Fleecat 백엔드 개발 표준 가이드

프로젝트명: Fleecat 멀티테넌트 쇼핑몰
버전: 1.0
작성일: 2025년 9월 30일
작성자: 백엔드 개발팀

개정 이력

버전	변경일	변경내용	작성자	승인자
1.0	2025.09.30	최초 작성	백엔드팀	-

목차

---

1. 프로젝트 약어 및 용어 정의

1.1 백엔드 약어

분야	영문명	약어	설명
API	Application Programming Interface	API	애플리케이션 인터페이스
ORM	Object-Relational Mapping	ORM	객체-관계 매핑
CRUD	Create, Read, Update, Delete	CRUD	기본 데이터 조작

분야	영문명	약어	설명
JWT	JSON Web Token	JWT	인증 토큰
RLS	Row Level Security	RLS	행 수준 보안
BaaS	Backend as a Service	BaaS	백엔드 서비스

1.2 도메인 용어

한글명	영문명	약어	설명
판매사	Tenant	-	공방, 스토어 운영 주체
판매자	Seller	-	판매사 소속 구성원
구매자	Buyer	-	상품 구매 회원
멀티테넌트	Multi-tenant	MT	여러 판매사 지원 구조

---

2. 프로젝트 개요

2.1 프로젝트 정보

항목	내용
프로젝트명	Fleecat 멀티테넌트 쇼핑몰
프로젝트 유형	E-commerce Platform (B2C, C2C)
핵심 기능	상품 관리, 주문/결제, 회원 관리, 판매자 관리
데이터베이스	PostgreSQL (via Supabase)
총 테이블 수	17개

2.2 주요 비즈니스 프로세스

1. 판매자 등록 프로세스
   회원가입 → 판매자 신청 → 관리자 승인 → 상품 등록

2. 상품 판매 프로세스
   상품 등록 → 관리자 승인 → 상품 노출 → 주문 → 배송

3. 주문 처리 프로세스
   장바구니 → 주문 생성 → 결제 → 배송 → 구매 확정

---

3. 기술 스택

3.1 핵심 기술

구분	기술	버전	역할
런타임	Node.js	20.x LTS	JavaScript 실행 환경
프레임워크	Express.js	4.x	웹 애플리케이션 프레임워크
ORM	Prisma	5.x	데이터베이스 ORM
BaaS	Supabase	Latest	데이터베이스, 인증, 스토리지

3.2 주요 라이브러리

라이브러리	용도
`@supabase/supabase-js`	Supabase 클라이언트
`@prisma/client`	Prisma 클라이언트
`jsonwebtoken`	JWT 인증
`bcrypt`	비밀번호 암호화
`express-validator`	입력 검증
`cors`	CORS 설정
`helmet`	보안 헤더
`morgan`	HTTP 로깅
`dotenv`	환경변수 관리

---

4. 프로젝트 구조

```
fleecat-backend/
├── prisma/                 # Prisma 설정
│   ├── schema.prisma       # 데이터베이스 스키마
│   ├── migrations/         # 마이그레이션 파일
│   └── seed.js             # 초기 데이터
│
├── src/                    # 소스 코드
│   ├── config/             # 설정 파일
│   │   ├── database.js     # DB 연결 설정
│   │   ├── supabase.js     # Supabase 설정
│   │   └── constants.js    # 상수 정의
│   │
│   ├── middlewares/        # 미들웨어
│   │   ├── auth.js         # 인증 미들웨어
│   │   ├── validation.js   # 검증 미들웨어
```

```
│   │   ├── errorHandler.js  # 에러 핸들러
│   │   └── rateLimiter.js   # Rate Limiting
│   │
│   ├── routes/              # 라우터
│   │   ├── index.js         # 라우터 통합
│   │   ├── auth.routes.js   # 인증 라우트
│   │   ├── product.routes.js # 상품 라우트
│   │   ├── order.routes.js   # 주문 라우트
│   │   ├── cart.routes.js    # 장바구니 라우트
│   │   ├── member.routes.js  # 회원 라우트
│   │   └── admin.routes.js   # 관리자 라우트
│   │
│   ├── controllers/         # 컨트롤러
│   │   ├── auth.controller.js
│   │   ├── product.controller.js
│   │   ├── order.controller.js
│   │   └── ...
│   │
│   ├── services/            # 비즈니스 로직
│   │   ├── auth.service.js
│   │   ├── product.service.js
│   │   ├── order.service.js
│   │   └── ...
│   │
│   ├── repositories/        # 데이터 접근 계층
│   │   ├── product.repository.js
│   │   ├── order.repository.js
│   │   └── ...
│   │
│   ├── utils/               # 유틸리티
│   │   ├── logger.js        # 로깅
│   │   ├── validator.js     # 검증 함수
│   │   ├── helpers.js       # 헬퍼 함수
│   │   └── response.js      # 응답 포맷터
│   │
│   ├── types/               # TypeScript 타입 정의
│   │   └── index.d.ts
│   │
│   └── app.js               # Express 앱 설정
│   └── server.js            # 서버 실행
│
├── tests/                   # 테스트 코드
│   ├── unit/                # 단위 테스트
│   ├── integration/         # 통합 테스트
│   └── e2e/                 # E2E 테스트
│
├── docs/                    # 문서
```

```
│   ├── api/                # API 문서
│   ├── architecture/       # 아키텍처 문서
│   └── database/           # DB 문서
│
├── scripts/                # 스크립트
│   ├── setup.sh            # 초기 설정
│   └── deploy.sh           # 배포 스크립트
│
├── .env.example            # 환경변수 예시
├── .env.development        # 개발 환경변수
├── .env.production         # 운영 환경변수
├── .gitignore              # Git 무시 파일
├── .eslintrc.js            # ESLint 설정
├── .prettierrc             # Prettier 설정
├── package.json            # 패키지 정보
├── package-lock.json       # 패키지 잠금
└── README.md               # 프로젝트 설명
```

