

WEB-GIS

인공위성 영상 표출 시스템

목차

- 프로젝트 설명
- 로드맵 설명
- 설계 내용
- 개발 내용
- 개선해야 할 점

프로젝트 목표

목표

인공위성 영상 표출용 Web-GIS 개발

회사 자체 개발한 큐브 인공위성이 찍은 영상을 처리한 LEVEL-1, LEVEL-2 영상을 Web으로 표출.

Solution을 사용하지 않고 기술 구현

영상을 표출해주는 Solution을 이용할 시 연간 1억의 고정비용이 발생. 오픈 소스를 통해 기술 구현이 필요.

프로젝트 로드맵

프로젝트 로드맵

기술 및 요구사항
분석

프로젝트 설계

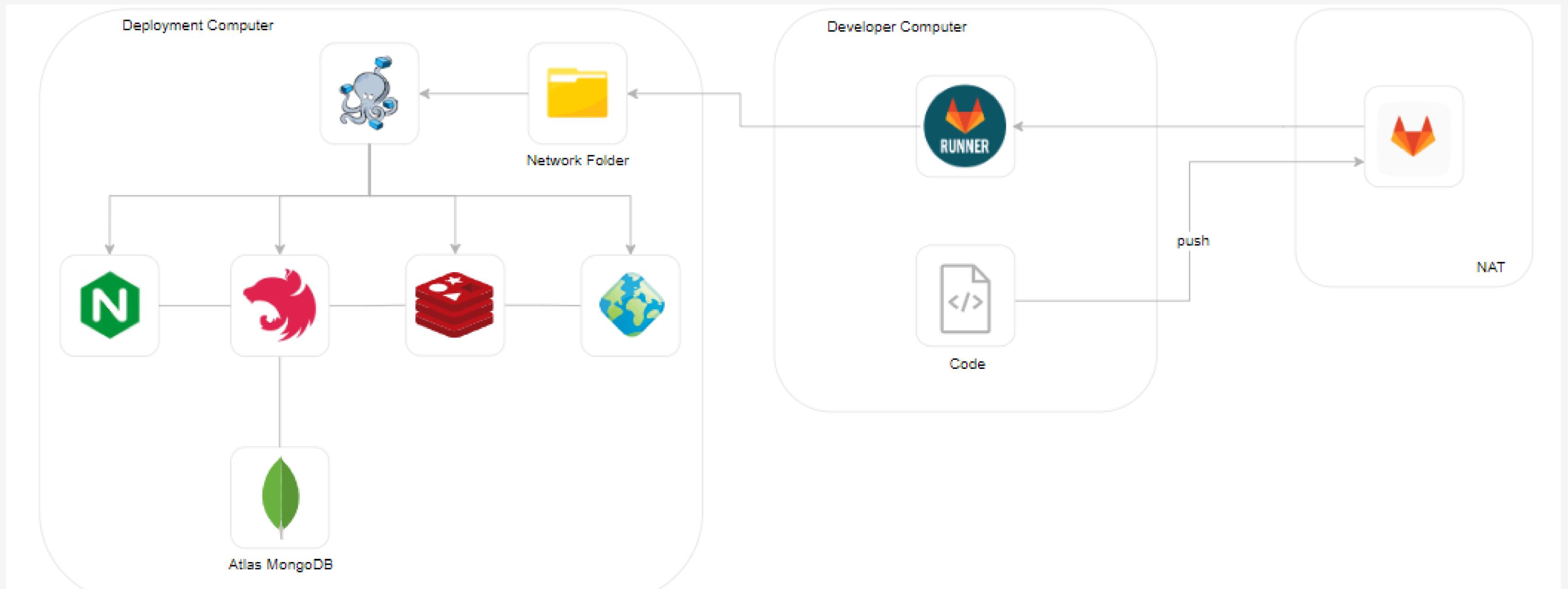
