**TALLER ARQUITECTURA DE APLICACIONES WEB**

La arquitectura de aplicaciones web es fundamental para diseñar sistemas eficientes y escalables. Permite que los distintos componentes de una aplicación interactúen de manera coherente.

1. **¿Qué es la Arquitectura de las Aplicaciones Web?**

La arquitectura de aplicaciones web es la organización estructural y la interacción de los componentes (cliente, servidor, datos, almacenamiento, redes y servicios auxiliares) que permiten que una aplicación web funcione, escale y sea mantenible. Define capas, patrones (por ejemplo, separación frontend/backend, APIs), protocolos de comunicación, y decisiones de infraestructura.

1. **¿Por qué es importante la Arquitectura de las Aplicaciones Web?**

La **arquitectura de aplicaciones web es importante** porque establece la base técnica sobre la cual funcionará el sistema. No se trata solo de la programación, sino de **cómo se organizan y comunican los componentes** (frontend, backend, base de datos, almacenamiento, seguridad).

Una buena arquitectura:

* **Garantiza el rendimiento:** los usuarios podrán consultar recetas, subir imágenes o filtrar ingredientes de manera rápida y sin retrasos.
* **Asegura la escalabilidad:** si tu aplicación de recetas se hace popular y recibe miles de visitas diarias, la arquitectura bien diseñada permitirá crecer sin rehacer todo el sistema.
* **Protege la información:** la arquitectura define cómo se implementan protocolos seguros (HTTPS, autenticación), lo que evita robos de cuentas o exposición de datos sensibles de los usuarios.
* **Facilita el mantenimiento y la evolución:** una app de recetas puede necesitar nuevas funciones (comentarios, valoraciones, recomendaciones personalizadas). Con una arquitectura modular, se pueden añadir sin romper lo ya existente.
* **Optimiza costos:** decisiones arquitectónicas (como usar almacenamiento en la nube para imágenes de recetas en lugar de guardarlas en el servidor) reducen gastos de infraestructura.

En resumen, la arquitectura web es importante porque determina **qué tan segura, rápida, confiable y sostenible será la aplicación a largo plazo**.

1. **¿Cuáles son los Componentes de la Aplicación Web?**

**Frontend (cliente):** interfaz del usuario —HTML/CSS/JS, SPA (React, Vue).

**Backend (servidor/aplicación):** lógica de negocio, APIs (Node.js, ASP.NET, Django).

**Base de datos:** persistencia de datos (SQL/NoSQL).

**Almacenamiento de objetos:** imágenes, vídeos, archivos grandes (S3, Spaces).

**API Gateway / Load Balancer:** enruta, balancea carga y proporciona capa de seguridad.

**Autenticación/Autorización:** gestión identidad (OAuth, JWT, Identity Server).

**Cache:** mejorar latencia (Redis, Memcached).

**Colas y procesamiento asíncrono:** tareas en background (RabbitMQ, Kafka).

Monitorización / Observabilidad: logs, métricas, trazas (Prometheus, ELK, Grafana).

CI/CD & Infraestructura como Código: despliegue y reproducibilidad (GitHub Actions, Terraform).

1. **Mencione las Mejores Prácticas para el desarrollo web**

Separación de responsabilidades (UI vs. lógica vs. datos).

APIs RESTful/GraphQL bien definidas y versionadas.

Seguridad por diseño: HTTPS, input validation, limitación de tasa, políticas CORS, gestión segura de secrets.

Automatizar CI/CD para tests y despliegues reproducibles.

Monitoreo y alertas desde el primer despliegue.

Cache inteligente para reducir latencia y coste.

Pruebas: unitarias, de integración y end-to-end.

Diseño para fallos: retries, circuit breakers, timeouts.

Documentación y contratos (OpenAPI, AsyncAPI) para equipos y terceros.

Optimización móvil y first-load (lazy loading, splitting bundles).

1. **Escriba y esplique brevemente los Tipos de Arquitectura de Aplicaciones Web, sus ventajas y desventajas**

**Monolito tradicional**

* Qué es: toda la app (UI, lógica, datos) en un único despliegue.
* Ventajas: simple de desarrollar; despliegue y testing iniciales más sencillos.
* Desventajas: escala mal con equipos grandes; despliegues arriesgados; cuesta introducir cambios a gran escala.

**Monolito modular (modular monolith)**

* Qué es: código único pero organizado en módulos/paquetes.
* Ventajas: equilibrio entre simplicidad y separación; transición hacia microservicios posible.
* Desventajas: aún un solo despliegue; riesgo de dependencias internas.

**Microservicios**

* Qué es: la app se divide en servicios pequeños e independientes (cada uno con su DB idealmente).
* Ventajas: escalabilidad y despliegue independiente, resiliencia, mejores límites de responsabilidad.
* Desventajas: complejidad operativa (observabilidad, transacciones distribuidas, orquestación), coste.

**Serverless / Functions as a Service (FaaS)**

* Qué es: funciones event-driven que se ejecutan bajo demanda (AWS Lambda, Azure Functions).
* Ventajas: coste basado en uso, sin gestionar servidores, escalado automático.
* Desventajas: cold starts, límites de ejecución, arquitecturas complejas para cargas constantes.

