백엔드 코드 튜토리얼

1 필수 조건
2 개요
3 상세 설계
4 도메인 레이어
  4.1 도메인
    4.1.1 서브 프로젝트를 추가한다.
    4.1.2 build.gradle을 추가한다.
    4.1.3 도메인 객체를 추가한다.
    4.1.4 도메인 서비스를 추가한다.
  4.2 인프라스트럭처 도메인 서버
    4.2.1 서브 프로젝트를 추가한다.
    4.2.2 build.gradle을 추가한다.
    4.2.3 메인 클래스를 추가한다.
    4.2.4 엔티티와 레포지토리를 추가한다.
    4.2.5 컨트롤러를 추가한다.
5 어그리게이션 레이어
  5.1 어그리게이션
    5.1.1 서브 프로젝트를 추가한다.
    5.1.2 build.gradle을 추가한다.
    5.1.3 어그리게이션 서비스를 추가한다.
  5.2 인프라스트럭처 도메인 HTTP
    5.2.1 서브 프로젝트를 추가한다.
    5.2.2 build.gradle을 추가한다.
    5.2.3 HTTP API 요청과 응답에 사용될 객체를 추가한다.
  5.3 인프라스트럭처 도메인 클라이언트
    5.3.1 서브 프로젝트를 추가한다.
    5.3.2 build.gradle을 추가한다.
    5.3.3 클라이언트 서비스를 추가한다.
    5.3.4 AutoConfiguration를 작성한다.
  5.4 인프라스트럭처 어그리게이션 서버
    5.4.1 서브 프로젝트를 추가한다.
    5.4.2 build.gradle을 추가한다.
    5.4.3 메인 클래스를 추가한다.
    5.4.4 서비스 설정을 추가한다.
    5.4.5 컨트롤러를 추가한다.

1 필수 조건

- 5. 백엔드 개발 가이드
  - Gradle 가이드
  - JUnit 5 & Mockito 5 테스트 작성 가이드

2 개요

본 문서는 AM 프로젝트의 개발 가이드를 토대로 실제 프로젝트를 개발하는 과정을 보여 준다.

예시를 위해 파일 서비스를 구현하는 도메인 레이어, 어그리게이션 레이어를 구성하는 과정을 간략하게 서술한다.

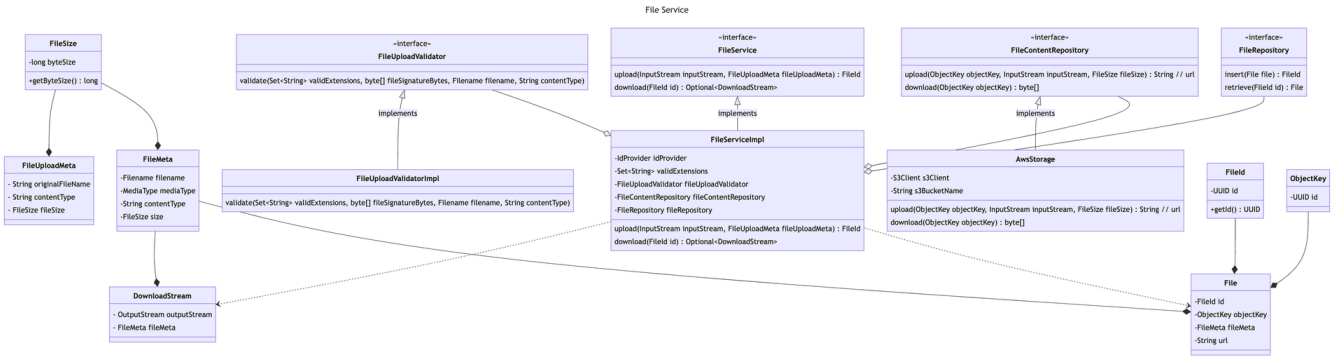
3 상세 설계

개발에 앞서 주요 도메인에 대한 상세 설계가 진행되어야 한다.

본 문서의 예시인 파일 서비스는 관계자들과 함께 모여 상세 설계에 필요한 정보들을 식별하였다.

