SHOUT · OUR · PASSION · TOGETHER

DO IT SOPT

6 차 세 미 나

SHOUT OUR PASSION TOGETHER
SOPT

모든 설명은 Window 10 설명 기준입니다.

01

Architecture

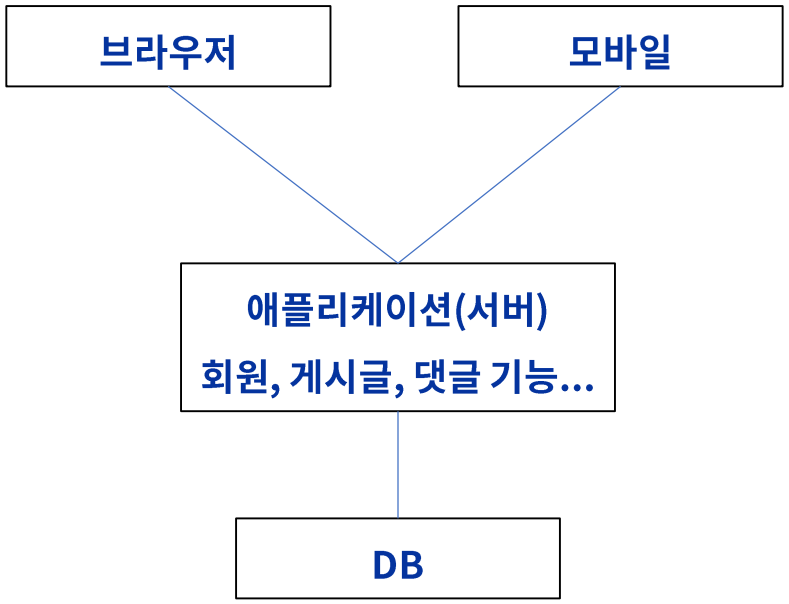
Architecture(아키텍처)

1. 시스템이 어떻게 작동하는지를 설명하는 프레임워크, 구조, 뼈대, 골격이다.
2. 시스템 구성 및 동작 원리를 나타낸다.
3. 소프트웨어 디자인 패턴과 유사하지만 더 큰 범주에 속한다.
4. 다양한 아키텍처 패턴에 대한 설명
5. https://mingrammer.com/translation-10-common-software-architectural-patterns-in-a-nutshell/

Monolithic Architecture(모놀리식 아키텍처)

1. 일체형 구조
2. 하나의 애플리케이션(서버) 안에 모든 로직이 들어가 있다.
3. 하나의 뭔가를 수정하려면 전체 애플리케이션(서버)을 다시 빌드하고 배포해야 한다.
4. 개발이 단순하고 배포 역시 단순하다.
5. 애플리케이션(서버) 구조가 커지고, 팀 규모가 성장하면 여러 단점이 발생한다.
6. 코드, 프레임워크에 강제 당하게 된다. (ex. Spring Framework를 사용한다면 나 역시 Spring Framework를 사용해 개발해야 한다.)

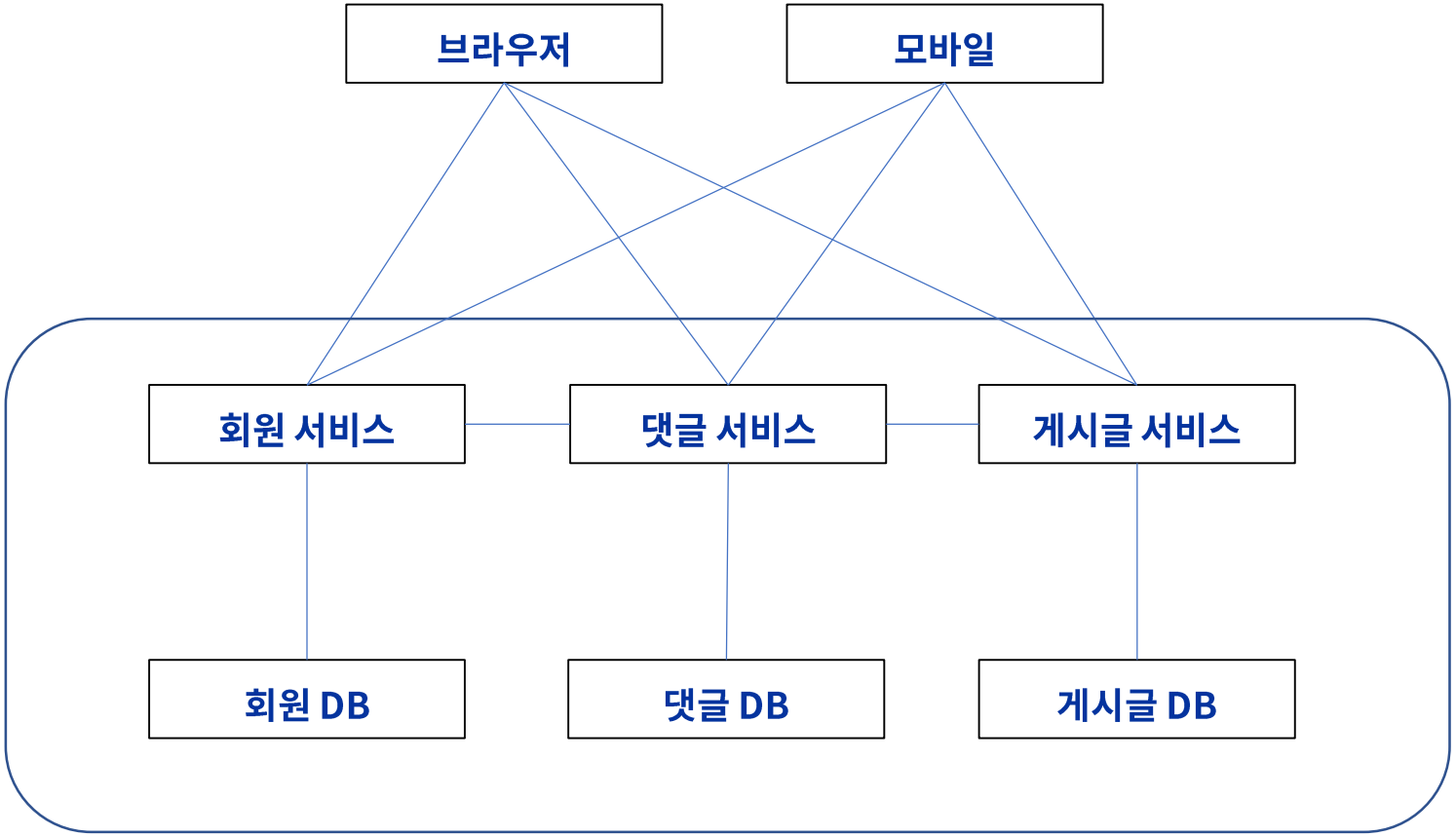
Monolithic Architecture(모놀리식 아키텍처)



