Appeler une API depuis le front

Table des matières

I. Contexte

Durée (en minutes) : 120

Environnement de travail :

- Ordinateur avec Windows, MacOS ou Linux
- Visual studio Code
- Si possible PHP > 7.2.5 installé avec les librairies / extensions suivantes :
    - Iconv
    - JSON
    - PCRE
    - Session
    - Tokenizer
- Composer
- Symfony CLI
- Postman

Pré requis : connaître Git, HTML, et JavaScript, des bases de Symfony

Objectifs

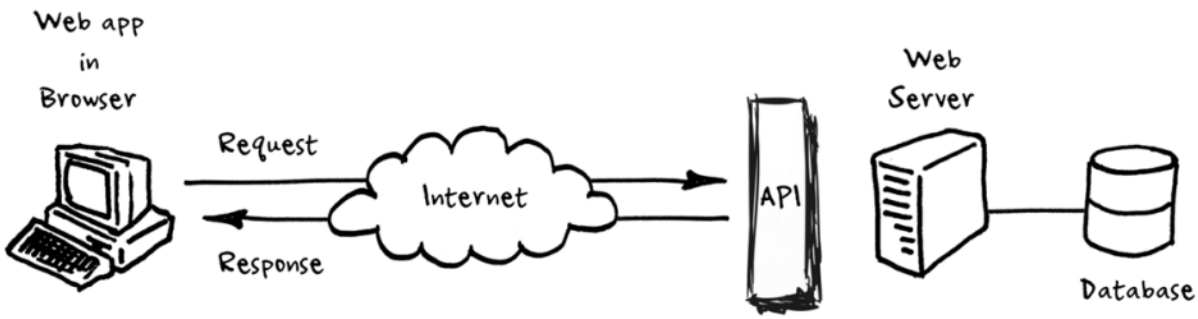
- Savoir installer un projet Symfony
- Savoir appeler une API via fetch
- Savoir récupérer des données depuis un formulaire

Contexte

Vous souhaitez comprendre comment les applications front-end et back-end communiquent entre elles ? Ce cours est là pour ça. En effet, dans le monde du développement web moderne, les API jouent un rôle central en permettant aux différentes parties d'une application de partager des données et de fonctionner de manière coordonnée. Ce cours vous fournira les connaissances et les compétences nécessaires pour travailler efficacement avec des API, que ce soit pour récupérer des données, envoyer des données, ou interagir avec des services externes. Nous allons coder la partie d'inscription et de connexion de notre site web Quai antique.

II. Travailler avec une API

A. Définition d'une API



Source : Blogspot[1]

Définition **API (Application Programming Interface)**

Un API (ou une API, les deux sont possibles), que l'on pourrait traduire en français par Interface de Programmation Applicative, est un ensemble de règles et de protocoles qui permettent à différents logiciels ou composants logiciels de communiquer entre eux.

Dans le contexte d'une application web, on peut distinguer deux principaux composants : le front-end et le back-end. L'API nous permettra de faire la liaison entre les deux.

Définition **Front-End**

Le front-end d'une application web est la partie visible et interactive que les utilisateurs voient et avec laquelle ils interagissent dans leur navigateur. Les API sont couramment utilisées dans le front-end pour effectuer des requêtes et obtenir des données du back-end.

Méthode

Voici comment cela fonctionne :

- Interface Utilisateur : le front-end est responsable de l'interface utilisateur, y compris la conception, les interactions et l'affichage des informations. Il peut inclure des éléments tels que des formulaires, des boutons, des graphiques, etc.
- Appel à l'API : pour obtenir des données dynamiques, le front-end envoie des requêtes à l'API du back-end. Ces requêtes peuvent être de différents types, tels que des requêtes GET (pour récupérer des données), POST (pour envoyer des données), PUT (pour mettre à jour des données) ou DELETE (pour supprimer des données).
- Réponse de l'API : l'API du back-end traite la requête et renvoie une réponse au front-end, généralement sous la forme de données structurées, telles que JSON (JavaScript Object Notation) ou XML (eXtensible Markup Language).
- Affichage des données : une fois que le front-end a reçu la réponse de l'API, il peut afficher les données à l'utilisateur, les mettre en forme, les filtrer ou les présenter de manière appropriée sur la page web.

