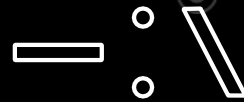


SHOUT OUR PASSION TOGETHER



PASSION

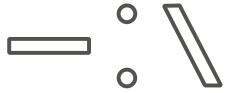
5th Seminar

EXPERIENCE

GROWTH

CHALLENGE





SHOUT OUR PASSION TOGETHER

조원

칠판

1조

2조

5조

3조

4조

6조

01 박형모 양희연 유희수 신윤재 허정민

02 김강희 박경선 박시현 박주연 양정훈 조하담

03 강영우 윤혁 이동훈 이소연 조수민 현주희

04 김해리 심다은 이소희 이재용 신정아 지현이

05 김민준 김채린 박승완 이시연 이재현 황채연

06 남궁권 양시연 양희찬 양승희 이다현 제갈윤

01 안내사항

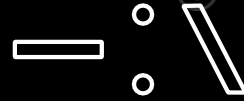
02 Multer

03 Multer with S3

04 JWT

05 과제 안내

SHOUT OUR PASSION TOGETHER

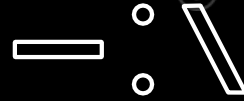


01

안내 사항

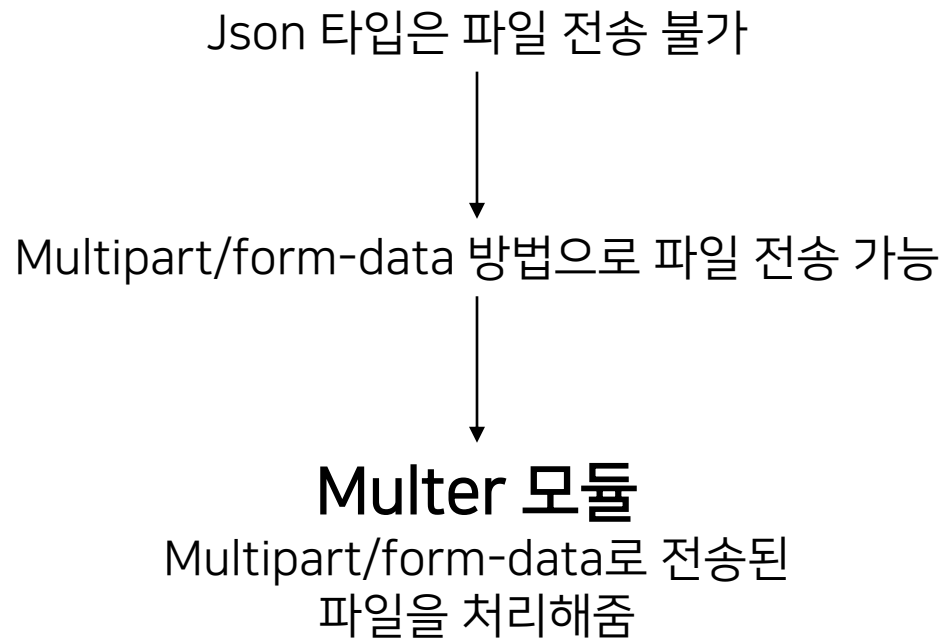
<공지>
5주차 세미나는
25기 서버 파트장의 개인사정으로
지난 기수 서버 파트장에게 진행 위임

SHOUT OUR PASSION TOGETHER



02

Multer



1. Multer 설치

```
$ npm install --save multer
```

2. MulterTest.js 라우팅 구현

localhost:3000

Routes/index.js

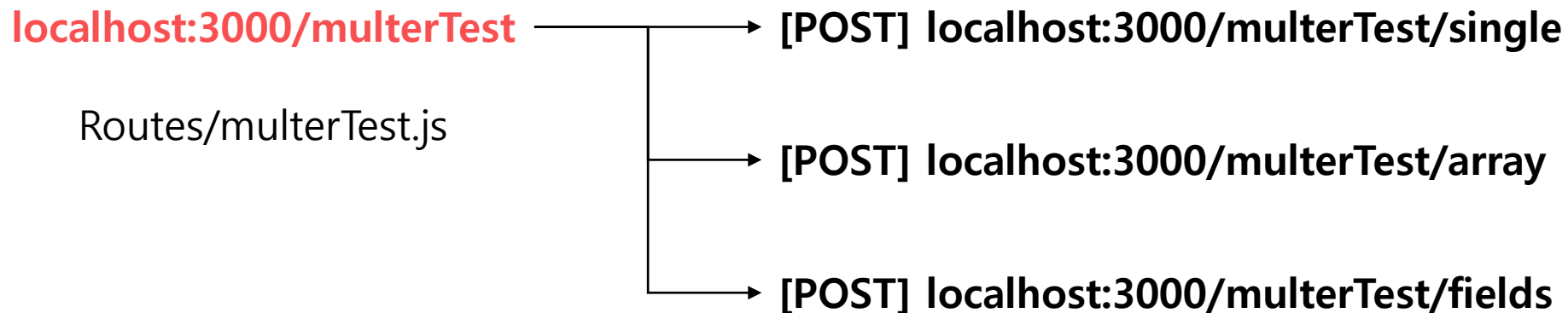
localhost:3000/multerTest

Routes/multerTest.js

요청 URL

파일

3. MulterTest.js 세부 라우팅 구현



Multer에서 3가지 메소드를 실습해볼 예정입니다.

single(fieldname)	Fieldname으로 받은 하나의 파일을 받아서 req.file에 저장
array(fieldname[, maxCount])	Fieldname으로 받은 여러 개의 파일을 받아서 req.files(배열)에 저장
fields(fields)	여러 개의 키로 받은 여러 개의 파일을 req.files(객체)에 저장

4. multerTest.js에서 multer 가져오기

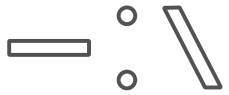
```
const multer = require('multer');  
const upload = multer({ dest: 'uploads/' });
```

Multer의 속성에서 dest는 클라이언트로 부터 받은 파일을 저장할 위치이다.

