HƯớng Dẫn TĂNG CườNG BẢO MẬT VÀ ỔN ĐỊNH HỆ THỐNG

Tổng Quan

Tài liệu này cung cấp hướng dẫn chi tiết để triển khai hệ thống tạp chí khoa học trong **môi trường mạng nội bộ quân đội (intranet)**, với các cơ chế phòng ngừa lỗi toàn diện.

I. HỆ Thống Cơ Sở Đã Triển Khai

1. Tầng Xử Lý Lỗi Thống Nhất (lib/error-handler.ts)

Chức năng:
- Xử lý tất cả các loại lỗi: Zod, Prisma, Custom Errors
- Trả về phản hồi JSON có cấu trúc rõ ràng
- Không bao giờ trả 500 mù mờ

Cách sử dụng:

```
import { handleError, ValidationError, NotFoundError } from '@/lib/error-handler';

export async function POST(req: NextRequest) {
  try {
    // Your logic here
    if (!data) {
      throw new ValidationError('Thiếu dữ liệu bắt buộc');
    }

    const result = await prisma.submission.findUnique({ where: { id } });
    if (!result) {
      throw new NotFoundError('Không tìm thấy bài nộp');
    }

    return NextResponse.json({ success: true, data: result });
  } catch (error) {
    return handleError(error, 'API_YOUR_ROUTE');
  }
}
```

2. Hệ Thống Logging Có Cấu Trúc (lib/logger.ts)

Tính năng:
- Ghi log ra console với màu sắc
- Tùy chọn ghi ra file (cho môi trường nội bộ)
- Hỗ trợ nhật ký theo ngày (logs/app-YYYY-MM-DD.log)

Cách sử dụng:

```
import { logger } from '@/lib/logger';

// Các mức log
logger.info('Thông tin chung');
logger.warn('Cảnh báo');
logger.error({ context: 'API_ERROR', error: 'Chi tiết lỗi', userId });

// Log chuyên biệt
logger.api('POST', '/api/submissions', { userId });
logger.db('create', 'Submission', { id });
logger.auth('login', userId);
logger.security('unauthorized_access', { ip, path });
```

Bật logging ra file:

Trong `.env`:

```
ENABLE_FILE_LOGGING=true
LOG_DIR=/var/log/tapchi
```

3. Authentication & Authorization Guards (lib/api-guards.ts)

Các hàm bảo vệ:

```
import { requireAuth, requireRole, requireEditor, requireAdmin } from '@/lib/api-guards';

// Yêu cầu đăng nhập
const session = await requireAuth(request);

// Yêu cầu vai trò cụ thể
const session = await requireRole(['EIC', 'MANAGING_EDITOR'], request);

// Yêu cầu là biên tập
const session = await requireEditor(request);

// Yêu cầu là quản trị viên
const session = await requireAdmin(request);

// Kiểm tra quyền truy cập tài nguyên
assertCanAccessResource(session, ownerId, ['EIC'], 'Không có quyền');
```

4. Validation Schemas (lib/validators.ts)

Các schema đã định nghĩa:

- `createSubmissionSchema` - Tạo bài nộp mới
- `submitReviewSchema` - Nộp đánh giá
- `createDecisionSchema` - Tạo quyết định
- `createIssueSchema` - Tạo số tạp chí
- `updateProfileSchema` - Cập nhật hồ sơ
- `changePasswordSchema` - Đổi mật khẩu

Cách sử dụng:

```
import { createSubmissionSchema } from '@/lib/validators';

const validatedData = createSubmissionSchema.parse(inputData);
// Nếu không hợp lệ, tự động throw ZodError (sẽ được handleError xử lý)
```

II. API Health Check

Endpoint: GET /api/health

Trạng thái trả về:
- `200` - healthy (hệ thống hoạt động tốt)
- `200` - degraded (có cảnh báo, nhưng vẫn hoạt động)
- `503` - unhealthy (hệ thống lỗi)

Phản hồi mẫu:

```
{
  "status": "healthy",
  "timestamp": "2025-01-15T10:30:00.000Z",
  "uptime": 3600,
  "checks": {
    "database": {
      "status": "ok",
      "latency": 45,
      "message": "Connected (45ms)"
    },
    "memory": {
      "status": "ok",
      "usage": {
        "used": 256,
        "total": 512,
        "percentage": 50
      }
    }
  },
  "version": "1.0.0",
  "environment": "production"
}
```

Cài Đặt Cron Job (Tự Động Khởi Động Lại)

Bước 1: Sao chép script cron

```
sudo cp scripts/cron-health-check.sh /usr/local/bin/tapchi-health-check.sh
sudo chmod +x /usr/local/bin/tapchi-health-check.sh
```

Bước 2: Chỉnh sửa script (nếu cần)

