Tapayz 암호화폐 결제솔루션 개발 가이드

📋 프로젝트 개요

시스템 아키텍처

```
[고객] → [Tapayz Frontend] → [Tapayz Backend] → [NowPayments API] → [블록체인]
          ↓                  ↓                  ↓
      [결제 UI/UX]       [비즈니스 로직]       [결제 처리]
```

핵심 구성요소

- **Frontend**: React/Vue.js 기반 결제 UI
- **Backend**: Node.js/Python 기반 API 서버
- **Database**: MySQL/PostgreSQL 고객 및 거래 데이터
- **Payment Engine**: NowPayments API 연동
- **Admin Panel**: 관리자 대시보드

🔧 기술 스택

Backend

- **Framework**: Node.js (Express) 또는 Python (FastAPI)
- **Database**: PostgreSQL
- **Cache**: Redis
- **Queue**: Bull (Node.js) / Celery (Python)

Frontend

- **Framework**: React.js / Next.js
- **UI Library**: Tailwind CSS / Material-UI
- **State Management**: Redux Toolkit / Zustand

DevOps

- **Deployment**: Docker + Kubernetes
- **Monitoring**: Prometheus + Grafana
- **Logging**: ELK Stack

🗄️ 데이터베이스 설계

핵심 테이블 구조

sql

```sql
-- 고객 테이블
CREATE TABLE customers (
    id BIGSERIAL PRIMARY KEY,
    external_id VARCHAR(255) UNIQUE,
    email VARCHAR(255) NOT NULL,
    phone VARCHAR(50),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    country VARCHAR(3),
    status VARCHAR(20) DEFAULT 'active',
    risk_level VARCHAR(20) DEFAULT 'medium',
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- 인보이스 테이블
CREATE TABLE invoices (
    id BIGSERIAL PRIMARY KEY,
    invoice_id VARCHAR(255) UNIQUE NOT NULL,
    customer_id BIGINT REFERENCES customers(id),
    title VARCHAR(255),
    amount DECIMAL(18,8) NOT NULL,
    currency VARCHAR(10) NOT NULL,
    crypto_amount DECIMAL(18,8),
    crypto_currency VARCHAR(10),
    price_at_request DECIMAL(18,8),
    status VARCHAR(20) DEFAULT 'new',
    expired_at TIMESTAMP,
    nowpayments_payment_id BIGINT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- 트랜잭션 테이블
CREATE TABLE transactions (
    id BIGSERIAL PRIMARY KEY,
    invoice_id BIGINT REFERENCES invoices(id),
    customer_id BIGINT REFERENCES customers(id),
    transaction_hash VARCHAR(255),
    from_address VARCHAR(255),
    to_address VARCHAR(255),
    amount DECIMAL(18,8) NOT NULL,
    currency VARCHAR(10) NOT NULL,
    network VARCHAR(50),
    status VARCHAR(20) DEFAULT 'pending',
    confirmations INT DEFAULT 0,
```

```
    block_number BIGINT,
    nowpayments_payment_id BIGINT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- 콜백 로그 테이블
CREATE TABLE callback_logs (
    id BIGSERIAL PRIMARY KEY,
    event_type VARCHAR(50) NOT NULL,
    payment_id BIGINT,
    invoice_id BIGINT,
    payload TEXT,
    status VARCHAR(20) DEFAULT 'received',
    processed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW()
);
```

🔐 인증 및 보안

API Key 관리

javascript

```
// config/auth.js
const API_KEYS = {
  NOWPAYMENTS_API_KEY: process.env.NOWPAYMENTS_API_KEY,
  NOWPAYMENTS_IPN_SECRET: process.env.NOWPAYMENTS_IPN_SECRET
};

// middleware/auth.js
const authenticateAPIKey = (req, res, next) => {
  const apiKey = req.headers['authorization']?.replace('Bearer ', '');

  if (!apiKey || !isValidAPIKey(apiKey)) {
    return res.status(401).json({ error: 'Invalid API key' });
  }

  req.partner = getPartnerByAPIKey(apiKey);
  next();
};
```

