Frontend

1. Contexte et présentation

1.1 Contexte GSB

Le laboratoire **GSB (Galaxy Swiss Bourdin)** est réputé pour la recherche, la production et la commercialisation de divers médicaments et produits de santé. Dans le cadre de ce projet **GestInv**, GSB souhaite mettre à disposition sa gamme de produits (médicaments, compléments, matériels, etc.) sur un site e-commerce destiné à différentes catégories d'utilisateurs :

- **Clients simples** (particuliers)
- **Pharmacies** et professionnels de santé

L'objectif est de gérer l'inventaire, passer des commandes, et proposer un espace d'administration pour la gestion des utilisateurs et des produits.

1.2 Description du Front-End

Le Front-End de **GestInv** est développé avec **React 18** et **TypeScript**, et utilise **Vite** comme outil de build. Il suit l'architecture suivante :

- **Pages** (dans src/page/) : Chaque route principale du site (Accueil, Login, AdminPanel, etc.).
- **Composants** (dans src/component/) : Composants réutilisables (formulaires, listings, layouts de produit, etc.).
- **Layout** (dans src/layout/) : Composants englobants (header commun, footer, etc.).
- **Utils** (dans src/utils/) : Fonctions utilitaires, notamment les appels API.
- **Router** (dans src/router.tsx) : Gestion des différentes routes (avec react-router-dom).
- **Styles** (dans src/index.css, tailwindcss, MUI pour les composants).

2. Stack technique

2.1 Langages, librairies et outils

1. **React 18** + **TypeScript**
    - Permet de bâtir une interface modulaire et robuste.
    - La typage TS améliore la fiabilité et la maintenabilité.
2. **Vite** (v5.1.6)
    - Outil de développement et de build rapide.
    - Configuration simple pour React + TypeScript.
3. **React Router DOM (v7.0.2)**
    - Gère la navigation entre les différentes pages (Home, Produits, Admin, etc.).
    - Support de la navigation déclarative via <Routes> et <Route>.
4. **Material-UI (MUI v6.2.1)**
    - Bibliothèque de composants UI prête à l'emploi.
    - Style customisable avec **@emotion/styled** et **@emotion/react**.
5. **TailwindCSS (v3.4.1)**
    - Un framework CSS utilitaire, complémentaire à MUI pour des ajustements rapides.
6. **JWT-Decode (v4.0.0)**
    - Pour décoder et manipuler les JSON Web Tokens (authentification).
7. **ESLint**, **Prettier**
    - Outils d'analyse statique et de formatage pour un code propre et cohérent.
8. **Jest**, **Testing Library**
    - Exécution de tests unitaires et de tests d'interface (composants).
    - Permet d'écrire des tests simples sur les composants React.

3. Organisation des fichiers

Arborescence simplifiée :

```
src
├── assets
│   └── react.svg
│
├── component
│   ├── Card.tsx
│   ├── Footer.tsx
│   ├── Header.tsx
│   ├── ProductForm.tsx
│   ├── ProductPanel.tsx
│   ├── ProductRow.tsx
│   ├── UserForm.tsx
│   ├── UserRow.tsx
│   └── UsersPanel.tsx
│
├── layout
│   └── mainLayout.tsx
│
├── page
│   ├── AdminPanel.tsx
│   ├── EspacePresse.tsx
│   ├── Home.tsx
│   ├── Login.tsx
│   ├── NosProduits.tsx
│   ├── NosProjets.tsx
```

```
|   ├── NousConnaitre.tsx
|   └── Panier.tsx
|
├── utils
|   └── api.ts
|
├── index.css
├── main.tsx
├── router.tsx
├── types.ts
└── vite-env.d.ts
```

3.1 Détails importants

- **main.tsx** : Point d'entrée du projet (montage de l'app React dans le DOM).
- **router.tsx** : Configuration du routing (react-router-dom).
- **types.ts** : Centralisation ou définition des types (interfaces TS) communs.
- **utils/api.ts** : Gestion des appels HTTP (fetch, axios, etc.), encapsule la logique d'authentification et la communication avec le back-end.
- **layout/mainLayout.tsx** : Mise en page globale du site (Header, Footer, etc.).

4. Installation et exécution

4.1 Prérequis

- **Node.js** >= 14
- **npm** ou **Yarn**

Vérifier la version de Node :

node -v

4.2 Installation des dépendances

1. Cloner le repo du front :

git clone https://github.com/votre-repo/gestinv-frontend.git

cd gestinv-frontend

2. Installer les dépendances :

npm install

ou

yarn install

4.3 Scripts disponibles

Dans le package.json :

- **dev** : Démarre l'application en mode développement.

