Ejercicio 1

Impulsado por eventos frente a basado en servicios e impulsado por orquestación

Nombre de la Arquitectura	**Impulsado por eventos**	**Basado en servicios e impulsado por orquestación**
Tipo de partición	Arquitectura técnicamente particionada. Los dominios se distribuyen en varios procesadores de eventos y se comunican a través de itermediarios, colas y temas.	Muy enfocado en el particionamiento técnico.
Número de cuantos	Varía de 1 a muchos. Esto se basa en la interacción con la base de datos dentro de cada procesador de eventos y el procesamiento de solicitud-respuesta.	1. Esto es debido a la naturaleza centralizada de esta arquitectura con una a pocas bases de datos y con un mecanismo de orquestación actuando como un punto de acoplamiento gigante.
Despliegue	⭐ ⭐ ⭐ Los servicios pueden ser desplegados de manera independiente y el impacto en los otros servicios puede ser abordado, pero podría, como contrapartida, se incremente considerablemente la complejidad.	⭐ Despliegues extremadamente difíciles que requieren una alta coordinación y pruebas holísticas.
Elasticidad	⭐ ⭐ ⭐ Altamente elástico pero con menos control sobre el flujo de procesamiento.	⭐ ⭐ ⭐ Recibe una calificación media debido a las herramientas proporcionadas por el proveedor para su implementación.
Evolutivo	⭐ ⭐ ⭐ ⭐ ⭐ La independencia de los servicios facilita la implementación de evolutivos del sistema sin causar interferencias entre ellos.	⭐ Demasiado monolítico para despliegues de evolutivos.
Tolerancia a fallos	⭐ ⭐ ⭐ ⭐ ⭐ En términos de independencia del sistema, en caso de una fallo, los componentes restantes del sistema pueden seguir operativos.	⭐ ⭐ ⭐ Comprometido debido a su naturaleza centralizada.
Modularidad	⭐ ⭐ ⭐ ⭐ Mayor modularidad al estructurar eventos de procesamiento en componentes distintos e independientes.	⭐ ⭐ ⭐ Estructurar servicios en diferentes capas facilita el desarrollo de componentes reutilizables, lo que se alinea con el objetivo de reutilización a nivel empresarial.

Nombre de la Arquitectura	Impulsado por eventos	Basado en servicios e impulsado por orquestación
Coste total	⭐ ⭐ ⭐ En el lado negativo tenemos un menor control del flujo de trabajo, menores capacidades de manejo de errores y, posiblemente, gastos de desarrollo y mantenimiento más altos. Por otro lado nos aporta beneficios de una mayor capacidad de respuesta del sistema y capacidad de recuperación mejorada.	⭐ Fuertemente condicionado por el particionamiento técnico intrincado y la alta complejidad, presenta un desafío ante posibles cambios incrementales: todo esto conduce a gastos de desarrollo y mantenimiento elevados.
Rendimiento	⭐ ⭐ ⭐ ⭐ ⭐ Muy optimizado, contamos con comunicaciones asíncronas combinadas con procesamiento paralelo.	⭐ ⭐ Los requerimientos de negocio se subdividen a lo largo de gran parte de la arquitectura.
Fiabilidad	⭐ ⭐ ⭐ Existe cierta contraprestación entre su buena tolerancia a fallos y su bajo control de flujo.	⭐ ⭐ Nuevamente, el particionamiento técnico intrincado puede introducir complejidades que desafían la confiabilidad general del sistema.
Escalabilidad	⭐ ⭐ ⭐ ⭐ ⭐ Manejo muy eficiente de inputs y balanceo gracias a la distribución y procesamiento independientes de eventos, lo que permite una escalado eficiente para acomodar demandas crecientes.	⭐ ⭐ ⭐ ⭐ El objetivo de esta arquitectura no era la escalabilidad, sin embargo, los distribuidores de TI invierten recursos sustanciales para mejorar la escalabilidad de los sistemas de todas las formas posibles.
Simplicidad	⭐ No hay líneas de procesamiento claramente definidas, los flujos son no deterministas y es difícil saber de antemano cómo reaccionará el sistema.	⭐ Simplicidad reducida debido al particionamiento técnico intrincado y su motor de orquestación centralizado.
Testabilidad	⭐ ⭐ Las arquitecturas orientadas a eventos son difíciles de probar debido a la naturaleza impredecible de los flujos de eventos, lo que puede llevar a escenarios de prueba complejos y en constante crecimiento.	⭐ En este sistema arquitectónico, es bastante difícil realizar pruebas aisladas de servicios individuales, debe ser altamente coordinado y holístico.

