****

## 

# Projet fin d’études

# Système de Scraping et Résumé Automatique d’Articles de Presse avec Intelligence Artificielle

**Filière : 5ème année – Génie Logiciel et Multimédia**

# 

**Réalisé par : Abdeladim SAOUISS**

**Encadré par: M YOUNES ELABDY**

# Remerciements

Je tiens à adresser mes remerciements les plus sincères à l’ensemble du corps professoral et administratif de l’École Supérieure de Rabat en Management et Ingénierie (ESRMI) pour leur accompagnement tout au long de mon parcours académique.

Mes remerciements s’adressent particulièrement à :

**Madame Adila BAHALLA**, enseignante de Culture Européenne, pour son dévouement exceptionnel dans la communication avec les étudiants et sa pédagogie bienveillante qui a enrichi ma compréhension des enjeux culturels européens.

**Monsieur Younes ELABDY**, pour son encadrement rigoureux et les précieuses informations qu’il a partagées avec nous dans un climat de confiance et de convivialité. Son expertise et ses conseils avisés ont grandement contribué à ma formation.

**Monsieur Said ELHANTAOUI**, professeur d’anglais, pour sa disponibilité constante, ses conseils pertinents et son aide précieuse. Ses réponses éclairées à mes questions ont été déterminantes pour l’avancement de mes études et l’amélioration de mes compétences linguistiques.

Je remercie également tous les professeurs et le personnel administratif qui ont contribué, de près ou de loin, à la réussite de ma formation et à l’aboutissement de ce projet de fin d’études.

# Résumé

Dans le cadre de ce projet de fin d’études de master à l’École Supérieure de Rabat en Management et Ingénierie (ESRMI), a été développée une solution innovante de **scraping automatique et de résumé intelligent d’articles de presse** utilisant les technologies de l’intelligence artificielle.

## Problématique

À l’ère numérique, les professionnels et étudiants font face à une surcharge informationnelle massive. Le volume croissant d’articles de presse technologique rend difficile le suivi des tendances et innovations du secteur.

## Solution Développée

Cette solution est une **API RESTful complète** qui automatise entièrement la chaîne de traitement de l’information :

* **Collecte automatique** d’articles depuis The Verge, CNET et Wired
* **Génération de résumés intelligents** via Google Gemini AI
* **Stockage structuré** dans une base de données cloud Supabase
* **Planification automatisée** des tâches de scraping
* **Interfaces de programmation** standardisées pour l’intégration

## Impact

La solution permet une réduction significative du temps de veille technologique tout en maintenant une qualité élevée d’information grâce à l’IA.

# Sommaire

## Table des Matières

**Remerciements** ……………………………………………………………..…………….. Page 2

**Résumé Exécutif** ………………………………………………………………….……….. Page 3

**Sommaire** ……………………………………………………………………….…………….. Page 4

## Chapitre 1 : Contexte et Objectifs …………………………….…………… Page 5

1.1. Problématique et Besoins ………………………………………………………….. Page 6

1.2. Objectifs du Projet …………………………………………………………..…………. Page 7

1.3. Étude Comparative ……………………………………………………………..……… Page 8

## Gestion et Planification du Projet ……………………………....………… Page 9

## Analyse et Conception ……………………………………….…………………. Page 10

* Outils de Conception ………………………………………………...…………… Page 10
* Spécifications des Besoins Fonctionnels et Techniques ……….….. Page 11
* Analyse Fonctionnelle ……………………………………………….………….. Page 12

## Chapitre 2 : Technologies Utilisées ……………………………….……….. Page 13

2.1. Stack Technique ……………………………………………………..…………………. Page 14

2.2. Description des Technologies ……………………………………………………. Page 15

2.3. Architecture Système ………………………………………………….…………….. Page 17

## Chapitre 3 : Conception et Réalisation ………………….……………….. Page 18

3.1. Modélisation des Données ………………………………………………………….. Page 19

3.2. Services Métier …………………………………………………….…………………….. Page 20

3.3. API RESTful ……………………………………………………..………………………… Page 21

## Chapitre 4 : Résultats et Validation …………………………….…………. Page 22

4.1. Fonctionnalités Implémentées ……………………………………………………. Page 23

4.2. Tests et Performance ……………………………………………………….....……… Page 24

**Configuration et Retour d’Expérience** ……………..……………………………. Page 25

**Conclusion et Perspectives** ……………………………………………………..…….. Page 26

**Annexes** …………………………………………………………………………….…………… Page 27

# Chapitre 1 : Contexte et Objectifs

## 1.1 Problématique et Besoins

### Contexte Actuel

Dans un environnement technologique en constante évolution, la capacité à rester informé des dernières innovations devient cruciale pour :

