PROYECTO TREVENQUE

Álvaro Becerril Robles

1ºDAW A

Proyecto Trevenque realizado en FCT

Fechas Estimada del Desarrollo: 3 semanas (±100H)

Índice

Primer punto →Requisitos previos y configuraciones varias

Para la fase de desarrollo, utilizaremos una máquina virtual en Debian para realizar el hosteo de la aplicación web, junto con toda la instalación y codificación necesaria.

1.1 Primeros Pasos

Para comenzar deberemos ejecutar los siguientes comandos para buscar actualizaciones y aplicarlas, esto lo haremos de la siguiente manera:

sudo apt upgrade && sudo apt upgrade

Ahora,instalaremos la conexión ssh con el fin de tener más solvencia a la hora de programar (Usaremos Visual Studio Code como IDE), para ello utilizamos:

sudo apt install openssh-server -y

1.2 Apache

Como he mencionado previamente, utilizaremos apache como servidor web. Para ello, realizamos su instalación con el siguiente comando:

sudo apt install apache2 -y

Para modificar la configuración de apache tendremos que irnos a su archivo de configuración y crear uno nuevo:

sudo nano /etc/apache2/sites-available/proyectoLaravel.conf

Una vez aquí deberemos tener un documento similar a este:

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName 10.211.64.1
        DocumentRoot /var/www/proyecto_Laravel/apirest/public

        <Directory /var/www/proyecto_Laravel/apirest/public>
                AllowOverride All
                Require all granted
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Este bloque de texto no tiene otra función más que la de atender peticiones HTTP a la dirección IP denominada en ServerName,vista previamente con el comando:

sudo ip a

Además,estaremos declarando las rutas de los logs, tanto de errores como de acceso. Por último pero no por ello menos importante, es fundamental saber que por defecto apache te muestra la página de inicio.

Para evitar esto y asegurarnos de enlazarlo con nuestro proyecto,cambiaremos el campo <Directory> donde insertaremos la ruta del proyecto de Laravel.

ATENCIÓN, independientemente de la ruta de nuestro proyecto, siempre debe apuntar a la carpeta /public, si no, no funcionará.

También deberemos modificar el archivo apache2.conf, situado en una ruta similar a la anterior mencionada, para configurar,entre otras permisiones,que las peticiones se realicen a Laravel. Sin esta configuración no podremos completar el enlace entre Apache y Laravel.

Para su correcta redirección editaremos este archivo con lo siguiente:

```
<Directory />
        Options FollowSymLinks
        AllowOverride None
        Require all denied
</Directory>

<Directory /usr/share>
        AllowOverride None
        Require all granted
</Directory>

<Directory /var/www/proyecto_Laravel/apirest/public>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>
```

1.3 MariaDB

Como base de datos vamos a utilizar MariaDB,alojado en la misma máquina virtual, para su instalación en Debian usaremos la siguiente línea de comandos:

sudo apt install mariadb-server

Para su correcta configuración accedemos a su archivo 50-server.cnf, esto con el fin de que escuche todas las direcciones de red.
El cambio puede parecer muy simple, pero es esencial para un correcto funcionamiento.

Una vez accedido al archivo cambiamos este campo:

bind-address = 127.0.0.1

Por este otro:

bind-address = 0.0.0.0

Ahora,deberemos crear la base de datos del proyecto junto con un usuario (asignándole permisos) y una contraseña robusta.

Para acceder a la terminal de mariaDB utilizaremos el siguiente comando (Hay que tener en cuenta que primero accederemos con root,ya que sino es imposible):

sudo mysql -u root -p

Una vez dentro ejecutamos las sentencias SQL correspondientes para crear un usuario,una base de datos, y darle permisos sobre esta.