Exemple

Si vous consultez une application météo en ligne, le front-end enverra une requête à une API météo, recevra les données de prévision, puis affichera ces données à l'utilisateur sous forme de graphiques et de chiffres.

Définition **Back-End**

Le back-end d'une application web est responsable du traitement des requêtes provenant du front-end, de l'accès à la base de données, de la logique métier et de la gestion de la sécurité. Les API peuvent être utilisées dans le back-end pour exposer des fonctionnalités ou des données aux clients (le front-end) de manière contrôlée et sécurisée.

- Exposition d'API : le back-end expose des API qui définissent les points d'accès aux fonctionnalités ou aux données de l'application. Ces API spécifient les méthodes acceptées, les paramètres nécessaires et les réponses attendues.
- Traitement des requêtes : lorsque le back-end reçoit une requête du front-end via une API, il effectue les opérations nécessaires, accède à la base de données, applique la logique métier et génère une réponse.

1 http://4.bp.blogspot.com/-uOEjIeHO2LY/VRcXVLYEjtI/AAAAAAAAWPU/YyBcbM3F7aQ/s1600/web20.png

- Sécurité : les API du back-end sont généralement sécurisées pour empêcher l'accès non autorisé et protéger les données sensibles. Cela peut impliquer l'utilisation d'authentification, d'autorisation et de protocoles de sécurité tels que HTTPS.

Exemple

Dans une application de vente en ligne, le back-end peut exposer une API permettant de rechercher des produits, d'ajouter des produits au panier, de passer des commandes, etc. Ces fonctionnalités sont exposées via des points d'accès API spécifiques.

Définition

En résumé, les API sont des interfaces qui permettent aux composants front-end et back-end d'une application web de communiquer de manière cohérente et structurée.

B. Installation de la partie back

Méthode

Nous voulons donc appeler notre API depuis notre application (le front). Notre site web est en local, donc uniquement sur notre machine. Pour pouvoir tester l'API, nous avons besoin de l'installer et de la lancer sur notre machine en local, afin de simuler un fonctionnement sur un serveur.

Ici, notre API est présente sur cette URL : https://github.com/GaetanRole/studi-restaurant-symfony-lts-api

Sur cette URL, nous avons accès à tous les fichiers de l'application, mais il nous faut savoir la lancer sur notre machine ! C'est pour nous aider qu'un fichier Readme.md est toujours présent. Il contient des instructions pour installer et lancer le programme.

Attention

Toutes les commandes présentes dans ce fichier sont à exécuter dans un terminal de commande, positionné sur le dossier de notre projet.

Méthode **Les prérequis**

La partie « *Project requirements* » de ce fichier nous mentionne les différents éléments que nous devons avoir installés sur notre machine pour pouvoir lancer l'application. S'il nous en manque, nous devons les installer, sans quoi l'application ne pourra jamais tourner sur notre machine.

Méthode **Installation**

Nous pouvons maintenant installer l'application. La documentation nous explique comment effectuer le 'git clone' pour copier l'application depuis github sur notre pc.

Ensuite, nous devons configurer l'application pour la faire fonctionner avec notre environnement. Il suffit de suivre les différentes instructions sur le document.

La commande `'$ cp .env .env.local'` va copier le fichier de configuration .env dans un nouveau fichier ayant pour nom .env.local. Ce fichier permet de contenir vos informations sensibles (connexion, clé d'authentification, etc.) à contrario du .env qui est une base, pouvant être versionnée.

