Node.js 교과서 개정2판

길벗

6장

6.1 익스프레스 프로젝트 시작하기

6.2 자주 사용하는 미들웨어

6.3 Router 객체로 라우팅 분리하기

6.4 req, res 객체 살펴보기

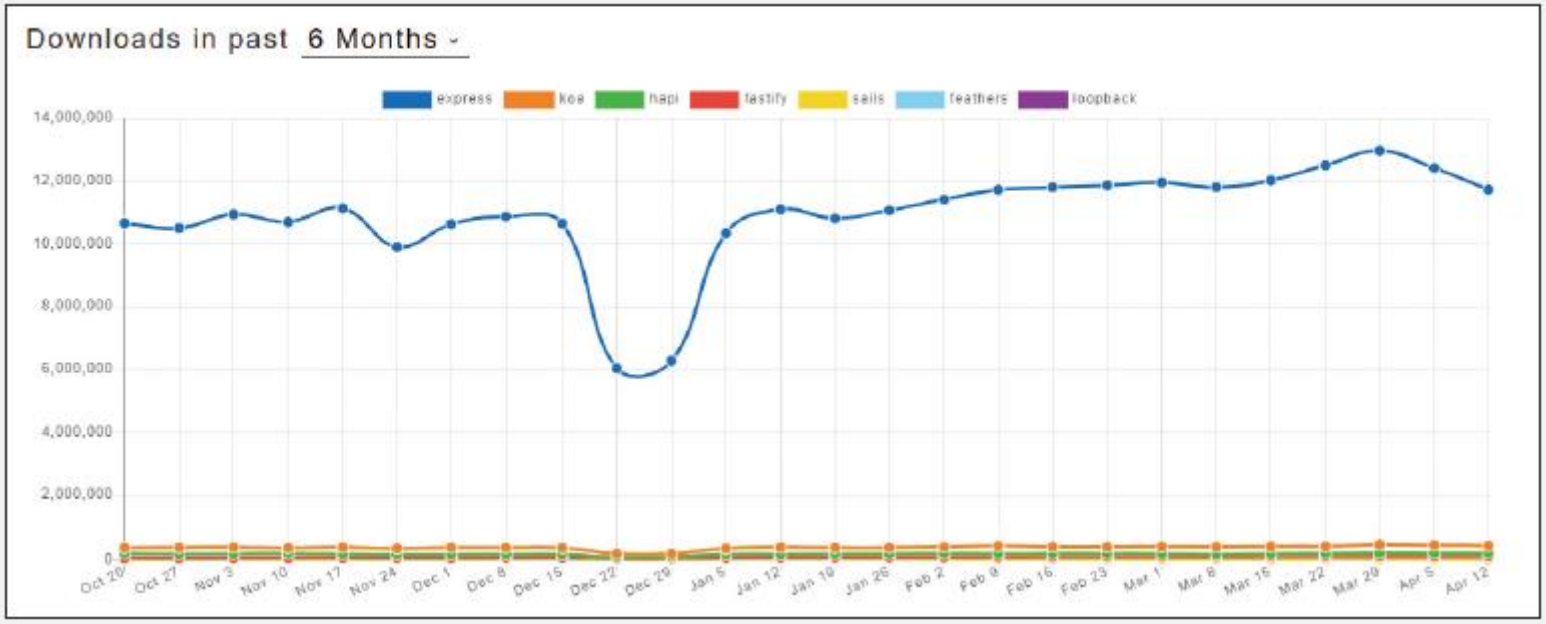
6.5 템플릿 엔진 사용하기

6.1 익스프레스 프로젝트 시작하기

1. Express 소개

- http 모듈로 웹 서버를 만들 때 코드가 보기 좋지 않고, 확장성도 떨어짐
  - 프레임워크로 해결
  - 대표적인 것이 Express(익스프레스), Koa(코아), Hapi(하피)
  - 코드 관리도 용이하고 편의성이 많이 높아짐

그림 6-1 express, koa, hapi 다운로드 수 비교



2. package.json 만들기

- 직접 만들거나 npm init 명령어 생성
  - nodemon이 소스 코드 변경 시 서버를 재시작해줌

package.json

```
{
  "name": "learn-express",
  "version": "0.0.1",
  "description": "익스프레스를 배우자",
  "main": "app.js",
  "scripts": {
    "start": "nodemon app"
  },
  "author": "ZeroCho",
  "license": "MIT"
}
```

