Plan de correction de bogue

Ce document décrit la méthodologie et les processus de correction de bogue pour les services du projet, en prenant `penpal-ai-db-service` comme exemple de référence. Il couvre l'identification, la classification, la reproduction, la correction et la vérification des bogues.

Vue d'ensemble

Le plan de correction s'appuie sur :

- **Templates d'issues GitHub** pour un signalement standardisé
- **Tests automatisés** (unitaires, E2E, couverture) pour validation
- **Logs structurés** pour diagnostic et traçabilité
- **Environnements multiples** (staging/production) pour tests sécurisés
- **Pipeline CI/CD** pour déploiement automatisé des corrections

Classification des bogues

Niveaux de sévérité

1. **Critique** : Service indisponible, perte de données, faille de sécurité
    - **SLA** : Correction sous 2h (production), 4h (staging)
    - **Process** : Hotfix immédiat, rollback si nécessaire
2. **Majeur** : Fonctionnalité principale cassée, impact significatif utilisateurs
    - **SLA** : Correction sous 24h (production), 48h (staging)
    - **Process** : Correction prioritaire sur develop → main
3. **Mineur** : Fonctionnalité secondaire affectée, contournement possible
    - **SLA** : Correction sous 1 semaine
    - **Process** : Correction dans le sprint courant
4. **Cosmétique** : Interface, performance non-critique
    - **SLA** : Correction planifiée selon roadmap
    - **Process** : Backlog, traitement différé

Processus de signalement

Template de rapport de bogue

Utilisation du template GitHub `.github/ISSUE_TEMPLATE/bug-report.md` :

```
## Description du bug

<!-- Une description claire et concise du bug -->
```

```
## Étapes pour reproduire

1. Aller à '...'
2. Cliquer sur '....'
3. Faire défiler jusqu'à '....'
4. Observer l'erreur

## Comportement attendu

<!-- Une description claire et concise de ce que vous attendiez qu'il se
passe -->

## Comportement actuel

<!-- Ce qui se passe actuellement -->

## Captures d'écran

<!-- Si applicable, ajoutez des captures d'écran pour aider à expliquer
votre problème -->

## Environnement

- Appareil: [ex. iPhone 12, MacBook Pro]
- OS: [ex. iOS 15, Windows 11]
- Navigateur: [ex. Chrome, Safari]
- Version: [ex. 22]

## Contexte additionnel

<!-- Ajoutez tout autre contexte concernant le problème ici -->

## Impact

- Sévérité: [Critique/Majeur/Mineur/Cosmétique]
- Utilisateurs affectés: [Tous/Certains/Peu]

## Possible solution

<!-- Si vous avez une idée de la solution, décrivez-la ici -->
```

Informations obligatoires

- **Description claire** : Comportement observé vs attendu
- **Reproduction** : Étapes détaillées et reproductibles
- **Environnement** : Staging/Production, version, navigateur, OS
- **Impact** : Sévérité et nombre d'utilisateurs affectés
- **Logs** : Extraits pertinents (sans données sensibles)

Processus d'investigation

1. Reproduction locale

```
# Cloner et configurer l'environnement local
git clone <repo> && cd penpal-ai-db-service
npm ci
cp .env.example .env

# Démarrer les dépendances (MongoDB, Redis)
cd compose && docker-compose up -d mongodb redis

# Lancer le service en mode debug
npm run start:dev
```

2. Analyse des logs

Le service utilise un système de logging structuré :

```
    logger: ["error", "warn", "log", "debug", "verbose"], // Active tous
les niveaux de log en développement
  });
  const configService = app.get(ConfigService);

  // Configuration du niveau de log basé sur l'environnement
  const logLevel = configService.get<string>("LOG_LEVEL") || "debug";
  logger.log(`Application running with LOG_LEVEL: ${logLevel}`);

  // Interceptor global pour le logging
  if (process.env.NODE_ENV !== "production") {
    app.useGlobalInterceptors(new LoggingInterceptor());
    logger.log("Logging interceptor enabled for detailed request/response
logging");
  }
```

Sources de logs :

- **Logs applicatifs** : `LoggingInterceptor` pour requêtes/réponses
- **Logs MongoDB** : Connexion, erreurs de requête, stats DB
- **Logs métier** : Services, contrôleurs, erreurs business
- **Logs d'infrastructure** : Coolify, Docker, systèmes de monitoring

Commandes utiles :

```
# Logs en temps réel (Coolify)
# Via interface web Coolify -> Application -> Logs

# Logs locaux avec niveau debug
export LOG_LEVEL=debug
npm run start:dev
```

```
# Filtrage des logs par type d'erreur
grep -i "error\|exception\|failed" logs/app.log
```

3. Tests de régression

Avant toute correction, s'assurer que les tests existants passent :

```
# Tests complets (comme en CI)
npm run lint
npm test -- --runInBand
npm run test:cov -- --runInBand
npm run test:e2e -- --runInBand
```

Processus de correction

1. Analyse de l'impact

- **Modules affectés** : Identifier les services/contrôleurs impactés
- **Dépendances** : API externes, base de données, cache Redis
- **Régression potentielle** : Fonctionnalités susceptibles d'être cassées
- **Compatibilité** : Impact sur les autres services du projet

2. Stratégie de correction

Option A : Hotfix (sévérité critique/majeure)

```
# Créer une branche hotfix depuis main
git checkout main
git pull origin main
git checkout -b hotfix/ISSUE-123-description-courte

# Développer la correction minimale
# Ajouter/modifier les tests
# Vérifier la correction localement

# Push et créer PR vers main
git push origin hotfix/ISSUE-123-description-courte
```

