express 3.0.0

web application framework for node

based 4.13.4

softcontext@gmail.com

based 1.2.16

softcontext@gmail.com

Express

익스프레스는 웹 애플리케이션을 만드는 데 필요한 기능을 제공하는 노드기반 웹 애플리케이션 프레임워크다.

익스프레스 첫 버전은 2010 년 11 월에 발표되었다.
익스프레스는 npm 레지스트리에서 가장 인기있는 모듈중의 하나가 되었다.

프레임워크를 사용하지 않고 개발자가 직접 웹 서버 프로그램이 기대하는 모든 기능을 노드의 핵심 모듈만 사용해서 개발하는 것은 생산적이지 않다.

익스프레스는 라우팅, 정적 리소스, 뷰 엔진 통합, 커뮤니티 연동 모듈 등을 제공한다.

Express.js is a Node.js web application server framework,
designed for building single-page, multi-page, and hybrid web applications.

It is the de facto standard server framework for node.js.

The original author, **TJ Holowaychuk**, described it
as a Sinatra-inspired server, meaning that
it is relatively minimal with many features available as plugins.

Express is the backend part of the MEAN stack,
together with MongoDB database and AngularJS frontend framework.

In June 2014, rights to manage the project were acquired by **StrongLoop**.
StrongLoop was acquired by **IBM** in September 2015;
in January 2016, IBM announced that
it would place Express.js under the stewardship of the **Node.js Foundation** incubator.

익스프레스를 사용하는 이유

1. 사용자의 요청을 구분하는 다양한 라우팅 방법을 제공한다.
   익스프레스를 사용하면 쉽게 Restful 서비스를 구축할 수 있다.

2. 사용자의 요청을 처리하는 다양한 미들웨어 연동설정 작업이 쉬워진다.
   서버의 처리 로직 흐름 중간에 배치하는 모듈을 미들웨어라 부른다.

3. 사용자의 요청에 응답하는 다양한 뷰를 연동하기 쉽게 만들어 준다.
   ejs, handlebars, jade, jshtml, hogan.js

4. 개발 생산성 및 유지보수성이 올라간다.
   라우팅 로직을 별도에 모듈로 쉽게 분리할 수 있다.

5. 체계적인 소프트웨어 구조를 제공함으로써 협업에 유리하다.
   개발자간의 의사소통이 원활해진다.

온라인 매뉴얼

http://expressjs.com/

Express 설치

1. 새로운 프로젝트를 위한 폴더를 만든다.

2. 다음 명령으로 프로젝트 정보와 디펜던시 정보를 관리할 package.json 파일을 만든다.

```
npm init
```

3. 익스프레스 모듈을 설치한다.
프로젝트의 루트 폴더에서 다음 명령으로 익스프레스 모듈을 설치한다.

```
npm install --save express
```

4. 프로젝트에 필요한 모듈들을 설치한다.

JSON 문자열 또는 x-www-form-urlencoded 형식으로 서버에 전송되는 데이터를 서버에서 쉽게 처리할 수 있도록 도와주는 모듈

```
npm install --save body-parser
```

브라우저의 쿠키를 쉽게 사용할 수 있게 도와주는 모듈

```
npm install --save cookie-parser
```

세션 모듈

```
npm install --save express-session
```

multipart/form-data 를 처리하는 파일 업로드 모듈

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

```
npm install --save formidable
```

몽고디비 연동 모듈

```
npm install --save mongoose
```

package.json : 프로젝트의 시작과 끝

사용하는 모듈의 버전관리는 크리티컬하다. 프로젝트 시작을 언제나 package.json 파일을 만들면서 시작하자. 개발자가 다른 모듈을 의존하여 새로 개발한 프로젝트도 github 를 통해서 공유한다면 또 하나의 모듈이 된다. 프로젝트 관리와 배포를 위해서 package.json 파일은 중요하다.

모듈 설치 시 --save 옵션을 설정하면 설치된 모듈 정보가 package.json 파일에 자동으로 추가된다. 노드 개발은 버전관리가 매우 중요함으로 프로젝트 관리문서는 필수라고 항상 생각하자.

package.json 샘플

```
{
    "name": "express-start",
    "version": "1.0.0",
    "description": "",
    "main": "app.js",
    "scripts":
    {
        "test": "echo ₩"Error: no test specified₩" && exit 1"
    },
    "author": "chris",
    "license": "MIT",
    "dependencies":
    {
        "body-parser": "^1.15.0",
        "cookie-parser": "^1.4.1",
        "express": "^4.13.4",
        "express-session": "^1.13.0",
        "multer": "^1.1.0"
    }
}
```

package.json 을 이용한 모듈 설치

package.json 파일에 dependencies 설정을 먼저하고 이를 바탕으로 모듈을 일괄적으로 설치할 수 있어 매우 편리하다. 파일이 있는 위치에서 다음 명령을 사용하면 된다. 모듈명이 생략되면 자동으로 package.json 파일을 찾아서 설정된 디펜더시 정보를 참고하여 모듈을 설치한다.

```
npm install
```

보통 모듈을 공유할 때 디펜던시 모듈의 실제 파일들은 제외하고 대신 package.json 파일만을 주는 관행은 위 기능이 있기 때문이라고 할 수 있다. 사이즈가 큰 디펜던시 모듈을 매번 주는 것은 사실 필요 없는 행위라 할 수 있다.

package.json 매뉴얼

https://docs.npmjs.com/files/package.json

디펜던시 표기법

https://docs.npmjs.com/files/package.json#dependencies

1. 첫 예제

간단히 첫 예제를 작성해 보자.

/1/app.js

```
var express = require('express');

var app = express();

app.get('/', function(req, res) {
    // 노드의 res.writeHead 함수에 해당한다.
    // 익스프레스가 확장한 메소드다.
    res.status(404);
    res.type('text/plain');
    // 노드의 res.end 함수에 해당한다.
    // 익스프레스가 확장한 메소드다.
    res.send('Hello World');
});

var server = app.listen(8080, '127.0.0.1', function() {

    var host = server.address().address;
    var port = server.address().port;

    // 서버 IP 를 생략하면 기본적으로 localhost 를 사용하기 때문에
    // host 정보 표시위치에 :: 문자열만 출력될 수 있다.
    console.log("sever is running on http://%s:%s", host, port);

});
```

서버를 시작하고 브라우저를 사용하여 루트('/')로 접근 해 보자.

Hello World 문자열이 보이면 성공이다.

2. 라우팅 기능 확장

GET, POST, PUT, DELETE 요청방식의 처리방법을 살펴보자.

테스트 툴로 구글의 postman 을 사용하면 다양한 요청방식을 쉽게 테스트할 수 있다.
참고사이트 https://www.getpostman.com/

/2/app.js

```
var express = require('express');

var app = express();

function print(url, method, res){
	console.log(url+' '+method);
	res.send(url+' '+method);
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// 서버 루트로 접근하는 GET 요청처리
app.get('/', function(req, res) {
	print(req.url, req.method, res);
});

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// 정보 요청
app.get('/user', function(req, res) {
	print(req.url, req.method, res);
});

// 정보저장 요청
app.post('/user', function(req, res) {
	print(req.url, req.method, res);
});

// 정보수정 요청
app.put('/user/:id', function(req, res) {
	console.log(req.params.id+' 값으로 대상을 찾아서 수정한다.');
```

```
    print(req.url, req.method, res);
});

// 정보삭제 요청
// delete 는 자바스크립트 예약어이다.
// 일부 디버거의 에러표시를 보이지 않게 하기 위해서 [] 표기법으로 바꾸어서 사용할 수 있다.
app['delete']('/user/:id', function (req, res) {
    console.log(req.params.id+' 값으로 대상을 찾아서 삭제한다.');
    print(req.url, req.method, res);
});

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// * 를 사용하면 해당 위치에 어떠한 문자(들)가 있어도 연동된다.
// abcd, abxcd, ab123cd ...
app.get('/ab*cd', function(req, res) {
    print(req.url, req.method, res);
});

var server = app.listen(8080, function() {
    var host = server.address().address;
    var port = server.address().port;
    console.log("sever is running on http://%s:%s", host, port);

});
```

온라인 매뉴얼

http://expressjs.com/en/guide/routing.html

정규표현식

메타 문자 중 +, ?, *, (), | 를 라우트 경로에 사용할 수 있다.

'/user(name)?'

/crazy|mad(ness)?|luna/

라우팅 연동 규칙

1. 대소문자를 구분하지 않는다.
2. 맨 뒤의 슬래시는 무시한다.
3. 파라미터는 스트링은 무시한다.

라우팅 로직의 분리

라우팅 로직을 서버 로직에서 분리해서 관리성을 증대시키자.

URL Pattern 처리의 중복이 있다면 앞에 있는 함수만이 작동한다.

/routes/index.js

```
var router = require('express').Router();

function print(req, res){
	console.log(req.url + ' ' + req.method);
	res.send(req.url + ' ' + req.method);
}

router.get('/', function(req, res) {
	print(req, res);
});

router.get('/about', function(req, res) {
	print(req, res);
});

// RESTful Router for member
var member = router.route('/member');

member.all(function(req, res, next) {
	console.log(req.url, req.method);
	next(); // 다음에 배치된 함수가 기동하게 만든다.
});
```

```
member.get(function(req, res) {
	print(req, res);
});

member.post(function(req, res) {
	print(req, res);
});

member.put(function(req, res) {
	print(req, res);
});

member['delete'](function(req, res) {
	print(req, res);
});

exports.router = router;
```

/app.js

```
var express = require('express'), app = express();
var router = require('./routes/index').router;
// 설정값을 저장한다.
app.set('port', 3000);

// 모든 URL Pattern 요청이 router 에게 전달된다.
app.use('/', router);

app.get('/admin', function(req, res) {
	console.log(req.url + ' ' + req.method);
	res.send(req.url + ' ' + req.method);
});

var server = app.listen(app.get('port'), function() {
	console.log("server is running on http://localhost:%s", app.get('port'));
});
```

3. 정적 리소스 제공 서비스

익스프레스가 제공하는 내장 미들웨어인 express.static 을 사용하면 간단하게 설정하여 정적리소스 파일을 서버에서 클라이언트에게 전달할 수 있다.

정적리소스 파일의 위치를 알려주는 것으로 설정이 끝난다.

app.js 파일이 있는 위치에서 public 폴더를 만들고 그 하부에 필요한 폴더와 파일을 배치한다.