Para comprobar que hemos realizado correctamente el proceso,podemos hacer la siguiente sentencia:

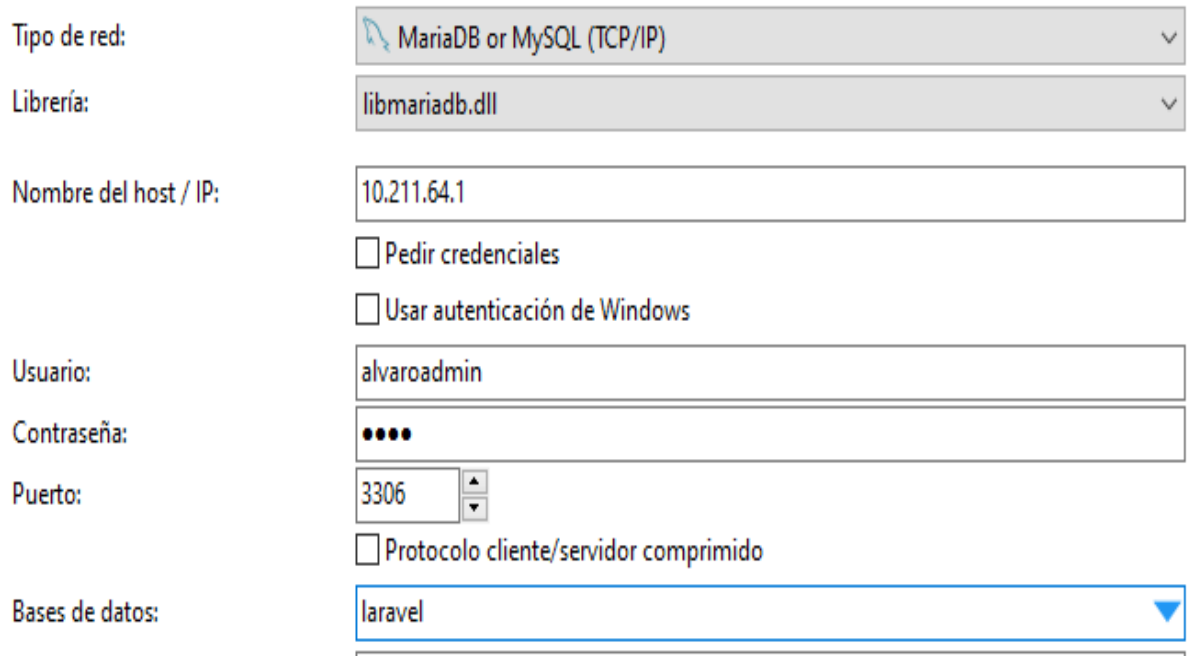
SHOW DATABASES;

Donde nos debería de dar una salida similar a la siguiente:

```
+--------------------+
| Database           |
+--------------------+
| PRUEBA             |
| information_schema |
| laravel            |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

Para darle mayor agilidad al proyecto, en cuanto a la visualización de las tablas y datos, he optado por la herramienta gráfica HeidiSQL

Para conectar esta herramienta con la máquina virtual,simplemente rellenamos el formulario con todos los datos correspondientes (IP,nombre de la BBDD, usuario/contraseña,...) y ya tendremos la conexión realizada.



En cuanto a la creación de tablas, se comentará más adelante en el documento.

Segundo punto → Laravel,estructura y configuración

Partiendo de la base de que Laravel es un framework de PHP, deberemos tener este instalado previamente.
También instalaremos la dependencia de Apache para archivos .php:

sudo apt install php libapache2-mod-php

El último paso previo a la instalación de Laravel será instalar Composer, siendo Composer el principal instalador de dependencias por defecto de PHP:

curl -sS https://getcomposer.org/installer | php

A partir de aquí pasaremos a la creación del proyecto de Laravel partiendo de Composer:

composer create-project laravel/laravel proyectoLaravel

Nos debería quedar un sistema de carpetas y archivos similar a este:

```
proyecto_Laravel/
├── proyecto_Laravel/            Lógica del backend (controladores, modelos, etc.)
│   ├── Console/
│   ├── Exceptions/
│   ├── Http/
│   └── Models/
│
├── bootstrap/
│   └── app.php
│
├── config/             # Configuración de la app (app.php, database.php, etc.)
│
├── database/           # Migraciones, seeders
│   ├── migrations/
│   └── seeders/
│
├── lang/               # Archivos de traducción (idiomas)
│
```

