



ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR
Membre de
HONORIS UNITED UNIVERSITIES

RAPPORT DE PROJET DE FIN D'ANNÉE

Conception et réalisation d'une application web de e-commerce

Réalisé par :

OKOUYI KONGO Laurent Franck Noël

Encadré par :

Pr. S. EL MOUMNI

Membres du jury :

- Pr. S. EL MOUMNI
- Pr. Chawki EL BALMANY

Septembre 2023

SOMMAIRE

REMERCIEMENTS	6
INTRODUCTION	7
Partie I : Définition des besoins	8
PARTIE II : Conception	9
1) Outils utilisés	9
2) Le diagramme de cas d'utilisation	10
3) Le diagramme de séquences	11
4) Le diagramme d'activité	12
5) Le diagramme de classes	13
Partie III : Réalisation	14
I. Le backend	14
a) Outils utilisés	14
b) Création de notre serveur	15
c) Création des schémas de données	16
d) Les contrôleurs	17
e) Les routes	18
f) La logique mot des mots de passe	19
g) Le token	20
h) Le middleware d'authentification	21
II. Le frontend	22
a) Outils utilisés	22
b) Liaison avec le backend	22
c) Les contextes	22
d) Les administrateurs	23
III. La gestion des données	24
CONCLUSION	25
INDEX	26
1. Diagramme de classes	26
2. Diagramme de cas d'utilisation	27
3. Diagrammes de séquence	28
a. Création du compte utilisateur	28
b. Modification du compte utilisateur	29
c. Authentification d'un utilisateur	30

d. Ajouter un article au panier 31

TABLES DES FIGURES

Figure 1: Le logo du logiciel PowerAMC.....	9
Figure 2: Logo du langage UML.....	9
Figure 3: Représentation de l'acteur "Utilisateur" et ses rôles.....	10
Figure 4: Diagramme de séquence du cas d'utilisation authentification.	11
Figure 5: diagramme de séquence du cas d'utilisation modifier panier.....	12
Figure 6: Diagramme d'activité du cas d'utilisation "Authentification"	12
Figure 7: Diagramme de classe mettant en évidence les classes Utilisateur, produit, catégorie, panier et commande en montrant leurs liens.....	13
Figure 8: Logo de NodeJs qui est un environnement Javascript	14
Figure 9:ExpressJs, utile pour créer des API rest.....	14
Figure 10:Logo de Bcrypt	15
Figure 11: Logo de JsonWebToken (JWT).....	15
Figure 12: Logo de l'outil Nodemon	15
Figure 13: importation du module http.....	15
Figure 14: Spécification du port et création de notre serveur.....	16
Figure 15: Définition du port d'écoute des requêtes.....	16
Figure 16: Importation de mongoose pour créer le modèle Produit	16
Figure 17: Définitions des attributs du model Commande.....	16
Figure 18: Exportation du modèle Commande	17
Figure 19: Les types ObjectId	17
Figure 20: Début du contrôleur qui permet à un utilisateur de passer une commande	17
Figure 21: Intégration de Express et de sa méthode Router()	19
Figure 22: 3 des routes mises en place pour gérer les commandes	19
Figure 24:Exportation des routes	19
Figure 23: Importation des routes pour une utilisation efficace	19
Figure 25: Création d'un compte avec hashage du mot de passe.....	20
Figure 26: Vérification du mot de passe lors de la connexion.....	20
Figure 27: Assignment du token.....	21
Figure 28: Middleware d'authentification.....	21
Figure 29: Logo de ReactJS	22
Figure 30: Logo de axios	22
Figure 31: Création d'une instance de axios.....	22
Figure 32: Importation du module axios	22
Figure 33: Stockage du token et du userId dans un contexte	23
Figure 34: Récupération du token stocké dans l'entête	23
Figure 35: Profil d'un utilisateur qui est également administrateur.....	23
Figure 36: Menu d'un utilisateur qui est uniquement utilisateur	23
Figure 37: La partie accueil du coté administrateur lorsqu'une personne qui n'est pas administrateur se connecte.....	24
Figure 38: La partie accueil coté administrateur lorsqu'un administrateur se connecte	24
Figure 39:La base de données de notre application.....	24
Figure 40: Diagramme de classes de notre application	26
Figure 41: Début du diagramme de cas d'utilisation	27
Figure 42: 1ère partie des cas d'utilisation d'un Administrateur	27

Figure 43: 2ème partie des cas d'utilisation de l'Administrateur	28
Figure 44: Scénario de la création d'un compte	28
Figure 45: Modification du compte utilisateur	29
Figure 46: Scénario de l'authentification.....	30

REMERCIEMENTS

Un travail peut être, à l'origine, le fait d'une seule personne. Mais au cours du temps, nous sommes aidés, encouragé et motivé par des personnes qui nous entourent même s'il arrive parfois que ces personnes nous aident sans s'en rendre compte. Leur apport qui peut leur paraître insignifiant peut contribuer à nous donner une nouvelle idée, est parfois un déclic lorsque nous faisons face à un obstacle.

Ainsi, avant d'entrer dans le vif du sujet, je tiens à affirmer toute ma gratitude à l'endroit de ces personnes qui font partie de mon entourage et qui ont été une source de motivation durant la réalisation de ce projet de fin d'année. Toutes ces personnes constituent un pilier sur lequel je m'appuie pour forger ma personne non seulement sur le plan professionnel, mais également sur le plan personnel.

Merci à l'Ecole Marocaine des Sciences de L'Ingénieur (EMSI) en général d'accompagner les étudiants que nous sommes dans notre développement professionnel.

Ma mère **KONGO RAÏSSA Lucie Victorine**, mon père **AUBEIZAULT Guy Roland** et ma sœur **DELICAT KONGO Ange Winelia**, ma famille que je remercie pour leur soutien sans faille et qui sont pour moi mon premier pilier

Dédicace particulière à **WORA DAVAIN Nell** qui m'a également aidé et soutenu durant la réalisation de mon projet de fin d'année.

Enfin, un remerciement à l'être le plus important, le tout puissant, sans qui ni vous ni moi ne serions présents aujourd'hui.

INTRODUCTION

L'Ecole Marocaine de Sciences de l'Ingénierie (EMSI) est un établissement qui a été créé en 1986 et qui est reconnu pour cinq (5) piliers : l'employabilité de ses lauréats, la recherche et l'innovation, la vie estudiantine, l'ouverture à l'international et l'excellence académique. A l'EMSI, une panoplie de notions, de techniques et de méthodes nous sont enseignées pour nous permettre de devenir les excellents ingénieurs que nous devons devenir.

Dans l'objectif de nous apprendre à gérer des projets, de nous faire grandir professionnellement parlant, nous sommes amenés à réaliser des projets de fin d'année. Pour réaliser ces projets, nous devons utiliser des notions apprises lors de notre formation au sein de l'EMSI. Mais nous devons également faire preuve de curiosité en approfondissant les notions apprises.

Ainsi pour réaliser mon projet qui a pour thème « *Conception et réalisation d'une application web de e-commerce* », j'ai eu à utiliser différentes notions. Tout d'abord la conception qui est l'étape la plus importante lors de la réalisation d'un projet, elle nous permet d'avoir une vision d'ensemble de ce que nous voulons réaliser avant de commencer la partie codage. Dans un deuxième temps nous avons programmer notre application web en utilisant les langages HTML, CSS et des Framework JavaScript que nous verrons en détail dans la suite. Notre application web est liée à une base de données qui permet de stocker différentes informations ; les informations de l'utilisateur et de produits entre autres.

Dans les pages qui suivent je vous présenterais étapes par étape la réalisation de notre application web.

Partie I : Définition des besoins

Buy_More qui est magasin d'électroménager souhaite fournir un service plus rapide et efficace en permettant à ses clients de commander leurs produits directement en ligne sur leur site web.

Un visiteur pourra consulter le catalogue de produits. Pour passer une commande ou ajouter des éléments au panier il devra s'authentifier. S'il n'a pas encore de comptes, il pourra en créer un, c'est une opération qui doit prendre juste quelques secondes. Depuis son profile, il pourra voir ses informations et les modifier s'il le souhaite. Il pourra également voir la liste de ses commandes et leur statut (En attente, En cours ou Livré).

Un administrateur est un utilisateur avec plus de privilèges. Depuis son profile utilisateur, il pourra basculer sur son compte administrateur pour effectuer des opérations d'ajout, de modification, de suppression des produits ou des catégories. Il pourra modifier le statut d'une commande, attribuer ou retirer le privilège administrateur à un utilisateur s'il le souhaite et il pourra consulter les différentes listes (Utilisateurs, produits, commandes et administrateur).

