Express

#01. Express 개요

> 01-setup.js

Node.js를 위한 빠르고 개방적인 간결한 웹 프레임워크

```
yarn add express
```

1) 프레임워크

프로그램을 만드는데 기본적으로 제공되는 기본 골격.

전체적인 틀이 이미 정해져 있기 때문에 개발자는 자신이 목표로 하는 세부 기능에만 집중할 수 있다.

코드 품질이 일정하게 유지될 수 있다.

2) Express와 함께 사용 할 수 있는 서드파티 패키지들

express-useragent

접근한 클라이언트의 정보(운영체제정보, 브라우저 정보, IP주소 등)를 취득할 수 있는 기능을 제공

```
yarn add express-useragent
```

serve-static

특정 폴더를 통째로 웹에 노출시키는 기능 (폴더 구조가 그대로 URL노출됨)

img, css, js, 정적 html파일등을 호스팅하고자 할 때 사용함.

```
yarn add serve-static
```

serve-favicon

favicon 파일을 브라우저에 전달한다.

```
yarn add serve-favicon
```

3) Express 기반 백엔드 구현 기본 코드 구조

```
/*-------------------------------------------------------------
 | 1) 모듈참조
 -------------------------------------------------------------*/


/*-------------------------------------------------------------
 | 2) Express 객체 생성
 -------------------------------------------------------------*/


/*-------------------------------------------------------------
 | 3) 클라이언트의 접속시 초기화
 -------------------------------------------------------------*/


/*-------------------------------------------------------------
 | 4) Express 객체의 추가 설정
 -------------------------------------------------------------*/


/*-------------------------------------------------------------
 | 5) 각 URL별 백엔드 기능 정의
 -------------------------------------------------------------*/


/*-------------------------------------------------------------
 | 6) 설정한 내용을 기반으로 서버 구동 시작
 -------------------------------------------------------------*/
```

#02. 웹 파라미터

> 02-GetParams.js

프론트엔드가 백엔드에게 전달하는 변수.

백엔드는 프론트엔드로부터 전달받은 변수에 따라 선택적인 결과를 생성하여 돌려줄 수 있다.

ex) 프론트엔드 -----(아이디,비밀번호)-------> 로그인

GET,POST,PUT,DELETE 방식의 변수 전달 방식이 있다.

1) GET 파라미터

URL 뒤에 `?`를 기준으로 `이름=값&이름=값` 형식으로 변수를 포함시키는 방법.

`?이름=값&이름=값` 형식의 문자열을 QueryString 이라고 하며 Client측 Javascript에서는 `window.location.search`로 문자열 전체를 취득하여 이를 활용할 수 있다.

주로 링크를 클릭한 경우 어떤 링크를 선택했는지를 감지하기 위해 사용한다.

HTML의 `<form>` 태그에서 `method`속성을 명시하지 않거나 `GET`으로 부여한 경우 `<input>` 태그의 name속성이 변수명이 되고, 사용자 입력값이 변수의 값이 되어 action 속성에 명시한 URL로 전송된다.

모든 변수가 URL에 노출되기 때문에 상대적으로 보안에 좋지 않다.

2) Path 파라미터

URL안에 변수값을 폴더 이름처럼 숨겨놓은 처리

Path 파라미터는 변수의 전달 형태만 다를 뿐 URL을 통해 전달한다는 점에서 GET 파라미터의 한 종류로 구분한다.

```
http://<hostname>:<port>/페이지이름/변수1/변수2
```

#03. 웹 파라미터 (2)

> 03-Post,Put,Delete.js

1) POST 파라미터

HTML의 `<form>` 태그에서 `method`속성을 `POST`로 부여한 경우를 의미.

변수가 URL에 노출되지 않기 때문에 상대적으로 보안에 유리하지만,
결코 전송되는 변수를 볼 수 없다는 것을 의미하지는 않는다.

Node.js 스스로는 POST 파라미터를 처리하지 못하므로 다음의 패키지를 설치해야 한다.

```
yarn add body-parser
```

2) PUT, DELETE 파라미터

HTTP 1.1에서는 지원하지 않는 속성 (HTTP 2.0부터 지원)

HTML의 `<form>` 태그는 이 방식을 지원하지 않는다.

