YMV 관리 프로그램 개발

1단계 : 엑셀 대용 + DB + 프린트 [ 자료 저장과 관리 ]

2단계 : 자동화 진행.

**핵심 사용기능**

**총무 기능**

* **구매품 관리**
  + 사무용품
  + 판매제품 (핫런너 및 판매용)
* **지출 요청서**
  + 결제 받기: 프린터 후 결제

**영업 기능**

* **견적서 관리**
  + 견적서 작성, 수정, 관리
  + 견적서 프린트, PDF

**기초 데이터**

* **제품 코드**
  + CSV로 업로드하여 사용

**설계 문서 작성 순서**

1. **시스템 아키텍처 설계**
   * 전체 구조도
   * 모듈 간 관계도
   * 데이터 흐름도
2. **데이터베이스 스키마 설계**
   * 테이블 구조
   * 관계도 (ERD)
   * 인덱스 설계
3. **기능별 상세 설계**
   * 각 기능의 세부 명세
   * 함수/클래스 설계
   * API 설계
4. **UI/UX 설계**
   * 화면 구성도
   * 메뉴 구조
   * 사용자 플로우
5. **개발 가이드라인**
   * 코딩 컨벤션
   * 파일 구조
   * 함수 명명 규칙

. **YMV 관리 프로그램 전체 설계 문서**

**📋 목차**

1. [시스템 아키텍처 설계](https://claude.ai/chat/591efa0c-8f3b-4dab-adc4-220d12145ff8#1-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%95%84%ED%82%A4%ED%85%8D%EC%B2%98-%EC%84%A4%EA%B3%84)
2. [데이터베이스 스키마 설계](https://claude.ai/chat/591efa0c-8f3b-4dab-adc4-220d12145ff8#2-%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4-%EC%8A%A4%ED%82%A4%EB%A7%88-%EC%84%A4%EA%B3%84)
3. [기능별 상세 설계](https://claude.ai/chat/591efa0c-8f3b-4dab-adc4-220d12145ff8#3-%EA%B8%B0%EB%8A%A5%EB%B3%84-%EC%83%81%EC%84%B8-%EC%84%A4%EA%B3%84)
4. [UI/UX 설계](https://claude.ai/chat/591efa0c-8f3b-4dab-adc4-220d12145ff8#4-uiux-%EC%84%A4%EA%B3%84)
5. [개발 가이드라인](https://claude.ai/chat/591efa0c-8f3b-4dab-adc4-220d12145ff8#5-%EA%B0%9C%EB%B0%9C-%EA%B0%80%EC%9D%B4%EB%93%9C%EB%9D%BC%EC%9D%B8)

**1. 시스템 아키텍처 설계**

**1.1 전체 구조도**

┌───────────────────────────────────┐

│ Frontend (Streamlit) │

├───────────────────────────────────┤

│ ┌─────────┐ ┌─────────┐ ┌─────────┐ │

│ │ 총무 │ │ 영업 │ │ 시스템 │ │

│ │ 모듈 │ │ 모듈 │ │ 관리 │ │

│ └─────────┘ └─────────┘ └─────────┘ │

├───────────────────────────────────┤

│ Shared Components │

│ ┌─────────┐ ┌─────────┐ ┌─────────┐ │

│ │ Auth │ │ Utils │ │Language │ │

│ └─────────┘ └─────────┘ └─────────┘ │

├────────────────────────────────────┤

│ Backend Services (Python) │

│ ┌─────────┐ ┌─────────┐ ┌─────────┐ │

│ │ DB │ │ File │ │ PDF │ │

│ │ Service │ │ Service │ │ Service │ │

│ └─────────┘ └─────────┘ └─────────┘ │

├───────────────────────────────────┤

│ Database (PostgreSQL) │

└──────────────────────────────────┘

**1.2 폴더 구조**

ymv\_management/

├── app/

│ ├── main.py # 메인 애플리케이션

│ ├── config/

│ │ ├── \_\_init\_\_.py

│ │ ├── database.py # DB 설정

│ │ ├── settings.py # 시스템 설정

│ │ └── constants.py # 상수 정의

│ ├── modules/

│ │ ├── \_\_init\_\_.py

│ │ ├── auth/

│ │ │ ├── \_\_init\_\_.py

│ │ │ ├── auth\_service.py # 인증 서비스

│ │ │ └── user\_management.py # 사용자 관리

│ │ ├── general\_affairs/ # 총무 모듈

│ │ │ ├── \_\_init\_\_.py

│ │ │ ├── purchase\_management.py # 구매 관리

│ │ │ └── expense\_request.py # 지출 요청서

│ │ ├── sales/ # 영업 모듈

│ │ │ ├── \_\_init\_\_.py

│ │ │ ├── quotation\_management.py # 견적서 관리

│ │ │ └── customer\_management.py # 고객 관리

│ │ └── system/ # 시스템 관리

│ │ ├── \_\_init\_\_.py

│ │ ├── product\_management.py # 제품 관리

│ │ └── code\_management.py # 코드 관리

│ ├── shared/

│ │ ├── \_\_init\_\_.py

│ │ ├── database.py # DB 연결

│ │ ├── utils.py # 공통 유틸

│ │ ├── file\_handler.py # 파일 처리

│ │ ├── pdf\_generator.py # PDF 생성

│ │ ├── language\_manager.py # 다국어 관리

│ │ └── components.py # UI 컴포넌트

│ └── static/

│ ├── css/

│ │ └── style.css

│ ├── js/

│ └── images/

├── database/

│ ├── schema.sql # DB 스키마

│ ├── initial\_data.sql # 초기 데이터

│ └── migrations/ # 마이그레이션

├── templates/ # 문서 템플릿

│ ├── quotation\_template.html # 견적서 템플릿

│ └── expense\_request\_template.html # 지출요청서 템플릿

├── uploads/ # 업로드 파일

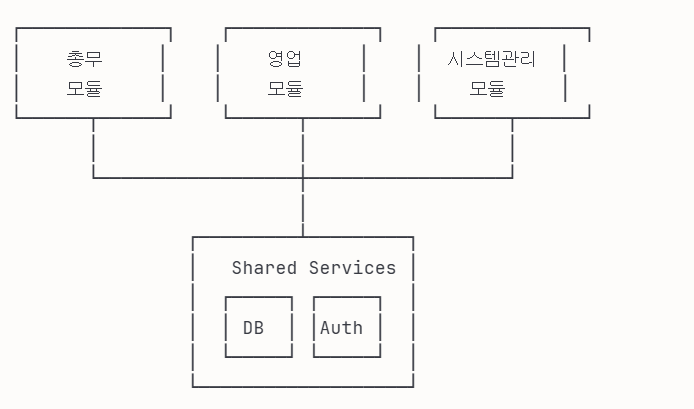
├── exports/ # 내보내기 파일

├── requirements.txt

├── .env

└── README.md

**1.3 모듈 간 관계도**

****

**2. 데이터베이스 스키마 설계**

**2.1 핵심 테이블 목록**

**인증 및 사용자 관리**

* users - 사용자 정보
* user\_sessions - 세션 관리

**제품 및 코드 관리**

* product\_categories - 제품 카테고리
* products - 제품 마스터
* product\_codes - 제품 코드

**총무 관리**

* purchase\_categories - 구매 카테고리
* purchase\_items - 구매 품목
* expense\_requests - 지출 요청서

**영업 관리**

* customers - 고객 정보
* quotations - 견적서 헤더
* quotation\_items - 견적서 상세

**시스템 설정**

* system\_settings - 시스템 설정
* translations - 다국어 번역

**2.2 상세 스키마**

YMV 데이터베이스 스키마

-- YMV 관리 프로그램 데이터베이스 스키마

-- PostgreSQL 기준

-- 1. 인증 및 사용자 관리

CREATE TABLE users (

user\_id SERIAL PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

full\_name VARCHAR(100) NOT NULL,

email VARCHAR(100),

department VARCHAR(50),

position VARCHAR(50),

is\_active BOOLEAN DEFAULT true,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE user\_sessions (

session\_id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(user\_id),

session\_token VARCHAR(255) UNIQUE NOT NULL,

expires\_at TIMESTAMP NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- 2. 제품 및 코드 관리

CREATE TABLE product\_categories (

category\_id SERIAL PRIMARY KEY,

category\_code VARCHAR(20) UNIQUE NOT NULL,

category\_name VARCHAR(100) NOT NULL,

parent\_id INTEGER REFERENCES product\_categories(category\_id),

level INTEGER CHECK (level >= 1 AND level <= 6),

sort\_order INTEGER DEFAULT 0,

is\_active BOOLEAN DEFAULT true,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE products (

product\_id SERIAL PRIMARY KEY,

product\_code VARCHAR(50) UNIQUE NOT NULL,

product\_name VARCHAR(200) NOT NULL,

category\_id INTEGER REFERENCES product\_categories(category\_id),

specification TEXT,

unit VARCHAR(20) DEFAULT 'EA',

cost\_price DECIMAL(15,2),

selling\_price DECIMAL(15,2),

currency VARCHAR(3) DEFAULT 'USD',

is\_active BOOLEAN DEFAULT true,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- 3. 총무 관리 - 구매 관리

CREATE TABLE purchase\_categories (

category\_id SERIAL PRIMARY KEY,

category\_name VARCHAR(50) NOT NULL, -- office, sales\_product, hotrunner

category\_name\_kr VARCHAR(50) NOT NULL,

description TEXT,

is\_active BOOLEAN DEFAULT true

);

CREATE TABLE purchase\_items (

item\_id SERIAL PRIMARY KEY,

item\_number VARCHAR(50) UNIQUE NOT NULL, -- YMV-Pyymmdd-count

category\_id INTEGER REFERENCES purchase\_categories(category\_id),

item\_name VARCHAR(200) NOT NULL,

quantity DECIMAL(10,2) NOT NULL,

unit VARCHAR(20) DEFAULT 'EA',

unit\_price DECIMAL(15,2),

total\_price DECIMAL(15,2),

currency VARCHAR(3) DEFAULT 'USD',

supplier\_name VARCHAR(200),

request\_date DATE NOT NULL,

needed\_date DATE,

status VARCHAR(20) DEFAULT 'pending', -- pending, approved, ordered, received

notes TEXT,

requested\_by INTEGER REFERENCES users(user\_id),

approved\_by INTEGER REFERENCES users(user\_id),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- 4. 총무 관리 - 지출 요청서

CREATE TABLE expense\_requests (

request\_id SERIAL PRIMARY KEY,

request\_number VARCHAR(50) UNIQUE NOT NULL, -- YMV-Eyymmdd-count

request\_date DATE NOT NULL,

requester\_id INTEGER REFERENCES users(user\_id),

department VARCHAR(50),

expense\_type VARCHAR(50), -- business\_trip, office\_supply, entertainment, etc.

description TEXT NOT NULL,

amount DECIMAL(15,2) NOT NULL,

currency VARCHAR(3) DEFAULT 'USD',

payment\_method VARCHAR(50), -- cash, bank\_transfer, corporate\_card

status VARCHAR(20) DEFAULT 'pending', -- pending, approved, rejected, paid

approved\_by INTEGER REFERENCES users(user\_id),

approved\_at TIMESTAMP,

paid\_at TIMESTAMP,

notes TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- 5. 영업 관리 - 고객 관리

CREATE TABLE customers (

customer\_id SERIAL PRIMARY KEY,

customer\_code VARCHAR(50) UNIQUE,

company\_name VARCHAR(200) NOT NULL,

contact\_person VARCHAR(100),

phone VARCHAR(50),

email VARCHAR(100),

address TEXT,

country VARCHAR(50),

business\_registration VARCHAR(50),

tax\_number VARCHAR(50),

payment\_terms INTEGER DEFAULT 30, -- 결제 조건 (일)

credit\_limit DECIMAL(15,2),

is\_active BOOLEAN DEFAULT true,

notes TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- 6. 영업 관리 - 견적서

CREATE TABLE quotations (

quotation\_id SERIAL PRIMARY KEY,

quotation\_number VARCHAR(50) UNIQUE NOT NULL, -- YMV-Qyymmdd-count

customer\_id INTEGER REFERENCES customers(customer\_id),

quotation\_date DATE NOT NULL,

valid\_until DATE,

total\_amount DECIMAL(15,2),

currency VARCHAR(3) DEFAULT 'USD',

exchange\_rate DECIMAL(10,4) DEFAULT 1,

discount\_rate DECIMAL(5,2) DEFAULT 0,

tax\_rate DECIMAL(5,2) DEFAULT 0,

payment\_terms VARCHAR(200),

delivery\_terms VARCHAR(200),

notes TEXT,

status VARCHAR(20) DEFAULT 'draft', -- draft, sent, accepted, rejected, expired

created\_by INTEGER REFERENCES users(user\_id),

approved\_by INTEGER REFERENCES users(user\_id),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE quotation\_items (

item\_id SERIAL PRIMARY KEY,

quotation\_id INTEGER REFERENCES quotations(quotation\_id),

product\_id INTEGER REFERENCES products(product\_id),

product\_code VARCHAR(50),

product\_name VARCHAR(200),

specification TEXT,

quantity DECIMAL(10,2) NOT NULL,

unit VARCHAR(20) DEFAULT 'EA',

unit\_price DECIMAL(15,2) NOT NULL,

total\_price DECIMAL(15,2) NOT NULL,

delivery\_days INTEGER,

sort\_order INTEGER DEFAULT 0

);

-- 7. 시스템 관리

CREATE TABLE system\_settings (

setting\_id SERIAL PRIMARY KEY,

setting\_key VARCHAR(100) UNIQUE NOT NULL,

setting\_value TEXT,

setting\_type VARCHAR(20) DEFAULT 'string', -- string, number, boolean, json

description TEXT,

updated\_by INTEGER REFERENCES users(user\_id),

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE translations (

translation\_id SERIAL PRIMARY KEY,

translation\_key VARCHAR(100) NOT NULL,

language\_code VARCHAR(5) NOT NULL, -- ko, en, vn

translation\_value TEXT NOT NULL,

module\_name VARCHAR(50),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

UNIQUE(translation\_key, language\_code)

);

-- 8. 인덱스 생성

CREATE INDEX idx\_users\_username ON users(username);

CREATE INDEX idx\_users\_active ON users(is\_active);

CREATE INDEX idx\_products\_code ON products(product\_code);

CREATE INDEX idx\_products\_category ON products(category\_id);

CREATE INDEX idx\_products\_active ON products(is\_active);

CREATE INDEX idx\_purchase\_items\_number ON purchase\_items(item\_number);

CREATE INDEX idx\_purchase\_items\_status ON purchase\_items(status);

CREATE INDEX idx\_purchase\_items\_date ON purchase\_items(request\_date);

CREATE INDEX idx\_expense\_requests\_number ON expense\_requests(request\_number);

CREATE INDEX idx\_expense\_requests\_status ON expense\_requests(status);

CREATE INDEX idx\_expense\_requests\_date ON expense\_requests(request\_date);

CREATE INDEX idx\_quotations\_number ON quotations(quotation\_number);

CREATE INDEX idx\_quotations\_customer ON quotations(customer\_id);

CREATE INDEX idx\_quotations\_status ON quotations(status);

CREATE INDEX idx\_quotations\_date ON quotations(quotation\_date);

CREATE INDEX idx\_customers\_code ON customers(customer\_code);

CREATE INDEX idx\_customers\_active ON customers(is\_active);

-- 9. 초기 데이터 삽입

-- Master 사용자

INSERT INTO users (username, password\_hash, full\_name, department, position)

VALUES ('Master', '$2b$12$LQv3c1yqBWVHxkd0LHAkCOYz6TtxMQJqhN8/LewAm4A6/YY3z.k.i', 'System Master', 'IT', 'Master');

-- 구매 카테고리

INSERT INTO purchase\_categories (category\_name, category\_name\_kr, description) VALUES

('office', '사무용품', '사무실에서 사용하는 일반 용품'),

('sales\_product', '판매제품', '판매를 위한 제품'),

('hotrunner', '핫런너', '핫런너 관련 제품 및 부품');

-- 시스템 설정

INSERT INTO system\_settings (setting\_key, setting\_value, setting\_type, description) VALUES

('company\_name', 'YMV Company', 'string', '회사명'),

('default\_currency', 'USD', 'string', '기본 통화'),

('default\_language', 'ko', 'string', '기본 언어'),

('quotation\_validity\_days', '30', 'number', '견적서 유효기간 (일)'),

('auto\_approve\_limit', '1000', 'number', '자동 승인 한도 (USD)');