PARTIE II : Conception

La conception étant une importante étape dans la réalisation d'une application, elle nous permet d'avoir une vision globale de notre application et de son fonctionnement. Elle nous permet de définir les acteurs et le rôle de chacun d'eux ainsi que leur limite. Elle nous permet de trouver et de définir le lien entre les différents acteurs. En clair, sans une bonne conception il peut être très difficile de réaliser notre application de manière efficace et efficiente.

1) Outils utilisés

Pour faire la conception de notre application et dessiner les différents diagrammes, nous avons le choix entre différents outils mais le choix a été porté sur le logiciel de modélisation **PowerDesigner**. PowerDesigner (anciennement PowerAMC) est un logiciel de conception créé par la société SAP, qui permet de modéliser les traitements informatiques et leurs bases de données associées.



Figure 1: Le logo du logiciel PowerAMC

Avec ce logiciel, nous avons utilisé le langage UML « Unified Model Language » qui est un langage unifié qui permet la modélisation à l'aide d'une panoplie de diagrammes. C'est un langage de communication qui va nous permettre de lever les ambiguïtés du langage naturel. En gros, il va nous permettre de voir le fonctionnement de notre application.



Figure 2: Logo du langage UML

2) Le diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est le premier diagramme que nous avons utilisé pour modéliser notre application. Ce diagramme permet de définir les différents acteurs en assignant à chacun des rôles spécifiques. Dans notre application, nous avons déterminé trois (3) acteurs : le visiteur, le client ou l'utilisateur et l'administrateur. Nous avons établi que le client et l'administrateur sont des expansions du visiteur. Ils vont donc hériter des rôles du visiteur tel que « voir le catalogue ». Mais chacun aura des limites ou, dans le cas de l'administrateur, il n'y aura pas de limites aux possibilités d'action.

- Le visiteur pourra voir le catalogue de produits et créer un compte
- Le client ou l'utilisateur pourra se connecter, ajouter des produits au panier, voir et modifier son profile, passer une commande et suivre une commande
- L'administrateur pourra gérer les produits, gérer les administrateurs, gérer les utilisateurs et gérer les commandes

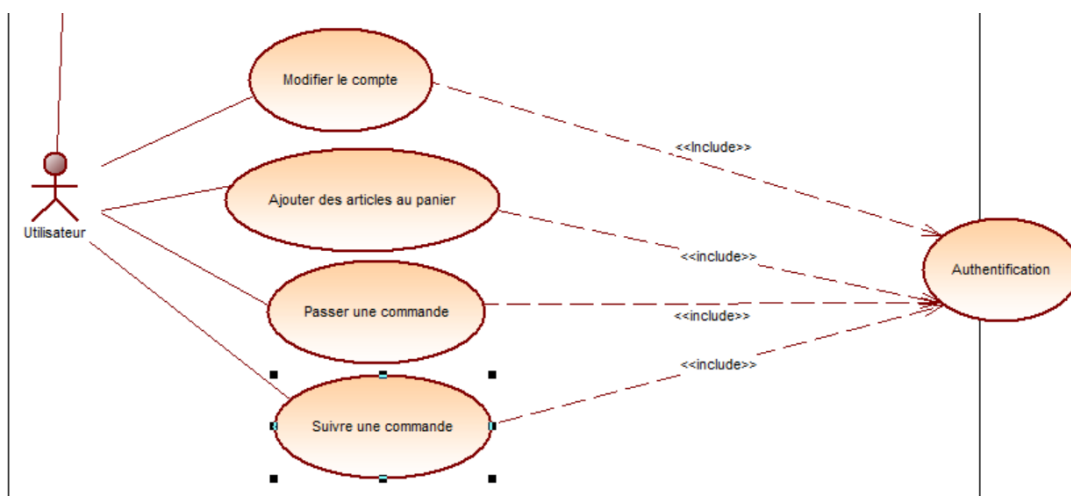


Figure 3: Représentation de l'acteur "Utilisateur" et ses rôles

Nous pouvons remarquer la notation « include » sur la liaison entre deux cas d'utilisations. Cela signifie qu'un cas d'utilisation est obligatoire pour la réalisation d'un autre cas. Dans notre figure (figure 3), le cas d'utilisation « Authentification » est obligatoire pour pouvoir modifier un compte, passer une commande ou encore suivre une commande.

La totalité du diagramme de cas d'utilisation sera visible dans la partie « Index » à la fin de notre rapport

3) Le diagramme de séquences

Le diagramme de séquence est un diagramme qui détaille un cas d'utilisation donné. Pour ce cas d'utilisation, il va montrer les acteurs qui sont concernés ainsi que l'enchaînement dans l'ordre chronologique des actions qui vont pousser à des réactions pour arriver à satisfaire un cas d'utilisation. Pour un cas d'utilisation donné, il existe un scénario nominal. Ce scénario nominal représente le fonctionnement dans les conditions normales de notre processus, il ne va pas prendre en compte les différentes possibilités.

Pour le cas d'utilisation Authentification par exemple, le scénario nominal sera le suivant :

L'utilisateur : entre dans l'application et choisi l'option de connexion

Le système : affiche le formulaire de connexion

L'utilisateur : entre les informations de connexion (courriel et mot de passe)

Le système : vérifie les informations de connexion, et redirige l'utilisateur vers la page d'accueil avec un message « connexion réussie »

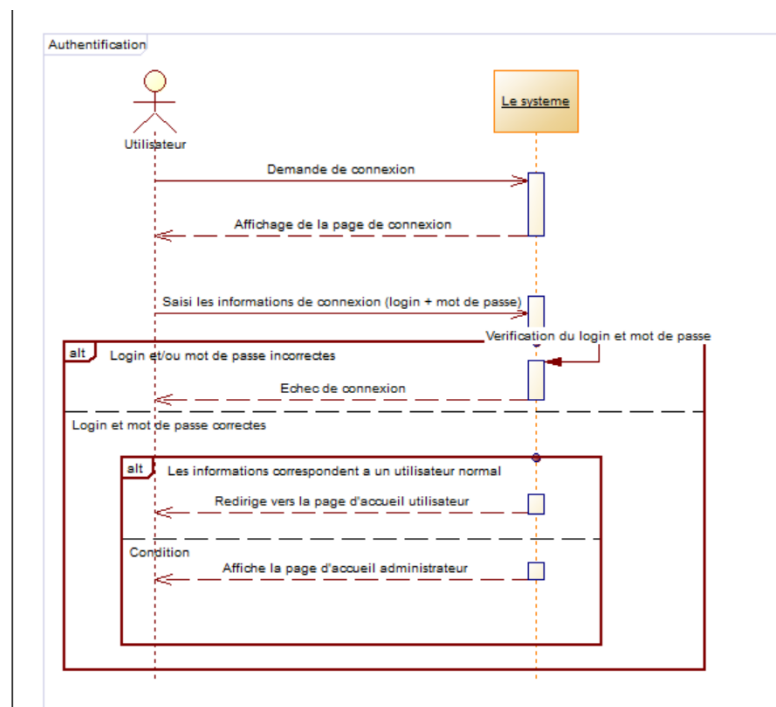


Figure 4: Diagramme de séquence du cas d'utilisation authentification.

Pour chaque cas d'utilisation nous avons fait un diagramme de séquence et parfois nous avons dû faire des références à des diagrammes de séquence dans d'autres diagrammes pour rendre plus clair les schémas. C'est le cas du diagramme de séquence du cas d'utilisation « Authentification » que nous avons inclus en début du diagramme de séquence du cas d'utilisation « Modifier panier » voire figure 5. Vous pouvez retrouver d'autres diagrammes de séquence de notre application dans la partie « Index » à la fin du rapport

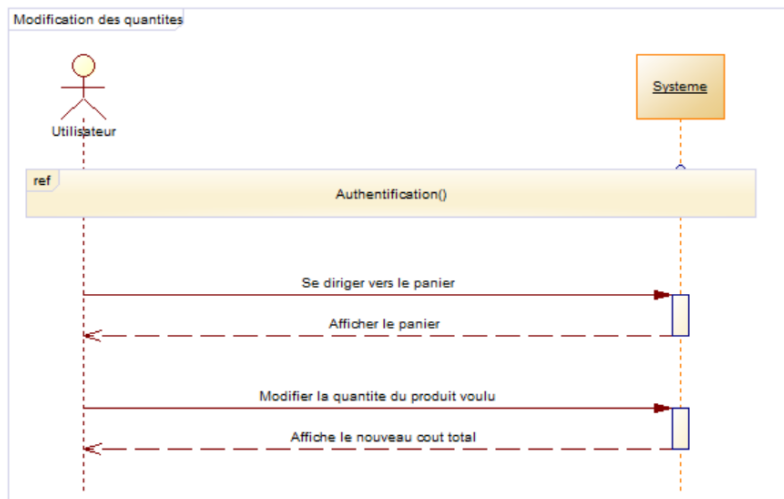


Figure 5: diagramme de séquence du cas d'utilisation modifier panier.