콘솔

```
$ npm i express
$ npm i -D nodemon
```

3. app.js 작성하기

» 서버 구동의 핵심이 되는 파일

- app.set('port', 포트)로 서버가 실행될 포트 지정
- app.get('주소', 라우터)로 GET 요청이 올 때 어떤 동작을 할지 지정
- app.listen('포트', 콜백)으로 몇 번 포트에서 서버를 실행할지 지정

app.js

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

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

app.get('/', (req, res) => {
  res.send('Hello, Express');
});

app.listen(app.get('port'), () => {
  console.log(app.get('port'), '번 포트에서 대기 중');
});
```

4. 서버 실행하기

- app.js: 핵심 서버 스크립트
- bin/www: 서버를 실행하는 스크립트
- public: 외부에서 접근 가능한 파일들 모아둠
- views: 템플릿 파일을 모아둠
- routes: 서버의 라우터와 로직을 모아둠
  - 추후에 models를 만들어 데이터베이스 사용

5. 익스프레스 서버 실행하기

» npm start(package.json의 start 스크립트) 콘솔에서 실행

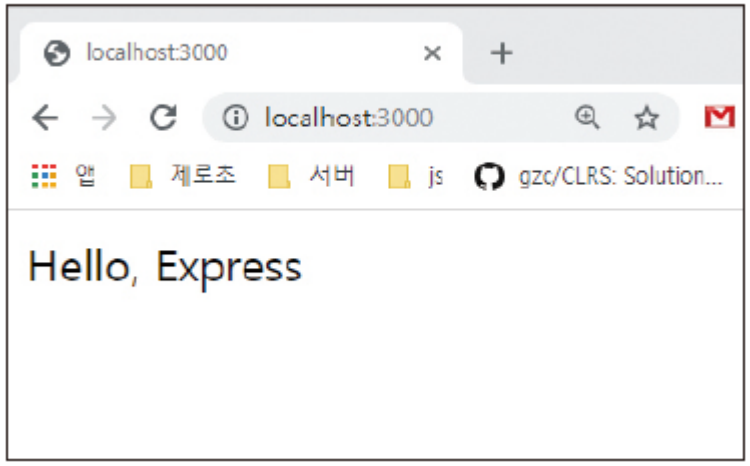
콘솔

```
$ npm start
> learn-express@0.0.1 start C:\Users\zerocho\learn-express
> nodemon app

[nodemon] 2.0.3
```

» localhost:3000

그림 6-2 localhost:3000 접속 화면



6. HTML 서빙하기

» res.sendFile로 HTML 서빙 가능

index.html

```html
<html>
<head>
  <meta charset="UTF-8" />
  <title>익스프레스 서버</title>
</head>
<body>
  <h1>익스프레스</h1>
  <p>배워봅시다.</p>
</body>
</html>
```

app.js

```js
const express = require('express');
const path = require('path');

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

app.get('/', (req, res) => {
  // res.send('Hello, Express');
  res.sendFile(path.join(__dirname, '/index.html'));
});

app.listen(app.get('port'), () => {
  console.log(app.get('port'), '번 포트에서 대기 중');
});
```

6.2 자주 사용하는 미들웨어

1. 미들웨어

익스프레스는 미들웨어로 구성됨

- 요청과 응답의 중간에 위치하여 미들웨어
- app.use(미들웨어)로 장착
- 위에서 아래로 순서대로 실행됨.
- 미들웨어는 req, res, next가 매개변수인 함수
- req: 요청, res: 응답 조작 가능
- next()로 다음 미들웨어로 넘어감.

표 6-1 미들웨어가 실행되는 경우

app.use(미들웨어)	모든 요청에서 미들웨어 실행
app.use('/abc', 미들웨어)	abc로 시작하는 요청에서 미들웨어 실행
app.post('/abc', 미들웨어)	abc로 시작하는 POST 요청에서 미들웨어 실행

app.js

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

app.use((req, res, next) => {
  console.log('모든 요청에 다 실행됩니다.');
  next();
});
app.get('/', (req, res, next) => {
  console.log('GET / 요청에서만 실행됩니다.');
  next();
}, (req, res) => {
  throw new Error('에러는 에러 처리 미들웨어로 갑니다.')
});

app.use((err, req, res, next) => {
  console.error(err);
  res.status(500).send(err.message);
});

app.listen(app.get('port'), () => {
...
```

