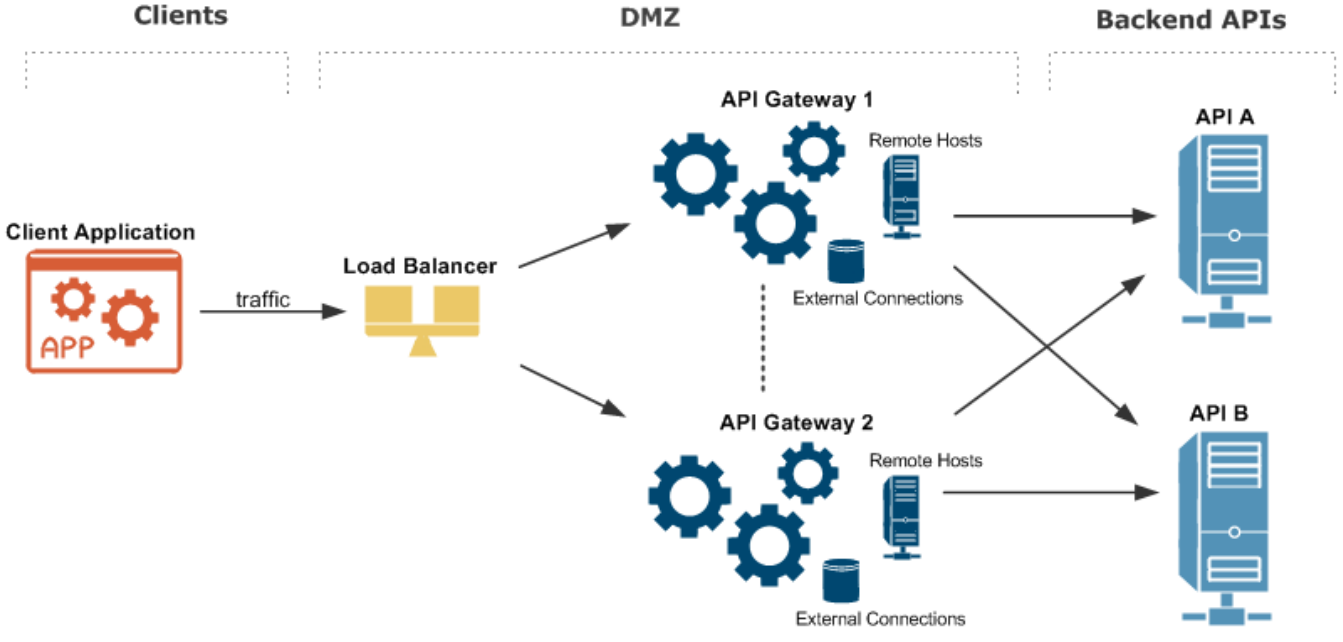
SERVICIOS WEB RESTFUL

PROGRAMACIÓN III

Abdel G. Martínez L.

CONCEPTO: API

- Interfaz de comunicación entre componentes de software.



CONCEPTO: REST

- Transferencia de estado representacional
- Está orientado a la transferencia de recursos
- Arquitectura cliente/servidor sin estado
- Ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes
- Hace uso del protocolo HTTP
- Cada recurso es únicamente accesible a través de su URI
- Los datos son ofrecidos en formato JSON o XML

CONCEPTO: URI

- Transferencia de estado representacional
- Está orientado a la transferencia de recursos
- Arquitectura cliente/servidor sin estado
- Ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes
- Hace uso del protocolo HTTP
- Cada recurso es únicamente accesible a través de su URI
- Los datos son ofrecidos en formato JSON o XML

CONCEPTO: CLIENTE/SERVIDOR

- Transferencia de estado representacional
- Está orientado a la transferencia de recursos
- Arquitectura cliente/servidor sin estado
- Ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes
- Hace uso del protocolo HTTP
- Cada recurso es únicamente accesible a través de su URI
- Los datos son ofrecidos en formato JSON o XML

OPERADORES

GET

- Obtiene información acerca un recurso
- http://ejemplo.com/api/articulos
- http://ejemplo.com/api/articulos/17

POST

- Crea un nuevo recurso
- http://ejemplo.com/api/articulos

PUT

- Actualiza un recurso
- http://ejemplo.com/api/articulos/17

DELETE

- Borra un recurso
- http://ejemplo.com/api/articulos/17

SIMPLEZA DE REST

- Utiliza solamente una URL
- Dicha URL se envía al servidor usando una petición GET
- La respuesta HTTP es la data resultante cruda, sin adicional
- Con REST, una conexión de red es todo lo necesario
- Se puede probar el API directamente, utilizando un navegador web
- Normalmente se utilizan sustantivos en lugar de verbos para denotar recursos de los API