4) Le diagramme d'activité

Le diagramme d'activité c'est un diagramme qui va simplifier le diagramme de séquence en affichant uniquement le parcours, l'enchaînement des opérations d'un cas d'utilisation. Voyons le diagramme d'activité du cas d'utilisation connexion à la figure 6.

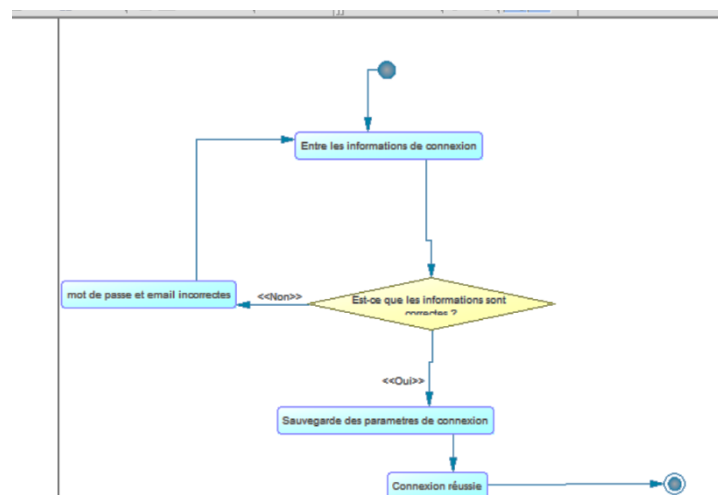


Figure 6: Diagramme d'activité du cas d'utilisation "Authentification"

5) Le diagramme de classes

Le diagramme de classes est un diagramme assez complet. Il représente les différentes classes de notre application, les liens entre les différentes classes ainsi que les opérations et les méthodes de chaque classe. Voyons une brève partie de notre diagramme de classes dans la figure 7.

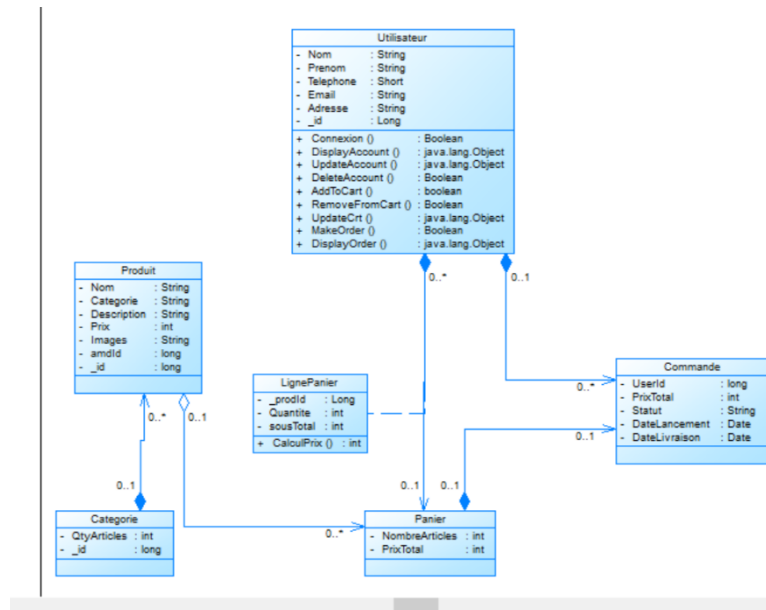


Figure 7: Diagramme de classe mettant en évidence les classes Utilisateur, produit, catégorie, panier et commande en montrant leurs liens

C'est sur ce dernier diagramme que nous terminons la partie conception de notre application. A la fin de notre rapport vous pourrez trouver la suite des différents diagrammes que nous avons présenté ci-dessus

Partie III : Réalisation

Nous voici arrivés à la partie réalisation de notre application. Dans cette partie nous allons parler de codes, de Framework, d'IDE et de tous les outils qui nous ont permis de mettre en place une application qui est « palpable ». Nous allons commencer par le backend, ensuite nous allons parler du frontend avant de parler de la gestion des données.

I. Le backend

Le backend est la partie de notre application où nous allons créer et préciser le comportement de notre application grâce à des contrôleurs. Nous allons définir des routes d'accès, des middlewares, les schémas de données etc...

Commençons par présenter les outils utilisés pour la mise en place de notre backend.

a) Outils utilisés



Figure 8: Logo de NodeJs qui est un environnement Javascript

Node.js est un environnement JavaScript côté serveur qui offre des performances élevées, une gestion efficace des E/S asynchrones, un écosystème riche de modules, et est largement utilisé pour développer une variété d'applications web et serveur modernes.

Express.js est un Framework web robuste, minimaliste et flexible qui permet aux développeurs de créer rapidement des applications web côté serveur en JavaScript avec Node.js. Il est largement utilisé dans l'industrie pour développer une variété d'applications web et d'API REST.



Figure 9: ExpressJs, utile pour créer des API rest



Figure 10: Logo de Bcrypt

Bcrypt est une bibliothèque de hachage de mot de passe qui offre une solution sécurisée pour le stockage des mots de passe en utilisant un algorithme de hachage de mot de passe adaptatif.

JSON Web Token (JWT) est un format de jeton flexible, sécurisé et compact utilisé pour l'authentification et la transmission d'informations entre des parties. Il est largement utilisé dans le développement d'applications web et de services web en raison de sa simplicité et de sa sécurité.



Figure 11: Logo de JsonWebToken (JWT)



Figure 12: Logo de l'outil Nodemon

Nodemon est un outil que nous avons utilisé pour relancer de manière automatique le serveur après la modification d'un fichier.

Pour le stockage de nos données, nous avons opté pour MongoDB qui est un gestionnaire de base de données noSQL



b) Création de notre serveur

Lors de la réalisation de notre backend, nous avons commencé par la création de notre serveur.

Nous avons commencé par importer le module « http » avec la commande :

```
kend > serverjs > ...
1 // Importer le package HTTP de Node
2 const http = require('http');
3
```

Figure 13: importation du module http

Nous avons par la suite défini un port que nous allons utiliser pour lancer notre serveur. Nous avons défini ce port dans une variable d'environnement créée avec le module dotenv. Par la suite nous avons créé notre serveur avec la commande

```
// Configuration du port
app.set('port', process.env.PORT || 3002);

// Création du server avec la méthode createServer de http
const server = http.createServer(app);
```

Figure 14: Spécification du port et création de notre serveur

Enfin, nous allons écouter les requêtes envoyées au port spécifier plus haut avec la commande suivante.

```
// Il faut écouter les requêtes envoyées au port *
server.listen(process.env.PORT || 3002);
```

Figure 15: Définition du port d'écoute des requêtes

c) Création des schémas de données

Pour pouvoir faire toutes les opérations de notre application web, nous devons d'abord définir le schéma de données qui va traduire notre diagramme de classe (Voici pourquoi la conception est importante). Pour créer ces schémas, nous devons d'abord importer l'outil mongoose de MongoDB.

```
// Importation de l'outil mongoose
const mongoose = require('mongoose');
```

Figure 16: Importation de mongoose pour créer le modèle Produit

Par la suite nous avons créé le schéma et défini les attributs avec la méthode **Schema** de mongoose.

