INSTITUT SUPÉRIEUR DES SCIENCES APPLIQUÉES
ET DE TECHNOLOGIE DE SOUSSE

Rapport de Stage : Gestion des stocks pour la société SAFI

Stagiaire :
Mehrez Ahmed Aziz

Superviseur :
Hbazi Walid

1ére Ingénierie en informatique

Remerciements

Je tiens dans ce modeste rapport à remercier en premier lieu la ISSAT qui m'a donner la possibilité de faire passer un stage de fin d'étude dans une établissement public à fin de faire le rapprochement de mes connaissances théoriques à celles pratiques.

Je tiens aussi à remercier Mr **Hbazi Walid** le Directeur de SAFI qui m'a accepté dans son honorable établissement

Je voulais, exprimer ma profonde et respectueuse reconnaissance à mon encadreur **Mr Oussama Mehrez** pour leurs conseils inestimables, leur encouragement incessant et leur contribution efficace, qui m'ont guidé tout au long de mon stage.

J'ai le grand plaisir d'adresser mes sincères remerciements au personnel de la société « **SAFI** .

Sans oublier tous mes enseignants pour leurs contributions durant mes études universitaires, et d'exprimer mes sentiments de gratitude aux membres de Jury qui ont bien voulu accepter d'évaluer ce travail et de bien assister à la soutenance.

Table des matières

1. Introduction

Afin de comprendre le monde de l'entreprise et de prendre part à un véritable projet informatique, j'ai réalisé un stage avec l'entreprise SAFI, stage d'une période de 8 semaines, celui-ci commença au Lundi 01/07 et se termina le Samedi 31/08.
Pendant ce stage, j'ai appris à travailler dans un environnement technique dans lequel j'ai pu mettre en application des concepts théoriques et pratiques acquis à l'ISSATso.

Ce projet ayant pour but d'être également utilisé par d'autres entreprises, il me permet également de travailler sur un projet en respectant les contraintes et demandes émises par les potentiels clients.

2. Présentation entreprise

SAFI est une entreprise Tunisienne fondée en 2020, elle est basée à Monastir.
Il s'agit d'une entreprise industriel spécialisé dans le chaud froid, posséde une branche de développement appelée SAFI Dev.

Pour le moment, le fondateur, Mr Walid et 6 employées travaillent dans l'entreprise mais celle-ci possède plusieurs projets annexes qui sont eux liés à d'autres acteurs.

Gestion de stock est un projet de cette entreprise et si celui-ci s'avère être prometteur, une structure à part entière lui sera dédié par le biais d'une nouvelle société.

3. Projet de stage

3.1. **Objectifs de stage**

De manière générale, le stage consiste à réaliser un projet qui respecte les exigences du maître de stage et donc de l'entreprise. Ainsi, il faut créer le projet depuis le début, imaginer le fonctionnement de celui-ci, le structurer et le programmer afin qu'il soit fonctionnel.

Le stage présenté ici consiste à réaliser un prototype d'un service web, programmé en Spring Boot avec une base de données MySQL ainsi que d'autres parties de code en React Js.

L'objectif est de réaliser un prototype permettant à un utilisateur de site web de se connecter avec un compte Google ou par son adresse email. Puis de faire la gestion de stock de l'entreprise avec un accès donné précédemment par l'admin.

La première partie du stage consiste à réaliser le site web , qui contiendra des articles et les type d'utilisateurs, simulant donc se site web qui demandera aux utilisateurs de se connecter.

Celui-ci possèdera sa propre base de données en SQL qui contiendra les données concernant les articles et aussi les utilisateurs qui peuvent être soit : admin, vendeur, client ou fournisseur.

La deuxième partie du stage consiste alors à réaliser les différent fonctionnalités qui se chargeront des connexions et aussi de savoir quel utilisateurs à acheter\vendre quel article, afin de bien gérer le stock.

Celui-ci possède également une base de données SQL qui contient les utilisateurs, les transactions que les utilisateurs ont fait afin d'accéder à leurs articles.

La dernière partie du stage consiste à réaliser un tableau de bord qui présenter les statistiques de dernier année de cette entreprise. Ainsi que de faciliter le contrôle d'autre utilisateur de site de priorité plus faible.

3.2. **Etapes du projet**

a) Site Web : Gestion de stock

La première étape du projet a consisté à concevoir et développer l'interface utilisateur du site web dédié à la gestion des stocks. Cette interface devait permettre d'afficher une liste complète des articles disponibles dans l'entrepôt. Pour cela, une base de données relationnelle a été créée avec des tables représentant les articles, incluant des champs tels que le nom, la description, la quantité disponible, la date d'ajout, et le prix unitaire.

Une attention particulière a été portée à l'ergonomie de l'interface, avec un affichage sous forme de tableau permettant aux utilisateurs d'avoir une vue claire et synthétique de l'ensemble des articles.

Des filtres ont également été intégrés pour faciliter la recherche, par exemple par nom d'article, catégorie ou disponibilité en stock.