기본적으로 POST 방식과 동일하면서 전송방식의 이름만 변경한 형태.

Node.js 스스로는 PUT, DELETE 파라미터를 처리하지 못하므로 다음의 패키지를 설치해야 한다.

```
yarn add method-override
```

3) RestfulAPI

하나의 URL이 어떤 개체(ex-상품,회원 등)를 의미하고 GET, POST, PUT, DELETE 전송방식에 따라 조회,입력,수정,삭제 기능을 구분하는 구현 형태

대부분 OpenAPI는 Restful API 방식을 따른다.

전송방식	수행할 동작	의미	SQL
POST	입력	Create	INSERT
GET	조회	Read	SELECT
PUT	수정	Update	UPDATE
DELETE	삭제	Delete	DELETE

위의 네가지 방식을 CRUD라 부른다.

즉, Restful API는 CRUD 방식을 따르는 표준.

전송방식에 따른 URL 예시

장바구니를 다루는 Restful API라고 가정할 경우

전송방식	URL	동작	파라미터
POST	http://localhost:3000/shopping/cart	장바구니에 담기	상품정보는 POST 방식으로 전달한다. 저장된 항목을 의미하는 고유한 일련번호값이 생성된다.
GET	http://localhost:3000/shopping/cart/:일련번호	해당 일련번호에 대한 장바구니 내역 조회	조회대상을 Path 파라미터로 전달한다.
PUT	http://localhost:3000/shopping/cart/:일련번호	해당 일련번호에 대한 장바구니 내역 수정(주문수량 등)	수정할 대상을 의미하는 일련번호는 Path 파라미터로 전달하고 수정할 내용은 PUT 파라미터로 전달한다.
DELETE	http://localhost:3000/shopping/cart/:일련번호	해당 일련번호에 대한 장바구니 항목 삭제	삭제대상을 Path 파라미터로 전달한다.

#04. 쿠키

> 04-Cookie.js

변수를 클라이언트(=웹브라우저)에 텍스트 파일 형식으로 저장해 놓은 것.

저장 주체가 백엔드일 경우 백엔드가 갖고 있는 변수 값을 프론트엔드에 보관시킨다는 의미.

백엔드에 접속하는 개인별로 맞춤형 정보를 저장할 수 있기 때문에 개인화 기능 구현에 활용됨.

텍스트 파일의 특성상 보안에 민감한 정보는 기록해서는 안된다.

백엔드와 프론트엔드가 변수값을 공유할 수 있는 가장 원시적인 방법.

저장위치가 브라우저이기 때문에 프론트엔드에서도 쿠키를 저장하거나 읽어올 수 있는 API가 존재함. (같은 값 공유 가능)

하지만 보안에 최악.

1) 쿠키의 특성

일반 변수는 프론트엔드가 페이지를 이동하면서 사라진다.

기본적으로 브라우저를 닫기 전까지는 사이트 내의 모든 페이지가 공유하는 전역변수의 역할을 한다.

설정에 따라 일정 유효기간 동안은 브라우저를 재시작해도 변수값을 유지할 수 있다.

#2) 패키지 설치

```
yarn add cookie-parser
```

3) 쿠키 저장에 필요한 조건

이름	설명
maxAge	유효시간 (ms단위). 설정하지 않을 경우 브라우저를 닫으면 삭제됨.
domain	유효 도메인 지정. 지정하지 않을 경우 현재 도메인 ex) 도메인이 itpaper.co.kr인 경우 ".itpaper.co.kr"로 설정하면 해당 도메인 내의 모든 사이트가 쿠키를 공유할 수 있다.
httpOnly	boolean (true/false). false인 경우 https에서도 식별가능.
path	유효경로 (기본값 "/") -> 특별한 일이 없다면 지정 안함.
signed	암호화 여부 app.use(cookieParser("암호화키")); 형식으로 암호화 키 지정 필요함.

#05. 세션

> 05-Session.js

쿠키와 마찬가지로 사이트 내의 모든 페이지가 공유하는 전역변수.