- 파일 서비스 기능 인터뷰: 03. HR 첨부파일 요건 구체화
- 파일 서비스 설계 및 모듈 구현: 파일 서비스

아래는 상세 설계의 결과의 예시이다.



(link)

추가로 API 사양을 문서화하면 개발자들 사이의 협업과 테스트 시에 큰 도움이 된다.

4 도메인 레이어

4.1 도메인

도메인은 업무에 대한 모델과 규칙에 대한 구현을 포함한다.

- 도메인 객체
- 애그리게잇
- 도메인 이벤트
- 도메인 저장소 인터페이스
- 도메인 서비스 (인터페이스와 구현 모두)

본 예시에서는 간단한 예시를 위해 도메인 객체와 서비스 구현을 서술한다.

4.1.1 서브 프로젝트를 추가한다.

루트 프로젝트의 settings.gradle에 도메인 모듈을 서브 프로젝트로 추가한다.

settings.gradle

```
rootProject.name = 'am-backend'
...
//   .
**include 'support:domain:file'**
```

4.1.2 build.gradle을 추가한다.

support/domain/file/build.gradle

```
plugins {
    // Spring Boot        .
    id 'com.koreanair.java-common-convention'
}
```

경우에 따라 도메인 모듈에서 value-object 모듈을 분리할 수 있다. 이 경우 value-object에 대한 의존성을 추가한다.

비동기적 특성이 요구되는 경우 io.projectreactor:reactor-core 의존성을 추가할 수 있으나, 이외의 의존성은 추가할 수 없다.

4.1.3 도메인 객체를 추가한다.

support/domain/file/src/main/java/com/koreanair/support/file/DownloadStream.java

```
package com.koreanair.support.file;

import java.io.OutputStream;
import java.util.Objects;

public record DownloadStream(
        OutputStream outputStream,
        FileMeta fileMeta
) {

    public DownloadStream {
        Objects.requireNonNull(outputStream);
        Objects.requireNonNull(fileMeta);
    }
}
```

- 자바에서 객체의 동등성(Equality)과 동일성(Identity) 구현 전략
  - equals() 메서드 구현
    - 두 객체의 내용이 동일한지 확인한다.
    - 오버라이딩 할 때 실제 클래스의 필드들을 비교하여 동등성을 판단하도록 재정의해야 한다.
  - hashCode() 메서드 구현
    - 객체의 해시 코드 값을 반환한다. 해시 기반의 자료 구조에서 객체를 식별하는 데 사용된다.
    - 동일한 객체라면 동일한 해시 코드를 반환해야 한다.
  - 자바 16 record를 사용하면 equals(), hashCode()를 자동 생성하여 별도의 구현이 필요하지 않다.
    - 필요 시 오버라이딩하여 커스텀이 가능하다.
    - 단, 객체가 변경 가능하다면 record를 사용한 선언은 사용할 수 없음에 주의해야 한다.

4.1.4 도메인 서비스를 추가한다.

> *주의, Spring Configuration은 인프라스트럭처 도메인 서버에 추가한다.*

4.1.4.1. 도메인 서비스 인터페이스를 선언한다.

support/domain/file/src/main/java/com/koreanair/support/file/FileService.java

```
package com.koreanair.support.file;
...

public interface FileService {
    ...

    Optional<DownloadStream> download(
            FileId id);
}
```

4.1.4.2. 서비스 인터페이스에서 발생시키는 예외를 선언한다.

support/domain/file/src/main/java/com/koreanair/support/file/FileService.java

```
package com.koreanair.support.file;
...

public interface FileService {
    ...

    final class InternalServerIOException extends RuntimeException {
    }

    final class InvalidUploadFileException extends RuntimeException {
    }
}
```

4.1.4.3. 순수 자바로만 구현한, 서비스의 구체 클래스를 작성한다.

