ChatGPT

Description détaillée d'Apprend + (branche `improved`)

Aperçu général

Apprend + est une application web développée en **Next.js 13**, **React 18**, **TypeScript**, **Tailwind CSS** et **Supabase**. Sa mission est d'accompagner les femmes dans un parcours de transformation personnelle et professionnelle structuré autour de la méthode **ACCEPTER**. L'utilisateur commence par une phase de **personnalisation** de son profil, suit ensuite un **programme de huit modules** où il consigne ses objectifs, croyances et actions, puis débloque un module de **renaissance** basé sur des jeux de mémoire pour ancrer les affirmations positives. Un **dashboard** centralise la progression et affiche des compétences clés (Confiance, Discipline et Action) sous forme de barres de progression. L'architecture repose sur des **hooks et services** réutilisables, un cache LRU pour limiter les appels réseau et des fonctions d'optimisation comme le **throttle** et le **debounce** [1] [2]. Les données utilisateur sont stockées sur Supabase dans des tables dédiées (profils, programmes, entrées du programme, axes de renaissance, tentatives de jeu, etc.) [3].

Mission et objectifs

L'application vise à aider chaque utilisatrice à devenir la femme qu'elle aspire à être. Pour cela, Apprend + propose :

- **Personnalisation du parcours :** un assistant en trois étapes recueille le prénom, l'année de naissance, la profession et, si elle le souhaite, le genre, le téléphone et le pays de l'utilisatrice. Les options sont validées et nettoyées (par exemple, l'année de naissance est limitée entre 13 et 100 ans) [4]. Les données sont sauvegardées dans Supabase via `UserProfileService.saveUserProfile` [5], puis un programme personnalisé est initialisé [6].
- **Programme structuré :** la méthode ACCEPTER est divisée en huit sous-parties (Ambitions, Caractère, Croyances, Emotions, Pensées, Travail, Environnement et Retention). Chaque sous-partie demande à l'utilisatrice de réfléchir et de saisir un certain nombre de phrases (objectifs, croyances à transformer, pensées à cultiver…). L'application impose des minimums et maximums d'entrées et suit la progression de manière granulaire.
- **Renaissance :** après avoir complété la totalité du programme (100 % de progression), l'utilisatrice débloque un module de renaissance. Elle choisit jusqu'à trois axes de développement (ex. Confiance, Discipline ou axes personnalisés) et s'exerce grâce à un **jeu flash** : des phrases positives sont affichées brièvement puis l'utilisatrice doit les retaper. L'algorithme compare la réponse en tolérant les erreurs mineures et calcule un score d'exactitude, le temps de réaction et d'autres statistiques [7]. Un tableau de bord Renaissance récapitule les statistiques et propose des conseils personnalisés selon l'exactitude obtenue [8] [9].
- **Dashboard central :** une page de tableau de bord affiche la progression globale, les compétences Confiance/Discipline/Action et les niveaux du parcours (Personnalisé, Programme, Renaissance et Évolution). Selon l'avancement, un message motivant est affiché ; par exemple, si l'utilisatrice

progresse bien, elle voit « Tu progresses bien ! Ta motivation et ta persévérance commencent à payer » [10] . Le passage au module Renaissance est protégé : il n'est accessible qu'une fois le programme terminé à 100 % [11] .

- **Performances et UX :** pour offrir une expérience fluide, le chargement des sous-parties du programme utilise un cache LRU et des opérations batched. Des indicateurs affichent les cache hits et le temps des dernières opérations [12] . Les actions de saisie sont **déclenchées via** `debounce` **et** `throttle` afin de réduire la charge réseau sans pénaliser l'utilisateur [13] . L'application logge les opérations qui dépassent un seuil et affiche des alertes si la durée est trop longue [14] .

