

PROJET TUTORÉ 1

Création de microservice

Licence Professionnelle

Développeur Web et Mobile pour le Commerce
Électronique

Sommaire :

Choix des microservices	2
Diagramme de Gantt	3
Répartition des tâches	4
Technologies utilisées	5
Microservice	6
Translate	6
Fidelity	14
Delivery	19
Conclusion	26
Annexe	27

Choix des microservices :

Translate – *Sujet imposé*

Le but de ce microservice est de pouvoir gérer des traductions afin de rendre nos applications traduisibles. Une application sera développée afin de voir, modifier et supprimer les traductions. Elle sera elle-même traduite en français et en anglais en utilisant ce microservice.

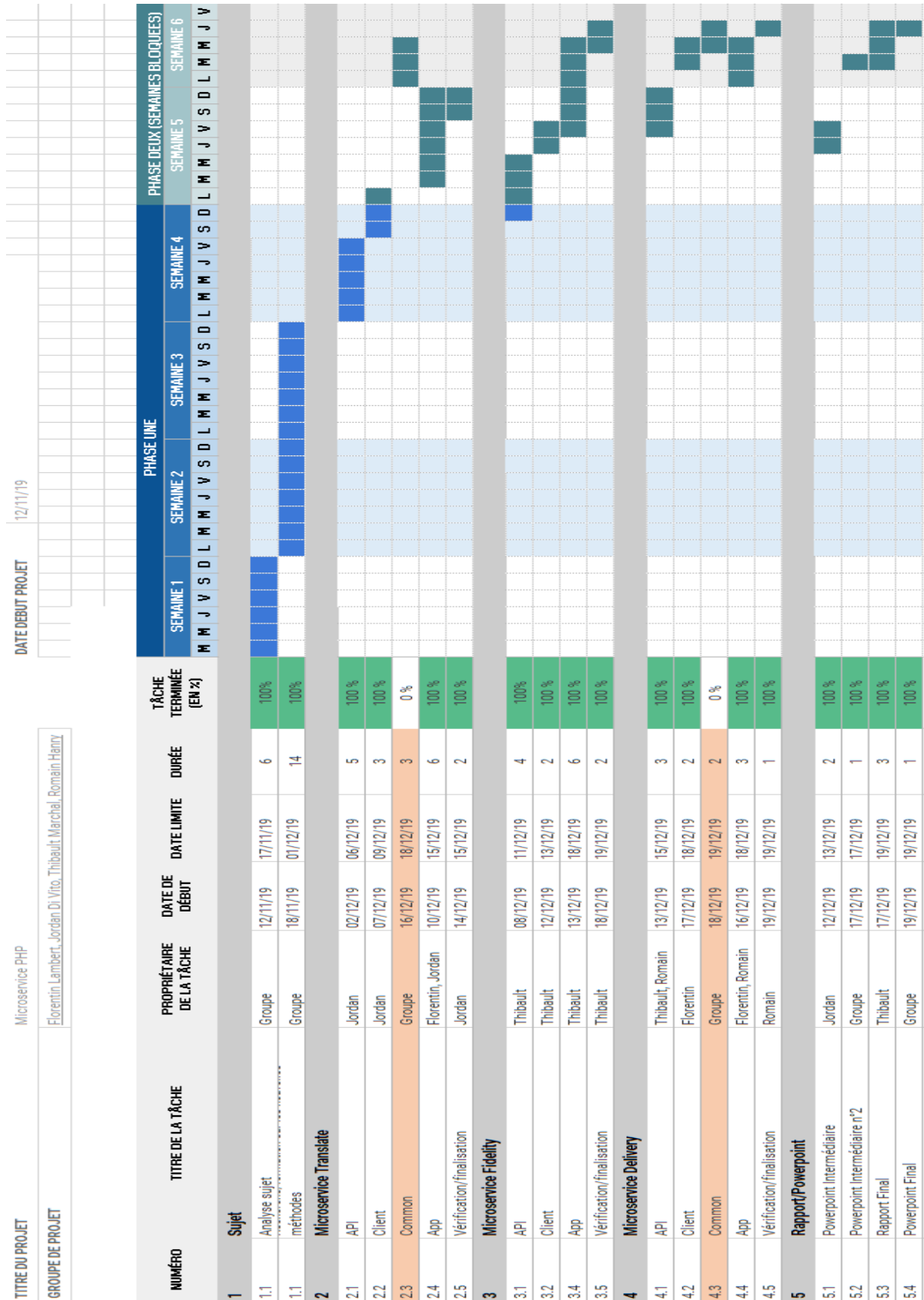
Fidelity – *Choix libre*

Le but de ce microservice est de pouvoir gérer les points de fidélité d'un client de magasin/site e-commerce. Une API en PHP sera développée afin d'ajouter des points de fidélité. Les points seront aussi diminués lorsque le client les utilisera.

Delivery – *Choix libre*

Le but de ce microservice est de pouvoir tracer la livraison d'un colis. Une API sera développée afin d'ajouter un point de localisation du colis pour une date et heure donnée. Il sera aussi possible de récupérer toutes les positions connues pour un produit à travers son numéro de colis.