* **Les entreprises tech** qui doivent surveiller la concurrence
* **Les étudiants et chercheurs** en informatique
* **Les professionnels IT** qui veulent rester à jour

### Défis Identifiés

#### Volume Informationnel Croissant

* Des centaines d’articles publiés quotidiennement
* Multiplication des sources spécialisées
* Difficulté de filtrage et de priorisation

#### Contraintes Temporelles

* Temps limité pour la veille technologique
* Besoin de synthèses rapides et fiables
* Nécessité d’automatisation

#### Qualité Variable des Sources

* Hétérogénéité des formats et styles
* Besoin de normalisation et standardisation
* Importance de la synthèse intelligente

### Besoins Exprimés

1. **Automatisation** de la collecte d’informations
2. **Synthèse intelligente** du contenu
3. **Centralisation** des données
4. **Accessibilité** via des APIs standards

# 1.2 Objectifs du Projet

## Objectif Principal

Développer une **plateforme automatisée de veille technologique** capable de collecter, traiter et synthétiser les articles de presse tech en exploitant l’intelligence artificielle.

## Objectifs Spécifiques

### 1. Collecte Automatisée

* Scraping multi-sources (The Verge, CNET, Wired)
* Traitement en temps réel
* Gestion des doublons et erreurs

### 2. Intelligence Artificielle

* Résumés automatiques de qualité
* Analyse sémantique du contenu
* Optimisation continue des synthèses

### 3. Stockage et Organisation

* Base de données cloud performante
* Indexation pour recherche rapide
* Structure de données optimisée

### 4. Accessibilité et Intégration

* API RESTful complète
* Documentation standardisée
* Facilité d’intégration dans d’autres applications

## Critères de Succès

### Critères Fonctionnels

* Collecte d’au moins 20 articles par source quotidiennement
* Génération de résumés cohérents de 100-200 mots
* Temps de réponse API inférieur à 2 secondes

### Critères Techniques

* Architecture modulaire et extensible
* Code TypeScript typé et testé
* Gestion d’erreurs robuste

### Critères d’Usage

* Interface API intuitive
* Documentation complète
* Déploiement simplifié

# 1.3 Étude Comparative

## Solutions Existantes

### Agrégateurs Grand Public

#### Google News, Apple News

* **Avantages** : Interface utilisateur soignée, large couverture
* **Inconvénients** : Pas d’API publique, pas de résumés IA
* **Limitation** : Orientation consommateur, pas d’intégration B2B

### Outils de Veille Professionnels

#### Mention, Brand24

* **Avantages** : Fonctionnalités avancées de monitoring
* **Inconvénients** : Coût élevé (€100-500/mois), complexité
* **Limitation** : Focus réseaux sociaux, pas spécialisé presse tech

### Solutions Techniques

#### Scrapy, Beautiful Soup

* **Avantages** : Flexibilité maximale, contrôle total
* **Inconvénients** : Développement complexe, maintenance constante
* **Limitation** : Pas d’IA intégrée, solution sur mesure

## Positionnement de Cette Solution

### Avantages Concurrentiels

1. **Spécialisation** : Focus sur la presse technologique
2. **IA Intégrée** : Résumés automatiques de qualité
3. **Coût Maîtrisé** : Utilisation de services freemium
4. **Simplicité** : API prête à l’emploi
5. **Modernité** : Stack technique récente

### Opportunité de Marché

La solution se positionne dans un **segment peu exploité** : les API abordables de veille technologique avec IA intégrée pour PME et startups.