-- 기본 번역 데이터

INSERT INTO translations (translation\_key, language\_code, translation\_value, module\_name) VALUES

('menu\_dashboard', 'ko', '대시보드', 'common'),

('menu\_general\_affairs', 'ko', '총무', 'common'),

('menu\_sales', 'ko', '영업', 'common'),

('menu\_system', 'ko', '시스템 관리', 'common'),

('menu\_purchase', 'ko', '구매 관리', 'general\_affairs'),

('menu\_expense', 'ko', '지출 요청', 'general\_affairs'),

('menu\_quotation', 'ko', '견적서 관리', 'sales'),

('menu\_customer', 'ko', '고객 관리', 'sales'),

('menu\_product', 'ko', '제품 관리', 'system'),

('btn\_save', 'ko', '저장', 'common'),

('btn\_cancel', 'ko', '취소', 'common'),

('btn\_edit', 'ko', '수정', 'common'),

('btn\_delete', 'ko', '삭제', 'common'),

('btn\_print', 'ko', '인쇄', 'common'),

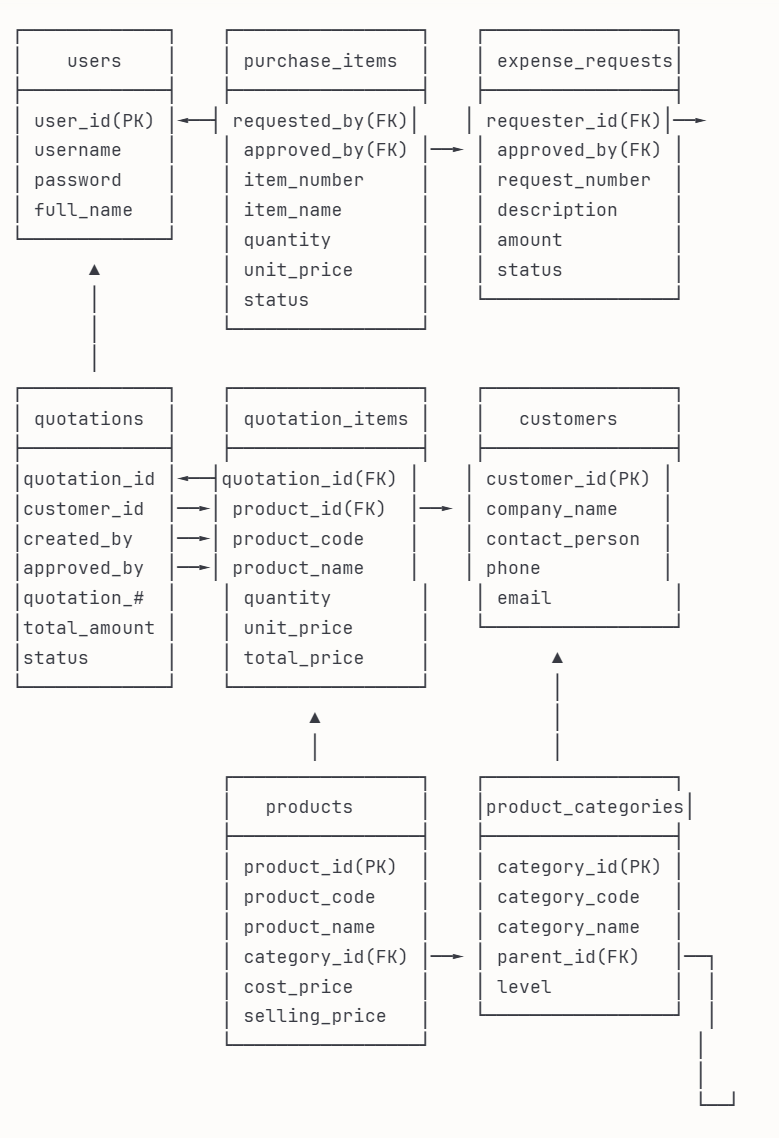
('btn\_export', 'ko', '내보내기', 'common'),

('status\_pending', 'ko', '대기중', 'common'),

('status\_approved', 'ko', '승인됨', 'common'),

('status\_rejected', 'ko', '거부됨', 'common');

**2.3 ERD (Entity Relationship Diagram)**



**3. 기능별 상세 설계**

**3.1 인증 시스템 (Authentication)**

*# 인증 시스템 함수 설계*

*# modules/auth/auth\_service.py*

import hashlib

import secrets

import bcrypt

from datetime import datetime, timedelta

from typing import Optional, Dict, Any

class AuthService:

"""인증 서비스 클래스"""

def \_\_init\_\_(self, db\_connection):

self.db = db\_connection

self.session\_duration = timedelta(hours=8) *# 8시간 세션*

def hash\_password(self, password: str) -> str:

"""비밀번호 해시화"""

salt = bcrypt.gensalt()

hashed = bcrypt.hashpw(password.encode('utf-8'), salt)

return hashed.decode('utf-8')

def verify\_password(self, password: str, hashed: str) -> bool:

"""비밀번호 검증"""

return bcrypt.checkpw(password.encode('utf-8'), hashed.encode('utf-8'))

def generate\_session\_token(self) -> str:

"""세션 토큰 생성"""

return secrets.token\_urlsafe(32)

def login(self, username: str, password: str) -> Optional[Dict[str, Any]]:

"""사용자 로그인"""

try:

*# 사용자 조회*

query = """

SELECT user\_id, username, password\_hash, full\_name,

department, position, is\_active

FROM users

WHERE username = %s AND is\_active = true

"""

result = self.db.execute\_query(query, (username,))

if not result:

return None

user = result[0]

*# 비밀번호 검증*

if not self.verify\_password(password, user['password\_hash']):

return None

*# 세션 토큰 생성*

session\_token = self.generate\_session\_token()

expires\_at = datetime.now() + self.session\_duration

*# 세션 저장*

session\_query = """

INSERT INTO user\_sessions (user\_id, session\_token, expires\_at)

VALUES (%s, %s, %s)

"""

self.db.execute\_query(session\_query, (user['user\_id'], session\_token, expires\_at))

*# 사용자 정보 반환 (비밀번호 제외)*

user\_info = {

'user\_id': user['user\_id'],

'username': user['username'],

'full\_name': user['full\_name'],

'department': user['department'],

'position': user['position'],

'session\_token': session\_token,

'expires\_at': expires\_at

}

return user\_info

except Exception as e:

print(f"Login error: {e}")

return None

def logout(self, session\_token: str) -> bool:

"""사용자 로그아웃"""

try:

query = "DELETE FROM user\_sessions WHERE session\_token = %s"

self.db.execute\_query(query, (session\_token,))

return True

except Exception as e:

print(f"Logout error: {e}")

return False

def validate\_session(self, session\_token: str) -> Optional[Dict[str, Any]]:

"""세션 검증"""

try:

query = """

SELECT us.user\_id, us.expires\_at, u.username, u.full\_name,

u.department, u.position

FROM user\_sessions us

JOIN users u ON us.user\_id = u.user\_id

WHERE us.session\_token = %s AND us.expires\_at > %s

"""

result = self.db.execute\_query(query, (session\_token, datetime.now()))

if result:

return result[0]

return None

except Exception as e:

print(f"Session validation error: {e}")

return None

def cleanup\_expired\_sessions(self):

"""만료된 세션 정리"""

try:

query = "DELETE FROM user\_sessions WHERE expires\_at < %s"

self.db.execute\_query(query, (datetime.now(),))

except Exception as e:

print(f"Session cleanup error: {e}")

def change\_password(self, user\_id: int, old\_password: str, new\_password: str) -> bool:

"""비밀번호 변경"""

try:

*# 현재 비밀번호 확인*

query = "SELECT password\_hash FROM users WHERE user\_id = %s"

result = self.db.execute\_query(query, (user\_id,))

if not result:

return False

if not self.verify\_password(old\_password, result[0]['password\_hash']):

return False

*# 새 비밀번호 해시화*

new\_hash = self.hash\_password(new\_password)

*# 비밀번호 업데이트*

update\_query = """

UPDATE users

SET password\_hash = %s, updated\_at = %s

WHERE user\_id = %s

"""

self.db.execute\_query(update\_query, (new\_hash, datetime.now(), user\_id))

return True

except Exception as e:

print(f"Password change error: {e}")

return False

def is\_master\_user(self, username: str) -> bool:

"""마스터 사용자 확인"""

return username.lower() == 'master'

*# 인증 관련 유틸리티 함수들*

def require\_login(func):

"""로그인 필요 데코레이터"""

def wrapper(\*args, \*\*kwargs):

if 'user\_info' not in st.session\_state:

st.error("로그인이 필요합니다.")

st.stop()

return func(\*args, \*\*kwargs)

return wrapper

def require\_master(func):

"""마스터 권한 필요 데코레이터"""

def wrapper(\*args, \*\*kwargs):

if 'user\_info' not in st.session\_state:

st.error("로그인이 필요합니다.")

st.stop()

if st.session\_state.user\_info['username'].lower() != 'master':

st.error("마스터 권한이 필요합니다.")

st.stop()

return func(\*args, \*\*kwargs)

return wrapper

**3.2 제품 관리 시스템 (Product Management)**

*# 제품 관리 시스템 함수 설계*

*# modules/system/product\_management.py*

import pandas as pd

import io

from typing import List, Dict, Any, Optional

from datetime import datetime

class ProductManagement:

"""제품 관리 클래스"""

def \_\_init\_\_(self, db\_connection):

self.db = db\_connection

def generate\_product\_code(self, category\_id: int, sequence: int = None) -> str:

"""제품 코드 자동 생성

Args:

category\_id: 카테고리 ID

sequence: 시퀀스 번호 (없으면 자동 생성)

Returns:

생성된 제품 코드 (예: CAT01-001)

"""

try:

*# 카테고리 코드 조회*

category\_query = """

SELECT category\_code FROM product\_categories

WHERE category\_id = %s

"""

category\_result = self.db.execute\_query(category\_query, (category\_id,))

if not category\_result:

raise ValueError("Invalid category ID")

category\_code = category\_result[0]['category\_code']

*# 시퀀스 번호 생성 (없는 경우)*

if sequence is None:

sequence\_query = """

SELECT COUNT(\*) + 1 as next\_seq

FROM products p

JOIN product\_categories pc ON p.category\_id = pc.category\_id

WHERE pc.category\_code = %s

"""

seq\_result = self.db.execute\_query(sequence\_query, (category\_code,))

sequence = seq\_result[0]['next\_seq']

return f"{category\_code}-{sequence:03d}"

except Exception as e:

print(f"Product code generation error: {e}")

return None

def create\_product(self, product\_data: Dict[str, Any]) -> bool:

"""제품 생성

Args:

product\_data: 제품 정보 딕셔너리

- product\_name: 제품명

- category\_id: 카테고리 ID

- specification: 사양

- unit: 단위

- cost\_price: 원가

- selling\_price: 판매가

- currency: 통화

Returns:

성공 여부

"""

try:

*# 제품 코드 생성*

product\_code = self.generate\_product\_code(product\_data['category\_id'])

if not product\_code:

return False

insert\_query = """

INSERT INTO products (

product\_code, product\_name, category\_id, specification,

unit, cost\_price, selling\_price, currency, created\_at

) VALUES (

%s, %s, %s, %s, %s, %s, %s, %s, %s

)

"""

values = (

product\_code,

product\_data['product\_name'],

product\_data['category\_id'],

product\_data.get('specification', ''),

product\_data.get('unit', 'EA'),

product\_data.get('cost\_price', 0),

product\_data.get('selling\_price', 0),

product\_data.get('currency', 'USD'),

datetime.now()

)

self.db.execute\_query(insert\_query, values)

return True

except Exception as e:

print(f"Product creation error: {e}")

return False

def get\_products(self, filters: Dict[str, Any] = None) -> List[Dict[str, Any]]:

"""제품 목록 조회

Args:

filters: 필터 조건

- category\_id: 카테고리 ID

- is\_active: 활성 상태

- search\_term: 검색어

Returns:

제품 목록

"""

try:

base\_query = """

SELECT p.product\_id, p.product\_code, p.product\_name,

pc.category\_name, p.specification, p.unit,

p.cost\_price, p.selling\_price, p.currency,

p.is\_active, p.created\_at

FROM products p

LEFT JOIN product\_categories pc ON p.category\_id = pc.category\_id

WHERE 1=1

"""

params = []

if filters:

if 'category\_id' in filters and filters['category\_id']:

base\_query += " AND p.category\_id = %s"

params.append(filters['category\_id'])

if 'is\_active' in filters:

base\_query += " AND p.is\_active = %s"

params.append(filters['is\_active'])

if 'search\_term' in filters and filters['search\_term']:

base\_query += " AND (p.product\_name ILIKE %s OR p.product\_code ILIKE %s)"

search\_param = f"%{filters['search\_term']}%"

params.extend([search\_param, search\_param])

base\_query += " ORDER BY p.product\_code"

return self.db.execute\_query(base\_query, params)

except Exception as e:

print(f"Get products error: {e}")

return []

def get\_product\_by\_id(self, product\_id: int) -> Optional[Dict[str, Any]]:

"""제품 상세 조회"""

try:

query = """

SELECT p.\*, pc.category\_name

FROM products p

LEFT JOIN product\_categories pc ON p.category\_id = pc.category\_id

WHERE p.product\_id = %s

"""

result = self.db.execute\_query(query, (product\_id,))

return result[0] if result else None

except Exception as e:

print(f"Get product by ID error: {e}")

return None

def update\_product(self, product\_id: int, product\_data: Dict[str, Any]) -> bool:

"""제품 정보 수정"""

try:

update\_query = """

UPDATE products SET

product\_name = %s,

category\_id = %s,

specification = %s,

unit = %s,

cost\_price = %s,

selling\_price = %s,

currency = %s,

updated\_at = %s

WHERE product\_id = %s

"""

values = (

product\_data['product\_name'],

product\_data['category\_id'],

product\_data.get('specification', ''),

product\_data.get('unit', 'EA'),

product\_data.get('cost\_price', 0),

product\_data.get('selling\_price', 0),

product\_data.get('currency', 'USD'),

datetime.now(),

product\_id

)

self.db.execute\_query(update\_query, values)

return True

except Exception as e:

print(f"Product update error: {e}")

return False

def delete\_product(self, product\_id: int) -> bool:

"""제품 삭제 (소프트 삭제)"""

try:

query = """

UPDATE products

SET is\_active = false, updated\_at = %s

WHERE product\_id = %s

"""

self.db.execute\_query(query, (datetime.now(), product\_id))

return True

except Exception as e:

print(f"Product deletion error: {e}")

return False

def import\_products\_from\_csv(self, csv\_file) -> Dict[str, Any]:

"""CSV 파일에서 제품 데이터 가져오기

Args:

csv\_file: 업로드된 CSV 파일

Returns:

처리 결과 (성공/실패 건수, 오류 목록)

"""

try:

*# CSV 파일 읽기*

df = pd.read\_csv(csv\_file)

*# 필수 컬럼 확인*

required\_columns = ['product\_name', 'category\_code']

missing\_columns = [col for col in required\_columns if col not in df.columns]

if missing\_columns:

return {

'success': False,

'message': f"필수 컬럼이 없습니다: {', '.join(missing\_columns)}"

}

success\_count = 0

error\_count = 0

errors = []

for index, row in df.iterrows():

try:

*# 카테고리 ID 조회*

category\_query = """

SELECT category\_id FROM product\_categories

WHERE category\_code = %s AND is\_active = true

"""

category\_result = self.db.execute\_query(

category\_query, (row['category\_code'],)

)

if not category\_result:

errors.append(f"행 {index + 1}: 카테고리 코드 '{row['category\_code']}'를 찾을 수 없습니다.")

error\_count += 1

continue

*# 제품 데이터 준비*

product\_data = {

'product\_name': row['product\_name'],

'category\_id': category\_result[0]['category\_id'],

'specification': row.get('specification', ''),

'unit': row.get('unit', 'EA'),

'cost\_price': float(row.get('cost\_price', 0)),

'selling\_price': float(row.get('selling\_price', 0)),

'currency': row.get('currency', 'USD')

}

*# 제품 생성*

if self.create\_product(product\_data):

success\_count += 1

else:

errors.append(f"행 {index + 1}: 제품 생성 실패")

error\_count += 1

except Exception as e:

errors.append(f"행 {index + 1}: {str(e)}")

error\_count += 1

return {

'success': True,

'success\_count': success\_count,

'error\_count': error\_count,

'errors': errors

}

except Exception as e:

return {

'success': False,

'message': f"CSV 파일 처리 오류: {str(e)}"