Architecture et technologies

- **Front-end :** Next.js 13 avec le routage `app/` et des composants React fonctionnels. Tailwind CSS est utilisé pour le design responsive et le thème. Les icônes proviennent de Lucide et des emojis.
- **Supabase** : joue à la fois le rôle de base de données (PostgreSQL), d'authentification et de stockage. Le schéma comprend des tables pour les profils (`user_profiles`), les programmes (`user_programmes`), les entrées du programme (`programme_entries`), les axes de renaissance (`renaissance_axes`), les phrases (`renaissance_phrases`), les sessions de jeu (`renaissance_game_sessions`), les tentatives (`renaissance_attempts`) et les sélections d'axes (`user_renaissance_selection`) [3] [15] [16] . Des vues comme `renaissance_dashboard_data` calculent des statistiques agrégées par utilisateur [17] .
- **Services et hooks** :
- `UserProfileService` gère la sauvegarde et la récupération des informations personnelles [5] .
- `programmeSupabaseService` crée ou charge le programme de l'utilisateur, met à jour la progression globale et fournit les sous-parties.
- `useSubPartData` est un hook complet qui charge les données d'un sous-partie, met en cache les entrées, vérifie les permissions et expose des méthodes pour ajouter, modifier et supprimer des champs [18] . Il utilise un **cache intelligent** avec politique LRU et enregistre le temps des opérations pour optimiser la réactivité [1] .
- `renaissanceService` et `renaissanceSupabaseService` gèrent la logique de jeu : création et mise à jour des sessions, enregistrement des tentatives, calcul des statistiques et gestion des axes sélectionnés [19] [20] .

Parcours utilisateur

1 – Personnalisation

L'utilisatrice débute par un assistant en trois écrans :

Étape	Description	Validation
Bienvenue	Présente la personnalisation et invite à commencer	Un bouton permet de revenir au tableau de bord ou de lancer le questionnaire [21] .

Étape	Description	Validation
Informations personnelles	Formulaire qui demande : nom, année de naissance, profession, genre (optionnel), téléphone (optionnel), pays (optionnel). Les listes déroulantes pour la profession et le pays utilisent des options pré-définies (CSP et pays avec drapeaux) [22] [23] .	Les champs obligatoires sont validés : l'année de naissance doit avoir entre 13 et 100 ans [4] , et la profession et le nom ne peuvent être vides. Les erreurs sont affichées en dessous des champs [24] .
Récapitulatif et finalisation	Un écran récapitule les informations saisies et félicite l'utilisatrice ; au clic sur « Continuer », les données sont enregistrées dans Supabase et le programme est initialisé [6] . L'utilisatrice est ensuite redirigée vers le tableau de bord.	Si la sauvegarde échoue, un message d'erreur apparaît mais les données sont tout de même stockées localement avant redirection [25] .

2 – Programme (méthode ACCEPTER)

Le programme est le cœur de la méthode ACCEPTER. Il se compose de huit modules que l'utilisatrice doit compléter dans l'ordre. Chaque module demande de renseigner plusieurs phrases. Les propriétés de chaque sous-partie sont décrites dans le tableau ci-dessous :

ID	Slug	Nom	Description	Min	Max
1	ambitions	Ambitions	Clarifier ses objectifs et résultats désirés	3	10
2	caractere	Caractère	Identifier les traits de caractère à développer	3	10
3	croyances	Croyances	Lister les croyances limitantes à transformer	3	10
4	emotions	Emotions	Décrire les émotions qui freinent la progression	3	10
5	pensees	Pensées	Inscrire des pensées positives à cultiver	3	10
6	travail	Travail	Définir les actions concrètes à mettre en place	3	10
7	environnement	Environnement	Imaginer l'environnement idéal qui soutient la réussite	3	10
8	retention	Retention	Prévoir des rituels pour ancrer durablement le changement	3	10

Ces informations proviennent de la constante `SUBPARTS_CONFIG` du code ; chaque objet définit le `id`, le `slug`, le `name`, la `description`, la `minEntries` et la `maxEntries` [26] .

Sur chaque page de module :

- Un **header** rappelle le numéro du module, son nom et un fil d'Ariane vers le programme et le tableau de bord.
- Un composant `SubPartForm` permet d'ajouter un champ de texte. Le nombre de champs est limité par `minEntries` et `maxEntries` ; un message d'état informe l'utilisatrice si elle doit ajouter davantage de contenu ou si elle peut passer à la suite. Les requêtes pour sauvegarder chaque champ sont regroupées et optimisées grâce au hook `useSubPartData` qui utilise un cache et un batch process [18] .
- Les actions (ajout, modification, suppression) sont **throttlées** pour éviter les appels réseau excessifs [13] . Un bouton de sauvegarde envoie les modifications sur Supabase. Des messages d'erreur ou de succès apparaissent selon le résultat.
- La progression (en pourcentage) est recalculée automatiquement et stockée dans la table `user_programmes` [27] . Une fonction `recalculateAllModulesProgress` est exécutée toutes les 30 secondes pour maintenir un suivi en temps réel [28] .
- Lorsque l'utilisatrice satisfait le minimum d'entrées, elle peut accéder au module suivant. Un message motivationnel personnalisé est affiché selon son avancée globale ; par exemple, entre 60 % et 99 % de progression, le message est « Tu es à quelques pas d'être la femme qui atteint ses objectifs les plus ambitieux » [29] .

