TWITCHPLANNER - RAPPORT TECHNIQUE

Auteur : ADEDEMI Fréjus
Date : 13/02/2026
Contexte : Projet Application Web B3 - Keyce Academy

---

1. MODÈLE CONCEPTUEL DE DONNÉES (MCD)

Looping1.loo

---

2. MODÈLE PHYSIQUE DE DONNÉES (MPD)

---

3. DIFFICULTÉS RENCONTRÉES ET SOLUTIONS

3.1. Gestion de l'Upload d'Images

Problème :
Les images uploadées depuis le frontend n'étaient pas accessibles et les chemins n'étaient pas corrects en base de données.

Causes identifiées :

- Le dossier `uploads/` n'était pas servi en tant que ressource statique
- Les chemins relatifs n'étaient pas correctement construits
- Problèmes CORS lors de l'accès aux images

Solution implémentée :

1. Utilisation de la bibliothèque **Multer** pour gérer l'upload côté serveur

Configuration d'un middleware Express pour servir les fichiers statiques :
app.use('/uploads', express.static('uploads'));

2.
3. Stockage des chemins relatifs en base de données (ex: `filename.jpg`)
4. Construction de l'URL complète côté frontend (`${API_BASE_URL}/uploads/${filename}`)

Résultat :
Les images sont maintenant correctement uploadées, stockées et affichées dans l'application.

3.2. Export PNG avec html2canvas

Problème :
Lors de l'export du planning en PNG, le titre était coupé en haut de l'image et certains boutons d'action apparaissaient dans l'export.

Causes identifiées :

- La référence `ref` de html2canvas pointait vers une div ne contenant pas le titre
- Les boutons d'ajout et d'édition n'étaient pas masqués lors de l'export
- Images externes posaient des problèmes CORS

Solution implémentée :

1. Déplacement du titre **à l'intérieur** de la div référencée par `ref={exportRef}`
2. Ajout d'un attribut `data-export-hide` sur tous les boutons interactifs

Masquage programmatique de ces éléments avant l'export :
const actionButtons = exportRef.current.querySelectorAll('[data-export-hide]');actionButtons.forEach(btn => btn.style.display = 'none');

3.
4. Conversion des images en base64 pour éviter les problèmes CORS
5. Restauration de l'affichage après l'export

Résultat :
L'export PNG génère maintenant une image complète et professionnelle du planning sans éléments parasites.

3.3. Authentification JWT et Persistance de Session

Problème :
Après rafraîchissement de la page, l'utilisateur était déconnecté car le token JWT n'était pas persisté.

Causes identifiées :

- Le token JWT était stocké uniquement en mémoire (variable d'état React)
- Pas de vérification du token au chargement de l'application

Solution implémentée :

Stockage du token dans `localStorage` après login :
localStorage.setItem('token', data.token);

1.
2. Création d'un hook personnalisé `useAuth` qui :
    - Vérifie la présence du token au chargement
    - Valide le token auprès du backend
    - Maintient l'état de connexion

Protection des routes avec `useEffect` :
useEffect(() => { if (!loading && !user) navigate('/auth');}, [user, loading, navigate]);

3.

Résultat :
La session utilisateur persiste maintenant après rafraîchissement de la page.

3.4. Gestion des Dates et Semaines

Problème :
Affichage incorrect de la semaine en cours et incohérences lors de la navigation entre semaines.

Causes identifiées :

- JavaScript considère par défaut dimanche comme premier jour de la semaine
- Problèmes de fuseau horaire lors de la conversion Date → String
- Incohérence entre dates affichées et dates stockées en BDD

Solution implémentée :

1. Utilisation de la bibliothèque **date-fns** pour la manipulation de dates

Configuration du début de semaine au lundi :
startOfWeek(date, { weekStartsOn: 1 })

2.
3. Format cohérent pour le stockage en BDD (ISO 8601 : `YYYY-MM-DD`)
4. Stockage de `week_start` et `week_end` en BDD pour éviter les recalculs

Résultat :
L'affichage des semaines est maintenant cohérent et la navigation fonctionne correctement.