5. [POST] localhost:3000/multerTest/single 구현

```
router.post('/single', upload.single('image'), (req, res) => {  
  console.log(req.file);  
  console.log(req.body);  
  res.send({ file: req.file, body: req.body });  
})
```

Image라는 필드로 파일을 받을 받으면 req.file에 FILE객체로 저장이 됩니다.



SHOUT OUR PASSION TOGETHER

Multer 실습

6. single 테스트

The screenshot shows the Postman interface for a POST request. The URL is `localhost:3000/multerTest/single`. The request body is set to `form-data`. The body contains two parts: a file named `image` (selected as `제목 없음-1.png`) and a text field named `test` with the value `1234`. The response status is `200 OK`, time is `22ms`, and size is `480 B`. The response body is shown in JSON format:

```
1 {
2   "file": {
3     "fieldname": "image",
4     "originalname": "제목 없음-1.png",
5     "encoding": "7bit",
6     "mimetype": "image/png",
7     "destination": "uploads/",
8     "filename": "066a80f57814353eed7d40e75356dc3b",
9     "path": "uploads\\066a80f57814353eed7d40e75356dc3b",
10    "size": 13554
11  },
12  "body": {
13    "test": "1234"
14  }
15 }
```

1. METHOD를 POST로 지정
2. url 입력 localhost:3000/multerTest/single
3. Body 선택 후 form-data 선택
4. key에 image 입력
5. 이때 image 입력란에 마우스를 올리면 우측에 드롭다운 메뉴가 나타난다. File을 선택 한 이후에 values에서 파일을 선택 해준다.
6. 다음 행에 key에 test 입력 후 Value에 임의의 값 입력.
7. send

7. [POST] localhost:3000/multerTest/array 구현

```
router.post('/array', upload.array('photos', 4), (req, res) => {  
  console.log(req.files);  
  console.log(req.body);  
  res.send({ file: req.files, body: req.body });  
})
```

Photos라는 필드로 파일을 받으며 이때 최대 개수는 4개로 지정합니다.

8. [POST] localhost:3000/multerTest/fields 구현

```
var cpUpload = upload.fields([ { name: 'thumbnail', maxCount: 1 }, { name: 'images', maxCount: 8 } ])  
router.post('/fields', cpUpload, (req, res) => {  
  console.log(req.files);  
  console.log(req.body);  
  res.send({ file: req.files, body: req.body });  
})
```

Thumbnail이라는 필드와 images라는 필드 2가지 key로 파일 배열을 받습니다.

9. array 테스트

1. POST

2. localhost:3000/multerTest/array

3. Body

4. photos

5. 4 files selected

6. id

7. Send

KEY	VALUE	DESCRIPTION
photos	4 files selected	
id	abc	
Key	Value	Description

Status: 200 OK Time: 78ms Size: 1.17 KB

```

1 {
2   "file": [
3     {
4       "fieldname": "photos",
5       "originalname": "kxt77t01nq1swf999bap.jpg",
6       "encoding": "7bit",
7       "mimetype": "image/jpeg",
8       "destination": "uploads/",
9       "filename": "bb68f300b1f929d8a2ced8ff1c06cbb3",
10      "path": "uploads\\bb68f300b1f929d8a2ced8ff1c06cbb3",
11      "size": 66816
12    },
13    {
14      "fieldname": "photos",
15      "originalname": "사진1.one".
  
```

1.METHOD를 POST로 지정

2.url 입력 localhost:3000/multerTest/array

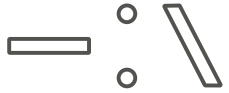
3.Body 선택 후 form-data 선택

4.key에 photos 입력한다.

5.다수의 파일을 선택

6.텍스트 필드 key, value 추가

7.send



SHOUT OUR PASSION TOGETHER

Multer 실습

10. fields 테스트

1 POST localhost:3000/multerTest/fields 7 Send

3 form-data

KEY	VALUE	DESCRIPTION
4 <input checked="" type="checkbox"/> thumbnail	사진1.png X	
5 <input checked="" type="checkbox"/> images	제목 없음-1.png X	
6 <input checked="" type="checkbox"/> test	abc	
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 29ms Size: 737 B Save