```
├── public/             # Carpeta expuesta al navegador (index.php, assets)
│   ├── index.php
│   └── ...
│
├── resources/          # Vistas Blade, archivos Vue, JS, SCSS...
│   ├── views/
│   ├── js/
│   └── css/
│
├── routes/             # Definición de rutas
│   ├── web.php
│   └── api.php
│
├── storage/            # Logs, archivos temporales, cachés, etc.
│   ├── app/
│   ├── framework/
│   └── logs/
│
├── tests/              # Tests unitarios y de integración
│
├── vendor/
│
├── .env                # Variables de entorno (base de datos, claves, etc.)
├── artisan             # CLI de Laravel (usa: `php artisan`)
├── composer.json       # Dependencias del proyecto
```

2.1 Conexión Laravel-MariaDB

Para realizar el enlace entre el famoso framework y nuestra BBDD, deberemos acceder al archivo .env de Laravel, situado en la carpeta principal del proyecto

Nota: Puede estar en oculto, para verlo utilizaremos el siguiente comando:

ls -la

Previamente a su acceso, deberemos generar la "key" del proyecto Laravel, con el fin de su correcto funcionamiento a la hora de validar las sesiones cifradas (cookies,..). Para ello utilizaremos el siguiente comando:

php artisan key:generate

Una vez listo, accederemos al archivo .env y modificaremos los siguientes valores:

```
DB_CONNECTION=mysql
DB_HOST=10.211.64.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=alvaroadmin
DB_PASSWORD=1234
```

Como podemos observar en la captura,debemos configurar usuario/contraseña,tipo de conexión….

Una vez configurado e iniciado el proyecto, estamos listos para empezar a codificar.

2.2 Conexión SSH Visual Studio - Debian.

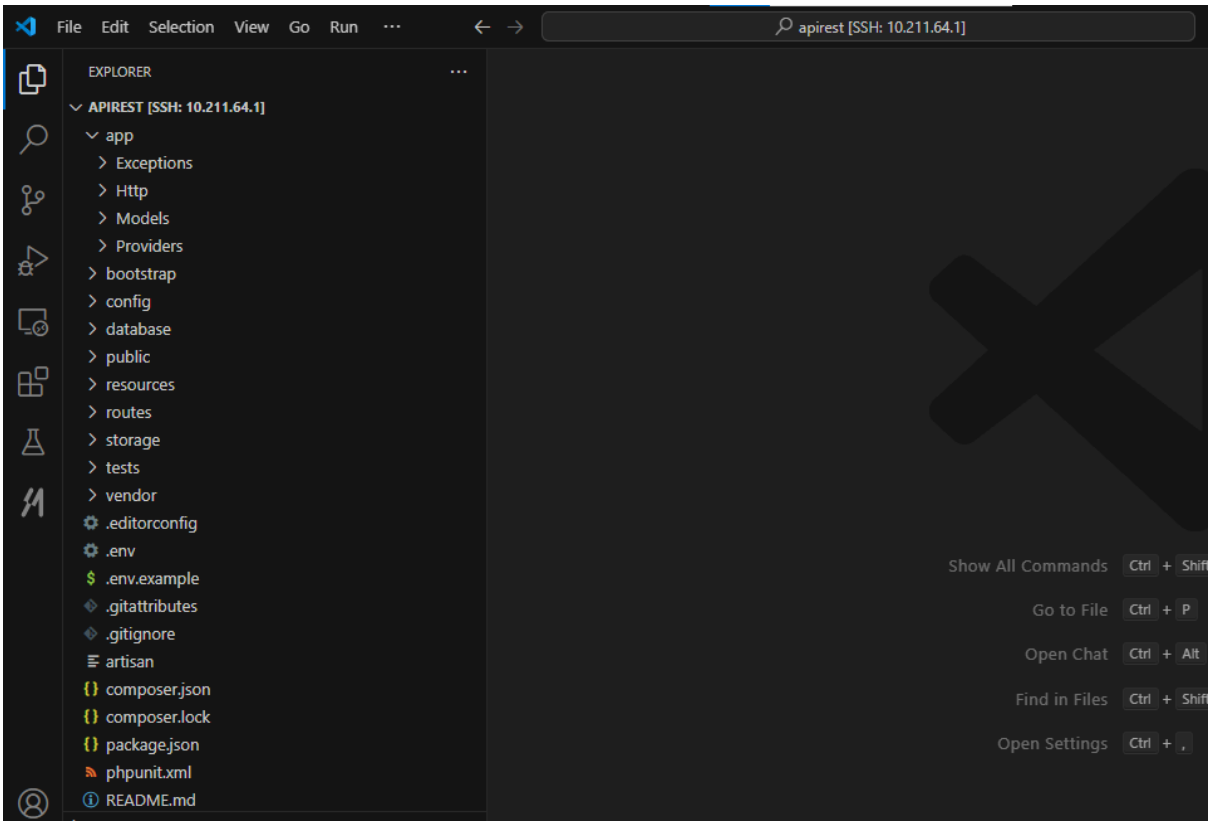
Para codificar de manera más eficiente, vamos a conectarnos a la máquina virtual a través de Visual Studio Code.

En primer lugar instalamos la extensión Remote-SSH en Visual Studio Code.

Una vez aquí, hacemos click a la barra de arriba y crearemos una nueva conexión, para ello nos pedirá la dirección de la máquina por la cual nos conectaremos (usuario@ip)

Cuando la hayamos creado nos conectaremos (nos pedirá la contraseña), y nos iremos a la ruta donde está el proyecto de Laravel.

Deberíamos de ver algo como esto:



Nota importante: Puede haber problemas de permisos a la hora de comenzar a editar los archivos,independientemente de su extensión, esto se soluciona con los siguientes comandos:

sudo chown -R $USER:$USER /var/www/miapp-laravel
sudo find /var/www/miapp-laravel -type f -exec chmod 644 {} \;
sudo find /var/www/miapp-laravel -type d -exec chmod 755 {} \;

Tercer punto →Codificación Backend

Para comenzar a generar los archivos para su posterior codificación, debemos primero entender cómo funciona un proyecto de Laravel.

Para ello vamos a definir de manera breve y concisa las principales partes del proyecto:

Vistas: Consiste en las páginas html/php que se despliegan al usuario, se utiliza principalmente un sistema de plantillas Blade, que se puede modificar al gusto del programador.

Controladores: Lógica completa del programa, son archivos .php que se van a encargar de todas las tareas realizadas con el procesamiento y modificación de datos y orquestar las instrucciones a las BBDD.

Rutas: Conexiones entre la URL y las vistas y/o controladores, de manera que al acceder a una url en concreto (Ejemplo: 10.211.64.1/bienvenida) se encargará de o bien mostrar la vista correspondiente, o bien realizar una llamada al controlador, cuya función sea definida previamente

Para la gestión de bases de datos,presenta tres herramientas:

Migración:Archivos .php que se encargan de realizar el CRUD de SQL pero en el lenguaje de programación php.

Laravel usa migraciones para mantener el control de versiones del esquema de la base de datos.

Se crean y se ejecutan con los siguientes comandos:

php artisan make:migration nombre_de_la_migracion

Una vez realizadas todas las migraciones, se ejecuta el siguiente comando:

php artisan migrate

Modelos:Clase PHP que representa una tabla de la base de datos y permite trabajar con sus registros como si fueran objetos.

Por convención, el modelo se llama igual (en singular y con mayúscula) que su tabla/migración correspondiente (en plural y en minúscula).

Para crear un modelo se utiliza el siguiente comando:

php artisan make:model Nombre_Modelo

Seeders: Archivos ejecutables cuya misión es rellenar las migraciones de datos, en este proyecto se han utilizado recibiendo datos de APIS o en formato .csv

Para crear y ejecutar un seeder se utilizan estos 2 comandos:

php artisan make:seeder NombreDelSeeder
php artisan db:seed --class=NombreDelSeeder

Nota: Es importante tener constancia de que para el apartado de la BBDD, los archivos se crean automáticamente en la carpeta del proyecto correspondiente (/migrations←Migraciones,/models←Modelos…)

Pero en cuanto a los controladores y a las vistas debemos encargarnos personalmente de crearlos en su carpeta correspondiente (/Controllers y /resources/views respectivamente).

