CAHIER DES CHARGES

ABSOLUTE STREAM

Plateforme communautaire de suivi de films, séries et animés

Candidat	Alexis DE JESUS
Épreuve	E6 – BTS SIO option SLAM
Projet	Absolute Stream
Binôme	Thomas Montout
Stack technique	TypeScript · React 19 · Next.js 16 · PostgreSQL (NeonDB)
Année	2025 – 2027

SOMMAIRE

I. Présentation du Projet

1. Contexte

Dans le cadre de l'épreuve E6 du BTS SIO option SLAM, j'ai développé le projet "Absolute Stream" en binôme avec Thomas Montout. Ce projet vise à valider notre maîtrise de la conception, du développement et du déploiement d'une application web full-stack en conditions réelles, en s'appuyant sur une stack moderne : TypeScript, React 19 et Next.js 16.

2. La Vision et les Objectifs

Absolute Stream est une plateforme communautaire inspirée de Letterboxd, unificant films, séries et animés en une seule interface. L'objectif central est de résoudre la fragmentation des historiques de visionnage éparpillés sur de nombreuses plateformes (Netflix, Disney+, Crunchyroll…).

Les objectifs principaux du projet sont :

- Centraliser le suivi des œuvres vues, en cours et à voir
- Proposer une dimension sociale (abonnements, critiques, partage de listes)
- Garantir une expérience fluide, rapide et accessible (Mobile First, Dark Mode)
- Anticiper des fonctionnalités futures : système de Match en duo et tournois communautaires

3. Périmètre du MVP présenté

Pour la soutenance, nous avons fait le choix délibéré de présenter un Produit Minimum Viable (MVP) fonctionnel et stable, centré sur l'exploration de médias. Une grande partie de notre travail de conception a consisté à modéliser la base de données de l'application finale (Match, réseau social, critiques, listes). Cependant, en termes d'interface utilisateur (UI) livrable aujourd'hui, le périmètre assumé couvre exclusivement les piliers fondamentaux : la consultation avancée du catalogue (API TMDB), la recherche globale, les fiches détaillées riches, l'authentification sécurisée et les fondations du profil utilisateur.

II. Analyse des Besoins et Périmètre

1. Acteurs du système

Acteur	Description
Visiteur	Utilisateur non authentifié. Peut naviguer librement dans le catalogue, utiliser la recherche globale et consulter les fiches détaillées des œuvres.
Membre	Utilisateur authentifié. Dispose d'un compte sécurisé et d'un profil public posant les bases des futures interactions sociales.
API TMDB	Fournisseur de données externe (The Movie Database). Sollicité dynamiquement pour alimenter l'ensemble du catalogue d'œuvres.

2. Besoins fonctionnels (MVP)

- **Catalogue Unifié :** Navigation par carrousels, pages de découverte avec filtres dynamiques (genres/tris), page Collections, et isolation technique automatique des animés.
- **Recherche Globale Avancée :** Moteur de recherche centralisé avec menu déroulant, optimisé pour les performances réseau.
- **Fiches Détaillées Enrichies :** Affichage immersif regroupant les métadonnées localisées (dates FR, durées), le casting principal et l'intégration automatisée de la bande-annonce YouTube officielle.
- **Authentification et Profil :** Inscription et connexion sécurisées via *Better Auth*, persistance de la session et page de profil public en lecture seule.

3. Besoins non-fonctionnels

- **Ergonomie** : interface Dark Mode, responsive Mobile First, navigation fluide sur tous supports
- **Performances** : rendu SSR via le routeur Next.js pour optimiser le temps de chargement et le SEO
- **Sécurité** : protection des clés API, sessions sécurisées, protection contre les injections SQL et les failles XSS
- **Maintenabilité** : typage strict TypeScript, architecture modulaire, documentation continue

III. Description Graphique et Ergonomique

1. Thème et Design

L'interface adopte un thème sombre (Dark Mode) avec des arrière-plans dynamiques qui s'adaptent à l'affiche du média consulté, grâce à un hook personnalisé useImageColor. Cette approche crée une immersion visuelle forte, donnant l'impression que l'interface « vit » en fonction du contenu affiché.

