

Journal pour le TP3

Travail présenté à Mohammed Salim Meflah

Dans le cadre du cours Application web (420-2CW-BB gr. 00003)

Par Mohamed Imadeddine Ait Benhammou

DA : 6238360

Le 30 mai 2024

Table des matières

[Le travail effectué 3](#_Toc167992385)

[Le travail effectué en fonction de la date 4](#_Toc167992386)

[Les difficultés rencontrées 6](#_Toc167992387)

[Les notions apprises pendant le travail 8](#_Toc167992388)

[La commande « git reflog » 10](#_Toc167992389)

# 

# Le travail effectué

Tout d’abord, j’ai commencé par faire la conception du modèle logique de la base de données du site web. Alors, j’ai décidé de créer cinq tables, soit la table « magasin » qui représente les différents magasins et qui contient leurs informations, la table « client » qui représente les clients des magasins, la table « produit » qui représente les produits que le magasin vend, la table « panier » qui correspond au panier qui peut contenir des produits du client et la table « facture » qui contient des informations sur l’achat effectué par le client.

Ensuite, j’ai créé le script d’insertion des données dans la base de données. Celles-ci seront affichées dynamiquement dans le site web en utilisant la méthode « fetch » en javascript. De plus, j’ai aussi créé le script afin d’activer l’API REST.

Par la suite, j’ai créé un dépôt distant sur GitHub nommé « 2cw-tp3-tp3\_mohamed\_aitbenhammou\_ayoub\_yassine » et j’ai ajouté les membres de mon équipe dans celui-ci. Le dépôt est utilisé afin de faire le site web pour le troisième travaille pratique. De plus, j’ai utilisé la commande « git clone HTTPS du dépôt distant » afin de cloner le dépôt distant dans ma machine locale, ce qui a créé un fichier « .git », qui représente la base de donnée pour le système de gestion de versions. Par ailleurs, j’ai créé deux répertoires nommés « Livrable1 » et « Livrable2 » et dans le répertoire « Livrable2 », j’ai créé le répertoire « Site\_Web » qui contient aussi le répertoire « html », qui contient les pages html, le répertoire « css », qui contient le fichier de style pour toutes les pages et le répertoire « javascript » qui contient le fichier javascript pour toutes les pages. De cette manière, le projet est organisé.

Dans mon dépôt local, j’ai créé trois branches. La première branche créée est la branche « MohamedAitBenhammou », qui est utilisée pour tout le travail effectuer sur les pages html. La deuxième branche créée est la branche « CSS », qui est utilisée pour le style de toutes les pages html. La dernière branche créée est la branche « Javascript », qui est utilisée lorsque je travaille dans le ficher « script.js ». Après avoir travaillé dans une des branches mentionnées ci-haut, je partais dans la branche « master » en utilisant la commande « git checkout master » et je fusionnais la branche sur laquelle je viens de travailler avec la branche « main » en utilisant la commande « git merge nomBranche », où « nomBranche » est le nom de la branche où je viens de travailler. Finalement, après avoir fait plusieurs « commits », j’utilisais la commande « git push » afin d’envoyer les changements effectués au projet dans le dépôt distant. Ainsi, les membres de mon équipe pouvaient utiliser la commande « git pull » afin d’importer les changements dans leurs dépôts locaux.

De surcroît, j’ai travaillé dans toutes les pages du site web, soit la page « index.html », la page « facture.html », la page « magasin.html », la page « panier.html » et la page « produits.html. ». J’avais aussi créé deux pages, soit les pages « inscription.html » et « connexion.html » que j’ai décidé d’enlever, car elles ne sont pas très utiles, puisque je ne peux pas utiliser l’option « POST » de la méthode « fetch » en javascript.

Finalement, j’ai créé tous les répertoires pour le site web, soit l’architecture de base du site web, et j’ai fait le document « Livrable1 » qui présente le site web au complet.

