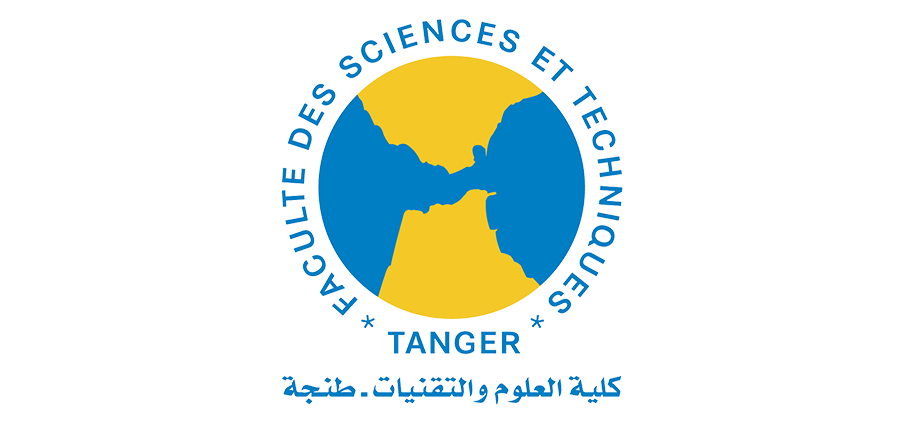
ROYAUME DU MAROC

UNIVERSITÉ ABDELMALEK ESSAÂDI

FACULTÉ DES SCIENCES JURIDIQUES,

ECONOMIQUES ET SOCIALES DE TANGER

Master Data Science pour l’économie et la finance

Module Langage de programmation

(S1)

(PAS ENCORE FINI JE M’EXCUSE POUR LE RETARD)

**Rapport du Projet de fin de module :**

**Application web FULL Stack basée sur le Framework Django.**

Réalisé par:

**GHARBI Nesrine**

Encadré par:

**Pr . ELAACHAk LOTFI**

Année universitaire: 2023/2024

# Table des matières

[**Introduction**](#_6lbdy134is0t) **3**

[**Conception**](#_sdrq1oothbei) **4**

[**Architecture du projet**](#_ybbi29brr6yi) **5**

[**Développement**](#_uvgvnnph1p7h) **8**

[**Conclusion et perspectives**](#_gxlpvhfku74n) **14**

# Introduction

Dans le cadre de nos études à la Faculté des sciences juridiques, économique et sociales de Tanger, plus précisément pendant le premier semestre du master Data Science pour l’économie et la finance, nous sommes appelés à mettre en place une application web full Stack basée sur le Framework Django.

Ce projet est une suite à notre cours et sous l'encadrement de Monsieur ELAACHAK, pour but d’approfondir nos connaissances en développement Web.

Nous allons traiter en premier lieu la conception de l’application, ensuite nous allons expliquer l’architecture suivie pendant la réalisation de cet atelier.

En troisième partie, nous allons mettre en concret les fonctionnalités de l’application.

Finalement nous allons conclure ce travail et nous citerons des idées afin d’améliorer davantage cette application.

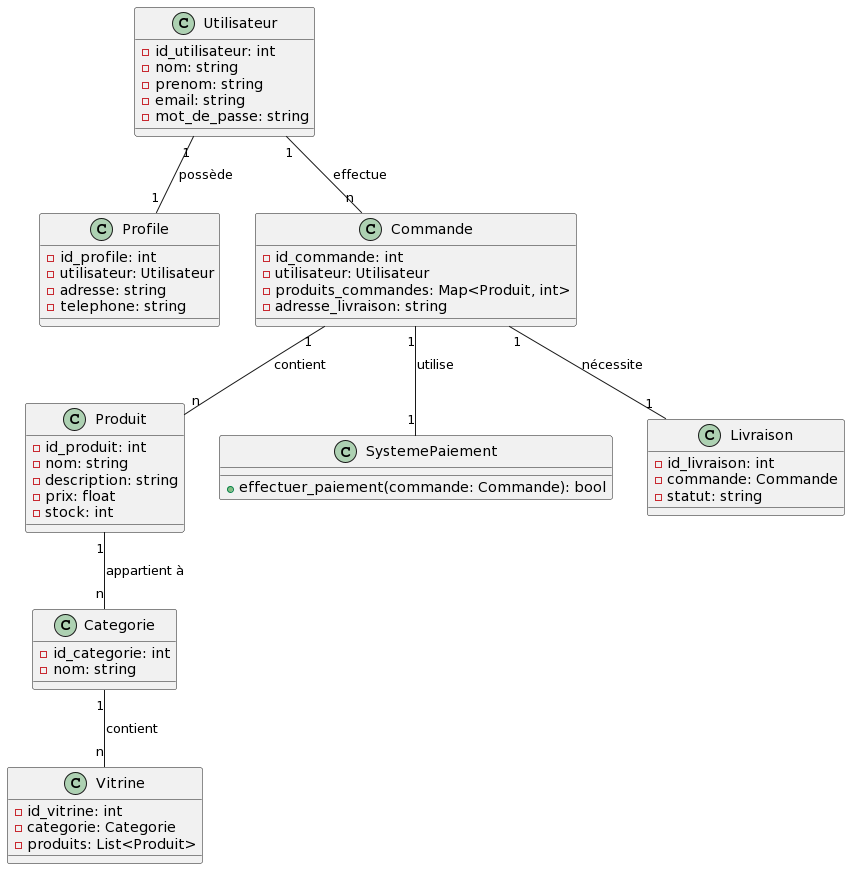
# 

# 

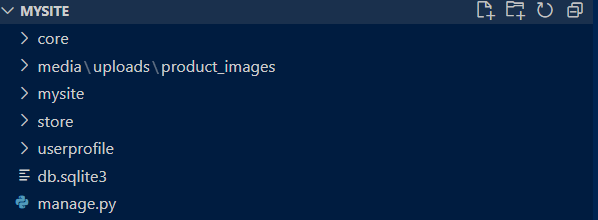
# Conception

Pour réaliser ce projet, nous avons choisi d'utiliser le framework Django en raison de sa robustesse, de sa simplicité de mise en œuvre et de sa capacité à fournir une application web complète. Django offre un ensemble d'outils puissants pour le développement web, y compris la gestion des bases de données, l'authentification utilisateur, et la gestion des formulaires, ce qui en fait un choix idéal pour un projet de cette envergure.