  - npm run dev

build : Compile (TypeScript) puis génère un build optimisé

- npm run build

preview : Lance un serveur local pour visualiser le build

- npm run preview

lint : Analyse statiquement le code (ESLint)

- npm run lint

test : Lance la suite de tests Jest en mode watch

- npm run test

5. Gestion de l'authentification

Le site prévoit la possibilité de se connecter via un token JWT fourni par l'API back-end (le laboratoire GSB). Le token est :

- Récupéré lors du **Login**
- Stocké localement (par localStorage)
- Décodé via jwt-decode pour en extraire des informations (userId, rôles, etc.)

Exemple de logique :

1. **Login** :
    - Appel à api.ts via loginUser(credentials)
    - Récupération du token en réponse
    - Stockage du token dans localStorage
2. **AuthGuard (optionnel)** :
    - Vérifie la présence et la validité du token pour restreindre l'accès à certaines routes (ex. /admin).

6. Système de routing

La navigation est définie dans router.tsx :

```tsx
import ReactDOM from 'react-dom/client'
import { BrowserRouter, Routes, Route } from 'react-router-dom'
import MainLayout from './layout/mainLayout'
import Home from './page/Home'
import NosProduits from './page/NosProduits'
import NousConnaitre from './page/NousConnaitre'
import EspacePresse from './page/EspacePresse'
import AdminPanel from './page/AdminPanel'
import Login from './page/Login'
import Panier from './page/Panier'
import NosProjets from './page/NosProjets'
import ProfilePage from './page/Profil'

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route path="/" element={<MainLayout />}>
          <Route index element={<Home />} />
          <Route path="Profil" element={<ProfilePage />} />
          <Route path="NosProduits" element={<NosProduits />} />
          <Route path="NousConnaitre" element={<NousConnaitre />} />
          <Route path="EspacePresse" element={<EspacePresse />} />
          <Route path="AdminPanel" element={<AdminPanel />} />
          <Route path="Panier" element={<Panier />} />
          <Route path="NosProjets" element={<NosProjets />} />
        </Route>
      </Routes>
    </BrowserRouter>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root')!)
root.render(<App />)
```

7. Composants et mises en page

1. **Header.tsx, Footer.tsx**
    - Composants d'en-tête et de pied de page communs.
    - Insérés généralement dans le layout global mainLayout.tsx.
    - Par exemple, ProductForm gère la saisie des détails d'un produit (nom, prix, stock, etc.).

2. ProductForm.tsx, ProductPanel.tsx, ProductRow.tsx

- Composants dédiés à la gestion des produits (Affichage, Ajout, Édition).

3. UsersPanel.tsx, UserRow.tsx, UserForm.tsx

- Modules pour l'administration et la gestion des utilisateurs.

4. Card.tsx

- composant générique pour afficher une carte produit dans le panier.

8. Style et theming

8.1 Material UI (MUI)

- Le projet utilise la version 6.x de MUI.
- Personnalisation possible via ThemeProvider et **@emotion**.
-

8.2 TailwindCSS

- Configuration usuelle via un fichier tailwind.config.js et l'activation dans postcss.config.js.

---

9. Tests

9.1 Outils

- **Jest** + **Testing Library**
    - Permet d'écrire des tests unitaires, d'intégration, et des tests d'interface.
    - npm test -w frontend

10. Déploiement et build

1. **Build** :
    - Génére un dossier dist/ optimisé avec vite build.
2. **Déploiement** :
    - Déposer le contenu de dist/ sur un hébergeur statique (Netlify, GitHub Pages, etc.) ou l'injecter dans un serveur Node (Express).
3. **Configuration** :
    - Vérifier les variables d'environnement (via import.meta.env dans Vite).
    - S'assurer que l'URL de l'API Back-End (ex. VITE_API_URL) est correctement paramétrée.

11. Conclusion

Le projet GestInv (Front-End) propose une architecture claire basée sur React 18, TypeScript, Vite et MUI/Tailwind. Il est destiné à la plateforme e-commerce du laboratoire GSB, permettant de présenter et de commander des produits pharmaceutiques.

