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

## Présenté par : [Votre Nom]

## École Supérieure de Rabat en Management et Ingénierie (ESRMI)

## Année Académique : 2024-2025

# 2 - 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.

# 3 - Résumé

Dans le cadre de mon stage de fin d’études à l’École Supérieure de Rabat en Management et Ingénierie (ESRMI), j’ai développé une solution innovante de **scraping automatique et de résumé intelligent d’articles de presse** utilisant les technologies de l’intelligence artificielle.

## Contexte du Projet

À l’ère de l’information numérique, la quantité d’articles de presse publiés quotidiennement sur internet est considérable. Les professionnels, les étudiants et le grand public font face au défi de traiter et de synthétiser cette masse d’informations en temps réel. Notre solution répond à ce besoin en automatisant la collecte, l’analyse et la synthèse d’articles provenant de sources médiatiques reconnues.

## Solution Développée

Le système développé est une **API RESTful complète** basée sur le framework AdonisJS qui :

* **Collecte automatiquement** les articles depuis des sources prestigieuses (The Verge, CNET, Wired)
* **Génère des résumés intelligents** grâce à l’IA Google Gemini
* **Stocke et organise** les données dans une base Supabase performante
* **Planifie les tâches** avec un système de cron jobs automatisé
* **Expose des API** pour l’intégration avec d’autres applications

## Impact et Valeur Ajoutée

Cette solution apporte une valeur significative en permettant aux utilisateurs de rester informés efficacement, en économisant du temps précieux tout en maintenant un haut niveau de qualité informationnelle grâce aux capacités de synthèse de l’intelligence artificielle moderne.

# 4 - Sommaire

## Table des Matières

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

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

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

## Chapitre 1 : Présentation du Cadre de Projet …………………… Page 5

1.1. Objectif de la Solution …………………………………… Page 6

1.2. Étude de l’Existant ………………………………………. Page 8

1.3. Gestion de Projet ………………………………………… Page 10

1.4. Planification du Projet …………………………………… Page 12

1.5. Spécifications des Besoins ………………………………… Page 14

## Chapitre 2 : Analyse et Conception …………………………… Page 16

2.1. Outils de Conception ……………………………………… Page 17

2.2. Spécifications des Besoins ………………………………… Page 19

2.2.1. Besoins Fonctionnels …………………………………. Page 20

2.2.2. Besoins Techniques …………………………………… Page 22

2.3. Analyse Fonctionnelle …………………………………….. Page 24

## Chapitre 3 : Réalisation ……………………………………. Page 26

3.1. Architecture Technique ……………………………………. Page 27

3.2. Technologies d’Intelligence Artificielle …………………… Page 28

3.3. Système de Scraping Web …………………………………… Page 29

3.4. Base de Données Supabase ………………………………….. Page 30

3.5. Tests et Validation ………………………………………. Page 31

**Conclusion** ………………………………………………… Page 32

# 5 - Chapitre 1 : Présentation du Cadre de Projet

## Introduction du Projet

Dans un monde où l’information circule à une vitesse sans précédent, la capacité à traiter, analyser et synthétiser rapidement le contenu journalistique devient un enjeu majeur pour les entreprises, les institutions académiques et les professionnels de l’information.

## Contexte Technologique

L’évolution rapide des technologies web et de l’intelligence artificielle ouvre de nouvelles possibilités pour automatiser des tâches complexes de traitement de l’information. Les techniques de web scraping, combinées aux modèles de langage avancés, permettent aujourd’hui de créer des solutions sophistiquées d’agrégation et d’analyse de contenu.

## Problématique Identifiée

Les professionnels font face à plusieurs défis :

* **Surcharge informationnelle** : Volume croissant d’articles à traiter quotidiennement
* **Temps limité** : Nécessité de rester informé malgré des contraintes temporelles
* **Qualité variable** : Difficulté à identifier les informations les plus pertinentes
* **Dispersion des sources** : Multiplication des plateformes d’information

## Enjeux du Projet

Ce projet s’inscrit dans une démarche d’innovation technologique visant à :

1. **Automatiser** la collecte d’informations depuis des sources fiables
2. **Optimiser** le temps de traitement de l’information
3. **Améliorer** la qualité de la synthèse grâce à l’IA
4. **Centraliser** l’accès aux données via une API moderne

## Portée du Projet

Le système développé couvre l’ensemble de la chaîne de traitement de l’information journalistique, depuis la collecte automatisée jusqu’à la mise à disposition via des interfaces de programmation standardisées.

# 6 - 1.1 Objectif de la Solution

## Vision Générale

Développer une **plateforme intelligente et automatisée** capable de collecter, traiter et synthétiser en temps réel les articles de presse provenant de sources médiatiques reconnues, en exploitant les capacités avancées de l’intelligence artificielle pour générer des résumés de qualité professionnelle.

## Objectifs Principaux

### 1. Automatisation Complète

* **Collecte autonome** d’articles depuis The Verge, CNET et Wired
* **Traitement en continu** sans intervention humaine
* **Planification intelligente** des tâches de scraping

### 2. Intelligence Artificielle Intégrée

* **Résumés automatiques** générés par Google Gemini
* **Analyse sémantique** du contenu des articles
* **Optimisation** de la qualité des synthèses

### 3. Performance et Scalabilité

* **API RESTful** haute performance
* **Base de données cloud** avec Supabase
* **Architecture modulaire** et extensible

### 4. Accessibilité et Intégration

* **Endpoints standardisés** pour l’intégration
* **Documentation complète** de l’API
* **Interface de démonstration** incluse

## Objectifs Spécifiques

* Réduire de **80%** le temps de lecture et d’analyse d’articles
* Maintenir une **précision élevée** dans les résumés générés
* Assurer une **disponibilité 24/7** du service
* Permettre un **déploiement facile** sur différents environnements

## Bénéfices Attendus

* **Gain de productivité** pour les professionnels de l’information
* **Veille technologique** automatisée et efficace
* **Centralisation** des sources d’information dispersées
* **Qualité constante** des synthèses grâce à l’IA

# 7 - 1.1 Objectif de la Solution (Suite)

## Cas d’Usage Principaux

### 1. Veille Technologique Professionnelle

Les entreprises du secteur technologique peuvent utiliser le système pour :

* Surveiller les tendances du marché
* Analyser les innovations concurrentielles
* Rester informées des évolutions réglementaires

### 2. Recherche Académique

Les étudiants et chercheurs bénéficient de :

* Synthèses rapides d’articles scientifiques
* Identification des sujets émergents
* Compilation automatisée de bibliographies

### 3. Journalisme et Médias

Les professionnels des médias peuvent :

* Effectuer une veille concurrentielle
* Identifier rapidement les sujets tendance
* Préparer des synthèses éditoriales

### 4. Consulting et Analyse

Les consultants utilisent la solution pour :

* Produire des rapports sectoriels
* Analyser l’évolution des marchés
* Conseiller leurs clients sur les tendances

## Critères de Réussite

### Critères Quantitatifs

* **Temps de traitement** : < 30 secondes par article
* **Taux de disponibilité** : > 99.5%
* **Précision des résumés** : > 85% de satisfaction utilisateur
* **Volume traité** : 100+ articles par jour

