로봇 서비스 API G/W 고도화
(오픈이노베이션 프로젝트)

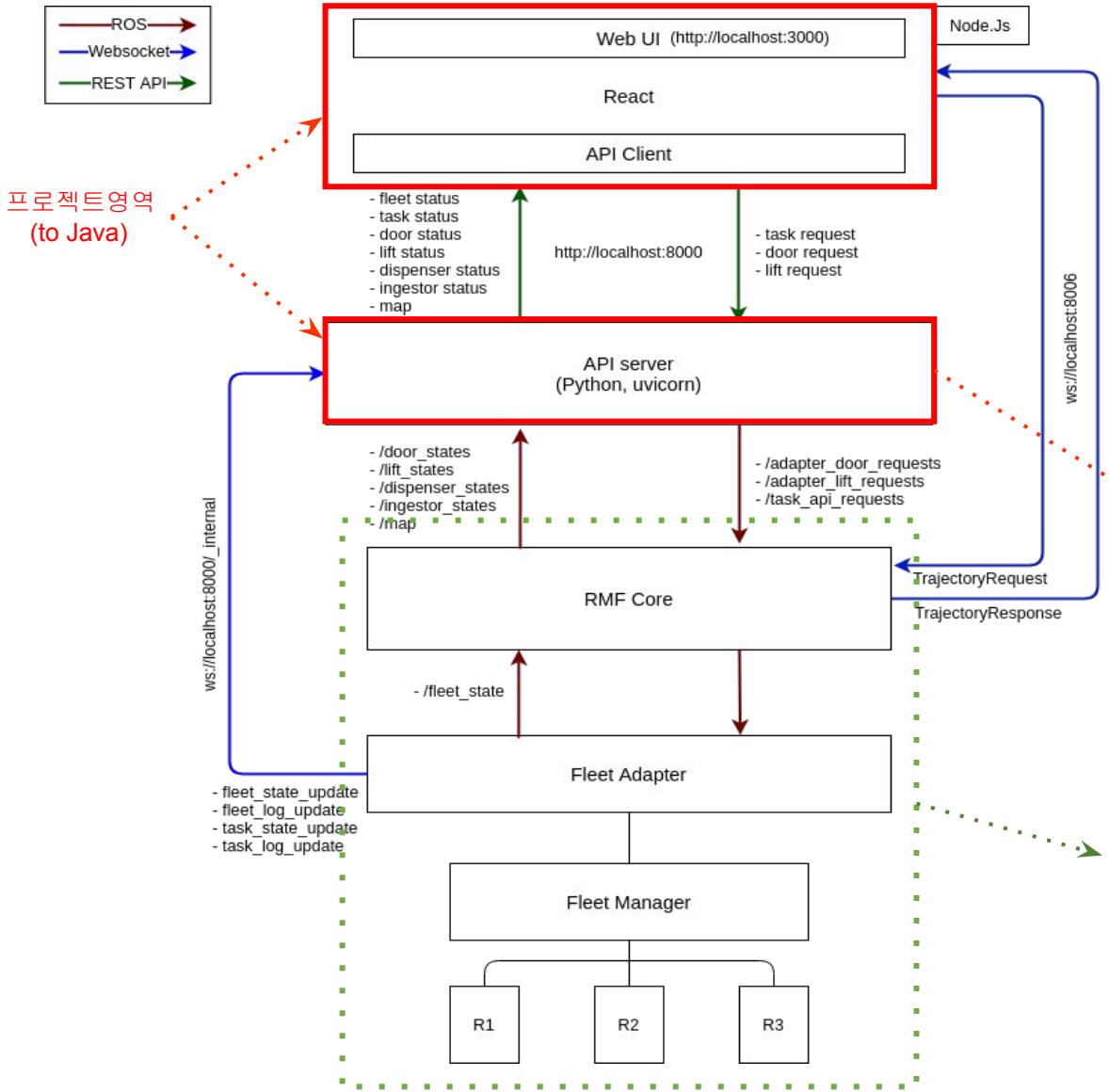
프로젝트 최종 발표

박정우

1. 프로젝트 개요

- Open-RMF

여러가지 로봇들을 관제, 제어, 모니터링 수집 및 피드백하는 시스템



오픈소스 리파지토리 참고
https://github.com/open-rmf

■ **활용 기술 스택 리스트**
ROS, Python, React
RestfulAPI
Websocket(socket.io)
SQLite

동일 기능을 Java 언어로 전환
(Spring Framework 기반)

Mock-Up 활용 또는
오픈소스 리파지토리
참고하여 환경 셋업
(시뮬레이션 가능)

1. 프로젝트 개요

- Python → Java 전환 방향성

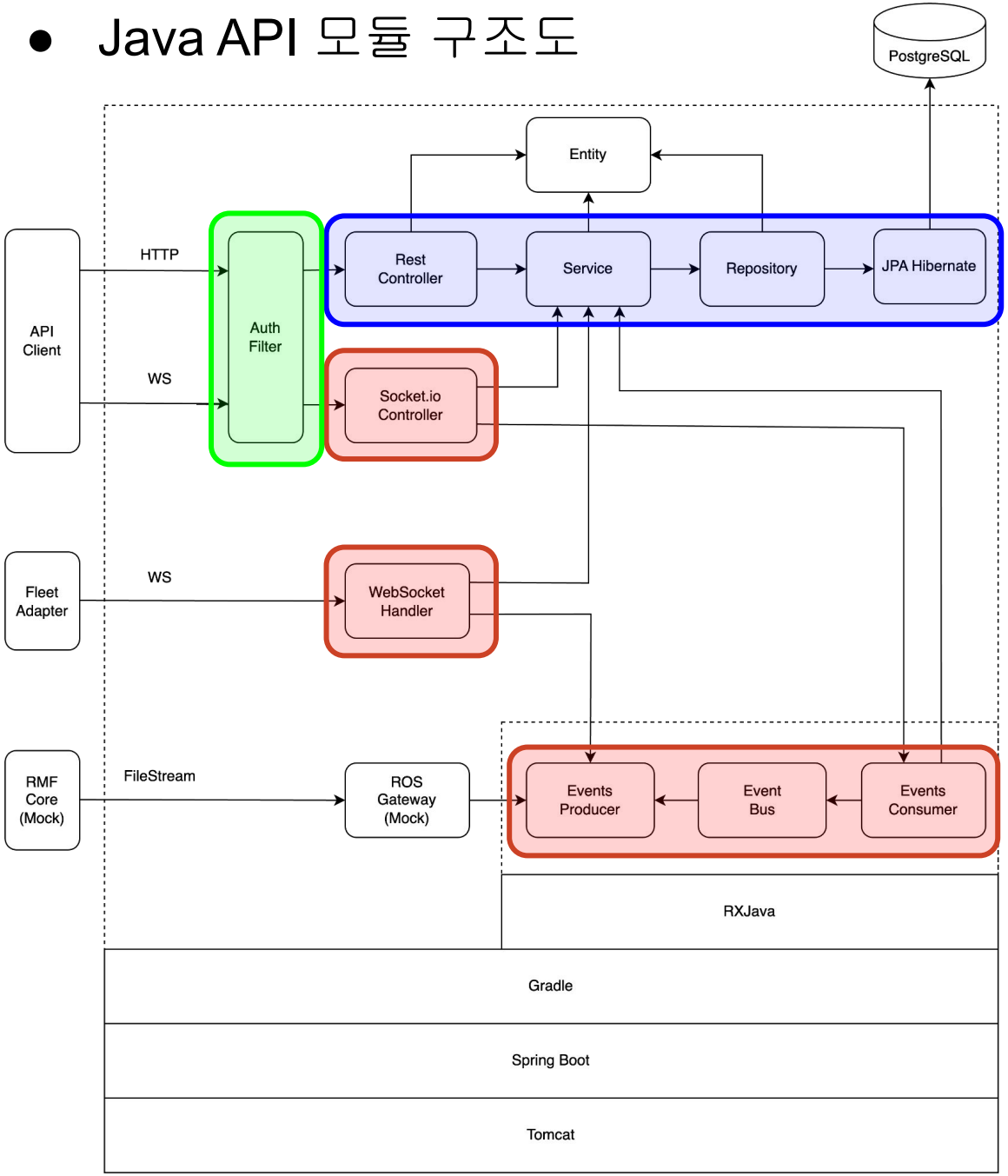
구분	Python		Java
Framework	Uvicorn	→	Spring Boot
	FastAPI		
	Pydantic		
Protocol	**WebSocket**		**WebSocket**
	socket.io		**socket.io**
ORM	Tortoise	→	JPA
Pub/Sub	**ReactiveX**		**ReactiveX**
Database	SQLite		PostgreSQL

- Java API Server 개발 범위

구분	내용	설명
아키텍처 전환 (Python → Java)	REST API	API Client에서 요청하는 REST API 구현
	WebSocket	Fleet Adapter에서 전달하는 WebSocket 데이터 처리 기능 구현
	socket.io	API Client에서 구독하는 socket.io API 구현
	ROS Mocking (선택)	RMF Core와 연결된 기능의 Mocking으로 ROS 의존성 제거
기능 추가	로그인	DB, REST API, 클라이언트 개발 및 인증과 인가 처리
	사용자 관리 (선택)	사용자 추가 및 정보 변경 기능 개발

