AI 인테리어 디자인 시스템 플로우 보고서

1. 시스템 개요

1.1 목적

- 사용자가 업로드한 원룸 사진을 AI가 분석하여 선택한 스타일에 맞게 개선된 인테리어 이미지 생성
- 원본 사진의 장점은 유지하고 문제점만 개선하는 현실적인 인테리어 제안

1.2 핵심 기술 스택

- Backend Framework: FastAPI (Python 3.x)
- AI Model: Google Gemini 2.5 Flash Image
- Image Processing: PIL (Pillow)
- API Communication: google-generativeai, google-genai SDK
- Frontend: HTML/JavaScript (Jinja2 Templates)

1.3 지원 인테리어 스타일

1. 미니멀리스트 - 깔끔하고 단순한 디자인
2. 스칸디나비안 - 밝고 자연스러운 북유럽 스타일
3. 모던 - 현대적이고 세련된 디자인
4. 빈티지 - 레트로 감성의 따뜻한 공간
5. 인더스트리얼 - 도시적이고 거친 매력

---

2. 시스템 아키텍처

2.1 디렉토리 구조

```
backend/
├── app/
│   ├── main.py                 # FastAPI 애플리케이션 진입점
│   ├── routes/
│   │   └── design.py           # API 엔드포인트 정의
│   ├── services/
│   │   └── gemini_service.py   # Gemini API 통합 로직
│   └── models/
│       └── schemas.py          # Pydantic 데이터 모델
├── templates/
│   └── index.html              # 사용자 인터페이스
├── uploads/                    # 업로드된 이미지 및 생성된 이미지 저장
├── static/                     # 정적 파일
└── requirements.txt            # Python 의존성
```

2.2 시스템 컴포넌트

2.2.1 FastAPI Application (main.py)

- 역할: 웹 서버 초기화 및 라우팅 설정
- 기능:
  - CORS 설정 (Cross-Origin Resource Sharing)
  - 정적 파일 및 템플릿 설정
  - 업로드 디렉토리 자동 생성
  - Health Check 엔드포인트 제공

2.2.2 API Routes (design.py)

- 역할: RESTful API 엔드포인트 제공
- 주요 엔드포인트:
  - `GET /api/styles`: 사용 가능한 스타일 목록 조회
  - `POST /api/upload`: 이미지 파일 업로드
  - `POST /api/analyze`: 원룸 사진 분석 (텍스트 분석)
  - `POST /api/design`: 디자인 가이드 생성 (텍스트 가이드)
  - `POST /api/generate-image`: 스타일 적용된 이미지 생성
  - `GET /api/images/{filename}`: 생성된 이미지 조회

2.2.3 Gemini Service (gemini_service.py)

- 역할: Google Gemini API와의 통신 및 AI 로직 처리
- 주요 메서드:
  - `analyze_room()`: 원룸 사진 분석 (JSON 반환)
  - `generate_design_guide()`: 디자인 가이드 생성 (JSON 반환)
  - `generate_interior_image()`: 개선된 인테리어 이미지 생성 (이미지 + 분석 텍스트)

2.2.4 Data Models (schemas.py)

- 역할: 요청/응답 데이터 구조 정의
- 주요 모델:
  - `StyleOption`: 스타일 정보
  - `RoomAnalysis`: 방 분석 결과
  - `DesignGuide`: 디자인 가이드
  - `DesignRequest`: 디자인 요청
  - `DesignResponse`: 디자인 응답