---

5. 명명 규칙

5.1 파일명 규칙

유형	규칙	예시
라우터	`[도메인].routes.js`	`product.routes.js`
컨트롤러	`[도메인].controller.js`	`product.controller.js`
서비스	`[도메인].service.js`	`product.service.js`
미들웨어	`[기능].js`	`auth.js`, `validation.js`
유틸리티	`[기능].js`	`logger.js`, `helpers.js`
테스트	`[대상].test.js`	`product.service.test.js`

5.2 변수명 규칙

javascript

```
// 상수: UPPER_SNAKE_CASE
const MAX_LOGIN_ATTEMPTS = 5;
const JWT_SECRET = process.env.JWT_SECRET;

// 변수/함수: camelCase
const userId = 123;
const productList = [];
function getUserById(id) { }

// 클래스: PascalCase
class ProductService { }
class OrderController { }

// Private 변수: _camelCase
class UserService {
  _connection = null;
  _cache = {};
}

// 데이터베이스 관련
const prisma = new PrismaClient();
const supabase = createClient(url, key);
```

5.3 함수명 규칙

유형	규칙	예시
조회	`get`, `find`, `fetch`	`getUserById`, `findProducts`
생성	`create`, `add`	`createOrder`, `addToCart`
수정	`update`, `modify`	`updateProduct`, `modifyStatus`
삭제	`delete`, `remove`	`deleteProduct`, `removeFromCart`
검증	`validate`, `check`	`validateEmail`, `checkStock`
처리	`process`, `handle`	`processPayment`, `handleOrder`

5.4 API 엔드포인트 규칙

기본 형식: /api/v1/[리소스]/[하위리소스]

```
GET    /api/v1/products          # 상품 목록 조회
GET    /api/v1/products/:id      # 상품 상세 조회
POST   /api/v1/products          # 상품 생성
PUT    /api/v1/products/:id      # 상품 수정
DELETE /api/v1/products/:id      # 상품 삭제
```

```
# 하위 리소스
GET    /api/v1/products/:id/images   # 상품 이미지 목록
POST   /api/v1/products/:id/images   # 상품 이미지 추가

# 특수 액션
POST   /api/v1/orders/:id/cancel     # 주문 취소
POST   /api/v1/orders/:id/confirm    # 주문 확정
```

---

6. 코딩 표준

6.1 JavaScript 코딩 표준

Import 순서

javascript

```javascript
// 1. Node.js 내장 모듈
const fs = require('fs');
const path = require('path');

// 2. 외부 라이브러리
const express = require('express');
const { PrismaClient } = require('@prisma/client');

// 3. 내부 모듈
const { authenticate } = require('../middlewares/auth');
const ProductService = require('../services/product.service');
```

