📱 Flutter 리포트 앱: 프로젝트 요구사항 명세서

📋 문서 정보

- **버전**: v2.1
- **작성일**: 2025년 6월 25일
- **프로젝트 타입**: Flutter 모바일/웹 + Spring Boot 백엔드
- **개발 방식**: 크로스플랫폼 (Flutter) + 마이크로서비스 (Spring Boot)
- **기술 스택**: Spring Boot + Spring Security + JPA + QueryDSL + Kafka + Docker
- **인증 방식**: OAuth2 + JWT
- **배포 방식**: Docker Compose + Nginx Load Balancing

---

1. 🎯 프로젝트 개요

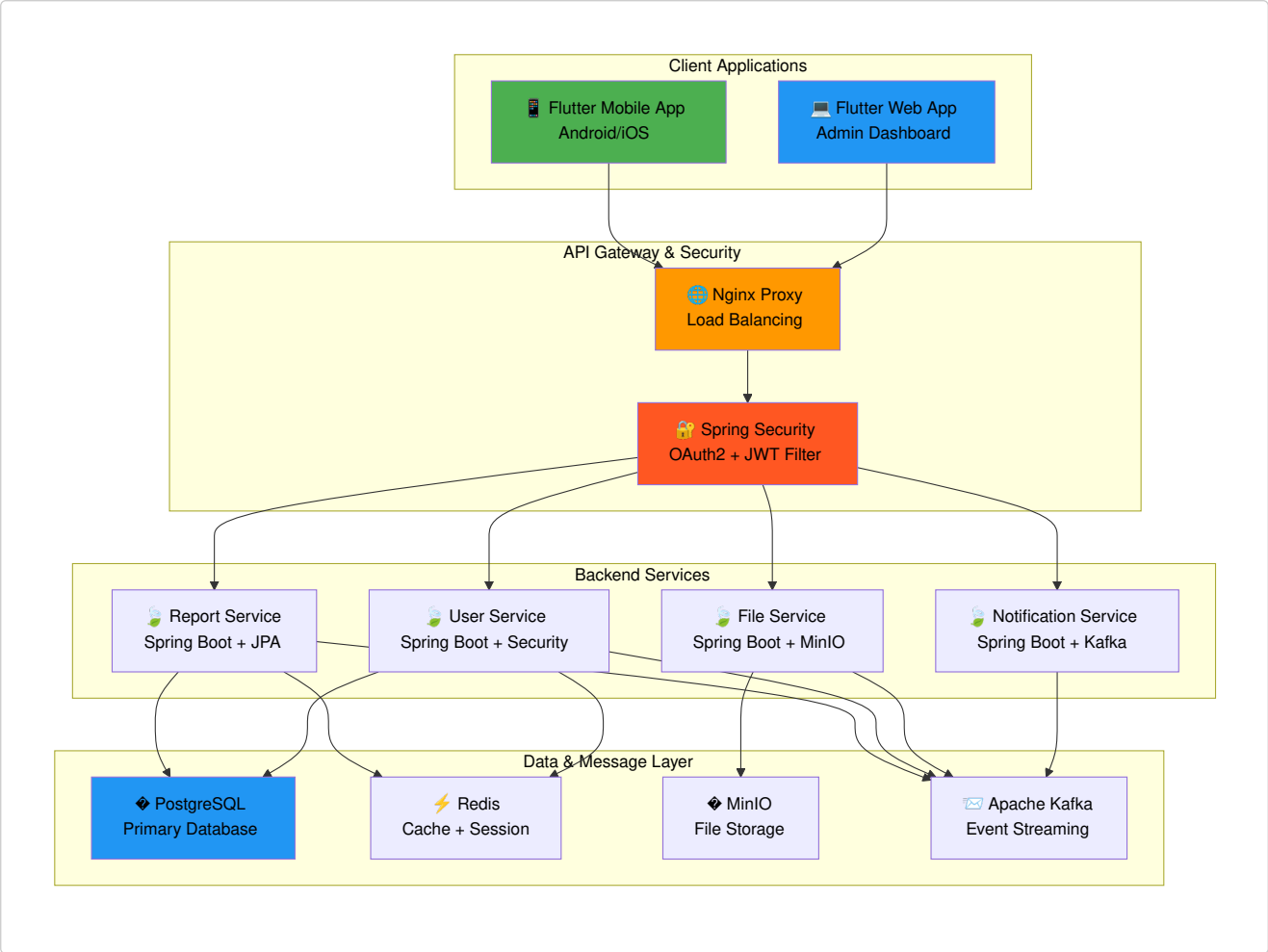
📋 항목	📝 내용
프로젝트명	현장 보고 및 관리 통합 플랫폼
프로젝트 목표	사용자가 모바일/웹을 통해 현장 상황(텍스트, 사진, 위치 등)을 신속하고 정확하게 보고하고, 관리자는 실시간 웹 대시보드에서 확인 및 관리할 수 있는 통합 시스템 구축
개발 프레임워크	Flutter (Mobile + Web)
대상 플랫폼	Android, iOS, Web (Desktop Browser)
주요 사용자	현장 직원 (모바일), 관리자 (웹 대시보드)
개발 기간	3개월 (MVP), 6개월 (전체 기능)

🎨 핵심 가치

- ⚡ **신속성**: 현장에서 즉시 보고서 작성 및 제출
- 🎯 **정확성**: 위치, 사진, 서명을 통한 정확한 현장 상황 기록
- 🔄 **실시간성**: 관리자 대시보드에서 실시간 현황 모니터링
- 📱 **접근성**: 모바일과 웹 모두에서 일관된 사용자 경험