/public/html/demo.html

/public/image/logo.png

/3/app.js

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

// app.use() 함수는 미들웨어를 설정하는 함수다.
// 정적리소스의 루트는 public 폴더가 된다.
// 뷰에서 정적리소스 사용 시 public 폴더는 생략한다.
app.use(express.static('public'));

app.get('/', function(req, res) {
	res.send('test express.static');
});

var server = app.listen(8080, function() {
	var host = server.address().address;
	var port = server.address().port;
	console.log("sever is running on http://%s:%s", host, port);
});
```

서버를 재 시작하고 다음 URL 로 접근하여 테스트 해 보자.

http://localhost:8080/html/demo.html

http://localhost:8080/image/logo.png

4. GET 방식 파라미터 처리

<a> 태그는 GET 요청방식이다. 사용자가 주소창에 직접 파라미터를 입력하는 것과 같다.

/4/public/html/home.html

```
<html>
<body>
    <h2>home.html</h2>
    <hr>
    <a href="user?name=chris&age=21">request and send parameters</a>
</body>
</html>
```

/4/app.js

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

app.use(express.static('public'));

app.get('/', function(req, res) {
    res.sendFile(__dirname + "/public/html/home.html");
});

app.get('/user', function(req, res) {
    var user = {
        name : req.query.name,
        age : req.query.age
    };
    console.log(user);

    res.send(JSON.stringify(user));
});

var server = app.listen(8080, function() {
    var host = server.address().address;
    var port = server.address().port;
```

```
    console.log("sever is running on http://%s:%s", host, port);
});
```

브라우저와 서버의 대화 순서

1. 사용자가 서버에 접속하면 첫 접근은 일반적으로 루트 접근이다.
서버의 라우팅 함수가 처리한다. 이 때 첫 HTML 페이지를 브라우저에게 보낸다.

```
app.get('/', function(req, res) {
    res.sendFile(__dirname + "/public/html/home.html");
});
```

2. 사용자가 home.html 에 설정되어 있는 링크를 클릭하면 미리 설정되어 있는 파라미터 정보가 GET 방식으로 서버로 전송된다.

```
<a href="user?name=chris&age=21">request and send parameters</a>
```

url pattern	parameter
user	name=chris&age=21

사용자는 직접 브라우저 주소창에 url pattern 과 parameter 를 입력하여 서버에 요청할 수도 있다.

3. 사용자의 요청인 url pattern 이 서버에 전달된다. 서버 함수에 설정되어 있는 라우팅 패턴과 일치하면 콜백함수가 기동한다. GET 방식으로 전달된 파라미터 정보는 req 객체에 query 프로퍼티가 객체 상태로 갖고 있다.

```
app.get('/user', function(req, res) {
    var user = {
        name : req.query.name,
        age : req.query.age
    };
    console.log(user);
});
```

4. 사용자가 보낸 파라미터 정보를 JSON 표기법의 문자열로 다시 브라우저에게 전송하여 확인한다. res.send(JSON.stringify(user));

5. POST 방식 파라미터 처리

사용자가 작성한 데이터를 서버로 보낼 때 POST 요청방식을 사용한다. 사이즈가 작다고 서버로 정보를 보낼 때 GET 방식을 사용하는 것은 가능은 하지만 잘 못된 용례다.
GET 방식은 사용자가 작성한 정보를 브라우저 주소창에 노출시키므로 사용을 삼가자.

/5/public/html/home.html

```
<html>
<body>
    <h2>home.html</h2>
    <hr>
    <form action="user" method="POST">
        <div>name: <input type="text" name="name"></div>
        <div>age: <input type="text" name="age"></div>
        <div><input type="submit" value="send"></div>
    </form>
</body>
</html>
```

POST 방식으로 전송된 데이터를 처리하기위해서 body-parser 모듈을 사용한다.

/5/app.js

```
var express = require('express');
var bodyParser = require('body-parser');

var app = express();

app.use(express.static('public'));

// x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false });

app.get('/', function (req, res) {
    res.sendFile( __dirname + "/public/html/home.html" );
});
```

```
app.post('/user', urlencodedParser, function (req, res) {
    var user = {
        name:req.body.name,
        age:req.body.age
    };
    console.log(user);

    res.send(JSON.stringify(user));
});

var server = app.listen(8080, function() {
     var host = server.address().address;
     var port = server.address().port;
     console.log("sever is running on http://%s:%s", host, port);
});
```

bodyParser.urlencoded({ extended: false })

string, array 를 처리한다.

bodyParser.urlencoded({ extended: true })

모든 종류의 데이터를 처리한다.

온라인 매뉴얼

https://www.npmjs.com/package/body-parser

6. Path Variable

접근 경로에 쓰여진 문자열을 파라미터화 하여 함수에서 받은 후 사용할 수 있다.
패스변수를 사용하여 보여주고자 하는 데이터와 연동하면 보다 사용자 친화적인 사이트 서비스를 제공할 수 있다.

/server-path-variable.js

```
var express = require('express'), app = express();

app.set('port', 3000);

app.get('/:category/product/:id', function(req, res) {
	console.log(req.params.category +'/product/' + req.params.id);
	res.send(req.params.category +'/product/' + req.params.id);
});

var server = app.listen(app.get('port'), function() {
	console.log("server is running on http://localhost:%s", app.get('port'));
});
```

7. 파일 업로드 with multer

다양한 기능을 제공하는 파일 업로드 모듈 멀터를 사용해 보자.

/public/html/home.html

```
<html>
<body>
    <h2>home.html</h2>
    <hr>

    <form action="upload" method="POST" enctype="multipart/form-data">
        <div>file: <input type="file" name="file" size="40"></div>
        <div><input type="submit" value="send"></div>
    </form>
    <hr>

    <form action="uploads" method="POST" enctype="multipart/form-data">
        <div>file: <input type="file" name="imgs" size="40"></div>
        <div>file: <input type="file" name="imgs" size="40"></div>
        <div><input type="submit" value="send"></div>
    </form>
    <hr>

    <form action="filterTest" method="POST" enctype="multipart/form-data">
        <div>name: <input type="text" name="photoName" size="20" value="abc"></div>
        <div>file: <input type="file" name="userPhoto" size="40"></div>
        <div><input type="submit" value="send"></div>
    </form>
</body>
</html>
```

/app.js

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
```

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

// dest 또는 storage 하나만 선택해서 설정해야 한다.
var storage = multer.diskStorage({
    destination: function (req, file, cb) {
        cb(null, './tmp/uploads');
    },
    filename: function (req, file, cb) {
        // 서버 측 저장 파일이름으로 originalname 을 사용한다.
        // 중복을 방지하기 위하여 시각정보를 파일이름 앞에 추가한다.
        cb(null, Date.now() + '!!' + file.originalname);
    }
});
var multiUpload = multer({ storage: storage });

app.use(express.static('public'));
app.use(bodyParser.urlencoded({ extended:false }));

app.get('/', function(req, res) {
    res.sendFile(__dirname + '/public/html/home.html');
});

// 기본적으로 서버에 저장되는 파일이름으로 originalname 을 사용하지 않으며
// 확장자가 누락된다.
app.post('/upload', upload.single('file'), function(req, res, next) {
    // 파일외 데이터 필드
    console.log(req.body);
    // 파일정보
    console.log(req.file);
    // 서버가 브라우저에게 응답으로 보낼 것이 없음을 의미한다.
    res.status(204).end();
});

// 파일 처리를 2 개만 하도록 설정한다. 추가로 이미지를 전송해도 누락은 아니다.
// 자세한 내용은 모듈 개발자 문서를 참조한다.
app.post('/uploads', multiUpload.array('imgs', 2), function(req, res, next) {
    // 파일외 데이터 필드
```

```
    console.log(req.body);
    // 파일정보
    console.log(req.files);

    res.status(204).end();
});

// 마임타입으로 비교하는 필터 예
function fileFilter1(req, file, cb) {
    if (file.mimetype !== 'image/png') {
        req.fileValidationError = 'goes wrong on the mimetype';
        return cb(null, false));
    }
    cb(null, true);
}

// 비디오와 이미지만 서버에 업로드 할 수 있다.
function fileFilter2 (req, file, cb){
    var type = file.mimetype;
    var typeArray = type.split("/");
    if (typeArray[0] == "video" || typeArray[0] == "image") {
        cb(null, true);
    }else {
        cb(null, false);
    }
}

// 10 메가 바이트 이하의 파일만 업로드 할 수 있다.
var filterUploadX = multer({ 'storage': storage, 'fileFilter': fileFilter }, {limits : {fieldNameSize :
10*1024*1024}}).array('pngs', 3);

var filterUpload = multer(
        {
            storage : storage,
            fileFilter : function(req, file, callback) {
                // 파일정보
                console.log(file);
```

```
                    var fileType = ['png', 'jpg'];
                    if (fileType.indexOf(file.originalname.split('.')
                            [file.originalname.split('.').length-1]) === -1) {
                        return callback(new Error('Wrong extension type'));
                    }
                    callback(null, true);
                }
            }).single('userPhoto');

app.post('/filterTest', function(req, res, next){
    console.log(req.url);

    filterUpload(req, res, function (err) {
        if (err) {
            console.log(err.message);
            return res.status(406).send('Not Acceptable');
        }
        // 파일외 데이터 필드
        console.log(req.body);

        res.status(204).end();
    });
});

var server = app.listen(8080, function() {
    var host = server.address().address;
    var port = server.address().port;
    console.log('sever is running on http://%s:%s', host, port);
});
```

8. 에러 핸들링

에러 발생 시 사용자에게 에러 메시지를 있는 그대로 전송하여 노출시키는 것은 좋지 않다.
적절한 에러 핸들링 처리 로직을 사용하자.

404 : Page Not Found

서버에 존재하지 않는 정적리소스나 서버에 설정되지 않은 라우팅 설정정보로 요청하는 경우 서버는 이에 대응할 수 없다. 사용자의 잘 못된 요청이다.

500: Internal Server Error

서버에서 장애가 발생했다. 서버에서 내부적인 처리 시 에러가 발생하여 사용자에게 응답할 수 없는 경우다.