En parallèle, le développement de la section relative à la gestion des utilisateurs a été réalisé. Le site prévoit plusieurs types d'utilisateurs: les administrateurs, les vendeurs, les clients et les fournisseurs.

Chaque type d'utilisateur possède des droits d'accès et des fonctionnalités spécifiques. Par exemple, les administrateurs ont un accès complet au système : ils peuvent ajouter, modifier ou supprimer des articles et gérer les comptes des autres utilisateurs.

Les vendeurs peuvent consulter le stock, enregistrer des ventes ou des entrées de produits. Les clients peuvent visualiser uniquement les articles auxquels ils ont été autorisés, et les fournisseurs peuvent ajouter de nouveaux produits sous autorisation.

Pour rendre cette gestion plus dynamique et sécurisée, un système de rôles a été intégré dans la base de données et géré au niveau du backend à travers des vérifications à chaque appel d'API.

Un autre point essentiel a été la mise en place du système d'**authentification** et de **gestion de la connexion**. L'utilisateur peut se connecter soit via un formulaire classique (email et mot de passe), soit via une authentification OAuth avec Google.

Lorsqu'un utilisateur tente d'accéder à une section protégée du site sans être connecté, il est redirigé vers une page de connexion.

Une fois connecté, le site lui donne un accès personnalisé selon son rôle. Cette logique permet de sécuriser les données sensibles et de restreindre l'accès aux fonctionnalités avancées.

La base de données MySQL joue ici un rôle central dans le projet. Elle permet de stocker toutes les informations importantes relatives aux produits, aux utilisateurs, aux transactions, et aux droits d'accès.

Elle a été conçue de manière relationnelle avec des jointures entre les différentes entités pour assurer l'intégrité des données. Des mécanismes de validation et des contraintes ont été mis en place afin de garantir que les données soient cohérentes, à jour et fiables. Par exemple, il est impossible d'enregistrer une vente si la quantité en stock d'un produit est insuffisante.

Enfin, une attention particulière a été portée à l'**affichage des données** sur le site. Une fois qu'un utilisateur est connecté, il peut visualiser les données qui le concernent via des interfaces modernes développées avec ReactJS.

Les articles sont présentés dans un tableau interactif, permettant des opérations comme le tri ou la pagination. Les informations affichées varient selon les privilèges de l'utilisateur : un vendeur peut voir toutes les transactions récentes, tandis qu'un client voit uniquement ses propres commandes.

Des composants dynamiques et réactifs ont été utilisés pour améliorer l'expérience utilisateur, avec des messages d'erreur ou de confirmation pour guider l'interaction.

Cet affichage est constamment mis à jour grâce à des appels API qui permettent de récupérer les dernières données en temps réel.

b) API

Le développement de l'API a représenté une étape cruciale dans l'architecture du projet, car elle constitue le pont entre le frontend développé en ReactJS et le backend construit avec Spring Boot.

L'API REST a été mise en place de manière à assurer une communication fluide, sécurisée et efficace entre ces deux parties. La première sous-étape a été la création des **fichiers de connexion** ainsi que la **configuration globale** de l'API.

Cela inclut la mise en place des contrôleurs REST (annotés avec `@RestController`), la définition des routes, et la configuration de l'accès à la base de données via JPA et Spring Data.

Des paramètres comme les ports, les chemins d'accès, le format JSON des réponses et les entêtes CORS ont été soigneusement définis dans le fichier `application.properties`.

Cette configuration initiale a permis de s'assurer que l'API pouvait communiquer correctement avec le frontend sans erreurs d'accès croisé, en particulier après le déploiement sur des serveurs différents.

Ensuite, une partie essentielle du développement de l'API a concerné la **gestion des requêtes SQL**. Même si l'ORM JPA a été utilisé pour simplifier les opérations CRUD (Create, Read, Update, Delete), certaines requêtes spécifiques ont nécessité l'écriture de requêtes SQL personnalisées (via `@Query`). Cela a permis par exemple de récupérer la liste des articles par type, de vérifier l'unicité d'un utilisateur, ou encore de générer des rapports de ventes filtrés par date.

Ces requêtes ont été encapsulées dans des repositories Java et testées pour s'assurer de leur efficacité. Un effort particulier a été fait pour éviter les failles de sécurité comme les injections SQL, en utilisant des paramètres liés (`?1`, `:paramName`) dans les requêtes.

Enfin, chaque fonctionnalité métier de l'API a été codée dans des **fichiers de fonctions spécifiques**. Ces fichiers, appelés services (`@Service`) ou contrôleurs, contiennent la logique métier du

projet: validation des données, traitement des entrées, gestion des erreurs, et formatage des réponses.

Par exemple, une fonction peut recevoir un identifiant d'article et un identifiant d'utilisateur, vérifier que cet utilisateur est autorisé à voir cet article, puis retourner les informations sous format JSON.

