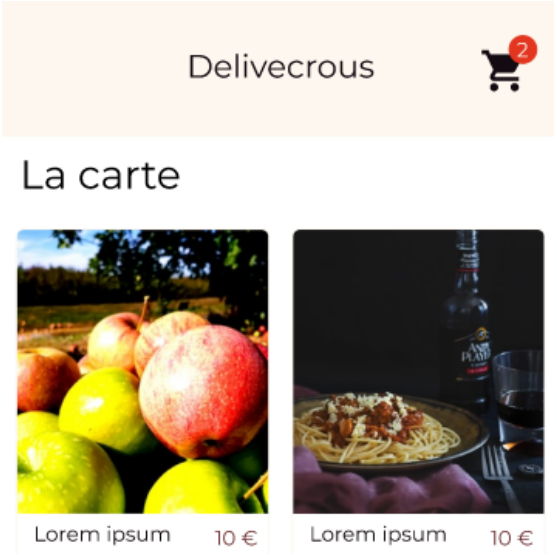
ATTENBOROUGH Elliott FISA 3 Informatique

CARLIER Amandine

TP - Technologie du Web

Réalisation de la partie Backend du site Web :

Delivecrous



Monsieur BENTIFRAOUINE Sid

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Université Polytechnique HAUTS-DE-FRANCE

Introduction

Dans le cadre du module de technologie web, nous avons eu l'occasion de réaliser la partie back (autrement dit toute la partie que l'utilisateur ne voit pas mais qui lui permet de réaliser des actions sur un site) d'un petit site web contenant plusieurs pages. Ainsi, au départ de notre projet, nous avons donc dû prendre connaissance du projet nommé Delivecrous qui nous était proposé. Afin de le se l'approprier rapidement, un résumé des fonctionnalités souhaitées, un prototype et une maquette nous a été mis à disposition.

De plus, ce travail nous a été confié avec une consigne à suivre pour que sachions, concrètement, ce qui nous était demandé. Celles-ci reprennent les routes nécessaires à notre back ainsi qu'une partie "challenge" nous permettant de nous surpasser et d'aller un peu plus loin dans ce projet.

Ce rapport va se diviser en plusieurs parties, une première traitera de la conception établie au cœur de ce back, une seconde des pour et des contre de l'utilisation du NodeJS suivie de ceux d'Express. Puis, nous expliquerons le code effectué dans une autre partie, pour terminer sur une conclusion portant sur la réalisation globale de Delivecrous.

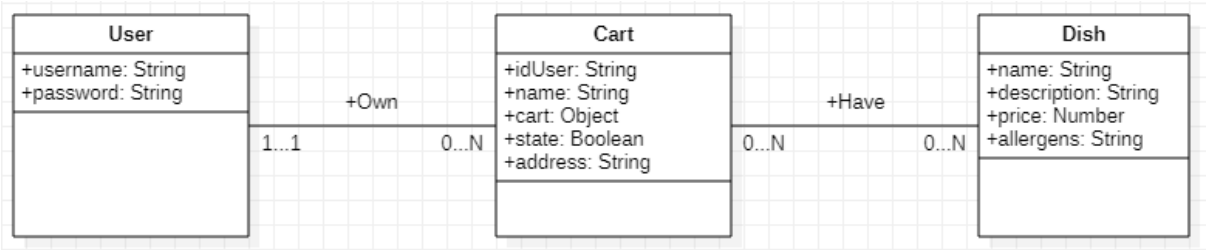
Pour consulter ce projet, voici le lien GitHub lui étant associé :
https://github.com/AvesSeven/Delivecrous.git

Celui-ci peut-être lancé via la commande : "npm run dev".

Conception

Pour la réalisation du back de Delivecrous, nous avons défini un diagramme de classes permettant de visualiser les différents objets ainsi que les divers liens entre ceux-ci.