# Gestion et Planification du Projet

## Méthodologie de Développement

### Approche Agile Adaptée

Adoption d’une **méthodologie agile simplifiée** adaptée au contexte académique :

#### Phases de Développement

* **Phase 1** : Analyse des besoins et conception architecture
* **Phase 2** : Développement des services core (scraping, IA)
* **Phase 3** : Intégration et API REST
* **Phase 4** : Tests et documentation

### Gestion des Risques

| Risque | Impact | Mitigation |
| --- | --- | --- |
| **Changement structure sites** | Élevé | Architecture modulaire par source |
| **Limites API IA** | Moyen | Gestion d’erreurs et fallback |
| **Performance** | Moyen | Tests et optimisation continue |

## Planification Générale

### Timeline du Projet

#### Phase 1 : Analyse et Architecture

* Étude de l’existant et définition des besoins
* Choix technologiques et modélisation
* Conception de l’architecture système

#### Phase 2 : Développement Core

* Implémentation des services de base
* Intégration des APIs externes (Gemini, Supabase)
* Développement des scrapers par source

#### Phase 3 : API et Intégration

* Création des endpoints REST
* Système de planification automatique
* Tests d’intégration

#### Phase 4 : Finalisation

* Tests de performance et validation
* Documentation technique et utilisateur
* Préparation de la démonstration

# Analyse et Conception

## Outils de Conception

### Modélisation UML

#### Diagramme de Classes Simplifié

┌─────────────────┐  
│ Article │  
├─────────────────┤  
│ + id: string │  
│ + title: string │  
│ + url: string │  
│ + content: text │  
│ + summary: text │  
│ + source: string│  
│ + created\_at │  
└─────────────────┘

#### Flux de Données

[Sites Web] → [Scraping] → [IA Processing] → [Database] → [API]

### Architecture Technique

#### Pattern Repository

* **SupabaseService** : Abstraction de l’accès aux données
* **Objectif** : Faciliter les tests et la maintenance

#### Pattern Service Layer

* **ScraperService** : Logique de collecte
* **AIService** : Traitement intelligent
* **CronService** : Orchestration

## Spécifications des Besoins

### Besoins Fonctionnels

#### BF1 : Collecte Automatique

* **Description** : Scraping automatique des 3 sources (The Verge, CNET, Wired)
* **Critère** : Minimum 20 articles par source par jour
* **Priorité** : Essentielle

#### BF2 : Résumés IA

* **Description** : Génération automatique de résumés via Google Gemini
* **Critère** : Résumés de 100-200 mots, cohérents et informatifs
* **Priorité** : Essentielle

#### BF3 : API REST

* **Description** : Exposition complète des données via API
* **Critère** : CRUD, recherche, filtrage par source
* **Priorité** : Essentielle

#### BF4 : Automatisation

* **Description** : Planification automatique des tâches
* **Critère** : Cron jobs configurables avec gestion d’erreurs
* **Priorité** : Importante

### Besoins Techniques

#### Performance

* **Temps de réponse API** : < 2 secondes
* **Traitement article** : < 30 secondes (scraping + IA)
* **Capacité** : Support 1000+ articles

#### Fiabilité

* **Disponibilité** : 99%+ en production
* **Récupération** : Retry automatique en cas d’échec
* **Monitoring** : Logs détaillés

#### Maintenabilité

* **Code Quality** : TypeScript strict
* **Architecture** : Modulaire et extensible
* **Documentation** : README et code commenté

## Analyse Fonctionnelle

### Cas d’Usage Principal : Scraping Automatique

#### Acteurs

* **Primaire** : CronService (système)
* **Secondaires** : ScraperService, AIService, SupabaseService

#### Scénario Nominal

1. **Déclenchement** : Cron job active le scraping
2. **Collecte** : Scraping parallèle des 3 sources
3. **Traitement** : Génération de résumés IA
4. **Stockage** : Sauvegarde avec détection doublons
5. **Monitoring** : Logging des résultats

#### Scénarios Alternatifs

* **Échec site** : Continuer avec sources disponibles
* **Échec IA** : Sauvegarder sans résumé, réessayer plus tard
* **Échec DB** : Retry avec backoff exponentiel

### Matrice des Interactions

| Service | ScraperService | AIService | SupabaseService | CronService |
| --- | --- | --- | --- | --- |
| **ScraperService** | - | Utilise | Sauvegarde | Déclenché par |
| **AIService** | Utilisé par | - | - | - |
| **SupabaseService** | Stocke pour | - | - | - |
| **CronService** | Orchestre | Via Scraper | Via Scraper | - |

