

## 데이터베이스와 함께 CRUD 구현 위해 정리할 부분

### 데이터베이스 연동을 위해 CRUD 구현을 위해 수정 할 부분들

1. 먼저 Service 와 Controller 파일에서 로직들을 다 수정해야 하기 때문에  
원래 있던 부분들을 주석 처리 합니다.

2. 이제는 메모리에 데이터 저장이 아니니 Service에 board 배열을 지워줍니다.

3. 게시물 데이터를 정의하는데 Entity를 이용하기 때문에 Board Model 파일에 있는 Board Interface는 지워주겠습니다.

4. 하지만 Status Enum 은 아직도 필요하기 때문에 이 부분만을 위한 파일을 생성해서 넣어줍니다.

- board.model.ts 파일 지운 후 board-status.enum.ts 생성

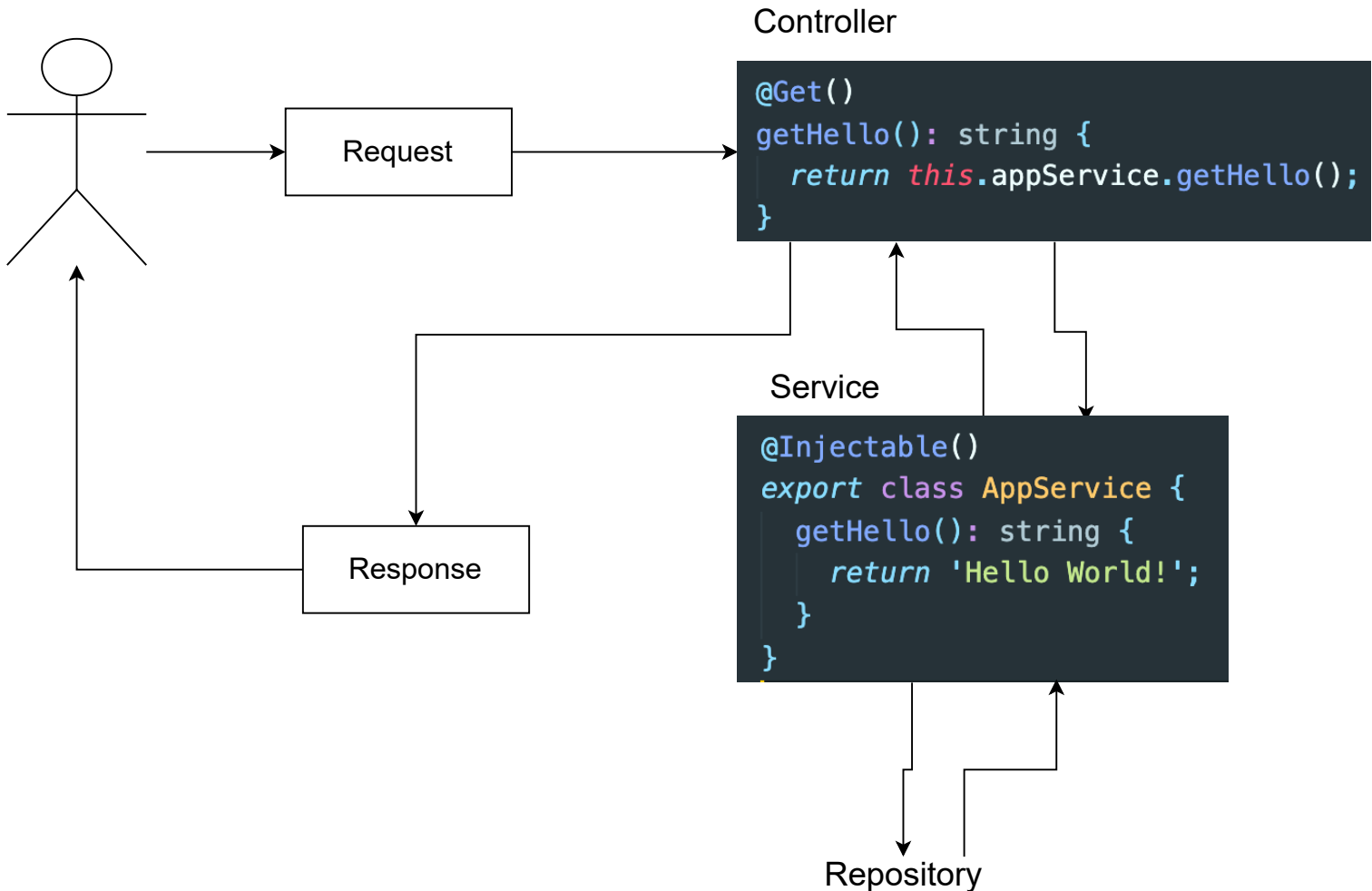
5. 데이터베이스 이용으로 인한 불필요한 경로 지워주기

`board-status-validation.pipe.ts` `BoardStatus`



## 아이디를 이용해서 특정 게시물 가져오기

이제는 메모리에서 데이터를 가져오는게 아닌 데이터베이스에서 가져오고 TypeORM을 쓸 때는 Repository 패턴을 사용한다고 했기 때문에 Board 서비스(service)에 Board 리포지터리 (Repository)를 넣어주겠습니다.(Inject)



## Service에 Repository 넣어주기 (Repository Injection)

```
@Injectable()
export class BoardsService {
    //Inject Repository to Service
    constructor(
        @InjectRepository(BoardRepository)
        private boardRepository: BoardRepository,
    ) {}
}
```

@InjectRepository

- 이 데코레이터를 이용해서 이 서비스에서 BoardRepository를 이용한다고 이걸 boardRepository 변수에 넣어줍니다.

## Service에서 getBoardById 메소드 생성하기



- typeOrm 에서 제공하는 findOne 메소드 사용하기
- async await을 이용해서 데이터베이스 작업이 끝난 후 결과값을 받을 수 있게 해주기



```
async getBoardById(id: number): Promise <Board> {  
    const found = await this.boardRepository.findOne(id);  
  
    if(!found) {  
        throw new NotFoundException(`Can't find Board with id ${id}`)  
    }  
  
    return found;  
}
```

## Controller 부분도 수정하기

```
@Get('/:id')  
getBoardById(@Param('id') id: number): Promise<Board> {  