Diagramme de Gantt :



Répartition des tâches :

Florentin LAMBERT

- ❖ Développement :
 - De l'application pour le microservice **Translate**
 - Du client pour le microservice **Delivery**
 - De l'application pour le microservice **Delivery**

Thibault MARCHAL

- ❖ Développement :
 - De l'API pour le microservice **Fidelity**
 - Du client pour le microservice **Fidelity**
 - De l'application pour le microservice **Fidelity**
 - De l'API pour le microservice **Delivery**
- ❖ Rédaction du rapport final

Romain HANRY

- ❖ Aide sur le développement de l'application pour le microservice **Delivery**
- ❖ Aide sur le développement de l'API du microservice **Delivery**

Jordan DI VITO

- ❖ Développement :
 - De l'API pour le microservice **Translate**
 - Du client pour le microservice **Translate**
- ❖ Aide sur le développement de l'application de **Translate**
- ❖ Rédaction du rapport intermédiaire

Technologies utilisées :

Microservice **Translate & Delivery**

- API : PHP pur / MySQL
- Client : PHP pur
- Application : HTML-CSS / PHP pur / Javascript

- **Composer** pour gérer les dépendances PHP
- **npm** pour gérer les dépendances Javascript.
- Librairie **GuzzleHTTP** pour les clients. (PHP)
- Librairie **Axios** pour les applications. (Javascript)

Microservice **Fidelity**


- API : PHP pur / MySQL
- Client : JS pur
- Application : HTML-CSS / Javascript

- **Composer** pour gérer les dépendances PHP
- **Fetch** pour le client (natif sur Google Chrome & Mozilla Firefox donc pas besoin de npm)

Microservices :

Microservice Translate

➤ Architecture de la base de données

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut
<input type="checkbox"/> 1	id 	int(11)			Non	Aucun(e)
<input type="checkbox"/> 2	locale	varchar(8)	utf8_general_ci		Non	Aucun(e)
<input type="checkbox"/> 3	key_trad	varchar(255)	utf8_general_ci		Non	Aucun(e)
<input type="checkbox"/> 4	translation	varchar(255)	utf8_general_ci		Non	Aucun(e)
<input type="checkbox"/> 5	name	varchar(255)	utf8_general_ci		Non	Aucun(e)

Capture d'écran de la structure de la base de données

« **id** » est un champ de type entier, il possède une clef primaire et il est en auto-increment.

« **locale** » est un champ de type varchar. Il permettra de stocker le langage.

« **key_trad** » est un champ de type varchar. Il permettra de stocker le texte à traduire.

« **translate** » est un champ de type varchar. Il permettra de stocker la traduction du mot.

« **name** » est un champ de type varchar. Il permettra de stocker le nom de l'application.

➤ Architecture du microservice

- translate-api

- Un dossier **public** où nous pouvons trouver le contrôleur principal qui appelle les différents fichiers. (*index.php*)
- Un dossier **src** qui contient :
 - Un dossier **Action**, il contient des fichiers avec une action spécifique par fichier. (*AddTranslate.php* ; *AllTranslate.php* ; *DeleteTranslate.php* ; *EditTranslate.php* ; *ShowTranslate.php*)
 - Un dossier **config**, il contient le fichier qui gère les routes. (liens, méthodes et actions) (*routing.php*)
 - Un dossier **include**, il contient le fichier de connexion à la base de données. (*connexion.php*)
 - Un dossier **Manager**, il contient la classe principale Translation ainsi que les différentes méthodes. (*Translation.php*)
- Le dossier **vendor**, créé avec l'aide de composer qui permet de gérer les dépendances PHP.

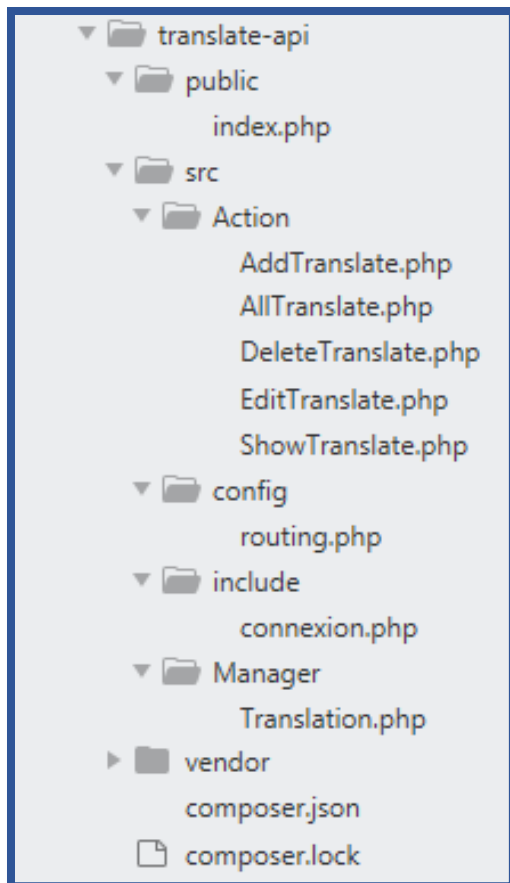
Nous avons utilisé la librairie **FastRoute** afin de gérer les routes facilement.

