

Proyecto Final

Integrantes: Florencia Miranda
Profesores: Javier Gil, Luis Fernández

1. Introducción

El objetivo de esta práctica es consolidar el conocimiento de los temas aprendidos en la asignatura, generando un proyecto que junte el trabajo realizado en todas las prácticas anteriores. En este caso se tendrá que terminar el trabajo que no fue completado para la práctica de javascript, extender la API que se creó en la práctica de PHP y finalmente cambiar las consultas realizadas al localStorage por consultas Ajax a la API.

2. Desarrollo

A continuación se explicarán los pasos seguidos para el desarrollo del proyecto separados entre la API (backend) y la aplicación del frontend.

2.1. API

2.1.1. Modelo de Datos y ORM

Se creó el modelo de datos para la aplicación usando MySQLWorkbench, para luego crear las entidades de la API. En la Figura 1 se encuentra el modelo de datos que se utilizará en la aplicación. Este modelo se generó en base al modelo ya existente de las cuestiones y usuarios de la práctica de PHP. Al tener el modelo listo, este se exportó a phpMyAdmin para generar las tablas en la base de datos. Finalmente a través del ORM Doctrine se transformaron las tablas de la base de datos en entidades de PHP con el siguiente comando:

```
$ ./bin/doctrine orm:convert-mapping annotation ./src/Entity/ --from-database
```

Luego se validó que el mapeo y la base de datos estuvieran sincronizados y resultó que la base de datos no estaba sincronizada. Para resolver esto se utilizó el comando

```
$ ./bin/doctrine orm:schema-tool:update --force --dump-sql
```

y hubo errores con las llaves foráneas, ya que se iba a actualizar el id de Soluciones y esto rompía una constraint de llave foránea del modelo de datos. Finalmente se optó por no crear las constraints al momento de crear la base de datos en MySQL y así se pudo resolver el problema. Luego de hacer algunas adaptaciones a las entidades se pudo ejecutar la API y los test de la práctica anterior sin problema.

2.1.2. Especificación de la API

Las nuevas rutas de la api se desarrollaron en base a las creadas para la práctica de PHP. A continuación se hará una pequeña descripción de las rutas creadas:

/solutions: Se creó un endpoint para las soluciones de las cuestiones. Aquí se puede hacer put, post, get y options. Se definió que para get todos los usuarios tienen acceso, ya que todos necesitarán ver las soluciones de una cuestión específica. Para el resto de los métodos solo un maestro tendrá acceso.

/propuestaSolucion : Este endpoint corresponde a las propuestas de solución que crea un aprendiz para cada cuestión. Para crear una propuesta de solución solo un aprendiz tiene acceso, y

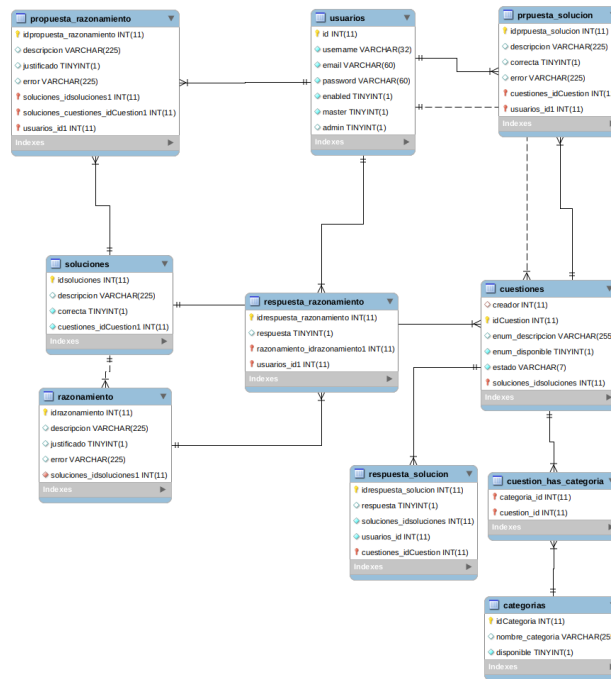


Figura 1: Modelo de datos

para los otros métodos solo un maestro tiene acceso, ya que se espera que el maestro pueda gestionar las propuestas. Existen dos métodos para get, uno para obtener todas las propuestas relacionadas con una cuestion, y otra para obtener la solución de un usuario específico para una cuestion.