Une fois les huit modules complétés, un écran de **conclusion** félicite l'utilisatrice pour avoir suivi la méthode et rappelle les transformations accomplies : atteindre ses objectifs, transformer ses croyances, gérer ses émotions et créer un environnement propice [30] .

3 – Dashboard

La page `/dashboard` est le centre de commande de l'application. Elle comporte :

- **Niveaux de progression :** quatre cartes représentent les étapes du parcours : **PERSONNALISÉ**, **PROGRAMME**, **RENAISSANCE** et **ÉVOLUTION**. Chaque carte possède un gradient de couleur, une icône (Target, TrendingUp, Award, Smile) et un pourcentage de progression calculé en fonction de l'avancement moyen et des statistiques Supabase [31] . Les cartes sont cliquables ; la carte Renaissance reste grisée tant que le programme n'est pas achevé à 100 % [11] .
- **Compétences clés :** trois barres de progression représentent la **Confiance**, la **Discipline** et l'**Action**. Les valeurs initiales (ex. 85 % de confiance) sont des défauts mais peuvent être mises à jour dans l'avenir. Les barres sont animées et colorées selon la compétence [31] .
- **Message motivant :** un encart affiche une plage de progression (« 0 – 19 % », « 20 – 59 % », « 60 – 99 % », « 100 % ») et le message d'encouragement correspondant, avec un emoji. Cette logique est gérée par `getProgressMessage` [32] .
- **Actions rapides :** trois cartes proposent de continuer le programme, analyser ses progrès ou définir un objectif. Certaines fonctionnalités (« Analyser mes progrès ») sont encore marquées comme à venir.
- **Chargement des données :** lors du rendu, la page vérifie l'authentification avec Supabase, récupère le profil, charge ou crée le programme, puis charge les statistiques Renaissance si le programme est terminé [33] . Le dashboard stocke les informations en local pour une meilleure UX offline.

4 – Renaissance

Le module **Renaissance** est un atelier d'ancrage conçu pour transformer les pensées en habitudes profondes. Il se déroule ainsi :

1. **Sélection des axes :** l'utilisatrice choisit jusqu'à trois axes parmi une liste (Confiance, Discipline, Leadership…). Chaque axe a un nom, une icône, une description et peut être personnalisable. Une carte indique si l'axe est sélectionné, commencé ou verrouillé [34]. Si l'option « Personnalisable » est activée, l'utilisatrice peut créer un **axe personnalisé** via une modale ; elle définit un nom et entre 3 à 10 phrases positives. La modale valide la longueur des phrases, supprime les doublons et affiche des conseils (formuler à la première personne, éviter les négations) [35].
2. **Gestion des sessions** : lorsque l'utilisatrice démarre un axe, `renaissanceService.createGameSession` crée une nouvelle session dans Supabase avec l'ID de l'axe, la durée d'affichage des flashs et l'ordre aléatoire des phrases [36]. Le service désactive les sessions actives existantes et enregistre des informations sur le navigateur et le périphérique.
3. **Jeu flash** : pour chaque phrase, le jeu affiche la phrase pendant un temps déterminé (p. ex. 500 ms), puis la cache et invite l'utilisatrice à retaper la phrase. Le moteur de jeu (`flashGameEngine.ts`) normalise le texte (mise en minuscules, suppression des accents) et calcule la distance de Levenshtein pour permettre des erreurs mineures. Il enregistre la réponse, la durée et la similarité [7]. La progression est mise à jour et l'interface affiche la phrase suivante ou le résultat final.
4. **Résultats et statistiques** : à la fin d'une session, `renaissanceService.getSessionStats` calcule le nombre total de phrases, le nombre de réponses correctes, l'exactitude (%) et la durée totale [20]. Le composant `ResultDisplay` présente un message personnalisé selon le score (exceptionnel, très bon travail, bon effort, continuez à vous entraîner) [8]. Les statistiques comprennent l'exactitude, le temps total et le temps moyen, la liste des réponses incorrectes avec l'analyse des erreurs (mots manquants, incorrects ou en trop) et des conseils pour progresser [9] [37].
5. **Suivi à long terme** : Supabase conserve les tentatives (`renaissance_attempts`) et les sessions (`renaissance_game_sessions`) avec la précision, le temps et les analyses détaillées. Les statistiques globales sont exposées via des vues (`renaissance_dashboard_data`) qui calculent le nombre d'axes sélectionnés, le pourcentage d'axes complétés et la précision moyenne [17]. Ces données alimentent le tableau de bord Renaissance.