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

/*
 * 파워쉘에서 포트 설정을 하면 기동 시 변경 값을 사용한다.
 * 쉘프롬프트> $env:PORT = 1234
 * 쉘프롬프트> node app.js
 */
app.set('port', process.env.PORT || 3000);
console.log('port: '+app.get('port'));

// 404 핸들러
// 클라이언트가 없는 url pattern 으로 서버에 요청 시 404 핸들러가 처리한다.
// 이 설정은 라우팅 설정함수들 맨 뒤에 위치해야 한다.
app.use(function(req, res){
    res.type('text/plain');
    res.status(404);
    res.send('404 - not found');
});

// 폴백 에러 핸들러, 모든 라우터 다음에 위치시킨다.
// next 파라미터를 명시해야만 익스프레스가 에러 핸들러라고 인식한다.
app.use(function(err, req, res, next){
```

```
    console.error(err.stack);
    res.type('text/plain');
    res.status(500);
    res.send('500 - server error');
});

app.listen(app.get('port'), function(){
    console.log('express is running on http://localhost:'+app.get('port'));
});
```

Handlebars

ejs 는 너무 단순하여 학습비용이 거의 들지 않는다. 반면에 개발자의 하드코딩양이 많아진다.
jade 는 너무 추상적이어서 많은 학습비용이 든다. 하지만 익숙해지면 커다란 도움이 될 것이다.

여기서는 ejs 와 jade 의 중간정도에 해당하는 핸들바를 사용 해 본다.
핸들바는 적절한 학습비용으로 개발 생산성을 높여주는 좋은 HTML Engine 모듈이다.

핸들바 모듈을 설치하자.

```
npm install --save express-handlebars
```

Layout

자바에서 tiles 라이브러리를 사용 해 본 개발자라면 레이아웃 개념이 낯설지 않을 것이다.
레이아웃이란 화면 처리 시 중복되는 뷰를 미리 만들어 놓고 설정으로 원하는 뷰에 적용하여 전체화면을 완성하는 기법에서 사용하는 기본 틀이다.

핸들바 설정 시 기본 레이아웃을 알려 줄 필요가 있다.
require('express-handlebars').create({**defaultLayout**:'**main**'});

/app.js

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

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// 핸들바 사용설정
var handlebars = require('express-handlebars').create({defaultLayout:'main'});
// handlebars 확장자를 사용하는 뷰 처리 시 handlebars.engine 을 사용하도록 만든다.
app.engine('handlebars', handlebars.engine);
// 핸들바 뷰 파일의 확장자를 handlebars 라고 설정한다.
// 확장자는 고정된 값이 아니다. hbs 라고 줄여 쓰기도 한다.
app.set('view engine', 'handlebars');
```

```
app.set('port', process.env.PORT || 3000);

// 뷰 엔진을 사용하면 기본적으로 콘텐츠 타입 text/html 과 상태코드 200 을 반환한다.
// 뷰 엔진으로 handlebars 설정했으므로 익스프레스는 home.handlebars 를 렌더링한다.
app.get('/', function(req, res){
	res.render('home');
});

app.get('/about', function(req, res){
	res.render('about');
});

app.use(function(req, res, next){
	res.type('text/plain');
	res.status(404);
	res.send('404 - not found');
});

app.use(function(err, req, res, next){
	console.error(err.stack);
	res.type('text/plain');
	res.status(500);
	res.send('500 - server error');
});

app.listen(app.get('port'), function(){
	console.log('express is running on http://localhost:'+app.get('port'));
});
```

데이터와 뷰를 결합하여 만들어지는 결과는 HTML 이다. 브라우저에게 전달되는 파일은 HTML 파일이 전송되는 것이다.

res.render() 함수를 사용하여 지정된 뷰 생성엔진을 통해 HTML 파일을 만들 수 있다.
render() 함수의 첫 번째 파라미터인 about 문자열은 about.handlebars 파일을 가리킨다.

기본적으로 views 폴더에 뷰 파일들을 배치한다.

layouts 폴더는 레이아웃 뷰가 배치되는 기본 폴더로 사용된다.

폴더를 변경할 수 있다. 자세한 설정은 모듈 개발자 문서를 참고하자.

/views/**layouts**/main.handlebars

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
</head>
<body>
    {{{body}}}
</body>
</html>
```

{{body}}

중괄호를 두 번 사용하면 컨텐츠 중 태그는 치환 문자열로 처리되어 표시된다.

{{{body}}}

중괄호를 세 번 사용하면 전달되는 컨텐츠의 있는 태그를 치환하지 않는다.

결론적으로 HTML 태그를 자바스크립트 파일에서 선 처리하여 넘길 수 있다는 의미다.

/views/about.handlebars

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
</head>
<body>
    <h2>about.handlebars</h2>
    <hr>
</body>
</html>
```

/views/home.handlebars

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
</head>
<body>
    <h2>home.handlebars</h2>
    <hr>
</body>
</html>
```

추가로 404, 500 에러 핸들러 함수에서도 render() 함수를 사용할 수 있도록 작업해 보자.

- 코드 생략

핸들바 뷰에서 정적 리소스 사용하기

정적 리소스란 서버에서 클라이언트에게 있는 그대로 전송하는 자원을 말한다.

대표적으로 image, css, js, html 등이 있다.

필요한 폴더를 만들고 정적리소스 파일을 준비한다.

/**public**/img/logo.png

/views/layouts/main.handlebars 파일을 수정한다.

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
</head>
<body>
    <!-- public 디렉토리는 클라이언트에게 보이지 않으므로 쓰지 않는다. -->
    <header><img src="/img/logo.png" style="width: 100px; height:auto;"></header>
    {{{body}}}
```

```
</body>
</html>
```

정적리소스를 사용할 수 있도록 app.js 에 설정을 추가한다.

/app.js

```
app.use(express.static(__dirname+'/public'));
```

서버를 재 시작해서 이미지가 잘 보이는지 테스트 한다.

main.handlebars 에 이미지를 배치했으므로 메인 레이아웃이 적용되는 모든 뷰에 이미지가 보이게 된다.

핸들바에 동적 데이터 전달

핸들바 같은 HTML 엔진을 사용하는 궁극적인 이유는 서버에서 동적으로 뷰를 만들어야 하기 때문이다. 서버에서 데이터베이스에 질의하여 동적으로 구한 데이터를 뷰에 전달하여 화면 디자인과 결합시킨다.

여기서는 간단히 테스트 용도로 더미 데이터를 사용하겠다.

app.js

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// 테스트 용 더미 데이터
var fortunes = [
      "good", "very good", "more than good", "not bad", "buy lotto"
];

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

```
var handlebars = require('express-handlebars').create({defaultLayout:'main'});

app.engine('handlebars', handlebars.engine);
app.set('view engine', 'handlebars');

app.set('port', process.env.PORT || 3000);

app.use(express.static(__dirname+'/public'));

app.get('/', function(req, res){
	res.render('home');
});

app.get('/about', function(req, res){
	var randomFortune = fortunes[Math.floor(Math.random()*fortunes.length)];
	// about.handlebars 파일에 데이터를 전달한다.
	res.render('about', {fortune: randomFortune});
});

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// 브라우저가 헤더로 보내는 정보를 확인한다.
app.get('/headers', function(req, res){
	res.set('Content-Type', 'text/plain');
	var s = '';
	for(var name in req.headers){
		s += name + ':' + req.headers[name] + '₩n';
	}
	res.send(s);
});

app.use(function(req, res, next){
	res.status(404);
	res.render('404');
});

app.use(function(err, req, res, next){
	console.error(err.stack);
	res.status(500);
```

```
    res.render('500');
});

app.listen(app.get('port'), function(){
    console.log('express is running on http://localhost:'+app.get('port'));
});
```

render() 함수에 두 번째 파라미터 자리에 데이터 객체를 설정하면 뷰에 전달된다.

/views/about.handlebars

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
</head>
<body>
    <h2>about.handlebars</h2>
    <hr>
    fortune for the day: <b>{{fortune}}</b>
</body>
</html>
```

app.js 에서 사용한 정적리소스를 모듈로 분리

서버 역할을 하는 app.js 는 기능이 계속해서 추가 되어야 할 것이고 그에 따라 복잡해지기 마련이다. 따라서 관리성 증대를 위해 일부 로직을 모듈로 분리하는 연습을 해 보자.

app.js

```
app.get('/about', function(req, res){
    res.render('about', {fortune: require('./lib/fortune').getFortune()});
});
```

/lib/fortune.js

```
var fortuneCookies = [
	"good",
	"very good",
	"more than good",
	"not bad",
	"buy lotto"
];

exports.getFortune = function() {
	var idx = Math.floor(Math.random() * fortuneCookies.length);
	return fortuneCookies[idx];
};
```

멀티 레이아웃 사용하기

한 사이트라고 하더라도 레이아웃을 다양하게 구성해야 할 필요가 있다.

app.js 파일에 다음 코드를 추가 해 보자.

app.js

```
// 기본 레이아웃(main.handlebars)을 적용하지 않는다.
app.get('/no-layout', function(req, res){
	res.render('no-layout', {layout:null});
});

// 기본 레이아웃을 사용하지 않고 다른 커스텀 레이아웃을 적용한다.
app.get('/custom-layout', function(req, res){
	res.render('custom-layout', {layout:'custom'});
});
```

lyaouts 폴더에 커스텀 레이아웃 파일을 만든다.

/views/layouts/custom.handlebars

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
</head>
<body>
    <header>
    <img src="https://www.google.co.kr/images/
        branding/googlelogo/2x/googlelogo_color_272x92dp.png"
        style="width: 200px; height:auto;">
    </header>
    {{{body}}}
</body>
</html>
```

/views/custom-layout.handlebars

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
</head>
<body>
    <h2>custom-layout.handlebars</h2>
    <hr>
    <h3>레이아웃 custom.handlebars 를 적용한다.</h3>
</body>
</html>
```

Section

섹션이란 레이아웃 body 에 배치되는 뷰에서 원하는 데이터를 레이아웃에 전달하여 레이아웃에 적용하는 것이다.

섹션을 사용하기 위해서 helpers 를 설정할 필요가 있다.

/app.js

```
var handlebars = require('express-handlebars').create({
    defaultLayout:'main',
    helpers: {
        section: function(name, options){
            if(!this._sections) {this._sections = {};}
            this._sections[name] = options.fn(this);
            return null;
        }
    }
});

app.get('/jquery-test', function(req, res){
    res.render('jquery-test');
});
```

/views/jquery-test.handlebars

```
{{#section 'head'}}
    <!-- we want Google to ignore this page -->
    <meta name="robots" content="noindex">
{{/section}}

<h1>Test Page</h1>
<p>We're testing some jQuery stuff.</p>

{{#section 'jquery'}}
    <script>
        $('document').ready(function(){
            $('h1').html('jQuery Works');
```

```
        });
    </script>
{{/section}}
```

/views/layouts/main.handlebars

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
    {{{_sections.head}}}
    <style>
        body {
            padding-top: 30px;
            padding-bottom: 20px;
            padding-left: 20px;
            padding-right: 20px;
        }
    </style>
</head>
<body>
    <header><img src="/img/logo.png" style="width: 100px; height:auto;"></header>
    <div class="container">
        {{{body}}}
        <hr>
        <footer>
            <p>&copy; korea travel</p>
        </footer>
    </div>

    <script src="//code.jquery.com/jquery-2.1.1.min.js"></script>
    {{{_sections.jquery}}}
</body>
</html>
```

main.handlerbars 에 선언된 {{{_sections.head}}}와 {{{_sections.jquery}}}는 섹션 선언이 없는 뷰와 랜더링 할 때는 작동하지 않지만 섹션 선언이 있는 jquery-test.handlebars 와 랜더링할 때는 사용된다.