**Arquitectura orientada a eventos (Event-driven / CQRS)**

* Qué es: componentes se comunican mediante eventos y colas.
* Ventajas: desacoplamiento, alta escalabilidad y eventual consistency.
* Desventajas: mayor complejidad conceptual, depuración más difícil.

**Arquitectura de micro-frontends**

* Qué es: dividir el frontend en piezas independientes por equipo/feature.
* Ventajas: despliegue independiente, equipos autónomos.
* Desventajas: integración, bundle size y experiencia de usuario si no está bien diseñada.

1. **Escriba Ejemplos de Tecnologías aplicada a:**
   1. **Frontend:**

* React, React, Vue.js, Angular, Svelte.
* Bundlers: Webpack, Vite.
* **Styling:** Tailwind, Sass.
* SSR / Meta frameworks: Next.js (React), Nuxt (Vue).
  1. **Backend:** Node.js (Express, NestJS), ASP.NET Core, Django (Python), Ruby on Rails, Spring Boot (Java).
  2. **Base de Datos:**
* **Relacionales:** PostgreSQL, MySQL, SQL Server.
* **NoSQL:** MongoDB, Redis (también cache), Cassandra.
  1. Almacenamiento de Imágenes: Amazon S3.

1. **Que es un protocolo de comunicación**

Un **protocolo de comunicación** es un conjunto de reglas y formatos que permiten el intercambio de información entre sistemas (formatos de mensaje, secuencia, control de errores, cifrado, etc.). Ejemplos: TCP/IP, HTTP, FTP, TLS.

1. **Explique los protocolos http, https**

* HTTP (HyperText Transfer Protocol): protocolo de capa aplicación usado para solicitar y servir recursos web (GET, POST, PUT, DELETE...). Opera típicamente sobre TCP (puerto 80). No cifra el contenido —por tanto, los mensajes pueden ser leídos o manipulados en tránsito.
* HTTPS: HTTP sobre TLS/SSL — es HTTP protegido por cifrado (puerto 443). Garantiza confidencialidad (cifrado), integridad (no modificación) y autenticidad (certificado que valida servidor). Siempre usar HTTPS en producción.

1. **Que es hosting, investigue los tipos de hosting y haga una tabla comparativa de mínimo 4 proveedores de este servicio, elija uno apropiado para su proyecto**

**Hosting:** es el servicio que pone en línea tu aplicación, almacenando sus archivos y datos en servidores accesibles desde Internet.

* **Shared Hosting:** barato y sencillo, pero limitado en rendimiento.
* **VPS (Servidor Privado Virtual):** equilibrio entre costo y control, recomendado para proyectos pequeños/medianos como tu aplicación de recetas.
* **Cloud Hosting (AWS, GCP, DigitalOcean):** muy escalable y flexible, ideal si esperas muchos usuarios.
* **PaaS (Heroku, Render, Railway):** despliegue muy sencillo, no te preocupas por servidores.

| **Proveedor** | **Tipo principal** | **Pros para tu app de recetas** | **Contras** | **Precio inicial aprox.** |
| --- | --- | --- | --- | --- |
| **Heroku** | PaaS (plataforma gestionada) | Fácil de desplegar el backend de recetas, escalado automático básico. | Planes gratuitos ya no existen permanentes, puede volverse caro. | Desde $5/mes |
| **Render** | PaaS + Hosting estático | Similar a Heroku, pero más barato, excelente para apps pequeñas. | Menos ecosistema que Heroku. | Gratis en pruebas / $7/mes básico |
| **DigitalOcean** | VPS + Managed DB + Spaces | Balance costo/facilidad, buen para alojar backend + base de datos de recetas + imágenes. | Requiere un poco más de configuración. | Desde $4/mes |
| **AWS (Amazon)** | Cloud Hosting + Serverless | Escalabilidad masiva, ideal si esperas millones de usuarios. | Complejidad alta, curva de aprendizaje fuerte. | Pago por uso (más caro si no controlas recursos). |

La opción de hosting elegida para la aplicación web de recetas es **Render**, ya que permite desplegar aplicaciones de forma sencilla sin necesidad de administrar servidores. Ofrece integración con bases de datos, escalabilidad automática y precios accesibles, comenzando desde 7 USD al mes, lo que lo hace ideal para proyectos pequeños y medianos como la aplicación de recetas.

1. **Que es un servidor de Dominio, escojan un dominio para su proyecto e investiga si está disponible, agregue capturas para comprobar su investigación**

**Servidor de dominio (DNS / servidor de nombres):**  
Es el sistema que traduce nombres legibles (ej. miapp.com) a direcciones IP y otros registros (A, AAAA, CNAME, MX, TXT). Un servidor de dominio o nameserver responde consultas DNS y dirige el tráfico a la IP correcta. Los registradores (Namecheap, Cloudflare, etc.) permiten gestionar registros DNS y delegar nameservers.

**el dominio elegido para este proyecto es el de TuRecetaHoy.com, si está disponible.**



