애자일 기반 건강관리 서비스 개발 프로젝트

염윤호 고민표 박희수 이재원 이창훈

목차

1-1. 팀 소개

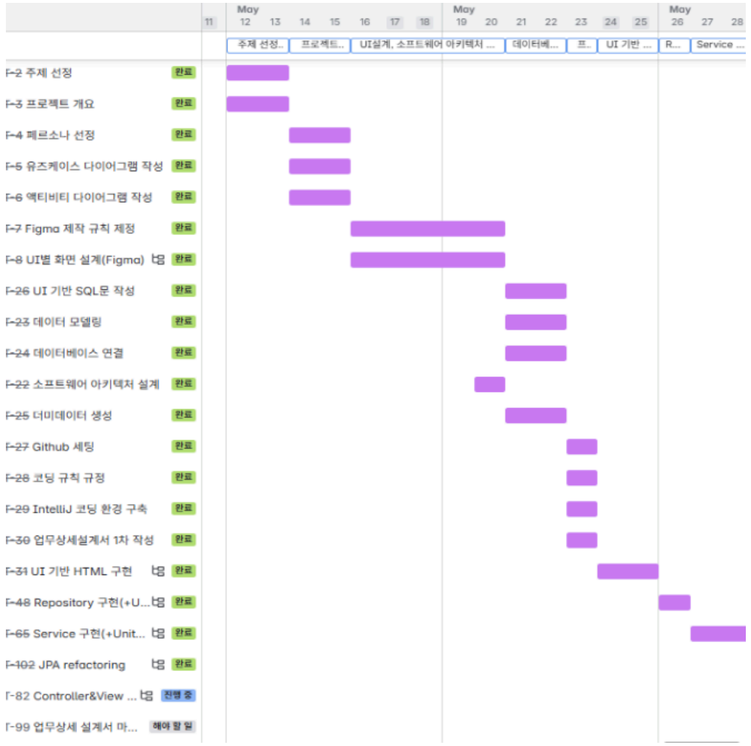
Selfit은 스스로 **운동을 계획하고 기록**하며, 건강한 습관을 만들어 가는 사람들을 위한 **동기부여 커뮤니티 플랫폼**입니다.

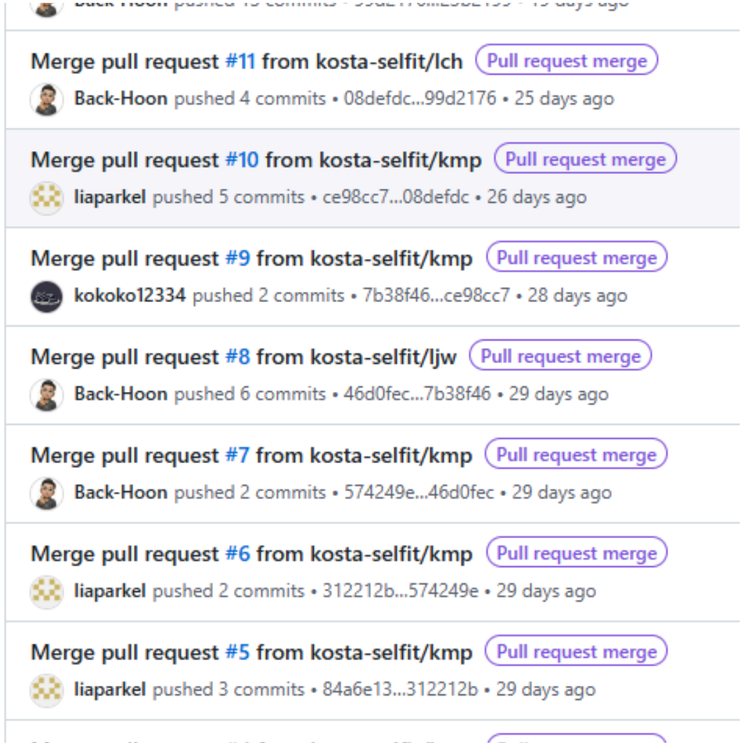
	구현 담당	그 외 담당
염윤호	대시보드(운동)	Jira 스케줄 관리
고민표	사용자계정, 인증/인가	Git/GitHub 관리, 컨벤션 관리
박희수	대시보드(음식)	회의록 작성, 업무상세 설계서 관리
이재원	커뮤니티(게시판)	Firebase 관리, UI서버 관리
이창훈	대시보드(체크리스트)	통합테스트, 인프라 관리