단, 유효시간은 설정할 수 없기 때문에 브라우저가 닫히거나 마지막 접속 이후 5분간 재접속이 없다면 자동 폐기된다. (시간은 설정 가능함)

특정 클라이언트에게 종속된 개인화 정보를 백엔드가 직접 저장/관리하는 형태.

많은 데이터를 보관할 수는 없다.

쿠키보다 보안에 유리하므로 로그인 정보 관리 등에 사용된다.

```
yarn add express-session
```

#06. 메일 발송

> 06-SendMail.js

웹 서버는 자체적으로 메일 발송을 할 수 없기 때문에 발송을 원하는 경우 SMTP서버와 연동해야 한다.

리눅스 서버등을 설치한 후 직접 SMTP서버를 운영하는 경우가 아니라면 타 플랫폼에 연동하는 형식으로 메일 발송 기능을 활용할 수 있다.

ex) 구글, 네이버 등

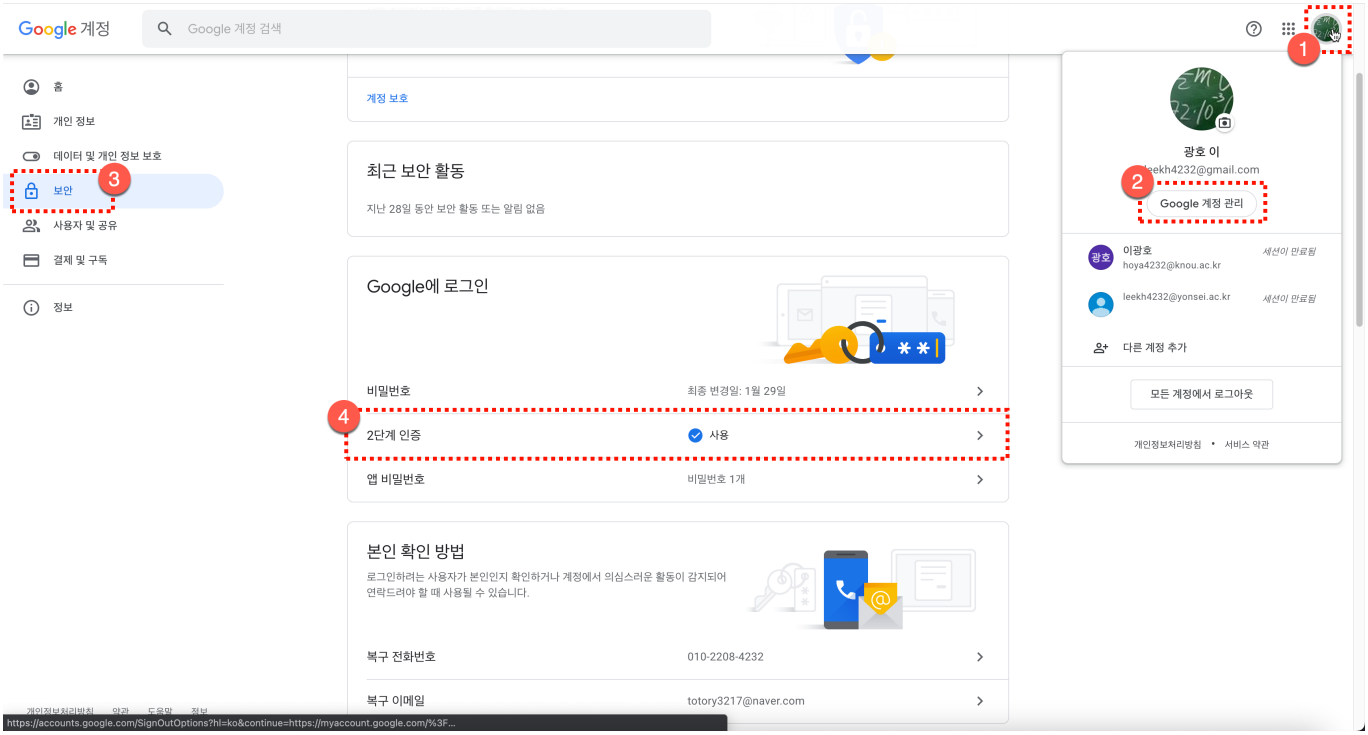
> 이 단원에서는 구글 SMTP와 연동하기 위한 구글 보안 설정 과정이 필요합니다.

