

Dossier Projet – Application de gestion de cours particuliers

1. Introduction

1.1 Contexte du projet

Dans le cadre d'une demande réelle émanant d'un professeur particulier, le projet vise à développer une application web permettant de gérer de manière centralisée les cours particuliers, de la réservation au paiement en ligne. Actuellement, la gestion des cours, des inscriptions et des paiements se fait de manière dispersée (échanges par téléphone, messages, virements manuels), ce qui entraîne des pertes de temps, un risque d'erreur et une expérience utilisateur peu fluide. L'objectif est donc de concevoir un outil simple et intuitif permettant :

. Aux élèves de s'inscrire, réserver un créneau disponible, effectuer un paiement sécurisé et consulter l'historique de leurs cours.

. Au professeur, également administrateur de la plateforme, de gérer son planning, ses tarifs, les inscriptions et la communication avec ses élèves à partir d'un espace unique.

Ce projet s'inscrit dans une démarche de digitalisation des services éducatifs, en offrant un gain de temps, une meilleure traçabilité et une expérience utilisateur moderne.

1.2 Objectifs généraux

- Permettre aux élèves de réserver et payer leurs cours en ligne.
- Centraliser la gestion des plannings pour le professeur.
- Faciliter la communication entre professeur et élèves.
- Permettre aux élèves de télécharger leurs factures à la demande

1.3 Public cible

- **Professeur/Administrateur** : gestion complète des cours, des paiements et des élèves.
- **Élèves** : inscription, réservation, paiement, suivi des cours.

1.4 Problématique

La gestion actuelle des cours particuliers repose sur des échanges éparés par téléphone, SMS ou e-mail, ainsi que sur des paiements manuels par virement ou espèces. Ce fonctionnement présente plusieurs inconvénients :

- Manque de visibilité sur les disponibilités du professeur.
- Risque de double réservation ou d'oubli d'un cours.
- Suivi administratif et financier chronophage.
- Communication dispersée entre différents canaux.

Ces contraintes nuisent à la fluidité de l'organisation, augmentent le risque d'erreurs et peuvent impacter la satisfaction des élèves. Il est donc nécessaire de mettre en place une application centralisée, offrant une vision claire du planning, une réservation simple, un paiement sécurisé et un historique consultable à tout moment, afin d'optimiser le temps du professeur et de faciliter la vie des élèves.

2. Présentation générale

2.1 Description fonctionnelle

L'application se présente sous la forme d'un site web ergonomique et intuitif, composé d'une page d'accueil (landing page), d'un profil public du professeur, et d'un espace utilisateur sécurisé appelé dashboard, qui regroupe l'ensemble des fonctionnalités interactives.

Page d'accueil (Landing page)

La page d'accueil, accessible à tous, présente brièvement le principe de fonctionnement de l'application et ses avantages. Elle intègre un menu de navigation permettant :

- D'accéder au profil public du professeur.
- De se connecter ou de s'inscrire pour accéder au dashboard (partie privée de l'application).

Espace utilisateur (Dashboard)

Une fois connecté, l'élève accède au tableau de bord. Par défaut, le menu principal est affiché à gauche et permet de naviguer entre les différentes sections.

1. Gestion des informations personnelles.
2. Historique des réservations.
3. Communication avec le professeur.
4. Calendrier interactif
5. Commandes et paiement

Le calendrier interactif

Pour les élèves, ce calendrier permet de consulter les créneaux disponibles du professeur, avec indication des prix et promotions éventuelles, d'afficher le planning sous différentes vues : jour, semaine ou mois et de sélectionner un ou plusieurs créneaux pour pré-réservation.

Lorsqu'un créneau est choisi, il est réservé temporairement (15 minutes) le temps de finaliser le paiement. Pendant cette période, il n'apparaît plus comme disponible pour les autres utilisateurs.

Pour le professeur, ce calendrier est le cœur de l'application, il permet de visualiser les créneaux, réserves, libres ou en cours de réservation, mais également, il permet l'ajout des nouveaux créneaux ou l'édition des créneaux libres. tel que le changement des prix, l'ajout de promotion ou la suppression.

Historique et réservations

Un onglet dédié permet de consulter :

- Les réservations à venir.
- Les réservations passées.

Profil élève

L'onglet Profil regroupe :

- Les informations personnelles (nom, prénom, coordonnées...).
- Les informations de formation (utiles au professeur pour préparer les cours).
- Les adresses (facturation, domicile...).
- Une description personnelle libre.
- Les liens vers les réseaux sociaux (LinkedIn, GitHub...).