Chaque fonction est pensée de manière modulaire et réutilisable, facilitant ainsi la maintenance future du projet. L'architecture MVC (Model-View-Controller) a été respectée afin d'assurer une séparation claire des responsabilités, ce qui a également permis d'implémenter facilement des tests unitaires sur chaque composant du backend.

c) Dialogue

Le **dialogue** entre le site web (frontend) et l'API (backend) a été soigneusement élaboré afin de garantir une communication fluide, sécurisée et performante.

La **mise en place de l'API sur le site web** a commencé par l'intégration des appels HTTP via des bibliothèques comme `axios` ou `fetch`. Ces appels sont effectués depuis les composants React, soit au chargement des pages (avec `useEffect`), soit après une action utilisateur (comme un clic sur un bouton).

Chaque requête est dirigée vers une route spécifique de l'API backend, en respectant les méthodes HTTP appropriées : `GET` pour récupérer des données, `POST` pour créer, `PUT` pour mettre à jour, et `DELETE` pour supprimer.

Afin d'éviter les erreurs de politique de sécurité de navigateur (CORS), une configuration spécifique a été appliquée côté backend pour autoriser les domaines du frontend.

La **gestion de la connexion** entre le frontend et le backend repose sur un système de **tokens d'authentification**. Lorsqu'un utilisateur se connecte, un token (JWT ou simple UUID) est généré par le backend et envoyé au frontend, qui le stocke dans la mémoire locale (localStorage ou sessionStorage).

Ce token est ensuite joint à chaque requête API dans les entêtes HTTP (`Authorization`) pour authentifier l'utilisateur. Le backend vérifie la validité du token avant de répondre. Si le token est invalide ou expiré, une réponse d'erreur est envoyée, demandant à l'utilisateur de se reconnecter.

Ce système permet d'assurer la sécurité des échanges tout en maintenant une expérience utilisateur fluide et sans rechargement de page.

Enfin, plusieurs **fonctions secondaires** ont été établies pour enrichir le dialogue entre le site et l'API. Cela inclut, par exemple, la gestion des rôles utilisateur (pour afficher certaines sections du site uniquement à certains types d'utilisateurs), la notification des erreurs (ex : mauvais mot de passe ou article indisponible), ou encore la récupération dynamique des données sans rechargement de page. Des messages de confirmation et d'alerte ont été ajoutés en cas de succès ou d'échec d'une opération. Par ailleurs, un système de logs côté backend permet de suivre toutes les interactions avec l'API, facilitant ainsi le diagnostic des erreurs lors des phases de test ou en production.

d) Déploiment

Le déploiement du projet représente l'une des étapes les plus critiques car il consiste à faire passer l'application d'un environnement de développement local à un environnement de production réel, accessible via Internet.

Dans ce cadre, le déploiement a été effectué sur **deux serveurs VPS (Virtual Private Server)** distincts : l'un dédié au **frontend** développé en ReactJS, et l'autre au **backend** construit avec Spring Boot. Cette séparation permet d'assurer une meilleure scalabilité, une sécurité renforcée, ainsi qu'une gestion plus souple des performances.

La première étape du déploiement a été la **préparation des serveurs VPS**. Chaque serveur a été configuré avec un système d'exploitation Linux (Ubuntu Server), choisi pour sa stabilité et sa compatibilité avec les outils utilisés. Sur le VPS backend, Java 17 a été installé pour faire fonctionner l'application Spring Boot. Sur le VPS frontend, Node.js et npm ont été installés pour compiler et exécuter l'application React.

Un serveur Nginx a été utilisé sur les deux serveurs pour gérer les requêtes HTTP, jouer le rôle de reverse proxy et servir les fichiers statiques du frontend. Les ports par défaut ont été modifiés pour renforcer la sécurité, et un certificat SSL a été mis en place via Let's Encrypt afin de permettre l'accès HTTPS sécurisé.

Une fois les serveurs prêts, le code source du projet a été transféré sur chaque machine à l'aide de Git ou par téléchargement via SCP (Secure Copy). Pour le **backend**, l'application Spring Boot a été empaquetée en fichier `.jar` puis lancée avec la commande `java -jar`.

Le backend a été configuré pour se connecter à une base de données distante MySQL, également hébergée sur un VPS ou sur un service cloud sécurisé. Pour le **frontend**, les fichiers de build React ont été générés avec `npm run build` puis déployés dans le répertoire de Nginx pour être accessibles via le navigateur.

Une attention particulière a été portée à la **gestion du CORS (Cross-Origin Resource Sharing)**, indispensable dans une architecture à domaines séparés. En effet, les requêtes effectuées par le frontend vers l'API du backend proviennent de domaines différents, ce qui peut être bloqué par défaut par les navigateurs modernes.