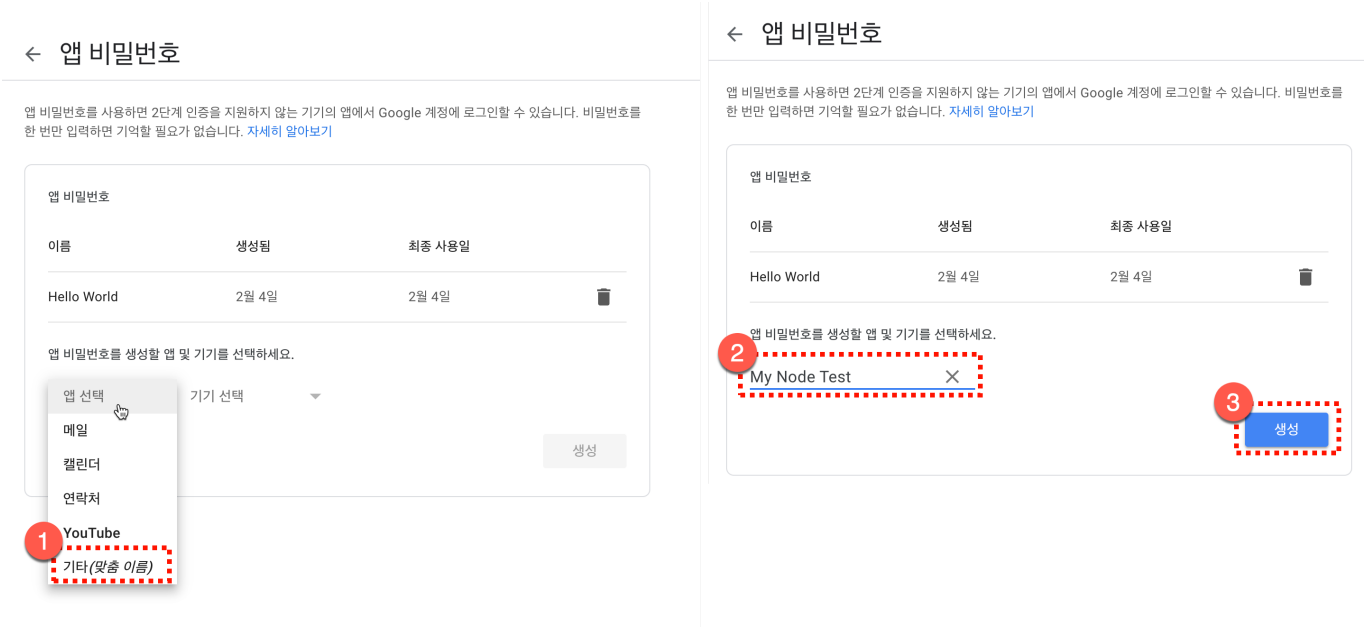
1) 패키지 설치하기

```
yarn add nodemailer
```

소스코드는 환경설정 과정에 불과하므로 응용의 여지는 적다.

2) 구글 앱 비밀번호 발급받기





#07. FileUpload.js

> 07-FileUpload.js

1) 파일 전송 (Frontend)

HTML의 `form` 상에서 `<input type='file'>` 요소를 사용하여 `form`에 명시된 `action` 페이지(=백엔드)로 전송

Ajax는 파일 업로드가 불가능하므로 Javascript의 file api를 사용하여 별도로 업로드 기능을 구현해야 함. (Frontend 영역)

파일 전송시에는 반드시 `<form>` 태그에 `enctype="multipart/form-data"` 속성이 명시되어야 함.

그런 이유로 파일 업로드를 백엔드에서는 multipart 전송이라고 표현하기도 함.