Ces informations sont visibles par le professeur afin d'adapter son enseignement.

Contact

L'onglet Contact permet d'envoyer un message directement au professeur pour :

- Demander un remboursement.
- Poser une question.
- Obtenir des renseignements divers.

Notifications

La page par défaut du dashboard est la page Notifications, affichant :

- Toutes les notifications par ordre chronologique.
- Un système de filtrage (par notifications vues / non vues).
- Un résumé synthétique de l'activité de la semaine.

Des extras information sont affihces pour le professeur

Onglet Utilisateurs (professeur)

Le professeur, également administrateur, dispose de fonctionnalités supplémentaires :

Onglet Utilisateurs qui permet de lister tous les élèves, rechercher un profil, consulter leurs informations.

Personnalisation visuelle

En bas de l'application, il propose un mode sombre et un mode clair, que l'utilisateur peut sélectionner selon ses préférences. Le choix est conservé tant qu'il n'est pas modifié.

2.2 Cas d'usage principaux

- Réserver un cours.
- Payer en ligne.
- Consulter l'historique.
- Envoyer un message au professeur.

2.3 Acteurs et rôles

Acteur	Rôle principal
Professeur	Administrer et donner les cours
Élève	Réserver et suivre les cours

3. Fonctionnalités détaillées

- 1. Inscription et authentification.
- 2. Réservation de créneaux disponibles.
- 3. Paiement sécurisé.
- 4. Consultation de l'historique.
- 5. Messagerie.
- 6. Gestion des tarifs et disponibilités.
- 7. Facturation

1. Inscription et authentification

L'inscription constitue le point d'entrée de l'application et s'articule autour d'un processus en deux étapes. Lors de l'inscription, l'utilisateur renseigne ses informations personnelles (nom, prénom, email, mot de passe) via un formulaire sécurisé avec validation en temps réel. Le système vérifie la robustesse du mot de passe (8 caractères minimum, combinaison de majuscules, minuscules, chiffres et caractères spéciaux) et l'unicité de l'adresse email. Deux consentements distincts sont requis : l'acceptation de la politique de confidentialité et l'autorisation de traitement des données personnelles conformément au RGPD. Une fois l'inscription validée, l'utilisateur reçoit un email de confirmation pour activer son compte.

L'authentification repose sur un système dual optimisant sécurité et expérience utilisateur. La connexion initiale génère un JWT court (30 minutes) stocké en mémoire et un refresh token long (7 jours) stocké dans un cookie sécurisé. Lors des visites ultérieures, un mécanisme automatique utilise le refresh token pour régénérer transparentement les credentials, évitant à l'utilisateur de se reconnecter manuellement. Cette approche protège contre les attaques XSS tout en maintenant une session persistante et fluide.


Plusde details sont detaillees dans la section Securite'

2. Réservation de créneaux disponibles

Le système de réservation s'appuie sur un calendrier interactif FullCalendar offrant trois vues (jour, semaine, mois) pour optimiser la visualisation selon les préférences utilisateur. Les créneaux disponibles apparaissent en temps réel avec leurs tarifs respectifs et d'éventuelles promotions. L'élève sélectionne un ou plusieurs créneaux consécutifs, déclenchant une pré-réservation temporaire de 15 minutes. Durant cette période critique, les créneaux choisis disparaissent de la disponibilité publique, évitant les conflits de réservation.


Le processus intègre une validation intelligente empêchant les réservations en double, les créneaux passés ou les chevauchements. Si le paiement n'est pas finalisé dans le délai imparti, les créneaux redeviennent automatiquement disponibles et une notification de libération est diffusée. Cette mécanique garantit une gestion optimale des disponibilités sans blocages inutiles. Un changement implique une annulation

immédiate du checkout de paiement.



- Tableau de bord
- Réservations
- Calendrier
- Mes Commandes
- Profil
- Contact**

Contacter l'administrateur



Sujet

Proposition

Message

Bonjour,
Est-il possible de ...

☒ Avoir une copie

Envoyer

- Paramètres
- Déconnexion

3. Paiement sécurisé

L'intégration Stripe assure un processus de paiement garantissant la sécurité maximale des données bancaires. L'interface de paiement s'adapte automatiquement au montant total (créneaux + promotions/réductions), affiche un récapitulatif détaillé et propose les principales méthodes de paiement européennes.

