Table des matières

I. Introduction

1. Présentation du projet

Hexaplanning est une application web de gestion de tâches, pensée pour transformer la to-do list classique en une expérience visuelle et ludique. Reprenant la nomenclature des jeux-vidéos, les tâches sont appelées "quêtes". Chacune d'entre elles peut être placée sur une carte d'hexagones, permettant à l'utilisateur de visualiser ses objectifs comme un parcours à accomplir.

Cette approche vise à rendre la planification plus motivante et interactive, en s'inspirant des mécaniques de jeu et de la gamification. Hexaplanning est destinée tout particulièrement aux personnes sujètes à un trouble de l'attention et ayant de la difficulté à se concentrer sur une tâche à la fois.

L'application a été développée en mobile-first, favorisant une utilisation quotidienne permettant à l'utilisateur d'avoir un aperçu de sa progression et de la mettre à jour régulièrement. Elle est bien entendu accessible également sur ordinateur, et l'utilisateur pourra se créer un compte pour accéder à sa progression depuis n'importe quel appareil.

2. Objectifs et contexte

Le projet est né du constat que la gestion des tâches peut rapidement devenir monotone et décourageante, surtout lorsqu'elle se limite à une simple liste. Hexaplanning propose une alternative visuelle et dynamique, où chaque utilisateur peut organiser ses quêtes selon ses priorités et ses envies, tout en bénéficiant d'un suivi clair de sa progression. L'application s'adresse à toute personne souhaitant mieux organiser son temps, que ce soit dans un cadre personnel, scolaire ou professionnel, et met l'accent sur l'ergonomie, la sécurité et la personnalisation de l'expérience.

II. Fonctionnalités principales

1. Page d'accueil

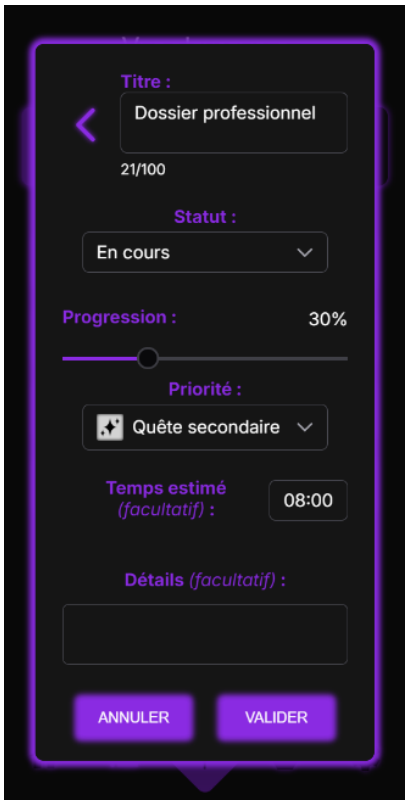
Page d'accueil d'Hexaplanning.

La page d'accueil apparaît dès la connexion de l'utilisateur, et affiche le nombre de quêtes qu'il lui reste à accomplir

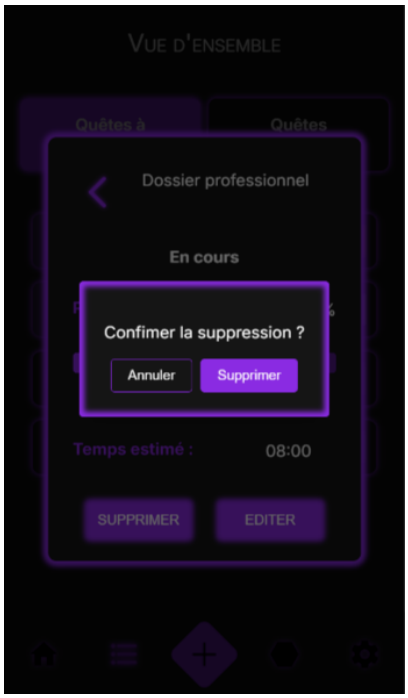
2. Gestion des quêtes (tâches)

Les tâches, appelées "quêtes", sont au cœur de l'application. Chaque quête possède un titre, un statut (en attente, en cours et terminée) et une priorité (primaire, secondaire ou tertiaire, avec une icône et un code couleur associés), ainsi qu'une description et un temps estimé en option, ainsi qu'un pourcentage de progression (associé à une barre de progression) dans le cas des quêtes en cours. L'utilisateur peut créer, éditer ou supprimer une quête, la marquer rapidement comme terminée ou la remettre en attente, et l'associer à un hexagone sur la carte prévue à cet effet.

Modale de détails d'une quête.

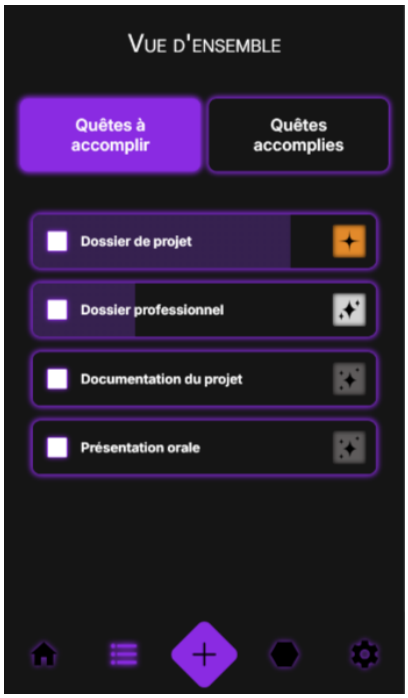


Edition d'une quête existante.