Le diagramme de classes est donc le suivant :



Sur celui-ci nous pouvons voir trois objets :

- User : fait référence à un utilisateur du site Delivecrous, il possède un “username”, soit un nom d’utilisateur, et un “password”, autrement dit un mot de passe. Ces attributs vont permettre à l’utilisateur de se connecter pour que celui-ci puisse créer et remplir ses propres paniers. En effet, nous pouvons remarquer qu’un utilisateur peut avoir zéro ou plusieurs “Cart”, c’est-à-dire panier.

- Cart : représente un panier d’un utilisateur, il contient l’identifiant de l’utilisateur qui le possède, un nom, un objet “cart” (qui correspond en fait à un tableau d’objet avec l’identifiant d’un plat et une quantité), une état (afin de savoir si le panier est validé ou non) et une adresse permettant de stocker l’adresse que l’utilisateur souhaite associer à son panier lors de la validation de celui-ci. Nous constatons qu’un panier appartient à un et un seul utilisateur et qu’il peut contenir de zéro à plusieurs plats.

- Dish : symbolise un plat, il est défini par un nom, une description, un prix et des allergènes (stocker simplement par une chaîne de caractères, soit un String). Ici, nous voyons qu’un plat peut-être contenu dans zéro ou plusieurs paniers.

De plus, pour ce projet, nous avons décidé de nous baser sur une architecture N-tiers, avec des contrôleurs, des services et des repositories. Nous reparlerons de cette architecture plus tard dans ce rapport afin de comprendre pourquoi nous avons décidé de l’utiliser.

Par le biais de ce projet, nous avons pu manipuler divers outils, frameworks et logiciels tels que :

- NodeJS : une plateforme logicielle libre en Javascript permettant de mettre en place des applications réseaux hautement concurrentes, ce n'est ni un framework, ni un serveur. Elle est basée sur le moteur V8 utilisé pour Google Chrome et Chromium. Node.js est très utilisé pour écrire des services côté serveur, c'est-à-dire des API, en Javascript car il est très performant et plutôt facile à prendre en main.

- ExpressJS : un des framework le plus utilisé pour construire rapidement une application web sur la plateforme NodeJS. Il est extrêmement flexible, minimaliste et se complète avec des modules disponibles dans les dépôts npm (ou « Node Package Manager » qui est un gestionnaire de paquets officiel NodeJS). Nous avons également utilisé express-validor qui a pour rôle de valider les données transmises par Postman (dans notre cas).

- Postman : une application qui permet de tester des API, nous pouvons y créer et tester des requêtes HTTP. Son interface simple et intuitive nous permet de personnaliser nos requêtes en fonction de nos besoins (en utilisant des paramètres particuliers, une méthode CRUD spécifique, etc.). De plus, nous pouvons conserver un historique de nos requêtes et travailler en collaboration sur un même espace de travail.

- Visual Studio Code : un éditeur de code open-source, développé par Microsoft, qui supporte de nombreux langages et dispose divers plugins compatibles avec NodeJS. De plus, il possède d'innombrables fonctionnalités telles que l'autocomplétions, la mise en évidence de la syntaxe (par de la couleur par exemple), le débogage, les commandes git. Malgré ces diverses fonctionnalités, cet éditeur reste léger et très puissant.

- GitHub : est une plateforme de développement collaborative qui permet d'utiliser l'outil Git, d'héberger, versionner le développement, de garder un historique des modifications de tous les fichiers et donc de gérer des projets dans leur ensemble. C'est un outil réactif et accessible qui offre la possibilité à toute une équipe de travailler sur un même projet, le tout de manière organisée.

- MongoDB : est une base de données NoSQL orientée document. Elle se distingue des bases de données relationnelles par sa flexibilité et ses performances.

Passons à présent aux pour et au contre de l'utilisation de NodeJS.

