NestJS Pipes

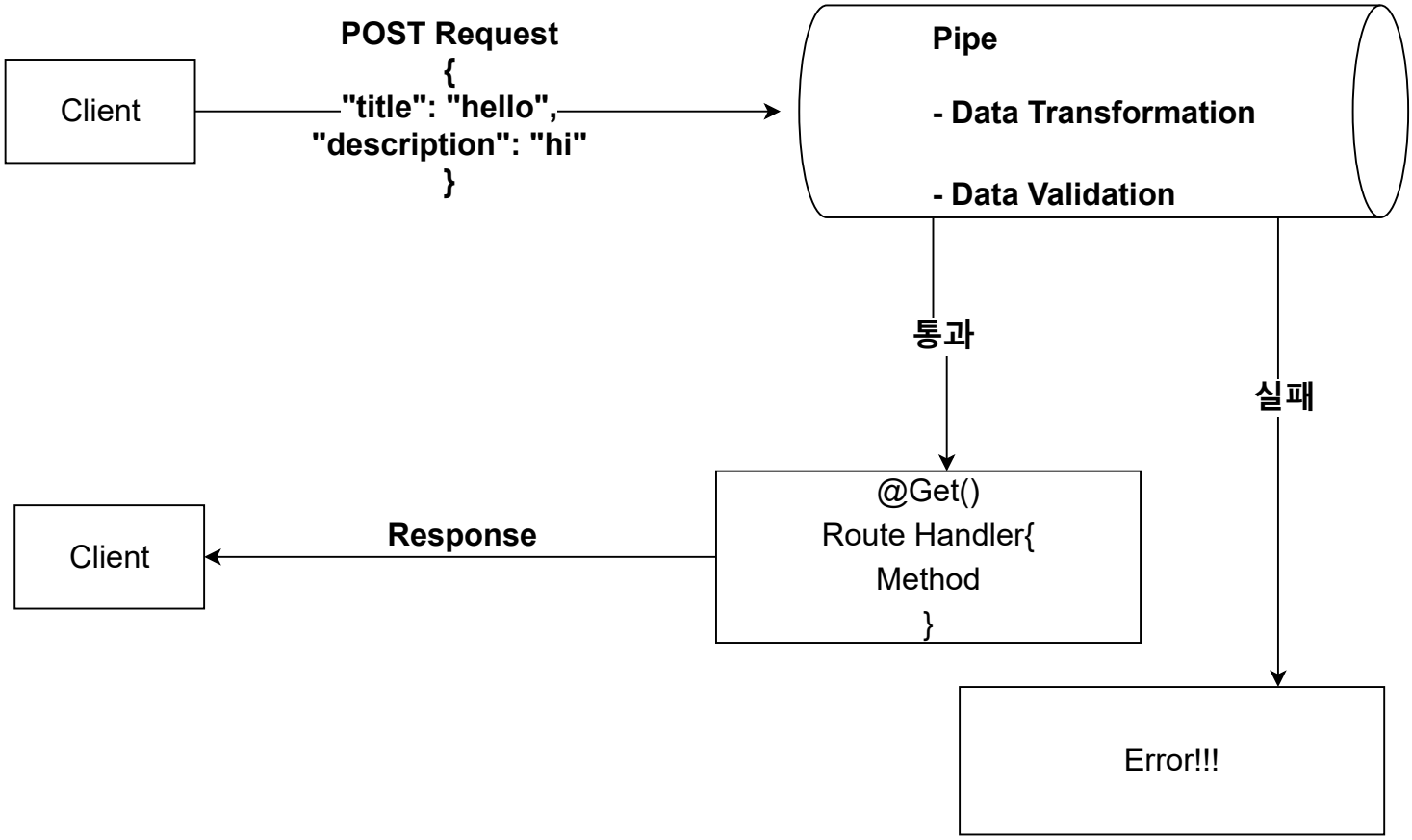
Pipe은 무엇인가요?

파이프는 @Injectable () 데코레이터로 주석이 달린 클래스입니다.

파이프는 **data transformation과 data validation을** 위해서 사용 됩니다**.**

파이프는 컨트롤러 경로 처리기에 의해 처리되는 인수에 대해 작동합니다.

