**PLAN DE TRAVAILLE SUR LE PROJET ANNULE**

**Étape 1 : Planifier l'architecture de l'application**

Avant de coder, il faut définir l'architecture de l'application. Voici ce que ce que je vous propose :

1. **Choix des technologies** :
   * **Front-end** : HTML, CSS, JavaScript (avec un framework comme React, Angular, ou Vue.js si nécessaire).
   * **Back-end** : PHP (comme suggéré dans le projet) ou Node.js si vous préférez JavaScript côté serveur.
   * **Base de données** : MySQL ou PostgreSQL pour stocker les données (clients, salariés, prestataires, contrats, etc.).
   * **API** : Vous pouvez utiliser un framework comme Laravel (PHP) ou Express (Node.js) pour créer l'API.
   * **Paiements** : Intégration de Stripe pour gérer les transactions.
   * **Notifications** : Utilisation de OneSignal pour les notifications push.

ETAPE 2 :

**1. Mission 1 : Gestion de la société (Développement)**

* **Objectif** : Moderniser et informatiser le système de gestion de Business Care.
* **Tâches principales** :
  + Développer une application web pour gérer les clients, les salariés et les prestataires.
  + Créer un backoffice pour l'administration générale.
  + Intégrer des fonctionnalités comme la gestion des contrats, des factures, des événements, des rendez-vous, etc.
  + Utiliser des technologies comme JavaScript, PHP, et des frameworks pour le développement.
  + Mettre en place une API pour gérer les traitements et utiliser des services cloud.
  + Configurer un serveur web pour héberger le site.

**2. Mission 2 : Services supplémentaires (Développement)**

* **Objectif** : Mettre en place des outils supplémentaires pour améliorer la gestion et l'expérience utilisateur.
* **Tâches principales** :
  + Développer une application autonome en Java pour générer des rapports graphiques sur l'état de l'activité de Business Care.
  + Créer une application Android pour les salariés, permettant d'accéder à leurs événements et plannings.
  + Intégrer des fonctionnalités comme la génération de PDF avec des diagrammes statistiques.
  + Gérer des cartes NFC pour l'accès aux événements.

**3. Mission 3 : Infrastructure Système, Réseau et Sécurité**

* **Objectif** : Mettre en place une infrastructure réseau robuste, sécurisée et hautement disponible pour l'entreprise.
* **Tâches principales** :
  + Configurer les serveurs (Active Directory, stockage, DNS, DHCP, etc.) pour les différents sites (Paris, Troyes, Biarritz, Nice).
  + Mettre en place des pare-feux (OPNSense) et des VPN pour sécuriser les connexions entre les sites.
  + Configurer des outils de supervision et de ticketing pour la gestion du réseau.
  + Assurer la redondance et la haute disponibilité des services.
  + Sécuriser l'accès au site web via HTTPS (SSL/TLS).
  + Mettre en place des règles de filtrage strictes pour les flux réseau.

**POUR LES FICHIERS STRIPE .PHP ET DB.PHP**

**✅ Avantages de db.php (Gestion de la base de données)**

1. **Réutilisabilité** 🔄
   * Centraliser la connexion à la base de données permet d’éviter de répéter le même code dans plusieurs fichiers.
   * Si les informations de connexion changent (ex: changement de serveur), il suffit de modifier **un seul fichier**.
2. **Facilité de maintenance** 🛠
   * Si on veux améliorer ou modifier la connexion à la base, on n’aurra qu’un seul fichier à modifier, ce qui simplifie la gestion.
3. **Sécurité améliorée** 🔐
   * En utilisant **require\_once 'db.php'**, on évites d’avoir tes identifiants de connexion dispersés dans plusieurs fichiers.
   * On peux aussi ajouter des sécurités comme **PDO avec des requêtes préparées** pour éviter les injections SQL.
4. **Structure et organisation propre** 📂
   * le projet est bien structuré : tous les fichiers qui se connectent à la base passent par **db.php**, ce qui rend le code plus lisible et logique.

**✅ Avantages de stripe.php (Gestion des paiements)**

1. **Séparation des responsabilités** 🎯
   * Le fichier **stripe.php** se concentre uniquement sur la gestion des paiements avec Stripe, ce qui rend le code plus **modulaire** et facile à gérer.
2. **Réutilisabilité** 🔄
   * Si on veux ajouter Stripe sur d’autres pages (ex: une page de facturation, un abonnement, etc.), il suffit d’appeler **require\_once 'stripe.php'**.
3. **Sécurité et gestion des erreurs** 🔐
   * En ayant un fichier dédié, on peux gérer les erreurs Stripe de manière centralisée, sans affecter le reste du projet.
   * On peux y intégrer des **logs** pour garder une trace des paiements.
4. **Évolutivité** 🚀
   * Si plus tard on veux **ajouter de nouvelles fonctionnalités Stripe** (remboursements, abonnements, webhooks, etc.), on modifies **uniquement** stripe.php sans toucher aux autres parties du projet.

**🌟 En résumé**

En séparant **db.php** et **stripe.php**, on rends le projet :  
✅ **Plus organisé**  
✅ **Facile à maintenir**  
✅ **Sécurisé**  
✅ **Évolutif**

📂 **Répartition claire des responsabilités**

* **API/** → Fichiers qui exposent les routes et gèrent les requêtes du Front-End.
* **CONTROLLERS/** → Gère la logique métier (validation, traitements).
* **MODELS/** → Interaction avec la base de données.
* **config.php** → Connexion et configuration de la base de données.
* **index.php** → Point d’entrée du Back-End, peut être utilisé pour router les requêtes.
* **stripe.php** → Gestion des paiements Stripe.

J’ai séparé **les différentes couches** du projet, ce qui rend le code **plus propre, maintenable et évolutif**. ✅

Le fichier **stripe.php** doit contenir donc :

1. Créer des paiements uniques et des abonnements Stripe
2. Confirmer le paiement et stocker l’information en base de données
3. Générer une facture PDF après paiement
4. Gérer les webhooks Stripe pour suivre l’état des paiements

Il m’a fallut créer un compte stripe avec se lien <https://dashboard.stripe.com/test/dashboard> pour obtenir les clés API Stripe qui vont me permettre de configurer le système de paiement sur l'application et aussi pour autres raisons comme :

1. Intégration des paiements : Utiliser les clés API dans votre code pour configurer les paiements par carte bancaire, gérer les abonnements, les paiements uniques, et bien plus.
2. Traitement des paiements : Cela vous permet de créer des tokens pour les informations de carte bancaire des clients, qui sont ensuite envoyées à Stripe pour effectuer des paiements sécurisés.
3. Gestion des paiements : Vous pourrez consulter le statut des paiements, les transactions effectuées, et gérer les remboursements via le tableau de bord Stripe.
4. Sécurisation des données : Stripe prend en charge la sécurité des données sensibles (comme les informations de carte bancaire) et vous permet de gérer les paiements sans avoir à stocker ces informations vous-même.