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "file": {
3     "thumbnail": [
4       {
5         "fieldname": "thumbnail",
6         "originalname": "사진1.png",
7         "encoding": "7bit",
8         "mimetype": "image/png",
9         "destination": "uploads/",
10        "filename": "acc9cb7159871f06e4abac7e8a0dbe9a",
11        "path": "uploads\\acc9cb7159871f06e4abac7e8a0dbe9a",
12        "size": 4513
13      }
14    ]
15  }
16 }
```

1.METHOD를 POST로 지정

2.url 입력 localhost:3000/multerTest/fields

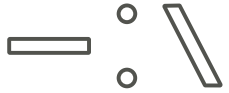
3.Body 선택 후 form-data 선택

4.key에 thumbnail 입력하고 단일 파일 선택

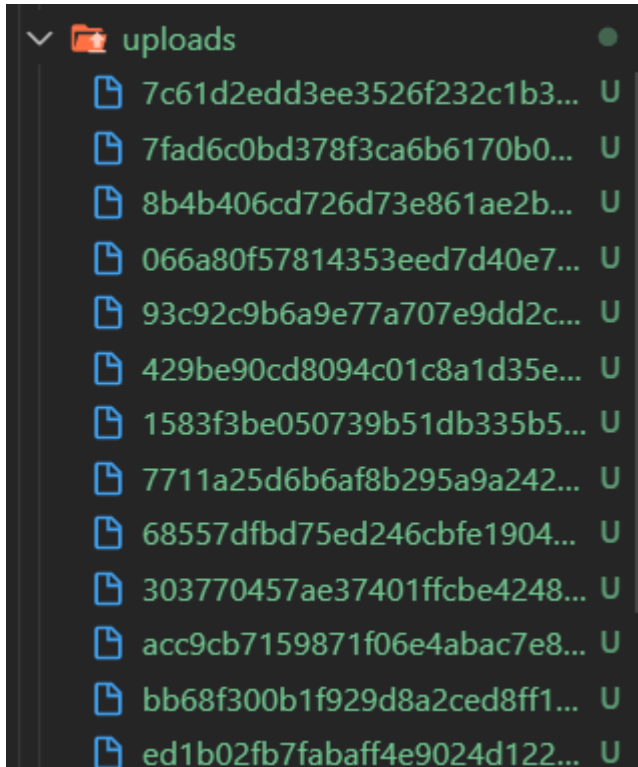
5.key에 images 입력하고 단일 복수 선택

6.텍스트 필드 key, value 추가

7.send



11. Uploads 폴더 확인



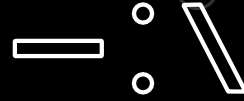
server에 파일을 저장한다면
위와 같이 파일이 생성됩니다.

12. 조별 토의

Multer 실습에서는
Console.log와 res.send를 통해서
결과 값을 확인할 수 있습니다.

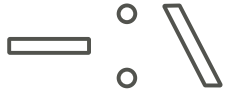
Single, array, fields 3가지 방법에 대해서
각각 어떤 특징이 있고 왜 이런 결과 나왔는지
조별 토의하는 시간을 갖겠습니다.

SHOUT OUR PASSION TOGETHER

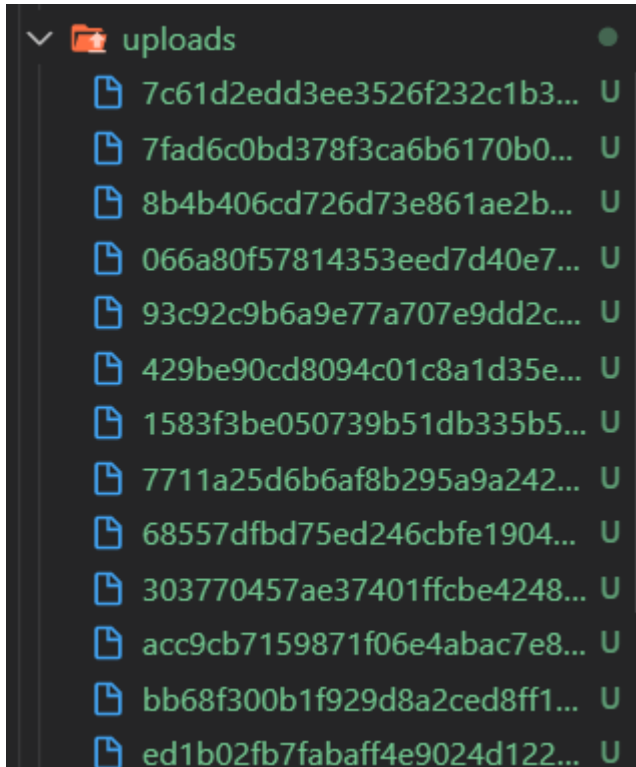


03

Multer with S3



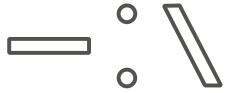
Multer의 문제점



1. EC2 서버가 가질 수 있는 용량에는 한계

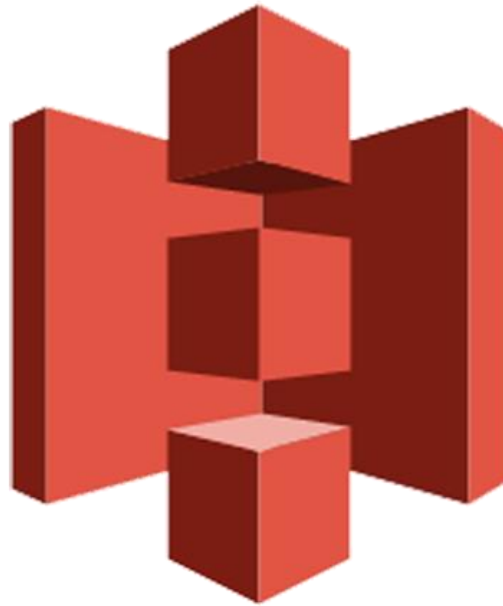
2. 빈번한 IO로 인해서 서버가 느려진다

=> 파일 저장으로 AWS의 S3를 많이 이용합니다



SHOUT OUR PASSION TOGETHER

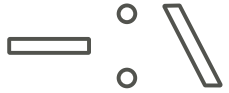
Multer with S3



Amazon S3

Amazon Simple Storage Service는
인터넷용 스토리지 서비스입니다.

이 서비스는 개발자가 더 쉽게 웹 규모 컴퓨팅 작업을 수행할 수 있도록 설계되었습니다.



SHOUT OUR PASSION TOGETHER

Multer with S3

Multer-s3 모듈

이미지 업로드 시 로컬 서버가 아닌
S3에 업로드하도록 만들어주는 모듈

AWS-SDK 모듈

AWS 서비스를 연결하기 위한 모듈

1. S3 bucket 생성

The screenshot shows the AWS S3 console interface. At the top, the AWS logo and navigation menu are visible. A red box with the number '1' highlights the 'S3' link in the left sidebar. Below the sidebar, the 'S3 버킷' (S3 Buckets) page is displayed. A red box with the number '2' highlights the '+ 버킷 만들기' (Create bucket) button. The main content area shows a message: '버킷이 없습니다. 다음은 Amazon S3를 시작하는 방법입니다.' (No buckets. Here are ways to get started with Amazon S3). Below this message are three cards: '버킷 새로 만들기' (Create new bucket), '데이터 업로드' (Upload data), and '권한 설정' (Set permissions). Each card has an icon and a brief description of the action.

1. AWS 서비스 중 S3 클릭

2. 버킷 만들기 클릭

1. S3 bucket 생성

버킷 만들기

1 이름 및 리전

2 옵션 구성

3 권한 설정

4 검토

이름 및 리전

1 버킷 이름 ⓘ

with-sopt-server

리전

아시아 태평양(서울)

기존 버킷에서 설정 복사

버킷이 없습니다.0 버킷

생성

2 다음

버킷 이름 지정하기

> 버킷 이름은 고유 값으로만 설정 가능

1. S3 bucket 생성

버킷 만들기

이름 및 리전

옵션 구성

권한 설정

검토

속성

버전 관리

☐ 동일 버킷 내에 한 객체의 모든 버전을 보관합니다. [세부 정보](#)

서버 액세스 로깅

☐ 버킷에 대한 액세스 요청을 기록합니다. [세부 정보](#)

태그

태깅을 사용하여 프로젝트 비용을 추적할 수 있습니다. [세부 정보](#)

키

값

+ 다른 항목 추가

객체 수준 로깅

☐ 추가 비용을 내고 AWS CloudTrail을 사용하여 객체 수준 API 활동을 기록합니다. 참고: [CloudTrail 요금](#) 또는 [세부 정보](#)

기본 암호화

☐ S3에 저장할 때 자동으로 객체를 암호화합니다. [세부 정보](#)

고급 설정

이전

1

다음

SHOUT OUR PASSION TOGETHER

SOPT