---

2. 🏗️ 시스템 아키텍처



---

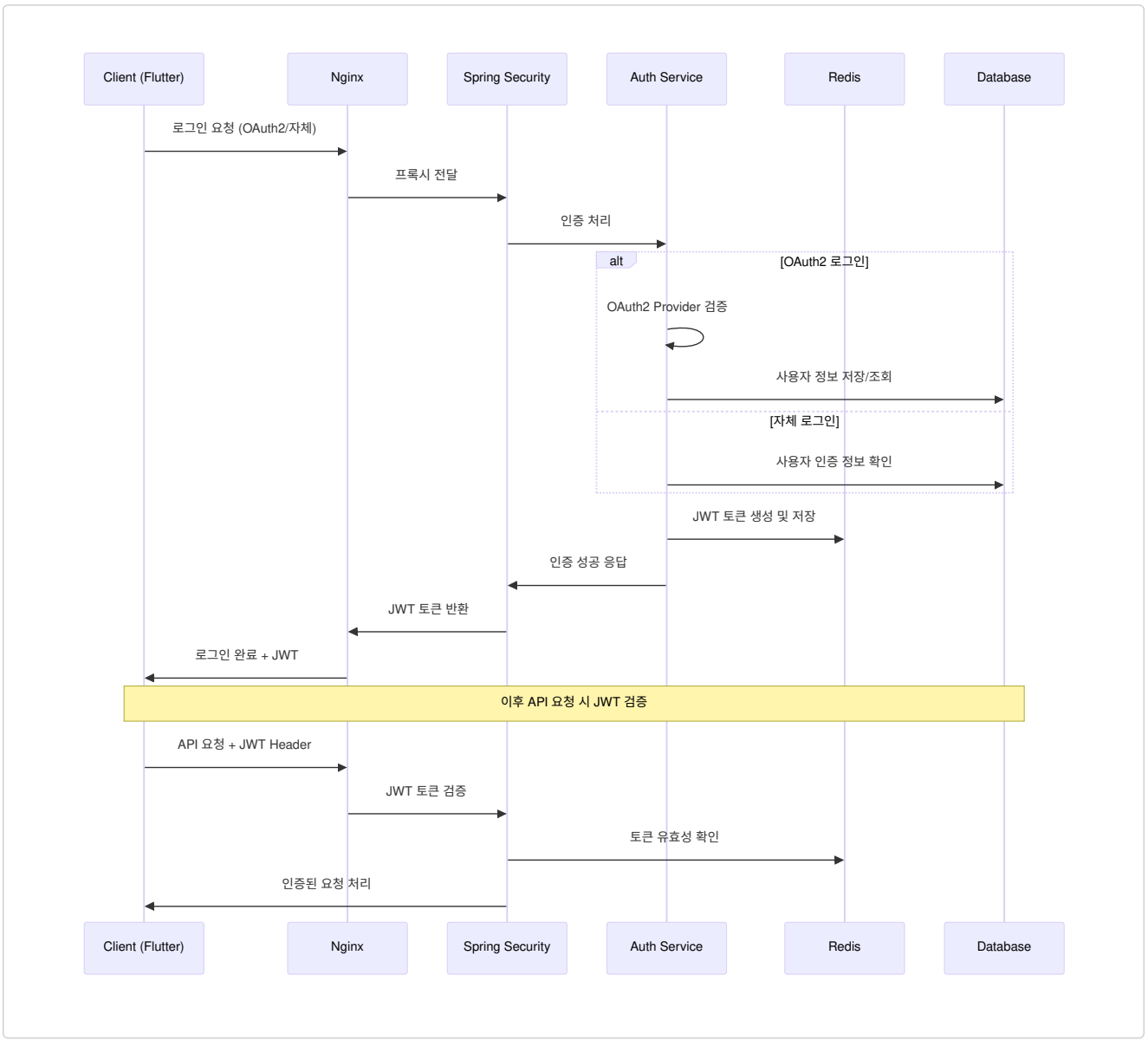
3. 📱 기능 요구사항

3.1. 👤 사용자 관리 기능 (필수)

🔐 인증 및 권한 관리

- **로그인/로그아웃**:
  - **OAuth2 소셜 로그인**: Google, Kakao, Naver 등
  - **자체 계정 로그인**: 이메일/비밀번호 + JWT
  - **생체인증**: 지문, 안면인식 (모바일)
  - **JWT 토큰 관리**: Access Token + Refresh Token
- **회원가입**:
  - 소셜 계정 연동 회원가입
  - 자체 계정 생성 (이메일 인증)
  - 관리자 승인 후 계정 활성화
  - Spring Security 기반 권한 관리
- **프로필 관리**:
  - 개인정보 수정 (이름, 연락처, 부서)
  - 비밀번호 변경 (자체 계정)
  - 연동 계정 관리 (OAuth2)
  - 프로필 사진 업로드

🔒 Spring Security 기반 보안 아키텍처

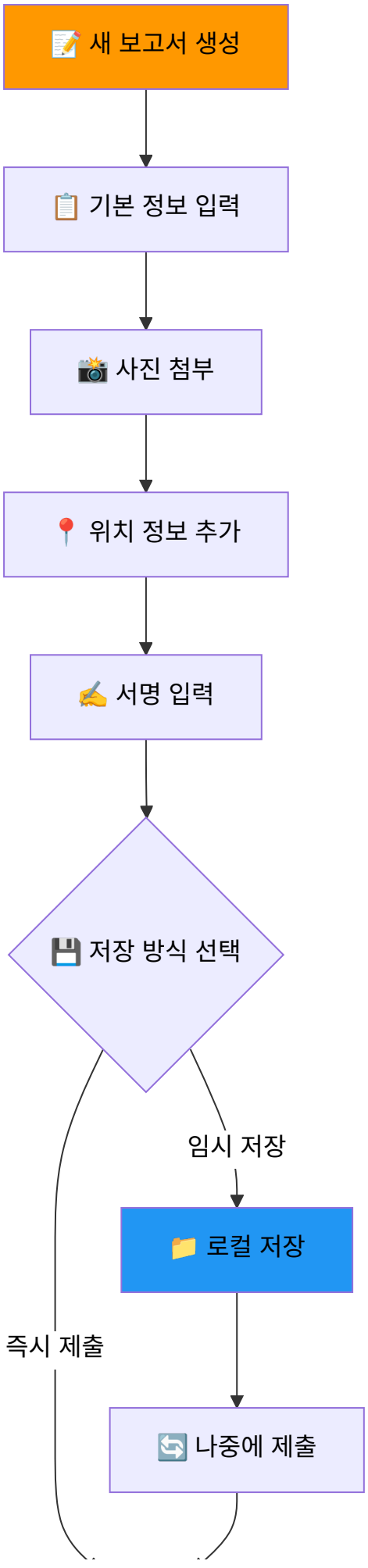


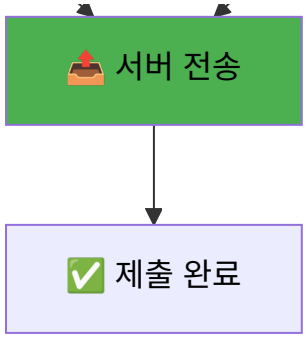
👥 사용자 역할 관리

🎭 역할	📋 권한	📱 접근 플랫폼
현장 직원	보고서 작성, 조회, 수정	Mobile App
팀 리더	팀 보고서 관리, 승인	Mobile + Web
관리자	전체 시스템 관리, 대시보드	Web Dashboard
최고 관리자	시스템 설정, 사용자 관리	Web Dashboard

3.2. 📝 보고서 작성 및 제출 기능 (핵심)

✍️ 보고서 생성 프로세스


📝 새 보고서 생성
📋 기본 정보 입력
📸 사진 첨부
📍 위치 정보 추가
✍️ 서명 입력
💾 저장 방식 선택
임시 저장
📁 로컬 저장
즉시 제출
🔄 나중에 제출



📋 입력 항목 상세

1. **텍스트 정보**:
    - 보고서 제목 (필수, 최대 100자)
    - 보고서 내용 (필수, 최대 2000자)
    - 카테고리 선택 (안전, 품질, 진행상황 등)
    - 우선순위 설정 (낮음, 보통, 높음, 긴급)
2. **사진 첨부**:
    - 카메라 직접 촬영 (최대 10장)
    - 갤러리에서 선택 (최대 10장)
    - 이미지 압축 및 최적화
    - 사진별 설명 추가 가능
3. **위치 정보**:
    - GPS 자동 획득 (정확도 표시)
    - 지도에서 수동 선택/수정
    - 주소 정보 자동 변환
    - 오프라인 상태에서도 기록
4. **서명 기능**:
    - 터치스크린 서명 입력
    - 서명 이미지 저장
    - 서명자 정보 자동 기록

💾 저장 및 제출 기능

- **임시 저장**:
    - 오프라인 상태에서도 로컬 저장
    - 자동 저장 (30초마다)
    - 저장된 항목 복구 기능
- **보고서 제출**:

    - 온라인 즉시 제출
    - 오프라인 큐잉 (연결 시 자동 전송)
    - 제출 상태 실시간 추적

3.3. 📊 보고서 조회 및 관리 기능

📱 모바일 앱 (현장 직원용)

- **보고서 목록**:
    - 상태별 필터링 (임시저장, 제출완료, 승인, 반려)
    - 날짜별 정렬 및 검색
    - 무한 스크롤 페이징
- **상세 조회**:
    - 모든 입력 정보 확인
    - 첨부 사진 갤러리 뷰
    - 처리 상태 및 피드백 확인
- **검색 기능**:
    - 제목/내용 키워드 검색
    - 날짜 범위 검색
    - 카테고리별 필터링

💻 웹 대시보드 (관리자용)

- **실시간 대시보드**:
    - 오늘의 보고서 통계
    - 진행 상태별 분포
    - 지역별/부서별 현황
- **보고서 관리**:
    - 일괄 승인/반려 처리
    - 댓글 및 피드백 작성
    - 보고서 상태 변경
- **분석 기능**:
    - 월간/주간 트렌드 분석
    - 부서별 성과 분석
    - 문제점 패턴 분석

