Práctica Final

Tecnologias para el desarrollo Web

Santiago Caro Duque [BQ0550]

Implementación de un servicio Web con Frontend
y Backend

Ingeniería Informática
Universidad Politecnica de Madrid
España
Junio 2020

Índice

1. Introducción

Durante la materia de Tecnología de Desarrollo para el Desarrollo Web se ha visto como se puede implementar todo un servicio Web desde cero y cuales son las grandes ventajas de esta tecnología y por que ha sido tan importante para el desarrollo de la humanidad.

La información es definitivamente uno de los activos mas importantes para la sociedad y esto se sabe desde el principio de toda esta historia y es gracias a la necesidad de querer compartir información de la forma más fácil posible es que hoy en día tenemos la red mundial que conocemos como internet.

La mayoría de la información que consumimos regularmente día a día en el internet llega a nosotros gracias a dos componentes básicos. La presentación o también conocida como el "frontend" es toda la parte gráfica que nos ayuda a ver la información de una manera placentera y fácil de recibir.

Dentro de este componente existen también varios tipos de archivos los HTML, estos ponen las "reglas" para indicar en donde se debe presentar la información, básicamente pone un orden a la presentación de la información. Después, están los archivos CSS los cuales indican el cómo se presentará la información, con que tipo de letra o que colores se van a usar o incluso de la forma de los elementos descritos dentro del anterior archivo.

Por ultimo también pueden existir archivos JavaScript los cuales describen el comportamiento de esta información según las acciones que el usuario indique. Todos estos archivos se ejecutan desde el lado del cliente, generalmente desde un navegador de internet como los grandes conocidos los cuales entienden cada uno de estos archivos y los saben presentar al cliente de la manera que ha sido querida por los desarrolladores.

El segundo componente importante para el funcionamiento de esta tecnología es la lógica detrás de todos los datos, también conocida como el "backend". En este caso no existe un lenguaje unificado que sirva solamente para realizar este componente o un lenguaje que se use universalmente para la parte lógica. Aunque si hay unos lenguajes que tienen integradas muchas mas facilidades para el cuando se desarrolla la parte lógica del sistema.

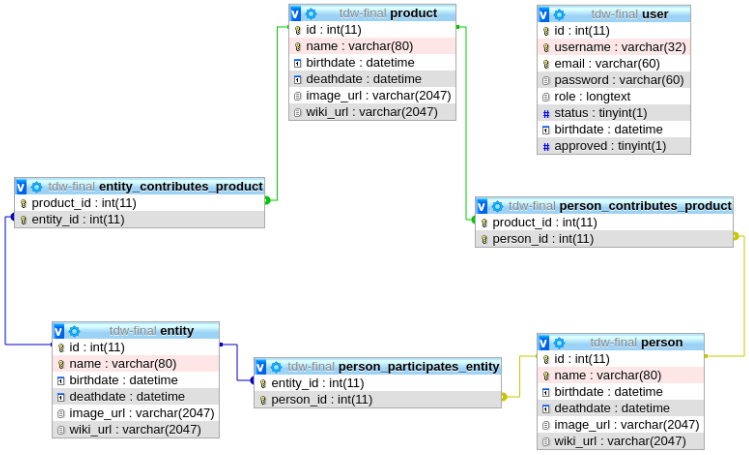
Este componente puede ser el que le parezca un poco mas "extraño" al usuario ya que no lo ve tan tangible dentro de su propio ordenador pero es muy necesario y útil. En el caso de este proyecto se tomó la decisión de hacer una API REST, esto quiere decir que desde las acciones realizadas por el usuario en el frontend por medio de la lógica dada por el Javascript se pueden hacer peticiones a un servidor y este según la información que llegue sabe como responder. El significado de que sea REST es que la lógica que responde va a dar como resultado el estado actual del estado de la información, ya lo dice por su definición en sus siglas en ingles "Representational state transfer". Para el desarrollo de este proyecto también se tomó la decisión de realizar este componente con el lenguaje PHP.