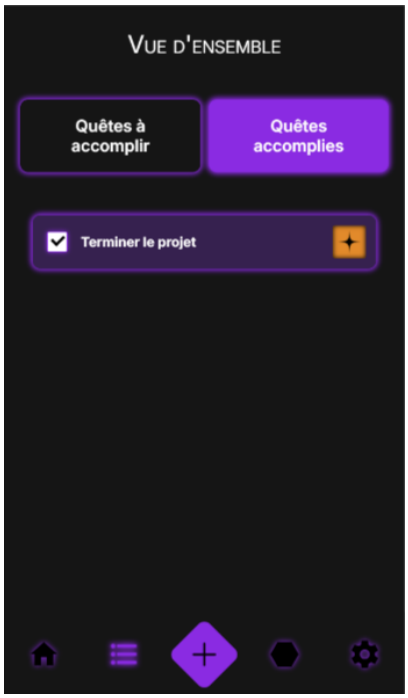


Modale de suppression d'une quête.

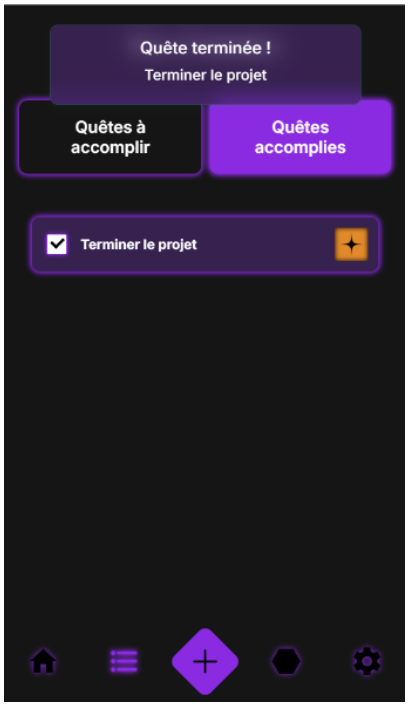
Un affichage standard des quêtes est proposé aux utilisateurs, sous forme de deux listes : l'une pour les quêtes à accomplir, l'autre pour les quêtes accomplies. La navigation se fait via un menu composé de deux onglets. Les quêtes à accomplir sont triées par ordre de priorité.



Liste des quêtes non accomplies.



Listes des quêtes accomplies.



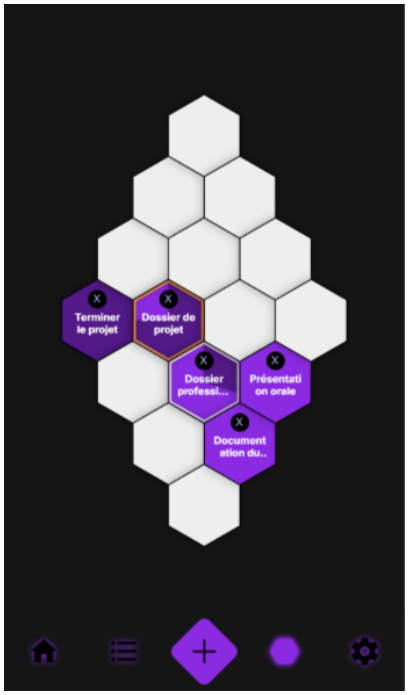
Toast de succès : quête accomplie.

Sur ces listes, l'utilisateur peut voir d'un coup d'oeil le titre de chaque quête ainsi qu'une icône représentant sa priorité, doublée d'un code couleur (orangé pour les principales, argenté pour les secondaires, gris foncé pour les tertiaires). Il dispose également d'un bouton à cocher pour aisément marquer une quête comme accomplie - ce qui déclenche un toast de succès - ou au contraire réhabiliter une quête terminée. Si la quête est indiquée comme "en cours", la barre de progression s'affiche directement sur l'aperçu de la quête, la remplissant progressivement d'une couleur plus sombre. Les quêtes terminées sont entièrement remplies.

3. Affichage visuel en hexagones (carte)

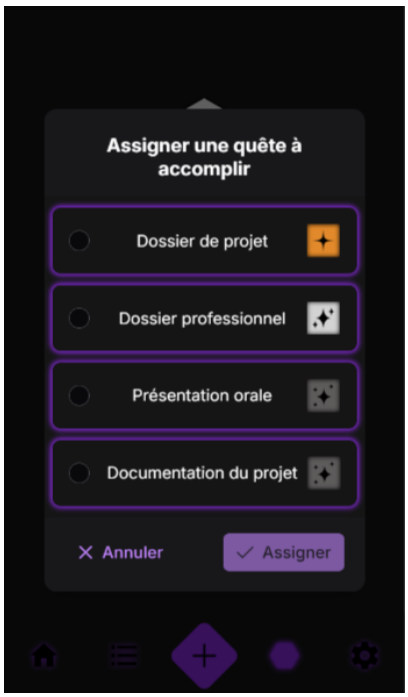
L'originalité d'Hexaplanning réside dans sa représentation visuelle : une carte d'hexagones sur laquelle l'utilisateur peut placer ses quêtes. Chaque hexagone peut accueillir une quête, et un code couleur sur le liseré

permet d'identifier rapidement sa priorité (orangé pour les principales, argenté pour les secondaires, et aucun liseré pour les tertiaires). Les quêtes terminées apparaissent avec un fond plus sombre, et les quêtes en cours disposent d'une barre de progression radiale qui remplit progressivement l'hexagone avec cette couleur, à la manière d'une horloge.



Page de la carte d'hexagones.

L'utilisateur peut assigner une quête en cliquant ou appuyant sur un hexagone vide, faisant apparaître une modale contenant la liste de toutes les quêtes non accomplies, et en sélectionnant la quête de son choix. Il pourra ensuite la désassigner d'un simple clic sur l'icône de croix au-dessus du titre de la quête, ce qui déclenchera une modale de confirmation.