1-2. 팀 협업 프로세스

스프린트 계획 | 개발 진행 | 일일 회의 | 회고

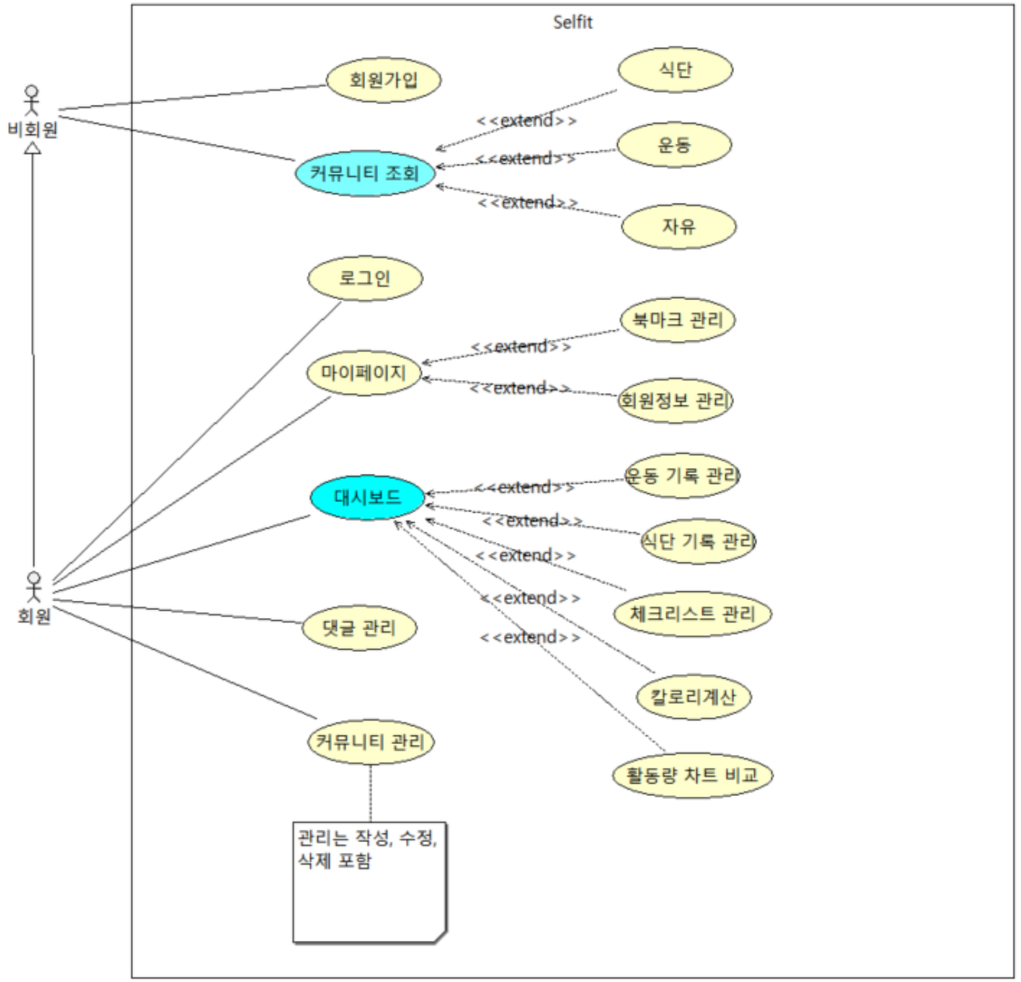
코드 개발
페어 프로그래밍 | 코드 리뷰 | 리팩토링



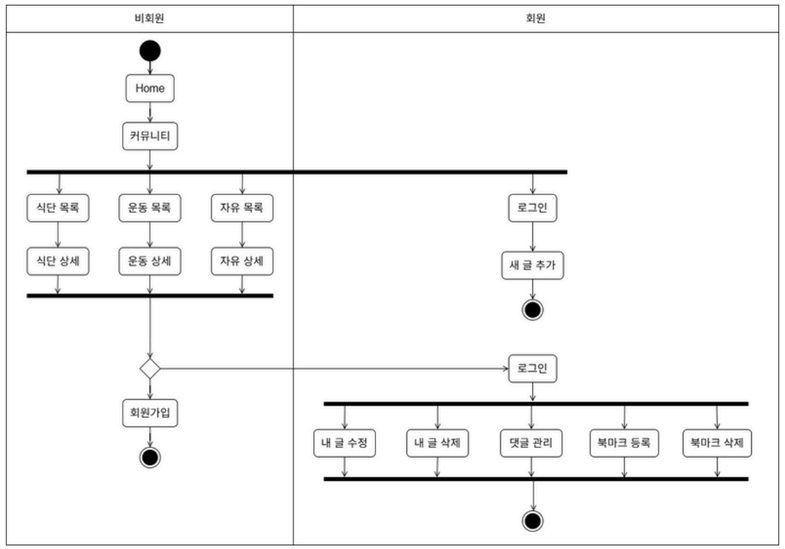


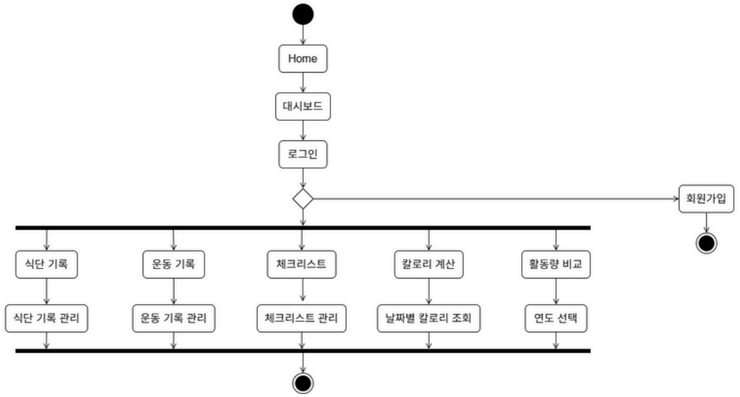
1-3. 업무 소개

유즈케이스



액티비티 다이어그램



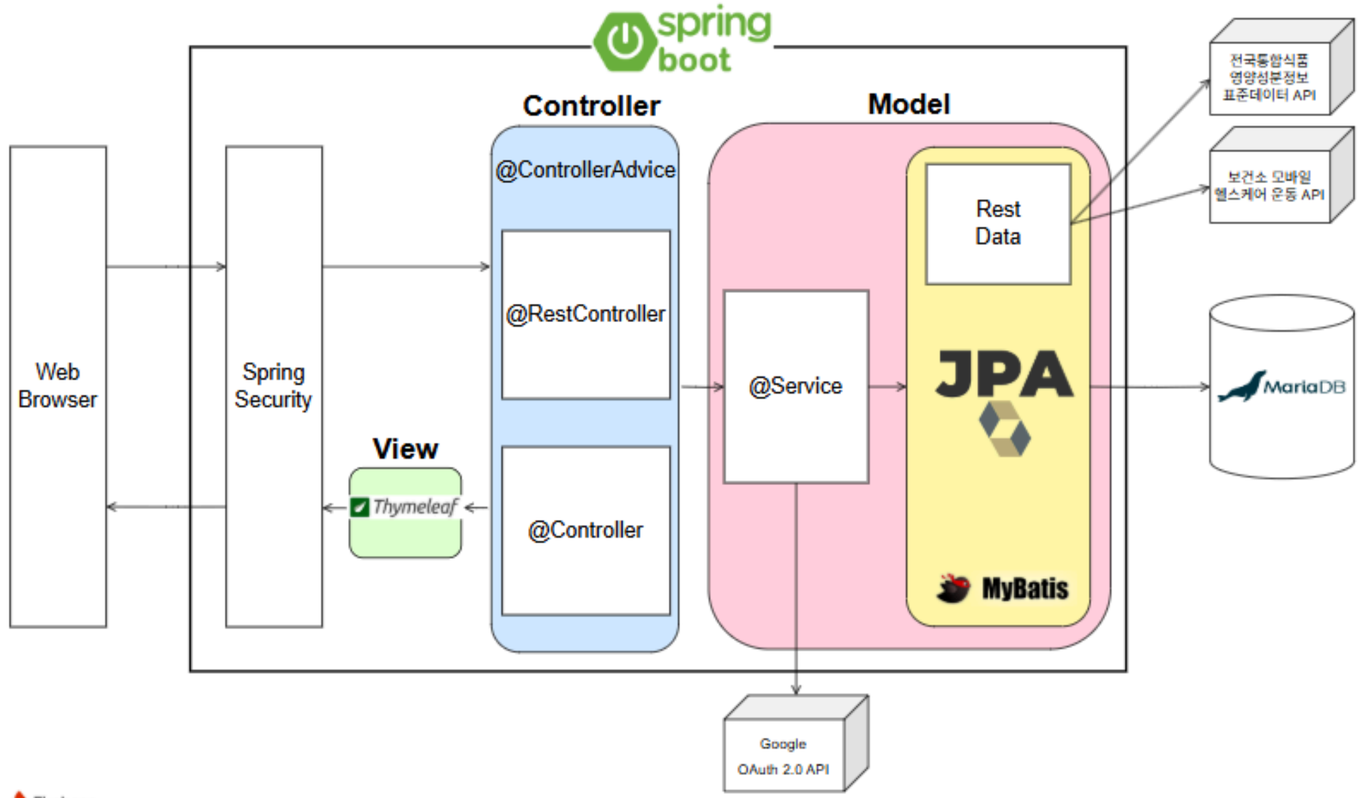


2-1. Spring Boot 기반 1차 스프린트 - 도입배경

객체 생성과 의존성 관리가 직접 이루어져 중복과 결합도가 높았고, 테스트가 어려웠습니다.

IoC 컨테이너와 DI가 가능한 스프링부트를 도입했습니다.