L'ensemble des styles est géré via Tailwind CSS v4, permettant une cohérence visuelle et une maintenance simplifiée.

2. Composants UI et Accessibilité

- **Radix UI** : primitives accessibles pour les menus déroulants, sliders et navigation au clavier (conformité ARIA)
- **Embla Carousel** : défilement horizontal fluide pour les listes de films et séries
- **shadcn/ui** : composants prêts à l'emploi cohérents avec le design system

3. Principales interfaces développées

Interfaces dont j'ai assuré le développement principal

- NavBar responsive avec gestion de l'état d'authentification
- Page profil dynamique (/profile/[id]) avec sous-onglets (Bibliothèque, Listes, Critiques)
- Page unifiée de détails des œuvres (/view-media) : notes, commentaires, castings
- Pages d'authentification /login et /register
- Composants DiscoverMedia, LibraryContainer et MediaContainer
- Gestion dynamique des thèmes et arrière-plans selon les pages

IV. Architecture Technique

Le projet exploite l'architecture Serverless de Next.js, qui permet au code client (Frontend) et à la logique serveur (Backend) de cohabiter de manière sécurisée et maintenable au sein d'un même repository.

Stack Technologique

Technologie	Version	Rôle dans le projet
Next.js 16	Framework principal	App Router, Server Actions, SSR, Middleware de protection des routes
React 19	Bibliothèque UI	Composants fonctionnels, hooks, gestion d'état (Context API)
TypeScript 5	Typage statique	Sécurisation du code, interfaces pour les objets TMDB, détection des erreurs à la compilation
NeonDB	Base de données	PostgreSQL hébergé dans le cloud, scalable et sans maintenance serveur
Prisma 6	ORM	Requêtes SQL sécurisées et typées, schéma déclaratif, migrations automatisées
Better Auth	Authentification	Hachage des mots de passe, sessions via cookies HTTP-Only, protection XSS
Tailwind CSS v4	Styles	Classes utilitaires, thème sombre, responsive design
Radix UI	Composants	Accessibilité ARIA, primitives headless pour menus et interactions

Schéma de l'Architecture

FRONTEND (Client)

- Composants React / Next.js
- Tailwind CSS + Radix UI
- AuthContext global
- Optimistic UI (favoris)

BACKEND (Serveur)

- Route Handlers (proxy TMDB)
- Server Actions (mutations BDD)
- Prisma ORM + NeonDB
- Better Auth (sessions)

V. Modèle Conceptuel et Dictionnaire de Données

Le catalogue d'œuvres n'est pas stocké dans notre base de données – il est géré dynamiquement via l'API TMDB. Notre base PostgreSQL se concentre exclusivement sur les entités liées aux utilisateurs et à leurs interactions.

Principales entités du schéma Prisma

Entité	Champs principaux	Description
User	id, name, email, image	Compte utilisateur, géré par Better Auth
Library	userId, tmdbId, type, status	Statut de visionnage unique par utilisateur et par œuvre (Vu / À voir / En cours)
Review	userId, tmdbId, rating?, content?, visibility	Critique avec note et texte optionnels. Visibilité : PUBLIC ou FRIENDS
Favorite	userId, tmdbId, mediaType	Mise en favoris d'une œuvre
List	userId, name, isPublic	Liste personnalisée publique ou privée
ListItem	listId, tmdbId, mediaType	Entrée dans une liste personnalisée
Follow	followerId, followingId	Relation d'abonnement entre deux utilisateurs
*MatchSession**	id, userId1, userId2, status	Anticipé : session de match en duo (non activé dans le MVP)

* Afin de préparer l'évolution future du projet, l'ensemble du modèle relationnel a été implémenté (incluant Library, Review, List, Follow et le système de Match). Cependant, dans le périmètre strict du MVP présenté, ces entités ne sont pas encore reliées à des interfaces graphiques utilisateur.

VI. Description Fonctionnelle et Documentation API

Pour garantir la sécurité de la clé `TMDB_API_KEY`, le navigateur ne communique jamais directement avec TMDB. Thomas a mis en place une architecture où notre serveur joue le rôle de proxy.