### Critères Qualitatifs

* **Pertinence** des informations extraites
* **Cohérence** des résumés générés
* **Facilité d’intégration** dans les workflows existants
* **Robustesse** face aux changements de structure des sites

# 8 - 1.2 Étude de l’Existant

## Analyse du Marché Actuel

### Solutions Existantes

#### 1. Agrégateurs de Contenu Traditionnels

**Google News, Apple News, Flipboard**

* **Avantages** : Large couverture, interface utilisateur soignée
* **Inconvénients** : Pas de résumés automatiques, pas d’API publique
* **Limitation** : Orientation grand public, pas d’intégration B2B

#### 2. Outils de Veille Professionnels

**Mention, Brand24, Hootsuite Insights**

* **Avantages** : Fonctionnalités de monitoring avancées
* **Inconvénients** : Coût élevé, complexité d’utilisation
* **Limitation** : Focus sur les réseaux sociaux plutôt que la presse

#### 3. Solutions de Web Scraping

**Scrapy, Beautiful Soup, Octoparse**

* **Avantages** : Flexibilité technique, personnalisation
* **Inconvénients** : Requiert des compétences techniques, pas d’IA intégrée
* **Limitation** : Pas de solution clé en main

### Analyse Comparative

| Critère | Solutions Existantes | Notre Solution |
| --- | --- | --- |
| **Automatisation** | Partielle | Complète |
| **IA intégrée** | Limitée | Avancée (Gemini) |
| **API disponible** | Rare | Complète |
| **Coût** | Élevé | Optimisé |
| **Facilité d’usage** | Variable | Simplifiée |

## Opportunités Identifiées

### 1. Gap Technologique

* Manque de solutions combinant scraping et IA
* Absence d’APIs abordables pour PME
* Besoin de solutions spécialisées par secteur

### 2. Évolution des Besoins

* Accélération de la transformation digitale
* Besoin croissant d’automatisation
* Importance de la réactivité informationnelle

# 9 - 1.2 Étude de l’Existant (Suite)

## Analyse Technique des Concurrents

### 1. Limitations Techniques Identifiées

#### Agrégateurs Classiques

* **Manque d’APIs** : Difficile d’intégrer dans des applications tierces
* **Absence de personnalisation** : Pas d’adaptation aux besoins spécifiques
* **Résumés limités** : Généralement de simples extraits, pas de vraie synthèse

#### Solutions de Scraping Existantes

* **Complexité de mise en œuvre** : Nécessitent des équipes techniques expertes
* **Maintenance constante** : Sensibles aux changements des sites cibles
* **Pas d’intelligence intégrée** : Collecte brute sans traitement intelligent

### 2. Avantages Concurrentiels de Notre Solution

#### Innovation Technologique

* **IA de dernière génération** : Utilisation de Google Gemini
* **Architecture moderne** : AdonisJS et TypeScript
* **Cloud-native** : Conçu pour le cloud avec Supabase

#### Facilité d’Intégration

* **API RESTful standard** : Intégration simple dans tout environnement
* **Documentation complète** : Réduction du temps d’implémentation
* **Endpoints flexibles** : Adaptation aux différents cas d’usage

#### Robustesse et Fiabilité

* **Gestion d’erreurs avancée** : Récupération automatique en cas d’échec
* **Rate limiting intelligent** : Respect des politiques des sites
* **Monitoring intégré** : Surveillance en temps réel

## Positionnement Stratégique

Notre solution se positionne comme une **alternative moderne et accessible** aux solutions existantes, combinant :

* La **simplicité d’usage** des agrégateurs grand public
* La **puissance technique** des outils professionnels
* L’**innovation IA** des solutions de nouvelle génération
* Un **modèle économique accessible** aux PME et startups

# 10 - 1.3 Gestion de Projet

## Méthodologie Adoptée

### Approche Agile Adaptée

Nous avons adopté une **méthodologie agile adaptée** au contexte académique et aux contraintes du projet de fin d’études :

#### 1. Sprints Hebdomadaires

* **Durée** : 1 semaine par sprint
* **Objectifs** : Fonctionnalités spécifiques livrables
* **Révision** : Point hebdomadaire sur l’avancement

#### 2. Développement Itératif

* **Phase 1** : Architecture et services de base
* **Phase 2** : Intégration IA et scraping
* **Phase 3** : API et endpoints
* **Phase 4** : Tests et optimisation

### Organisation du Travail

#### 1. Planification Initiale

* **Analyse des besoins** : 1 semaine
* **Conception architecture** : 1 semaine
* **Développement core** : 3 semaines
* **Tests et documentation** : 1 semaine

#### 2. Gestion des Risques

| Risque | Probabilité | Impact | Mitigation |
| --- | --- | --- | --- |
| **Changement API sites** | Moyenne | Élevé | Architecture modulaire |
| **Limites API IA** | Faible | Moyen | Gestion d’erreurs robuste |
| **Performance** | Moyenne | Moyen | Tests de charge |
| **Complexité technique** | Élevée | Élevé | Formation continue |

## Outils de Gestion Utilisés

### 1. Versioning et Collaboration

* **Git** : Contrôle de version distribuée
* **GitHub** : Hébergement et collaboration
* **Branches** : Feature branching strategy

### 2. Documentation

* **README** : Documentation utilisateur
* **Commentaires code** : Documentation technique
* **Diagrammes** : Architecture et flux de données

### 3. Tests et Qualité

* **TypeScript** : Typage statique pour la robustesse
* **ESLint** : Analyse statique du code
* **Tests unitaires** : Validation des composants

# 11 - 1.3 Gestion de Projet (Suite)

## Défis Rencontrés et Solutions

### 1. Défis Techniques

#### Gestion de la Diversité des Sources

* **Problème** : Chaque site (The Verge, CNET, Wired) a sa propre structure HTML
* **Solution** : Architecture modulaire avec des scrapers spécialisés par source
* **Résultat** : Flexibilité et maintenabilité accrues

#### Intégration de l’IA

* **Problème** : Gestion des limites et coûts de l’API Google Gemini
* **Solution** : Système de fallback et gestion d’erreurs robuste
* **Résultat** : Service fiable même en cas de problèmes d’API

#### Performance et Scalabilité

* **Problème** : Traitement en masse d’articles peut être lent
* **Solution** : Traitement asynchrone et mise en cache intelligente
* **Résultat** : Temps de réponse optimisés

### 2. Défis de Gestion

#### Contraintes de Temps

* **Problème** : Délais académiques serrés
* **Solution** : Priorisation des fonctionnalités essentielles (MVP)
* **Résultat** : Livraison dans les temps avec qualité

#### Complexité Technologique

* **Problème** : Stack technique avancée à maîtriser
* **Solution** : Formation progressive et documentation extensive
* **Résultat** : Montée en compétence efficace

## Leçons Apprises

### 1. Importance de l’Architecture

Une **architecture bien pensée** dès le début permet d’éviter de nombreux problèmes de maintenance et d’évolution.

### 2. Tests Continus

Les **tests réguliers** durant le développement sont essentiels pour maintenir la qualité.

### 3. Documentation

Une **documentation à jour** facilite grandement la maintenance et les évolutions futures.