함수 작성 규칙

javascript

```javascript
/**
 * 상품 목록 조회
 * @param {Object} filters - 필터 조건
 * @param {number} filters.categoryId - 카테고리 ID
 * @param {number} filters.minPrice - 최소 가격
 * @param {number} filters.maxPrice - 최대 가격
 * @param {Object} pagination - 페이지네이션
 * @param {number} pagination.page - 페이지 번호
 * @param {number} pagination.limit - 페이지당 개수
 * @returns {Promise<Object>} 상품 목록과 총 개수
 */
async function getProducts(filters = {}, pagination = {}) {
  const { categoryId, minPrice, maxPrice } = filters;
  const { page = 1, limit = 20 } = pagination;

  // 구현...
}
```

에러 처리

javascript

```javascript
// 커스텀 에러 클래스
class ValidationError extends Error {
  constructor(message) {
    super(message);
    this.name = 'ValidationError';
    this.statusCode = 400;
  }
}

class NotFoundError extends Error {
  constructor(message) {
    super(message);
    this.name = 'NotFoundError';
    this.statusCode = 404;
  }
}

// 사용 예시
if (!product) {
  throw new NotFoundError('상품을 찾을 수 없습니다');
}
```

6.2 비동기 처리

javascript

```javascript
// ✅ 권장: async/await 사용
async function createOrder(orderData) {
  try {
    const order = await prisma.order.create({
      data: orderData
    });
    return order;
  } catch (error) {
    throw new Error(`주문 생성 실패: ${error.message}`);
  }
}

// ❌ 비권장: 콜백 사용
function createOrder(orderData, callback) {
  // 콜백 헬 발생 가능
}
```

6.3 환경변수 관리

javascript

```
// config/constants.js
require('dotenv').config();

module.exports = {
  // 서버 설정
  PORT: process.env.PORT || 3000,
  NODE_ENV: process.env.NODE_ENV || 'development',

  // 데이터베이스
  DATABASE_URL: process.env.DATABASE_URL,

  // Supabase
  SUPABASE_URL: process.env.SUPABASE_URL,
  SUPABASE_KEY: process.env.SUPABASE_KEY,

  // JWT
  JWT_SECRET: process.env.JWT_SECRET,
  JWT_EXPIRES_IN: process.env.JWT_EXPIRES_IN || '7d',

  // 기타
  MAX_FILE_SIZE: 5 * 1024 * 1024, // 5MB
  ALLOWED_ORIGINS: process.env.ALLOWED_ORIGINS?.split(',') || []
};
```

---

7. 데이터베이스 표준

7.1 Prisma Schema 규칙

```
prisma
```

```
// schema.prisma

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

// 모델 정의 규칙
model Product {
  // 1. ID 필드 (항상 첫 번째)
  product_id          BigInt    @id @default(autoincrement())

  // 2. 필수 필드
  product_name        String    @db.VarChar(100)
  product_price       Decimal   @db.Decimal(10, 2)
  product_status      String    @default("inactive") @db.VarChar(20)

  // 3. Foreign Key
  tenant_member_id    BigInt
  category_id         BigInt    @default(0)

  // 4. 타임스탬프
  product_created_at  DateTime  @default(now())
  product_updated_at  DateTime  @updatedAt

  // 5. 관계 정의
  tenant_member       TenantMember @relation(fields: [tenant_member_id], references: [tenant_member_id])
  category            Category     @relation(fields: [category_id], references: [category_id])
  product_images      ProductImage[]

  // 6. 테이블명 매핑
  @@map("product")

  // 7. 인덱스
  @@index([tenant_member_id])
  @@index([category_id])
  @@index([product_status])
}
```

7.2 쿼리 작성 규칙

```
javascript
```

```javascript
// ✅ 권장: 필요한 필드만 선택
const products = await prisma.product.findMany({
  select: {
    product_id: true,
    product_name: true,
    product_price: true,
    tenant_member: {
      select: {
        tenant: {
          select: {
            tenant_name: true
          }
        }
      }
    }
  }
});

// ❌ 비권장: 모든 필드 조회
const products = await prisma.product.findMany();
```

7.3 트랜잭션 처리

javascript

```
// 주문 생성 트랜잭션
async function createOrder(orderData, items) {
  return await prisma.$transaction(async (tx) => {
    // 1. 주문 생성
    const order = await tx.order.create({
      data: orderData
    });

    // 2. 주문 상세 생성
    await tx.orderItem.createMany({
      data: items.map(item => ({
        order_id: order.order_id,
        ...item
      }))
    });

    // 3. 재고 감소
    for (const item of items) {
      await tx.product.update({
        where: { product_id: item.product_id },
        data: {
          product_quantity: {
            decrement: item.quantity
          }
        }
      });
    }

    return order;
  });
}
```