1. Route Handlers – Proxy TMDB (Lecture)

L'intégralité des flux de données externes a été sécurisée via nos propres routes API.

Endpoints TMDB implémentés

Films (`/api/movies/`)

- `GET /api/movies/discoverMovies` Liste filtrée et paginée (params : page, genre)
- `GET /api/movies/popularMovies` Films populaires du moment
- `GET /api/movies/topRated` Films les mieux notés par la communauté TMDB
- `GET /api/movies/nowPlaying` Films actuellement à l'affiche au cinéma
- `GET /api/movies/movieGenres` Référentiel des identifiants de genres de films

Séries (`/api/tvshows/`)

- `GET /api/tvshows/discoverTvshows` Séries filtrées et paginées
- `GET /api/tvshows/popularTv` Séries populaires
- `GET /api/tvshows/topRated` Séries les mieux notées
- `GET /api/tvshows/onTheAir` Séries en cours de diffusion (nouveaux épisodes)
- `GET /api/tvshows/tvGenres` Référentiel des identifiants de genres de séries

Animés (`/api/animes/`)

- `GET /api/animes/discoverAnimes` Catalogue d'animés (avec isolation technique des flux)
- `GET /api/animes/popularAnimes` Animés populaires du moment
- `GET /api/animes/topRated` Animés les mieux notés
- `GET /api/animes/animeGenres` Référentiel des genres d'animation japonaise

Global & Recherche

- `GET /api/search` Moteur de recherche global (`/search/multi`)
- `GET /api/collections` Recherche de sagas et univers cinématographiques
- `GET /api/findByID/[external_id]` Lookup universel d'une œuvre via un ID externe

2. Server Actions – Mutations en écriture

Les actions modifiant la base de données utilisent les Server Actions Next.js. Elles s'exécutent côté serveur, vérifient la session via Better Auth, puis effectuent la requête Prisma appropriée.

Action	Table concernée	Comportement
toggleFavorite(mediaId, mediaType)	Favorite	Crée ou supprime un favori, UI mise à jour en Optimistic UI

VII. Gestion de Session et Sécurité

La sécurité a été un axe central de mon développement, notamment sur les aspects authentification et gestion de session.

1. Authentification avec Better Auth

- Hachage des mots de passe via l'algorithme scrypt (algorithme par défaut de Better Auth), conçu pour être lent et gourmand en mémoire afin de résister aux attaques par force brute.
- Cookies de session en HTTP-Only : en cas de faille XSS, le token de session reste illisible par JavaScript et ne peut donc pas être volé.
- Protection native contre les failles XSS par l'échappement automatique des variables au rendu par React (JSX), qui neutralise l'injection de balises.
- Gestion propre de l'état de session côté client via un AuthContext global

2. Protection des Routes (Middleware Next.js)

Un fichier middleware intercepte toutes les requêtes réseau. Il vérifie la validité de la session et redirige les utilisateurs non authentifiés vers la page de connexion lorsqu'ils tentent d'accéder à des pages privées.

3. Protection contre les Injections SQL

L'utilisation stricte de Prisma paramétrise automatiquement toutes les requêtes SQL en arrière-plan, éliminant le risque d'injection SQL sans effort supplémentaire du développeur.

4. Sécurité de l'API TMDB

La clé TMDB_API_KEY est stockée comme variable d'environnement serveur et n'est jamais exposée côté client. Le proxy via les Route Handlers garantit que le navigateur ne peut pas accéder directement à cette clé.

Corrections anti-patterns React (contribution spécifique)

- Résolution des problèmes de redirections dans les composants React (side effects hors useEffect)
- Mise en place d'une gestion correcte du cache pour éviter les clignotements de la NavBar lors du rechargement
- Correction des états de session inconsistants lors de la navigation entre pages protégées et publiques

VIII. Tests et Validation

1. Intégration Continue (CI/CD)

Un workflow GitHub Actions (ci.yml) a été mis en place pour garantir la stabilité du projet à chaque push. Ce pipeline automatique :