FORMATOS DE SALIDA: XML vs JSON

```xml
<empinfo>
  <employees>
    <employee>
      <name>Scott Philip</name>
      <salary>£44k</salary>
      <age>27</age>
    </employee>
    <employee>
      <name>Tim Henn</name>
      <salary>£40k</salary>
      <age>27</age>
    </employee>
    <employee>
      <name>Long yong</name>
      <salary>£40k</salary>
      <age>28</age>
    </employee>
  </employees>
</empinfo>
```

```json
{ "empinfo" :
    {
        "employees" : [
        {
          "name" : "Scott Philip",
          "salary" : £44k,
          "age" : 27,

        },

        {
          "name" : "Tim Henn",
          "salary" : £40k,
          "age" : 27,
        },

        {
         "name" : "Long Yong",
         "salary" : £40k,
         "age" : 28,

        }
                  ]
    }
}
```

AJAX Y REST

- AJAX es una técnica de desarrollo web popular que hacer las páginas web interactivas utilizando JavaScript.
- En AJAX, las peticiones son enviadas al servidor utilizando objetos XMLHttpRequest. La respuesta es utilizada por el código JavaScript para cambiar dinámicamente la página actual.
- Las aplicaciones AJAX siguen los principios de diseño REST. Cada XMLHttpRequest puede ser visto como una petición de servicio REST, utilizando GET. Usualmente, la respuesta es JSON.

COMPONENTES DE REST

Recursos

- Identificados por URL lógico
  - Implica que los recursos son universalmente identificables por otras partes del sistema
- Representan estado y funcionalidad
- Son el elemento clave de un diseño RESTful

Web de Recursos

- Un recurso individual no debe ser tan grande y no contener detalles muy finos
- Un recurso debe contener enlaces a información adicional
- Es similar al funcionamiento de las páginas web

Cliente-Servidor

- Un componente de un servidor puede ser un componente de un cliente

COMPONENTES DE REST

Sin Estado de Conexión

- La interacción es stateless (a pesar que los servidores y recursos pueden ser stateful).
- Cada petición debe llevar la información necesaria para completarla y no debe depender de las anteriores

Cacheable

- Deben tener fecha de expiración
- Debe permitir al servidor guardar en caché ciertos recursos
- Se puede utilizar la cabecera de control de HTTP

Servidores Proxy

- Mejoran rendimiento y escalabilidad
- Se puede utilizar cualquier proxy HTTP estándar

LINEAMIENTOS EN EL DISEÑO REST

1. No usar URL físicas
   Ejemplo: http://www.acme.com/inventory/product03.xml [NO]
   Ejemplo: http://www.acme.com/inventory/product/003 [SI]
2. Consultas no deben retornar sobrecarga de datos
   Debe mostrar primeros n resultados, con enlaces Siguiente/Anterior
3. Documentar las salidas de las respuestas REST
   Si la salida es XML, se debe documentar en DTD
   Incluso si la salida es human-readable, los clientes al final no son humanos
4. Incluir la URL inicial en la respuesta REST
   La salida es más larga, pero mejor controlada
5. Las peticiones de acceso GET no deben hacer cambios
   En otras palabras, DELETE.

EJEMPLOS

- Google
  - https://console.developers.google.com/apis
- Facebook
  - https://developers.facebook.com
- Twitter
  - https://dev.twitter.com/rest/public
- Ebay
  - https://go.developer.ebay.com

EXTRA: BORRADO DE ELEMENTOS

- Importante: NO se debe exponer enlaces de eliminación, es decir, enlaces de cambio de estado, como simples enlaces web.
- Se debe utilizar el verbo HTTP llamado DELETE.

EXTRA: AUTENTICACIÓN

- Existen dos alcances:
  - HTTP Auth: El usuario ingresará credenciales una sola vez. El software cliente incluirá las credenciales codificadas en cada petición HTTP que se haga al servidor (cabecera "Authorization").
  - Crear servicio de login dedicado: Acepta credenciales y retorna un token. El token se incluye como un argumento URL. Obviamente debe estar bien definido en la creación del API.
- El primer alcance es útil cuando se utiliza contra navegadores web, mientras que la segunda porque se puede mantener expiraciones.
- En estos casos una cookie no sería válida ya que viola la noción REST de transferir los estados, donde el estado debe ser pasado por cada petición.

EXTRA: OBJETOS COMPLEJOS

- La manera más sencilla es crear un constructor para las clases de objeto complejas que acepte un par de llave/valores (diccionario); los cuales deben ser parámetros detallados pasados por el cliente.
- Sin embargo, se considera mejor evitar los chequeos de validación en el constructor. Se pudiese implementar para valores que no sean cadenas (ejemplo: enteros).
- Para evitar riesgos de seguridad, se puede validar cada llave existente en el mapa. Si encuentra un valor idéntico, entonces se aplica el método setter del campo.