2. 에러 처리 미들웨어

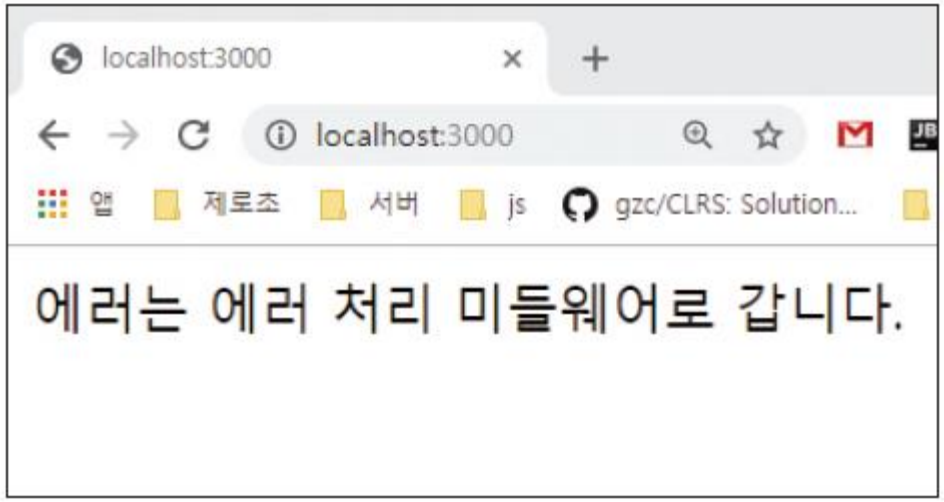
» 에러가 발생하면 에러 처리 미들웨어로

- err, req, rs, next까지 매개변수가 4개
- 첫 번째 err에는 에러가 관한 정보가 담김
- res.status 메서드로 HTTP 상태 코드를 지정 가능(기본값 200)
- 에러 처리 미들웨어를 안 연결해도 익스프레스가 에러를 알아서 처리해주긴 함.
- 특별한 경우가 아니면 가장 아래에 위치하도록 함.

콘솔

```
모든 요청에 다 실행됩니다.
GET / 요청에서만 실행됩니다.
Error: 에러는 에러 처리 미들웨어로 갑니다.
...
```

그림 6-4 localhost:3000 접속 화면



3. 자주 쓰는 미들웨어

- morgan, cookie-parser, express-session 설치
  - app.use로 장착
  - 내부에서 알아서 next를 호출해서 다음 미들웨어로 넘어감
  - dotenv는 다음 장에 설명

콘솔

```
$ npm i morgan cookie-parser express-session dotenv
```

.env

```
COOKIE_SECRET=cookiesecret
```

app.js

```
const express = require('express');
const morgan = require('morgan');
const cookieParser = require('cookie-parser');
const session = require('express-session');
const dotenv = require('dotenv');
const path = require('path');

dotenv.config();
const app = express();
app.set('port', process.env.PORT || 3000);

app.use(morgan('dev'));
app.use('/', express.static(path.join(__dirname, 'public')));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser(process.env.COOKIE_SECRET));
app.use(session({
  resave: false,
  saveUninitialized: false,
  secret: process.env.COOKIE_SECRET,
  cookie: {
    httpOnly: true,
    secure: false,
  },
  name: 'session-cookie',
}));

app.use((req, res, next) => {
  console.log('모든 요청에 다 실행됩니다.');
  next();
});
...
```

4. dotenv

- .env 파일을 읽어서 process.env로 만듦
  - dot(점) + env
  - Process.env.COOKIE_SECRET에 cookiesecret 값이 할당됨(키=값 형식)
  - 비밀 키들을 소스 코드에 그대로 적어두면 소스 코드가 유출되었을 때 비밀 키도 같이 유출됨
  - .env 파일에 비밀 키들을 모아두고 .env 파일만 잘 관리하면 됨

