# MEMORIA PROYECTO FINAL

# TDW

# Tecnologías de Desarrollo para la Web

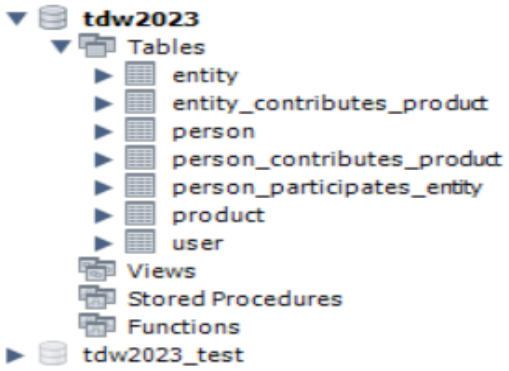
**Alumnado: LIUPENG JI**

**Matrícula: br0158**

# ÍNDICE

## 1. Modelo de datos de la aplicación

Para el funcionamiento del proyecto, se ha diseñado el siguiente modelo de datos relacional en MySQL:




- **entity:** tabla que representa una entidad científica

**Table: entity**

**Columns:**

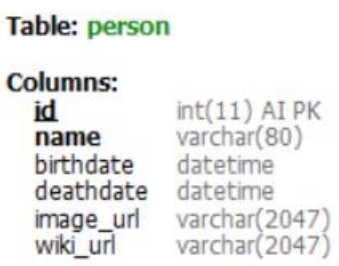
| | |
|---|---|
| **id** | int(11) AI PK |
| **name** | varchar(80) |
| birthdate | datetime |
| deathdate | datetime |
| image_url | varchar(2047) |
| wiki_url | varchar(2047) |

  - id: valor numérico único para cada entidad
  - name: cadena de caracteres que representa el nombre de la entidad
  - birthdate: fecha de creación de la entidad
  - deathdate: fecha de cierre de la entidad
  - image_url: enlace a la imagen de la entidad
  - wiki_url: enlace a la página Wikipedia de la entidad

- **entity_contributes_product:** tabla relación existente entre un elemento producto con un elemento. Indica qué entidades han contribuido en la creación de un producto


- producto_id: id del producto que participa en la relación
- entity_id: id de la entidad que participa en la relación

- **person:** tabla que representa una persona.




- id: valor numérico único para cada persona
- name: cadena de caracteres que representa el nombre de la persona
- birthdate: fecha de nacimiento de la persona
- deathdate: fecha de fallecimiento de la persona
- image_url: enlace a la imagen de la persona
- wiki_url: enlace a la página Wikipedia de la persona

- **person_contributes_product:** tabla reclación existene entre persona y producto. Indica qué personas han contribuido en la creación de un producto.

    - producto_id: id del producto que participa en la relación
    - person_id: id de la persona que participa en la relación

- **person_participates_entity:** tabla reclación existene entre persona y entidad. Indica qué personas han formado parte de una entidad.


    - entity_id: id de la entidad que participa en la relación
    - person_id: id de la persona que participa en la relación

- **product:** tabla que representa un producto


    - id: valor numérico único para cada producto
    - name: cadena de caracteres que representa el nombre del producto
    - birthdate: fecha de creación del producto
    - deathdate: fecha de baja del producto
    - image_url: enlace a la imagen del producto
    - wiki_url: enlace a la página Wikipedia del producto

- **user:** tabla que representa a los usuarios que interaccionan con la aplicación.

**Table: user**

**Columns:**

| | |
|---|---|
| id | int(11) AI PK |
| username | varchar(32) |
| email | varchar(60) |
| password | varchar(60) |
| role | varchar(10) |
| birthdate | datetime |
| state | varchar(20) |

- id: valor numérico único para cada usuario
- username: cadena de caracteres que representa el nombre del usuario
- email: correo electrónico de cada usuario
- password: contraseña de cada usuario
- role: valor enumerado que dota de permisos a cada usuario.
  - READER: pueden visualizar los productos, personas y entidades almacenados en el sistema; aparte de poder visualizar y modificar sus datos personales.
  - WRITER: hereda todas las funcionalidades del Reader, además de poder crear, modificar y eliminar los elementos almacenados en el sistema; y el privilegio de activar/inactivar un usuario, modificar las informaciones de otros usuarios, dar de baja a un usuario existente, y de poder aceptar la petición de alta de nuevos usuarios.
- birthdate: valor que representa la fecha de nacimiento de cada usuario.
- state: valor enumerado que indicar el estado actuar de cada usuario:
  - ACTIVE: estado que permite al usuario acceder al sistema
  - INACTIVE: estado que bloquea al usuario el acceso al sistema
  - PENDINGACCEPTANCE: estado único para aquellos usuarios registrado pero que aún no han sido dado de alta por un WRITER. Al igual que INACTIVE, no puede acceder al sistema