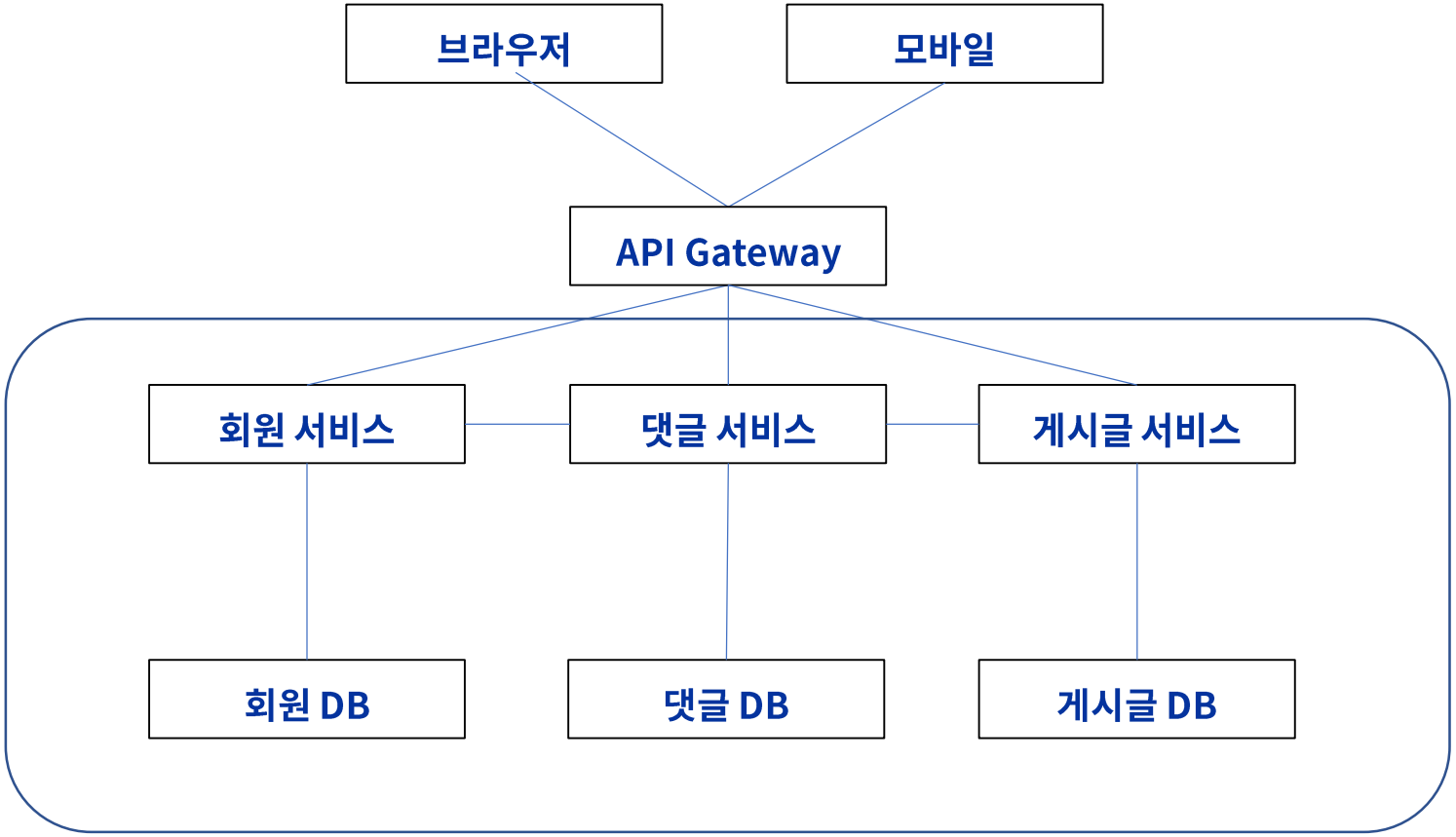
Micro Service Architecture(MSA, 마이크로 서비스 아키텍처)

1. 각 컴포넌트를 서비스라는 개념으로 정의한다.
2. REST API같은 표준 인터페이스로 그 기능을 외부로 제공한다.
3. 쉽게 생각해 각각의 서비스별로 서버로 분리하고 따로 개발 하는 것이다.
4. MSA에서는 URI/API 설계가 무척 중요하다.
5. 애플리케이션 로직을 분리해서 여러 개의 애플리케이션으로 나눠서 배포한다.
6. API Gateway를 사용하면 모든 API에 대해 End Point를 통합함으로써 다른 서비스, 클라이언트 간의 통신하는 것에 도움을 준다.
7. 개발에 있어서 언어/프레임워크에 구속 받지 않게 된다.
8. 설계가 무척 중요하기 때문에 초기 프로젝트 설계 시간이 오래 걸린다.
9. 단시간에 개발하기에는 적합한 아키텍처는 아니다.
10. 추후 서비스 확장에 유리한 구조이다.