Partial

파셜이란 작은 뷰를 다른 뷰에 인클루드 시키는 것이다.
뷰를 생성하는 작업이 모든 끝난 후 파셜이 오버라이드 된다.

/app.js

```
//mocked weather data
function getWeatherData(){
    return {
        locations: [
            {
                name: 'Portland',
                forecastUrl: 'http://www.wunderground.com/US/OR/Portland.html',
                iconUrl: 'http://icons-ak.wxug.com/i/c/k/cloudy.gif',
                weather: 'Overcast',
                temp: '54.1 F (12.3 C)',
            },
            {
                name: 'Bend',
                forecastUrl: 'http://www.wunderground.com/US/OR/Bend.html',
                iconUrl: 'http://icons-ak.wxug.com/i/c/k/partlycloudy.gif',
                weather: 'Partly Cloudy',
                temp: '55.0 F (12.8 C)',
            },
            {
                name: 'Manzanita',
                forecastUrl: 'http://www.wunderground.com/US/OR/Manzanita.html',
                iconUrl: 'http://icons-ak.wxug.com/i/c/k/rain.gif',
                weather: 'Light Rain',
                temp: '55.0 F (12.8 C)',
            },
        ]
    };
}

app.use(function(req, res, next){
    // 파셜을 모든 뷰에서 쓸 수 있도록 하기 위해서 res.loclas 를 사용한다.
    if(!res.locals.partials) {res.locals.partials = {};}
```

```
    // 각 뷰에서 사용하는 컨텍스트와 충돌하지 않도록 모든 파셜 컨텍스트를
    // partials 객체에 넣는다. 파셜을 뷰에서 {{> foo}} 표기법으로 사용할 수 있다.
    res.locals.partials.foo = 'partial foo';
    res.locals.partials.weatherContext = getWeatherData();
    next();
});
```

/views/partials/weather.handlebars

```
<div class="weatherWidget">
    {{#each partials.weatherContext.locations}}
        <div class="location">
            <h3>{{name}}</h3>
            <a href="{{forecastUrl}}">
                <img src="{{iconUrl}}" alt="{{weather}}"> {{weather}}, {{temp}}
            </a>
        </div>
    {{/each}}
</div>
```

/views/home.handlebars

```
<h1>welcome to korea travel</h1>
<hr>

<!-- app.js 에 선언된 res.locals.partials.foo 가 가리키는 문자열을 사용한다. -->
{{> foo}}
<hr>

{{! app.js 에 선언된 res.locals.partials.weatherContext 가 가리키는
    객체가 갖고 있는 날씨정보와 views/partials/weather.handlerbars 를 사용하여 렌더링한다.}}
{{> weather}}
```

{{> weather}} 설정은 기본적으로 views/partials 폴더 밑에 weather.handlebars 를 찾아서 있으면 대상으로 삼는다.

서버코드에서 res.locals.partials.weatherContext 를 설정하면 weather.handlebars 랜더링 시 weatherContext 가 갖고 있는 정보를 전달하여 사용한다.

{{! 브라우저에게 전송되지 않는 서버사이드 주석을 애용하자 }}
<!-- 브라우저에게 전송되어 소스보기 시 보이는 주석은 삼가하자 -->

Helpers

헬퍼스란 핸들바 표현식에서 사용할 수 있는 함수다.

핸들바 뷰인 home 에게 배열을 전달한다.

```
var names = ['chris', 'aaron', 'david'];

res.render('home', {names: names});
```

/views/home.handlebars

```
{{#each names}}
    {{@index}} : {{this}}<br>
{{/each}}
```

#each 배열

배열이 갖고 있는 데이터만큼 루프를 돈다.

@index

루프 처리 횟수를 가리키는 인덱스다. 0 부터 시작한다.

this

루프 처리 시 해당 데이터를 가리킨다. 위 경우 배열이 갖고 있는 데이터는 문자열이므로 this 는 문자열을 가리킨다.

결과

```
0 : chris
1 : aaron
2 : david
```

넘버링을 1 부터 하고 싶다고 생각할 수 있다.

그런데, 핸들바는 {{@index + 1}} 같은 문법을 지원하지 않는다.

어떻게 하면 가능할까?

해답은 헬퍼스에 함수를 만들고 그 함수가 처리하게 하는 것이다.

/app.js

```
var handlebars = require('express-handlebars').create({
    defaultLayout:'main',
    helpers: {
        section: function(name, options){
            if(!this._sections) {this._sections = {};}
            this._sections[name] = options.fn(this);
            return null;
        },
        inc: function(value){
            return parseInt(value, 10) + 1;
        }
});
```

헬퍼스에 추가한 함수 inc 를 핸들바 표현식에서 사용할 수 있다.

/views/home.handlebars

```
{{#each names}}
    {{inc @index}} : {{this}}<br>
{{/each}}
```

@index 값이 inc 함수에 파라미터로 전달되고 그 결과가 리턴되어 화면 표시된다.

헬퍼스에 원하는 기능의 함수를 추가해서 표현식에서 사용하는 방식은 상당히 매력적이다.

추가로 다음을 살펴보자.

```
res.render('home', {
    data: {by_width:[1, 2, 3], by_height:['a', 'b', 'c']}
});
```

이번에는 data 가 가리키는 데이터를 뷰에서 출력 해 보자.

/views/home.handlebars

```
{{#each data.by_width}}
    {{#each ../data.by_height}}
        {{../this}} x {{this}}<br>
    {{/each}}
{{/each}}
```

결과

```
1 x a
1 x b
1 x c
2 x a
2 x b
2 x c
3 x a
3 x b
3 x c
```

2 중 for 문처럼 동작한다. 핸들바는 JSTL 문법에서 제공하는 begin, end, step 같은 기능을 제공하지 않는다. 그러나, 앞에서 살펴 본 것처럼 2 중 for 문을 돌 수 있으며 헬퍼스 함수를 곁들인다면 충분히 원하는 화면을 구성할 수 있다. 구구단 로직을 작성 해 보자.

```
res.render('home', {
        gugudan:{
            rows:['1','2','3','4','5','6','7','8','9'],
            columns:[1,2,3,4,5,6,7,8,9]}
    });
```

```
helpers : {
    multiply : function (a, b) {
        return parseInt(a, 10) * parseInt(b, 10);
    }
}
```

```
{{#each gugudan.rows}}
    {{#each ../gugudan.columns}}
        {{../this}} * {{this}} = {{multiply ../this this}}<br>
    {{/each}}
    <br>
{{/each}}
```

다양한 구조의 데이터 처리를 추가로 연습하자.

실습 #1

```
var contact = {
    user: {
        contact: {
            email: 'kyssra@gmail.com',
            twitter: 'frozen'
        },
        address: {
            city: 'Seoul',
            state: 'Korea'
        },
        name: 'Chris'
    }
};
```

```
{{#with contact.user}}
    <div>{{name}}</div>

    <div>
```

```
    {{#with contact}}
        Email: {{email}}<br>
        Twitter: {{twitter}}
    {{/with}}
    </div>

    <div>
        Address: {{address.city}}, {{address.state}}
    </div>
{{/with}}
```

실습 #2

```
var fullstack = {mean:[
    {name: 'mongo', desc: 'no-sql databse'},
    {name: 'express', desc: 'web framework'},
    {name: 'angular', desc: 'front-end framework'},
    {name: 'node', desc: 'server javascript'}]
};
```

```
helpers : {
    table : function (data) {
        var str = '<table border="1px">';
        for (var i = 0; i < data.length; i++) {
            str += '<tr>';
            for (var key in data[i]) {
                if (data[i].hasOwnProperty(key)) {
                    str += '<td>' + data[i][key] + '</td>';
                }
            }
            str += '</tr>';
        }
        str += '</table>';
        return str;
    }
}
```

```
{{{table fullstack.mean}}}
```

실습 #3

```
var bands = [];
bands.push({
    Name : "Band",
    Date : "Aug 14th, 2012",
    Albums : [{Name : "Generic Name"},{Name : "Something Else!!"}]
});
bands.push({
    Name : "Other Guys",
    Date : "Jan 22nd, 2013",
    Albums : [{Name : "Album One"}]
});
```

```
{{#if bands}}
    <table border='1px'>
        <tr>
            <th>No</th>
            <th>Band Name</th>
            <th>Date</th>
            <th>Album</th>
        </tr>
        {{#each bands}}
        <tr>
            <td>{{inc @index}}</td>
            <td>{{Name}}</td>
            <td>{{Date}}</td>
            <td>
                {{#each Albums}}
                    {{comma @index}}{{Name}}
                {{/each}}
            </td>
        </tr>
        {{/each}}
    </table>
{{else}}
    <h3>There are no concerts coming up.</h3>
{{/if}}
```

Cookie

쿠키를 맛있게 구워보자. 냠냠냠!

쿠키는 브라우저(클라이언트)와 서버가 상대방을 인식하기 위한 도구다. HTTP 프로토콜에 비연결성을 극복하기 위해 탄생한 기술이다. 브라우저는 서버에 접속할 때 항상 쿠키를 전달한다.