/respuestaSolucion: Las respuestas de solución guardan si el aprendiz marcó como correcta o no una solución. Es por esto que solo los aprendices tienen acceso a hacer post. Por otro lado, se puede hacer get de las soluciones entregando el Id de un usuario. En el método get todos los usuarios identificados tienen acceso.

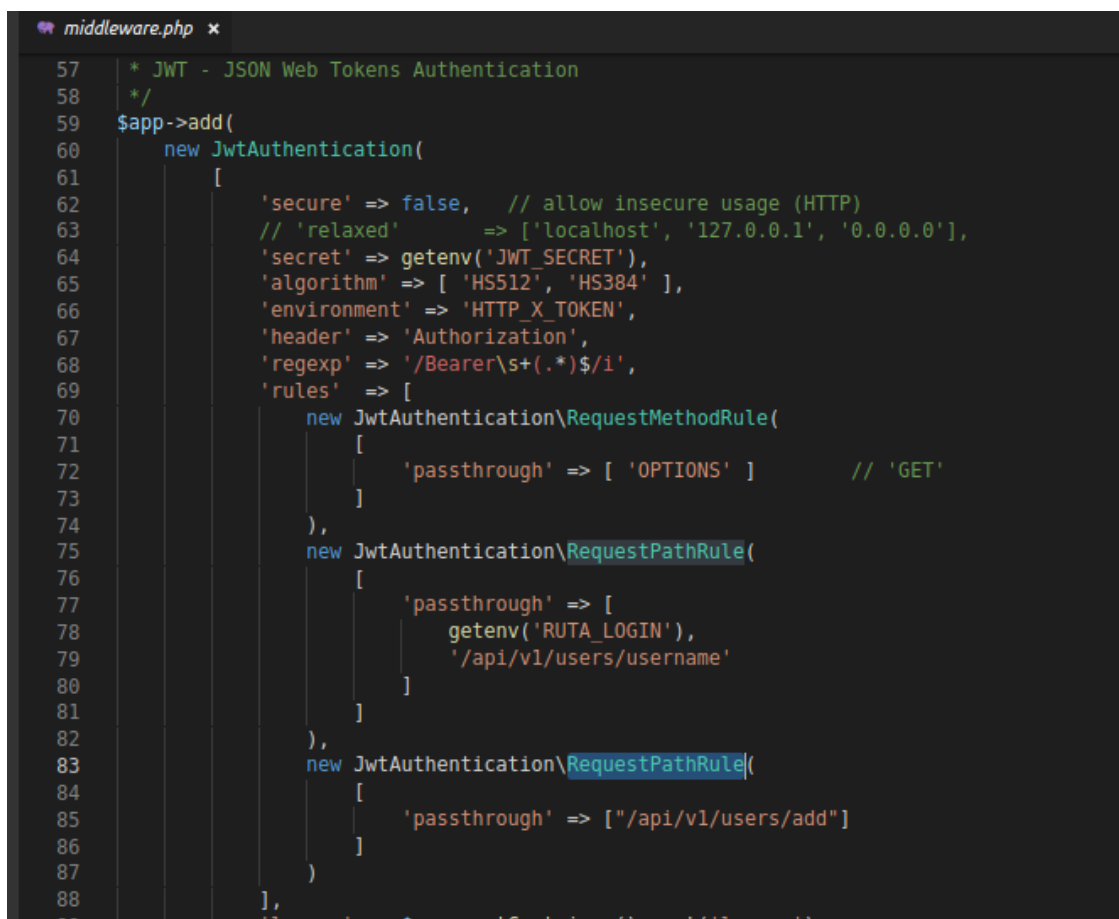
/users/add: Como la aplicación necesita crear usuarios sin tener autenticación previa (registro), se creó una nueva ruta para agregar usuarios. Para poder hacer post sin estar autenticado se modificó el fichero middleware.php y se especificó un nuevo RequestPathRule como se muestra en la Figura 2.

Por otro lado, se modificó el método getToken, ya que el valor “enabled” del token siempre era true y ahora se necesitaba especificar si el usuario está activo o no.