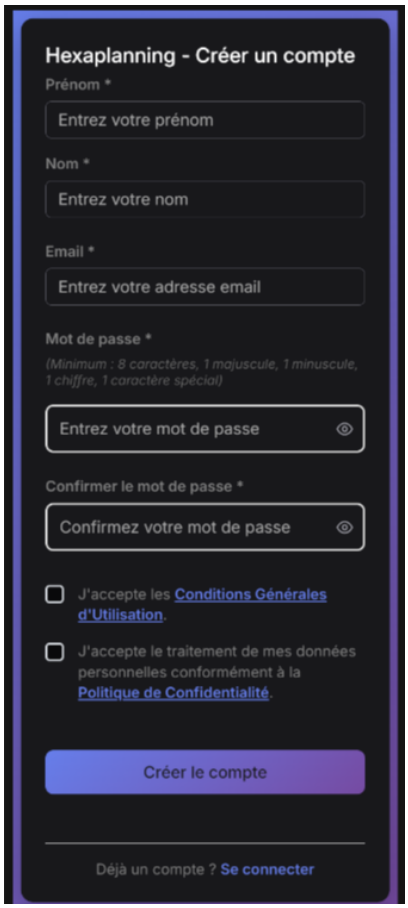
Modale d'assignation d'une quête à un hexagone.

Modale de désassignation d'une quête à un hexagone.

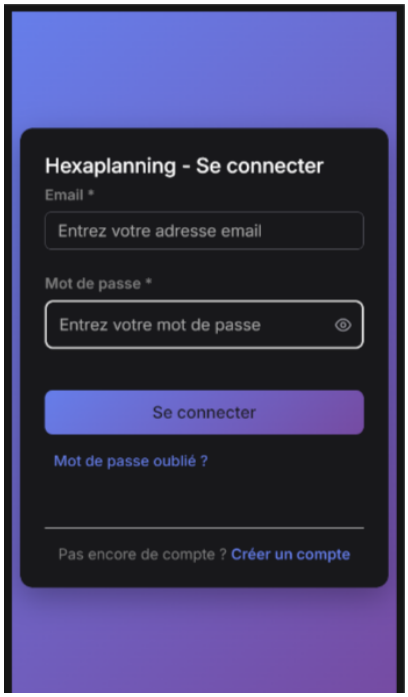
Tout comme sur les listes des quêtes, il suffit de cliquer ou d'appuyer sur un hexagone associé à une quête pour afficher les détails de la quête en question, et éventuellement modifier ou supprimer la quête (ce qui la fera disparaître de la carte et des listes).

4. Système d'authentification et gestion des utilisateurs

L’accès à l’application nécessite la création d’un compte et une authentification sécurisée. L'utilisateur devra accepter les CGU et la politique de confidentialité, accessibles via des liens sur le formulaire de création de compte. Le mot de passe choisi devra respecter les normes standard : au minimum 8 caractères dont 1 lettre majuscule, 1 lettre minuscule, 1 chiffre et 1 caractère spécial. Après son enregistrement, l'utilisateur sera redirigé vers la page de connexion, et il peut aisément naviguer entre la connexion et la création de compte via un lien en bas de page.



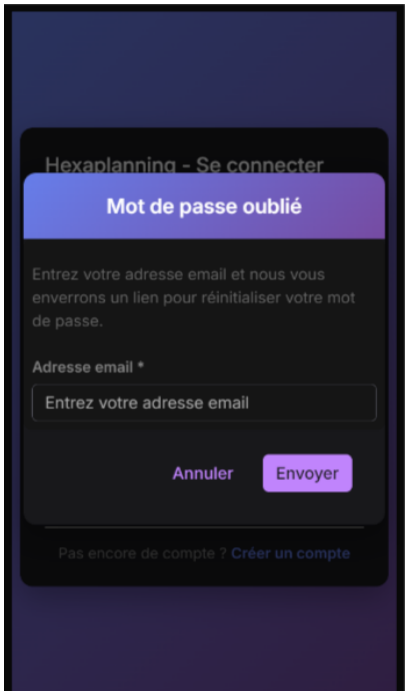
Page de création de compte.



Page de connexion.

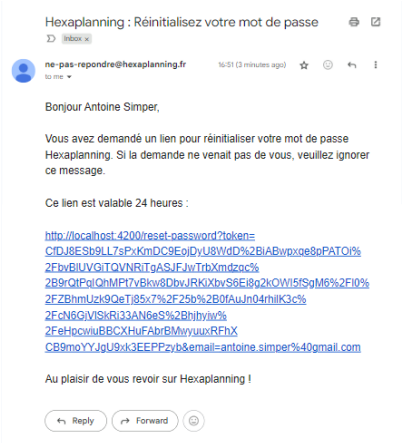
Un système de gestion des mots de passe oubliés est en place, avec envoi d'email pour la réinitialisation. Lorsque l'utilisateur clique sur "mot de passe oublié", une modale s'ouvre. Si l'utilisateur avait déjà rentré son adresse e-mail dans le champ de connexion, il sera automatiquement reporté dans le champ de la modale. Au clic sur le bouton d'envoi, un toast informe l'utilisateur qu'un mail a été délivré à l'adresse indiquée, si elle existe. En effet, il s'agit de ne pas confirmer ou infirmer la présence de cette adresse e-mail dans la base de

données. De plus, il ne peut y avoir qu'une seule requête vers la même adresse toutes les 5 minutes, afin d'éviter le spam d'une adresse e-mail et la saturation du service de mail.