- *translate-client*

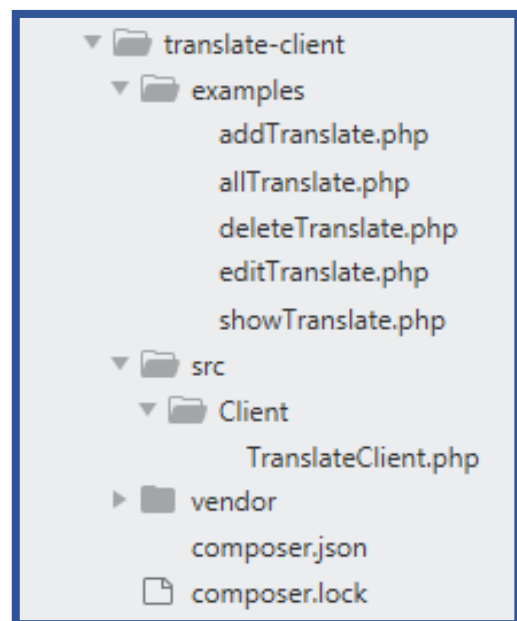
- Un dossier **examples**, il contient les différents fichiers qui permettent les requêtes vers l'API.
 - **addTranslate.php** : Permet l'ajout d'une nouvelle traduction. (*Paramètres : la locale, le mot à traduire, le mot traduit, le nom de l'application*)
 - **allTranslate.php** : Permet de voir toutes les traductions possibles.
 - **deleteTranslate.php** : Permet de supprimer une traduction. (*Paramètre : id*)
 - **editTranslate.php** : Permet de modifier une traduction. (*Paramètres : la locale, le mot à traduire, le mot traduit, le nom de l'application*)
 - **showTranslate.php** : Permet de voir les traductions d'une application. (*Paramètres : la locale, le nom de l'application*)
- Un dossier **src\client** :
 - Dans ce dossier se trouve le fichier **TranslateClient.php**, il contient la classe principale TranslateClient qui contient les méthodes utilisées dans le dossier examples.
- Le dossier **vendor**, crée avec l'aide de composer qui permet de gérer les dépendances PHP.

- *translate-app*

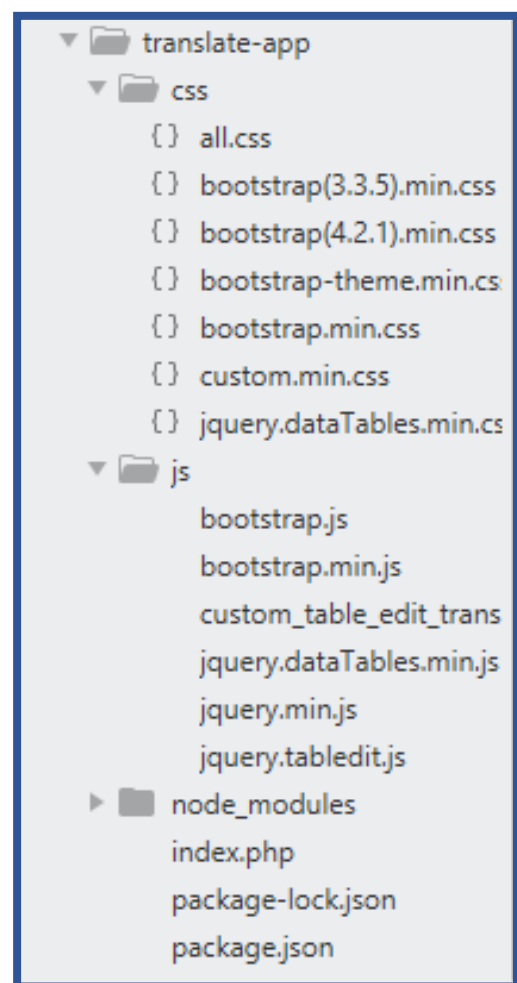
- Un dossier **css**, il contient les différents fichiers de style. (Utilisation du framework Bootstrap)
- Un dossier **js**, il contient les différents fichiers Javascript.
- Le dossier **node_modules**, crée avec l'aide de npm qui permet de gérer les dépendances Javascript.
- Le fichier **index.php**, il permet la gestion des traductions. (Ajout, suppression, modification et filtre)



Dossier translate-api



Dossier translate-client



Dossier translate-app

➤ Liste des routes et actions

Action : Afficher toutes les traductions pour les applications.

Méthode : GET

Route : « /Translate/translate-api/public/index.php/ translate-api/trad »

Action : Récupérer toutes les traductions d'une application avec la locale et son application renseignée.

Méthode : GET

Route : « /Translate/translate-api/public/index.php/ translate-api/trad/show?locale=**local**&nameApp= **Application** »

Action : Ajouter une nouvelle traduction.