3.1 Deroulement

une fois la commande est prête, le client dispose de 15 minutes pour régler la commande, un compte à rebours est placé pour indiquer le temps restant. En cliquant sur payer, la redirection se fait automatiquement après avoir créé un checkout de paiement côté serveur de validité de 15 minutes. Une fois le délai est passé, le checkout sera annulé automatiquement. L'interface de paiement Stripe, détaille les articles à payer ainsi que le montant total, à la fin de paiement et si le paiement est abouti, le client sera redirigé vers la page de validation de paiement. Stripe indique le non succès du paiement dans le cas contraire.

3.2 précaution et sécurité

Les créneaux sont des articles limités de nombre limités avec des contraintes de temps, un créneau réservé est unique et doit être payé le plus vite possible, sinon ça bloquera le créneau pour les élèves qui en ont besoin. C'est pour cela la contrainte de paiement est imposée. Côté serveur le checkout déclenche un background-service qui annulera la réservation si aucun checkout n'est créé dans le délai de 15 minutes. Le checkout de paiement déclenche à son tour un autre service de fonctionnement similaire et qui sera annulé si le paiement est accepté ou si le délai est terminé.

J'ai pris des précautions supplémentaires si une fois le checkout créé, l'élève modifie sa commande dans un autre onglet ou sur un autre appareil, chaque modification de commande annule automatiquement le checkout.

de paiement et le paiement sera refusé automatiquement.

```
public async Task<bool> BookSlot(BookingCreateDTO newBookingCreateDTO,
UserApp booker)
{
    ...
    Order order = await
orderService.GetOrCreateCurrentOrderByUserAsync(booker);
    ...
    if (order.CheckoutID is not null)
    {
        try
        {
            await jobChron.ExpireCheckout(order.CheckoutID);
            order.ResetCheckout();
        }
        ...
    }
}
```

Lors de la réservation, on vérifie si un checkout est déjà en cours, on l'annule

```
public async Task ExpireCheckout(string checkoutId)
{
    try
    {
        ...
        StripeConfiguration.ApiKey =
EnvironmentVariables.STRIPE_SECRET_KEY;
        var service = new Stripe.Checkout.SessionService();
        Stripe.Checkout.Session session = service.Expire(checkoutId);
    }
    ...
}
```

cote serveur, j'ai mis en place un webhook responsable uniquement de l'écoute de stripe et qui met à jour les commandes et réservations en fonction de l'aboutissement de paiement.

```
public async Task<bool> CheckPaymentAndUpdateOrder(...)
{
    ...
    // si paiement termine'
    if (stripeEvent.Type == "checkout.session.completed")
    {
        var session = stripeEvent.Data.Object as Session;
        // si paiement accepté'
        if (session.PaymentStatus == "paid")
        {
            ...
        }
    }
}
```

```
        if (orderId is not null && session.PaymentIntentId is not
null)
        {
            // notifications
            ...
            // annulation du service de nettoyage

jobChron.CancelScheuledJob(newOrder.Id.ToString());
            // mettre a jour la commande, la marquer comme
paye'

            return await orderService.UpdateOrderStatus(
                orderGuid,
                EnumBookingStatus.Paid,
                session.PaymentIntentId
            );
        }
    }
}
```

4. Consultation de l'historique

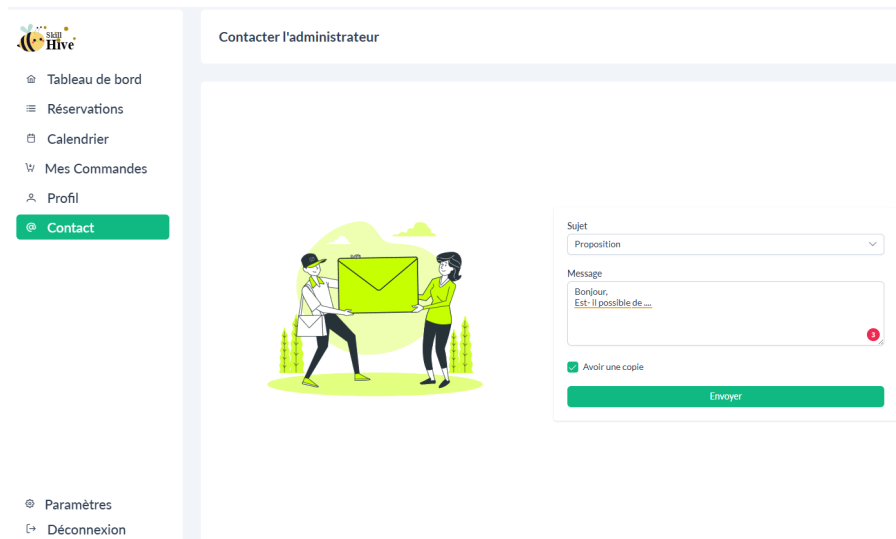
Les réservations sont classées par statut (à venir ou passées) avec pagination. Chaque entrée affiche les détails complets : date, heure, durée, prix payé... En cliquant sur une reservation, un modal détaillant la reservation sera affiche', et qui permettent également d avoir un suivi de la seession de cours via une mini interface de communication de type chat.