Authentification

Apprend + utilise Supabase Auth. Les pages d'authentification proposent :

- **Connexion/Inscription par e-mail et mot de passe** : un formulaire vérifie la validité de l'adresse e-mail et la longueur du mot de passe. Une erreur est renvoyée si l'utilisateur n'existe pas ou si les identifiants sont incorrects [38]. Lors de l'inscription, un e-mail de confirmation est envoyé.
- **Connexion via Google** : l'utilisateur peut s'identifier avec son compte Google via OAuth [39]. Un bouton déclenche `supabase.auth.signInWithOAuth` et redirige vers la page de dashboard en cas de succès.
- **Déconnexion** : le tableau de bord propose un menu qui appelle `supabase.auth.signOut` et efface les données du local storage [40].

Base de données (Supabase)

Les principales tables utilisées par Apprend + sont décrites ci-dessous. Seules les colonnes pertinentes pour la logique applicative sont listées :

Table	Rôle	Principales colonnes
user_profiles	Stocke les informations personnelles saisies lors de la personnalisation.	`user_id` (clé étrangère vers `auth.users`), `name`, `birth_year`, `profession`, `gender`, `phone`, `country`, `updated_at`.
user_programmes	Contient le programme de chaque utilisatrice (progrès général).	`user_id`, `subpart_statuses` (array indiquant l'état de chaque sous-partie), `overall_progress`, `last_updated`, etc.
programme_entries [3]	Regroupe les entrées saisies par les utilisatrices pour chaque module du programme.	`id` (UUID), `user_id`, `subpart_id`, `value` (texte), `created_at`, `updated_at`, `validation_status` (pending/valid/invalid), `word_count` (colonnes générées) [3].
renaissance_axes	Référence les axes disponibles (nom, description, icône, personnalisable) et l'ordre d'affichage.	`id`, `name`, `icon`, `description`, `is_active`, `is_customizable`.
renaissance_phrases	Liste les phrases associées à chaque axe.	`id`, `axe_id`, `phrase_number`, `content`.
user_renaissance_selection [16]	Stocke les axes choisis par l'utilisatrice et l'ordre de sélection.	`id`, `user_id`, `axe_id`, `custom_name`, `custom_phrases` (JSON), `selection_order`, `is_started`, `is_completed`, `selected_at`, `started_at`, `completed_at`.
renaissance_game_sessions	Enregistre chaque session de jeu (stage, ordre des phrases, durée flash) [41].	`id`, `user_id`, `axe_id`, `stage` (`discovery`, `level1`, `level2`, `level3`), `flash_duration_ms`, `phrases_order`, `current_phrase_index`, `is_active`, `is_completed`, `correct_count`, `total_attempts`, `session_accuracy`, `started_at`, `completed_at`.

Table	Rôle	Principales colonnes
renaissance_attempts [15]	Log toutes les réponses d'un utilisateur lors d'une session.	`id`, `session_id`, `phrase_id`, `phrase_number`, `user_input`, `expected_text`, `is_correct`, `response_time_ms`, `input_length`, `similarity_score`, `error_analysis` (JSON), `submitted_at`.
user_renaissance_progress [42]	Suit la progression d'un utilisateur sur un axe et un stage particuliers.	`id`, `user_id`, `axe_id`, `stage`, `current_phrase`, `attempts` (JSON), `stage_completed`, `stage_completed_at`, `last_attempt_at`.

Ces tables sont complétées par les vues `renaissance_dashboard_data` et `renaissance_dashboard_summary` qui agrègent les données par utilisateur pour le tableau de bord Renaissance [17]. Des triggers mettent à jour automatiquement `updated_at` lors de modifications et des politiques de sécurité limitent l'accès aux données à l'utilisateur connecté.