/credentials.js

```
// 쿠키시크릿은 서버만 알고 있는 문자열이다.
// 쿠키를 클라이언트에 보내기 전에 암호화할 때 사용한다.
// 비밀번호가 아니므로 기억할 필요가 없고 무작위 문자열을 써도 된다.
module.exports = {
    cookieSecret: 'your cookie secret goes here'
}
```

/app.js

```
// 쿠키에 서명할 키를 관리하는 객체의 모듈
var credentials = require('./credentials.js');

// 비서명 쿠키 또는 서명된 쿠키를 사용하기 위한 설정
app.use(require('cookie-parser')(credentials.cookieSecret));

app.get('/cookie-test', function(req, res){
    res.render('cookie-test', {'savedValue':req.cookies.keyString});
});

app.get('/cookie-save', function(req, res){
    var value = req.query.keyString || '';
    console.log('value: '+value);

    if (value) {
        // 밀리세컨드 단위다. 1 초는 1000 을 사용하면 된다.
        res.cookie('keyString', value, {maxAge:1000*10});
    }
    res.redirect('/cookie-test');
```

```
});

app.get('/cookie-delete', function(req, res){
    res.clearCookie('keyString');
    res.redirect('/cookie-test');
});
```

/views/cookie-test.handlebars

```
<!doctype html>
<html>
<head>
    <title>cookie</title>
</head>
<body>
    <h2>cookie-test.handlebars</h2>
    <hr>

    keyString: {{savedValue}}
    <hr>

    <a href="/cookie-save?keyString=777">save cookie</a><br>
    <a href="/cookie-delete">delete cookie</a><br>
</body>
</html>
```

쿠키 삭제

res.**clearCookie**(키문자열);

서명된 쿠키 사용법

저장

res.cookie('signed_cookie', 'this is value', **{ signed:true }**);

불러오기

var signedCookie = req.**signedCookies**.signed_cookie;

Session

세션은 고객의 정보를 서버 측 메모리에 유지할 때 사용하는 객체다.
세션은 고객을 구분하기 위해서 쿠키를 기반으로 사용한다. 사용자가 쿠키 사용을 거부하는 경우 브라우저는 그 대안으로 서버가 발급한 세션 아이디를 파라미터로 전송한다.

세션의 기본적인 사용방식은 쿠키에 고유한 식별자만 저장하고 나머지 정보는 모두 서버에 저장한다. 따라서 서버에 정보를 저장할 저장소가 필요하다. 세션은 메모리에만 존재하기 때문에 서버를 재 시작할 때마다 정보가 사라진다. 세션 영속화에는 redis 모듈을 추천한다.

세션의 최대 사용처는 사용자의 인증(로그인) 처리다. 또한 세션은 사용자의 행동방식에 따라 사용자에게 필요한 정보를 유지하는 기법에서도 많이 사용된다.

/app.js

```
var session = require('express-session');
var sessionOptions = {
    secret: "secret",
    resave : true,
    saveUninitialized : false
};
app.use(session(sessionOptions));

app.get('/greeting', function(req, res){
    if (!req.session.views){
        req.session.views = 1;
    }else{
        req.session.views += 1;
    }
    res.render('greeting', {
            message: 'welcome',
            counter: req.session.views
        });
});
```

greeting 으로 접근 할 때 마다 counter 값을 증가시켜 뷰에 counter 값을 전달한다.

Session Options

Session Options	Description
cookie	Options object for the session ID cookie. The default value is { path: '/', httpOnly: true, secure: false, maxAge: null }.
genid	Function to generate the session ID. Default is to use uuid
name	The name of the session ID cookie to set in the response (and read from in the request).
proxy	Trust the reverse proxy when setting secure cookies.
resave	If true forces a session to be saved back to store even if it was not modified in the request. 요청이 바뀌지 않았어도 세션 정보를 강제로 다시 저장한다. **false 권장**
rolling	Forces a cookie to be set on every request.
saveUninitialized	If true it forces a newly created session without any modifications to be saved to the session store. true: 초기화되지 않은 새로운 세션도 저장한다. **false: 권장** 쿠키를 설정하기 전에 사용자의 허락을 받아야 한다면 false 로 설정해야 한다.
secret	It is a required option and is used for signing the session ID cookie. 쿠키에 서명할 때 사용하는 키. **cookie-parser 에 사용하는 키를 써도 무방**하다.
store	Session store instance. Default is to use memory store. 세션이 저장될 인스턴스, 기본값은 MemoryStore 의 인스턴스를 사용한다.
unset	Controls the handling of session object in the store after it is unset. Either delete or keep the session object. Default is to keep the session object

/views/greeting.handlebars

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
</head>
<body>
    <h2>greeting.handlebars</h2>
    <hr>
```

```
    message: {{message}}<br>
    counter: {{counter}}<br>
</body>
</html>
```

POST - REDIRECT - GET

사용자가 전송한 정보를 서버에서 받아서 데이터베이스에 저장하는 흐름을 분석해 보자.

간단한 처리 방식 : 하지만 나쁜!

1. POST: 사용자가 폼에 정보를 입력하고 전송버튼을 눌러서 서버에 전송한다.
2. 바로 처리 결과 통보: 데이터를 저장하고 정보 저장이 성공했음을 사용자에게 알려준다.

위 경우, 브라우저의 특성 상 기 전송한 POST 데이터를 캐시하고 있으므로 화면 갱신 작업 시 데이터를 서버에 재 전송한다. 앞서서 전송하여 이미 처리된 데이터가 또 서버에 전송되게 된다.

일반적으로 위 문제점을 피하기 위해서 다음과 같이 처리한다.

재 전송을 막기위한 처리 방식

1. **POST**: 사용자가 폼에 정보를 입력하고 전송버튼을 눌러서 서버에 전송한다.
2. **REDIRECT**: 데이터 저장 후 브라우저에게 다른 주소를 지정하여 재 접속하라는 REDIRECT 를 요청한다. 다른 경로로 접속하게 되면 브라우저에 캐시 된 정보는 사라진다.
3. **GET**: 브라우저는 REDIRECT 요청을 받고 GET 방식으로 요청받은 주소로 접근한다.

서버에서 처리 결과를 클라이언트에게 통보한다. 이 때 메시지는 flash 처리를 한다.

/app.js

```
var express = require('express');
var app = express();
var path = require('path');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

//------------------------------------------------
// configuration
app.set('port', process.env.PORT || 3000);

var handlebars = require('express-handlebars').create({
    defaultLayout:'main',
    // .handlebars 확장자를 .hbs 로 변경해서 사용한다.
```

```
    extname: '.hbs'
});
app.engine('hbs', handlebars.engine);
app.set('view engine', 'hbs');

//----------------------------------------------------
// middle-ware
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
    extended : false
}));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

//----------------------------------------------------
// router
app.use('/', require('./routes/default'));

//----------------------------------------------------
// error handler
app.use(function(req, res, next) {
    var err = new Error('Not Found');
    err.status = 404;
    next(err);
});

app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.type('text/plain');
    res.send(err);
});

//----------------------------------------------------
// server start
app.listen(app.get('port'), function(){
    console.log('running on http://localhost:'+app.get('port'));
});
```

/routes/default.js

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res, next) {
	res.render('home', {
		title : 'Express'
	});
});

router.get('/newsletter', function(req, res) {
	res.render('newsletter', {
		csrf : 'CSRF token goes here'
	});
});

router.post('/process', function(req, res) {
	// 접근경로 뒤에 붙어서 오는 GET 방식 파라미터 구하기
	var action = req.query.action;
	if(action){
		action = action.toLowerCase();
	}
	console.log(action);

	var next = '/';
	switch (action) {
	case 'newsletter':
		// POST 방식으로 전송된 파라미터 구하기
		console.log(req.body._csrf);
		console.log(req.body.name);
		console.log(req.body.email);
		next = '/thank-you';
		break;
	default:
		break;
	}
```

```
    // 301 : 영구적인 리다이렉션으로 브라우저가 캐시할 수 있다.
    // 일시적 리다이렉션을 의미하는 303 을 쓰자.
    res.redirect(303, next);
});

router.get('/thank-you', function(req, res) {
    res.render('thank-you');
});

module.exports = router;
```

/views/layouts/main.hbs

```
<!doctype html>
<html>
<head>
    <title>what's up?</title>
</head>
<body>
    {{{body}}}
</body>
</html>
```

/views/home.hbs

```
<!doctype html>
<html>
<head>
    <title>home</title>
</head>
<body>

    {{#if title}}
        <h2>{{title}}</h2>
    {{else}}
        <h2>home.hbs</h2>
    {{/if}}
```

```
    <hr>

    <a href="newsletter">regist for newsletter</a>
</body>
</html>
```

/views/newsletter.hbs

```
<h2>sign up for our newsletter</h2>
<form action="/process?action=newsletter" method="post">
    <input type="hidden" name="_csrf" value="{{csrf}}">
    <div>
        <label>name</label>
        <div>
            <input type="text" name="name">
        </div>
    </div>
    <div>
        <label>email</label>
        <div>
            <input type="email" name="email" required>
        </div>
    </div>
    <div>
        <div>
            <button type="submit">register</button>
        </div>
    </div>
</form>
```

/views/thank-you.hbs

```
<h2>Thank You!</h2>
<p>
    We appreciate your business.<br>
    Please <a href="#">contact us</a> if you have any questions or concerns.
</p>
```

Flash

사용자가 전달한 정보의 처리가 끝나면 그 결과를 사용자에게 알려 줄 필요가 있다.
이 때 결과 메시지는 한 번만 전달하는 것이 좋으므로 플래쉬를 사용한다.

/app.js

```
// ## session
app.use(require('express-session')({
	resave: false,
	saveUninitialized: false,
	secret: 'secret'
}));

// ## flash
// 한 번만 뷰에  데이터를 전달한다.
app.use(function(req, res, next){
	res.locals.flash = req.session.flash;
	delete req.session.flash;
	next();
});
```

/routes/default.js

```
router.post('/process', function(req, res) {
	var action = req.query.action;
	if(action){
		action = action.toLowerCase();
	}
	console.log(action);

	var next = '/';
	switch (action) {
	case 'newsletter':
		console.log(req.body._csrf);
		console.log(req.body.name);
		console.log(req.body.email);
```

```
            // ## flash message
            req.session.flash = {
                        type: 'success',
                        intro: 'Thank you!',
                        message: 'You have now been signed up for the newsletter service.',
            };

            next = '/thank-you';
            break;
      default:
            break;
      }

      res.redirect(303, next);
});
```

/views/thank-you.hbs

```
<h2>Thank You!</h2>
<p>
      We appreciate your business.<br>
      Please <a href="#">contact us</a> if you have any questions or concerns.
</p>

{{#if flash}}
      {{#section 'jquery'}}
            <script>
                  $('document').ready(function(){
                        alert('{{flash.message}}');
                  });
            </script>
      {{/section}}
{{/if}}
```

/views/layouts/main.hbs

```
<!doctype html>
<html>
<head>
    <title>what's up?</title>
</head>
<body>
    {{{body}}}

    <script src="//code.jquery.com/jquery-2.1.1.min.js"></script>
    {{{_sections.jquery}}}
</body>
</html>
```

AJAX

사용자의 요청을 비동기 방식으로 처리하면 사용자에게는 화면처리의 부드러운 느낌을 서버에는 뷰 랜더링 작업의 부담을 덜 수 있다. 웹 개발은 서서히 AJAX 처리를 늘리는 방향으로 발전하고 있다.

