LOG4420 – Conception de sites web dynam. et transact.

Travail pratique 4

Chargés de laboratoire:

Antoine Béland

Emilio Rivera

Automne 2018

Département de génie informatique et génie logiciel

1 Objectifs

Le but de ce travail pratique est de vous familiariser avec l'environnement du côté serveur en utilisant le *framework* express et l'engin Pug (anciennement Jade). De plus, vous aurez à vous familiariser avec les services web et avec l'utilisation d'une base de données avec MongoDB.

Plus particulièrement, vous aurez à migrer votre projet que vous avez réalisé au travail pratique 3 vers le *framework* express tout en utilisant Pug pour réaliser le rendu des différentes vues du site web. De plus, vous aurez à définir plusieurs API pour manipuler les données utilisées par le site web (panier d'achats, produits et commandes) et vous aurez à utiliser MongoDB pour gérer ces mêmes données.

2 Introduction

Lors des trois premiers travaux pratiques, vous aviez à mettre en place un site web qui était uniquement géré du côté client. En effet, le traitement des données se faisait grâce à des scripts JavaScript qui étaient exécutés sur le navigateur web du client. En ce sens, il n'était pas possible de garantir l'intégrité des données puisque l'usager pouvait modifier les scripts ou le contenu des variables dans son navigateur web pour changer le comportement du site. Ainsi, cela posait plusieurs problèmes au point de vue de la sécurité.

Afin de s'assurer de l'intégrité des données manipulées, il est nécessaire de gérer les données du côté serveur. En effet, cela permet de masquer le traitement des données tout en cachant certaines données sensibles qui ne doivent pas être divulguées aux différents clients. Le présent travail pratique vous permettra donc de vous initier aux différents aspects et avantages de la programmation du côté serveur.

3 Travail à réaliser

À partir du code que vous avez produit lors des travaux pratiques précédents, vous aurez à réaliser l'application du côté serveur pour le site web d'achats en ligne. Dans le cas où votre travail pratique 3 n'aurait pas été bien réussi, vous pouvez utiliser le corrigé du TP3 qui se trouve sur Moodle afin de partir du bon pied pour ce quatrième travail. Il est à noter que

les scripts du côté client vous sont fournis. Ainsi, vous n'aurez pas à mettre à jour les scripts que vous avez écrits au TP3, afin de communiquer avec les API à réaliser.

Avant de débuter, assurez-vous d'avoir récupéré l'archive associée à ce travail pratique sur Moodle. Cette archive contient le squelette d'une application utilisant le *framework* express.

3.1 Migration du projet vers express

Une fois l'archive associée à ce travail pratique téléchargée et décompressée, la première étape à réaliser est d'installer toutes les dépendances nécessaires au fonctionnement du *framework*. Pour ce faire, tapez la commande suivante dans un terminal à la racine du projet :

```
npm install
```

Une fois toutes les dépendances installées, il vous suffit de taper la commande suivante dans le terminal pour démarrer l'application :

```
npm start
```

Si tout se passe comme prévu, l'application *express* devrait être fonctionnelle à l'adresse `http://localhost:8000`. De plus, la page d'accueil devrait indiquer le message suivant : « Ça semble fonctionner ! ». Il est à noter que toutes les ressources nécessaires au bon fonctionnement de l'application client sont fournies dans le dossier « `public` » (images, feuilles de style, scripts, etc.).

3.2 Définition de *templates* avec Pug

Comme vous l'avez sans doute remarqué, une portion importante de votre code HTML était répétée sur chacune des pages du site web. En effet, seul le contenu principal était modifié de page en page ; l'entête et le pied de page restaient toujours les mêmes. L'utilisation de Pug permet justement d'éviter cette duplication de code en définissant des *templates*.

La deuxième étape de ce travail pratique consiste d'ailleurs à définir des *templates* pour les pages du site web à l'aide de Pug. Pour ce faire, vous devrez convertir votre code HTML en code Pug en vous basant sur la documentation du langage. Vous pouvez également utiliser des outils en ligne pour vous assister dans votre conversion de code, tel que celui-ci. Afin de

vous assurer du bon fonctionnement du site web avec les scripts du côté client qui vous sont fournis, veuillez réutiliser **le code des pages HTML du corrigé du TP3** afin de vous assurer de la compatibilité de ces mêmes scripts avec les vues.