Pour et contre de l'utilisation de NodeJS

Répertorions les avantages et les inconvénients du NodeJS que j'ai pu relevé au cours de ce projet au sein d'un un tableau :

Avantages	**Inconvénients**
NodeJS est basé sur un seul langage de programmation : le javascript. Ainsi, NodeJS nous permet d'utiliser du Javascript pour développer à la fois du front et du back. De plus, le Javascript n'est pas un langage très difficile à prendre en main comparé à d'autres.	L'utilisation du Javascript peut poser des problèmes. En effet, c'est un langage très permissif en ce qui concerne le typage des variables.
C'est un outil très rapide car il a l'avantage de fonctionner via le moteur V8 de Google. Cela lui permet de compiler le Javascript directement dans le code machine.	Il n'y a pas d'architecture prédéfinie, nous sommes obligés, pour chaque projet de télécharger les dépendances une à une.
Il possède une communauté très active permettant le partage de paquet NPM (eux même possède des outils et des modules autonomes que nous pouvons télécharger et utiliser).	En raison des différentes dépendances installées, NodeJS peut utiliser beaucoup d'espace disque.
Il permet de déployer rapidement et facilement des applications web en temps réel.	Étant donné que le NodeJS se base sur le Javascript qui est un langage peu optimisé, la machine tend à être beaucoup sollicitée en fonction des opérations effectuées.

Désormais, intéressons-nous aux avantages et aux inconvénients d'ExpressJS.

Pour et contre de l'utilisation d’ExpressJS

Pour cette partie, nous allons également regrouper nos informations au sein d’un tableau récapitulatif :

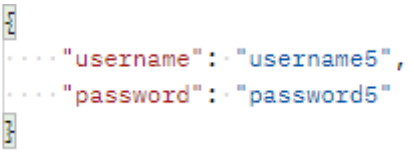
Avantages	**Inconvénients**
Express est un framework extrêmement flexible, minimaliste et se complète avec des modules disponible dans les dépôts npm. Ainsi, il nous laisse la liberté d’ajouter les librairies et modules de notre choix.	Avec Express les fichiers sont lus de manière séquentielle, c’est-à-dire de haut en bas et ceci peut amener à de potentiels conflits si l’une des routes est mal positionnée dans un des fichiers.
Il nous permet de construire rapidement une application web sur la plateforme NodeJS.	
Il permet de différencier les différents types de méthode CRUD (create, read, update, delete) pour chaque routes différentes.	

Dans la suite de ce rapport, analysons les différentes fonctionnalités implémentées, l’architecture choisie et concentrons nous l’une des fonctionnalités bonus proposées.

Explication du code

Les fonctionnalités implémentées dans ce back sont les suivantes :

- Pour l'objet utilisateur :
  - Ajouter un utilisateur :
    - La route est : localhost:3000/api/users avec la méthode POST ;
    - Il faut donner son nom d'utilisateur et son mot de passe sous le format JSON :



  - Récupérer l'ensemble des utilisateurs ;
    - La route est : localhost:3000/api/users avec la méthode GET ;
  - Récupérer un utilisateur spécifique via son identifiant :
    - La route est : localhost:3000/api/users/:id avec la méthode GET (:id représente une variable correspondant à l'identifiant d'un utilisateur) ;
  - Supprimer un utilisateur via son identifiant :
    - La route est : localhost:3000/api/users/:id avec la méthode DELETE (:id représente une variable correspondant à l'identifiant d'un utilisateur) ;
  - Modifier un objet utilisateur via son identifiant également :
    - La route est : localhost:3000/api/users/:id avec la méthode PUT (:id représente une variable correspondant à l'identifiant d'un utilisateur) ;
    - Il faut donner son mot de passe et/ou son nom d'utilisateur sous le format JSON :

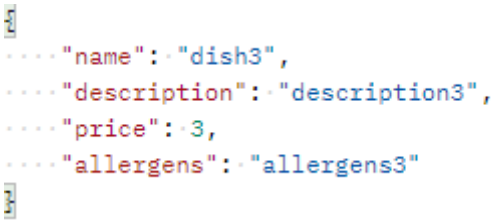
- Pour l’objet panier :
    - Ajouter un panier à un utilisateur :
        - La route est : localhost:3000/api/carts avec la méthode POST ;
        - Il faut donner un nom sous le format JSON :

```
{
    "name": "Panier"
}
```