Figure 17: Définitions des attributs du model Commande

```
const mongoose = require('mongoose')

const commandesSchema = mongoose.Schema({
  UserId: {type: mongoose.Schema.Types.ObjectId, ref: 'User'},
  Produits: [{type: mongoose.Schema.Types.ObjectId, ref: 'Produit' }],
  Quantite: {type: Number},
  Statut: {type: String},
  DateLancement: {type: Date},
  DateLivraison: {type: Date},
  PrixTot: {type: Number},
  Numero: {type: Number}
})

class Schema<EnforcedDocType = any, TModelType =
Model<EnforcedDocType, any, any, any, IfAny<EnforcedDocType,
any, Document<unknown, any, EnforcedDocType> &
Require_id<EnforcedDocType>>, any>, TInstanceMethods = {},
TQueryHelpers = {}, TVirtuals = {}, TStaticMethods = {},
TSchemaOptions = DefaultSchemaOptions, DocType extends
ResolveTimestamps<...> = ResolveTimestamps<...>,
THydratedDocumentType = IfAny<...>.Types.ObjectId
```


Pour finir et exporter nos modèles, nous allons utiliser la ligne suivante

```
module.exports = mongoose.model('Commande', commandesSchema)
```

Figure 18: Exportation du modèle Commande

C'est ainsi que nous avons créé nos différents schémas. Nous avons défini l'option « required » sur certains attributs pour spécifier qu'il est obligatoire ou non de les renseigner.

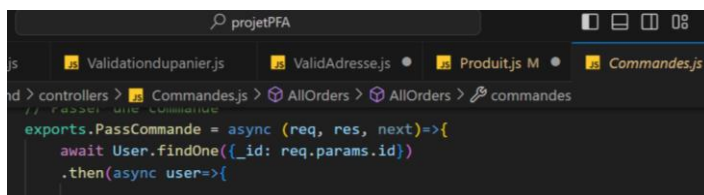
Nous remarquons également que quelques-uns des attributs sont de type « mongoose.Schema.Types.ObjectId », c'est tout simplement pour spécifier que ces attributs ont pour valeur un Id faisant référence à un autre schéma, User ou Produit dans l'exemple suivant

```
const commandesSchema = mongoose.Schema({
  UserId: {type: mongoose.Schema.Types.ObjectId, ref: 'User'},
  Produits: [{type: mongoose.Schema.Types.ObjectId, ref: 'Produit' }],
  Quantites: {type: mongoose.Schema.Types.Number, required: true}
});
```

Figure 19: Les types ObjectId

d) Les contrôleurs

Les contrôleurs sont, en gros, les fonctions qui vont être au cœur de notre api. Ce sont les contrôleurs qui vont juger si des opérations peuvent être réalisées ou pas. Que ce soit la lecture, la modification, la suppression ou la création d'un élément, sans contrôleur aucune de ces opérations n'est possible. Analysons un peu plus en détail un contrôleur.

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'projetPFA' with a folder 'controllers' containing 'Commandes.js'. The code editor shows the following code:

```
// passer une commande
exports.PassCommande = async (req, res, next) => {
  await User.findOne({ _id: req.params.id })
    .then(async user => {
```

Figure 20: Début du contrôleur qui permet à un utilisateur de passer une commande

Dans la toute première ligne, on export la fonction « PassCommande ».

Ensuite on va vérifier un `_id` qui a été passé en paramètre. On vérifie avec la fonction `findOne()` du modèle User qui décrit le schéma de l'utilisateur pour vérifier si cet `_id` est correct et qu'il appartient bien à un utilisateur existant. Si c'est le cas, on va renvoyer un objet de type utilisateur.

```
const prices = []
const quantity = user.cart.quantity
const prods = []

user.cart.article.forEach(e => {
  prods.push(e.prod)
  prices.push(e.prixT)
});

if(prods.length == 0)
  return res.status(400).json({message: "Votre panier est vide !!"});

let PrixTotal = 0
prices.forEach(e=>{
  PrixTotal = PrixTotal + e
})
```

Ensuite on va déclarer deux tableaux : `prices` et `prods` qui vont stocker respectivement la liste des sommes de chaque ligne du panier et la liste des produits du panier grâce à une boucle `forEach`.

Par la suite on va vérifier le panier, si le panier est vide on arrête l'opération et on affiche un message côté front. Si le panier n'est pas vide, on va calculer le prix total du panier à l'aide d'une nouvelle variable : `PrixTotal` initialisée à 0 et mise à jour avec les valeurs de `prices` défini plus haut.

```
commandeCount.addCommand()
const numero = commandeCount.getCommandCount()

const newCommande = new Commande({
  UserId: req.params.id,
  Produits: prods,
  Quantite: quantity,
  PrixTot: PrixTotal,
  DateLancement: Date.now(),
  Numero: numero
})

newCommande.save()
user.cart.article = []
user.cart.quantity = 0
user.save()

res.status(200).json({message: "Votre commande a bien été passée"})
.catch(error => res.status(404).json({message: "La commande n'a pas été créée"}))
```

Par la suite on va récupérer le nombre de commandes à l'aide d'une constante.

Nous allons finalement créer la nouvelle commande en renseignant les informations d'une commande et nous allons sauvegarder la nouvelle commande. Avant de sortir, nous allons prendre le soin de vider le panier de l'utilisateur. En cas de réussite de tout ce

processus, nous allons attribuer le statut 200 à la réponse et on va afficher un message. Dans le cas contraire, un statut 404 sera renvoyé.

e) Les routes

Ce sont les routes qui vont définir les chemins à suivre pour mettre en pratique un contrôleur. Pour créer des routes, nous devons utiliser la méthode Router de ExpressJs

```
backend > routes > CommandesRoutes.js > ...
1 const express = require('express')
2 const router = express.Router()
3 const CmdCtrl = require('../controllers/Commandes')
4
```

Figure 21: Intégration de Express et de sa méthode Router()

Nous déclarons également une variable qui va contenir tous les contrôleurs créés pour la gestion des commandes.

Par la suite nous pouvons utiliser les méthodes post, delete, get, put grâce à Router() de ExpressJs.

```
// Affichage des commandes d'un utilisateur par l'utilisateur connecté
router.get('/MyAccount/orders/:id', CmdCtrl.DisplayCommandeUser )

// Passer une commande
router.post('/MyAccount/MakeOrder/:id', CmdCtrl.PassCommande)

// Affichage de toutes les commandes
router.get('/Admin/AllOrders', CmdCtrl.AllOrders)
```

Figure 22: 3 des routes mises en place pour gérer les commandes

Pour que les routes soient utilisables, nous devons les exporter et les importer notre serveur.

```
16
17 module.exports = router
```

Figure 24: Exportation des routes

```
// Place pour définir les différentes routes du projet
const ProductRoute = require('./routes/ProductsRoute');
const UserRoutes = require('./routes/userRoutes')
const CategoriesRoutes = require('./routes/Categories')
const CommandesRoutes = require('./routes/CommandesRoutes')
```

Figure 23: Importation des routes pour une utilisation efficace

f) La logique mot des mots de passe

Dans une application web, il existe des informations qui sont très sensibles, c'est le cas des mots de passe. C'est pourquoi dans notre application nous avons décidé de prendre des mesures importantes pour masquer et surtout protéger les mots de passes. L'utilisation de bcrypt

```
exports.SignUp = (req, res, next) =>{  
  // On va commencer par hacher le mot de passe en faisant 12 tours  
  bcrypt.hash(req.body.Password, 12)  
    .then(hash=>{
```

Figure 25: Création d'un compte avec hashage du mot de passe

comme algorithme de hashage nous permet d'avoir des hash qui sont irréversibles. Donc même si un attaquant arrive à récupérer les mots de passe dans

notre base de données il sera quasi impossible qu'il arrive à voir le mot de passe exact, il aura devant lui un hash qui n'a pas encore d'algorithme de décryptage.

Nous allons commencer par hasher le mot de passe de l'utilisateur avant de l'créer le nouveau compte utilisateur et insérer le hash à la place du mot de passe brut entré par l'utilisateur.

