📋 Consignes d'Évaluation - Projet Tamagotcho

Date de remise : Jeudi 13 Novembre 2025 à 23h59

🎯 Objectif

Vous devez implémenter **4 features principales** pour finaliser et enrichir l'application Tamagotcho. Chaque feature sera évaluée selon les critères de fonctionnalité, de qualité du code, et de respect des principes d'architecture (SOLID, Clean Code).

📦 Feature 1 : Système d'Accessoires et Arrière-plans

1.1 Accessoires

Objectif : Implémenter un système complet permettant d'acheter, équiper et retirer des accessoires sur les monstres. Les accessoires doivent être dessinés dans le visuel du monstre.

Fonctionnalités requises :

✅ **3 catégories d'accessoires** :

- Chapeaux (hats) : 🎩🪖🧢
- Lunettes (glasses) : 😎🤓🕶️
- Chaussures (shoes) : 👟🥾🩰

✅ **Actions disponibles** :

- Acheter un accessoire depuis la boutique (débit en Koins)
- Équiper un accessoire sur un monstre
- Retirer un accessoire d'un monstre
- Visualiser les accessoires possédés

✅ **Affichage** :

- **Dans le détail du monstre** (`/app/creatures/[id]`) : Visualisation sur les monstres
- **Dans la liste du dashboard** (`/app`) : Visualisation sur les monstres

✅ **BONUS - Système de rareté** :

- Implémenter 4 niveaux de rareté (Commun, Rare, Épique, Légendaire)
- Affichage visuel avec badges colorés
- Prix ajustés selon la rareté

Ressources disponibles :

Un document de spécification détaillée existe dans `docs/specs/ACCESSORIES_BACKGROUNDS_SYSTEM.md` qui contient :

- Architecture complète
- Structure de base de données
- Types TypeScript
- Exemples de composants
- Configuration du catalogue

Vous devez adapter et implémenter cette spécification selon vos besoins.

1.2 Arrière-plans

Objectif : Permettre l'achat et l'application d'arrière-plans spéciaux depuis la boutique.

Fonctionnalités requises :

✅ **Achat d'arrière-plans** :

- Catalogue d'arrière-plans avec différents thèmes (gradient, pattern, animé, etc.)
- Achat depuis la boutique avec Koins
- Gestion de la possession (un utilisateur peut posséder plusieurs arrière-plans)

✅ **Application sur un monstre** :

- Sélection et application d'un arrière-plan acheté

- Retrait/réinitialisation à l'arrière-plan par défaut
- Affichage visuel dans le détail et la liste

✅ **Affichage** :

- Prévisualisation dans la boutique
- Aperçu avant application
- Affichage réel sur le monstre

🔧 Feature 2 : Finalisation de la Base

2.1 Redirections

✅ **Routes à finaliser** :

- `/` → Redirection intelligente :
  - Si non connecté → Reste sur `/` (landing page)
  - Si connecté → Redirection vers `/app` (dashboard)
- `/sign-in`, `/sign-up` → Redirection après connexion réussie vers `/app`
- Routes protégées → Redirection vers `/sign-in` si non authentifié

✅ **Navigation** :

- Vérifier que toutes les redirections fonctionnent correctement
- Gérer les cas d'erreur (session expirée, etc.)

2.2 Personnalisation Stripe

✅ **Page de paiement Stripe** :

- Personnaliser le branding Stripe Checkout avec le thème de l'application
- Ajouter le logo de l'application
- Personnaliser les couleurs et le style
- Messages personnalisés sur la page de paiement

Documentation Stripe : Stripe Checkout Customization

2.3 Design de l'Application

✅ **Ajustements de design** :

- Cohérence visuelle sur toutes les pages
- Palette de couleurs harmonieuse
- Typographie cohérente
- Espacements et marges réguliers
- Animations et transitions fluides

Vous êtes libres de choisir votre direction artistique mais gardez :

- L'esprit "kawaii" et fun du Tamagotcho
- La lisibilité et l'accessibilité
- Le responsive design (mobile first)

2.4 Gains de Koins pour les Actions

✅ **Système de récompenses** :

- Chaque action sur un monstre rapporte des Koins :
  - Nourrir le monstre → +X Koins
  - Jouer avec le monstre → +X Koins
  - Soigner le monstre → +X Koins
  - Etc.
- Affichage d'un message de gain (toast/notification)
- Mise à jour immédiate du solde de Koins

✅ **Configuration** :

- Extraire les montants de récompenses dans un fichier de configuration (`src/config/rewards.ts` ou similaire)

2.5 Extraction des Configurations

✅ **Fichiers de configuration à créer/structurer** :

- `src/config/rewards.ts` → Montants de Koins pour chaque action
- `src/config/accessories.config.ts` → Catalogue d'accessoires (si pas déjà fait)
- `src/config/backgrounds.config.ts` → Catalogue d'arrière-plans (si pas déjà fait)
- `src/config/quests.config.ts` → Configuration des quêtes journalières

- Tout autre fichier de configuration manquant

Principe : Tous les valeurs magiques (nombres, textes, etc.) doivent être extraites dans des fichiers de configuration.