1. 프로젝트 개요

- Java API 모듈 구조도



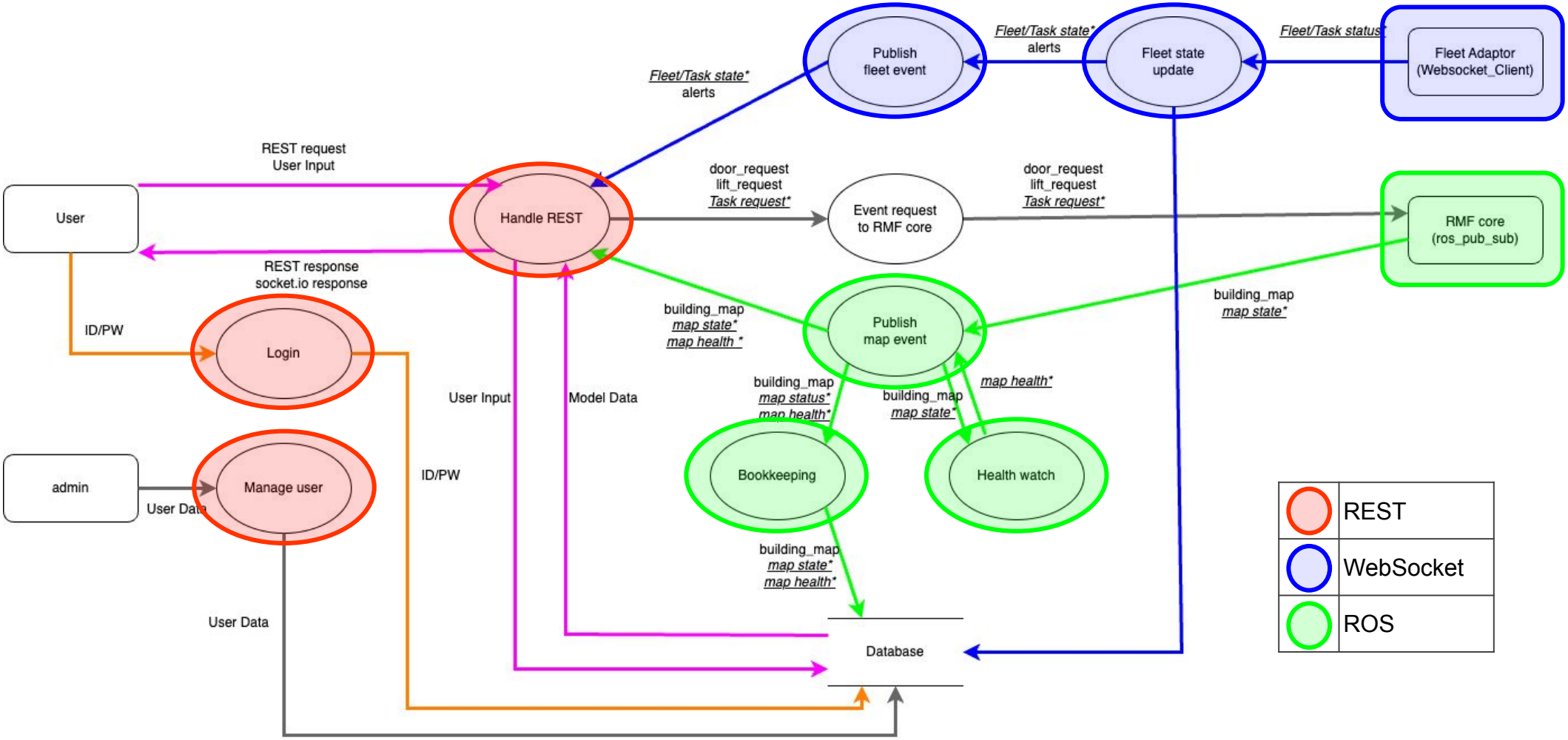
- **Controller**
- **Service**
- **Repository**
- **JPA**

- **WebSocketHandler**
- **SocketIoController**
- **RxJava**
  - EventBus
  - EventProducer
  - EventConsumer

- **Auth Filter**

1. 프로젝트 개요

- 데이터 플로우 다이어그램



2. 개발목표 대비 성과

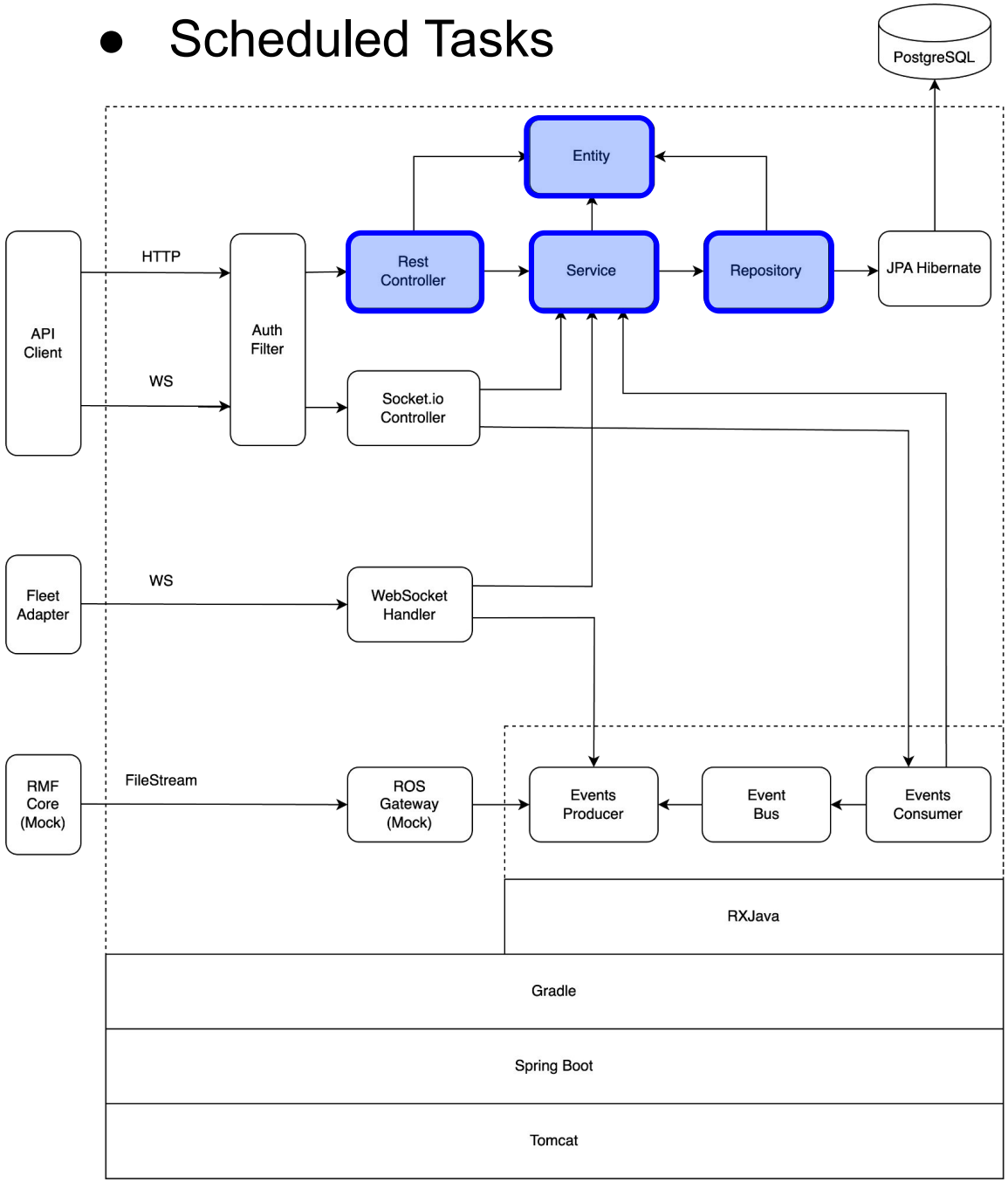
- **개발 목표**
  - Server ↔ Client (REST API)
  - Server ↔ Client (socket.io)
  - Fleet Adapter → Server (WebSocket)
  - RMF-Core → Server (ROS)
    - BookKeeper
    - HealthWatchDog
  - 로그인/로그아웃 화면 및 기능
  - Server → RMF-Core (ROS)
  - 사용자 관리 화면 및 기능
    - 사용자 추가
    - 패스워드 변경
    - 이메일 변경
    - 사용자 권한 기능 (Roles, Permissions)

- **성과**
  - Server ↔ Client (REST API)
  - Server ↔ Client (socket.io)
  - Fleet Adapter → Server (WebSocket)
  - RMF-Core → Server (ROS)
    - BookKeeper
    - HealthWatchDog
  - 로그인/로그아웃 화면 및 기능
  - 사용자 관리 화면 및 기능 *
    - 사용자 추가 *
    - 패스워드 변경 *

- **미흡**
  - Server → RMF-Core (ROS)
  - 사용자 관리 화면 및 기능
    - 이메일 변경
    - 사용자 권한 기능 (Roles, Permissions)