# Le travail effectué en fonction de la date

|  |  |
| --- | --- |
| Date | Travail effectué |
| LE 11 mai 2024 | Création de la base de données (modèle logique). |
| Le 12 mai 2024 | Activation de l’API REST. |
| Le 18 mai 2024 | Ajout de la table « magasin » dans la base de données, création du script d’insertion des données et activation de l’API REST. |
| LE 19 mai 2024 | Création du dépôt distant, ajout des membres de l’équipe dans le projet GitHub, création du dépôt local et création des branches « MohamedAitBenhammou », « CSS » et « Javascript ». |
| LE 20 mai 2024 | Création de l’architecture de base du site web, soit les répertoires « html », « css » et « javascript » et création des fichiers « index.html », « facture.html », « magasin.html », « panier.html », « produits.html », « style.css » et « script.js ».  Finir la page « index.html. ». |
| le 21 mai 2024 | Création du répertoire « image » et création du fichier « .gitignore » afin d’ignorer toutes les images dans ce fichier. Ajout de toutes les images des produits dans le répertoire. |
| Le 24 mai 2024 | Création des méthodes « createNode », « append » et « fetch ». Relier les pages html aux fichier « style.css » et « script.js ». Création des barres de navigation dans les pages html.  Travailler sur la page « produits.html », afficher dynamiquement les images et les informations des produits. |
| Le 25 mai 2024 | Travailler sur l’affichage dynamique des magasins dans la page « magasin.html » en utilisant javascript, afficher l’identifiant du magasin, le nom du magasin, son adresse, son numéro de téléphone et ses heures d’ouverture. |
| Le 29 mai 2024 | Travailler sur l’affichage du panier du client, affichage des produits dans le panier, leurs noms, leurs prix et le magasin qui les livre. Affichage du résumé du panier, affichage du nombre de produits dans le panier, le coût total avant taxes, les taxes et le coût total après taxes. Création de deux boutons afin d’aller dans les pages « produits.html » et « facture.html ». Travailler dans la page « facture.html ». Afficher le numéro de la facture, le nom du magasin, le coût total avant taxes, les taxes, le coût total après taxes, la date d’achat un message de la part du magasin. |
| Le 30 mai 2024 | Ajout de commentaire dans le code, modification du style pour la page « panier.html ». Vérification du code et du bon fonctionnement du site web. |

# Les difficultés rencontrées

Pendant le travail, j’ai rencontré plusieurs difficultés lors de l’activation de l’API REST et aussi avec le fichier « script.js ».

La première difficulté que j’ai rencontrée est l’activation de l’API REST. En effet, lorsque j’essayais d’activer l’API REST, cela ne fonctionnait pas. Plus tard, après de nombreuses recherches, j’ai trouvé un site web d’Oracle[[1]](#endnote-1) qui m’a expliqué de quelle manière je devais activer l’API.

La deuxième difficulté que j’ai rencontrée est l’utilisation de l’url afin d’accéder au fichier « JSON ». En effet, après avoir activé l’API REST, lorsque j’allais sur l’url, je ne trouvais pas les données de la base de données. Après avoir demander de l’aide au professeur, j’ai compris le problème. Le problème était qu’il fallait d’abord insérer les données, puis activer l’API REST, puisque, une fois activée, l’url ne contiendra pas les nouvelles données ajoutées à la base de données. Donc, il faut d’abord créer la base de données, insérer les données dans celle-ci et finalement activer l’API REST. De cette manière, toutes les données insérées précédemment dans la base de données seront présentes dans le fichier « JSON » accessible avec l’url.

La troisième difficulté que j’ai rencontrée est l’utilisation de la méthode « fetch » en javascript. En effet, je ne comprenais pas la manière de l’utiliser, puisque je n’arrivais pas arrivais pas à afficher dynamiquement les données présentes dans la base de données dans mon site web. Alors, après de nombreuses recherches, j’ai compris comment l’utiliser grâce aux documents de cours et aussi grâce à un site web[[2]](#endnote-2). De cette manière, j’ai compris que la méthode « fetch » permet d’obtenir les données de la base de données en utilisant l’url que j’ai obtenu en activant l’API REST. De plus, j’ai aussi compris que la méthode « fetch » utilise l’option « GET » par défaut, mais qu’on peut aussi utiliser d’autres options comme l’option « POST ». Afin d’obtenir les données, il faut créer deux fonction « then » et une fonction « map » dans la deuxième fonction « then ».