GET 방식 전송으로는 처리가 불가.

```
<form method='post' action='/upload/simple' enctype='multipart/form-data'>
    <input type='file' name='myphoto' />
    <input type='submit'>
</form>
```

2) 파일 수신 (Backend)

node의 `multer` 패키지 사용

```
yarn add multer
```

업로드 객체에 대한 환경 설정

```
var multipart = multer({
    storage: multer.diskStorage({
        destination: function(req, file, callback) {
            callback(null, 업로드_될_파일이_저장될_폴더경로);
        },
        filename: function(req, file, callback) {
            callback(null, 저장될_파일이름);
        }
    }),
    limits: { // 업로드 제약
        files: 최대업로드가능_파일수(int),
        fileSize: 최대업로드가능_크기(byte단위)
    }
});
```

백엔드 페이지에서 업로드 처리

```
// 업로드를 위한 함수 클로저 리턴받기 (멀티업로드인 경우 single 대신 array 함수 사용)
var upload = multipart.single('input태그의 name속성값');

upload(req, res, function(err) {
    if (err) { ... 에러가 발생한 경우 ... }

    ... 업로드 성공시 처리 ...
});
```

아래의 URL에서 자세한 레퍼런스 참조 가능함.

https://github.com/expressjs/multer/blob/master/doc/README-ko.md

---

Multer

Multer는 파일 업로드를 위해 사용되는 `multipart/form-data` 를 다루기 위한 node.js 의 미들웨어 입니다. 효율성을 최대화 하기 위해 busboy 를 기반으로 하고 있습니다.

주: Multer는 multipart (`multipart/form-data`)가 아닌 폼에서는 동작하지 않습니다.

번역

이 문서는 아래의 언어로도 제공됩니다:

- English (영어)
- Español (스페인어)
- 简体中文 (중국어)
- Русский язык (러시아)
- Português (포르투갈어 BR)

설치

```
$ yarn add multer
```

사용법

Multer는 `body` 객체와 한 개의 `file` 혹은 여러개의 `files` 객체를 `request` 객체에 추가합니다. `body` 객체는 폼 텍스트 필드의 값을 포함하고, 한 개 혹은 여러개의 파일 객체는 폼을 통해 업로드된 파일들을 포함하고 있습니다.

기본 사용 예제:

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

const app = express()

app.post('/profile', upload.single('avatar'), function (req, res, next) {
  // req.file 은 `avatar` 라는 필드의 파일 정보입니다.
  // 텍스트 필드가 있는 경우, req.body가 이를 포함할 것입니다.
})

app.post('/photos/upload', upload.array('photos', 12), function (req, res, next) {
  // req.files 는 `photos` 라는 파일정보를 배열로 가지고 있습니다.
  // 텍스트 필드가 있는 경우, req.body가 이를 포함할 것입니다.
})

const cpUpload = upload.fields([{ name: 'avatar', maxCount: 1 }, { name:
'gallery', maxCount: 8 }])
app.post('/cool-profile', cpUpload, function (req, res, next) {
  // req.files는 (String -> Array) 형태의 객체 입니다.
  // 필드명은 객체의 key에, 파일 정보는 배열로 value에 저장됩니다.
  //
  // e.g.
  //  req.files['avatar'][0] -> File
  //  req.files['gallery'] -> Array
  //
  // 텍스트 필드가 있는 경우, req.body가 이를 포함할 것입니다.
})
```

텍스트 전용 multipart 폼을 처리해야 하는 경우, 어떠한 multer 메소드 (`.single()`, `.array()`, `fields()`) 도 사용할 수 있습니다. 아래는 `.array()` 를 사용한 예제 입니다 :

```
const express = require('express')
const app = express()
const multer  = require('multer')
```