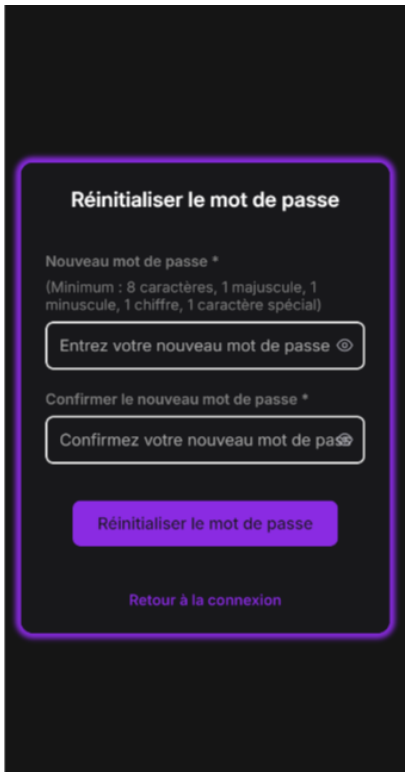


Modale de mot de passe oublié.

Le destinataire recevera un mail contenant un lien de réinitialisation de mot de passe. Ce lien le redirigera vers la page prévue à cet effet, avec dans l'url un token valable une heure, et l'adresse e-mail du compte à modifier. Sans ces deux éléments valides, la requête ne pourra être acceptée. L'utilisateur n'a plus qu'à rentrer son nouveau mot de passe et à le confirmer, avant d'être redirigé vers la page de connexion.

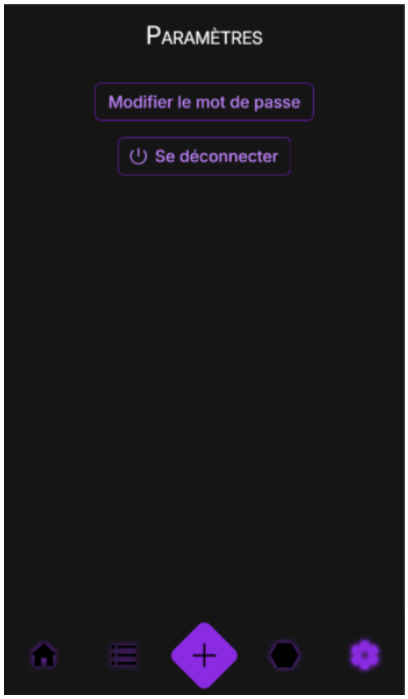


Mail de réinitialisation de mot de passe.



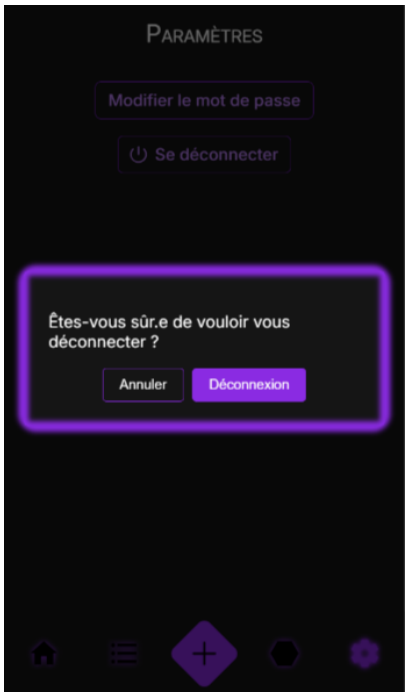
Page de réinitialisation de mot de passe.

L'utilisateur peut également changer son mot de passe depuis l'interface : en accédant au menu des paramètres, il aura la possibilité d'ouvrir une modale lui demandant son mot de passe actuel ainsi que le nouveau. Depuis ce même menu, il pourra se déconnecter de l'application.



Page de paramètres.

Modale de changement de mot de passe.



Modale de déconnexion.

La sécurité des données et la protection contre les accès non autorisés sont assurées par des mécanismes robustes côté backend.

5. Navigation et ergonomie

L'application propose un menu apparaissant en permanence en bas de page, et permettant de naviguer entre l'accueil, les listes de quêtes, la carte des hexagones et les paramètres. Un bouton dédié au centre du menu permet de créer rapidement une nouvelle quête, qui viendra s'insérer dans la liste qui lui correspond, et sera accessible dans la modale d'assignation à un hexagone.

Modale de création de quête.

L'interface est pensée pour être intuitive, responsive et agréable à utiliser, afin de maximiser l'engagement et la productivité de l'utilisateur.

III. Modélisation des données

1. MCD (Modèle Conceptuel de Données)

Schéma de la base de données relationnelle d'Hexaplanning, réalisé avec dbdiagram.io.

2. MLD (Modèle Logique de Données)

- Table **UserApp** (Id PK, FirstName, LastName, Email, PasswordHash, CreatedAt, UpdatedAt, IsArchived, ...)
- Table **Quest** (Id PK, Title, Description, EstimatedTime, Advancement, UserId FK, PriorityId FK, StatusId FK, HexAssignmentId FK, CreatedAt, UpdatedAt, IsArchived)
- Table **Priority** (Id PK, Name, Color, BorderColor, Icon, CreatedAt, UpdatedAt, IsArchived)
- Table **Status** (Id PK, Name, Color, Icon, CreatedAt, UpdatedAt, IsArchived)
- Table **HexAssignment** (Id PK, Q, R, S, QuestId FK, CreatedAt, UpdatedAt, IsArchived)
- Table **Mail** (MailTo, MailSubject, MailBody, MailFrom, Receiver)

3. Description des entités et relations

UserApp (Utilisateur)

- Un utilisateur peut créer plusieurs quêtes. Il possède :
    - Un nom et un prénom.
    - Une adresse e-mail unique.
    - Un mot de passe (hashé dans la base de données).
    - Une liste de quêtes.
    - Des métadonnées : date de création, date de mise à jour, statut d'archivage.