Proponer un ejemplo de aplicación que encaje en cada uno de los dos estilos arquitectónicos

- **Arquitectura impulsada por eventos:** un caso de uso para este tipo de arquitecturas serían las aplicaciones IoT. Consideremos un sistema de monitorización basado en dispositivos IoT: este tipo de sistema contrará con sensores midiendo la temperatura, la velocidad del viento y otro tipo de medidas. En este sistema, los sensores obtienen métricas, pero cierto evento sólo se disparará si se cumplen ciertas condiciones, como pueden ser un repentino incremento de la velocidad del viento y,

adicionalmente, la temperatura ha atravesado cierto umbral crítico. Con esta arquitectura podemos contar con diferentes procesadores de eventos que emitirán (dispararán) alertas de distintos tipos.

- **Basado en servicios e impulsado por orquestación:** un ejemplo típico de este tipo de esta arquitectura sería un ERP creado ad hoc con una base relacional en el backend y la típica capa de aplicación con módulos para finanzas, compras, ventas, distribución, etc. Pensemos en un ERP de una empresa constructora y, para ilustrar esta arquitectura, en la clase `FixedAsset` que se usa tanto para máquinaria de obra, como puede ser una hormigonera, como un edificio en construcción. Si bien la selección de esta arquitectura nos ha permitido la reutilización de `FixedAsset` a lo largo de la aplicación, en el caso del edificio hay muchísimos datos de la maquinaria (como puede ser la matrícula), que nada aporta en el caso del edificio. Tal y cómo indica el libro de texto de referencia, este tipo de arquitectura no se aplicaría en implementaciones más modernas y es un producto de las limitaciones existentes en términos de licencia y procesamiento en la década de los 90s.

Ejercicio 2

Considerando las premisas y las restricciones del proyecto, el estílo arquitectónico más adecuado para este estilo de proyectos sería el monolítico. Más precisamente sería una ***arquitectura en capas***. Tal y cómo se indica en el libro de referencia de la asignatura: "*La arquitectura en capas es una buena opción para pequeñas y simples aplicaciones o sitios web. Es también una buena elección de arquitecutra, especialmente como punto de partida, para situaciones con un presupuesto muy reducido y restricciones temporales.*"

En la siguiente tabla se analizan requerimientos, restricciones, pros y contras ante la posibile implementación de este tipo de arquitectura para el caso de estudio. Vemos que, a pesar de contar con varios inconvenientes, ninguno incide de forma determinante. Ante una aplicación de este tipo no estamos especialmente interesados en la modularidad y escalabilidad. Si es un inconveniente la posible difícultad de su mantenimiento, sin embargo, esto aplicaría a proyectos más ambiciosos. En el caso de este restaurante familiar probablemente optar por computación distribuida revertiría en más costes tanto iniciales como de mantenimiento, ya que contaríamos con una estructura más potente pero también que implica mayor nivel de complejidad.

Requerimientos	**Restricciones**	**Pros**	**Contras**
Funcionalidad básica	Presupuesto limitado	Simplicidad	Puede ser difícil de mantener
Solución rápida y de bajo coste	Equipo de desarrollo pequeño: 1 UI + 2 PHP	Bajo coste	Falta de modularidad
Expectativas de bajo tráfijo	Infrastructura "On premise"	Encaja con proyectos pequeños con requerimientos básicos	Escalabilidad limitada

Ejercicio 3

De acuerdo con los requerimientos iniciales del proyecto y, especialmente, los nuevos necesidades podemos descartar cualquier tipo de arquitectura monolítica. Cualquiera de las tres arquitecturas monolíticas (en capas, tuberías o micronúcleo) carecen de escalabilidad, esto es debido al alto nivel de acoplamiento entre componentes, haciendo, por lo tanto, realmente difícil la escalabilidad de componentes individuales de forma independiente.