Nest는 메소드가 호출되기 직전에 파이프를 삽입하고 파이프는 메소드로 향하는 인수를 수신하고 이에 대해 작동합니다.



Data Transformation?

입력 데이터를 원하는 형식으로 변환 (예 : 문자열에서 정수로)

만약 숫자를 받길 원하는데 문자열 형식으로 온다면 파이프에서 자동으로 숫자로 바꿔줍니다.

Data validation?

입력 데이터를 평가하고 유효한 경우 변경되지 않은 상태로 전달하면됩니다. 그렇지 않으면 데이터가 올바르지 않을 때 예외를 발생시킵니다.

만약 이름의 길이가 10자 이하여야 하는데 10자 이상 되면 에러를 발생시킵니다.

파이프는 위에 두가지 모든 경우에서....

라우트 핸들러(Route Handler)가 처리하는 인수에 대해서 작동합니다.

그리고 파이프는 메소드를 바로 직전에 작동해서 메소드로 향하는 인수에 대해서 변환할 것이 있느면 변환하고 유효성 체크를 위해서도 호출됩니다.

PIPE 사용하는 법(Binding Pipes)

파이프를 사용하는 방법(Binding pipes)은 세가지로 나눠질수 있습니다.
Handler-level Pipes ,Parameter-level Pipes, Global-level Pipes 입니다
이름에서 말하는 것 그대로 핸들러 레벨, 파라미터 레벨, 글로벌 레벨로 파이프 사용할 수 있습니다.

Handler-level Pipes

핸들러 레벨에서 @UsePipes() 데코레이터를 이용해서 사용 할 수 있습니다.
이 파이프는 모든 파라미터에 적용이 됩니다. (title, description)

```
@Post()
@UsePipes(pipe)
createBoard(
  @Body('title') title,
  @Body('description') description
) {

}
```

Parameter-level Pipes

파라미터 레벨의 파이프 이기에
특정한 파라미터에게만 적용이 되는 파이프 입니다.
아래와 같은 경우에는 title만 파라미터 파이브가 적용이 됩니다.

```
@Post()
createBoard(
  @Body('title', ParameterPipe) title,
  @Body('description') description
) {

}
```

Global Pipes

글로벌 파이프로서 애플리케이션 레벨의 파이브 입니다.
클라이언트에서 들어오는 모든 요청에 적용이 됩니다.
가장 상단 영역인 main.ts에 넣어주시면 됩니다.

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(GlobalPipes);
  await app.listen(3000);
}
bootstrap();
```

Built-in Pipes

Nest JS 에 기본적으로 사용할 수 있게 만들어 놓은 6가지의 파이프가 있습니다.

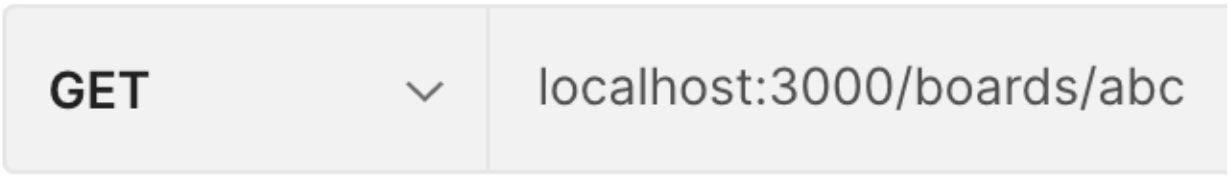
- ValidationPipe
- ParseIntPipe
- ParseBoolPipe
- ParseArrayPipe
- ParseUUIDPipe
- DefaultValuePipe

이름을 보면 각각의 파이프가 어떠한 역할을 하는지 짐작을 할 수 있습니다.
그중에서 ParseIntPipe를 이용해서 간단히 파이프를 체험해보겠습니다.

이렇게 원래는 파라미터 값으로 숫자가 와야하는 핸들러가 있습니다..

```
@Get(':id')
findOne(@Param('id', ParseIntPipe) id: number) {
  return ;
}
```

하지만 파라미터 값으로 숫자가 아닌 abc 문자열을 보냅니다...



그래서 이렇게 에러가 발생하게 됩니다....