1. S3 bucket 생성

버킷 만들기

이름 및 리전

옵션 구성

3 권한 설정

4 검토

참고: 버킷 생성 후에는 특정 사용자에게 액세스 권한을 부여할 수 있습니다.

퍼블릭 액세스 차단(버킷 설정)

퍼블릭 액세스는 ACL(액세스 제어 목록), 버킷 정책 또는 둘 다를 통해 버킷 및 객체에 부여됩니다. 모든 S3 버킷 및 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 [모든 퍼블릭 액세스 차단](#)을 활성화합니다. 이 설정은 이 버킷에만 적용됩니다. AWS에서는 [모든 퍼블릭 액세스 차단](#)을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없어도 애플리케이션이 올바르게 작동하는지 확인합니다. 버킷 또는 내부 객체에 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 특정 스토리지 사용 사례에 맞게 아래 개별 설정을 사용자 지정할 수 있습니다. [세부 정보](#)

1

모든 퍼블릭 액세스 차단

이 설정을 활성화하면 아래 4개의 설정을 모두 활성화한 것과 같습니다. 다음 설정 각각은 서로 독립적입니다.

- ACL(액세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

S3은 새로 추가된 버킷 또는 객체에 적용되는 퍼블릭 액세스 권한을 차단하며, 기존 버킷 및 객체에 대한 새 퍼블릭 액세스 ACL 생성을 금지합니다. 이 설정은 ACL을 사용하여 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 권한을 변경하지 않습니다.
- 임의의 ACL(액세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 모든 ACL을 무시합니다.
- 퍼블릭 버킷 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 새 버킷 정책을 차단합니다. 이 설정은 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 정책을 변경하지 않습니다.
- 임의의 퍼블릭 버킷 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 및 교차 계정 액세스 차단

2

이전

다음

모든 퍼블릭 액세스 차단 **체크 해제**

1. S3 bucket 생성

버킷 만들기

이름 및 리전

옵션 구성

권한 설정

4 검토

CloudWatch 요청 추정치

비활성

객체 잠금

비활성

권한

편집

모든 퍼블릭 액세스 차단

비활성화

ACL(엑세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

비활성화

임의의 ACL(엑세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

비활성화

퍼블릭 버킷 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

비활성화

임의의 퍼블릭 버킷 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 및 교차 계정 액세스 차단

비활성화

시스템 권한

비활성

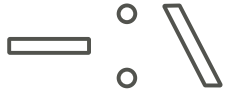
1

이전

버킷 만들기

SHOUT OUR PASSION TOGETHER

SOPT



SHOUT OUR PASSION TOGETHER

Multer with S3 실습

2. S3 bucket 정책 설정

버킷 검색

모든 액세스 유형

+ 버킷 만들기 퍼블릭 액세스 설정 편집 비우기 삭제

1 버킷 1 리전

버킷 이름	액세스	리전	생성 날짜
with-sopt-server	객체 퍼블릭화 가능	아시아 태평양(서울)	11월 9, 2019 8:34:58 오후 GMT+0900

Amazon S3 > with-sopt-server

개요 권한 관리

2-1

2-2

퍼블릭 액세스 차단 퍼블릭 액세스 제어 목록 버킷 정책 CORS 구성

※

버킷 정책 편집기 ARN: arn:aws:s3:::with-sopt-server

아래 텍스트 영역에 새 정책을 추가하거나 기존 정책을 편집하려면 입력합니다.

삭제 취소 저장

1

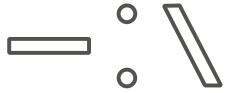
2-3

정책 생성기

1. 생성된 버킷 클릭

2. 권한 -> 버킷 정책 -> 정책생성기

※ 이때 ARN을 복사한다.



2. S3 bucket 정책 설정

Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an [IAM Policy](#), an [S3 Bucket Policy](#), an [SNS Topic Policy](#), a [VPC Endpoint Policy](#), and an [SQS Queue Policy](#).

1 Select Type of Policy

Step 2: Add Statement(s)

A statement is the formal description of a single permission. See [a description of elements](#) that you can use in statements.

2 Effect ☒ Allow ☐ Deny

3 Principal
Use a comma to separate multiple values.

4 AWS Service ☐ All Services (**)
Use multiple statements to add permissions for more than one service.

5 Actions ☐ All Actions (**)

6 Amazon Resource Name (ARN)
ARN should follow the following format: arn:aws:s3:::<bucket_name>/<key_name>.