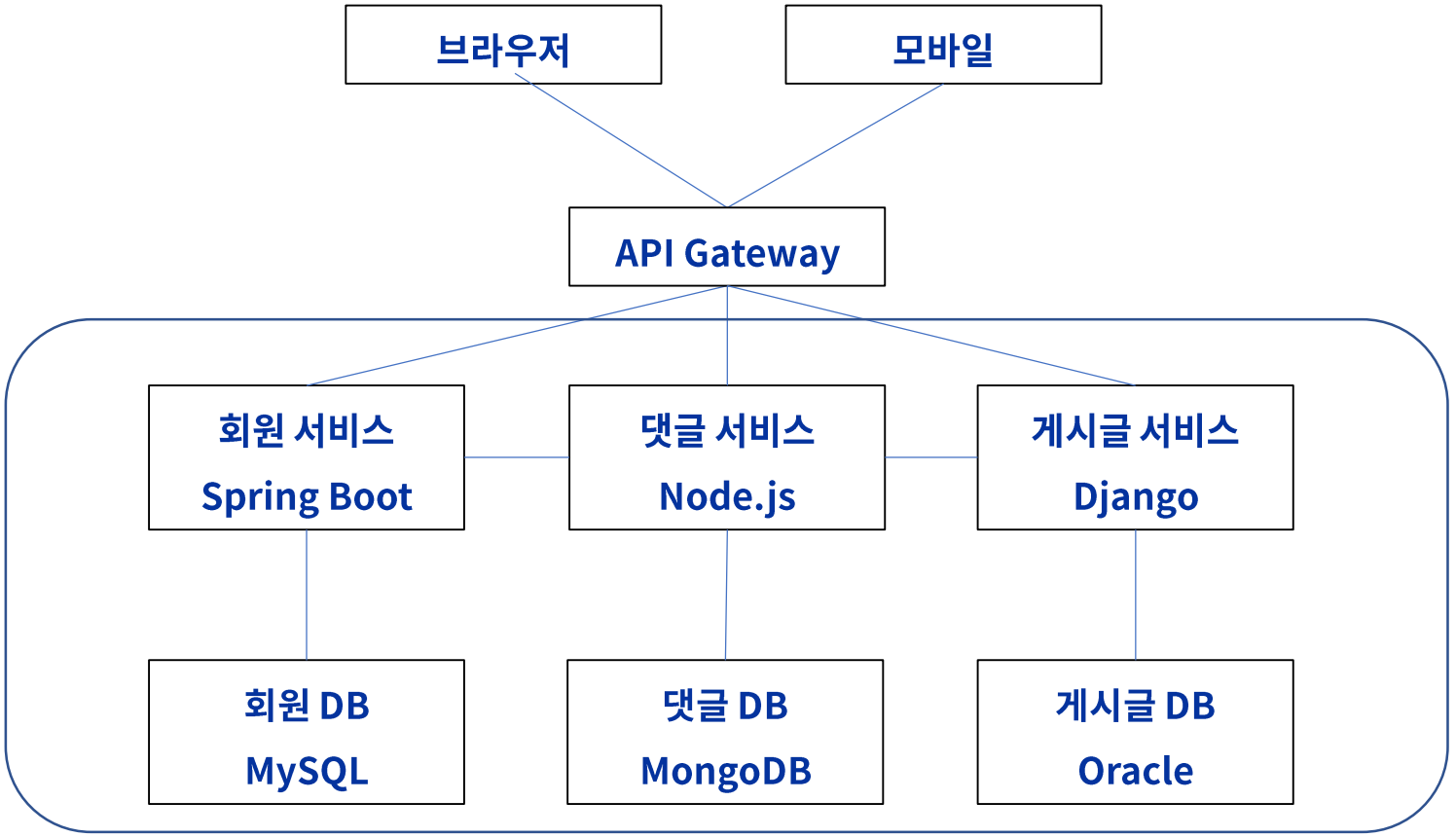
Micro Service Architecture(MSA, 마이크로 서비스 아키텍처)



Micro Service Architecture(MSA, 마이크로 서비스 아키텍처)



Micro Service Architecture(MSA, 마이크로 서비스 아키텍처)



API Gateway

1. API Gateway는 프론트 엔드 혹은 다른 플랫폼이 들어오는 출입구다. 오직 API Gateway를 통해서만 정보를 얻는다. 그 내부가 어떤 구조, 어떤 통신, 스키마 설계 등은 중요하지 않다.
2. API Gateway는 각 서비스가 어떤 구성, 어떤 스키마 등을 가졌는지 내용을 알 필요가 없다. 오직 기능만 알고 있으면 된다. 다시 말해서 각 서비스에선 API Gateway에게 기능만 알려주면 될 정도로 기능이 명확해야 한다.
3. API Gateway입장에선 내부를 생각하는 것 보단 외부에서 볼 때 어떤 의미로 나뉘는지가 중요하다. 그렇지 않으면 API Gateway를 쓸 필요가 없다

02

REST API

프로젝트 단계

1. 프로젝트 기능 확정
2. 기능별 개발 우선순위 부여 및 역할 분배
3. URI 설계 & DB 설계
4. URI 설계를 토대로 API 설계
5. 작업 시작...
6. 기능 테스트
7. API 문서 작성
8. 클라이언트 연동
9. 버그 수정

URI(Uniform Resource Identifier, 통합 자원 식별자)

1. 자원을 나타내는 유일한 주소이자 자원을 식별할 수 있는 문자열이다.
2. 하위 개념으로 URL, URN이 있다.
3. 어떤 자원의 위치를 의미하고, Method가 그 위치에 대한 행위를 뜻한다.
4. 네이밍이 직관적이어야 한다.
5. 형용사보다는 명사가 이해하기 좋고 복수 형태로 만든다.
6. URI 설계에 정답은 없지만 항상 일관된 규칙으로 작성해야 혼동하지 않는다.