Backend

1. Contexte et présentation

1.1 Rappel : laboratoire GSB

Le projet **GestInv** est développé pour le laboratoire **GSB (Galaxy Swiss Bourdin)**, un groupe pharmaceutique . Le Back-End fournit une API REST permettant de :

- Gérer les utilisateurs (enregistrement, droits d'accès, rôles éventuels)
- Gérer l'authentification (JWT)
- Gérer les produits (consultation, création, mise à jour, suppression)
- Gérer les commandes (création, suivi)
- Le Front-End communique avec ce Back-End pour offrir un site e-commerce (mise à disposition de produits, ajout au panier, gestion de commandes, etc.).

1.2 Aperçu de la stack technique

- **TypeScript** : pour la robustesse du typage et la maintenabilité.
- **Node.js** : environnement d'exécution JavaScript côté serveur.
- **Express** (ou autre framework HTTP, selon la config) : gère les routes, middleware, etc.
- **JWT** : gestion de l'authentification et des autorisations.
- **Base de données** : connectée via le fichier config/db.ts

3. Structure de l'application

```
|  .env
|  .eslintrc.json
|  .prettierrc.json
|  jest.config.ts
|  nodemon.json
|  package.json
|  server.ts
|  tsconfig.json
|
├──Authent
|      auth.controller.ts
|      auth.middleware.ts
|      auth.model.ts
|      auth.route.ts
|      auth.service.ts
├──Commande
|      commande.controller.ts
|      commande.interfaces.ts
|      commande.route.ts
|      commande.service.ts
```

```
|
├──config
|      db.ts
|
├──Produits
|      produit.controller.ts
|      produit.interfaces.ts
|      produit.route.ts
|      produit.service.ts
|
└──Users
      users.controller.ts
      users.interfaces.ts
      users.route.ts
      users.service.ts
```

2.1 Fichiers racine

1. **.env** : Contient les variables d'environnement
2. **.eslintrc.json / .prettierrc.json** : Configuration d'ESLint et Prettier pour le linting et le formatage du code.
3. **jest.config.ts** : Configuration pour les tests Jest
4. **nodemon.json** : Configuration pour nodemon (redémarrage automatique du serveur en dev).
5. **package.json** : Liste des scripts et dépendances Node.
6. **server.ts** : Point d'entrée de l'application, instancie le serveur et monte les routes.
7. **tsconfig.json** : Configuration TypeScript

2.2 Dossiers principaux

2.2.1 Authent

Gère tout le cycle d'authentification et d'autorisation.

- **auth.controller.ts** : Logique de validation et de réponse HTTP pour la connexion, inscription, refresh token
- **auth.middleware.ts** : Middleware qui vérifie la présence et la validité du token JWT dans la requête (protège certaines routes).
- **auth.model.ts** : Définition du schéma de données.
- **auth.route.ts** : Définit les endpoints Express (e.g. /login, /register, /refresh).
- **auth.service.ts** : Logique métier (génération de token, hashing de mot de passe, etc.).

2.2.2 Commande

Gère la création, la lecture, la mise à jour et la suppression des commandes.

- **commande.controller.ts** : Contient les handlers (fonctions) qui traitent les requêtes liées aux commandes (ex: créer une commande, lister, etc.).
- **commande.interfaces.ts** : Définit les interfaces TypeScript pour les commandes (ex: ICommande, ICommandeItem).
- **commande.route.ts** : Fichier où les routes (URL) spécifiques aux commandes sont définies.
- **commande.service.ts** : Logique métier (calcul du total, vérification du stock, etc.).
-

2.2.3 Produits

Gère la partie Inventaire/Produits (médicaments, matériel, etc.).

- **produit.controller.ts** : Logique pour créer, afficher, modifier, supprimer un produit.
- **produit.interfaces.ts** : Interfaces TypeScript décrivant la structure du produit (IProduit).
- **produit.route.ts** : Déclaration des endpoints.
- **produit.service.ts** : Logique métier pour la gestion des produits (vérifier la disponibilité, mettre à jour le stock, etc.).

2.2.4 Users

Gère les données et actions liées aux utilisateurs.

- **users.controller.ts** : Fonctions de contrôle (afficher la liste des utilisateurs, éditer un profil).
- **users.interfaces.ts** : Interfaces TS .
- **users.route.ts** : Définit les routes associées (/users, /users/:id).
- **users.service.ts** : Règles de gestion (rôles, vérification droits d'accès, etc.).