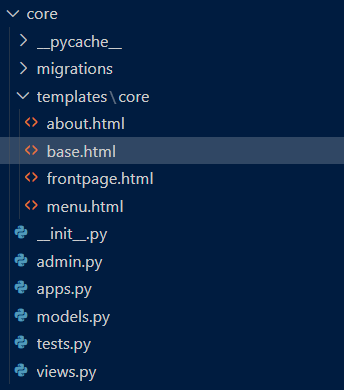
Commençons d’abord par un diagramme modélisant le plan de notre site :



L’application web est composée de 5 dossiers (également appelé « projets »)



## Projet : Core

  
Le dossier "core" dans notre application Django est utilisé pour regrouper les composants fondamentaux et centraux de l'application.

* **init.py** : Ce fichier indique à Python que le dossier "core" doit être traité comme un module, ce qui permet d'importer facilement des éléments de ce dossier dans d'autres parties de l'application.
* **admin.py** : Ce fichier contient les configurations pour l'interface d'administration de Django. Il permet de personnaliser l'apparence et le comportement de l'interface d'administration pour les modèles définis dans ce module.
* **apps.py** : Ce fichier contient la configuration de l'application Django. Il permet de définir des métadonnées pour l'application, telles que son nom et ses options de configuration.
* **models.py** : Ce fichier contient les modèles de données de l'application. Les modèles définissent la structure des données stockées dans la base de données et fournissent des interfaces pour interagir avec ces données.
* **tests.py** : Ce fichier contient les tests unitaires pour l'application. Les tests sont utilisés pour s'assurer que le code fonctionne comme prévu et pour détecter les erreurs éventuelles.
* **views.py** : Ce fichier contient les vues de l'application. Les vues sont responsables de la logique métier de l'application et définissent le comportement des pages web.

En plus des fichiers Python, le dossier "core" contient également un dossier "migrations" qui stocke les migrations de base de données générées par Django pour les modèles définis dans ce module. Cela permet de garder une trace des changements de schéma de base de données au fil du temps.

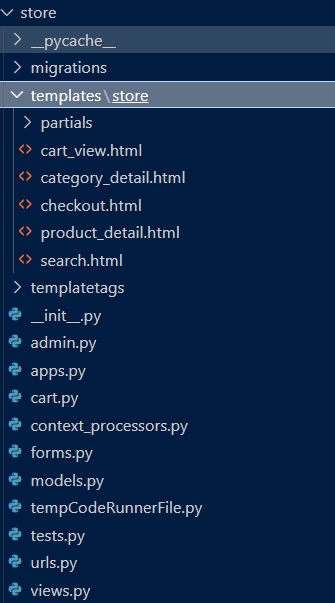
Enfin, le dossier "templates" contient des templates HTML utilisés par les vues pour générer du contenu dynamique. Ces templates sont organisés en sous-dossiers pour faciliter la gestion et la maintenance de l'interface utilisateur de l'application.

## Projet : mysite

Le dossier "mysite" il contient les fichiers et les configurations nécessaires pour définir et gérer le projet dans son ensemble, notamment les configurations principales, les routes URL, et les paramètres de déploiement et de serveur.

* **init.py** : Ce fichier indique à Python que le dossier "mysite" doit être traité comme un module, ce qui permet d'importer facilement des éléments de ce dossier dans d'autres parties de l'application.
* **asgi.py** : Ce fichier contient la configuration pour l'ASGI (Asynchronous Server Gateway Interface). ASGI est utilisé pour gérer les connexions asynchrones dans les applications web Django, telles que les connexions WebSocket.
* **settings.py** : Ce fichier contient les configurations principales de l'application. On y trouve des paramètres tels que les paramètres de base de données, les paramètres de sécurité, les paramètres d'authentification, les paramètres de localisation, etc. Modifier ce fichier permet de personnaliser le comportement global de l'application.
* **urls.py** : Ce fichier contient les configurations des URL de l'application. Il définit les correspondances entre les URL entrantes et les vues correspondantes, ce qui permet de router les requêtes HTTP vers les bonnes parties de l'application.
* **wsgi.py** : Ce fichier contient la configuration pour le WSGI (Web Server Gateway Interface). WSGI est utilisé pour gérer les connexions synchrones dans les applications web Django, telles que les connexions HTTP classiques.

## Projet : store

* **init.py** : Ce fichier indique à Python que le dossier "store" doit être traité comme un module, ce qui permet d'importer facilement des éléments de ce dossier dans d'autres parties de l'application.
* **admin.py** : Ce fichier contient les configurations pour l'interface d'administration de Django spécifiques à l'application "store". Il permet de personnaliser l'apparence et le comportement de l'interface d'administration pour les modèles définis dans ce module.
* **apps.py** : Ce fichier contient la configuration de l'application Django "store". Il permet de définir des métadonnées pour l'application, telles que son nom et ses options de configuration.
* **models.py** : Ce fichier contient les modèles de données spécifiques à l'application "store". Les modèles définissent la structure des données liées à la boutique en ligne, telles que les produits, les catégories, les commandes, etc.
* **tests.py** : Ce fichier contient les tests unitaires spécifiques à l'application "store". Les tests sont utilisés pour s'assurer que le code fonctionne comme prévu et pour détecter les erreurs éventuelles dans la logique de la boutique en ligne.
* **views.py** : Ce fichier contient les vues spécifiques à l'application "store". Les vues sont responsables de la logique métier de la boutique en ligne, telles que l'affichage des produits, la gestion du panier d'achat, le traitement des commandes, etc.
* **cart.py** : Ce fichier contient les fonctionnalités spécifiques au panier d'achat de l'application "store", telles que l'ajout de produits, la suppression de produits, le calcul du total, etc.
* **context\_processors.py** : Ce fichier contient des context processors spécifiques à l'application "store". Les context processors permettent d'ajouter des variables contextuelles aux templates HTML pour faciliter l'affichage des données dynamiques.
* **forms.py** : Ce fichier contient les formulaires spécifiques à l'application "store". Les formulaires sont utilisés pour collecter et valider les données entrées par les utilisateurs lors de l'interaction avec la boutique en ligne.
* **urls.py** : Ce fichier contient les configurations des URL spécifiques à l'application "store". Il définit les correspondances entre les URL entrantes et les vues correspondantes de la boutique en ligne.

