostrar, procesa los eventos del usuario y solicita cambios al modelo.

Además, es habitual que la comunicación con el Controlador se realice a través de servicios RESTful, permitiendo una mejor integración con sistemas distribuidos y facilitando la interoperabilidad entre distintas aplicaciones

## Organización de la capa de negocio

La capa de negocio se encarga de procesar las solicitudes provenientes de la capa de presentación, realizar cálculos y gestionar el flujo de datos entre las distintas capas del sistema. Para organizar esta capa, se pueden utilizar diferentes patrones como Transaction Script, Table Module y Domain Model, siendo este último el más recomendado para la asignatura Diseño y Pruebas I debido a su capacidad para manejar lógica de negocio compleja de manera estructurada. Además, se puede incluir una Service Layer para dividir la lógica en dos tipos: la lógica de dominio, que maneja las reglas del negocio, y la lógica de aplicación, que se encarga de tareas como la gestión de transacciones, seguridad y comunicación con sistemas externos.

## Organización de la capa de recursos

Diagrama

El contenido generado por IA puede ser incorrecto.

Figura 7: Diagrama de capas usado en Diseño y pruebas 1.

La capa de recursos es la responsable de la gestión de datos almacenados en bases de datos, archivos, servicios web y otros sistemas de almacenamiento. Para manejar la persistencia de datos, se utilizan patrones como Active Record y Data Mapper, siendo este último el elegido para la asignatura Diseño y Pruebas I por su flexibilidad y desacoplamiento entre objetos y la base de datos. Esta capa hace uso de herramientas y frameworks como JPA (Java Persistence API), Spring Data y Object-Relational Mapping (ORM) para definir entidades, administrar repositorios y ejecutar consultas sobre la base de datos de manera eficiente.

# Integración Software

## Definición de integración de aplicaciones

La integración de aplicaciones permite que sistemas independientes se comuniquen y trabajen juntos, manteniendo un acoplamiento bajo y utilizando, en muchos casos, comunicación asíncrona para combinar funcionalidades y datos con el fin de habilitar nuevos casos de uso.

## Mecanismo de integración Web

### Mashups

Un mashup (o aplicación web híbrida) es una aplicación web que combina datos o funcionalidades procedentes de una o más fuentes para crear un nuevo servicio. La nueva aplicación logra ofrecer un valor añadido a través de un nuevo uso de las aplicaciones originales para el que no fueron concebidas. El contenido y los elementos de presentación suelen provenir de feeds RSS o Atom, servicios web o JavaScript con APIs abiertas. Su entorno natural es la web, y sus principales características incluyen la agregación, la transformación y la visualización de datos.

### Sindicación de contenido

La sindicación web permite que parte del material de un sitio web esté disponible para otros usuarios a través de lenguajes estándar como XML. Los mecanismos de sindicación hacen que la información de un sitio web público sea reutilizable por terceros, permitiendo, por ejemplo, consultar noticias de prensa o entradas de un blog sin necesidad de acceder al sitio web original. Podemos encontrar varios tipos de formatos, como RSS, Atom o GeoRSS.

Gracias a esto es posible integrar información de distintas fuentes web y generar nuevos feeds añadiendo, recortando o transformando los datos. También se puede utilizar la información de las fuentes para procesarla de distintas maneras.

### Servicios web

El W3C (World Wide Web Consortium) define un servicio web como un sistema software diseñado para dar soporte a la interacción interoperable máquina-a-máquina a través de una red. Los servicios web permiten intercambiar información entre aplicaciones software desarrolladas en lenguajes de programación diferentes y ejecutadas sobre distintas plataformas, siendo la base para la integración web. Podemos usar los servicios web no solo para transmitir datos, sino también para hacer uso de la funcionalidad implementada por otros sistemas.

Los servicios web tradicionales (SOAP) son los servicios clásicos definidos por el W3C, y se caracterizan por el uso de WSDL para describir la interfaz pública y SOAP para el intercambio de mensajes en formato XML. Por otro lado, los servicios web RESTful son más recientes, orientados a recursos, y utilizan el protocolo HTTP para manejar un conjunto restringido de operaciones sobre estos recursos, siendo RESTful ampliamente más usado en la actualidad que SOAP.

### APIs

Una API define la forma en la que un programa puede comunicarse con otro programáticamente. Permite que un programa use la funcionalidad y los datos de otros sin necesidad de conocer la estructura interna del programa que implementa la interfaz. Las APIs web suelen implementarse a través de servicios web, permitiendo la comunicación entre aplicaciones mediante protocolos estándar como HTTP. Son clave para la reutilización y la integración.

# Conclusión

Tras discutir los anteriores apartados, se llegó a varias conclusiones.

La arquitectura de un Sistema de Información Web (WIS) es un elemento clave para estructurar y gestionar la complejidad del desarrollo, permitiendo una clara identificación de componentes y sus interacciones, lo que resulta esencial para garantizar la escalabilidad y el mantenimiento del sistema. Es fundamental comprender y aplicar estilos arquitectónicos como Capas, Tuberías y Filtros o Publicar-Suscribir, así como patrones como MVC, adaptándolos al contexto y los requisitos del proyecto para optimizar su diseño y funcionalidad.

Además, la integración web, mediante mecanismos como servicios RESTful, mashups y APIs, se presenta como un pilar crítico para conectar sistemas independientes, favoreciendo la reutilización y la interoperabilidad, aspectos que deben planificarse cuidadosamente para evitar un acoplamiento excesivo.

El diseño arquitectónico y su documentación, incluyendo vistas como las del modelo 4+1, deben definirse con precisión desde el inicio, asegurando que todos los miembros del equipo compartan una visión clara y detallada, lo que facilita la toma de decisiones, reduce imprevistos y contribuye al éxito global del proyecto.

# Bibliografía

* Diapositivas de la asignatura AISS (Temas 2, 4, 5, 6, 7 y 9).
* Diapositivas de la asignatura DP1(Temas 2 y 4).
* Diapositivas de la asignatura ISSI1(Temas 1).

AISS: Tema 2,4,5,6,7

DP1: Tema 2,4

ISSI: Tema 1

AISS: Patrón MVC y Microservicios  
TEMAS 6-9

Diseño de la arquitectura  
Mashup

RESTful APIs

Web Services

Integración de Datos

Integración de Aplicaciones

Portales de Información

Sindicación de Contenido

DP1: Arquitectura de capas

csr