2.6 Connexions Tierces

✅ **Obligatoire - GitHub OAuth** :

- Intégrer la connexion via GitHub (Better Auth supporte déjà GitHub, vérifier la configuration)
- Bouton "Se connecter avec GitHub" dans la page de connexion
- Gérer les redirections après OAuth

✅ **BONUS - Google OAuth** :

- Ajouter la connexion via Google
- Suivre le même pattern que GitHub

Documentation : Voir `docs/authentication/auth-system.md` et la configuration dans `src/lib/auth.ts`

2.7 Optimisation de la Base de Code

✅ **Plan d'optimisation** :

- Créer un document (`OPTIMIZATION_PLAN.md`) listant :
  - Les composants à optimiser avec `useMemo`
  - Les fonctions à mémoriser avec `useCallback`
  - Les données à mettre en cache
  - Les chargements à optimiser (lazy loading, code splitting)

✅ **Implémentation** :

- Appliquer au moins 5 optimisations concrètes :
  - Utiliser `useMemo` pour les calculs coûteux
  - Utiliser `useCallback` pour les fonctions passées en props
  - Optimiser les re-renders inutiles
  - Implémenter le lazy loading où approprié
  - Optimiser les requêtes DB avec des index si nécessaire

Documentation : Décrire chaque optimisation dans le plan.

📸 Feature 3 : Système de Galerie

3.1 Mode Public des Monstres

✅ **Fonctionnalités** :

- Ajouter un champ `isPublic: boolean` au modèle Monster
- Interface pour activer/désactiver le mode public d'un monstre
- Toggle dans le détail du monstre pour rendre public/privé
- Indicateur visuel que le monstre est public

3.2 Page Galerie Communautaire

✅ **Page dédiée** : `/app/gallery` ou `/gallery`

Fonctionnalités :

- Affichage de tous les monstres publics de tous les utilisateurs
- Design inspiré d'une galerie d'art/photos
- Filtres possibles :
  - Par niveau
  - Par humeur/état
  - Par date de création
- Pagination ou scroll infini
- Affichage du nom du créateur (anonymisé ou avec username)

✅ **Affichage** :

- Grille ou liste de cartes de monstres
- Preview des accessoires et arrière-plans
- Informations essentielles (niveau, nom, créateur)

3.3 Respect de la Vie Privée

✅ **Gestion des données** :

- Un utilisateur peut rendre son monstre privé à tout moment

- Les monstres privés n'apparaissent jamais dans la galerie
- Gestion des permissions et de la visibilité

🎮 Feature 4 : Système de Quêtes Journalières

4.1 Quêtes du Jour

✅ **Fonctionnalités** :

- **3 quêtes journalières** uniques par utilisateur
- Renouvellement automatique à minuit (heure serveur ou locale)
- Chaque quête rapporte des Koins quand complétée
- Système de progression pour suivre l'avancement

4.2 Types de Quêtes

Exemples de quêtes :

- "Nourris 5 fois ton monstre aujourd'hui" → +20 Koins
- "Fais évoluer un monstre d'un niveau" → +50 Koins
- "Interagis avec 3 monstres différents" → +30 Koins
- "Achète un accessoire dans la boutique" → +40 Koins
- "Rends un monstre public" → +15 Koins

✅ **Système flexible** :

- Configuration centralisée des quêtes dans `src/config/quests.config.ts`
- Types de quêtes extensibles
- Suivi de progression en temps réel

4.3 Renouvellement à Minuit

✅ **Mécanisme de renouvellement** :

- Utiliser un système de cron job ou scheduler
- Réinitialiser les quêtes à 00:00 (minuit)
- Option 1 : Utiliser le système de cron existant (`CRON_SUMMARY.md`)
- Option 2 : Utiliser Vercel Cron Jobs (si déployé sur Vercel)