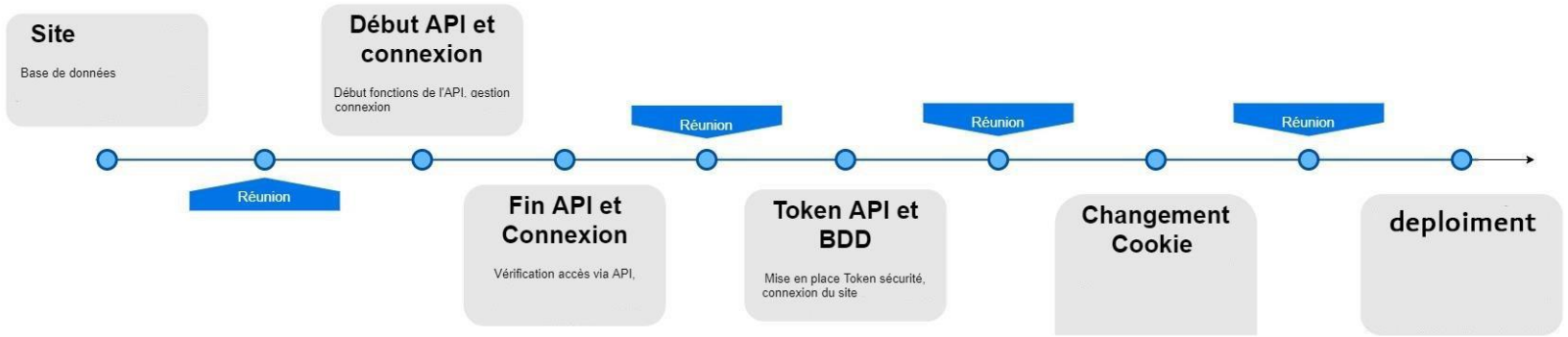
Pour y remédier, une configuration explicite a été ajoutée dans l'API Spring Boot afin d'autoriser les requêtes en provenance du domaine du frontend, tout en maintenant un niveau de sécurité élevé.

Enfin, après la mise en ligne des deux parties du projet, des **tests de bout en bout** ont été effectués pour vérifier que l'ensemble des fonctionnalités fonctionnait correctement dans un environnement réel. Ces tests ont révélé certaines erreurs liées aux chemins d'accès, aux cookies de session et à l'authentification croisée.

Grâce à l'analyse des logs serveur et à l'outil Postman, ces erreurs ont été rapidement identifiées et corrigées. Une fois stabilisé, le site a été accessible à tout utilisateur disposant d'une connexion internet, offrant ainsi un aperçu réaliste de la solution finale.

Ce processus de déploiement a permis non seulement de valider la faisabilité technique du projet, mais aussi de simuler son utilisation dans un contexte professionnel réel, avec toutes les contraintes que cela implique : sécurité, performances, accessibilité et robustesse.

4. Suivi du projet



Ce schéma illustre, sous la forme d’une ligne du temps, l’avancée du projet et la fréquence des réunions avec le maitre de stage ainsi que le superviseur.

5. Description du déroulement du stage

5.1. Site Web: Gestion de stock

5.1.1. Réalisation du site

La réalisation du site web a débuté par la conception de la structure de la base de données, qui constitue le cœur du système de gestion des stocks.

Tous les articles sont enregistrés dans une **base de données SQL**, où chaque article dispose d'attributs essentiels tels que le nom, la description, la quantité, le prix, la date d'ajout, et le fournisseur associé. Ces informations permettent de gérer efficacement l'état du stock et d'en assurer le suivi précis.

À ce stade du projet, **aucune interface d'ajout ou de modification d'articles n'a encore été intégrée dans le site web**. Ainsi, l'insertion des articles se fait directement via des requêtes SQL exécutées dans l'interface d'administration de la base de données (ex. : MySQL Workbench ou phpMyAdmin).

Cette méthode, bien que temporaire, a permis de se concentrer sur le développement des fonctionnalités de consultation, d'affichage et de gestion des droits d'accès des utilisateurs avant de développer une interface d'administration complète.

Concernant les utilisateurs, le site repose sur un **système de rôles**, répartis en quatre catégories distinctes : **administrateur, vendeur, client et fournisseur**.

Chaque type d'utilisateur possède des privilèges spécifiques adaptés à ses besoins. Par exemple, l'administrateur a un accès complet à toutes les fonctionnalités du site, notamment la gestion des utilisateurs et des articles ; le vendeur peut effectuer des ventes et consulter les stocks ; le client a un accès limité aux articles autorisés à la vente ; et le fournisseur peut consulter et, à terme, proposer des produits à intégrer dans le stock.

Cette structure permet une gestion fine des autorisations et garantit la sécurité et la pertinence des actions effectuées par chaque utilisateur.

5.1.2. **Connexion**

Le système de connexion constitue une partie essentielle du site, permettant de filtrer l'accès aux informations en fonction des droits attribués à chaque utilisateur. Lorsqu'un utilisateur **non connecté** tente d'accéder à un article, l'accès lui est **automatiquement restreint**. Ce dernier ne pourra consulter les détails de l'article que **si l'administrateur lui a accordé les droits nécessaires**.

