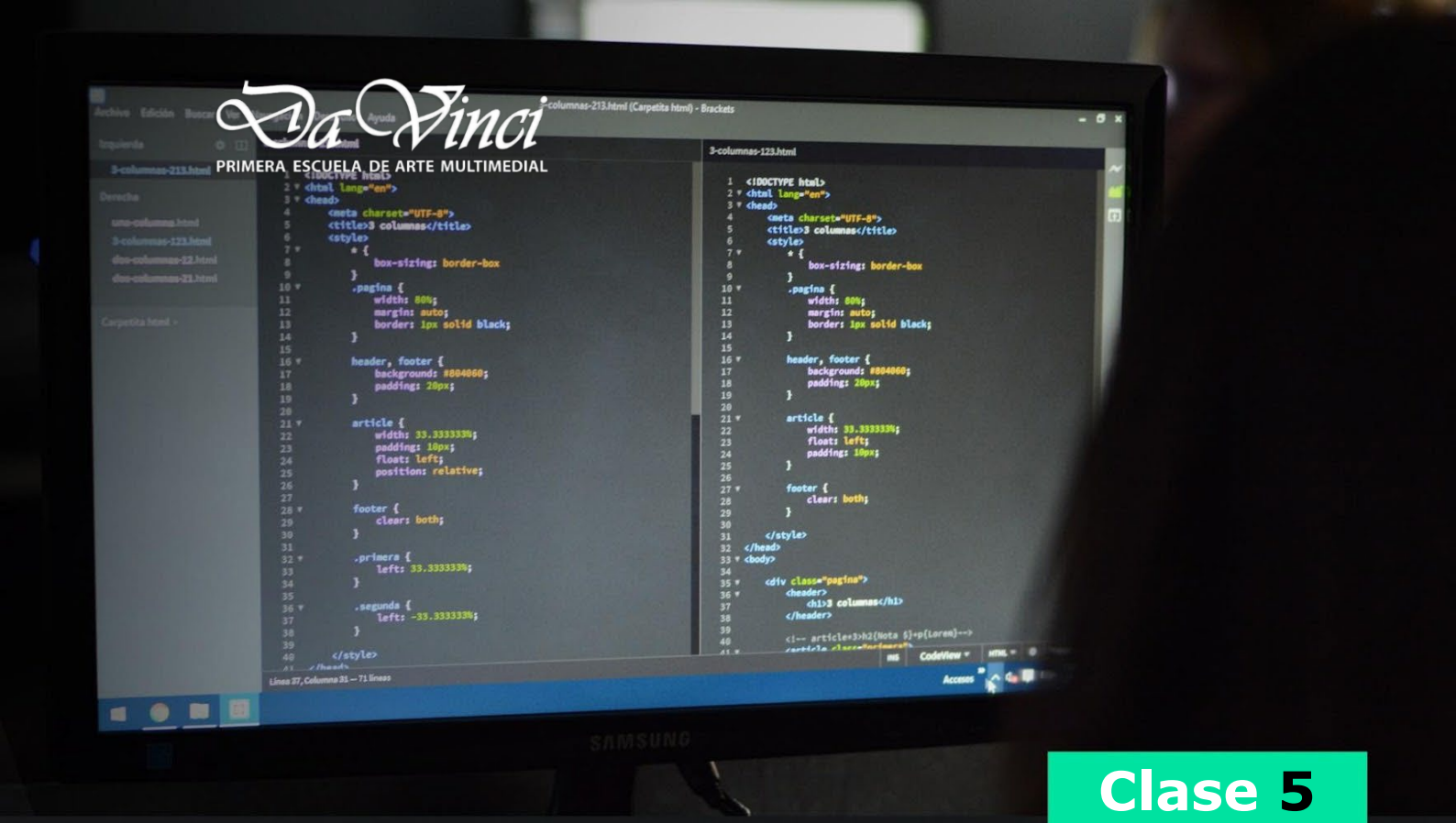

Clase 5

Diseño y Programación Web

Materia:
Aplicaciones Híbridas

Docente contenidista: MARCOS GALBÁN, Camila Belén

APIs

Las API (Application Programming Interfaces) son un conjunto de reglas y definiciones que permiten que dos aplicaciones o componentes de software se comuniquen entre sí. Las API son esenciales en la programación moderna porque permiten que diferentes sistemas intercambien datos y funcionalidad de una manera estandarizada y eficiente.