}

def export\_products\_to\_csv(self, filters: Dict[str, Any] = None) -> io.StringIO:

"""제품 데이터를 CSV로 내보내기"""

try:

products = self.get\_products(filters)

*# DataFrame 생성*

df = pd.DataFrame(products)

*# 컬럼 순서 및 이름 정리*

if not df.empty:

df = df[['product\_code', 'product\_name', 'category\_name',

'specification', 'unit', 'cost\_price', 'selling\_price',

'currency', 'is\_active']]

df.columns = ['제품코드', '제품명', '카테고리', '사양', '단위',

'원가', '판매가', '통화', '활성상태']

*# CSV 문자열로 변환*

output = io.StringIO()

df.to\_csv(output, index=False, encoding='utf-8-sig')

output.seek(0)

return output

except Exception as e:

print(f"Export products error: {e}")

return None

class CategoryManagement:

"""제품 카테고리 관리 클래스"""

def \_\_init\_\_(self, db\_connection):

self.db = db\_connection

def create\_category(self, category\_data: Dict[str, Any]) -> bool:

"""카테고리 생성"""

try:

insert\_query = """

INSERT INTO product\_categories (

category\_code, category\_name, parent\_id, level, sort\_order

) VALUES (%s, %s, %s, %s, %s)

"""

values = (

category\_data['category\_code'],

category\_data['category\_name'],

category\_data.get('parent\_id'),

category\_data['level'],

category\_data.get('sort\_order', 0)

)

self.db.execute\_query(insert\_query, values)

return True

except Exception as e:

print(f"Category creation error: {e}")

return False

def get\_categories\_tree(self) -> List[Dict[str, Any]]:

"""카테고리 트리 구조 조회"""

try:

query = """

SELECT category\_id, category\_code, category\_name,

parent\_id, level, sort\_order, is\_active

FROM product\_categories

WHERE is\_active = true

ORDER BY level, sort\_order, category\_name

"""

return self.db.execute\_query(query)

except Exception as e:

print(f"Get categories tree error: {e}")

return []

def get\_categories\_by\_level(self, level: int) -> List[Dict[str, Any]]:

"""레벨별 카테고리 조회"""

try:

query = """

SELECT category\_id, category\_code, category\_name, parent\_id

FROM product\_categories

WHERE level = %s AND is\_active = true

ORDER BY sort\_order, category\_name

"""

return self.db.execute\_query(query, (level,))

except Exception as e:

print(f"Get categories by level error: {e}")

return []