Les commandes sont également classees par ordre chronologique descendant, et qui permettent a l utilisateur de telecharger sa facture.

La fonctionnalité permet le téléchargement groupé de factures par période, l'export des données au format CSV pour la comptabilité personnelle, et l'accès aux communications échangées avec le professeur pour chaque réservation. Un système de recherche textuelle facilite la localisation rapide d'un cours spécifique. Les statistiques personnelles (nombre d'heures de cours, montant total dépensé, fréquence de réservation) enrichissent la vue d'ensemble de l'activité éducative.

5. Messagerie

La messagerie intégrée facilite la communication directe entre élève et professeur via Trevo. Il s agit d une simple interface de communication pour les demandes types (remboursement, report, information pédagogique).



Plusieurs possibilités de titres et la possibilité d'avoir une copie dans sa propre boîte mail

6. Gestion des tarifs et disponibilités

Cette fonctionnalité, exclusive au professeur, constitue le cœur opérationnel de l'application. Via le calendrier, le professeur peut créer, supprimer ou éditer les créneaux. Il peut également ajouter des promotions pour promouvoir des créneaux spécifiques.

7. Facturation

Le système de facturation automatisé génère des documents conformes aux obligations légales françaises (numérotation séquentielle et TVA). La création des factures se fait à la demande, en PDF via PuppeteerSharp, avec template professionnel personnalisable incluant les coordonnées de l'auto-entreprise.

4. Architecture technique

4.1 Technologies utilisées

Frontend : Angular 19

Angular 19 représente la dernière version du framework développé par Google, offrant une approche moderne et robuste pour le développement d'applications web. Le choix d'Angular se justifie par plusieurs avantages significatifs par rapport aux autres frameworks :

- **Architecture structurée** : Angular impose une architecture claire basée sur les composants, services et modules, facilitant la maintenance et l'évolutivité du code
- **TypeScript natif** : L'intégration native de TypeScript offre une meilleure détection d'erreurs à la compilation et améliore la productivité des développeurs
- **Écosystème complet** : Angular CLI, Angular Material, et un ensemble d'outils intégrés accélèrent le développement
- **Performance optimisée** : Le système de détection des changements et la compilation AOT (Ahead-of-Time) garantissent des performances élevées
- **Support à long terme** : Google assure un support LTS (Long Term Support) offrant une stabilité pour les projets d'entreprise

Librairies Frontend utilisées

L'application s'appuie sur un ensemble de librairies spécialisées pour offrir une expérience utilisateur riche :

PrimeNG (v19.0.5) : Cette suite de composants UI pour Angular fournit plus de 100 composants prêts à l'emploi (calendriers, tableaux, formulaires, modales). PrimeNG a été choisi pour sa compatibilité native avec Angular, sa documentation exhaustive et ses thèmes personnalisables qui s'intègrent parfaitement avec notre design system.

Tailwind CSS (v3.4.17) avec tailwindcss-primeui : Framework CSS utility-first qui permet un développement rapide et une personnalisation fine. L'intégration avec PrimeUI assure une cohérence visuelle entre les composants custom et ceux de PrimeNG.

FullCalendar (v6.1.15) : Bibliothèque spécialisée dans l'affichage de calendriers interactifs, essentielle pour la gestion des créneaux de cours. Elle offre des vues multiples (jour, semaine, mois) et une intégration native avec Angular.

openapi-typescript-codegen : Outil automatisant la génération du code TypeScript client à partir de la spécification OpenAPI du backend. Cette approche garantit une synchronisation parfaite entre l'API et le frontend, élimine les erreurs de typage et accélère le développement en générant automatiquement les services, modèles et types TypeScript correspondant aux endpoints de l'API .NET.