Assurez-vous de créer vos fichiers `.pug` dans le dossier « `views` ». De plus, vous devez définir les éléments qui ne changent pas d'une page web à l'autre (entête, pied de page, inclusions de styles ou des scripts, etc.) dans le fichier « `layout.pug` ». Cela permettra donc de régler le problème du code dupliqué. Vous pouvez consulter ce lien pour en savoir plus.

Vous devez également définir certaines routes précises pour accéder aux différentes pages du site. Pour ce faire, vous devez ajouter les routes suivantes dans le fichier « `index.js` » se trouvant dans le dossier « `routes` » :

1. « `/` » et « `/accueil` » (page d'accueil) ;
2. « `/produits` » (page des produits) ;
3. « `/produits/:id` » (page d'un produit) ;
4. « `/contact` » (page de contact) ;
5. « `/panier` » (page du panier d'achats) ;
6. « `/commande` » (page de commande) ;
7. « `/confirmation` » (page de confirmation).

Contrairement au travail pratique 3 où vous deviez effectuer le rendu initial d'une page du côté client, vous devrez faire le rendu initial d'une page du côté serveur. Ainsi, les éléments suivants devront être réalisés avec Pug en passant des variables aux vues tout en effectuant des conditions et des boucles :

1. Le nombre de produits ajoutés au panier d'achats indiqué dans l'entête lorsqu'une page est chargée ;
2. La liste des produits à afficher par défaut lorsque la page des produits est chargée ;
3. Les informations associées au produit à afficher lorsque la page d'un produit est chargée ;
4. Les éléments se trouvant dans le panier d'achats lorsque la page du panier est chargée ;
5. Le numéro de confirmation et le nom du client lorsque la page de confirmation est chargée.

Notez bien

Assurez-vous de définir l'attribut « `data-product-id` » sur l'élément « `form` » de la page d'un produit en lui associant comme valeur l'identifiant du produit qui est rendu sur la page (p. ex. `data-product-id="1"`).

Également, assurez-vous que l'attribut « `data-product-id` » est défini avec l'identifiant du bon produit pour chacun des éléments « `tr` » se trouvant dans l'élément « `tbody` » sur la page du panier d'achats.

Les autres éléments qui étaient demandés lors du TP3 resteront du côté client. Comme cela a été mentionné plus tôt, tous les scripts du côté client vous sont fournis. Consultez l'annexe A pour connaître les scripts à inclure sur vos pages afin que le site web soit fonctionnel. Enfin, il est conseillé de réaliser les différentes fonctionnalités demandées une fois que vous aurez complété la logique nécessaire pour récupérer vos données qui seront utilisées par vos API (voir section suivante). Il est à noter que **vous ne devez pas lancer de requêtes AJAX** vers vos API pour récupérer les données nécessaires pour le rendu initial d'une vue avec Pug.

Il vous est également demandé de gérer les éléments simples des pages web (titre de la page, menu de navigation actif, etc.) du côté serveur grâce à des variables passées aux vues. Par ailleurs, n'oubliez pas de vérifier que les liens vers les différentes ressources du site web sont toujours valides et fonctionnels.

3.3 Création de services web

La troisième et dernière étape de ce travail est de mettre en place plusieurs services web. Les sites d'achats en ligne intègrent habituellement une section administrative qui permet aux gestionnaires des sites de consulter et de modifier facilement les données. Afin de vous simplifier la tâche, vous aurez uniquement à implémenter certaines API qui permettront de faire la gestion du site. Il est à noter que cette partie est la plus importante puisque vous réutiliserez intégralement les services web que vous réaliserez lors du travail pratique 5.

Avant d'aller plus loin, suivez les instructions se trouvant à l'annexe B de ce document afin de mettre en place une base de données MongoDB sur mLab. En effet, vous aurez besoin de communiquer avec MongoDB pour manipuler les produits et les commandes du site web. Pour ce faire, vous devrez utiliser Mongoose (la dépendance est déjà installée) pour manipuler la base de données via l'application côté serveur. Vous pouvez consulter ce tutoriel pour savoir comment utiliser Mongoose avec une application Node.js.