Le processus de hashage sera le même lorsqu'il s'agira de la connexion d'un utilisateur. Etant donné qu'on ne peut pas comparer le mot de passe brut entré par l'utilisateur avec le mot de passe stocké dans la base directement, le seul moyen de vérifier un mot de passe sera donc de hasher le mot de passe fourni par un utilisateur lors du remplissage du formulaire et le comparer au hash du mot de passe stocké.

```
}  
bcrypt.compare(req.body.Password, user.Password)  
  .then(valid => {  
    if (!valid){
```

Figure 26: Vérification du mot de passe lors de la connexion

g) Le token

Un token est jeton (ensemble de caractères) permettant de protéger des routes. En utilisant des token dans notre application, on peut empêcher des utilisateurs d'avoir accès à des parties de notre application en l'absence de ce token. Ce sera notamment le cas lorsqu'il s'agira d'afficher les informations d'un seul compte utilisateur. Ce token est assigné à un utilisateur lors de la connexion.

```
const token = jwt.sign({ userId: user._id }, SECRET_TOKEN, { expiresIn: '2h' });
res.header('Authorization', token);
```

Figure 27: Assignement du token

Premièrement nous allons déclarer jsonwebtoken. Par la suite, nous devons utiliser la méthode **sign()** de

jsonwebtoken pour créer un token. Notre token va comporter le **userId** de l'utilisateur, une chaîne de caractères qui a été définie en amont et la durée de validité. Pour finir nous allons envoyer ce token dans le header. Ensuite lorsque nous en aurons besoin nous allons juste le récupérer.

h) Le middleware d'authentification

Ce middleware va nous permettre de protéger les routes. Ce middleware va récupérer le token que nous avons stocké dans l'en-tête. Si le token n'est pas valide ou absent, la page ne sera pas accessible

```
// Verification du token pour la connexion
exports.auth=(req,res,next)=>{
  try {
    const token = req.headers.authorization.split(' ')[1];
    const decodedToken = jwt.verify(token, SECRET_TOKEN);
    const userId = decodedToken.userId;
    req.auth = {
      userId: userId
    };
    next();
  } catch(error) {
    res.status(401).json({ error });
  }
}
```

Figure 28: Middleware d'authentification

II. Le frontend

Le frontend correspond à la partie qui sera visible par les utilisateurs, nous allons donc décrire les outils que nous avons utilisé ainsi que les logiques mises en place pour respecter le backend.

a) Outils utilisés



Figure 29: Logo de ReactJS

React.js est une bibliothèque JavaScript populaire pour la création d'interfaces utilisateur interactives et réactives dans les applications web

Axios est une bibliothèque JavaScript populaire pour effectuer des requêtes HTTP de manière simple et efficace dans les applications web et Node.js.



Figure 30: Logo de axios

b) Liaison avec le backend

Pour utiliser l'api que nous avons créé, nous devons importer le module axios qui va nous permettre d'effectuer des requêtes http de manière simple en fournissant, lors de la création d'une instance, l'url de base qui correspond à l'adresse de notre api.

Figure 32: Importation du module axios

```
9 import axios from 'axios';
```

```
9
0 const api = axios.create({
1   |   baseURL: "http://localhost:3002/api"
2   | })
```

Figure 31: Création d'une instance de axios

c) Les contextes

Plusieurs Hook seront utilisés dans notre application, parmi lesquels nous trouverons **useContext**. Un contexte va permettre de sauvegarder des informations et de les partager dans

l'application de manière simple. Telles que des informations de l'utilisateur lors de la création tel que le nom, l'adresse électronique ou le panier.

```
const SubmitFormAuth = async(data) =>{
  api.post('/login', {
    "Email":data.Email,
    "Password":data.Password
  })
  .then(rep=>{
    console.log(rep.data)
    setAuthInfo(() =>({token:rep.data.token, userId:rep.data.userId, isAuthenticated:true}))
  })
}
```

Figure 33: Stockage du token et du userId dans un contexte

Ces informations seront utilisées dans les composants reactJs où nous auront besoin de vérifier

l'utilisateur.

```
const api = axios.create({
  baseURL: "http://localhost:3002/api",
  headers: {
    Authorization: `Bearer ${authInfo.token}`,
  },
})
```

Figure 34: Récupération du token stocké dans l'entête

d) Les administrateurs

Comme décrits dans la partie conception, les administrateurs sont des utilisateurs avec l'attribut IsAdmin égal à true. Cela se manifeste par le fait qu'ils ont accès à partie management du site contrairement aux utilisateurs lambda.

ofil

Informations du comptes
Vos commandes
Modifier le compte

Figure 36: Menu d'un utilisateur qui est uniquement utilisateur

u profil

Informations du comptes
Vos commandes
Modifier le compte
Mode Administrateur

Figure 35: Profil d'un utilisateur qui est également administrateur

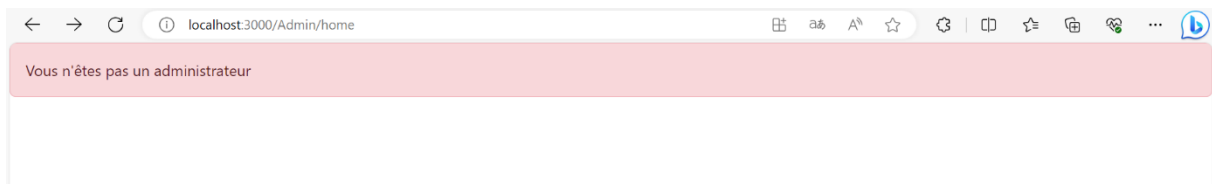


Figure 37: La partie accueil du coté administrateur lorsqu'une personne qui n'est pas administrateur se connecte

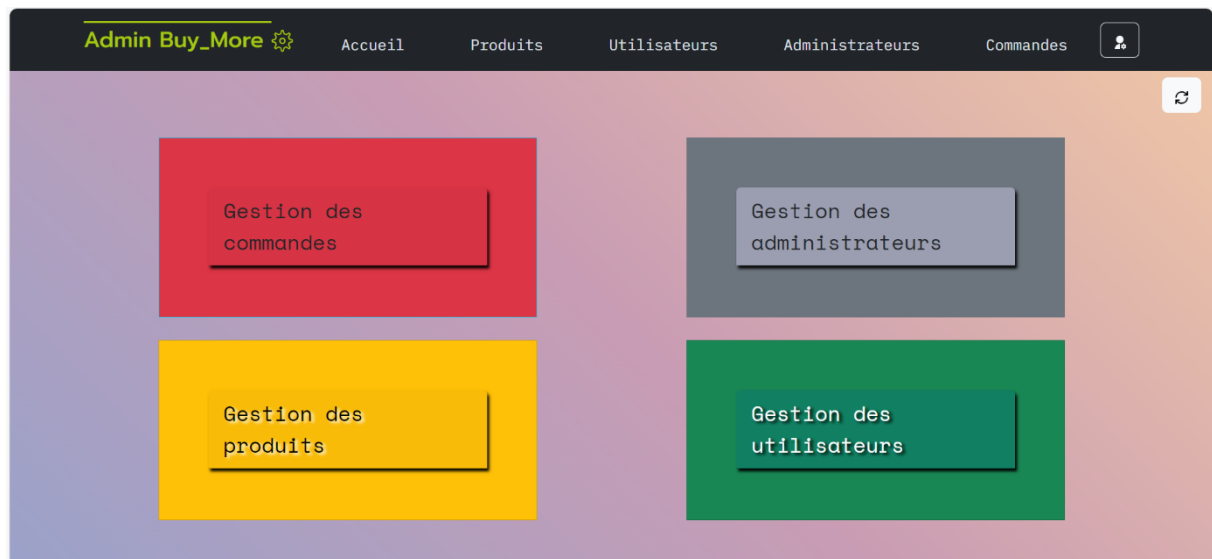


Figure 38: La partie accueil coté administrateur lorsqu'un administrateur se connecte

III. La gestion des données

Comme base de données nous avons choisi MongoDB et come gestionnaire de base de données, nous avons choisi MongoDB Atlas. MongoDB Atlas permet de gérer nos bases de données MongoDB en offrant un service Cloud ce qui nous permet d'avoir accès à nos données depuis n'importe quel appareil.

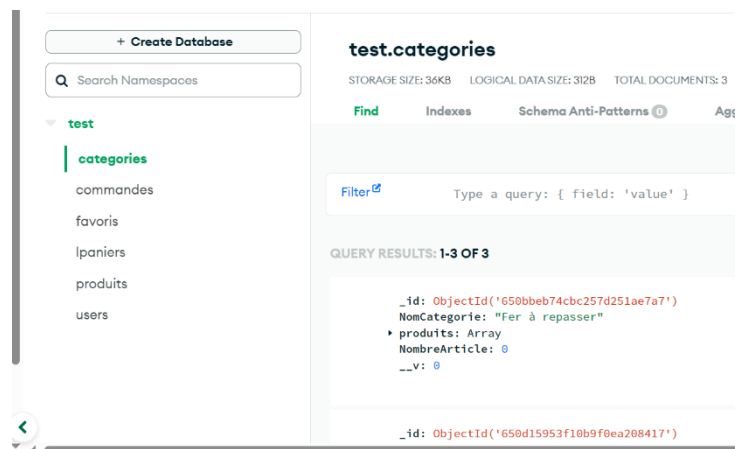


Figure 39: La base de données de notre application.

CONCLUSION

L'objectif du précédent travail était de mettre sur pied une application en passant de la conception et en faisant la réalisation partie backend et frontend.