3.4. 🔔 알림 및 커뮤니케이션 기능

📱 푸시 알림

🎯 알림 유형	📱 모바일	💻 웹	📧 이메일
보고서 승인	✅	✅	⭕
보고서 반려	✅	✅	✅
긴급 보고서 접수	⭕	✅	✅
시스템 공지사항	✅	✅	⭕
정기 리마인더	✅	⭕	⭕

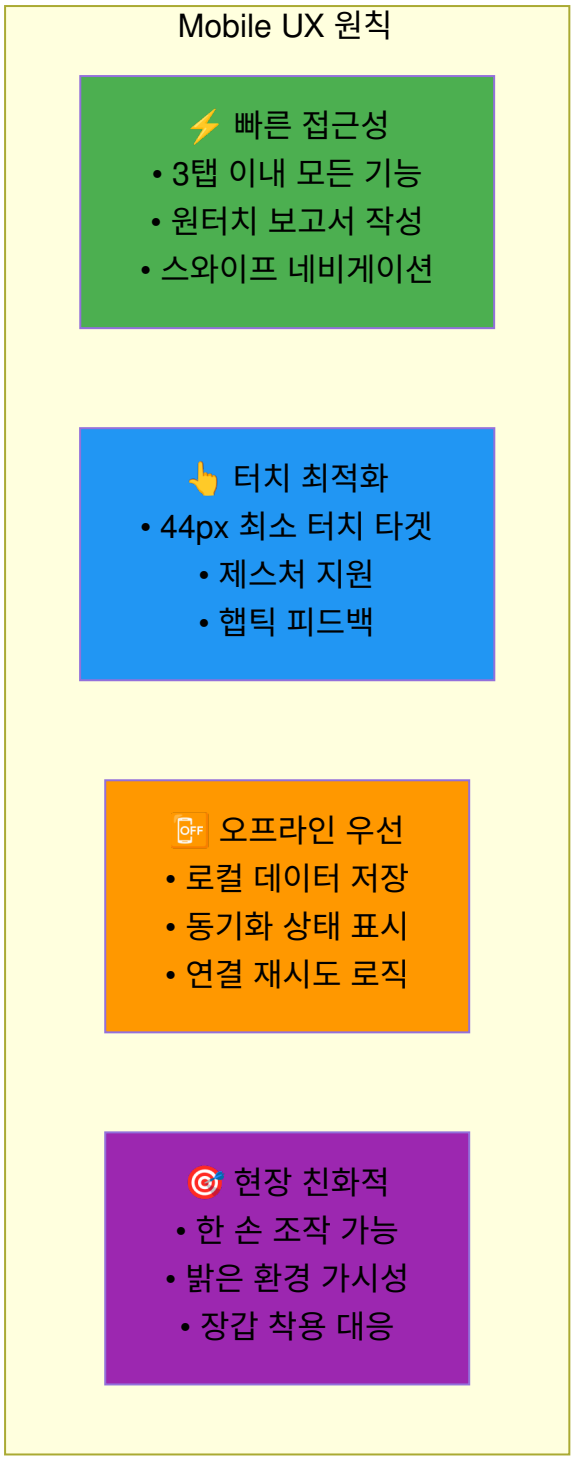
💬 커뮤니케이션

- **댓글 시스템**: 보고서별 질문/답변
- **상태 알림**: 실시간 처리 상태 업데이트
- **공지사항**: 전체 공지 및 개별 메시지

---

4. 🎨 UX/UI 설계 원칙

4.1. 📱 모바일 앱 UX 원칙



4.2. 💻 웹 대시보드 UX 원칙

- **정보 중심 설계**: 데이터 시각화 우선
- **반응형 레이아웃**: 다양한 화면 크기 지원
- **키보드 네비게이션**: 업무 효율성 향상
- **다중 창 지원**: 여러 보고서 동시 관리

---

5. 🚀 비기능 요구사항

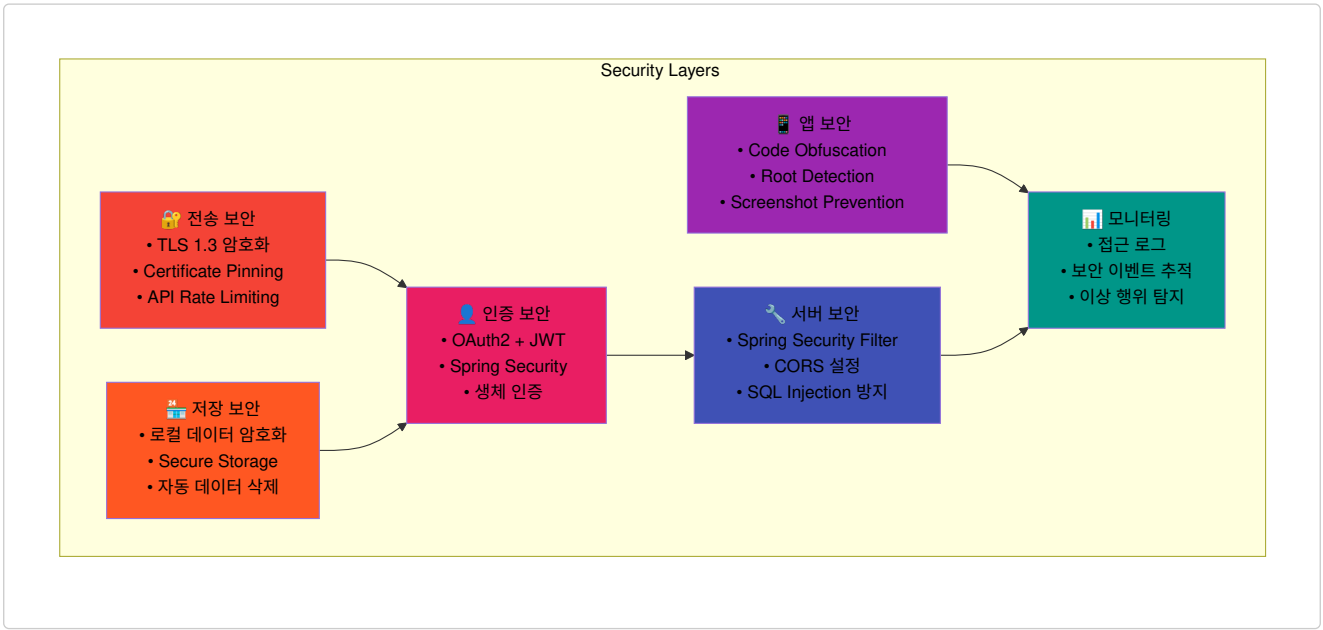
5.1. 📱 플랫폼 및 호환성

🎯 플랫폼	📋 지원 버전	🔧 특별 고려사항
Android	API 21+ (Android 5.0+)	Material Design 3 적용
iOS	iOS 12.0+	Cupertino Design 적용
Web	Chrome 90+, Safari 14+, Firefox 88+	Progressive Web App

5.2. ⚡ 성능 목표

📊 메트릭	🎯 목표	📱 모바일	💻 웹
앱 로딩	첫 화면 표시	< 2초	< 1.5초
API 응답	평균 응답 시간	< 500ms	< 300ms
이미지 업로드	5MB 사진 전송	< 10초	< 5초
오프라인 동기화	데이터 동기화	< 30초	N/A

5.3. 🔒 보안 요구사항



🔐 보안 상세 요구사항

🛡️ 보안 영역	📋 구현 방식	🔧 기술 스택
API 인증	OAuth2 + JWT	Spring Security + JWT
데이터 암호화	AES-256 + TLS 1.3	JPA Encryption + HTTPS
세션 관리	Stateless JWT	Redis Token Store
파일 보안	서명된 URL + 접근 제어	MinIO + Pre-signed URLs
모바일 보안	Certificate Pinning	Flutter dio_pinning