Backend : .NET 8.0

.NET 8.0 constitue la plateforme backend, offrant performance, sécurité et maintenabilité. Ses avantages incluent :

- **Performance native** : Compilation native et optimisations avancées
- **Écosystème riche** : Vaste bibliothèque de packages NuGet
- **Sécurité intégrée** : Fonctionnalités de sécurité built-in et conformité aux standards
- **Interopérabilité** : Support multi-plateforme (Windows, Linux, macOS)
- **Support Microsoft** : Maintenance et évolutions assurées par Microsoft

Librairies Backend principales

Entity Framework Core avec Npgsql.EntityFrameworkCore.PostgreSQL (v8.0.10) : ORM moderne permettant l'interaction avec PostgreSQL via des objets .NET, avec support des migrations automatiques et optimisations de requêtes.

ASP.NET Core Identity (v8.0.10) : Framework d'authentification et d'autorisation intégré gérant les utilisateurs, rôles et politiques de sécurité.

Stripe.net (v47.3.0) : SDK officiel pour l'intégration des paiements Stripe, garantissant sécurité et conformité PCI DSS.

Hangfire (v1.8.18) : Framework de gestion des tâches en arrière-plan pour le traitement asynchrone (envoi d'emails, nettoyage des réservations expirées).

Swashbuckle.AspNetCore (v6.9.0) : Génération automatique de la documentation API OpenAPI/Swagger facilitant l'intégration frontend et les tests.

PuppeteerSharp (v20.1.3) : Génération de PDF (factures, récapitulatifs) via contrôle programmatique de navigateur Chrome.

RazorLight (v2.3.1) : Moteur de templates pour la génération d'emails HTML et de documents dynamiques.

Bogus (v35.6.1) : Générateur de données de test facilitant le développement et les tests avec des jeux de données réalistes.

Base de données : PostgreSQL 15

PostgreSQL a été retenu pour ses performances, sa fiabilité et ses fonctionnalités avancées (JSONB, indexation sophistiquée, contraintes complexes). Sa compatibilité native avec .NET via Npgsql garantit une intégration optimale.

Paielement : Stripe

Stripe s'impose comme référence pour les paiements en ligne grâce à sa sécurité PCI DSS Level 1, son API intuitive, et son support international. L'intégration avec .NET via le SDK officiel assure fiabilité et conformité réglementaire.

Hébergement : VPS chez Hostinger

Le choix d'un VPS offre flexibilité, contrôle total sur l'environnement, et rapport qualité-prix optimal pour une application de cette envergure. L'architecture conteneurisée avec Docker facilite le déploiement et la scalabilité.

4.2 Schéma d'architecture

(Insérer un diagramme)

4.3 Structure de la base de données

(Lister les tables principales et relations)

5. Conception

- Diagrammes UML : cas d'utilisation, séquence, classes.
- Maquettes écran.
- Modèle de données.

6. Sécurité

La sécurité constitue un enjeu majeur pour une application gérant des données personnelles et des transactions financières.

6.1 Authentification et autorisation

Système d'authentification dual

L'application implémente un système d'authentification à double mécanisme pour optimiser à la fois la sécurité et l'expérience utilisateur :

Authentification par identifiants : Lors de la première connexion, l'utilisateur fournit ses identifiants (email/mot de passe). Le backend valide ces informations et génère :

- Un **JWT (JSON Web Token)** à durée de vie limitée (30 minutes) contenant les informations utilisateur et ses permissions
- Un **refresh token** à durée de vie étendue (7 jours) permettant le renouvellement automatique du JWT

Authentification automatique par cookies : Pour les sessions ultérieures, le mécanisme fonctionne ainsi :

1. Le refresh token est stocké dans un cookie **secure, strict et httpOnly**
2. Lors du rafraîchissement de la page, un interceptor Angular déclenche automatiquement une requête vers l'endpoint `/auth/refresh-token`
3. Le backend valide le refresh token et retourne un nouveau JWT avec les données utilisateur
4. Toutes les requêtes suivantes utilisent ce JWT pour l'authentification

Gestion des tokens côté frontend

Le frontend adopte une stratégie de stockage sécurisée :

