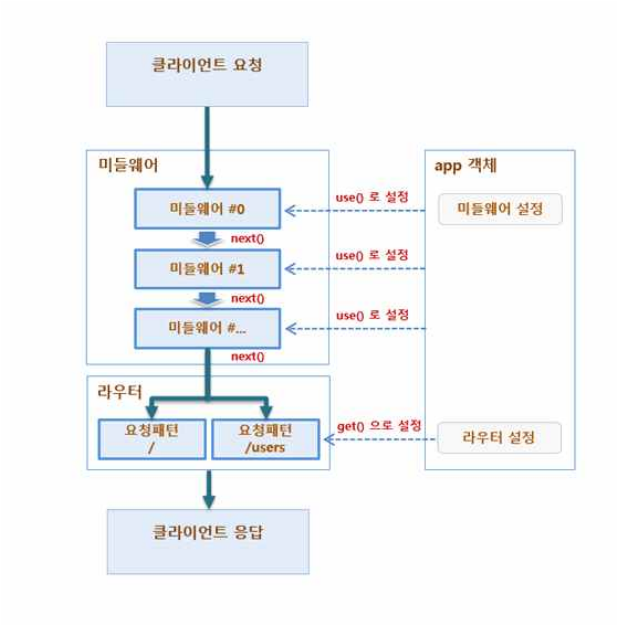
multer 미들웨어를 사용해 파일 업로드 하기



글로벌 미들웨어: 모든 요청에 대해 동작하며, 일반적으로 라우터 전에 설정된다.
예를 들어, 로깅, 보안, 요청 본문 파싱(body-parser) 등이 여기에 해당함

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

// 모든 요청에 대해 동작하는 글로벌 미들웨어
app.use((req, res, next) => {
    console.log(`${req.method} ${req.url}`);
    next();
});
```

라우터 수준 미들웨어: 특정 라우트에 대해서만 동작하도록 설정된다. 이 경우, 라우터와 함께 정의된다.

```
const router = express.Router();

// 특정 라우터에만 적용되는 미들웨어
router.use((req, res, next) => {
    if (!req.headers['x-auth']) {
        return res.status(403).send('Forbidden');
    }
    next();
});

router.get('/profile', (req, res) => {
    res.send('Profile page');
});

app.use('/user', router);


//에러 처리 미들웨어
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).send('Server Error');
});
```

동작 순서

- **전역 미들웨어 → 라우터 → 라우터 수준 미들웨어 → 응답 → 에러 처리 미들웨어**

코드 작성 순서에서 미들웨어와 라우터를 어떻게 배치하느냐에 따라 애플리케이션의 동작이 달라질 수 있다.
일반적으로 다음과 같은 규칙을 따르는 것이 좋다:

1. 전역 미들웨어는 라우터 전에 배치
로깅, 인증, 요청 본문 파싱 등 모든 요청에 적용해야 하는 미들웨어는 라우터 정의 전에 배치한다.
이렇게 하면 모든 요청이 라우터로 전달되기 전에 이 미들웨어를 거치게 된다.

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

// 전역 미들웨어
app.use(express.json()); // 요청 본문 파싱
app.use((req, res, next) => {
    console.log('Request received:', req.method, req.url);
    next();
});

// 라우터
app.get('/hello', (req, res) => {
    res.send('Hello, world!');
});
```

2. 특정 라우터에만 적용할 미들웨어는 해당 라우터와 함께 정의
특정 경로에만 적용할 미들웨어는 해당 라우터와 함께 정의한다.
이렇게 하면 특정 요청에 대해서만 미들웨어가 적용된다.

```
const router = express.Router();

// 특정 라우터에만 적용되는 미들웨어
router.use((req, res, next) => {
    if (!req.headers['x-auth']) {
        return res.status(403).send('Forbidden');
    }
    next();
});

router.get('/profile', (req, res) => {
    res.send('Profile page');
});

app.use('/user', router);
```

3. 에러 처리 미들웨어는 라우터와 다른 미들웨어 뒤에 배치
에러 처리 미들웨어는 요청 처리 중 발생한 에러를 잡아내야 하기 때문에, 가장 마지막에 배치해야 한다.
그렇지 않으면 에러가 발생했을 때, 에러 처리 미들웨어가 실행되지 않을 수 있다.

```
app.use((err, req, res, next) => {
    console.error('Error occurred:', err);
    res.status(500).send('Internal Server Error');
});
```