개발 및 환경 구축



프로젝트 설계

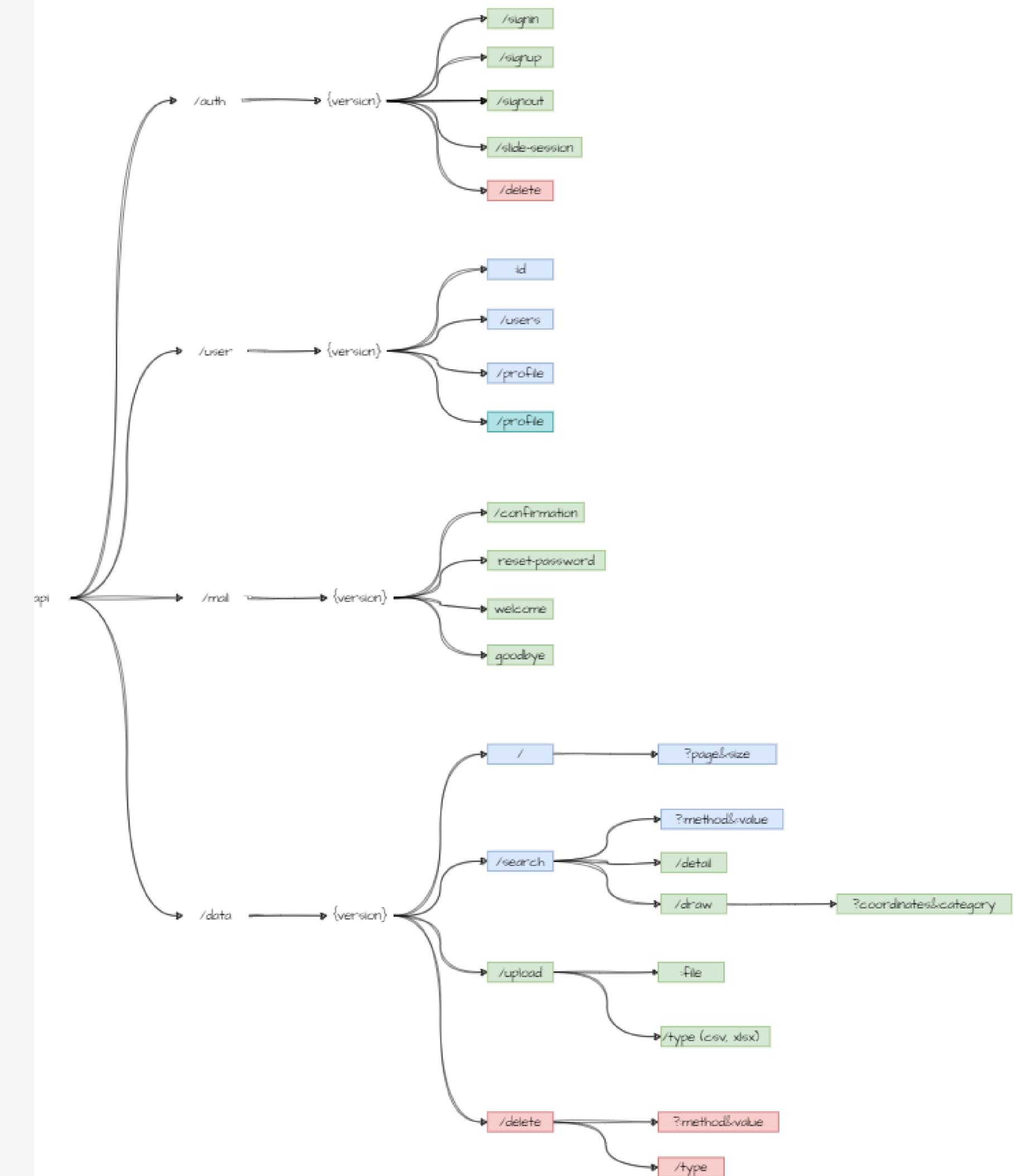
아키텍처

기술 스택: Node.js, Nest.js, NginX, MongoDB, Docker, GitLab