2-2. Spring Boot 기반 1차 스프린트 - S/W 아키텍처



3-1. REST API 기반 2차 스프린트 - 도입배경

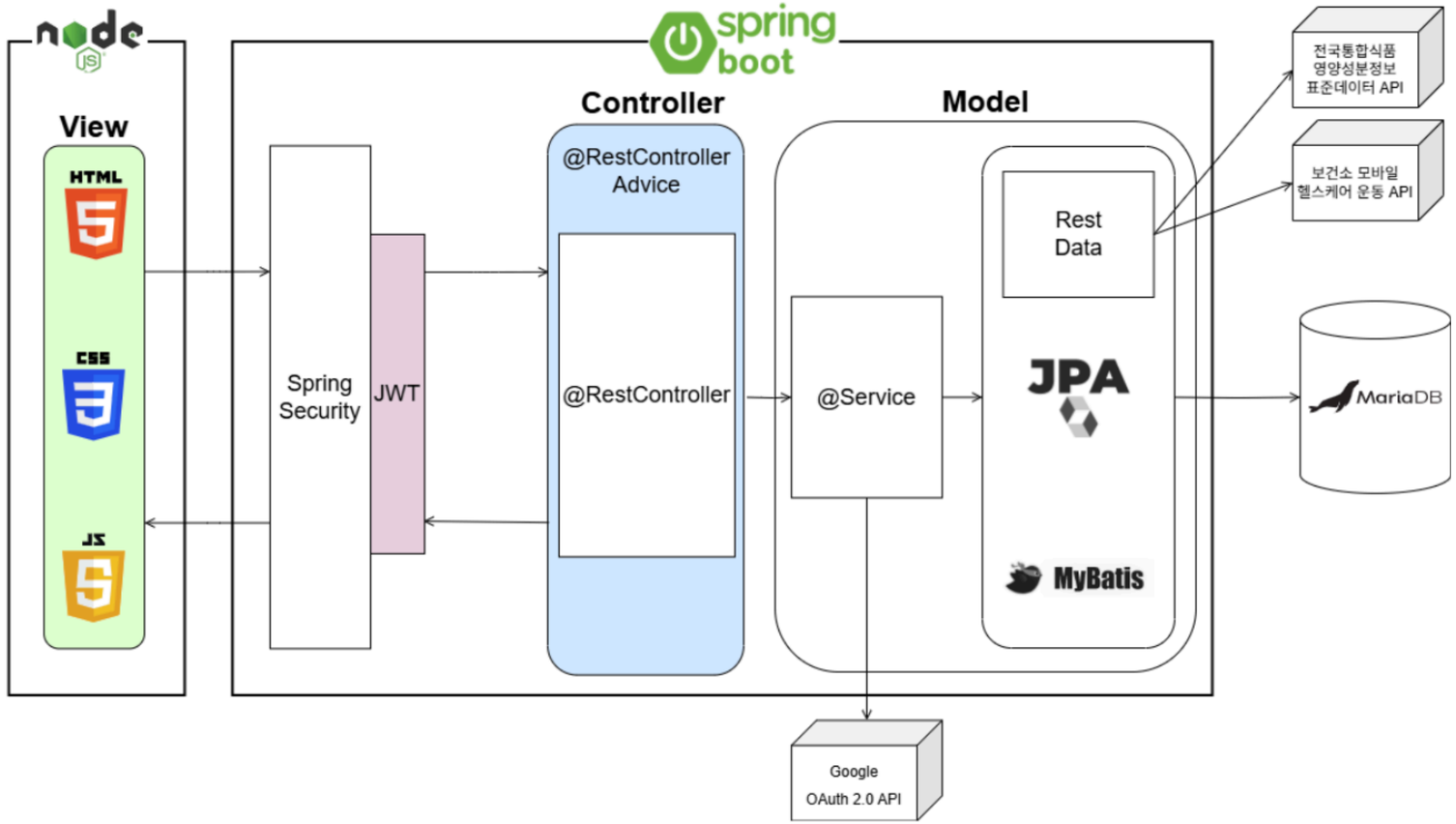
1.백엔드와 프론트엔드가 한 서버 내에서 실행되어서 코드를 수정해도 전체 서버를 재가동해야 하는 불편함이 있었습니다.

서버 분리를 통해 결합도를 낮추어 독립적인 개발과 배포가 가능하도록 개선했습니다.

2. Controller+Thymeleaf 사용으로 BackEnd에서도 FrontEnd의 정보를 알아야 하는 의존성 문제가 있었습니다.

REST API를 적용하여 Back-Front 역할을 명확히 분리하고 독립적인 개발 환경을 구축했습니다.