Use a comma to separate multiple values.

Add Conditions (Optional)

6 Add Statement

Step 3: Generate Policy

A *policy* is a document (written in the [Access Policy Language](#)) that acts as a container for one or more statements.

Add one or more statements above to generate a policy.

1. Type of Policy에서 **S3 Bucket Policy** 선택
2. Principal에 *****
3. AWS Service에 **Amazon S3**
4. Action에 **getObject** 선택
5. ARN에 **'복사한 값/*'**
6. Add Statement

2. S3 bucket 정책 설정

You added the following statements. Click the button below to Generate a policy.

Principal(s)	Effect	Action	Resource	Conditions
1. *	Allow	• s3:GetObject	arn:aws:s3:::with-sopt-server/*	None

Step 3: Generate Policy

A *policy* is a document (written in the [Access Policy Language](#)) that acts as a container for one or more statements.

2 **Generate Policy** [Start Over](#)

1. ARN에 복사한 값과 /*가 제대로 들어갔는지 확인한다.
2. Generate Policy
3. Get Policy를 클릭하고 해당 내용을 복사한다.

2. S3 bucket 정책 설정

Amazon S3 > with-sopt-server

개요 속성 권한 관리

퍼블릭 액세스 차단 액세스 제어 목록 버킷 정책 CORS 구성

버킷 정책 편집기 ARN: arn:aws:s3:::with-sopt-server
아래 텍스트 영역에 새 정책을 추가하거나 기존 정책을 편집하려면 입력합니다.

삭제 취소 저장

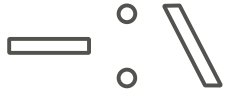
1

설명서 정책 생성기

1. 복사한 정책(텍스트)을 버킷 정책 편집기에 추가한다.
2. 저장
3. 아래와 같이 경고가 뜨면 성공입니다.

아래 경고로 종색에 새 정책을 추가하거나 기존 정책을 편집할때도 표시됩니다
버킷 외부 공유가 | Visit: <https://www.amazonaws.cn/>

이 버킷에 퍼블릭 액세스를 생성했습니다. 이 버킷에 대하여 어떤 퍼블릭 액세스도 되거나 최대 속도와도 같이 올랐습니다.
이 버킷에 퍼블릭 액세스도 되거나 성공



3. IAM 설정하기

IAM 이란

AWS Identity and Access Management(IAM)는

AWS 리소스에 대한 액세스를 안전하게 제어할 수 있는 웹 서비스입니다.

IAM을 사용하여 리소스를 사용하도록 인증(로그인) 및 권한 부여(권한 있음)된 대상을 제어합니다.

1

사용자 추가

사용자 삭제

Q 사용자 이름 또는 액세스 키로 사용자 찾기

☐

사용자 이름 ▼

- 1. AWS 서비스
- > IAM
- > 사용자
- > 사용자 추가

3. IAM 설정하기

사용자 세부 정보 설정

동일한 액세스 유형 및 권한을 사용하여 한 번에 여러 사용자를 추가할 수 있습니다. [자세히 알아보기](#)

1 사용자 이름*

25withsopt

+ 다른 사용자 추가

AWS 액세스 유형 선택

해당 사용자가 AWS에 액세스하는 **2** 을 선택합니다. 마지막 단계에서는 액세스 키와 자동 생성된 비밀번호가 제공됩니다. [자세히 알아보기](#)

액세스 유형*



프로그래밍 방식 액세스

AWS API, CLI, SDK 및 기타 개발 도구에 대해 액세스 키 ID 및 비밀 액세스 키 을(를) 활성화합니다.



AWS Management Console 액세스

사용자가 AWS Management Console에 로그인할 수 있도록 허용하는 비밀번호 을(를) 활성화합니다.

1. 사용자 이름 지정
2. 프로그래밍 방식 액세스 체크

3. IAM 설정하기

▼ 권한 설정

그룹에 사용자 추가

기존 사용자에서 권한 복사

1 기존 정책 직접 연결

정책 생성

정책 필터 ▼ AmazonS3Full 1 결과 표시

2 정책 이름 ▼	유형	사용 용도
<input checked="" type="checkbox"/> ▶ AmazonS3FullAccess	AWS 관리형	없음

1. 기본 정책 연결
2. AmazonFullAccess 체크
3. 다음: 태그 -> 다음: 검토 -> 사용자 만들기

3. IAM 설정하기



성공

아래에 표시된 사용자를 생성했습니다. 사용자 보안 자격 증명을 보고 다운로드할 수 있습니다. AWS Management Console 로그인에 위한 사용자 지침을 이메일로 보낼 수도 있습니다. 지금이 이 자격 증명을 다운로드할 수 있는 마지막 기회입니다. 하지만 언제든지 새 자격 증명을 생성할 수 있습니다.

AWS Management Console 액세스 권한이 있는 사용자가 [redacted]에 로그인할 수 있습니다.

다운로드 .csv 다운로드

	사용자	액세스 키 ID	비밀 액세스 키
▶	25withsopt	[redacted]	[redacted] 표시

액세스 키와 비밀키를 확인할 수 있는 페이지가 나옵니다.
이 페이지를 나가면 해당 Key를 다시 확인 할 수 없기 때문에 **반드시 메모**합니다.
또한 이 Key를 **git에 올리거나 유출**될 경우 **과금**의 원인이 될 수 있습니다.

4. aws-sdk, multer-s3 설치