Le fichier « `db.js` » se trouvant dans le répertoire « `lib` » vous fournit les lignes de code pour configurer Moogose. Les schémas permettant de définir le format d'enregistrement d'un produit et d'une commande vous sont fournis (ne pas les modifier). Assurez-vous de mettre à jour le *connect string* (`mongodb://...`) afin que vous soyez en mesure de vous connecter à votre base de données sur mLab. Il est à noter que le fichier « `db.js` » est déjà inclus dans le fichier « `app.js` ».

Assurez-vous de définir les routes décrites par les API dans le dossier « `routes` » de l'application. N'hésitez pas à ajouter de nouveaux fichiers dans ce dossier afin de séparer les différentes API (p. ex. un fichier par service web). Également, il est conseillé de séparer la logique permettant de récupérer les données (MongoDB) de celle liée aux routes. Cela vous facilitera la tâche pour récupérer les données à utiliser pour le rendu de vos vues. Enfin, vous êtes libres d'utiliser des paquets sur npm pour vous aider à réaliser vos services web.

Les sous-sections qui suivent décrivent les services web qui devront être mis en place. Il est à noter que toutes les données qui seront spécifiées aux API devront être validées du côté serveur. En effet, afin d'assurer l'intégrité de ces mêmes données, il est important de s'assurer que les données manipulées sont valides. Pour effectuer cette validation, vous pouvez utiliser des paquets sur npm pour vous aider (p. ex. `validator`).

? Comment tester vos API ?

Un moyen simple de tester vos services web est d'utiliser Postman. De plus, des tests d'intégration vous sont fournis afin que vous puissiez tester vos API (voir la section §Exécution de tests d'intégration et d'acceptation).

3.3.1 Produits

La première API à réaliser est celle qui permet la manipulation des différents produits disponibles sur le site web. Contrairement au travail pratique 3 où vous aviez à faire une requête AJAX vers le fichier « `products.json` » pour récupérer la liste des produits, le client devra effectuer des requêtes à l'API suivante pour gérer les produits sur le site web. En ce sens, le code du côté client a été modifié pour utiliser cette nouvelle API. En plus de récupérer les produits disponibles, cette API permettra de créer de nouveaux produits et d'en supprimer. De plus, les produits devront être enregistrés dans la base de données MongoDB. En ce sens, toutes les modifications que vous effectuerez sur les produits devront être enregistrées dans la base de données. Les routes à définir pour cette API sont les suivantes :

1. **GET** `/api/products`

   Obtient tous les produits se trouvant dans la base de données. Par défaut, les produits sont triés en ordre croissant de prix.

   Paramètres d'URL

`category`	La catégorie de produit à utiliser pour filtrer la liste à retourner. Par défaut, lorsque ce paramètre n'est pas spécifié, tous les produits doivent être présents dans la liste retournée. Les catégories possibles sont les suivantes : — `cameras` : appareils photo ; — `computers` : ordinateurs ; — `consoles` : consoles de jeu ; — `screens` : écrans.
`criteria`	Le critère de tri qui doit être utilisé pour trier la liste de produits qui doit être retournée. Par défaut, lorsque ce paramètre n'est pas spécifié, le critère de tri à utiliser est « `price-asc` ». Les valeurs possibles pour le critère sont les suivantes : — `alpha-asc` : nom des produits en ordre alphabétique ; — `alpha-dsc` : nom des produits en ordre alphabétique inverse ; — `price-asc` : prix des produits en ordre croissant ; — `price-dsc` : prix des produits en ordre décroissant.

   Réponses

`200 (OK)`	Retourne une liste contenant les produits répondants aux paramètres spécifiés. Le format demandé pour un produit est illustré à l'annexe C. S'il n'y a aucun produit dans la base de données, une liste vide est retournée.
`400 (Bad Request)`	Indique que les valeurs associées pour un ou plusieurs paramètres d'URL sont invalides (p. ex. critère ou catégorie invalides).

2. **GET** `/api/products/`**`:id`**

   Obtient le produit associé à l'identifiant spécifié (**`:id`**) qui se trouve dans la base de données. Le produit retourné doit être de la même forme que l'exemple illustré à l'annexe C.

   Réponses

`200 (OK)`	Retourne le produit se trouvant dans la base de données associé à l'identifiant spécifié au format JSON.
`404 (Not Found)`	Indique que l'identifiant spécifié n'est pas associé à un produit se trouvant dans la base de données.