Dans le cas contraire, il a la possibilité d'envoyer une **demande d'accès**, qui sera ensuite examinée par un utilisateur disposant d'un niveau de priorité supérieur, comme un administrateur ou un gestionnaire.

Pour initier le processus de connexion, l'utilisateur peut cliquer sur le bouton **« Connexion »** situé sur la page d'accueil. Ce bouton déclenche l'ouverture d'un **modal (fenêtre contextuelle)** contenant les formulaires d'**inscription** et de **connexion**. Ainsi, un nouvel utilisateur peut créer un compte, tandis qu'un utilisateur déjà enregistré peut se connecter en saisissant ses identifiants.

Une fois la connexion établie, l'utilisateur est redirigé vers son espace personnel, où il peut désormais **accéder aux articles disponibles** en fonction de son rôle. Ce système

d’authentification permet de sécuriser l’accès aux données sensibles tout en assurant une navigation personnalisée selon le profil de l’utilisateur.

5.1.3. **Base de données**

La **base de données** constitue la fondation technique du projet de gestion de stock. Elle est implémentée en utilisant **MySQL**, un système de gestion de base de données relationnelle robuste, performant et largement utilisé dans le monde du développement web. Toutes les informations importantes de l’application y sont centralisées : données des utilisateurs, liste des articles, historique des transactions, rôles et permissions, ainsi que les journaux d’activité.

La structure de la base de données repose sur un **modèle relationnel bien normalisé**, composé de plusieurs tables interconnectées à travers des relations (clés étrangères). Les principales tables sont :

- **`utilisateurs`** : contient les informations de base de chaque utilisateur (nom, prénom, email, mot de passe crypté, rôle, date d'inscription).
- **`articles`** : regroupe les produits disponibles dans le stock avec des champs tels que l'identifiant unique, le nom de l'article, la quantité disponible, la description, le prix unitaire, la date d'ajout et le fournisseur associé.

- **`transactions`** : enregistre toutes les opérations effectuées sur les articles (vente, achat, ajout ou retrait de stock). Chaque ligne mentionne l'article concerné, l'utilisateur responsable, la date et le type d'opération.
- **`roles`** : permet d'associer chaque utilisateur à un rôle spécifique (admin, vendeur, client, fournisseur), facilitant la gestion des autorisations au sein de l'application.
- **`demandes_acces`** : stocke les demandes envoyées par les utilisateurs non autorisés souhaitant accéder à certains articles.

L'intégration avec l'application est assurée via **JPA (Java Persistence API)**, qui permet d'interagir avec la base de données en utilisant des entités Java au lieu d'écrire directement des requêtes SQL complexes. Cela simplifie considérablement le code et garantit une meilleure maintenabilité.

Des **contraintes d'intégrité** (clé primaire, clé étrangère, contraintes `NOT NULL`, `UNIQUE`, etc.) ont été mises en place pour garantir la cohérence des données. De plus, une **validation côté backend** est effectuée avant toute insertion ou mise à jour, afin d'éviter les incohérences ou les saisies erronées.

Pour renforcer la **sécurité**, les mots de passe des utilisateurs sont **cryptés** à l'aide d'algorithmes de hachage comme **BCrypt**, rendant impossible leur lecture en cas d'accès illégal à la base. Les tokens d'authentification sont également stockés temporairement dans une table séparée, avec une date d'expiration, permettant une gestion sécurisée des sessions.

Enfin, la base de données est régulièrement **sauvegardée** (backup) à l'aide de scripts automatisés, afin d'éviter toute perte d'information en cas de panne système ou d'erreur humaine. Des outils tels que **MySQL Workbench** ou **DBeaver** ont été utilisés pour modéliser, interroger et administrer la base de données de façon intuitive tout au long du développement du projet.

5.1.4. **Affichage données**

L’affichage des données est un élément central de toute application web orientée utilisateur. Dans le cadre de ce projet, cette responsabilité a été confiée au **frontend**, développé à l’aide de **ReactJS**, une bibliothèque JavaScript moderne, populaire et performante, conçue pour la création d’interfaces utilisateur interactives et réactives.

Le choix de ReactJS a été motivé par sa capacité à rendre les interfaces dynamiques tout en assurant une séparation claire des composants, ce qui facilite le développement, la maintenance et l’extensibilité de l’application.

Une fois qu’un utilisateur est connecté, les données le concernant (par exemple ses informations personnelles, la liste des articles qu’il peut consulter ou acheter, ses crédits disponibles, etc.) sont récupérées via des **appels à l’API backend** (sous forme de requêtes HTTP) et affichées dynamiquement dans l’interface.

L’interaction entre le frontend et le backend repose sur des **requêtes asynchrones** envoyées grâce à des fonctions

JavaScript, souvent en utilisant la bibliothèque `axios` pour simplifier la communication HTTP.

