Cahier des charges : Youcus

Sommaire

---

1. Présentation du projet

1.1 Contexte

YouTube est devenu une plateforme d'apprentissage majeure : cours universitaires, tutoriels techniques, conférences, MOOCs informels. Selon Pew Research, plus de la moitié des utilisateurs de YouTube s'en servent à des fins éducatives.

Cependant, son interface est conçue pour maximiser le temps de visionnage : recommandations latérales, suggestions de fin de vidéo, shorts, commentaires, notifications. Pour un apprenant qui utilise YouTube comme une bibliothèque éducative, ces éléments constituent autant de distractions qui rompent la concentration et réduisent la qualité de l'apprentissage.

1.2 Problématique

Comment offrir à un apprenant autodidacte un environnement épuré pour consulter ses playlists YouTube éducatives, suivre sa progression et prendre des notes structurées, sans subir les mécanismes d'engagement de la plateforme native, tout en restant strictement conforme aux Conditions Générales d'Utilisation de YouTube ?

1.3 Objectifs fonctionnels

- Permettre l'authentification sécurisée via Google avec autorisation YouTube.
- Importer les playlists YouTube de l'utilisateur via l'API officielle.
- Offrir un lecteur intégré dans un environnement sans distraction.
- Suivre la progression par playlist et par vidéo.
- Permettre la prise de notes texte attachées aux vidéos.
- Permettre l'export des notes dans des formats réutilisables.

1.4 Objectifs techniques

- Démontrer la maîtrise d'une stack JavaScript/TypeScript complète.
- Mettre en œuvre une architecture en couches propre et testable.
- Intégrer plusieurs API tierces (OAuth, REST, IFrame).
- Conteneuriser l'application pour reproductibilité et déploiement.
- Couvrir les services critiques par des tests automatisés.

1.5 Public cible

- **Cible primaire** : étudiants supérieurs autodidactes, développeurs en formation continue, professionnels en reconversion.
- **Cible secondaire** : formateurs curant des playlists pour leurs apprenants, parents structurant le visionnage éducatif de leurs enfants.

2. Périmètre fonctionnel

2.1 Fonctionnalités MVP (semaines 2 à 5)

#	Fonctionnalité	Priorité
F1	Authentification Google OAuth 2.0 avec scope YouTube readonly	MUST
F2	Déconnexion et suppression de compte (RGPD)	MUST
F3	Import de playlists depuis le compte YouTube	MUST
F4	Tableau de bord listant les playlists importées	MUST
F5	Vue détail d'une playlist avec liste des vidéos	MUST
F6	Lecteur YouTube intégré en mode focus	MUST
F7	Marquage manuel d'une vidéo comme vue	MUST
F8	Calcul automatique de la progression de playlist	MUST
F9	Prise de notes texte par vidéo avec sauvegarde automatique	MUST
F10	Suppression d'une playlist	MUST
F16	Prise d'une note attachée à une playlist (objectifs, conclusions, prérequis)	MUST

2.2 Fonctionnalités bonus (semaines 6-7)

#	Fonctionnalité	Priorité
F11	Notes timestampées avec navigation cliquable	SHOULD
F12	Synchronisation d'une playlist déjà importée	SHOULD
F13	Export PDF des notes d'une playlist	COULD
F14	Export Markdown des notes d'une playlist	COULD
F15	Tableau de bord d'apprentissage avec statistiques	COULD

2.3 Hors périmètre

- Téléchargement de vidéos (interdit par les CGU YouTube).
- Application mobile native.
- Multi-utilisateurs / partage de playlists.
- Résumés ou quiz générés par IA.
- Mode déblocage séquentiel de vidéos.

Ces éléments seront mentionnés dans la roadmap produit mais explicitement exclus du MVP.

3. Exigences techniques

3.1 Architecture générale

Architecture client-serveur séparée :

- **Frontend** : Single Page Application React, build statique servi par Nginx.
- **Backend** : API REST Express en architecture en couches (routes → controllers → services → repositories).
- **Persistance** : MySQL 8 via Prisma ORM.
- **Communication** : HTTPS uniquement, JSON, authentification par cookie de session HttpOnly.