- **JWT et données utilisateur** : Stockés en mémoire (variables JavaScript) pour éviter la persistance locale
- **Refresh token** : Stocké exclusivement dans un cookie avec les attributs de sécurité suivants :
 - **Secure** : Transmission uniquement via HTTPS
 - **SameSite=Strict** : Protection contre les attaques CSRF
 - **HttpOnly** : Inaccessible au JavaScript côté client

Cette approche offre une protection optimale contre les principales vulnérabilités :

6.2 Protection contre les attaques courantes

Protection XSS (Cross-Site Scripting)

- **Stockage en mémoire** : Les tokens JWT ne sont jamais persistés dans le localStorage ou sessionStorage, éliminant le risque d'exfiltration via du code JavaScript malveillant
- **Cookie HttpOnly** : Le refresh token est inaccessible au JavaScript, empêchant son vol par des scripts injectés
- **Validation des entrées** : Angular intègre nativement une protection contre l'injection de scripts dans les templates

Protection CSRF (Cross-Site Request Forgery)

- **Cookie SameSite=Strict** : Empêche l'envoi automatique du refresh token lors de requêtes cross-origin
- **JWT en headers** : L'utilisation de JWT dans les headers Authorization nécessite une action JavaScript explicite, impossible depuis un site tiers
- **Validation d'origine** : Vérification systématique de l'origine des requêtes côté backend

Protection contre l'injection SQL

L'utilisation d'**Entity Framework Core** comme ORM fournit une protection native contre les injections SQL :

- **Requêtes paramétrées** : Toutes les requêtes utilisent des paramètres typés, empêchant l'injection de code SQL
- **LINQ to SQL** : Les requêtes LINQ sont automatiquement converties en requêtes SQL sécurisées
- **Validation des modèles** : Les annotations de validation sur les modèles filtrent les données en amont

6.3 Chiffrement et protection des données

Gestion des mots de passe

- **Hachage BCrypt** : Les mots de passe sont hachés avec l'algorithme BCrypt (work factor 12) avant stockage
- **Salt unique** : Chaque mot de passe dispose d'un salt généré aléatoirement
- **Politique de mots de passe** : Validation de la complexité (8 caractères minimum, majuscules, minuscules, chiffres, caractères spéciaux)

Chiffrement des communications

- **HTTPS obligatoire** : Toutes les communications sont chiffrées via TLS 1.3
- **HSTS (HTTP Strict Transport Security)** : Headers configurés pour forcer l'utilisation d'HTTPS
- **Certificats SSL** : Utilisation de certificats Let's Encrypt avec renouvellement automatique

6.4 Conformité RGPD

Gestion des consentements

- **Consentement explicite** : L'application utilise uniquement les cookies essentielles à l'authentification, et c'est marqué explicitement. Il faut bien noter que lors de l'inscription, l'utilisateur doit accepter l'utilisation des cookies essentiels, sinon il ne peut pas s'inscrire, vu que sans les cookies, l'authentification sera perdue toutes les 30 minutes.

Droits des utilisateurs

- **Droit d'accès** : L'utilisateur peut consulter son profil en tout moment, via la page de profil.
- **Droit de rectification** : Interface de modification des données personnelles
- **Droit à l'effacement** : En cours.

6.5 Sécurité applicative

Validation et sanitisation

- **Validation côté client et serveur** : Double validation avec Angular Validators et FluentValidation (.NET)
- **Sanitisation des entrées** : Nettoyage automatique des données utilisateur (Angular)
- **Protection contre le brute force** : Verrouillage temporaire après échecs multiples (Dotnet 5 tentatives avant le blocage de compte)

6.6 Sécurité des paiements

Intégration Stripe

- **PCI DSS Level 1** : Conformité aux standards de sécurité des données de cartes de paiement
- **Tokenisation** : Aucune donnée de carte stockée localement, utilisation des tokens Stripe
- **3D Secure** : Authentification forte pour les paiements européens
- **Webhooks sécurisés** : Vérification cryptographique des notifications Stripe

Cette architecture de sécurité multicouche garantit une protection robuste des données utilisateur et des transactions financières, tout en maintenant une expérience utilisateur fluide et conforme aux réglementations en vigueur.

7. Déploiement

Le déploiement de l'application suit une approche moderne basée sur la conteneurisation Docker et l'intégration continue via GitHub Actions. Cette stratégie garantit la reproductibilité, la scalabilité et la fiabilité du processus de mise en production.