노드 프로젝트의 기본 폴더 구조 예시

```
my-node-project/
├── node_modules/
├── public/
│   ├── css/
│   ├── js/
│   └── images/
├── src/
│   ├── controllers/
│   ├── models/
│   ├── routes/
│   ├── services/
│   ├── middlewares/
│   └── utils/
├── views/
│   ├── layouts/
│   ├── partials/
│   └── pages/
├── tests/
├── config/
├── .env
├── .gitignore
├── package.json
├── package-lock.json
└── server.js (또는 app.js)
```

- **클라이언트 요청 → 라우터가 요청을 컨트롤러로 전달**.
- **컨트롤러**는 **서비스**를 호출하여 필요한 비즈니스 로직을 처리.
- **서비스**는 **모델**과 상호작용하여 데이터를 처리하거나 외부 API를 호출.
- **컨트롤러**는 서비스로부터 받은 데이터를 클라이언트에 **응답**으로 전달

- **1. node_modules/**
- 프로젝트의 종속성(dependencies)이 설치되는 폴더입니다. npm install 명령어로 설치된 모든 패키지가 여기에 저장됩니다.
- **2. public/**
- 정적 파일(CSS, JavaScript, 이미지 등)을 저장하는 폴더입니다. 서버에서 클라이언트로 직접 제공됩니다.
- css/, js/, images/: 스타일 시트, 클라이언트 측 JavaScript 파일, 이미지 파일 등을 위한 서브 폴더입니다.
- **3. src/**
- 애플리케이션의 주요 소스 코드가 위치하는 폴더입니다.
- controllers/: 요청을 처리하고, 비즈니스 로직을 수행하며, 데이터를 모델과 뷰 사이에서 주고받는 역할을 합니다. 예: userController.js
- models/: 데이터베이스와 상호작용하는 로직을 포함한 파일들이 위치합니다. 데이터 스키마와 관련된 파일들도 여기에 위치합니다. 예: userModel.js
- routes/: URL 경로와 해당 경로에 연결된 컨트롤러를 정의합니다. 라우팅 설정이 포함됩니다. 예: userRoutes.js
- services/: 컨트롤러에서 호출되는 비즈니스 로직을 포함합니다. 모델과 컨트롤러 사이에서 중간 역할을 하는 경우가 많습니다.
- middlewares/: 미들웨어 함수들을 정의하는 폴더입니다. 예: 인증, 로깅, 에러 핸들링 등.
- utils/: 유틸리티 함수들을 모아 놓는 폴더입니다. 예: 공통으로 사용되는 함수들, 헬퍼 함수들.

- **views/**
- 템플릿 파일들이 위치합니다. 보통 템플릿 엔진(e.g., EJS, Pug)과 함께 사용됩니다.
- layouts/: 페이지의 공통 레이아웃 파일들을 저장합니다.
- partials/: 페이지의 재사용 가능한 부분들(헤더, 푸터 등)을 저장합니다.
- pages/: 개별 페이지 템플릿 파일들이 위치합니다.
- 5. tests/
- 테스트 코드가 위치하는 폴더입니다. 유닛 테스트, 통합 테스트 등을 위한 파일들을 저장합니다.
- 6. config/
- 설정 파일들을 저장하는 폴더입니다. 예를 들어, 데이터베이스 설정, 환경 설정 파일들이 여기에 위치할 수 있습니다. 또한, 환경 변수 파일을 정의할 수도 있습니다.
- 7. .env
- 환경 변수를 정의하는 파일입니다. 데이터베이스 URL, API 키, 포트 번호 등 민감한 정보를 저장합니다.
-

multer 미들웨어를 사용해 파일 업로드 하기 1

참조

http://expressjs.com/ko/4x/api.html#req.body

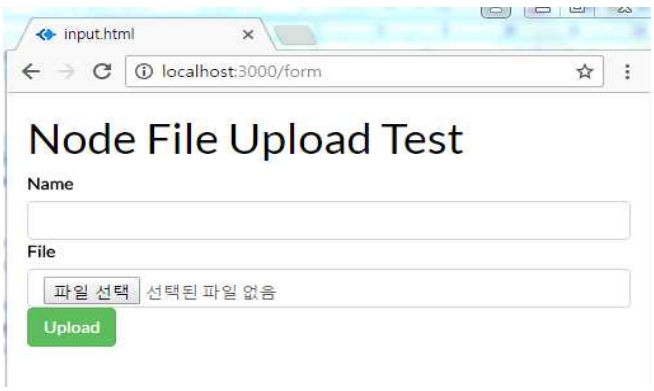
https://opentutorials.org/course/2136/11959
http://bcho.tistory.com/1078
http://blog.naver.com/hyoun1202/220670669034