3.2 Stack technique imposée

Frontend - React 18 + TypeScript - Vite (build tool) - React Router v6 (routing) - TanStack Query (cache et data fetching) - Tailwind CSS v3 (styling) - YouTube IFrame Player API (lecture)

Backend - Node.js 20 + TypeScript - Express 4 - Prisma ORM - Zod (validation des entrées) - Passport.js (stratégie Google OAuth 2.0) - Pino (logging structuré) - Jest + Supertest (tests unitaires et d'intégration)

Données - MySQL 8

Infrastructure - Docker + Docker Compose - Conteneurs : `frontend` (Nginx + build), `backend` (Node), `db` (MySQL) - Volumes persistants pour MySQL - Variables d'environnement via `.env` non versionné

3.3 Exigences de sécurité

- Communication HTTPS systématique en production.
- Refresh tokens Google chiffrés en base (AES-256-GCM).
- Sessions par cookies HttpOnly, SameSite=Lax, Secure.
- Protection CSRF sur toutes les routes mutatives.
- Validation systématique des entrées via Zod.
- Rate limiting sur les endpoints d'authentification et d'import.
- Headers de sécurité via Helmet (CSP, HSTS, X-Frame-Options).
- Secrets jamais commités, gérés via variables d'environnement.
- Logs sans données personnelles sensibles (pas de tokens, emails masqués).

3.4 Conformité

RGPD - Politique de confidentialité accessible. - Consentement explicite au traitement lors de l'inscription. - Droit à l'effacement : suppression complète du compte et de toutes les données associées en une action. - Droit à la portabilité : export JSON de toutes les données utilisateur. - Conservation limitée : suppression automatique des comptes inactifs > 24 mois (mécanisme prévu post-MVP).

CGU YouTube - Aucun téléchargement, aucune extraction de flux vidéo. - Lecture exclusive via le YouTube IFrame Player officiel. - Affichage des attributions conformes aux YouTube Branding Guidelines (titre, chaîne, lien vers la vidéo source). - Quotas API respectés via mise en cache côté serveur des métadonnées.

3.5 Exigences de performance

En production

Exigence	Cible	Méthode de vérification
Temps de réponse API P95	< 300 ms (hors appels YouTube)	Sentry Performance
Time to Interactive (TTI) front	< 3 s sur connexion 4G	Lighthouse / WebPageTest
Bundle front initial	< 250 ko gzippé	Vite bundle analyzer en intégration continue (CI)
Time to First Byte (TTFB)	< 200 ms	Monitoring synthétique
Pagination des listes	50 éléments par page	Implémenté côté API et UI
Cache des métadonnées YouTube	TTL adapté (immutable pour ID vidéo, 24 h pour métadonnées)	Configuration Redis ou cache mémoire applicatif

En développement

Exigence	Cible	Méthode de vérification
Démarrage en local via Docker Compose	< 30 s	`time docker compose up`
Hot reload front (Vite)	< 1 s	Mesure manuelle
Durée du build CI complet	< 5 min	Logs GitHub Actions
Délai entre push et déploiement staging	< 10 min	Pipeline CI/CD

3.6 Exigences de qualité

Qualité du code

Exigence	Cible	Méthode de vérification
Couverture de tests backend (services critiques)	> 70 %	Coverage report Jest
Couverture de tests frontend (hooks, utilitaires)	> 50 %	Coverage report Vitest
Tests d’intégration sur routes critiques	Auth, import, notes, suppression compte	Supertest
Lint backend et frontend	ESLint + Prettier sans erreur	CI check obligatoire
TypeScript strict	`strict: true` sans any non justifié	`tsc --noEmit` en CI
Hook pre-commit	Lint + format + type-check	Husky + lint-staged

Qualité du processus de développement

Exigence	Cible	Méthode de vérification
Conventions de commit	Conventional Commits	Hook `commitlint`
Conventions de branches	`feat/`, `fix/`, `chore/`, `docs/`	Manuel + linter optionnel
Pull Request mergeables sans rework	> 80 %	Statistiques GitHub
Taux d'échec build CI	< 10 %	Logs GitHub Actions sur 7 jours
Documentation OpenAPI / Swagger	Générée à partir du code, à jour	`tsoa` ou annotations
Architecture Decision Records (ADR) à jour	Une ADR par décision structurante	Dossier `docs/adr/`

4. Contraintes

Contrainte	Détail
Durée	8 semaines au total (7 développement + 1 soutenance)
Budget	Personnel, hébergement cible < 20 €/mois
Équipe	Développeur unique
Légal	Respect strict CGU YouTube et RGPD
Technique	Stack imposée ci-dessus, pas de bibliothèque non libre

5. Livrables attendus

1. **Code source** versionné sur GitHub (deux dépôts : `youcus-api`, `youcus-web`).
2. **Documentation technique** :
   - README d'installation par dépôt
   - Documentation API (Swagger ou Postman collection)
   - Décisions d'architecture (ADR)
3. **Dossier de conception** (le présent document + business plan + user stories + diagrammes).
4. **Application déployée** accessible publiquement à une URL stable.
5. **Présentation de soutenance** (slides + démonstration live).

6. Planning prévisionnel

Le planning est conçu sur 8 semaines (7 développement + 1 soutenance). Trois principes guident sa structure : un **MVP testable dès la semaine 4** pour valider la trajectoire à mi-parcours et activer le plan B si nécessaire, une **marge de 20 % par sprint** absorbée dans le périmètre des semaines bonus et de polish, et un **feature freeze à partir de la semaine 7** pour garantir la stabilité de la livraison finale.