Otro punto es la implementación, dadas las premisas de este proyecto nos estamos enfocando a un despliegue continuo. Cualquiera de las arquitecturas monolíticas puntuan bajo (especialmente la arquitectura en capas), pero incluso en el mejor caso con cualquier arquitectura monolítica, en términos de despliegue, estamos hablando de una punturación media relativamente baja.

Además de lo expuesto en términos de escalabilidad e implementación, destacamos la escasa elasticidad de este tipo de arquitecturas. Nuevamente, en base a los muy requerimientos del proyecto, que son muy diversos y específicos por área, entendemos que esta especificidad será difícilmente atendida por un equipo de desarrollo pequeño, más bien estaremos enfocándonos en equipos de trabajo especializados en dominio y usando técnicas de programación actuales. Por lo tanto, las arquitecturas monolíticas se descartan y, a partir de ahora, nos enfocaremos tan solo en arquitecturas distribuidas, que ofrecen gran escalabilidad, despliege así como adaptación a áreas de trabajo muy especializadas.

Si nos enfocamos en los atributos que buscamos en este proyecto, los principales retos que nos presenta se sitúan en los siguientes atributos:

- **Despliegue**: se requiere implementar/desplegar nuevas funcionalidades muy rápidamente, idealmente sin intervención manual. Los servicios podrían necesitar varios despliegues diarios.
- **Elasticidad**: la variabilidad inherente y las necesidades de escalabilidad del proyecto sugieren que la elasticidad sería muy beneficiosa, por lo que también nos vamos a centrar en este atributo.
- **Evolutivo**: se sugiere la mejora continua, los requisitos del proyecto apuntan a un sistema con alta capacidad de adaptación al cambio y con capacidad para evolucionar con el tiempo.
- **Modularidad**: se sugieren requisitos muy especializados y muy diferentes específicos del dominio, lo que resalta la necesidad de un sistema modular.
- **Escalabilidad**: se mencionan la demanda fluctuante y las variaciones estacionales, así como las necesidades de crecimiento y expansión. Por lo tanto, priorizaremos arquitecturas que sobresalgan en este aspecto.

Exponemos a continuación la puntuación en estrellas de estos atributos para cada una de las arquitectura distribuidas:

Atributo de la arquitectura	**Basada en servicios**	**Impulsado por eventos**	**Basado en espacio**	**Basado en servicios e impulsado por orquestación**	**Microservicios**
Despliegue	⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐	⭐ ⭐ ⭐ ⭐
Elasticidad	⭐ ⭐	⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐
Evolutivo	⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐	⭐ ⭐ ⭐ ⭐ ⭐
Modularidad	⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐
Escalabilidad	⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐

Parece evidente que la arquitectura de microservicios es la mejor opción para este proyecto, especialmente debido al crecimiento de la aplicación y los problemas de rendimiento detectados. La arquitectura de microservicios nos permite fragmentar la aplicación en componentes independientes y escalables. Cada microservicio puede ser desarrollado, desplegado y escalado de forma independiente, lo que también se ajusta al propósito de abordar problemas actuales de concurrencia y sobrecarga.

En general, los microservicios parecen ser la mejor opción según los principales desafíos del proyecto, pero también tenemos algunos aspectos negativos que, en el caso de esta arquitectura, son el **coste**, el **rendimiento** y la **simplicidad**: analicémoslos uno por uno.

- **Coste total**: Aunque el coste es una aspecto negativo significativo de esta arquitectura, vale la pena señalar que este proyecto no especifica ninguna restricción presupuestaria. Por lo tanto, el costo potencialmente más alto de los microservicios podría ser manejable dada la (supuesta) flexibilidad del proyecto en recursos financieros.
- **Rendimiento**: Si bien el rendimiento es un factor crírito en el caso de la elección de microservicios, se trata de un aspecto abordable mediante la implementación de estrategias de optimización dentro de los servicios, como la instalación de ERP para usuarios de oficina, por otro lado, cierta latencia es inherente (y hasta cierto punto esperable por parte de los usuarios) en las operaciones del frontend para usuarios externos.
- **Simplicidad**: A pesar de la mayor complejidad en la gestión de sistemas distribuidos y la comunicación entre servicios, es importante reconocer que los microservicios -aún teniendo en cuenta este factor de considerable complejidad- ofrecen ventajas significativas para el proyecto Photo&Film4You.