Estructura de una API

La estructura de una API puede variar según su propósito y la tecnología utilizada, pero generalmente se compone de los siguientes elementos:

Endpoint

La URL que expone la API. Cada endpoint corresponde a una ruta específica de la API y define un recurso concreto.

Métodos HTTP

Los métodos que se utilizan para interactuar con los endpoints. Los métodos más comunes son:

- GET: Recupera información.
- POST: Crea un nuevo recurso.
- PUT: Actualiza un recurso existente.
- DELETE: Elimina un recurso.

Headers

Información adicional enviada con la solicitud, como el tipo de contenido (Content-Type), la autorización (Authorization), etc.

Cuerpo de la solicitud (Request Body)

Datos enviados en una solicitud POST o PUT, generalmente en formato JSON.

Cuerpo de la respuesta (Response Body)

Datos devueltos por la API, también generalmente en formato JSON.

Códigos de estado HTTP

Indicadores del resultado de la solicitud (200 OK, 404 Not Found, 500 Internal Server Error).

Tipos de API

Las APIs pueden clasificarse en varios tipos según su diseño y propósito:

APIs REST (Representational State Transfer)

- Basadas en el protocolo HTTP.
- Utilizan métodos HTTP estándar.
- Son stateless (sin estado).
- Los datos se transfieren normalmente en formato JSON o XML.
- Ejemplo de endpoint: https://api.example.com/users.

APIs SOAP (Simple Object Access Protocol)

- Basadas en XML.
- Más estructuradas y estrictas que REST.
- Utilizan el protocolo HTTP o SMTP.
- Ejemplo de endpoint: https://api.example.com/soap-endpoint.

APIs GraphQL

- Lenguaje de consulta desarrollado por Facebook.
- Permite solicitar solo los datos necesarios.
- Un único endpoint para todas las operaciones.
- Ejemplo de endpoint: https://api.example.com/graphql.

APIs RPC (Remote Procedure Call)

- Permiten la ejecución de procedimientos en un servidor remoto.
- Pueden ser implementadas en JSON-RPC, XML-RPC, etc.
- Ejemplo de endpoint: https://api.example.com/rpc.

API REST

Una API REST (Representational State Transfer) es un estilo arquitectónico para diseñar servicios web. RESTful APIs permiten la interacción entre sistemas utilizando el protocolo HTTP y siguiendo ciertos principios y restricciones que facilitan la escalabilidad, la interoperabilidad y la independencia del sistema.

Principios de una API REST

Client-Server

Separación entre el cliente y el servidor. El cliente no necesita saber nada sobre la lógica de negocio del servidor y viceversa.

Stateless

Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud. El servidor no guarda ningún estado del cliente entre solicitudes.

Cacheable

Las respuestas deben ser definidas como cacheables o no cacheables, para que los clientes puedan almacenar en caché las respuestas y mejorar la eficiencia de la red.

Uniform Interface

Una interfaz uniforme entre componentes que simplifica y desacopla la arquitectura, permitiendo que cada parte evolucione de manera independiente. Esto se logra a través de:

- Identificación de recursos en las solicitudes.
- Manipulación de recursos a través de representaciones.
- Mensajes autodescriptivos.
- Hipermedios como motor del estado de la aplicación (HATEOAS).

Layered System

La arquitectura puede estar compuesta por capas, con cada capa sirviendo una función específica y sin que las capas sepan las funciones de las otras capas.

Code on Demand (opcional)

Los servidores pueden proporcionar código ejecutable al cliente bajo demanda, para extender la funcionalidad del cliente.

Componentes Clave de una API REST

Recursos

Representan entidades del sistema (por ejemplo, usuarios, pedidos, productos) y se identifican mediante URLs.

Métodos HTTP

Se utilizan para realizar operaciones sobre los recursos.

