

Architecture logicielle - Penpal AI

Ce document présente l'architecture logicielle complète de la plateforme Penpal AI, incluant les schémas d'architecture, les justifications des choix technologiques et les patterns d'implémentation.

Vue d'ensemble de l'architecture

Penpal AI est construit selon une **architecture microservices** permettant une scalabilité horizontale, une maintenance indépendante des services et une résilience élevée. La plateforme se compose de 7 services principaux orchestrés via Docker et déployés sur une infrastructure cloud moderne.

Schéma d'architecture globale

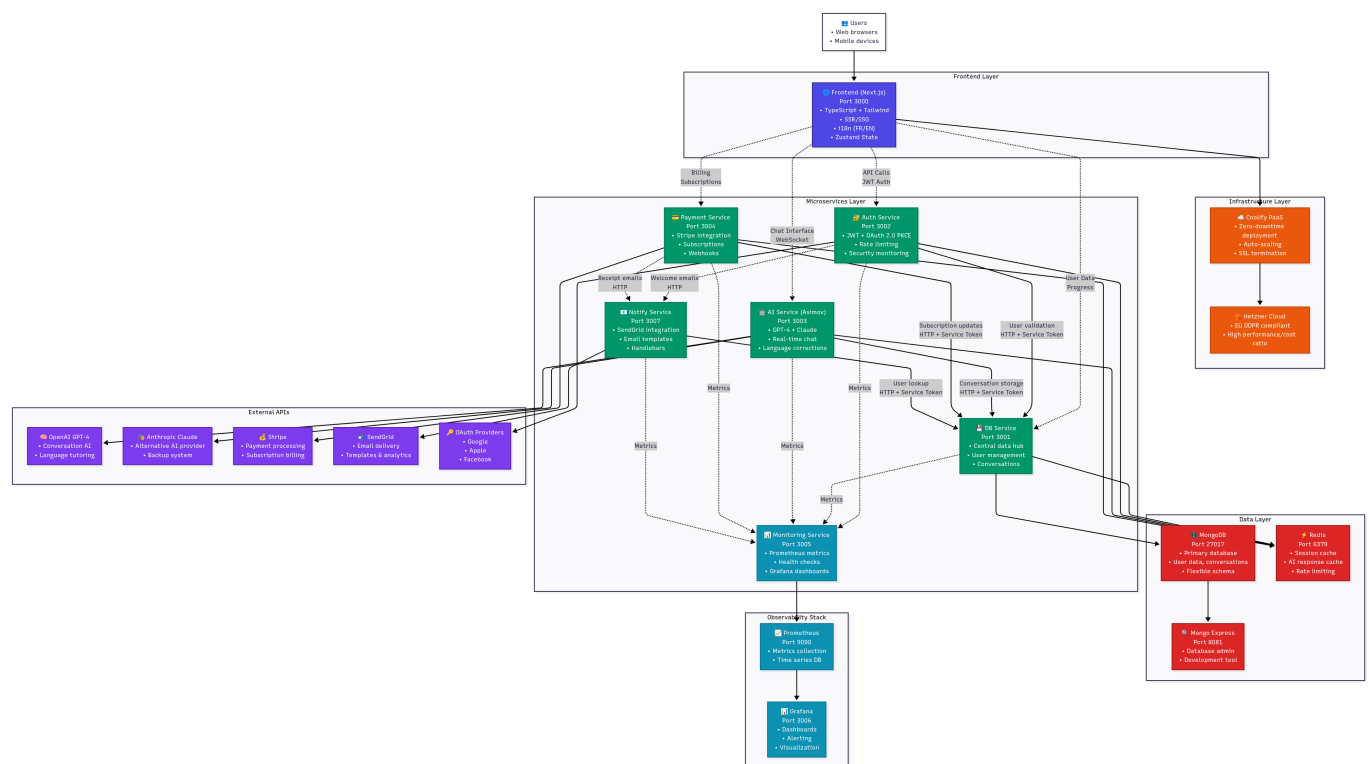


Figure 1 : Vue d'ensemble détaillée de l'architecture microservices Penpal AI

Cette architecture illustre parfaitement :

Layers d'architecture :

- **Frontend Layer** (Bleu) : Interface Next.js avec TypeScript, SSR/SSG et internationalisation
- **Microservices Layer** (Vert) : 6 services NestJS spécialisés avec leurs responsabilités
- **Infrastructure Layer** (Orange) : Coolify PaaS sur Hetzner Cloud pour déploiement
- **Data Layer** (Rouge) : MongoDB comme base principale et Redis pour cache/sessions
- **Observability Stack** (Bleu clair) : Prometheus et Grafana pour monitoring complet
- **External APIs** (Violet) : Services tiers intégrés (OpenAI, Stripe, SendGrid, OAuth)