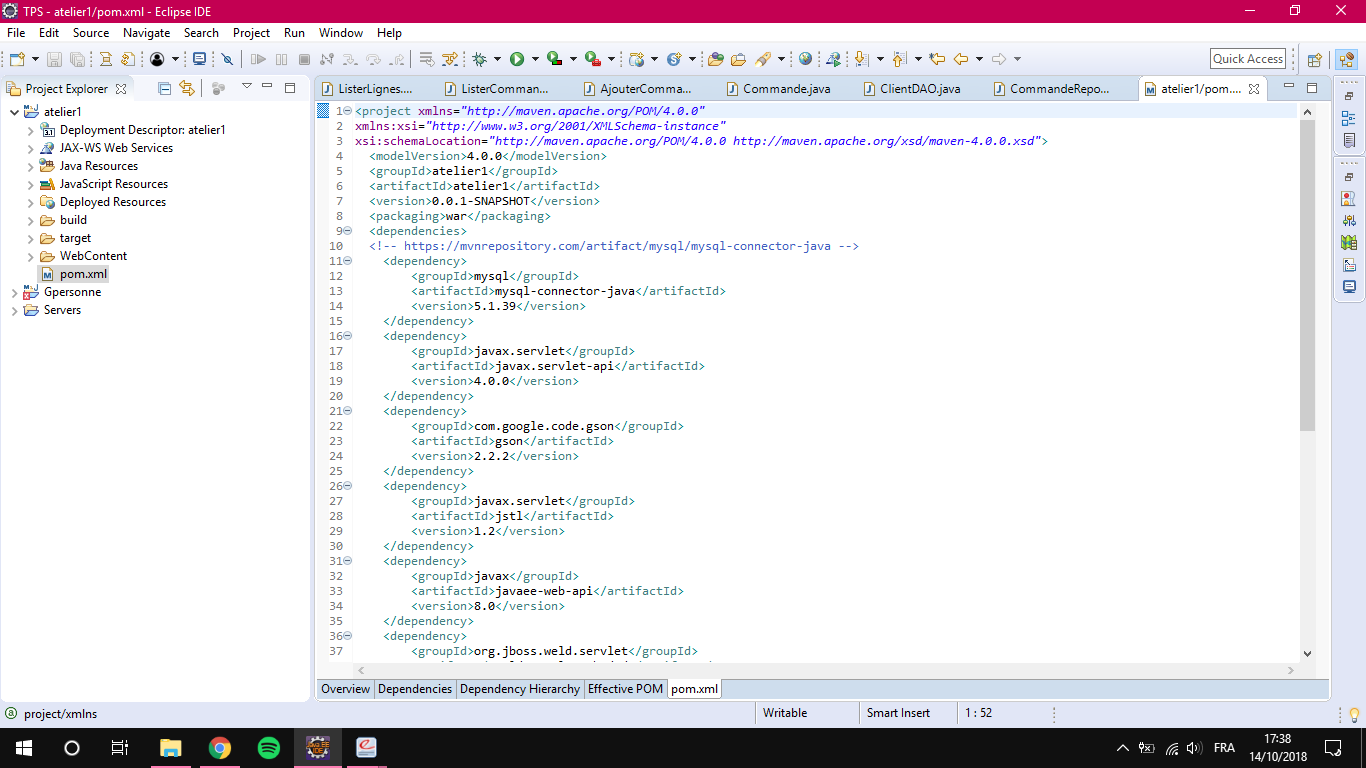
Le dossier de cache de migrations contient les migrations de base de données générées par Django pour les modèles définis dans l'application "store", afin de gérer les changements de schéma de base de données au fil du temps.

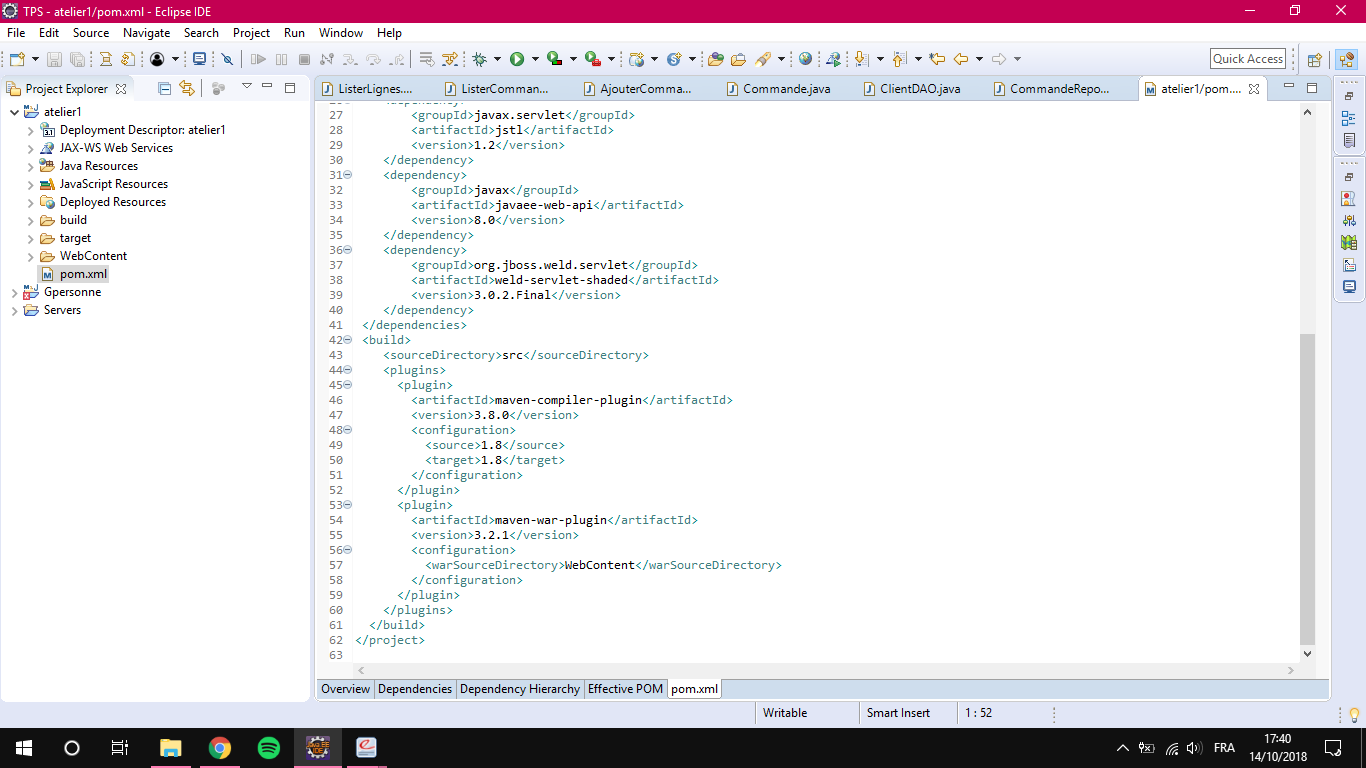
Enfin, le dossier "templates" contient des templates HTML spécifiques à l'application "store", utilisés pour afficher les pages de la boutique en ligne, telles que la vue du panier d'achat, la vue de détail du produit, la vue de recherche, etc. Ces templates sont organisés en sous-dossiers pour une meilleure organisation et gestion.