Méthode : POST

Route : « /Translate/translate-api/public/index.php/ translate-api/trad/new ?locale=**locale**
&key_trad=**mot_a_traduire**&translation=**la_traduction**&nameApp=**nom_application** »

Action : Permet la modification d'une traduction.

Méthode : PUT

Route : « /Translate/translate-api/public/index.php/ translate-api/trad/new ?locale=**locale**
&key_trad=**mot_a_traduire**&translation=**la_traduction**&nameApp=**nom_application** »

The screenshot shows the home page of an application. At the top, there are four tabs: 'locale', 'key_trad', 'translation', and 'nameApp'. The 'translation' tab is currently selected. On the left side, there is a sidebar with the following elements: a button labeled 'Toutes traductions', a section titled 'Locale : (Ex : fr_FR)' with an input field, a section titled 'Application :' with an input field, a button labeled 'Recherche', and a button labeled 'Ajouter une traduction'. The main content area is currently empty.

Capture d'écran de la page d'accueil

The screenshot shows a modal window titled 'Ajouter une traduction'. It contains four input fields: 'Locale : (Ex : fr_FR)', 'Clé de traduction :', 'Traduction :', and 'Application :'. At the bottom right of the modal, there are two buttons: 'Annuler' and 'Sauvegarder'.

Capture d'écran du modal pour ajouter une traduction

Modifier une traduction

Locale :
(Ex : fr_FR)

en_GB

Clé de traduction :

Nombre

Traduction :

Number

Application :

ApplicationN1Mod

Annuler Modifier

Capture d'écran du modal pour modifier une traduction

Etes-vous sûr de vouloir supprimer cette traduction ?

Locale :
(Ex : fr_FR)

en_GB

Clé de traduction :

Nom

Traduction :

Name

Application :

ApplicationN1Mod

Annuler Supprimer

Capture d'écran du modal pour supprimer une traduction

locale	key_trad	translation	nameApp		
en_GB	Nombre	Number	ApplicationN1Mod	Modifier	Supprimer
en_GB	Nom	Name	ApplicationN1Mod	Modifier	Supprimer

Toutes traductions

Locale :
(Ex : fr_FR)

en_GB

Application :

ApplicationN1Mod

Recherche

Ajouter une traduction

Capture d'écran du filtre

Microservice Fidelity

➤ Architecture de la base de données

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut
<input type="checkbox"/>	1	id 	int(11)			Non	Aucun(e)
<input type="checkbox"/>	2	email	varchar(255) latin1_swedish_ci			Non	Aucun(e)
<input type="checkbox"/>	3	point	int(11)			Non	Aucun(e)

Capture d'écran de la structure de la base de données

« **id** » est un champ de type entier, il possède une clef primaire et il est en auto-increment.

« **email** » est un champ de type varchar. Il permettra de stocker l'adresse mail de l'utilisateur.

« **point** » est un champ de type entier. Il permettra de stocker le nombre de points de l'utilisateur.

➤ Architecture du microservice

- *fidelity-api*

- Un dossier **public** où nous pouvons trouver le contrôleur principal qui appelle les différents fichiers.
- Un dossier **src** qui contient :
 - Un dossier **Action**, il contient des fichiers avec une action spécifique par fichier. (*addPoint.php* : *decreasePoint.php* et *LoginClient.php*)
 - Un dossier **config**, il contient le fichier qui gère les routes. (liens, méthodes et actions) (*routing.php*)

- Un dossier **include**, il contient le fichier de connexion à la base de données. (*connexion.php*)
- Un dossier **Manager**, il contient la classe principale Card ainsi que les différentes méthodes. (*Card.php*)
- Le dossier **vendor**, crée avec l'aide de composer qui permet de gérer les dépendances PHP.

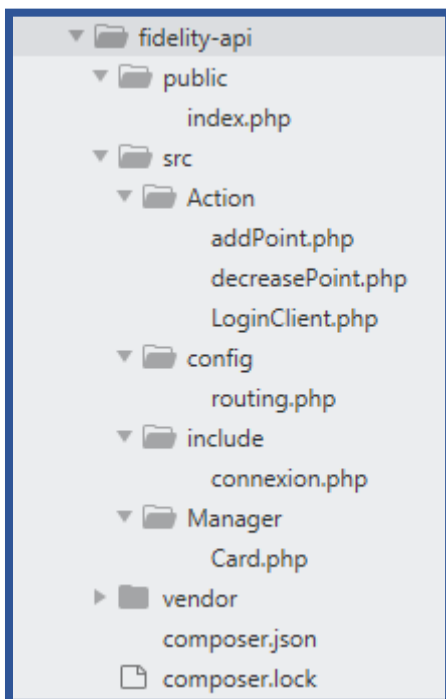
Nous avons utilisé la librairie **FastRoute** afin de gérer les routes facilement.