# 12 - 1.4 Planification du Projet

## Timeline Générale du Projet

### Phase 1 : Analyse et Conception (Semaines 1-2)

#### Semaine 1 : Étude Préliminaire

* **Lundi-Mardi** : Analyse du marché et des solutions existantes
* **Mercredi-Jeudi** : Définition des besoins et spécifications
* **Vendredi** : Validation du périmètre et des objectifs

#### Semaine 2 : Conception Détaillée

* **Lundi-Mardi** : Architecture technique et choix technologiques
* **Mercredi-Jeudi** : Modélisation de la base de données
* **Vendredi** : Finalisation des spécifications techniques

### Phase 2 : Développement Core (Semaines 3-5)

#### Semaine 3 : Infrastructure de Base

* **Services fondamentaux** : SupabaseService, configuration environnement
* **Modèles de données** : Article, interfaces TypeScript
* **Architecture AdonisJS** : Controllers, routes de base

#### Semaine 4 : Intégration IA et Scraping

* **AIService** : Intégration Google Gemini
* **ScraperService** : Développement des scrapers par source
* **Tests unitaires** : Validation des services core

#### Semaine 5 : API et Orchestration

* **ArticlesController** : Endpoints RESTful complets
* **CronService** : Automatisation et planification
* **Gestion d’erreurs** : Robustesse et fiabilité

### Phase 3 : Tests et Finalisation (Semaine 6)

#### Tests et Optimisation

* **Tests d’intégration** : Validation end-to-end
* **Tests de performance** : Optimisation des requêtes
* **Documentation** : README et guide d’utilisation
* **Déploiement** : Préparation pour la production

# 13 - 1.4 Planification du Projet (Suite)

## Diagramme de Gantt Simplifié

Semaines 1 2 3 4 5 6  
 ----+----+----+----+----+----  
Analyse ████████  
Conception ████████  
Services ████████  
IA/Scraping ████████  
API ████████  
Tests ████████

## Jalons Importants

### Jalons Techniques

#### J1 - Fin Semaine 2 : Architecture Validée

* ✅ Spécifications techniques finalisées
* ✅ Stack technologique confirmée
* ✅ Modèle de données approuvé

#### J2 - Fin Semaine 3 : Services de Base

* ✅ Connexion Supabase opérationnelle
* ✅ Structure AdonisJS en place
* ✅ Premiers tests unitaires

#### J3 - Fin Semaine 4 : Intégrations Majeures

* ✅ IA Google Gemini fonctionnelle
* ✅ Scraping multi-sources opérationnel
* ✅ Pipeline de traitement complet

#### J4 - Fin Semaine 5 : API Complète

* ✅ Tous les endpoints implémentés
* ✅ Cron jobs automatisés
* ✅ Gestion d’erreurs robuste

#### J5 - Fin Semaine 6 : Projet Finalisé

* ✅ Tests de validation réussis
* ✅ Documentation complète
* ✅ Prêt pour démonstration

### Critères de Validation

#### Critères Fonctionnels

* Scraping automatique des 3 sources
* Génération de résumés par IA
* API RESTful complète et documentée

#### Critères Techniques

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

#### Critères Qualité

* Documentation utilisateur claire
* Code commenté et maintenable
* Tests de validation réussis

# 14 - 1.5 Spécifications des Besoins

## Analyse des Besoins Utilisateurs

### 1. Profils d’Utilisateurs Cibles

#### Développeurs et Intégrateurs

* **Besoins** : API simple et bien documentée
* **Attentes** : Réponses JSON standardisées, gestion d’erreurs claire
* **Contraintes** : Compatibilité avec différents langages et frameworks

#### Professionnels de l’Information

* **Besoins** : Accès rapide aux dernières actualités tech
* **Attentes** : Résumés précis et synthétiques
* **Contraintes** : Fiabilité et fraîcheur des informations

#### Entreprises et Startups

* **Besoins** : Veille concurrentielle automatisée
* **Attentes** : Solution économique et scalable
* **Contraintes** : Facilité d’intégration dans l’infrastructure existante

### 2. Exigences Fonctionnelles Principales

#### EF1 : Collecte Automatique d’Articles

* **Description** : Le système doit collecter automatiquement les articles depuis The Verge, CNET et Wired
* **Critères** : Minimum 20 articles par source et par jour
* **Priorité** : Essentielle

#### EF2 : Génération de Résumés IA

* **Description** : Chaque article collecté doit être résumé par l’IA
* **Critères** : Résumé de 100-200 mots, cohérent et informatif
* **Priorité** : Essentielle

#### EF3 : API RESTful Complète

* **Description** : Exposition de tous les services via API REST
* **Critères** : CRUD complet, recherche, filtrage par source
* **Priorité** : Essentielle

#### EF4 : Planification Automatique

* **Description** : Automatisation des tâches de scraping
* **Critères** : Cron jobs configurables, gestion des erreurs
* **Priorité** : Importante

#### EF5 : Gestion des Doublons

* **Description** : Éviter les articles dupliqués
* **Critères** : Détection par URL, optimisation du stockage
* **Priorité** : Importante

# 15 - 1.5 Spécifications des Besoins (Suite)

## Exigences Non-Fonctionnelles

### 1. Performance

* **Temps de réponse API** : < 2 secondes pour les requêtes simples
* **Traitement article** : < 30 secondes par article (scraping + IA)
* **Capacité** : Support de 1000+ articles en base
* **Concurrence** : 10+ requêtes simultanées

### 2. Fiabilité et Disponibilité

* **Uptime** : 99%+ (objectif de production)
* **Récupération d’erreur** : Automatique avec retry logic
* **Gestion pannes** : Fallback et messages d’erreur explicites
* **Monitoring** : Logs détaillés pour le debugging

### 3. Sécurité

* **Protection API** : Rate limiting pour éviter l’abuse
* **Données sensibles** : Variables d’environnement sécurisées
* **Validation** : Sanitization des inputs utilisateur
* **CORS** : Configuration appropriée pour les appels cross-origin

### 4. Maintenabilité

* **Code quality** : TypeScript avec typage strict
* **Architecture** : Modulaire et extensible
* **Documentation** : Code commenté et README détaillé
* **Tests** : Couverture des fonctionnalités critiques

### 5. Évolutivité

* **Nouvelles sources** : Ajout facile de nouveaux sites à scraper
* **APIs IA** : Possibilité de changer de fournisseur IA
* **Scaling** : Architecture prête pour le scaling horizontal
* **Features** : Extensibilité pour nouvelles fonctionnalités

## Contraintes Techniques

### 1. Contraintes Environnementales

* **Runtime** : Node.js 18+
* **Base de données** : Compatible Supabase/PostgreSQL
* **APIs externes** : Google Gemini, sites web publics
* **Déploiement** : Compatible cloud (Heroku, Vercel, etc.)

### 2. Contraintes de Développement

* **Langage** : TypeScript obligatoire
* **Framework** : AdonisJS v6
* **Qualité code** : ESLint, formatage automatique
* **Versioning** : Git avec branches dédiées

# 16 - Chapitre 2 : Analyse et Conception

## Introduction à l’Analyse et Conception

La phase d’analyse et de conception constitue le **cœur stratégique** de notre projet. Elle transforme les besoins identifiés en une architecture technique robuste et une conception logicielle optimisée.