ReactJS, en tant que technologie de rendu côté client (**client-side rendering**), permet de construire l'interface utilisateur à partir de **composants réutilisables**, comme par exemple : `Navbar`, `TableProduits`, `FormulaireConnexion`, etc. Chaque composant est responsable d'un aspect précis de l'interface et peut être facilement modifié ou mis à jour indépendamment des autres.

Par exemple, le tableau d'affichage des articles (produits en stock) est un composant qui récupère les données de l'API et les affiche dans un format lisible, souvent sous forme de tableau interactif avec tri, pagination, ou recherche dynamique.

Le langage **JavaScript**, utilisé avec React, permet d'ajouter une couche d'interactivité et de logique métier dans le navigateur de l'utilisateur. Il permet par exemple d'actualiser certaines parties de la page sans la recharger entièrement, ce qui améliore significativement l'expérience utilisateur.

Lorsqu'un utilisateur clique sur un bouton pour consulter les détails d'un article ou pour filtrer les résultats, JavaScript intercepte l'événement, envoie la demande à l'API, récupère la réponse et met à jour uniquement le composant concerné.

Pour améliorer encore l'expérience utilisateur, des éléments visuels tels que des **modales**, des **menus déroulants**, des **notifications de succès ou d'erreur**, et des **icônes dynamiques** ont été intégrés. Ces éléments sont pilotés par des états internes de React (`useState`, `useEffect`, etc.), permettant une gestion fine de l'affichage conditionnel selon les actions de l'utilisateur.

De plus, ReactJS offre un **écosystème riche** de bibliothèques complémentaires comme **React Router** (pour gérer la navigation entre les pages), **Formik** (pour la gestion des formulaires), ou **Bootstrap/Material UI** (pour le design visuel des composants).

Dans notre cas, certains composants de style ont été utilisés pour garantir une interface moderne, responsive, et agréable à utiliser sur différents types d'appareils (ordinateurs, tablettes, téléphones).

En résumé, le couple **ReactJS + JavaScript** a joué un rôle fondamental dans le développement du frontend du site, en rendant l'affichage des données non seulement fluide et dynamique, mais aussi adaptable aux différents besoins des utilisateurs selon leur rôle.

Cette technologie assure une **expérience utilisateur moderne, réactive et sécurisée**, tout en facilitant le travail du développeur via une architecture modulaire et bien structurée.

5.2. **API**

5.2.1. **Affichage données**

L'API va permettre au site de récupérer des informations à l'aide de l'API Google afin de gérer la connexion et l'accès à certains articles.

Plusieurs fichiers permettent de configurer cette API, notamment la connexion
à la base de données du site et la création de l'objet Client, qui contiendra certains attributs par défaut et qui permettra de stocker les informations d'un client.

Ces informations peuvent être récupérer depuis la base de données, et ensuite enregistrés, lorsque qu'un nouveau client souhaite s'inscrire sur le site par exemple.

5.2.2. **Requête SQL**

La gestion des demandes d’informations dans une base de données à travers des requêtes SQL est une composante essentielle dans le développement d’applications modernes. Pour que cela soit opérationnel, il est nécessaire de passer par des requêtes HTTP, qui permettent de communiquer avec le backend de l'application, de solliciter des données et de recevoir une réponse. Cependant, ce processus nécessite plusieurs étapes pour garantir que la communication entre l'interface utilisateur (front-end) et le serveur (back-end) fonctionne correctement.

La gestion des requêtes SQL via HTTP

Dans un système où des données doivent être extraites d'une base de données, il est courant d'utiliser des requêtes SQL. Ces requêtes permettent de sélectionner, insérer, mettre à jour ou supprimer des données selon les besoins de l’application. Pour intégrer ces requêtes dans une application web, on utilise souvent des requêtes HTTP qui permettent d’envoyer des informations au serveur et d'obtenir une réponse. L'API (Application Programming Interface) qui sert de passerelle entre le client (le front-end) et la base de données est donc essentielle pour gérer ces échanges.

Dans ce cadre, l'API expose des points de terminaison (endpoints) que le front-end peut appeler via des requêtes HTTP. Chaque appel à l'API correspond à une opération spécifique qui sera traduite en une requête SQL. Par exemple, un appel HTTP `GET` pourrait être utilisé pour récupérer des données, et un appel `POST` pour envoyer des informations à la base de données. Ces requêtes HTTP ne se contentent pas de demander des données ; elles contiennent aussi des paramètres nécessaires à l'exécution de la requête SQL.

L'implémentation dans l'objet Client

Dans un système orienté objet, l'objet **Client** joue souvent le rôle central dans la gestion des interactions avec l'API. Cet objet peut être conçu pour inclure des fonctions spécifiques qui appellent des requêtes HTTP et manipulent les réponses. Cependant, ce n'est pas suffisant de simplement appeler une fonction au sein de l'objet **Client** pour garantir une gestion correcte de l'API. Il faut aussi s'assurer que les paramètres nécessaires pour la requête SQL sont correctement définis et envoyés.