2.2. Aplicación Frontend

El trabajo realizado en el Frontend fue pasar los datos que se almacenaban en el LocalStorage del navegador a la base de datos. Para esto se crearon peticiones Ajax con la API y así obtener la información necesaria.

Además, de añadieron funcionalidades que no habían sido implementadas para la práctica anterior de javascript. Algunos elementos aún se guardan en localStorage tales como el token del usuario conectado o la cuestión que ha sido seleccionada.



```
57  * JWT - JSON Web Tokens Authentication
58  */
59  $app->add(
60      new JwtAuthentication(
61          [
62              'secure' => false, // allow insecure usage (HTTP)
63              'relaxed' => ['localhost', '127.0.0.1', '0.0.0.0'],
64              'secret' => getenv('JWT_SECRET'),
65              'algorithm' => [ 'HS512', 'HS384' ],
66              'environment' => 'HTTP_X_TOKEN',
67              'header' => 'Authorization',
68              'regexp' => '/Bearer\s+(.*)$/i',
69              'rules' => [
70                  new JwtAuthentication\RequestMethodRule(
71                      [
72                          'passthrough' => [ 'OPTIONS' ] // 'GET'
73                      ]
74                  ),
75                  new JwtAuthentication\RequestPathRule(
76                      [
77                          'passthrough' => [
78                              getenv('RUTA_LOGIN'),
79                              '/api/v1/users/username'
80                          ]
81                      ]
82                  ),
83                  new JwtAuthentication\RequestPathRule(
84                      [
85                          'passthrough' => [ "/api/v1/users/add" ]
86                      ]
87                  )
88              ]
89          ),
90      'logger' => $app->getContainer()->get('logger')
```

Figura 2: Modificación del fichero middleware.php

2.3. Arquitectura de la aplicación

La arquitectura de la aplicación está formada por un cliente, desarrollado en html, javascript y css. Un servidor desarrollado en PHP y conectado a una base de datos MySQL. La parte del cliente se encuentra en la carpeta *public* del servidor, para evitar tener problemas de CORS .

2.4. Mecanismos de protección

Para hacer mas segura la aplicación, se utilizaron Json Web Tokens para la autenticación de los usuarios en la aplicación. Cuando un usuario inicia sesión, se genera el token que luego será enviado en cada petición al servidor. Por otro lado, la API tiene definiciones estrictas de qué tipo de usuario puede realizar cada acción, por lo tanto la información mas crítica no será accesible a cualquier usuario con acceso a la API.

3. Ejecución de la aplicación

- Correr Xampp y correr el servidor de mySQL.

- Para ejecutar por primera vez la API, seguir las instrucciones del fichero README.md.
- Ejecutar la API con el comando

```
$ php -S localhost:8000 -t public
```
- Ir a al enlace “<http://localhost:8000/practicaFinalFront/html/login.html>” para iniciar la aplicación.

Para hacer mas fáciles las pruebas de la aplicación, al hacer *composer install* se crearán los usuarios especificados en la práctica de javascript, con los cuales se podrá iniciar sesión y hacer pruebas. Todo esto está especificado en el fichero README.md.

4. Conclusiones

A pesar de no haber implementado todas las funcionalidades requeridas para la práctica, se trabajó en ambas partes de la aplicación y se logró conectar al cliente con el servidor de forma segura.

La creación de entidades tuvo problemas en un principio lo que generó que las relaciones entre entidades no se crearan como se esperaba. Solo hay conexión entre las entidades a través de un id de la llave foránea y no a través del objeto completo. Un problema que trajo esto es que al eliminar un elemento de la base de datos, no se eliminan todos los relacionados con este automáticamente, sino que hay que agregar la eliminación a mano. No se alcanzó a modificar la eliminación por lo que al eliminar una cuestión que tiene soluciones y propuestas de solución relacionadas la aplicación falla. Esto no hace que la implementación sea incorrecta, pero no se pudo aprovechar todos beneficios que trae utilizar un ORM como Doctrine.

Para las nuevas funcionalidades de la aplicación Frontend se utilizó JQuery para trabajar con el DOM y esto permitió generar código mas limpio y desarrollar de forma mas rápida. Esto es beneficioso ya que a la hora de mantener el código o de que alguien externo lea el código es mas fácil comprenderlo.