## Approche Méthodologique

### 1. Analyse Descendante (Top-Down)

Nous avons adopté une approche **top-down** partant des besoins utilisateurs globaux pour descendre vers les détails techniques d’implémentation :

* **Niveau Système** : Architecture globale et flux de données
* **Niveau Service** : Décomposition en services métier
* **Niveau Composant** : Détail des classes et interfaces
* **Niveau Implémentation** : Choix techniques spécifiques

### 2. Conception Orientée Services

L’architecture suit les **principes SOA** (Service-Oriented Architecture) adaptés au contexte d’une application moderne :

* **Séparation des responsabilités** : Chaque service a un rôle défini
* **Faible couplage** : Services indépendants et interchangeables
* **Haute cohésion** : Fonctionnalités liées regroupées logiquement
* **Réutilisabilité** : Services conçus pour être réutilisés

### 3. Patterns de Conception Appliqués

#### Pattern Repository

* **Objectif** : Abstraction de la couche d’accès aux données
* **Implémentation** : SupabaseService comme repository unifié
* **Avantage** : Facilite les tests et le changement de base de données

#### Pattern Service Layer

* **Objectif** : Encapsulation de la logique métier
* **Implémentation** : AIService, ScraperService, CronService
* **Avantage** : Séparation claire entre logique métier et présentation

#### Pattern Factory

* **Objectif** : Création d’objets complexes
* **Implémentation** : Factory pour les scrapers spécialisés
* **Avantage** : Extensibilité pour nouvelles sources

## Vue d’Ensemble de l’Architecture

Notre solution s’articule autour de **quatre couches principales** :

1. **Couche Présentation** : API REST et contrôleurs
2. **Couche Logique Métier** : Services spécialisés
3. **Couche Intégration** : Connecteurs externes (IA, scraping)
4. **Couche Données** : Supabase et modèles de données

# 17 - 2.1 Outils de Conception

## Outils de Modélisation Utilisés

### 1. Diagrammes UML

#### Diagramme de Classes

┌─────────────────┐ ┌─────────────────┐  
│ Article │ │ ArticleService │  
├─────────────────┤ ├─────────────────┤  
│ + id: string │◄───┤ + getArticles() │  
│ + title: string │ │ + searchArticles│  
│ + url: string │ │ + createArticle │  
│ + content: text │ └─────────────────┘  
│ + summary: text │  
│ + source: string│  
│ + createdAt │  
└─────────────────┘

#### Diagramme de Séquence - Processus de Scraping

User → CronService → ScraperService → AIService → SupabaseService  
 │ │ │ │ │  
 │ startScrape() │ │ │  
 │ │ scrapeAll() │ │  
 │ │ │ generateSummary() │  
 │ │ │ │ saveArticle()  
 │ │ │ │ │  
 │ │ ◄────────────────────────────────┘  
 │ ◄────┘

### 2. Architecture en Couches

#### Diagramme d’Architecture

┌─────────────────────────────────────────────┐  
│ COUCHE PRÉSENTATION │  
│ ArticlesController │ DemoController │  
├─────────────────────────────────────────────┤  
│ COUCHE LOGIQUE MÉTIER │  
│ ScraperService │ AIService │ CronService │  
├─────────────────────────────────────────────┤  
│ COUCHE INTÉGRATION │  
│ Puppeteer │ Cheerio │ Google Gemini │  
├─────────────────────────────────────────────┤  
│ COUCHE DONNÉES │  
│ SupabaseService │ PostgreSQL │  
└─────────────────────────────────────────────┘

### 3. Flux de Données

#### Pipeline de Traitement

[Sites Web] → [Scraping] → [Extraction] → [IA] → [Stockage] → [API]  
 │ │ │ │ │ │  
 The Verge Puppeteer Cheerio Gemini Supabase REST  
 CNET Browser Parser AI Database JSON  
 Wired Automation Content Summary PostgreSQL Endpoints

## Outils de Développement

### 1. Environnement de Développement

* **IDE** : Visual Studio Code
* **Extensions** : TypeScript, AdonisJS, ESLint
* **Terminal** : Intégration Git et npm

### 2. Outils de Conception de Base de Données

* **Supabase Dashboard** : Interface graphique pour la BDD
* **SQL Scripts** : Scripts de création et migration
* **Modélisation** : Relationships et contraintes

# 18 - 2.1 Outils de Conception (Suite)

## Standards et Conventions Adoptés

### 1. Conventions de Nommage

#### TypeScript/JavaScript

// Classes : PascalCase  
class ScraperService { }  
  
// Méthodes et variables : camelCase  
async scrapeArticles() { }  
const articleData = {};  
  
// Constantes : UPPER\_SNAKE\_CASE  
const MAX\_ARTICLES\_PER\_SOURCE = 10;  
  
// Interfaces : PascalCase avec préfixe I ou suffixe Interface  
interface ArticleData { }

#### Base de Données

-- Tables : snake\_case pluriel  
CREATE TABLE articles (  
 -- Colonnes : snake\_case  
 id UUID PRIMARY KEY,  
 created\_at TIMESTAMP,  
 updated\_at TIMESTAMP  
);

### 2. Architecture des Fichiers

#### Structure des Services

app/services/  
├── ai\_service.ts # Service IA Gemini  
├── scraper\_service.ts # Service scraping web  
├── supabase\_service.ts # Service base de données  
└── cron\_service.ts # Service planification

#### Structure des Contrôleurs

app/controllers/  
├── articles\_controller.ts # API articles  
└── demo\_controller.ts # Démonstrations

### 3. Patterns de Gestion d’Erreurs

#### Try-Catch Systématique

async someMethod() {  
 try {  
 // Logique métier  
 return successResponse;  
 } catch (error) {  
 console.error('Error details:', error);  
 return errorResponse;  
 }  
}

#### Validation des Entrées

if (!query) {  
 return response.badRequest({  
 success: false,  
 message: 'Query parameter is required',  
 })  
}

## Documentation Technique

### 1. Documentation du Code

* **JSDoc** : Commentaires standardisés pour les méthodes
* **README** : Guide d’installation et d’utilisation
* **API Docs** : Documentation des endpoints

### 2. Schémas de Données

* **Interfaces TypeScript** : Typage fort des données
* **Schémas Supabase** : Structure de base de données documentée
* **Exemples JSON** : Formats de réponse API

# 19 - 2.2 Spécifications des Besoins Détaillées

## Analyse Approfondie des Besoins

### 1. Cartographie des Acteurs

#### Acteurs Primaires

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Développeur │ │ Entreprise IT │ │ Chercheur │  
│ │ │ │ │ │  
│ • API simple │ │ • Veille auto │ │ • Synthèses │  
│ • Documentation │ │ • Intégration │ │ • Trends tech │  
│ • Fiabilité │ │ • Scalabilité │ │ • Bibliographie │  
└─────────────────┘ └─────────────────┘ └─────────────────┘

#### Acteurs Secondaires

┌─────────────────┐ ┌─────────────────┐  
│ Sites Source │ │ Services IA │  
│ │ │ │  
│ • The Verge │ │ • Google Gemini │  
│ • CNET │ │ • APIs externes │  
│ • Wired │ │ • Rate limits │  
└─────────────────┘ └─────────────────┘

### 2. Scénarios d’Usage Détaillés

#### Scénario 1 : Développeur Intègre l’API

1. Le développeur consulte la documentation  
2. Il obtient les endpoints disponibles  
3. Il teste avec curl/Postman  
4. Il intègre dans son application  
5. Il gère les réponses JSON