Prenons un exemple concret : si un utilisateur souhaite consulter les détails d'un produit dans une base de données, l'API doit être capable de recevoir l'ID du produit sous forme de paramètre. L'objet **Client** doit donc préparer cet ID et l'envoyer dans la requête HTTP. Mais il faut également que l'API sur le serveur soit capable de gérer cet ID, de l'utiliser dans une requête SQL afin de rechercher les informations dans la base de données et renvoyer la réponse adéquate.

Gestion de la réponse et des paramètres

L'un des défis principaux réside dans la gestion de la réponse et l’accès aux paramètres qui seront utilisés pour effectuer la requête SQL. Lorsque la requête HTTP est effectuée, le serveur doit être capable de traiter les paramètres reçus, d’exécuter la requête SQL correspondante et de renvoyer une réponse au client.

La gestion des réponses est cruciale pour que l’application puisse réagir correctement. Par exemple, si la requête SQL retourne une erreur (comme une donnée manquante ou une violation de contrainte), l’API doit être capable de retourner une réponse d'erreur claire, de préférence sous un format structuré (souvent en JSON). Cette réponse doit être gérée par le côté client pour informer l'utilisateur de l'erreur ou pour prendre les mesures appropriées.

De plus, l’accès aux paramètres pour exécuter la requête SQL doit être correctement sécurisé. Par exemple, les paramètres reçus via HTTP doivent être validés pour éviter des attaques telles que l'injection SQL, où des données malveillantes pourraient être insérées dans la requête SQL. Pour cette raison, il est essentiel que l’API vérifie les paramètres avant de les inclure dans une requête SQL.

Fichier API pour la gestion des requêtes

Pour organiser cette logique, chaque requête HTTP est généralement associée à un fichier dédié dans l'API, qui contient la logique spécifique pour gérer cette requête. Ce fichier API est responsable de la gestion des différentes étapes de la requête : réception des paramètres, validation, exécution de la requête SQL, et envoi de la réponse.

Dans un cadre structuré, chaque fichier de l'API pourrait correspondre à un ensemble de fonctionnalités bien définies. Par exemple, un fichier `produits.php` pourrait gérer toutes les requêtes liées aux produits (comme obtenir la liste des produits, ajouter un nouveau produit, ou mettre à jour un produit existant). Ce fichier serait chargé de récupérer les paramètres nécessaires à partir de la requête HTTP, d'effectuer la requête SQL et de renvoyer la réponse appropriée au client.

En résumé, la gestion des requêtes SQL via des requêtes HTTP implique de bien structurer l'API pour qu'elle puisse accepter les paramètres, effectuer les requêtes SQL et gérer les réponses de manière adéquate. L'objet **Client** sert de médiateur pour envoyer les requêtes HTTP, mais la logique de traitement et de gestion des erreurs réside dans les fichiers de l'API qui doivent être soigneusement développés et sécurisés.

5.2.3. **Client et Fonction API**

Afin d'utiliser convenablement l'API, chaque fonction de celle-ci est représentée par un fichier .java, celui-ci va gérer la demande en provenance du site, exécuté la demande et envoyer une réponse au format JSON permettant au site de modifier son comportement en fonction de la réponse reçue.

Les fichiers fonctions de l'API vont d'abord établir la connexion avec la base de données, dans le but de récupérer ou y stocker des informations.
Ensuite, il faut gérer les paramètres qui seront utilisés par l'API lors du déroulement de sa requête.

Le plus souvent, ces paramètres sont des ID de client ou d'article ou alors des informations de connexion (nom de compte et mot de passe).

Une fois les paramètres obtenus, le code va les traiter et appeler une fonction contenu dans l'objet Client de l'API, les paramètres seront alors transmis à la fonction de l'objet Client qui va les utiliser afin de réaliser une requête SQL avec la base de données.

Une fois la requête effectuée, le code du fichier .java vérifie le contenu des informations données par la requête, les traite si celles-ci doivent être traité et va les encoder en JSON dans la réponse de l'API afin de les transmettre au site. Cependant, il faut avant cela interpréter ses informations afin de renvoyer une réponse correcte au site.
Si la requête à échouer ou que la base de données ne possèdent pas l'information demandée, la réponse sera alors adaptée et permettra au site d'agir en conséquence.

La réponse contient soit l'information demandée, soit un OK affirmant que tout s'est passé normalement, soit un message d'erreur.

5.3. **Dialogue**

5.3.1. **Site et API**

Le site doit installer une partie de code qui va permettre d'utiliser l'API et gérer son fonctionnement en fonction de la réponse de celle-ci.

Cette portion de code est installée soit dans le Contrôleur du site, soit dans un code JavaScript. Pour déterminer dans quelle zone la portion de code sera installée, il faut faire attention au rôle de celle-ci.

En effet, le code installé dans le contrôleur sera exécuté coté Serveur, alors que le code qui est installé en JavaScript (React Js) sera quant à lui exécuté coté Client, et donc depuis l'ordinateur de l'utilisateur, sans passé par le site.