🛡️ 보안 영역	📋 구현 방식	🔧 기술 스택
로그 보안	민감정보 마스킹	Logback 필터

5.4. 📈 확장성 요구사항

- **사용자 규모**: 동시 접속 1,000명
- **데이터 처리**: 일일 보고서 10,000건
- **파일 저장**: 월 100GB 증가 대응
- **지역 확장**: 다국어 지원 준비

---

6. 🛠️ 기술 스택 및 아키텍처

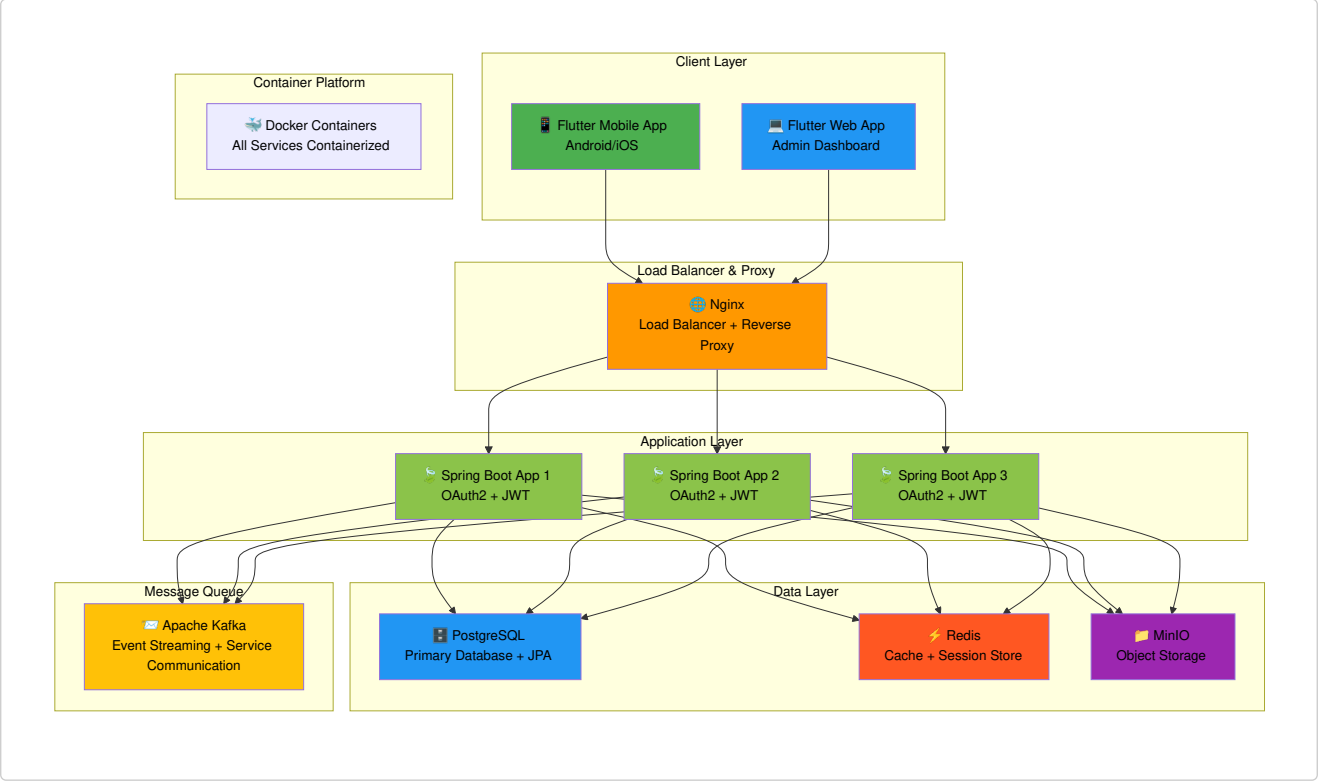
6.1. 🎨 프론트엔드 기술 스택

```
프레임워크:
  - Flutter 3.16+
  - Dart 3.2+

상태관리:
  - Provider (가벼운 앱용)
  - Riverpod (복잡한 상태관리)
  - Bloc (엔터프라이즈급)

UI 라이브러리:
  - Material Design 3
  - Cupertino (iOS)
  - Custom Design System

인증 연동:
  - google_sign_in: Google OAuth2
  - kakao_flutter_sdk: Kakao Login
  - flutter_naver_login: Naver Login
  - oauth2: 커스텀 OAuth2 처리

주요 패키지:
  - dio: HTTP 클라이언트 + JWT 인터셉터
  - camera: 카메라 기능
  - geolocator: GPS 위치
  - hive: 로컬 저장소 (오프라인)
  - firebase_messaging: 푸시 알림
  - image_picker: 이미지 선택
  - signature: 서명 입력
  - pdf: PDF 생성
  - flutter_secure_storage: 보안 토큰 저장
```

6.2. ⚙️ 백엔드 기술 스택

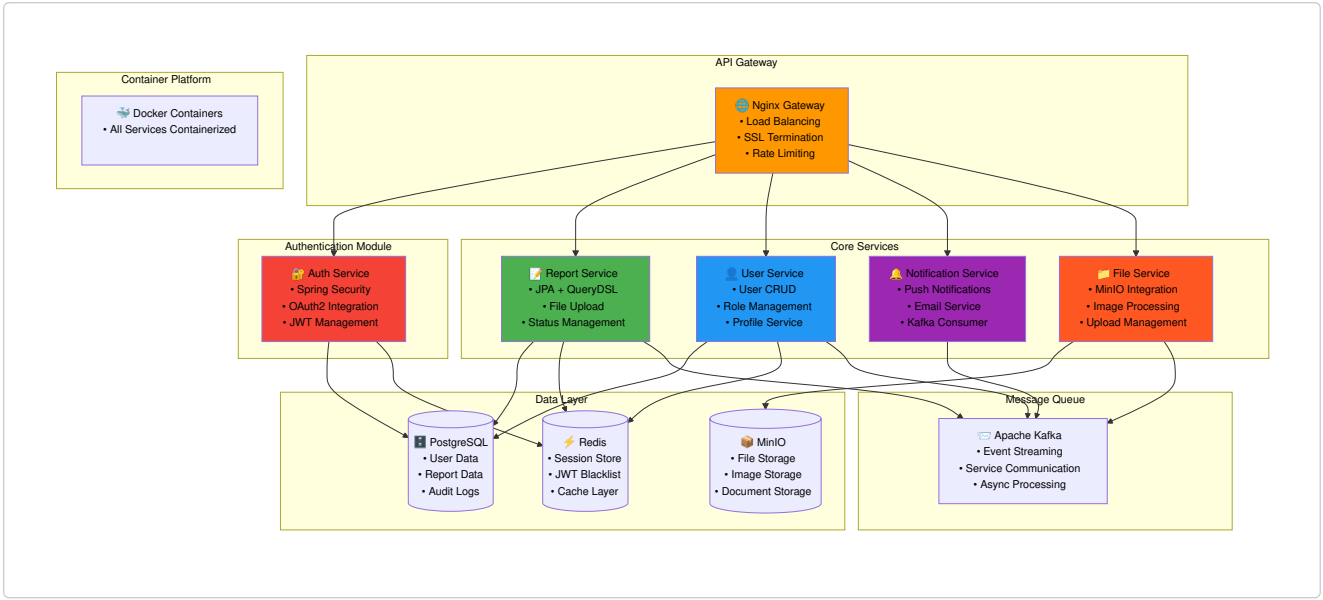
🔧 구분	🛠️ 기술	📝 설명
프레임워크	Spring Boot 3.2+	Java 기반 엔터프라이즈 애플리케이션
보안	Spring Security 6+	OAuth2 + JWT 통합 인증
데이터 액세스	Spring Data JPA + QueryDSL	타입 안전 쿼리 + ORM
빌드 도구	Gradle 8+	의존성 관리 및 빌드 자동화
데이터베이스	PostgreSQL + Redis	관계형 DB + 캐싱
메시징	Apache Kafka	내부 서비스 간 비동기 통신
파일 저장소	MinIO (S3 호환)	오브젝트 스토리지
인증 방식	OAuth2 + 자체 JWT	소셜 로그인 + 커스텀 인증
API 문서	SpringDoc OpenAPI 3	자동 API 문서 생성
컨테이너화	Docker + Docker Compose	배포 환경 표준화
프록시/로드밸런싱	Nginx	리버스 프록시 + 로드 밸런싱