Communications inter-services :

- **API calls sécurisés** entre frontend et services via JWT
- **HTTP + Service Token** pour communications service-to-service
- **Métriques centralisées** vers le monitoring service
- **Intégrations externes** avec gestion d'erreurs et fallbacks

Schéma des flux de données

 Flux de données *Figure 2 : Flux de données entre les services et composants*

Architecture microservices

Services backend (NestJS)

L'écosystème backend est composé de 6 microservices spécialisés :

1. DB Service (Port 3001)

Rôle : Service central de gestion des données

- **Responsabilités** :
 - Gestion des utilisateurs, rôles et permissions
 - Stockage des conversations et messages
 - Gestion des langues et caractères IA
 - Cache Redis pour les performances
- **Technologies** : NestJS, MongoDB, Redis, Mongoose
- **Pattern** : Single Source of Truth pour les données

2. Auth Service (Port 3002)

Rôle : Authentification et autorisation

- **Responsabilités** :
 - Inscription/connexion utilisateurs
 - OAuth 2.0 avec PKCE (Google, Apple, Facebook)
 - JWT avec gestion sécurisée des tokens
 - Rate limiting et protection CSRF
- **Technologies** : NestJS, JWT, OAuth 2.0, bcrypt
- **Pattern** : Security-first avec audit trails

3. AI Service - Asimov (Port 3003)

Rôle : Intelligence artificielle conversationnelle

- **Responsabilités** :
 - Conversations temps réel avec GPT-4/Claude
 - Corrections linguistiques intelligentes
 - Templates de prompts dynamiques
 - Analytics et suivi des progressions
- **Technologies** : NestJS, OpenAI API, Anthropic Claude, WebSocket

- **Pattern** : Provider abstraction pour multi-IA

4. Payment Service (Port 3004)

Rôle : Gestion des paiements et abonnements

- **Responsabilités** :
 - Intégration Stripe pour paiements
 - Gestion des abonnements et essais gratuits
 - Webhooks pour synchronisation état
 - Billing et facturation automatisée
- **Technologies** : NestJS, Stripe API, Webhooks
- **Pattern** : Event-driven pour transactions

5. Monitoring Service (Port 3005)

Rôle : Observabilité et métriques

- **Responsabilités** :
 - Collecte métriques Prometheus
 - Health checks des services
 - Dashboards Grafana intégrés
 - Alerting automatisé
- **Technologies** : NestJS, Prometheus, Grafana
- **Pattern** : Centralized monitoring

6. Notify Service (Port 3007)

Rôle : Notifications et communications

- **Responsabilités** :
 - Emails transactionnels (inscription, mot de passe)
 - Templates Handlebars pour personnalisation
 - Intégration SendGrid pour délivrabilité
 - Gestion des préférences de notification
- **Technologies** : NestJS, SendGrid, Handlebars
- **Pattern** : Template-based messaging

Service frontend (Next.js)

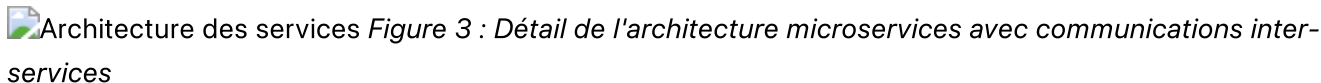
Frontend Application (Port 3000)

Rôle : Interface utilisateur et expérience client

- **Responsabilités** :
 - SPA avec rendu côté serveur (SSR)
 - Interface de chat temps réel
 - Gestion d'état avec Zustand
 - Internationalisation (FR/EN)

- **Technologies** : Next.js, TypeScript, Tailwind CSS, i18n
- **Pattern** : JAMstack avec API-first

Schéma d'architecture des services

 Architecture des services *Figure 3 : Détail de l'architecture microservices avec communications inter-services*

Choix technologiques et justifications

Frontend : Next.js + TypeScript

Next.js (Framework React)

Justifications :

- **Performance** : SSR/SSG pour SEO et temps de chargement optimaux
- **Developer Experience** : Hot reload, optimisations automatiques, API routes
- **Ecosystem** : Large communauté, intégrations tierces riches
- **Scalabilité** : Support natif du déploiement serverless et edge computing
- **SEO** : Rendu côté serveur crucial pour acquisition utilisateurs

```
{
  "name": "penpal-ai",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
```