Lo que se requiere como funcionalidad del sistema es el control de la información de Personas, Entidades y Productos los cuales han influido de manera significativa en el desarrollo de las tecnologías de la web y el internet. Para cada uno de estas entidades se van a guardar su información básica así como enlaces de interés como su página en Wikipedia. Así mismo se conoce que pueden existir relaciones entre varias entidades de diferente tipo, por lo tanto también se puede guardar estas relaciones entre las entidades. Además, se requiere que el acceso a esta información sea restringido por medio de nombres de usuario y contraseñas.

Existen dos tipos de usuarios, los lectores los cuales se les es permitido ver toda la información dentro del sistema y ver su propia información. Los Escritores además de ver toda la información también pueden añadir, modificar y eliminar tanto entidades de cualquier tipo como otros usuarios. Cuando se registra un nuevo usuario se le da el rol de lector y solo un escritor tiene los permisos para ascenderlo a rol de escritor. Además no puede acceder a la información hasta que un escritor no lo haya verificado como usuario.

2. Diagramas

2.1. Modelo de Datos



2.1.1. Explicación

En el anterior diagrama se puede ver el modelo de datos de la aplicación desarrollada, cual es su estructura y como se relacionan las diferentes entidades entre si. Lo primero que hay que notar es que las tres entidades principales tienen la misma estructura interna, con los mismos atributos ya que se quiere guardar la misma información de cada uno de ellos. Esta información es:

- **id** Identificador único del elemento de tipo numérico.

- **name** Cadena de caracteres única en su categoría que da el nombre del elemento especifico.
- **birthDate** Fecha de nacimiento o creación del elemento.
- **deathDate** Fecha de muerte o terminación del elemento.
- **image_url** Cadena de caracteres con formato de URL que indica una dirección a una imagen relacionada con el elemento.
- **wiki_url**Cadena de caracteres con formato de URL que indica una dirección a la pagina de Wikipedia del elemento.

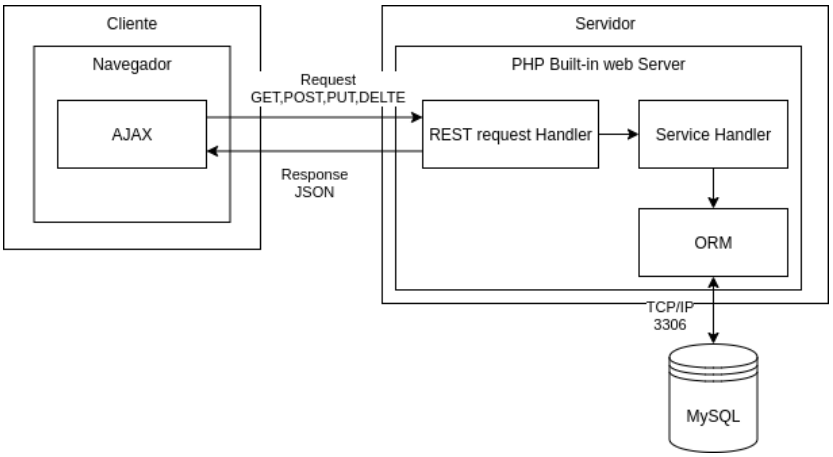
Además entre cada una de las entidades se puede ver que hay una relación de muchos a muchos, ya que pueden existir relaciones entre las entidades y pueden ser múltiples entre ellas. Por lo tanto, existe una tabla intermedia que facilita el vinculo entre las dos otras tablas. Todas tienen la misma estructura la cual es:

- **element1_id** Numero identificador del elemento numero 1.
- **element2_id** Numero identificador del elemento numero 2.