- Lance ESLint pour vérifier la conformité du code aux règles définies
- Exécute tsc --noEmit pour détecter toutes les erreurs de typage TypeScript
- Bloque les merges si l'une des vérifications échoue

2. Tests Fonctionnels Manuels

Les parcours utilisateurs critiques ont été validés manuellement :

Scénario testé	Résultat attendu	Statut
Inscription d'un nouvel utilisateur	Compte créé, session ouverte, redirection profil	**Validé**
Connexion / Déconnexion	Session persistante, NavBar cohérente	**Validé**
Ajout en favori (Optimistic UI)	UI mise à jour immédiatement, BDD persistée	**Partiellement validé**
Navigation vers une page privée sans session	Redirection vers /login par le Middleware	**Validé**
Suivi d'un autre utilisateur	Compteurs Followers/Following mis à jour	**Indisponible**
Création d'une liste privée	Liste visible uniquement pour l'auteur	**Indisponible**
Recherche globale d'un film	Affichage dynamique dans le menu déroulant	**Validé**

IX. Mes Réalisations Personnelles

Au sein du binôme, j'ai piloté le setup initial du projet, l'authentification complète, les interfaces dynamiques de l'application et le refactoring transverse. Ces travaux représentent 66 commits majeurs sur le repository.

1. Setup et Architecture Initiale

- Création et configuration initiale du projet Next.js 16 avec App Router
- Intégration de Radix UI et organisation rigoureuse du dossier app en Route Groups thématiques
- Mise en place de la structure de navigation et des conventions de nommage du projet

2. Authentification et Sécurité

Ce module représente l'une de mes contributions techniques les plus significatives :

- **Schéma Prisma** : conception et implémentation de l'ensemble du modèle de données
- **Better Auth** : intégration côté client et serveur, gestion des sessions et des cookies
- **AuthContext global** : contexte React partagé permettant à tous les composants d'accéder à l'état d'authentification
- **Pages /login et /register** : formulaires avec validation et gestion des erreurs
- **Middleware de protection** : redirection automatique des utilisateurs non authentifiés
- **Correction anti-patterns React** : résolution des problèmes de redirections et de clignotements de la NavBar

3. Navigation et UX/UI Applicatif

- **NavBar responsive** : affichage dynamique selon l'état d'authentification, menu mobile
- **Page profil /profile/[id]** : page dynamique avec sous-onglets : Bibliothèque, Listes, Critiques, Abonnements
- **Page /view-media** : fiche unifiée pour films, séries et animés : synopsis, notes, casting, critiques
- **Système de favoris Optimistic UI** : mise à jour instantanée de l'interface sans rechargement

4. Refactoring Transverse

- **DiscoverMedia** : wrapper unifié pour les pages de découverte multi-types
- **LibraryContainer et MediaContainer** : composants réutilisables pour l'affichage des œuvres
- **Gestion dynamique des thèmes** : arrière-plans adaptatifs selon le média consulté, via useImageColor

Résumé quantitatif de mes contributions

- 66 commits majeurs sur le repository GitHub
- Développement de l'ensemble du système d'authentification (Better Auth + Prisma)
- Conception et implémentation du schéma Prisma (8 entités dont 3 anticipées pour le futur)
- Développement de la NavBar, de la page profil, de la page /view-media
- Implémentation de l'Optimistic UI pour les favoris
- Résolution des anti-patterns React liés à la gestion de session

X. Bilan et Perspectives

1. Compétences validées

Le développement d'Absolute Stream m'a permis de valider des compétences concrètes de développeur Full-Stack dans un contexte réel et exigeant :

- Maîtrise de l'architecture Next.js hybride (SSR, App Router, Server Actions, Middleware)
- Intégration sécurisée d'une authentification tierce (Better Auth) dans un projet React
- Modélisation d'une base de données relationnelle avec Prisma et PostgreSQL
- Développement d'interfaces dynamiques et accessibles avec React 19 et Radix UI
- Mise en place d'un pipeline CI avec GitHub Actions (lint + typage TypeScript)
- Travail collaboratif en binôme avec Git (branches, code review, gestion des conflits)