[1] 입력 폼 작성 - 파일 업로드를 1개 할 경우

TestNodeFileUp/public/input.html

TestNodeFileUp/public/input.html

```
<!DOCTYPE html>
<html><head><title>input.html</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<script type="text/javascript"
        src="http://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script>
<script type="text/javascript"

src="http://netdna.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js"></script>
<link

href="http://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.3.0/css/font-awesome.min.css"
        rel="stylesheet" type="text/css">
<link

href="http://pingendo.github.io/pingendo-bootstrap/themes/default/bootstrap.css"
        rel="stylesheet" type="text/css">
</head>
<body>
<div class="container">
<h1>Node File Upload Test</h1>
<form action="/uploadEnd" method="post" enctype="multipart/form-data"
role="form">
    <label for="username">Name</label>
    <input type="text" name="username" id="username" class="form-control">
    <p>
<label for="myfile1">File</label>
    <input type="file" name="myfile1" id="myfile1"  class="form-control">
    <input type="submit" value="Upload" class="btn btn-success">
 </form>
</div>
</body></html>
```



[2] multer모듈 설치

도스창에서 npm install –save multer 으로 설치하던지
package.json파일에 의존 모듈 추가한다. 우리는 두 번째 방법으로

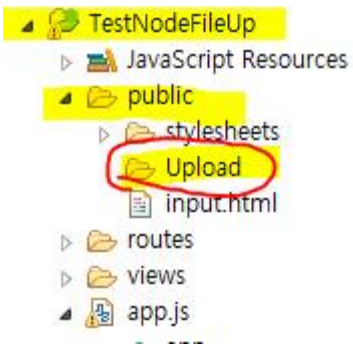
TestNodeFileUp/package.json

```
{
  "name": "TestNodeFileUp",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
                "body-parser": "~1.13.2",
              "cookie-parser": "~1.3.5",
              "debug": "~2.2.0",
              "ejs": "~2.3.3",
              "express": "3.2.6",
              "morgan": "~1.6.1",
              "serve-favicon": "~2.3.0",
              "multer":"~1.1.0"
  }
}
```

package.json 선택후 오른 마우스 클릭 ->Run As -> npm install 선택하여 설치한다.

[3] 업로드할 디렉토리를 생성한다.