TypeScript

Justifications :

- **Type Safety** : Détection d'erreurs à la compilation, réduction bugs production
- **Refactoring** : Modifications sûres à grande échelle
- **Developer Experience** : Auto-complétion, IntelliSense, documentation inline
- **Maintenabilité** : Code plus lisible et auto-documenté
- **Ecosystem** : Support natif Next.js, librairies typées

Backend : NestJS + Node.js

NestJS (Framework Node.js)

Justifications :

- **Architecture Enterprise** : Modules, dependency injection, decorators pour scalabilité
- **TypeScript First** : Cohérence technologique avec le frontend

- **Microservices Ready** : Support natif gRPC, Redis, message queues
- **Security Built-in** : Guards, interceptors, pipes pour sécurité par défaut
- **Testing** : Framework de test intégré avec mocking avancé

This service follows a microservice architecture pattern where:

- **Auth Service** (this service): Manages authentication, user registration, and OAuth integrations with advanced security
- **DB Service** (separate): Handles all database operations and is the only service with direct database access
- **Notification Service** (separate): Handles email notifications including welcome emails for new users

The Auth Service communicates with the DB Service and Notification Service via HTTP requests, maintaining a clear separation of concerns.

Node.js

Justifications :

- **Performance I/O** : Event loop non-bloquant pour applications temps réel
- **Ecosystem** : npm avec packages riches pour IA, paiements, notifications
- **Unified Language** : JavaScript/TypeScript sur frontend et backend
- **Real-time** : Support natif WebSocket pour chat temps réel
- **Deployment** : Containerisation Docker simple et efficace

Bases de données : MongoDB + Redis

MongoDB (Base principale)

Justifications :

- **Schema Flexibility** : Évolution agile du modèle de données
- **JSON Native** : Alignement parfait avec APIs REST/GraphQL
- **Scalabilité Horizontale** : Sharding natif pour croissance future
- **Rich Queries** : Agrégations complexes pour analytics utilisateurs
- **Development Speed** : Pas de migrations complexes, itération rapide

Architecture

The application is built with NestJS and uses:

- **MongoDB** as the primary database
- **Redis** as a caching system
- **JWT** for authentication
- **Swagger** for API documentation

Data Model

![Data Model](MVP-penpal-ai-diagram.png)

- ****Users****: User management, profiles, and preferences
- ****Roles****: Role-based access control
- ****Languages****: Languages available for learning
- ****AI Characters****: AI personalities to converse with
- ****Conversations****: Exchanges between users and AI
- ****Messages****: Conversation content

Redis (Cache et sessions)

Justifications :

- **Performance** : Cache in-memory pour réponses sub-milliseconde
- **Sessions** : Stockage tokens JWT et états temporaires
- **Rate Limiting** : Compteurs distribués pour protection API
- **Real-time** : Pub/Sub pour notifications temps réel
- **AI Caching** : Cache des réponses IA pour optimisation coûts

Communication : SendGrid

SendGrid (Service email)

Justifications :

- **Deliverability** : Réputation IP et domaine pour inbox placement
- **Scale** : Gestion millions d'emails sans infrastructure
- **Templates** : Système de templates professionnels
- **Analytics** : Métriques détaillées (ouvertures, clics, bounces)
- **Compliance** : GDPR, CAN-SPAM automatiquement gérés

Patterns architecturaux

1. API Gateway Pattern

Le frontend Next.js agit comme un API Gateway léger :

```
// Pattern de proxy API pour centraliser les appels
export async function apiCall(endpoint: string, options: RequestInit) {
  const response = await fetch(`/api/proxy${endpoint}`, {
    ...options,
    headers: {
      'Authorization': `Bearer ${getToken()}`,
      'Content-Type': 'application/json',
      ...options.headers,
    },
  });
  return response.json();
}
```

2. Service Discovery Pattern

Configuration centralisée des services via environnement :

```
# Docker Compose service discovery
services:
  db-service:
    ports: ["3001:3001"]
  auth-service:
    ports: ["3002:3002"]
  ai-service:
    ports: ["3003:3003"]
```

3. Circuit Breaker Pattern

Protection contre les défaillances en cascade :

```
// Exemple dans ai-service pour OpenAI API
export class CircuitBreaker {
  async callWithBreaker(fn: Function) {
    if (this.isOpen()) {
      throw new Error('Circuit breaker is open');
    }
    try {
      const result = await fn();
      this.onSuccess();
      return result;
    } catch (error) {
      this.onFailure();
      throw error;
    }
  }
}
```