Option B : Correction standard (sévérité mineure/cosmétique)

```
# Créer une branche feature depuis develop
git checkout develop
git pull origin develop
git checkout -b fix/ISSUE-123-description-courte
```

```
# Développer la correction
# Ajouter/modifier les tests
# Push et créer PR vers develop
```

3. Développement de la correction

1. **Code minimal** : Corriger uniquement le problème identifié
2. **Tests unitaires** : Ajouter des tests couvrant le cas d'erreur
3. **Tests E2E** : Valider le workflow complet si nécessaire
4. **Documentation** : Mettre à jour si changement d'API

Exemple de test unitaire pour correction :

```
// test/modules/users/users.service.spec.ts
describe('UserService - Bug #123 Correction', () => {
  it('should handle null email gracefully', async () => {
    // Test du cas d'erreur spécifique
    const userData = { name: 'Test', email: null };

    await expect(userService.createUser(userData))
      .rejects
      .toThrow('Email is required');
  });

  it('should create user with valid email after fix', async () => {
    // Test du cas de succès
    const userData = { name: 'Test', email: 'test@example.com' };

    const result = await userService.createUser(userData);
    expect(result.email).toBe('test@example.com');
  });
});
```

Processus de validation

1. Tests automatisés

Vérification que tous les tests CI passent :

```
# Séquence exacte du pipeline CI
npm ci
npm run lint
npm test -- --runInBand
npm run test:cov -- --runInBand
npm run test:e2e -- --runInBand
```

2. Test en staging

```
# Déploiement automatique sur push develop
git push origin fix/ISSUE-123-description-courte

# Vérification staging
curl -H "Authorization: Bearer $STAGING_API_KEY" \
  https://staging.db-service.penpal-ai.maksou.dev/api/v1/health

# Tests manuels sur staging."service".penpal-ai.maksou.dev
```

3. Validation production

```
# Merge vers main déclenche déploiement production
git checkout main
git merge fix/ISSUE-123-description-courte
git push origin main

# Monitoring post-déploiement
curl -H "Authorization: Bearer $PROD_API_KEY" \
  https://prod.db-service.penpal-ai.maksou.dev/api/v1/health

# Vérification métriques et logs (Coolify/Grafana)
```

Suivi et communication

1. Documentation de la correction

Mise à jour de l'issue GitHub avec :

- **Root cause** : Cause racine identifiée
- **Solution** : Description technique de la correction
- **Impact** : Changements de comportement éventuels
- **Tests** : Tests ajoutés/modifiés
- **Déploiement** : Statut staging/production

2. Release notes

Pour les corrections majeures, ajouter dans les notes de version :

```
## [v1.2.3] - 2024-01-15

### Fixed
- [#123] Correction gestion des emails null dans la création d'utilisateur
- [#124] Résolution timeout connexion MongoDB en production
```

3. Post-mortem (sévérité critique)

Pour les incidents critiques, réaliser un post-mortem incluant :

- **Timeline** : Chronologie de l'incident
- **Impact** : Durée, utilisateurs affectés, perte éventuelle
- **Root cause** : Analyse technique détaillée
- **Correction** : Solution mise en place
- **Prévention** : Actions pour éviter la récurrence

Outils et ressources

Debugging local

```
# Variables d'environnement debug
export LOG_LEVEL=debug
export NODE_ENV=development

# Lancement avec debugger Node.js
npm run start:debug

# Tests avec output détaillé
npm test -- --verbose --runInBand
```

Monitoring production

- **Logs** : Interface Coolify, filtrage par niveau/service
- **Métriques** : Grafana dashboards (si configuré)
- **Alerting** : Notifications automatiques via webhooks
- **Health checks** : `/api/v1/health` endpoints

Base de connaissances

- **Issues GitHub** : Historique des problèmes et solutions
- **Runbooks** : Procédures de diagnostic spécifiques
- **Documentation API** : Swagger `/docs` pour validation endpoints
- **Tests E2E** : `test/` dossier avec scénarios de régression

Procédures d'urgence

Rollback rapide

```
# Option 1: Via Coolify (interface web)
# Sélectionner version précédente et redéployer

# Option 2: Revert Git + redéploiement
git revert <commit-hash>
git push origin main  # Déclenche redéploiement auto
```

Communication d'incident

1. **Équipe technique** : Slack/Teams avec contexte immédiat
2. **Management** : Email avec impact et ETA de résolution
3. **Utilisateurs** : Status page si impact client significatif
4. **Stakeholders** : Mise à jour périodique selon sévérité

Escalation

- **< 30min** : Tentative de résolution par développeur assigné
- **< 2h** : Escalation vers senior/lead developer
- **< 4h** : Implication de l'équipe infrastructure/DevOps
- **> 4h** : Escalation management + communication externe

Métriques et amélioration continue

KPI de suivi

- **MTTR** (Mean Time To Recovery) : Temps moyen de résolution
- **MTBF** (Mean Time Between Failures) : Fiabilité du service
- **Bug escape rate** : % de bugs atteignant la production
- **Test coverage** : Couverture de code par les tests

Amélioration des processus

- **Retrospectives** : Analyse mensuelle des incidents
- **Automation** : Extension de la couverture de tests automatisés
- **Monitoring** : Ajout d'alertes préventives
- **Formation** : Sessions de partage sur les patterns de bugs fréquents

Référence

- Templates d'issues : `penpal-ai-db-service/.github/ISSUE_TEMPLATE/`
- Pipeline CI : `penpal-ai-db-service/.github/workflows/ci.yml`
- Configuration logging : `penpal-ai-db-service/src/main.ts`
- Tests : `penpal-ai-db-service/test/` et `penpal-ai-db-service/src/**/*.spec.ts`