TestNodeFileUp/public/ 디렉토리 안에 Upload란 폴더를 만들고 그곳에 업로드할 예정

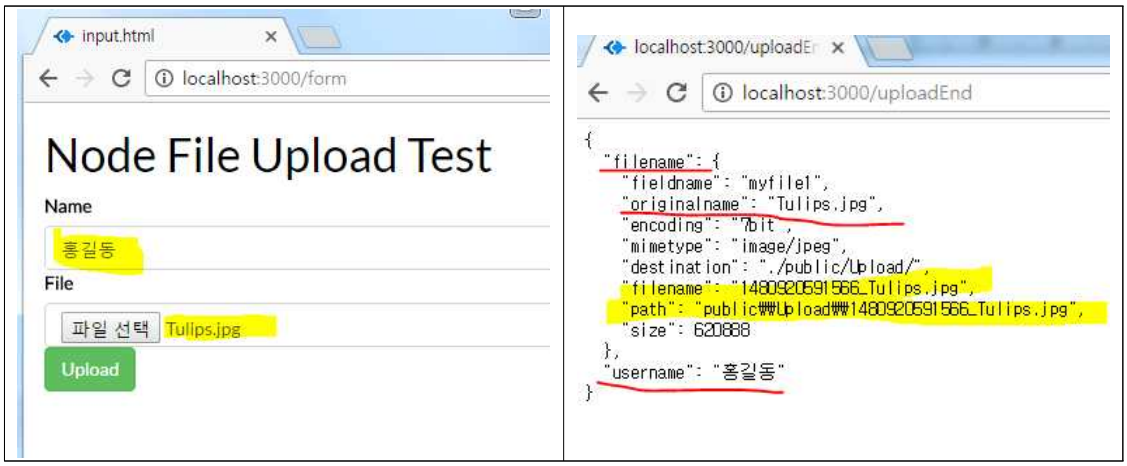


[4] app.js 작성

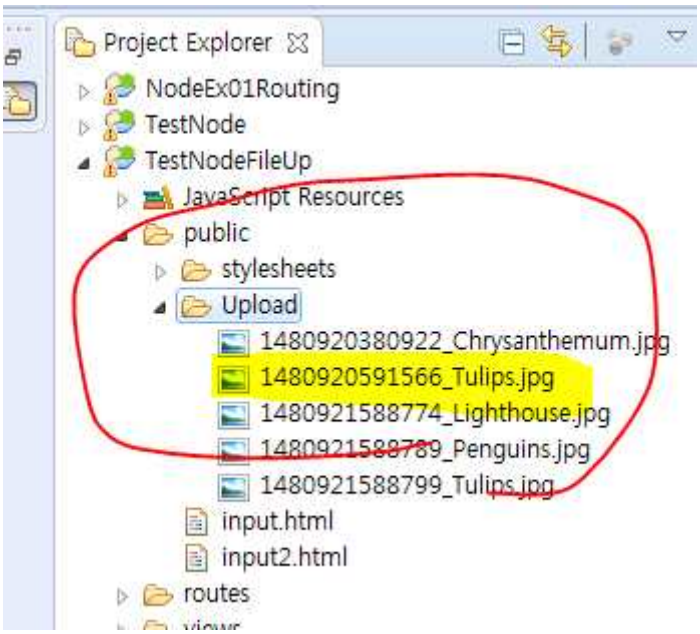
TestNodeFileUp/app.js

```
var express = require('express')
  , routes = require('./routes')
  , user = require('./routes/user')
  , http = require('http')
  , path = require('path'),
    fs=require('fs');
//파일업로드시 multer를 쓰던지 bodyParser를 쓰던지..둘 중 하나만 해야 파일 업로드가 됨
//var bodyParser=require('body-parser'); //주석 처리
////////////////////////////////////
var multer=require('multer');
////////////////////////////////////
```

```
var app = express();

// all environments
app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
app.use(express.favicon());
app.use(express.logger('dev'));
//app.use(express.bodyParser());
//파일업로드시 multer를 쓰던지 bodyParser를 쓰던지..둘 중 하나만 해야 파일 업로드가 됨
//app.use(bodyParser.json());
////////////////////////
app.use(multer());
////////////////////////
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));


// development only
if ('development' == app.get('env')) {
  app.use(express.errorHandler());
}

app.get('/', routes.index);
app.get('/users', user.list);
//파일 입력 폼 요청 처리
app.get('/form', function(req, res, next) {
        fs.readFile("public/input.html", function(err, data) {
                res.send(data.toString());
        });
});
//multer를 이용해 저장할 곳 지정 및 파일명 중복을 피하기 위해 업로드한 날짜와 파일명을 결합한다.
var storage=multer.diskStorage({
        destination: function(req,file,callback){
                callback(null,"./public/Upload/");
        },
        filename: function(req,file,callback){
                callback(null,Date.now()+"_"+file.originalname);
                console.log(">>>"+file.originalname);
        }
});

var upmulter=multer({storage:storage});

app.post("/uploadEnd",upmulter.single('myfile1'),function(req,res){
        console.log("File: "+req.file);
        //파일 정보는 req.file로, 파라미터값은 req.body.파라미터명 으로 추출
        res.json({'filename' : req.file,
                'username' : req.body.username});
        //업로드된 파일 정보를 json형태로 브라우저에 출력해보자.

});

http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```



[5] 실제로 업로드 됐는지 확인 해본다. Upload폴더를 선택해 새로고침 해보자.



폼을 통해 업로드한 파일이 1개일 때 req.file을 이용해 업로드한 파일 정보를 추출할 수 있다. 반면 업로드한 파일이 여러 개 일 때는 req.files 임에 주의하자.

또한 multer를 통해 업로드시 **파일이 1개일 때는 multer.single('필드명')**을 사용하는 반면 **여러 개 일 때는 multer.array('필드명')**을 사용한다.

다음 예제는 여러 파일을 업로드 할 때의 예제이다.
input2.html 파일과 app2.js를 작성해 테스트 해보자.

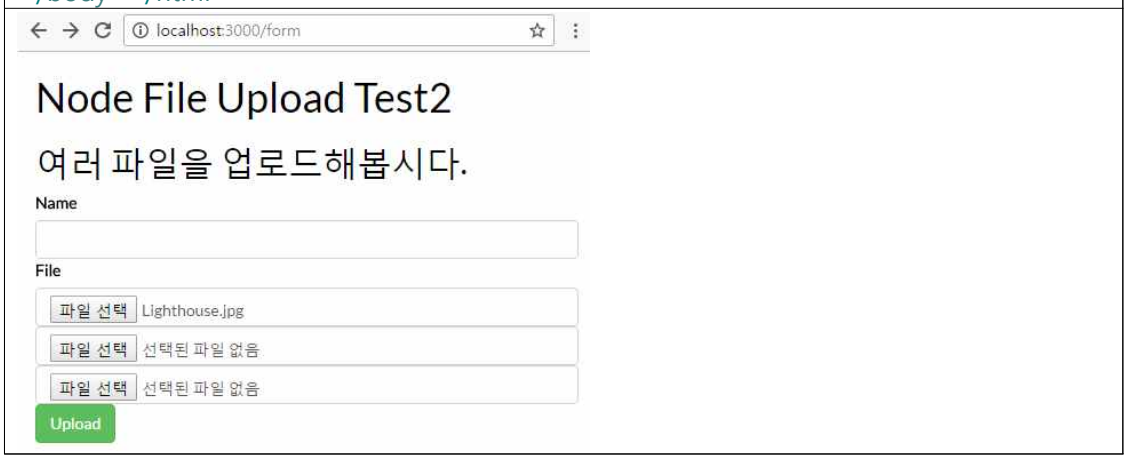
multer 미들웨어 사용해 파일 업로드 하기 2

[1] 입력 폼 작성 - 파일 업로드를 여러 개 할 경우

TestNodeFileUp/public/input.html

TestNodeFileUp/public/input2.html