6.3. 🏗️ 인프라 구성



6.4. 🍃 Spring Boot 백엔드 아키텍처

🏛️ 마이크로서비스 구조

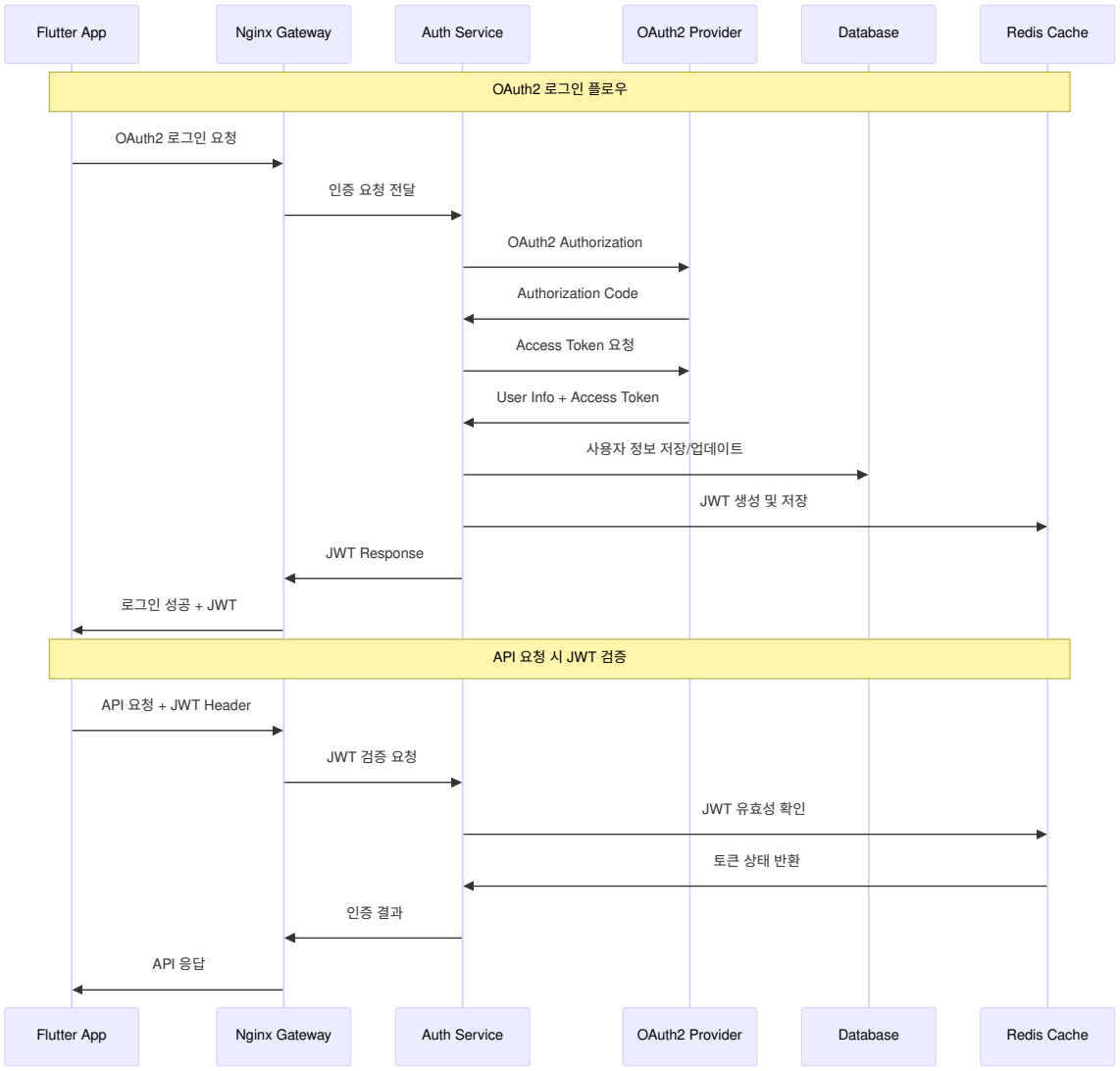


🔧 기술 스택 상세

📊 Spring Boot 구성

```
Backend Stack:
  Framework: Spring Boot 3.2+
  Language: Java 17+
  Build Tool: Gradle 8+

Security:
  - Spring Security 6+
  - OAuth2 Resource Server
  - JWT Token Authentication
  - Method Level Security

Data Access:
  - Spring Data JPA
  - QueryDSL 5+
  - Database Migration: Flyway
  - Connection Pool: HikariCP

Messaging:
  - Spring Kafka
  - Event-Driven Architecture
  - Async Processing

Monitoring:
  - Spring Boot Actuator
  - Micrometer Metrics
  - Logging: Logback
```

🔐 OAuth2 + JWT 인증 플로우



📁 프로젝트 구조

```
spring-boot-report-api/
├── 📁 gateway-service/              # Nginx 설정 (Docker)
├── 📁 auth-service/                 # 인증 서비스
│   ├── 📁 src/main/java/
│   │   ├── 📁 config/               # Security Config
│   │   ├── 📁 controller/           # OAuth2 Controller
│   │   ├── 📁 service/              # JWT Service
│   │   └── 📁 entity/               # User Entity
├── 📁 report-service/               # 보고서 서비스
│   ├── 📁 src/main/java/
│   │   ├── 📁 entity/               # Report Entity
│   │   ├── 📁 repository/           # JPA + QueryDSL
│   │   ├── 📁 service/              # Business Logic
│   │   └── 📁 controller/           # REST API
├── 📁 file-service/                 # 파일 서비스
│   ├── 📁 src/main/java/
│   │   ├── 📁 config/               # MinIO Config
│   │   ├── 📁 service/              # File Upload/Download
```

```
│   │   └── 📁 controller/          # File API
├── 📁 notification-service/       # 알림 서비스
│   ├── 📁 src/main/java/
│   │   ├── 📁 kafka/               # Kafka Consumer
│   │   ├── 📁 service/             # Push Service
│   │   └── 📁 config/              # Kafka Config
├── 📁 common/                      # 공통 모듈
│   ├── 📁 src/main/java/
│   │   ├── 📁 entity/              # 공통 Entity
│   │   ├── 📁 dto/                 # 공통 DTO
│   │   ├── 📁 exception/           # 예외 처리
│   │   └── 📁 util/                # 유틸리티
└── 📁 docker/                      # Docker 구성
    ├── 📄 docker-compose.yml       # 전체 서비스 구성
    ├── 📄 nginx.conf               # Nginx 설정
    └── 📁 init/                    # DB 초기화 스크립트
```