La quatrième difficulté que j’ai rencontrée est l’utilisation de la méthode « window.prompt() » dans une page spécifique afin d’obtenir le numéro du client entré par l’utilisateur. En effet, lorsque j’avais créé cette méthode dans le fichier « script.js », elle s’appliquait à toutes les pages html liées au fichier javascript. Ensuite, après avoir cherché une solution, j’ai trouvé, dans le site web W3Schools[[3]](#endnote-3), l’événement « onload » qui est similaire à l’événement « onclick » qu’on a utilisé avec les boutons html. Alors, j’ai créé des fonctions en javascript pour chacune des pages html et dans la balise « body » de chaque page html, j’ai ajouté l’événement « onload » qui permet d’appeler une méthode javascript lorsque la page html se charge. De cette manière, j’ai pu régler mon problème et j’ai pu appeler une méthode javascript spécifique afin d’afficher les données dans une page spécifique.

La cinquième difficulté que j’ai rencontrée est l’affichage dynamique des champs dans la base de données. En effet, je ne savais pas comment les afficher d’une manière dynamique, mais j’ai compris plus tard comment le faire. La solution était de créer un élément de type « div » dans la page html où je veux afficher les champs de la base de données dynamiquement et lui donner un id. Ensuite, dans le fichier javascript, créer une constante qui contient cet objet. Finalement, dans la méthode « fetch » créer des éléments html dynamiquement en utilisant la fonction « createNode() » et les ajouter à la page html (dans l’élément de type « div » de la page html) en utilisant le fonction « append(parent, el) », où « parent » est l’élément de type « div » de la page html. Donc, tous les éléments créés dynamiquement en utilisant javascript seront placés dans l’élément de type « div » de la page html.

La sixième difficulté que j’ai rencontrée est le fusionnement des branches. En effet, j’avais un problème lorsque je fusionnais ma branche personnelle avec la branche « master », je n’arrivais pas à la faire. Cependant, j’ai réussi à le faire après avoir géré le conflit qui était causé, puisque la même partie du fichier a été modifié dans deux branches différentes. Alors, j’ai dû partir régler ce conflit dans Visual Studio Code et accepter les changements avant de fusionner les braches.

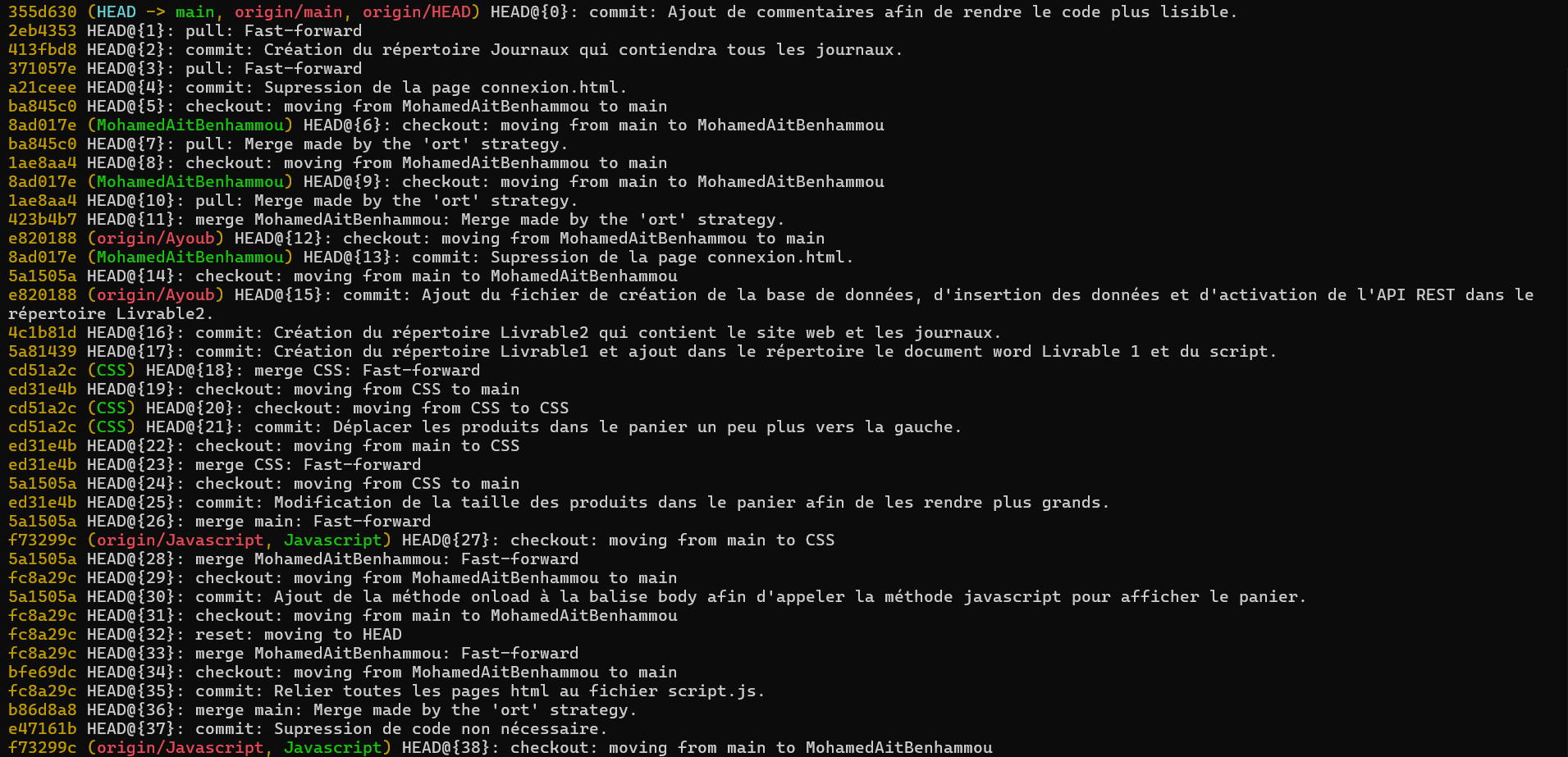
# Les notions apprises pendant le travail