---

8. API 개발 표준

8.1 계층 구조

```
Client Request
    ↓
Route (라우팅)
    ↓
Middleware (검증, 인증)
    ↓
Controller (요청/응답 처리)
    ↓
```

```
Service (비즈니스 로직)
    ↓
Repository (데이터 접근)
    ↓
Database
```

8.2 라우터 작성

javascript

```javascript
// routes/product.routes.js
const express = require('express');
const router = express.Router();
const ProductController = require('../controllers/product.controller');
const { authenticate } = require('../middlewares/auth');
const { validateProduct } = require('../middlewares/validation');

// 공개 API
router.get('/', ProductController.getProducts);
router.get('/:id', ProductController.getProductById);

// 인증 필요 API
router.post('/',
  authenticate,
  validateProduct,
  ProductController.createProduct
);

router.put('/:id',
  authenticate,
  validateProduct,
  ProductController.updateProduct
);

router.delete('/:id',
  authenticate,
  ProductController.deleteProduct
);

module.exports = router;
```

8.3 컨트롤러 작성

javascript

```
// controllers/product.controller.js
const ProductService = require('../services/product.service');
const { successResponse, errorResponse } = require('../utils/response');

class ProductController {
  /**
   * 상품 목록 조회
   */
  async getProducts(req, res, next) {
    try {
      const { categoryId, minPrice, maxPrice, page, limit } = req.query;

      const result = await ProductService.getProducts({
        categoryId: categoryId ? parseInt(categoryId) : undefined,
        minPrice: minPrice ? parseFloat(minPrice) : undefined,
        maxPrice: maxPrice ? parseFloat(maxPrice) : undefined
      }, {
        page: page ? parseInt(page) : 1,
        limit: limit ? parseInt(limit) : 20
      });

      return successResponse(res, result, '상품 목록 조회 성공');
    } catch (error) {
      next(error);
    }
  }

  /**
   * 상품 생성
   */
  async createProduct(req, res, next) {
    try {
      const userId = req.user.id; // 인증 미들웨어에서 설정
      const productData = req.body;

      const product = await ProductService.createProduct(userId, productData);

      return successResponse(res, product, '상품 등록 성공', 201);
    } catch (error) {
      next(error);
    }
  }
}

module.exports = new ProductController();
```

8.4 서비스 작성

```javascript
```

```
// services/product.service.js
const ProductRepository = require('../repositories/product.repository');
const { ValidationError, NotFoundError } = require('../utils/errors');

class ProductService {
  /**
   * 상품 목록 조회
   */
  async getProducts(filters, pagination) {
    const products = await ProductRepository.findMany(filters, pagination);
    const total = await ProductRepository.count(filters);

    return {
      products,
      pagination: {
        page: pagination.page,
        limit: pagination.limit,
        total,
        totalPages: Math.ceil(total / pagination.limit)
      }
    };
  }

  /**
   * 상품 생성
   */
  async createProduct(userId, productData) {
    // 비즈니스 로직 검증
    if (productData.product_price <= 0) {
      throw new ValidationError('가격은 0보다 커야 합니다');
    }

    // 판매자 권한 확인
    const seller = await this._checkSellerPermission(userId);

    // 상품 생성
    const product = await ProductRepository.create({
      ...productData,
      tenant_member_id: seller.tenant_member_id
    });

    return product;
  }

  /**
   * 판매자 권한 확인 (private)
```

```
   */
  async _checkSellerPermission(userId) {
    // 구현...
  }
}

module.exports = new ProductService();
```

8.5 Repository 작성

javascript

```
// repositories/product.repository.js
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

class ProductRepository {
  /**
   * 상품 목록 조회
   */
  async findMany(filters = {}, pagination = {}) {
    const { categoryId, minPrice, maxPrice } = filters;
    const { page = 1, limit = 20 } = pagination;

    const where = {};

    if (categoryId) where.category_id = parseInt(categoryId);
    if (minPrice) where.product_price = { gte: parseFloat(minPrice) };
    if (maxPrice) {
      where.product_price = {
        ...where.product_price,
        lte: parseFloat(maxPrice)
      };
    }

    return await prisma.product.findMany({
      where,
      include: {
        tenant_member: {
          select: {
            tenant: {
              select: {
                tenant_name: true
              }
            }
          }
        },
        product_images: {
          where: { product_image_sequence: 0 },
          take: 1
        }
      },
      skip: (page - 1) * limit,
      take: limit,
      orderBy: { product_created_at: 'desc' }
    });
  }
```

```
  /**
   * 상품 생성
   */
  async create(data) {
    return await prisma.product.create({
      data,
      include: {
        tenant_member: true,
        category: true
      }
    });
  }

  /**
   * 상품 개수 조회
   */
  async count(filters = {}) {
    const { categoryId, minPrice, maxPrice } = filters;
    const where = {};

    if (categoryId) where.category_id = parseInt(categoryId);
    if (minPrice || maxPrice) {
      where.product_price = {};
      if (minPrice) where.product_price.gte = parseFloat(minPrice);
      if (maxPrice) where.product_price.lte = parseFloat(maxPrice);
    }

    return await prisma.product.count({ where });
  }
}

module.exports = new ProductRepository();
```