# 

# Architecture du projet J2EE

Nous avons créé un nouveau projet Web dynamique avec un web module de version 3.1, ensuite on l’a converti vers un projet Maven, le fichier pom.xml contient toutes les dépendances utilisées pour ce projet.



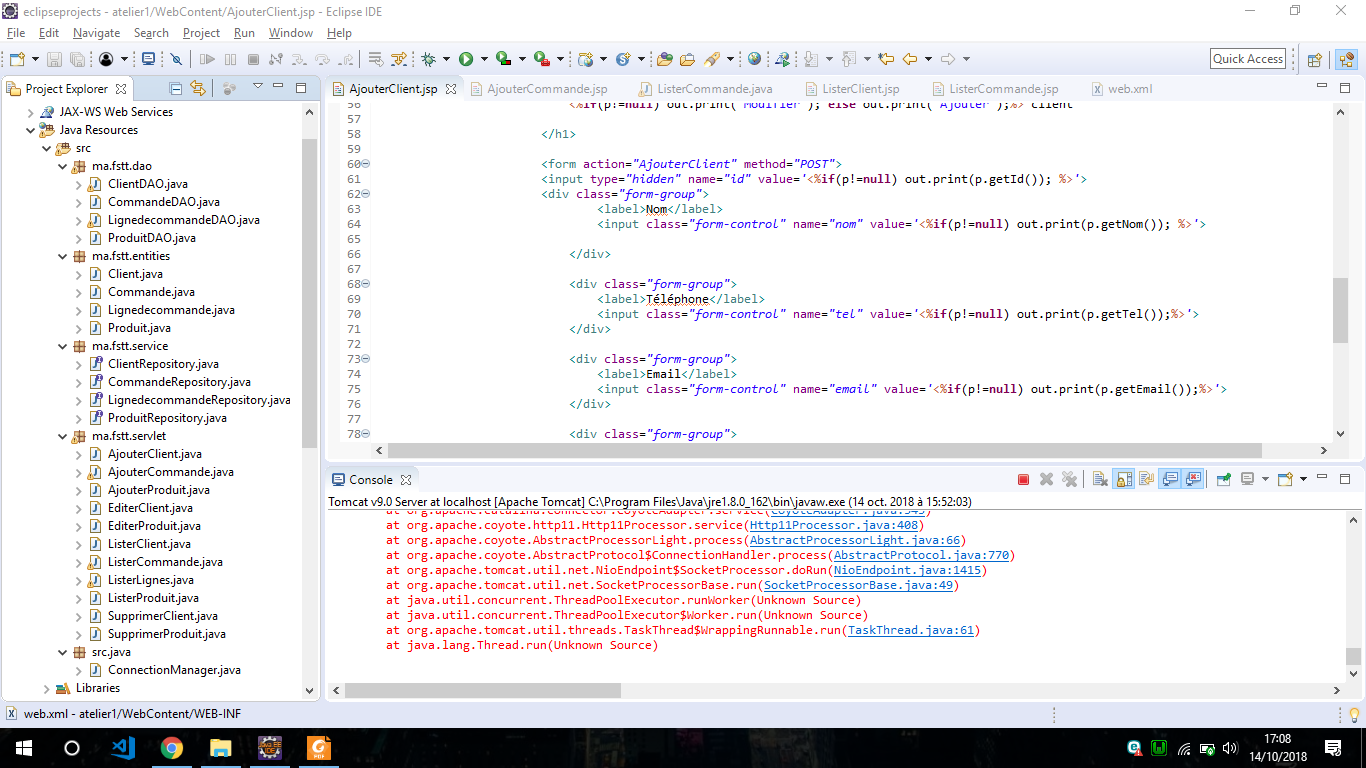


Le projet suit l’architecture du MVC (Model - View - Controller):

Couche Model/Persistence : Nous avons créé des classes dans le package ma.fstt.entities qui représentent les entités, et des interfaces dans le package ma.fstt.service, qui contiennent les méthodes qu'on va redéfinir dans les classes DAO. Ces classes DAO contiennent l’implémentation des méthodes utilisées pour le mapping Objet Relationnel, ces méthodes sont : ajouter, modifier, supprimer, trouverbyId et lister. Finalement, la classe ConnectionManager qui implémente le patron de conception (singleton) afin d’assurer la connexion à la base de données MySQL.

Couche Controller : Le package ma.fstt.servlet contient toutes les servlets essentielles pour chacune des espaces de l’application web « Espace client, Espace produit, Espace commande », ces servlets garantissent le traitement des données selon les actions de l'utilisateur pour envoyer les réponses ou le redirige vers les vues spécifiées (sous forme de pages jsp).

Cette illustration ci-dessous montre toutes les classes et les interfaces citées auparavant.



Couche vues : Les vues sont des pages JSP (JAVA SERVER PAGES), structurées avec du code HTML, ces pages incluent le code JAVA, en utilisant les scriptlets, et aussi les balises JSTL (JSP STANDARD TAG LIBRARY), afin de représenter les données reçues par la servlet.