Quest (Quête)

- Une quête appartient à un seul utilisateur. Elle possède :
    - Un titre (limité à 100 caractères).
    - Optionnellement, une description.
    - Optionnellement, un temps estimé.
    - Un pourcentage d'avancement (Advancement) pour les quêtes en cours.
    - Un UserId pour la rattacher à son utilisateur.
    - Un PriorityId pour définir sa priorité.
    - Un StatusId pour définir son statut.
    - Optionnellement, un HexAssignmentId pour l'assigner à un hexagone.
    - Des métadonnées : date de création, date de mise à jour, statut d'archivage.

Priority (Priorité)

- Une priorité peut être associée à plusieurs quêtes. Elle possède :
    - Un nom (PRIMARY, SECONDARY, TERTIARY).
    - Une couleur principale.
    - Une couleur de bordure (BorderColor) pour l'affichage sur la carte.
    - Optionnellement, une icône.
    - Des métadonnées : date de création, date de mise à jour, statut d'archivage.

Status (Statut)

- Un statut peut être associé à plusieurs quêtes. Il possède :
    - Un nom (en attente, en cours, terminée).
    - Une couleur pour l'affichage.
    - Optionnellement, une icône.
    - Des métadonnées : date de création, date de mise à jour, statut d'archivage.

HexAssignment (Assignation d'hexagone)

- Un hexagone (HexAssignment) est lié à une seule quête. Il possède :
    - Un jeu de coordonnées q, r, s qui lui est unique (système de coordonnées hexagonales).
    - Un QuestId pour la quête assignée.
    - Des métadonnées : date de création, date de mise à jour, statut d'archivage.

Mail

- Un mail est indépendant et permet d'envoyer des communications (réinitialisation de mot de passe, bienvenue, etc.). Il possède :
    - Un destinataire (MailTo).
    - Un sujet (MailSubject).
    - Un corps de message (MailBody).
    - Un expéditeur (MailFrom).
    - Un destinataire lié à un utilisateur (Receiver).

Relations principales :

- **UserApp 1:N Quest** : Un utilisateur possède plusieurs quêtes.
- **Quest N:1 Priority** : Une quête a une priorité.
- **Quest N:1 Status** : Une quête a un statut.
- **Quest 1:1 HexAssignment** : Une quête peut être assignée à un hexagone (optionnel).

IV. Architecture technique

1. Vue d'ensemble

L'architecture d'Hexaplanning repose sur un frontend Angular (avec PrimeNG pour les composants de base), un backend .NET et une base de données PostgreSQL. Elle fait également appel à Brevo pour permettre l'envoi de mails. Cette approche modulaire facilite la maintenance, l'évolutivité et la sécurité de l'application. La communication entre les différentes couches s'effectue via une API REST sécurisée.

Schéma global

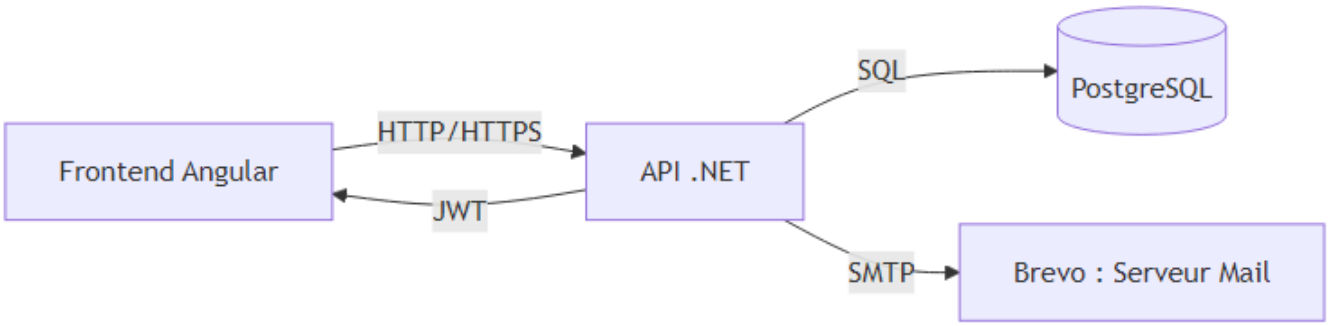


Schéma global de l'architecture d'Hexaplanning.

2. Frontend (Angular)

- **Technologie** : Angular 18
- **Structure** : Organisation en modules, composants, services et modèles TypeScript.
- **Responsabilité** : Gestion de l'interface utilisateur, navigation, appels API, gestion du token JWT, affichage dynamique de la carte d'hexagones, gestion des quêtes.
- **Sécurité** : Intercepteur HTTP pour l'ajout automatique du JWT, guards de navigation pour protéger les routes sensibles.
- **Tests** : Utilisation de Jest et Cypress pour les tests unitaires et end-to-end.

3. Backend (.NET)

- **Technologie** : ASP.NET Core 8
- **Structure** : Architecture en couches (Controllers, Services, Models, DataContext, Utilities).
- **Responsabilité** : Exposition d'une API RESTful, gestion de l'authentification (JWT), logique métier (création/gestion des quêtes, hexagones, utilisateurs), validation et sécurisation des données. Un service d'envoi d'e-mails est intégré pour la gestion du mot de passe oublié et d'autres notifications : il s'appuie sur la solution Brevo (anciennement Sendinblue), permettant l'envoi fiable et sécurisé de courriels transactionnels depuis l'API .NET.
- **Sécurité** : Middleware d'authentification JWT, validation des entrées, gestion des droits d'accès, protection contre les attaques courantes.
- **Tests** : Couverture par des tests unitaires (xUnit) et des tests d'intégration.