8.6 응답 포맷

javascript

```javascript
// utils/response.js

/**
 * 성공 응답
 */
function successResponse(res, data, message = 'Success', statusCode = 200) {
  return res.status(statusCode).json({
    success: true,
    message,
    data
  });
}

/**
 * 에러 응답
 */
function errorResponse(res, message, statusCode = 500, errors = null) {
  const response = {
    success: false,
    message
  };

  if (errors) {
    response.errors = errors;
  }

  return res.status(statusCode).json(response);
}

module.exports = {
  successResponse,
  errorResponse
};
```

---

9. 인증 및 보안

9.1 JWT 인증

javascript

```
// middlewares/auth.js
const jwt = require('jsonwebtoken');
const { JWT_SECRET } = require('../config/constants');
const { errorResponse } = require('../utils/response');

/**
 * JWT 토큰 생성
 */
function generateToken(payload) {
  return jwt.sign(payload, JWT_SECRET, {
    expiresIn: '7d'
  });
}

/**
 * JWT 인증 미들웨어
 */
function authenticate(req, res, next) {
  try {
    const token = req.headers.authorization?.split(' ')[1];

    if (!token) {
      return errorResponse(res, '인증 토큰이 필요합니다', 401);
    }

    const decoded = jwt.verify(token, JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    return errorResponse(res, '유효하지 않은 토큰입니다', 401);
  }
}

/**
 * 권한 확인 미들웨어
 */
function authorize(...roles) {
  return (req, res, next) => {
    if (!req.user) {
      return errorResponse(res, '인증이 필요합니다', 401);
    }

    if (!roles.includes(req.user.role)) {
      return errorResponse(res, '권한이 없습니다', 403);
    }
```

```
    next();
  };
}

module.exports = {
  generateToken,
  authenticate,
  authorize
};
```

9.2 입력 검증

javascript

```
// middlewares/validation.js
const { body, validationResult } = require('express-validator');
const { errorResponse } = require('../utils/response');

/**
 * 검증 결과 확인
 */
function validateResult(req, res, next) {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return errorResponse(res, '입력값이 올바르지 않습니다', 400, errors.array());
  }
  next();
}

/**
 * 상품 검증
 */
const validateProduct = [
  body('product_name')
    .trim()
    .notEmpty().withMessage('상품명은 필수입니다')
    .isLength({ max: 100 }).withMessage('상품명은 100자 이하여야 합니다'),

  body('product_price')
    .isFloat({ min: 0 }).withMessage('가격은 0 이상이어야 합니다'),

  body('product_quantity')
    .isInt({ min: 0 }).withMessage('재고는 0 이상이어야 합니다'),

  body('category_id')
    .isInt().withMessage('카테고리 ID가 필요합니다'),

  validateResult
];

/**
 * 주문 검증
 */
const validateOrder = [
  body('order_recipient_name')
    .trim()
    .notEmpty().withMessage('수령인 이름은 필수입니다'),

  body('order_recipient_phone')
    .matches(/^010-\d{4}-\d{4}$/).withMessage('휴대폰 형식이 올바르지 않습니다'),
```

```
  body('order_recipient_address')
    .trim()
    .notEmpty().withMessage('배송지 주소는 필수입니다'),

  validateResult
];

module.exports = {
  validateProduct,
  validateOrder,
  validateResult
};
```

9.3 보안 미들웨어

```javascript
```

```javascript
// app.js
const helmet = require('helmet');
const cors = require('cors');
const rateLimit = require('express-rate-limit');

// 보안 헤더
app.use(helmet());

// CORS 설정
app.use(cors({
  origin: process.env.ALLOWED_ORIGINS?.split(',') || ['http://localhost:3000'],
  credentials: true
}));

// Rate Limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15분
  max: 100, // 최대 100 요청
  message: '너무 많은 요청을 보냈습니다. 잠시 후 다시 시도해주세요.'
});

app.use('/api', limiter);

// API 특정 엔드포인트에 대한 엄격한 제한
const authLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 5, // 15분에 5번만 허용
  message: '로그인 시도 횟수를 초과했습니다.'
});

app.use('/api/v1/auth/login', authLimiter);
```