2. Difficultés rencontrées et solutions apportées

Difficulté	Solution apportée
Clignotements de la NavBar lors des rechargements	Gestion fine du cache et de l'état de session dans le AuthContext
Anti-patterns React dans les redirections	Déplacement des effets de bord dans useEffect, refactoring des composants concernés
Cohérence de l'état d'authentification entre pages	Mise en place d'un contexte global unique et d'un middleware Next.js fiable
Complexité du schéma Prisma évolutif	Modélisation anticipée des entités futures (Match, Tournoi) sans impacter le MVP

3. Perspectives d'évolution

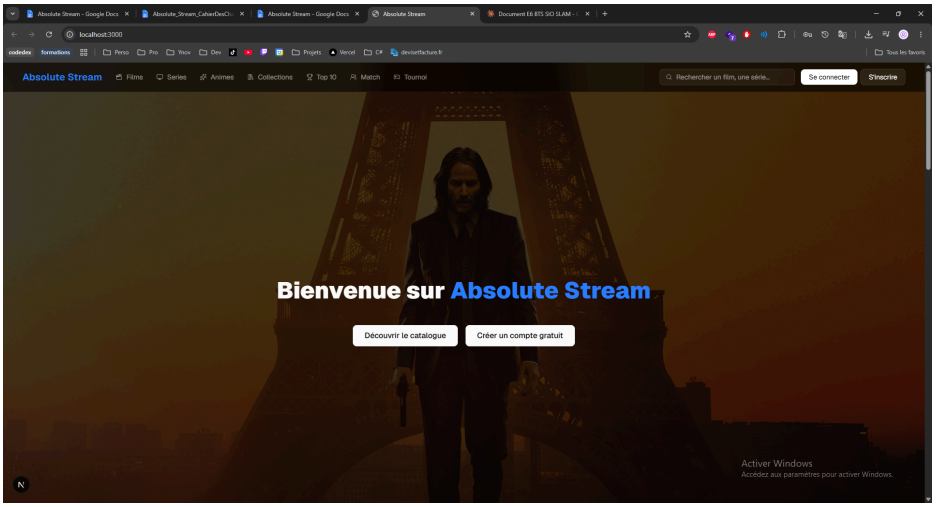
- **Système de Match** : développer l'interface de Swipe (les tables BDD sont déjà prêtes) pour activer la fonctionnalité phare
- **Application Mobile** : exploiter la séparation de notre logique Backend pour développer une app React Native
- **Recommandations personnalisées** : algorithme basé sur l'historique de visionnage de l'utilisateur
- **Notifications en temps réel** : WebSockets pour les interactions sociales (nouveau follower, réponse à une critique)

XI. Annexes

A. Captures d'écran de l'application

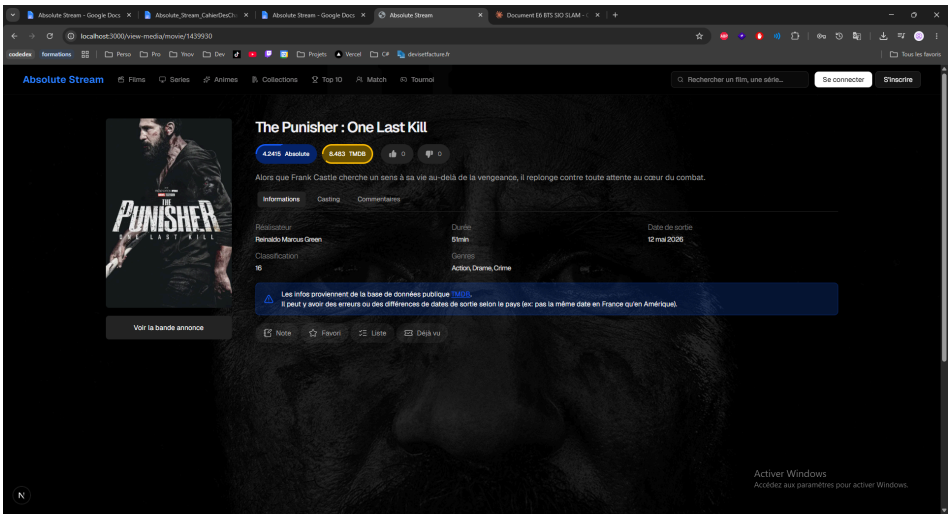
A1 – Page d'accueil (Home)

La page d'accueil présente les fonctionnalités d'Absolute Stream. Les arrière-plans s'adaptent dynamiquement au média survolé grâce au hook useImageColor.