4. Base de données (PostgreSQL)

- **Modélisation** : Respect du MCD/MLD présenté plus haut.
- **Gestion** : Migrations Entity Framework Core pour la création et l'évolution du schéma.
- **Sécurité** : Accès restreint via le backend uniquement, aucune exposition directe.

5. Communication API

- **Format** : JSON via HTTP(S)
- **Endpoints** : Authentification, gestion des quêtes, gestion des hexagones, gestion des utilisateurs.
- **Sécurité** : Toutes les routes sensibles sont protégées par JWT, CORS configuré pour limiter les origines autorisées.

6. Conteneurisation et déploiement

- **Docker** : Chaque composant (frontend, backend, base de données) dispose de son propre Dockerfile pour faciliter le déploiement et l'isolation.
- **Orchestration** : Utilisation de docker-compose pour le développement local et le déploiement sur serveur.
- **Reverse Proxy** : Nginx Proxy Manager pour la gestion des domaines et des certificats SSL.
- **CI/CD** : Pipelines GitHub Actions pour l'intégration et le déploiement continus.

7. Sécurité et bonnes pratiques

- **Authentification JWT** pour toutes les opérations sensibles.
- **Validation systématique** des données côté backend.
- **Gestion des erreurs** centralisée.
- **Logs** pour le suivi et l'audit des actions critiques.
- **Séparation stricte** des responsabilités entre frontend et backend.

Cette architecture garantit robustesse, évolutivité et sécurité, tout en permettant une expérience utilisateur fluide et moderne.

V. Qualité logicielle et tests

La qualité logicielle d'Hexaplanning repose sur une stratégie de tests complète, principalement concentrée sur l'API .NET, afin de garantir la robustesse, la fiabilité et la maintenabilité du backend.

1. Tests unitaires (backend)

Les tests unitaires sont réalisés avec xUnit et couvrent les principaux services métiers, notamment le service de gestion des quêtes (`QuestService`). Ces tests vérifient le bon fonctionnement des méthodes de création, lecture, mise à jour et suppression de quêtes, ainsi que la gestion des cas limites (identifiants invalides, absence de données, etc.).

Exemples de méthodes testées :

- Création d'une quête (`CreateQuestAsync`)
- Récupération d'une quête par ID (`GetQuestByIdAsync`)
- Mise à jour et suppression de quêtes (`UpdateQuestAsync`, `DeleteQuestAsync`)
- Récupération des quêtes selon leur statut (en attente, terminées, non assignées)

2. Tests d'intégration

Des tests d'intégration automatisés valident l'ensemble du pipeline API, de la couche HTTP jusqu'à la base de données PostgreSQL (via Testcontainers). Ils simulent des scénarios réels, comme la récupération de quêtes via des requêtes authentifiées, la gestion des droits d'accès, et la cohérence des données persistées.

Caractéristiques :

- Utilisation de `WebApplicationFactory` pour lancer l'API en environnement de test
- Base de données PostgreSQL éphémère (Testcontainers)
- Données de test injectées automatiquement (utilisateur, quêtes)
- Vérification de la sécurité (JWT requis, accès refusé si non authentifié)

3. Tests de charge et fixtures

Des fixtures de données sont utilisées pour simuler des volumes importants de quêtes et d'utilisateurs, grâce à la librairie Bogus. Cela permet de valider la tenue en charge de l'API et la stabilité des traitements sur de grands ensembles de données. Les tests ont été réalisés avec 100000 utilisateurs et 1000000 de quêtes pour s'assurer de la robustesse de la base de données.

4. Stratégie de validation

Chaque nouvelle fonctionnalité ou correction de bug s'accompagne de tests dédiés. Les tests sont exécutés automatiquement lors des pipelines CI/CD (GitHub Actions), garantissant l'absence de régressions avant chaque déploiement. La couverture de code est régulièrement analysée pour cibler les zones à renforcer.

Cette démarche assure un haut niveau de confiance dans la qualité logicielle du backend, tout en facilitant l'évolution continue du projet.

VI. Déploiement et intégration continue

L'automatisation du déploiement et de l'intégration continue est assurée par des pipelines GitHub Actions distincts pour le frontend Angular et l'API .NET. Cette organisation garantit des mises en production fiables, rapides et reproductibles.

1. Intégration continue (CI) de l'API

Un pipeline CI dédié à l'API .NET s'exécute à chaque push sur la branche `main` :

- **Tests unitaires** : Compilation et exécution des tests unitaires (`dotnet test ./TestsUnitaires`)
- **Tests d'intégration** : Lancement des tests d'intégration sur une base PostgreSQL éphémère (`dotnet test ./TestsIntegration`)
- **Vérification de la qualité** : Toute régression ou échec bloque la suite du pipeline

Extrait du workflow :

```
jobs:
   test-unitaire:
      ...
      - run: dotnet test --no-build --verbosity normal ./TestsUnitaires
   test-integration:
      ...
      - run: dotnet test --no-build --verbosity detailed  ./TestsIntegration
```

2. Déploiement continu (CD) du backend

Le backend .NET dispose également d'un pipeline CD automatisé. Celui-ci ne se déclenche que si le pipeline CI de l'API s'est terminé avec succès (`workflow_run`). Il effectue les étapes suivantes :

- **Build Docker** : Construction de l'image Docker de l'API
- **Push Docker** : Publication de l'image sur Docker Hub
- **Déploiement VPS** : Connexion SSH au serveur OVH, pull de la nouvelle image et redémarrage du conteneur backend via `docker compose`