Gestionando cuidadosamente la complejidad a través de estrategias efectivas de diseño, implementación y monitoreo, los beneficios de los microservicios pueden compensar con creces los aspectos negativos que presenta esta arquitectura, convirtiéndolos en la opción arquitectónica preferida para el este proyecto.

Ejercicio 4

4.1. Top 3 de falacias de la computación distribuida

Falacia	Ejemplo	¿Porqué se ha seleccionado?
La red es confiable	Supongamos un sistema de comercio electrónico que funciona en una red distribuida, los servidores de tienda o de tienda en línea necesitan comunicarse con un servidor backend que tenga información de disponibilidad de inventario en línea. En un día determinado, debido a problemas en la infraestructura o mantenimiento no planificado, la red entre estos servicios se interrumpe. Como resultado, los clientes que están usando la página experimentan lentitud durante el proceso de compra, ya que la tienda o tienda en línea no puede verificar la disponibilidad de los artículos.	Hoy en día, la computación en la nube está tomando una relevancia enorme, muchas aplicaciones prácticamente no cuentan con versiones 'on-premise'. En este contexto, donde la conectividad de red a menudo se da por sentada pero está lejos de estar garantizada. Las redes constan de múltiples capas, y aunque un proveedor de software o servicio en particular esté operativo, nuestra propia infraestructura de red puede experimentar interrupciones periódicas. Los proveedores de software ofrecen cada vez más soluciones de Punto de Venta (POS) que dependen de sistemas basados en la nube. Estos sistemas vinculan estrechamente las ventas con la confiabilidad de la red, lo cual podría resultar ser un error significativo en caso de que ocurra una interrupción de red imprevista.

Falacia	Ejemplo	¿Porqué se ha seleccionado?
La red es segura	El reciente ciberataque a Air Europa, comprometiendo los datos de tarjetas de crédito de más de 100,000 clientes, sirve como un ejemplo contundente que destaca la falacia de "La red es segura". Como aerolínea, es más que probable que Air Europa haya desplegado sistemas de procesamiento distribuido para gestionar varios módulos como operaciones de vuelo, reservaciones, servicios para pasajeros, además de la necesidad de realizar la distribución del sistema en múltiples ubicaciones. Sin embargo, este incidente subraya la vulnerabilidad de tales sistemas interconectados a brechas de seguridad, enfatizando la importancia de medidas robustas de ciberseguridad en redes distribuidas.	Elijo esta falacia como la top 2 porque tengo la sensación de que es una idea equivocada común. Si hablamos con muchos usuarios promedio, es probable que compartan la misma creencia de que la red es completamente segura y confiable, incluso muchos usuarios alardean de usar contraseñas simples para multiples servicios. Sin embargo, casos como el reciente ciberataque a Air Europa demuestran lo contrario, destacando la necesidad de una mayor conciencia sobre la seguridad cibernética y la implementación de medidas de protección adecuadas en entornos de red distribuida.
Solo hay un administrador	Imaginemos que una empresa desarrolla una API (v1) utilizada por muchos clientes. La empresa anuncia el lanzamiento de la versión v2 con nuevas características, asumiendo la responsabilidad exclusiva de la gestión de la API. Sin embargo, algunos clientes no reciben la actualización a la v2 debido a servicios dentro de nuestro sistema que aún no la han adoptado. Esto crea discrepancias: los clientes se adaptan a la v2, pero los servicios a los que acceden aún no la han implementado.	Hemos de tener en cuenta que los múltiples administradores que existen en una red distribuida representan un reto en términos de seguridad, fiabilidad, trasparencia, gestión, etc. Si bien es cierto que tener multiples administradores puede representar beneficios (como por ejemplo que un solo administrador no compromete todo el sistema), esta falácia hace hincapié en una de las características en las que estos sistemas suelen puntuar bajo y que es la **complejidad**, en este caso la existencia de numerosos administradores implican dificultades adicionales en cuanto a coordinación y gestión.

4.2. Responabilidades de un arquitecto frente a las responsabilidades del equipo de desarrollo

Arquitecto	Equipo de desarrollo
Definir como se gestiona a nivel de infraestructura el incremento de uso durante las campañas de marketing	Realizar un diagrama de estados por los que pasa un pedido.

