# Documentation technique – Application Web

## Présentation de contexte :

L'application web GSB est une plateforme permettant aux utilisateurs d'acheter des produits pharmaceutiques en ligne. Elle est développée en monorepo pour une gestion simplifiée du frontend et du backend.

## Technologies utilisées :

### Front-End :

- React
- Typescript
- TailwindCSS
- Vite

### Back-End :

- Node.js
- Express

### Base de données :

- MySQL

### Outils :

- Postman
- MySQLWorkbench
- Jest

## Organisation du projet :

Le projet est organisé en monorepo et utilise npm workspaces pour gérer le frontend et le backend dans un même dépôt. Une seule commande permet de démarrer l'ensemble du projet.

### Structure du Monorepo :

```
root/
├── back/
├── front/
├── package.json (gestion des workspaces)
├── .gitignore
└── README.md
```

## Scripts globaux :

Le fichier package.json à la racine contient :

```
"scripts": {
  "back": "npm run dev -w back",
  "front": "npm run dev -w front",
  "dev": "concurrently \"npm run back\" \"npm run front\"",
  "test:back": "npm test -w back",
  "test:front": "npm test -w front"
}
```

# Front-End :

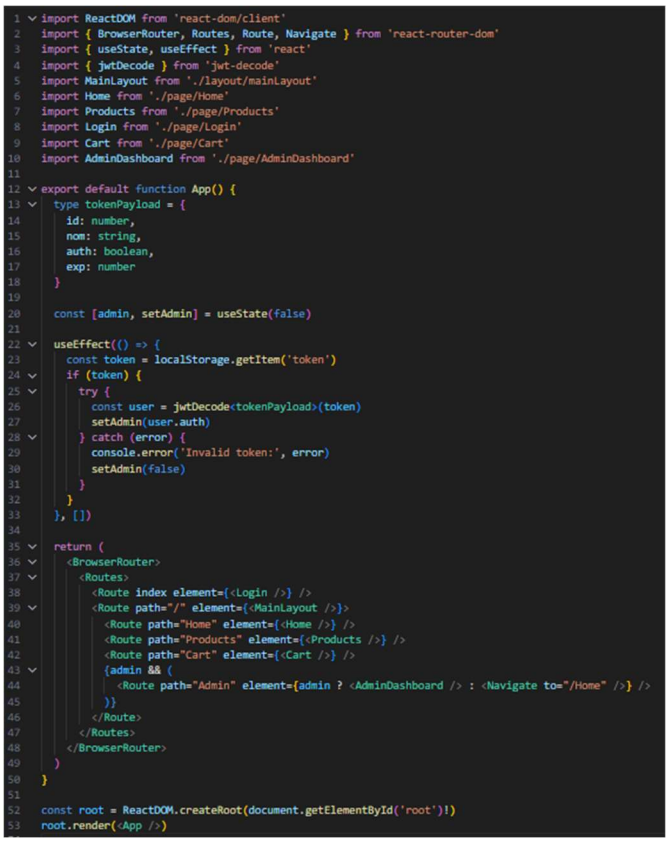
## Structure du Front-End :

- src/components/ : Composants réutilisables
- src/page/ : Pages principales (Accueil, Produits, Connexion, ...)
- src/interface/ :
- src/router.tsx : Gestion de la navigation avec react-router-dom
- src/utils/api.ts : Gestion centralisée des requêtes et des routes de l'api

## Gestion des styles :

Configuration TailwindCSS dans tailwind.config.js.

## Router principal :



```tsx
import ReactDOM from 'react-dom/client'
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'
import { useState, useEffect } from 'react'
import { jwtDecode } from 'jwt-decode'
import MainLayout from './layout/mainLayout'
import Home from './page/Home'
import Products from './page/Products'
import Login from './page/Login'
import Cart from './page/Cart'
import AdminDashboard from './page/AdminDashboard'

export default function App() {
  type tokenPayload = {
    id: number,
    nom: string,
    auth: boolean,
    exp: number
  }

  const [admin, setAdmin] = useState(false)

  useEffect(() => {
    const token = localStorage.getItem('token')
    if (token) {
      try {
        const user = jwtDecode<tokenPayload>(token)
        setAdmin(user.auth)
      } catch (error) {
        console.error('Invalid token:', error)
        setAdmin(false)
      }
    }
  }, [])

  return (
    <BrowserRouter>
      <Routes>
        <Route index element={<Login />} />
        <Route path="/" element={<MainLayout />}>
          <Route path="Home" element={<Home />} />
          <Route path="Products" element={<Products />} />
          <Route path="Cart" element={<Cart />} />
          {admin && (
            <Route path="Admin" element={admin ? <AdminDashboard /> : <Navigate to="/Home" />} />
          )}
        </Route>
      </Routes>
    </BrowserRouter>
  )
}

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

# Back-End :

## Structure du Back-end :

- server.js : point d'entrée du back-end.
- db.config.js : Configuration de la base de données.
- /routes : Dossier contenant les fichiers des routes (users, products, categories, ...).
- /controllers : Dossier contenant les fichiers des requêtes faites vers la base de données (user, product, category, ...).
- jsonwebtoken/check.js : Vérification sur une route protégée de la présence du token.

## Dépendance utilisée :

- Express : Gestion des routes
- JWT : Authentification et gestion des tokens
- Bcryptjs : Hash des mots de passe
- MySQL2 : base de données

## Exécution seul :

- Commande : npm run back dans le dossier root ou nodemon -r dotenv/config server.js dans le dossier back.

## API REST :

Chaque entité possède sa propre route, contrôleur et service, assurant une séparation claire des responsabilités.

## Middleware JsonWebToken :

```
import { compare } from 'bcrypt'
import pkg from 'jsonwebtoken'
import db from '../db.config.js'

const { sign } = pkg

export async function login(req, res) {
  const { mail, mdp } = req.body

  if (!mail || !mdp) {
    return res.status(400).json({ message: 'Il manque un paramètre' })
  }

  try {
    const req = await db.query('SELECT * FROM user WHERE mail = ?', [mail])
    let user = req[0][0]

    if (user.length === 0) {
      return res.status(404).json({ message: 'Utilisateur n\'existe pas' })
    }

    let match = await compare(mdp, user.mdp)

    if (!match) {
      return res.status(401).json({ message: 'Mauvais mot de passe' })
    }

    const token = sign({
      id: user.id_user,
      nom: user.nom,
      auth: user.admin
    }, process.env.JWT_SECRET, { expiresIn: process.env.JWT_DURING })

    return res.json({ access_token: token, admin: user.admin })
  } catch (err) {
    res.status(500).json({ message: 'Erreur lors du login', error: err })
  }
```

# Base de Données :

## Modèle Conceptuel de Données (MCD) :