    return this.boardsService.getBoardById(id);  
}
```

GET	localhost:3000/boards/234					
Params	Auth	Headers (7)	Body	Pre-req.	Tests	Settings
Query Params						
	KEY				VALUE	

Body   404

PrettyRawPreviewVisualizeJSON  

```
1 {  
2   ... "statusCode": 404,  
3   ... "error": "Not Found",  
4   ... "message": "Can't find Board with id 234"  
5 }
```

---

## 게시물 생성하기

이번에는 게시물을 생성하는 부분을 처리해보겠습니다.

### board.service.ts

```
async createBoard(createBoardDto: CreateBoardDto) : Promise<Board> {
  const { title, description } = createBoardDto;

  const board = this.boardRepository.create({
    title,
    description,
    status: BoardStatus.PUBLIC
  })

  await this.boardRepository.save(board);
  return board;
}
```

### board.controller.ts

```
@Post()
@UsePipes(ValidationPipe)
createBoard(@Body() createBoardDto: CreateBoardDto): Promise<Board> {
  return this.boardsService.createBoard(createBoardDto);
}
```


Nest JS / localhost:3000/boards

POST localhost:3000/boards

Params Auth Headers (9) Body ● Pre-req. Tests

Query Params

KEY	VALUE
-----	-------

Body 


Pretty Raw Preview Visualize JSON

```

1 {
2   ... "title": "board title",
3   ... "description": "board description",
4   ... "status": "PUBLIC",
5   ... "id": 1

```

Data Output Explain Messages Notifications

	id [PK] integer 	title character varying 	description character varying 	status character varying 
1	1	board title	board description	PUBLIC

## 데이터베이스에 관련된 로직은 Repository로 이동...

Repository Pattern에 대해서 배웠습니다.

리포지토리 패턴은 서비스에 있는 데이터베이스관련 로직을 Repository 쪽으로 모아주면 됩니다.