- *fidelity-client*

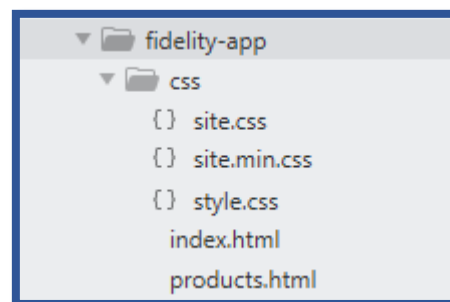
- Un dossier **src** où nous pouvons trouver les différents fichiers qui permettent les requêtes vers l'API.
 - Fichier **addPoint.js**, permet d'ajouter des points en fonction de l'adresse mail.
 - Fichier **decreasePoint.js**, permet de retirer des points en fonction de l'adresse mail.
 - Fichier **connexion.js**, permet de savoir si l'adresse mail est bien dans la base de données.

- *fidelity-app*

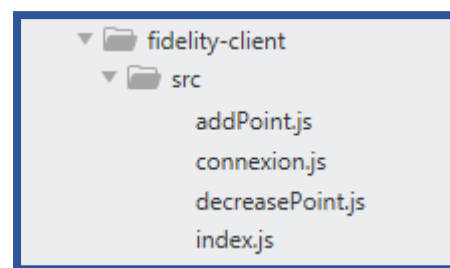
- Un dossier **css**, il contient les différents fichiers de style. (Utilisation du framework Bootflat, qui est un dérivé du framework Bootstrap)
- Fichier **index.html**, il contient le formulaire de connexion.
- Fichier **products.html**, une fois connecté, nous arrivons sur cette page qui permet d'ajouter 100 points de fidélité et d'acheter des produits.



Dossier fidelity-api



Dossier fidelity-app



Dossier fidelity-client

➤ Liste des routes et actions

Action : Ajouter des points en fonction de l'adresse mail.

Méthode : POST

Route : « /Fidelity/fidelity-api/public/index.php/ fidelity-api/point/add?email=test@test.com&point=100 »

Action : Retirer des points en fonction de l'adresse mail.

Méthode : POST

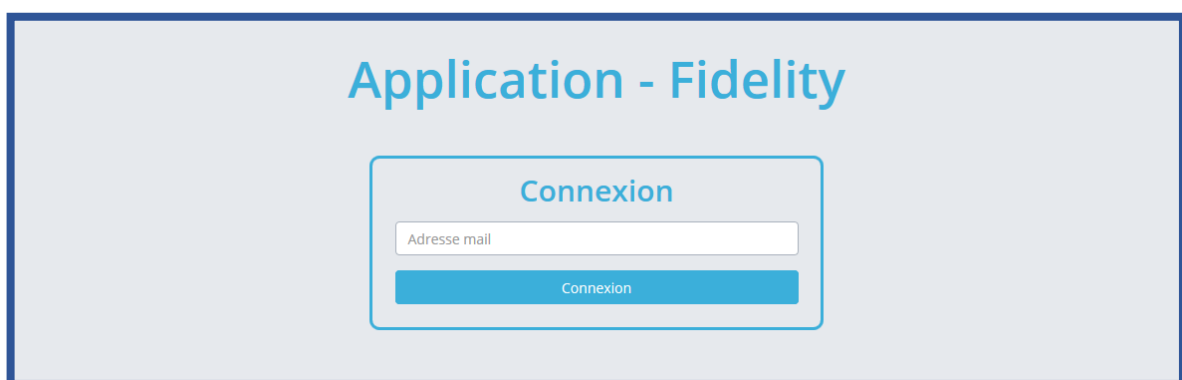
Route : « /Fidelity/fidelity-api/public/index.php/ fidelity-api/point/ decrease?email=test@test.com&point=100 »

Action : Savoir si l'adresse mail existe dans la base de données.

Méthode : POST


Route : « /Fidelity/fidelity-api/public/index.php/ fidelity-api/point/ login?email=test@test.com »

➤ Résultat

The image shows a screenshot of a web application titled "Application - Fidelity". In the center, there is a login form box. The form has a title "Connexion" at the top. Below the title, there is a text input field labeled "Adresse mail". At the bottom of the form box, there is a blue button labeled "Connexion". The entire form is set against a light gray background.

Capture d'écran de la page de connexion (index.html)

Fidelity
Connecté avec l'adresse mail : test@test.com
Vous avez : 125 points
Ajouter 100 points




Produit N°1

Donec id elit non mi porta gravida at eget metus. Nullam id.

Points : 125

Acheter




Produit N°2

Donec id elit non mi porta gravida at eget metus. Nullam id.

Points : 150

Acheter




Produit N°3

Donec id elit non mi porta gravida at eget metus. Nullam id.

Points : 485

Acheter



Produit N°4


Donec id elit non mi porta gravida at eget metus. Nullam id.

Points : 481

Acheter

Page permettant d'acheter des produits et d'ajouter des points de fidélité
Boutons grisés si nous n'avons pas assez de points de fidélité.
(products.html)

➤ Architecture de la base de données

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut
<input type="checkbox"/>	1	id 	int(11)			Non	Aucun(e)
<input type="checkbox"/>	2	parcel_number	int(11)			Non	Aucun(e)
<input type="checkbox"/>	3	latitude	float			Non	Aucun(e)
<input type="checkbox"/>	4	longitude	float			Non	Aucun(e)
<input type="checkbox"/>	5	date	date			Non	Aucun(e)
<input type="checkbox"/>	6	heure	time			Non	Aucun(e)

Capture d'écran de la structure de la base de données

« **id** » est un champ de type entier, il possède une clef primaire et il est en auto-increment.