USERS – URI 예시

USERS

메소드	경로	설명
GET	/users	모든 회원 조회
GET	/users/{userIdx}	회원 조회
POST	/users	회원 가입
PUT	/users/{userIdx}	회원 수정
DELETE	/users/{userIdx}	회원 삭제

Cowalker – URI 예시

Users

메소드	경로	설명
GET	/users/{user_idx}	회원 조회
POST	/users	회원 가입
PUT	/users/{user_idx}	회원 정보 수정
DELETE	/users/{user_idx}	회원 탈퇴
GET	/users/check?email=bghgu	이메일 중복 확인

Cowalker – URI 예시

RECRUIT

메소드	경로	설명
GET	/project/{project_idx}/recruit	모집 공고 전체 조회
POST	/project/{project_idx}/recruit	모집 공고 작성
PUT	/project/{project_idx}/recruit/{recruit_idx}	모집 공고 수정
DELETE	/project/{project_idx}/recruit/{recruit_idx}	모집 공고 삭제
GET	/project/{project_idx}/recruit/{recruit_idx}	모집 공고 세부 조회

APPLY

Cowalker – URI 예시

메소드	경로	설명
GET	/users/{user_idx}/apply	내가쓴 자소서 조회
GET	/users/{user_idx}/apply/{apply_idx}	내가쓴 자소서 세부 조회
POST	/users/{user_idx}/apply	자소서 생성
PUT	/users/{user_idx}/apply/{apply_idx}	자소서 수정
DELETE	/users/{user_idx}/apply/{apply_idx}	자소서 삭제
GET	/recruit/{recruit_idx}/apply	공고의 자소서 조회
GET	/recruit/{recruit_idx}/apply/{apply_idx}	공고의 자소서 세부 조회

PROJECT 생성

Cowalker – URI 예시

메소드	경로	설명
POST	/users/{user_idx}/project	프로젝트 생성
PUT	/users/{user_idx}/project/{project_idx}	프로젝트 수정
DELETE	/users/{user_idx}/project/{project_idx}	프로젝트 삭제

탐색 & 검색

메소드	경로	설명
GET	/project?aim=창업&area=서울&position=PM&department=블록체인&keyword=검색어	프로젝트 조회
GET	/project/{project_idx}	프로젝트 세부 조회

Cowalker – URI 예시

메시지

메소드	경로	설명
GET	/message/users/{user_idx}	사용자가 주고 받은 메시지 리스트 조회
GET	/message/users/{user_idx}/partner/{user_idx}	상대방과 주고 받은 메시지 조회
POST	/message/users/{user_idx}/partner/{user_idx}	상대방에게 메시지 전송
DELETE	/message/users/{user_idx}/partner/{user_idx}	메시지 삭제

알람

메소드	경로	설명
GET	/users/{user_idx}/alarm?count=3	알람 조회

상품결제 – URI 예시

Products

메소드	경로	설명
GET	/products?offset={offset}&limit={limit}&sort={sort}&order={order}	상품 목록 조회
GET	/products/{productIdx}	상품 정보 조회

Payments

메소드	경로	설명
GET	/payments?offset={offset}&limit={limit}&sort={sort}&order={order}	결제 목록 조회
GET	/payments/{paymentIdx}	결제 내역 조회
POST	/payments	상품 결제

서울시 앱 공모전 – URI 예시

Places

메소드	경로	설명
GET	/places	모든 장소 조회
GET	/places/{place_id}	특정 장소 조회

Stamps

메소드	경로	설명
GET	/users/{user_id}/stamps	사용자의 스탬프 조회
GET	/users/{user_id}/stamps/{stamp_id}	스탬프 상세 조회
POST	/users/{user_id}/places/{place_id}/stamps	스탬프 작성

서울시 앱 공모전 – URI 예시

Ranks

메소드	경로	설명
GET	/ranks	실시간 순위 조회

Comments

메소드	경로	설명
GET	/places/{place_id}/comments	해당 장소의 댓글들 조회
GET	/comments/{comment_id}	댓글 조회
POST	/users/{user_id}/comments/{comment_id}	댓글 작성

URL(Uniform Resource Locator, 통합 자원 지시자)

1. 네트워크 상에서 해당 자원이 어디 있는지를 알려주기 위한 규약이다.
2. 특정 서버의 한 자원에 대한 구체적인 위치를 나타낸다.
3. 127.0.0.1:8080/users?name=배다슬은 URI
4. 127.0.0.1:8080/users/배다슬/profile.jpg 는 URL이면서 URI

Typora

1. 마크다운 문법 편집기(.md)
2. Git의 readme.md 파일을 작성할 수 있다.