3-2. REST API 기반 2차 스프린트 - S/W 아키텍처



4-1. MSA 기반 3차 스프린트 - 도입배경

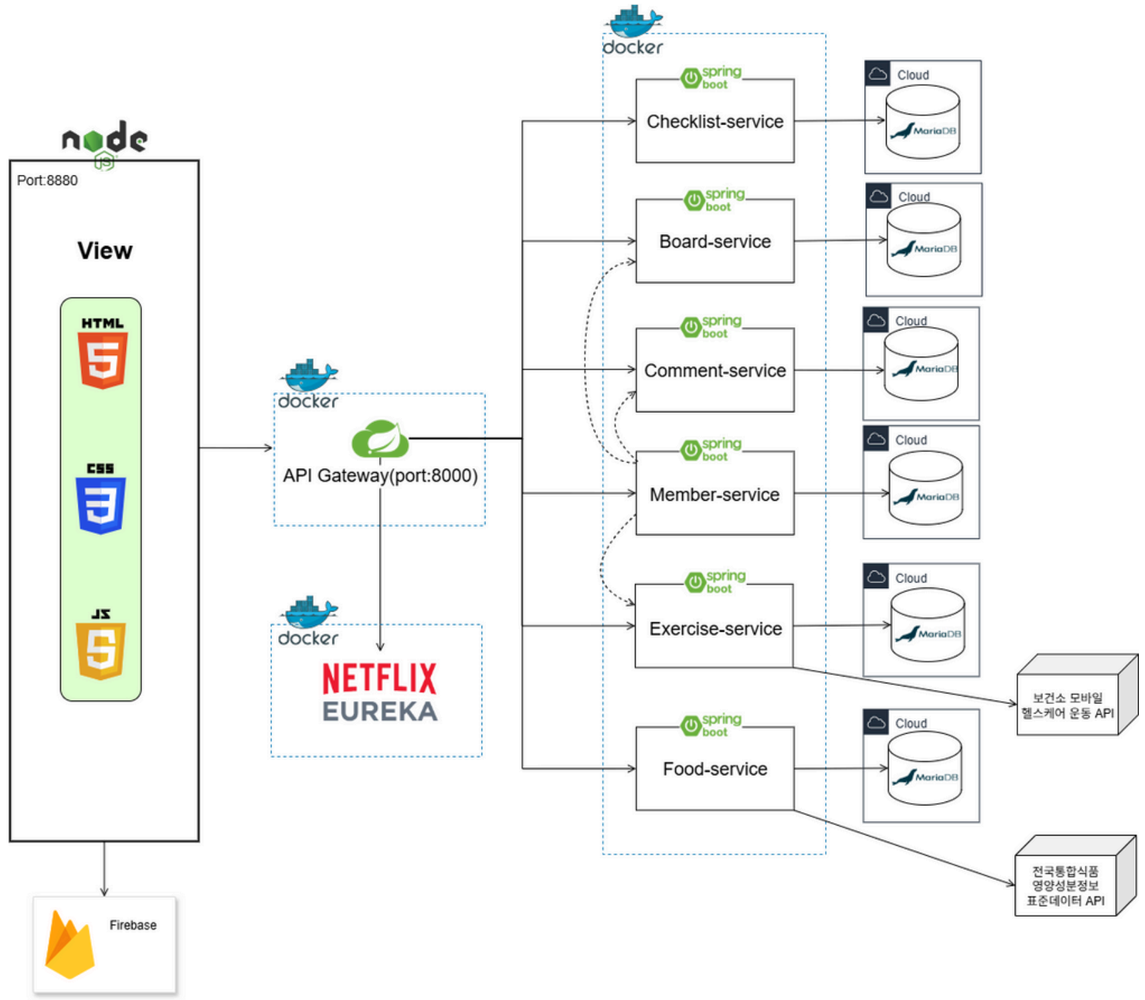
1.기존 모놀리식 구조에서 다른 사람의 코드가 내 코드에도 영향을 주는 문제가 있었습니다.

마이크로 서비스로 나누어 영향을 최소화하고 독립성을 확보합니다.