🐳 Docker 구성

Docker Compose 구조

```yaml
# docker-compose.yml
version: "3.8"
services:
  # 데이터베이스
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: report_db
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init:/docker-entrypoint-initdb.d
    ports:
      - "5432:5432"

  # 캐시 & 세션 스토어
  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    command: redis-server --appendonly yes
    volumes:
      - redis_data:/data

  # 오브젝트 스토리지
  minio:
    image: minio/minio:latest
    ports:
      - "9000:9000"
```

```
      - "9001:9001"
    environment:
      MINIO_ACCESS_KEY: minio_access_key
      MINIO_SECRET_KEY: minio_secret_key
    command: server /data --console-address ":9001"
    volumes:
      - minio_data:/data

  # 메시지 큐
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

  # 백엔드 서비스들
  auth-service:
    build: ./auth-service
    depends_on:
      - postgres
      - redis
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/report_db
      SPRING_REDIS_HOST: redis
    ports:
      - "8081:8080"

  report-service:
    build: ./report-service
    depends_on:
      - postgres
      - redis
      - kafka
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/report_db
      SPRING_REDIS_HOST: redis
      SPRING_KAFKA_BOOTSTRAP_SERVERS: kafka:9092
    ports:
      - "8082:8080"

  file-service:
    build: ./file-service
```

```
    depends_on:
      - minio
      - kafka
    environment:
      MINIO_ENDPOINT: http://minio:9000
      SPRING_KAFKA_BOOTSTRAP_SERVERS: kafka:9092
    ports:
      - "8083:8080"

  notification-service:
    build: ./notification-service
    depends_on:
      - kafka
      - redis
    environment:
      SPRING_KAFKA_BOOTSTRAP_SERVERS: kafka:9092
      SPRING_REDIS_HOST: redis
    ports:
      - "8084:8080"

  # 리버스 프록시 & 로드 밸런서
  nginx:
    image: nginx:alpine
    depends_on:
      - auth-service
      - report-service
      - file-service
      - notification-service
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/ssl:/etc/nginx/ssl
    restart: unless-stopped

volumes:
  postgres_data:
  redis_data:
  minio_data:
```

🌐 Nginx 로드 밸런싱 설정

```
# nginx.conf
upstream auth_backend {
    server auth-service:8080;
}

upstream report_backend {
    server report-service:8080;
}
```

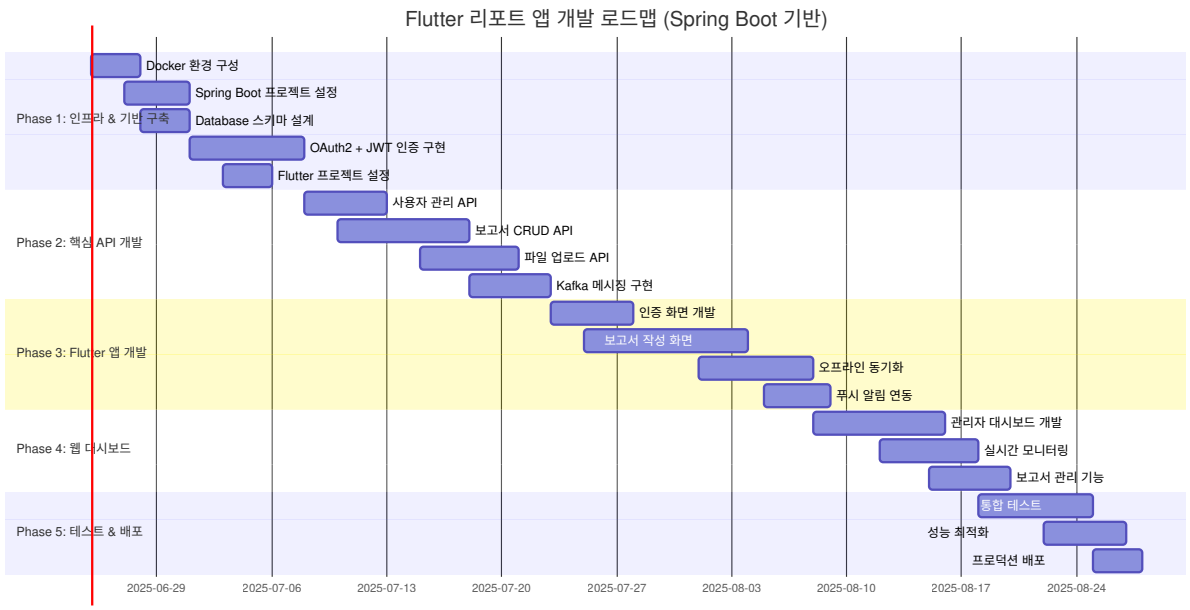
```
upstream file_backend {
    server file-service:8080;
}

upstream notification_backend {
    server notification-service:8080;
}

server {
    listen 80;
    server_name localhost;

    # API Gateway 역할
    location /api/auth/ {
        proxy_pass http://auth_backend/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api/reports/ {
        proxy_pass http://report_backend/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api/files/ {
        proxy_pass http://file_backend/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # 파일 업로드를 위한 설정
        client_max_body_size 100M;
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;
    }

    location /api/notifications/ {
        proxy_pass http://notification_backend/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # 헬스체크 엔드포인트
    location /health {
```

```
        access_log off;
        return 200 "healthy\n";
        add_header Content-Type text/plain;
    }
}
```

---

7. 📅 개발 로드맵

7.1. 🚀 개발 단계별 계획



7.2. 🎯 마일스톤별 목표

📅 Phase	🎯 목표	📋 주요 기능	🔧 완료 기준
Phase 1	기반 인프라 구축	Docker, Spring Boot, OAuth2	인증 API 완성
Phase 2	백엔드 API 개발	CRUD API, 파일 업로드, Kafka	Postman 테스트 완료
Phase 3	모바일 앱 개발	Flutter 앱, 오프라인 지원	MVP 앱 완성
Phase 4	웹 대시보드 개발	관리자 기능, 실시간 모니터링	실무 사용 가능
Phase 5	테스트 & 배포	통합 테스트, 성능 최적화	프로덕션 준비 완료

---

8. 🧪 테스트 계획

8.1. 📱 모바일 앱 테스트

🔍 테스트 유형	🛠️ 도구	📋 범위

🔍 테스트 유형	🛠️ 도구	📋 범위
Unit Test	Flutter Test	비즈니스 로직, 유틸리티 함수
Widget Test	Flutter Test	UI 컴포넌트, 상호작용
Integration Test	Flutter Driver	전체 사용자 플로우
Performance Test	Flutter Inspector	메모리, CPU, 렌더링 성능

8.2. 🌐 웹 앱 테스트

- **Cross-browser Testing**: Chrome, Safari, Firefox
- **Responsive Testing**: 다양한 화면 크기
- **Accessibility Testing**: WCAG 2.1 준수
- **Load Testing**: 동시 접속자 1,000명

8.3. ⚙️ Spring Boot 백엔드 테스트

🔍 테스트 유형	🛠️ 도구	📋 범위
Unit Test	JUnit 5 + Mockito	Service Layer, Repository Layer
Integration Test	@SpringBootTest	API 엔드포인트 통합 테스트
Security Test	Spring Security Test	OAuth2 인증/인가 테스트
Repository Test	@DataJpaTest	JPA + QueryDSL 쿼리 테스트
Kafka Test	@EmbeddedKafka	메시지 발행/구독 테스트
API Contract Test	Spring Cloud Contract	서비스 간 계약 테스트

8.4. 🐳 Docker & 인프라 테스트

```
테스트 환경:
  Container Testing:
    - Docker Compose 서비스 통합 테스트
    - 컨테이너 간 네트워크 연결 테스트
    - 볼륨 마운트 및 데이터 지속성 테스트

  Load Testing:
    - Nginx 로드 밸런싱 테스트
    - 동시 접속 1000명 부하 테스트
    - 데이터베이스 성능 테스트

  Security Testing:
    - JWT 토큰 보안 테스트
    - OAuth2 플로우 보안 검증
    - API Rate Limiting 테스트
```

---

9. 📊 성공 지표 (KPI)

9.1. 👥 사용자 지표

📊 메트릭	🎯 목표값	📈 측정 방법
일일 활성 사용자	80%	Spring Boot Actuator 메트릭
보고서 작성 완료율	95%	작성 시작 vs 제출 추적
평균 보고서 작성 시간	< 3분	시간 측정 로그 분석
사용자 만족도	4.5/5	인앱 설문조사