« **parcel_number** » est un champ de type entier. Il permettra de stocker le numéro de colis.

« **latitude** » est un champ de type décimal. Il permettra de stocker la latitude.

« **longitude** » est un champ de type décimal. Il permettra de stocker la longitude.

« **date** » est un champ de type décimal. Il permettra de stocker la date.

« **heure** » est un champ de type décimal. Il permettra de stocker la l'heure.

➤ Architecture du microservice

- delivery-api

- Un dossier **public** où nous pouvons trouver le contrôleur principal qui appelle les différents fichiers.
- Un dossier **src** qui contient :
 - Un dossier **Action**, il contient des fichiers avec une action spécifique par fichier.
(*AddPoint.php : ajout.php et GetPosition.php*)
 - Un dossier **config**, il contient le fichier qui gère les routes. (liens, méthodes et actions)
(*routing.php*)
 - Un dossier **include**, il contient le fichier de connexion à la base de données.
(*connexion.php*)
 - Un dossier **Manager**, il contient la classe principale Position ainsi que les différentes méthodes. (*Position.php*)
- Le dossier **vendor**, crée avec l'aide de composer qui permet de gérer les dépendances PHP.

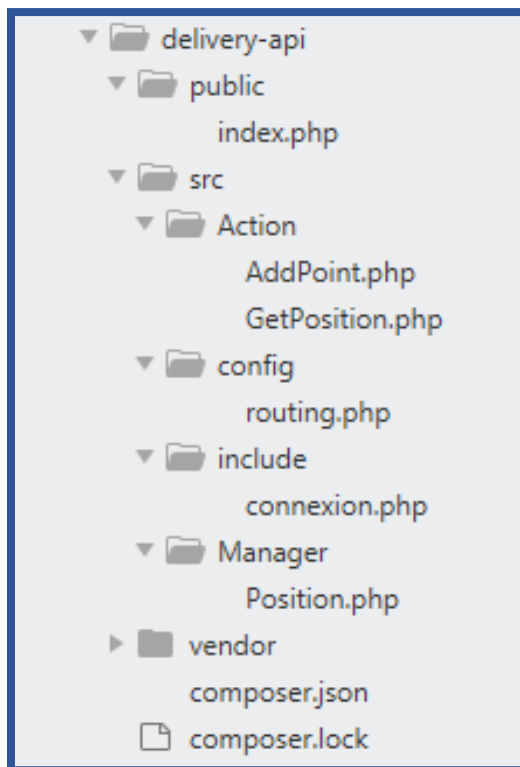
Nous avons utilisé la librairie **FastRoute** afin de gérer les routes facilement.

- delivery-client

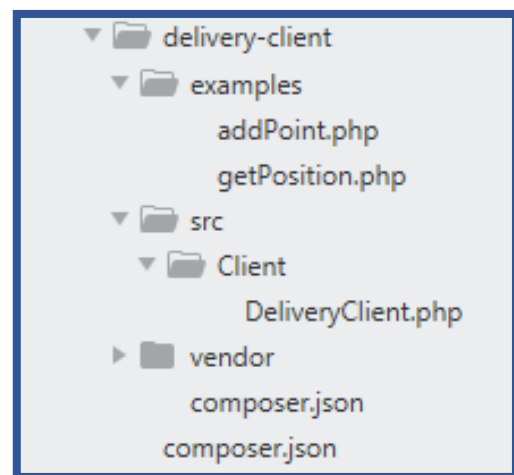
- Un dossier **examples**, il contient les différents fichiers qui permettent les requêtes vers l'API.
 - **addPoint.php** : Permet l'ajout d'un colis.
(Paramètres : le numéro de colis, la latitude, la longitude, la date et l'heure)
 - **getPosition.php** : Permet de récupérer les positions. (Paramètres : le numéro de colis)
- Un dossier **src\Client** :
 - Dans ce dossier se trouve le fichier **DeliveryClient.php**, il contient la classe principale DeliveryClient qui contient les méthodes utilisées dans le dossier examples.
- Le dossier **vendor**, crée avec l'aide de composer qui permet de gérer les dépendances PHP.

- delivery-app

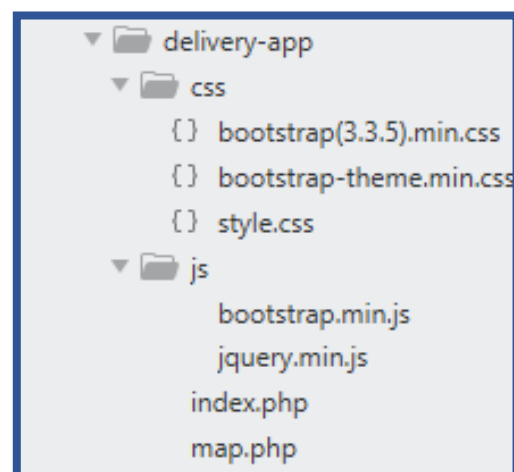
- Un dossier **css**, il contient les différents fichiers de style. (Utilisation du framework Bootstrap)
- Un dossier **js**, il contient les différents fichiers Javascript.
- Fichier **index.php**, il contient le formulaire de d'ajout d'un colis.
- Fichier **map.php**, il permet d'afficher la map.



