# **Glosario Aplicaciones Moviles**

NOMBRE: Bruno Exequiel Rojas Maldonado

CARRERA: Ingeniería en informática

ASIGNATURA: Aplicaciones Móviles para IoT

PROFESOR: Miguel Acuña

FECHA: 07/10/2024

# **Modelo OSI (Open Systems Interconnection)**

**¿Qué es?** Es un marco de referencia estandarizado que divide las funciones de una red de comunicación en siete capas distintas.

**¿Para qué sirve y se utiliza?**

* **Facilita la interoperabilidad** entre diferentes sistemas y tecnologías de red.
* **Estandariza** las funciones de red para facilitar el diseño y la implementación de sistemas de comunicación.
* **Diagnostica problemas** de red al identificar en qué capa ocurre el fallo.

**Las siete capas del Modelo OSI:**

1. **Capa Física:** Transmisión de bits a través del medio físico.
2. **Capa de Enlace de Datos:** Manejo de errores y control de flujo entre dispositivos directamente conectados.
3. **Capa de Red:** Enrutamiento de datos entre diferentes redes.
4. **Capa de Transporte:** Transferencia de datos de extremo a extremo y control de errores.
5. **Capa de Sesión:** Gestión de sesiones de comunicación entre aplicaciones.
6. **Capa de Presentación:** Traducción y encriptación de datos.
7. **Capa de Aplicación:** Interfaz para aplicaciones y servicios de red.

# **Frontend**

**¿Qué es?** Es la parte del desarrollo web que se encarga de la interfaz y la experiencia del usuario.

**Lenguajes comunes:**

* **HTML:** Estructura del contenido web.
* **CSS:** Estilos y diseño visual.
* **JavaScript:** Interactividad y funcionalidades dinámicas.

**Frameworks populares:**

* **React:** Biblioteca de JavaScript para construir interfaces de usuario.
* **Angular:** Framework de Google para aplicaciones web robustas.
* **Vue.js:** Framework progresivo para construir interfaces de usuario.

**¿Para qué sirve y se utiliza?**

* **Desarrollar la interfaz de usuario** de sitios y aplicaciones web.
* **Garantizar una experiencia de usuario** intuitiva y atractiva.
* **Optimizar la interacción** entre el usuario y el sistema.

# **Backend**

**¿Qué es?** Es la parte del desarrollo web que se encarga de la lógica del servidor, bases de datos y la comunicación entre el servidor y el frontend.

**Lenguajes comunes:**

* **Java:** Lenguaje versátil y robusto.
* **Python:** Fácil de aprender y con múltiples aplicaciones.
* **Ruby:** Conocido por su simplicidad y productividad.
* **PHP:** Popular en el desarrollo web.
* **Node.js (JavaScript):** Permite usar JavaScript en el servidor.

**Frameworks populares:**

* **Django (Python):** Framework de alto nivel que facilita el desarrollo rápido.
* **Spring (Java):** Framework completo para aplicaciones empresariales.
* **Ruby on Rails (Ruby):** Facilita el desarrollo rápido con convenciones.
* **Express (Node.js):** Framework minimalista para aplicaciones web.

**¿Para qué sirve y se utiliza?**

* **Gestionar la lógica de negocio** y las operaciones del servidor.
* **Interactuar con bases de datos** para almacenar y recuperar información.
* **Procesar solicitudes** del frontend y enviar respuestas adecuadas.

# **Repositorio**

**¿Qué es?** Es un sistema para almacenar y gestionar el código fuente y otros archivos de un proyecto de software.

**Tipos principales:**

* **SQL (Bases de Datos Relacionales):** Utilizan tablas y relaciones entre ellas. Ejemplos: MySQL, PostgreSQL, SQL Server.
* **NoSQL (Bases de Datos No Relacionales):** Diseñadas para manejar grandes volúmenes de datos no estructurados. Ejemplos: MongoDB, Cassandra, Redis.

**¿Para qué sirve y se utiliza?**

* **Almacenar y gestionar el código fuente** de manera colaborativa.
* **Controlar versiones** del código para facilitar el seguimiento de cambios.
* **Facilitar la colaboración** entre desarrolladores en proyectos compartidos.

**Herramientas populares de repositorios:**

* **Git:** Sistema de control de versiones distribuido.
* **GitHub:** Plataforma para alojar repositorios Git.
* **GitLab:** Plataforma similar a GitHub con funcionalidades adicionales.
* **Bitbucket:** Otra opción para alojar repositorios Git.

# **API (Interfaz de Programación de Aplicaciones)**

**¿Qué es?** Es un conjunto de definiciones y protocolos que permiten que diferentes aplicaciones se comuniquen entre sí.