---

3. 데이터 플로우

3.1 전체 프로세스

```
[사용자]
   |
   | 1. 이미지 업로드
   v
[FastAPI /api/upload]
```

```
    |
    | 2. 파일 저장 (UUID 기반 고유 파일명)
    v
[uploads/ 디렉토리]
    |
    | 3. 스타일 선택 + 이미지 생성 요청
    v
[FastAPI /api/generate-image]
    |
    | 4. Gemini Service 호출
    v
[GeminiService.generate_interior_image()]
    |
    | 5. Gemini 2.5 Flash Image API 호출
    |    (2-Step Workflow Prompt)
    v
[Google Gemini API]
    |
    | 6. 분석 + 이미지 생성
    |    - STEP 1: 원본 분석 (장점/개선점 파악)
    |    - STEP 2: 개선점에만 스타일 적용
    v
[Response: 분석 텍스트 + 이미지 데이터]
    |
    | 7. 이미지 디코딩 및 저장
    v
[uploads/generated_*.png]
    |
    | 8. 응답 반환 (파일명 + 분석 텍스트)
    v
[사용자 화면에 표시]
```

3.2 이미지 업로드 플로우 (POST /api/upload)

1. **파일 수신**: 클라이언트에서 multipart/form-data로 이미지 전송
2. **검증**: 파일 확장자 검사 (.jpg, .jpeg, .png, .webp만 허용)
3. **저장**: UUID 기반 고유 파일명 생성 후 uploads/ 디렉토리에 저장
4. **응답**: 파일명 반환 (클라이언트는 이 파일명으로 이후 요청)

3.3 이미지 생성 플로우 (POST /api/generate-image)

3.3.1 요청 처리

```
{
  "image_filename": "uuid.jpg",
  "style_id": "minimalist"
}
```

3.3.2 Gemini API 호출 구조

```
# 원본 이미지 로드
original_image = Image.open(image_path)

# 2-Step Workflow Prompt 생성
prompt = """
STEP 1: ANALYZE THE CURRENT ROOM
- 잘되어 있는 점 (유지할 부분)
- 개선이 필요한 점

STEP 2: APPLY IMPROVEMENTS
- 원본의 좋은 점은 유지
- 문제점만 선택한 스타일로 개선
"""

# API 호출
response = client.models.generate_content(
    model="gemini-2.5-flash-image",
    contents=[prompt, original_image]
)
```

3.3.3 응답 처리

1. **응답 구조 파싱**: `response.candidates[0].content.parts` 순회
2. **텍스트 추출**: 분석 내용 수집 (한국어)
3. **이미지 추출**:
    - `inline_data.data`에서 base64 인코딩된 이미지 데이터 추출
    - base64 디코딩 후 PIL Image로 변환
    - PNG 형식으로 저장
4. **결과 반환**:

```
{
  "success": true,
  "generated_image": "generated_uuid.png",
  "original_image": "original_uuid.jpg",
  "style": "미니멀리스트",
  "analysis": "현재 방 분석:\n- 잘되어 있는 점: ...\n개선 내용: ..."
}
```

---

4. AI 프롬프트 엔지니어링 전략

4.1 핵심 원칙

기존 방식의 문제점을 해결하기 위해 **분석 우선(Analysis-First) 접근법** 채택

4.1.1 기존 방식 (문제점)

- 원본 사진 무시
- 스타일 가이드의 프리셋 색상/재료 무조건 적용
- 예: 미니멀리스트 = 무조건 흰색 벽, 스칸디나비안 = 무조건 원목

4.1.2 개선된 방식 (현재)

```
1단계: 원본 분석
   - 현재 상태의 장점 식별 → 유지
   - 개선이 필요한 부분 식별 → 수정 대상

2단계: 선택적 개선
   - 문제점에만 선택한 스타일 적용
   - 좋은 점은 그대로 유지
```

4.2 프롬프트 구조

4.2.1 분석 단계 (STEP 1)

```
Look at THIS specific room and identify:
- What is GOOD and should be KEPT
- What NEEDS IMPROVEMENT
- Current colors, furniture, layout, lighting
- Problems: clutter, poor lighting, bad furniture placement, etc.
```

4.2.2 개선 단계 (STEP 2)

```
IMPORTANT RULES:
✓ KEEP what's already good in the original room
✓ IMPROVE only what needs fixing
✓ Apply style to improvements, not blindly to everything
✓ Respect the original room's character

DON'T:
✗ Change colors just because style guide says so
✗ Replace furniture that already fits the style
✗ Add unnecessary items
```