```
{
    "statusCode": 400,
    "error": "Bad Request",
    "message": "Validation failed (numeric string is expected)"
}
```

파이프를 이용한 유효성 체크

파이프에 대해서 알아보았기 때문에
이번에는 파이프를 이용해서 게시물을 생성할 때 유효성 체크를 해보겠습니다.

필요한 모듈

class-validator , class-transformer

npm install class-validator class-transformer --save

Documentation 페이지

- https://github.com/typestack/class-validator#manual-validation

파이프 생성하기

현재는 게시물을 생성할 때 이름과 설명에 아무런 값을 주지 않아도
아무 문제 없이 생성이 됩니다. 이 부분을 파이프를 이용해서 수정해주겠습니다.

create-board.dto.ts

```ts
import { IsNotEmpty } from "class-validator";

export class CreateBoardDto {
  @IsNotEmpty()
  title: string;

  @IsNotEmpty()
  description: string;
}
```

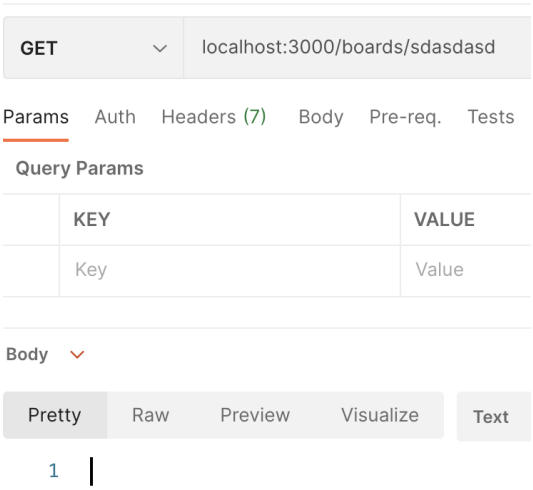
boards.controller.ts

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


```
{
    "statusCode": 400,
    "error": "Bad Request",
    "message": [
        {
            "target": {},
            "property": "title",
            "children": [],
            "constraints": {
                "isNotEmpty": "title should not be empty"
            }
        },
        {
            "target": {},
            "property": "description",
            "children": [],
            "constraints": {
                "isNotEmpty": "description should not be empty"
            }
        }
    ]
}
```

특정 게시물을 찾을 때 없는 경우 결과 값 처리

현재 특정 게시물을 ID로 가져올 때 만약 없는 아이디의 게시물을 가져오려고 한다면 결과값으로 아무 내용이 없이 돌아옵니다.
그래서 그 부분을 게시물이 없는 것이면 없다고 내용을 넣어서 클라이언트로



에러를 표출해주기 위해서는 ...

찾는 게시물이 없을 때는
예외 인스턴스를 생성해서 이용해주시면 됩니다.

```
getBoardById(id: string): Board {
  const found = this.boards.find(board => board.id === id);

  if(!found) {
    throw new NotFoundException();
  }

  return found;
}
```

이렇게 하면 아래와 같이 포맷이 잡힌 에러 문구가 나옵니다.

이번에는 에러 메시지를 넣어주겠습니다.

NotFoundException()에 텍스트를 넣어주시면 됩니다.

```
getBoardById(id: string): Board {
  const found = this.boards.find(board => board.id === id);

  if(!found) {
    throw new NotFoundException(`Can't find Board with id ${id}`);
  }

  return found;
}
```

```
{
    "statusCode": 404,
    "error": "Not Found",
    "message": "Can't find Board with id sdasdasd"
}
```

없는 게시물을 지우려 할 때 결과 값 처리

앞서 특정 게시물을 ID로 가져올 때 만약 없는 아이디의 게시물을 가져오려고 하면 그에 대한 에러 값을 전달해주었던것처럼
없는 게시물을 지우려 할 때도 에러 값을 주겠습니다.

구현 방법

이미 있는 메소드인 getBoardById를 이용해서
지우려고 하는 게시물이 있는지 체크를 해준 후에 있다면
지워주고 없다면 에러 문구를 보내주면 됩니다.