---

10. 테스트 표준

10.1 테스트 구조

```javascript
```

```
// tests/unit/services/product.service.test.js
const ProductService = require('../../../src/services/product.service');
const ProductRepository = require('../../../src/repositories/product.repository');

// Mock 설정
jest.mock('../../../src/repositories/product.repository');

describe('ProductService', () => {
  describe('getProducts', () => {
    it('필터 없이 상품 목록을 조회할 수 있다', async () => {
      // Given
      const mockProducts = [
        { product_id: 1, product_name: '테스트 상품' }
      ];
      ProductRepository.findMany.mockResolvedValue(mockProducts);
      ProductRepository.count.mockResolvedValue(1);

      // When
      const result = await ProductService.getProducts({}, { page: 1, limit: 20 });

      // Then
      expect(result.products).toEqual(mockProducts);
      expect(result.pagination.total).toBe(1);
    });

    it('카테고리 필터로 상품을 조회할 수 있다', async () => {
      // Given
      const filters = { categoryId: 10 };
      ProductRepository.findMany.mockResolvedValue([]);

      // When
      await ProductService.getProducts(filters, {});

      // Then
      expect(ProductRepository.findMany).toHaveBeenCalledWith(
        expect.objectContaining({ categoryId: 10 }),
        expect.any(Object)
      );
    });
  });

  describe('createProduct', () => {
    it('유효한 데이터로 상품을 생성할 수 있다', async () => {
      // Given
      const userId = 1;
      const productData = {
```

```
      product_name: '새 상품',
      product_price: 10000
    };

    // When & Then
    // 테스트 구현...
  });

  it('가격이 0 이하이면 에러를 반환한다', async () => {
    // Given
    const productData = {
      product_name: '새 상품',
      product_price: -1000
    };

    // When & Then
    await expect(
      ProductService.createProduct(1, productData)
    ).rejects.toThrow('가격은 0보다 커야 합니다');
  });
 });
});
```

10.2 통합 테스트

javascript

```
// tests/integration/product.test.js
const request = require('supertest');
const app = require('../../src/app');
const { PrismaClient } = require('@prisma/client');

const prisma = new PrismaClient();

describe('Product API Integration Tests', () => {
  let authToken;

  beforeAll(async () => {
    // 테스트 데이터베이스 초기화
    await prisma.$executeRaw`TRUNCATE TABLE product CASCADE`;

    // 테스트 사용자 로그인
    const response = await request(app)
      .post('/api/v1/auth/login')
      .send({
        email: 'test@example.com',
        password: 'password123'
      });

    authToken = response.body.data.token;
  });

  afterAll(async () => {
    await prisma.$disconnect();
  });

  describe('GET /api/v1/products', () => {
    it('상품 목록을 조회할 수 있다', async () => {
      const response = await request(app)
        .get('/api/v1/products')
        .expect(200);

      expect(response.body.success).toBe(true);
      expect(Array.isArray(response.body.data.products)).toBe(true);
    });
  });

  describe('POST /api/v1/products', () => {
    it('인증된 사용자는 상품을 생성할 수 있다', async () => {
      const productData = {
        product_name: '테스트 상품',
        product_price: 10000,
        category_id: 1
```

```
    };

    const response = await request(app)
      .post('/api/v1/products')
      .set('Authorization', `Bearer ${authToken}`)
      .send(productData)
      .expect(201);

    expect(response.body.success).toBe(true);
    expect(response.body.data.product_name).toBe('테스트 상품');
  });

  it('인증 없이는 상품을 생성할 수 없다', async () => {
    await request(app)
      .post('/api/v1/products')
      .send({ product_name: '테스트' })
      .expect(401);
  });
 });
});
```

---

11. 배포 및 운영

11.1 환경 구성

환경	용도	URL	데이터베이스
Development	로컬 개발	localhost:3000	로컬 Supabase
Staging	통합 테스트	staging.fleecat.com	Staging DB
Production	실제 서비스	api.fleecat.com	Production DB