4.2.3 출력 형식 (한국어)

```
1. 현재 방 분석:
   - 잘되어 있는 점 (유지할 부분)
   - 개선이 필요한 점

2. [스타일] 개선 내용:
```

```
    - 구체적으로 무엇을 어떻게 바꿨는지
    - 추가한 가구/소품 (있다면)
    - 왜 이렇게 바꿨는지

3. GENERATE THE IMPROVED IMAGE
```

4.3 스타일 적용 전략

4.3.1 스타일 원칙 (참고용, 강제 아님)

```
style_principles = {
    "미니멀리스트": "심플함, 필수 요소만, 여백 강조, 깔끔한 라인",
    "스칸디나비안": "밝은 원목, 자연스러움, 아늑함, 따뜻한 조명, 식물",
    "모던": "현대적, 세련됨, 기하학적, 중성 색상, 금속 포인트",
    "빈티지": "앤티크, 따뜻함, 복고풍, 장식적 디테일, 낭만적",
    "인더스트리얼": "원자재 노출, 거친 질감, 금속/콘크리트, 산업적 느낌"
}
```

4.3.2 가구 추가 전략 (BM 연계)

- 기능적으로 부족한 부분에만 가구 추가
- 추가된 가구는 분석 텍스트에 명시 (향후 제품 태깅/판매 활용)
- 예: "수납공간 부족으로 미니멀 선반 추가"

---

5. 에러 처리 및 안정성

5.1 파일 처리 안정성

5.1.1 절대 경로 사용

```
BASE_DIR = Path(__file__).resolve().parent.parent
UPLOAD_DIR = BASE_DIR / "uploads"
```

- 상대 경로 문제 해결
- FastAPI 서빙 위치와 무관하게 동작

5.1.2 디렉토리 자동 생성

```
UPLOAD_DIR.mkdir(exist_ok=True)
```

5.2 API 응답 처리

5.2.1 다중 응답 구조 대응

```
if hasattr(response, 'parts'):
    parts = response.parts
elif hasattr(response, 'candidates'):
    parts = response.candidates[0].content.parts
```

5.2.2 이미지 데이터 추출

```
# base64 디코딩 시도
try:
    image_bytes = base64.b64decode(image_data)
    image = Image.open(BytesIO(image_bytes))
except:
    # 디코딩 없이 직접 시도
    image = Image.open(BytesIO(image_data))
```

5.3 에러 핸들링

5.3.1 파일 검증

- 업로드 파일 확장자 검증
- 파일 존재 여부 확인 (404 반환)

5.3.2 API 호출 실패 처리

```
if not image_found:
    raise Exception("Gemini가 이미지를 생성하지 못했습니다.")
```

5.3.3 JSON 파싱 실패 대응

- 마크다운 코드 블록 자동 제거
- 파싱 실패 시 기본 구조 반환

---

6. 주요 기능 상세

6.1 원룸 분석 기능 (analyze_room)

입력

- 원룸 이미지 파일 경로

처리

- Gemini 2.5 Flash Image 모델에 분석 요청

- JSON 형식으로 구조화된 분석 결과 요청

출력

```
{
  "room_type": "원룸/투룸 등",
  "size_estimate": "평수 추정",
  "current_layout": "현재 레이아웃 설명",
  "issues": ["문제점 1", "문제점 2"],
  "strengths": ["장점 1", "장점 2"]
}
```

분석 항목

- 방문 옆 침대 위치 (동선 문제)
- 창문 위치와 채광
- 공간 활용도
- 수납 공간
- 가구 배치 효율성

6.2 디자인 가이드 생성 (generate_design_guide)

입력

- 원룸 이미지
- 분석 결과
- 선택한 스타일

처리

- 분석 결과 기반 개선 제안
- 스타일에 맞는 구체적 가이드 생성

출력

```
{
  "recommendations": ["추천사항 1", "추천사항 2"],
  "layout_suggestions": "레이아웃 제안 상세 설명",
  "color_scheme": "색상 배치 제안",
  "furniture_suggestions": ["가구 제안 1", "가구 제안 2"]
}
```

