Activité-type 2 **Développer la partie back-end d'une application web ou web mobile sécurisée**

CP 7 Développer des composants métier côté serveur

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Dans le projet Soundora, j'ai conçu et développé les composants métier du back-end en Node.js/Express.

J'ai implémenté la logique métier pour la gestion des utilisateurs, des produits, des catégories, du panier, des commandes et du paiement en ligne (Stripe).
J'ai structuré le code en contrôleurs (ex : orderController.js, cartController.js, authController,js) pour séparer la logique métier de l'accès aux données; voir exemple en partie 5.

Précision de la gestion des règles métier :

•J'ai mis en place des contrôleurs côté serveur (Node.js/Express) pour centraliser la logique métier de l'application.

•Pour chaque fonctionnalité (création de commande, ajout au panier, paiement, etc.), j'ai défini des règles précises :

•Vérification que l'utilisateur est bien authentifié avant d'accéder à certaines routes (ex : passer commande, voir l'historique).

•Validation des données reçues du frontend : contrôle des champs obligatoires, des formats (ex : email, prix positif, quantité non nulle), et gestion des erreurs en cas de données invalides.

•Calcul automatique du total du panier et des frais de livraison lors de la création d'une commande.

•Gestion des stocks : vérification de la disponibilité des produits avant validation d'une commande, et mise à jour des quantités en base après achat.

•Attribution de droits spécifiques selon le rôle de l'utilisateur (ex : seuls les admins peuvent ajouter ou supprimer des produits, seuls les clients peuvent commander).

•Intégration du paiement Stripe : la commande n'est validée que si le paiement est accepté, grâce à la gestion des webhooks Stripe.

•Suivi de l'état des commandes (en attente, payée, expédiée, annulée) et mise à jour automatique selon les actions de l'utilisateur ou du backoffice.

•J'ai documenté chaque règle métier dans le code et dans la documentation technique pour faciliter la compréhension et la maintenance du projet.

Conditions de réalisation :

Environnement technique :

Le projet a été développé sur un poste sous Linux, avec Node.js et Angular installés localement. Le code source est versionné sur GitHub, ce qui permet un suivi des évolutions et un travail collaboratif. L'éditeur principal utilisé est Visual Studio Code, avec des extensions facilitant le développement (lint, auto-format, intégration Git…).

•**Contraintes matérielles et logicielles** :

Le développement nécessite un ordinateur avec une connexion Internet, Node.js, npm, et un navigateur moderne pour le front-end. L'API back-end fonctionne en local ou via Docker, et la base de données est gérée via Supabase (PostgreSQL) ou MySQL selon les besoins de test.

•**Outils et ressources** :

•**Gestion de version** : Git et GitHub pour le suivi, la sauvegarde et la collaboration.

•**Gestion de projet** : Kanban (GitHub Projects), documentation dans le dossier docs.

•**Tests et validation** : Postman pour tester les API, console navigateur pour le front-end, logs serveur pour le back-end.

•**Sécurité** : Utilisation de variables d'environnement (.env), gestion des tokens JWT, et authentification via Supabase.

•**Déploiement** : Docker Compose pour simuler l'environnement de production, scripts npm pour automatiser les tâches courantes.

•**Organisation du travail** :

Le projet a été découpé en tâches : conception de la base de données, développement du back-end (API REST, logique métier), puis du front-end (UI responsive, intégration Stripe, gestion de l'authentification). Chaque étape a été validée par des tests manuels et des revues de code.

•**Contraintes spécifiques** :

•Respect des bonnes pratiques de sécurité (protection des routes, gestion des erreurs, validation des entrées).

•Accessibilité et responsive design pour garantir une expérience utilisateur optimale sur mobile et desktop.

•Documentation systématique du code et des choix techniques pour faciliter la maintenance et la transmission du projet.

2. Précisez les moyens utilisés :

•**Environnement technique** :

•Développement du back-end avec Node.js et Express pour la gestion des routes, de la logique métier et des contrôleurs.