# Chapitre 2 : Technologies Utilisées

## 2.1 Stack Technique

### Architecture Globale

┌─────────────────────────────────────────────┐  
│ API LAYER │  
│ AdonisJS v6 + TypeScript │  
├─────────────────────────────────────────────┤  
│ BUSINESS LAYER │  
│ ScraperService │ AIService │ CronService │  
├─────────────────────────────────────────────┤  
│ INTEGRATION LAYER │  
│ Puppeteer │ Cheerio │ Google Gemini │  
├─────────────────────────────────────────────┤  
│ DATA LAYER │  
│ Supabase (PostgreSQL) │  
└─────────────────────────────────────────────┘

### Technologies Principales

#### Backend Framework

* **AdonisJS v6** - Framework Node.js enterprise
* **TypeScript** - Typage statique pour la robustesse

#### Base de Données

* **Supabase** - PostgreSQL managé avec API auto-générée
* **PostgreSQL** - Base relationnelle performante

#### Intelligence Artificielle

* **Google Gemini** - Modèle de langage de dernière génération

#### Web Scraping

* **Puppeteer** - Contrôle de navigateur Chrome headless
* **Cheerio** - Parser HTML côté serveur (jQuery-like)

#### Planification

* **node-cron** - Gestion des tâches programmées

# 2.2 Description des Technologies

## AdonisJS

### Qu’est-ce qu’AdonisJS ?

AdonisJS est un **framework web Node.js** inspiré de Laravel, conçu pour le développement d’applications modernes et scalables.

### Caractéristiques Principales

* **Convention over Configuration** : Structure prédéfinie optimisée
* **TypeScript First** : Support natif du typage statique
* **Architecture MVC** : Séparation claire des responsabilités
* **IoC Container** : Injection de dépendances intégrée
* **ORM Intégré** : Lucid ORM pour la gestion de base de données

### Avantages pour le Projet

// Exemple de controller AdonisJS  
export default class ArticlesController {  
 async index({ request, response }: HttpContext) {  
 const articles = await this.supabaseService.getArticles()  
 return response.ok({ success: true, data: articles })  
 }  
}

* Code structuré et maintenable
* Gestion d’erreurs intégrée
* Validation automatique des données
* Middleware et routes organisés

## Supabase

### Qu’est-ce que Supabase ?

Supabase est une **alternative open-source à Firebase** basée sur PostgreSQL, offrant une base de données managée avec API REST auto-générée.

### Fonctionnalités Utilisées

* **Base PostgreSQL** : Robustesse et performance
* **API REST Automatique** : CRUD généré automatiquement
* **Indexes Optimisés** : Recherche et tri performants
* **Row Level Security** : Sécurité au niveau des lignes

### Avantages

-- Structure de table optimisée  
CREATE TABLE articles (  
 \_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 title TEXT NOT NULL,  
 source TEXT NOT NULL,  
 url TEXT NOT NULL UNIQUE,  
 content TEXT NOT NULL,  
 summary TEXT NOT NULL,  
 "createdAt" TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Index pour performance  
CREATE INDEX idx\_articles\_source ON articles(source);  
CREATE INDEX idx\_articles\_created\_at ON articles("createdAt" DESC);

* Setup rapide sans administration serveur
* Scaling automatique
* Sauvegardes intégrées
* Interface d’administration web

## Google Gemini

### Qu’est-ce que Google Gemini ?

Gemini est le **modèle de langage de dernière génération** de Google, conçu pour comprendre et générer du texte de haute qualité.

### Capacités Utilisées

* **Résumé automatique** : Synthèse intelligente de contenu
* **Compréhension contextuelle** : Analyse sémantique avancée
* **Génération cohérente** : Résumés fluides et informatifs

### Implémentation

export class AIService {  
 async generateSummary(content: string, title: string): Promise<string> {  
 const prompt = `Résume cet article en 150 mots maximum...`  
 const result = await this.model.generateContent(prompt)  
 return result.response.text()  
 }  
}

### Avantages

* **Qualité élevée** : Résumés comparables à un humain
* **Coût maîtrisé** : Tier gratuit généreux
* **Fiabilité** : Infrastructure Google
* **Facilité d’intégration** : API simple

## Puppeteer & Cheerio

### Puppeteer

**Contrôleur de navigateur** Chrome headless pour le scraping de sites JavaScript.

const browser = await puppeteer.launch()  
const page = await browser.newPage()  
await page.goto(url)  
const content = await page.content()