#### Scénario 2 : Veille Automatique d’Entreprise

1. L'entreprise configure les cron jobs  
2. Le système scrape automatiquement  
3. Les résumés IA sont générés  
4. Les données sont stockées  
5. Les équipes accèdent via dashboard

#### Scénario 3 : Recherche Académique

1. Le chercheur recherche par mots-clés  
2. Le système filtre les articles pertinents  
3. Il obtient des résumés structurés  
4. Il export les données pour analyse  
5. Il cite les sources originales

### 3. Matrice de Traçabilité

| Besoin Utilisateur | Exigence Fonctionnelle | Service Technique | Test de Validation |
| --- | --- | --- | --- |
| Accès articles récents | EF1 - Collecte auto | ScraperService | Test scraping |
| Résumés intelligents | EF2 - IA Gemini | AIService | Test qualité résumé |
| Intégration facile | EF3 - API REST | ArticlesController | Test endpoints |
| Mise à jour auto | EF4 - Cron jobs | CronService | Test planification |

# 20 - 2.2.1 Besoins Fonctionnels Détaillés

## Catalogue Complet des Besoins Fonctionnels

### Groupe 1 : Collecte et Extraction

#### BF-01 : Scraping Multi-Sources

* **Description** : Collecter automatiquement les articles depuis 3 sources
* **Sources** : The Verge, CNET, Wired
* **Fréquence** : Configurable (par défaut : quotidienne)
* **Volume** : 10+ articles par source
* **Critères qualité** : Articles récents (< 24h), contenu substantiel

#### BF-02 : Extraction de Contenu

* **Description** : Extraire le contenu textuel complet des articles
* **Éléments** : Titre, URL, contenu principal, date de publication
* **Filtrage** : Suppression des publicités, menus, éléments non-pertinents
* **Format** : Texte brut structuré

#### BF-03 : Gestion de la Robustesse

* **Description** : Gérer les échecs de scraping et les changements de structure
* **Retry logic** : 3 tentatives avec backoff exponentiel
* **Fallback** : Messages d’erreur explicites
* **Adaptation** : Détection automatique des changements de structure

### Groupe 2 : Intelligence Artificielle

#### BF-04 : Génération de Résumés

* **Description** : Produire des résumés intelligents via Google Gemini
* **Longueur** : 100-200 mots optimaux
* **Qualité** : Préservation des informations clés
* **Cohérence** : Style journalistique professionnel
* **Langue** : Français ou langue source selon configuration

#### BF-05 : Optimisation IA

* **Description** : Optimiser les appels API et gérer les limites
* **Rate limiting** : Respect des quotas Google Gemini
* **Caching** : Éviter les appels redondants
* **Fallback** : Génération de résumés de base si IA indisponible

### Groupe 3 : Stockage et Recherche

#### BF-06 : Persistance des Données

* **Description** : Sauvegarder tous les articles et métadonnées
* **Déduplication** : Éviter les doublons par URL
* **Indexation** : Optimisation pour la recherche
* **Historique** : Conservation de l’historique des modifications

#### BF-07 : Recherche Avancée

* **Description** : Recherche full-text dans les articles
* **Critères** : Titre, contenu, résumé, source
* **Filtres** : Par source, date, longueur
* **Performance** : Résultats < 2 secondes

# 21 - 2.2.1 Besoins Fonctionnels (Suite)

### Groupe 4 : API et Intégration

#### BF-08 : API RESTful Complète

* **Description** : Exposer toutes les fonctionnalités via API REST
* **Endpoints** :
  + GET /api/articles - Liste paginée
  + GET /api/articles/search - Recherche
  + GET /api/articles/source/:source - Par source
  + GET /api/articles/latest - Plus récents
  + POST /api/scrape - Déclenchement manuel
  + GET /api/status - Statut du système

#### BF-09 : Formats de Réponse

* **Description** : Standardisation des réponses JSON
* **Structure** :

{  
 "success": boolean,  
 "data": object|array,  
 "message": string,  
 "pagination": object  
}

#### BF-10 : Gestion des Erreurs API

* **Description** : Codes d’erreur HTTP appropriés et messages explicites
* **Codes** : 200, 400, 404, 500 avec descriptions
* **Logging** : Traçabilité complète des erreurs

### Groupe 5 : Automatisation

#### BF-11 : Planification Automatique

* **Description** : Exécution automatique des tâches de scraping
* **Configuration** : Cron jobs paramétrables
* **Monitoring** : Statut en temps réel des jobs
* **Récupération** : Reprise automatique en cas d’échec

#### BF-12 : Monitoring et Logs

* **Description** : Surveillance du système et logging détaillé
* **Métriques** : Nombre d’articles traités, temps de traitement, erreurs
* **Alertes** : Notification en cas de problème
* **Historique** : Logs persistants pour debugging

### Groupe 6 : Configuration et Administration

#### BF-13 : Configuration Flexible

* **Description** : Paramétrage via variables d’environnement
* **Paramètres** : APIs keys, intervalles de scraping, limites
* **Validation** : Vérification de la configuration au démarrage
* **Documentation** : Guide de configuration complet

#### BF-14 : Mode Démonstration

* **Description** : Fonctionnement avec données de démonstration
* **Objectif** : Test sans APIs externes configurées
* **Données** : Articles d’exemple pré-chargés
* **Endpoints** : /api/demo/\* pour les démonstrations

# 22 - 2.2.2 Besoins Techniques Détaillés

## Infrastructure et Performance

### Groupe 1 : Architecture Technique

#### BT-01 : Stack Technologique

* **Framework** : AdonisJS v6 (Node.js/TypeScript)
* **Base de données** : Supabase (PostgreSQL cloud)
* **IA** : Google Gemini API
* **Scraping** : Puppeteer + Cheerio
* **Planification** : node-cron

#### BT-02 : Qualité du Code

* **Typage** : TypeScript strict mode
* **Linting** : ESLint avec règles strictes
* **Formatage** : Prettier automatique
* **Tests** : Couverture des services critiques

#### BT-03 : Architecture Modulaire

* **Séparation** : Couches distinctes (contrôleurs, services, modèles)
* **Injection** : Dépendances injectées via le conteneur AdonisJS
* **Extensibilité** : Ajout facile de nouvelles sources/fonctionnalités

### Groupe 2 : Performance et Scalabilité

#### BT-04 : Performance Cible

* **Réponse API** : < 2 secondes pour requêtes simples
* **Traitement article** : < 30 secondes (scraping + IA)
* **Concurrence** : 10+ requêtes simultanées
* **Mémoire** : < 512 MB utilisation normale

#### BT-05 : Optimisations

* **Caching** : Mise en cache des résultats fréquents
* **Pagination** : Limitation des résultats par page
* **Asynchrone** : Traitement non-bloquant
* **Rate limiting** : Protection contre l’abus

#### BT-06 : Monitoring Performance

* **Métriques** : Temps de réponse, utilisation mémoire/CPU
* **Profiling** : Identification des goulots d’étranglement
* **Alertes** : Seuils de performance configurables