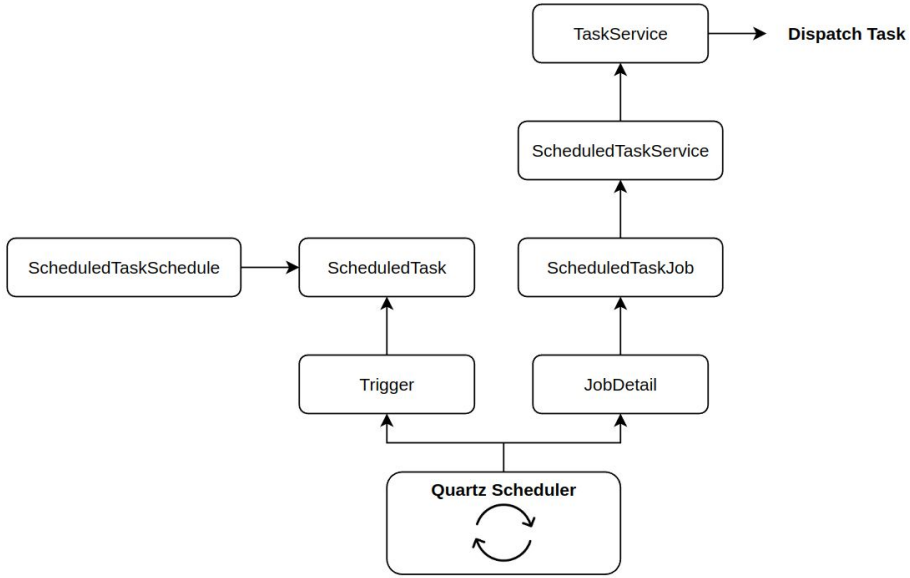
3. 개발 상세

- Scheduled Tasks



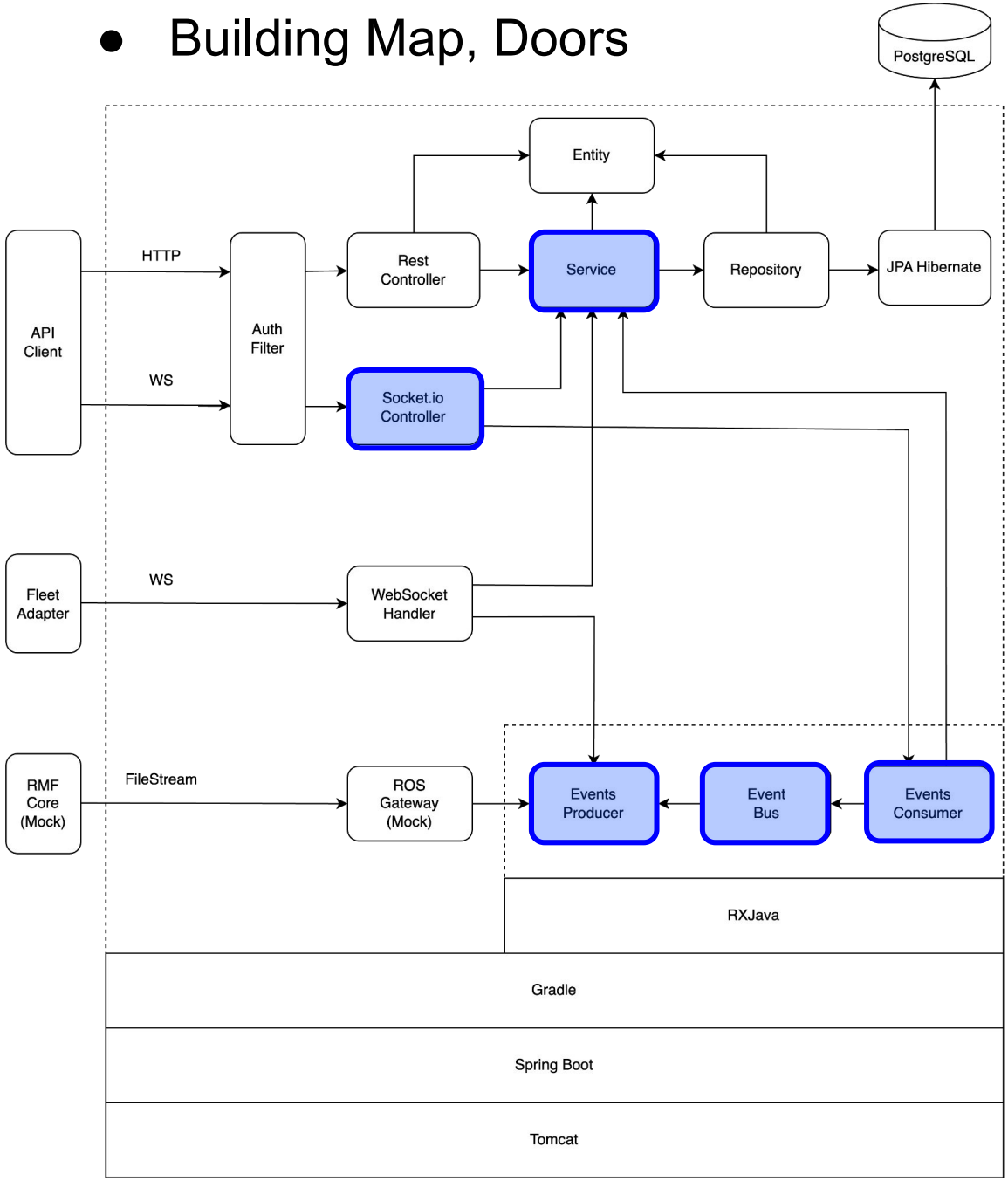
- REST API
  - GET /scheduled_tasks
  - POST /scheduled_tasks
  - GET /scheduled_tasks/{task_id}
  - DELETE /scheduled_tasks/{task_id}
    - 전체 schedule 삭제
  - PUT /scheduled_tasks/{task_id}/clear
    - 특정 날짜 단일 schedule 삭제
  - POST /scheduled_tasks/{task_id}/update
    - 특정 날짜의 단일 schedule update
    - 모든 날짜에 대한 schedule update

- 스케쥴러
  - Spring Quartz 라이브러리 활용



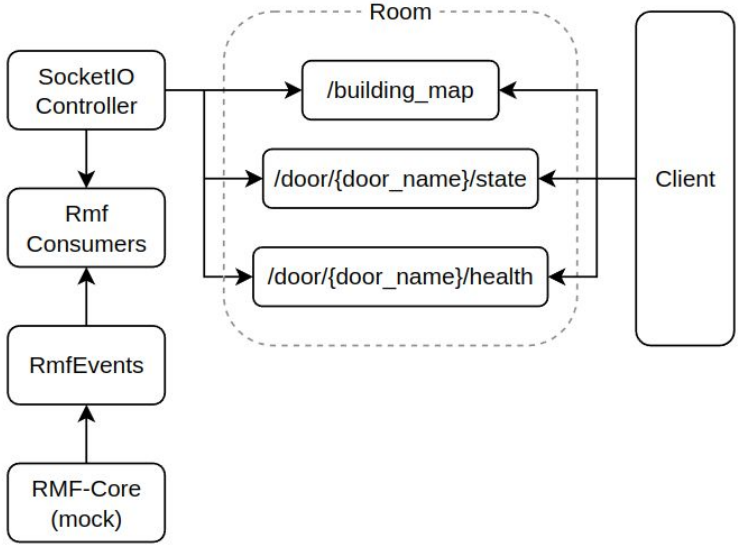
3. 개발 상세

- Building Map, Doors



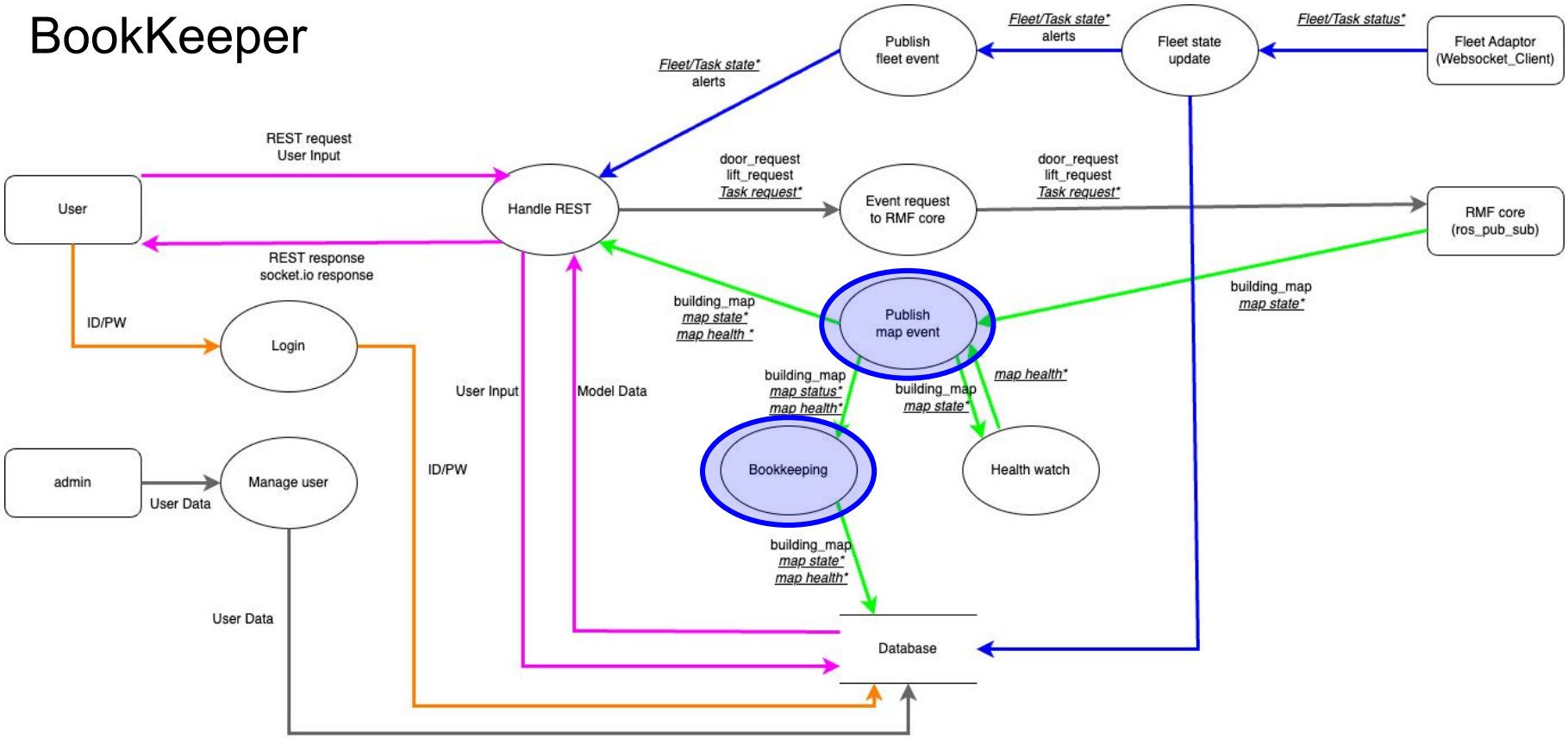
- REST API
  - GET /building_map
  - GET /doors
  - GET /doors/{door_name}/health
  - GET /doors/{door_name}/state