4. CQRS Pattern

Séparation lecture/écriture pour les données complexes :

```
// Command pour écriture
export class CreateConversationCommand {
  constructor(
    public readonly userId: string,
    public readonly aiCharacterId: string,
    public readonly language: string,
  ) {}
}
```

```
// Query pour lecture optimisée
export class GetUserConversationsQuery {
  constructor(
    public readonly userId: string,
    public readonly limit: number,
    public readonly offset: number,
  ) {}
}
```

Architecture de déploiement

Containerisation : Docker

Dockerfile Multi-stage

Justifications :

- **Optimisation** : Images finales légères sans dépendances dev
- **Sécurité** : Images minimales réduisent surface d'attaque
- **Performance** : Layers cachés pour builds rapides
- **Consistency** : Environnements identiques dev/staging/prod

```
# Exemple Dockerfile multi-stage
FROM node:20-alpine AS base
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

FROM node:20-alpine AS development
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
CMD ["npm", "run", "start:dev"]

FROM base AS production
COPY --from=development /app/dist ./dist
CMD ["npm", "run", "start:prod"]
```

Orchestration : Docker Compose

Configuration des services et dépendances :

```
services:
  db-service:
    build:
      context: ..
```



```

    dockerfile: Dockerfile
    target: development
    container_name: penpal-ai-db-service
    volumes:
      - ../:/app
      - /app/node_modules
    ports:
      - "3001:3001"
    command: npm run start:dev
    env_file:
      - ./db-service/.env
    depends_on:
      mongodb:
        condition: service_healthy
      redis:
        condition: service_healthy
    networks:
      - penpal-network
    healthcheck:
      test: [CMD, curl, -f, "http://localhost:3001/api/v1/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

```

Infrastructure : Hetzner + Coolify

Hetzner Cloud

Justifications :

- **Performance/Prix** : Excellent ratio performance/coût vs AWS/GCP
- **Latence Europe** : Serveurs européens pour conformité GDPR
- **Simplicité** : Interface claire, pas de vendor lock-in
- **Scaling** : Augmentation ressources à la demande
- **Network** : Bande passante généreuse incluse

Coolify (PaaS)

Justifications :

- **Simplicité DevOps** : Déploiement git-push sans complexité Kubernetes
- **Self-hosted** : Contrôle total infrastructure vs Vercel/Netlify
- **Multi-environment** : Staging/production avec promotion automatique
- **Zero-downtime** : Rolling deployments sans interruption service
- **Cost-effective** : Une instance vs orchestrateurs entreprise

Schéma d'infrastructure de déploiement



Infrastructure Hetzner *Figure 4 : Architecture de déploiement sur Hetzner avec Coolify*

Monitoring et observabilité

Prometheus + Grafana

Configuration intégrée pour métriques temps réel :

```
prometheus:
  image: prom/prometheus:latest
  container_name: penpal-prometheus
  restart: unless-stopped
  ports:
    - "9090:9090"
  volumes:
    - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
    - prometheus_data:/prometheus
  command:
    - --config.file=/etc/prometheus/prometheus.yml
    - --storage.tsdb.path=/prometheus
    - --web.console.libraries=/etc/prometheus/console_libraries
    - --web.console.templates=/etc/prometheus/consoles
    - --storage.tsdb.retention.time=200h
    - --web.enable-lifecycle
  networks:
    - penpal-network
  depends_on:
    - monitoring-service
```

Schéma de monitoring



Monitoring Stack *Figure 5 : Stack de monitoring avec Prometheus, Grafana et alerting*

Sécurité et conformité

Authentification et autorisation

JWT + OAuth 2.0 avec PKCE

- **Tokens stateless** : Scalabilité sans state serveur
- **PKCE** : Protection contre interception codes d'autorisation
- **Refresh tokens** : Sécurité accrue avec rotation automatique
- **Rate limiting** : Protection contre attaques brute force

Architecture de sécurité multicouche

```
// Guard NestJS pour authentication service-to-service
@Injectable()
export class ServiceAuthGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const request = context.switchToHttp().getRequest();
```

```

    const serviceToken = request.headers['x-service-token'];

    return this.validateServiceToken(serviceToken);
  }
}