**3.3 견적서 관리 시스템 (Quotation Management)**

**# 견적서 관리 시스템 함수 설계**

**# modules/sales/quotation\_management.py**

**from typing import List, Dict, Any, Optional**

**from datetime import datetime, timedelta**

**import re**

**class QuotationManagement:**

**"""견적서 관리 클래스"""**

**def \_\_init\_\_(self, db\_connection):**

**self.db = db\_connection**

**self.default\_validity\_days = 30**

**def generate\_quotation\_number(self, date: datetime = None) -> str:**

**"""견적서 번호 자동 생성**

**Format: YMV-Qyymmdd-count**

**Args:**

**date: 견적서 날짜 (기본값: 오늘)**

**Returns:**

**생성된 견적서 번호**

**"""**

**try:**

**if date is None:**

**date = datetime.now()**

**# 날짜 포맷 (yymmdd)**

**date\_str = date.strftime("%y%m%d")**

**# 해당 날짜의 견적서 개수 조회**

**count\_query = """**

**SELECT COUNT(\*) as count**

**FROM quotations**

**WHERE quotation\_number LIKE %s**

**"""**

**pattern = f"YMV-Q{date\_str}-%"**

**result = self.db.execute\_query(count\_query, (pattern,))**

**count = result[0]['count'] + 1**

**return f"YMV-Q{date\_str}-{count:03d}"**

**except Exception as e:**

**print(f"Quotation number generation error: {e}")**

**return None**

**def create\_quotation(self, quotation\_data: Dict[str, Any], items: List[Dict[str, Any]]) -> Optional[int]:**

**"""견적서 생성**

**Args:**

**quotation\_data: 견적서 헤더 정보**

**items: 견적서 상세 항목 리스트**

**Returns:**

**생성된 견적서 ID (실패시 None)**

**"""**

**try:**

**# 견적서 번호 생성**

**quotation\_number = self.generate\_quotation\_number(quotation\_data.get('quotation\_date'))**

**if not quotation\_number:**

**return None**

**# 유효기간 계산**

**quotation\_date = quotation\_data.get('quotation\_date', datetime.now().date())**

**valid\_until = quotation\_date + timedelta(days=self.default\_validity\_days)**

**# 총액 계산**

**total\_amount = sum(item['total\_price'] for item in items)**

**# 견적서 헤더 생성**

**header\_query = """**

**INSERT INTO quotations (**

**quotation\_number, customer\_id, quotation\_date, valid\_until,**

**total\_amount, currency, exchange\_rate, discount\_rate, tax\_rate,**

**payment\_terms, delivery\_terms, notes, status, created\_by, created\_at**

**) VALUES (**

**%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s**

**) RETURNING quotation\_id**

**"""**

**header\_values = (**

**quotation\_number,**

**quotation\_data['customer\_id'],**

**quotation\_date,**

**valid\_until,**

**total\_amount,**

**quotation\_data.get('currency', 'USD'),**

**quotation\_data.get('exchange\_rate', 1.0),**

**quotation\_data.get('discount\_rate', 0),**

**quotation\_data.get('tax\_rate', 0),**

**quotation\_data.get('payment\_terms', ''),**

**quotation\_data.get('delivery\_terms', ''),**

**quotation\_data.get('notes', ''),**

**'draft',**

**quotation\_data['created\_by'],**

**datetime.now()**

**)**

**result = self.db.execute\_query(header\_query, header\_values)**

**quotation\_id = result[0]['quotation\_id']**

**# 견적서 상세 항목 생성**

**for sort\_order, item in enumerate(items, 1):**

**item\_query = """**

**INSERT INTO quotation\_items (**

**quotation\_id, product\_id, product\_code, product\_name,**

**specification, quantity, unit, unit\_price, total\_price,**

**delivery\_days, sort\_order**

**) VALUES (**

**%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s**

**)**

**"""**

**item\_values = (**

**quotation\_id,**

**item.get('product\_id'),**

**item['product\_code'],**

**item['product\_name'],**

**item.get('specification', ''),**

**item['quantity'],**

**item.get('unit', 'EA'),**

**item['unit\_price'],**

**item['total\_price'],**

**item.get('delivery\_days'),**

**sort\_order**

**)**

**self.db.execute\_query(item\_query, item\_values)**

**return quotation\_id**

**except Exception as e:**

**print(f"Quotation creation error: {e}")**

**return None**

**def get\_quotations(self, filters: Dict[str, Any] = None) -> List[Dict[str, Any]]:**

**"""견적서 목록 조회**

**Args:**

**filters: 필터 조건**

**- customer\_id: 고객 ID**

**- status: 상태**

**- date\_from: 시작 날짜**

**- date\_to: 종료 날짜**

**- search\_term: 검색어**

**Returns:**

**견적서 목록**

**"""**

**try:**

**base\_query = """**

**SELECT q.quotation\_id, q.quotation\_number, q.quotation\_date,**

**q.valid\_until, q.total\_amount, q.currency, q.status,**

**c.company\_name, u.full\_name as created\_by\_name,**

**q.created\_at**

**FROM quotations q**

**LEFT JOIN customers c ON q.customer\_id = c.customer\_id**

**LEFT JOIN users u ON q.created\_by = u.user\_id**

**WHERE 1=1**

**"""**

**params = []**

**if filters:**

**if 'customer\_id' in filters and filters['customer\_id']:**

**base\_query += " AND q.customer\_id = %s"**

**params.append(filters['customer\_id'])**

**if 'status' in filters and filters['status']:**

**base\_query += " AND q.status = %s"**

**params.append(filters['status'])**

**if 'date\_from' in filters and filters['date\_from']:**

**base\_query += " AND q.quotation\_date >= %s"**

**params.append(filters['date\_from'])**

**if 'date\_to' in filters and filters['date\_to']:**

**base\_query += " AND q.quotation\_date <= %s"**

**params.append(filters['date\_to'])**

**if 'search\_term' in filters and filters['search\_term']:**

**base\_query += " AND (q.quotation\_number ILIKE %s OR c.company\_name ILIKE %s)"**

**search\_param = f"%{filters['search\_term']}%"**

**params.extend([search\_param, search\_param])**

**base\_query += " ORDER BY q.quotation\_date DESC, q.quotation\_number DESC"**

**return self.db.execute\_query(base\_query, params)**

**except Exception as e:**

**print(f"Get quotations error: {e}")**

**return []**

**def get\_quotation\_details(self, quotation\_id: int) -> Optional[Dict[str, Any]]:**

**"""견적서 상세 조회**

**Args:**

**quotation\_id: 견적서 ID**

**Returns:**

**견적서 상세 정보 (헤더 + 항목)**

**"""**

**try:**

**# 헤더 정보 조회**

**header\_query = """**

**SELECT q.\*, c.company\_name, c.contact\_person, c.phone, c.email,**

**c.address, u.full\_name as created\_by\_name**

**FROM quotations q**

**LEFT JOIN customers c ON q.customer\_id = c.customer\_id**

**LEFT JOIN users u ON q.created\_by = u.user\_id**

**WHERE q.quotation\_id = %s**

**"""**

**header\_result = self.db.execute\_query(header\_query, (quotation\_id,))**

**if not header\_result:**

**return None**

**header = header\_result[0]**

**# 상세 항목 조회**

**items\_query = """**

**SELECT qi.\*, p.specification as product\_specification**

**FROM quotation\_items qi**

**LEFT JOIN products p ON qi.product\_id = p.product\_id**

**WHERE qi.quotation\_id = %s**

**ORDER BY qi.sort\_order**

**"""**

**items = self.db.execute\_query(items\_query, (quotation\_id,))**

**return {**

**'header': header,**

**'items': items**

**}**

**except Exception as e:**

**print(f"Get quotation details error: {e}")**

**return None**

**def update\_quotation(self, quotation\_id: int, quotation\_data: Dict[str, Any],**

**items: List[Dict[str, Any]]) -> bool:**

**"""견적서 수정"""**

**try:**

**# 총액 재계산**

**total\_amount = sum(item['total\_price'] for item in items)**

**# 헤더 업데이트**

**header\_query = """**

**UPDATE quotations SET**

**customer\_id = %s,**

**quotation\_date = %s,**

**valid\_until = %s,**

**total\_amount = %s,**

**currency = %s,**

**exchange\_rate = %s,**

**discount\_rate = %s,**

**tax\_rate = %s,**

**payment\_terms = %s,**

**delivery\_terms = %s,**

**notes = %s,**

**updated\_at = %s**

**WHERE quotation\_id = %s**

**"""**

**header\_values = (**

**quotation\_data['customer\_id'],**

**quotation\_data['quotation\_date'],**

**quotation\_data['valid\_until'],**

**total\_amount,**

**quotation\_data.get('currency', 'USD'),**

**quotation\_data.get('exchange\_rate', 1.0),**

**quotation\_data.get('discount\_rate', 0),**

**quotation\_data.get('tax\_rate', 0),**

**quotation\_data.get('payment\_terms', ''),**

**quotation\_data.get('delivery\_terms', ''),**

**quotation\_data.get('notes', ''),**

**datetime.now(),**

**quotation\_id**

**)**

**self.db.execute\_query(header\_query, header\_values)**

**# 기존 상세 항목 삭제**

**delete\_items\_query = "DELETE FROM quotation\_items WHERE quotation\_id = %s"**

**self.db.execute\_query(delete\_items\_query, (quotation\_id,))**

**# 새로운 상세 항목 추가**

**for sort\_order, item in enumerate(items, 1):**

**item\_query = """**

**INSERT INTO quotation\_items (**

**quotation\_id, product\_id, product\_code, product\_name,**

**specification, quantity, unit, unit\_price, total\_price,**

**delivery\_days, sort\_order**

**) VALUES (**

**%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s**

**)**

**"""**

**item\_values = (**

**quotation\_id,**

**item.get('product\_id'),**

**item['product\_code'],**

**item['product\_name'],**

**item.get('specification', ''),**

**item['quantity'],**

**item.get('unit', 'EA'),**

**item['unit\_price'],**

**item['total\_price'],**

**item.get('delivery\_days'),**

**sort\_order**

**)**

**self.db.execute\_query(item\_query, item\_values)**

**return True**

**except Exception as e:**

**print(f"Quotation update error: {e}")**

**return False**

**def update\_quotation\_status(self, quotation\_id: int, status: str) -> bool:**

**"""견적서 상태 변경"""**

**try:**

**query = """**

**UPDATE quotations**

**SET status = %s, updated\_at = %s**

**WHERE quotation\_id = %s**

**"""**

**self.db.execute\_query(query, (status, datetime.now(), quotation\_id))**

**return True**

**except Exception as e:**

**print(f"Quotation status update error: {e}")**

**return False**

**def delete\_quotation(self, quotation\_id: int) -> bool:**

**"""견적서 삭제"""**

**try:**

**# 상세 항목 삭제**

**delete\_items\_query = "DELETE FROM quotation\_items WHERE quotation\_id = %s"**

**self.db.execute\_query(delete\_items\_query, (quotation\_id,))**

**# 헤더 삭제**

**delete\_header\_query = "DELETE FROM quotations WHERE quotation\_id = %s"**

**self.db.execute\_query(delete\_header\_query, (quotation\_id,))**

**return True**

**except Exception as e:**

**print(f"Quotation deletion error: {e}")**

**return False**

**def get\_quotation\_statistics(self, year: int = None, month: int = None) -> Dict[str, Any]:**

**"""견적서 통계 조회"""**

**try:**

**base\_query = """**

**SELECT**

**COUNT(\*) as total\_count,**

**COUNT(CASE WHEN status = 'draft' THEN 1 END) as draft\_count,**

**COUNT(CASE WHEN status = 'sent' THEN 1 END) as sent\_count,**

**COUNT(CASE WHEN status = 'accepted' THEN 1 END) as accepted\_count,**

**COUNT(CASE WHEN status = 'rejected' THEN 1 END) as rejected\_count,**

**SUM(total\_amount) as total\_amount,**

**SUM(CASE WHEN status = 'accepted' THEN total\_amount ELSE 0 END) as accepted\_amount**

**FROM quotations**

**WHERE 1=1**

**"""**

**params = []**

**if year:**

**base\_query += " AND EXTRACT(YEAR FROM quotation\_date) = %s"**

**params.append(year)**

**if month:**

**base\_query += " AND EXTRACT(MONTH FROM quotation\_date) = %s"**

**params.append(month)**

**result = self.db.execute\_query(base\_query, params)**

**return result[0] if result else {}**

**except Exception as e:**

**print(f"Quotation statistics error: {e}")**

**return {}**

**class QuotationValidator:**

**"""견적서 유효성 검증 클래스"""**

**@staticmethod**

**def validate\_quotation\_data(data: Dict[str, Any]) -> List[str]:**

**"""견적서 데이터 유효성 검증**

**Returns:**

**오류 메시지 목록 (빈 리스트면 유효함)**

**"""**

**errors = []**

**# 필수 필드 검증**

**required\_fields = ['customer\_id', 'quotation\_date']**

**for field in required\_fields:**

**if field not in data or not data[field]:**

**errors.append(f"{field}는 필수 입력 항목입니다.")**

**# 날짜 검증**

**if 'quotation\_date' in data and data['quotation\_date']:**

**if data['quotation\_date'] > datetime.now().date():**

**errors.append("견적일은 오늘 이후 날짜일 수 없습니다.")**

**# 환율 검증**

**if 'exchange\_rate' in data and data['exchange\_rate']:**

**if data['exchange\_rate'] <= 0:**

**errors.append("환율은 0보다 큰 값이어야 합니다.")**

**# 할인율 검증**

**if 'discount\_rate' in data and data['discount\_rate']:**

**if not (0 <= data['discount\_rate'] <= 100):**

**errors.append("할인율은 0-100% 사이의 값이어야 합니다.")**

**return errors**

**@staticmethod**

**def validate\_quotation\_items(items: List[Dict[str, Any]]) -> List[str]:**

**"""견적서 항목 유효성 검증"""**

**errors = []**

**if not items:**

**errors.append("견적서 항목이 없습니다.")**

**return errors**

**for i, item in enumerate(items, 1):**

**# 필수 필드 검증**

**required\_fields = ['product\_code', 'product\_name', 'quantity', 'unit\_price']**

**for field in required\_fields:**

**if field not in item or not item[field]:**

**errors.append(f"항목 {i}: {field}는 필수 입력입니다.")**

**# 수량 검증**

**if 'quantity' in item and item['quantity']:**

**if item['quantity'] <= 0:**

**errors.append(f"항목 {i}: 수량은 0보다 큰 값이어야 합니다.")**

**# 단가 검증**

**if 'unit\_price' in item and item['unit\_price']:**

**if item['unit\_price'] <= 0:**

**errors.append(f"항목 {i}: 단가는 0보다 큰 값이어야 합니다.")**

**return errors**

**3.4 구매 관리 시스템 (Purchase Management)**

**# 구매 관리 시스템 함수 설계**

**# modules/general\_affairs/purchase\_management.py**

**from typing import List, Dict, Any, Optional**

**from datetime import datetime, timedelta**

**import pandas as pd**

**import io**

**class PurchaseManagement:**

**"""구매 관리 클래스"""**

**def \_\_init\_\_(self, db\_connection):**

**self.db = db\_connection**

**def generate\_purchase\_number(self, date: datetime = None) -> str:**

**"""구매 번호 자동 생성**

**Format: YMV-Pyymmdd-count**

**Args:**

**date: 구매 요청 날짜 (기본값: 오늘)**

**Returns:**

**생성된 구매 번호**

**"""**

**try:**

**if date is None:**

**date = datetime.now()**

**# 날짜 포맷 (yymmdd)**

**date\_str = date.strftime("%y%m%d")**

**# 해당 날짜의 구매 요청 개수 조회**

**count\_query = """**

**SELECT COUNT(\*) as count**

**FROM purchase\_items**

**WHERE item\_number LIKE %s**

**"""**

**pattern = f"YMV-P{date\_str}-%"**

**result = self.db.execute\_query(count\_query, (pattern,))**

**count = result[0]['count'] + 1**

**return f"YMV-P{date\_str}-{count:03d}"**

**except Exception as e:**

**print(f"Purchase number generation error: {e}")**

**return None**

**def create\_purchase\_item(self, purchase\_data: Dict[str, Any]) -> Optional[int]:**

**"""구매 품목 생성**

**Args:**

**purchase\_data: 구매 품목 정보**

**- category\_id: 구매 카테고리 ID**

**- item\_name: 품목명**

**- quantity: 수량**

**- unit: 단위**

**- unit\_price: 단가**

**- supplier\_name: 공급업체**

**- request\_date: 요청일**

**- needed\_date: 필요일**

**- notes: 비고**

**- requested\_by: 요청자 ID**

**Returns:**

**생성된 구매 품목 ID (실패시 None)**

**"""**

**try:**

**# 구매 번호 생성**

**item\_number = self.generate\_purchase\_number(purchase\_data.get('request\_date'))**

**if not item\_number:**

**return None**

**# 총 금액 계산**

**total\_price = purchase\_data['quantity'] \* purchase\_data['unit\_price']**

**insert\_query = """**

**INSERT INTO purchase\_items (**

**item\_number, category\_id, item\_name, quantity, unit,**

**unit\_price, total\_price, currency, supplier\_name,**

**request\_date, needed\_date, notes, requested\_by, created\_at**

**) VALUES (**

**%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s**

**) RETURNING item\_id**

**"""**

**values = (**

**item\_number,**

**purchase\_data['category\_id'],**

**purchase\_data['item\_name'],**

**purchase\_data['quantity'],**

**purchase\_data.get('unit', 'EA'),**

**purchase\_data['unit\_price'],**

**total\_price,**

**purchase\_data.get('currency', 'USD'),**

**purchase\_data.get('supplier\_name', ''),**

**purchase\_data['request\_date'],**

**purchase\_data.get('needed\_date'),**

**purchase\_data.get('notes', ''),**

**purchase\_data['requested\_by'],**

**datetime.now()**

**)**

**result = self.db.execute\_query(insert\_query, values)**

**return result[0]['item\_id']**

**except Exception as e:**

**print(f"Purchase item creation error: {e}")**

**return None**

**def get\_purchase\_items(self, filters: Dict[str, Any] = None) -> List[Dict[str, Any]]:**

**"""구매 품목 목록 조회**

**Args:**

**filters: 필터 조건**

**- category\_id: 카테고리 ID**

**- status: 상태**

**- date\_from: 시작 날짜**

**- date\_to: 종료 날짜**

**- requested\_by: 요청자 ID**

**- search\_term: 검색어**

**Returns:**

**구매 품목 목록**

**"""**

**try:**

**base\_query = """**

**SELECT pi.item\_id, pi.item\_number, pi.item\_name, pi.quantity,**

**pi.unit, pi.unit\_price, pi.total\_price, pi.currency,**

**pi.supplier\_name, pi.request\_date, pi.needed\_date,**

**pi.status, pi.notes,**

**pc.category\_name\_kr as category\_name,**

**u1.full\_name as requested\_by\_name,**

**u2.full\_name as approved\_by\_name,**

**pi.created\_at**

**FROM purchase\_items pi**

**LEFT JOIN purchase\_categories pc ON pi.category\_id = pc.category\_id**

**LEFT JOIN users u1 ON pi.requested\_by = u1.user\_id**

**LEFT JOIN users u2 ON pi.approved\_by = u2.user\_id**

**WHERE 1=1**

**"""**

**params = []**

**if filters:**

**if 'category\_id' in filters and filters['category\_id']:**

**base\_query += " AND pi.category\_id = %s"**

**params.append(filters['category\_id'])**

**if 'status' in filters and filters['status']:**

**base\_query += " AND pi.status = %s"**

**params.append(filters['status'])**

**if 'date\_from' in filters and filters['date\_from']:**

**base\_query += " AND pi.request\_date >= %s"**

**params.append(filters['date\_from'])**

**if 'date\_to' in filters and filters['date\_to']:**

**base\_query += " AND pi.request\_date <= %s"**

**params.append(filters['date\_to'])**

**if 'requested\_by' in filters and filters['requested\_by']:**

**base\_query += " AND pi.requested\_by = %s"**

**params.append(filters['requested\_by'])**

**if 'search\_term' in filters and filters['search\_term']:**

**base\_query += " AND (pi.item\_name ILIKE %s OR pi.item\_number ILIKE %s OR pi.supplier\_name ILIKE %s)"**

**search\_param = f"%{filters['search\_term']}%"**

**params.extend([search\_param, search\_param, search\_param])**

**base\_query += " ORDER BY pi.request\_date DESC, pi.item\_number DESC"**

**return self.db.execute\_query(base\_query, params)**

**except Exception as e:**

**print(f"Get purchase items error: {e}")**

**return []**

**def get\_purchase\_item\_by\_id(self, item\_id: int) -> Optional[Dict[str, Any]]:**

**"""구매 품목 상세 조회"""**

**try:**

**query = """**

**SELECT pi.