DEVELOPPEMENT D'APPLICATION WEB

FullStack

OBJECTIFS

A travers cet exercice, vous allez créer une application de vente aux enchères dans une version ultra wish de eBay.

Parmi les points abordés :

- Créer un backend Java qui fournira une API Rest permettant d'interroger et de manipuler une base de données.
- Créer un frontend en HTML / CSS / JavaScript qui interagira avec votre API Rest dans une architecture micro-services.
- Mettre en place des Server Sent Events pour rendre votre application davantage réactive.

PREPARATION

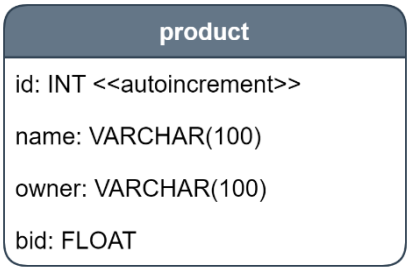
BOITE A OUTILS

Un dépôt GitHub a été créé pour vous permettre de télécharger des éléments utiles pour la suite du sujet.

- Téléchargez le contenu du dépôt https://github.com/cmeunier-ub/daw-fullstack/tree/main et conservez-le pour plus tard. On appellera cela, **`la boîte à outil`**.

MYSQL

- Utilisez XAMPP, WAMPP ou MAMPP pour démarrer les services Apache et MySQL.
- Accédez à PHPMyAdmin et créez une nouvelle base de données **`poly_bay`**.
- Ajoutez la table suivante à votre base de données :



- Ajoutez quelques enregistrements dans cette nouvelle table.

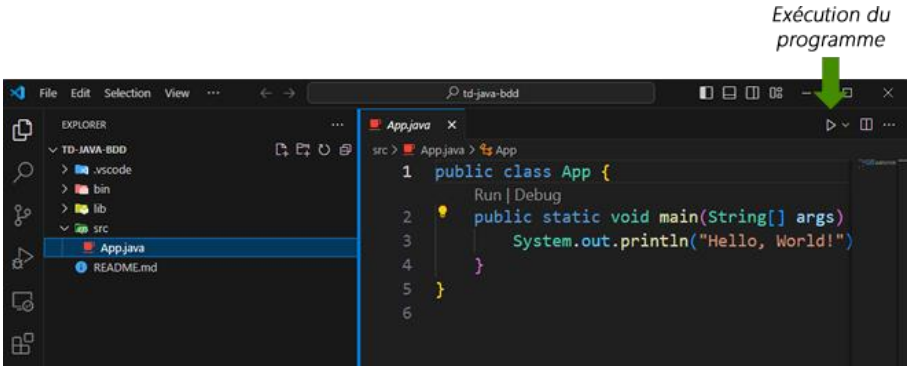
BACKEND - BASE DE DONNEES

NOUVEAU PROJET

- Démarrez Visual Studio Code
- Appuyez sur la touche **`F1`** de votre clavier.
- Saisissez **`Java`** dans l'invite qui apparaît en haut de la fenêtre VS Code.
- Cliquez sur **`Java: Create Java Project...`**.
- Sélectionnez **`No build tools`**

- Sélectionnez le dossier dans lequel vous souhaitez créer votre projet à l'aide de l'explorateur de dossiers qui apparaît.
- Saisissez le nom de votre projet (ex : backend).

Une nouvelle fenêtre VSCode apparaît avec une arborescence de projet prête à l'emploi :