Representaciones

La forma en que los recursos se representan en los mensajes HTTP, generalmente en formatos como JSON o XML.

Códigos de Estado HTTP

Informan sobre el resultado de la solicitud, como 200 OK, 404 Not Found, 500 Internal Server Error, etc.

URI

Una URI (Uniform Resource Identifier) es una cadena de caracteres utilizada para identificar un recurso en Internet. Las URIs son una generalización de las URLs (Uniform Resource Locators) y URNs (Uniform Resource Names). Una URI puede ser tanto un nombre, una ubicación, o ambos, para un recurso.

Estructura de una URI

La URI tiene una estructura jerárquica que generalmente se divide en varias partes:

```
scheme:[//authority]path[?query][#fragment]
```

- scheme: Especifica el protocolo a utilizar (por ejemplo, 'http', 'https', 'ftp', 'mailto', etc.).
- authority (opcional): Incluye la información sobre el servidor (por ejemplo, 'user:password@host:port').
    - 'user:password' (opcional): Credenciales de autenticación.
    - 'host': Dirección del servidor (puede ser un nombre de dominio o una dirección IP).
    - 'port' (opcional): Número de puerto en el servidor.
- path: Ruta al recurso en el servidor.
- query (opcional): Información adicional en forma de pares clavevalor.

- fragment (opcional): Una referencia interna dentro del recurso.

Ejemplo de URI

URL (Una forma específica de URI que localiza un recurso):

```
https://www.example.com:8080/path/to/resource?query=param#section
```

- scheme: 'https'
- authority: 'www.example.com:8080'
    - host: 'www.example.com'
    - port: '8080'
- path: '/path/to/resource'
- query: 'query=param'
- fragment: 'section'

URN (Una forma de URI que nombra un recurso de manera única):

```
urn:isbn:0451450523
```

- scheme: 'urn'
- path: 'isbn:0451450523'

Uso de URIs en APIs REST

En el contexto de las APIs REST, las URIs se utilizan para identificar y acceder a los recursos. Los endpoints de la API se definen utilizando URIs, y los métodos HTTP (GET, POST, PUT, DELETE) se emplean para realizar operaciones sobre esos recursos.

Ejemplo de URIs en una API REST

Consideremos una API que maneja recursos de usuarios. Las URIs podrían estar estructuradas de la siguiente manera:

Obtener todos los usuarios:

URI: 'https://api.example.com/users'

Método: 'GET'

Obtener un usuario específico por ID:

URI: 'https://api.example.com/users/1'

Método: 'GET'

Crear un nuevo usuario:

URI: 'https://api.example.com/users'

Método: 'POST'

Actualizar un usuario existente:

URI: 'https://api.example.com/users/1'

Método: 'PUT'

Eliminar un usuario:

URI: 'https://api.example.com/users/1'

Método: 'DELETE'

Model-View-Controller (MVC)

El patrón de diseño Model-View-Controller (MVC) es un enfoque arquitectónico utilizado para separar la lógica de la aplicación en tres componentes interconectados. Esto ayuda a separar las preocupaciones, facilitando el desarrollo, mantenimiento y escalabilidad del software. En el contexto de una aplicación web utilizando Node.js y Express, cada componente del patrón MVC tiene un rol específico:

- **Model**: Representa los datos y la lógica de negocio. Se encarga de la interacción con la base de datos y la validación de datos.
- **View**: Presenta los datos al usuario. Se encarga de la interfaz de usuario y de la presentación de la información.
- **Controller**: Gestiona la comunicación entre el modelo y la vista. Procesa las solicitudes del usuario, interactúa con el modelo y selecciona la vista adecuada para renderizar la respuesta.

Ejemplo de una aplicación MVC con Node.js y Express

A continuación se muestra un ejemplo básico de cómo estructurar una aplicación utilizando el patrón MVC con Node.js y Express.

```
myapp/
├── controllers/
│   └── userController.js
├── models/
│   └── userModel.js
├── views/
│   └── userView.ejs
├── routes/
│   └── userRoutes.js
├── app.js
└── package.json
```