Après avoir fait une conception grâce au langage UML, en passant par la réalisation des différents diagrammes (cas d'utilisation, séquence, activité, classes), nous avons réalisé notre application en utilisant les outils MERN (MongoDB, ExpressJs, ReactJs et NodeJs) en plus d'autres outils pour fournir cette application fonctionnelle.

Ce travail n'a pas été de tout repos, il y a eu des obstacles, beaucoup d'ailleurs, mais il a suffi d'un peu de curiosité pour le mener à bien.

Il reste encore du chemin car cette application bien qu'elle soit fonctionnelle, a tout de même des zones de progrès, telles que l'aspect visuel et la sécurité pour ne citer que ceux-là. Ces points feront l'objet d'autres séances de travail.

INDEX

1. Diagramme de classes

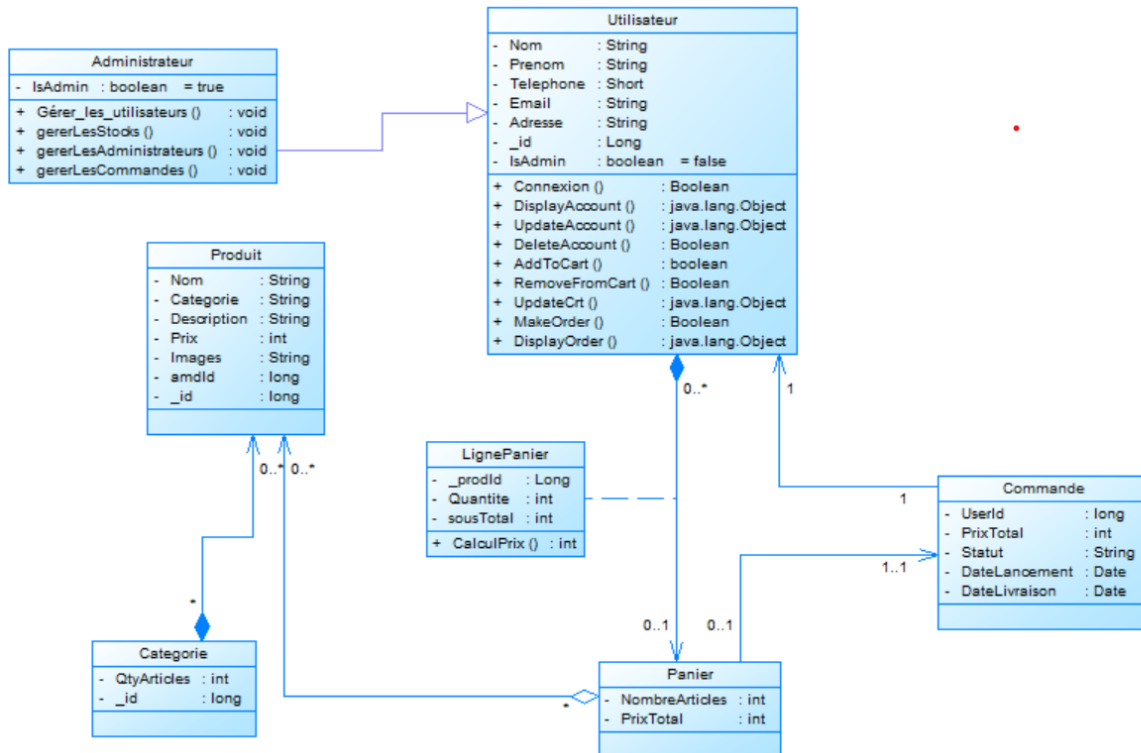


Figure 40: Diagramme de classes de notre application

2. Diagramme de cas d'utilisation

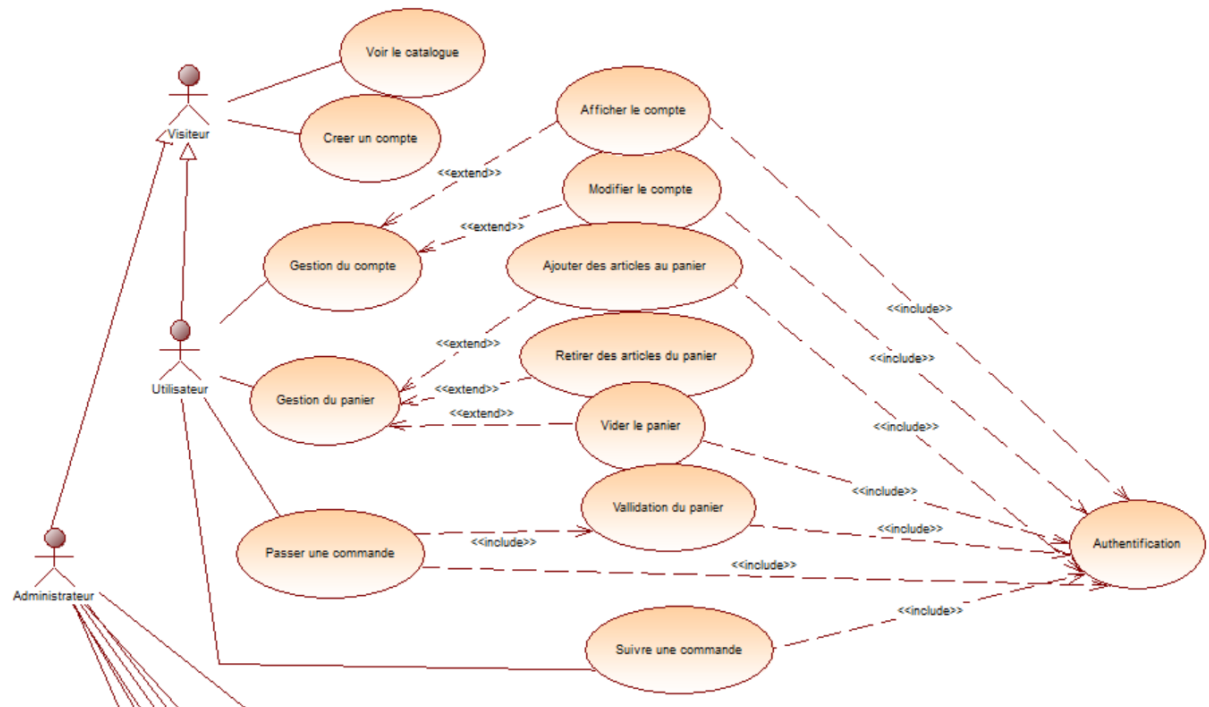


Figure 41: Début du diagramme de cas d'utilisation

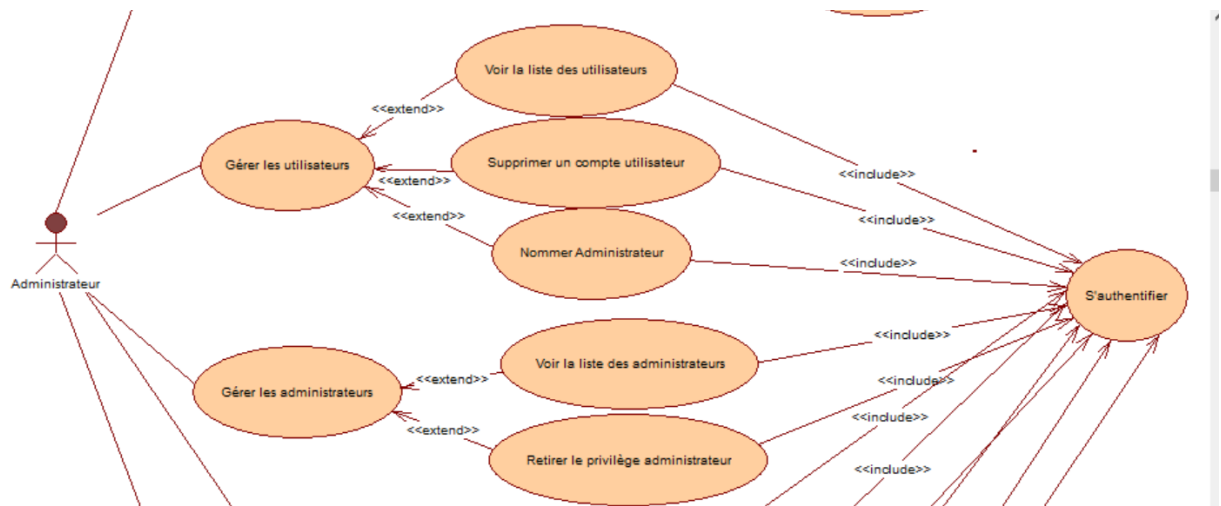


Figure 42: 1ère partie des cas d'utilisation d'un Administrateur

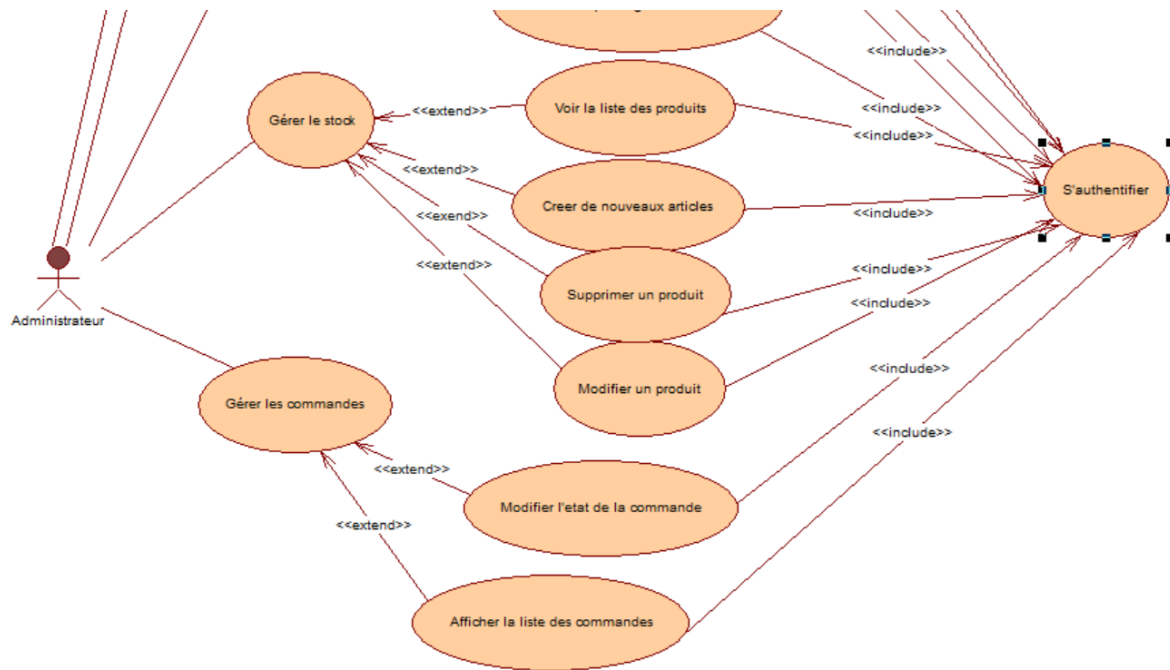