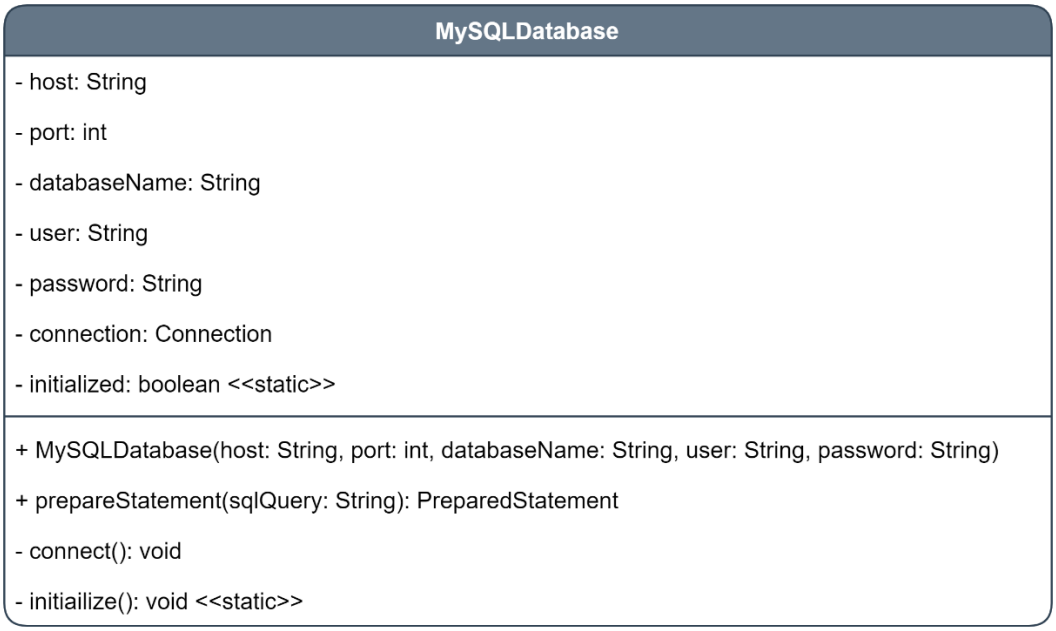
- Sélectionnez le fichier **App.java** présent dans le dossier **src** et cliquez sur le bouton permettant de démarrer le programme pour tester que tout fonctionne.

DRIVER MYSQL

- Récupérez le fichier **backend > libs > mysql-connector-j-8.4.0.jar** dans la boîte à outils et et glissez le dans le dossier **lib** de votre projet via l'interface de Visual Studio Code. Cela ajoutera directement le fichier aux dépendances externes de votre projet.

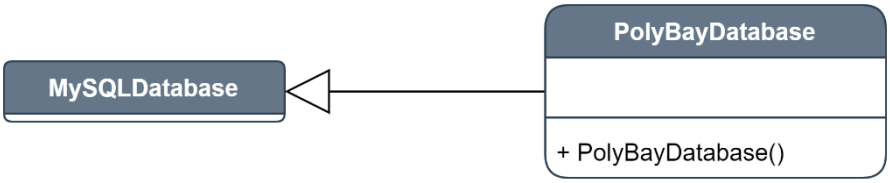
MYSQLDATABASE

- Créez un dossier **database** dans le dossier src de votre projet.
- Récupérez le fichier **backend > database > MySQLDatabase.java** dans la boîte à outils et placez le dans ce dossier. Voici la modélisation de cette classe :



POLYBAYDATABASE

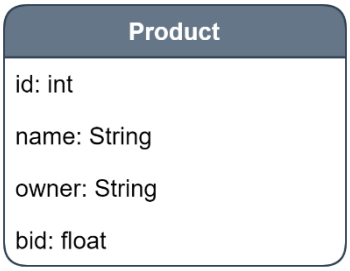
- Dans le dossier **`database`** précédemment créé, créez une classe **`PolyBayDatabase`** qui héritera de **`MySQLDatabase`** et initialisera les information de connexion à votre base de données (pas de singleton aujourd'hui :-)) :



- Modifiez la fonction **`main`** de votre projet (App.java) pour créer une instance de la classe **`PolyBayDatabase`**.
- Exécutez l'application pour vérifier qu'il n'y a pas d'erreur.

MODELE

- Créez un dossier **`models`** dans le dossier **`src`** de votre projet
- Créez dans ce dossier un record **`Product`** (clic-droit sur le dossier > New Java File > Record) ayant la structure suivante :

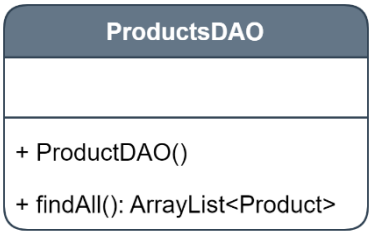


Note : Record a été intégré à Java 14 pour manipuler des classes de données de manière simple et concise. C'est en quelque sorte un équivalent aux objets de base en JavaScript. Les attributs d'un record sont immutables (ne peuvent pas être modifiés après avoir été initialisés) et les getters sont générés automatiquement par le compilateur Java.

```
//Définition d'un record
public record MyRecord(
    String stringAttribute,
    int integerAttribute,
    boolean booleanAttribute
);

//Création d'une instance de record
MyRecord myRecord = new MyRecord("first", 2, true);

//Accès aux attributs d'un record
String str = myRecord.stringAttribute();
boolean bool = myRecord.booleanAttribute();
```

DAO

- Créez un dossier **dao** dans le dossier **src** de votre projet.
- Dans ce dossier, créez une classe **ProductsDAO** selon la modélisation suivante :



La méthode **findAll** récupère tous les produits présents dans la base de données, triés par nom croissant, et retourne un **ArrayList** de **Product**.