- socket.io API
  - /building_map
    - subscribe시 초기 맵 데이터 전송
  - /doors/{door_name}/state
  - /doors/{door_name}/health

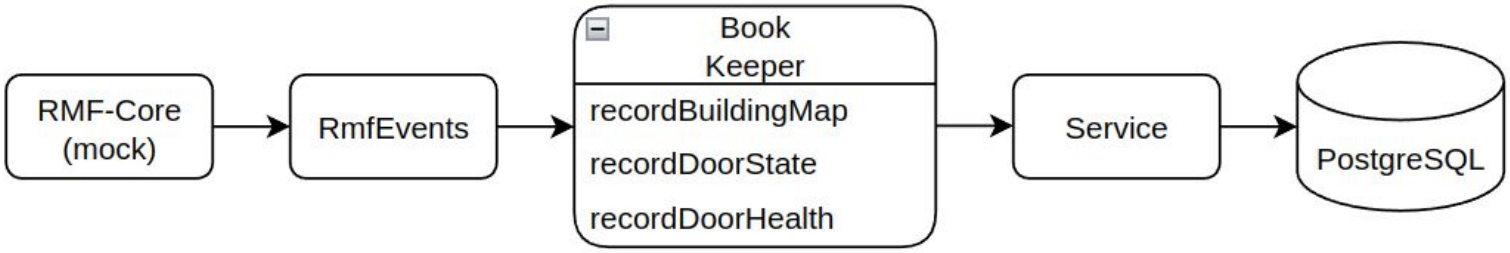


3. 개발 상세

- BookKeeper

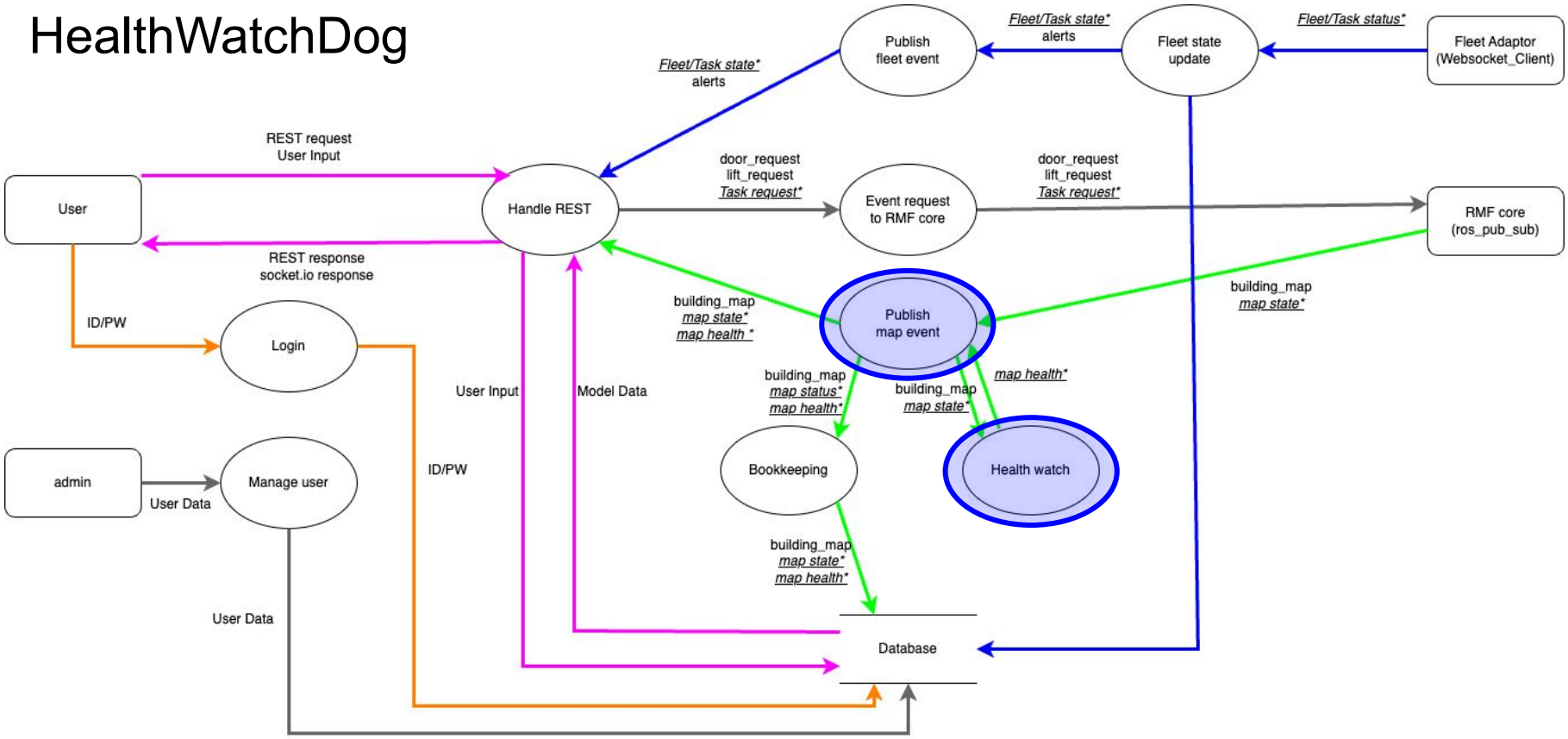


- BookKeeper
  - BuilingMap
  - DoorState
  - DoorHealth

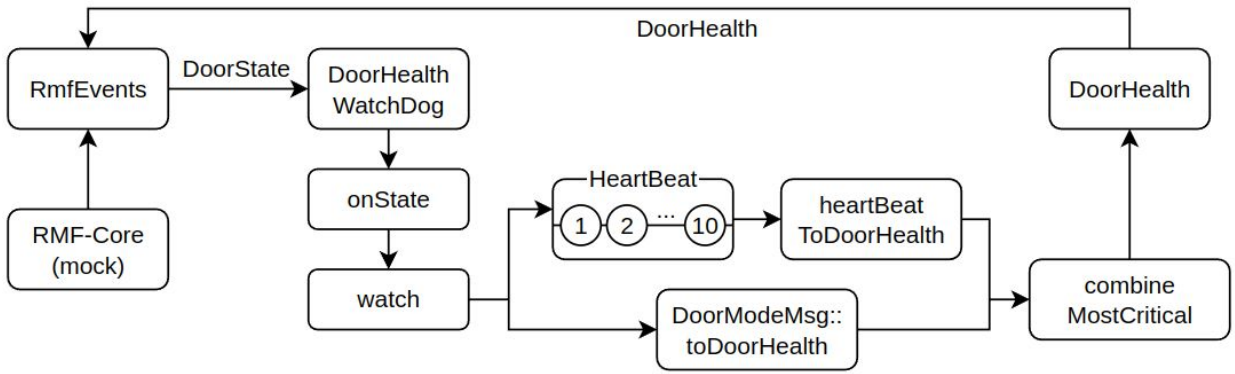


3. 개발 상세

- HealthWatchDog

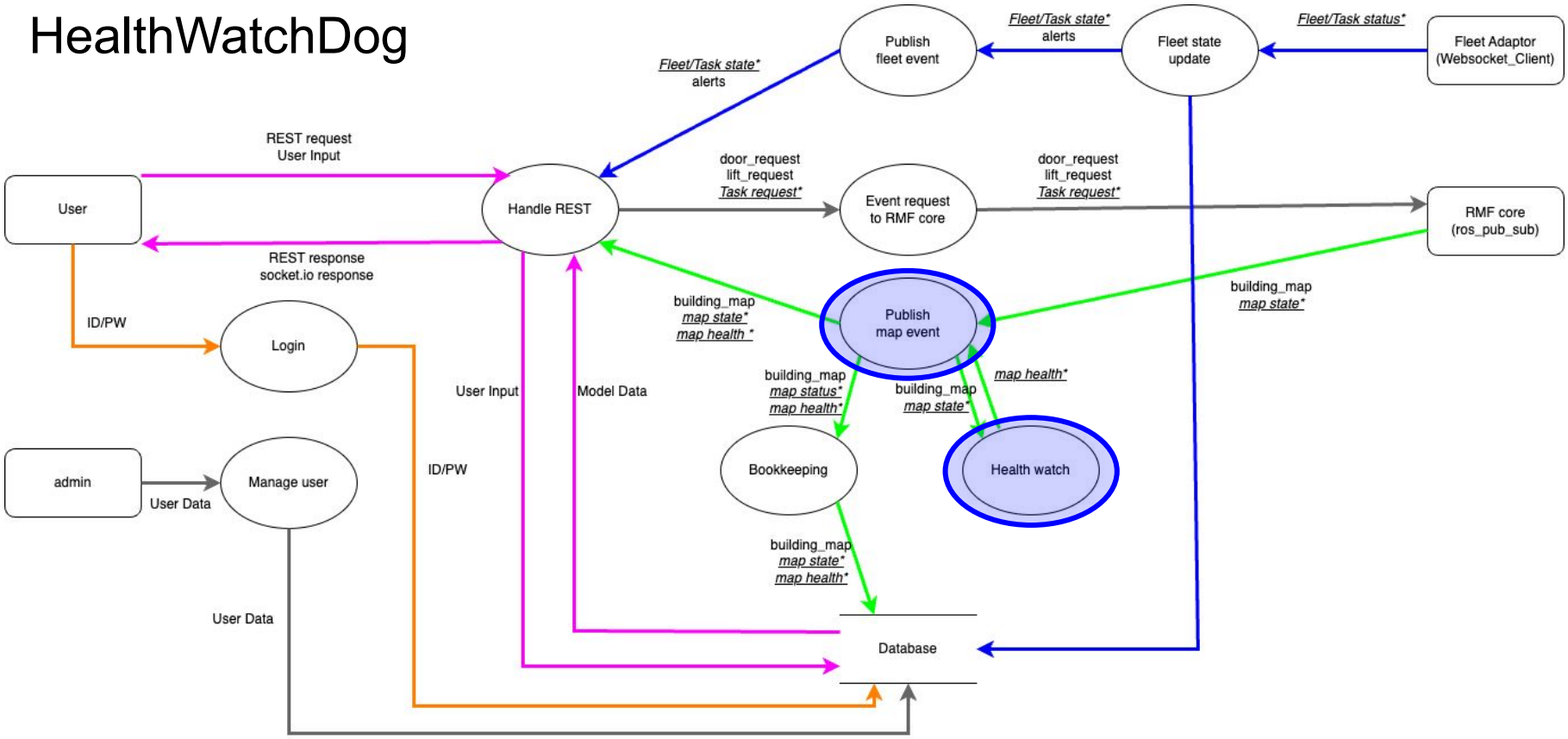


- DoorHealthWatchDog
  - BuildingMap 초기 데이터에서 door를 조회하여 subjects로 관리