En cuanto al FrontEnd, he programado las vistas con html,css y JS, nos vamos a centrar en este último.

Cuarto punto →Consideraciones Extraordinarias

4.1 JavaScript

Para conectar las rutas con el frontend, he utilizado la función fetch en JavaScript, esto con el fin de que se “active” la ruta, y de paso a la siguiente acción (cargado de vista o controlador).

He utilizado un JavaScript de tipo modular para mayor reutilización de los elementos y una mayor estructuración del contenido.

En el main de mi JavaScript (nueva_Rest.js) se encuentran las instancias de las clases y las funciones de los botones, mientras que en cada clase se encuentra el código innerHTML y filtrado de datos,cálculos intermedios….

Para la elaboración de gráficas he utilizado la librería Chart.Js y para el mapa interactivo he utilizado la librería propuesta Leaflet.Js

4.2 APIs y Token CSRF

Ambas peticiones a las APIs se han realizado de manera independiente en su respectivo controlador,siendo necesaria en la API de la AEMET una "key", la cual se ha generado y se ha almacenado en el archivo .env de Laravel.

Para informarse acerca de la API del tiempo se ha usado el siguiente enlace.
Con respecto a la API de la AEMET se ha investigado en esta web.

Aunque son bastante diferentes, la lógica seguida en el proyecto es muy similar: Dada una lista de provincias o municipios, se extrae el código de cada uno y se almacena en su respectiva tabla.

Cuando el usuario ingresa un nombre se hace una comparativa con su respectivo modelo para sacar el código de la migración y se obtienen los datos en formato JSON.

Los datos JSON son interpretados y cargados mediante **JavaScript**.

Respecto al Token CSRF, para el cifrado de las peticiones **POST**, se ha utilizado ya que en la BBDD se guarda el punto previamente clicado, concretamente el nombre, la longitud y latitud (coordenadas).

4.3 Otras herramientas utilizadas.

También quiero destacar el uso de 2 herramientas fundamentales para el desarrollo, en primer lugar **PostMan**, para comprobar y corregir las peticiones **GET** y **POST** de mi API y **Docker**, para realizar la imagen que genera los 2 contenedores necesarios para hacer mi aplicación más ligera,portable y eficaz. Se desarrollará este proceso más adelante.

En el proceso de pasar a Docker mi aplicación web, he utilizado **WinSCP** para descargar los archivos con el protocolo SFTP desde mi máquina virtual hasta el ordenador local, más adelante se explicará el motivo

Por último, cabe destacar que he cambiado la configuración de red de la máquina virtual de **DHCP a IP estática** para evitar posibles problemas con cambios de dirección IP durante la fase de desarrollo (previa al uso de Docker).

Quinto punto →Docker

Para la entrega del proyecto, y una vez finalizado el desarrollo, las pruebas y validaciones correspondientes, he procedido a crear un archivo **Dockerfile** junto con un **docker-compose**.**yml** para la generación de la imagen Docker y la gestión de los servicios necesarios.

5.1 Extracción del proyecto + BBDD

Previamente al Dockerfile, y,tomando la decisión de crear la imagen y gestionar los contenedores con Docker Desktop, he tenido que extraer de Debian mi proyecto Laravel y la BBDD que he utilizado.

Para el proyecto simplemente he utilizado la herramienta WinSCP como he comentado anteriormente en el punto ***4.3 Otras herramientas utilizadas.***

En cuanto a la base de datos, he utilizado el siguiente comando para generar un dump(copia de seguridad):

mysqldump -u usuario -p laravel > backup.sql

Este archivo contiene todos los datos y la estructura de las tablas de la base de datos,listo para ejecutarse en cualquier momento.

Para extraerlo de la máquina virtual también he usado WinSCP.