### board.repository.ts

```

@EntityRepository(Board)
export class BoardRepository extends Repository<Board> {
  async createBoard(createBoardDto: CreateBoardDto): Promise<Board> {
    const { title, description } = createBoardDto;






    const board = new Board();
    board.title = title;
    board.description = description;
    board.status = BoardStatus.PUBLIC;
    await board.save();

    return board;
  }
}

```

### board.service.ts

```
async createBoard(createBoardDto: CreateBoardDto): Promise<Board> {  
    return this.boardRepository.createBoard(createBoardDto);  
}
```

Data Output		Explain	Messages	Notifications
	<b>id</b> [PK] integer 	<b>title</b> character varying 	<b>description</b> character varying 	<b>status</b> character varying 
1	1	board title	board description	PUBLIC
2	2	board title	board description	PUBLIC

## 게시물 삭제하기

이번에는 게시물을 삭제하는 부분을 처리해보겠습니다.

### remove() vs delete() ?

- remove : 무조건 존재하는 아이템을 remove 메소드를 이용해서 지워야합니다 그러지 않으면 에러가 발생합니다.(404 Error)

- delete: 만약 아이템이 존재하면 지우고 존재하지 않으면 아무런 영향이 없습니다.

이러한 차이 때문에 remove를 이용하면 하나의 아이템을 지울 때 두번 데이터베이스를 이용해야하기 때문에 (아이템 유무 + 지우기) 데이터베이스에 한번만 접근해도 되는 delete 메소드를 사용해주겠습니다.

다큐멘테이션:

<https://github.com/typeorm/typeorm/blob/master/docs/repository-api.md>

### board.service.ts

```
async deleteBoard(id: number): Promise<void> {  
  const result = await this.boardRepository.delete(id);  
  
  console.log('result', result);  
}
```

### board.controller.ts

```
@Delete('/:id')  
deleteBoard(@Param('id', ParseIntPipe) id: number): Promise<void> {  
  return this.boardsService.deleteBoard(id);  
}
```

### board.service.ts

```
async deleteBoard(id: number): Promise<void> {  
  const result = await this.boardRepository.delete(id);  
  
  if(result.affected === 0) {
```



```
17(result.affected == 0) {  
    throw new NotFoundException(`Can't find Board with id ${id}`)  
}  
}
```

	<div><div><div><div></div></div></div><div><div>id</div><div>[PK] integer</div></div><div><div></div><div></div></div></div>	<div><div><div><div></div></div></div><div><div>title</div><div>character varying</div></div><div><div></div><div></div></div></div>	<div><div><div><div></div></div></div><div><div>description</div><div>character varying</div></div><div><div></div><div></div></div></div>	<div><div><div><div></div></div></div><div><div>status</div><div>character varying</div></div><div><div></div><div></div></div></div>
1	2	board title	board description	PUBLIC
2				

## 게시물 상태 업데이트하기

이번에는 게시물의 상태를 업데이트 하는 부분을 처리해보겠습니다

### board.service.ts

```
async updateBoardStatus(id: number, status: BoardStatus): Promise<Board> {  
    const board = await this.getBoardById(id);  
  
    board.status = status;  
    await this.boardRepository.save(board);  
  
    return board;  
}
```

### board.controller.ts

```
@Patch('/:id/status')  
updateBoardStatus(  
    @Param('id', ParseIntPipe) id: number,  
    @Body('status', BoardStatusValidationPipe) status: BoardStatus,  
): Promise<Board> {  
    return this.boardsService.updateBoardStatus(id, status);  
}
```

The screenshot shows a REST client interface with the following details:

- Method:** PATCH
- URL:** localhost:3000/boards/2/status
- Body Tab:** Selected, showing a JSON body: 

```
{  
  "status": "PRIVATE"  
}
```
- Response Tab:** Shows a JSON response: 

```
{  
  "id": 2,  
  "title": "board title",
```

```
4     ...."description": "board description",
5     ...."status": "PRIVATE"
6 }
```

## 모든 게시물 가져오기

이번에는 모든 게시물 가져오는 부분을 처리해보겠습니다.

### board.service.ts

```
async getAllBoards(): Promise <Board[]> {  
  return this.boardRepository.find();  
}
```

### board.controller.ts

```
@Get()  
getAllBoard(): Promise<Board[]> {  
  return this.boardsService.getAllBoards();  
}
```