  - 발행된 DoorState의 Health와 HeartBeat Buffer의 Health 값을 비교

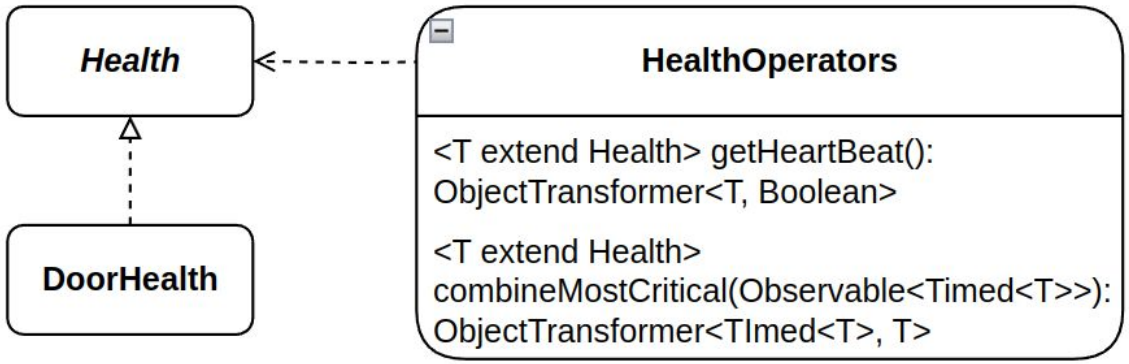


3. 개발 상세

- HealthWatchDog

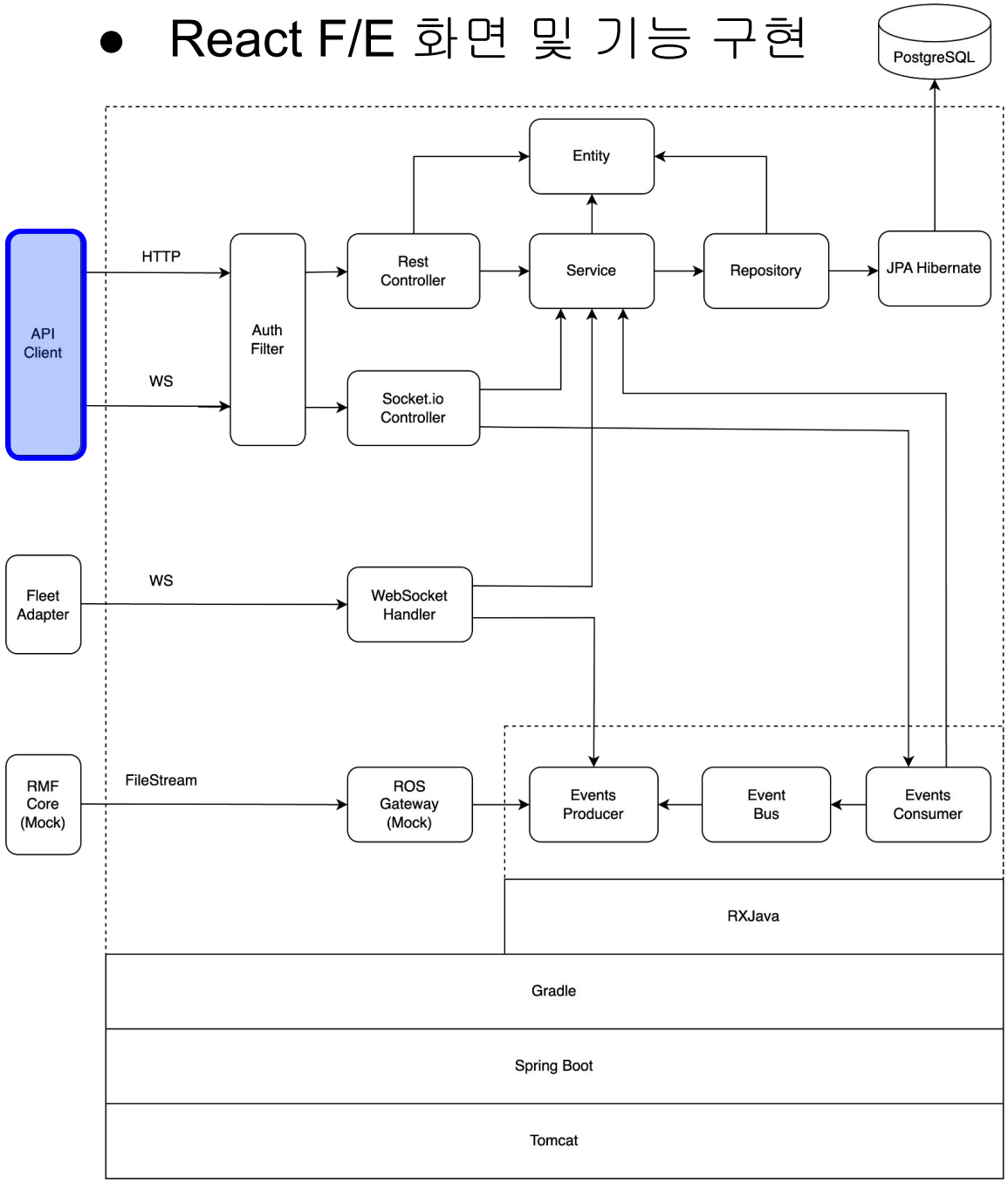


- HealthOperators
  - DoorHealth 등 Health 관련 엔티티가 Health 추상 클래스를 구현하도록 작성
  - 제네릭을 활용하여 HealthOperators를 재사용할 수 있도록 작성



3. 개발 상세

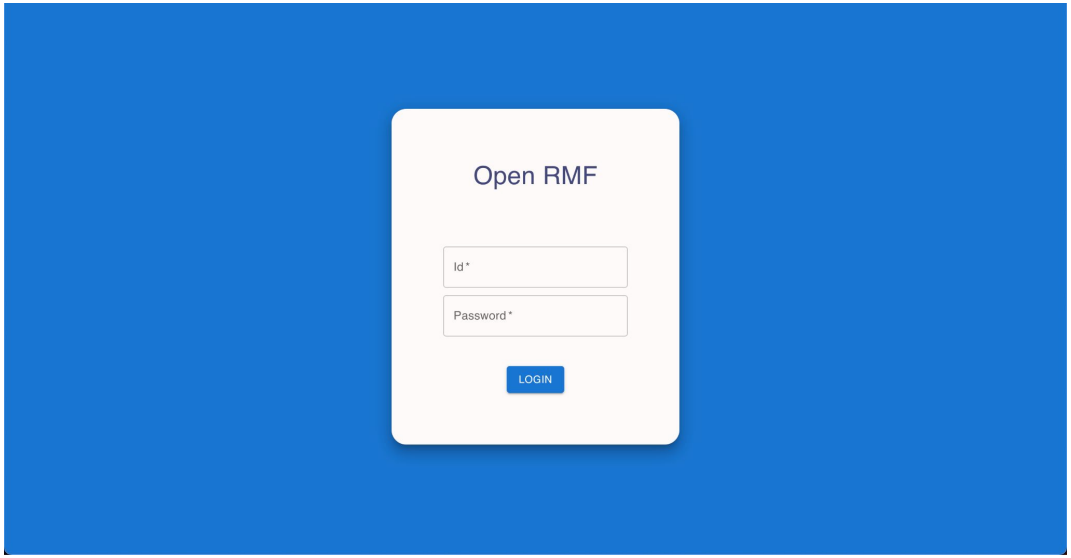
- React F/E 화면 및 기능 구현

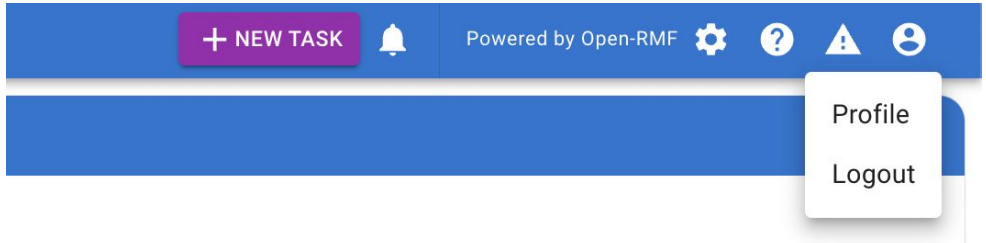


- API 테스트를 위한 화면 및 기능 구현
  - 로그인화면
  - 프로필, 로그아웃 버튼
  - 사용자 생성 화면
  - 사용자 정보 및 수정 화면
    - 비밀번호 변경