- Supprimez l'instance de **PolyBayDatabase** crée dans la fonction main de votre projet.
- Créez une instance de **ProductsDAO** dans la fonction main de votre projet et affichez le résultat de la méthode **findAll**.
- Testez le bon fonctionnement.

BACKEND - API

SERVEUR WEB

- Créez un dossier **webserver** dans le dossier **src** de votre projet.
- Récupérez les fichiers situés dans **backend > webserver** de la boîte à outils et placez-les dans ce dossier.
- Récupérez le fichier **backend > libs > gson-2.10.1.jar** de la boîte à outils et placez-le dans le dossier lib de votre projet.

La classe **WebServer** fournie encapsule la classe **HTTPServer** de Java pour simplifier la mise en place d'API et la gestion des Server Sent Event (SSE) que vous utiliserez plus tard dans ce sujet.

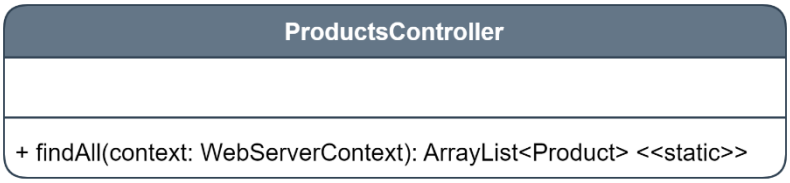
```
//Utilisation de WebServer
WebServer webserver = new WebServer();

webserver.listen(port_number);
```

- Dans la fonction main de votre projet, créez une instance de **WebServer** qui écoutera sur le port **8080**.
- Démarrez l'application.
- Ouvrez un navigateur et saisissez l'adresse **http://localhost:8080**

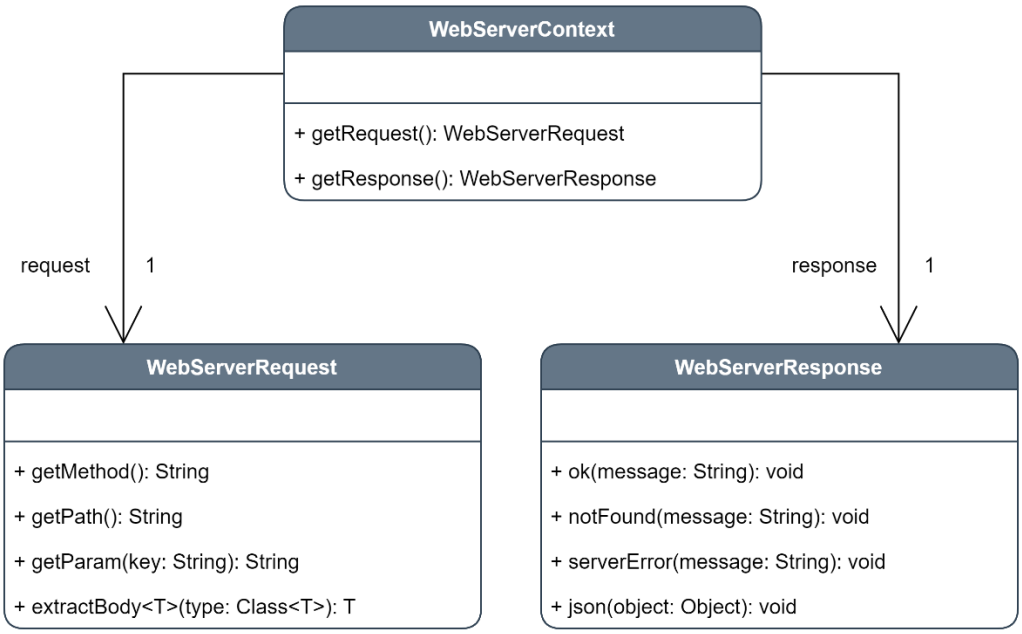
  Le texte **Not found** devrait apparaître à l'écran. C'est normal :-)

CONTROLEUR

- Créez un dossier **controllers** dans le dossier **src** de votre projet
- Créez une classe **ProductsController** à partir de la modélisation suivante :



La méthode statique **findAll** prend en paramètre une instance de **WebServerContext** dont voici une modélisation simplifiée :



Si cela vous rappelle quelque chose, c'est que vos efforts ont porté leurs fruits. Sinon, il faut vraiment travailler davantage :'(

- Faites en sorte que la méthode **`findAll`** appelle la méthode **`ok`** de **`WebServerResponse`** (par pitié réfléchissez avant de faire n'importe quoi !) avec le message **`"Tous les produits"`**.