Por ultimo están los datos de los usuarios que pueden acceder al sistema. Se tienen sus datos necesarios para ingresar y otros datos adicionales que sirven tanto por tener información del usuario como para el funcionamiento de la aplicación. Su estructura es la siguiente:

- **id** Numero de identificador único de usuario asignado en el registro.
- **username** Nombre de usuario por medio del cual se puede identificar en el sistema.
- **email** Cadena de caracteres con formato de correo electrónico del usuario.
- **birthDate** Fecha de nacimiento del usuario.
- **password** Cadena de caracteres en formato hash que representa la contraseña del usuario para ingresar al sistema. No se guarda la contraseña como texto plano, se le aplica una función hash que permite su verificación posterior.
- **role** Rol el cual identifica los permisos con los cuales el usuario puede realizar operaciones dentro del sistema.
- **status** Valor Booleano que indica el estado del sistema. En caso de ser 0, este usuario esta desactivado y no puede entrar al sistema. En caso contrario no tiene restricciones para entrar.
- **approved** Valor Booleano que representa si el usuario ya ha sido aprobado por otro usuario de tipo Escritor. Hasta no ser aprobado el usuario no puede realizar operaciones dentro del sistema.

2.2. Arquitectura



2.2.1. Explicación

Esta es la arquitectura de un sistema que funciona con una API REST. Se tiene principalmente una arquitectura Cliente-Servidor, en donde el cliente hace peticiones al servidor y el servidor responde con información.

Desde el lado del cliente se puede ver un componente principal que es el navegador web que es la forma como interactúa el cliente. Dentro de este navegador esta el modulo AJAX el cual se encarga de enviar y recibir las peticiones al servidor. Se puede ver en el diagrama que se envían peticiones con métodos HTTP y el servidor retorna datos en formato JSON.

En la parte del servidor hay también una parte principal que es la ejecución logica del servidor que en este caso se trata del servidor que puede ser desplegado directamente desde un ambiente PHP. Dentro de este servidor hay muchas funcionalidades y paquetes que hacen que funcione de la manera que deberia, pero principalmente se puede dividir en tres partes.

La primera parte es el modulo que maneja las peticiones REST, según la petición que llegue crea el controlador necesario y lo ejecuta. Este controlador es el segundo modulo importante que es el responsable de la parte lógica de la petición. Se encarga de preparar la respuesta que va a ser enviada al cliente y verificar la autenticidad de la petición en caso de ser necesario. Este controlador tendrá que hacer peticiones a la base de datos la cual guarda la información que representa el estado de la aplicación y esto lo hace por medio de un ORM [Object Relation Model] quien es el que se encarga de hacer un mapeo entre las clases lógicas del controlador y las tablas de la base de datos, siendo esta el tercer modulo importante para el funcionamiento de este componente.

Una vez se tiene la información de la base de datos en formato JSON por medio del manejador de peticiones se devuelve esta información al cliente quien

debe tener el código que permite mostrar la información recibida de una forma amigable para el usuario.

3. Especificación de la API

A continuación se describen todos los puntos a los cuales se les puede hacer peticiones por medio de peticiones HTTP. Cada uno tiene su dirección, los métodos que soporta y sus explicaciones.

Existen algunos métodos para los cuales es necesario que en la cabecera de la petición exista un JWT valido para poder ejecutar la petición. Estos métodos están identificados con "**[Identificación Necesaria]** " ó "**[IDN]**" . Un usuario con rol de lector puede ejecutar solamente los métodos de tipo GET que necesiten identificación. Un usuario de tipo escritor puede ejecutar todos los métodos descritos.

3.1. Usuarios

3.1.1. /access_token

POST Dentro del cuerpo de esta petición van las credenciales con las cuales algún usuario quiera acceder al sistema. En caso de que las credenciales sean validas se le asigna un JWT (JSON Web Token) con el cual podrá identificarse para hacer peticiones al sistema. Con estas peticiones puede acceder a la información y en caso que tenga los permisos suficientes modificarla.