\*, pc.category\_name\_kr as category\_name,**

**u1.full\_name as requested\_by\_name,**

**u2.full\_name as approved\_by\_name**

**FROM purchase\_items pi**

**LEFT JOIN purchase\_categories pc ON pi.category\_id = pc.category\_id**

**LEFT JOIN users u1 ON pi.requested\_by = u1.user\_id**

**LEFT JOIN users u2 ON pi.approved\_by = u2.user\_id**

**WHERE pi.item\_id = %s**

**"""**

**result = self.db.execute\_query(query, (item\_id,))**

**return result[0] if result else None**

**except Exception as e:**

**print(f"Get purchase item by ID error: {e}")**

**return None**

**def update\_purchase\_item(self, item\_id: int, purchase\_data: Dict[str, Any]) -> bool:**

**"""구매 품목 수정"""**

**try:**

**# 총 금액 재계산**

**total\_price = purchase\_data['quantity'] \* purchase\_data['unit\_price']**

**update\_query = """**

**UPDATE purchase\_items SET**

**category\_id = %s,**

**item\_name = %s,**

**quantity = %s,**

**unit = %s,**

**unit\_price = %s,**

**total\_price = %s,**

**currency = %s,**

**supplier\_name = %s,**

**request\_date = %s,**

**needed\_date = %s,**

**notes = %s,**

**updated\_at = %s**

**WHERE item\_id = %s**

**"""**

**values = (**

**purchase\_data['category\_id'],**

**purchase\_data['item\_name'],**

**purchase\_data['quantity'],**

**purchase\_data.get('unit', 'EA'),**

**purchase\_data['unit\_price'],**

**total\_price,**

**purchase\_data.get('currency', 'USD'),**

**purchase\_data.get('supplier\_name', ''),**

**purchase\_data['request\_date'],**

**purchase\_data.get('needed\_date'),**

**purchase\_data.get('notes', ''),**

**datetime.now(),**

**item\_id**

**)**

**self.db.execute\_query(update\_query, values)**

**return True**

**except Exception as e:**

**print(f"Purchase item update error: {e}")**

**return False**

**def update\_purchase\_status(self, item\_id: int, status: str, approved\_by: int = None) -> bool:**

**"""구매 상태 변경"""**

**try:**

**if status == 'approved' and approved\_by:**

**query = """**

**UPDATE purchase\_items**

**SET status = %s, approved\_by = %s, updated\_at = %s**

**WHERE item\_id = %s**

**"""**

**params = (status, approved\_by, datetime.now(), item\_id)**

**else:**

**query = """**

**UPDATE purchase\_items**

**SET status = %s, updated\_at = %s**

**WHERE item\_id = %s**

**"""**

**params = (status, datetime.now(), item\_id)**

**self.db.execute\_query(query, params)**

**return True**

**except Exception as e:**

**print(f"Purchase status update error: {e}")**

**return False**

**def delete\_purchase\_item(self, item\_id: int) -> bool:**

**"""구매 품목 삭제"""**

**try:**

**# 승인된 상태가 아닌 경우에만 삭제 가능**

**status\_query = "SELECT status FROM purchase\_items WHERE item\_id = %s"**

**result = self.db.execute\_query(status\_query, (item\_id,))**

**if result and result[0]['status'] in ['approved', 'ordered', 'received']:**

**return False # 승인 이후 상태에서는 삭제 불가**

**delete\_query = "DELETE FROM purchase\_items WHERE item\_id = %s"**

**self.db.execute\_query(delete\_query, (item\_id,))**

**return True**

**except Exception as e:**

**print(f"Purchase item deletion error: {e}")**

**return False**

**def get\_purchase\_categories(self) -> List[Dict[str, Any]]:**

**"""구매 카테고리 목록 조회"""**

**try:**

**query = """**

**SELECT category\_id, category\_name, category\_name\_kr, description**

**FROM purchase\_categories**

**WHERE is\_active = true**

**ORDER BY category\_name\_kr**

**"""**

**return self.db.execute\_query(query)**

**except Exception as e:**

**print(f"Get purchase categories error: {e}")**

**return []**

**def get\_purchase\_statistics(self, year: int = None, month: int = None) -> Dict[str, Any]:**

**"""구매 통계 조회"""**

**try:**

**base\_query = """**

**SELECT**

**COUNT(\*) as total\_count,**

**COUNT(CASE WHEN status = 'pending' THEN 1 END) as pending\_count,**

**COUNT(CASE WHEN status = 'approved' THEN 1 END) as approved\_count,**

**COUNT(CASE WHEN status = 'ordered' THEN 1 END) as ordered\_count,**

**COUNT(CASE WHEN status = 'received' THEN 1 END) as received\_count,**

**SUM(total\_price) as total\_amount,**

**SUM(CASE WHEN status = 'approved' THEN total\_price ELSE 0 END) as approved\_amount**

**FROM purchase\_items**

**WHERE 1=1**

**"""**

**params = []**

**if year:**

**base\_query += " AND EXTRACT(YEAR FROM request\_date) = %s"**

**params.append(year)**

**if month:**

**base\_query += " AND EXTRACT(MONTH FROM request\_date) = %s"**

**params.append(month)**

**result = self.db.execute\_query(base\_query, params)**

**return result[0] if result else {}**

**except Exception as e:**

**print(f"Purchase statistics error: {e}")**

**return {}**

**def export\_purchase\_items\_to\_csv(self, filters: Dict[str, Any] = None) -> io.StringIO:**

**"""구매 품목을 CSV로 내보내기"""**

**try:**

**items = self.get\_purchase\_items(filters)**

**# DataFrame 생성**

**df = pd.DataFrame(items)**

**# 컬럼 순서 및 이름 정리**

**if not df.empty:**

**df = df[['item\_number', 'category\_name', 'item\_name', 'quantity',**

**'unit', 'unit\_price', 'total\_price', 'currency',**

**'supplier\_name', 'request\_date', 'needed\_date', 'status',**

**'requested\_by\_name', 'approved\_by\_name', 'notes']]**

**df.columns = ['구매번호', '카테고리', '품목명', '수량', '단위',**

**'단가', '총금액', '통화', '공급업체', '요청일',**

**'필요일', '상태', '요청자', '승인자', '비고']**

**# CSV 문자열로 변환**

**output = io.StringIO()**

**df.to\_csv(output, index=False, encoding='utf-8-sig')**

**output.seek(0)**

**return output**

**except Exception as e:**

**print(f"Export purchase items error: {e}")**

**return None**

**class PurchaseApprovalWorkflow:**

**"""구매 승인 워크플로우 클래스"""**

**def \_\_init\_\_(self, db\_connection):**

**self.db = db\_connection**

**self.auto\_approval\_limit = 1000 # USD 기준 자동 승인 한도**

**def get\_approval\_limit(self) -> float:**

**"""자동 승인 한도 조회"""**

**try:**

**query = """**

**SELECT setting\_value FROM system\_settings**

**WHERE setting\_key = 'auto\_approve\_limit'**

**"""**

**result = self.db.execute\_query(query)**

**if result:**

**return float(result[0]['setting\_value'])**

**return self.auto\_approval\_limit**

**except Exception as e:**

**print(f"Get approval limit error: {e}")**

**return self.auto\_approval\_limit**

**def check\_auto\_approval(self, purchase\_amount: float) -> bool:**

**"""자동 승인 가능 여부 확인"""**

**approval\_limit = self.get\_approval\_limit()**

**return purchase\_amount <= approval\_limit**

**def auto\_approve\_purchase(self, item\_id: int, system\_user\_id: int = 1) -> bool:**

**"""자동 승인 처리"""**

**try:**

**# 구매 품목 정보 조회**

**item = self.get\_purchase\_item\_by\_id(item\_id)**

**if not item:**

**return False**

**# 자동 승인 가능 여부 확인**

**if not self.check\_auto\_approval(item['total\_price']):**

**return False**

**# 자동 승인 처리**

**purchase\_mgmt = PurchaseManagement(self.db)**

**return purchase\_mgmt.update\_purchase\_status(item\_id, 'approved', system\_user\_id)**

**except Exception as e:**

**print(f"Auto approve purchase error: {e}")**

**return False**

**def get\_pending\_approvals(self, user\_id: int = None) -> List[Dict[str, Any]]:**

**"""승인 대기 목록 조회"""**

**try:**

**query = """**

**SELECT pi.item\_id, pi.item\_number, pi.item\_name,**

**pi.total\_price, pi.currency, pi.request\_date,**

**u.full\_name as requested\_by\_name,**

**pc.category\_name\_kr as category\_name**

**FROM purchase\_items pi**

**LEFT JOIN users u ON pi.requested\_by = u.user\_id**

**LEFT JOIN purchase\_categories pc ON pi.category\_id = pc.category\_id**

**WHERE pi.status = 'pending'**

**ORDER BY pi.request\_date DESC**

**"""**

**return self.db.execute\_query(query)**

**except Exception as e:**

**print(f"Get pending approvals error: {e}")**

**return []**

**class PurchaseValidator:**

**"""구매 요청 유효성 검증 클래스"""**

**@staticmethod**

**def validate\_purchase\_data(data: Dict[str, Any]) -> List[str]:**

**"""구매 데이터 유효성 검증**

**Returns:**

**오류 메시지 목록 (빈 리스트면 유효함)**

**"""**

**errors = []**

**# 필수 필드 검증**

**required\_fields = ['category\_id', 'item\_name', 'quantity', 'unit\_price', 'request\_date']**

**for field in required\_fields:**

**if field not in data or not data[field]:**

**errors.append(f"{field}는 필수 입력 항목입니다.")**

**# 수량 검증**

**if 'quantity' in data and data['quantity']:**

**if data['quantity'] <= 0:**

**errors.append("수량은 0보다 큰 값이어야 합니다.")**

**# 단가 검증**

**if 'unit\_price' in data and data['unit\_price']:**

**if data['unit\_price'] <= 0:**

**errors.append("단가는 0보다 큰 값이어야 합니다.")**

**# 날짜 검증**

**if 'request\_date' in data and data['request\_date']:**

**if data['request\_date'] > datetime.now().date():**

**errors.append("요청일은 오늘 이후 날짜일 수 없습니다.")**

**if 'needed\_date' in data and data['needed\_date'] and 'request\_date' in data:**

**if data['needed\_date'] < data['request\_date']:**

**errors.append("필요일은 요청일 이후여야 합니다.")**

**return errors**

**3.5 지출 요청서 시스템 (Expense Request)**

**# 지출 요청서 시스템 함수 설계**

**# modules/general\_affairs/expense\_request.py**

**from typing import List, Dict, Any, Optional**

**from datetime import datetime**

**import pandas as pd**

**import io**

**class ExpenseRequestManagement:**

**"""지출 요청서 관리 클래스"""**

**def \_\_init\_\_(self, db\_connection):**

**self.db = db\_connection**

**def generate\_expense\_number(self, date: datetime = None) -> str:**

**"""지출 요청서 번호 자동 생성**

**Format: YMV-Eyymmdd-count**

**Args:**

**date: 요청 날짜 (기본값: 오늘)**

**Returns:**

**생성된 지출 요청서 번호**

**"""**

**try:**

**if date is None:**

**date = datetime.now()**

**# 날짜 포맷 (yymmdd)**

**date\_str = date.strftime("%y%m%d")**

**# 해당 날짜의 지출 요청서 개수 조회**

**count\_query = """**

**SELECT COUNT(\*) as count**

**FROM expense\_requests**

**WHERE request\_number LIKE %s**

**"""**

**pattern = f"YMV-E{date\_str}-%"**

**result = self.db.execute\_query(count\_query, (pattern,))**

**count = result[0]['count'] + 1**

**return f"YMV-E{date\_str}-{count:03d}"**

**except Exception as e:**

**print(f"Expense number generation error: {e}")**

**return None**

**def create\_expense\_request(self, expense\_data: Dict[str, Any]) -> Optional[int]:**

**"""지출 요청서 생성**

**Args:**

**expense\_data: 지출 요청 정보**

**- requester\_id: 요청자 ID**

**- department: 부서**

**- expense\_type: 지출 유형**

**- description: 설명**

**- amount: 금액**

**- currency: 통화**

**- payment\_method: 결제 방법**

**- request\_date: 요청일**

**- notes: 비고**

**Returns:**

**생성된 지출 요청서 ID (실패시 None)**

**"""**

**try:**

**# 요청서 번호 생성**

**request\_number = self.generate\_expense\_number(expense\_data.get('request\_date'))**

**if not request\_number:**

**return None**

**insert\_query = """**

**INSERT INTO expense\_requests (**

**request\_number, requester\_id, department, expense\_type,**

**description, amount, currency, payment\_method, request\_date,**

**notes, created\_at**

**) VALUES (**

**%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s**

**) RETURNING request\_id**

**"""**

**values = (**

**request\_number,**

**expense\_data['requester\_id'],**

**expense\_data.get('department', ''),**

**expense\_data['expense\_type'],**

**expense\_data['description'],**

**expense\_data['amount'],**

**expense\_data.get('currency', 'USD'),**

**expense\_data.get('payment\_method', 'bank\_transfer'),**

**expense\_data['request\_date'],**

**expense\_data.get('notes', ''),**

**datetime.now()**

**)**

**result = self.db.execute\_query(insert\_query, values)**

**return result[0]['request\_id']**

**except Exception as e:**

**print(f"Expense request creation error: {e}")**

**return None**

**def get\_expense\_requests(self, filters: Dict[str, Any] = None) -> List[Dict[str, Any]]:**

**"""지출 요청서 목록 조회**

**Args:**

**filters: 필터 조건**

**- requester\_id: 요청자 ID**

**- department: 부서**

**- expense\_type: 지출 유형**

**- status: 상태**

**- date\_from: 시작 날짜**

**- date\_to: 종료 날짜**

**- search\_term: 검색어**

**Returns:**

**지출 요청서 목록**

**"""**

**try:**

**base\_query = """**

**SELECT er.request\_id, er.request\_number, er.request\_date,**

**er.department, er.expense\_type, er.description,**

**er.amount, er.currency, er.payment\_method, er.status,**

**u1.full\_name as requester\_name,**

**u2.full\_name as approved\_by\_name,**

**er.approved\_at, er.paid\_at, er.created\_at**

**FROM expense\_requests er**

**LEFT JOIN users u1 ON er.requester\_id = u1.user\_id**

**LEFT JOIN users u2 ON er.approved\_by = u2.user\_id**

**WHERE 1=1**

**"""**

**params = []**

**if filters:**

**if 'requester\_id' in filters and filters['requester\_id']:**

**base\_query += " AND er.requester\_id = %s"**

**params.append(filters['requester\_id'])**

**if 'department' in filters and filters['department']:**

**base\_query += " AND er.department = %s"**

**params.append(filters['department'])**

**if 'expense\_type' in filters and filters['expense\_type']:**

**base\_query += " AND er.expense\_type = %s"**

**params.append(filters['expense\_type'])**

**if 'status' in filters and filters['status']:**

**base\_query += " AND er.status = %s"**

**params.append(filters['status'])**

**if 'date\_from' in filters and filters['date\_from']:**

**base\_query += " AND er.request\_date >= %s"**

**params.append(filters['date\_from'])**

**if 'date\_to' in filters and filters['date\_to']:**

**base\_query += " AND er.request\_date <= %s"**

**params.append(filters['date\_to'])**

**if 'search\_term' in filters and filters['search\_term']:**

**base\_query += " AND (er.request\_number ILIKE %s OR er.description ILIKE %s)"**

**search\_param = f"%{filters['search\_term']}%"**

**params.extend([search\_param, search\_param])**

**base\_query += " ORDER BY er.request\_date DESC, er.request\_number DESC"**

**return self.db.execute\_query(base\_query, params)**

**except Exception as e:**

**print(f"Get expense requests error: {e}")**

**return []**

**def get\_expense\_request\_by\_id(self, request\_id: int) -> Optional[Dict[str, Any]]:**

**"""지출 요청서 상세 조회"""**

**try:**

**query = """**

**SELECT er.