•Utilisation de Supabase (PostgreSQL) comme base de données relationnelle, avec gestion des tables, des relations et des requêtes SQL.

•Intégration de l'API Stripe pour la gestion des paiements en ligne et des webhooks de confirmation.

•Utilisation de JWT (JSON Web Token) pour l'authentification sécurisée et la gestion des sessions utilisateurs.

•Mise en place de middlewares Express pour la validation des entrées, la gestion des droits d'accès et la sécurisation des routes sensibles.

•**Outils et pratiques de développement** :

•J'ai utilisé **Visual Studio Code** comme éditeur principal pour écrire et organiser le code du back-end.

•Le code a été versionné avec **Git** et hébergé sur **GitHub**, ce qui m'a permis de sauvegarder chaque étape du projet, de revenir en arrière si besoin, et de travailler proprement.

•J’ai utilisé **Postman** pour tester les différentes routes de l’API : cela m’a permis de vérifier que chaque fonctionnalité (connexion, ajout au panier, paiement, etc.) fonctionnait bien indépendamment du frontend.

•J’ai mis en place des **variables d’environnement** (fichier .env) pour sécuriser les informations sensibles comme les clés d’API ou les identifiants de la base de données.

•J’ai documenté le code avec des commentaires clairs pour expliquer la logique métier, les paramètres attendus et les retours des fonctions.

•J’ai structuré le projet en plusieurs fichiers et dossiers : contrôleurs pour la logique métier, routes pour les points d’entrée de l’API, services pour l’accès aux données, ce qui facilite la maintenance et l’évolution du projet.

•J’ai appliqué les bonnes pratiques de sécurité : validation des données reçues, gestion des erreurs, séparation des droits d’accès selon le rôle de l’utilisateur (admin, client…).

3. Avec qui avez-vous travaillé ?

Pour ce projet j’ai travaillé en total autonomie

4. Contexte

Nom de l’entreprise, organisme ou association ► *La Plateforme*

Chantier, atelier, service ► *Boutique en ligne d’instruments et d’accessoires de musique “Soundora”*

Période d’exercice ► Du : ***17/06/2025*** au : **31/08/2025**

5. Informations complémentaires (facultatif)

Voici ci dessous le code du fichier api.js représentant les différentes routes de l’application :

routes > JS api.js > ...

```
import express from "express"; // Importe Express pour créer le routeur
import * as authController from "../controllers/authController.js"; // Contrôleur d'authentification
import * as productSupabaseController from "../controllers/productSupabaseController.js"; //  NOUVEAU : Contrôleur produits Supabase (remplace productController)
import * as categoryController from "../controllers/categoryController.js"; // Contrôleur catégories
import * as brandController from "../controllers/brandController.js"; // Contrôleur marques
import * as cartController from "../controllers/cartController.js"; // Contrôleur panier
import * as orderController from "../controllers/orderController.js"; // Contrôleur commandes
import * as stripeController from "../controllers/stripeController.js"; // NOUVEAU : Contrôleur Stripe pour paiements
import * as testController from "../controllers/testController.js"; // Contrôleur de test
import checkJwt from "../middleware/checkJwt.js"; // Middleware JWT pour protéger les routes
import checkSupabaseAuth from "../middleware/checkSupabaseAuth.js"; // Middleware Supabase Auth

const router = express.Router(); // Crée un routeur Express

// ==================================
// ROUTES DE TEST
// Permettent de vérifier le bon fonctionnement de Supabase
// ==================================
router.get("/test/connection", testController.testConnection); // Test connexion Supabase
router.get("/test/tables", testController.listTables); // Liste des tables disponibles

// ==================================
// ROUTES D'AUTHENTIFICATION
// Gestion des utilisateurs avec Supabase Auth
// ==================================
router.post("/auth/register", authController.register); // Inscription d'un utilisateur
router.post("/auth/login", authController.login); // Connexion et obtention du token JWT
router.post("/auth/logout", authController.logout); // Déconnexion utilisateur
router.get("/auth/me", checkSupabaseAuth, authController.getCurrentUser); // Récupération de l'utilisateur actuel (protégé)

// ==================================
// ROUTES POUR LES PRODUITS - Version Supabase Avancée
// Remplace l'ancien productController par productSupabaseController
// Nouvelles fonctionnalités : pagination, filtres, recherche, slugs SEO
// ==================================

//  ROUTE PRINCIPALE : Liste des produits avec filtres et pagination
// Exemples d'utilisation :
// GET /api/products                                    → Tous les produits (page 1, 10 par page)
// GET /api/products?page=2&limit=20                    → Page 2, 20 produits par page
```