Dossier delivery-api



Dossier delivery-client



Dossier delivery-app

➤ Liste des routes et actions

Action : Ajouter un colis avec la date et l'heure.

Méthode : POST

Route : « /Delivery/delivery-api/public/index.php/delivery-api/colis/add?parcel_number=1&latitude=51.515&longitude=15.515&date=2019-01-01&heure=12:54 »

Action : Récupérer les coordonnées du colis avec son numéro.

Méthode : GET

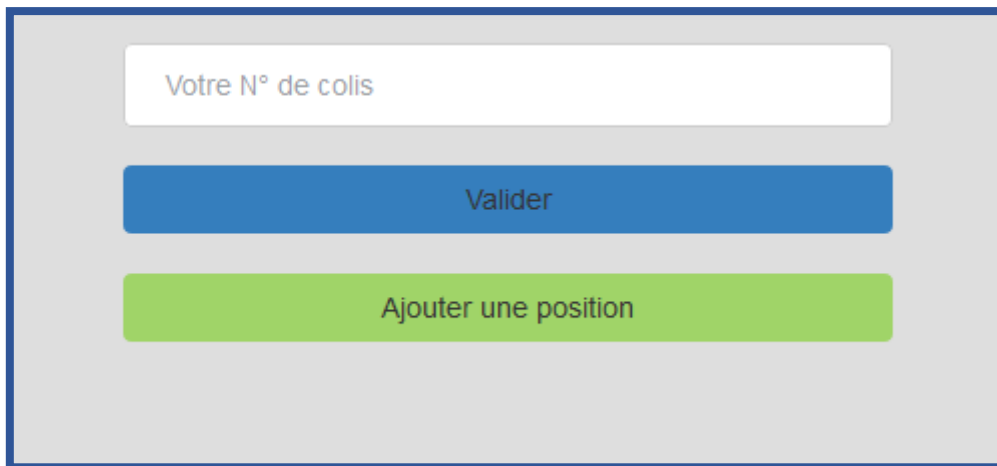
Route : « /Delivery/delivery-api/public/index.php/delivery-api/colis/get?parcel_number=1 »

Action : Récupérer les coordonnées de tous les colis.

Méthode : GET

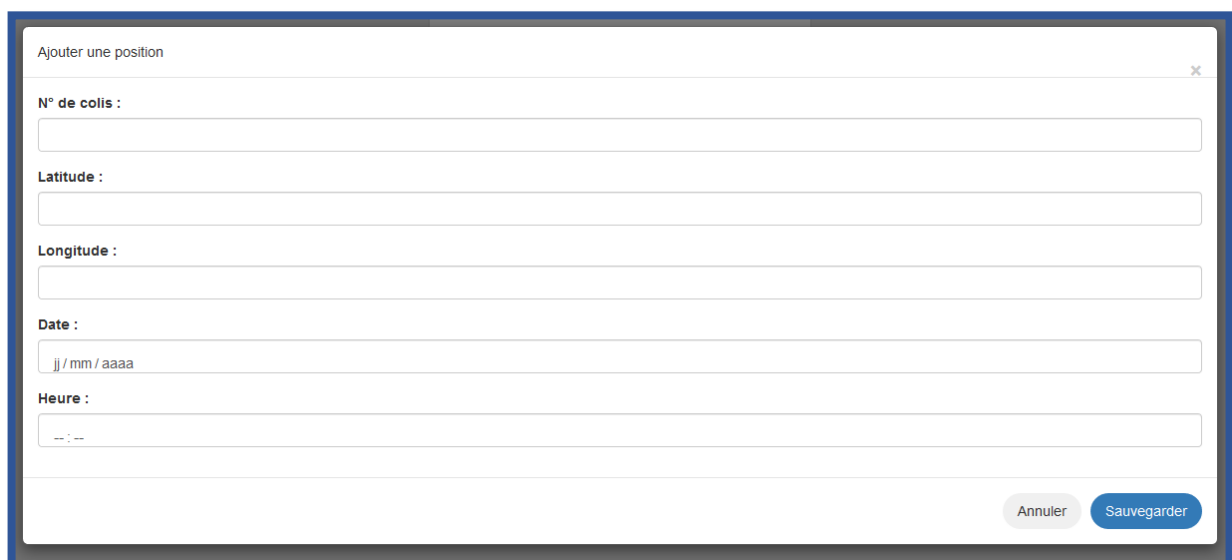
Route : « /Delivery/delivery-api/public/index.php/delivery-api/colis/get?parcel_number=all »

➤ Résultat



A screenshot of a web interface for a microservice named 'Delivery'. It features a light gray background with a white input field at the top containing the placeholder text 'Votre N° de colis'. Below the input field are two buttons: a blue button labeled 'Valider' and a green button labeled 'Ajouter une position'.