\*, u1.full\_name as requester\_name,**

**u2.full\_name as approved\_by\_name**

**FROM expense\_requests er**

**LEFT JOIN users u1 ON er.requester\_id = u1.user\_id**

**LEFT JOIN users u2 ON er.approved\_by = u2.user\_id**

**WHERE er.request\_id = %s**

**"""**

**result = self.db.execute\_query(query, (request\_id,))**

**return result[0] if result else None**

**except Exception as e:**

**print(f"Get expense request by ID error: {e}")**

**return None**

**def update\_expense\_request(self, request\_id: int, expense\_data: Dict[str, Any]) -> bool:**

**"""지출 요청서 수정"""**

**try:**

**update\_query = """**

**UPDATE expense\_requests SET**

**department = %s,**

**expense\_type = %s,**

**description = %s,**

**amount = %s,**

**currency = %s,**

**payment\_method = %s,**

**request\_date = %s,**

**notes = %s,**

**updated\_at = %s**

**WHERE request\_id = %s AND status = 'pending'**

**"""**

**values = (**

**expense\_data.get('department', ''),**

**expense\_data['expense\_type'],**

**expense\_data['description'],**

**expense\_data['amount'],**

**expense\_data.get('currency', 'USD'),**

**expense\_data.get('payment\_method', 'bank\_transfer'),**

**expense\_data['request\_date'],**

**expense\_data.get('notes', ''),**

**datetime.now(),**

**request\_id**

**)**

**result = self.db.execute\_query(update\_query, values)**

**return result.rowcount > 0 if hasattr(result, 'rowcount') else True**

**except Exception as e:**

**print(f"Expense request update error: {e}")**

**return False**

**def approve\_expense\_request(self, request\_id: int, approved\_by: int, notes: str = None) -> bool:**

**"""지출 요청서 승인"""**

**try:**

**update\_query = """**

**UPDATE expense\_requests SET**

**status = 'approved',**

**approved\_by = %s,**

**approved\_at = %s,**

**notes = CASE**

**WHEN %s IS NOT NULL THEN CONCAT(COALESCE(notes, ''), '\n승인 메모: ', %s)**

**ELSE notes**

**END,**

**updated\_at = %s**

**WHERE request\_id = %s AND status = 'pending'**

**"""**

**values = (**

**approved\_by,**

**datetime.now(),**

**notes,**

**notes,**

**datetime.now(),**

**request\_id**

**)**

**self.db.execute\_query(update\_query, values)**

**return True**

**except Exception as e:**

**print(f"Expense request approval error: {e}")**

**return False**

**def reject\_expense\_request(self, request\_id: int, rejected\_by: int, reason: str) -> bool:**

**"""지출 요청서 거부"""**

**try:**

**update\_query = """**

**UPDATE expense\_requests SET**

**status = 'rejected',**

**approved\_by = %s,**

**approved\_at = %s,**

**notes = CONCAT(COALESCE(notes, ''), '\n거부 사유: ', %s),**

**updated\_at = %s**

**WHERE request\_id = %s AND status = 'pending'**

**"""**

**values = (**

**rejected\_by,**

**datetime.now(),**

**reason,**

**datetime.now(),**

**request\_id**

**)**

**self.db.execute\_query(update\_query, values)**

**return True**

**except Exception as e:**

**print(f"Expense request rejection error: {e}")**

**return False**

**def mark\_as\_paid(self, request\_id: int) -> bool:**

**"""지출 요청서 결제 완료 처리"""**

**try:**

**update\_query = """**

**UPDATE expense\_requests SET**

**status = 'paid',**

**paid\_at = %s,**

**updated\_at = %s**

**WHERE request\_id = %s AND status = 'approved'**

**"""**

**values = (datetime.now(), datetime.now(), request\_id)**

**self.db.execute\_query(update\_query, values)**

**return True**

**except Exception as e:**

**print(f"Mark expense as paid error: {e}")**

**return False**

**def delete\_expense\_request(self, request\_id: int) -> bool:**

**"""지출 요청서 삭제 (대기 상태에서만 가능)"""**

**try:**

**# 상태 확인**

**status\_query = "SELECT status FROM expense\_requests WHERE request\_id = %s"**

**result = self.db.execute\_query(status\_query, (request\_id,))**

**if result and result[0]['status'] != 'pending':**

**return False # 대기 상태가 아니면 삭제 불가**

**delete\_query = "DELETE FROM expense\_requests WHERE request\_id = %s"**

**self.db.execute\_query(delete\_query, (request\_id,))**

**return True**

**except Exception as e:**

**print(f"Expense request deletion error: {e}")**

**return False**

**def get\_expense\_types(self) -> List[Dict[str, str]]:**

**"""지출 유형 목록 조회"""**

**expense\_types = [**

**{'value': 'business\_trip', 'label': '출장비'},**

**{'value': 'office\_supply', 'label': '사무용품'},**

**{'value': 'entertainment', 'label': '접대비'},**

**{'value': 'training', 'label': '교육비'},**

**{'value': 'transport', 'label': '교통비'},**

**{'value': 'meal', 'label': '식비'},**

**{'value': 'communication', 'label': '통신비'},**

**{'value': 'equipment', 'label': '장비구입'},**

**{'value': 'maintenance', 'label': '유지보수'},**

**{'value': 'marketing', 'label': '마케팅'},**

**{'value': 'other', 'label': '기타'}**

**]**

**return expense\_types**

**def get\_payment\_methods(self) -> List[Dict[str, str]]:**

**"""결제 방법 목록 조회"""**

**payment\_methods = [**

**{'value': 'cash', 'label': '현금'},**

**{'value': 'bank\_transfer', 'label': '계좌이체'},**

**{'value': 'corporate\_card', 'label': '법인카드'},**

**{'value': 'check', 'label': '수표'},**

**{'value': 'other', 'label': '기타'}**

**]**

**return payment\_methods**

**def get\_expense\_statistics(self, year: int = None, month: int = None) -> Dict[str, Any]:**

**"""지출 요청서 통계 조회"""**

**try:**

**base\_query = """**

**SELECT**

**COUNT(\*) as total\_count,**

**COUNT(CASE WHEN status = 'pending' THEN 1 END) as pending\_count,**

**COUNT(CASE WHEN status = 'approved' THEN 1 END) as approved\_count,**

**COUNT(CASE WHEN status = 'rejected' THEN 1 END) as rejected\_count,**

**COUNT(CASE WHEN status = 'paid' THEN 1 END) as paid\_count,**

**SUM(amount) as total\_amount,**

**SUM(CASE WHEN status = 'approved' THEN amount ELSE 0 END) as approved\_amount,**

**SUM(CASE WHEN status = 'paid' THEN amount ELSE 0 END) as paid\_amount**

**FROM expense\_requests**

**WHERE 1=1**

**"""**

**params = []**

**if year:**

**base\_query += " AND EXTRACT(YEAR FROM request\_date) = %s"**

**params.append(year)**

**if month:**

**base\_query += " AND EXTRACT(MONTH FROM request\_date) = %s"**

**params.append(month)**

**result = self.db.execute\_query(base\_query, params)**

**return result[0] if result else {}**

**except Exception as e:**

**print(f"Expense statistics error: {e}")**

**return {}**

**def get\_department\_expenses(self, year: int = None, month: int = None) -> List[Dict[str, Any]]:**

**"""부서별 지출 통계"""**

**try:**

**base\_query = """**

**SELECT**

**department,**

**COUNT(\*) as request\_count,**

**SUM(amount) as total\_amount,**

**SUM(CASE WHEN status = 'paid' THEN amount ELSE 0 END) as paid\_amount**

**FROM expense\_requests**

**WHERE department IS NOT NULL AND department != ''**

**"""**

**params = []**

**if year:**

**base\_query += " AND EXTRACT(YEAR FROM request\_date) = %s"**

**params.append(year)**

**if month:**

**base\_query += " AND EXTRACT(MONTH FROM request\_date) = %s"**

**params.append(month)**

**base\_query += " GROUP BY department ORDER BY total\_amount DESC"**

**return self.db.execute\_query(base\_query, params)**

**except Exception as e:**

**print(f"Department expenses error: {e}")**

**return []**

**def export\_expense\_requests\_to\_csv(self, filters: Dict[str, Any] = None) -> io.StringIO:**

**"""지출 요청서를 CSV로 내보내기"""**

**try:**

**requests = self.get\_expense\_requests(filters)**

**# DataFrame 생성**

**df = pd.DataFrame(requests)**

**# 컬럼 순서 및 이름 정리**

**if not df.empty:**

**df = df[['request\_number', 'request\_date', 'department',**

**'expense\_type', 'description', 'amount', 'currency',**

**'payment\_method', 'status', 'requester\_name',**

**'approved\_by\_name', 'approved\_at', 'paid\_at']]**

**df.columns = ['요청번호', '요청일', '부서', '지출유형', '설명',**

**'금액', '통화', '결제방법', '상태', '요청자',**

**'승인자', '승인일', '결제일']**

**# CSV 문자열로 변환**

**output = io.StringIO()**

**df.to\_csv(output, index=False, encoding='utf-8-sig')**

**output.seek(0)**

**return output**

**except Exception as e:**

**print(f"Export expense requests error: {e}")**

**return None**

**class ExpenseValidator:**

**"""지출 요청서 유효성 검증 클래스"""**

**@staticmethod**

**def validate\_expense\_data(data: Dict[str, Any]) -> List[str]:**

**"""지출 요청서 데이터 유효성 검증**

**Returns:**

**오류 메시지 목록 (빈 리스트면 유효함)**

**"""**

**errors = []**

**# 필수 필드 검증**

**required\_fields = ['expense\_type', 'description', 'amount', 'request\_date']**

**for field in required\_fields:**

**if field not in data or not data[field]:**

**errors.append(f"{field}는 필수 입력 항목입니다.")**

**# 금액 검증**

**if 'amount' in data and data['amount']:**

**if data['amount'] <= 0:**

**errors.append("금액은 0보다 큰 값이어야 합니다.")**

**if data['amount'] > 1000000: # 100만 달러 제한**

**errors.append("금액이 너무 큽니다. 관리자에게 문의하세요.")**

**# 날짜 검증**

**if 'request\_date' in data and data['request\_date']:**

**if data['request\_date'] > datetime.now().date():**

**errors.append("요청일은 오늘 이후 날짜일 수 없습니다.")**

**# 설명 길이 검증**

**if 'description' in data and data['description']:**

**if len(data['description']) < 5:**

**errors.append("설명은 최소 5자 이상 입력해야 합니다.")**

**if len(data['description']) > 500:**

**errors.append("설명은 500자를 초과할 수 없습니다.")**

**return errors**

**3.6 공통 유틸리티 및 파일 처리**

**# 공통 유틸리티 및 파일 처리 함수 설계**

**# shared/utils.py**

**import re**

**import json**

**from typing import Any, Dict, List, Optional, Union**

**from datetime import datetime, date**

**from decimal import Decimal**

**import locale**

**class FormatUtils:**

**"""포맷팅 유틸리티 클래스"""**

**@staticmethod**

**def format\_currency(amount: Union[int, float, Decimal], currency: str = 'USD',**

**locale\_code: str = 'ko\_KR') -> str:**

**"""통화 포맷팅 (천단위 콤마)**

**Args:**

**amount: 금액**

**currency: 통화 코드**

**locale\_code: 로케일 코드**

**Returns:**

**포맷된 통화 문자열**

**"""**

**try:**

**# 천단위 콤마 추가**

**formatted\_amount = f"{amount:,.2f}"**

**# 통화별 기호 매핑**

**currency\_symbols = {**

**'USD': '$',**

**'KRW': '₩',**

**'VND': '₫',**

**'CNY': '¥',**

**'THB': '฿'**

**}**

**symbol = currency\_symbols.get(currency, currency)**

**# 소수점 제거 (원화, 동화의 경우)**

**if currency in ['KRW', 'VND']:**

**formatted\_amount = f"{int(amount):,}"**

**return f"{symbol} {formatted\_amount}"**

**except Exception as e:**

**return str(amount)**

**@staticmethod**

**def format\_date(date\_obj: Union[date, datetime, str], format\_str: str = '%Y-%m-%d') -> str:**

**"""날짜 포맷팅"""**

**try:**

**if isinstance(date\_obj, str):**

**# 문자열인 경우 datetime으로 변환**

**date\_obj = datetime.strptime(date\_obj, '%Y-%m-%d').date()**

**elif isinstance(date\_obj, datetime):**

**date\_obj = date\_obj.date()**

**return date\_obj.strftime(format\_str)**

**except Exception as e:**

**return str(date\_obj)**

**@staticmethod**

**def format\_phone(phone: str) -> str:**

**"""전화번호 포맷팅"""**

**try:**

**# 숫자만 추출**

**digits = re.sub(r'\D', '', phone)**

**# 한국 전화번호 포맷**

**if len(digits) == 11 and digits.startswith('010'):**

**return f"{digits[:3]}-{digits[3:7]}-{digits[7:]}"**

**elif len(digits) == 10:**

**return f"{digits[:3]}-{digits[3:6]}-{digits[6:]}"**

**return phone**

**except Exception as e:**

**return phone**

**@staticmethod**

**def truncate\_text(text: str, max\_length: int = 50, suffix: str = '...') -> str:**

**"""텍스트 자르기"""**

**if len(text) <= max\_length:**

**return text**

**return text[:max\_length - len(suffix)] + suffix**

**class ValidationUtils:**

**"""유효성 검증 유틸리티 클래스"""**

**@staticmethod**

**def is\_valid\_email(email: str) -> bool:**

**"""이메일 유효성 검증"""**

**pattern = r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'**

**return re.match(pattern, email) is not None**

**@staticmethod**

**def is\_valid\_phone(phone: str) -> bool:**

**"""전화번호 유효성 검증"""**

**# 숫자, 하이픈, 괄호, 공백, + 기호만 허용**

**pattern = r'^[\d\-\(\)\s\+]+$'**

**return re.match(pattern, phone) is not None**

**@staticmethod**

**def is\_valid\_business\_number(business\_number: str) -> bool:**

**"""사업자등록번호 유효성 검증 (한국 기준)"""**

**# 숫자만 추출**

**digits = re.sub(r'\D', '', business\_number)**

**if len(digits) != 10:**

**return False**

**# 체크섬 검증**

**check\_digits = [1, 3, 7, 1, 3, 7, 1, 3, 5]**

**total = sum(int(digits[i]) \* check\_digits[i] for i in range(9))**

**check\_digit = (10 - (total % 10)) % 10**

**return int(digits[9]) == check\_digit**

**@staticmethod**

**def sanitize\_filename(filename: str) -> str:**

**"""파일명 안전하게 만들기"""**

**# 위험한 문자 제거**

**sanitized = re.sub(r'[<>:"/\\|?\*]', '\_', filename)**

**# 연속된 공백을 하나로**

**sanitized = re.sub(r'\s+', ' ', sanitized)**

**# 앞뒤 공백 제거**

**return sanitized.strip()**

**class DateTimeUtils:**

**"""날짜/시간 유틸리티 클래스"""**

**@staticmethod**

**def get\_current\_date() -> date:**

**"""현재 날짜 반환"""**

**return datetime.now().date()**

**@staticmethod**

**def get\_current\_datetime() -> datetime:**

**"""현재 날짜시간 반환"""**

**return datetime.now()**

**@staticmethod**

**def get\_month\_range(year: int, month: int) -> tuple:**

**"""월의 시작일과 종료일 반환"""**

**from calendar import monthrange**

**start\_date = date(year, month, 1)**

**\_, last\_day = monthrange(year, month)**

**end\_date = date(year, month, last\_day)**

**return start\_date, end\_date**

**@staticmethod**

**def get\_quarter\_range(year: int, quarter: int) -> tuple:**

**"""분기의 시작일과 종료일 반환"""**

**quarter\_months = {**

**1: (1, 3),**

**2: (4, 6),**

**3: (7, 9),**

**4: (10, 12)**

**}**

**start\_month, end\_month = quarter\_months[quarter]**

**start\_date = date(year, start\_month, 1)**

**from calendar import monthrange**

**\_, last\_day = monthrange(year, end\_month)**

**end\_date = date(year, end\_month, last\_day)**

**return start\_date, end\_date**

**@staticmethod**

**def is\_weekend(date\_obj: date) -> bool:**

**"""주말 여부 확인"""**

**return date\_obj.weekday() >= 5 # 5: 토요일, 6: 일요일**

**# shared/file\_handler.py**

**import os**

**import csv**

**import json**

**import pandas as pd**

**from io import StringIO, BytesIO**

**from typing import List, Dict, Any, Optional**

**import zipfile**

**import tempfile**

**class FileHandler:**

**"""파일 처리 클래스"""**

**def \_\_init\_\_(self, upload\_dir: str = 'uploads', export\_dir: str = 'exports'):**

**self.upload\_dir = upload\_dir**

**self.export\_dir = export\_dir**

**self.\_ensure\_directories()**

**def \_ensure\_directories(self):**

**"""디렉토리 생성"""**

**os.makedirs(self.upload\_dir, exist\_ok=True)**

**os.makedirs(self.export\_dir, exist\_ok=True)**

**def save\_uploaded\_file(self, uploaded\_file, subfolder: str = '') -> str:**

**"""업로드된 파일 저장**

**Args:**

**uploaded\_file: Streamlit 업로드 파일 객체**

**subfolder: 하위 폴더명**

**Returns:**

**저장된 파일 경로**

**"""**

**try:**

**# 안전한 파일명 생성**

**safe\_filename = ValidationUtils.sanitize\_filename(uploaded\_file.name)**

**timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')**

**filename = f"{timestamp}\_{safe\_filename}"**

**# 저장 경로**

**save\_dir = os.path.join(self.upload\_dir, subfolder) if subfolder else self.upload\_dir**

**os.makedirs(save\_dir, exist\_ok=True)**

**file\_path = os.path.join(save\_dir, filename)**

**# 파일 저장**

**with open(file\_path, 'wb') as f:**

**f.write(uploaded\_file.getbuffer())**

**return file\_path**

**except Exception as e:**

**print(f"File save error: {e}")**

**return None**

**def read\_csv\_file(self, file\_path: str, encoding: str = 'utf-8') -> Optional[pd.DataFrame]:**

**"""CSV 파일 읽기"""**

**try:**

**# 여러 인코딩 시도**

**encodings = [encoding, 'utf-8-sig', 'cp949', 'euc-kr']**

**for enc in encodings:**

**try:**

**df = pd.read\_csv(file\_path, encoding=enc)**

**return df**

**except UnicodeDecodeError:**

**continue**

**# 모든 인코딩 실패시 에러**

**raise ValueError("Unable to read CSV file with any encoding")**

**except Exception as e:**

**print(f"CSV read error: {e}")**

**return None**

**def write\_csv\_file(self, data: List[Dict[str, Any]], filename: str,**

**subfolder: str = '') -> str:**

**"""CSV 파일 쓰기"""**

**try:**

**# 저장 경로**

**save\_dir = os.path.join(self.export\_dir, subfolder) if subfolder else self.export\_dir**

**os.makedirs(save\_dir, exist\_ok=True)**

**timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')**

**safe\_filename = ValidationUtils.sanitize\_filename(filename)**

**file\_path = os.path.join(save\_dir, f"{timestamp}\_{safe\_filename}")**

**# DataFrame으로 변환 후 CSV 저장**

**df = pd.DataFrame(data)**

**df.to\_csv(file\_path, index=False, encoding='utf-8-sig')**

**return file\_path**

**except Exception as e:**

**print(f"CSV write error: {e}")**

**return None**

**def create\_zip\_file(self, files: List[str], zip\_filename: str) -> str:**

**"""여러 파일을 ZIP으로 압축"""**

**try:**

**zip\_path = os.path.join(self.export\_dir, zip\_filename)**

**with zipfile.ZipFile(zip\_path, 'w', zipfile.ZIP\_DEFLATED) as zipf:**

**for file\_path in files:**

**if os.path.exists(file\_path):**

**# 파일명만 추출하여 ZIP 내부 경로로 사용**

**arcname = os.path.basename(file\_path)**

**zipf.write(file\_path, arcname)**

**return zip\_path**

**except Exception as e:**

**print(f"ZIP creation error: {e}")**

**return None**

**def cleanup\_old\_files(self, days: int = 7):**

**"""오래된 파일 정리"""**

**try:**

**current\_time = datetime.now()**

**for directory in [self.upload\_dir, self.export\_dir]:**

**if not os.path.exists(directory):**

**continue**

**for root, dirs, files in os.walk(directory):**

**for file in files:**

**file\_path = os.path.join(root, file)**

**# 파일 수정 시간 확인**

**file\_time = datetime.fromtimestamp(os.path.getmtime(file\_path))**

**# 지정된 일수보다 오래된 파일 삭제**

**if (current\_time - file\_time).days > days:**

**os.remove(file\_path)**

**print(f"Deleted old file: {file\_path}")**

**except Exception as e:**

**print(f"Cleanup error: {e}")**

**def get\_file\_info(self, file\_path: str) -> Dict[str, Any]:**

**"""파일 정보 조회"""**

**try:**

**if not os.path.exists(file\_path):**

**return None**

**stat = os.stat(file\_path)**

**return {**

**'filename': os.path.basename(file\_path),**

**'size': stat.st\_size,**

**'size\_mb': round(stat.st\_size / (1024 \* 1024), 2),**

**'created\_time': datetime.fromtimestamp(stat.st\_ctime),**

**'modified\_time': datetime.fromtimestamp(stat.st\_mtime),**

**'extension': os.path.splitext(file\_path)[1].lower()**

**}**

**except Exception as e:**

**print(f"File info error: {e}")**

**return None**

**class ConfigManager:**

**"""설정 관리 클래스"""**

**def \_\_init\_\_(self, config\_file: str = 'config.json'):**

**self.config\_file = config\_file**

**self.