```
<!DOCTYPE html>
<html><head><title>input.html</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<script type="text/javascript"
        src="http://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script>
<script type="text/javascript"

src="http://netdna.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js"></script>
<link

href="http://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.3.0/css/font-awesome.min.css"
        rel="stylesheet" type="text/css">
<link

href="http://pingendo.github.io/pingendo-bootstrap/themes/default/bootstrap.css"
        rel="stylesheet" type="text/css">
</head>
<body>
<div class="container">
<h1>Node File Upload Test2</h1>
<h2>여러 파일을 업로드해봅시다.</h2>
<form action="/uploadEnd" method="post" enctype="multipart/form-data"
role="form">
    <label for="username">Name</label>
    <input type="text" name="username" id="username" class="form-control">
    <p>
<label for="myfile1">File</label>
    <input type="file" name="myfile1" id="myfile1" class="form-control">
    <input type="file" name="myfile1" id="myfile2" class="form-control">
    <input type="file" name="myfile1" id="myfile3" class="form-control">
    <input type="submit" value="Upload" class="btn btn-success">
 </form>
</div>
</body></html>
```

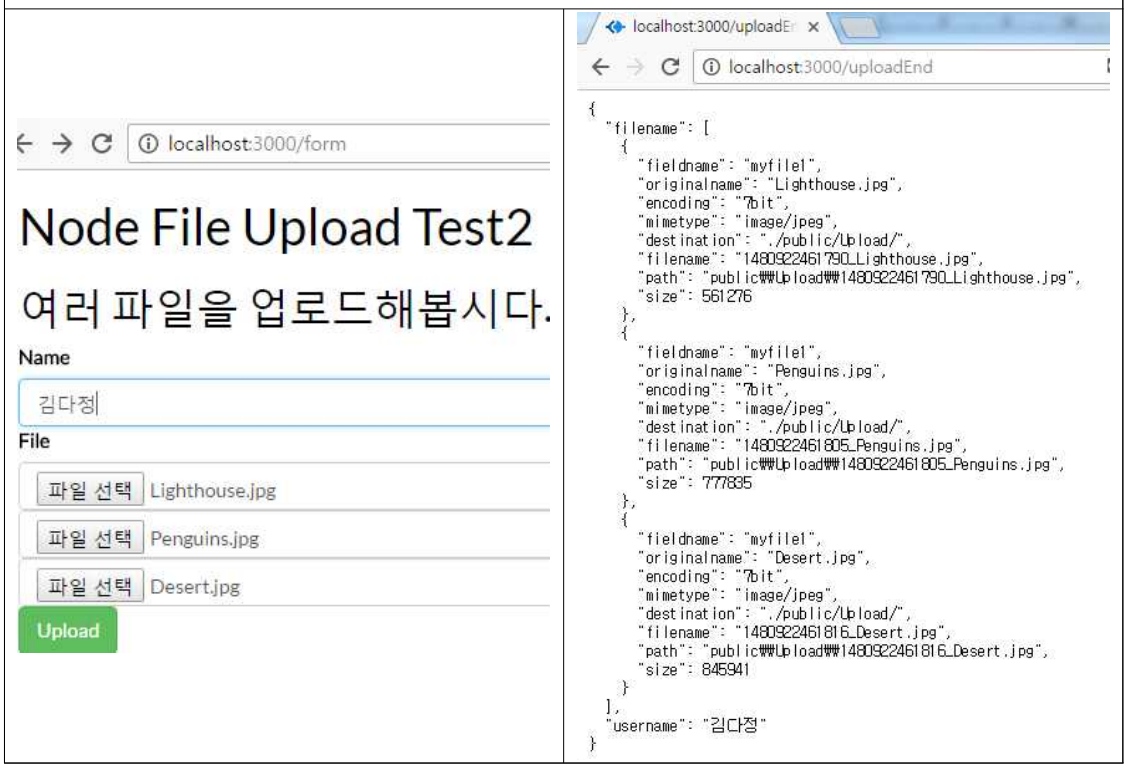


[2] app2.js 작성

TestNodeFileUp/app2.js