Nous allons modifier la variable d'environnement DATABASE_URL de ce nouveau fichier. C'est une chaine de connexion, elle permettra à votre application Symfony à se connecter à votre base de données. En utilisant une base de données MySql, nous obtenons une chaîne de connexion composée ainsi :

```
mysql://user:pwd@127.0.0.1:3306/bdd?serverVersion=8&charset=utf8mb4
```

- mysql://: C'est le protocole utilisé pour la connexion à MySQL.
- user: C'est le nom d'utilisateur MySQL que vous utilisez pour vous connecter.
- pwd: C'est le mot de passe de l'utilisateur (il est important de noter que le mot de passe doit être gardé confidentiel et sécurisé).
- 127.0.0.1: C'est l'adresse IP du serveur MySQL auquel vous souhaitez vous connecter. Dans ce cas, il s'agit de l'adresse IP locale, ce qui signifie que le serveur MySQL est sur la même machine que celle à partir de laquelle vous tentez de vous connecter.
- 3306: C'est le port MySQL par défaut.
- bdd : C'est le nom de la base de données à laquelle vous souhaitez vous connecter.
- serverVersion=8&charset=utf8mb4: Ici des paramètres supplémentaires spécifiant la version du serveur MySQL et l'encodage de caractères.

Il faut donc modifier cette chaîne de connexion pour l'adapter aux besoins.

Méthode **Création et mise en place de la BDD**

Nous allons maintenant utiliser les outils mis en place dans l'API pour créer la base de données et sa structure automatiquement. Il suffit pour cela de taper ces trois commandes :

```
composer install
php bin/console doctrine:database:create
php bin/console doctrine:migrations:migrate
```

Méthode **Lancer l'API**

Notre API est maintenant installée, nous pouvons essayer de la lancer en local, en tapant la commande suivante :

```
symfony server:start
```

Nous pouvons voir dans les messages après notre commande, l'url sur laquelle l'API est lancée. Si nous accédons à cette url, nous avons le message « *Bienvenue sur votre accueil !* ».

Bravo ! Vous venez d'installer une API en Symfony sur votre PC !

C. La documentation d'API

Définition **Documentation d'API**

Une documentation d'API est un ensemble de documents qui expliquent comment utiliser une API. La documentation d'API est essentielle pour les développeurs, car elle fournit des informations détaillées sur les *endpoints* (points d'accès) de l'API, les paramètres requis, les formats de données acceptés, les méthodes HTTP supportées, les erreurs possibles, et d'autres informations importantes pour intégrer l'API dans une application.

Si nous comparons notre API à un restaurant, la documentation d'API serait la carte de menu du restaurant.

Définition **OpenAPI**

OpenAPI est un outil qui permet de décrire de manière standardisée et structurée une API. Elle utilise un format JSON ou YAML pour documenter les détails techniques de l'API. Il permet également de générer automatiquement des éléments pour notre API, notamment une interface de test, appelée SWAGGER UI.

Définition **Swagger UI**

L'interface visuelle d'OpenAPI, appelée « *Swagger UI,* » est un outil qui permet de visualiser et de tester une API en utilisant la documentation OpenAPI. Swagger UI génère une interface web à partir de la spécification OpenAPI, ce qui facilite la compréhension et l'utilisation de l'API sans avoir à écrire de code.

Nous pouvons y accéder sur notre API via l'url : http://localhost:8000/api/doc

Sur cette interface, vous pouvez visualiser tous les endpoints (URL à requetter) de l'API, et vous pouvez même le tester en direct !

Méthode

Vous pouvez donc ici essayer d'inscrire un utilisateur, en testant depuis cette interface la méthode d'inscription, comme dans la vidéo suivante.

D. Postman

Définition **Postman**

Postman est un des outils les plus connus pour tester vos API. Il permet de configurer et d'envoyer des requêtes HTTP vers votre API de façon simple et pratique. La configuration est visuelle, et il vous permet même de produire du code !

Vous pouvez donc essayer d'inscrire votre utilisateur, directement depuis Postman, comme dans la vidéo suivante.

III. Appeler une API depuis notre code

A. Le fetch

Méthode

Maintenant que notre API est lancée, et que nous avons pu vérifier qu'elle fonctionne, nous voulons l'appeler, directement depuis notre code, en JavaScript. Mais comment faire ? Il existe pour cela différentes méthodes. Dans ce cours, nous allons utiliser « *fetch* ».