/views/newsletter.handlebars

```
<div class="formContainer">
    <form class="form-horizontal newsletterForm" role="form"
            action="process?action=newsletter" method="POST">
        <input type="hidden" name="_csrf" value="{{csrf}}">
        <div class="form-group">
            <label for="fieldName" class="col-sm-2 control-label">Name</label>
            <div class="col-sm-4">
                <input type="text" class="form-control"
                id="fieldName" name="name">
            </div>
        </div>
        <div class="form-group">
            <label for="fieldEmail" class="col-sm-2 control-label">Email</label>
            <div class="col-sm-4">
                <input type="email" class="form-control" required
                    id="fieldEmail" name="email">
            </div>
        </div>
        <div class="form-group">
            <div class="col-sm-offset-2 col-sm-4">
                <button type="submit" class="btn btn-default">Register</button>
            </div>
        </div>
    </form>
</div>

{{#section 'jquery'}}
    <script>
        $(document).ready(function(){
            $('.newsletterForm').on('submit', function(evt){
```

```
                evt.preventDefault(); // 이벤트의 전파를 막는다.
                var action = $(this).attr('action');
                var $container = $(this).closest('.formContainer');
                $.ajax({
                    url: action,
                    type: 'POST',
            data: $(this).serialize(),
                    success: function(data){
                        if(data.success){
                            $container.html('<h2>Thank you!</h2>');
                        } else {
                            $container.html('There was a problem.');
                        }
                    },
                    error: function(){
                        $container.html('There was a problem.');
                    }
                });
            });
        });
    </script>
{{/section}}
```

파일 업로드 with formidable

formidable 은 간단한 파일 업로드 처리 시 인기있는 모듈이다.

/app.js

```
var formidable = require('formidable');

app.get('/contest/vacation-photo', function(req, res){
	var now = new Date();
	res.render('contest/vacation-photo', { year: now.getFullYear(), month: now.getMonth() });
});

app.post('/contest/vacation-photo/:year/:month', function(req, res){
	var form = new formidable.IncomingForm();

	form.parse(req, function(err, fields, files){
		if(err) {return res.redirect(303, '/error');}

		console.log('received fields:');
		console.log(fields);
		console.log('received files:');
		console.log(files);

		res.redirect(303, '/thank-you');
	});
});
```

/views/contest/vacation-photo.handlebars

```
<form class="form-horizontal" role="form"
		enctype="multipart/form-data" method="POST"
		action="/contest/vacation-photo/{{year}}/{{month}}">

	<div class="form-group">
		<label for="fieldName" class="col-sm-2 control-label">Name</label>
		<div class="col-sm-4">
```

```
            <input type="text" class="form-control" id="fieldName" name="name">
        </div>
    </div>

    <div class="form-group">
        <label for="fieldEmail" class="col-sm-2 control-label">Email</label>
        <div class="col-sm-4">
            <input type="email" class="form-control" required
                id="fieldEmail" name="email">
        </div>
    </div>

    <div class="form-group">
        <label for="fieldPhoto" class="col-sm-2 control-label">Vacation photo</label>
        <div class="col-sm-4">
            <input type="file" class="form-control" required accept="image/*"
                id="fieldPhoto" data-url="/upload" name="photo">
        </div>
    </div>

    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-4">
            <button type="submit" class="btn btn-primary">Submit</button>
        </div>
    </div>

</form>
```

파라미터 정보의 유효성 테스트

때때로 사용자는 악의적이다. 사용자가 작성한 정보가 해당 정보 작성법에 맞게 입력되었는지 테스트 할 필요가 있다. 브라우저에서 서버로 보내기 전 클라이언트 측의 자바스크립트에서 최대한 유효성 테스트를 해서 쓸모 없는 정보가 서버로 가지 않도록 조치하는 것이 좋다. 서버에서 진행하는 유효성 테스트는 2 차 저지선으로 생각하자.

/app.js

```
//for now, we're mocking NewsletterSignup:
function NewsletterSignup(){}

NewsletterSignup.prototype.save = function(cb){
    cb();
};

var VALID_EMAIL_REGEX = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:₩.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)+$/;

app.get('/newsletter', function(req, res){
    res.render('newsletter', { csrf: 'CSRF token goes here' });
});

app.post('/newsletter', function(req, res){
    var name = req.body.name || '', email = req.body.email || '';

    // 입력정보 유효성 검사
    if(!email.match(VALID_EMAIL_REGEX)) {
        // AJAX 요청인 경우 req.xhr 값은 true 다.
        if(req.xhr) return res.json({ error: 'Invalid name email address.' });

        req.session.flash = {
            type: 'danger',
            intro: 'Validation error!',
            message: 'The email address you entered was not valid.',
        };
```

```
        return res.redirect(303, '/newsletter/archive');
    }

    // 진짜 로직 구현은 직접 해 봅니다.
    new NewsletterSignup({ name: name, email: email }).save(function(err){

        if(err) {
            if(req.xhr) return res.json({ error: 'Database error.' });
            req.session.flash = {
                type: 'danger',
                intro: 'Database error!',
                message: 'There was a database error; please try again later.',
            };
            return res.redirect(303, '/newsletter/archive');
        }

        // AJAX 요청도 처리
        if(req.xhr) return res.json({ success: true });

        req.session.flash = {
            type: 'success',
            intro: 'Thank you!',
            message: 'You have now been signed up for the newsletter.',
        };

        return res.redirect(303, '/newsletter/archive');
    });
});

app.get('/newsletter/archive', function(req, res){
    res.render('newsletter/archive');
});
```

/views/newsletter/archive.handlebars

```
<h2>Past Newsletters</h2>
```

Middle-Ware

익스프레스는 서버 측의 처리 로직에서 중간 중간 처리해야 하는 추가 로직을 미들웨어라 부른다. app.use() 함수로 미들웨어를 설정할 수 있다. 콜백 함수의 파라미터로 받을 수 있는 next 로 다음 로직이 기동할 것인지를 결정할 수 있다.

/app.js

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

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//next() 호출이 없으면 처리함수 다음 함수는 기동하지 않는다.

app.get('/favicon.ico', function(req, res){
    res.end();
});

// 라우팅 정보가 없는 함수는 미들웨어다.
app.use(function(req, res, next){
    console.log('1~~~~~~~~~~~~~~~~');
    // 다음 함수를 기동시킨다.
    next();
});

// 라우팅 정보와 요청방식이 일치하는 경우에만 기동한다.
app.get('/a', function(req, res, next){
    console.log('a - 1');
    next();
});

app.get('/a', function(req, res){
    console.log('a - 2');
    // 응답 처리는 여기서 끝난다.
    res.send('a - 2');
});
```

```
app.use(function(req, res, next){
    console.log('2~~~~~~~~~~~~~~~');
    next();
});

app.get('/b', function(req, res, next){
    console.log('b - 1');
    res.send('b - 1');
    next();
});

app.get('/b', function(req, res){
    console.log('b - 2');

    // Can't set headers after they are sent.
    // 앞서서 send() 함수를 사용했으므로 다시 사용할 수 없다.
    // res.send('b - 2');
});

app.use(function(req, res, next){
    console.log('3~~~~~~~~~~~~~~~');
    next();
});

app.get('/c', function(req, res, next){
    console.log('c - 1');
    throw new Error('c - 1 failed');
});

// 앞선 /c 처리 함수에서 에러가 발생한 경우만 잡아서 처리하는 에러 핸들러가 된다.
app.use('/c', function(err, req, res, next){
    console.log('c - 2');
    console.log('err.message: '+err.message);

    // 에러를 잡아서 사용하고 다음으로 넘겨주지 않으면(자신이 이벤트를 소비)
    // 폴백 에러 핸들러는 기동하지 않는다.
    // 대신 404 핸들러가 처리한다.
    // next();
```

```
    // 에러를 넘겨주면 폴백 에러 핸들러가 기동한다.
    next(err);
});

// 앞서서 에러가 있다면 일반로직 함수는 기동하지 않는다.
app.use(function(req, res, next){
    console.log('4~~~~~~~~~~~~~~~~');
    next();
});

// 404 핸들러
app.use(function(req, res){
    console.log('404 not found');
    res.send('404 not found');
});

// fall-back 에러 핸들러
app.use(function(err, req, res, next){
    console.log('fallback error handler: '+err.message);
    res.send('500 server error');
});

var server = app.listen(8080, function() {
    var host = server.address().address;
    var port = server.address().port;
    console.log("sever is running on http://%s:%s", host, port);
});
```

Express with JSON

JSON 포맷의 문자열을 취급하는 파일을 데이터베이스 용도로 사용 해 보자.

/public/index.html

```
<html>
<body>
    <h2>index.html</h2>
    <hr>

    <form action="user" method="POST">
        <div>name: <input type="text" name="name"></div>
        <div>password: <input type="text" name="password"></div>
        <div>profession: <input type="text" name="profession"></div>
        <div><input type="submit" value="send"></div>
    </form>
</body>
</html>
```

/lib/users.json

```
[
    {
        "id": 1,
        "name": "chris",
        "password": "1111",
        "profession": "teacher"
    },
    {
        "id": 2,
        "name": "david",
        "password": "1111",
        "profession": "librarian"
    },
    {
        "id": 3,
```

```
        "name": "aaron",
        "password": "1111",
        "profession": "clerk"
    }
]
```

/app.js

```
var express = require('express');
var app = express();
var fs = require("fs");
var bodyParser = require('body-parser');

app.use(express.static('public'));
var urlencodedParser = bodyParser.urlencoded({ extended: false });

String.prototype.escapeSpecialChars = function() {
    return this.replace(/₩₩n/g, "")
        .replace(/₩₩t/g, "");
};

app.get('/', function(req, res) {
    res.sendFile('/public/index.html');
});

app.get('/user', function(req, res) {
    fs.readFile(__dirname + "/lib/users.json", 'utf8', function(err, data) {
        console.log(data);

        res.end('{"users":'+data.escapeSpecialChars()+'}');
    });
});

app.post('/user', urlencodedParser, function(req, res) {
    var user = {
            id:0,
            name:req.body.name,
            password:req.body.password,
```

```
                profession:req.body.profession
        };
        console.log(user);

        addUser(user, res);
});

var server = app.listen(8080, function() {
        var host = server.address().address;
        var port = server.address().port;
        console.log("server is running on http://%s:%s", host, port);
});

function addUser(user, res){
        getMaxId(function(id, users){
                user.id = id;
                users.push(user);
                console.log('~~~~~~~~~~~after added~~~~~~~~~~~');
                console.log(users);

                fs.writeFile(__dirname+'/lib/users.json', JSON.stringify(users), function(error){
                        if (error) {
                                return console.error(error.message);
                        }

                        res.redirect('user');
                });
        });
}

function getMaxId(cb){
        fs.readFile(__dirname + "/lib/users.json", 'utf8', function(err, data) {
                var users = JSON.parse(data);
                console.log('~~~~~~~~~~~current~~~~~~~~~~~');
                console.log(users);

                var maxId = 0;
                for (var i = 0; i < users.length; i++) {
```

```
                if (maxId < users[i].id) {
                    maxId = users[i].id;
                }
            }
            maxId = maxId + 1;

            cb(maxId, users);
        });
}
```

Express with MongoDB

1. 새 프로젝트를 생성

File > New > Node.js Express Project

2. jade 대신 handlebars 를 사용하도록 설정을 변경

예제코드에서 파일명을 명시하지 않으면 서버 역할을 수행하는 app.js 파일을 대상으로 작업한다.