```
deleteBoard(id: string): void {
  const found = this.getBoardById(id);
  this.boards = this.boards.filter(board => board.id !== found.id);
}
```

커스텀 파이프를 이용한 유효성 체크

지금까지는 NestJS에서 이미 구성해놓은 built-in 파이프를 사용했습니다.

하지만 이것 말고도 따로 생성해서 사용할 수 있는 CUSTOM PIPE도 있습니다.
그래서 이번 시간에는 커스텀 파이프를 만들어서 사용해보겠습니다.

커스텀 파이프 구현 방법

먼저 PipeTransform이란 인터페이스를 새롭게 만들 커스텀 파이프에 구현해줘야 합니다. 이 PipeTransform 인터페이스는 모든 파이프에서 구현해줘야 하는 인터페이스입니다. 그리고 이것과 함께 모든 파이프는 transform() 메소드를 필요합니다. 이 메소드는 NestJS가 인자(arguments)를 처리하기 위해서 사용됩니다.

```
import { ArgumentMetadata, PipeTransform } from "@nestjs/common";

export class BoardStatusValidationPipe implements PipeTransform {
    transform(value: any, metadata: ArgumentMetadata) {
        console.log('value', value)
        console.log('metadata', metadata)

        return value;
    }
}
```

transform() 메소드

이 메소드는 두개의 파라미터를 가집니다.

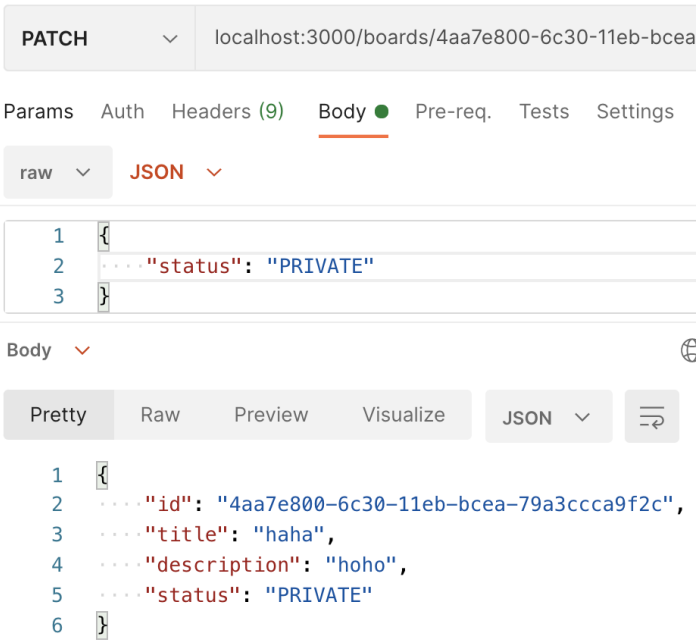
첫번째 파라미터는 처리가 된 인자의 값(value)이며
두번째 파라미터는 인자에 대한 메타 데이터를 포함한 객체입니다.

transform()메소드에서 Return 된 값은 Route 핸들러로 전해집니다.
만약 예외(Exception)가 발생하면 클라이언트에 바로 전해집니다.

실제로 value 와 metadata값 콘솔로 찍어보기

1. 커스텀 파이프 생성
2. 게시물에 업데이트하는 핸들러에 커스텀 파이프 넣어주기
3. 포스트 맨으로 요청 보내기

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




```
value PRIVATE
metadata { metatype: [Function: String], type: 'body', data: 'status' }
```

커스텀 파이프로 실제 기능 구현하기

구현 할 기능 : 상태(Status)는 PUBLIC과 PRIVATE만 올 수 있기 때문에

readonly class property

접두사(prefix) readonly는 속성을 읽기 전용으로 만드는 데 사용됩니다. 읽기 전용 멤버는 클래스 외부에서 액세스 할 수 있지만 해당 값은 변경 할 수 없습니다.

```
export class BoardStatusValidationPipe implements PipeTransform {

    readonly StatusOptions = [
        BoardStatus.PRIVATE,
        BoardStatus.PRIVATE
    ]

    transform(value: any) {
        value = value.toUpperCase();

        if (!this.isStatusValid(value)) {
            throw new BadRequestException(`${value} isn't in the status
        }

        return value;
    }

    private isStatusValid(status: any) {
        const index = this.StatusOptions.indexOf(status);
        return index !== -1;
    }
}
```