```
var express = require('express')
  , routes = require('./routes')
  , user = require('./routes/user')
  , http = require('http')
  , path = require('path'),
    fs=require('fs');
//파일업로드시 multer를 쓰던지 bodyParser를 쓰던지..둘 중 하나만 해야 파일 업로드가 됨
//var bodyParser=require('body-parser');
////////////////////////////////////
var multer=require('multer');
////////////////////////////////////
//첨부파일을 구분하기 위한 변수 선언
var maxFileCount=3;//첨부파일 허용 갯수
var maxFileSize=10*1024*1024;//첨부 가능한 최대 파일 사이즈 10Mb로 설정

////////////////////////////////////
var app = express();

// all environments
app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
app.use(express.favicon());
app.use(express.logger('dev'));
//app.use(express.bodyParser());
//파일업로드시 multer를 쓰던지 bodyParser를 쓰던지..둘 중 하나만 해야 파일 업로드가 됨
//app.use(bodyParser.json());
////////////////////////
app.use(multer());
////////////////////////
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));


// development only
if ('development' == app.get('env')) {
  app.use(express.errorHandler());
}

app.get('/', routes.index);
app.get('/users', user.list);
//파일 입력 폼 요청 처리
app.get('/form', function(req, res, next) {
        fs.readFile("public/input2.html", function(err, data) {
                res.send(data.toString());
        });
});
//multer를 이용해 저장할 곳 지정 및 파일명 중복을 피하기 위해 업로드한 날짜와 파일명을 결합한다.
var storage=multer.diskStorage({
        destination: function(req,file,callback){
                callback(null,"./public/Upload/");
        },
        filename: function(req,file,callback){
                callback(null,Date.now()+"_"+file.originalname);
                console.log(">>>"+file.originalname);
        }
```

```
});

var upmulter=multer({storage:storage,
        limits:{fileSize: maxFileSize}});   //파일 크기 10Mb로 제한하였음

//app.post("/uploadEnd",upmulter.single('myfile1'),function(req,res){ //1개 업로드시
//여러 개 업로드시는 array() 사용
app.post("/uploadEnd",upmulter.array('myfile1',maxFileCount),function(req,res){
        console.log("File: "+req.files);
        //파일 정보는 req.files (복수형임에 주의 files)로, 파라미터값은
req.body.파라미터명 으로 추출
        res.json({'filename' : req.files,
                'username':req.body.username});
        //업로드된 파일 정보를 json형태로 브라우저에 출력해보자.

});

http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server2 listening on port ' + app.get('port'));
});
```



public/Upload디렉토리를 F5를 눌러 새로고침하면 업로드 된 것 확인할 수 있다.