```
const upload = multer()

app.post('/profile', upload.array(), function (req, res, next) {
  // req.body는 텍스트 필드를 포함합니다.
})
```

API

파일 정보

각각의 파일은 아래의 정보를 포함하고 있습니다:

Key	Description	Note
`fieldname`	폼에 정의된 필드 명	
`originalname`	사용자가 업로드한 파일 명	
`encoding`	파일의 엔코딩 타입	
`mimetype`	파일의 Mime 타입	
`size`	파일의 바이트(byte) 사이즈	
`destination`	파일이 저장된 폴더	`DiskStorage`
`filename`	`destination` 에 저장된 파일 명	`DiskStorage`
`path`	업로드된 파일의 전체 경로	`DiskStorage`
`buffer`	전체 파일의 `Buffer`	`MemoryStorage`

`multer(opts)`

Multer는 옵션 객체를 허용합니다. 그 중 가장 기본 옵션인 `dest` 요소는 Multer에게 파일을 어디로 업로드 할지를 알려줍니다. 만일 옵션 객체를 생략했다면, 파일은 디스크가 아니라 메모리에 저장될 것 입니다.

기본적으로 Multer는 이름이 중복되는 것을 방지하기 위해서 파일의 이름을 재작성 합니다. 필요에 따라 해당 함수는 커스터마이징이 가능합니다.

Multer로 전달 가능한 옵션들은 다음과 같습니다.

Key	Description
`dest` or `storage`	파일이 저장될 위치
`fileFilter`	어떤 파일을 허용할지 제어하는 함수
`limits`	업로드 된 데이터의 한도
`preservePath`	파일의 base name 대신 보존할 파일의 전체 경로

보통의 웹 앱에서는 `dest` 옵션 정도만 필요할지도 모릅니다. 설정 방법은 아래의 예제에 나와있습니다.

```

```

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

만일 업로드를 더 제어하고 싶다면, `dest` 옵션 대신 `storage` 옵션을 사용할 수 있습니다. Multer는 스토리지 엔진인 `DiskStorage` 와 `MemoryStorage` 를 탑재하고 있습니다. 써드파티로부터 더 많은 엔진들을 사용할 수 있습니다.

`.single(fieldname)`

`fieldname` 인자에 명시된 이름의 단수 파일을 전달 받습니다. 이 파일은 `req.file` 에 저장될 것 입니다.

`.array(fieldname[, maxCount])`

`fieldname` 인자에 명시된 이름의 파일 전부를 배열 형태로 전달 받습니다. 선택적으로 `maxCount` 에 명시된 값 이상의 파일이 업로드 될 경우 에러를 출력할 수 있습니다. 전달 된 배열 형태의 파일은 `req.files` 에 저장될 것입니다.

`.fields(fields)`

`fields` 인자에 명시된 여러 파일을 전달 받습니다. 파일 객체는 배열 형태로 `req.files` 에 저장될 것입니다.

`fields` 는 `name` 과 `maxCount` (선택사항) 을 포함하는 객체의 배열이어야 합니다.
예제:

```
[
  { name: 'avatar', maxCount: 1 },
  { name: 'gallery', maxCount: 8 }
]
```

`.none()`

오직 텍스트 필드만 허용합니다. 만일 파일이 업로드 되었을 경우, "LIMIT_UNEXPECTED_FILE" 와 같은 에러 코드가 발생할 것입니다. 이는 `upload.fields([])` 와 같은 동작을 합니다.

`.any()`

전달된 모든 파일을 허용합니다. 파일 배열은 `req.files` 에 저장될 것입니다.

주의: 항상 사용자가 업로드한 파일을 다룬다는 점을 명심하세요. 악의적인 사용자가 여러분이 예측하지 못한 곳으로 파일을 업로드 할 수 있으므로 절대 multer를 글로벌 미들웨어로 사용하지 마세요.

`storage`

`DiskStorage`

디스크 스토리지 엔진은 파일을 디스크에 저장하기 위한 모든 제어 기능을 제공합니다.

```
```

```
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, '/tmp/my-uploads')
  },
  filename: function (req, file, cb) {
    cb(null, file.fieldname + '-' + Date.now())
  }
})