JWT 토큰 관리

javascript

```javascript
// utils/jwt.js
const jwt = require('jsonwebtoken');

const generateTokens = (user) => {
  const accessToken = jwt.sign(
    { userId: user.id, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: '1h' }
  );

  const refreshToken = jwt.sign(
    { userId: user.id },
    process.env.JWT_REFRESH_SECRET,
    { expiresIn: '30d' }
  );

  return { accessToken, refreshToken };
};
```

💳 NowPayments API 연동

기본 설정

javascript

```
// services/nowpayments.js
class NowPaymentsService {
  constructor() {
    this.apiKey = process.env.NOWPAYMENTS_API_KEY;
    this.baseURL = 'https://api.nowpayments.io/v1';
  }

  async makeRequest(endpoint, method = 'GET', data = null) {
    const config = {
      method,
      headers: {
        'x-api-key': this.apiKey,
        'Content-Type': 'application/json'
      }
    };

    if (data) {
      config.body = JSON.stringify(data);
    }

    const response = await fetch(`${this.baseURL}${endpoint}`, config);
    return await response.json();
  }

  // API 상태 확인
  async getStatus() {
    return await this.makeRequest('/status');
  }

  // 지원 암호화폐 목록
  async getCurrencies() {
    return await this.makeRequest('/currencies');
  }

  // 최소 결제 금액 조회
  async getMinAmount(currencyFrom, currencyTo) {
    return await this.makeRequest(
      `/min-amount?currency_from=${currencyFrom}&currency_to=${currencyTo}`
    );
  }

  // 예상 금액 계산
  async getEstimate(amount, currencyFrom, currencyTo) {
    return await this.makeRequest(
      `/estimate?amount=${amount}&currency_from=${currencyFrom}&currency_to=${currencyTo}`
    );
```

```
  }

  // 결제 생성
  async createPayment(paymentData) {
    return await this.makeRequest('/payment', 'POST', paymentData);
  }

  // 결제 상태 조회
  async getPayment(paymentId) {
    return await this.makeRequest(`/payment/${paymentId}`);
  }
}
```

결제 생성 API

```
javascript
```

```
// controllers/paymentController.js
const createPayment = async (req, res) => {
  try {
    const { customerId, amount, currency, cryptoCurrency } = req.body;

    // 1. 고객 존재 확인
    const customer = await Customer.findById(customerId);
    if (!customer) {
      return res.status(404).json({ error: 'Customer not found' });
    }

    // 2. 최소 금액 확인
    const minAmount = await nowPayments.getMinAmount(currency, cryptoCurrency);
    if (amount < parseFloat(minAmount.min_amount)) {
      return res.status(400).json({ error: 'Amount below minimum' });
    }

    // 3. 예상 금액 계산
    const estimate = await nowPayments.getEstimate(amount, currency, cryptoCurrency);

    // 4. NowPayments 결제 생성
    const paymentData = {
      price_amount: amount,
      price_currency: currency,
      pay_currency: cryptoCurrency,
      order_id: generateOrderId(),
      order_description: `Payment for customer ${customerId}`,
      ipn_callback_url: `${process.env.BASE_URL}/callbacks/nowpayments`
    };

    const nowPayment = await nowPayments.createPayment(paymentData);

    // 5. 인보이스 저장
    const invoice = await Invoice.create({
      invoice_id: generateInvoiceId(),
      customer_id: customerId,
      amount: amount,
      currency: currency,
      crypto_amount: estimate.estimated_amount,
      crypto_currency: cryptoCurrency,
      status: 'new',
      nowpayments_payment_id: nowPayment.payment_id,
      expired_at: new Date(Date.now() + 30 * 60000) // 30분 후 만료
    });

    res.status(201).json({
```

```
      success: true,
      data: {
        invoice_id: invoice.invoice_id,
        payment_address: nowPayment.pay_address,
        crypto_amount: estimate.estimated_amount,
        crypto_currency: cryptoCurrency,
        qr_code: generateQRCode(nowPayment.pay_address, estimate.estimated_amount),
        expires_at: invoice.expired_at
      }
    });

  } catch (error) {
    console.error('Payment creation error:', error);
    res.status(500).json({ error: 'Payment creation failed' });
  }
};
```