Arquitecto	Equipo de desarrollo
Definir la mejor tecnología para implementar los procesos.	Crear un prototipo con la interfaz de usuario para verificar la experiencia de los usuarios con la aplicación antes de desarrollarla*
	Automatización de las pruebas.

4.3. Ejemplo de una aplicación que sigue el estílo arquitectónico basado en microservicios pero donde se ve que no es posible maximizar TODAS las características deseables para esta arquitectura

Analizamos la maximización de las características de los microservicios, podemos considerar dos perspectivas:

1. **Aplicaciones que sí se ajustan al estilo arquitectónico de microservicios:** Tomemos como ejemplo la aplicación Photo&Film4You. Si bien los microservicios ofrecen ventajas como la escalabilidad y la modularidad, es importante tener en cuenta los *trade-offs* inherentes. Inicialmente, fragmentamos la aplicación en microservicios para gestionar la autenticación de usuarios, el catálogo, los alquileres, los pagos y la interfaz de usuario. Esto nos brinda un sistema completo y modular, pero conforme la aplicación crece, podemos enfrentarnos a un dilema entre rendimiento y complejidad. Si buscamos maximizar el rendimiento a costa de la complejidad, podemos llegar a un escenario donde la fragmentación excesiva dificulte la gestión y el mantenimiento del sistema. En el caso de Photo&Film4You, es posible que los microservicios propuestos satisfagan las necesidades de rendimiento, sin embargo, es importante tener en cuenta el balance entre escalabilidad y complejidad.
2. **Aplicaciones que no encajan con el estilo arquitectónico de microservicios:** Los microservicios son una arquitectura eficaz para ciertos tipos de aplicaciones, pero pueden no ser adecuados para aplicaciones más tradicionales como los sistemas de gestión empresarial (ERP), de relaciones con los clientes (CRM) o de gestión de almacenes (WMS). Estas aplicaciones suelen manejar datos centrales y únicos, lo que dificulta la maximización de características como la modularidad o la evolutividad. Al fragmentar el sistema en microservicios, aumentamos la complejidad sin obtener los beneficios deseados.

4.4. Expectativas asociadas al rol de arquitecto

El diplomático técnico

En el debate de la actividad se han remarcado las ***soft skills*** y las habilidades de comunicación y negociación por muchos compañeros, remarcando la figura del arquitecto como una suerte de **diplomático técnico**, esto también encaja con la expectativa indicada en el libro de texto de referencia donde se indica que el arquitecto de software debe disponer de habilidades interpersonales. En este línea un arquitecto de software no solo debe ser capaz de seleccionar la distribución y componentes básicos de un sistema sino que será la figura de referencia para explicar de cara a los **stakeholders** porqué se han tomado decisiones técnicas en un determinado sentido.

A modo de ejemplo, de cara a presentar una posible arquitectura ante el consejo de administración de **Photo&Film4You**, esta figura debe ser capaz de buscar los recursos apropiados para explicar porqué se ha elegido el estilo arquitectónico que se ha elegido teniendo en cuenta que en el citado consejo hay profanos

en la materia. La exposición debe ser clara remarcando los pros y contras de la decisión y debe ser percibida por los no técnicos de forma clara y concisa.

El conocimiento del dominio

Otro rol que se espera de un arquitecto de software es que tenga **conocimiento del negocio del dominio específo** en el que se está desarrollando la solución. En este sentido un arquitecto de software no es meramente un técnico que ofrece soluciones tecnológicas indistintamente del sector, debe tener conocimiento de qué problema se está resolviendo y porqué necesita resolverse. Volviendo al ejemplo de **Photo&Film4You**, lo que se intenta resolver en este caso con las herramientas ofrecidas por el arquitecto no es simplemente una plataforma web donde cruzar las demandas de los clientes con la oferta de la empresa, del mismo modo que todos los sectores, el sector de la fotografía tiene ciclos, casuísticas particulares, picos estacionales, etc. No se busca un arquitecto aficcionado a la fotografía, pero idealmente sí debe conocer las particularidades del negocio, si el tema de garantías de los productos es clave, debe saber cómo se suele solucionar esa característica determinada del sector, debe tener *olfato* para los puntos críticos del negocio en particular.