3.1.2. /users

GET **[Identificación Necesaria]** El usuario puede obtener la lista completa de usuarios que están registrados dentro del sistema con su información básica, excepto la contraseña.

POST Dentro del cuerpo de la petición va como contenido la información de un nuevo usuario el cual va a ser registrado dentro del sistema. La respuesta del servidor es la representación de como queda registrado este usuario dentro del sistema y cual fue el numero identificador que se le asigno.

OPTIONS Retorna un cuerpo vacio pero dentro de sus cabeceras esta una de tipo "ALLOW" la cual nombra cuales son aquellos servicios que pueden ser utilizados dentro del la API.

EJ: GET,POST,OPTIONS

3.1.3. /users/{user_id}

Parametro {user_id}:

Un numero que identifica a un usuario en especifico. Cada usuario tiene su numero identificador único.

GET [**IDN**] Obtiene toda la información que se tiene dentro del sistema sobre este usuario en especifico en formato JSON, dentro del cuerpo de la respuesta.

PUT [**IDN**] Permite enviar dentro del cuerpo de a petición información que se quiera modificar al usuario especifico.

DELETE [**IDN**] Permite eliminar toda la información almacenada sobre el usuario especifico.

OPTIONS Retorna un cuerpo vacio pero dentro de sus cabeceras esta una de tipo "ALLOW" la cual nombra cuales son aquellos servicios que pueden ser utilizados dentro del la API.

EJ: GET,POST,OPTIONS

3.1.4. /users/username/{username}

GET Permite determinar si el nombre de usuario ya exista, en caso de que no exista retorna un mensaje 404, lo que significa que el nombre de usuario puede ser registrado a un nuevo usuario. En caso de que exista retorna un mensaje 204 y no se puede registrar el nuevo usuario.

3.2. Elementos

En este caso como tenemos 3 entidades que tienen la misma información y se pueden realizar las mismas operaciones sobre todos las entidades, decidí agrupar estas tres partes de la API en una sola por facilidad de lectura y omitir repetición. Por lo tanto en todos los end points que en adelante que se refieran a un ítem, pueden tomar cualquiera de los siguientes valores:

- products → Para el manejo de productos
- persons → Para el manejo de personas
- entities → Para el manejo de entidades

3.2.1. /items

GET [**IDN**] En el cuerpo del retorno devuelve toda la información que tiene el sistema sobre toda la colección de un tipo de entidad en formato JSON.

POST [**IDN**] Sirve para crear una nueva entidad. Dentro del cuerpo de la petición tiene que ir un nuevo objeto describiendo la entidad nueva. Como retorno se devuelve el estado de como queda reflejado el estado de esta nueva entidad dentro del sistema, además de el numero identificador único asignado por el sistema.

OPTIONS Retorna un cuerpo vacio pero dentro de sus cabeceras esta una de tipo "ALLOW" la cual nombra cuales son aquellos servicios que pueden ser utilizados dentro del la API.
EJ: GET,POST,OPTIONS

3.2.2. /items/{item_id}

Parametro {item_id}:
Numero identificador único que representa a un elemento dentro del sistema del tipo que sea el {item}

GET [**IDN**] Retorna dentro del cuerpo de la respuesta en formato JSON el estado actual del elemento que se ha solicitado. En caso de que el elemento con ese numero identificador no este registrado a ninguno retornara un mensaje 404 ya que no lo encontro.

PUT [**IDN**] Permite la modificación del estado del elemento. Dentro del cuerpo de la petición se envía cuales son los valores que se que se quieran modificar. Como respuesta, en el cuerpo de esta retorna el nuevo estado que ha sido guardado dentro del sistema.

DELETE [**IDN**] Permite eliminar toda la información almacenada del elemento.