support/domain/file/src/main/java/com/koreanair/support/file/FileServiceImpl.java

```
package com.koreanair.support.file;
...

public class FileServiceImpl implements FileService {
    ...
    private final FileContentRepository fileContentRepository;
    private final FileRepository fileRepository;

    public FileServiceImpl(
            ...
            final FileContentRepository fileContentRepository,
            final FileRepository fileRepository) {
        ...
        this.fileContentRepository = Objects.requireNonNull(fileContentRepository);
        this.fileRepository = Objects.requireNonNull(fileRepository);
    }
    ...
    @Override
    public Optional<DownloadStream> download(
            final FileId id) {
        return fileRepository.retrieve(id)
                .map(file -> {
                    final var downloadBytes = fileContentRepository.download(file.objectKey());
                    final var outputStream = new ByteArrayOutputStream();
                    try {
                        outputStream.write(downloadBytes);
                    } catch (IOException e) {
                        throw new InternalServerIOException(e);
                    }
                    return new DownloadStream(outputStream, file.fileMeta());
                });
    }
}
```

support/domain/file/src/test/java/com/koreanair/support/file/FileServiceImplTest.java

```
package com.koreanair.support.file;
...
// testImplementation project(':test')        .
import static com.koreanair.test.RandomTestUtils.*;
// AssertJ  assertion    .
import static org.assertj.core.api.Assertions.assertThat;
...

@ExtendWith(MockitoExtension.class)
class FileServiceImplTest {
  public static final String PDF_EXTENSION= "pdf";

  //   underTest  .
  @InjectMocks
  private FileServiceImpl underTest;
  ...

  // Mock  .
  @Mock
  private FileContentRepository fileContentRepository;
  @Mock
  private FileRepository fileRepository;
  ...

  @Test
  void download() {
    // given
    final var file = new File(
      new FileId(randomUUID()),
      new ObjectKey(randomUUID()),
      mock(FileMeta.class),
      randomAlphanumeric()// url
    );
    given(fileRepository.retrieve(any()))
   .willReturn(Optional.of(file));

    given(fileContentRepository.download(any()))
    .willReturn(new byte[0]);

    // when
    final var id = new FileId(randomUUID());
    final var actual = underTest.download(id);

    // then
    assertThat(actual.isPresent()).isTrue();
    assertThat(actual.get().fileMeta()).isEqualTo(file.fileMeta());

    then(fileRepository).should().retrieve(id);
    then(fileContentRepository).should().download(file.objectKey());
  }
}
```

4.2 인프라스트럭처 도메인 서버

인프라스트럭처 도메인 서버는 인프라스트럭처 레이어에 해당하며 메인 클래스를 가지고 있기 때문에 실행 가능하다.

또한 아래의 기능 중 하나 이상을 담당한다.

- HTTP API 제공
- HTTP API 호출
- 이벤트 발행
- 이벤트 구독

4.2.1 서브 프로젝트를 추가한다.

루트 프로젝트의 settings.gradle에 인프라스트럭처 도메인 서버 모듈을 서브 프로젝트로 추가한다.

settings.gradle

```
rootProject.name = 'am-backend'
...
// subproject .
include 'example:file:infrastructure:domain:file-server'
```

4.2.2 build.gradle을 추가한다.

example/file/infrastructure/domain/file-server/build.gradle

```
plugins {
    // Java domain server convention   .
    id 'com.koreanair.java-domain-server-convention'
    // Java jpa conventiont   .
    id 'com.koreanair.java-jpa-convention'
}

dependencies {
  //       .
  implementation project(':support:domain:file')
  // HTTP        .
  implementation project(':example:file:infrastructure:domain:file-http')
}
```

4.2.3 메인 클래스를 추가한다.

example/file/infrastructure/domain/file-server/src/main/java/com/koreanair/example/file/infrastructure/domain/file_server/ExampleFileServer.java

```
package com.koreanair.example.file.infrastructure.domain.file_server;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ExampleFileServer {

    public static void main(final String[] args) {
        SpringApplication.run(ExampleFileServer.class, args);
    }
}
```

4.2.4 엔티티와 레포지토리를 추가한다.