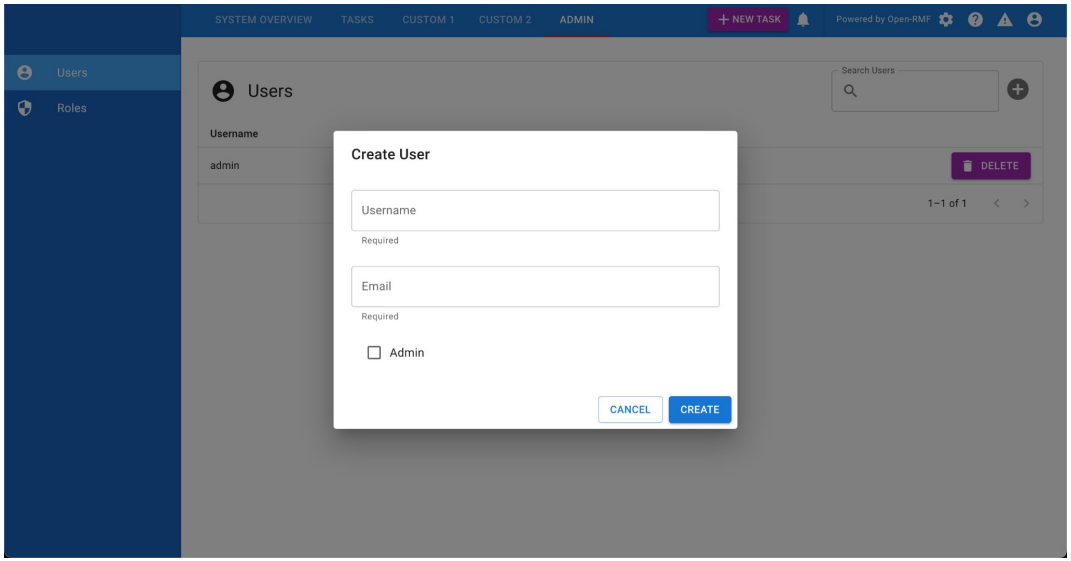
3. 개발 상세

- React F/E 화면 및 기능 구현





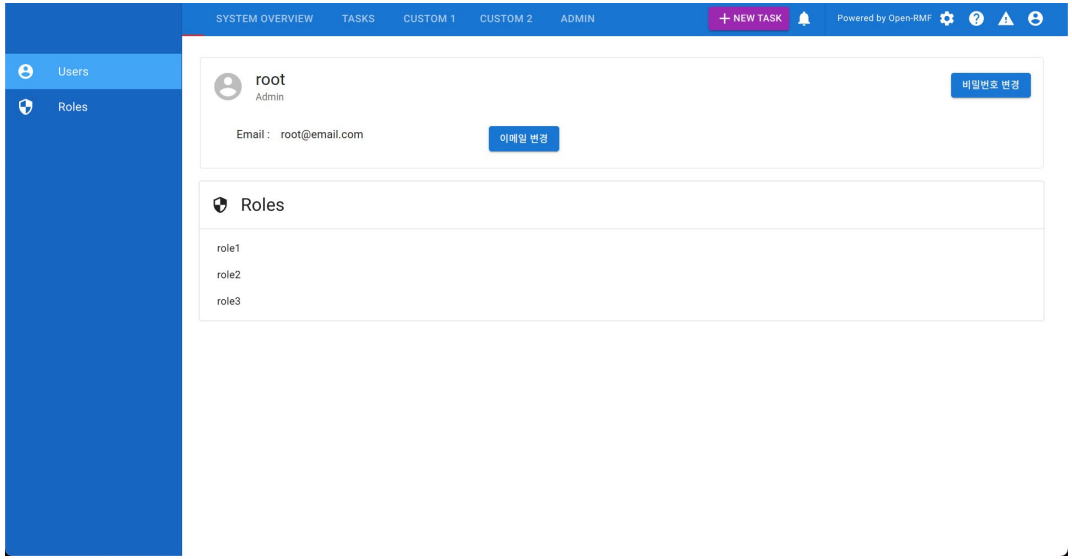
로그인 / 로그아웃



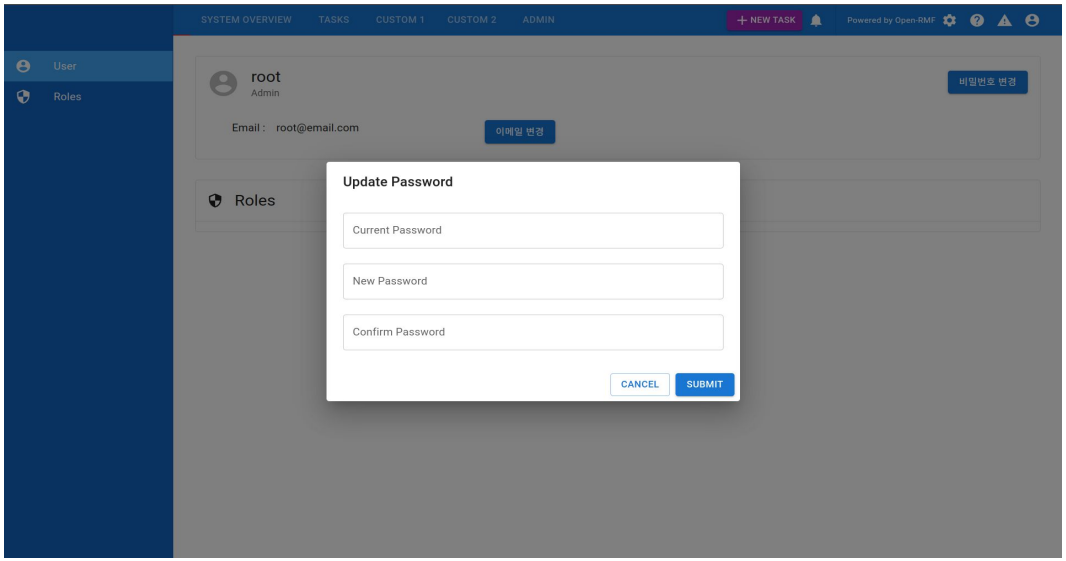
사용자 생성

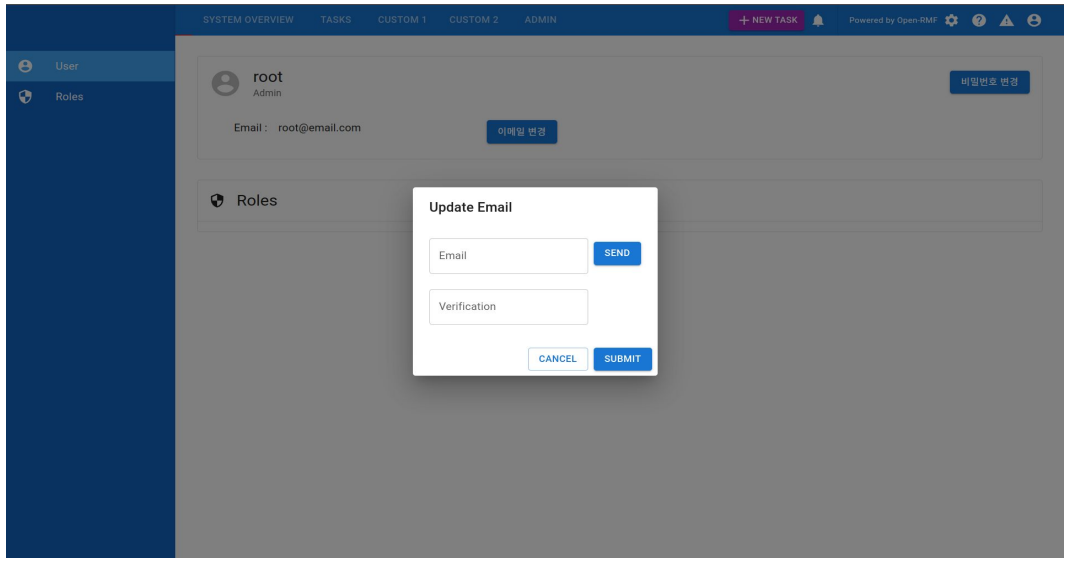
3. 개발 상세

- React F/E 화면 및 기능 구현



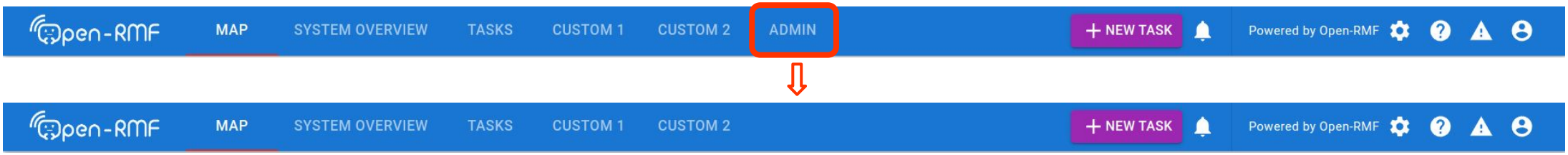
사용자 정보 및 수정



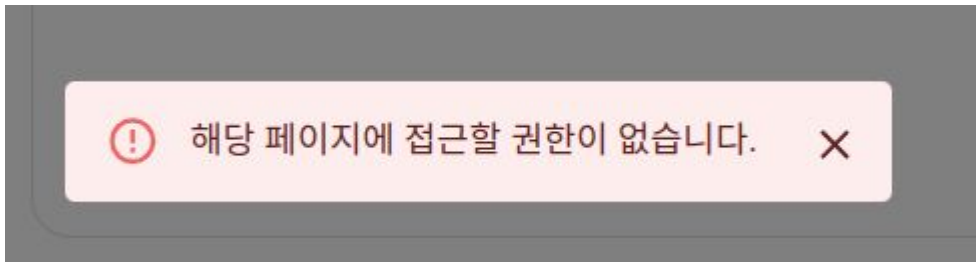


3. 개발 상세

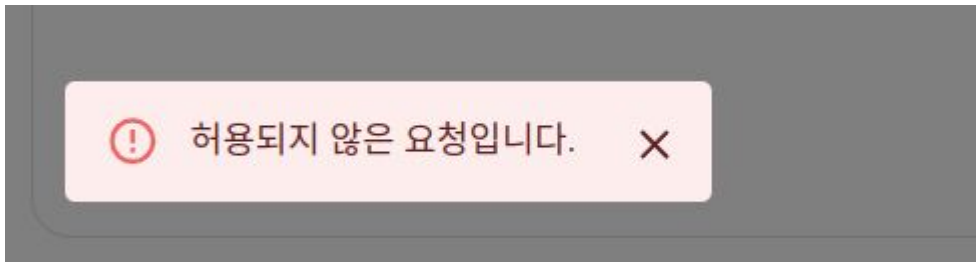
- React F/E 화면 및 기능 구현
  - admin 사용자인 경우에만 상단 AppBar에서 ADMIN 탭 보이도록 설정