6.1 Découpage hebdomadaire

Semaine	Phase	Livrables clés	Critère de validation
S1	Conception	Dossier validé (business plan, cahier des charges, user stories, diagrammes), schéma Prisma, dépôts initialisés, Docker Compose dev fonctionnel	Tous les documents validés + `docker compose up` opérationnel
S2	Auth + Données	OAuth Google opérationnel, modèle de données déployé, sessions sécurisées (cookies HttpOnly + CSRF)	Parcours login/logout end-to-end + tests d'intégration sur l'authentification
S3	Import YouTube	Récupération playlists via API officielle, persistance en base, vue dashboard	Import d'une vraie playlist + vérification des quotas API
S4	Lecteur + Progression	Lecteur focus (IFrame API), marquage vidéos vues, calcul de progression playlist	**MVP testable de bout en bout** : connexion + import + visionnage + suivi
S5	Notes + Tests	CRUD notes textuelles, sauvegarde automatique, couverture de tests > 70 % backend / > 50 % frontend	Tests automatisés verts en intégration continue + démonstration des notes fonctionnelle
S6	Bonus + Polish	Notes timestampées, export PDF/Markdown selon temps disponible, polish d'interface	Au moins une fonctionnalité SHOULD livrée + lint et types verts
S7	Déploiement + Feature freeze	Mise en production sur serveur Hetzner, monitoring Sentry, certificats HTTPS, smoke tests	URL publique stable + headers de sécurité grade A
S8	Dossier + Soutenance	Finalisation des quatre documents du dossier, slides, trois répétitions de soutenance	Dossier complet remis + soutenance prête

6.2 Jalons critiques

- **Fin S1** : dossier de conception validé → autorisation de démarrer le développement
- **Fin S4** : MVP testable de bout en bout → décision go/no-go pour les fonctionnalités bonus
- **Fin S7** : feature freeze et déploiement → aucune nouvelle fonctionnalité après ce point
- **S8** : finalisation administrative et soutenance

6.3 Plan B en cas de retard

Si un retard supérieur à une semaine est constaté avant la fin de S4 : - Descente de fonctionnalités SHOULD en COULD (notes timestampées, export PDF, statistiques) - Concentration sur les dix fonctionnalités MUST exclusivement - S6 et S7 fusionnent en une seule semaine de finition + déploiement

Cette flexibilité garantit que le MVP minimum est livré quoi qu'il arrive, conformément à la mitigation du risque personnel décrit en section 8.4.

7. Critères de réussite

Les critères de réussite du projet portent à la fois sur les **livrables** (le MVP est-il fonctionnel ?), la **qualité technique** (le code respecte-t-il les standards visés ?), la **conformité** (le service respecte-t-il ses engagements légaux ?) et la **validation pédagogique** (le titre RNCP est-il obtenu ?). Ces critères sont mesurables à la fin du projet et constituent les seuils minimaux ; les objectifs de pilotage plus larges (rétention, NPS, performance en production) sont détaillés dans le business plan, section 9.

7.1 Critères fonctionnels (production)

Critère	Mesure	Seuil minimal
Fonctionnalités MVP livrées	Comptage des 10 MUST en production	100 %
Application déployée	URL publique accessible et stable	OK ou KO
Authentification fonctionnelle	Parcours OAuth Google complet sans erreur	OK ou KO
Suppression de compte RGPD	Suppression effective des données en une action	OK ou KO

7.2 Critères techniques (qualité du code)

Critère	Mesure	Seuil minimal
Couverture de tests backend	Coverage report Jest sur services critiques	> 70 %
Couverture de tests frontend	Coverage report Vitest sur hooks + utilitaires	> 50 %
Lint backend et frontend	ESLint + Prettier sans erreur en CI	100 %
TypeScript strict	`strict: true` sans any non justifié	OK ou KO
Vulnérabilités hautes en dépendances	`npm audit --audit-level=high` en CI	0

7.3 Critères techniques (production)

Critère	Mesure	Seuil minimal
Temps de réponse API P95	Sentry Performance sur 7 jours	< 300 ms
Time to Interactive (TTI) front	Lighthouse	< 3 s sur 4G
Bundle front initial	Taille gzippée	< 250 ko
Uptime sur la dernière semaine du projet	UptimeRobot / Better Stack	> 99 %

7.4 Critères de documentation

Critère	Mesure	Seuil minimal
README technique	Installation d'un repo depuis zéro par un tiers	< 10 min
Documentation API	Swagger / OpenAPI à jour, accessible	OK ou KO
Dossier de conception	4 documents complets (business plan, cahier des charges, user stories, diagrammes)	Tous validés
ADR (Architecture Decision Records)	Au moins 3 décisions documentées	≥ 3

