

Feuille de Route - Plateforme de Génération de Bilans Neuropsychologiques

Vue d'ensemble du projet

Objectif : Créer un MVP permettant aux neuropsychologues de générer automatiquement des bilans à partir de leurs notes brutes et résultats de tests.

Stack technique :

- Frontend : Next.js + React + Tailwind CSS + Shadcn/ui
 - Backend : NestJS
 - Base de données : PostgreSQL + Prisma ORM
 - IA : OpenAI GPT API
 - Stockage fichiers : AWS S3 ou local
-

Phase 1 : Architecture et Setup (Semaines 1-2)

1.1 Setup du projet

- ☐ Initialisation du projet Next.js avec TypeScript
- ☐ Configuration Tailwind CSS + Shadcn/ui
- ☐ Setup NestJS backend avec TypeScript
- ☐ Configuration PostgreSQL + Prisma
- ☐ Setup Docker pour le développement
- ☐ Configuration ESLint/Prettier

1.2 Architecture de base

```
frontend/
├─ app/
│   ├─ dashboard/
│   ├─ patients/
│   ├─ bilans/
│   └─ settings/
├─ components/
│   ├─ ui/ (shadcn)
│   ├─ forms/
│   ├─ charts/
│   └─ layout/
└─ lib/
```

```
backend/
├─ src/
│   ├─ auth/
│   ├─ patients/
│   ├─ bilans/
│   ├─ tests/
│   ├─ uploads/
│   └─ ai/
```

1.3 Base de données

```
sql

-- Tables principales
- users (thérapeutes)
- patients
- bilans
- test_results
- documents
- templates
```

Phase 2 : Authentification et Base (Semaines 3-4)

2.1 Système d'authentification

- ☐ JWT avec NestJS + Passport
- ☐ Pages login/register (Shadcn forms)
- ☐ Middleware de protection des routes
- ☐ Gestion des rôles (thérapeute, admin)

2.2 Interface de base

- ☐ Layout principal avec sidebar
 - ☐ Dashboard d'accueil
 - ☐ Navigation responsive
 - ☐ Composants Shadcn configurés
-

Phase 3 : Gestion des Patients (Semaines 5-6)

3.1 CRUD Patients

- ☐ Formulaire création patient (Shadcn Form)
- ☐ Liste des patients avec recherche/filtres
- ☐ Fiche patient détaillée
- ☐ Historique des bilans par patient

3.2 Structure Patient

```
typescript

interface Patient {
  id: string;
  firstName: string;
  lastName: string;
  birthDate: Date;
  age: number;
  category: 'enfant' | 'adolescent' | 'adulte' | 'personne_agee';
  contactInfo: ContactInfo;
  medicalHistory: string;
  bilans: Bilan[];
}
```

Phase 4 : Gestion des Tests et Cotations (Semaines 7-9)

4.1 Référentiel des tests

- ☐ Base de données des tests par catégorie d'âge
- ☐ WISC-V, WAIS-IV, NEPSY-II, TEA-Ch, etc.
- ☐ Normes et barèmes de cotation
- ☐ Interface de sélection des tests

4.2 Saisie des résultats

- ☐ Formulaires dynamiques par test
- ☐ Calcul automatique des scores
- ☐ Comparaison avec les normes
- ☐ Graphiques de résultats (Recharts)

4.3 Composants spécialisés

```
typescript

// Composant de cotation automatique
<TestScoring
  testType="WISC-V"
  rawScores={scores}
  patientAge={age}
  onScoresCalculated={handleScores}
/>
```

Phase 5 : Upload et Traitement des Documents (Semaines 10-11)

5.1 Upload de fichiers

- ☐ Drag & drop zone (Shadcn)
- ☐ Support PDF, images, audio
- ☐ Stockage sécurisé (S3/local)
- ☐ Preview des documents

5.2 OCR et traitement

- ☐ Extraction de texte (Tesseract.js)
 - ☐ Transcription audio (OpenAI Whisper)
 - ☐ Parsing des résultats de tests
 - ☐ Classification automatique des documents
-

Phase 6 : Intégration IA et Génération (Semaines 12-14)

6.1 Service IA

```
typescript

@Injectable()
export class AIService {
  async generateBilan(data: BilanInput): Promise<GeneratedBilan> {
    // Prompts structurés par section
    // Gestion des tokens et coûts
    // Validation du contenu généré
  }
}
```