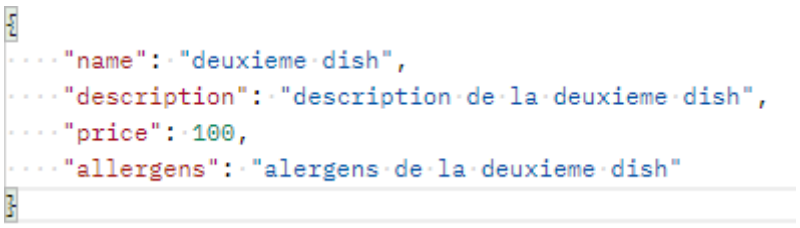
    - Récupérer l'ensemble des paniers d’un utilisateur :
        - La route est : localhost:3000/api/carts avec la méthode GET ;
    - Récupérer un panier spécifique via son identifiant :
        - La route est : localhost:3000/api/carts/:id avec la méthode GET (:id représente une variable correspondant à l'identifiant d'un panier) ;
    - Supprimer un panier via son identifiant :
        - La route est : localhost:3000/api/carts/:id avec la méthode DELETE (:id représente une variable correspondant à l'identifiant d'un panier) ;
    - Modifier un objet panier via son identifiant également :
        - La route est : localhost:3000/api/carts/:id avec la méthode PUT (:id représente une variable correspondant à l'identifiant d'un panier) ;
        - Il faut donner son nom et ou l’identifiant et la quantité du plat à ajouter sous le format JSON :

```
{
    "name": "Mon nouveau panier de Noel",
    "cart": [{
        "idDish": "61bb6b699348434170a883e1",
        "quantity": 3
    }]
}
```

- Pour l’objet plat :
  - Ajouter un plat :
    - La route est : localhost:3000/api/dishes avec la méthode POST ;
    - Il faut donner un nom, une description, un prix et ses éventuels allergènes sous le format JSON :

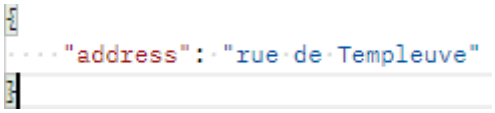


  - Récupérer l'ensemble des plats :
    - La route est : localhost:3000/api/dishes avec la méthode GET ;
  - Récupérer un plat spécifique via son identifiant :
    - La route est : localhost:3000/api/dishes/:id avec la méthode GET (:id représente une variable correspondant à l'identifiant d'un plat) ;
  - Récupérer un plat spécifique via un mot clé :
    - La route est : localhost:3000/api/dishes/search?keyword= suivi du mot clé avec la méthode GET ;
  - Supprimer un plat via son identifiant (celui-ci est également supprimé dans les paniers des utilisateurs le contenant) :
    - La route est : localhost:3000/api/dishes/:id avec la méthode DELETE (:id représente une variable correspondant à l'identifiant d'un plat) ;
  - Modifier un objet plat via son identifiant également :
    - La route est : localhost:3000/api/dishes/:id avec la méthode PUT (:id représente une variable correspondant à l'identifiant d'un plat) ;
    - Il faut donner un nom et ou une description et ou un prix et ou ses éventuels allergènes sous le format JSON :



- Pour l'authentification d'un utilisateur :
    - La route est : localhost:3000/api/auth avec la méthode GET ;
    - Il faut donner le nom de l'utilisateur et son mot de passe sous le format JSON afin de récupérer un token :

- Pour la validation d'un panier d'un utilisateur :
    - La route est : localhost:3000/api/carts/:id/checkout avec la méthode GET (:id représente une variable correspondant à l'identifiant d'un panier) ;
    - Il faut donner l'adresse de l'utilisateur associée au panier sous le format JSON afin de valider la commande d'un panier en particulier :



Intéressons-nous, désormais, en détail à l'architecture choisie. En effet, nous avons choisi d'utiliser une architecture N-tiers pour ce back en NodeJS afin d'obtenir un découpage intelligent permettant une bonne pérennisation du projet. En effet, celle-ci permet de rendre indépendant chaque partie, ou « couche », d'un projet donc dans notre cas, du back de Delivecrous et de nous laisser une certaine souplesse dans la gestion de ces couches.