```
npm install aws-sdk multer-s3
```

5. awsconfig.json 생성

config 폴더 아래에 awsconfig.json 파일을 생성하고 아래와 같이 입력한다.

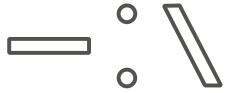
이때 gitignore에 config를 꼭 추가하고 commi시 해당 파일이 업로드 되는지 주의를 기울여야 한다.

```
{  
  "accessKeyId": "엑세스키",  
  "secretAccessKey": "시크릿키",  
  "region": "ap-northeast-2"  
}
```

accessKeyId와 SecretAccessKey에 IAM에서 생성한 2가지 키를 입력해주면 된다.

➤ Region의 경우 S3에 접속했을 때 URL을 통해서 확인할 수 있다.

<https://s3.console.aws.amazon.com/s3/buckets/with-sopt-server/?region=ap-northeast-2&tab=management>



6. multer.js 생성

config폴더에 multer.js파일을 생성하고 아래의 코드를 입력한다. 또한 버킷이름을 변경해준다.

```
const multer = require('multer');
const multerS3 = require('multer-s3');
const aws = require('aws-sdk');
aws.config.loadFromPath(__dirname + '/awsconfig.json');

const s3 = new aws.S3();

const upload = multer({
  storage: multerS3({
    s3: s3,
    bucket: '버킷이름',
    acl: 'public-read',
    key: function(req, file, cb) {
      cb(null, Date.now() + '.' + file.originalname.split('.').pop());
    }
  })
});
module.exports = upload;
```

7. MulterTest.js 코드 수정

변경 전

```
const multer = require('multer');  
const upload = multer({  
  dest: 'uploads/'  
});
```

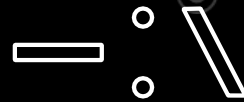
변경 후

```
const upload = require('../config/multer');
```

좌측의 코드를 우측으로 바꾸기만 하면 local 서버가 아닌 S3에 올라가는 것을 확인할 수 있다.

이를 POSTMAN으로 실습해보고 조원들과 결과를 공유해보자.

SHOUT OUR PASSION TOGETHER



04

JWT

SHOUT OUR PASSION TOGETHER
SOPT



JWT란 JSON Web Token의 약자로 [클레임 토큰 기반 인증 방식](#) 입니다. 클라이언트의 세션 상태를 저장하는 것이 아니라 필요한 정보를 토큰 body에 저장해서 클라이언트가 가지고 이를 증명서 처럼 사용

JWT의 구성

{Header}.{Payload}.{Verify Signature}
3가지 정보를 '.'로 연결하여 사용한다.

Header	JWT 토큰의 유형이나 사용된 해시 알고리즘의 정보가 들어간다.
Payload	클라이언트에 대한 정보가 담겨있다. 또한 여기에는 iss,sub,aud,exp,nbf,iat,jti 와 같은 기본 정보가 들어간다.
Signature	header에서 지정한 알고리즘과 secret key로 Header와 Payload를 담는다.

<https://jwt.io/> 해당 사이트를 통해서 JWT Token을 확인, 검증해 볼 수 있다.

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1IjoiMTIzNCIsImh0dCI6MTU3Mzc0DEzNCw
iZXhwIjoxNTczNzUxNzM0LCJpc3MiOiJnZW5pZSJ9.AKCK9dq8W9G0He8VjSpELPCyBRuMIAF1VZPLk_
4T2Jc
```