Lors de ce travail pratique, j’ai appris plusieurs notions importantes, notamment pour l’activation de l’API REST et l’utilisation de javascript pour l’affichage dynamique des pages.

Les notions principales apprises sont :

* La fonction « createNode(element) »[[4]](#endnote-4): J’ai appris à utiliser cette méthode qui permet de créer des éléments html dynamiquement. Je l’ai utilisée afin d’afficher dynamiquement les magasins, les produits, les paniers et les factures en créent des éléments de type « h1 », « h2 », « div », « p » et plusieurs autres éléments html.
* La fonction « append(parent, el) »[[5]](#endnote-5) : J’ai appris à utiliser cette méthode qui permet de placer des éléments html dans d’autres éléments html. En effet, le paramètre « parent » contient l’élément où je veux ajouter un élément html et le paramètre « el » sera ajouté dans l’élément « parent ». Ainsi, j’ai pu créer des éléments html dynamiquement et les ajouter aux pages html afin d’afficher dynamiquement les magasins, les produits, les paniers et les factures.
* Accéder aux éléments du DOM (Document Object Model) avec « document.getElementById() »[[6]](#endnote-6) : J’ai appris la manière d’accéder aux élément du DOM afin de les modifier. Je peux stocker un élément html dans une variable javascript et modifier cet élément en utilisant le nom de la variable et en ajoutant l’aspect que je veux modifier.
* Donner une classe à un élément html dynamiquement[[7]](#endnote-7) : Lors de la création d’élément dynamiquement en utilisant la fonction « createNode(element) », je devais donner une classe à ces éléments afin de pouvoir modifier leurs styles dans le fichier « style.css ». Pour ce faire, j’ai du stocker l’élément créé dans une variable en ensuite utiliser le code suivant afin de donner une classe à l’élément créé : nomVar.className = « nom-classe ». De cette manière, j’ai pu modifier le style des éléments dynamiquement créés dans le fichier « style.css ».
* Modifier un élément html en utilisant javascript[[8]](#endnote-8) : J’ai appris la manière de modifier un élément html en utilisant un « innerHTML ». Ainsi, j’ai pu afficher les champs de la base de données dans les éléments html créé dynamiquement.
* L’activation de l’API REST[[9]](#endnote-9) : J’ai appris la procédure à suivre afin d’activer l’API REST, ce qui m’a permis d’utiliser le url de la table et d’afficher dynamiquement les informations présentes dans le tableau « JSON ».
* L’événement « onload »[[10]](#endnote-10) : J’ai appris à utiliser cette fonction dans les balises « body » des pages html. Celle-ci permet d’appeler une méthode javascript lorsque la page html est chargée, ce qui est très utile, puisque j’ai utilisé un seul fichier javascript pour toutes les pages html. Donc, cela m’a permis d’appeler seulement une méthode pour une page html.
* La méthode « fetch »[[11]](#endnote-11) : J’ai appris à utiliser cette fonction qui permet d’obtenir les informations présentes dans la base de données en utilisant l’url de l’API. Ensuite, dans les « then » de cette méthode, je peux effectuer un traitement avec les données présentes dans la base de données. Ainsi, j’ai créé des éléments html à partir des données présentes dans la base de données.
* Le « catch » dans la méthode « fetch »[[12]](#endnote-12) : J’ai appris à l’utiliser, ce qui m’a permis de savoir lorsqu’il y a une erreur, car elle est affichée dans le navigateur de la console. Cela a rendu le débogage plus facile et plus rapide.
* Le « finally » dans la méthode « fetch »[[13]](#endnote-13) : J’ai appris à l’utiliser, ce qui m’a permis de faire un traitement impossible avant. En effet, je voulais calculer le nombre de produits et le coût total des produits dans le panier afin d’afficher un résumé du panier, mais cela ne fonctionnait pas sans le « finally », car le code s’exécuté avant la fin de l’exécution du code dans les « then » de la méthode « fetch ». Alors, afin de régler ce problème, j’ai utilisé le « finally » qui permet d’attendre la fin de l’exécution de la méthode « fetch », puis d’exécuter le code dans le « finally ».