Fondamental **Le HTTP**

La méthode fetch consiste en finalité, à envoyer une requête HTTP depuis notre code Javascript. Pour savoir utiliser cette méthode, il est donc fondamental de savoir ce qu'est le HTTP.

Le but d'une requête est de faire communiquer deux appareils entre eux. L'objectif final étant l'envoi et la réception de données entre un client et un serveur. Pour que les échanges soient compréhensibles, Tim Berners-Lee a inventé le protocole HTTP en 1990.

Quand un utilisateur communique avec un serveur via le protocole HTTP, il envoie une requête HTTP au serveur. En d'autres termes, lorsque vous accédez à un site Web depuis votre ordinateur en utilisant le protocole HTTP, votre ordinateur envoie une requête contenant diverses informations et le serveur renvoie une réponse HTTP.

Une requête HTTP est composée de plusieurs éléments :

- Ligne de requête : contient le type de méthode (GET, POST, etc.) et l'URL cible.
- En-têtes (headers) : fournissent des informations sur la requête, telles que les types de contenu acceptés, les cookies, etc.
- Corps de la requête : optionnel, il contient les données à envoyer au serveur, souvent utilisé avec les méthodes POST ou PUT.

Une réponse HTTP est également composée de plusieurs parties :

- Ligne de statut : indique le statut de la réponse, par exemple « *200 OK* » pour une réponse réussie.
- En-têtes (headers) : fournissent des informations sur la réponse, telles que le type de contenu, la date, etc.
- Corps de la réponse : contient les données renvoyées par le serveur, telles que le HTML pour une page web, le JSON pour une API, etc.

Définition **FETCH**

« *fetch* » est une fonction JavaScript intégrée qui permet de récupérer des données à partir de ressources externes, généralement des API, via des requêtes HTTP. Elle retourne une Promise, ce qui la rend idéale pour gérer des opérations asynchrones.

Méthode

Voici comment l'utiliser de base :

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Erreur HTTP : ' + response.status);
    }
    return response.json(); // Convertit la réponse en JSON
  })
  .then(data => {
    console.log(data); // Utilisez les données récupérées ici
  })
  .catch(error => {
    console.error('Erreur :', error);
  });
```

Le code ci-dessus effectue une requête GET à l'URL https://api.example.com/data traite la réponse en vérifiant d'abord si la réponse HTTP est réussie, puis la convertit en JSON pour l'utiliser dans notre application. Si une erreur survient à n'importe quelle étape, elle est capturée par le bloc .catch.

Méthode

La méthode fetch accepte plusieurs paramètres optionnels qui vous permettent de personnaliser votre requête. Voici les principaux paramètres possibles :

1. Méthode (method)

La méthode HTTP à utiliser pour la requête. Par défaut, fetch utilise la méthode GET. Vous pouvez spécifier d'autres méthodes comme POST, PUT, DELETE, etc.

```
fetch('https://api.example.com/resource', {
  method: 'POST',
  // Autres options...
})
```

2. En-têtes (headers)

Vous pouvez définir des en-têtes HTTP personnalisés pour votre requête. Cela peut être utile pour l'authentification ou pour spécifier le type de contenu que vous attendez en réponse.

```
fetch('https://api.example.com/resource', {
  headers: {
    'Authorization': 'Bearer your-token',
    'Content-Type': 'application/json',
  },
  // Autres options...
})
```

3. Corps (body)

Lorsque vous effectuez des requêtes POST, PUT, ou d'autres méthodes qui permettent un corps de requête, vous pouvez spécifier le contenu de ce corps. Cela peut être utile pour envoyer des données JSON ou d'autres formats.