Mở file `/usr/local/bin/tapchi-health-check.sh` và cập nhật:
- `APP_URL` - URL của ứng dụng
- Phần restart (systemd, PM2, hoặc Docker)

Bước 3: Thêm vào crontab

```
sudo crontab -e
```

Thêm dòng:

```
*/5 * * * * /usr/local/bin/tapchi-health-check.sh
```

(Kiểm tra mỗi 5 phút)

III. Scripts Chẩn Đoán Hệ Thống

1. Kiểm Tra Tổng Thể (scripts/diagnostics/check-system.ts)

Chạy script:

```
cd nextjs_space
yarn tsx scripts/diagnostics/check-system.ts
```

Kiểm tra:
- Kết nối database
- Bảng dữ liệu
- Biến môi trường
- Hệ thống file
- Các vấn đề thường gặp

Kết quả: Lưu tại `logs/diagnostics/diagnostic-[timestamp].json`

2. Kiểm Tra Hệ Thống Nộp Bài (scripts/diagnostics/check-submissions.ts)

```
yarn tsx scripts/diagnostics/check-submissions.ts
```

Phát hiện:
- Bài nộp không có file
- Bài bị kẹt trong phản biện
- Bài chưa có chuyên mục
- Tiêu đề trùng lặp

Lịch chạy tự động: Thêm vào crontab

```
0 2 * * * cd /path/to/app/nextjs_space && yarn tsx scripts/diagnostics/check-submissions.ts >> /var/log/tapchi-diagnostics.log 2>&1
```

IV. Mẫu API Route Với Full Error Handling

Template API Route Hoàn Chỉnh

```typescript
/**
 * API Route Example with Full Error Handling
 * /app/api/example/route.ts
 */

import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/lib/prisma';
import { logger } from '@/lib/logger';
import { handleError, ValidationError, NotFoundError } from '@/lib/error-handler';
import { requireAuth, assertCanAccessResource } from '@/lib/api-guards';
import { z } from 'zod';

// 1. Định nghĩa schema validation
const createSchema = z.object({
  name: z.string().min(3, 'Tên phải có ít nhất 3 ký tự'),
  description: z.string().optional(),
});

// 2. POST Handler
export async function POST(request: NextRequest) {
  try {
    // Authenticate
    const session = await requireAuth(request);

    logger.api('POST', '/api/example', { userId: session.user.id });

    // Parse & validate
    const body = await request.json();
    const validatedData = createSchema.parse(body);

    // Business logic
    const result = await prisma.yourModel.create({
      data: {
        ...validatedData,
        createdBy: session.user.id,
      },
    });

    logger.info({
      context: 'EXAMPLE_CREATED',
      id: result.id,
      userId: session.user.id,
    });

    return NextResponse.json({
      success: true,
      data: result,
    });
  } catch (error) {
    return handleError(error, 'API_EXAMPLE_POST');
  }
}

// 3. GET Handler
export async function GET(request: NextRequest) {
  try {
    const session = await requireAuth(request);

    logger.api('GET', '/api/example', { userId: session.user.id });

    // Query params
    const searchParams = request.nextUrl.searchParams;
```

```
    const id = searchParams.get('id');

    if (!id) {
      throw new ValidationError('Thiếu tham số id');
    }

    // Fetch data
    const result = await prisma.yourModel.findUnique({
      where: { id },
    });

    if (!result) {
      throw new NotFoundError('Không tìm thấy dữ liệu');
    }

    // Check permissions
    assertCanAccessResource(
      session,
      result.ownerId,
      ['EIC', 'MANAGING_EDITOR'],
      'Không có quyền truy cập'
    );

    return NextResponse.json({
      success: true,
      data: result,
    });
  } catch (error) {
    return handleError(error, 'API_EXAMPLE_GET');
  }
}
```

V. Best Practices Cho Môi Trường Nội Bộ

1. Không Phụ Thuộc Vào Dịch Vụ Bên Ngoài

✅ **Đúng:**
- Lưu file trong hệ thống nội bộ (`/mnt/uploads/`)
- Sử dụng database nội bộ (PostgreSQL)
- Logging ra file hệ thống (`/var/log/tapchi/`)

❌ **Sai:**
- Upload lên AWS S3/Cloudflare (không có internet)
- Sử dụng external logging service (DataDog, Sentry)
- CDN cho static assets

2. Xử Lý Timeout & Retry

```
// Prisma client configuration
import { PrismaClient } from '@prisma/client';

export const prisma = new PrismaClient({
  datasources: {
    db: {
      url: process.env.DATABASE_URL,
    },
  },
  // Cấu hình cho môi trường nội bộ
  log: ['error', 'warn'],
  // Tăng timeout cho mạng chậm
  // Note: Configure this in DATABASE_URL connection string:
  // postgresql://user:pass@host:5432/db?connect_timeout=30&pool_timeout=30
});

// Auto-reconnect on disconnect
prisma.$on('beforeExit', async () => {
  await prisma.$disconnect();
});
```