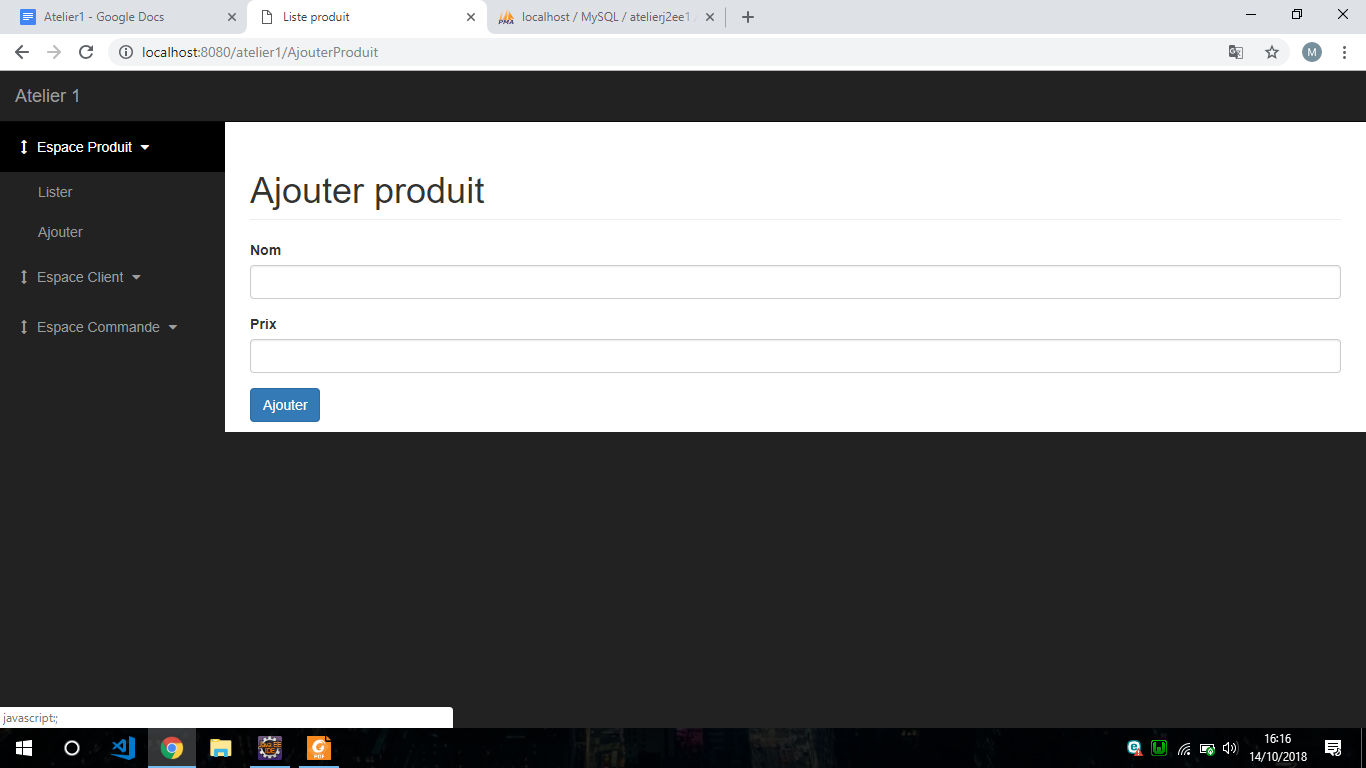
Nous avons également introduit du code Javascript et la bibliothèque de jQuery, pour dynamiser les pages et pour transférer des requêtes en arrière-plan (Ajax).

Pour les styles CSS, nous avons préféré d’utiliser une template d’Admin Dashboard, basée sur le framework de Bootstrap et jQuery, nous avons séparé les parties répétitives tel que la barre de navigation (navbar) du reste du contenu dynamique de la page, dans une autre page JSP, et on l’a réintroduit dans les pages avec la directive include.

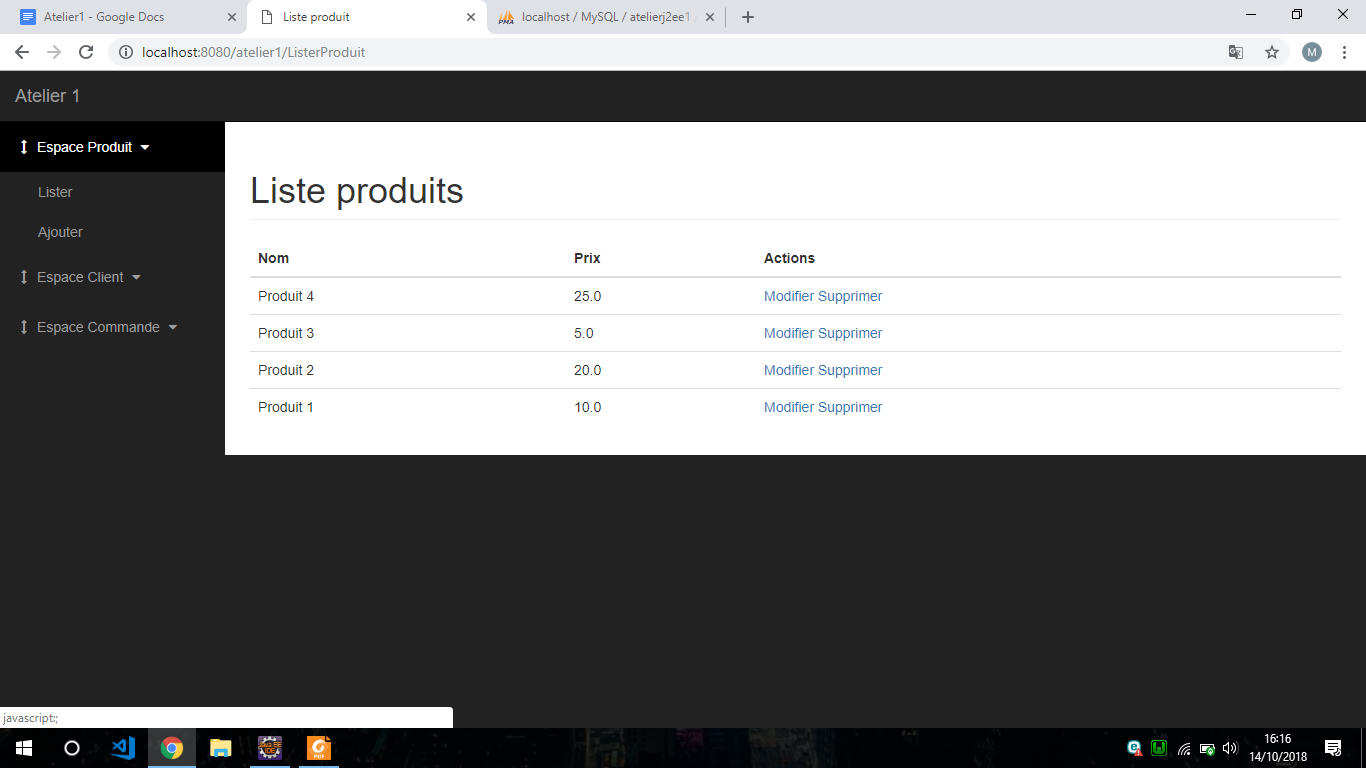
# Développement

Cette partie du rapport, consiste à la description de la phase de mise en œuvre de l’application. Nous y illustrons les fonctionnalités de notre application à travers quelques interfaces.