Voici alors les différentes couches de cette architecture :

- Une couche « controller » : qui nous permettra de communiquer avec le web et qui permet, dans une implémentation plus complète, d'assurer les conversions entre modèle métier et DTO ;
- Une couche « service » : qui va nous permettre de gérer nos modèles métier (en fonction de la demande), celle-ci regroupe l'intelligence pure du projet ;
- Une couche « repository » : qui va nous permettre de communiquer avec la base de données et qui permet, dans une implémentation plus complète, d'assurer les conversions entre modèle métier et DAO.

L’idée de cette architecture est de pouvoir rendre facile toute modification du projet en cas de changement de l’une des couches puisqu’elles sont toutes indépendantes les unes des autres.

Expliquons à présent plus en profondeur la fonctionnalité qui permet à un utilisateur de s’authentifier.

Effectivement, l’utilisateur doit, pour gérer son ou ses paniers, se connecter. Un utilisateur peut se connecter avec son identifiant et son mot de passe. En effectuant ceci, la partie back du projet va alors retourner ce que nous appelons un token.

Pour retourner ce token, l'utilisateur est extrait de la base de données avec le nom fourni et la couche repository renverra le modèle qui contient toutes les informations de cet utilisateur. Ensuite, la méthode checkPassword qui se trouve dans UserService compare le mot de passe de notre modèle et le mot de passe que l'utilisateur a saisi. Par ailleurs, le mot de passe de la base de données a été chiffré plus tôt grâce à la bibliothèque bcrypt. De plus, c'est la méthode de comparaison de la bibliothèque bcrypt qui vérifie si les deux mots de passe correspondent. Ce n’est qu’à la suite de cela que le token est retourné, grâce à la librairie JsonWebToken (seulement si les mots de passe correspondent). Attention, un token possède une certaine durée.

Une fois que l’utilisateur a récupéré son token il doit, pour l’instant, le fournir dans l'entête d'autorisation de la requête http pour pouvoir faire une action particulière. Celui-ci est bien évidemment vérifié au préalable car il peut être erroné ou tout simplement expiré. Pour le vérifier, c’est la librairie JsonWebToken qui est utilisée et plus précisément la méthode verify, qui retourne l'état du token. Le token est revérifié à chaque fois qu'un utilisateur souhaite faire quelque chose qui nécessite une authentification.

Maintenant que nous avons fait le tour sur ce projet Delivecrous, passons à une conclusion regroupant ce que ce projet nous a apporté et les éventuelles améliorations envisageables.

Conclusion

Pour conclure sur ce projet, nous pouvons dire qu'il a été très enrichissant car il nous a permis de réaliser un service web et plus particulièrement le back d'un site web du début à la fin avec un cahier des charges à respecter. Il nous a également permis de mettre en pratique l'ensemble des notions qui nous ont été donnés en cours. Ainsi, malgré la dose de travail durant ce premier semestre, nous sommes satisfait d'avoir pu rendre un back plus que fonctionnel, ainsi que de ce que nous a apporté ce module tout au long du semestre. De plus, ce projet représente aussi pour nous l'opportunité de pouvoir recréer un back entier de manière plus instinctive.

Néanmoins, comme tout projet, nous pouvons bien évidemment envisager de multiples améliorations. Effectivement, nous avons évoqué dans ce rapport l'utilisation d'une architecture N-tiers mais si nous avions été un petit peu plus loin, nous aurions pu renforcer cette idée de rendre indépend les diverses couches en utilisant des DTO ("Data Transfert Object"), des modèles métiers, des DAO ("Data Access Object") et en faisant les conversions qui leurs sont associées. Pour renforcer une nouvelle fois cette idée, nous aurions pu notamment mettre en place de l'injection de dépendances. Nous aurions pu également mettre en place une meilleure gestion des éventuelles erreurs, ou instaurer un système d'administrateur qui aurait des identifiants particuliers et qui aurait la main sur l'ensemble des objets de ce back (sur les utilisateurs ou les plats par exemple). Ainsi nous avons pu voir à travers ce projet de nombreuses notions, mais nous nous sommes également rendu compte que nous pouvions toujours pousser plus loin l'amélioration de celui-ci.