### Groupe 3 : Sécurité et Fiabilité

#### BT-07 : Sécurité

* **Variables d’environnement** : Secrets protégés
* **Validation inputs** : Sanitization des données utilisateur
* **CORS** : Configuration cross-origin appropriée
* **Rate limiting** : Protection DDoS basique

#### BT-08 : Gestion d’Erreurs

* **Try-catch** : Encapsulation systématique
* **Logging** : Traçabilité complète des erreurs
* **Fallback** : Dégradation gracieuse
* **Recovery** : Reprise automatique quand possible

# 23 - 2.2.2 Besoins Techniques (Suite)

### Groupe 4 : Intégration et APIs

#### BT-09 : Intégrations Externes

* **Google Gemini** : Gestion des quotas et authentification
* **Sites web** : Adaptation aux changements de structure
* **Supabase** : Optimisation des requêtes PostgreSQL
* **Resilience** : Timeout et retry logic appropriés

#### BT-10 : Compatibilité

* **Node.js** : Version 18+ LTS
* **Navigateurs** : Support des APIs REST standards
* **Cloud** : Déployable sur Heroku, Vercel, AWS
* **Environnements** : Development, staging, production

### Groupe 5 : Maintenance et Évolution

#### BT-11 : Maintenabilité

* **Documentation** : Code auto-documenté et README détaillé
* **Structure** : Organisation logique des fichiers
* **Debugging** : Logs détaillés et stack traces
* **Refactoring** : Architecture facilitant les modifications

#### BT-12 : Évolutivité

* **Nouvelles sources** : Ajout facile de scrapers
* **Nouvelles APIs** : Changement de fournisseur IA
* **Nouvelles fonctionnalités** : Extension sans régression
* **Migration** : Compatibilité ascendante

## Contraintes Techniques Spécifiques

### Contraintes Externes

* **Rate limiting sites** : Respect des robots.txt et politiques
* **Quotas API IA** : Gestion des limites Google Gemini
* **Stabilité réseau** : Gestion des timeouts et connexions
* **Formats changeants** : Adaptation aux évolutions des sites

### Contraintes Internes

* **Ressources limitées** : Optimisation mémoire et CPU
* **Temps de développement** : Priorisation des features essentielles
* **Compétences techniques** : Stack moderne mais accessible
* **Budget** : Solutions gratuites ou peu coûteuses privilégiées

# 24 - 2.3 Analyse Fonctionnelle

## Modélisation des Processus Métier

### 1. Processus Principal : Cycle de Vie d’un Article

┌─────────────┐ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐  
│ Détection │───▶│ Extraction │───▶│ Traitement │───▶│ Stockage │  
│ │ │ │ │ │ │ │  
│ • Crawling │ │ • Scraping │ │ • IA Résumé │ │ • Supabase │  
│ • Nouveauté │ │ • Parsing │ │ • Validation│ │ • Indexation│  
└─────────────┘ └─────────────┘ └─────────────┘ └─────────────┘

### 2. Flux de Données Détaillé

#### Phase 1 : Découverte et Collecte

Sites Sources → Puppeteer → Parser HTML → Extraction Métadonnées  
 │ │ │ │  
 Page Lists Browser Cheerio Article Objects  
 (HTML) Automation Parsing (TypeScript)

#### Phase 2 : Enrichissement IA

Article Brut → Validation → API Gemini → Résumé Généré → Article Enrichi  
 │ │ │ │ │  
 Raw Content Sanitize IA Request AI Response Complete Object

#### Phase 3 : Persistance et Accès

Article Final → Déduplication → Stockage BDD → Indexation → API Access  
 │ │ │ │ │  
Complete Data Check Existing Supabase Search Index REST JSON

### 3. Matrice des Interactions

| Composant | ScraperService | AIService | SupabaseService | CronService |
| --- | --- | --- | --- | --- |
| **ScraperService** | - | Appelle pour résumés | Sauvegarde articles | Déclenché par |
| **AIService** | Utilisé par | - | - | - |
| **SupabaseService** | Stocke pour | - | - | - |
| **CronService** | Orchestre | Via Scraper | Via Scraper | - |

# 25 - 2.3 Analyse Fonctionnelle (Suite)

## Analyse des Cas d’Usage Complexes

### Cas d’Usage 1 : Scraping Intelligent Multi-Sources

#### Acteurs

* **Primaire** : CronService (automatique)
* **Secondaire** : ScraperService, AIService, SupabaseService

#### Préconditions

* Services configurés avec APIs keys
* Connexion internet stable
* Base de données accessible

#### Scénario Principal

1. **Déclenchement** : Cron job active le scraping
2. **Collecte parallèle** : Scraping simultané des 3 sources
3. **Filtrage** : Élimination des doublons et contenu non-pertinent
4. **Enrichissement** : Génération de résumés IA pour chaque article
5. **Persistance** : Sauvegarde avec vérification de doublons
6. **Notification** : Logging du succès/échec

#### Scénarios Alternatifs

* **Échec site** : Continuer avec les sources disponibles
* **Échec IA** : Sauvegarder sans résumé, retry plus tard
* **Échec BDD** : Retry avec backoff exponentiel

### Cas d’Usage 2 : Recherche Avancée Utilisateur

#### Acteurs

* **Primaire** : Utilisateur/Application cliente
* **Secondaire** : ArticlesController, SupabaseService

#### Préconditions

* Articles présents en base
* API accessible

#### Scénario Principal

1. **Requête** : Client envoie requête de recherche
2. **Validation** : Vérification des paramètres
3. **Recherche** : Full-text search en base
4. **Pagination** : Application des limites
5. **Formatage** : Sérialisation JSON standardisée
6. **Réponse** : Retour des résultats

#### Optimisations

* **Cache** : Mise en cache des recherches fréquentes
* **Index** : Utilisation des index PostgreSQL
* **Filtres** : Pré-filtrage par source/date

# 26 - Chapitre 3 : Réalisation

## Vue d’Ensemble de l’Implémentation

La phase de réalisation transforme notre conception en **solution opérationnelle** en exploitant un stack technologique moderne et robuste. Cette implémentation suit une approche **modulaire et scalable** garantissant la maintenabilité et l’évolutivité du système.

## Architecture Technique Globale

### Stack Technologique Choisi

┌─────────────────────────────────────────────────────────┐  
│ FRONTEND/API │  
│ AdonisJS v6 + TypeScript │  
├─────────────────────────────────────────────────────────┤  
│ SERVICES MÉTIER │  
│ ScraperService │ AIService │ SupabaseService │  
├─────────────────────────────────────────────────────────┤  
│ INTÉGRATIONS │  
│ Puppeteer │ Cheerio │ Google Gemini │ Node-Cron │  
├─────────────────────────────────────────────────────────┤  
│ DONNÉES │  
│ Supabase (PostgreSQL) │  
└─────────────────────────────────────────────────────────┘

### Justification des Choix Techniques

#### AdonisJS + TypeScript

* **Robustesse** : Framework enterprise avec TypeScript natif
* **Productivité** : Convention over configuration
* **Moderne** : Support ES modules, async/await natif
* **Écosystème** : ORM, validation, middleware intégrés

#### Supabase

* **Simplicité** : PostgreSQL managé avec API auto-générée
* **Performance** : Index optimisés, requêtes rapides
* **Sécurité** : RLS (Row Level Security) intégré
* **Coût** : Tier gratuit généreux pour développement

#### Google Gemini

* **Qualité** : IA de dernière génération
* **Fiabilité** : Infrastructure Google robuste
* **Coût** : Modèle freemium accessible
* **Innovation** : Mises à jour régulières du modèle

## Principes d’Implémentation

### 1. Séparation des Responsabilités

Chaque service a un **rôle unique et bien défini** :

* **ScraperService** : Collecte et extraction uniquement
* **AIService** : Traitement IA exclusivement
* **SupabaseService** : Persistance et requêtes seulement
* **CronService** : Orchestration et planification

### 2. Gestion d’Erreurs Robuste

**Try-catch systématique** avec logging détaillé et fallback intelligents pour assurer la continuité de service même en cas de problème partiel.