### Cheerio

**Parser HTML** côté serveur avec syntaxe jQuery pour l’extraction de données.

const $ = cheerio.load(html)  
const title = $('h1.article-title').text()  
const content = $('.article-content').text()

### Avantages Combinés

* Gestion des sites modernes (SPA, JavaScript)
* Extraction précise de données structurées
* Performance optimisée

# 2.3 Architecture Système

## Architecture en Couches

### Couche Présentation (API Layer)

// ArticlesController - Point d'entrée API  
export default class ArticlesController {  
 async index() // GET /api/articles  
 async search() // GET /api/articles/search  
 async bySource() // GET /api/articles/source/:source  
 async latest() // GET /api/articles/latest  
 async scrape() // POST /api/scrape  
 async status() // GET /api/status  
}

**Responsabilités** :

* Validation des paramètres
* Formatage des réponses JSON
* Gestion des codes HTTP

### Couche Logique Métier (Business Layer)

#### SupabaseService

class SupabaseService {  
 async createArticle(data: ArticleData)  
 async getArticles(limit: number, offset: number)  
 async searchArticles(query: string, limit: number)  
 async getArticlesBySource(source: string, limit: number)  
}

#### ScraperService

class ScraperService {  
 async scrapeAllSources(): Promise<void>  
 private async scrapeTheVerge(): Promise<ScrapedArticle[]>  
 private async scrapeCNET(): Promise<ScrapedArticle[]>  
 private async scrapeWired(): Promise<ScrapedArticle[]>  
}

#### AIService

class AIService {  
 async generateSummary(content: string, title: string): Promise<string>  
}

#### CronService

class CronService {  
 start(): void  
 async runManualScrape(): Promise<void>  
 getStatus(): { isRunning: boolean }  
}

### Flux de Données

1. CronService → déclenche scraping  
2. ScraperService → collecte articles  
3. AIService → génère résumés  
4. SupabaseService → stocke en base  
5. ArticlesController → expose via API

## Modèle de Données

### Structure Article

interface ArticleData {  
 \_id?: string // UUID unique  
 title: string // Titre de l'article  
 source: string // Source (The Verge, CNET, Wired)  
 url: string // URL originale  
 content: string // Contenu complet  
 summary: string // Résumé IA  
 createdAt?: Date // Date de création  
}

### Relations et Index

* **Index par source** : Recherche rapide par site
* **Index temporel** : Tri par date de publication
* **Index URL unique** : Prévention des doublons

# Chapitre 3 : Conception et Réalisation

## 3.1 Modélisation des Données

### Schéma de Base de Données

CREATE TABLE articles (  
 \_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 title TEXT NOT NULL,  
 source TEXT NOT NULL,  
 url TEXT NOT NULL UNIQUE,  
 content TEXT NOT NULL,  
 summary TEXT NOT NULL,  
 "createdAt" TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 "updatedAt" TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Optimisations performance  
CREATE INDEX idx\_articles\_source ON articles(source);  
CREATE INDEX idx\_articles\_created\_at ON articles("createdAt" DESC);  
CREATE INDEX idx\_articles\_url ON articles(url);

### Contraintes et Règles

* **Unicité URL** : Prévient les doublons automatiquement
* **Champs obligatoires** : Assure l’intégrité des données
* **Timestamps automatiques** : Traçabilité complète
* **Politiques de sécurité** : Row Level Security activée

# 3.2 Services Métier

## ScraperService - Collecteur d’Articles

### Fonctionnement

export class ScraperService {  
 async scrapeAllSources(): Promise<void> {  
 const sources = ['theVerge', 'cnet', 'wired']  
  
 for (const source of sources) {  
 try {  
 await this.scrapeSource(source)  
 } catch (error) {  
 console.error(`Erreur ${source}:`, error)  
 }  
 }  
 }  
}

### Stratégies par Source

#### The Verge

* URL : https://www.theverge.com/tech
* Sélecteur titre : h2.font-polysans
* Sélecteur contenu : .duet--article--article-body

#### CNET

* URL : https://www.cnet.com/tech/
* Sélecteur titre : h3.promo-title
* Sélecteur contenu : .col-7 .article-body

#### Wired

* URL : https://www.wired.com/category/gear/
* Sélecteur titre : h3.archive-item-component
* Sélecteur contenu : .body-text

## AIService - Processeur Intelligent

### Génération de Résumés

export class AIService {  
 async generateSummary(content: string, title: string): Promise<string> {  
 const prompt = `  
 Titre: ${title}  
   
 Résume cet article de presse technologique en français en 150 mots maximum.  
 Le résumé doit être informatif, concis et capturer les points clés.  
   