A2 – Page de détail d'une œuvre (/view-media)

La fiche unifiée affiche les métadonnées localisées (dates FR, durées), le synopsis, le casting principal et intègre automatiquement la bande-annonce YouTube officielle.

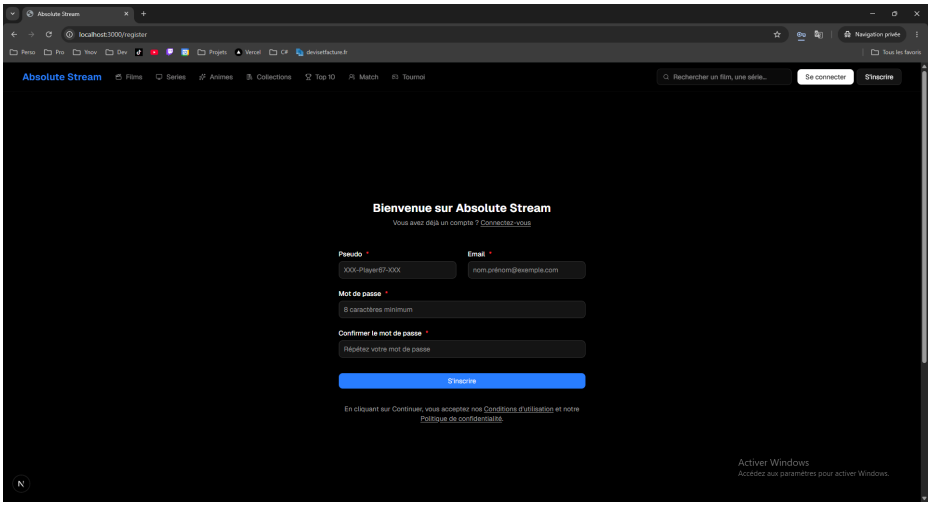


A3 – NavBar (état connecté vs non connecté)

La NavBar s'adapte dynamiquement selon l'état d'authentification : menu simplifié pour le visiteur, accès au profil et aux favoris pour le membre connecté.

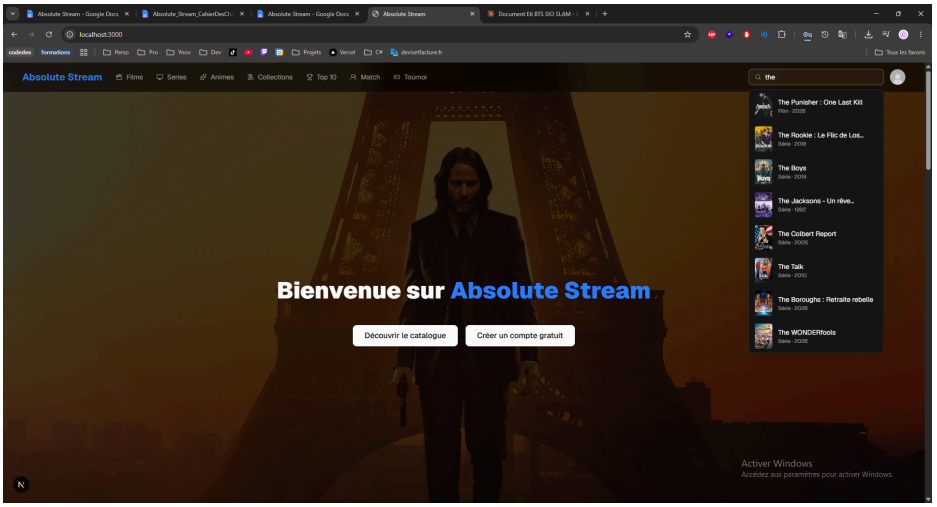
A4 – Page d'authentification (/login)

Formulaire de connexion avec validation des champs et gestion des erreurs. L'inscription (/register) suit la même structure visuelle.