PREMIERE ROUTE

- Dans la fonction main de votre projet ajoutez une route **`/products`** pour la méthode **`GET`** qui sera associée à **`findAll`** de **`ProductsController`**. Pour cela, adaptez le code suivant :

```
//Créer une route GET
webServer.getRouter().get(
    "/path/to/the/route",
    (WebServerContext context) -> { MyController.myFunction(context) }
);
```

- Démarrez votre programme et testez l'adresse **`http://localhost:8080/products`** depuis votre navigateur. Le message **`"Tous les produits"`** devrait apparaître.

Quelques explications : la construction d'une route pour l'API nécessite trois éléments : l'URL de la route (/product), la méthode HTTP permettant d'y accéder (GET) et la méthode à appeler pour réaliser le traitement associé. Ici, la notation **`() -> { }`** permet de créer une fonction anonyme (plus exactement une expression lambda) comme on le ferait en JavaScript avec la notation **`() => { }`**.

LISTE DES PRODUITS

- Modifiez la methode **`findAll`** pour qu'elle appelle la méthode **`json`** de **`WebServerResponse`** en lui passant la liste des produits présents dans la base de données (n'oubliez pas que vous avez un DAO).
- Testez à nouveau depuis votre navigateur. La liste de vos produits devrait apparaître au format JSON.

  Bravo ! Vous venez de créer votre première API :-)

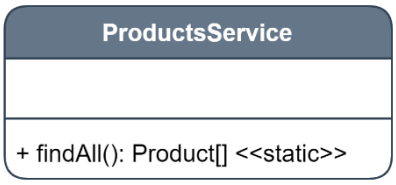
FRONTEND

PREPARATION

- Créez un nouveau dossier (ex: frontend) dans le dossier racine de votre projet, au même niveau que le dossier backend créé au début du sujet, et ouvrez ce dossier **dans une seconde fenêtre** VisualStudio Code.
- Créez un fichier **`index.html`** et initialisé sont contenu avec le snippet **`html:5`**
- Créez un fichier **`main.js`** et associé le comme module JavaScript à votre fichier HTML.
- Démarrez votre projet avec Go Live de Live Server et vérifiez que vous n'avez pas d'erreur dans la console de développement de votre navigateur.

FETCH

- Créez un dossier **`services`** dans votre projet **`frontend`** et ajoutez un fichier **`products-service.js`** dans lequel vous implémenterez la classe suivante :



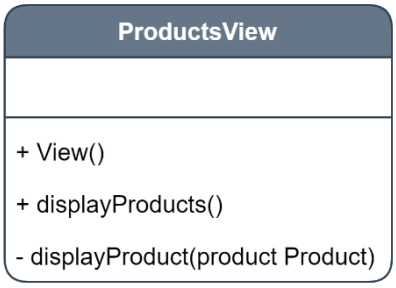
La méthode statique **`findAll`** récupérera la liste des produits auprès de votre API **`/product`** et cette liste.

```
//Rappel sur l'utilisation de fetch
async myFunction()
{
    const response = await fetch("http://my.domain.com/my/api/path");

    if(response.status === 200)
    {
        const data = await response.json();
    }
}
```

- Dans le fichier **main.js**, créez une fonction **run** qui affichera dans la console le résultat renvoyé par la méthode statique **findAll** de **ProductsService** (n'oubliez pas que **findAll** est asynchrone).
- Appelez la fonction **run** une fois que tous les éléments de la page ont bien été chargés.

VIEW

- Dans le fichier **index.html**, ajoutez une balise **div** ayant la classe CSS **products**.
- Créez un dossier **views** dans votre projet **frontend** et implémentez la classe suivante :



La méthode **displayProducts** appellera la méthode **findAll** de **ProductsService**. Pour chaque produit retourné, elle appellera sa méthode **displayProduct**.