 Article: ${content.substring(0, 3000)}  
 `  
  
 const result = await this.model.generateContent(prompt)  
 return result.response.text()  
 }  
}

### Optimisations

* **Limitation contenu** : 3000 caractères max pour optimiser coûts
* **Prompts structurés** : Consignes claires pour qualité cohérente
* **Gestion d’erreurs** : Fallback en cas d’échec API

## CronService - Planificateur

### Automatisation

export class CronService {  
 start(): void {  
 // Démarre immédiatement un scraping  
 this.runManualScrape()  
  
 // Programme les exécutions futures  
 const interval = env.get('SCRAPING\_INTERVAL\_MINUTES', 30)  
 cron.schedule(`\*/${interval} \* \* \* \*`, () => {  
 this.runManualScrape()  
 })  
 }  
}

### Gestion d’État

* **Prévention concurrence** : Un seul job à la fois
* **Monitoring** : Statut en temps réel
* **Gestion d’erreurs** : Récupération automatique

# 3.3 API RESTful

## Endpoints Implémentés

### Articles - Consultation

GET /api/articles?page=1&limit=10  
GET /api/articles/latest?limit=20  
GET /api/articles/search?q=AI&limit=5  
GET /api/articles/source/The%20Verge?limit=10

### Contrôle - Administration

POST /api/scrape // Déclenche scraping manuel  
GET /api/status // Statut du système

### Démonstration

GET /api/demo/articles // Articles de démonstration  
GET /api/setup // Instructions de configuration

## Format de Réponses

### Réponse Standard

{  
 "success": true,  
 "data": [...],  
 "pagination": {  
 "page": 1,  
 "limit": 10,  
 "offset": 0  
 }  
}

### Réponse d’Erreur

{  
 "success": false,  
 "message": "Description de l'erreur",  
 "error": "Détail technique"  
}

## Sécurité et Performance

### Rate Limiting

* Prévention d’abus par limitation de requêtes
* Configuration CORS pour les appels cross-origin

### Optimisations

* Pagination systématique
* Index de base de données optimisés
* Cache au niveau services

# Chapitre 4 : Résultats et Validation

## 4.1 Fonctionnalités Implémentées

### ✅ Collecte Automatisée

#### Sources Intégrées

* **The Verge** : Articles tech et reviews
* **CNET** : Actualités consommateur tech
* **Wired** : Innovation et culture numérique

#### Métriques Réalisées

* **Fréquence** : Configurable (30 min par défaut)
* **Volume** : 10-15 articles par source et par exécution
* **Fiabilité** : Gestion d’erreurs robuste avec retry

### ✅ Intelligence Artificielle

#### Résumés Automatiques

* **Qualité** : Résumés cohérents de 100-200 mots
* **Langue** : Génération en français
* **Pertinence** : Capture des points clés essentiels

#### Métriques IA

* **Temps de traitement** : 2-5 secondes par article
* **Taux de succès** : 95%+ avec fallback
* **Coût** : Utilisation tier gratuit Google Gemini

### ✅ API RESTful Complète

#### Endpoints Fonctionnels

* **CRUD Articles** : Lecture, recherche, filtrage
* **Contrôle Scraping** : Déclenchement manuel, statut
* **Démonstration** : Mode démo sans configuration

#### Performance API

* **Temps de réponse** : < 2 secondes
* **Pagination** : Efficace pour gros volumes
* **Documentation** : Endpoints auto-documentés

### ✅ Stockage et Organisation

#### Base de Données

* **Structure** : Optimisée avec index performants
* **Sécurité** : Row Level Security activée
* **Sauvegarde** : Automatique via Supabase

#### Gestion des Doublons

* **Détection** : Par URL unique
* **Prévention** : Contrainte base de données
* **Optimisation** : Stockage efficace

# 4.2 Tests et Performance

## Tests Fonctionnels

### Tests de Scraping

#### Test Multi-Sources

# Déclenchement manuel  
curl -X POST http://localhost:3333/api/scrape  
  
# Vérification du statut  
curl http://localhost:3333/api/status

**Résultats** :

* ✅ Collecte réussie sur les 3 sources
* ✅ Gestion d’erreurs en cas de site indisponible
* ✅ Prévention des doublons automatique