5. morgan

» 서버로 들어온 요청과 응답을 기록해주는 미들웨어

- 로그의 자세한 정도 선택 가능(dev, tiny, short, common, combined)

콘솔

```
3000 번 포트에서 대기 중
모든 요청에 다 실행됩니다.
GET / 요청에서만 실행됩니다.
Error: 에러는 에러 처리 미들웨어로 갑니다.
// 에러 스택 트레이스 생략
GET / 500 7.409 ms - 50
```

```
app.use(morgan('dev'));
```

- 예시) GET / 200 51.267 ms – 1539
- 순서대로 HTTP요청 요청주소 상태코드 응답속도 – 응답바이트
- 개발환경에서는 dev, 배포환경에서는 combined를 애용함.

» 더 자세한 로그를 위해 winston 패키지 사용(15장에서)

6. static

- 정적인 파일들을 제공하는 미들웨어
  - 인수로 정적 파일의 경로를 제공
  - 파일이 있을 때 fs.readFile로 직접 읽을 필요 없음
  - 요청하는 파일이 없으면 알아서 next를 호출해 다음 미들웨어로 넘어감
  - 파일을 발견했다면 다음 미들웨어는 실행되지 않음

```
app.use('요청 경로', express.static('실제 경로'));

app.use('/', express.static(path.join(__dirname, 'public')));
```

- 컨텐츠 요청 주소와 실제 컨텐츠의 경로를 다르게 만들 수 있음
  - 요청 주소 localhost:3000/stylesheets/style.css
  - 실제 컨텐츠 경로 /public/stylesheets/style.css
  - 서버의 구조를 파악하기 어려워져서 보안에 도움이 됨

9. body-parser

- 요청의 본문을 해석해주는 미들웨어
  - 폼 데이터나 AJAX 요청의 데이터 처리
  - json 미들웨어는 요청 본문이 json인 경우 해석, urlencoded 미들웨어는 폼 요청 해석
  - [illegible]게서 온 데이터를 넣어줌

```
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
```

- 콘솔

```
$ npm i body-parser
```

body-parser를 직접 설치해야 함

```
const bodyParser = require('body-parser');
app.use(bodyParser.raw());
app.use(bodyParser.text());
```

- Multipart 데이터(이미지, 동영상 등)인 경우는 다른 미들웨어를 사용해야 함
  - multer 패키지(9장에서)

10. cookie-parser

- 요청 헤더의 쿠키를 해석해주는 미들웨어
  - parseCookies 함수와 기능 비슷
  - req.cookies 안에 쿠키들이 들어있음

app.js

```
app.use(cookieParser(비밀키));
```

  - 비밀 키로 쿠키 뒤에 서명을 붙여 내 서버가 만든 쿠키임을 검증할 수 있음
- 실제 쿠키 옵션들을 넣을 수 있음
  - expires, domain, httpOnly, maxAge, path, secure, sameSite 등
  - 지울 때는 clearCookie로(expires와 maxAge를 제외한 옵션들이 일치해야 함)

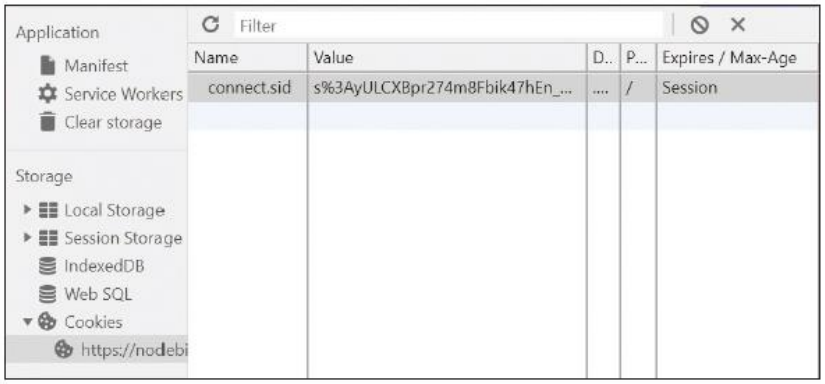
```
res.cookie('name', 'zerocho', {
  expires: new Date(Date.now() + 900000),
  httpOnly: true,
  secure: true,
});
res.clearCookie('name', 'zerocho', { httpOnly: true, secure: true });
```