3. **POST** `/api/products/`

   Crée un nouveau produit dans la base de données.

   Paramètres *Body* (format JSON)

`id`	L'identifiant du produit (nombre entier **unique**).
`name`	Le nom du produit (chaîne de caractères non vide).
`price`	Le prix du produit (nombre réel positif).
`image`	L'image associée au produit (chaîne de caractères non vide).
`category`	La catégorie du produit. Les valeurs possibles sont les suivantes : `cameras`, `computers`, `consoles`, `screens`.
`description`	La description associée au produit (chaîne de caractères non vide).
`features`	La liste de caractéristiques du produit (liste de chaînes de caractères non vides).

   Réponses

`201 (Created)`	Indique que le produit a été créé dans la base de données.
`400 (Bad Request)`	Indique qu'un ou plusieurs paramètres spécifiés sont invalides (p. ex. erreur de type, erreur de logique, valeurs invalides, identifiant déjà utilisé par un autre produit dans la base de données, etc.).

4. **DELETE** `/api/products/:id`

   Supprime le produit associé à l'identifiant spécifié (**`:id`**) dans la base de données.

   Réponses

`204 (No Content)`	Indique que le produit associé à l'identifiant spécifié a été supprimé de la base de données.
`404 (Not Found)`	Indique que l'identifiant spécifié n'est pas associé à un produit se trouvant dans la base de données.

5. **DELETE** `/api/products/`

   Supprime tous les produits qui se trouvent dans la base de données.

   Réponse

`204 (No Content)`	Indique que tous les produits ont été supprimés de la base de données.

3.3.2 Panier d'achats

La deuxième API à réaliser est celle qui permet de gérer le panier d'achats. Contrairement au troisième travail pratique où les informations du panier d'achats étaient conservées du côté client grâce au *Local Storage*, ce travail pratique demande de gérer le panier du côté serveur avec l'aide d'une session (la dépendance `express-session` est déjà installée et a été

configurée dans le fichier « app.js »). En ce sens, vous devez mettre en place l'API suivante tout en conservant les données du panier dans une variable de session qui se trouve du côté serveur. Il est à noter que la logique du côté client a été modifiée pour vous afin que le site web utilise cette API afin de conserver les produits ajoutés au panier. Voici les routes que vous devez définir dans votre application :

1. **GET** `/api/shopping-cart`

   Obtient tous les éléments se trouvant dans le panier.

   Réponse

`200 (OK)`	Retourne une liste d'éléments se trouvant dans le panier au format JSON. Lorsqu'aucun produit ne se trouve dans le panier, une liste vide (`[]`) est retournée. Le format demandé pour la liste est le suivant : `[ { "productId": #id, "quantity": #quantity }, ... ]`

2. **GET** `/api/shopping-cart/`**`:productId`**

   Obtient l'élément associé à l'identifiant du produit (**`:productId`**) se trouvant dans le panier.

   Réponses

`200 (OK)`	Retourne l'élément associé à l'identifiant spécifié au format JSON. Le format demandé pour l'objet est le suivant : `{ "productId": #id, "quantity": #quantity }`
`404 (Not Found)`	Indique que l'identifiant spécifié n'est pas associé à un élément qui se trouve dans le panier.

3. **POST** `/api/shopping-cart/`

   Ajoute un élément dans le panier d'achats. Assurez-vous que l'identifiant du produit spécifié est associé à un produit dans la base de données.

   Paramètres *Body* (format `JSON`)

`productId`	L'identifiant du produit à ajouter (nombre entier).
`quantity`	La quantité liée au produit (nombre entier positif).

   Réponses

`201 (Created)`	Indique que l'item a été ajouté au panier.
`400 (Bad Request)`	Indique que l'identifiant (`productId`) spécifié n'existe pas, que le produit associé à l'identifiant spécifié a déjà été ajouté dans le panier ou que la quantité spécifiée est invalide.

4. **PUT** `/api/shopping-cart/:productId`

   Met à jour la quantité associée au produit se trouvant dans le panier qui est lié à l'identifiant du produit spécifié (**`:productId`**).

   Paramètre *Body* (format `JSON`)

`quantity`	La quantité liée au produit spécifié à mettre à jour (nombre entier positif).