example/file/infrastructure/domain/file-server/src/main/java/com/koreanair/example/file/infrastructure/domain/file_server/repository/FileEntity.java

```
package com.koreanair.example.file.infrastructure.domain.file_server.repository;

import jakarta.persistence.*;

import java.util.Objects;

@Entity
@Table(name = "file")
class FileEntity {
    @EmbeddedId
    private FileEntityId id;
    @Embedded
    @AttributeOverride(name = "id", column = @Column(name = "object_key"))
    private ObjectKeyEntity objectKey;
    @Embedded
    private FileMetaEntity fileMetaData;
    private String url;

    protected FileEntity() {
    }

    FileEntity(
            final FileEntityId id,
            final ObjectKeyEntity objectKey,
            final FileMetaEntity fileMetaData,
            final String url) {
        this.id = Objects.requireNonNull(id);
        this.objectKey = Objects.requireNonNull(objectKey);
        this.fileMetaData = Objects.requireNonNull(fileMetaData);
        this.url = Objects.requireNonNull(url);
    }

    FileEntityId getId() {
        return id;
    }

    ObjectKeyEntity getObjectKey() {
        return objectKey;
    }

    FileMetaEntity getFileMetaData() {
        return fileMetaData;
    }

    String getUrl() {
        return url;
    }
}
```

example/file/infrastructure/domain/file-server/src/main/java/com/koreanair/example/file/infrastructure/domain/file_server/repository/FileJpaRepository.java

```
package com.koreanair.example.file.infrastructure.domain.file_server.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
interface FileJpaRepository extends JpaRepository<FileEntity, FileEntityId> {
}
```

Spring Data Repository Interface

- 만약 JpaSpecification 사용한다면 `JpaSpecificationExecutor` 를 상속한다.
- Pagination이나 Sort를 사용한다면 `JpaRepository`를 상속한다.
- 이외에는 `ListCrudRepository`를 상속한다.

example/file/infrastructure/domain/file-server/src/main/java/com/koreanair/example/file/infrastructure/domain/file_server/repository/FileSpringDataRepository.java

```
package com.koreanair.example.file.infrastructure.domain.file_server.repository;
...

public class FileSpringDataRepository implements FileRepository {

    private final FileJpaRepository fileJpaRepository;

    public FileSpringDataRepository(
            final FileJpaRepository fileJpaRepository) {
        this.fileJpaRepository = Objects.requireNonNull(fileJpaRepository);
    }

    @Override
    public FileId insert(
            final File file) {
        final var newFileEntity = toEntity(file);
        return fromEntityId(fileJpaRepository.save(newFileEntity).getId());
    }

    @Override
    public Optional<File> retrieve(
            final FileId id) {
        return fileJpaRepository.findById(toEntityId(id))
                .map(FileSpringDataRepository::fromEntity);
    }

    // ,  .
    private static FileEntity toEntity(
            final File file) {
        return new FileEntity(
                toEntityId(file.id()),
                toEntityObjectKey(file.objectKey()),
                toEntityMetaData(file.fileMeta()),
                file.url()
        );
    }
    ...

    private static File fromEntity(
            final FileEntity fileEntity) {
        return new File(
                fromEntityId(fileEntity.getId()),
                fromEntityObjectKey(fileEntity.getObjectKey()),
                fromEntityMetaData(fileEntity.getFileMetaData()),
                fileEntity.getUrl()
        );
    }
    ...
}
```

- 코딩 컨벤션은 5. 백엔드 개발 가이드 를 참고한다.

example/file/infrastructure/domain/file-server/src/main/java/com/koreanair/example/file/infrastructure/domain/file_server/repository/FileRepositoryConfig.java

```
package com.koreanair.example.file.infrastructure.domain.file_server.repository;

import com.koreanair.support.file.FileRepository;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
class FileRepositoryConfig {

    @Bean
    FileRepository fileRepository(
            final FileJpaRepository fileJpaRepository) {
        return new FileSpringDataRepository(fileJpaRepository);
    }
}
```

4.2.5 컨트롤러를 추가한다.