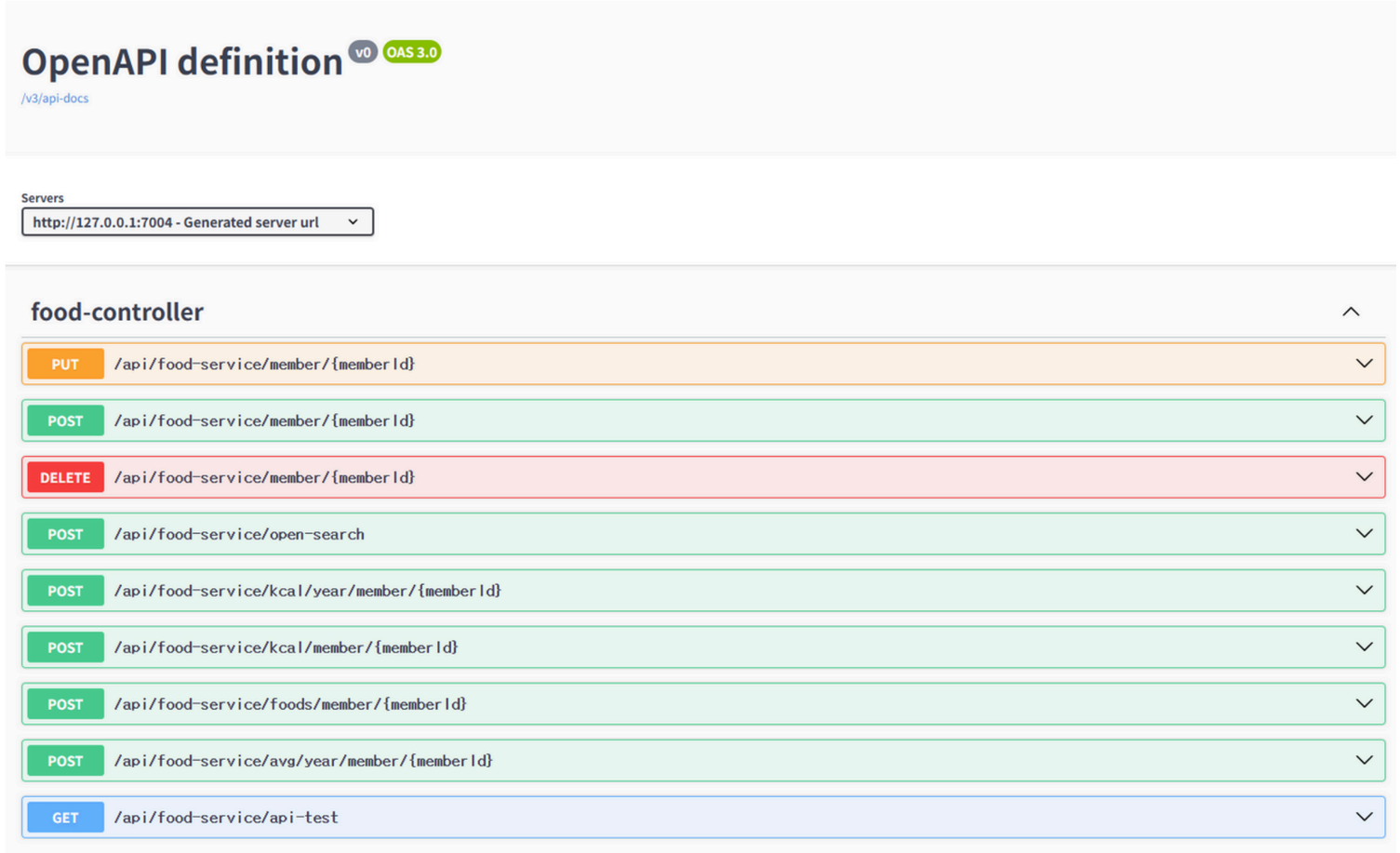
2. 특정 서비스 장애 시 전체 시스템을 확장해야 하는 비효율성 문제가 있습니다.

트래픽이 많은 마이크로 서비스만 선택적으로 인스턴스를 추가 확장할 수 있도록 구조를 변경합니다.