6.2 Prompts structurés

- ☐ Template prompts par section
- ☐ Prompts adaptatifs selon l'âge
- ☐ Gestion du contexte et des références
- ☐ Validation des outputs

6.3 Génération par sections

typescript

```
interface BilanSections {  
  anamnese: string;  
  testsProposed: string;  
  resultatsChiffres: string;  
  interpretation: string;  
  questionnaires: string;  
  synthese: string;  
  recommandations: string;  
}
```

Phase 7 : Éditeur de Bilans (Semaines 15-16)

7.1 Éditeur riche

- ☐ Éditeur WYSIWYG (TipTap ou similaire)
- ☐ Templates de bilans prédéfinis
- ☐ Sections modifiables
- ☐ Suggestions temps réel

7.2 Fonctionnalités d'édition

- ☐ Sauvegarde automatique
- ☐ Historique des versions
- ☐ Commentaires et annotations
- ☐ Export PDF professionnel

Phase 8 : Visualisation et Analytics (Semaine 17)

8.1 Tableaux de bord

- ☐ Statistiques des bilans générés
- ☐ Temps de traitement moyen
- ☐ Coûts IA par bilan
- ☐ Métriques d'utilisation

8.2 Visualisations

- ☐ Graphiques de résultats de tests
 - ☐ Comparaisons normatives
 - ☐ Évolution patient dans le temps
 - ☐ Rapports d'activité
-

Phase 9 : Optimisation et Déploiement (Semaines 18-19)

9.1 Performance

- ☐ Optimisation des requêtes
- ☐ Cache des prompts IA
- ☐ Compression des images
- ☐ Lazy loading des composants

9.2 Sécurité et conformité

- ☐ Chiffrement des données médicales
- ☐ Audit trails
- ☐ Conformité RGPD
- ☐ Sauvegarde automatique

9.3 Déploiement

- ☐ CI/CD avec GitHub Actions
 - ☐ Déploiement Vercel (frontend)
 - ☐ Déploiement backend (Railway/AWS)
 - ☐ Monitoring et logs
-

Phase 10 : Tests et Validation (Semaine 20)

10.1 Tests automatisés

- ☐ Tests unitaires (Jest)
- ☐ Tests d'intégration
- ☐ Tests E2E (Playwright)
- ☐ Tests de performance

10.2 Validation métier

- ☐ Tests avec vrais neuropsychologues
 - ☐ Validation des bilans générés
 - ☐ Ajustements des prompts
 - ☐ Documentation utilisateur
-

Composants Clés à Développer

1. Composants Shadcn personnalisés

typescript

// Sélecteur de tests par âge

```
<TestSelector ageCategory="adolescent" onTestsSelected={handleTests} />
```

// Éditeur de scores

```
<ScoreEditor test="WISC-V" onScoresChange={handleScores} />
```

// Préview du bilan

```
<BilanPreview sections={bilanSections} editable={true} />
```

2. Services backend critiques

typescript

// Service de cotation

```
@Injectable()
```

```
export class CotationService {
```

```
  calculateNormativeScores(test: TestType, rawScores: number[], age: number)  
}
```

// Service de génération

```
@Injectable()
```

```
export class BilanGeneratorService {
```

```
  async generateSection(section: BilanSection, context: PatientContext)  
}
```

3. Types TypeScript essentiels

typescript

```
interface BilanContext {
  patient: Patient;
  testResults: TestResult[];
  clinicalNotes: string[];
  previousBilans?: Bilan[];
}

interface GenerationSettings {
  sections: BilanSection[];
  template: BilanTemplate;
  aiModel: 'gpt-4' | 'gpt-3.5-turbo';
  maxTokens: number;
}
```

Estimation Budget IA

- **Développement** : ~500€/mois (tests et développement)
- **Production** : ~2-5€ par bilan généré
- **Optimisations** : Cache, prompts efficaces, modèles adaptés

Métriques de Succès

- Temps de génération : < 5 minutes par bilan
- Qualité : 80% du contenu utilisable sans modification
- UX : Interface intuitive, onboarding < 10 minutes
- Performance : Chargement < 3 secondes

Prochaines Itérations

1. **V2** : IA locale pour la confidentialité
2. **V3** : Intégration systèmes hospitaliers
3. **V4** : Mobile app pour saisie terrain
4. **V5** : Analytics prédictifs et recommandations