**Tipos principales:**

* **REST API (Representational State Transfer):**
  + **Arquitectura basada en recursos.**
  + Utiliza métodos HTTP (GET, POST, PUT, DELETE).
  + Es ligera y fácil de usar con formatos como JSON y XML.
* **SOAP (Simple Object Access Protocol):**
  + **Protocolo basado en XML.**
  + Más complejo y formal en comparación con REST.
  + Soporta operaciones avanzadas y transacciones seguras.

**¿Para qué sirve y se utiliza?**

* **Permitir la comunicación** entre diferentes sistemas y aplicaciones.
* **Facilitar la integración** de servicios y funcionalidades externas.
* **Exponer funcionalidades** de una aplicación para que otras puedan utilizarlas.

# **Microservicios**

**¿Qué es?**

Es una arquitectura de software donde una aplicación se construye como un conjunto de servicios pequeños e independientes.

**¿Para qué sirve y se utiliza?**

* **Mejorar la escalabilidad y flexibilidad** de las aplicaciones.
* **Facilitar el mantenimiento y actualización** de componentes individuales sin afectar al sistema completo.
* **Permitir el desarrollo independiente** por diferentes equipos.

**Características principales:**

* **Descentralización:** Cada microservicio gestiona su propia base de datos.
* **Despliegue independiente:** Cada servicio puede ser desplegado y escalado de manera autónoma.
* **Comunicación ligera:** Generalmente mediante APIs REST o mensajería.

# **HTTP vs HTTPS**

**HTTP (HyperText Transfer Protocol)**

* **¿Qué es?** Protocolo para la transferencia de información en la web.
* **Características:** No cifrado, susceptible a interceptaciones.

**HTTPS (HTTP Secure)**

* **¿Qué es?** Versión segura de HTTP que utiliza SSL/TLS para cifrar la comunicación.
* **Características:** Cifrado de datos, autenticación del servidor, integridad de los datos.

**Métodos HTTP comunes:**

* **GET:** Solicita datos del servidor.
* **POST:** Envía datos al servidor.
* **PUT:** Actualiza datos en el servidor.
* **DELETE:** Elimina datos en el servidor.

**Códigos de respuesta HTTP:**

* **200 OK:** Solicitud exitosa.
* **201 Created:** Recurso creado.
* **400 Bad Request:** Solicitud inválida.
* **401 Unauthorized:** Autenticación requerida.
* **404 Not Found:** Recurso no encontrado.
* **500 Internal Server Error:** Error en el servidor.

**URL vs URI:**

* **URL (Uniform Resource Locator):** Dirección específica de un recurso en la web, incluyendo el protocolo, dominio y ruta. Ejemplo: https://www.ejemplo.com/pagina.
* **URI (Uniform Resource Identifier):** Identificador general que puede referirse a un recurso mediante un nombre o ubicación. Toda URL es una URI, pero no toda URI es una URL.

# **MVC (Modelo-Vista-Controlador)**

**¿Qué es?** Es un patrón de diseño de software que separa una aplicación en tres componentes principales: Modelo, Vista y Controlador.

**Componentes:**

* **Modelo:**
  + **Representa la lógica de negocio** y los datos de la aplicación.
  + Gestiona el acceso a la base de datos y las reglas de negocio.
* **Vista:**
  + **Presenta la interfaz de usuario**.
  + Muestra los datos del modelo al usuario y envía las interacciones del usuario al controlador.
* **Controlador:**
  + **Gestiona la comunicación** entre el modelo y la vista.
  + Procesa las solicitudes del usuario, actualiza el modelo y selecciona la vista adecuada.

**¿Para qué sirve y se utiliza?**

* **Facilita la organización** del código y la separación de responsabilidades.
* **Mejora la mantenibilidad** y escalabilidad de las aplicaciones.
* **Permite el desarrollo paralelo** de componentes por diferentes equipos.

# **POO (Programación Orientada a Objetos)**

**¿Qué es?** Es un paradigma de programación que organiza el software en objetos que contienen datos y métodos.

**Conceptos clave:**

* **Clases:** Plantillas para crear objetos, definiendo sus propiedades y comportamientos.
* **Objetos:** Instancias de clases que representan entidades individuales.
* **Encapsulamiento:** Ocultar los detalles internos del objeto y exponer solo lo necesario.
* **Herencia:** Permite crear nuevas clases basadas en clases existentes.
* **Polimorfismo:** Permite que diferentes clases sean tratadas como instancias de la misma clase base.
* **Abstracción:** Simplificar complejidad al modelar entidades relevantes y ocultar detalles innecesarios.

**¿Para qué sirve y se utiliza?**

* **Organizar el código** de manera modular y reutilizable.
* **Facilitar el mantenimiento** y la expansión de las aplicaciones.
* **Representar entidades del mundo real** de manera más intuitiva en el código.

**Lenguajes de programación orientados a objetos:**

* **Java**
* **C++**
* **Python**
* **C#**
* **Ruby**