11.2 배포 체크리스트

```
markdown
```

```
## 배포 전 체크리스트

### 코드 품질
- [ ] ESLint 검사 통과
- [ ] 모든 테스트 통과
- [ ] 코드 리뷰 완료

### 환경 설정
- [ ] 환경변수 설정 확인
- [ ] Prisma 마이그레이션 적용
- [ ] 데이터베이스 백업 완료

### 보안
- [ ] API 키 및 시크릿 환경변수 확인
- [ ] CORS 설정 확인
- [ ] Rate Limiting 설정 확인

### 성능
- [ ] 데이터베이스 인덱스 확인
- [ ] 쿼리 최적화 확인
- [ ] 로그 레벨 설정

### 문서
- [ ] API 문서 업데이트
- [ ] CHANGELOG 작성
- [ ] README 업데이트
```

11.3 배포 스크립트

```bash
```

```
#!/bin/bash
# scripts/deploy.sh

echo "🚀 Fleecat 배포 시작..."

# 1. 환경 확인
if [ -z "$NODE_ENV" ]; then
  echo "❌ NODE_ENV가 설정되지 않았습니다"
  exit 1
fi

echo "📦 환경: $NODE_ENV"

# 2. 의존성 설치
echo "📥 패키지 설치 중..."
npm ci

# 3. Prisma 마이그레이션
echo "🗄️ 데이터베이스 마이그레이션..."
npx prisma migrate deploy

# 4. Prisma Client 생성
echo "⚙️ Prisma Client 생성..."
npx prisma generate

# 5. 테스트 실행
echo "🧪 테스트 실행..."
npm test

# 6. 빌드 (필요시)
# npm run build

# 7. 서버 재시작
echo "🔄 서버 재시작..."
pm2 restart fleecat-api

echo "✅ 배포 완료!"
```

11.4 로깅 전략

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: [
    // 에러 로그 파일
    new winston.transports.File({
      filename: 'logs/error.log',
      level: 'error'
    }),
    // 전체 로그 파일
    new winston.transports.File({
      filename: 'logs/combined.log'
    })
  ]
});

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

module.exports = logger;
```

11.5 모니터링

javascript

```
// 주요 모니터링 지표

1. API 성능
   - 응답 시간
   - 에러율
   - 요청 처리량 (RPS)

2. 데이터베이스
   - 쿼리 실행 시간
   - 커넥션 풀 사용률
   - 슬로우 쿼리

3. 시스템
   - CPU 사용률
   - 메모리 사용률
   - 디스크 사용률

4. 비즈니스 지표
   - 주문 생성 수
   - 결제 성공률
   - 사용자 활동
```

---

12. 버전 관리 및 협업

12.1 Git 브랜치 전략

```
main (프로덕션)
├── develop (개발)
│   ├── feature/product-management
│   ├── feature/order-system
│   ├── feature/payment-integration
│   └── feature/admin-dashboard
├── release/v1.0.0
└── hotfix/critical-bug-fix
```

12.2 브랜치 규칙

브랜치 타입	명명 규칙	설명
main	`main`	프로덕션 배포 브랜치
develop	`develop`	개발 통합 브랜치
feature	`feature/[기능명]`	새 기능 개발
bugfix	`bugfix/[버그명]`	버그 수정

브랜치 타입	명명 규칙	설명
hotfix	`hotfix/[긴급수정]`	긴급 수정
release	`release/v[버전]`	릴리스 준비

12.3 커밋 메시지 규칙

```
[타입]: [간단한 설명]

[상세 설명 (선택)]

[이슈 번호 (선택)]
```

커밋 타입

타입	설명	예시
feat	새로운 기능	`feat: 상품 필터링 기능 추가`
fix	버그 수정	`fix: 주문 생성 시 재고 검증 오류 수정`
docs	문서 수정	`docs: API 문서 업데이트`
style	코드 포맷팅	`style: ESLint 규칙 적용`
refactor	코드 리팩토링	`refactor: 상품 서비스 로직 개선`
test	테스트 추가	`test: 주문 API 통합 테스트 추가`
chore	빌드/설정 변경	`chore: 패키지 버전 업데이트`
perf	성능 개선	`perf: 상품 조회 쿼리 최적화`

예시