Extrait du workflow :

```
on:
   workflow_run:
      workflows: ["CI pipeline for the API"]
      types:
         - completed
jobs:
   build-and-deploy:
      ...
      - run: docker build -t antoinespr/hexaplanning-api:dev1 .
      - run: docker push antoinespr/hexaplanning-api:dev1
      - uses: appleboy/ssh-action@v1.0.0
         with:
            script: |
               docker pull antoinespr/hexaplanning-api:dev1
               docker compose -f /home/ubuntu/backend/docker-compose.yml up -d --
force-recreate
```

Le frontend Angular dispose d'un pipeline CD qui automatise la construction, la publication et le déploiement sur le serveur de production :

- **Build Docker** : Construction de l'image Docker de l'application Angular
- **Push Docker** : Publication de l'image sur Docker Hub
- **Déploiement VPS** : Connexion SSH au serveur OVH, pull de la nouvelle image et redémarrage du conteneur via `docker compose`

Extrait du workflow :

```
jobs:
   deploy:
      ...
      - run: docker build --target prod-runtime -t antoinespr/hexaplanning-
front:dev1 .
      - run: docker push antoinespr/hexaplanning-front:dev1
      - uses: appleboy/ssh-action@v1.0.0
         with:
            script: |
               docker pull antoinespr/hexaplanning-front:dev1
               docker compose -f /home/ubuntu/frontend/docker-compose.yml up -d --
force-recreate
```

3. Déploiement continu (CD) du frontend

Le frontend Angular bénéficie d'un pipeline CD dédié, déclenché à chaque mise à jour de la branche principale. Ce pipeline prend en charge l'ensemble du cycle de livraison : il construit l'application en mode production, génère une image Docker optimisée, la publie sur Docker Hub, puis orchestre le déploiement sur le serveur distant. L'automatisation garantit que la dernière version du frontend est toujours disponible en production, sans intervention manuelle.

Le pipeline s'appuie sur GitHub Actions et utilise des secrets pour sécuriser l'accès au registre Docker et au serveur. L'étape de déploiement s'effectue via SSH, assurant un redémarrage fluide du conteneur frontend sans interruption de service pour les utilisateurs.

Extrait du workflow :

```
jobs:
   deploy:
      ...
      - run: docker build --target prod-runtime -t antoinespr/hexaplanning-
front:dev1 .
      - run: docker push antoinespr/hexaplanning-front:dev1
      - uses: appleboy/ssh-action@v1.0.0
         with:
            script: |
               docker pull antoinespr/hexaplanning-front:dev1
               docker compose -f /home/ubuntu/frontend/docker-compose.yml up -d --
force-recreate
```

4. Conteneurisation et orchestration

Chaque composant (frontend, backend, base de données) dispose de son propre Dockerfile. Le déploiement s'effectue via `docker compose`, facilitant la gestion, la montée en charge et la maintenance.

5. Hébergement et reverse proxy

L'application est hébergée sur un VPS OVH, avec Nginx Proxy Manager pour la gestion des domaines et des certificats SSL. Cette architecture assure la sécurité, la disponibilité et la scalabilité du service.

Cette chaîne CI/CD garantit des livraisons rapides, sûres et automatisées, tout en limitant les interventions manuelles et les risques d'erreur.

Le résultat final est disponible sous le nom de domaine hexaplanning.fr.

VII. Sécurité

La sécurité est un pilier central d'Hexaplanning, intégrée à tous les niveaux de l'architecture pour garantir la confidentialité, l'intégrité et la disponibilité des données utilisateurs.

1. Authentification et gestion des accès

- **JWT (JSON Web Token)** : Toutes les opérations sensibles nécessitent une authentification par token JWT, généré lors de la connexion et vérifié à chaque requête côté backend.
- **Guards et Intercepteurs** : Le frontend Angular utilise des guards pour protéger les routes et un intercepteur HTTP pour injecter automatiquement le token dans les requêtes API.

2. Validation et intégrité des données

- **Validation systématique** : Toutes les entrées utilisateur sont validées côté backend (.NET) pour éviter les injections, incohérences ou données malformées.
- **Gestion des erreurs** : Les erreurs sont centralisées et les messages d'erreur ne révèlent jamais d'informations sensibles.

3. Protection contre les attaques

- **Force brute** : Limitation du nombre de tentatives de connexion et gestion des comptes bloqués après plusieurs échecs.
- **CORS** : Configuration stricte des origines autorisées pour l'API.
- **Sécurité des mots de passe** : Hashage fort (ASP.NET Identity), politique de complexité et gestion sécurisée du reset via email (Brevo).

4. Sécurité de la conteneurisation et du déploiement

- **Isolation** : Chaque composant (frontend, backend, base de données) s'exécute dans un conteneur dédié.

- **Secrets** : Les secrets (tokens, clés, mots de passe) sont stockés dans les variables d'environnement et jamais dans le code source.
- **Reverse proxy** : Nginx Proxy Manager gère les certificats SSL et protège l'accès aux services.

5. Surveillance et audit

- **Logs** : Les actions critiques sont journalisées côté backend pour permettre un audit et une détection rapide d'anomalies.
- **Mises à jour** : Les dépendances et images Docker sont régulièrement mises à jour pour corriger les vulnérabilités.

Cette approche globale permet de garantir un haut niveau de sécurité pour les utilisateurs et les données de la plateforme.

VIII. Technologies utilisées

Le projet Hexaplanning s'appuie sur un ensemble de technologies modernes, choisies pour leur robustesse, leur écosystème et leur adéquation avec les besoins fonctionnels et non fonctionnels du projet.