```
fetch('https://api.example.com/resource', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({ key: 'value' }),
  // Autres options...
})
```

Ces paramètres permettent de personnaliser vos requêtes fetch en fonction de vos besoins spécifiques. Il existe d'autres paramètres optionnels, mais ceux-ci sont les plus couramment utilisés.

B. Notre premier fetch

Méthode

Pour commencer, nous pouvons essayer d'exécuter le JavaScript généré par postman, pour essayer d'inscrire un utilisateur depuis notre application, après un clic sur un bouton. Voici le code généré par Postman :

```
var myHeaders = new Headers();
myHeaders.append("Content-Type", "application/json");

var raw = JSON.stringify({
  "firstName": "Test postman",
  "lastName": "test postman",
  "email": "testdepuisPostman@email.com",
  "password": "Azerty11"
});

var requestOptions = {
  method: 'POST',
  headers: myHeaders,
  body: raw,
  redirect: 'follow'
};

fetch("https://127.0.0.1:8000/api/registration", requestOptions)
  .then(response => response.text())
  .then(result => console.log(result))
  .catch(error => console.log('error', error));
```

C. Envoyer des données

Méthode

Notre requête s'envoie bien, nous arrivons à inscrire un utilisateur, mais celui-ci est inscrit en dur dans notre JavaScript. Nous voulons maintenant envoyer à l'API, les données que l'utilisateur aura mis dans le formulaire. Voici comment faire :

```
// Crée un nouvel objet FormData à partir du formulaire contenu dans la variable "formInscription"
let dataForm = new FormData(formInscription);

// Crée un nouvel objet Headers pour définir les en-têtes de la requête HTTP
let myHeaders = new Headers();
// Ajoute l'en-tête "Content-Type" avec la valeur "application/json"
myHeaders.append("Content-Type", "application/json");

// Convertit les données du formulaire en une chaîne JSON
let raw = JSON.stringify({
    "firstName": dataForm.get("nom"),
    "lastName": dataForm.get("prenom"),
    "email": dataForm.get("email"),
    "password": dataForm.get("mdp")
});

// Configure les options de la requête HTTP
let requestOptions = {
    // Méthode de la requête : "POST" pour envoyer des données au serveur
    method: 'POST',
    // Définit les en-têtes de la requête en utilisant l'objet Headers créé précédemment
    headers: myHeaders,
    // Corps de la requête : les données JSON converties en chaîne
    body: raw,
    // Redirection à suivre en cas de besoin ("follow" suit automatiquement les redirections)
    redirect: 'follow'
};
```

D. Action après la requête

Méthode

Maintenant que l'utilisateur peut s'inscrire sur le formulaire d'inscription, nous ne pouvons pas le laisser sans aucun message de validation. Nous devons donc gérer le fonctionnement de notre application en cas de réussite de l'inscription, mais aussi en cas d'échec. Voici le code permettant cela :

```
fetch(apiUrl+"registration", requestOptions)
    .then(response => {
        if(response.ok){
            return response.json();
        }
        else{
            alert("Erreur lors de l'inscription");
        }
    })
    .then(result => {
        alert("Bravo "+dataForm.get("prenom")+", vous êtes maintenant inscrit, vous pouvez vous connecter.");
        document.location.href="/signin";
    })
    .catch(error => console.log('error', error));
```

E. La connexion

Méthode

Nous avons implémenté l'inscription, et une fois que l'utilisateur est inscrit, nous le redirigeons sur la page de connexion. Maintenant, il nous faut donc coder cette connexion, que nous avions créée en dur auparavant. Voici la fonction permettant cela :

```
function checkCredentials(){
    let dataForm = new FormData(signinForm);

    let myHeaders = new Headers();
    myHeaders.append("Content-Type", "application/json");

    let raw = JSON.stringify({
        "username": dataForm.get("email"),
        "password": dataForm.get("mdp")
    });

    let requestOptions = {
        method: 'POST',
        headers: myHeaders,
        body: raw,
        redirect: 'follow'
    };

    fetch(apiUrl+"login", requestOptions)
    .then(response => {
        if(response.ok){
            return response.json();
        }
        else{
            mailInput.classList.add("is-invalid");
            passwordInput.classList.add("is-invalid");
        }
    })
    .then(result => {
        const token = result.apiToken;
        setToken(token);
        //placer ce token en cookie