Markdown 문법

```
# 제목1

## 제목2

### 제목3

#### 제목4

##### 제목5

###### 제목6
```

제목1

제목2

제목3

제목4

제목5

제목6

Markdown 문법

```
* 목록
    * 세부목록
        * 세부 세부 목록

**강조**

```김현진 최고```

```
운영팀장 최고
```

* 이것은 수평선

```

- 목록
    - 세부목록
        - 세부 세부 목록

**강조**

`김현진 최고`

```
운영팀장 최고
```

- 이것은 수평선

```

Markdown 문법

```
* 표도 만들수 있다.

| 첫행 | 첫행 | 첫행 |
| ---- | ---- | ---- |
| 박다예 | 이상윤 | 송지은 |
| 권서연 | 강수진 | 이민형 |
```

- 표도 만들수 있다.

첫행	첫행	첫행
박다예	이상윤	송지은
권서연	강수진	이민형

마크 다운 파일 양식

API 문서 예시

https://github.com/bghgu/SOPT-23-Server/blob/master/6%EC%B0%A8/api%20%EC%98%88%EC%8B%9C.md

URI 파일 예시

https://github.com/bghgu/SOPT-23-Server/blob/master/6%EC%B0%A8/uri%20%EC%98%88%EC%8B%9C.md

readme 파일 예시

https://github.com/bghgu/SOPT-23-Server/blob/master/6%EC%B0%A8/readme%20%EC%98%88%EC%8B%9C.md

URI 설계 실습

기능

1. 컨텐츠(글) CRUD
2. 팔로잉(Following) & 팔로워(Followers)
3. 일단 로그인(Auth) 기능은 배제하고 설계한다.

Follow

1. 팔로우(Follow) 기능을 사용하여 다른 회원의 소식을 만날 수 있다.
2. 팔로우(Follow) : 특정 대상을 Follow 하는 경우 그 대상의 소식을 만날 수 있다.
3. 팔로잉(Following) : 나 또는 특정 대상이 Follow 하는 사람
4. 팔로워(Followers) : 나 또는 특정 대상을 Follow 하는 사람

API(Application Programming Interface)

1. 서버 애플리케이션의 기능을 사용하기 위한 수단/방법
2. 프론트 엔드는 다양한 API를 사용해 서버의 기능을 사용한다.

URI & API

1. URI 가 서버 설계 도면이라면 API는 서버 사용 설명서
2. URI는 서버의 구성 요소들을 나타낸다.
3. API를 사용해 서버의 내부 구조, 정보 등을 몰라도 서버의 기능을 사용할 수 있다.
4. URI & API 둘 다 명확하고 직관적으로 구성해야 다른 사람이 볼 때 문제가 없다.

REST(REpresentational State Transfer) API

1. 브라우저 뿐만 아니라 모바일에서도 통신할 수 있어야 한다.
2. Resource-URI, Method, Message 3가지 요소로 구성된다.
3. 모든 것을 자원, 명사로 표현한다.
4. Self-Descriptiveness(자체 표현 구조) : REST API 메시지만 보고도 이를 쉽게 이해 할 수 있는 자체 표현 구조로 되어 있다.
5. Stateless(무상태성) : 상태 정보를 저장하지 않고 각 API 서버는 들어오는 요청만을 들어오는 메시지로 처리하면 된다.

HTTP 메소드

Method	설명	CRUD
GET	조회	Select
POST	생성	Create
PUT	수정	Update
DELETE	삭제	Delete

HTTP 상태 코드

Code	Name	설명
200	Ok	요청을 성공적으로 처리했다.
201	Created	새 자원을 저장했다.
204	No Content	요청을 성공적으로 처리했지만 컨텐츠를 제공하지 않는다.
400	Bad Request	서버가 요청을 이해하지 못하고 있다.
401	UnAuthorized	인증이 필요하다.
403	Forbidden	요청을 거부하고 있다. 권한이 없다.
404	Not Found	자원, 페이지를 찾을 수 없다.
500	Internal Server Error	서버 내부 오류
503	Service Unavailable	일시적으로 서비스를 사용할 수 없다.

API 문서

1. 클라이언트에게 API 명세서를 제공해야 한다.
2. 다양한 방법으로 제공할 수 있다.
3. github wiki에 작성할 경우 swagger보다 작업량은 늘어나지만 서버를 구동하지 않아도 언제든지 API를 볼 수 있다.
4. 엑셀에 작성할 경우 별도의 파일/구글 드라이브 주소를 공유해 줘야 한다.
5. git wiki에 작성해서 나중에 포트폴리오로도 활용할 수 있다.
6. 항상 API 문서는 누가 봐도 이해 할 수 있게 명확하고 직관적으로 적어야 한다.
7. github wiki는 마크다운 문법으로도 작성이 가능하다.
8. 다양한 방법이 있다.

API 문서의 구성 요소

1. API 이름
2. 메소드(HTTP Method)
3. 경로(URL)
4. 요청 헤더(Request Header)
5. 요청 바디(Request Body)
6. 응답 바디(Response Body)

요청과 응답

1. 요청 헤더(Request Header)에는 토큰을 필요로 할 수도 있다.
2. 토큰이 있다면 로그인을 했다는 의미이고, 토큰이 없다면 로그인을 하지 않았다는 의미이다.
3. 요청 바디(Request Body)에 따라 응답이 바뀔 수 있다.
4. 응답 바디(Response Body)는 다양한 예외를 가질 수 있다.
5. 토큰의 유무(로그인의 유무)에 따라 응답 데이터가 바뀔 수 있다.(ex. 글에 대한 수정, 삭제 권한, 프로필 변경 권한 등등..)