```

Protection des données

RGPD Compliance

- **Chiffrement** : Données sensibles chiffrées au repos et en transit
- **Pseudonymisation** : Identifiants utilisateurs non réversibles
- **Right to be forgotten** : APIs de suppression complète
- **Data minimization** : Collecte limitée aux besoins fonctionnels

Schéma de sécurité

 Architecture de sécurité *Figure 6 : Layers de sécurité et flux d'authentification*

Scalabilité et performances

Horizontal Scaling

Microservices indépendants

- **Scaling sélectif** : Scale uniquement les services sous charge
- **Load balancing** : Distribution charge via reverse proxy
- **Database sharding** : Partition MongoDB par région/utilisateur
- **CDN** : Assets statiques distribués mondialement

Optimisations performances

```

// Cache Redis pour réponses IA coûteuses
export class AIResponseCache {
  async getCacheResponse(prompt: string, context: string): Promise<string
| null> {
    const cacheKey = this.generateCacheKey(prompt, context);
    return await this.redis.get(cacheKey);
  }

  async setCacheResponse(prompt: string, context: string, response:
string): Promise<void> {
    const cacheKey = this.generateCacheKey(prompt, context);
    await this.redis.setex(cacheKey, 3600, response); // 1h TTL
  }
}

```

Schéma de scalabilité

 Stratégie de scalabilité *Figure 7 : Stratégie de scalabilité horizontale et optimisations*

Pipeline CI/CD et DevOps

Intégration continue

- **Tests automatisés** : Unitaires, intégration, E2E pour chaque service
- **Quality gates** : Linting, coverage >80%, security scanning
- **Multi-environment** : Dev → Staging → Production avec promotion automatique

Déploiement continu

- **Blue-green deployment** : Zero-downtime via Coolify
- **Rollback automatique** : Retour version précédente en cas d'échec
- **Feature flags** : Déploiement progressif nouvelles fonctionnalités

Schéma CI/CD

 Pipeline CI/CD *Figure 8 : Pipeline d'intégration et déploiement continu*

Architecture future et évolutions

Roadmap technique

Court terme (3-6 mois)

- **WebSocket** : Chat temps réel avec notifications push
- **GraphQL** : APIs unifiées pour frontend mobile
- **Kafka** : Message streaming pour events cross-services
- **Redis Cluster** : High availability cache distribué

Moyen terme (6-12 mois)

- **Kubernetes** : Migration orchestrateur entreprise
- **Service Mesh** : Istio pour communication inter-services
- **Event Sourcing** : Historique complet des événements utilisateur
- **Multi-région** : Déploiement global avec data residency

Long terme (12+ mois)

- **Edge Computing** : Serveurs régionaux pour latence minimale
- **AI/ML Pipeline** : MLOps pour modèles personnalisés
- **Blockchain** : Tokens pour gamification apprentissage
- **Voice AI** : Reconnaissance vocale pour pratique orale

Schéma d'évolution architecture

 Roadmap architecture *Figure 9 : Évolution planifiée de l'architecture vers une solution entreprise*

Métriques et KPIs techniques

Performance

- **Response time** : <200ms API calls, <2s page load
- **Throughput** : 1000+ req/sec par service en pic
- **Availability** : 99.9% uptime avec SLA monitoring
- **Error rate** : <0.1% erreurs 5xx en production

Scalabilité

- **Horizontal scaling** : Auto-scaling basé métriques CPU/memory
- **Database performance** : <100ms query time 95e percentile
- **Cache hit ratio** : >90% pour données fréquemment accédées
- **AI API costs** : Optimisation via caching et prompt engineering

Développement

- **Deployment frequency** : Multiple deployments/jour sans friction
- **Lead time** : <1h feature → production en moyenne
- **MTTR** : <15min temps de résolution incidents
- **Code coverage** : >80% lignes couvertes par tests automatisés

Conclusion

L'architecture Penpal AI combine **modernité technologique** et **pragmatisme opérationnel** :

- **Microservices** pour flexibilité et scalabilité indépendante
- **Stack TypeScript** pour cohérence et productivité développeurs
- **Technologies éprouvées** (Next.js, NestJS, MongoDB) pour stabilité
- **Infrastructure cloud** optimisée coût/performance avec Hetzner
- **DevOps simplifié** via Coolify sans complexité excessive
- **Sécurité by design** avec authentification moderne et protection données

Cette architecture permet de **délivrer rapidement de la valeur** tout en préparant une **croissance à long terme** vers une plateforme entreprise d'apprentissage linguistique par IA.