example/file/infrastructure/domain/file-server/src/main/java/com/koreanair/example/file/infrastructure/domain/file_server/controller/FileController.java

```
package com.koreanair.example.file.infrastructure.domain.file_server.controller;
...

@RestController
//  (Entry Point)
@RequestMapping("/files")
class FileController {
    private final FileService fileService;

    FileController(
            final FileService fileService) {
        this.fileService = Objects.requireNonNull(fileService);
    }
    ...
    //  (Endpoint)
    @GetMapping("/{id}/display")
    void display(
            @PathVariable(name = "id") final UUID id,
            final HttpServletResponse servletResponse
    ) {
        final var downloadStream = fileService.download(new FileId(id))
                .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND));

        servletResponse.setContentType(MediaType.APPLICATION_OCTET_STREAM_VALUE);
        writeOutputStream(downloadStream, servletResponse);
    }
    ...

    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler(FileService.InternalServerIOException.class)
    void handleInternalServerException() {
    }
    ...
}
```

- 컨트롤러 클래스 내 경로에 대한 코드 컨벤션은 다음과 같다.
    - Entrypoint는 Controller 위에 나타낸다
    - Endpoint는 Handler Method 위에 나타낸다
- 예외에 대한 처리는 에러 핸들링 가이드를 참고한다.
    - 에러 핸들링 가이드

5 어그리게이션 레이어

5.1 어그리게이션

5.1.1 서브 프로젝트를 추가한다.

루트 프로젝트의 settings.gradle에 어그리게이션 모듈을 서브 프로젝트로 추가한다.

settings.gradle

```
rootProject.name = 'am-backend'
...
// subproject .
include 'example:file:aggregation'
```

5.1.2 build.gradle을 추가한다.

example/file/aggregation/build.gradle

```
plugins{
                id 'com.koreanair.java-common-convention'
}

dependencies{
                implementation 'io.projectreactor:reactor-core'
}
```

5.1.3 어그리게이션 서비스를 추가한다.

비동기적으로 동작하는 서비스 인터페이스는 접미사로 ReactiveService를 사용한다.

example/file/aggregation/src/main/java/com/koreanair/example/file/aggregation/file/FileReactiveService.java

```
package com.koreanair.example.file.aggregation.file;

import reactor.core.publisher.Flux;

public interface FileReactiveService {

    Flux<byte[]> retrieve(
            String id);
}
```

어그리게이션 서비스 인터페이스는 질의와 명령으로 그 성격을 분류하여 접미사로 각각 AggregationQueryService, AggregationCommandService 를 사용한다.

example/file/aggregation/src/main/java/com/koreanair/example/file/aggregation/file/FileAggregationQueryService.java

```
package com.koreanair.example.file.aggregation.file;

import reactor.core.publisher.Flux;

public interface FileAggregationQueryService {

    Flux<byte[]> retrieve(
            String id);
}
```

example/file/aggregation/src/main/java/com/koreanair/example/file/aggregation/file/FileAggregateQueryServiceImpl.java

```
package com.koreanair.example.file.aggregation.file;

import reactor.core.publisher.Flux;

import java.util.Objects;

public record FileAggregateQueryServiceImpl(
        FileReactiveService fileReactiveService
) implements FileAggregationQueryService {

    public FileAggregateQueryServiceImpl(
            final FileReactiveService fileReactiveService) {
        this.fileReactiveService = Objects.requireNonNull(fileReactiveService);
    }

    @Override
    public Flux<byte[]> retrieve(
            final String id) {
        return fileReactiveService.retrieve(id);
    }
}
```

example/file/aggregation/src/test/java/com/koreanair/example/file/aggregation/file/FileAggregateQueryServiceImplTest.java

```
package com.koreanair.example.file.aggregation.file;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import reactor.core.publisher.Flux;
import reactor.test.StepVerifier;

import static com.koreanair.test.RandomTestUtils.randomAlphanumeric;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.BDDMockito.given;
import static org.mockito.BDDMockito.then;

@ExtendWith(MockitoExtension.class)
class FileAggregateQueryServiceImplTest {

    @InjectMocks
    private FileAggregateQueryServiceImpl underTest;

    @Mock
    private FileReactiveService fileReactiveService;

    @Test
    void retrieve() {
        // given
        final var bytes = new byte[0];
        given(fileReactiveService.retrieve(any())).willReturn(Flux.just(bytes));

        // when
        final var id = randomAlphanumeric();
        final var actual = underTest.retrieve(id);

        // StepVerifier      .
        StepVerifier.create(actual)
                .expectNext(bytes)
                .verifyComplete();

        // then
        then(fileReactiveService).should().retrieve(id);
    }
}
```

5.2 인프라스트럭처 도메인 HTTP

인프라스트럭처 도메인 HTTP는 HTTP API의 요청과 응답에 사용될 객체를 포함한다. 이러한 객체는 JSON 직렬화 및 역직렬화의 대상이다.