회원 정보 수정

회원 정보 수정 API - 1

메소드	경로	짧은 설명
PUT	/users/{userIdx}	회원 정보 수정

요청 헤더

```
Content-Type: multipart/form-data
Application: token
```

요청 바디

- 요청하지 않은 값은 자동으로 이전 값으로 반영

```
{
    "name" : "테스트",
    "email" : "2",
    "part" : "서버",
    "profile" : "파일"
}
```

응답 바디 API 문서

회원 정보 수정 API - 2

회원 정보 수정 성공


```
{
    "status": 200,
    "message": "회원 정보 수정 성공",
    "data": {
        "u_id": 1,
        "u_name": "배다솔",
        "u_part": "서버",
        "u_profile": "https://s3.ap-northeast-2.amazonaws.com/sopt-23-
        "u_email": "1",
        "auth": true
    }
}
```


회원 정보 수정 실패


```
{
    "status": 400,
    "message": "회원 정보 수정 실패",
    "data": null
}
```

회원 정보 수정 API - 3

인증 실패

```
{
    "status": 401,
    "message": "인증 실패",
    "data": null
}
```

다른 회원 정보 수정 시도

```
{
    "status": 403,
    "message": "인가 실패",
    "data": false
}
```

회원 정보 수정 API - 4

DB 에러

```
{
    "status": 600,
    "message": "데이터베이스 에러",
    "data": null
}
```

INTERNAL SERVER ERROR

```
{
    "status": 500,
    "message": "서버 내부 에러",
    "data": null
}
```

원본 링크

API 문서 예시

1. SOPT 23 클라이언트 파트 통신 연결 테스트 API
2. https://github.com/bghgu/SOPT-23-API-SERVER/wiki
3. 2018 서울시 앱 공모전 잘생겼다 서울 API
4. https://github.com/project-handsomeGo/project-handsomeGo-Server/wiki
5. SOPT 22 App-Jam Cowalker API
6. https://github.com/project-cowalker/project-cowalker-server/wiki
7. SOPT 20 App-Jam INDIVUS API
8. https://github.com/bghgu/project_INDIVUS_rest_api/wiki

API 문서 작성 실습

기능

1. 컨텐츠(글) CRUD
2. 글에는 사진을 포함한다.
3. 로그인(Auth) 기능까지 생각하고 작성한다.
4. 모든 상황은 가정하고 작성한다.(글이 존재 하지 않는다, DB 에러, 등등..)