```bash
feat: 상품 장바구니 담기 기능 추가

- 장바구니 API 엔드포인트 구현
- 중복 상품 수량 증가 로직 추가
- 재고 부족 시 에러 처리

Closes #123
```

12.4 Pull Request 템플릿

```markdown
```

```
## 변경 사항
<!-- 이 PR에서 변경된 내용을 설명해주세요 -->

## 변경 이유
<!-- 왜 이 변경이 필요한지 설명해주세요 -->

## 테스트 방법
<!-- 이 변경사항을 어떻게 테스트할 수 있는지 설명해주세요 -->

## 체크리스트
- [ ] 코드가 컨벤션을 따릅니다
- [ ] 테스트를 추가/수정했습니다
- [ ] 문서를 업데이트했습니다
- [ ] 모든 테스트가 통과합니다
- [ ] 브레이킹 체인지가 없습니다

## 스크린샷 (선택)
<!-- API 응답이나 로그 등을 첨부해주세요 -->

## 관련 이슈
<!-- 관련된 이슈 번호를 적어주세요 -->
Closes #이슈번호
```

12.5 코드 리뷰 체크리스트

```
markdown
```

```
## 기능성
- [ ] 요구사항을 충족하는가?
- [ ] 엣지 케이스를 처리하는가?
- [ ] 에러 처리가 적절한가?

## 코드 품질
- [ ] 코딩 컨벤션을 따르는가?
- [ ] 함수/변수명이 명확한가?
- [ ] 중복 코드가 없는가?
- [ ] 주석이 적절한가?

## 성능
- [ ] 불필요한 데이터베이스 쿼리가 없는가?
- [ ] N+1 쿼리 문제가 없는가?
- [ ] 메모리 누수 가능성이 없는가?

## 보안
- [ ] 입력 검증이 충분한가?
- [ ] SQL 인젝션 위험이 없는가?
- [ ] 민감한 데이터가 노출되지 않는가?
- [ ] 인증/인가가 적절한가?

## 테스트
- [ ] 단위 테스트가 있는가?
- [ ] 통합 테스트가 필요한가?
- [ ] 테스트 커버리지가 충분한가?
```

부록

A. 자주 사용하는 Prisma 쿼리 패턴

```
javascript
```

```javascript
// 1. 기본 CRUD
const product = await prisma.product.create({ data: {...} });
const products = await prisma.product.findMany();
const product = await prisma.product.findUnique({ where: { product_id: 1 } });
const updated = await prisma.product.update({ where: {...}, data: {...} });
const deleted = await prisma.product.delete({ where: {...} });

// 2. 관계 포함 조회
const product = await prisma.product.findUnique({
  where: { product_id: 1 },
  include: {
    tenant_member: true,
    category: true,
    product_images: true
  }
});

// 3. 선택적 필드 조회
const products = await prisma.product.findMany({
  select: {
    product_id: true,
    product_name: true,
    product_price: true
  }
});

// 4. 필터링
const products = await prisma.product.findMany({
  where: {
    product_status: 'active',
    product_price: {
      gte: 10000,
      lte: 50000
    },
    OR: [
      { category_id: 1 },
      { category_id: 2 }
    ]
  }
});

// 5. 정렬 및 페이지네이션
const products = await prisma.product.findMany({
  orderBy: { product_created_at: 'desc' },
  skip: 20,
  take: 10
```

```
});

// 6. 집계
const count = await prisma.product.count();
const sum = await prisma.order.aggregate({
  _sum: { order_total_amount: true }
});
```

B. 환경변수 예시

bash

```
# .env.example

# Server
NODE_ENV=development
PORT=3000

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

# Supabase
SUPABASE_URL=https://xxxxx.supabase.co
SUPABASE_ANON_KEY=xxxxx
SUPABASE_SERVICE_KEY=xxxxx

# JWT
JWT_SECRET=your-secret-key-here
JWT_EXPIRES_IN=7d

# CORS
ALLOWED_ORIGINS=http://localhost:3000,http://localhost:3001

# File Upload
MAX_FILE_SIZE=5242880

# Rate Limiting
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100

# Logging
LOG_LEVEL=info
```

C. 유용한 NPM 스크립트

json

```
{
  "scripts": {
    "dev": "nodemon src/server.js",
    "start": "node src/server.js",
    "test": "jest --coverage",
    "test:watch": "jest --watch",
    "lint": "eslint src/**/*.js",
    "lint:fix": "eslint src/**/*.js --fix",
    "prisma:generate": "prisma generate",
    "prisma:migrate": "prisma migrate dev",
    "prisma:studio": "prisma studio",
    "prisma:seed": "node prisma/seed.js"
  }
}
```

문서 이력

버전	날짜	변경 내용	작성자
1.0	2025.09.30	최초 작성	백엔드팀

문서 끝

이 문서는 Fleecat 백엔드 개발의 모든 표준을 정의합니다.
모든 개발자는 이 가이드를 숙지하고 준수해야 합니다.