11. express-session

세션 관리용 미들웨어

```
app.use(session({
  resave: false,
  saveUninitialized: false,
  secret: process.env.COOKIE_SECRET,
  cookie: {
    httpOnly: true,
    secure: false,
  },
  name: 'session-cookie',
}));
```

```
req.session.name = 'zerocho'; // 세션 등록
req.sessionID; // 세션 아이디 확인
req.session.destroy(); // 세션 모두 제거
```



- 세션 쿠키에 대한 설정(secret: 쿠키 암호화, cookie: 세션 쿠키 옵션)
- 세션 쿠키는 앞에 s%3A가 붙은 후 암호화되어 프런트에 전송됨

- resave: 요청이 왔을 때 세션에 수정사항이 생기지 않아도 다시 저장할지 여부
- saveUninitialized: 세션에 저장할 내역이 없더라도 세션을 저장할지
- req.session.save로 수동 저장도 가능하지만 할 일 거의 없음

12. 미들웨어의 특성

» req, res, next를 매개변수로 가지는 함수

```
app.use((req, res, next) => {
  console.log('모든 요청에 다 실행됩니다.');
  next();
});
```

» 익스프레스 미들웨어들도 다음과 같이 축약 가능

- 순서가 중요
- static 미들웨어에서 파일을 찾으면 next를 호출 안 하므로 json, urlencoded, cookieParser는 실행되지 않음

```
app.use(
  morgan('dev'),
  express.static('/', path.join(__dirname, 'public')),
  express.json(),
  express.urlencoded({ extended: false }),
  cookieParser(process.env.COOKIE_SECRET),
);
```

13. next

- next를 호출해야 다음 코드로 넘어감
  - next를 주석 처리하면 응답이 전송되지 않음
  - 다음 미들웨어(라우터 미들웨어)로 넘어가지 않기 때문
  - next에 인수로 값을 넣으면 에러 핸들러로 넘어감('route'인 경우 다음 라우터로)

그림 6-6 next의 동작

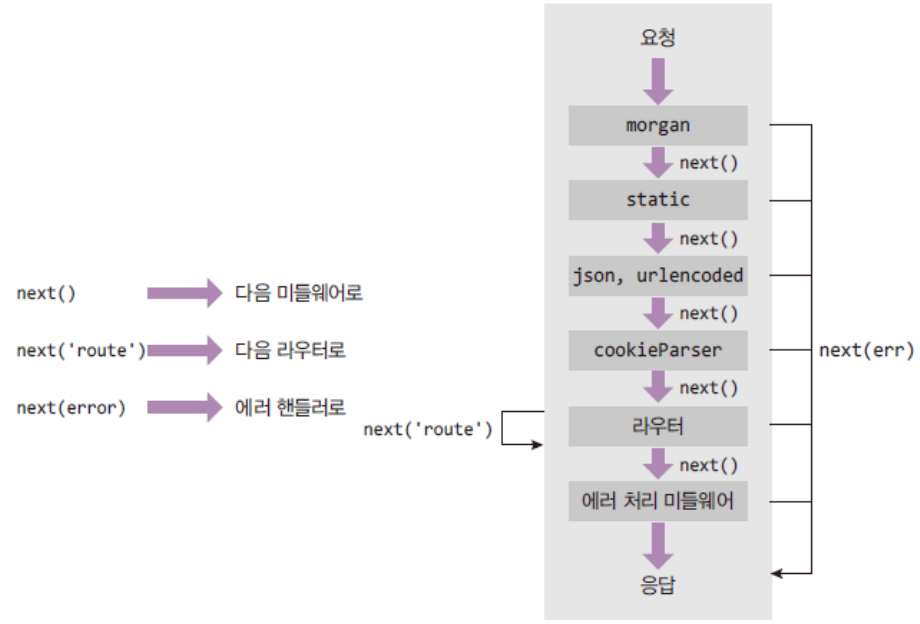


그림 6-7 에러 처리 미들웨어로 에러 보내기

```
next(err)

(err, req, res, next) => { }
```

14. 미들웨어간 데이터 전달하기

- req나 res 객체 안에 값을 넣어 데이터 전달 가능
  - app.set과의 차이점: app.set은 서버 내내 유지, req, res는 요청 하나 동안만 유지
  - req.body나 req.cookies같은 미들웨어의 데이터와 겹치지 않게 조심

```
app.use((req, res, next) => {
  req.data = '데이터 넣기';
  next();
}, (req, res, next) => {
  console.log(req.data); // 데이터 받기
  next();
});
```