OPTIONS Retorna un cuerpo vacio pero dentro de sus cabeceras esta una de tipo "ALLOW" la cual nombra cuales son aquellos servicios que pueden ser utilizados dentro del la API.
EJ: GET,POST,OPTIONS

3.2.3. /item/itemname/{itemname}

Parametro {itemname}:
Cadena de caracteres describiendo el nombre de un elemento

GET En caso de que ya exista un elemento dentro de la categoría seleccionada con el nombre que viene por parámetro retorna un mensaje con código 204 y por lo tanto no se puede usar el mismo nombre para registrar un nuevo elemento. En caso de que no exista retorna un mensaje con código 404 y se puede registrar el nuevo elemento.

3.2.4. /item/{item_id}/relation1

Cada elemento de una categoría puede tener relaciones con las elementos de las otras dos categorias. Para evitar repetición las otras dos categorías en este caso van a ser referidas como "relation1" y "relation2".

GET [**IDN**] Retorna la colección de elementos completos de la primera categoría con los cuales se relaciona el elemento identificado con el {item_id}.

3.2.5. /item/{item_id}/relation2

GET [**IDN**] Retorna la colección de elementos completos de la segunda categoría con los cuales se relaciona el elemento identificado con el {item_id}.

3.2.6. /item/{item_id}/relation1/add/{item_id}

PUT [**IDN**] Permite agregar una relación entre el elemento que se esta modificando con un elemento de la primera categoría identificado con el segundo identificador de elemento dado en la URL del endpoint.

3.2.7. /item/{item_id}/relation1/rem/{item_id}

PUT [**IDN**] Permite eliminar una relación entre el elemento que se esta modificando con un elemento de la primera categoría identificado con el segundo identificador de elemento dado en la URL del endpoint.

3.2.8. /item/{item_id}/relation2/add/{item_id}

PUT [**IDN**]Permite agregar una relación entre el elemento que se esta modificando con un elemento de la segunda categoría identificado con el segundo identificador de elemento dado en la URL del endpoint.

3.2.9. /item/{item_id}/relation2/rem/{item_id}

PUT [**IDN**]Permite eliminar una relación entre el elemento que se esta modificando con un elemento de la segunda categoría identificado con el segundo identificador de elemento dado en la URL del endpoint.

4. Problemas durante el desarrollo

Uno de los principales problemas para el desarrollo de este proyecto fue entender la forma en como AJAX realiza las peticiones al servidor por su propia naturalidad de se Asíncrono. Por lo tanto cuando se hacen peticiones paralelas o anidadas existen casos en los cuales los resultados de las peticiones pueden "mezclarse" o confundirse, o por lo menos así fue el comportamiento que obtuve. Así mismo, entender que los datos que retorna una petición solo existen mientras las petición continué ejecutándose, por lo tanto si se quiere que estos datos persistan toca aplicar otras técnicas para presentar la información después de que la petición haya terminado de ejecutarse.

La solución para estos problemas fue principalmente leer en internet los ejemplos y documentación del funcionamiento de estas peticiones. Además de hacer bastantes ensayos para entender como es que es lo que hace este modulo con los datos y las peticiones. Confidencialmente la solución para ambos problemas

fue la misma y fue obligar a que las peticiones que sean necesarias se hagan de manera asíncrona, sobre escribiendo la forma natural con la cual AJAX hace las peticiones.

Otra dificultad que tuve durante el desarrollo de este proyecto un poco menos grave pero que igual pudo disminuir un poco mi ritmo de desarrollo fue que había que cambiar un par de cosas de las funcionalidades de la implementación de la API.

Para solucionar este problema lo principal que hice fue leer el código de la implementación y entender su funcionamiento a fondo y el por que de las cosas estaban implementadas como estaban funcionando. Una vez entendido por completo su estructura y funcionamiento pude cambiar el comportamiento a como debería ser. Lo cual al final resultó siendo un punto a mi favor ya que termine entendiendo a fondo como funciona la otra parte de la API que en la entrega pasada no había tenido la oportunidad de modificar.