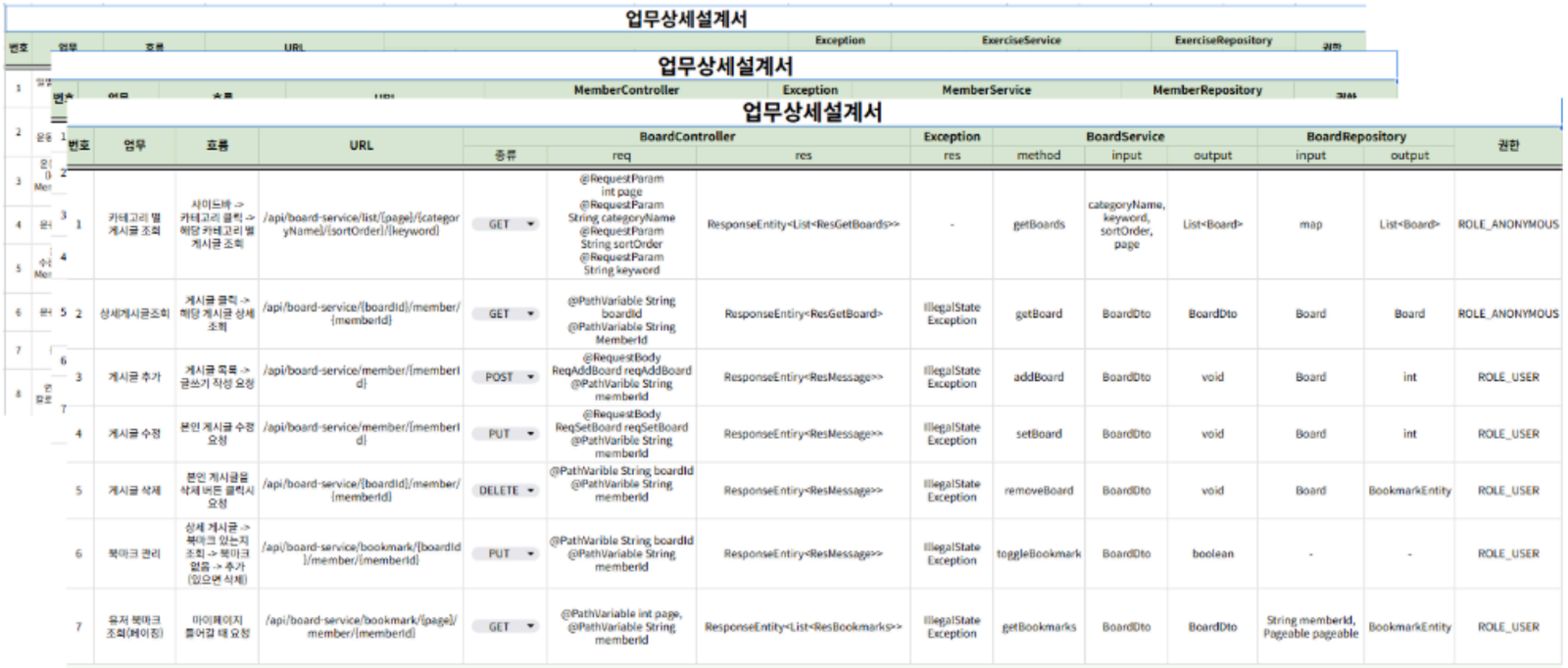
4-2. MSA 기반 3차 스프린트 - S/W 아키텍처



5. 최종 산출물 - SWAGGER



5. 최종 산출물 - 업무 상세 설계서



업무상세설계서

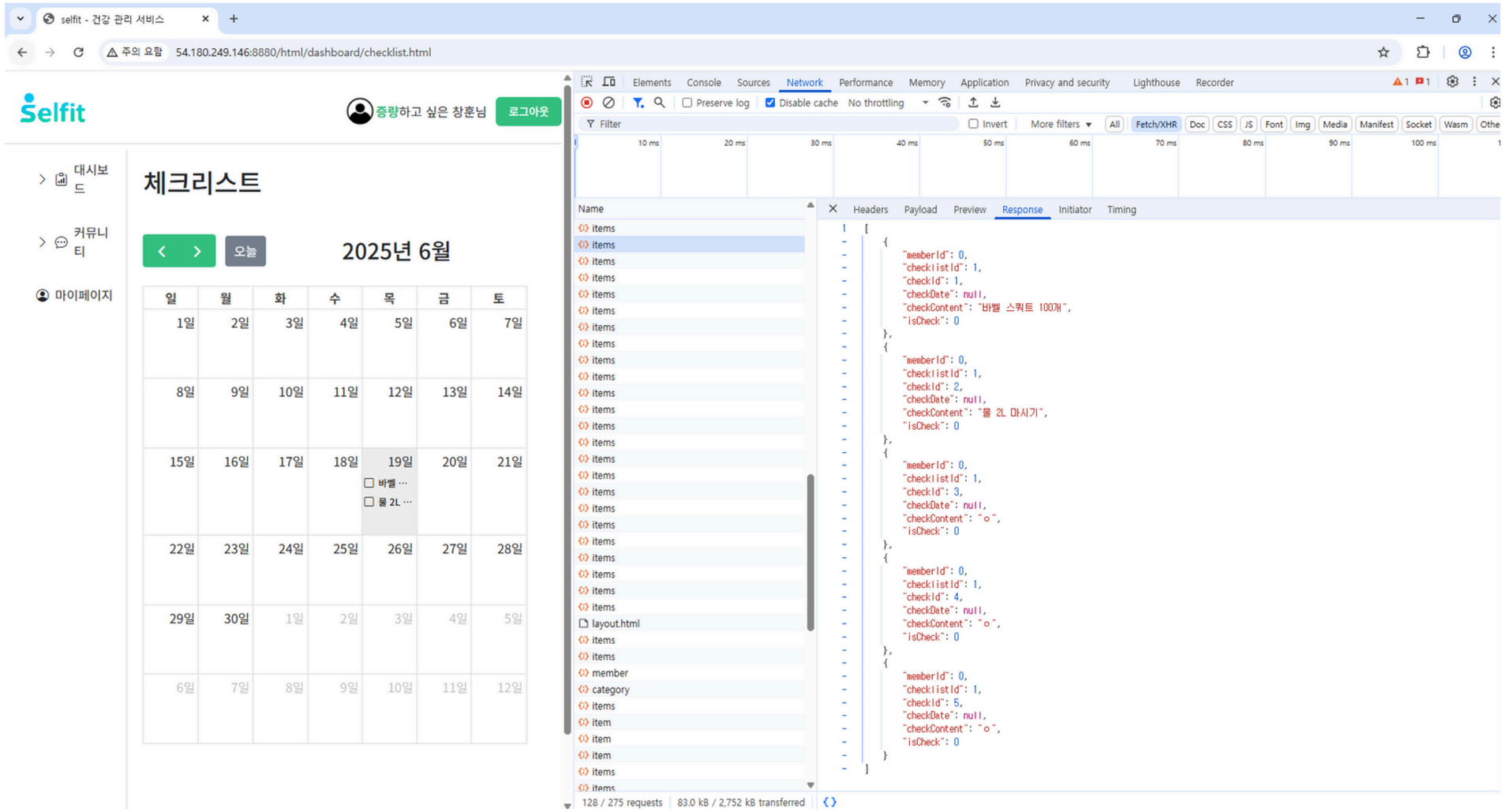
번호	업무	흐름	URL	BoardController			Exception	BoardService			BoardRepository		권한
				종류	req	res	res	method	input	output	input	output	
1	카테고리 별 게시글 조회	사이드바 -> 카테고리 클릭 -> 해당 카테고리별 게시글 조회	/api/board-service/list/{page}/{categoryName}/{sortOrder}/{keyword}	GET	@RequestParam int page @RequestParam String categoryName @RequestParam String sortOrder @RequestParam String keyword	ResponseEntity<List<ResGetBoards>>	-	getBoards	categoryName, keyword, sortOrder, page	List<Board>	map	List<Board>	ROLE_ANONYMOUS
2	상세게시글조회	게시글 클릭 -> 해당 게시글 상세 조회	/api/board-service/{boardId}/member/{memberId}	GET	@PathVariable String boardId @PathVariable String MemberId	ResponseEntiry<ResGetBoard>	IllegalState Exception	getBoard	BoardDto	BoardDto	Board	Board	ROLE_ANONYMOUS
3	게시글 추가	게시글 목록 -> 글쓰기 작성 요청	/api/board-service/member/{memberId}	POST	@RequestBody ReqAddBoard reqAddBoard @PathVaribe String memberId	ResponseEntiry<ResMessage>>	IllegalState Exception	addBoard	BoardDto	void	Board	int	ROLE_USER