🔔 콜백 처리

NowPayments IPN 콜백

javascript

```
// controllers/callbackController.js
const crypto = require('crypto');

const handleNowPaymentsCallback = async (req, res) => {
  // 즉시 200 응답
  res.status(200).send('OK');

  try {
    const payload = req.body;
    const signature = req.headers['x-nowpayments-sig'];

    // 1. 서명 검증
    if (!verifySignature(payload, signature)) {
      console.error('Invalid signature');
      return;
    }

    // 2. 중복 이벤트 체크
    const existingLog = await CallbackLog.findOne({
      payment_id: payload.payment_id,
      event_type: 'payment_status_update'
    });

    if (existingLog && existingLog.status === 'processed') {
      console.log('Event already processed');
      return;
    }

    // 3. 콜백 로그 저장
    const callbackLog = await CallbackLog.create({
      event_type: 'payment_status_update',
      payment_id: payload.payment_id,
      payload: JSON.stringify(payload),
      status: 'received'
    });

    // 4. 백그라운드에서 처리
    await processPaymentCallback(payload, callbackLog.id);

  } catch (error) {
    console.error('Callback processing error:', error);
  }
};

const verifySignature = (payload, signature) => {
  const sortedParams = {};
```

```
  Object.keys(payload).sort().forEach(key => {
    sortedParams[key] = payload[key];
  });

  const sortedString = JSON.stringify(sortedParams);
  const expectedSignature = crypto
    .createHmac('sha512', process.env.NOWPAYMENTS_IPN_SECRET)
    .update(sortedString)
    .digest('hex');

  return crypto.timingSafeEqual(
    Buffer.from(signature, 'hex'),
    Buffer.from(expectedSignature, 'hex')
  );
};

const processPaymentCallback = async (payload, logId) => {
  try {
    // 인보이스 및 트랜잭션 업데이트
    const invoice = await Invoice.findOne({
      nowpayments_payment_id: payload.payment_id
    });

    if (!invoice) {
      console.error('Invoice not found for payment:', payload.payment_id);
      return;
    }

    // 상태별 처리
    switch (payload.payment_status) {
      case 'waiting':
        await updateInvoiceStatus(invoice.id, 'pending');
        break;
      case 'confirming':
        await updateInvoiceStatus(invoice.id, 'confirming');
        await createOrUpdateTransaction(invoice, payload);
        break;
      case 'confirmed':
        await updateInvoiceStatus(invoice.id, 'confirmed');
        break;
      case 'finished':
        await updateInvoiceStatus(invoice.id, 'completed');
        await sendCompletionNotification(invoice);
        break;
      case 'failed':
        await updateInvoiceStatus(invoice.id, 'failed');
        break;
```

```
      case 'expired':
        await updateInvoiceStatus(invoice.id, 'expired');
        break;
    }

    // 콜백 로그 업데이트
    await CallbackLog.update(logId, { status: 'processed', processed_at: new Date() });

  } catch (error) {
    console.error('Payment callback processing error:', error);
    await CallbackLog.update(logId, { status: 'failed' });
  }
};
```