5.2 Dockerfile

El Dockerfile creado tiene como función principal preparar y configurar un entorno de ejecución optimizado para una aplicación Laravel, asegurando la instalación de todas las dependencias necesarias, la configuración adecuada del servidor Apache, la gestión correcta de permisos y la automatización de tareas esenciales como la generación de la clave de aplicación, facilitando así un despliegue consistente y fiable en contenedores Docker.

Sigue el siguiente flujo:

→Basado en la imagen php:apache
→ Instalación de dependencias necesarias (curl,mysql,php…)
→Configuración de Apache.
→Instalación composer + librerías PhP
→ Copiado del proyecto completo.
→Activar el puerto 80.
→ Ajuste de permisos y ejecución del script de entrada

5.3 Script de Entrada

En el script de entrada he ido realizando pequeños echos, para hacer debugging de la conexión del contenedor que lleva mariaDB y el contenedor que lleva al aplicación web como tal.

Se presentan también algunas funciones para chequear la conexión y su número de intentos para realizar dicho enlace. Se procede a la limpieza de las variables de las migraciones (Por si hay datos obsoletos) .

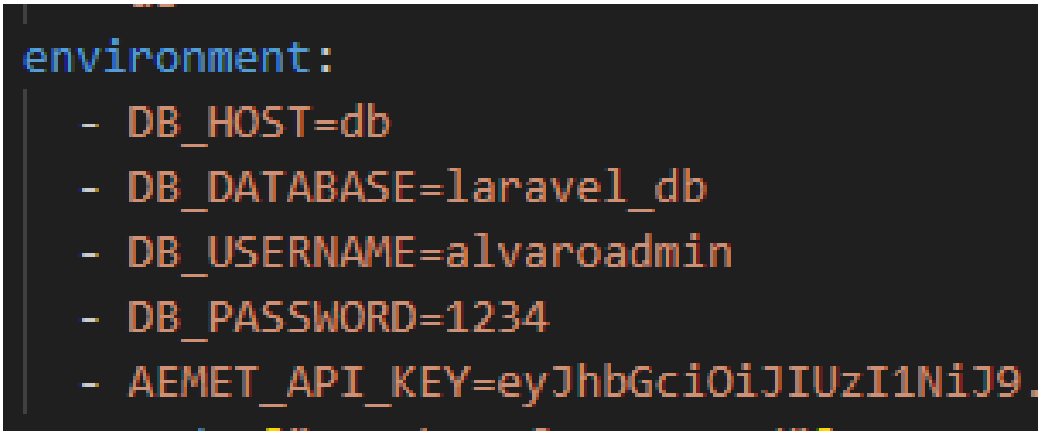
La función principal del script es verificar continuamente que la base de datos MariaDB esté disponible y aceptar conexiones antes de iniciar la aplicación Laravel, asegurando así que la conexión entre ambos contenedores sea exitosa antes de ejecutar cualquier comando o servicio.

5.4. docker-compose.yml

En el archivo dentro del proyecto nos encontraremos con las siguientes configuraciones:

Contenedor Laravel:

→Le asigna el puerto 80.
→Le indica que depende del contenedor db (El de MariaDB) (Es decir, debe inicializar primero el de MariaDB y hasta que no esté activo)
→Define un comando que se ejecuta cuando el contenedor se inicia y el script de entrada que se ejecuta previo a ese comando.
→Le asigna valores a las siguientes variables:



Entre ellas, la API_KEY de la AEMET, fundamental para el desarrollo del programa.

Contenedor MariaDB:

→ Se asigna la imagen a utilizar
→ Se indica el puerto correspondiente
→Se asocian 2 volúmenes, uno que se crea al iniciar el contenedor (volumen Docker) y un Bind Mount. Si es la primera vez que se carga el contenedor, se monta el archivo backup.sql en el contenedor para cargar la BBDD y en el volumen se guardan los datos de la BBDD una vez insertados para que sean persistentes aunque el contenedor se borre.

→Le asigna valores a las siguientes variables:

```
environment:
  MYSQL_ROOT_PASSWORD: root
  MYSQL_DATABASE: laravel_db
  MYSQL_USER: alvaroadmin
  MYSQL_PASSWORD: 1234
```

NOTA: Como podemos ver, las variables de MYSQL // DB respectivamente, deben ser iguales en ambos contenedores, sino la conexión no se realizará correctamente.

Cabe destacar que estos datos también deben ser los mismos en el archivo .env de Laravel.

Sexto punto →Reflexión Final

Como conclusión final y más personal, he disfrutado muchísimo de este proyecto, no solo por la idea tan completa y la temática apetecible, sino por la cantidad ingente de contenido,herramientas y tecnologías que he aprendido a lo largo de todo el desarrollo del proyecto.

Ha supuesto un antes y un después en mi experiencia tanto personal como profesional, ya que cada día se presentaba un nuevo reto a afrontar y, al final, disfrutas tanto el proceso como el final del desarrollo, del que por supuesto me siento orgulloso.

Como último párrafo del informe, quiero dar las gracias a Grupo Trevenque por la oportunidad y espero bien realizar las prácticas el año que viene con esta gran empresa o bien trabajar en un futuro con ellos.

→Álvaro Becerril Robles.