토큰 예시

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1IjoiMTIzNCIsImh0dCI6MTU3Mzc0DEzNCw
iZXhwIjoxNTczNzUxNzM0LCJpc3MiOiJnZW5pZSJ9.AKCK9dq8W9G0He8VjSpELPCyBRuMIAF1VZPLk_
4T2Jc
```

복사해서 사용

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "name": "1234",
  "iat": 1573748134,
  "exp": 1573751734,
  "iss": "genie"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Payload는 공개 데이터

JWT에 정보는 누구나 <https://jwt.io/> 페이지에 접속해서 정보를 확인할 수 있습니다.

따라서 비밀번호와 같은 보안이 필요한 정보는 payload에 저장하면 안됩니다.

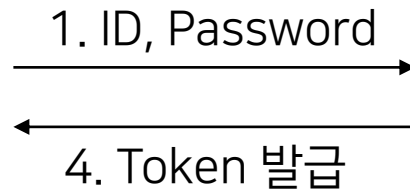
JWT의 Secret Key

JWT에서는 정보는 공개가 되어있지만 해시 값을 통해서 정보가 유효한지 확인을 하게 됩니다. 따라서 시크릿 키가 유출이 된다면 JWT에서 보안상에 큰 위협이 됩니다.

JWT 과정

로그인(토큰 발행)

1. 클라이언트가 유저에 대한 정보(ex. ID, Password)에 대한 정보를 서버에게 보낸다.
2. 서버는 DB를 이용해서 정보의 유효성을 확인한다.
3. User 정보 중 일부를 JWT body에 넣고 토큰을 발행
4. 클라이언트에게 응답한다.



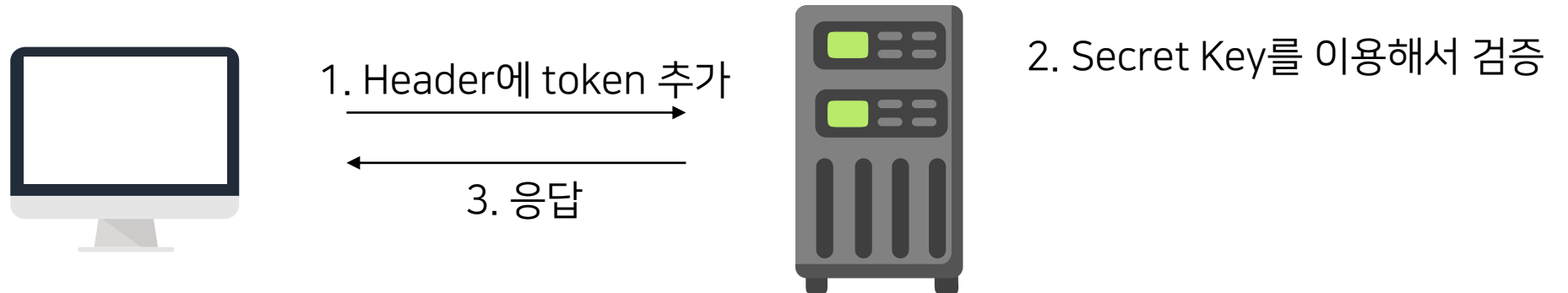
2. DB를 이용해서 비밀번호 체크
3. Secret Key를 이용해서 토큰 발행

JWT 과정

토큰 검증

1.HTTP header에 토큰값을 넣어서 보낸다.

2.서버는 토큰값을 받아서 JWT 정보와 서버가 가지고 있는 secret key를 이용해서 서명을 만든다.
이때 JWT의 서명과 일치하다면 유효하고 일치하지 않는다면 유효하지 않는 요청으로 판단한다.



1. Jsonwebtoken 및 rand-token 모듈 설치

```
npm install jsonwebtoken rand-token
```

2. Config 폴더에 secretKey.js 생성

```
module.exports = {  
  secretOrPrivateKey: "jwtSecretKey!"  
}
```

secretOfPrivateKey에는 JWT에서 사용할 암호를 지정

3. Module에 jwt.js 생성

```
const randToken = require('rand-token');
const jwt = require('jsonwebtoken');
const {secretOrPrivateKey} = require('../config/secretKey');
const options = {
  algorithm: "HS256",
  expiresIn: "1h",
  issuer: "genie"
};

module.exports = {
  sign: (user) => {
  },
  verify: (token) => {
    let decoded;
    return decoded;
  },
  refresh: (user) => {
  }
};
```

Option에는 3가지 옵션을 지정했다.

1. Algorithm
서명을 만들기 위해 사용된 알고리즘을 지정
2. expiresIn
토큰의 유효기간을 지정
1h -> 한시간
1m -> 1분
60 * 60 * 24 -> 1일
3. Issuer
토큰을 발행한 사람의 이름을 지정

4. jwt.js에서 sign 함수 구현