4	게시글 수정	본인 게시글 수정 요청	/api/board-service/member/{memberId}	PUT	@RequestBody ReqSetBoard reqSetBoard @PathVaribe String memberId	ResponseEntiry<ResMessage>>	IllegalState Exception	setBoard	BoardDto	void	Board	int	ROLE_USER
5	게시글 삭제	본인 게시글을 삭제 버튼 클릭시 요청	/api/board-service/{boardId}/member/{memberId}	DELETE	@PathVaribe String boardId @PathVaribe String memberId	ResponseEntiry<ResMessage>>	IllegalState Exception	removeBoard	BoardDto	void	Board	BookmarkEntity	ROLE_USER
6	북마크 관리	상세 게시글 -> 북마크 있는지 조회 -> 북마크 없음 -> 추가 (있으면 삭제)	/api/board-service/bookmark/{boardId}/member/{memberId}	PUT	@PathVaribe String boardId @PathVaribe String memberId	ResponseEntiry<ResMessage>>	IllegalState Exception	toggleBookmark	BoardDto	boolean	-	-	ROLE_USER
7	유저 북마크 조회(페이징)	마이페이지 들어갈 때 요청	/api/board-service/bookmark/{page}/member/{memberId}	GET	@PathVariable int page, @PathVariable String memberId	ResponseEntity<List<ResBookmarks>>	IllegalState Exception	getBookmarks	BoardDto	BoardDto	String memberId, Pageable pageable	BookmarkEntity	ROLE_USER