Figure 43: 2ème partie des cas d'utilisation de l'Administrateur

3. Diagrammes de séquence

a. Création du compte utilisateur

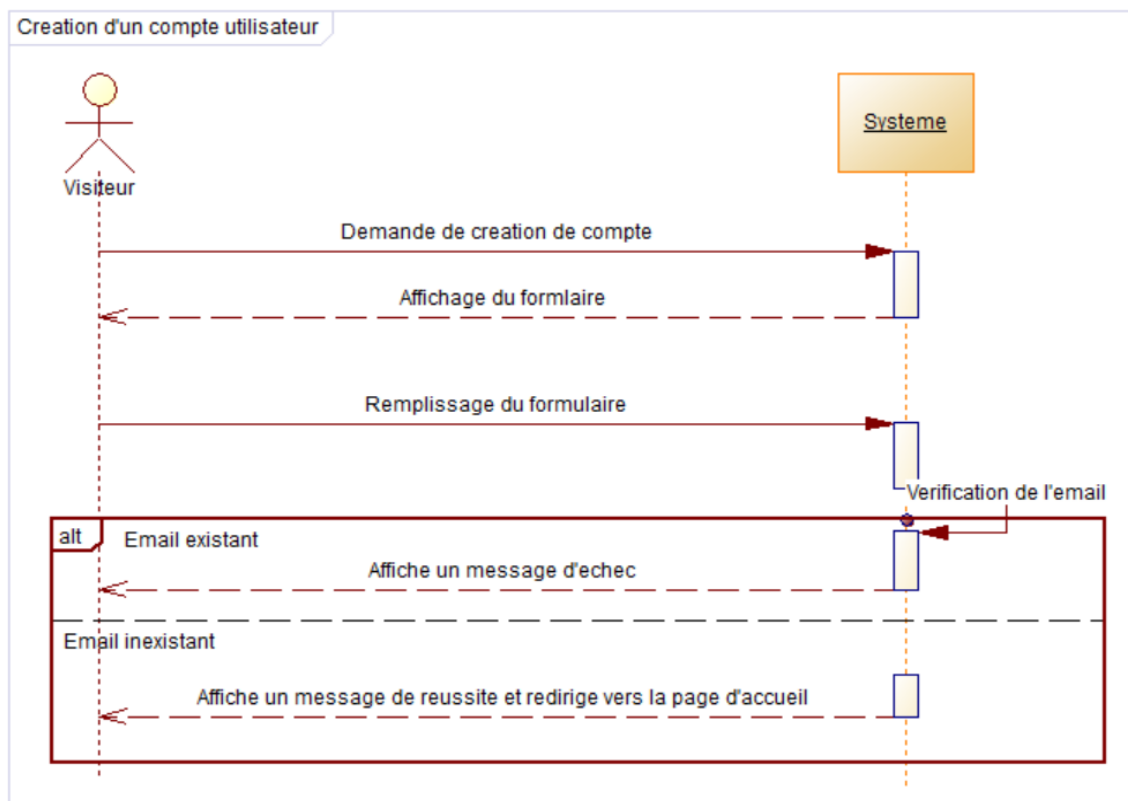


Figure 44: Scénario de la création d'un compte

b. Modification du compte utilisateur

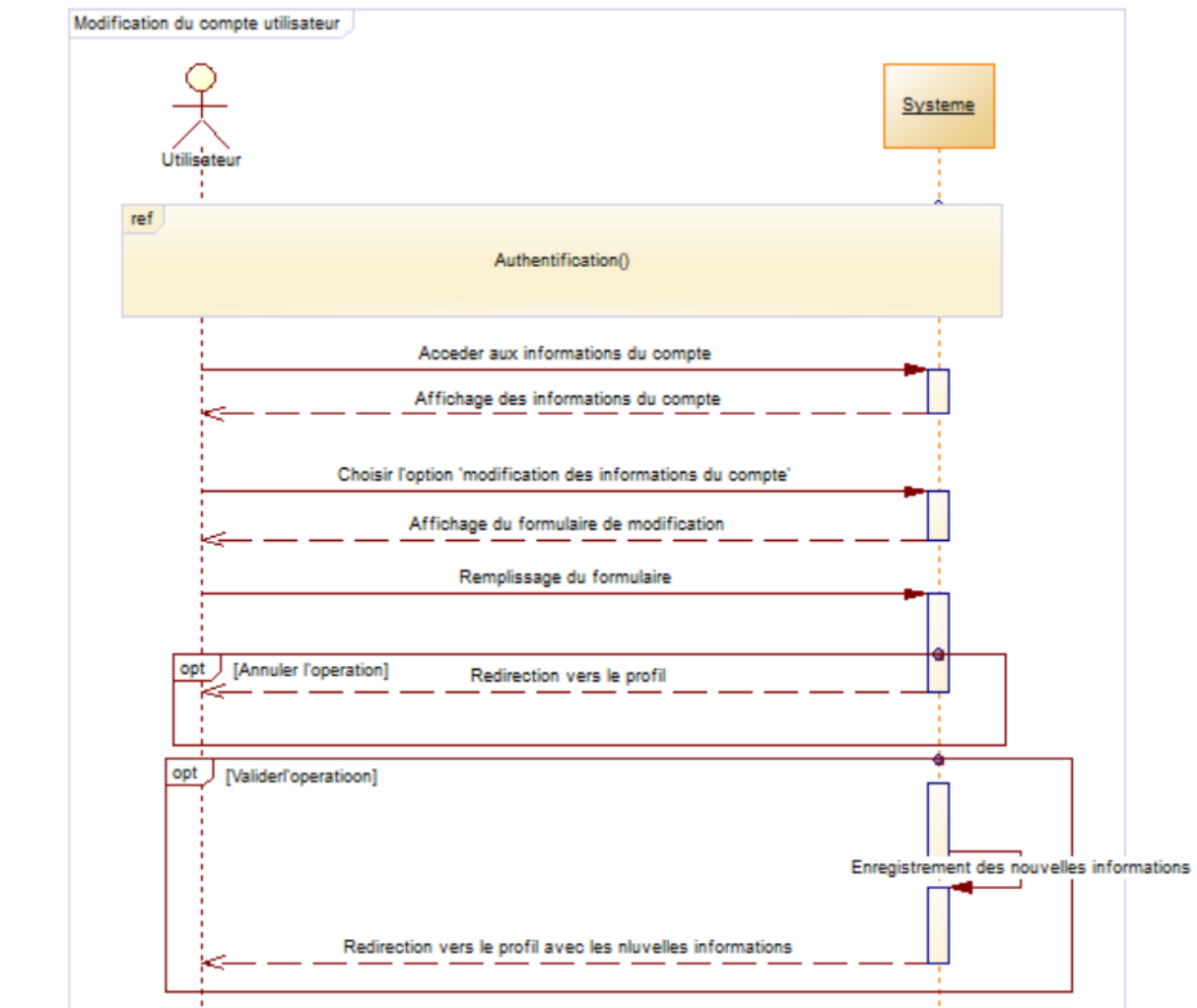


Figure 45: Modification du compte utilisateur

c. Authentification d'un utilisateur

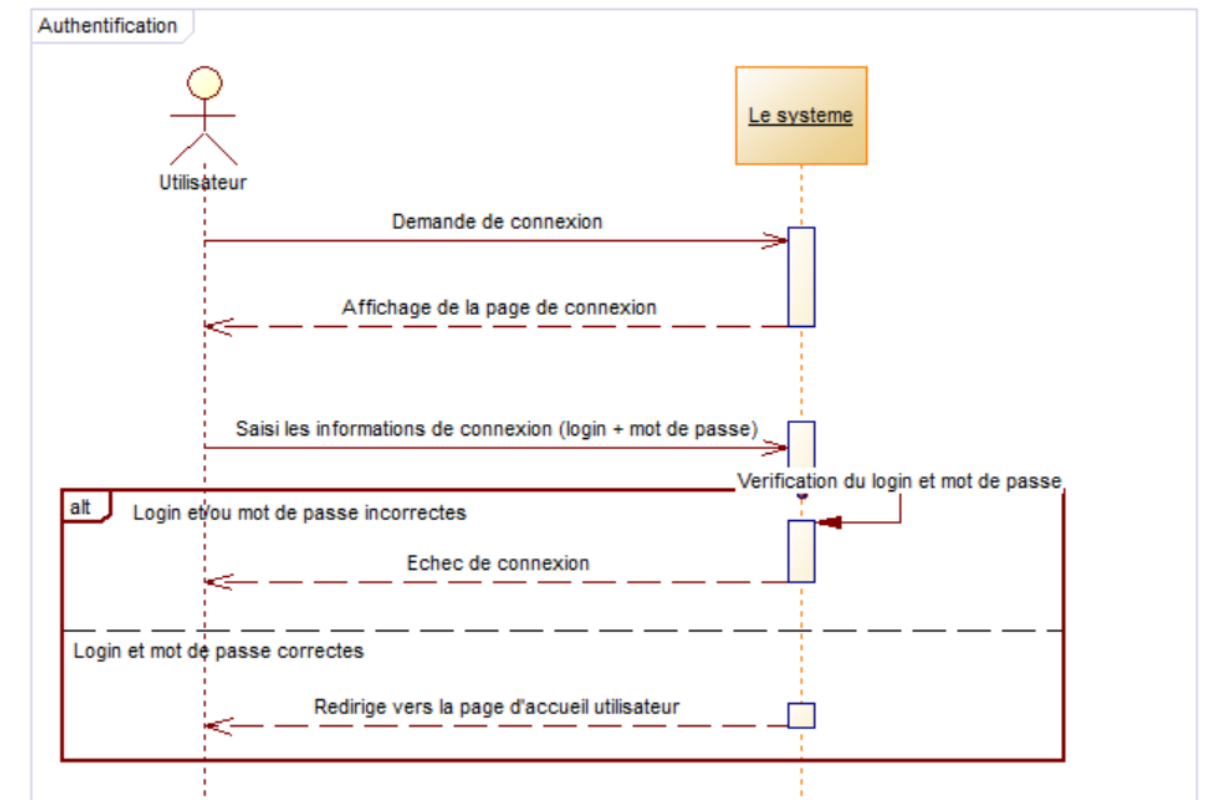
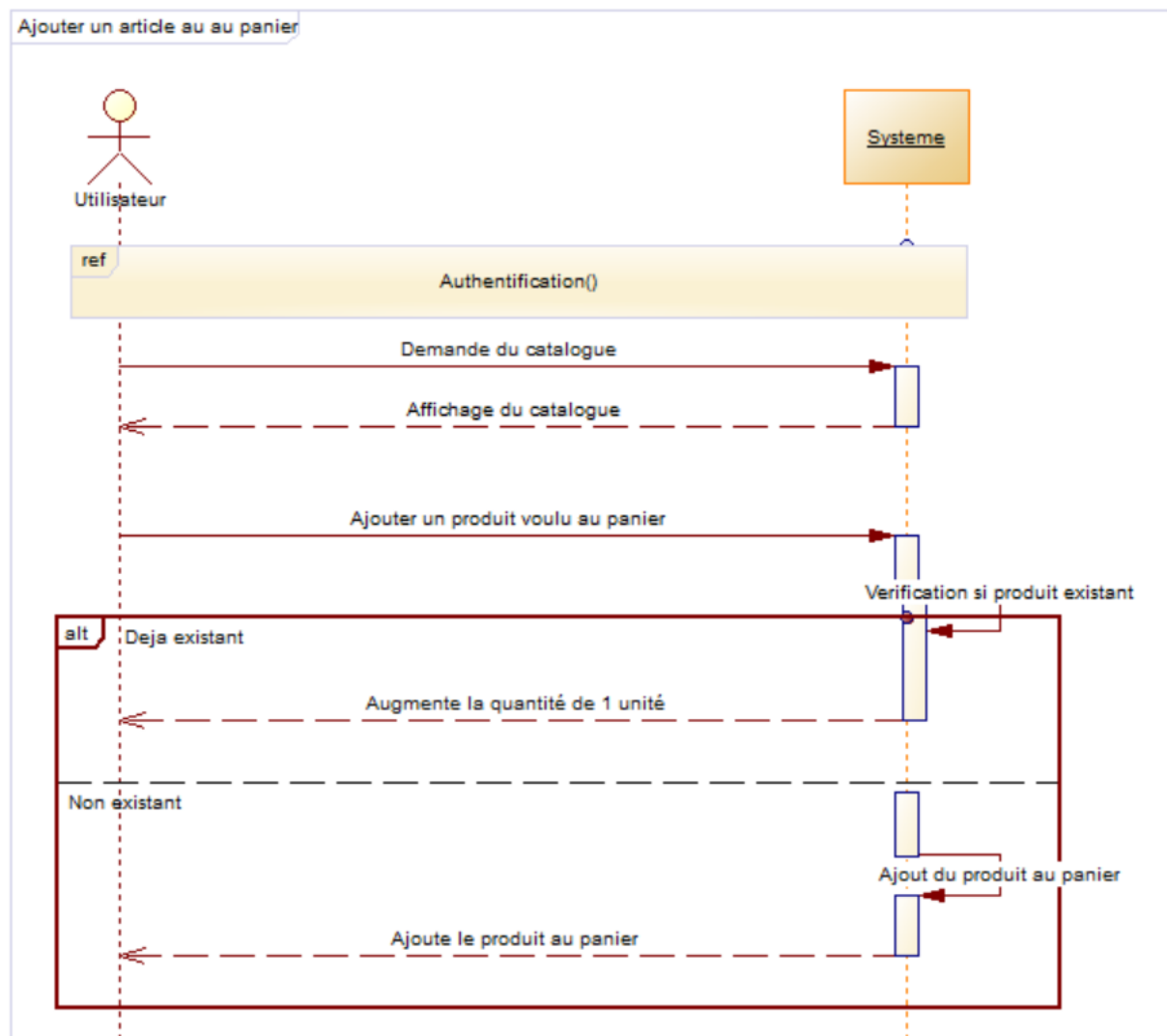


Figure 46: Scénario de l'authentification

d. Ajouter un article au panier



e.