const upload = multer({ storage: storage })
```

`destination` 과 `filename` 의 두가지 옵션이 가능합니다. 두 옵션 모두 파일을 어디에 저장할 지를 정하는 함수입니다.

`destination` 옵션은 어느 폴더안에 업로드 한 파일을 저장할 지를 결정합니다. 이는 `string` 형태로 주어질 수 있습니다 (예. `'/tmp/uploads'`). 만일 `destination` 옵션이 주어지지 않으면, 운영체제 시스템에서 임시 파일을 저장하는 기본 디렉토리를 사용합니다.

주: `destination` 을 함수로 사용할 경우, 디렉토리를 생성해야 할 책임이 있습니다. 문자열이 전달될 때, multer는 해당 디렉토리가 생성되었는지 확인합니다.

`filename` 은 폴더안에 저장되는 파일 명을 결정하는데 사용됩니다.
만일 `filename` 이 주어지지 않는다면, 각각의 파일은 파일 확장자를 제외한 랜덤한 이름으로 지어질 것입니다.

주: Multer는 어떠한 파일 확장자도 추가하지 않습니다. 사용자 함수는 파일 확장자를 온전히 포함한 파일명을 반환해야 합니다.

결정을 돕기 위해 각각의 함수는 요청 정보 (`req`) 와 파일 (`file`) 에 대한 정보를 모두 전달 받습니다.

`req.body` 는 완전히 채워지지 않았을 수도 있습니다. 이는 클라이언트가 필드와 파일을 서버로 전송하는 순서에 따라 다릅니다.

`MemoryStorage`

메모리 스토리지 엔진은 파일을 메모리에 `Buffer` 객체로 저장합니다. 이에 대해서는 어떤 옵션도 없습니다.

```
const storage = multer.memoryStorage()
const upload = multer({ storage: storage })
```

메모리 스토리지 사용시, 파일 정보는 파일 전체를 포함하는 `buffer` 라고 불리는 필드를 포함할 것입니다.

주의: 메모리 스토리지를 사용시, 매우 큰 사이즈의 파일을 업로드 하거나 많은 양의 비교적 작은 파일들을 매우 빠르게 업로드 하는 경우 응용 프로그램의 메모리 부족이 발생 할 수 있습니다.

`limits`

다음의 선택적 속성의 크기 제한을 지정하는 객체입니다. Multer 는 이 객체를 busboy로 직접 전달합니다. 속성들에 대한 자세한 내용은 busboy's page 에서 확인 하실 수 있습니다.

다음과 같은 정수 값들이 가능합니다:

속성	설명	기본값
`fieldNameSize`	필드명 사이즈 최대값	100 bytes
`fieldSize`	필드값 사이즈 최대값	1MB
`fields`	파일형식이 아닌 필드의 최대 개수	무제한
`fileSize`	multipart 형식 폼에서 최대 파일 사이즈(bytes)	무제한
`files`	multipart 형식 폼에서 파일 필드의 최대 개수	무제한
`parts`	For multipart forms, the max number of parts (fields + files)	무제한
`headerPairs`	multipart 형식 폼에서 파싱할 헤더의 key=>value 쌍의 최대 개수	2000

사이즈 제한을 지정하면 서비스 거부 (DoS) 공격으로부터 사이트를 보호하는데 도움이 됩니다.

`fileFilter`

어느 파일을 업로드 할지, 혹은 건너뛸지 제어할 수 있게 함수에 설정합니다. 해당 함수는 아래와 같을 것입니다 :

```
function fileFilter (req, file, cb) {

  // 이 함수는 boolean 값과 함께 `cb`를 호출함으로써 해당 파일을 업로드 할지 여부를 나타낼 수 있습니다.
  // 이 파일을 거부하려면 다음과 같이 `false` 를 전달합니다:
  cb(null, false)

  // 이 파일을 허용하려면 다음과 같이 `true` 를 전달합니다:
  cb(null, true)

  // 무언가 문제가 생겼다면 언제나 에러를 전달할 수 있습니다:
  cb(new Error('I don\'t have a clue!'))

}
```

에러 핸들링

에러가 발생할 때, multer는 에러를 express에 위임할 것입니다. 여러분은 the standard express way 를 이용해서 멋진 오류 페이지를 보여줄 수 있습니다.

만일 multer 로부터 특별히 에러를 캐치하고 싶다면, 직접 미들웨어 함수를 호출하세요.

```
const upload = multer().single('avatar')

app.post('/profile', function (req, res) {
  upload(req, res, function (err) {
```

```
    if (err) {
      // 업로드할때 오류가 발생함
      return
    }

    // 정상적으로 완료됨
  })
})
```