A5 – Recherche globale dynamique

Le moteur de recherche affiche les résultats en temps réel dans un menu déroulant, optimisé pour limiter les appels réseau (debounce).

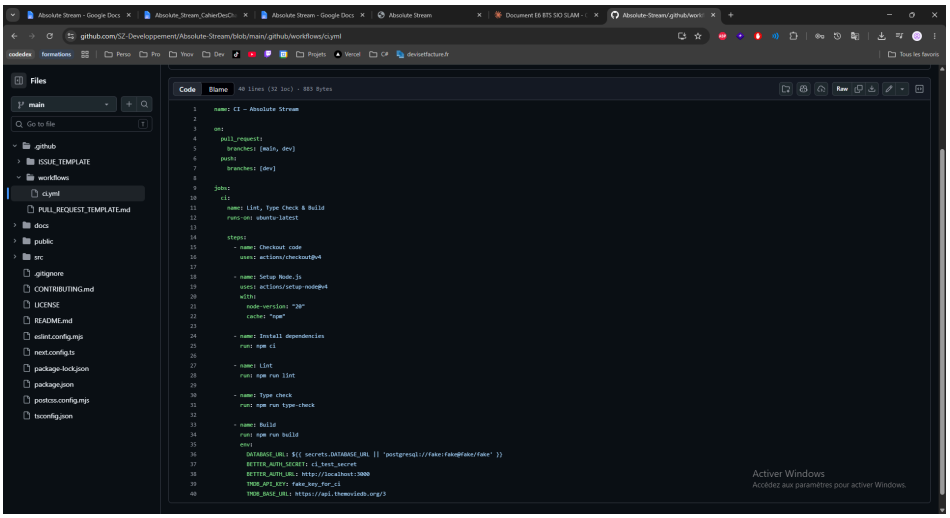


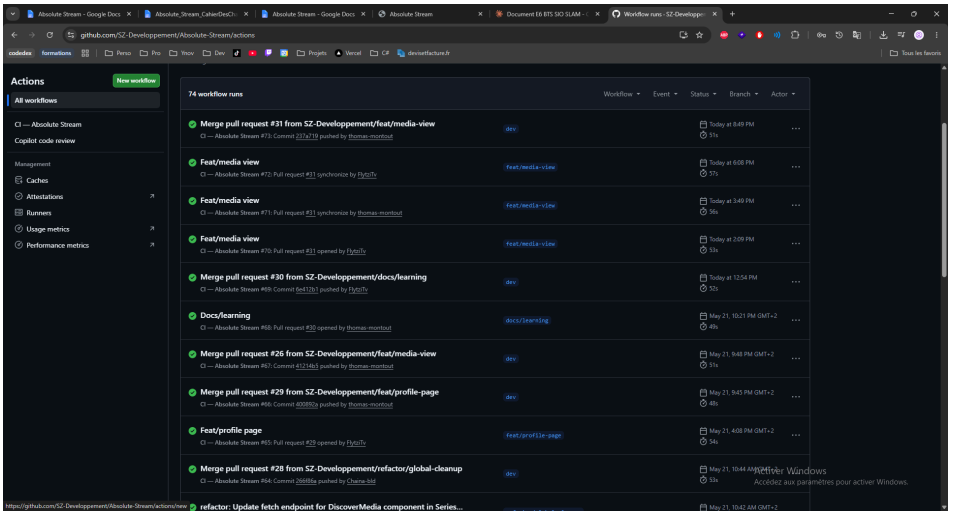
B. Schéma de base de données (Prisma)

Le schéma Prisma modélise l'ensemble des entités de l'application, y compris les entités anticipées pour les fonctionnalités futures (MatchSession, Swipe, MatchResult).

C. Pipeline CI/CD – GitHub Actions

Le workflow GitHub Actions valide automatiquement chaque push sur le repository : lint ESLint et vérification du typage TypeScript (tsc --noEmit).





D. AuthContext global

Le contexte React partagé expose l'état de session à tous les composants de l'application, évitant les props drilling et garantissant la cohérence de l'affichage.