   Réponses

`204 (No Content)`	Indique que la quantité associée à l'identifiant spécifié a été mise à jour dans le panier.
`404 (Not Found)`	Indique que l'identifiant spécifié n'existe pas dans le panier.
`400 (Bad Request)`	Indique que la quantité spécifiée est invalide.

5. **DELETE** `/api/shopping-cart/:productId`

   Supprime l'élément associé à l'identifiant (**`:productId`**) spécifié qui se trouve dans le panier.

   Réponses

`204 (No Content)`	Indique que l'item associé à l'identifiant spécifié a été supprimé dans le panier.
`404 (Not Found)`	Indique que l'identifiant spécifié n'existe pas dans le panier.

6. **DELETE** `/api/shopping-cart/`

   Supprime tous les éléments qui se trouve dans le panier.

   Réponse

`204 (No Content)`	Indique que tous les éléments qui se trouvaient dans le panier ont été supprimés.

3.3.3 Commandes

La dernière API à réaliser est celle qui permet de gérer les commandes. Lors du troisième travail pratique, seuls le prénom et le nom du client étaient conservés dans le *Local Storage*. Cela était bien entendu insuffisant puisque beaucoup d'informations étaient manquantes (adresse courriel, numéro de téléphone, produits achetés, etc.) pour une commande. Cette API permettra donc de créer, d'accéder et de supprimer des commandes. De plus, toutes ces commandes devront être enregistrées dans la base de données MongoDB. Il est à noter que le script du côté client a été mis à jour afin que les commandes soumises sur le site

web puissent être enregistrées dans la base de données en communiquant avec cette API. Les routes à créer pour cette API sont les suivantes :

1. **`GET`** `/api/orders`

   Obtient toutes les commandes se trouvant dans la base de données.

   Réponse

`200 (OK)`	Retourne une liste contenant toutes les commandes. Le format demandé pour une commande est illustré à l'annexe C. S'il n'y a aucune commande dans la base de données, une liste vide est retournée.

2. **`GET`** `/api/orders/`**`:id`**

   Obtient la commande associée à l'identifiant spécifié (**`:id`**) qui se trouve dans la base de données. La commande retournée doit être de la même forme que l'exemple montré à l'annexe C.

   Réponses

`200 (OK)`	Retourne la commande se trouvant dans la base de données associée à l'identifiant spécifié au format JSON.
`404 (Not Found)`	Indique que l'identifiant spécifié n'est pas associé à une commande se trouvant dans la base de données.

3. **`POST`** `/api/orders/`

   Crée une nouvelle commande dans la base de données. Assurez-vous que les produits spécifiés existent dans la base de données.

   Paramètres *Body* (format `JSON`)

`id`	L'identifiant de la commande (nombre entier **unique**). L'identifiant correspond au numéro de commande.
`firstName`	Le prénom du client (chaîne de caractères non vide).
`lastName`	Le nom du client (chaîne de caractères non vide).
`email`	L'adresse courriel du client (adresse courriel valide).
`phone`	Le numéro de téléphone du client (numéro de téléphone valide).
`products`	Une liste des produits achetés par le client. Chacun des éléments de la liste doit posséder les propriétés suivantes : — `id` : L'identifiant du produit (nombre entier) ; — `quantity` : la quantité de produits qui a été achetée (nombre entier positif).

Réponses

`201 (Created)`	Indique que la commande a été créée dans la base de données.
`400 (Bad Request)`	Indique qu'un ou plusieurs paramètres spécifiés sont invalides (p. ex. erreur de type, erreur de logique, identifiant déjà utilisé par une autre commande, identifiant de produit invalide, etc.).

4. **DELETE** `/api/orders/`**`:id`**

   Supprime la commande associée à l'identifiant spécifié (**`:id`**) dans la base de données.

   Réponses

`204 (No Content)`	Indique que la commande associée à l'identifiant spécifié a été supprimée de la base de données.
`404 (Not Found)`	Indique que l'identifiant spécifié n'est pas associé à une commande se trouvant dans la base de données.

5. **DELETE** `/api/orders/`

   Supprime toutes les commandes qui se trouvent dans la base de données.

   Réponse

`204 (No Content)`	Indique que toutes les commandes ont été supprimées de la base de données.