\_config = self.\_load\_config()**

**def \_load\_config(self) -> Dict[str, Any]:**

**"""설정 파일 로드"""**

**try:**

**if os.path.exists(self.config\_file):**

**with open(self.config\_file, 'r', encoding='utf-8') as f:**

**return json.load(f)**

**return {}**

**except Exception as e:**

**print(f"Config load error: {e}")**

**return {}**

**def \_save\_config(self):**

**"""설정 파일 저장"""**

**try:**

**with open(self.config\_file, 'w', encoding='utf-8') as f:**

**json.dump(self.\_config, f, indent=2, ensure\_ascii=False)**

**except Exception as e:**

**print(f"Config save error: {e}")**

**def get(self, key: str, default: Any = None) -> Any:**

**"""설정값 조회"""**

**return self.\_config.get(key, default)**

**def set(self, key: str, value: Any):**

**"""설정값 설정"""**

**self.\_config[key] = value**

**self.\_save\_config()**

**def update(self, config\_dict: Dict[str, Any]):**

**"""설정 업데이트"""**

**self.\_config.update(config\_dict)**

**self.\_save\_config()**

**class LogManager:**

**"""로그 관리 클래스"""**

**def \_\_init\_\_(self, log\_file: str = 'app.log'):**

**self.log\_file = log\_file**

**def log(self, level: str, message: str, user\_id: int = None):**

**"""로그 기록"""**

**try:**

**timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')**

**user\_info = f"[User: {user\_id}]" if user\_id else "[System]"**

**log\_entry = f"{timestamp} [{level.upper()}] {user\_info} {message}\n"**

**with open(self.log\_file, 'a', encoding='utf-8') as f:**

**f.write(log\_entry)**

**except Exception as e:**

**print(f"Logging error: {e}")**

**def info(self, message: str, user\_id: int = None):**

**"""정보 로그"""**

**self.log('info', message, user\_id)**

**def warning(self, message: str, user\_id: int = None):**

**"""경고 로그"""**

**self.log('warning', message, user\_id)**

**def error(self, message: str, user\_id: int = None):**

**"""에러 로그"""**

**self.log('error', message, user\_id)**

**def get\_recent\_logs(self, lines: int = 100) -> List[str]:**

**"""최근 로그 조회"""**

**try:**

**if not os.path.exists(self.log\_file):**

**return []**

**with open(self.log\_file, 'r', encoding='utf-8') as f:**

**all\_lines = f.readlines()**

**return all\_lines[-lines:]**

**except Exception as e:**

**print(f"Get logs error: {e}")**

**return []**

**# 전역 인스턴스들**

**format\_utils = FormatUtils()**

**validation\_utils = ValidationUtils()**

**datetime\_utils = DateTimeUtils()**

**file\_handler = FileHandler()**

**config\_manager = ConfigManager()**

**log\_manager = LogManager()**

**4. UI/UX 설계**

**4.1 메뉴 구조**

**YMV 관리 시스템**

**├── 대시보드**

**├── 총무**

**│ ├── 구매 관리**

**│ │ ├── 구매 요청 목록**

**│ │ ├── 새 구매 요청**

**│ │ └── 구매 승인 관리**

**│ └── 지출 요청서**

**│ ├── 지출 요청 목록**

**│ ├── 새 지출 요청**

**│ └── 지출 승인 관리**

**├── 영업**

**│ ├── 고객 관리**

**│ │ ├── 고객 목록**

**│ │ └── 고객 등록**

**│ └── 견적서 관리**

**│ ├── 견적서 목록**

**│ ├── 견적서 작성**

**│ └── 견적서 통계**

**└── 시스템 관리**

**├── 제품 관리**

**│ ├── 제품 목록**

**│ ├── 제품 등록**

**│ └── CSV 업로드**

**├── 사용자 관리**

**└── 시스템 설정**

**4.2 화면 레이아웃 설계**

**# UI 컴포넌트 설계**

**# shared/components.py**

**import streamlit as st**

**import pandas as pd**

**from typing import List, Dict, Any, Optional, Callable**

**from datetime import datetime, date**

**class UIComponents:**

**"""공통 UI 컴포넌트 클래스"""**

**@staticmethod**

**def render\_page\_header(title: str, subtitle: str = None, show\_divider: bool = True):**

**"""페이지 헤더 렌더링"""**

**st.title(title)**

**if subtitle:**

**st.caption(subtitle)**

**if show\_divider:**

**st.divider()**

**@staticmethod**

**def render\_info\_box(title: str, value: str, delta: str = None,**

**delta\_color: str = "normal"):**

**"""정보 박스 렌더링"""**

**st.metric(**

**label=title,**

**value=value,**

**delta=delta,**

**delta\_color=delta\_color**

**)**

**@staticmethod**

**def render\_status\_badge(status: str) -> str:**

**"""상태 배지 렌더링"""**

**status\_colors = {**

**'pending': '',**

**'approved': '',**

**'rejected': '',**

**'draft': '⚪',**

**'sent': '',**

**'accepted': '✅',**

**'expired': '⚫',**

**'paid': '',**

**'ordered': '',**

**'received': '✅'**

**}**

**status\_texts = {**

**'pending': '대기중',**

**'approved': '승인됨',**

**'rejected': '거부됨',**

**'draft': '임시저장',**

**'sent': '발송됨',**

**'accepted': '수락됨',**

**'expired': '만료됨',**

**'paid': '결제완료',**

**'ordered': '주문됨',**

**'received': '입고됨'**

**}**

**icon = status\_colors.get(status, '⚪')**

**text = status\_texts.get(status, status)**

**return f"{icon} {text}"**

**@staticmethod**

**def render\_data\_table(data: List[Dict[str, Any]],**

**columns: List[Dict[str, str]] = None,**

**actions: List[Dict[str, Any]] = None,**

**key\_column: str = 'id',**

**page\_size: int = 20) -> Optional[Any]:**

**"""데이터 테이블 렌더링**

**Args:**

**data: 테이블 데이터**

**columns: 컬럼 설정 [{'key': 'field\_name', 'label': '컬럼명', 'type': 'text|currency|date|status'}]**

**actions: 액션 버튼 [{'label': '수정', 'key': 'edit', 'color': 'primary'}]**

**key\_column: 키 컬럼명**

**page\_size: 페이지 크기**

**Returns:**

**선택된 행의 키 값 또는 액션 정보**

**"""**

**if not data:**

**st.info("표시할 데이터가 없습니다.")**

**return None**

**# DataFrame 생성**

**df = pd.DataFrame(data)**

**# 페이지네이션**

**total\_rows = len(df)**

**total\_pages = (total\_rows - 1) // page\_size + 1**

**if total\_pages > 1:**

**col1, col2, col3 = st.columns([1, 2, 1])**

**with col2:**

**page = st.selectbox("페이지", range(1, total\_pages + 1)) - 1**

**else:**

**page = 0**

**start\_idx = page \* page\_size**

**end\_idx = min(start\_idx + page\_size, total\_rows)**

**page\_df = df.iloc[start\_idx:end\_idx]**

**# 컬럼 설정이 있으면 적용**

**if columns:**

**display\_df = page\_df.copy()**

**for col\_config in columns:**

**field\_key = col\_config['key']**

**if field\_key in display\_df.columns:**

**col\_type = col\_config.get('type', 'text')**

**if col\_type == 'currency':**

**display\_df[field\_key] = display\_df[field\_key].apply(**

**lambda x: f"${x:,.2f}" if pd.notnull(x) else ""**

**)**

**elif col\_type == 'date':**

**display\_df[field\_key] = pd.to\_datetime(display\_df[field\_key]).dt.strftime('%Y-%m-%d')**

**elif col\_type == 'status':**

**display\_df[field\_key] = display\_df[field\_key].apply(**

**lambda x: UIComponents.render\_status\_badge(x) if pd.notnull(x) else ""**

**)**

**# 컬럼명 변경**

**column\_mapping = {col['key']: col['label'] for col in columns if col['key'] in display\_df.columns}**

**display\_df = display\_df.rename(columns=column\_mapping)**

**else:**

**display\_df = page\_df**

**# 테이블 표시**

**selected\_rows = st.dataframe(**

**display\_df,**

**use\_container\_width=True,**

**hide\_index=True,**

**selection\_mode="single-row" if actions else None**

**)**

**# 액션 버튼 처리**

**if actions and selected\_rows and len(selected\_rows.selection.rows) > 0:**

**selected\_idx = selected\_rows.selection.rows[0]**

**selected\_row = page\_df.iloc[selected\_idx]**

**st.write("\*\*선택된 항목에 대한 작업:\*\*")**

**action\_cols = st.columns(len(actions))**

**for i, action in enumerate(actions):**

**with action\_cols[i]:**

**if st.button(**

**action['label'],**

**key=f"action\_{action['key']}\_{selected\_row[key\_column]}",**

**type=action.get('color', 'secondary')**

**):**

**return {**

**'action': action['key'],**

**'row\_data': selected\_row.to\_dict(),**

**'key\_value': selected\_row[key\_column]**

**}**

**return None**

**@staticmethod**

**def render\_search\_filters(filters\_config: List[Dict[str, Any]]) -> Dict[str, Any]:**

**"""검색 필터 렌더링**

**Args:**

**filters\_config: 필터 설정**

**[{'key': 'status', 'label': '상태', 'type': 'selectbox', 'options': [...]}]**

**Returns:**

**필터 값들의 딕셔너리**

**"""**

**st.subheader("🔍 검색 및 필터")**

**filter\_values = {}**

**# 필터 그룹을 2개씩 나누어 배치**

**for i in range(0, len(filters\_config), 2):**

**cols = st.columns(2)**

**for j, col in enumerate(cols):**

**if i + j < len(filters\_config):**

**filter\_config = filters\_config[i + j]**

**with col:**

**filter\_key = filter\_config['key']**

**filter\_type = filter\_config['type']**

**filter\_label = filter\_config['label']**

**if filter\_type == 'text':**

**filter\_values[filter\_key] = st.text\_input(**

**filter\_label,**

**key=f"filter\_{filter\_key}"**

**)**

**elif filter\_type == 'selectbox':**

**options = ['전체'] + filter\_config.get('options', [])**

**selected = st.selectbox(**

**filter\_label,**

**options,**

**key=f"filter\_{filter\_key}"**

**)**

**filter\_values[filter\_key] = None if selected == '전체' else selected**

**elif filter\_type == 'date':**

**filter\_values[filter\_key] = st.date\_input(**

**filter\_label,**

**key=f"filter\_{filter\_key}"**

**)**

**elif filter\_type == 'date\_range':**

**date\_col1, date\_col2 = st.columns(2)**

**with date\_col1:**

**filter\_values[f"{filter\_key}\_from"] = st.date\_input(**

**f"{filter\_label} 시작",**

**key=f"filter\_{filter\_key}\_from"**

**)**

**with date\_col2:**

**filter\_values[f"{filter\_key}\_to"] = st.date\_input(**

**f"{filter\_label} 종료",**

**key=f"filter\_{filter\_key}\_to"**

**)**

**# 검색 버튼**

**search\_col, reset\_col = st.columns([1, 1])**

**with search\_col:**

**search\_clicked = st.button("🔍 검색", type="primary")**

**with reset\_col:**

**reset\_clicked = st.button("🔄 초기화")**

**if reset\_clicked:**

**# 세션 스테이트의 필터 관련 키들 삭제**

**for config in filters\_config:**

**key\_to\_delete = f"filter\_{config['key']}"**

**if key\_to\_delete in st.session\_state:**

**del st.session\_state[key\_to\_delete]**

**# date\_range의 경우 추가 키들도 삭제**

**if config['type'] == 'date\_range':**

**for suffix in ['\_from', '\_to']:**

**key\_to\_delete = f"filter\_{config['key']}{suffix}"**

**if key\_to\_delete in st.session\_state:**

**del st.session\_state[key\_to\_delete]**

**st.rerun()**

**# None 값 제거**

**return {k: v for k, v in filter\_values.items() if v is not None and v != ''}**

**@staticmethod**

**def render\_form\_field(field\_config: Dict[str, Any], value: Any = None) -> Any:**

**"""폼 필드 렌더링**

**Args:**

**field\_config: 필드 설정**

**{**

**'key': 'field\_key',**

**'label': '필드명',**

**'type': 'text|number|date|selectbox|textarea|file',**

**'required': True/False,**

**'options': [...], # selectbox용**

**'help': '도움말'**

**}**

**value: 기본값**

**Returns:**

**입력된 값**

**"""**

**field\_key = field\_config['key']**

**field\_type = field\_config['type']**

**field\_label = field\_config['label']**

**required = field\_config.get('required', False)**

**help\_text = field\_config.get('help', None)**

**# 필수 필드 표시**

**if required:**

**field\_label += " \*"**

**if field\_type == 'text':**

**return st.text\_input(**

**field\_label,**

**value=value or "",**

**key=f"form\_{field\_key}",**

**help=help\_text**

**)**

**elif field\_type == 'number':**

**return st.number\_input(**

**field\_label,**

**value=float(value) if value else 0.0,**

**key=f"form\_{field\_key}",**

**help=help\_text**

**)**

**elif field\_type == 'date':**

**default\_value = value if value else date.today()**

**return st.date\_input(**

**field\_label,**

**value=default\_value,**

**key=f"form\_{field\_key}",**

**help=help\_text**

**)**

**elif field\_type == 'selectbox':**

**options = field\_config.get('options', [])**

**index = 0**

**if value and value in options:**

**index = options.index(value)**

**return st.selectbox(**

**field\_label,**

**options,**

**index=index,**

**key=f"form\_{field\_key}",**

**help=help\_text**

**)**

**elif field\_type == 'textarea':**

**return st.text\_area(**

**field\_label,**

**value=value or "",**

**key=f"form\_{field\_key}",**

**help=help\_text**

**)**

**elif field\_type == 'file':**

**return st.file\_uploader(**

**field\_label,**

**key=f"form\_{field\_key}",**

**help=help\_text**

**)**

**else:**

**st.error(f"지원하지 않는 필드 타입: {field\_type}")**

**return None**

**@staticmethod**

**def render\_form(form\_config: List[Dict[str, Any]],**

**submit\_label: str = "저장",**

**cancel\_label: str = "취소",**

**initial\_values: Dict[str, Any] = None) -> Dict[str, Any]:**

**"""폼 렌더링**

**Args:**

**form\_config: 폼 필드 설정 리스트**

**submit\_label: 제출 버튼 레이블**

**cancel\_label: 취소 버튼 레이블**

**initial\_values: 초기값 딕셔너리**

**Returns:**

**폼 데이터 또는 액션 정보**

**"""**

**form\_data = {}**

**# 폼 필드들 렌더링**

**for field\_config in form\_config:**

**field\_key = field\_config['key']**

**initial\_value = initial\_values.get(field\_key) if initial\_values else None**

**form\_data[field\_key] = UIComponents.render\_form\_field(field\_config, initial\_value)**

**# 버튼들**

**st.divider()**

**col1, col2 = st.columns([1, 1])**

**with col1:**

**submit\_clicked = st.button(submit\_label, type="primary")**

**with col2:**

**cancel\_clicked = st.button(cancel\_label)**

**if submit\_clicked:**

**# 필수 필드 검증**

**errors = []**

**for field\_config in form\_config:**

**if field\_config.get('required', False):**

**field\_key = field\_config['key']**

**if not form\_data[field\_key]:**

**errors.append(f"{field\_config['label']}는 필수 입력 항목입니다.")**

**if errors:**

**for error in errors:**

**st.error(error)**

**return None**

**return {'action': 'submit', 'data': form\_data}**

**if cancel\_clicked:**

**return {'action': 'cancel'}**

**return None**

**@staticmethod**

**def render\_confirmation\_dialog(message: str, confirm\_label: str = "확인",**

**cancel\_label: str = "취소") -> Optional[bool]:**

**"""확인 다이얼로그 렌더링"""**

**st.warning(message)**

**col1, col2 = st.columns(2)**

**with col1:**

**if st.button(confirm\_label, type="primary"):**

**return True**

**with col2:**

**if st.button(cancel\_label):**

**return False**

**return None**

**@staticmethod**

**def render\_statistics\_cards(stats: List[Dict[str, Any]]):**

**"""통계 카드들 렌더링**

**Args:**

**stats: 통계 데이터**

**[{'title': '제목', 'value': '값', 'delta': '변화량', 'color': '색상'}]**

**"""**

**if not stats:**

**return**

**# 카드 수에 따라 컬럼 분할**

**num\_cards = len(stats)**

**cols = st.columns(num\_cards)**

**for i, stat in enumerate(stats):**

**with cols[i]:**

**UIComponents.render\_info\_box(**

**title=stat['title'],**

**value=stat['value'],**

**delta=stat.get('delta'),**

**delta\_color=stat.get('color', 'normal')**

**)**

**@staticmethod**

**def render\_export\_section(export\_functions: List[Dict[str, Any]]):**

**"""내보내기 섹션 렌더링**

**Args:**

**export\_functions: 내보내기 함수들**

**[{'label': 'CSV 내보내기', 'key': 'csv', 'function': export\_csv\_func}]**

**"""**

**st.subheader("📤 데이터 내보내기")**

**cols = st.columns(len(export\_functions))**

**for i, export\_func in enumerate(export\_functions):**

**with cols[i]:**

**if st.button(export\_func['label'], key=f"export\_{export\_func['key']}"):**

**try:**

**# 내보내기 함수 실행**

**result = export\_func['function']()**

**if result:**

**st.success(f"{export\_func['label']} 완료!")