### 3. Configuration Externalisée

Toute la configuration via **variables d’environnement** pour faciliter le déploiement multi-environnements et la sécurité.

# 27 - 3.1 Architecture Technique Détaillée

## Architecture en Couches

### Couche Présentation (API Layer)

#### ArticlesController

export default class ArticlesController {  
 // Endpoints RESTful standards  
 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 d’entrée
* Orchestration des appels aux services
* Formatage des réponses JSON standardisées
* Gestion des codes de statut HTTP

### Couche Logique Métier (Business Layer)

#### Services Principaux

**SupabaseService** - Gestionnaire de données

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

**ScraperService** - Collecteur d’articles

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

**AIService** - Processeur intelligent

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

### Patterns Architecturaux Implémentés

#### 1. Repository Pattern

Le SupabaseService fait office de repository unifié, abstrayant la complexité de l’accès aux données et facilitant les tests unitaires.

#### 2. Factory Pattern

Les scrapers spécialisés par source permettent l’extensibilité facile vers de nouvelles sources.

#### 3. Observer Pattern

Le système de cron jobs observe et déclenche les tâches de scraping selon la planification.

# 28 - 3.2 Technologies d’Intelligence Artificielle

## Intégration Google Gemini

### Configuration et Authentification

#### Setup de l’API

import { GoogleGenerativeAI } from '@google/generative-ai'  
  
export class AIService {  
 private genAI: GoogleGenerativeAI  
 private model: any  
  
 constructor() {  
 this.genAI = new GoogleGenerativeAI(env.get('GEMINI\_API\_KEY'))  
 this.model = this.genAI.getGenerativeModel({ model: 'gemini-pro' })  
 }  
}

### Optimisation des Prompts

#### Template de Prompt Optimisé

private formatPrompt(content: string, title: string): string {  
 return `  
 Analyse l'article suivant et génère un résumé professionnel de 100-150 mots.  
  
 Titre: ${title}  
  
 Contenu: ${content.substring(0, 4000)}  
  
 Exigences du résumé:  
 - Préserver les informations clés  
 - Style journalistique professionnel  
 - Structure logique et cohérente  
 - Langage accessible mais précis  
  
 Résumé:  
 `;  
}

### Gestion des Limitations API

#### Rate Limiting et Retry Logic

async generateSummary(content: string, title: string): Promise<string> {  
 const maxRetries = 3;  
 let attempt = 0;  
  
 while (attempt < maxRetries) {  
 try {  
 const prompt = this.formatPrompt(content, title);  
 const result = await this.model.generateContent(prompt);  
 return result.response.text();  
 } catch (error) {  
 attempt++;  
 if (attempt >= maxRetries) {  
 return this.generateFallbackSummary(content);  
 }  
 await new Promise(resolve => setTimeout(resolve, 1000 \* attempt));  
 }  
 }  
}

### Métriques et Monitoring IA

#### Suivi de la Qualité

* **Longueur moyenne** des résumés générés
* **Temps de traitement** par article
* **Taux de succès** des appels API
* **Coût** estimé par utilisation

#### Optimisations Continues

* **A/B Testing** de différents prompts
* **Feedback loop** pour améliorer la qualité
* **Cache intelligent** pour éviter les appels redondants

# 29 - 3.3 Système de Scraping Web

## Technologies de Scraping

### Stack Puppeteer + Cheerio

#### Puppeteer pour l’Automation

private async scrapeTheVerge(): Promise<ScrapedArticle[]> {  
 const browser = await puppeteer.launch({ headless: true });  
 const page = await browser.newPage();  
  
 try {  
 await page.goto('https://www.theverge.com', {  
 waitUntil: 'networkidle2'  
 });  
 const html = await page.content();  
 const $ = cheerio.load(html);  
 // Parsing avec Cheerio...  
 } finally {  
 await browser.close();  
 }  
}

#### Cheerio pour l’Extraction

$('article h2 a, article h3 a').each((\_, element) => {  
 const $element = $(element)  
 const title = $element.text().trim()  
 const url = $element.attr('href')  
  
 if (title && url) {  
 articles.push({  
 title,  
 url: url.startsWith('http') ? url : `https://www.theverge.com${url}`,  
 source: 'The Verge',  
 publishedAt: new Date(),  
 })  
 }  
})

### Stratégies par Source

#### The Verge

* **Sélecteurs** : article h2 a, article h3 a
* **Défi** : Structure dynamique avec JavaScript
* **Solution** : Puppeteer pour le rendu complet

#### CNET

* **Sélecteurs** : article h3 a, .latestScrollItems h4 a
* **Défi** : Lazy loading des articles
* **Solution** : Attente du chargement réseau

#### Wired

* **Sélecteurs** : .summary-item\_\_hed a, article h3 a
* **Défi** : Paywall et contenu restreint
* **Solution** : Extraction des teasers publics

### Robustesse et Resilience

#### Gestion des Échecs

try {  
 const articles = await this.scrapeTheVerge()  
 return articles  
} catch (error) {  
 console.error('Error scraping The Verge:', error)  
 return [] // Retour gracieux  
}

#### Adaptation aux Changements

* **Sélecteurs multiples** : Fallback si structure change
* **Validation du contenu** : Vérification de la qualité
* **Monitoring** : Alertes en cas de baisse de volume

# 30 - 3.4 Base de Données Supabase

## Architecture de Données

### Schéma de Base de Données

#### Table Articles

CREATE TABLE articles (  
 id UUID DEFAULT gen\_random\_uuid() PRIMARY KEY,  
 title TEXT NOT NULL,  
 url TEXT UNIQUE NOT NULL,  
 content TEXT,  
 summary TEXT,  
 source TEXT NOT NULL,  
 created\_at TIMESTAMPTZ DEFAULT now(),  
 updated\_at TIMESTAMPTZ DEFAULT now()  
);

#### Index pour Performance

-- Recherche full-text  
CREATE INDEX articles\_search\_idx ON articles  
USING gin(to\_tsvector('english', title || ' ' || content || ' ' || summary));  
  
-- Filtrage par source et date  
CREATE INDEX articles\_source\_date\_idx ON articles(source, created\_at DESC);  
  
-- Recherche par URL (déduplication)  
CREATE UNIQUE INDEX articles\_url\_idx ON articles(url);