La méthode **displayProduct** prendra en paramètre les données d'un produit et ajoutera à la balise ayant la classe CSS **products** le code HTML permettant de produire le résultat suivant :

ASUS ROG Strix GeForce RTX 4090 24GB OC Edition	qchassel	2000.00 €	Enchérir
Prototype Carmat Alpha 0.0.1	cmeunier	12.00 €	Enchérir

- Modifiez le code de la fonction **run** du fichier **main.js** de manière à afficher les produits dans votre page HTML.

ENCHERES

Retournez sur la session VS Code qui gère votre projet backend (Java).

DAO

- Ajoutez une méthode **bid** à la classe **ProductsDAO** qui prendra en paramètre l'id d'un produit et ajoutera 50 € à sa valeur dans la base de données.

```
//exécution d'une requête INSERT/UPDATE/DELETE
boolean result = preparedStatement.execute();
```

Attention aux injections SQL !

CONTROLEUR

- Ajoutez une méthode **bid** à la classe **ProductsController** qui appellera la méthode **bid** de **ProductsDAO** en lui passant, en dur, l'id d'un produit existant dans votre base de données (exemple : 1).

  - Si la méthode **bid** lève une exception, vous appellerez **serverError** de **WebServerResponse**. Sinon, vous appellerez **ok** de **WebServerResponse**.

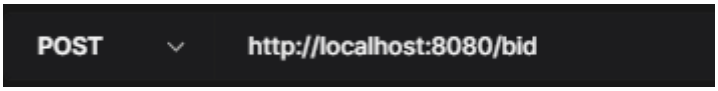
ROUTE

- Dans la fonction **main** de votre projet, ajoutez une route **/bid** pour la méthode **POST** à votre serveur web.

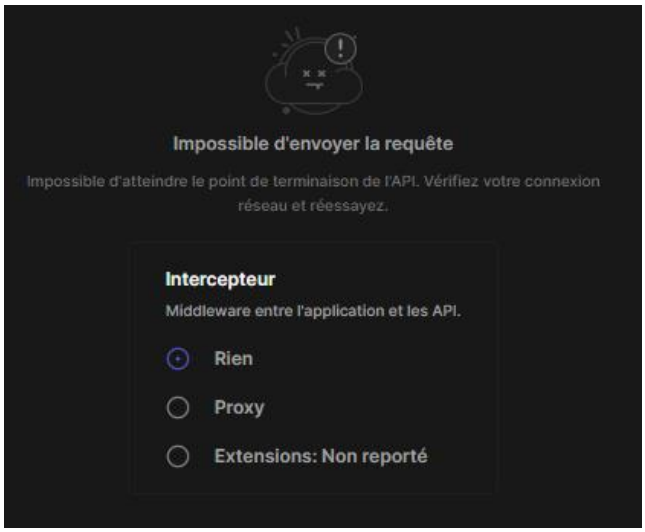
```
//Créer une route POST
webServer.getRouter().post(
    "/path/to/the/route",
    (WebServerContext context) -> { MyController.myFunction(context) }
);
```

TESTER UNE API SANS FRONT

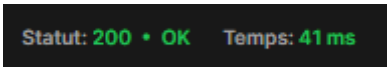
- Démarrez votre application
- Accédez au site internet **https://hoppscotch.io/**
- Sélectionnez la méthode **POST** et saisissez l'URL de votre API :



- Cliquez sur le bouton **Envoyer**
- Si le point de terminaison de l'API ne peut être atteint, sélectionnez **Rien** puis cliquez de nouveau sur le bouton **Envoyer**.



- Si tout fonctionne correctement, vous devriez obtenir un code 200



- Vérifiez dans votre base de données (via PHPMyAdmin) que la valeur (**bid**) de votre produit a bien augmenté.

ROUTE PARAMETREE

Il est possible de définir des routes pour lesquelles des paramètres vont pouvoir être directement intégrés à l'URL.

- Modifiez la route **/bid** en **/bid/:productId** (attention les **:** sont importants)

  Ici, on indique que si une requête est faire avec l'URL **/bid/1** alors la valeur **1** sera affectée au paramètre **productId**.

- Dans la classe **ProductsController**, modifiez la fonction **bid** pour que celle-ci récupère la valeur du paramètre **productId** via la méthode **getParam** de **WebServerRequest** (là encore, observez la modélisation de **WebServerContext** présentée plus tôt et réfléchissez).
- Passez cette valeur en paramètre à la méthode **bid** de **ProductsDAO**.
- Testez avec **HoppScotch** plusieurs URL dont certaines avec des id qui n'existent pas dans la base de données.