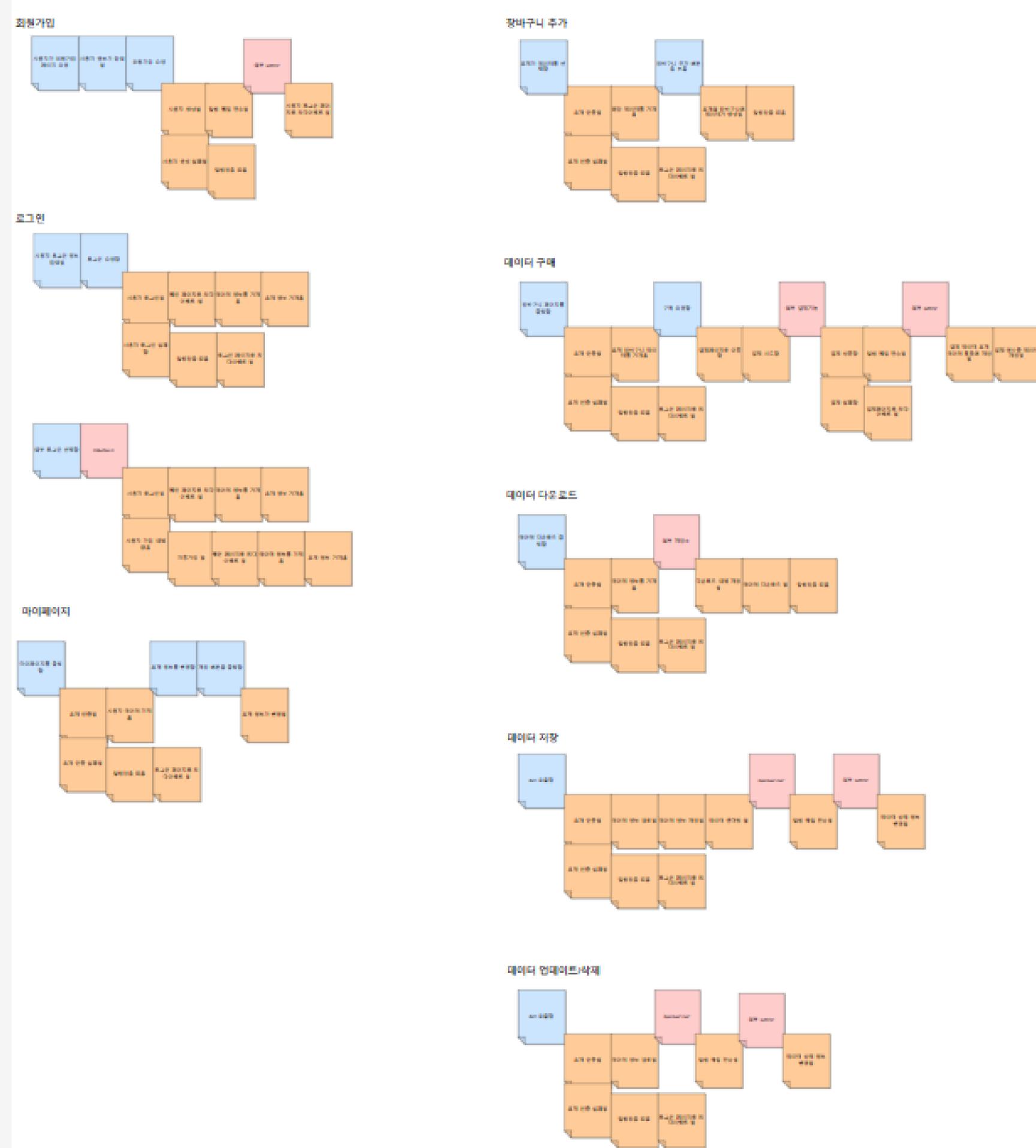
API 작성

프로토타입 프로젝트 기반으로 요구사항을
만족시킬 수 있는 API 들을 작성



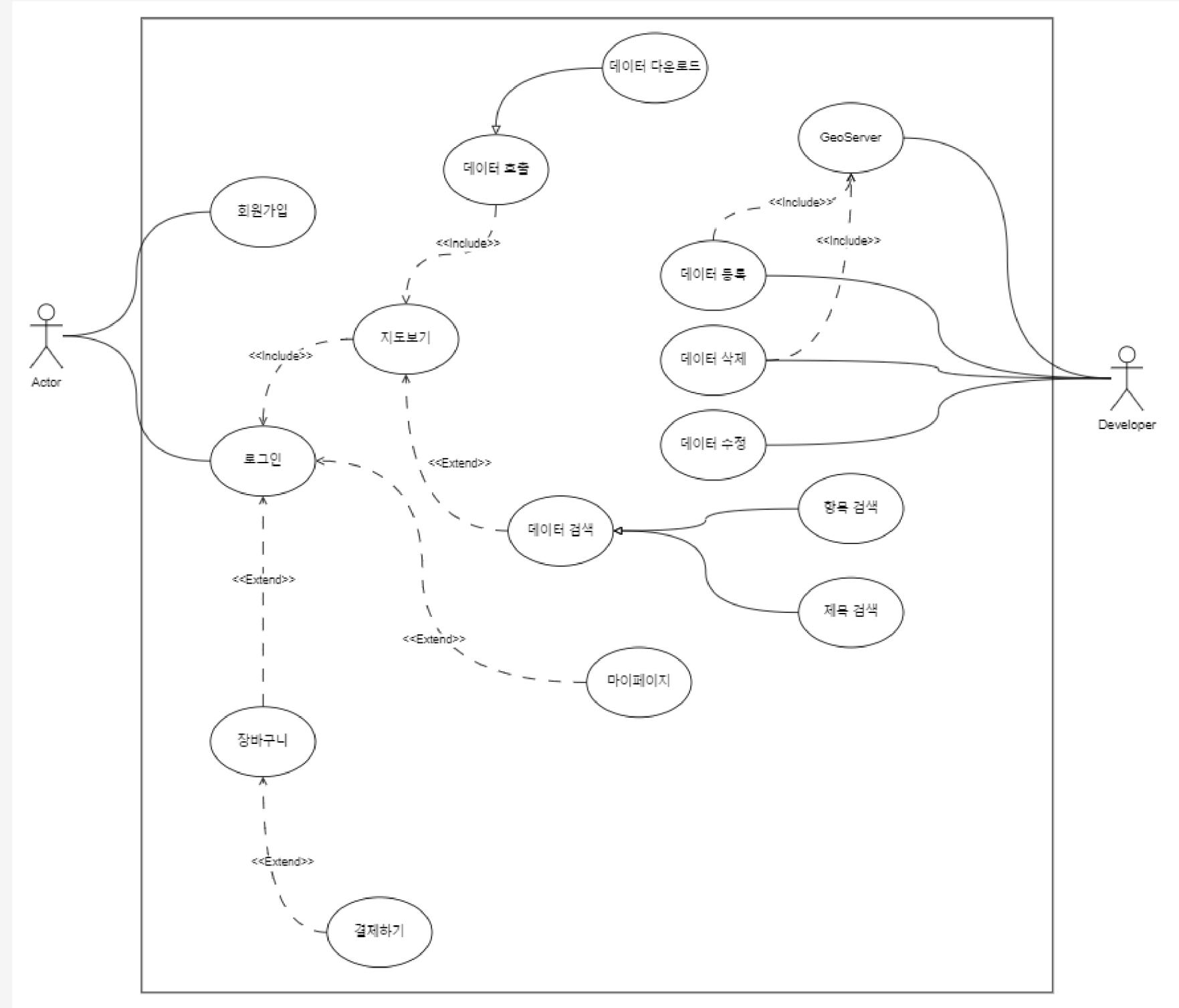
이벤트 스토밍

각 도메인에 대한 이해와 격리성을 위한 이
벤트 스토밍 도구 사용.