### Service de Données

#### CRUD Operations

export class SupabaseService {  
 private supabase: SupabaseClient  
  
 async createArticle(data: ArticleData): Promise<Article> {  
 const { data: article, error } = await this.supabase  
 .from('articles')  
 .insert(data)  
 .select()  
 .single()  
  
 if (error) throw error  
 return article  
 }  
  
 async searchArticles(query: string, limit: number = 10): Promise<Article[]> {  
 const { data, error } = await this.supabase  
 .from('articles')  
 .select('\*')  
 .textSearch('title,content,summary', query)  
 .order('created\_at', { ascending: false })  
 .limit(limit)  
  
 if (error) throw error  
 return data || []  
 }  
}

### Optimisations Performance

#### Pagination Efficace

async getArticles(limit: number, offset: number): Promise<Article[]> {  
 const { data, error } = await this.supabase  
 .from('articles')  
 .select('id, title, summary, source, created\_at, url')  
 .order('created\_at', { ascending: false })  
 .range(offset, offset + limit - 1);  
  
 return data || [];  
}

#### Déduplication Intelligente

async getArticleByUrl(url: string): Promise<Article | null> {  
 const { data, error } = await this.supabase  
 .from('articles')  
 .select('\*')  
 .eq('url', url)  
 .single();  
  
 return error ? null : data;  
}

# 31 - 3.5 Tests et Validation

## Stratégie de Tests

### Tests Unitaires

#### Tests des Services

// Test du AIService  
describe('AIService', () => {  
 it('should generate summary for valid content', async () => {  
 const aiService = new AIService()  
 const content = 'Sample article content...'  
 const title = 'Test Article'  
  
 const summary = await aiService.generateSummary(content, title)  
  
 expect(summary).toBeDefined()  
 expect(summary.length).toBeGreaterThan(50)  
 })  
})  
  
// Test du SupabaseService  
describe('SupabaseService', () => {  
 it('should create and retrieve articles', async () => {  
 const service = new SupabaseService()  
 const articleData = {  
 title: 'Test Article',  
 url: 'https://test.com/article',  
 content: 'Test content',  
 source: 'Test Source',  
 }  
  
 const created = await service.createArticle(articleData)  
 const retrieved = await service.getArticleByUrl(articleData.url)  
  
 expect(retrieved?.title).toBe(articleData.title)  
 })  
})

### Tests d’Intégration

#### Test du Pipeline Complet

describe('Complete Article Pipeline', () => {  
 it('should scrape, process and store articles', async () => {  
 const scraperService = new ScraperService()  
  
 // Test sur une source  
 await scraperService.scrapeSource('theverge')  
  
 // Vérifier que des articles ont été créés  
 const supabaseService = new SupabaseService()  
 const articles = await supabaseService.getArticlesBySource('The Verge', 5)  
  
 expect(articles.length).toBeGreaterThan(0)  
 expect(articles[0].summary).toBeDefined()  
 })  
})

### Tests de Performance

#### Benchmarking

describe('Performance Tests', () => {  
 it('should handle concurrent requests', async () => {  
 const startTime = Date.now()  
  
 const promises = Array(10)  
 .fill(0)  
 .map(() => request(app).get('/api/articles').expect(200))  
  
 await Promise.all(promises)  
  
 const duration = Date.now() - startTime  
 expect(duration).toBeLessThan(5000) // < 5 secondes  
 })  
})

### Validation Manuelle

#### Tests API avec curl

# Test de base  
curl -X GET "http://localhost:3333/api/articles" \  
 -H "Accept: application/json"  
  
# Test de recherche  
curl -X GET "http://localhost:3333/api/articles/search?q=AI&limit=5" \  
 -H "Accept: application/json"  
  
# Test de scraping manuel  
curl -X POST "http://localhost:3333/api/scrape" \  
 -H "Content-Type: application/json"

#### Métriques de Qualité

* **Couverture de code** : >80% pour les services critiques
* **Temps de réponse** : <2s pour les requêtes API
* **Taux de succès** : >95% pour le scraping
* **Qualité des résumés** : Validation manuelle sur échantillon

# 32 - Conclusion

## Bilan du Projet

### Objectifs Atteints

Ce projet de fin d’études a permis de **développer avec succès** une solution complète et innovante de scraping et résumé automatique d’articles de presse. Les **objectifs initiaux ont été pleinement réalisés** :

✅ **Automatisation complète** : Collecte autonome depuis 3 sources majeures  
✅ **Intelligence artificielle intégrée** : Résumés de qualité via Google Gemini  
✅ **API RESTful moderne** : Interface standardisée et documentée  
✅ **Architecture robuste** : Solution scalable et maintenable  
✅ **Documentation exhaustive** : Guide complet pour utilisateurs et développeurs

### Apports Techniques et Pédagogiques

#### Compétences Développées

* **Maîtrise des technologies modernes** : AdonisJS, TypeScript, Supabase
* **Intégration IA** : Utilisation pratique des modèles de langage avancés
* **Web scraping avancé** : Techniques robustes avec Puppeteer et Cheerio
* **Architecture logicielle** : Conception modulaire et patterns avancés
* **DevOps et déploiement** : Configuration cloud et automatisation

#### Défis Techniques Surmontés

* **Gestion de la complexité** : Architecture multi-services cohérente
* **Robustesse** : Gestion d’erreurs et récupération automatique
* **Performance** : Optimisation pour le traitement en masse
* **Intégration** : Coordination de multiples APIs externes

### Valeur Ajoutée et Innovation

#### Innovation Technologique

Notre solution se distingue par **l’intégration native de l’IA** dans un pipeline de scraping automatisé, créant une chaîne de valeur complète depuis la collecte brute jusqu’à l’information synthétisée et exploitable.

#### Impact Pratique

Le système développé répond à un **besoin réel du marché** en proposant une alternative moderne, accessible et performante aux solutions existantes, particulièrement adaptée aux PME et startups.

### Perspectives d’Évolution

#### Extensions Fonctionnelles

* **Nouvelles sources** : Intégration d’autres sites d’actualité
* **Analyse sentimentale** : Ajout de l’analyse des sentiments
* **Catégorisation automatique** : Classification intelligente des articles
* **Notifications** : Alertes en temps réel sur des sujets spécifiques

#### Améliorations Techniques

* **Mise en cache avancée** : Redis pour les performances
* **Monitoring complet** : Métriques détaillées et dashboards
* **API GraphQL** : Alternative à REST pour plus de flexibilité
* **Container** : Dockerisation pour faciliter le déploiement

### Remerciements et Reconnaissance

Ce projet n’aurait pu aboutir sans l’**excellence de la formation** dispensée à l’ESRMI et l’**accompagnement bienveillant** du corps professoral. Les compétences acquises en management, ingénierie et technologies modernes ont été déterminantes dans la réussite de cette réalisation.

L’**approche pédagogique innovante** de l’école, combinant théorie et pratique, a permis d’aborder ce projet complexe avec confiance et méthode, en appliquant les meilleures pratiques de l’industrie.

### Conclusion Générale

Ce projet illustre parfaitement la **convergence entre formation académique de qualité et innovation technologique**. Il démontre qu’avec les bonnes bases théoriques et un encadrement adapté, il est possible de créer des solutions techniques sophistiquées répondant aux enjeux contemporains de la transformation digitale.

L’**École Supérieure de Rabat en Management et Ingénierie** forme ainsi des profils capables de **porter l’innovation** et de contribuer activement au développement technologique du Maroc et de la région.

**Fin de Présentation**

*Merci pour votre attention*