```
var handlebars = require('express-handlebars').create({
    defaultLayout:'main',
    helpers: {
        section: function(name, options){
            if(!this._sections) {this._sections = {};}
            this._sections[name] = options.fn(this);
            return null;
        }
    }
});
app.engine('handlebars', handlebars.engine);
app.set('view engine', 'handlebars');

// 다음 코드를 삭제한다.
//app.set('views', path.join(__dirname, 'views'));
//app.set('view engine', 'jade');

app.get('/', function(req, res){
    res.render('index');
});

// 다음 코드를 삭제한다.
//app.use('/', routes);
//app.use('/users', users);
```

```
app.listen(3000, function(){
    console.log('server is running on http://localhost:3000');
});

// 다음 코드를 삭제한다.
//module.exports = app;
```

3. 메인 레이아웃 뷰

/views/layouts/main.handlebars

```
<!doctype html>
<html>
<head>
    <title>korea travel</title>
    <link rel="stylesheet" href="/vendor/bootstrap/css/bootstrap.min.css">
    <style>
        body {
            padding-top: 30px;
            padding-bottom: 20px;
            padding-left: 20px;
            padding-right: 20px;
        }
    </style>
    <link rel="stylesheet" href="/vendor/bootstrap/css/bootstrap-theme.min.css">
    <script src="/vendor/js/modernizr-2.6.2-respond-1.1.0.min.js"></script>
</head>
<body>
    <header><img src="/img/logo.png" style="width: 100px; height:auto;"></header>
    <div class="container">
        {{#if flash}}
            <div class="alert alert-dismissible alert-{{flash.type}}">
                <button type="button" class="close"
                    data-dismiss="alert" aria-hidden="true">&times;</button>
                <strong>{{flash.intro}}</strong> {{{flash.message}}}
            </div>
```

```
        {{/if}}

        {{{body}}}

        <hr>
        <footer>
            <p>&copy; korea travel</p>
        </footer>
    </div>

    <script src="//code.jquery.com/jquery-2.1.1.min.js"></script>
    {{{_sections.jquery}}}
    <script src="/vendor/bootstrap/js/bootstrap.min.js"></script>
</body>
</html>
```

4. 정적리소스

main.handlebars 뷰에서 사용하는 css, js 파일들을 준비하여 /public 폴더를 만들고 그 밑으로 경로에 맞게 폴더를 만들고 리소스를 배치한다.

5. 첫 페이지 뷰

앵커태그 연동 처리는 12 번에서 살펴본다.

/views/index.handlebars

```
<h1>index.handlebars</h1>
<hr>

{{title}}

<h2>Check out our featured tour packages:</h2>
<ul>
    <li><a href="/vacation/hood-river-day-trip">Hood River Day Trip</a></li>
    <li><a href="/vacation/oregon-coast-getaway">Oregon Coast Getaway</a></li>
```

```
    <li><a href="/vacation/rock-climbing-in-bend">Rock Climbing in Bend</a></li>
</ul>
<br><br>

<a href="/vacations">Available Packages Information Specifically</a>

{{#if flash}}
    {{#section 'jquery'}}
        <script>
            $('document').ready(function(){
                alert('{{flash.message}}');
            });
        </script>
    {{/section}}
{{/if}}
```

TEST: 서버를 시작하고 뷰가 보이는지 테스트 한다.

6. 플래쉬 메시지

다음 미들웨서는 세션에 저장한 플래쉬 정보를 사용 후 세션에서 삭제하는 기능이다.

```
app.use(function(req, res, next){
	res.locals.flash = req.session.flash;
	delete req.session.flash;
	next();
});
```

7. 기동 설정값

시작 시 콘솔모드에서 값을 주지 않으면 오른쪽 값이 디폴트 값으로 사용된다.

```
app.set('port', process.env.PORT || 3000);
app.set('mode', process.env.MODE || 'development');
```

8. 세션

```
var credentials = require('./credentials');

app.use(require('express-session')({
	resave: false,
	saveUninitialized: false,
	secret: credentials.cookieSecret,
}));
```

/credentials.js

```
module.exports = {
	cookieSecret: 'your cookie secret goes here',
	mongo: {
		development: {
			connectionString: 'mongodb://scott:tiger@127.0.0.1:27017/mydb'
		},
```

```
        producction: {
            connectionString: 'mongodb://scott:tiger@127.0.0.1:27017/mydb'
        }
    }
};
```

9. 몽구스

```
var mongoose = require('mongoose');

// 웹사이트처럼 오래 실행되는 앱에서 데이터베이스 연결 에러를 막기위해 설정한다.
var options = {
    server: {
        socketOptions: { keepAlive: 1 }
    }
};

switch (app.get('mode')) {
case 'development':
    mongoose.connect(credentials.mongo.development.connectionString, options, function(err){
        if(err){
            return console.log('there was a problem connecting to the database! '+err);
        }
        console.log('database connected!');
    });
    break;
case 'production':
    mongoose.connect(credentials.mongo.production.connectionString, options);
    break;
default:
    throw new Error('unknown execution enviroment: '+app.get('mode'));
}
```

10. 스키마

```
var Vacation = require('./models/vacation.js');
```

/models/vacation.js

```
var mongoose = require('mongoose');

var vacationSchema = mongoose.Schema({
	name: String,
	slug: String,
	category: String,
	sku: String,
	description: String,
	priceInCents: Number,
	tags: [String],
	inSeason: Boolean,
	available: Boolean,
	requiresWaiver: Boolean,
	maximumGuests: Number,
	notes: String,
	packagesSold: Number,
});

// 가격을 표시하게 좋게 바꾸는 메소드다.
// model() 함수 사용전에 정의한다.
vacationSchema.methods.getDisplayPrice = function(){
	return '$' + (this.priceInCents / 100).toFixed(2);
};

var Vacation = mongoose.model('Vacation', vacationSchema);

module.exports = Vacation;
```

11. 테스트 더미 데이터

```
// 테스트를 위한 더미 데이터 입력하기
Vacation.find(function(err, vacations){
    if(err) return console.error(err);
    // 데이터가 존재하면 입력하지 않는다.
    if(vacations.length) return;

    new Vacation({
        name: 'Hood River Day Trip',
        slug: 'hood-river-day-trip',
        category: 'Day Trip',
        sku: 'HR199',
        description: 'Spend a day sailing on the Columbia and ' +
        'enjoying craft beers in Hood River!',
        priceInCents: 9995,
        tags: ['day trip', 'hood river', 'sailing', 'windsurfing', 'breweries'],
        inSeason: true,
        maximumGuests: 16,
        available: true,
        packagesSold: 0,
    }).save();

    new Vacation({
        name: 'Oregon Coast Getaway',
        slug: 'oregon-coast-getaway',
        category: 'Weekend Getaway',
        sku: 'OC39',
        description: 'Enjoy the ocean air and quaint coastal towns!',
        priceInCents: 269995,
        tags: ['weekend getaway', 'oregon coast', 'beachcombing'],
        inSeason: false,
        maximumGuests: 8,
        available: true,
        packagesSold: 0,
    }).save();
```

```
    new Vacation({
        name: 'Rock Climbing in Bend',
        slug: 'rock-climbing-in-bend',
        category: 'Adventure',
        sku: 'B99',
        description: 'Experience the thrill of rock climbing in the high desert.',
        priceInCents: 289995,
        tags: ['weekend getaway', 'bend', 'high desert', 'rock climbing', 'hiking', 'skiing'],
        inSeason: true,
        requiresWaiver: true,
        maximumGuests: 4,
        available: false,
        packagesSold: 0,
        notes: 'The tour guide is currently recovering from a skiing accident.',
    }).save();
});
```

12. /vacation/:vacation GET

```
// 첫 페이지(index) > 상품 클릭
app.get('/vacation/:vacation', function(req, res, next){
    Vacation.findOne({ slug: req.params.vacation }, function(err, vacation){
        if(err) return next(err);
        if(!vacation) return next();
        res.render('vacation', { vacation: vacation });
    });
});
```

13. vacation

/views/vacation.handlebars

```
<h1>{{vacation.name}}</h1>
```

```
<form class="form-horizontal newsletterForm" role="form" action="/cart/add" method="POST">
    <input type="hidden" name="sku" value="{{vacation.sku}}">
    <div class="form-group">
        <label for="fieldGuests" class="col-sm-4 control-label">Number of guests in your party</label>
        <div class="col-sm-1">
            <input type="number" class="form-control" id="fieldGuests" name="guests">
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-4">
            <button type="submit" class="btn btn-default">Add to Cart</button>
        </div>
    </div>
</form>
```

14. /cart/add POST

```
//첫 페이지(index) > 상품 클릭(vacation) > 인원수 입력, Add to Cart 클릭
app.post('/cart/add', function(req, res, next){
    var cart = req.session.cart || (req.session.cart = { items: [] });

    Vacation.findOne({ sku: req.body.sku }, function(err, vacation){
        if(err) return next(err);
        if(!vacation) return next(new Error('Unknown vacation SKU: ' + req.body.sku));
        cart.items.push({
            vacation: vacation,
            guests: req.body.guests || 1,
        });

        res.redirect(303, '/cart');
    });
});
```

15. /cart GET

```
//첫 페이지(index) > 상품 클릭(vacation) > 인원수 입력, Add to Cart 클릭 > 카트 조회
app.get('/cart', function(req, res, next){
	var cart = req.session.cart;
	if(!cart) next();

	res.render('cart', { cart: cart });
});
```

16. cart

```
<h1>Your Shopping Cart</h1>
{{#each cart.errors}}
	<div class="alert alert-danger">{{.}}</div>
{{/each}}
{{#each cart.warnings}}
	<div class="alert alert-warning">{{.}}</div>
{{/each}}