3.4 Exécution de tests d'intégration et d'acceptation

Afin que vous puissiez valider vos API, des tests d'intégration vous sont fournis. Ainsi, ces tests seront utilisés par le correcteur pour évaluer vos services web. Il est donc important de valider que l'ensemble des tests fonctionnent **avant** que vous remettiez votre travail. Pour exécuter les tests d'intégration, veuillez entrer la commande suivante dans un terminal à la racine du projet :

```
npm test
```

Tout comme au travail pratique 3, des tests d'acceptation automatisés vous ont également été fournis afin que vous puissiez vérifier votre travail. En effet, ces tests permettront de valider que tous les changements que vous avez apportés sont valides. De plus, ces tests serviront à l'évaluation de votre travail. En ce sens, il est important que vous vous assuriez que tous les tests fonctionnent **avant** de remettre votre travail pratique. Pour exécuter les tests d'acceptation, vous devez entrer la commande suivante dans un terminal :

```
npm run e2e
```

Il est à noter que l'environnement de tests d'acceptation automatisés est actuellement configuré pour fonctionner dans le laboratoire de Polytechnique (système d'exploitation Linux). Si vous souhaitez exécuter les tests sur un autre environnement, vous pouvez jeter un coup d'œil à ce lien (voir fichier « `nightwatch.json` » dans le dossier « `tests` »).

Avertissement

Il est important de n'apporter **aucune** modification aux fichiers dans le dossier `tests` (sauf « `nightwatch.json` »). En effet, ces fichiers contiennent tous les tests automatisés qui sont exécutés sur le site web et se doivent de ne pas être modifiés.

Conseils pour la réalisation du travail pratique

1. Référez-vous aux tutoriels sur express et sur MongoDB sur le site de référence du cours.
2. Utilisez un débogueur pour vous aider à identifier plus rapidement les bogues dans vos API (voir Webstorm).
3. Utilisez Postman pour vérifier rapidement les routes de vos API.
4. Assurez-vous que votre base de données contient les bonnes valeurs. Pour ce faire, vérifiez l'état de votre base de données sur le site web de mLab.
5. N'attendez pas à la dernière minute pour commencer le laboratoire ! Le débogage de services web peut s'avérer long et ardu.

4 Remise

Voici les consignes à suivre pour la remise de ce travail pratique :

1. Vous devez placer le code de votre projet dans un dossier compressé au format ZIP nommé « `TP4_matricule1_matricule2.zip` ». Assurez-vous d'exclure les dossiers « `node_modules` » et « `tests` » avant de remettre votre travail.
2. Vous devrez également créer un fichier nommé « `temps.txt` » à l'intérieur du dossier de votre projet. Vous indiquerez le temps passé au total pour ce travail.
3. Le travail pratique doit être remis avant **23h55**, le **19 novembre 2018** sur Moodle.

Aucun retard ne sera accepté pour la remise de ce travail. En cas de retard, le travail se verra attribuer la note de zéro. Également, si les consignes 1 et 2 concernant la remise ne sont pas respectées, une pénalité de -5% est applicable.

Le navigateur web **Google Chrome** sera utilisé pour tester votre site web.

À vérifier

Avant de remettre votre travail, assurez-vous que tous les paquets de npm que vous avez installés dans votre projet sont listés dans le fichier « `package.json` ». Si vous avez des dépendances manquantes, assurez-vous de les installer avec l'argument « `--save` ».

5 Évaluation

Globalement, vous serez évalué sur le respect des exigences des API à réaliser. Plus précisément, le barème de correction est le suivant :

Exigences	Points
Utilisation convenable de Pug	4
Création de services web	
API des produits	4
API du panier d'achats	4
API des commandes	4
Bon fonctionnement du site web avec express	
Résultats des tests d'acceptation automatisés	4
Total	20

L'évaluation se fera en grande partie grâce aux tests d'intégration et d'acceptation qui vous sont fournis.

Ce travail pratique a une pondération de **6 %** sur la note du cours.

6 Questions

Si vous avez des interrogations concernant ce travail pratique, vous pouvez poser vos questions sur le canal `#tp4` sur Slack. N'hésitez pas à poser vos questions sur ce canal afin qu'elles puissent également profiter aux autres étudiants.