Coté serveur, il s'agit principalement des fonctions permettant d'effectuer des requêtes HTTP à l'API de la BDD, afin d'achever les actions du client à certains articles du site.

Coté client, tous fait avec React Js qui est « client side rendering ».

5.3.2. **Token API**

L'API est le principal moyen permettant de dialoguer entre le frontend et le backend.

Cependant, il faut faire une vérification afin de déterminer quel site a le droit d'utiliser l'API et ses fonctions. Cette vérification va permettre d'éviter que n'importe quels sites internet puissent accéder à l'API et utiliser sans autorisations les fonctions de celle-ci ou spammer l'API et la rendre non fonctionnelle.
Les fonctions de l'API, en plus de demander des paramètres lorsque ceux-ci sont nécessaires, demandent également un **token**, celui-ci ne doit être connu que par l'utilisateur. Le token va permettre d'authentifier l'utilisateur du site, lui laissé l'accès au fonction de l'API, cependant si le token est incorrecte, la requête HTTP est interrompue et la réponse de l'API est un message précisant que le token est faux.

La mise en place de ce token se fait lorsque le backend et le frontend se mettent d'accord et acceptent de collaborer. Dans la base de données, on enregistre le provider ainsi qu'un nom de compte et un mot de passe.

Ces 2 informations de connections sont transmises de façon privé au site, et lorsque celui-ci souhaite se connecter au site, il inscrit dans sa requête ses informations de connexion, ensuite l'API vérifie et si elles sont correctes, lui renvoi le token.

5.4. **deploiment**

Le déploiement d'un site web créé avec Spring Boot, MySQL et ReactJS implique plusieurs étapes pour s'assurer que les différentes parties de l'application fonctionnent ensemble de manière fluide sur un serveur en production.

Tout d'abord, le backend Spring Boot, qui sert à la gestion des requêtes, des données et de la logique métier, doit être empaqueté sous forme d'un fichier `.jar` (ou `.war`) via la commande `mvn clean package`.

Ce fichier est ensuite déployé sur un serveur d'application tel que Apache Tomcat ou tout autre serveur compatible avec Spring Boot, ou bien il peut être exécuté directement avec une commande Java. Il est crucial que la configuration du backend inclut les paramètres nécessaires à la connexion à la base de données MySQL, tels que l'URL de la base de données, le nom d'utilisateur, le mot de passe, ainsi que le driver JDBC.

Ensuite, pour le frontend développé avec ReactJS, il doit être construit en utilisant la commande `npm run build`, ce qui génère un dossier `build/` contenant les fichiers statiques (HTML, CSS, JS) prêts à être servis. Ces fichiers peuvent être déployés sur un serveur web, tel qu'Apache ou Nginx, qui servira les fichiers statiques aux utilisateurs.

Une fois que le frontend est configuré, il doit être relié au backend Spring Boot via des API REST, pour récupérer et envoyer des données.

Le déploiement complet nécessite également la mise en place d'un environnement de base de données MySQL, qui peut être hébergé sur un serveur distant ou local, en veillant à ce que les configurations du backend pointent vers cette base de données et que la structure des tables soit en place.

Enfin, des outils de gestion de serveur comme Docker peuvent être utilisés pour faciliter le déploiement de l'application dans des containers, permettant une gestion plus simple des dépendances et des environnements.

Une fois déployé, le site est accessible via un nom de domaine ou une adresse IP, et la communication entre le frontend et le backend doit être vérifiée pour s'assurer que les requêtes sont correctement traitées et que les données sont bien échangées entre le client et le serveur.

6. Conclusion

6.1. **Bilan objectifs**

- ✓ Le site Web a été correctement organisé et beaucoup des fonctionalité son livrable.

- ✓ Le site de backend permet bien d'accéder aux informations du compte d'un client/vendeur/admin/fournisseur.

- ✓ Le dialogue entre les 2 sites a été mis en place, que ce soit à travers une API qui va permettre de gérer les accès ou via les redirections dans lesquels certaines informations sont stockées, le dialogue est établi et permet aux 2 sites de communiquer.

- ✓ La mise en place sur les VPS s'est faite correctement et permet aux 2 sites de profiter d'un domaine différent sans pour autant nuire au fonctionnement du projet.

6.2. **Perpectives TFA**

- Tester le prototype sur un véritable site Web extérieur au projet et permettre la compatibilité entre ceux-ci

- Etablir la demande des utilisateurs et la demande des sites de frontend afin de faire un état des lieux de ce qui intéresse les potentiels utilisateurs.

- Continuer à travailler sur l'aspect sécurité afin de protéger le site de backend.

7. Bibliographie

https://www.w3schools.com/bootstrap/bootstrap_modal.asp

https://legacy.reactjs.org/tutorial/tutorial.html

https://spring.io/guides/gs/spring-boot

https://developer.mozilla.org/fr/docs/Web/HTTP/CORS

Fait le 12/09/2024