Performances et optimisation

Apprend + accorde une grande importance à la réactivité :

- **Cache intelligent** : un cache local de type LRU stocke les données des sous-parties pour éviter de refaire les mêmes requêtes. Lorsqu'une entrée est demandée, le hook `useSubPartData` vérifie si elle se trouve dans le cache et la renvoie immédiatement ; sinon, il la charge depuis Supabase et met à jour le cache [18]. Des statistiques de cache (hits, misses) sont affichées sur l'interface pour le débogage [12].
- **Débounce et throttle** : les actions de modification de champ sont regroupées (debounce) pour ne pas envoyer de requêtes à chaque frappe. Les appels sont aussi limités en fréquence (throttle) afin d'éviter les surcharges réseau [13].
- **Performance monitoring** : un utilitaire log les opérations qui durent plus de 200 ms et affiche des avertissements dans la console [14]. Cela incite les développeurs à maintenir un temps de réponse faible.
- **Recalcul périodique** : la progression globale du programme est recalculée automatiquement toutes les 30 secondes pour que le tableau de bord soit toujours à jour [28].

Conception et design

L'interface mêle sobriété et modernité : palette de couleurs pastel (violet, rose, turquoise, jaune), utilisation d'icônes Lucide et d'emojis pour renforcer l'engagement, barres de progression linéaires et circulaires pour représenter l'avancement [43]. Les pages utilisent les polices **Inter** et **JetBrains Mono** et sont organisées en sections claires. Un fil d'Ariane permet de revenir à l'étape précédente. L'ensemble est entièrement **responsive** et s'adapte à différents appareils.

Perspectives et évolutions

La branche `improved` met l'accent sur la structure du parcours, la personnalisation et l'ancrage via le jeu flash. Le niveau **ÉVOLUTION**, présent dans le dashboard, est pour l'instant un placeholder et représente les futures fonctionnalités (coaching avancé, défis communautaires, analyse fine des progrès). Certaines actions, comme « Analyser mes progrès », sont également marquées comme **à venir** et pourront intégrer des statistiques plus approfondies.

Conclusion

Apprend + propose une expérience complète mêlant introspection, planification et entraînement cognitif. En combinant un programme structuré de réflexion, un tableau de bord ludique et un atelier d'ancrage, l'application accompagne l'utilisatrice sur la durée et lui fournit des outils pour mesurer et améliorer sa progression. L'architecture modulaire et les optimisations intégrées (cache, debounce, throttle) assurent une navigation fluide, tandis que Supabase centralise l'authentification, la base de données et les fonctions de serveur. Cette description peut être intégrée au README pour présenter en détail le fonctionnement et les choix techniques de la branche `improved`.

---

1 2 14 performanceUtils.ts
https://github.com/Kutoh07/apprend-final/blob/improved/src/lib/utils/performanceUtils.ts

3 15 16 17 42 schema.sql
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/supabase/schema.sql

4 24 PersonalInfoStep.tsx
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/components/personalisation/PersonalInfoStep.tsx

5 userProfileService.ts
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/lib/userProfileService.ts

6 25 SuccessStep.tsx
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/components/personalisation/SuccessStep.tsx

7 flashGameEngine.ts
https://github.com/Kutoh07/apprend-final/blob/improved/src/lib/utils/flashGameEngine.ts

8 9 37 ResultDisplay.tsx
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/app/renaissance/components/ResultDisplay.tsx

10 11 31 32 33 40 page.tsx
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/app/dashboard/page.tsx

12 SubPartPage.tsx
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/app/programme/components/SubPartTemplate/SubPartPage.tsx

13 18 useSubPartData.ts
https://github.com/Kutoh07/apprend-final/blob/improved/src/hooks/useSubPartData.ts

19 20 36 renaissanceService.ts
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/lib/services/renaissanceService.ts

21 WelcomeStep.tsx
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/components/personalisation/WelcomeStep.tsx

22 23 constants.tsx
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/lib/constants.tsx

26 programme.ts
https://github.com/Kutoh07/apprend-final/blob/improved/src/lib/types/programme.ts

27 28 29 page.tsx
https://github.com/Kutoh07/apprend-final/blob/improved/src/app/programme/page.tsx

30 page.tsx
https://github.com/Kutoh07/apprend-final/blob/improved/src/app/programme/conclusion/page.tsx

34 AxeSelectionCard.tsx
https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/app/renaissance/components/AxeSelectionCard.tsx

35 CustomAxeModal.tsx

https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/app/renaissance/components/CustomAxeModal.tsx

38 39 page.tsx

https://github.com/Kutoh07/apprend-final/blob/improved/src/app/auth/page.tsx

41 renaissance.ts

https://github.com/Kutoh07/apprend-final/blob/improved/src/lib/types/renaissance.ts

43 ProgressBar.tsx

https://github.com/Kutoh07/apprend-final/blob/122f962c17157f6748502c63606024199a685f09/src/app/renaissance/components/ProgressBar.tsx