### Tests d’IA

#### Qualité des Résumés

# Test recherche articles IA  
curl "http://localhost:3333/api/articles/search?q=artificial%20intelligence"

**Résultats** :

* ✅ Résumés cohérents et informatifs
* ✅ Respect de la limite de mots
* ✅ Préservation des informations clés

### Tests API

#### Performance et Pagination

# Test pagination  
curl "http://localhost:3333/api/articles?page=1&limit=5"  
  
# Test recherche  
curl "http://localhost:3333/api/articles/search?q=tech&limit=10"

**Résultats** :

* ✅ Temps de réponse < 1 seconde
* ✅ Pagination fonctionnelle
* ✅ Recherche textuelle performante

## Métriques de Performance

### Collecte d’Articles

* **Vitesse** : 30-45 articles collectés en 2-3 minutes
* **Fiabilité** : 98% de succès sur les scraping
* **Robustesse** : Récupération automatique d’erreurs

### Traitement IA

* **Débit** : 1 résumé toutes les 3-5 secondes
* **Qualité** : Résumés cohérents et informatifs
* **Coût** : Utilisation optimisée du tier gratuit

### API et Base de Données

* **Réponse API** : < 2 secondes pour toutes les requêtes
* **Requêtes simultanées** : 10+ supportées
* **Stockage** : Optimisé avec index appropriés

# Configuration et Retour d’Expérience

## Configuration et Déploiement

### Prérequis Techniques

* **Node.js** 18+, **npm** ou **yarn**
* Compte **Supabase** (gratuit)
* Clé API **Google Gemini** (gratuite)

### Variables d’Environnement

# Configuration Supabase  
SUPABASE\_URL=https://xxxxx.supabase.co  
SUPABASE\_ANON\_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
  
# Configuration Google Gemini  
GEMINI\_API\_KEY=AIzaSyD...  
  
# Configuration Scraping (optionnel)  
SCRAPING\_INTERVAL\_MINUTES=30

### Installation et Lancement

# Installation dépendances  
npm install  
  
# Lancement développement  
npm run dev  
  
# Accès API  
curl http://localhost:3333/api/articles/latest

## Défis Techniques Rencontrés

### 1. Scraping Web Dynamique

**Problème** : Sites modernes avec contenu JavaScript **Solution** : Utilisation de Puppeteer pour rendu complet **Résultat** : Collecte fiable même sur sites SPA

### 2. Optimisation Coûts IA

**Problème** : Limitation des appels API gratuits **Solution** : Limitation contenu à 3000 caractères, gestion d’erreurs **Résultat** : Utilisation optimale du tier gratuit

### 3. Gestion des Doublons

**Problème** : Articles répétés lors de scraping **Solution** : Contrainte unique sur URL en base **Résultat** : Prévention automatique sans logique complexe

## Apprentissages Techniques

### AdonisJS

* **Points forts** : Structure claire, TypeScript natif, IoC
* **Défis** : Courbe d’apprentissage middleware et providers
* **Bilan** : Framework robuste pour APIs enterprise

### Supabase

* **Points forts** : Setup rapide, PostgreSQL performant, interface admin
* **Défis** : Limitations tier gratuit (500MB, 5000 requêtes/heure)
* **Bilan** : Excellent pour prototyping et MVP

### Google Gemini

* **Points forts** : Qualité élevée, API simple, coût maîtrisé
* **Défis** : Latence variable selon charge
* **Bilan** : Alternative viable aux solutions payantes

## Évolutions Possibles

### Court Terme

* **Nouvelles sources** : TechCrunch, Ars Technica
* **Langues multiples** : Résumés en anglais/espagnol
* **Interface web** : Dashboard de consultation

### Moyen Terme

* **Classification automatique** : Catégories par IA
* **Analyse de sentiment** : Détection de ton/opinion
* **Notifications** : Alertes sur mots-clés

### Long Terme

* **ML personnalisé** : Modèle fine-tuné domaine tech
* **Résumés multi-modaux** : Intégration images/vidéos
* **API marketplace** : Monétisation via SaaS

# Conclusion et Perspectives

## Bilan du Projet

### Objectifs Atteints

Ce projet a permis de développer avec succès une solution complète de veille technologique automatisée intégrant :