        setCookie(RoleCookieName, result.roles[0], 7);
        window.location.replace("/");
    })
    .catch(error => console.log('error', error));
}
```

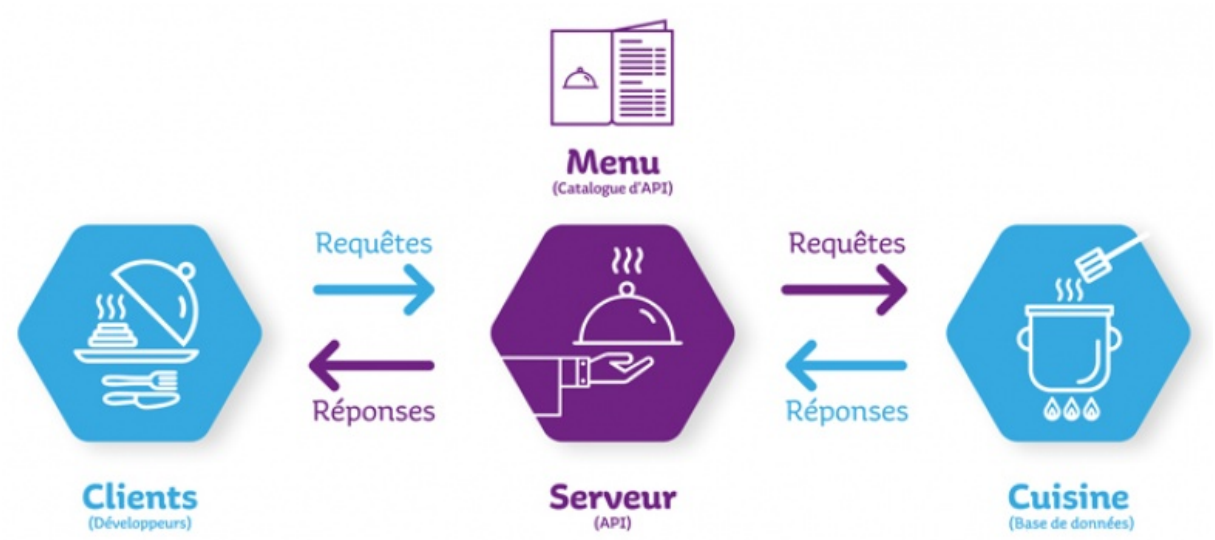
IV. L'essentiel

Les API permettent aux différents morceaux d'une application de se parler de manière organisée. Imaginez une application web comme une salle de restaurant : le front-end est la salle où les clients interagissent, tandis que le back-end est la cuisine. Les API servent de lien entre ces deux mondes, on pourrait dire que ce sont les serveurs du restaurant. La documentation d'API, c'est comme le menu du restaurant. Elle explique comment utiliser l'API en détaillant les options disponibles, les ingrédients nécessaires et les étapes à suivre. OpenAPI et Swagger UI sont des outils simplifiant la création et la compréhension de l'API. Swagger UI transforme la documentation en une interface

visuelle où vous pouvez explorer et tester l'API sans écrire de code. Pour communiquer avec une API depuis le front-end, nous utilisons la fonction fetch(). C'est un peu comme passer une commande au serveur et attendre que le plat soit servi.

Gérer les réponses de l'API, c'est comme déguster le plat. Vous vérifiez si tout est parfait. Si ce n'est pas le cas, vous prenez des mesures pour corriger le tir.

Notre application communique maintenant avec notre API, et gère donc des données en base de données, nous nous rapprochons de plus en plus du produit final !



Source : Manutan[1]

V. Pour aller plus loin

Vous pouvez maintenant implémenter d'autres appels à l'API pour récupérer les informations. Vous pourrez ainsi implémenter :

- Affichage des images de la galerie récupérées depuis l'API
- Affichage des menus récupérés depuis l'API

1 https://www.manutan.com/blog/medias/file_bank/Images/2019/12_D%C3%A9cembre/thumbs/863_Infographie-API-restaurant-FR-800-100.jpg