```
sign: (user) => {  
  const payload = {  
    idx: user.idx,  
    grade: user.grade,  
    name: user.name  
  };  
  const result = {  
    token: jwt.sign(payload, secretOrPrivateKey, options),  
    refreshToken: randToken.uid(256)  
  };  
  return result;  
},
```

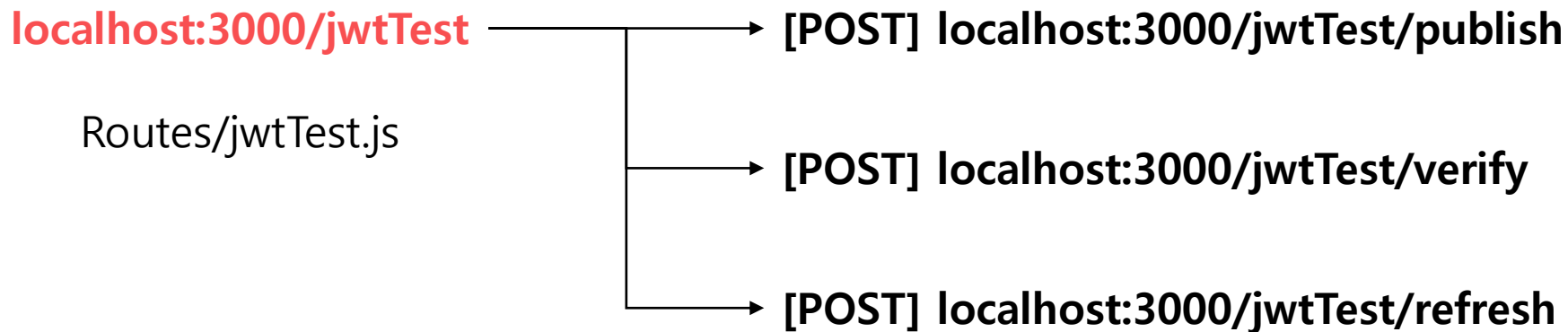
5. jwt.js에서 verify 함수 구현

```
verify: (token) => {  
  let decoded;  
  try {  
    decoded = jwt.verify(token, secretOrPrivateKey);  
  } catch (err) {  
    if (err.message === 'jwt expired') {  
      console.log('expired token');  
      return -3;  
    } else if (err.message === 'invalid token') {  
      console.log('invalid token');  
      return -2;  
    } else {  
      console.log("invalid token");  
      return -2;  
    }  
  }  
  return decoded;  
},
```

6. jwt.js에서 refresh 함수 구현

```
refresh: (user) => {  
  const payload = {  
    idx: user.idx,  
    grade: user.grade,  
    name: user.name  
  };  
  return jwt.sign(payload, secretOrPrivateKey, options);  
}
```

7. jwtTest.js 라우팅 구현



8. jwtTest.js에서 publish 구현

```
router.post('/publish', (req, res) => {  
  const {idx, grade, name} = req.body;  
  if(!idx || !grade || !name){  
    res.send('wrong parameter');  
    return;  
  }  
  const result = jwt.sign({idx, grade, name});  
  res.json(result);  
})
```

9. jwtTest.js에서 verify 구현

```
router.post('/verify', (req, res) => {  
  const {token} = req.headers;  
  const result = jwt.verify(token);  
  if(result == -1) {  
    return res.status(statusCode.UNAUTHORIZED)  
      .send(util.successFalse(resMessage.EXPIRED_TOKEN));  
  }  
  if(result == -2) {  
    return res.status(statusCode.UNAUTHORIZED)  
      .send(util.successFalse(resMessage.INVALID_TOKEN));  
  }  
  res.json(result);  
});
```


10. jwtTest.js에서 Refresh 구현

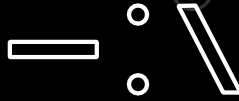
```
router.post('/refresh', (req, res) => {  
  const refreshToken = req.headers.refreshtoken;  
  const selectUser = {  
    idx: 1,  
    grade: 1,  
    id: 'genie',  
    name: 'genie'  
  };  
  
  const newAccessToken = jwt.refresh(selectUser);  
  res.status(statusCode.OK).send(util.successTrue(resMessage.REFRESH_TOKEN, newAccessToken));  
});
```

5주차 전체 프로젝트 코드

<https://github.com/WITH-SOPT-SERVER/SOPT-SERVER-SEMINAR/tree/develop/seminar5>

5주차 보충 세미나에서는
JWT refresh Token과 미들웨어 설정에 대해서 다룰 예정입니다.
- 25기 서버 파트장 윤희성 -

SHOUT OUR PASSION TOGETHER



05

과제 안내

4주차 과제에 다음과 같은 기능을 추가합니다.

1. Article Table에 Image Field를 추가합니다.
Article을 작성할 때 이미지 파일도 업로드 할 수 있습니다.

[심화] 만약 하나의 게시글 에서 복수개의 이미지를 올릴 수 있다면 어떻게 할지 고민해보세요.

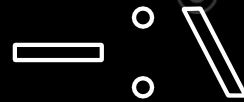
2. 로그인 회원가입을 추가합니다.
이때 로그인에서 성공한 경우 token값을 반환해줍니다.

[심화] Token의 payloa에는 어떤 값을 넣는 것이 좋은 지 고민해조세요.

3. 모든 객체(블로그, 게시글, 댓글)의 작성, 수정, 삭제는 token을 이용해서 검증단계를 거칩니다.

[심화] 보충세미나에서 다룬 미들웨어를 어떻게 적용할 지 고민해보세요.

SHOUT OUR PASSION TOGETHER



PASSION

Thank You :)

EXPERIENCE

GROWTH

CHALLENGE

SHOUT OUR PASSION TOGETHER
SOPT