9.2. 🔧 기술 지표

📊 메트릭	🎯 목표값	📈 측정 도구
API 응답 시간	< 200ms	Spring Boot Actuator
데이터베이스 쿼리 시간	< 100ms	QueryDSL 성능 모니터링
Kafka 메시지 처리 시간	< 50ms	Kafka 메트릭
Docker 컨테이너 재시작 횟수	< 1회/월	Docker 로그 모니터링
OAuth2 인증 성공률	> 99.9%	Spring Security 로그

---

10. 🔧 운영 및 유지보수

10.1. 🐳 Docker 기반 배포 및 운영

```
배포 전략:
  환경 구성:
    - 개발환경: docker-compose.dev.yml
    - 스테이징: docker-compose.staging.yml
    - 프로덕션: docker-compose.prod.yml

  자동화:
    - CI/CD: GitHub Actions
    - 이미지 빌드: Docker Multi-stage Build
    - 배포 스크립트: Shell Script

  모니터링:
    - 컨테이너 상태: Docker Health Check
    - 로그 수집: ELK Stack (Elasticsearch, Logstash, Kibana)
    - 메트릭 수집: Prometheus + Grafana
```

10.2. 🏗️ Spring Boot 운영

- **프로파일 관리**: dev, staging, prod 환경별 설정

- **로그 관리**: Logback을 통한 구조화된 로깅
- **모니터링**: Spring Boot Actuator + Micrometer
- **보안 패치**: 정기적인 의존성 업데이트

10.3. 📞 고객 지원

- **문의 채널**: 앱 내 문의, 이메일, 전화
- **FAQ 관리**: Spring Boot 기반 FAQ API
- **사용자 교육**: 사용 매뉴얼, 비디오 가이드
- **피드백 수집**: Kafka를 통한 실시간 피드백 수집

---

11. 💰 예산 및 리소스

11.1. 👥 개발 팀 구성

👤 역할	👥 인원	⏰ 기간	💰 예상 비용
Flutter 개발자	2명	6개월	₩120,000,000
Spring Boot 백엔드 개발자	2명	6개월	₩120,000,000
DevOps 엔지니어	1명	3개월	₩30,000,000
UI/UX 디자이너	1명	2개월	₩20,000,000
QA 엔지니어	1명	3개월	₩24,000,000
프로젝트 매니저	1명	6개월	₩36,000,000

11.2. 🔧 인프라 비용 (월간)

💻 항목	💰 비용	📝 설명
서버 호스팅	₩400,000	Docker 컨테이너 호스팅 (4vCPU, 16GB RAM)
데이터베이스	₩200,000	PostgreSQL + Redis 클러스터
도메인 및 SSL	₩30,000	도메인, SSL 인증서
백업 스토리지	₩100,000	데이터 백업 및 MinIO 스토리지
모니터링 도구	₩80,000	ELK Stack + Prometheus + Grafana

---

12. 🔮 향후 확장 계획

12.1. 🚀 단계별 확장 로드맵

12.2. 🎯 기술적 확장 계획

- **Kubernetes 도입**: Docker Compose → Kubernetes 마이그레이션
- **서비스 메시 도입**: Istio를 통한 마이크로서비스 관리
- **AI 기능 추가**: Spring Boot AI 프레임워크 도입
- **GraphQL 지원**: REST API → GraphQL 점진적 전환
- **클라우드 네이티브**: AWS/GCP/Azure 클라우드 배포

---

13. 📚 개발 가이드 및 Best Practices

13.1. 🍃 Spring Boot 개발 가이드

📁 프로젝트 구조 (Gradle Multi-Module)

```
flutter-report-backend/
├── 📁 build.gradle                    # Root Gradle 설정
├── 📁 settings.gradle                 # Multi-module 설정
├── 📁 docker-compose.yml              # 개발환경 Docker 구성
├── 📁 auth-service/                   # 인증 서비스
│   ├── 📁 src/main/java/
│   │   ├── 📁 config/                # Security, OAuth2 Config
│   │   ├── 📁 controller/            # Auth API Controller
```

```
│   │   ├── 📁 service/                    # JWT, OAuth2 Service
│   │   ├── 📁 entity/                     # User, Role Entity
│   │   └── 📁 repository/                 # JPA Repository
│   └── 📄 build.gradle                    # Auth Service Dependencies
├── 📁 report-service/                     # 보고서 서비스
│   ├── 📁 src/main/java/
│   │   ├── 📁 entity/                     # Report, Comment Entity
│   │   ├── 📁 repository/                 # JPA + QueryDSL Repository
│   │   ├── 📁 service/                    # Business Logic
│   │   ├── 📁 controller/                 # REST API Controller
│   │   └── 📁 dto/                        # Request/Response DTO
│   └── 📄 build.gradle
├── 📁 file-service/                       # 파일 서비스
│   ├── 📁 src/main/java/
│   │   ├── 📁 config/                     # MinIO Configuration
│   │   ├── 📁 service/                    # File Upload/Download
│   │   └── 📁 controller/                 # File API
│   └── 📄 build.gradle
├── 📁 notification-service/               # 알림 서비스
│   ├── 📁 src/main/java/
│   │   ├── 📁 kafka/                      # Kafka Consumer/Producer
│   │   ├── 📁 service/                    # Push Notification
│   │   └── 📁 config/                     # Kafka Configuration
│   └── 📄 build.gradle
├── 📁 common/                             # 공통 모듈
│   ├── 📁 src/main/java/
│   │   ├── 📁 entity/                     # 공통 Base Entity
│   │   ├── 📁 dto/                        # 공통 DTO
│   │   ├── 📁 exception/                  # 글로벌 예외 처리
│   │   ├── 📁 util/                       # 유틸리티 클래스
│   │   └── 📁 config/                     # 공통 Configuration
│   └── 📄 build.gradle
└── 📁 api-gateway/                        # API Gateway (선택적)
    ├── 📁 src/main/java/
    │   ├── 📁 filter/                     # Gateway Filter
    │   ├── 📁 config/                     # Gateway Configuration
    │   └── 📁 security/                   # Gateway Security
    └── 📄 build.gradle
```

🔧 Gradle 의존성 설정

```gradle
// build.gradle (Root)
plugins {
    id 'org.springframework.boot' version '3.2.0'
    id 'io.spring.dependency-management' version '1.1.4'
    id 'java'
}

allprojects {
    group = 'com.flutterreport'
    version = '1.0.0'
```

```
    sourceCompatibility = '17'

    repositories {
        mavenCentral()
    }
}

subprojects {
    apply plugin: 'java'
    apply plugin: 'org.springframework.boot'
    apply plugin: 'io.spring.dependency-management'

    dependencies {
        // Spring Boot Starters
        implementation 'org.springframework.boot:spring-boot-starter-
web'
        implementation 'org.springframework.boot:spring-boot-starter-
data-jpa'
        implementation 'org.springframework.boot:spring-boot-starter-
security'
        implementation 'org.springframework.boot:spring-boot-starter-
oauth2-resource-server'
        implementation 'org.springframework.boot:spring-boot-starter-
validation'
        implementation 'org.springframework.boot:spring-boot-starter-
actuator'

        // Database
        runtimeOnly 'org.postgresql:postgresql'
        implementation 'org.springframework.boot:spring-boot-starter-
data-redis'

        // QueryDSL
        implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'
        annotationProcessor 'com.querydsl:querydsl-apt:5.0.0:jakarta'

        // Kafka
        implementation 'org.springframework.kafka:spring-kafka'

        // JWT
        implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
        runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.5'
        runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.5'

        // API Documentation
        implementation 'org.springdoc:springdoc-openapi-starter-webmvc-
ui:2.2.0'

        // Testing
        testImplementation 'org.springframework.boot:spring-boot-
starter-test'
        testImplementation 'org.springframework.security:spring-
security-test'
        testImplementation 'org.springframework.kafka:spring-kafka-test'
```

```
        testImplementation 'org.testcontainers:junit-jupiter'
        testImplementation 'org.testcontainers:postgresql'
    }
}
```