- Option 3 : Utiliser un hook/scheduler côté client avec vérification serveur

✅ **Base de données** :

- Collection `daily_quests` ou champs dans la collection `users`
- Stocker :
  - Date du jour
  - Quêtes actives
  - Progression de chaque quête
  - Quêtes complétées

4.4 Interface Utilisateur

✅ **Affichage** :

- Section des quêtes dans le dashboard (peut remplacer/enrichir la section existante)
- Progress bars pour chaque quête
- Badges "Complété" ✅
- Animation lors de la complétion
- Notification de gain de Koins

✅ Checklist Globale

Avant de soumettre votre travail, vérifiez :

Fonctionnalités

- ☐ Toutes les features sont implémentées
- ☐ Pas de fonctionnalités cassées
- ☐ Les erreurs sont gérées proprement
- ☐ Les messages d'erreur sont clairs

Code

- ☐ Code commenté et documenté
- ☐ Types TypeScript corrects

- [ ] Pas de `any` non justifiés
- [ ] Respect des principes SOLID
- [ ] Code réutilisable et modulaire

Base de données

- [ ] Schémas MongoDB cohérents
- [ ] Index sur les champs fréquemment utilisés
- [ ] Migrations claires

UI/UX

- [ ] Design cohérent
- [ ] Responsive (mobile + desktop)
- [ ] Animations fluides
- [ ] Feedback utilisateur (toasts, loaders)

Tests

- [ ] Application testée manuellement
- [ ] Cas limites testés
- [ ] Gestion d'erreurs testée

📝 Livrables

1. Code

- Pull Request sur le repository ou code livré
- Code propre et bien organisé
- Commits avec des messages clairs

2. Documentation

- Documenter les nouvelles features dans `/docs` ou à la racine
- README mis à jour si nécessaire
- `OPTIMIZATION_PLAN.md` pour la Feature 2.7

3. Explication

- Un court document (`IMPLEMENTATION_NOTES.md`) expliquant :
  - Vos choix d'implémentation
  - Les difficultés rencontrées
  - Les optimisations appliquées
  - Les améliorations futures possibles

🎓 Critères d'Évaluation

Fonctionnalité (40%)

- ✅ Fonctionnalités implémentées et opérationnelles
- ✅ Cas limites gérés
- ✅ Pas de bugs majeurs

Qualité du Code (30%)

- ✅ Architecture respectée (SOLID)
- ✅ Code propre et lisible
- ✅ Réutilisabilité
- ✅ Gestion d'erreurs

Design & UX (20%)

- ✅ Interface cohérente et agréable
- ✅ Responsive design
- ✅ Feedback utilisateur

Bonus & Extras (10%)

- ✅ Système de rareté
- ✅ Connexion Google
- ✅ Optimisations avancées
- ✅ Fonctionnalités bonus personnalisées

💡 Conseils

1. **Commencez par comprendre l'architecture existante**
   - Lisez `ARCHITECTURE.md`
   - Explorez les composants existants
   - Comprenez le système de wallet et de boutique
2. **Planifiez votre travail**
   - Priorisez les features
   - Estimez le temps nécessaire
   - Répartissez les tâches si en équipe
3. **Testez régulièrement**
   - Ne laissez pas les bugs s'accumuler
   - Testez sur mobile et desktop
   - Vérifiez les cas limites
4. **Documentez au fur et à mesure**
   - N'attendez pas la fin pour documenter
   - Commentez votre code
   - Notez vos décisions importantes
5. **Demandez de l'aide si bloqué**
   - Utilisez les ressources disponibles
   - Consultez la documentation
   - Posez des questions

📚 Ressources Utiles

- **Documentation du projet** : `/docs` et `/documentation`
- **Spécifications accessoires** : `docs/specs/ACCESSORIES_BACKGROUNDS_SYSTEM.md`
- **Système Wallet** : `docs/WALLET_SYSTEM.md` et `docs/WALLET_SHOP_SYSTEM.md`
- **Authentification** : `docs/authentication/auth-system.md`

- **Cron Jobs** : `CRON_SUMMARY.md` et `docs/CRON_SYSTEM.md`
- **Next.js** : Documentation officielle
- **MongoDB** : Documentation Mongoose
- **Stripe** : Documentation Stripe
- **Better Auth** : Documentation Better Auth

Bonne chance et bon développement ! 🚀

Document créé le : [Date actuelle]

Dernière mise à jour : [Date actuelle]

Deadline : Jeudi 13 Novembre 2025 à 23h59