1. Frontend

- **Angular 18** : Framework SPA reconnu pour sa structure modulaire, sa maintenabilité et sa communauté active. Il facilite la création d'interfaces dynamiques, responsives et testables.
- **TypeScript** : Apporte la sécurité de typage et la clarté du code, essentielle pour un projet d'envergure.
- **Jest & Cypress** : Outils de référence pour les tests unitaires et end-to-end, assurant la fiabilité de l'interface utilisateur.

2. Backend

- **ASP.NET Core 8** : Framework backend performant, sécurisé et multiplateforme, idéal pour exposer une API REST robuste et scalable.
- **Entity Framework Core** : ORM facilitant la gestion et la migration de la base de données, tout en assurant la cohérence des modèles.
- **xUnit** : Framework de tests unitaires moderne et flexible, intégré à l'écosystème .NET.

3. Base de données

- **PostgreSQL** : SGBD open source reconnu pour sa fiabilité, ses performances et ses capacités avancées (transactions, indexation, JSON, etc.).

4. Infrastructure et DevOps

- **Docker** : Conteneurisation de chaque composant pour garantir la portabilité, l'isolation et la reproductibilité des environnements.
- **docker-compose** : Orchestration simplifiée du déploiement multi-conteneurs.
- **GitHub Actions** : Automatisation des pipelines CI/CD pour des livraisons rapides et sûres.
- **Nginx Proxy Manager** : Gestion centralisée des domaines, des certificats SSL et du reverse proxy.
- **OVH VPS** : Hébergement flexible et sécurisé, adapté à la montée en charge.

5. Services externes

- **Brevo (ex-Sendinblue)** : Service d'envoi d'e-mails transactionnels fiable et simple à intégrer.

Ces choix technologiques assurent la robustesse, la sécurité et l'évolutivité de la plateforme, tout en facilitant la maintenance et l'intégration de nouvelles fonctionnalités.

IX. Conclusion et perspectives

1. Bilan du projet

Hexaplanning a permis de concevoir et de mettre en production une application web moderne, robuste et sécurisée, centrée sur l'expérience utilisateur et la gamification de la gestion de tâches. Le découpage clair entre frontend Angular et backend .NET, la modélisation soignée des entités (quêtes, utilisateurs, hexagones), ainsi que l'automatisation des tests et du déploiement, ont permis d'atteindre un haut niveau de qualité logicielle.

Les fonctionnalités principales sont opérationnelles : création et gestion de quêtes, affichage visuel sur carte hexagonale, authentification sécurisée, gestion des mots de passe, et notifications par email. L'architecture modulaire et la conteneurisation facilitent la maintenance et l'évolutivité.

2. Perspectives d'évolution

Les évolutions futures d'Hexaplanning s'articulent autour de plusieurs axes fonctionnels et techniques, en lien direct avec les besoins utilisateurs et la structure du code :

- **Sécurité et gestion des comptes**
    - Ajout d'un système de refresh token (stocké localement ou en cookies) pour renforcer la sécurité et la gestion de session.
    - Envoi d'un email de bienvenue et de confirmation à la création du compte.
    - Création d'un dashboard administrateur pour gérer les utilisateurs.
- **Liste de quêtes**
    - Ajout d'options de tri pour l'ordre d'affichage des quêtes : par date de création, par priorité ou personnalisé.
    - Ajout du drag & drop pour réorganiser les quêtes dans un ordre personnalisé.
    - Ajout de catégories personnalisables pour faciliter l'organisation.
    - Ajout d'un sélecteur de priorité directement depuis la liste.
    - Implémentation du glissement tactile sur mobile pour naviguer d'un menu à l'autre (quêtes à accomplir / quêtes accomplies).
- **Détails des quêtes**
    - Ajout d'une option pour rendre les quêtes répétables et permettre de les placer plusieurs fois sur la carte d'hexagones.
    - Ajout d'options dates pour organiser les quêtes dans le temps (date d'exécution prévue ou deadline).

    - Regroupement de quêtes en "expédition" avec un objectif final, chaque quête devenant une étape de la progression.

- **Carte d'hexagones**

    - Extension de la carte pour ajouter davantage de quêtes, voire extension automatique dès qu'un hexagone proche du bord est rempli.
    - Multiplication des cartes pour séparer les quêtes par catégorie.
    - Implémentations d'une navigation plus intuitive avec option de zoom et navigation à la souris ou au doigt.
    - Ajout de filtres pour masquer les quêtes par état (accomplies / non accomplies) et par priorité, pour permettre à l'utilisateur de se concentrer sur les tâches les plus urgentes sans être distrait par les suivantes, ou simplement de personnaliser son affichage.
    - Amélioration du système d'assignation des quêtes aux hexagones en permettant de déplacer une quête en drag & drop.
    - Ajout de flèches pour indiquer le sens de progression d’une quête à l’autre.
    - Ajout d’une mécanique de personnages se déployant sur la carte comme des soldats conquérant un territoire hexagonal après l'autre, ou d'un personnage seul progressant de façon linéaire jusqu'à un objectif.

- **Personnalisation**

    - Ajout d'un avatar pour l'utilisateur.
    - Personnalisation des couleurs (thème de l'application, texte des priorités dans les détails de quêtes, liseré des priorités dans la carte d'hexagones).
    - Personnalisation des unités déployables sur la carte (une fois faite l'implémentation d'un ou plusieurs personnages évoluant sur la carte).

- **Déploiement futur**

    - Création d'une application mobile en utilisant Ionic, et déploiement sur les stores Android et iOS.
    - Système de notifications.
    - Persistance des données utilisateurs en les stockant sur l'AsyncStorage de l'appareil afin d'éviter d'avoir à se reconnecter à chaque ouverture de l'app.

L’architecture actuelle, modulaire et évolutive, permet d’intégrer ces améliorations de façon progressive, tout en maintenant la stabilité et la sécurité de la plateforme.