7.1 Environnements de déploiement

Environnement de développement

- **Système d'exploitation** : Windows 11
- **IDE** : Visual Studio Code avec extensions Angular et Docker
- **Outils** : Angular CLI v19, Node.js v20, Docker Desktop
- **Base de données** : PostgreSQL 15 en conteneur Docker local

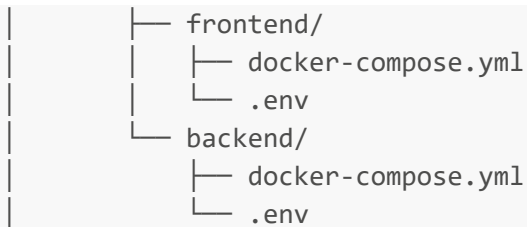
Environnement de production

- **Serveur** : VPS Ubuntu 24.04 LTS chez Hostinger
- **Orchestration** : Docker Compose pour la gestion des conteneurs
- **Reverse Proxy** : Nginx Proxy Manager pour la gestion des domaines et certificats SSL
- **Monitoring** : Logs centralisés et surveillance des performances

7.2 Architecture de déploiement sur VPS

L'infrastructure sur le VPS est organisée selon une structure hiérarchique optimisant la séparation des environnements :

```
/root/
├── nginx-proxy-manager/      # Reverse proxy centralisé
│   └── docker-compose.yml
├── skillhive/                # Environnement de production
│   ├── frontend/
│   │   ├── docker-compose.yml
│   │   └── .env
│   └── backend/
│       ├── docker-compose.yml
│       └── .env
└── skillhive-test            # Environnement de test
```



Cette organisation permet :

- **Isolation des environnements** : Production et test complètement séparés
- **Gestion centralisée des proxy** : Un seul point d'entrée pour tous les services
- **Configuration sécurisée** : Variables d'environnement isolées par contexte
- **Scalabilité horizontale** : Possibilité d'ajouter facilement de nouveaux environnements

7.3 Conteneurisation avec Docker

Dockerfile multi-stage

La conteneurisation utilise une approche multi-stage optimisant la taille des images et permettant la génération de plusieurs environnements :

```
# Stage de construction
FROM node:20 AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci

# Stage de production
FROM build AS production
COPY . .
RUN npm run build:prod

# Stage de test
FROM build AS testing
COPY . .
RUN npm run build:test

# Runtime de production
FROM nginx:alpine as prod-runtime
COPY --from=production /app/dist/skill-hive/browser /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

# Runtime de test
FROM nginx:alpine as test-runtime
COPY --from=testing /app/dist/skill-hive/browser /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Avantages de cette approche :

- **Optimisation de taille** : Les images finales ne contiennent que le strict nécessaire (nginx + fichiers statiques)
- **Séparation des environnements** : Builds distincts pour production et test avec configurations appropriées
- **Sécurité renforcée** : Images basées sur Alpine Linux (surface d'attaque minimale)
- **Performance** : Nginx optimisé pour le serving de fichiers statiques

7.4 Intégration continue avec GitHub Actions

Pipeline de déploiement automatisé

Le fichier `.github/workflows/cd.yml` orchestre le processus de déploiement continu :

```
name: CD Pipeline for Angular Project

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Set version
        id: set_version
        run: echo "FRONT_IMAGE_VERSION=prod" >> $GITHUB_ENV

      - name: Checkout the branch
        uses: actions/checkout@v4

      - name: Login to docker hub
        uses: docker/login-action@v2
        with:
          username: ${ secrets.DOCKER_HUB_USERNAME }
          password: ${ secrets.DOCKER_HUB_ACCESS_TOKEN }

      - name: Build the docker image
        run: docker build --target prod-runtime -t mahdimcheik/skill-hive-front:${ env.FRONT_IMAGE_VERSION } .

      - name: Push the docker image to the docker hub
        run: docker push mahdimcheik/skill-hive-front:${ env.FRONT_IMAGE_VERSION }

      - name: Deploy on VPS via SSH
        uses: appleboy/ssh-action@v1.0.0
        with:
          host: ${ secrets.VPS_HOST }
```