ASSOCIER LE FRONT

- Retournez sur la session VS Code qui gère la partie front de votre projet.
- Ajoutez une méthode statique **bid** à la classe **ProductsService** qui prendra en paramètre l'id d'un produit et contactera l'API précédemment créée. Attention, cette API attend une requête de type **POST**.

  Si la code de retour du serveur est **200**, la méthode **bid** renverra **true**, sinon **false**.

- Modifiez la classe **ProductsView** pour que chaque bouton de la page appelle la méthode **bid** de **ProductsService** avec l'id du produit qui lui est associé (une méthode possible avec **dataset**).

  Si la méthode **bid** de **ProductsService** renvoie **true**, actualisez la page. Attention, **bid** est asynchrone.

```
//Recharger la page
location.reload();
```

- Testez le bon fonctionnement.

SANS RECHARGEMENT COMPLET

Le rechargement de la page implique de tout recharger (HTML, CSS, liste complète des produits). Même si le navigateur possède un cache qui limite le rechargement des fichiers, la solution n'est pas optimisée pour ce qui est de l'actualisation de la liste des produits alors que seule une valeur change à chaque clic sur un bouton.

- Modifiez votre API pour que cette dernière renvoie la nouvelle valeur du produit au format JSON.
- Récupérez cette valeur dans le front et actualisez uniquement la valeur modifiée sans rechargement complet de la page.
- Testez le bon fonctionnement.

SERVER SENT EVENT

OBSERVATION

- Ouvrez plusieurs onglets de navigateur pointant vers votre front.
- Cliquez sur un bouton d'enchère. Si vous observez les autres onglets, vous voyez que la valeur n'est pas actualisée. Heureusement, si vous cliquez enchérissez sur les autres pages, la valeur actualisée est cohérente, car directement fournie par le serveur.

Une première solution serait d'actualiser la page à intervalles réguliers. Ainsi nous pourrions voir les évolutions des valeurs. Mais quel intervalle de temps choisir entre deux rafraichissements ? 1 seconde, 5 secondes ? De plus, cela va générer beaucoup d'échanges entre le client et le serveur pour rien car la plupart du temps, les valeurs n'auront pas changé.

Une autre solution est de laisser le serveur informer le client qu'une valeur a changé. Pour cela, le client ouvre une connexion avec le serveur et, à la différence d'un requête HTTP standard qui se referme sitôt traitée, ici nous allons maintenir la connexion ouverte. Ainsi le serveur pourra directement envoyer des informations au client. Bienvenue dans le monde des Server Sent Events.

Important : les SSE sont à sens unique. Seul le serveur peut envoyer des informations au client. Pour une communication bidirectionnelle, renseignez-vous sur les WebSockets.

PREPARATION

- Dans votre projet front, créez un dossier **`lib`** dans lequel vous placerez les fichiers issus de **`frontend > libs`** de la boîte à outils.

VIEW

- Ajoutez une méthode **`updateBid`** qui prendra en paramètre des données **`data`** qui lui seront transmises. Peu importe le type de ces données pour le moment, contentez-vous simplement de les afficher dans la console avec un **`console.log`**.

SSECLIENT

- Dans la fonction **`run`** du fichier **`main.js`**, créez une instance de **`SSEClient`** en précisant l'adresse et le port de votre serveur.

```
//Créer une instance de SSEClient
const mySSEClient = new SSEClient("my.domain.com:port_number");
mySSEClient.connect();
```

- Puis abonnez-vous au canal **bids** en fournissant la méthode **updateBid** de **ProductsView** comme **callback**.

```
//Abbonement à un canal
mySSEClient.subscribe("channel_name", myCallback);
```

BACKEND

- Retournez sur la session VS Code qui gère le projet backend (Java).

La classe WebServerContext possède un attribut sse, accessible via getSSE, qui permet d'envoyer des informations (string uniquement) sur un canal donné :

```
context.getSSE().emit("channel_name", "data_to_send");
```

- Modifiez la méthode **bid** de **ProductsController** afin d'envoyer sur le canal **bids** l'id du produit modifié ainsi que sa nouvelle valeur.
- Redémarrez le back et actualisez le front et observez la console lorsque vous enchérissez sur un produit.

FINAL

- Retournez sur la session VS Code qui gère le front.
- Modifiez la méthode **updateBid** de **ProductsView** afin que cette dernière actualise la valeur du produit sur la page.
- Testez avec au moins deux navigateurs côte à côte.

Toutes les pages s'actualisent simultanément ? Bravo, vous êtes parés pour le projet !