👥 고객 관리 API

고객 생성

javascript

```
// controllers/customerController.js
const createCustomer = async (req, res) => {
  try {
    const { external_id, email, phone, first_name, last_name, country } = req.body;

    // 중복 확인
    const existingCustomer = await Customer.findOne({
      $or: [{ email }, { external_id }]
    });

    if (existingCustomer) {
      return res.status(400).json({ error: 'Customer already exists' });
    }

    // 고객 생성
    const customer = await Customer.create({
      external_id,
      email,
      phone,
      first_name,
      last_name,
      country,
      status: 'active',
      risk_level: 'medium'
    });

    res.status(201).json({
      success: true,
      data: {
        customer_id: customer.id,
        external_id: customer.external_id,
        email: customer.email,
        status: customer.status,
        created_at: customer.created_at
      }
    });

  } catch (error) {
    console.error('Customer creation error:', error);
    res.status(500).json({ error: 'Customer creation failed' });
  }
};

const getCustomer = async (req, res) => {
  try {
    const { customer_id } = req.params;
```

```
    const customer = await Customer.findById(customer_id);
    if (!customer) {
      return res.status(404).json({ error: 'Customer not found' });
    }

    // 고객 거래 통계
    const stats = await getCustomerStats(customer_id);

    res.json({
      success: true,
      data: {
        ...customer.toJSON(),
        stats
      }
    });

  } catch (error) {
    console.error('Customer fetch error:', error);
    res.status(500).json({ error: 'Customer fetch failed' });
  }
};
```

💰 환율 및 가격 관리

실시간 환율 조회

```
javascript
```

```
// services/priceService.js
class PriceService {
  constructor() {
    this.nowPayments = new NowPaymentsService();
    this.cache = new Map();
    this.cacheTimeout = 30000; // 30초 캐시
  }

  async getExchangeRate(fromCurrency, toCurrency) {
    const cacheKey = `${fromCurrency}_${toCurrency}`;
    const cached = this.cache.get(cacheKey);

    if (cached && Date.now() - cached.timestamp < this.cacheTimeout) {
      return cached.data;
    }

    try {
      const estimate = await this.nowPayments.getEstimate(1, fromCurrency, toCurrency);
      const rate = {
        from_currency: fromCurrency,
        to_currency: toCurrency,
        rate: estimate.estimated_amount,
        inverse_rate: 1 / estimate.estimated_amount,
        last_updated: new Date().toISOString()
      };

      this.cache.set(cacheKey, {
        data: rate,
        timestamp: Date.now()
      });

      return rate;
    } catch (error) {
      console.error('Exchange rate fetch error:', error);
      throw error;
    }
  }

  async convertAmount(amount, fromCurrency, toCurrency, includeFees = false) {
    const rate = await this.getExchangeRate(fromCurrency, toCurrency);
    const convertedAmount = amount * rate.rate;

    let result = {
      original_amount: amount.toString(),
      from_currency: fromCurrency,
      converted_amount: convertedAmount.toString(),
```

```
      to_currency: toCurrency,
      exchange_rate: rate.rate.toString(),
      timestamp: new Date().toISOString()
    };

    if (includeFees) {
      const feePercentage = 0.5; // 0.5% 기본 수수료
      const feeAmount = amount * (feePercentage / 100);
      const netAmount = amount - feeAmount;

      result = {
        ...result,
        fee_percentage: feePercentage.toString(),
        fee_amount: feeAmount.toString(),
        net_amount: netAmount.toString()
      };
    }

    return result;
  }
}
```