7.5 Critères de conformité

Critère	Mesure	Seuil minimal
Conformité CGU YouTube	Audit visuel : pas de téléchargement, attributions présentes	OK ou KO
Conformité RGPD	Politique de confidentialité, consentement, droits utilisateur effectifs	OK ou KO
Headers de sécurité	Score SecurityHeaders.com	A ou supérieur
HTTPS production	Certificat valide, redirection HTTP → HTTPS	OK ou KO

7.6 Critère pédagogique

Critère	Mesure	Seuil minimal
Validation du titre RNCP CDA	Décision du jury	Obtenu

8. Risques identifiés et mitigations

Cette section identifie les risques pesant sur la réalisation du projet et expose les mesures de mitigation prévues. Les risques sont regroupés en quatre catégories, conformément à l'analyse présentée dans le business plan (section 7), à laquelle le présent document se réfère pour le détail argumentaire.

8.1 Risques techniques et produit

Changement des CGU YouTube ou évolutions de l'API. YouTube peut modifier ses Conditions Générales d'Utilisation ou restreindre les fonctionnalités de l'API Data v3 ou du player IFrame. Mitigation : veille active sur les release notes officielles, architecture en ports & adapters isolant la couche d'accès YouTube, permettant un pivot vers une autre source (Vimeo, PeerTube) si nécessaire.

Dépassement des quotas API YouTube. L'API Data v3 alloue 10 000 unités/jour. Mitigation : cache agressif des métadonnées en base, traitement asynchrone via file d'attente, pagination intelligente, demande de quota étendu auprès de Google si nécessaire.

Évolutions du player IFrame. Le YouTube IFrame Player API peut changer. Mitigation : suivi des release notes, tests visuels automatisés sur les scénarios critiques (lecture, pause, seek, fin de vidéo).

8.2 Risques marché et adoption

Concurrence directe par Google. Risque structurel : Google peut intégrer nativement un mode focus à YouTube. Mitigation : la singularité du positionnement éthique (cost-recovery non lucratif) que Google ne peut pas adopter sans contredire son modèle économique constitue une protection structurelle.

Faible adoption initiale. Produit de niche par construction. Mitigation : MVP minimaliste, acquisition organique patiente, modèle économique ne supposant pas une adoption massive pour rester viable (break-even à 5 abonnés Premium).

8.3 Risques juridiques et sécurité

Conformité RGPD. L'application collecte des données personnelles (identité Google, playlists, notes). Mitigation : application complète du RGPD dès le MVP (consentement, droit à l'effacement, portabilité, hébergement européen via Hetzner Allemagne, chiffrement AES-256-GCM des tokens OAuth en base).

Vulnérabilités de sécurité. Risques classiques d'application web manipulant des tokens OAuth et exposant des endpoints publics. Mitigation : application du référentiel OWASP Top 10 documenté dans le dossier sécurité, cookies HttpOnly + SameSite=Lax + Secure, Helmet, rate limiting, Dependabot et `npm audit` en CI, validation Zod systématique des entrées.

8.4 Risques personnels

Contrainte de temps liée à l'alternance. Le projet est mené en solo dev sur le temps personnel, en parallèle de l'alternance chez Oktogone / VDL (développement Neatemys à temps plein). Fenêtre globale : 7 semaines de développement + 1 semaine de soutenance. Mitigation : périmètre MVP volontairement restreint (11 fonctionnalités MUST), priorisation MoSCoW stricte, MVP testable dès la semaine 4 pour valider la trajectoire à mi-parcours, plan B explicite de réduction du scope si retard supérieur à une semaine.

8.5 Tableau récapitulatif

Risque	Catégorie	Probabilité	Impact	Mitigation principale
Changement des CGU YouTube	Technique / Produit	Faible	Élevé	Veille, architecture en ports & adapters
Dépassement quotas API	Technique / Produit	Moyenne	Moyen	Cache, pagination, quota étendu
Évolutions player IFrame	Technique / Produit	Faible	Faible	Suivi release notes, tests visuels
Concurrence directe (Google)	Marché / Adoption	Faible (court terme)	Élevé	Différenciation éthique cost-recovery
Faible adoption initiale	Marché / Adoption	Élevée	Moyen	MVP minimal, cost-recovery
Non-conformité RGPD	Juridique / Sécurité	Faible	Élevé	Application complète RGPD dès MVP
Vulnérabilités sécurité	Juridique / Sécurité	Moyenne	Élevé	OWASP Top 10, scan, headers, rate limit

Risque	Catégorie	Probabilité	Impact	Mitigation principale
Contrainte de temps person-nelle	Personnel	Élevée	Élevé	Périmètre restreint, MoSCoW strict, plan B