---

4. APPRENTISSAGES ET COMPÉTENCES MISES EN PRATIQUE

4.1. Compétences Techniques Acquises

Backend (Node.js / Express)

Architecture API REST :

- ✅ Création de routes RESTful (GET, POST, PUT, DELETE)
- ✅ Séparation des responsabilités (routes, middleware, logique métier)
- ✅ Gestion des erreurs avec try/catch
- ✅ Validation des données entrantes

Sécurité :

- ✅ Authentification JWT (génération et vérification de tokens)
- ✅ Hashage des mots de passe avec bcrypt (10 rounds de salt)
- ✅ Middleware de protection des routes
- ✅ Configuration CORS pour limiter les origines autorisées

Upload de Fichiers :

- ✅ Configuration de Multer pour gérer les uploads
- ✅ Validation des types de fichiers
- ✅ Limitation de la taille des fichiers
- ✅ Stockage et service de fichiers statiques

Frontend (React / Vite)

Architecture React Moderne :

- ✅ Composants fonctionnels avec Hooks
- ✅ Custom hooks (useAuth, useToast)
- ✅ Gestion d'état avec useState et useEffect
- ✅ Props drilling et composition de composants

Communication API :

- ✅ Appels asynchrones avec async/await
- ✅ Gestion des erreurs et des états de chargement
- ✅ Envoi de FormData pour upload de fichiers
- ✅ Headers d'authentification (Bearer token)

Styling et UI :

- ✅ Tailwind CSS pour un design moderne

- ✅ Composants shadcn/ui réutilisables
- ✅ Design responsive
- ✅ Animations et transitions CSS

Fonctionnalités Avancées :

- ✅ Manipulation du DOM avec html2canvas
- ✅ Gestion de dates avec date-fns
- ✅ Routing avec React Router
- ✅ Formulaires contrôlés

Base de Données (PostgreSQL)

Conception :

- ✅ Modélisation MCD/MPD
- ✅ Normalisation des données
- ✅ Définition des clés primaires et étrangères
- ✅ Contraintes d'intégrité (UNIQUE, NOT NULL, CHECK)

Requêtes SQL :

- ✅ SELECT avec JOIN pour récupérer des données liées
- ✅ INSERT pour créer des enregistrements
- ✅ UPDATE pour modifier des données
- ✅ DELETE avec CASCADE pour supprimer en cascade
- ✅ Utilisation de paramètres pour éviter les injections SQL

4.2. Méthodologie et Bonnes Pratiques

Gestion de Projet :

- ✅ Versioning avec Git
- ✅ Commits réguliers et messages descriptifs
- ✅ Branches pour développer des fonctionnalités
- ✅ Utilisation de GitHub pour le partage

Documentation :

- ✅ README complet avec instructions d'installation
- ✅ Commentaires dans le code
- ✅ Documentation de l'API (endpoints, paramètres, réponses)

Sécurité :

- ✅ Variables d'environnement (.env) pour les secrets

- ✅ .gitignore pour ne pas versionner les données sensibles
- ✅ Validation des données côté serveur
- ✅ Hashage des mots de passe

Architecture :

- ✅ Séparation frontend/backend
- ✅ Structure de dossiers claire
- ✅ Réutilisabilité du code (composants, fonctions)

---

5. CONCLUSION

Ce projet TwitchPlanner m'a permis de **mettre en pratique l'ensemble des compétences** acquises durant le cours d'Application Web :

1. **Conception de base de données** : Modélisation MCD/MPD, normalisation, contraintes
2. **Développement backend** : API REST avec Express, authentification JWT, upload de fichiers
3. **Développement frontend** : Application React moderne, hooks, appels API, export PNG

La **principale difficulté** rencontrée fut la gestion de l'export PNG avec html2canvas, notamment pour capturer exactement la zone voulue et gérer les images avec CORS. Cette expérience m'a appris à **debugger efficacement**, à **lire attentivement la documentation** des bibliothèques tierces, et à **tester rigoureusement** chaque fonctionnalité.

Le résultat final est une **application web fonctionnelle et utilisable** par de vrais streamers pour communiquer leurs plannings à leur communauté. Elle démontre ma capacité à **concevoir, développer et déployer** une application full-stack complète.

Ce projet constitue une excellente **base de portfolio** et m'a donné confiance dans ma capacité à mener à bien des projets web de A à Z.