  - admin router에서 로그인된 사용자에 대한 admin 권한 확인 및 리다이렉션



  - axios interceptors를 추가하여 에러코드 401/403 인 경우 메인 페이지로 리다이렉션



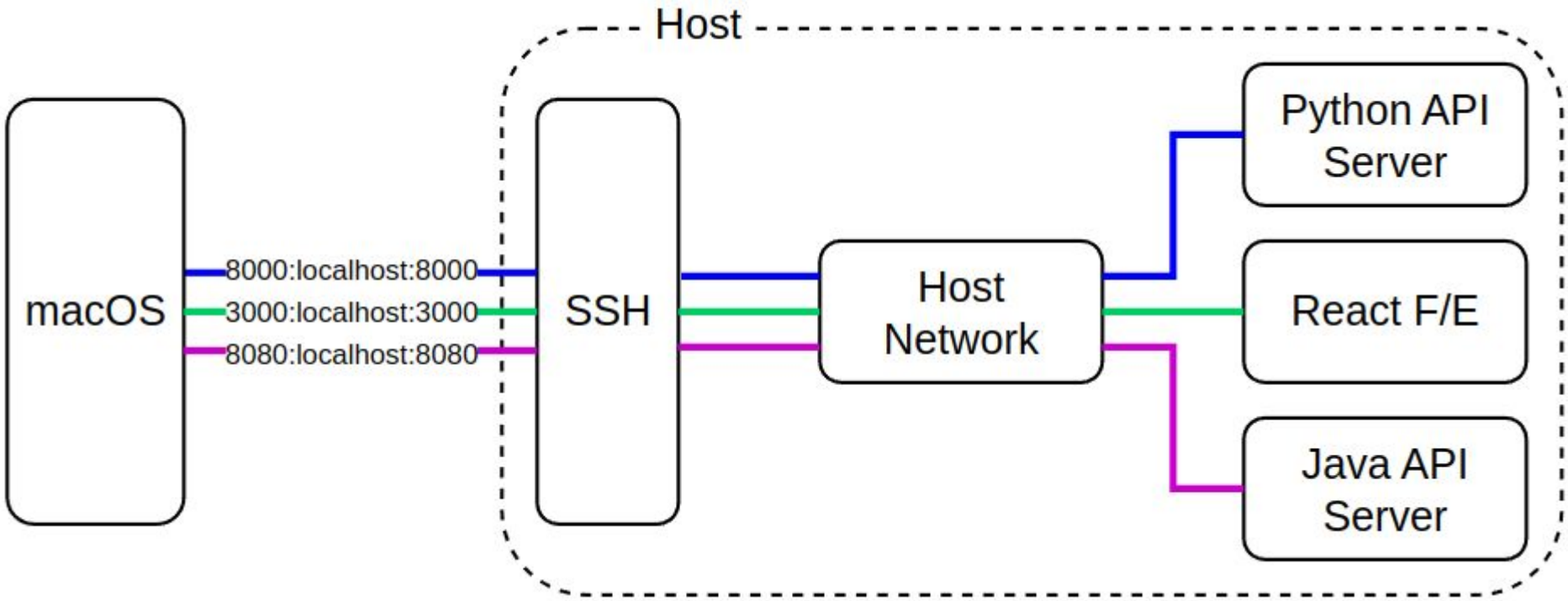
4. 이슈사항 및 해결과정

- Docker 호스트 모드 네트워크

  도커 네트워크가 **Host Mode**로 설정

  - 클라이언트에서 서버 요청시 **localhost** 사용
  - macOS에서 Host Mode 미지원

  → **SSH Tunneling을 사용하여 외부에서 localhost 접근**



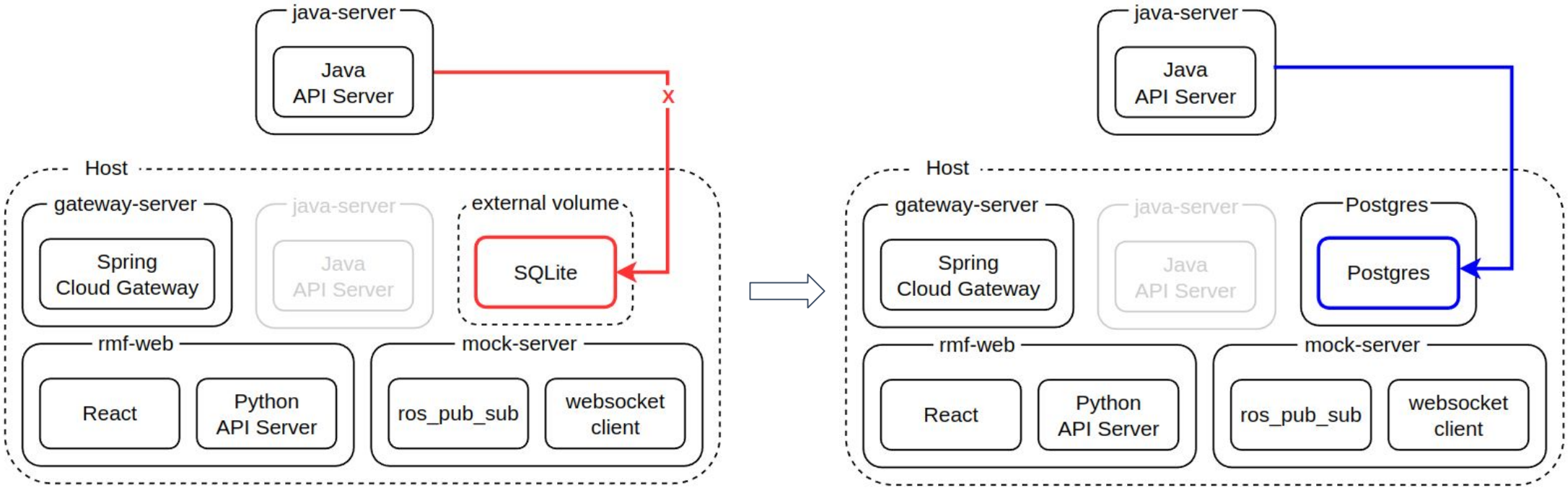
4. 이슈사항 및 해결과정

- SQLite → PostgreSQL

  SQLite 사용시 데이터베이스가 파일 형태(.db3)로 저장

  - External Volume을 사용하여 컨테이너 간 DB 공유
  - 외부에서 External Volume 접근 불가

  → **PORT를 통해 접근하는 MySQL/PostgreSQL로 변경**



4. 이슈사항 및 해결과정

- Spring Quartz 라이브러리 통합

  Spring Scheduler → 동적 스케쥴링 기능 부족

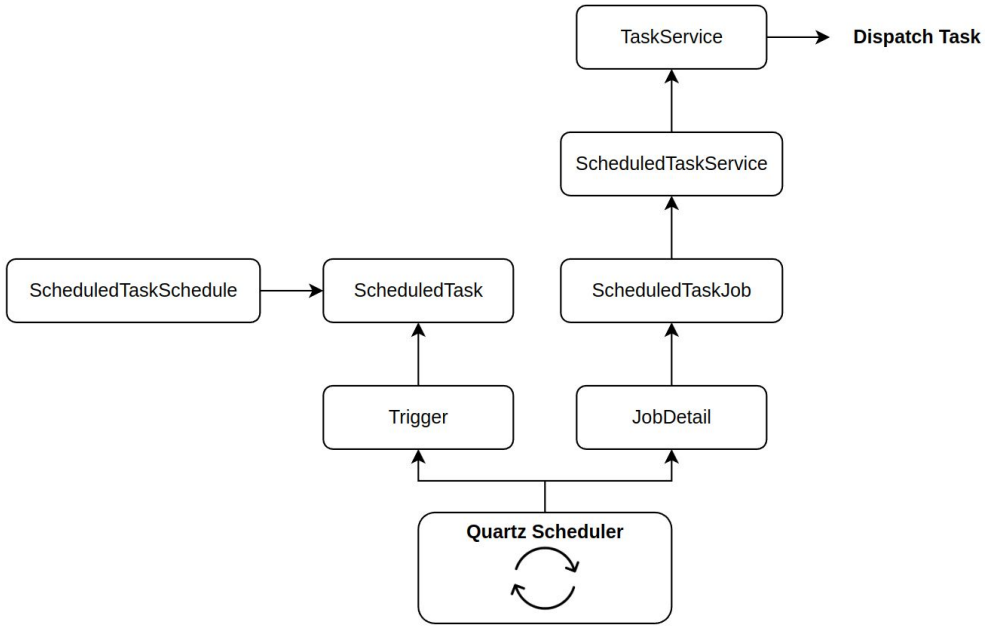
  Spring Quartz 라이브러리 활용

  - 동적 스케쥴러 관리 용이
  - Trigger, JobDetail과 같은 개념이 Python Scheduler와 유사
  - Autowiring 기능 부족, Service 인스턴스 의존성 주입 불가