🎨 프론트엔드 구현

React 결제 컴포넌트

jsx

```jsx
// components/PaymentModal.jsx
import React, { useState, useEffect } from 'react';
import QRCode from 'qrcode.react';

const PaymentModal = ({ isOpen, onClose, paymentData }) => {
  const [timeLeft, setTimeLeft] = useState(null);
  const [paymentStatus, setPaymentStatus] = useState('waiting');

  useEffect(() => {
    if (!paymentData?.expires_at) return;

    const timer = setInterval(() => {
      const now = new Date().getTime();
      const expiry = new Date(paymentData.expires_at).getTime();
      const difference = expiry - now;

      if (difference > 0) {
        const minutes = Math.floor((difference % (1000 * 60 * 60)) / (1000 * 60));
        const seconds = Math.floor((difference % (1000 * 60)) / 1000);
        setTimeLeft(`${minutes}:${seconds.toString().padStart(2, '0')}`);
      } else {
        setTimeLeft('Expired');
        setPaymentStatus('expired');
      }
    }, 1000);

    return () => clearInterval(timer);
  }, [paymentData?.expires_at]);

  useEffect(() => {
    if (!paymentData?.invoice_id) return;

    // WebSocket 연결로 실시간 상태 업데이트
    const ws = new WebSocket(`${process.env.REACT_APP_WS_URL}/payment/${paymentData.invoice_id}`);

    ws.onmessage = (event) => {
      const data = JSON.parse(event.data);
      setPaymentStatus(data.status);

      if (data.status === 'completed') {
        setTimeout(() => {
          onClose();
          // 성공 페이지로 리다이렉트
        }, 2000);
      }
    };
```

```
  return () => ws.close();
}, [paymentData?.invoice_id]);

const copyToClipboard = (text) => {
  navigator.clipboard.writeText(text);
  // 토스트 알림 표시
};

if (!isOpen) return null;

return (
  <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">
    <div className="bg-white rounded-lg p-6 max-w-md w-full mx-4">
      <div className="flex justify-between items-center mb-4">
        <h2 className="text-xl font-bold">암호화폐 결제</h2>
        <button onClick={onClose} className="text-gray-500 hover:text-gray-700">
          ×
        </button>
      </div>

      {paymentStatus === 'waiting' && (
        <>
          <div className="text-center mb-4">
            <div className="text-2xl font-bold text-blue-600 mb-2">
              {paymentData.crypto_amount} {paymentData.crypto_currency}
            </div>
            <div className="text-gray-600">
              결제 시간: {timeLeft}
            </div>
          </div>

          <div className="flex justify-center mb-4">
            <QRCode
              value={`${paymentData.crypto_currency}:${paymentData.payment_address}?amount=${paymentData.crypto_
              size={200}
            />
          </div>

          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-700 mb-2">
              결제 주소
            </label>
            <div className="flex items-center">
              <input
                type="text"
                value={paymentData.payment_address}
```

```
            readOnly
            className="flex-1 p-2 border rounded-l-md bg-gray-50"
          />
          <button
            onClick={() => copyToClipboard(paymentData.payment_address)}
            className="px-3 py-2 bg-blue-500 text-white rounded-r-md hover:bg-blue-600"
          >
            복사
          </button>
        </div>
      </div>

      <div className="text-sm text-gray-600 mb-4">
        <p>1. 위의 QR 코드를 스캔하거나 주소를 복사하세요</p>
        <p>2. 정확한 금액을 전송하세요</p>
        <p>3. 결제 확인까지 잠시 기다려주세요</p>
      </div>
    </>
  )}

  {paymentStatus === 'confirming' && (
    <div className="text-center py-8">
      <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-500 mx-auto mb-4"></div>
      <p className="text-lg font-medium">결제 확인 중...</p>
      <p className="text-gray-600">블록체인에서 거래를 확인하고 있습니다</p>
    </div>
  )}

  {paymentStatus === 'completed' && (
    <div className="text-center py-8">
      <div className="text-green-500 text-6xl mb-4">✓</div>
      <p className="text-lg font-medium text-green-600">결제 완료!</p>
      <p className="text-gray-600">결제가 성공적으로 처리되었습니다</p>
    </div>
  )}

  {paymentStatus === 'expired' && (
    <div className="text-center py-8">
      <div className="text-red-500 text-6xl mb-4">⏰</div>
      <p className="text-lg font-medium text-red-600">결제 시간 만료</p>
      <p className="text-gray-600">새로운 결제를 요청해주세요</p>
    </div>
  )}
 </div>
</div>
);
};
```

```
export default PaymentModal;
```

결제 API 클라이언트

javascript

```javascript
// services/api.js
class TapayzAPI {
  constructor() {
    this.baseURL = process.env.REACT_APP_API_URL;
    this.apiKey = process.env.REACT_APP_API_KEY;
  }

  async request(endpoint, options = {}) {
    const config = {
      headers: {
        'Authorization': `Bearer ${this.apiKey}`,
        'Content-Type': 'application/json',
        ...options.headers
      },
      ...options
    };

    const response = await fetch(`${this.baseURL}${endpoint}`, config);

    if (!response.ok) {
      throw new Error(`API Error: ${response.status}`);
    }

    return await response.json();
  }

  // 고객 생성
  async createCustomer(customerData) {
    return await this.request('/customers', {
      method: 'POST',
      body: JSON.stringify(customerData)
    });
  }

  // 결제 생성
  async createPayment(paymentData) {
    return await this.request('/payments', {
      method: 'POST',
      body: JSON.stringify(paymentData)
    });
  }

  // 환율 조회
  async getExchangeRate(from, to) {
    return await this.request(`/exchange-rates?from=${from}&to=${to}`);
  }
```

```
  // 지원 암호화폐 목록
  async getCurrencies() {
    return await this.request('/currencies');
  }

  // 결제 상태 조회
  async getPaymentStatus(invoiceId) {
    return await this.request(`/payments/${invoiceId}`);
  }
}

export default new TapayzAPI();
```