<table class="table table-striped">
	<thead>
		<tr>
			<td>Vacation</td>
			<td>Guests</td>
		</tr>
	</thead>
	<tbody>
		{{#each cart.items}}
			<tr>
				<td>{{vacation.name}}</td>
				<td>{{guests}}</td>
			</tr>
		{{/each}}
	</tbody>
```

```
</table>

<a class="btn btn-primary" href="/cart/checkout">Checkout</a>
```

17. /cart/checkout GET

```
// 카트 조회(cart) > Checkout 클릭
app.get('/cart/checkout', function(req, res, next){
    var cart = req.session.cart;
    if(!cart) next();

    res.render('cart-checkout');
});
```

18. cart-checkout

/views/cart-checkout.handlebars

```
<h1>Your Shopping Cart</h1>
<p>Please provide your email address so we can send you a confirmation of your order.</p>
<form class="form-horizontal newsletterForm" role="form" action="/cart/checkout"
method="POST">
    <div class="form-group">
        <label for="fieldName" class="col-sm-2 control-label">Name</label>
        <div class="col-sm-4">
            <input type="text" class="form-control" required
                id="fieldName" name="name">
        </div>
    </div>
    <div class="form-group">
        <label for="fieldEmail" class="col-sm-2 control-label">Email</label>
        <div class="col-sm-4">
            <input type="email" class="form-control" required
                id="fieldEmail" name="email">
```

```
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-4">
            <button type="submit" class="btn btn-default">Checkout</button>
        </div>
    </div>
</form>
```

19. /cart/checkout POST

```
var VALID_EMAIL_REGEX = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:₩.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)+$/;

// 카트 조회(cart) > Checkout 클릭(cart-checkout) > 이름, 이메일 작성, Checkout 클릭
app.post('/cart/checkout', function(req, res){
    var cart = req.session.cart;
    if(!cart) next(new Error('Cart does not exist.'));

    var name = req.body.name || '', email = req.body.email || '';
    // input validation
    if(!email.match(VALID_EMAIL_REGEX)) return res.next(new Error('Invalid email address.'));

    // assign a random cart ID; normally we would use a database ID here
    cart.number = Math.random().toString().replace(/^0₩.0*/, '');
    cart.billing = {
        name: name,
        email: email,
    };

    /*
     * send email to client
     */
    console.log('~~~~~~~~~~~~~~~send email to client~~~~~~~~~~~~~~~');
    console.log(JSON.stringify(cart));
    console.log('~~~~~~~~~~~~~~~send email to client~~~~~~~~~~~~~~~');
```

```
    res.render('cart-thank-you', { cart: cart });
});
```

20. cart-thank-you

/views/cart-thank-you.handlebars

```
<p>Thank you for booking your tirp with Korea Travel, {{cart.billing.name}}!</p>
<p>Your reservation number is {{cart.number}}, <br>
and an email has been sent to {{cart.billing.email}} for your records.</p>
```

TEST: 여기서 잠시 작업을 멈추고 지금까지 한 작업을 테스트 한다.

21. /vacations GET

첫 화면의 하단에 있는 /vacations 앵커를 클릭했을 때 필요한 연동처리를 진행한다.

```
app.get('/vacations', function(req, res){
    Vacation.find({ available: true }, function(err, vacations){

        // 데이터베이스에서 반환한 데이터를 뷰에서 사용 할 정보만 넘긴다.
        var context = {
            vacations: vacations.map(function(vacation){
                return {
                    sku: vacation.sku,
                    name: vacation.name,
                    description: vacation.description,
                    inSeason: vacation.inSeason,
                    price: vacation.getDisplayPrice(),
                    qty: vacation.qty,
                };
            })
```

```
        };

        res.render('vacations', context);
    });
});
```

22. vacations

/views/vacations.handlebars

```
{{! 상품 페이지 }}
<h1>Vacations</h1>
{{#each vacations}}
    <div class="vacation">
        <h3>{{name}}</h3>
        <p>{{description}}</p>
        {{#if inSeason}}
            <span class="price">{{price}}</span>
            <a href="/cart/add?sku={{sku}}" class="btn btn-default">Buy Now!</a>
        {{else}}
            <span class="outOfSeason">We're sorry, this vacation is currently
            not in season.
            {{! The "notify me when this vacation is in season"
                page will be our next task. }}
            <a href="/notify-me-when-in-season?sku={{sku}}">Notify me when
            this vacation is in season.</a>
        {{/if}}
    </div>
{{/each}}
```

23. /cart/add GET

```
app.get('/cart/add', function(req, res, next){
    var cart = req.session.cart || (req.session.cart = { items: [] });
```

```
    Vacation.findOne({ sku: req.query.sku }, function(err, vacation){
        if(err) return next(err);
        if(!vacation) return next(new Error('Unknown vacation SKU: ' + req.query.sku));

        cart.items.push({
            vacation: vacation,
            guests: req.body.guests || 1,
        });

        res.redirect(303, '/cart');
    });
});
```

TEST: 작업 내용을 테스트 한다.

24. /notify-me-when-in-season GET

```
app.get('/notify-me-when-in-season', function(req, res){
    res.render('notify-me-when-in-season', { sku: req.query.sku });
});
```

25. notify-me-when-in-season

/views/notify-me-when-in-season.handlebars

```
<div class="formContainer">
    <form class="form-horizontal newsletterForm" role="form"
        action="/notify-me-when-in-season" method="POST">
        <input type="hidden" name="sku" value="{{sku}}">
        <div class="form-group">
            <label for="fieldEmail" class="col-sm-2 control-label">Email</label>
```

```
                <div class="col-sm-4">
                    <input type="email" class="form-control" required
                        id="fieldName" name="email">
                </div>
            </div>
            <div class="form-group">
                <div class="col-sm-offset-2 col-sm-4">
                    <button type="submit" class="btn btn-default">Submit</button>
                </div>
            </div>
        </form>
</div>
```

26. /notify-me-when-in-season POST

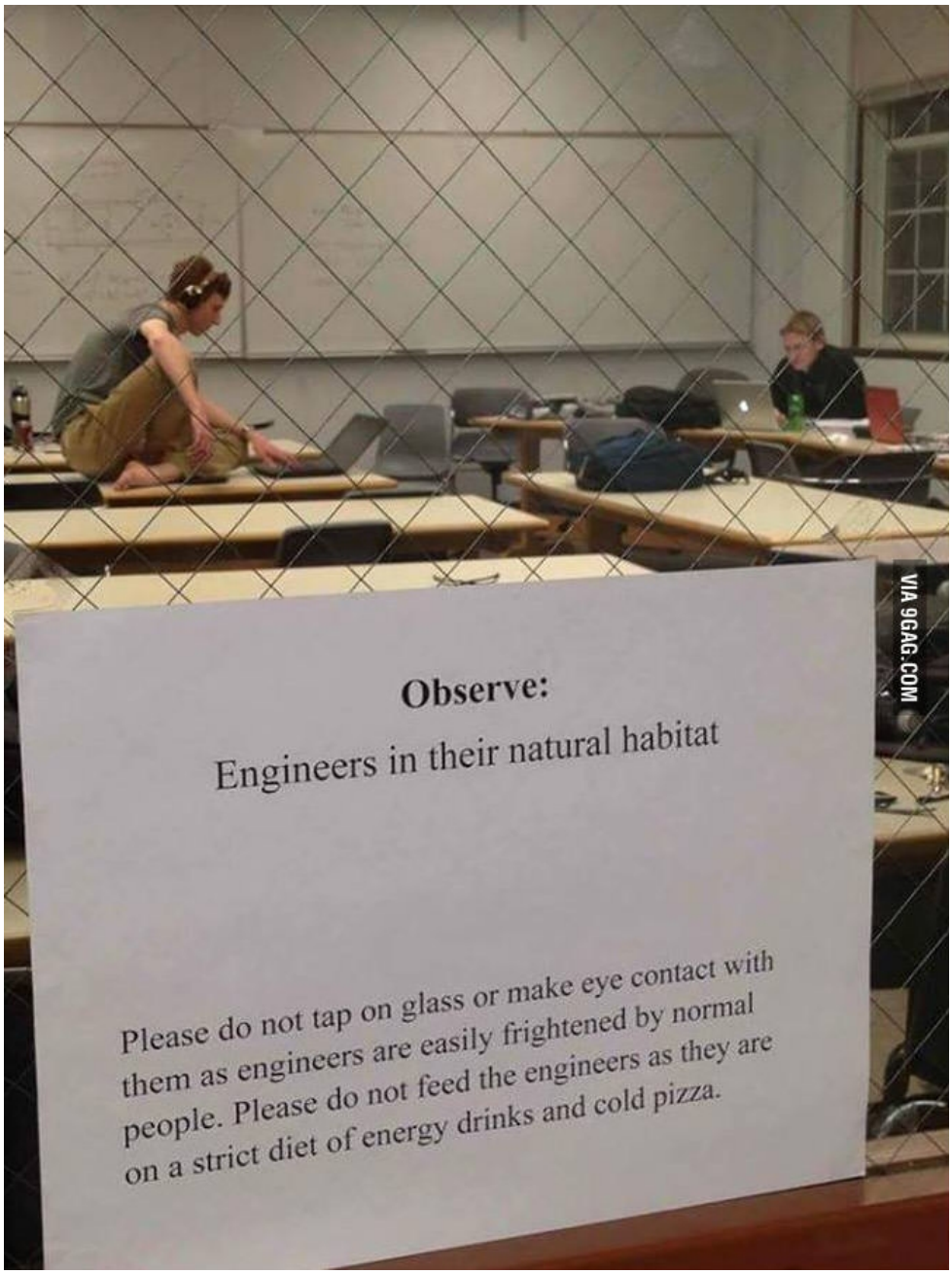
홈(/)으로 리다이렉트 하면서 플래쉬 메시지를 설정했으므로 홈 뷰에서 사용자에게 플래쉬 메시지가 전달될 것이다.

```
app.post('/notify-me-when-in-season', function(req, res){

    /*
     * 이메일을 저장했다가 휴가철이 돌아오면 메일발송으로 알려준다.
     */
    req.session.flash = {
            type: 'success',
            intro: 'Thank you!',
            message: 'You have now been signed up for the notification service.',
        };

    res.redirect('/');
});
```

수고하셨습니다.


Observe:
Engineers in their natural habitat
Please do not tap on glass or make eye contact with them as engineers are easily frightened by normal people. Please do not feed the engineers as they are on a strict diet of energy drinks and cold pizza.
VIA 9GAG.COM