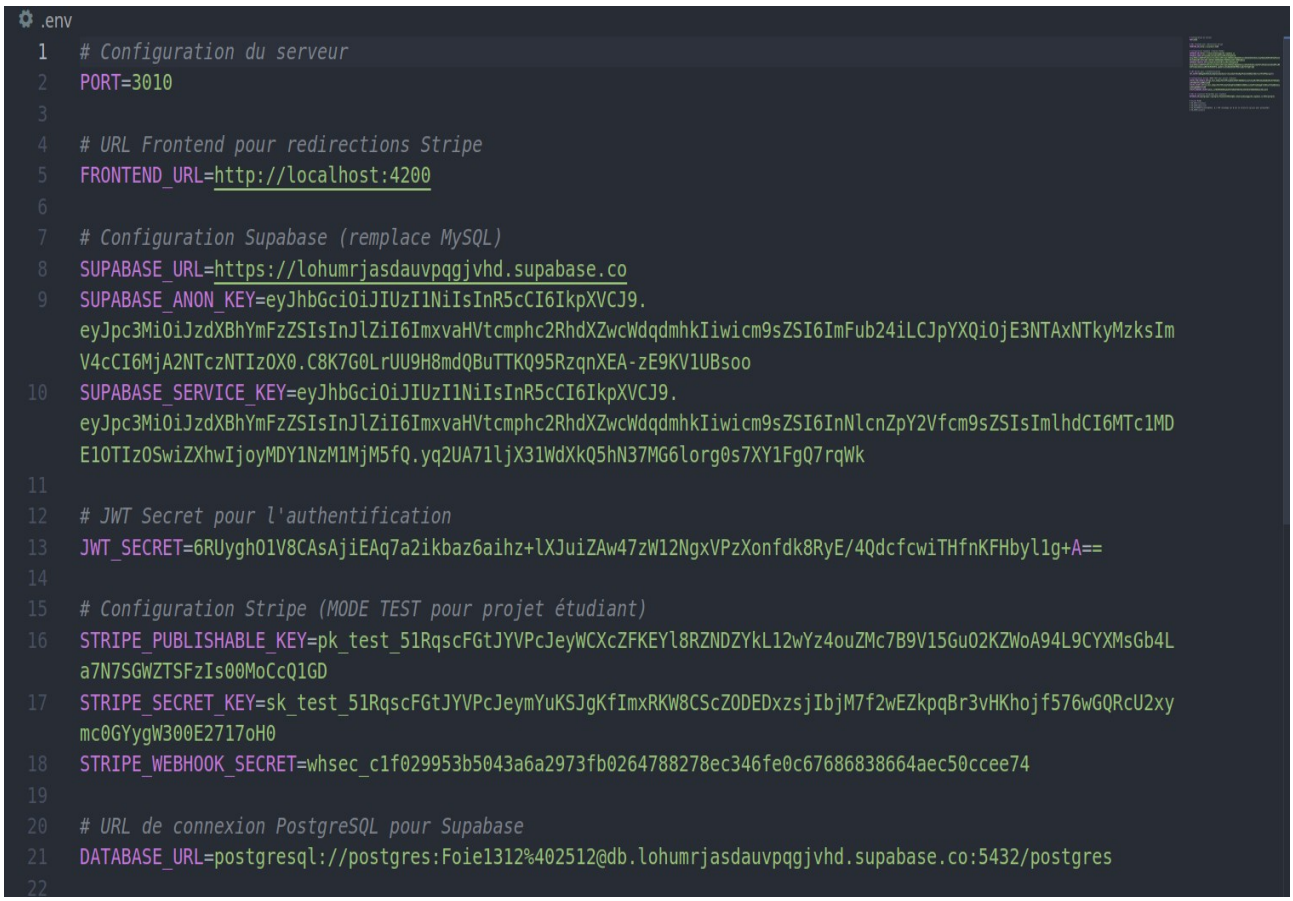

routes > JS api.js > ...

```
// GET /api/products?category=guitares                  → Seulement les guitares
// GET /api/products?brand=fender&min_price=500         → Marque Fender, prix min 500€
// GET /api/products?search=stratocaster                → Recherche "stratocaster"
router.get("/products", productSupabaseController.getAllProducts);

// ROUTE PRODUITS FEATURED : Produits mis en avant
// GET /api/products/featured?limit=6                   → 6 produits featured (défaut)
// GET /api/products/featured?limit=12                  → 12 produits featured
router.get("/products/featured", productSupabaseController.getFeaturedProducts);

// ROUTE RECHERCHE : Recherche textuelle rapide
// GET /api/products/search?q=guitare&limit=10          → Recherche "guitare", max 10 résultats
// GET /api/products/search?q=gibson                    → Recherche "gibson"
router.get("/products/search", productSupabaseController.searchProducts);

//  ROUTE PRODUIT INDIVIDUEL : Récupération par slug (SEO-friendly)
// GET /api/products/gibson-les-paul-standard-2024      → Produit via slug
// GET /api/products/fender-stratocaster-american       → Autre exemple
// IMPORTANT : Cette route doit être EN DERNIER pour éviter les conflits avec "featured" et "search"
router.get("/products/:slug", productSupabaseController.getProductBySlug);

// ====================================
// ROUTES POUR LES CATÉGORIES
// Gestion des catégories de produits (guitares, basses, etc.)
// ====================================
router.get("/categories", categoryController.getAllCategories); // Liste toutes les catégories
router.get("/categories/:id", categoryController.getCategoryById); // Récupère une catégorie spécifique
router.post("/categories", checkJwt, categoryController.createCategory); // Création d'une catégorie (protégé)
router.put("/categories/:id", checkJwt, categoryController.updateCategory); // Mise à jour (protégé)
router.delete("/categories/:id", checkJwt, categoryController.deleteCategory); // Suppression (protégé)

// ====================================
// ROUTES POUR LES MARQUES
// Gestion des marques de produits (Fender, Gibson, etc.)
// ====================================
router.get("/brands", brandController.getAllBrands); // Liste toutes les marques
router.get("/brands/:id", brandController.getBrandById); // Récupère une marque par ID
router.get("/brands/slug/:slug", brandController.getBrandBySlug); // Récupère une marque par slug
```