# La commande « git reflog »

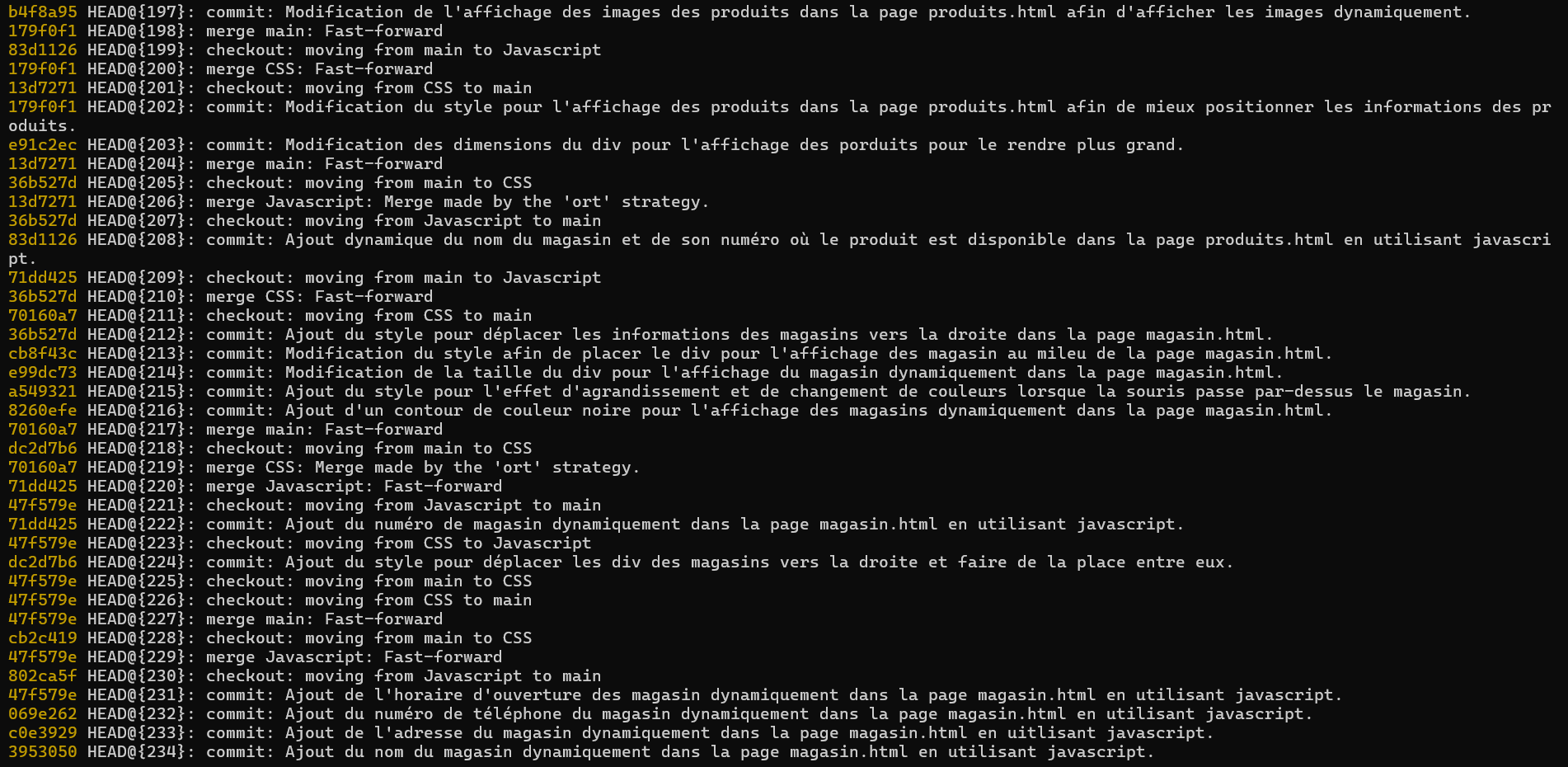


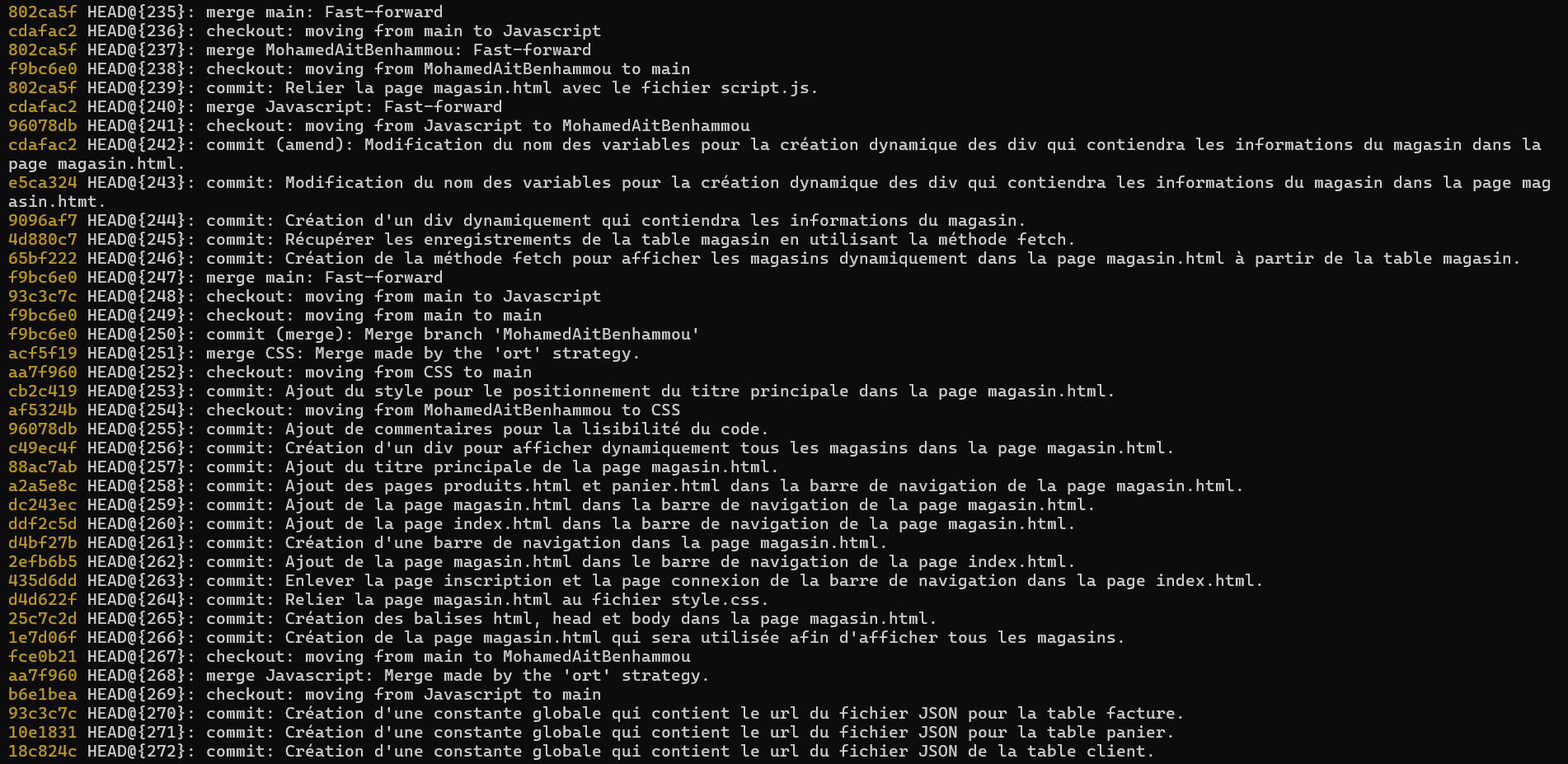






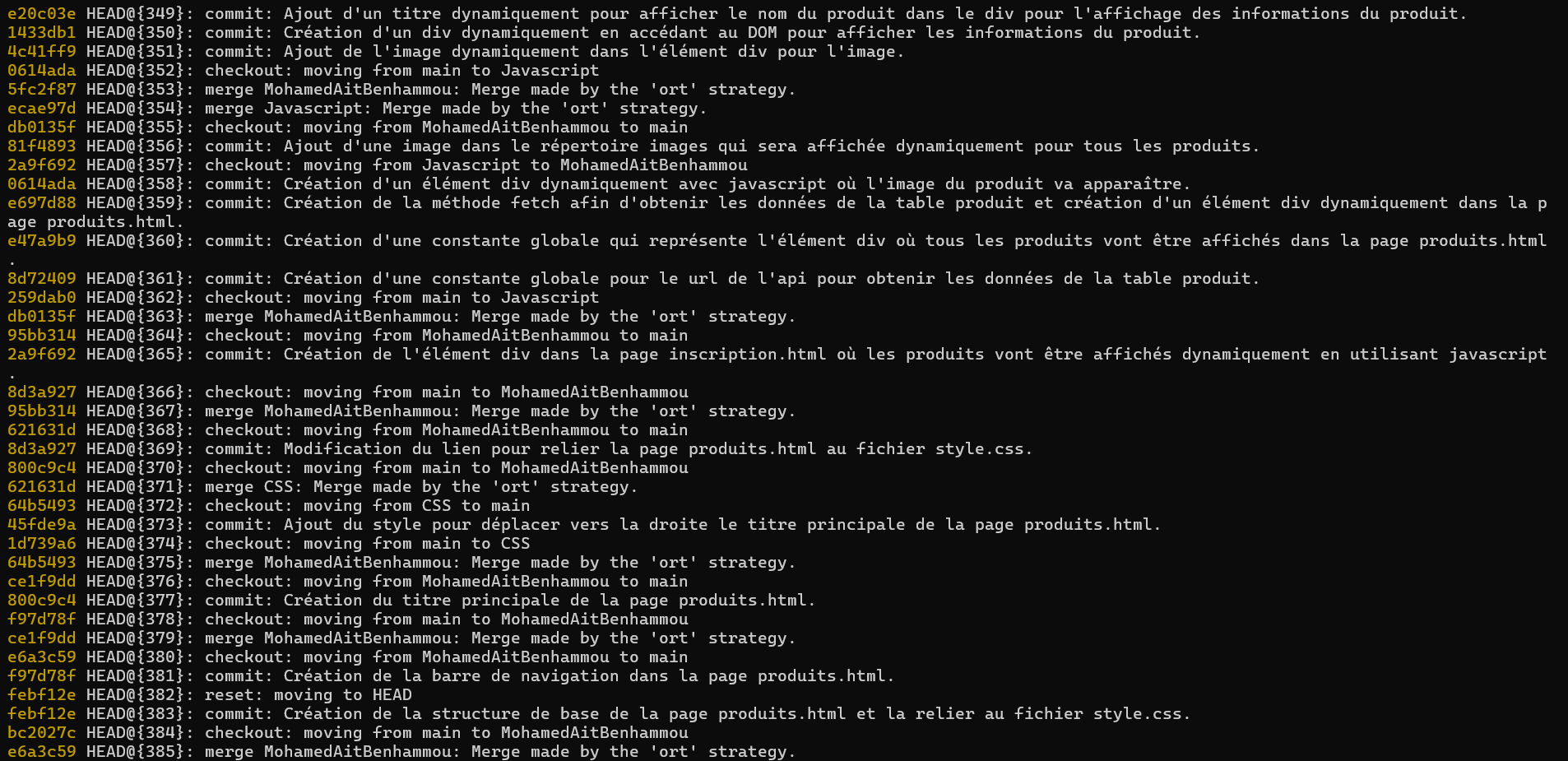






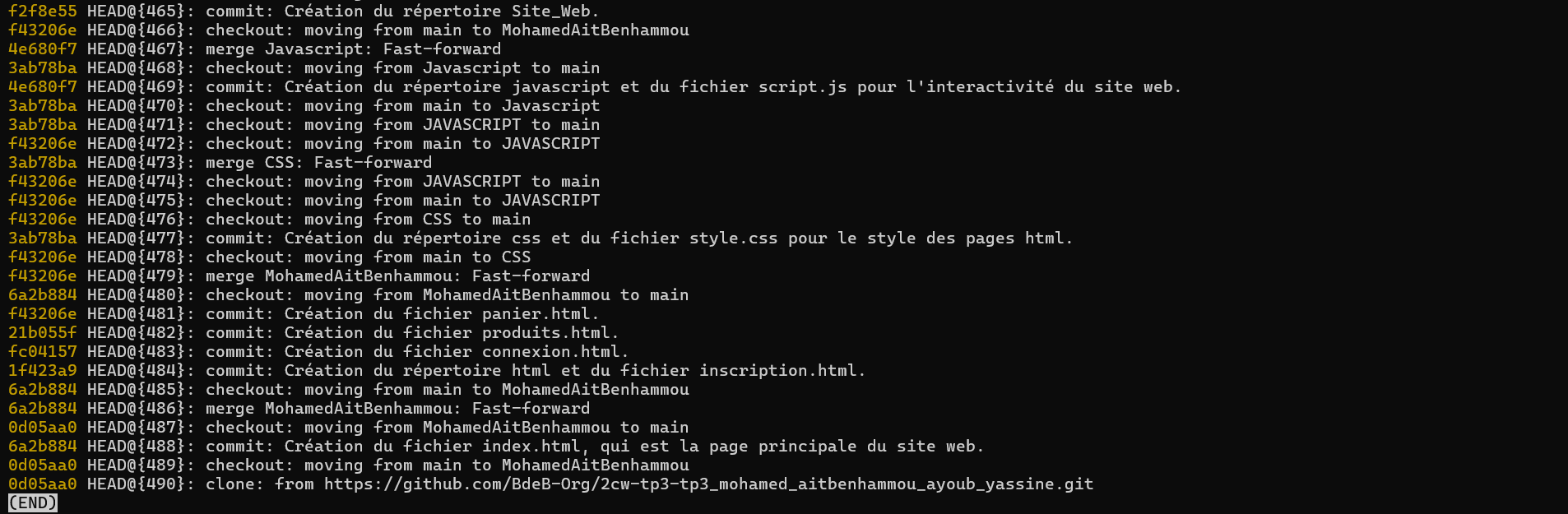












1. <https://oracle-base.com/articles/misc/oracle-rest-data-services-ords-create-basic-rest-web-services-using-plsql#enable-ords> [↑](#endnote-ref-1)
2. <https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch> [↑](#endnote-ref-2)
3. <https://www.w3schools.com/jsref/event_onload.asp> [↑](#endnote-ref-3)
4. <https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement> [↑](#endnote-ref-4)
5. <https://developer.mozilla.org/en-US/docs/Web/API/Element/append> [↑](#endnote-ref-5)
6. <https://www.w3schools.com/jsref/met_document_getelementbyid.asp> [↑](#endnote-ref-6)
7. <https://www.w3schools.com/jsref/prop_html_classname.asp> [↑](#endnote-ref-7)
8. <https://www.w3schools.com/jsref/prop_html_innerhtml.asp> [↑](#endnote-ref-8)
9. <https://oracle-base.com/articles/misc/oracle-rest-data-services-ords-create-basic-rest-web-services-using-plsql#google_vignette> [↑](#endnote-ref-9)
10. <https://www.w3schools.com/jsref/event_onload.asp> [↑](#endnote-ref-10)
11. <https://www.w3schools.com/jsref/api_fetch.asp> [↑](#endnote-ref-11)
12. <https://dev.to/thecharacterv/error-handling-in-the-javascript-fetch-api-1f7a> [↑](#endnote-ref-12)
13. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/finally> [↑](#endnote-ref-13)