**

**# 다운로드 링크 제공 로직 추가 가능**

**else:**

**st.error("내보내기 실패")**

**except Exception as e:**

**st.error(f"내보내기 중 오류가 발생했습니다: {str(e)}")**

**class NavigationManager:**

**"""네비게이션 관리 클래스"""**

**@staticmethod**

**def render\_sidebar\_menu(menu\_config: Dict[str, Any], user\_info: Dict[str, Any]) -> str:**

**"""사이드바 메뉴 렌더링**

**Args:**

**menu\_config: 메뉴 설정**

**user\_info: 사용자 정보**

**Returns:**

**선택된 페이지 키**

**"""**

**with st.sidebar:**

**# 사용자 정보**

**st.write(f"👤 \*\*{user\_info['full\_name']}\*\*")**

**st.write(f"🏢 {user\_info.get('department', '')}")**

**st.divider()**

**# 메뉴 항목들**

**selected\_page = None**

**for menu\_key, menu\_item in menu\_config.items():**

**if menu\_item.get('type') == 'page':**

**if st.button(menu\_item['label'], use\_container\_width=True):**

**selected\_page = menu\_key**

**elif menu\_item.get('type') == 'group':**

**with st.expander(menu\_item['label']):**

**for submenu\_key, submenu\_item in menu\_item.get('items', {}).items():**

**if st.button(submenu\_item['label'], key=submenu\_key):**

**selected\_page = submenu\_key**

**# 로그아웃 버튼**

**st.divider()**

**if st.button("🚪 로그아웃", use\_container\_width=True):**

**selected\_page = 'logout'**

**return selected\_page**

**@staticmethod**

**def get\_menu\_config(user\_role: str = 'user') -> Dict[str, Any]:**

**"""사용자 역할에 따른 메뉴 설정 반환"""**

**base\_menu = {**

**'dashboard': {**

**'type': 'page',**

**'label': '📊 대시보드',**

**'icon': ''**

**},**

**'general\_affairs': {**

**'type': 'group',**

**'label': '🏢 총무',**

**'items': {**

**'purchase\_list': {**

**'label': '📦 구매 관리',**

**'icon': ''**

**},**

**'expense\_list': {**

**'label': '💰 지출 요청서',**

**'icon': ''**

**}**

**}**

**},**

**'sales': {**

**'type': 'group',**

**'label': '💼 영업',**

**'items': {**

**'customer\_list': {**

**'label': '👥 고객 관리',**

**'icon': ''**

**},**

**'quotation\_list': {**

**'label': '📋 견적서 관리',**

**'icon': ''**

**}**

**}**

**}**

**}**

**# Master 사용자는 시스템 관리 메뉴 추가**

**if user\_role == 'master':**

**base\_menu['system'] = {**

**'type': 'group',**

**'label': '⚙️ 시스템 관리',**

**'items': {**

**'product\_list': {**

**'label': '📦 제품 관리',**

**'icon': ''**

**},**

**'user\_management': {**

**'label': '👤 사용자 관리',**

**'icon': ''**

**},**

**'system\_settings': {**

**'label': '⚙️ 시스템 설정',**

**'icon': '⚙️'**

**}**

**}**

**}**

**return base\_menu**

**4.3 PDF 생성 시스템**

**# PDF 생성 시스템 설계**

**# shared/pdf\_generator.py**

**from reportlab.lib.pagesizes import A4**

**from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle**

**from reportlab.lib.units import mm, inch**

**from reportlab.lib import colors**

**from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer, Image**

**from reportlab.pdfbase import pdfmetrics**

**from reportlab.pdfbase.ttfonts import TTFont**

**from reportlab.lib.enums import TA\_CENTER, TA\_LEFT, TA\_RIGHT**

**from typing import Dict, List, Any, Optional**

**from datetime import datetime**

**import os**

**import io**

**class PDFGenerator:**

**"""PDF 생성 클래스"""**

**def \_\_init\_\_(self):**

**self.setup\_fonts()**

**self.page\_width, self.page\_height = A4**

**self.margin = 20 \* mm**

**def setup\_fonts(self):**

**"""한글 폰트 설정"""**

**try:**

**# 한글 폰트 등록 (나눔고딕)**

**font\_path = "fonts/NanumGothic.ttf"**

**if os.path.exists(font\_path):**

**pdfmetrics.registerFont(TTFont('NanumGothic', font\_path))**

**pdfmetrics.registerFont(TTFont('NanumGothic-Bold', 'fonts/NanumGothic-Bold.ttf'))**

**else:**

**# 기본 폰트 사용**

**print("한글 폰트를 찾을 수 없습니다. 기본 폰트를 사용합니다.")**

**except Exception as e:**

**print(f"폰트 설정 오류: {e}")**

**def get\_styles(self) -> Dict[str, ParagraphStyle]:**

**"""스타일 설정"""**

**styles = getSampleStyleSheet()**

**# 커스텀 스타일 정의**

**custom\_styles = {**

**'Title': ParagraphStyle(**

**'CustomTitle',**

**parent=styles['Title'],**

**fontName='NanumGothic-Bold',**

**fontSize=16,**

**alignment=TA\_CENTER,**

**spaceAfter=20**

**),**

**'Heading': ParagraphStyle(**

**'CustomHeading',**

**parent=styles['Heading2'],**

**fontName='NanumGothic-Bold',**

**fontSize=12,**

**alignment=TA\_LEFT,**

**spaceAfter=10**

**),**

**'Normal': ParagraphStyle(**

**'CustomNormal',**

**parent=styles['Normal'],**

**fontName='NanumGothic',**

**fontSize=9,**

**alignment=TA\_LEFT**

**),**

**'Center': ParagraphStyle(**

**'CustomCenter',**

**parent=styles['Normal'],**

**fontName='NanumGothic',**

**fontSize=9,**

**alignment=TA\_CENTER**

**),**

**'Right': ParagraphStyle(**

**'CustomRight',**

**parent=styles['Normal'],**

**fontName='NanumGothic',**

**fontSize=9,**

**alignment=TA\_RIGHT**

**)**

**}**

**return custom\_styles**

**def create\_quotation\_pdf(self, quotation\_data: Dict[str, Any],**

**company\_info: Dict[str, Any],**

**output\_path: str = None) -> bytes:**

**"""견적서 PDF 생성**

**Args:**

**quotation\_data: 견적서 데이터 (header + items)**

**company\_info: 회사 정보**

**output\_path: 출력 파일 경로 (None이면 바이트 반환)**

**Returns:**

**PDF 바이트 데이터**

**"""**

**buffer = io.BytesIO()**

**# PDF 문서 생성**

**doc = SimpleDocTemplate(**

**buffer,**

**pagesize=A4,**

**rightMargin=self.margin,**

**leftMargin=self.margin,**

**topMargin=self.margin,**

**bottomMargin=self.margin**

**)**

**# 스타일 가져오기**

**styles = self.get\_styles()**

**# 컨텐츠 리스트**

**content = []**

**header = quotation\_data['header']**

**items = quotation\_data['items']**

**# 1. 문서 제목**

**title = Paragraph("견 적 서", styles['Title'])**

**content.append(title)**

**content.append(Spacer(1, 20))**

**# 2. 회사 정보 및 견적서 정보 테이블**

**header\_data = [**

**['회사명:', company\_info.get('company\_name\_ko', ''), '견적번호:', header['quotation\_number']],**

**['주소:', company\_info.get('address\_ko', ''), '견적일:', header['quotation\_date'].strftime('%Y-%m-%d')],**

**['전화:', company\_info.get('phone', ''), '유효기간:', header['valid\_until'].strftime('%Y-%m-%d')],**

**['이메일:', company\_info.get('email', ''), '통화:', header['currency']]**

**]**

**header\_table = Table(header\_data, colWidths=[25\*mm, 70\*mm, 25\*mm, 50\*mm])**

**header\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 9),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),**

**('BACKGROUND', (0, 0), (0, -1), colors.lightgrey),**

**('BACKGROUND', (2, 0), (2, -1), colors.lightgrey),**

**]))**

**content.append(header\_table)**

**content.append(Spacer(1, 20))**

**# 3. 고객 정보**

**customer\_title = Paragraph("고객 정보", styles['Heading'])**

**content.append(customer\_title)**

**customer\_data = [**

**['회사명:', header['company\_name']],**

**['담당자:', header.get('contact\_person', '')],**

**['전화:', header.get('phone', '')],**

**['이메일:', header.get('email', '')],**

**['주소:', header.get('address', '')]**

**]**

**customer\_table = Table(customer\_data, colWidths=[30\*mm, 140\*mm])**

**customer\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 9),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),**

**('BACKGROUND', (0, 0), (0, -1), colors.lightgrey),**

**]))**

**content.append(customer\_table)**

**content.append(Spacer(1, 20))**

**# 4. 견적 상세 항목**

**items\_title = Paragraph("견적 내역", styles['Heading'])**

**content.append(items\_title)**

**# 테이블 헤더**

**items\_data = [**

**['No.', '제품코드', '제품명', '사양', '수량', '단위', '단가', '합계']**

**]**

**# 항목 데이터 추가**

**total\_amount = 0**

**for i, item in enumerate(items, 1):**

**item\_row = [**

**str(i),**

**item['product\_code'],**

**item['product\_name'],**

**item.get('specification', ''),**

**f"{item['quantity']:,.0f}",**

**item['unit'],**

**f"{item['unit\_price']:,.2f}",**

**f"{item['total\_price']:,.2f}"**

**]**

**items\_data.append(item\_row)**

**total\_amount += item['total\_price']**

**# 합계 행 추가**

**items\_data.append(['', '', '', '', '', '', '총 합계:', f"{total\_amount:,.2f}"])**

**items\_table = Table(**

**items\_data,**

**colWidths=[15\*mm, 25\*mm, 40\*mm, 30\*mm, 20\*mm, 15\*mm, 25\*mm, 25\*mm]**

**)**

**items\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 8),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),**

**('ALIGN', (0, 0), (-1, -1), 'CENTER'),**

**('ALIGN', (2, 1), (2, -2), 'LEFT'), # 제품명은 왼쪽 정렬**

**('ALIGN', (3, 1), (3, -2), 'LEFT'), # 사양은 왼쪽 정렬**

**('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey), # 헤더 배경**

**('BACKGROUND', (0, -1), (-1, -1), colors.lightgrey), # 합계 행 배경**

**('FONTNAME', (0, 0), (-1, 0), 'NanumGothic-Bold'), # 헤더 볼드**

**('FONTNAME', (0, -1), (-1, -1), 'NanumGothic-Bold'), # 합계 행 볼드**

**]))**

**content.append(items\_table)**

**content.append(Spacer(1, 20))**

**# 5. 조건 및 비고**

**if header.get('payment\_terms') or header.get('delivery\_terms') or header.get('notes'):**

**conditions\_title = Paragraph("조건 및 비고", styles['Heading'])**

**content.append(conditions\_title)**

**conditions\_data = []**

**if header.get('payment\_terms'):**

**conditions\_data.append(['결제조건:', header['payment\_terms']])**

**if header.get('delivery\_terms'):**

**conditions\_data.append(['납기조건:', header['delivery\_terms']])**

**if header.get('notes'):**

**conditions\_data.append(['비고:', header['notes']])**

**if conditions\_data:**

**conditions\_table = Table(conditions\_data, colWidths=[30\*mm, 140\*mm])**

**conditions\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 9),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'TOP'),**

**('BACKGROUND', (0, 0), (0, -1), colors.lightgrey),**

**]))**

**content.append(conditions\_table)**

**# PDF 생성**

**doc.build(content)**

**# 바이트 데이터 반환**

**buffer.seek(0)**

**pdf\_data = buffer.getvalue()**

**buffer.close()**

**# 파일로 저장 (옵션)**

**if output\_path:**

**with open(output\_path, 'wb') as f:**

**f.write(pdf\_data)**

**return pdf\_data**

**def create\_expense\_request\_pdf(self, expense\_data: Dict[str, Any],**

**company\_info: Dict[str, Any],**

**output\_path: str = None) -> bytes:**

**"""지출 요청서 PDF 생성"""**

**buffer = io.BytesIO()**

**doc = SimpleDocTemplate(**

**buffer,**

**pagesize=A4,**

**rightMargin=self.margin,**

**leftMargin=self.margin,**

**topMargin=self.margin,**

**bottomMargin=self.margin**

**)**

**styles = self.get\_styles()**

**content = []**

**# 1. 문서 제목**

**title = Paragraph("지 출 요 청 서", styles['Title'])**

**content.append(title)**

**content.append(Spacer(1, 20))**

**# 2. 기본 정보**

**basic\_info\_data = [**

**['요청번호:', expense\_data['request\_number'], '요청일:', expense\_data['request\_date'].strftime('%Y-%m-%d')],**

**['요청자:', expense\_data['requester\_name'], '부서:', expense\_data.get('department', '')],**

**['지출유형:', expense\_data['expense\_type'], '결제방법:', expense\_data.get('payment\_method', '')]**

**]**

**basic\_info\_table = Table(basic\_info\_data, colWidths=[25\*mm, 70\*mm, 25\*mm, 50\*mm])**

**basic\_info\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 9),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),**

**('BACKGROUND', (0, 0), (0, -1), colors.lightgrey),**

**('BACKGROUND', (2, 0), (2, -1), colors.lightgrey),**

**]))**

**content.append(basic\_info\_table)**

**content.append(Spacer(1, 20))**

**# 3. 지출 내역**

**expense\_details\_data = [**

**['지출 내역', expense\_data['description']],**

**['요청 금액', f"{expense\_data['currency']} {expense\_data['amount']:,.2f}"],**

**['비고', expense\_data.get('notes', '')]**

**]**

**expense\_details\_table = Table(expense\_details\_data, colWidths=[40\*mm, 130\*mm])**

**expense\_details\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 9),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'TOP'),**

**('BACKGROUND', (0, 0), (0, -1), colors.lightgrey),**

**]))**

**content.append(expense\_details\_table)**

**content.append(Spacer(1, 30))**

**# 4. 승인란**

**approval\_title = Paragraph("승인란", styles['Heading'])**

**content.append(approval\_title)**

**approval\_data = [**

**['요청자', '부서장', '총무', '대표이사'],**

**['', '', '', ''],**

**['서명:', '서명:', '서명:', '서명:'],**

**['날짜:', '날짜:', '날짜:', '날짜:']**

**]**

**approval\_table = Table(approval\_data, colWidths=[42.5\*mm, 42.5\*mm, 42.5\*mm, 42.5\*mm])**

**approval\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 9),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),**

**('ALIGN', (0, 0), (-1, -1), 'CENTER'),**

**('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),**

**('ROWBACKGROUNDS', (0, 1), (-1, 1), [colors.white]),**

**('SPAN', (0, 1), (-1, 1)), # 빈 공간 (서명용)**

**]))**

**# 서명 공간 높이 설정**

**approval\_table.\_argH[1] = 30 # 두 번째 행 높이 30**

**content.append(approval\_table)**

**# PDF 생성**

**doc.build(content)**

**buffer.seek(0)**

**pdf\_data = buffer.getvalue()**

**buffer.close()**

**if output\_path:**

**with open(output\_path, 'wb') as f:**

**f.write(pdf\_data)**

**return pdf\_data**

**def create\_purchase\_request\_pdf(self, purchase\_data: Dict[str, Any],**

**company\_info: Dict[str, Any],**

**output\_path: str = None) -> bytes:**

**"""구매 요청서 PDF 생성"""**

**buffer = io.BytesIO()**

**doc = SimpleDocTemplate(**

**buffer,**

**pagesize=A4,**

**rightMargin=self.margin,**

**leftMargin=self.margin,**

**topMargin=self.margin,**

**bottomMargin=self.margin**

**)**

**styles = self.get\_styles()**

**content = []**

**# 1. 문서 제목**

**title = Paragraph("구 매 요 청 서", styles['Title'])**

**content.append(title)**

**content.append(Spacer(1, 20))**

**# 2. 기본 정보**

**basic\_info\_data = [**

**['요청번호:', purchase\_data['item\_number'], '요청일:', purchase\_data['request\_date'].strftime('%Y-%m-%d')],**

**['요청자:', purchase\_data['requested\_by\_name'], '카테고리:', purchase\_data['category\_name']],**

**['공급업체:', purchase\_data.get('supplier\_name', ''), '필요일:', purchase\_data.get('needed\_date', '').strftime('%Y-%m-%d') if purchase\_data.get('needed\_date') else '']**

**]**

**basic\_info\_table = Table(basic\_info\_data, colWidths=[25\*mm, 70\*mm, 25\*mm, 50\*mm])**

**basic\_info\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 9),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),**

**('BACKGROUND', (0, 0), (0, -1), colors.lightgrey),**

**('BACKGROUND', (2, 0), (2, -1), colors.lightgrey),**

**]))**

**content.append(basic\_info\_table)**

**content.append(Spacer(1, 20))**

**# 3. 구매 상세**

**purchase\_details\_data = [**

**['품목명', purchase\_data['item\_name']],**

**['수량', f"{purchase\_data['quantity']:,.0f} {purchase\_data['unit']}"],**

**['단가', f"{purchase\_data['currency']} {purchase\_data['unit\_price']:,.2f}"],**

**['총 금액', f"{purchase\_data['currency']} {purchase\_data['total\_price']:,.2f}"],**

**['비고', purchase\_data.get('notes', '')]**

**]**

**purchase\_details\_table = Table(purchase\_details\_data, colWidths=[40\*mm, 130\*mm])**

**purchase\_details\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 9),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'TOP'),**

**('BACKGROUND', (0, 0), (0, -1), colors.lightgrey),**

**]))**

**content.append(purchase\_details\_table)**

**content.append(Spacer(1, 30))**

**# 4. 승인란 (구매용)**

**approval\_title = Paragraph("승인란", styles['Heading'])**

**content.append(approval\_title)**

**approval\_data = [**

**['요청자', '부서장', '구매담당', '승인자'],**

**['', '', '', ''],**

**['서명:', '서명:', '서명:', '서명:'],**

**['날짜:', '날짜:', '날짜:', '날짜:']**

**]**

**approval\_table = Table(approval\_data, colWidths=[42.5\*mm, 42.5\*mm, 42.5\*mm, 42.5\*mm])**

**approval\_table.setStyle(TableStyle([**

**('FONTNAME', (0, 0), (-1, -1), 'NanumGothic'),**

**('FONTSIZE', (0, 0), (-1, -1), 9),**

**('GRID', (0, 0), (-1, -1), 0.5, colors.black),**

**('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),**

**('ALIGN', (0, 0), (-1, -1), 'CENTER'),**

**('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),**

**]))**

**# 서명 공간 높이 설정**

**approval\_table.\_argH[1] = 30**

**content.append(approval\_table)**

**# PDF 생성**

**doc.build(content)**

**buffer.seek(0)**

**pdf\_data = buffer.getvalue()**

**buffer.close()**

**if output\_path:**

**with open(output\_path, 'wb') as f:**

**f.write(pdf\_data)**

**return pdf\_data**

**# PDF 생성 팩토리**

**class PDFFactory:**

**"""PDF 생성 팩토리 클래스"""**

**def \_\_init\_\_(self):**

**self.generator = PDFGenerator()**

**def create\_pdf(self, document\_type: str, data: Dict[str, Any],**

**company\_info: Dict[str, Any], output\_path: str = None) -> bytes:**

**"""문서 타입에 따른 PDF 생성**

**Args:**

**document\_type: 문서 타입 (quotation, expense\_request, purchase\_request)**

**data: 문서 데이터**

**company\_info: 회사 정보**

**output\_path: 출력 파일 경로**

**Returns:**

**PDF 바이트 데이터**

**"""**

**if document\_type == 'quotation':**

**return self.generator.create\_quotation\_pdf(data, company\_info, output\_path)**

**elif document\_type == 'expense\_request':**

**return self.generator.create\_expense\_request\_pdf(data, company\_info, output\_path)**

**elif document\_type == 'purchase\_request':**

**return self.generator.create\_purchase\_request\_pdf(data, company\_info, output\_path)**

**else:**

**raise ValueError(f"지원하지 않는 문서 타입: {document\_type}")**

**# 전역 PDF 팩토리 인스턴스**

**pdf\_factory = PDFFactory()**