03

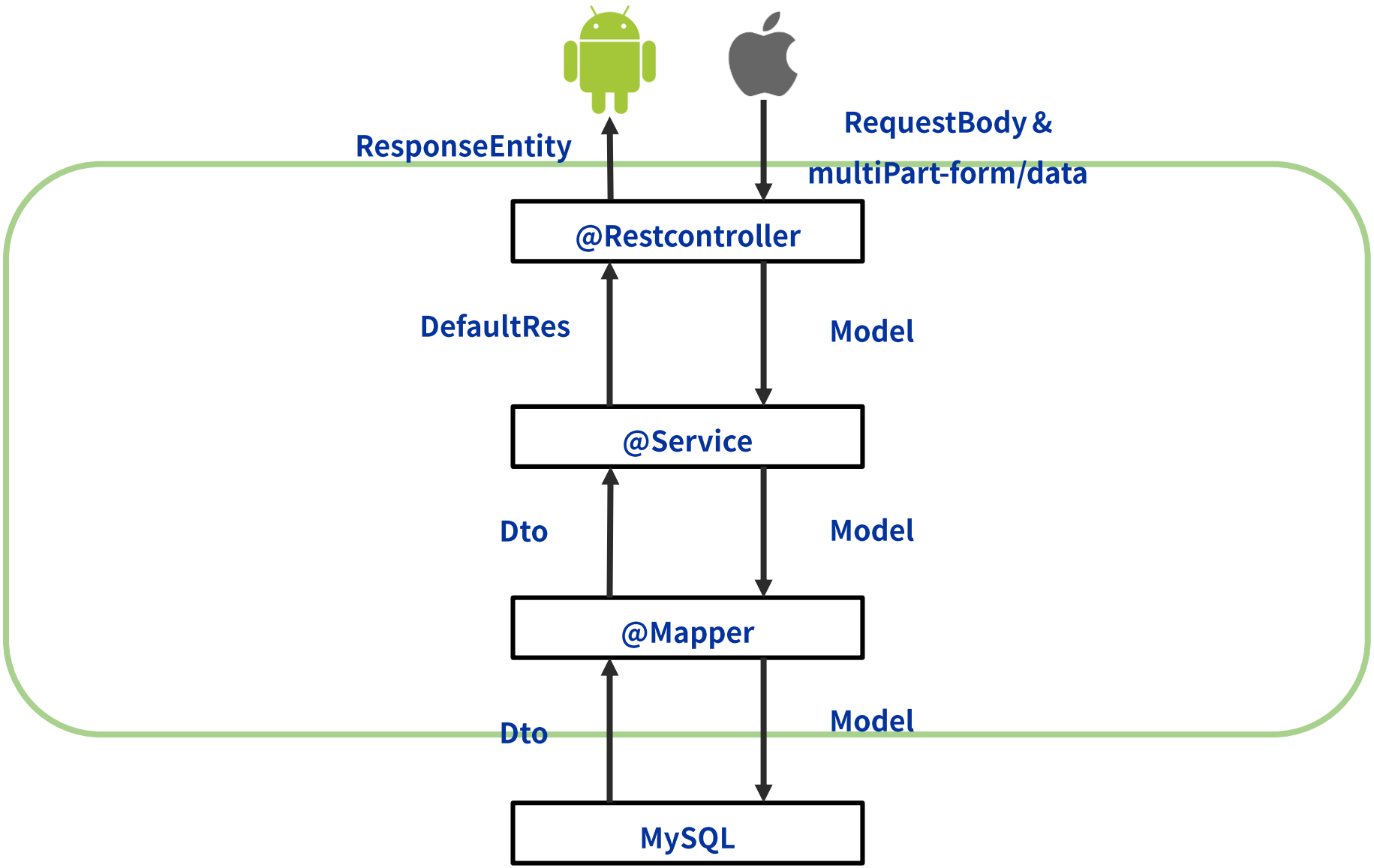
복 습

1. @RestController & @Service & @Mapper가 하는 일

1. @RestController
    1. 로그인 확인 (@Auth)
    2. URL에 따른 API 맵핑
    3. 파라미터 확인, 파일 업로드라면 파일 확인
    4. 서비스 호출
    5. 자원에 대한 권한 확인
    6. ResponseEntity 반환
2. @Service
    1. 실제 기능 구현
    2. 의미 : 공통 기능, 싱글톤, node의 require의 모듈, static, 컴포넌트, 모듈화, 부품화
    3. 생성자 의존성주입을 사용해 컨트롤러에서 서비스를 이용한다.
3. @Mapper
    1. @Select, @Insert, @Update, @Delete와 SQL을 사용해 DB를 사용
    2. 생성자 의존성주입을 사용해 서비스에서 맵퍼를 이용한다.

1. @RestController & @Service 나누는 이유

1. 사실 모든 기능을 @RestController에서 구현 해도 상관 없다.
2. 이렇게 하면 시간도 단축되고 신경 쓸 것이 많이 없어진다.
3. 그렇지만 굳이 @Service를 만들어서 나눠주는 이유는 분명 중복되는 코드가 생기기 때문이다.
    1. ex. 회원 정보 조회
    2. 회원 고유 번호로 회원의 정보를 조회하는 코드는 게시글, 댓글, 프로필조회 등등 다양한 곳에서 사용한다.
    3. 이 코드를 그때 그때 작성하기엔 너무 너무 비 효율 적이다.
4. 중복 되는 코드가 발생하면 따로 모듈화를 해 나눠주는 것이 후에 유지 보수하기에도 더 쉬워질 수 있다.
5. 이렇게 모든 기능들을 세분화해서 @Service에 작성하게 되면 나중에는 서비스의 기능들을 조합만 해서 새로운 기능을 만들 수도 있다.
6. 서비스에서 다른 서비스를 의존성 참조 하는 것도 가능하다.


ResponseEntity
RequestBody &
multiPart-form/data
@Restcontroller
DefaultRes
Model
@Service
Dto
Model
@Mapper
Dto
Model
MySQL

2. ResponseEntity

1. ResponseEntity의 상태코드는 항상 HttpStatus.OK(200)로 맞춰줘야 한다.
2. try-catch의 에러는 HttpStatus.INTERNAL_SERVER_ERROR(500)로 맞춰줘야 한다.

```
@PostMapping("login")
public ResponseEntity login(@RequestBody final LoginReq loginReq) {
    try {
        return new ResponseEntity<>(authService.login(loginReq), HttpStatus.OK);
    } catch (Exception e) {
        log.error(e.getMessage());
        return new ResponseEntity<>(FAIL_DEFAULT_RES, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Login API의 예시

2. ResponseEntity

1. ResponseEntity의 진짜 상태코드는 DefaultRes의 status에 넣어준다.
2. 모든 응답에 항상 한국말로 메시지를 넣어준다.(message)
3. 응답 데이터가 있다면 제네릭 타입을 이용해 data에 넣어준다.

DefaultRes 코드

DefaultRes 소스코드 링크

```
@Setter
@Getter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class DefaultRes<T> {

    private int status;

    private String message;

    private T data;

    public DefaultRes(final int status, final String message) {
        this.status = status;
        this.message = message;
        this.data = null;
    }

    public static<T> DefaultRes<T> res(final int status, final String

    public static<T> DefaultRes<T> res(final int status, final String
        return DefaultRes.<T>builder()
                .data(t)
                .status(status)
                .message(message)
                .build();
    }

    public static final DefaultRes FAIL_DEFAULT_RES = new DefaultRes(
}
```

2. ResponseEntity의 상태코드는 200/500으로 하고 진짜 상태코드는 DefaultRes의 status에 넣어 주는 이유

1. 프론트엔드(Android/iOS)는 서버로 부터 받은 응답 데이터를 파싱해 객체에 저장한다.
   - 이때 파싱 데이터 형식이 다를 경우 프론트엔드는 힘들어한다.
2. 상태코드에 따라 때로는 아예 ResponseEntity가 전송 되지 않을 때도 있다.(204, NO CONTENT)
   - 이때 프론트엔드는 응답 데이터를 파싱 할 수 없기 때문에 앱이 터져버린다.
3. 프론트엔드는 상태 코드를 읽을 줄 알고 이거에 따라 다르게 파싱을 할 수 있지만 잘 안 한다.
   - 상태 코드에 따라 파싱 객체를 따로 만들 경우 프론트엔드가 할 일이 많아진다.
   - 프론트엔드는 view도 개발해야 하고 할 일이 많다.
4. 모든 응답에 대한 메시지를 한국말로 DefaultRes의 message에 넣어주면 프론트엔드와 의사 소통이 더 빠르게 진행된다.
   - 이렇게 해야 클라이언트가 에러 찾기에 더욱 수월하다.
   - 그냥 에러가 나오고 서버를 불러서 찾기에는 너무 시간도 오래 걸리고 비 효율 적이다.
   - 서버 로그를 다 뒤져서 에러 원인을 직접 찾아 줘야한다.

3. Controller try-catch

1. 우리가 제어할 수 없는 다양한 에러가 발생할 수 있다.
2. 그럴 땐 try-catch로 일단 묶어 버리고 에러 로그를 찍은 다음 발생한 뒤에 해결하자.
3. log.error(e.getMessage());