🔒 Spring Security + OAuth2 설정

```
// SecurityConfig.java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        http
            .cors(cors ->
cors.configurationSource(corsConfigurationSource()))
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session ->

session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**", "/api/public/
**").permitAll()
                .requestMatchers("/actuator/health").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/reports/
**").hasRole("USER")
                .requestMatchers(HttpMethod.POST, "/api/reports/
**").hasRole("USER")
                .requestMatchers("/api/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated())
            .oauth2ResourceServer(oauth2 -> oauth2
                .jwt(jwt -> jwt.jwtDecoder(jwtDecoder())))

.authenticationEntryPoint(jwtAuthenticationEntryPoint()))
            .addFilterBefore(jwtAuthenticationFilter(),
                UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOriginPatterns(Arrays.asList("*"));
        configuration.setAllowedMethods(Arrays.asList("GET", "POST",
"PUT", "DELETE", "OPTIONS"));
        configuration.setAllowedHeaders(Arrays.asList("*"));
        configuration.setAllowCredentials(true);
```

```
        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
}
```

13.2. 📱 Flutter 개발 가이드

🏗️ 프로젝트 구조 (Clean Architecture)

```
lib/
├── 📁 main.dart                    # 앱 진입점
├── 📁 app/                         # 앱 설정
│   ├── 📁 config/                  # 환경 설정
│   ├── 📁 theme/                   # 테마 설정
│   └── 📁 routes/                  # 라우팅 설정
├── 📁 core/                        # 핵심 기능
│   ├── 📁 network/                 # HTTP 클라이언트
│   ├── 📁 storage/                 # 로컬 저장소
│   ├── 📁 auth/                    # 인증 관리
│   └── 📁 utils/                   # 유틸리티
├── 📁 features/                    # 기능별 모듈
│   ├── 📁 auth/                    # 인증 기능
│   │   ├── 📁 data/                # Repository Implementation
│   │   ├── 📁 domain/              # Entity, Repository Interface
│   │   └── 📁 presentation/        # UI, State Management
│   ├── 📁 reports/                 # 보고서 기능
│   │   ├── 📁 data/
│   │   ├── 📁 domain/
│   │   └── 📁 presentation/
│   └── 📁 profile/                 # 프로필 기능
├── 📁 shared/                      # 공통 위젯
│   ├── 📁 widgets/                 # 재사용 위젯
│   ├── 📁 constants/               # 상수
│   └── 📁 extensions/              # 확장 함수
└── 📁 l10n/                        # 다국어 지원
```

🔧 HTTP 클라이언트 설정 (Dio + JWT)

```
// api_client.dart
class ApiClient {
  late final Dio _dio;
  final AuthService _authService = GetIt.instance<AuthService>();

  ApiClient() {
    _dio = Dio(BaseOptions(
```

```
      baseUrl: AppConfig.apiBaseUrl,
      connectTimeout: const Duration(seconds: 30),
      receiveTimeout: const Duration(seconds: 30),
      headers: {
        'Content-Type': 'application/json',
        'Accept': 'application/json',
      },
    ));

    _setupInterceptors();
  }

  void _setupInterceptors() {
    // JWT 토큰 자동 추가
    _dio.interceptors.add(InterceptorsWrapper(
      onRequest: (options, handler) async {
        final token = await _authService.getAccessToken();
        if (token != null) {
          options.headers['Authorization'] = 'Bearer $token';
        }
        handler.next(options);
      },
      onError: (error, handler) async {
        if (error.response?.statusCode == 401) {
          // 토큰 만료 시 자동 갱신
          final refreshed = await _authService.refreshToken();
          if (refreshed) {
            final newToken = await _authService.getAccessToken();
            error.requestOptions.headers['Authorization'] = 'Bearer
$newToken';
            return _dio.fetch(error.requestOptions);
          } else {
            _authService.logout();
          }
        }
        handler.next(error);
      },
    ));

    // 로깅 인터셉터
    _dio.interceptors.add(LogInterceptor(
      requestBody: true,
      responseBody: true,
      logPrint: (log) => debugPrint(log.toString()),
    ));
  }
}
```

13.3. 🐳 Docker 개발 환경 설정

개발용 Docker Compose

```yaml
# docker-compose.dev.yml
version: "3.8"
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: report_dev
      POSTGRES_USER: dev_user
      POSTGRES_PASSWORD: dev_password
    ports:
      - "5432:5432"
    volumes:
      - postgres_dev_data:/var/lib/postgresql/data
      - ./sql/init.sql:/docker-entrypoint-initdb.d/init.sql

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    command: redis-server --appendonly yes

  minio:
    image: minio/minio:latest
    ports:
      - "9000:9000"
      - "9001:9001"
    environment:
      MINIO_ACCESS_KEY: dev_access_key
      MINIO_SECRET_KEY: dev_secret_key
    command: server /data --console-address ":9001"
    volumes:
      - minio_dev_data:/data

  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

volumes:
  postgres_dev_data:
```

```
  minio_dev_data:
```

간편한 개발 스크립트

```
#!/bin/bash
# start-dev.sh

echo "🚀 Flutter Report App 개발 환경 시작..."

# Docker 서비스 시작
docker-compose -f docker-compose.dev.yml up -d

# 데이터베이스 마이그레이션 대기
echo "⏳ 데이터베이스 준비 중..."
sleep 10

# Spring Boot 애플리케이션 시작
echo "🍃 Spring Boot 서비스 시작..."
./gradlew :auth-service:bootRun &
./gradlew :report-service:bootRun &
./gradlew :file-service:bootRun &
./gradlew :notification-service:bootRun &

echo "✅ 개발 환경이 준비되었습니다!"
echo "📱 Flutter 앱: flutter run"
echo "📊 API 문서: http://localhost:8080/swagger-ui.html"
echo "🗄️ MinIO Console: http://localhost:9001"
```

---

14. 📋 부록

A. 📚 참고 문서

- Spring Boot 공식 문서
- Spring Security OAuth2 가이드
- QueryDSL 문서
- Flutter 공식 문서
- Apache Kafka 문서
- Docker Compose 문서

B. 🛠️ 개발 도구 및 리소스

- **IDE**: IntelliJ IDEA (Spring Boot), VS Code (Flutter)
- **버전 관리**: Git, GitHub
- **API 테스트**: Postman, Insomnia
- **데이터베이스 도구**: DBeaver, pgAdmin
- **디자인**: Figma, Adobe XD

- **프로젝트 관리**: Jira, Notion
- **커뮤니케이션**: Slack, Discord

C. 📞 연락처 및 지원

- **기술 지원**: tech-support@flutterreport.com
- **프로젝트 매니저**: pm@flutterreport.com
- **DevOps 지원**: devops@flutterreport.com
- **긴급 연락처**: +82-10-1234-5678

📊 문서 정보

- **버전**: v2.1
- **최종 수정**: 2025년 6월 25일
- **작성자**: Flutter + Spring Boot 개발팀
- **기술 스택**: Spring Boot 3.2+ + Flutter 3.16+ + Docker
- **인증 방식**: OAuth2 + JWT
- **배포 방식**: Docker Compose + Nginx
- **검토자**: 백엔드 아키텍트, Flutter 개발자, DevOps 엔지니어
- **다음 리뷰**: 2025년 7월 10일

본 문서는 Spring Boot 백엔드와 Flutter 프론트엔드를 기반으로 한 현장 보고 및 관리 플랫폼의 개발 및 운영을 위한 종합 가이드입니다.