Use Case DiaGram

유저의 사용 패턴을 분석하기 위한 Use Case 다이어 그램 작성



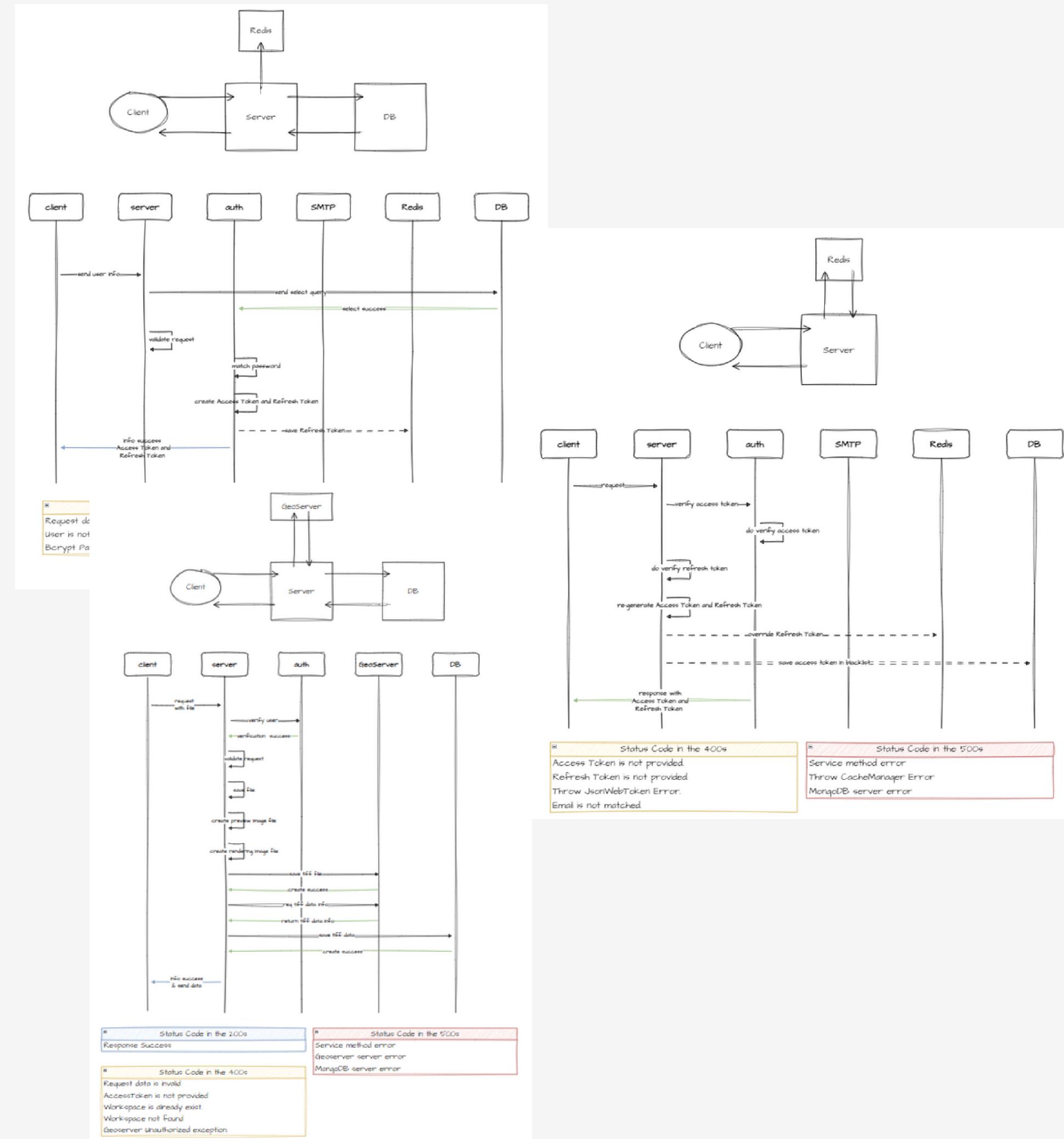
프로젝트 개발

Sequence Diagram

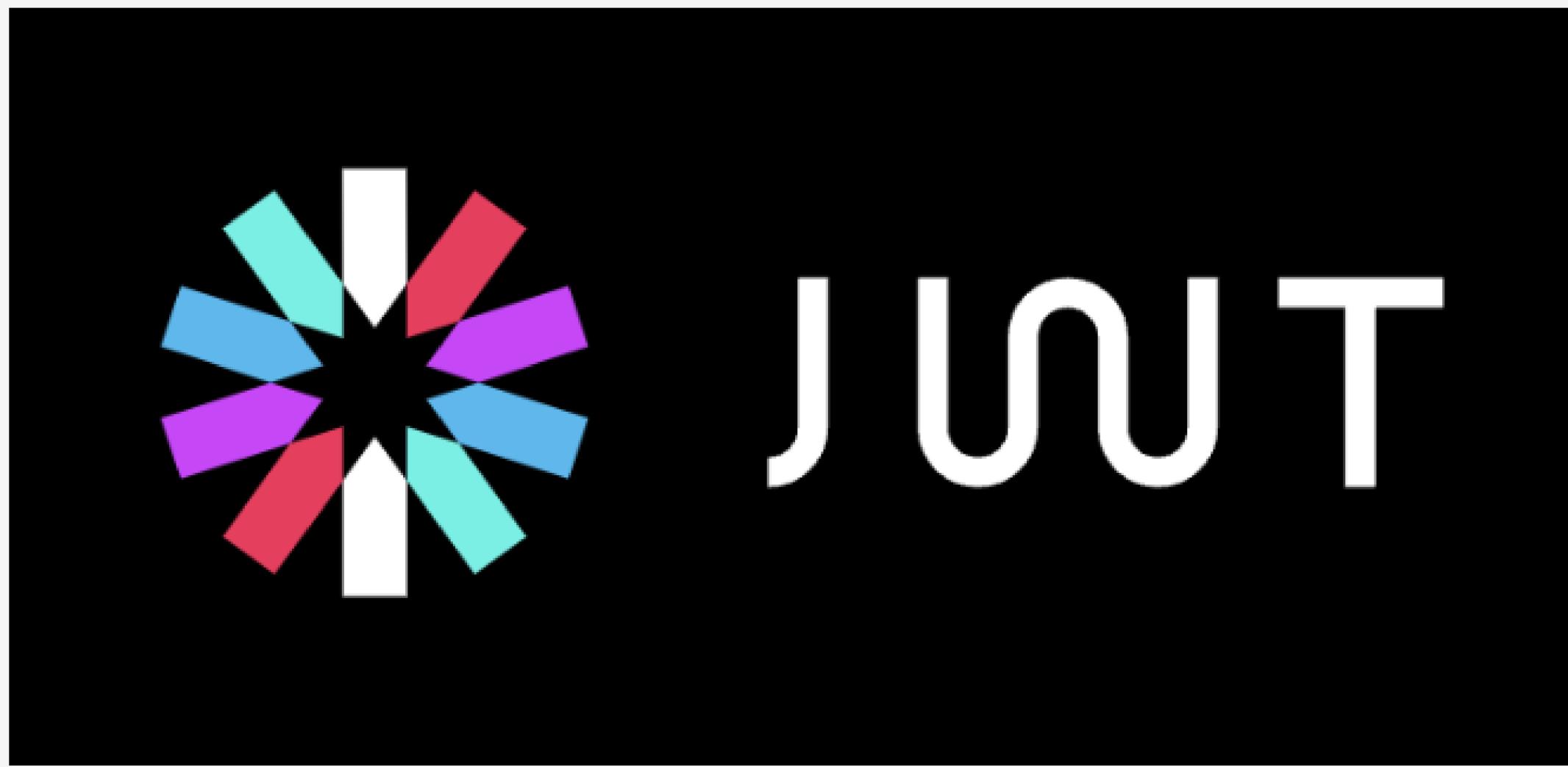
기반 로직 작성

주니어 개발자라는 단점을 보완하기 위해 매 서비스 로직을 구현할 때마다 시퀀스 다이어그램을 작성.

최대한 빠트릴 수 있는 에러 부분들을 반복적으로 고민할 수 있는 툴로서 다이어그램을 사용.



JWT 인증 구현



JWT 을 통한 인증 로직 구현.

Slide-session 처리를 하기 위한 Refresh Token 적용.

Roles 을 구현하여 인가 기능 구현.
(ADMIN, USER)

Swagger 작성

API를 테스트 할 수 있는 문서 작성.

프론트엔드와 수월한 소통을 하기 위한 문서화 수행.

The screenshot shows a Swagger UI interface with three main sections: **Auth**, **User**, and **Mail**.