```
@PostMapping("login")
public ResponseEntity login(@RequestBody final LoginReq loginReq) {
    try {
        return new ResponseEntity<>(authService.login(loginReq), HttpStatus.OK);
    } catch (Exception e) {
        log.error(e.getMessage());
        return new ResponseEntity<>(FAIL_DEFAULT_RES, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Login API의 예시

4. 권한 확인

1. 예를 들어...
2. 내 프로필은 나만 수정할 수 있어야 한다. 수정 버튼 표시가 필요하다.
3. 다른 사람이 내 프로필을 조회할 경우 수정/삭제 버튼이 표시 되면 안된다. 권한이 없다.
4. 따라서 회원 조회 API에 해당 회원의 정보를 수정할 수 있는 권한을 표시해서 프론트 엔드에게 표시해 줘야 한다.


```
{
    "status": 200,
    "message": "회원 정보 조회 성공",
    "data": {
        "u_id": 36,
        "u_name": "배다솔",
        "u_part": "서버",
        "u_profile": "https://s3.ap-no
        "u_email": "test",
        "auth": true
    }
}
```


내가 조회 했을 때


```
{
    "status": 200,
    "message": "회원 정보 조회 성공",
    "data": {
        "u_id": 36,
        "u_name": "배다솔",
        "u_part": "서버",
        "u_profile": "https://s3.ap-nor
        "u_email": "test",
        "auth": false
    }
}
```


타인이 조회 했을 때

04

차후 일정 안내

7차 세미나

1. 클라이언트 합동 세미나
2. 특정한 기능을 당일 공지하고, 해당 기능에 대한 DB 설계부터 API 문서 작성 까지 세미나 시간 동안 개발
3. 다 완성 못해도 됩니다.

8차 세미나

1. Spring Data JPA
2. Interceptor
3. 그동안 했던 거 총 정리
4. 정상적으로 출석 합니다.

앱잼 일정

1. 한 기수동안 진행한 세미나 내용을 바탕으로, 기획자/개발자/디자이너가 한 팀을 이뤄 하나의 애플리케이션을 협업해서 만드는 한 기수 마지막 프로젝트
2. **SOPT를 하는 이유 – 세미나 & 앱잼**
3. 앱잼 참가 신청 : 11월 24일 토요일 18:30 ~ 12월 2일 일요일 23:59
4. 접수 방식 : 접수 시작 시 신청 링크 공지 예정
5. 네트워킹 데이 진행 : 12월 8일 토요일 12:00 ~ 14:00 (7차 세미나 시작 전)
6. 네트워킹 데이 시작 전 시드 공개(A시드 & 일반)
7. 자율 팀 빌딩 기간 : 12월 8일 토요일 ~ 12월 21일 금요일 18:00
8. 기획안 경선 : 12월 22일 토요일 12:00 ~ 14:00 서울 창업 허브 10층 대강당
9. 데모데이 : 12월 22일 토요일 14:00 ~ 18:00 서울 창업 허브 10층 대당강
10. 앱잼 시작 : 12월 22일 토요일 ~ 2019년 1월 12일 토요일
11. 최종 발표 : 2019년 1월 12일 토요일 12:00 ~ 18:00 MARU180 지하 1층 이벤트홀

앱잼 팀 빌딩 하자 마자 해야 할 일 & 서버 파트장 전달 사항

1. 기획자 기선 제압
2. 기능 축소 및 명확한 기능 확정 및 기능 개발의 우선 순위를 정한다.
3. 기능의 우선 순위대로 모든 파트원이 개발 싱크를 맞춰야 한다.
4. 앱잼 프로젝트 조에서 서버 파트원의 위치를 알려야 한다.
5. 모든 기능/뷰의 수정/변경/추가는 꼭! 서버와 함께 논의하도록 강요한다.
   - 그렇지 않으면 퇴근/자고 나서 숙소에 왔는데 기능이 변경되어서 DB구조를 바꿔야 하는 일이 발생할 수도 있다.
6. 기획자가 힘들게 할 경우 언제든지 서버 파트장을 불러라.
7. git Organization을 생성하고, 서버, 안드, iOS 파트장을 Collaborator로 추가한다.
8. 항상 힘들고 어렵고 두렵고 짜증날 땐 서버 파트장을 소환한다.
9. 상태 코드표를 출력해서 붙여 놔라
10. 프론트 엔드 개발자한테 지지마라
11. 프론트 엔드 개발자에게 지지 않기 위해 항상 꼭 API 문서는 최신화해라
12. POST MAN 되면 되는 거다.
13. 기획자가 변경점을 통보 할 경우 강하게 따져라
14. A 시드도 사람이다. 너무 많이 힘들게 하지 말고 서버 파트장을 불러라
15. 마감 1주일 전에는 그 무엇도 바꾸면 안된다. 아무도 못한다. 싸움의 시작이다.

앱잼 팀 빌딩 하자 마자 해야 할 일

서버 파트 앱잼 최종 제출물

1. Github 링크
2. DB 모델링 PDF & 출력물
3. API 문서 링크

앱잼 이후 일정

1. 2019년 1월 19일 ~ 1월 20일 후반기 MT

2. 2019년 1월 16일 ~ 1월 23일 서버 파트장 입후보 기간

3. 2019년 1월 26일 종무식 및 23대 임원진 선거

SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPTO

THANK U

PPT 디자인 **한승미** 세미나 자료 **배다슬**

SHOUT OUR PASSION TOGETHER
SOPT