Annexes

A Scripts à inclure sur chacune des pages du site web

Afin de simplifier votre travail, tous les scripts du côté client vous sont fournis pour ce travail pratique. Vous devez donc uniquement vous attarder sur la partie côté serveur. Bien entendu, vous devez inclure les scripts selon un certain ordre pour que le site web puisse fonctionner. Ainsi, la liste suivante énumère les scripts qui devront être inclus sur **toutes les pages** de votre site web, en conservant l'ordre indiqué :

1. « `/assets/js/jquery-3.2.1.min.js` »
2. « `/assets/scripts/utils.js` »
3. « `/assets/scripts/products.service.js` »
4. « `/assets/scripts/shopping-cart.service.js` »
5. « `/assets/scripts/shopping-cart.controller.js` »

En plus des scripts qui doivent être inclus sur toutes les pages, certaines d'entre elles doivent inclure d'autres scripts. En ce sens, le tableau 1 illustre les scripts supplémentaires à inclure pour les pages des produits et du panier d'achats.

Tableau 1 – Scripts supplémentaires à inclure

Page	Scripts à inclure
Produits	« `/assets/scripts/products.controller.js` »
Commande	1. « `/assets/js/jquery.validate.min.js` » 2. « `/assets/js/additional-methods.min.js` » 3. « `/assets/js/messages_fr.js` » 4. « `/assets/scripts/orders.service.js` » 5. « `/assets/scripts/order.controller.js` »

B Créer une base de données avec mLab

Voici les étapes à suivre afin de mettre en place une base de données avec mLab :

1. Rendez-vous sur le site web de mLab pour vous créer un compte. Assurez-vous de choisir un mot de passe que vous pourrez partager avec votre coéquipier et le chargé de laboratoire.

2. Une fois votre compte créé, vous recevrez un courriel de mLab afin de confirmer votre adresse courriel. Assurez-vous donc de cliquer sur le lien du courriel que vous avez reçu afin de confirmer votre adresse courriel.

3. Une fois votre adresse courriel confirmée auprès de mLab, il vous sera possible de créer une base de données MongoDB. Pour ce faire, vous devez cliquer sur le bouton « *Create new* » se trouvant dans la section « *MongoDB Deployments* ».

4. Lorsque vous aurez cliqué sur le bouton « *Create new* », vous serez dirigé vers un formulaire qui permettra de créer la base de données. Assurez-vous de sélectionner « Amazon Web Services » pour le « *Cloud Provider* » et « ***Sandbox*** » pour le « *Plan Type* ». Par la suite, cliquez sur le bouton « *Continue* ».

5. La page suivante du formulaire vous demandera de sélectionner une région pour la base de données. Assurez-vous de sélectionner la région la plus près du Canada parmi les choix afin de réduire la latence puis cliquez sur le bouton « *Continue* ».

6. La page suivante vous invitera à saisir un nom pour la base de données. Saisissez un nom (p. ex. *online-shop*) puis cliquez sur le bouton « *Continue* ».

7. Une fois arrivé sur la dernière page du formulaire, cliquez sur le bouton « *Submit Order* » afin de créer la base de données. Vous serez alors redirigé vers la liste des « *MongoDB Deployments* ». Si tout s'est passé comme prévu, le nom de votre base de données devrait être affiché dans cette liste.

8. Cliquez sur le nom de votre base de données pour accéder aux paramètres de celle-ci. Par la suite, cliquez sur l'onglet « *Users* » puis sur le bouton « *Add database user* » pour créer un utilisateur pouvant se connecter à la base de données. Remplissez le formulaire puis appuyez sur le bouton « *Create* ». Assurez-vous de saisir un mot de passe qui pourra être partagé entre plusieurs personnes.

C Format des objets de l'API

Cette annexe illustre le format attendu pour les objets retournés par les API que vous avez à implémenter.

Produit

```
{
  "id": 14,
  "name": "Nom du produit",
  "price": 99.99,
  "image": "image.png",
  "category": "computers",
  "description": "La description du produit...",
  "features": [
    "Caractéristique 1",
    "Caractéristique 2",
    "Caractéristique 3"
  ]
}
```

Commande

```
{
  "id": 1,
  "firstName": "Antoine",
  "lastName": "Béland",
  "email": "antoine.beland@polymtl.ca",
  "phone": "514-340-4711",
  "products": [
    {
      "id": 1,
      "quantity": 2
    },
    {
      "id": 2,
      "quantity": 1
    }
  ]
}
```