6.3 이미지 생성 (generate_interior_image)

입력

- 원본 이미지
- 스타일 이름
- 스타일 설명

처리 단계

1. 원본 이미지 로드 (PIL)
2. 2-Step Workflow 프롬프트 생성
3. Gemini 2.5 Flash Image API 호출
4. 응답에서 텍스트와 이미지 분리
5. 이미지 디코딩 및 저장 (PNG)
6. 결과 반환

출력

```
{
  'filename': 'generated_uuid.png',
  'analysis': '현재 방 분석:\n- 잘되어 있는 점...\n개선 내용...'
}
```

---

7. 성능 및 확장성

7.1 현재 성능 특성

7.1.1 처리 시간

- 이미지 업로드: 즉시 (로컬 저장)
- 분석: 3-5초 (Gemini API 호출)
- 이미지 생성: 10-15초 (Gemini 이미지 생성)

7.1.2 제약사항

- 동기 처리: 한 번에 하나의 요청만 처리
- 메모리: 이미지를 메모리에 로드하여 처리
- API Rate Limit: Google Gemini API 제한에 종속

7.2 확장 가능성

7.2.1 수평 확장

- FastAPI는 ASGI 기반으로 비동기 처리 지원
- 멀티 워커 배포 가능 (Gunicorn + Uvicorn)

7.2.2 스토리지 확장

- 현재: 로컬 파일 시스템

- 확장 가능: AWS S3, Google Cloud Storage 등

---

8. 보안 고려사항

8.1 현재 구현된 보안 기능

8.1.1 파일 업로드 검증

```
allowed_extensions = {".jpg", ".jpeg", ".png", ".webp"}
```

8.1.2 API 키 보호

- 환경변수로 관리 (.env)
- 코드에 하드코딩 금지

8.2 추가 필요 보안 조치

8.2.1 파일 크기 제한

- 현재 미구현
- 대용량 파일 업로드 방지 필요

8.2.2 인증/인가

- 현재 미구현
- 사용자 인증 및 요청 제한 필요

8.2.3 입력 검증

- 파일 내용 검증 (실제 이미지 파일인지)
- 악성 파일 업로드 방지

---

9. 배포 및 운영

9.1 개발 환경 실행

```
# 의존성 설치
pip install -r requirements.txt

# 환경변수 설정
export GEMINI_API_KEY="your-api-key"

# 서버 실행
python -m uvicorn app.main:app --reload
```

9.2 프로덕션 배포 고려사항

9.2.1 웹 서버

- Uvicorn (ASGI 서버)
- Gunicorn (프로세스 관리자)

9.2.2 리버스 프록시

- Nginx (정적 파일 서빙, SSL 종료)

9.2.3 컨테이너화

- Docker 이미지 생성
- Docker Compose로 서비스 오케스트레이션

9.2.4 모니터링

- Health Check 엔드포인트 활용
- 로그 수집 및 분석

---

10. 결론

10.1 시스템 강점

1. **분석 기반 개선**: 원본을 무시하지 않고 분석 후 개선
2. **현실적인 변환**: 극단적 변화 대신 실현 가능한 제안
3. **다양한 스타일**: 5가지 인테리어 스타일 지원
4. **한국어 지원**: 분석 및 제안이 모두 한국어로 제공
5. **BM 연계**: 가구 추가 기능으로 제품 판매 가능성

10.2 핵심 기술 성과

- Google Gemini 2.5 Flash Image API 통합
- 2-Step Workflow 프롬프트 엔지니어링
- FastAPI 기반 RESTful API 구현
- Base64 이미지 데이터 처리

10.3 개선 이력

1. 이미지 경로 문제 해결 (상대 → 절대 경로)
2. API 응답 파싱 안정화
3. 극단적 변환 방지 (보존적 접근법)
4. 분석 텍스트 한국어 출력
5. 프리셋 강제 적용 → 분석 기반 개선으로 전환