Capture d'écran de la page d'accueil du microservice Delivery



A screenshot of a modal form titled 'Ajouter une position' with a close button (X) in the top right corner. The form contains several input fields: 'N° de colis :', 'Latitude :', 'Longitude :', 'Date :', and 'Heure :'. The 'Date :' field has a placeholder 'jj / mm / aaaa' and the 'Heure :' field has a placeholder '-- : --'. At the bottom right of the form are two buttons: 'Annuler' (gray) and 'Sauvegarder' (blue).

Capture d'écran du formulaire pour ajouter un colis et sa position

Information du colis				
N° de colis	Latitude	Longitude	Date	Heure
25	58	25	2019-12-04	18:55:00

Capture d’écran du modal présentant les informations du colis



Capture d’écran de la map qui affiche les coordonnées du colis

Conclusion :

Malgré les difficultés rencontrées, nous avons pu terminer à temps les trois microservices. Il manque cependant le dossier **common** dans le microservice Translate & Delivery.

Annexe :

Microservice Translate

➤ Manuel d'installation du microservice

Pour installer ce microservice, il faut commencer par télécharger l'archive **TranslateFinal.zip**.

1. Créer une base de données.
2. Importer le fichier .sql dans cette base de données.
3. Créer un dossier **Translate** à la racine de votre serveur.
4. Mettre les trois dossiers dedans.
5. Modifier les informations de connexion dans le **translate-api/src/include**.
6. Se rendre dans le dossier **translate-client/src/client**.
7. Modifier les différentes requêtes. (*/projetTut/ProjetTut/* devient */VOTRE_URL/*)
8. Se rendre dans le dossier **translate-client/examples** et modifier tous les fichiers présents.
 - a. Modifier la base URI par l'adresse racine d'où le dossier Translate est mis
 - b. Modifier la location. (*http://127.0.0.1/projetTut/ProjetTut/* devient http://VOTRE_URL/)
9. Se rendre dans le dossier **translate-app/js/script.js**
 - a. Modifier les liens de la fonction `getTranslate` et `showTranslate`. (*/projetTut/ProjetTut/* devient */VOTRE_URL/*)
10. Utiliser un navigateur et aller à l'adresse : *http://VOTRE_URL/Translate/translate-app* pour utiliser l'application

Microservice Fidelity

➤ Manuel d'installation du microservice

Pour installer ce microservice, il faut commencer par télécharger l'archive **FidelityFinal.zip**.

1. Créer une base de données.
2. Importer le fichier .sql dans cette base de données.
3. Créer un dossier **Fidelity** à la racine de votre serveur.
4. Mettre les trois dossiers dedans.
5. Se rendre dans le dossier **fidelity-api/src/include**
6. Modifier les informations de connexion à la base de données.
7. Se rendre dans le dossier **fidelity-client/src**
8. Modifier l'URL (ligne 2) dans chaque fichier avec le format suivant:
 - a. Pour addPoint.js : *VOTRE_URL/Fidelity/fidelity-api/public/index.php/fidelity-api/point/add?*
 - b. Pour connexion.js : *VOTRE_URL/Fidelity/fidelity-api/public/index.php/fidelity-api/point/login?*
 - c. Pour decreasePoint.js : *VOTRE_URL/Fidelity/fidelity-api/public/index.php/fidelity-api/point/decrease?*
9. Se rendre sur : **VOTRE_URL/Fidelity/fidelity-app** et se connecter avec l'adresse mail : test@test.com

➤ Manuel d'intégration du microservice

Afin d'utiliser ce microservice sur un site web déjà existant, il faut télécharger l'archive **FidelityIntregation.zip**. Considérons que vous avez une page **boutique.php** avec les différents produits que vous vendez.

1. Copier le dossier **fidelity-client** sur votre serveur.

Normalement, vous avez déjà un système pour récupérer l'adresse mail du client connecté sur votre site internet.

2. Se rendre dans le dossier **fidelity-client/src**
3. Modifier la ligne 4 dans chaque fichier afin de transmettre l'adresse mail du client connecté.
4. Importer les fichiers dans votre boutique.php avec ces lignes.

```
<script src="fidelity-client/src/addPoint.js"></script>
<script src="fidelity-client/src/decreasePoint.js"></script>
```

5. Pour ajouter des points de fidélité à ce client connecté, il faut créer un bouton avec une action au clique. (Exemple pour ajouter 100 points :

```
<button type="button" class="btnAddPoint"
onclick="addPoints(100);">Ajouter 100 points</button> )
```

Il ne faut surtout pas changer le nom de la fonction « addPoints ». Vous pouvez changer le nombre à l'intérieur.

6. Pour réduire les points de fidélité d'un client, c'est le même principe. (Exemple :

```
<button type="button"
class="btnBuyProduct" onclick="decreasePoint(ID_PRODUIT,
PRIX_DU_PRODUIT);">Acheter le produit</button>
```

)

Il ne faut surtout pas changer le nom de la fonction « decreasePoint ». Veuillez remplacer « ID_PRODUIT » et « PRIX_DU_PRODUIT » avec vos données.