3. Quản Lý Session & Token

```
# .env configuration
NEXTAUTH_SECRET=<your-secret-key>
NEXTAUTH_URL=http://your-internal-server:3000

# Tăng thời gian session cho mạng nội bộ
SESSION_MAX_AGE=86400  # 24 hours
JWT_MAX_AGE=86400      # 24 hours
```

4. Backup & Recovery

Backup Database (Hàng Ngày):

```
# Thêm vào crontab
0 3 * * * cd /path/to/app/nextjs_space && ./scripts/backup-db.sh
```

Restore Database:

```
./scripts/restore-db.sh /path/to/backup.sql.gz
```

VI. Checklist Triển Khai

Trước Khi Triển Khai

- [ ] Cấu hình tất cả biến môi trường trong `.env`
- [ ] Chạy `yarn tsx scripts/diagnostics/check-system.ts`
- [ ] Kiểm tra kết nối database: `yarn prisma db pull`
- [ ] Build ứng dụng: `yarn build`

- [ ] Test health check: `curl http://localhost:3000/api/health`

Sau Khi Triển Khai

- [ ] Cài đặt cron job health check
- [ ] Cài đặt cron job backup database
- [ ] Cài đặt cron job diagnostic scripts
- [ ] Cấu hình log rotation cho `/var/log/tapchi/`
- [ ] Kiểm tra quyền truy cập thư mục upload
- [ ] Test tất cả chức năng quan trọng

Giám Sát Hàng Ngày

- [ ] Kiểm tra log: `tail -f /var/log/tapchi-health.log`
- [ ] Xem diagnostic reports: `ls -lh logs/diagnostics/`
- [ ] Kiểm tra dung lượng đĩa: `df -h`
- [ ] Kiểm tra backup: `ls -lh /path/to/backups/`

VII. Troubleshooting

Vấn Đề: Database Connection Timeout

Nguyên nhân: Mạng nội bộ chậm hoặc database idle timeout

Giải pháp:

1. Tăng timeout trong DATABASE_URL:

```
DATABASE_URL="postgresql://user:pass@host:5432/db?con-
nect_timeout=30&pool_timeout=30&statement_timeout=30000"
```

1. Bật connection pooling:

```
DATABASE_URL="...?connection_limit=10&pool_timeout=30"
```

Vấn Đề: 500 Internal Server Error

Nguyên nhân: Lỗi chưa được xử lý trong API route

Cách kiểm tra:

1. Xem log chi tiết:

```
tail -f logs/app-$(date +%Y-%m-%d).log | grep ERROR
```

1. Chạy diagnostic:

```
yarn tsx scripts/diagnostics/check-system.ts
```

1. Kiểm tra health:

```
curl http://localhost:3000/api/health | jq
```

Vấn Đề: File Upload Failed

Nguyên nhân: Quyền truy cập thư mục

Giải pháp:

```
# Tạo thư mục upload
sudo mkdir -p /mnt/uploads
sudo chown -R appuser:appuser /mnt/uploads
sudo chmod 755 /mnt/uploads

# Cấu hình trong .env
UPLOAD_STORAGE_PATH=/mnt/uploads
```

VIII. Liên Hệ Hỗ Trợ

Nếu gặp vấn đề không giải quyết được, vui lòng:

1. Thu thập thông tin:
   - Chạy diagnostic script
   - Export logs gần đây
   - Chụp màn hình lỗi

2. Cung cấp thông tin hệ thống:
   - Phiên bản ứng dụng
   - Phiên bản Node.js
   - Hệ điều hành
   - Cấu hình database

Phụ Lục: Cấu Trúc Thư Mục

```
nextjs_space/
├── lib/
│   ├── error-handler.ts    # Tầng xử lý lỗi
│   ├── logger.ts           # Hệ thống logging
│   ├── api-guards.ts       # Auth & authorization
│   └── validators.ts       # Zod schemas
├── app/api/
│   ├── health/route.ts     # Health check endpoint
│   └── submissions/route.ts # API với error handling
├── scripts/
│   ├── diagnostics/
│   │   ├── check-system.ts
│   │   └── check-submissions.ts
│   ├── backup-db.sh
│   └── cron-health-check.sh
└── logs/
    ├── app-YYYY-MM-DD.log
    └── diagnostics/
        └── diagnostic-[timestamp].json
```

Tài liệu được cập nhật: 2025-01-15
Phiên bản hệ thống: 2.0
Tác giả: DeepAgent - Abacus.AI