```

username: ${ secrets.VPS_USER }}
key: ${ secrets.VPS_SSH_PRIVATE_KEY }}
script: |
    export FRONT_IMAGE_VERSION=${ env.FRONT_IMAGE_VERSION }}
    docker pull mahdimcheik/skill-hive-front:${ env.FRONT_IMAGE_VERSION
}}

    docker compose -f /root/skillhive/frontend/docker-compose.yml up -d --
force-recreate

```

7.5 Processus de déploiement détaillé

Étape 1 : Déclenchement automatique

Le déploiement s'active automatiquement lors d'un push sur la branche `main` ou `test`, garantissant une mise en production immédiate des changements validés. En fonction de la branche, une série différente des instructions sera exécutée.

Étape 2 : Gestion des versions

```
echo "FRONT_IMAGE_VERSION=rc-1.0.1" >> $GITHUB_ENV
```

Le système de versioning permet de différencier les builds et facilite les rollbacks si nécessaire.

Étape 3 : Construction de l'image Docker

```
docker build --target prod-runtime -t mahdimcheik/skill-hive-front:${ env.FRONT/BACK_IMAGE_VERSION }} .
```

- Dans cet exemple, l'utilisation du stage `prod-runtime` permet de séparer les variables liées au test de celles liées à la production. remarquez que dans le dockerfile `FROM nginx:alpine as prod-runtime COPY --from=production /app/dist/skill-hive/browser /usr/share/nginx/html` le profil `prod-runtime` permet préciser à angular de build le profil de production et donc utiliser les variables qui y sont liées grâce à la configuration

```

    "production": {
      ...
      "fileReplacements": [
        {
          "replace": "src/environments/environment.ts",
          "with":
"src/environments/environment.production.ts"
        }
      ],
      ...
    },

```


un traitement similaire est réservé au testing.

Par rapport au backend, les variables et secrets sont fournis grâce aux fichiers .env séparés.

Étape 4 : Publication sur Docker Hub

```
docker push mahdimcheik/skill-hive-front:${{ env.FRONT/BACK_IMAGE_VERSION }}
```

Le registre Docker Hub centralise les images, permettant leur déploiement sur n'importe quel environnement disposant de Docker.

Étape 5 : Déploiement sur VPS

La connexion SSH sécurisée avec clé privée exécute les commandes de déploiement :

```
docker pull mahdimcheik/skill-hive-front:${{ env.FRONT/BACK_IMAGE_VERSION }}
docker compose -f /root/skillhive/frontend/docker-compose.yml up -d --force-recreate
```

7.7 Sécurité du déploiement

Gestion des secrets

- **Variables d'environnement** : Stockage sécurisé dans GitHub Secrets
- **Clés SSH** : Authentification par clé privée, sans mot de passe
- **Tokens Docker Hub** : Utilisation de tokens d'accès plutôt que mots de passe
- **Fichiers .env** : Variables sensibles isolées

Réseau et accès

- **SSL/TLS** : Certificats automatiques via Let's Encrypt
- **Isolation des conteneurs** : Réseaux Docker dédiés par environnement

7.8 Monitoring et maintenance

Surveillance automatisée

- **Health checks** : Vérification automatique de l'état des conteneurs
- **Logs centralisés** : Agrégation des logs pour analyse et debugging
- **Alertes** : Notifications en cas de dysfonctionnement

Stratégie de rollback

En cas de problème, le rollback s'effectue en modifiant la variable d'environnement :

```
export FRONT_IMAGE_VERSION=previous-version
docker compose up -d --force-recreate
```

7.9 Optimisations et bonnes pratiques

Performance

- **Images multi-architecture** : Support AMD64
- **Cache Docker** : Optimisation des layers pour accélérer les builds
- **CDN** : Nginx configuré avec compression gzip et cache headers

Fiabilité

- **Zero-downtime deployment** : Mise à jour sans interruption de service
- **Configuration immutable** : a venir
- **Backup automatique** : a venir

Cette architecture de déploiement offre une solution scalable et sécurisée, permettant une mise en production rapide et fiable tout en maintenant une séparation claire entre les environnements de développement, test et production.

8. Tests et assurance qualité

- Tests unitaires.
 - Tests d'intégration.
 - Tests end-to-end.
 - Validation fonctionnelle.
-

10. Maintenance et évolutivité

- Correctifs de bugs.
 - Ajout futur de nouvelles fonctionnalités.
 - Amélioration de l'expérience utilisateur.
-

11. Gestion de projet

- Méthodologie : Agile/Scrum.
 - Planning.
 - Suivi des tâches.
-

12. Conclusion

- Bilan.
- Limites.

- Perspectives.
-

13. Annexes

- Captures d'écran.
- Documentation API.
- Schéma BDD.