5.2.1 서브 프로젝트를 추가한다.

루트 프로젝트의 settings.gradle에 인프라스트럭처 도메인 HTTP 모듈을 서브 프로젝트로 추가한다.

settings.gradle

```
rootProject.name = 'am-backend'
...
// subproject .
include 'example:file:infrastructure:domain:file-http
```

5.2.2 build.gradle을 추가한다.

```
plugins {
    id 'com.koreanair.java-common-convention'
}
```

이외에 특별하게 직렬화 및 역직렬화가 필요한 경우 관련 의존성을 추가할 수 있다.

5.2.3 HTTP API 요청과 응답에 사용될 객체를 추가한다.

5.3 인프라스트럭처 도메인 클라이언트

5.3.1 서브 프로젝트를 추가한다.

루트 프로젝트의 settings.gradle에 인프라스트럭처 클라이언트 HTTP 모듈을 서브 프로젝트로 추가한다.

settings.gradle

```
rootProject.name = 'am-backend'
...
// subproject .
**include 'example:file:infrastructure:domain:file-client'**
```

5.3.2 build.gradle을 추가한다.

example/file/infrastructure/domain/file-client/build.gradle

```
plugins {
     // Java client convention   .
     id 'com.koreanair.java-client-convention'
}

dependencies {
    implementation project(':example:file:aggregation')
    // HTTP        .
    implementation project(':example:file:infrastructure:domain:file-http')
}
```

5.3.3 클라이언트 서비스를 추가한다.

example/file/infrastructure/domain/file-client/src/main/java/com/koreanair/example/file/infrastructure/domain/file_client/FileClientReactiveService.java

```
package com.koreanair.example.file.infrastructure.domain.file_client;

import com.koreanair.example.file.aggregation.file.FileReactiveService;
import org.springframework.http.MediaType;
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Flux;

import java.util.Objects;

record FileClientReactiveService(
        WebClient webClient
) implements FileReactiveService {

    public static final String FILE_PATH = "/files";
    public static final String FILE_DISPLAY_PATH = "/display";

    FileClientReactiveService(
            final WebClient webClient) {
        this.webClient = Objects.requireNonNull(webClient);
    }

    @Override
    public Flux<byte[]> retrieve(
            final String id) {
        return webClient.get()
                .uri(uriBuilder ->
                        uriBuilder.path(FILE_PATH + "/" + id + FILE_DISPLAY_PATH)
                                .build()
                )
                .accept(MediaType.APPLICATION_OCTET_STREAM)
                .retrieve()
                .bodyToFlux(byte[].class);
    }
}
```

5.3.4 AutoConfiguration를 작성한다.

인프라스트럭처 어그리게이션 서버에서의 의존성 주입을 위해 인프라스트럭처 도메인 클라이언트를 모듈화한다.

example/file/infrastructure/domain/file-client/src/main/java/com/koreanair/example/file/infrastructure/domain/file_client/FileReactiveServiceAutoConfiguration.java

```
package com.koreanair.example.file.infrastructure.domain.file_client;

import com.koreanair.example.file.aggregation.file.FileReactiveService;
import org.springframework.boot.autoconfigure.AutoConfigureAfter;
import org.springframework.boot.autoconfigure.condition.ConditionalOnBean;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.boot.autoconfigure.web.reactive.function.client.WebClientAutoConfiguration;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.reactive.function.client.WebClient;

@Configuration
@AutoConfigureAfter(WebClientAutoConfiguration.class)
@EnableConfigurationProperties(FileClientProperties.class)
class FileReactiveServiceAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    @ConditionalOnBean(WebClient.Builder.class)
    FileReactiveService fileReactiveService(
            final WebClient.Builder webClientBuilder,
            final FileClientProperties fileClientProperties
    ) {
        final var webClient = webClientBuilder
                .baseUrl(fileClientProperties.baseUrl())
                .build();
        return new FileClientReactiveService(
                webClient);
    }
}
```