📊 관리자 대시보드

관리자 페이지 컴포넌트

jsx

```jsx
// components/AdminDashboard.jsx
import React, { useState, useEffect } from 'react';

const AdminDashboard = () => {
  const [stats, setStats] = useState({});
  const [recentPayments, setRecentPayments] = useState([]);

  useEffect(() => {
    fetchDashboardData();
  }, []);

  const fetchDashboardData = async () => {
    try {
      const [statsResponse, paymentsResponse] = await Promise.all([
        TapayzAPI.request('/admin/stats'),
        TapayzAPI.request('/admin/payments?limit=10')
      ]);

      setStats(statsResponse.data);
      setRecentPayments(paymentsResponse.data);
    } catch (error) {
      console.error('Dashboard data fetch error:', error);
    }
  };

  return (
    <div className="p-6">
      <h1 className="text-2xl font-bold mb-6">Tapayz 관리자 대시보드</h1>

      {/* 통계 카드 */}
      <div className="grid grid-cols-1 md:grid-cols-4 gap-6 mb-8">
        <div className="bg-white p-6 rounded-lg shadow">
          <h3 className="text-sm font-medium text-gray-500">오늘 거래량</h3>
          <p className="text-2xl font-bold text-gray-900">{stats.todayVolume || 0}</p>
        </div>
        <div className="bg-white p-6 rounded-lg shadow">
          <h3 className="text-sm font-medium text-gray-500">성공률</h3>
          <p className="text-2xl font-bold text-green-600">{stats.successRate || 0}%</p>
        </div>
        <div className="bg-white p-6 rounded-lg shadow">
          <h3 className="text-sm font-medium text-gray-500">활성 고객</h3>
          <p className="text-2xl font-bold text-blue-600">{stats.activeCustomers || 0}</p>
        </div>
        <div className="bg-white p-6 rounded-lg shadow">
          <h3 className="text-sm font-medium text-gray-500">대기 중 결제</h3>
          <p className="text-2xl font-bold text-orange-600">{stats.pendingPayments || 0}</p>
```

```
      </div>
    </div>

    {/* 최근 거래 */}
    <div className="bg-white rounded-lg shadow">
      <div className="px-6 py-4 border-b">
        <h2 className="text-lg font-medium">최근 거래</h2>
      </div>
      <div className="overflow-x-auto">
        <table className="min-w-full divide-y divide-gray-200">
          <thead className="bg-gray-50">
            <tr>
              <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase">ID</th>
              <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase">고객</th>
              <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase">금액</th>
              <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase">상태</th>
              <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase">생성일</th>
            </tr>
          </thead>
          <tbody className="bg-white divide-y divide-gray-200">
            {recentPayments.map((payment) => (
              <tr key={payment.id}>
                <td className="px-6 py-4 whitespace-nowrap text-sm font-medium text-gray-900">
                  {payment.invoice_id}
                </td>
                <td className="px-6 py-4 whitespace-nowrap text-sm text-gray-500">
                  {payment.customer_email}
                </td>
                <td className="px-6 py-4 whitespace-nowrap text-sm text-gray-500">
                  {payment.amount} {payment.currency}
                </td>
                <td className="px-6 py-4 whitespace-nowrap">
                  <span className={`px-2 py-1 text-xs rounded-full ${getStatusColor(payment.status)}`}>
                    {payment.status}
                  </span>
                </td>
                <td className="px-6 py-4 whitespace-nowrap text-sm text-gray-500">
                  {new Date(payment.created_at).toLocaleDateString()}
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  </div>
);
```

```
};

const getStatusColor = (status) => {
  switch (status) {
    case 'completed': return 'bg-green-100 text-green-800';
    case 'pending': return 'bg-yellow-100 text-yellow-800';
    case 'failed': return 'bg-red-100 text-red-800';
    default: return 'bg-gray-100 text-gray-800';
  }
};
```