routes > JS api.js > ...

```
// ====================================
router.get("/cart", checkJwt, cartController.getCart); // Récupère le panier de l'utilisateur (protégé)
router.post("/cart/items", checkJwt, cartController.addToCart); // Ajoute un article au panier (protégé)
router.put("/cart/items/:id", checkJwt, cartController.updateCartItem); // Met à jour un article du panier (protégé)
router.delete("/cart/items/:id", checkJwt, cartController.removeFromCart); // Supprime un article du panier (protégé)
router.get("/cart/count", checkJwt, cartController.getCartCount); // Récupère le nombre d'articles dans le panier (protégé)

// ====================================
//  ROUTES POUR LES COMMANDES
// Gestion des commandes et historique d'achat
// ====================================
router.post("/orders", checkJwt, orderController.createOrder); // Création d'une commande (protégé)
router.get("/orders", checkJwt, orderController.getUserOrders); // Récupère les commandes de l'utilisateur (protégé)
router.get("/orders/:order_id", checkJwt, orderController.getOrderDetails); // Récupère une commande spécifique (protégé)

// ====================================
// ROUTES STRIPE PAIEMENT
// Gestion des paiements via Stripe Checkout
// ====================================
router.post("/stripe/create-checkout-session", checkSupabaseAuth, stripeController.createCheckoutSession); // Créer session Stripe (protégé)
router.get("/stripe/session-status/:sessionId", stripeController.getSessionStatus); // Vérifier statut session

// ====================================
// ROUTES STRIPE TEST (POUR DÉVELOPPEMENT) - ACTIVÉES POUR TESTS
// Routes simples pour tester Stripe sans authentification
// ====================================
router.post("/stripe/test-simple", stripeController.createTestSessionSimple); // Test session 10€ (non protégé)
router.post("/stripe/test-complete", stripeController.createTestSessionComplete); // Test session 99€ (non protégé)

// ====================================
// WEBHOOK STRIPE (SANS AUTHENTIFICATION)
// Stripe appelle cette route pour confirmer les paiements
// IMPORTANT: Cette route ne doit PAS avoir de middleware d'auth
// Le middleware raw est déjà appliqué dans server.js
// ====================================
router.post("/stripe/webhook", stripeController.stripeWebhook);

// === EXPORT DU ROUTEUR POUR L'UTILISER DANS server.js ===
export default router;
```

Et voici le fichier .env pour la sécurité des variables d'environnement :



.env

```
# Configuration du serveur
PORT=3010

# URL Frontend pour redirections Stripe
FRONTEND_URL=http://localhost:4200

# Configuration Supabase (remplace MySQL)
SUPABASE_URL=https://lohumrjasdauvpqgjvhd.supabase.co
SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImxvaHVtcmphc2RhdXZwcWdqdmhkIiwicm9sZSI6ImFub24iLCJpYXQiOjE3NTAxNTkyMzksIm
V4cCI6MjA2NTczNTIzOX0.C8K7G0LrUU9H8mdQBuTTKQ95RzqnXEA-zE9KV1UBsoo
SUPABASE_SERVICE_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImxvaHVtcmphc2RhdXZwcWdqdmhkIiwicm9sZSI6InNlcnZpY2Vfcm9sZSIsImlhdCI6MTc1MD
E1OTIzOSwiZXhwIjoyMDY1NzM1MjM5fQ.yq2UA71ljX31WdXkQ5hN37MG6lorg0s7XY1FgQ7rqWk

# JWT Secret pour l'authentification
JWT_SECRET=6RUyghO1V8CAsAjiEAq7a2ikbaz6aihz+lXJuiZAw47zW12NgxVPzXonfdk8RyE/4QdcfcwiTHfnKFHbyl1g+A==

# Configuration Stripe (MODE TEST pour projet étudiant)
STRIPE_PUBLISHABLE_KEY=pk_test_51RqscFGtJYVPcJeyWCXcZFKEYl8RZNDZYkL12wYz4ouZMc7B9V15GuO2KZWoA94L9CYXMsGb4L
a7N7SGWZTSFzIs00MoCcQ1GD
STRIPE_SECRET_KEY=sk_test_51RqscFGtJYVPcJeymYuKSJgKfImxRKW8CScZODEDxzsjIbjM7f2wEZkpqBr3vHKhojf576wGQRcU2xy
mc0GYygW300E2717oH0
STRIPE_WEBHOOK_SECRET=whsec_c1f029953b5043a6a2973fb0264788278ec346fe0c67686838664aec50ccee74

# URL de connexion PostgreSQL pour Supabase
DATABASE_URL=postgresql://postgres:Foie1312%402512@db.lohumrjasdauvpqgjvhd.supabase.co:5432/postgres
```