  → **Spring에서 지원하는 SpringBeanJobFactory를 상속받아 autowireBean 설정 추가**

```java
@Bean  yhames
public SchedulerFactoryBean schedulerFactoryBean() {
    SchedulerFactoryBean schedulerFactoryBean = new SchedulerFactoryBean();
    schedulerFactoryBean.setJobFactory(autowiringSpringBeanJobFactory);
    return schedulerFactoryBean;
}
```

```java
@Component  yhames
@RequiredArgsConstructor
public static class AutowiringSpringBeanJobFactory extends SpringBeanJobFactory {

    private final AutowireCapableBeanFactory beanFactory;

    @Override  1 usage  yhames
    protected Object createJobInstance(final TriggerFiredBundle bundle) throws Exception {
        final Object job = super.createJobInstance(bundle);
        beanFactory.autowireBean(job);  // Job 클래스에 의존성 주입
        return job;
    }
}
```



4. 이슈사항 및 해결과정

- Spring Quartz 라이브러리 통합

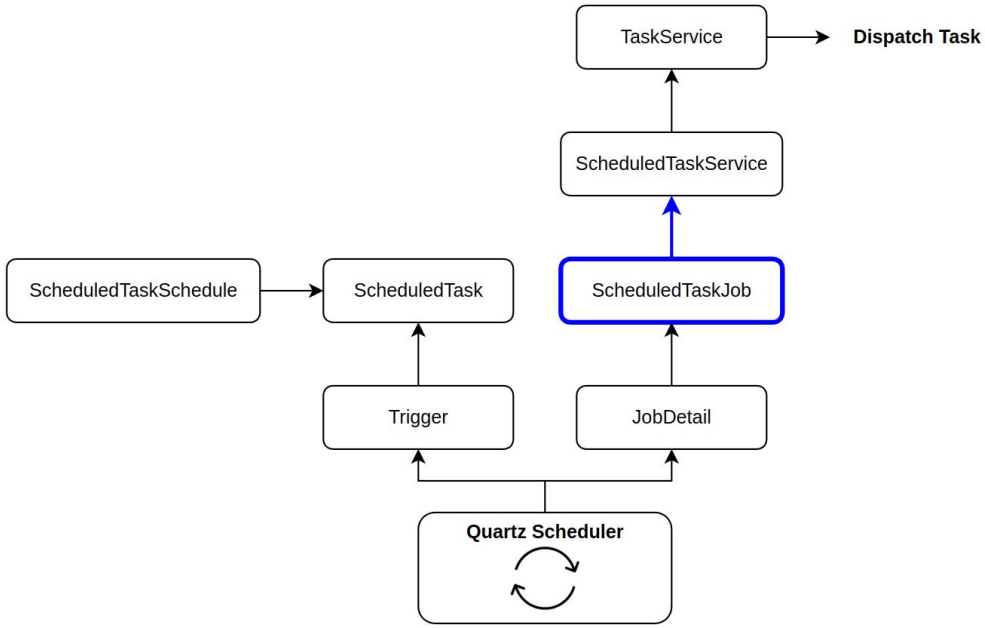
  Spring Scheduler → 동적 스케쥴링 기능 부족

  Spring Quartz 라이브러리 활용

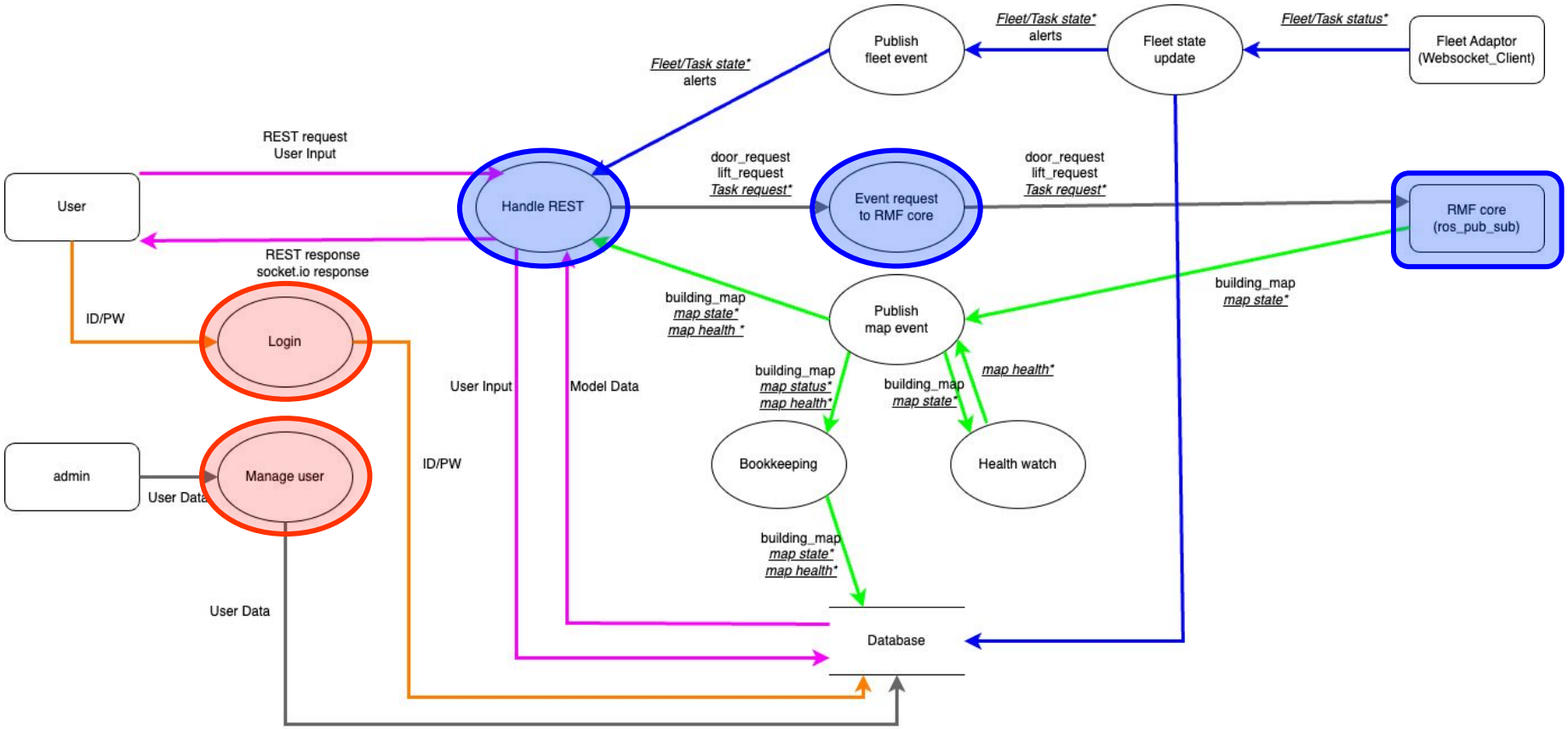
  - 동적 스케쥴러 관리 용이
  - Trigger, JobDetail과 같은 개념이 Python Scheduler와 유사
  - Autowiring 기능 부족, Service 인스턴스 의존성 주입 불가

  → **Spring에서 지원하는 SpringBeanJobFactory를 상속받아 autowireBean 설정 추가**

```java
@Component
@RequiredArgsConstructor
public class ScheduledTaskJob implements Job {

    public static final String KEY = "SCHEDULED_TASK_KEY";  2 usages

    public static final String GROUP = "SCHEDULED_TASK_GROUP";  4 usages

    private final ScheduledTaskService scheduledTaskService;    // 의존성 주입

    @Override  yhames
    public void execute(JobExecutionContext jobExecutionContext) {
        JobDataMap jobDataMap = jobExecutionContext.getJobDetail().getJobDataMap();
        int scheduledTaskId = jobDataMap.getIntValue(ScheduledTaskJob.KEY);
        scheduledTaskService.dispatchScheduledTask(scheduledTaskId);
    }
}
```



5. 향후 계획



- 미흡사항 보완
  - 이메일 인증 및 변경
  - 사용자 권한 기능 (Roles, Permissions)
- ROS 통신을 위한 Gateway 구현
  - Door Request
  - Task Request
  - MQTT 활용

6. Lessons Learned

- 개발 프로세스 준수
  - \+ 문서화를 통한 원활한 커뮤니케이션 → 낭비되는 시간 없이 개발에 집중
  - \- 결함관리와 공유 미흡, 예상치 못한 문제 상황 발생
  - \- 하루 12시간 이상 투자, 일정과 리소스 관리 미흡

  → **더 명확한 분석/설계와 엄격한 문서화 필요**

- 통합 및 단위 테스트 코드 작성
  - \+ 명확한 커버리지를 설정 → 구현과 테스트를 적절히 분배
  - \+ 짧은 개발 기간에도 코드 변경에 대한 신뢰성을 높임
  - \- 효율적인 테스트 코드 작성 미흡, 가독성 저하

  → **효율적인 테스트 코드 작성을 위한 고민이 필요**

- 기술적인 다양성과 숙련도의 필요성
  - \+ socket.io, RxJava, Quartz, React 등 다양한 언어와 프레임워크 활용
  - \- 기술적인 숙련도가 낮아서 성능면에서 비효율적인 코드 작성
  - \- 기본적인 DB설계, JPA, Security 부분에 대한 경험 부족

  → **기술적인 고도화 및 경험이 필요**