5. 최종 산출물 - EUREKA

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
APIGATEWAY-SERVICE	n/a (1)	(1)	UP (1) -
BOARD-SERVICE	n/a (1)	(1)	UP (1) -
CHECKLIST-SERVICE	n/a (1)	(1)	UP (1) -
COMMENT-SERVICE	n/a (1)	(1)	UP (1) -
EXERCISE-SERVICE	n/a (1)	(1)	UP (1) -
FOOD-SERVICE	n/a (1)	(1)	UP (1) -
MEMBER-SERVICE	n/a (1)	(1)	UP (1) -

5. 최종 산출물 - UI 페이지



6. 팀 회고

1.서버 환경 설정 파일 관리의 복잡함

모놀리식 구조에서는 단일 .env 파일로 모든 환경 설정을 관리했지만, 서비스가 분리되면서 각 서비스마다 개별 환경 설정 파일로 관리하였습니다. 하지만 흩어진 설정 파일로 인해 관리가 어려웠습니다. 따라서 Config Server를 도입하여 하나의 중앙 집중식 서버에서 환경 설정을 관리할 필요성을 느꼈습닌다.

2.마이크로서비스 분리 기준

기존 모놀리식에서 업무 방식 자체가 바뀌기 때문에 마이크로서비스를 어떤 기준으로 나눌지 정하기 어려웠습니다. 팀 내 논의를 통해 기존 RestController 기준으로 업무 크기를 확인하여 분리하기로 결정했습니다. 하지만 이 과정에서 많은 시간이 소요되었습니다. 이를 통해 MSA에서 업무를 나누는 것에 대한 사전 계획과 논의가 중요하다는 경험을 했습니다.