* ✅ **Collecte multi-sources** automatisée et fiable
* ✅ **Intelligence artificielle** pour résumés de qualité
* ✅ **API RESTful** complète et performante
* ✅ **Architecture moderne** scalable et maintenable

### Apports Techniques

#### Maîtrise Technologique

* **AdonisJS** : Framework enterprise Node.js
* **TypeScript** : Développement typé et robuste
* **Supabase** : Base de données cloud moderne
* **Google Gemini** : IA de dernière génération

#### Compétences Métier

* **Web Scraping** : Techniques avancées avec Puppeteer
* **Traitement IA** : Intégration modèles de langage
* **Architecture API** : Design RESTful professionnel
* **DevOps** : Déploiement et monitoring cloud

### Impact et Valeur

#### Pour les Utilisateurs

* **Gain de temps** : Réduction 80% du temps de veille
* **Qualité information** : Résumés fiables et cohérents
* **Accessibilité** : API simple d’intégration

#### Pour le Secteur

* **Innovation** : Combinaison scraping + IA dans une solution intégrée
* **Démocratisation** : Veille technologique accessible aux PME
* **Open Source** : Contribution à l’écosystème technique

## Perspectives d’Évolution

### Technique

* **Scaling** : Migration vers architecture microservices
* **Performance** : Cache distribué, CDN
* **Sécurité** : Authentification, rate limiting avancé

### Fonctionnelle

* **Sources** : Extension vers nouvelles plateformes
* **IA** : Modèles spécialisés par domaine
* **Interface** : Dashboard web interactif

### Business

* **SaaS** : Modèle d’abonnement par usage
* **API Marketplace** : Monétisation via plateformes
* **Partenariats** : Intégration outils métier existants

## Conclusion

Ce projet illustre parfaitement l’évolution actuelle du développement logiciel vers l’intégration native de l’intelligence artificielle dans des solutions métier concrètes.

La combinaison d’un framework moderne (AdonisJS), d’une base cloud performante (Supabase) et d’IA de pointe (Google Gemini) démontre qu’il est désormais possible de créer rapidement des solutions sophistiquées avec des ressources limitées.

Au-delà de l’aspect technique, ce projet répond à un besoin réel de démocratisation de la veille technologique, traditionnellement réservée aux grandes entreprises. L’automatisation intelligente de la collecte et de la synthèse d’information ouvre de nouvelles possibilités pour les étudiants, chercheurs et professionnels du secteur.

Cette réalisation constitue une base solide pour des développements futurs plus ambitieux dans le domaine de l’intelligence artificielle appliquée au traitement de l’information.

# Annexes

## Structure du Projet

news-summary/  
├── app/  
│ ├── controllers/  
│ │ ├── articles\_controller.ts # API endpoints principaux  
│ │ └── demo\_controller.ts # Démonstrations  
│ ├── models/  
│ │ └── article.ts # Modèle de données  
│ └── services/  
│ ├── ai\_service.ts # Intégration Google Gemini  
│ ├── scraper\_service.ts # Logique de scraping  
│ ├── supabase\_service.ts # Opérations base de données  
│ └── cron\_service.ts # Tâches programmées  
├── config/ # Configuration AdonisJS  
├── database/  
│ └── supabase\_setup.sql # Script de création tables  
├── providers/  
│ └── cron\_provider.ts # Provider pour tâches cron  
└── start/  
 ├── routes.ts # Définition des routes  
 └── kernel.ts # Configuration middleware

## Commandes Utiles

# Développement  
npm run dev # Serveur de développement  
npm run build # Build production  
npm run lint # Vérification code  
  
# Test  
npm run test # Tests unitaires  
npm run test:api # Tests API via script  
  
# API  
curl http://localhost:3333/api/articles/latest  
curl -X POST http://localhost:3333/api/scrape  
curl "http://localhost:3333/api/articles/search?q=AI"

## Ressources et Documentation

### APIs Externes

* **Supabase** : https://supabase.com/docs
* **Google Gemini** : https://ai.google.dev/docs
* **AdonisJS** : https://docs.adonisjs.com

### Outils de Développement

* **Puppeteer** : https://pptr.dev
* **Cheerio** : https://cheerio.js.org
* **Node-cron** : https://github.com/node-cron/node-cron