- Auth** section:
 - POST /api/auth/v1/signup
 - POST /api/auth/v1/signin
 - POST /api/auth/v1/slidesession (locked)
 - POST /api/auth/v1/signout
 - DELETE /api/auth/v1/delete (highlighted in red, locked)
- User** section:
 - GET /api/user/v1
 - GET /api/user/v1/users
 - GET /api/user/v1/profile
 - PATCH /api/user/v1/profile (highlighted in green, locked)
- Mail** section:
 - POST /api/mail/v1/confirmation
 - POST /api/mail/v1/reset-password

TDD

```
PASS  src/user/user.controller.spec.ts  
PASS  src/auth/auth.controller.spec.ts  
PASS  src/data/data.service.spec.ts  
PASS  src/auth/auth.service.spec.ts  
PASS  src/data/data.repository.spec.ts  
PASS  src/geoserver/geoserver.service.spec.ts  
PASS  src/mail/mail.service.spec.ts  
PASS  src/user/user.repository.spec.ts  
PASS  src/mail/mail.controller.spec.ts  
PASS  src/user/user.service.spec.ts  
PASS  src/image/image.service.spec.ts  
PASS  src/logger/logger.service.spec.ts
```

```
Test Suites: 13 passed, 13 total  
Tests:       236 passed, 236 total  
Snapshots:   0 total  
Time:        36.963 s  
Ran all test suites.
```

```
title is valid', async () => {  
  Model.find({});  
  
  aModel.findOne({  
  
    itory.findOneByTitle(title);  
  
    al(targetData.workspace);  
  
    throw error', async () => {  
      ne').mockImplementationOnce(() => {  
  
        await expect(  
          dataRepository.findOneByTitle(tiffData.title),  
          ).rejects.toThrowError(InternalServerErrorException);  
    });
  });
}
```

서비스로직의 메소드 단위를 테스트하는 단위 테스트 작성.

엔드포인트를 호출하여 전체적인 실제로직을 검증하는 통합 테스트 작성.

Mockist 보다 Classicist를 적용하여 서비스로직에 대한 신뢰성 확립.

Exception Handling

```
export class HttpExceptionFilter implements ExceptionFilter {
    private readonly logger = new Logger(HttpExceptionFilter.name);

    // Helper function to extract message
    private extractMessage(errorResponse: unknown): string {
        if (typeof errorResponse === 'object' && errorResponse !== null) {
            const message = (errorResponse as { message: unknown }).message;
            return Array.isArray(message) ? message[0] : message;
        }
        return 'An error occurred';
    }

    // Helper function to extract Location
    private extractLocation(errorResponse: unknown): string {
        if (
            typeof errorResponse === 'object' &&
            errorResponse !== null &&
            'at' in errorResponse
        ) {
            return errorResponse.at;
        }
        return '';
    }

    catch(exception: unknown, context: ExecutionContext) {
        const error = this.extractMessage(exception);
        const location = this.extractLocation(exception);
        const stack = exception.stack;

        this.logger.error(`Error ${error} at ${location}. Stack: ${stack}`);
        context.switchToHttp().handleException(exception);
    }
}
```

Global Exception Handler Filter를 적용.

좀 더 세분화된 에러처리 하기 위해 메소드 별 발생 할 수 있는 에러들을 구분.

디버깅시 추적하기 쉽게 할 수 있도록 각 Exception에 대한 추가적인 오브젝트 리터럴 값을 추가.

개선해야 할 부분

더 나은 비즈니스 모델을 위해



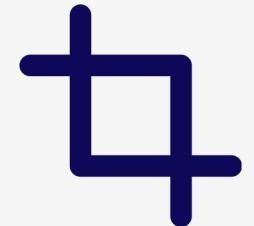
좀 더 폭넓은 TDD 작성

외부 서버와 의존성이 높은 코드들을 검증하는 테스트들은 어떻게 효율적으로 검증 할 수 있을지.



Observe

서버를 운영하면서 발생하는 에러들을 관리할 방안을 검토. 현재로는 그라파나, 도커 스웜 등을 검토.



코드 추상화

재사용성이 높은 코드를 작성하기 위한 방법 검토

감사합니다.