🚀 배포 및 운영

Docker 설정

dockerfile

```
# Dockerfile
FROM node:16-alpine

WORKDIR /app

COPY package*.json ./
RUN npm ci --only=production

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

환경 변수 설정

bash

```
# .env
NODE_ENV=production
PORT=3000

# Database
DATABASE_URL=postgresql://user:password@localhost:5432/tapayz

# NowPayments
NOWPAYMENTS_API_KEY=your_api_key_here
NOWPAYMENTS_IPN_SECRET=your_ipn_secret_here
NOWPAYMENTS_SANDBOX=false

# JWT
JWT_SECRET=your_jwt_secret_here
JWT_REFRESH_SECRET=your_refresh_secret_here

# Redis
REDIS_URL=redis://localhost:6379

# Callback URLs
BASE_URL=https://api.tapayz.com
FRONTEND_URL=https://tapayz.com
```

Kubernetes 배포

yaml

```yaml
# k8s/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tapayz-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: tapayz-api
  template:
    metadata:
      labels:
        app: tapayz-api
    spec:
      containers:
      - name: tapayz-api
        image: tapayz/api:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: tapayz-secrets
              key: database-url
        - name: NOWPAYMENTS_API_KEY
          valueFrom:
            secretKeyRef:
              name: tapayz-secrets
              key: nowpayments-api-key
```

📈 모니터링 및 로깅

로깅 설정

```javascript
```

```
// utils/logger.js
const winston = require('winston');

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.errors({ stack: true }),
    winston.format.json()
  ),
  transports: [
    new winston.transports.File({ filename: 'logs/error.log', level: 'error' }),
    new winston.transports.File({ filename: 'logs/combined.log' }),
  ]
});

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.simple()
  }));
}

module.exports = logger;
```

헬스체크 엔드포인트

javascript

```
// routes/health.js
const express = require('express');
const router = express.Router();

router.get('/health', async (req, res) => {
  try {
    // 데이터베이스 연결 확인
    await db.query('SELECT 1');

    // NowPayments API 상태 확인
    const nowPaymentsStatus = await nowPayments.getStatus();

    res.json({
      status: 'healthy',
      timestamp: new Date().toISOString(),
      services: {
        database: 'healthy',
        nowpayments: nowPaymentsStatus.message === 'OK' ? 'healthy' : 'unhealthy',
        redis: 'healthy'
      }
    });
  } catch (error) {
    res.status(503).json({
      status: 'unhealthy',
      error: error.message
    });
  }
});

module.exports = router;
```

🔒 보안 체크리스트

필수 보안 조치

- [ ] API 키 환경 변수 관리
- [ ] HTTPS 전용 통신
- [ ] Rate Limiting 구현
- [ ] SQL Injection 방지
- [ ] XSS 방지
- [ ] CSRF 토큰 사용
- [ ] 콜백 서명 검증
- [ ] IP 화이트리스트 설정

- [ ] 민감 정보 로깅 금지
- [ ] 정기적인 보안 감사

운영 체크리스트

- [ ] 백업 전략 수립
- [ ] 모니터링 알림 설정
- [ ] 장애 대응 절차 마련
- [ ] 성능 최적화
- [ ] 부하 테스트 수행
- [ ] 문서화 완료
- [ ] 팀 교육 실시

이 문서는 Tapayz 개발팀이 실제 프로덕션 환경에서 사용할 수 있도록 구성되었습니다. 추가 질문이나 특정 부분에 대한 상세한 설명이 필요하시면 언제든 문의해주세요.