example/file/infrastructure/domain/file-client/src/main/resources/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports

```
com.koreanair.example.file.infrastructure.domain.file_client.FileReactiveServiceAutoConfiguration
com.koreanair.example.file.infrastructure.domain.file_client.FileClientPropertiesAutoConfiguration
```

5.4 인프라스트럭처 어그리게이션 서버

5.4.1 서브 프로젝트를 추가한다.

루트 프로젝트의 settings.gradle에 인프라스트럭처 어그리게이션 서버 모듈을 서브 프로젝트로 추가한다.

settings.gradle

```
rootProject.name = 'am-backend'
...
// subproject .
include 'example:file:infrastructure:aggregation:file-server'
```

5.4.2 build.gradle을 추가한다.

example/file/infrastructure/aggregation/file-server/build.gradle

```
plugins{
  id 'com.koreanair.java-aggregation-server-convention'
}

dependencies{
    implementation project(':example:file:aggregation')
    implementation project(':example:file:infrastructure:domain:file-client')
}
```

5.4.3 메인 클래스를 추가한다.

example/file/infrastructure/aggregation/file-server/src/main/java/com/koreanair/example/file/infrastructure/aggregation/file_server/ExampleFileAggregationServer.java

```
package com.koreanair.example.file.infrastructure.aggregation.file_server;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ExampleFileAggregationServer {

    public static void main(final String[] args) {
        SpringApplication.run(ExampleFileAggregationServer.class, args);
    }
}
```

5.4.4 서비스 설정을 추가한다.

example/file/infrastructure/aggregation/file-server/src/main/java/com/koreanair/example/file/infrastructure/aggregation/file_server/service/FileAggregationQueryServiceConfig.java

```
package com.koreanair.example.file.infrastructure.aggregation.file_server.service;

import com.koreanair.example.file.aggregation.file.FileAggregateQueryServiceImpl;
import com.koreanair.example.file.aggregation.file.FileAggregationQueryService;
import com.koreanair.example.file.aggregation.file.FileReactiveService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
class FileAggregationQueryServiceConfig {

    @Bean
    FileAggregationQueryService fileAggregationQueryService(
            final FileReactiveService fileReactiveService
    ) {
        return new FileAggregateQueryServiceImpl(fileReactiveService);
    }
}
```

5.4.5 컨트롤러를 추가한다.

example/file/infrastructure/aggregation/file-server/src/main/java/com/koreanair/example/file/infrastructure/aggregation/file_server/controller/FileAggregationController.java

```
package com.koreanair.example.file.infrastructure.aggregation.file_server.controller;

import com.koreanair.example.file.aggregation.file.FileAggregationQueryService;
import org.springframework.core.io.buffer.DataBuffer;
import org.springframework.http.MediaType;
import org.springframework.http.server.reactive.ServerHttpResponse;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import java.util.Objects;

@RestController
//  API URL Path    .
@RequestMapping("/api/v1/files")
class FileAggregationController {

    private final FileAggregationQueryService fileAggregationQueryService;

    FileAggregationController(
            final FileAggregationQueryService fileAggregationQueryService) {
        this.fileAggregationQueryService = Objects.requireNonNull(fileAggregationQueryService);
    }

    @GetMapping("/{id}/display")
    Mono<Void> retrieve(
            ServerHttpResponse response,
            @PathVariable final String id) {
        final var dataStream = fileAggregationQueryService.retrieve(id);
        Flux<DataBuffer> dataBufferFlux = dataStream.map(data -> response.bufferFactory().wrap(data));
        response.getHeaders().setContentType(MediaType.APPLICATION_OCTET_STREAM);
        return response.writeWith(dataBufferFlux);
    }
}
```