## 2. Especificación de la API REST

En el fichero openapi.yaml se define los datos de la API usada, el modelo de datos que se usará (de users, elements, products, entities, responses, requests… ); además de definir los endpoints y funcionalidades que el API ofrecerá, estas son :

- '/access_token':
  - post: retorna token de autorización (JWT)
- '/users':
  - get: retorna todos los usuarios del sistema
  - post: crea un nuevo usuario
  - options: lista de métodos http
- '/users/{userId}':
  - get: retorna el usuario con un id específico
  - put: actualiza la información del usuario con un id específico
  - delete: elimina un usuario con un id específico del sistema
  - options: lista de métodos http
- '/users/username/{username}':
  - get: determina si el nombre del usuario ya existe en el sistema
- '/products':
  - get: retorna todos los productos almacenados
  - post: creación de un nuevo producto
  - options: lista de métodos http
- '/products/{productId}':
  - get: retorna un producto con un id específico
  - put: actualiza un producto con un id específico
  - delete: elimina un producto con un id específico
  - options: lista de métodos http
- '/products/productname/{productname}':
  - get: determina si el nombre del producto ya existe
- '/products/{productId}/entities':
  - get: retorna todas las entidades relacionadas con el producto

- '/products/{productId}/persons':
  - get: retorna todas las personas relacionadas con el producto
- '/products/{productId}/entities/{operation}/{entityId}':
  - put: relaciona una entidad con un id específico con un producto con un id específico
- '/products/{productId}/persons/{operation}/{personId}':
  - put: relaciona una persona con un id específico con un producto con un id específico
- '/persons':
  - get: retorna todas las personas almacenadas
  - post: crea una nueva persona
  - options: lista de métodos http
- '/persons/{personId}':
  - get: retorna una persona con un id específico
  - put: actualiza la información de la persona con un id específico
  - delete: elimina la persona con un id específico del sistema
  - options: lista de métodos http
- '/persons/personname/{personname}':
  - get: determina si el nombre de la persona ya existe en el sistema
- '/persons/{personId}/entities':
  - get: retorna todas las entidades relacionadas con la persona
- '/persons/{personId}/products':
  - get: retorna todas los productos relacionados con la persona
- '/persons/{personId}/entities/{operation}/{entityId}':
  - put: retorna todas las entidades relacionadas con la persona
- '/persons/{personId}/products/{operation}/{productId}':
  - put: retorna todos los productos relacionados con la persona
- '/entities':

    - get: retorna todas las entidades del sistema
    - post: crea una nueva entidad
    - options: lista de métodos http
-
- '/entities/{entityId}':
    - get: retorna la entidad con un id específico
    - put: actualiza la información de la entidad con un id específico
    - delete: elimina la entidad con un id específico del sistema
    - options: lista de métodos http
- '/entities/entityname/{entityname}':
    - get: determina si el nombre de la entidad ya existe
- '/entities/{entityId}/products':
    - get: retorna todos los productos relacionados con la entidad
- '/entities/{entityId}/persons':
    - get: retorna todas las personas relacionadas con la entidad
- '/entities/{entityId}/products/{operation}/{productId}':
    - put: relaciona un producto con un id específico con una entidad con un id específico
- '/entities/{entityId}/persons/{operation}/{personId}':
    - put: relaciona una persona con un id específico con una entidad con un id específico

## 3. Arquitectura del sistema

Se ha implementado una arquitectura de cliente - servidor donde la aplicación actúa de cliente, quien realiza peticiones al API REST, quien actúa de servidor. El servidor utiliza las entradas de datos enviadas por el cliente para realizar las funcionalidades especificadas por el usuario (GET, PUT, POST, etc.) en la Base de Datos. Todos los intercambios de datos entre cliente – servidor se realiza mediante mensajes HTTP.

Cabe mencionar que el API REST utiliza Doctrine ORM para el intercambio de datos con la Base de Datos.

## 4. Problemas encontrados

- Ausencia de atributo birthdate en la tabla user, lo impedía al usuario insertar y modificar su fecha de nacimeinto.
- Ausencia de un atributo state en la tabla user, lo que impedía determinar si un usuario esta activo o inactivo.

Para ello, se ha tenido que modificar la entidad User y su controlador correspondiente, además de actualizar el fichero openapi.yml.

## 5. Mecanismos de protección utilizados

El principal mecanismo de protección es el empleo del JWT (JSON Web Token) para la autenticación del usuario en futuros acceso una vez hecho el login, sin tener que introducir los datos de login otra vez.

Por otra parte, en relación con la Base de Datos, se realiza un proceso de encriptación de la contraseña del usuario a la hora de almacenar dicha información.