2.2.5 config

- **db.ts** : Fichier de configuration et d'initialisation de la base de données.
    - MySQL

3. Installation et exécution

3.1 Prérequis

- **Node.js** >= 14
- **npm** ou **Yarn**
- Une base de données fonctionnelle configurée dans le .env.

3.2 Configuration .env

variables d'environnement à placer dans .env :

DB_PORT = 3307
DB_USER = «root
DB_PASSWORD = admin
DB_HOST = localhost
DB_Name = db_slam_ap

3.3 Installation des dépendances

Depuis la racine du projet Back-End :

- npm install

3.4 Scripts disponibles

Dans le package.json, vous pouvez retrouver des scripts de base, par exemple :

- **npm run dev** : Lance l'application en mode développement (avec nodemon), recharge automatique.
- **npm run build** : Compile TypeScript vers JavaScript (dossier dist/).
- **npm run start** : Démarre le serveur en production (en utilisant le code buildé).
- **npm run test** : Lance la suite de tests Jest.
- **npm run lint** : Lance ESLint sur le code.

4. Structure Express et montage des routes

4.1 server.ts

```
import express from 'express'
import cors from 'cors'
import authRoute from './Authent/auth.route'
import produitRoute from './Produits/produit.route'
import userRoute from './Users/users.route'
import commandeRoute from './Commande/commande.route'
import * as dotenv from 'dotenv'

dotenv.config()
import pharmacieRoute from './Pharmacie/pharmacie.route'

const app = express()
const port = 3000

app.use(cors({}))

app.use(express.json())
app.use('/api/commande', commandeRoute)
app.use('/api/produits', produitRoute)
app.use('/api/auth', authRoute)
app.use('/api/users', userRoute)
app.use('/api/pharmacie', pharmacieRoute)

app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`)
})
```

Ce fichier est le point d'entrée principal :

1. Initialise l'application Express.
2. Charge la config de la DB.
3. Monte les middlewares nécessaires.
4. Monte les routes dédiées à l'authentification, produits, etc.

5. Authentification JWT

5.1 Principes généraux

1. **Login** : L'utilisateur envoie ses identifiants (email, password).
2. **Contrôleur Auth** : Vérifie l'authenticité, génère un token JWT (via auth.service.ts).
3. **Réponse** : Renvoie le token JWT au Front-End.
4. **Middleware** : Les routes protégées exigent la présence du token dans le header (Authorization: Bearer <token>). Ce token est vérifié par auth.middleware.ts.

5.2 middleware

```ts
import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';

interface JwtPayload {
  id: number;
}

export const verifyToken = (req: any, res: Response, next: NextFunction): void => {
  const authHeader = req.headers.authorization;

  if (!authHeader) {
    res.status(401).json({ message: 'Token d\'acces manquant' });
    return;
  }

  const token = authHeader.split(' ')[1];

  if (!token) {
    res.status(401).json({ message: 'Pas de token' });
    return;
  }

  try {
    const decoded = jwt.verify(token, 'secretKey') as JwtPayload;
    req.user = decoded;
    next();
  } catch (error) {
    res.status(403).json({ message: 'token invalide ou déjà expiré' });
  }
};
```

};

6. Base de données (config/db.ts)

Ce fichier gère la connexion à la base de données.

```ts
import mysql from 'mysql2/promise'
import * as dotenv from 'dotenv'

dotenv.config()

const db = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  port: Number(process.env.DB_PORT),
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
})

export default db
```

7. Tests et qualité

7.1 Tests (Jest)

- **jest.config.ts** : Fichier de configuration.
- Tests d'intégration et de contrôleur possibles via des bibliothèques comme supertest.
- script : npm run test (lance Jest).

7.2 Lint et formatage

- **ESLint** : Vérifie la syntaxe, l'usage des variables, etc.
- **Prettier** : Applique le formatage défini dans .prettierrc.json.

8. Déploiement

8.1 Build

1. Compiler le TypeScript :

   - npm run build

   Génère le dossier dist/ contenant le code JavaScript transpilé.

9.Conclusion

Le Back-End de **GestInv** offre un socle pour gérer l'inventaire, les utilisateurs et les commandes liés au laboratoire **GSB**.

- L'API suit une organisation modulaire : **Authent**, **Commande**, **Produits**, **Users**, centralisant le code dans des controller, service, route, etc.
- L'authentification s'appuie sur un système JWT, protégé par un middleware dédié.
- La base de données est connectée via config/db.ts et peut être adaptée à différents SGBD.