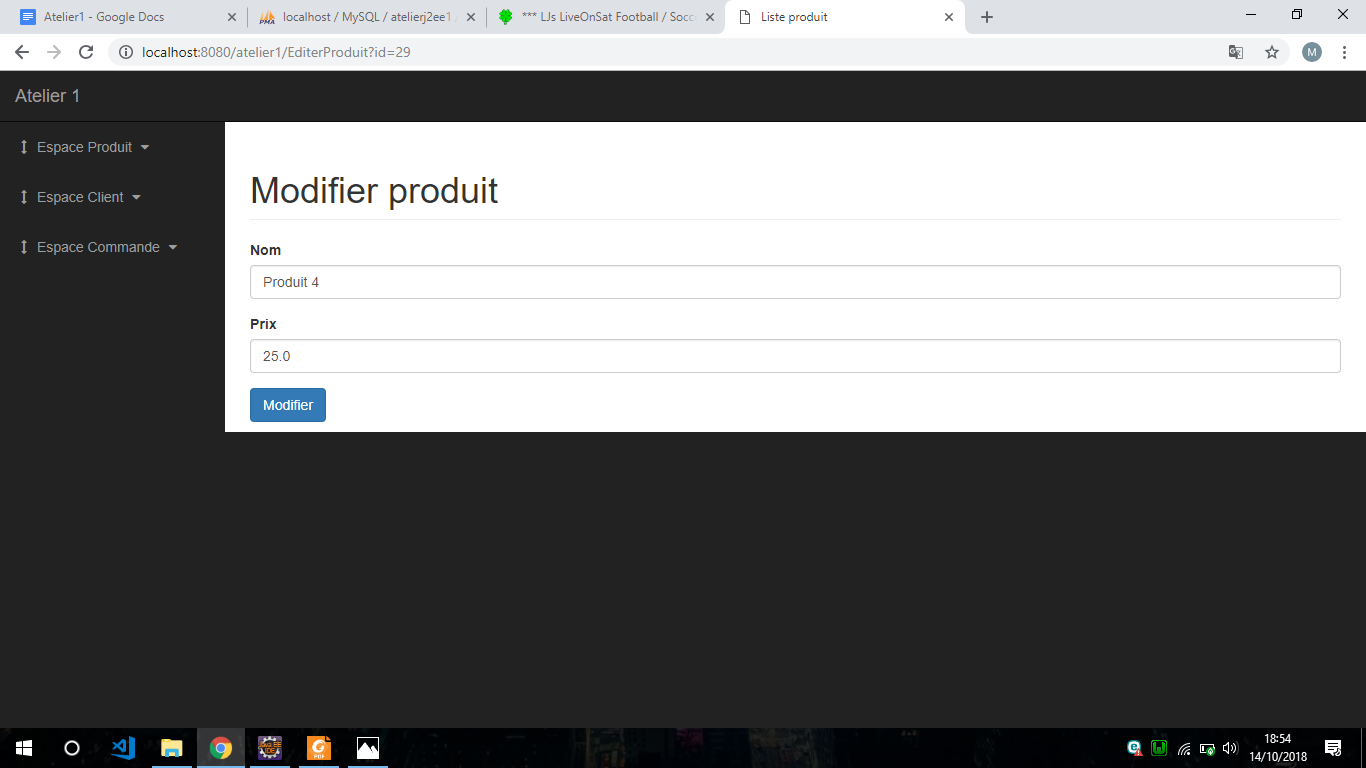
**Interface Ajouter produit :**



**Interface Liste produits :**



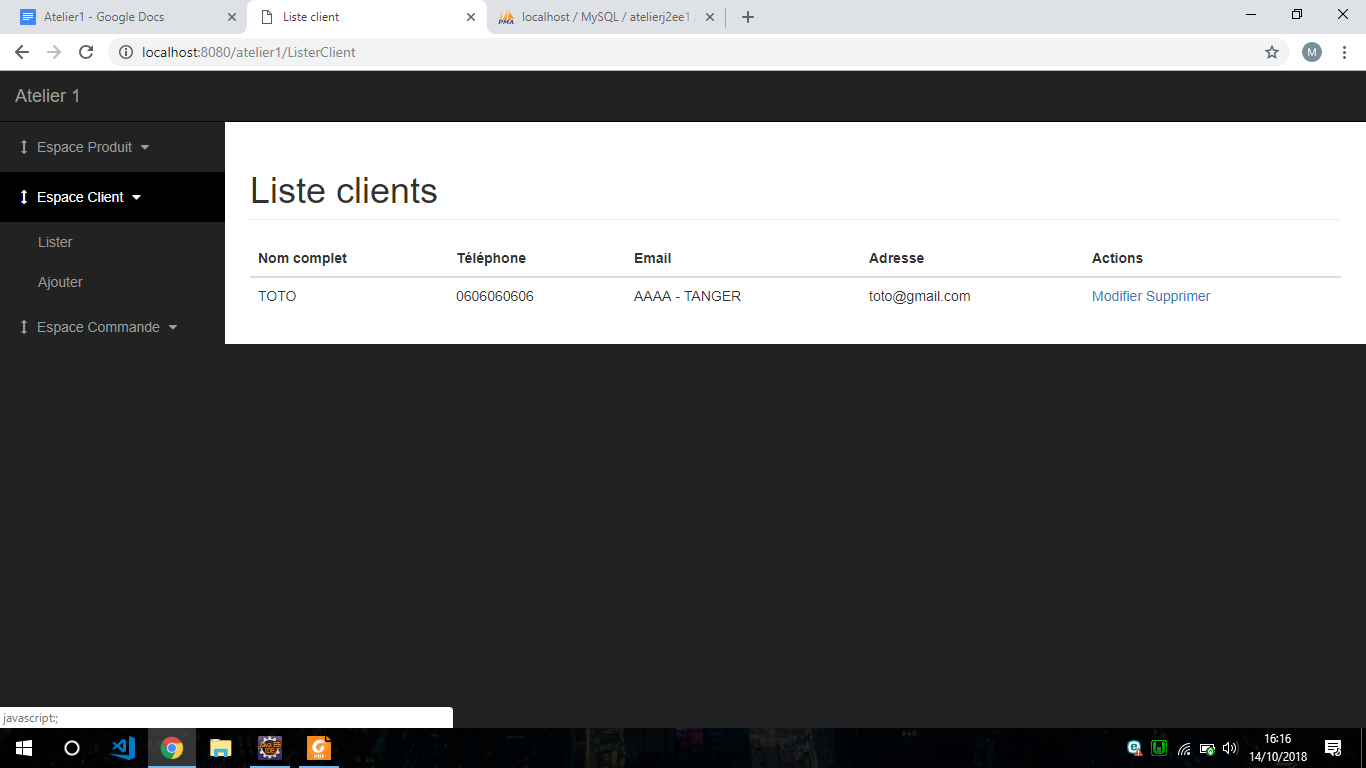
**Interface Modifier produits:**



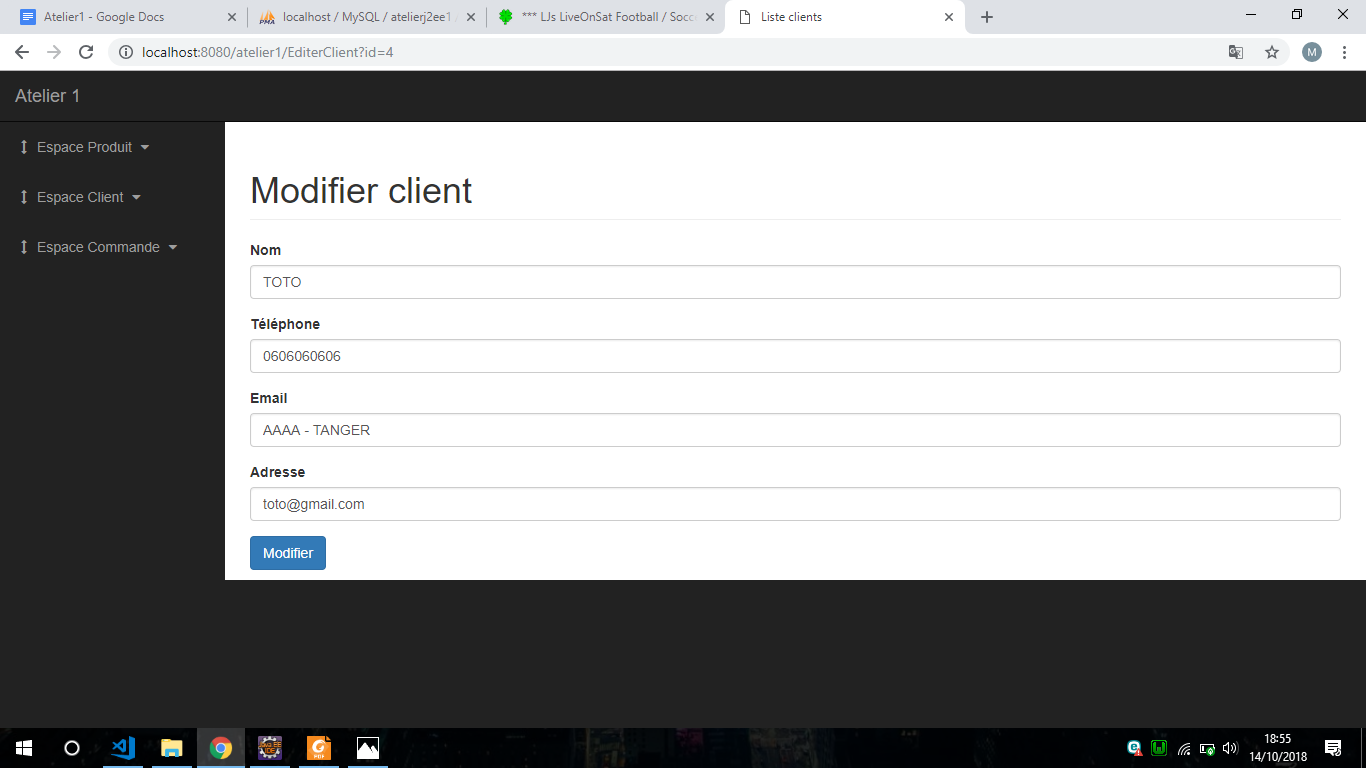
**Interface Ajouter client:**

# 

**Interface Lister clients:**



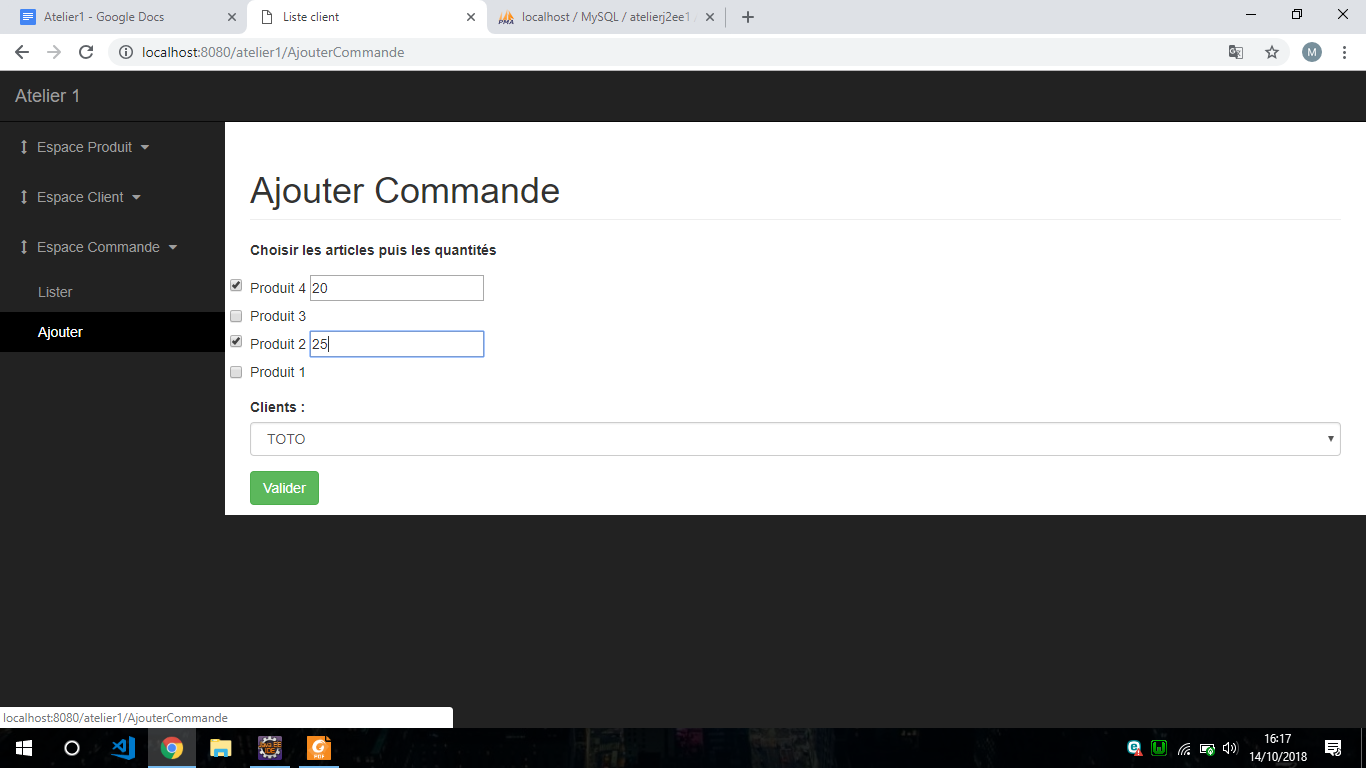
**Interface Modifier client:**



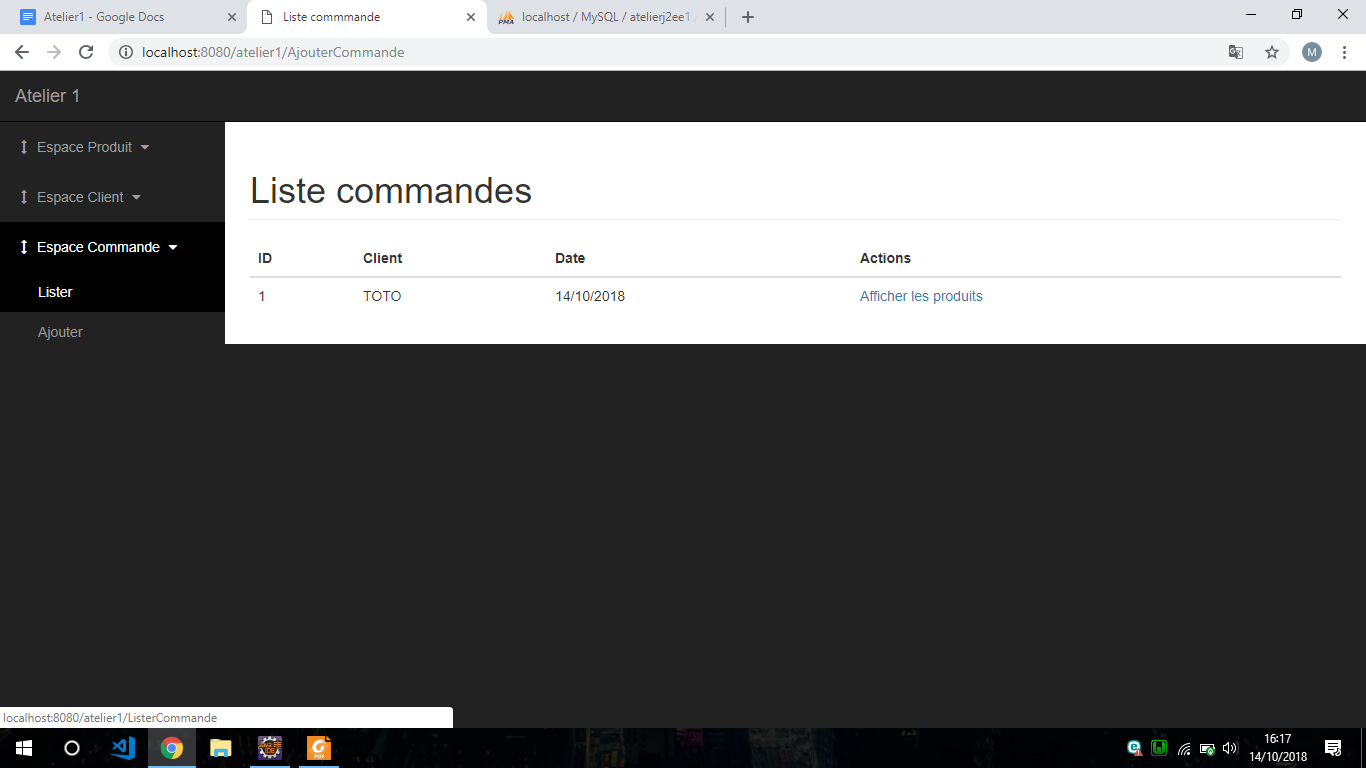
**Interface Ajouter commande:**

# 

Quand l’utilisateur coche un produit, un input s’affiche pour introduir la quantité respective.



**Interface Liste commandes:**



Quand l’utilisateur clique sur Afficher les produits, une boîte de dialogue s’affiche contient la liste des produits commandés, avec leurs quantités.

# 

# 

# Conclusion et perspectives

Dans notre cursus du cycle ingénieur en Logiciels et Systèmes Intelligents, nous sommes amenés à réaliser plusieurs projets et ateliers dans des différents modules, celui du module d’Architecture Web Distribuée J2EE d’une application Web pour la gestion des commandes.

Pendant la réalisation de cette application, nous avons pu se familiariser davantage avec l’environnement d’Eclipse et manipuler en toute aisance les composants du projet Web Dynamique sous JEE.

Notre travail ne s’arrête pas à ce stade, nous visons à améliorer cette application en perfectionnant d’abord les fonctionnalités présentes, et d’introduire d’autres fonctionnalités interactives.