T6 – PHP Frameworks

E-Commerce

Using Symfony

E-Commerce

language: PHP (Symfony), Ansible

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.

THE PROJECT

The goal of the project is to create a generic API for e-commerce merchant sites with a maximum of features. In our example, we will sell computer components.
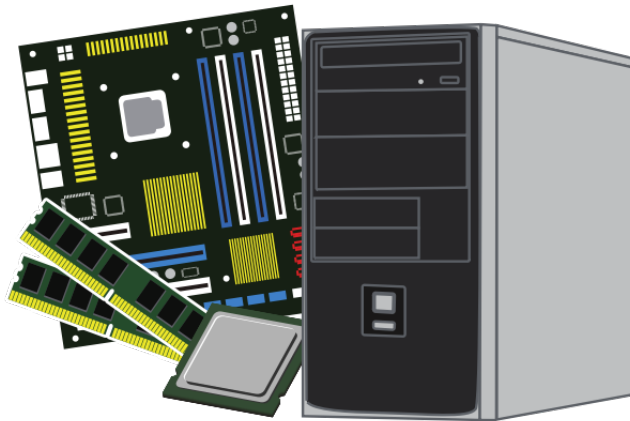
Your API must also be automatically deployable with Ansible on a Debian 11 (Bullseye) server.

- **Back-end**
  You **must** use the Symfony 6 framework to create this API.
- **Front-end**
  You are free to use the libraries and frameworks of your choice.
  This step is **bonus** and allows you to test your API.
- **Deployment**
  You **must** use Ansible and it **must** deploy on a fresh Debian 11 (Bullseye) server.

Your project, and especially the back-end, must be located in an `app` folder at the root of your repository. Thus we should be able to launch it using:

```
~/T-WEB-600> cd app/
~/T-WEB-600> symfony server:start
```

Both the back-end and deployment part are required !

API

Your API must respect the **REST** standard and using JSON format to represent data.

Each endpoint must use the **most** appropriate HTTP verb (GET, POST, PUT or DELETE).
The HTTP code associated with the response from the server must be consistent with the response sent (200 when all is well, 404 when a resource cannot be found, …) and as precise as possible (201 when a resource is created for instance).

Your API must be implemented using the power of the Symfony framework and use the JSON format for data exchange (in requests and responses bodies).

Below you'll find all the endpoints that we ask for and their uses in the following format:

```
- what the endpoint is used for (`/api/endpoint/{variablePart}`)
```

Where `{variablePart}` is to be replaced by a variable that'll be used in a request (such as an id) and `/api/endpoint/{variablePart}` is the URI to access the endpoint on the host. If the host is `localhost` we must be able to access it using `http://localhost/api/endpoint/{variablePart}` (with the required HTTP verb).

> Start by making a simple but functional API before adding more advanced features!

In case of errors, the body must be as follows:

```
{
    "error": "The error message explaining what went wrong."
}
```

USERS

A *user* is represented as such:

```
{
    "login": "foobar",
    "password": "mypassword",
    "email": "my@email.com",
    "firstname": "Foo",
    "lastname": "Bar"
}
```

- registration of user (`/api/register`)

- connection of user, retrieving the authentication token (`/api/login`)

The login process, in case of success must return:

```
{
    "token": "XXXXXXXXX"
}
```

This token will then be needed in the "authorization" header of the requests that needs authentication (indicated by the **AUTHED** flag)

- update current user information (`/api/users`) – **AUTHED**
- display current user information (`/api/users`) – **AUTHED**

> When displaying the information, is it secure to send the password?

Catalog

A *product* is represented as such:

```
{
    "id": 1,
    "name": "Item 3000",
    "description": "Best item in the shop!",
    "photo": "https://path/to/image.png",
    "price": 13.37
}
```

- Retrieve list of products (`/api/products`)
- Retrieve information on a specific product (`/api/products/{productId}`)
- Add a product (`api/products`) – **AUTHED**
- Modify and delete a product (`/api/products/{productId}`) – **AUTHED**
- Add a product to the shopping cart. (`/api/carts/{productId}`) – **AUTHED**
- Remove a product to the shopping cart. (`/api/carts/{productId}`) – **AUTHED**
- State of the shopping cart (list of products in the cart). (`/api/carts`) – **AUTHED**
- Validation of the cart (aka converting the cart to an order) (`/api/carts/validate`) – **AUTHED**

> It is a very trustfull-minded e-commerce site, everyone can add and remove items.

ORDERS

An **order** is represented as such:

```json
{
    "id": 1,
    "totalPrice": 42.01,
    "creationDate": "2021-04-01 08:32:00Z",
    "products": [
        {
            "id": 1,
            "name": "Item 3000",
            "description": "Best item in the shop!",
            "photo": "https://path/to/image.png",
            "price": 13.37
        },
        {
            "id": 2,
            "name": "Another item",
            "description": "Still good",
            "photo": "https://path/to/image2.png",
            "price": 28.64
        }
    ]
}
```

- recover all orders of the current user (`/api/orders/`) — **AUTHED**
- Get information about a specific order (`/api/orders/{orderId}`) — **AUTHED**
- Is only authorized if the order belong to the logged user.

STORAGE

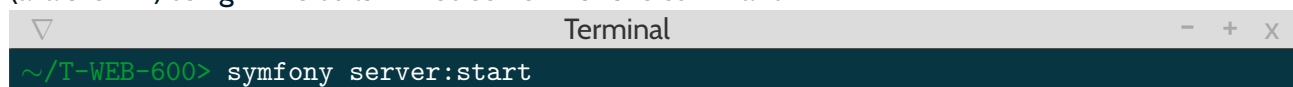To store the data you need to install a database.
In this project, we will use MariaDB which integrates very well with the symfony framework.
The following environment variable **must** be used to connect the database:
`DATABASE_URL` (eg: 'mysql://db_user:db_password@127.0.0.1:3306/db_name')

DEVELOPMENT ENVIRONMENT

Independently from the deployment part (see below), we **must** be able to launch your Symfony application (aka the API) using PHP's built-in web server with this command:

```
~/T-WEB-600> symfony server:start
```

Deployment

You **must** be able to deploy your API automatically using Ansible.
The application must be able to be deployed on Debian 11 (Bullseye) servers.
The solution must be deployable with the simple execution of the playbook.
For your api symfony version 6.2 : it is necessary to install php8.0 or higher. do not forget the additional modules
if the operating system is not debian 11 (Bullseye), the playbook should shut down properly.

symfony server and docker are forbidden to use for the deployment process.

web server as apache2 or nginx

You must have two files at the root of your repository:

```
- hosts
- playbook.yml
```

The "hosts" file represents the list of your servers.

inventory

The deployment will be executed as follows :

```
Terminal                                                       −  +  x
~/T-WEB-600> "ansible-playbook playbook.yml -i hosts"
```

authentication : SSH key

Organization

It is without saying that we cannot really test your deployment if there's no application to deploy.
But the deployment is an important part of the project and should not be considered at the end.
Do the whole project gradually, by step (and don't forget that you're in group, so you can work in parallel on several things).
Here's to help you, the beginning of an organization scheme:

- Work on parallel on the API and on the deployment.
- When you have a basic API (for instance when the retrieval, creation, deletion and updating of products is done), focus your effort on the deployment part.
- When the deployment part is done, then focus on adding the rest of your API.

> **Focussing** on something don't necessary mean to dedicate all your ressources to it. It depends of your particular team organizations and the skills of each one of you.