15. 미들웨어 확장하기

» 미들웨어 안에 미들웨어를 넣는 방법

- 아래 두 코드는 동일한 역할

```
app.use(morgan('dev'));
// 또는
app.use((req, res, next) => {
  morgan('dev')(req, res, next);
});
```

- 아래처럼 다양하게 활용 가능

```
app.use((req, res, next) => {
  if (process.env.NODE_ENV === 'production') {
    morgan('combined')(req, res, next);
  } else {
    morgan('dev')(req, res, next);
  }
});
```

16. 멀티파트 데이터 형식

- form 태그의 enctype이 multipart/form-data인 경우
  - body-parser로는 요청 본문을 해석할 수 없음
  - multer 패키지 필요

콘솔

```
$ npm i multer
```

multipart.html

```
<form action="/upload" method="post" enctype="multipart/form-data">
  <input type="file" name="image" />
  <input type="text" name="title" />
  <button type="submit">업로드</button>
</form>
```

▼ Form Data view parsed

```
------WebKitFormBoundaryOa6rH3D3NJ1cNo85
Content-Disposition: form-data; name="image"; filename="퇴사.jpg"
Content-Type: image/jpeg


------WebKitFormBoundaryOa6rH3D3NJ1cNo85
Content-Disposition: form-data; name="title"

제목
------WebKitFormBoundaryOa6rH3D3NJ1cNo85--
```

17. multer 설정하기

multer 함수를 호출

- storage는 저장할 공간에 대한 정보
- diskStorage는 하드디스크에 업로드 파일을 저장한다는 것
- destination은 저장할 경로
- filename은 저장할 파일명(파일명+날짜+확장자 형식)
- Limits는 파일 개수나 파일 사이즈를 제한할 수 있음.

```
const multer = require('multer');

const upload = multer({
  storage: multer.diskStorage({
    destination(req, file, done) {
      done(null, 'uploads/');
    },
    filename(req, file, done) {
      const ext = path.extname(file.originalname);
      done(null, path.basename(file.originalname, ext) + Date.now() + ext);
    },
  }),
  limits: { fileSize: 5 * 1024 * 1024 },
});
```

- 실제 서버 운영 시에는 서버 디스크 대신에 S3같은 스토리지 서비스에 저장하는 게 좋음
  - Storage 설정만 바꿔주면 됨

18. multer 미들웨어들

- single과 none, array, fields 미들웨어 존재
  - single은 하나의 파일을 업로드할 때, none은 파일은 업로드하지 않을 때
  - req.file 안에 업로드 정보 저장

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

```
app.post('/upload',  upload.none(),  (req, res) => {
  console.log(req.body);
  res.send('ok');
});
```

```
{
  fieldname: 'img',
  originalname: 'nodejs.png',
  encoding: '7bit',
  mimetype: 'image/png',
  destination: 'uploads/',
  filename: 'nodejs1514197844339.png',
  path: 'uploads\\nodejs1514197844339.png',
  size: 53357
}
```

  - array와 fields는 여러 개의 파일을 업로드 할 때 사용
  - array는 하나의 요청 body 이름 아래 여러 파일이 있는 경우
  - fields는 여러 개의 요청 body 이름 아래 파일이 하나씩 있는 경우
  - 두 경우 모두 업로드된 이미지 정보가 req.files 아래에 존재

```
app.post('/upload', upload.array('many'), (req, res) => {
  console.log(req.files, req.body);
  res.send('ok');
});
```

```
app.post('/upload',
  upload.fields([{ name: 'image1' }, { name: 'image2' }]),
  (req, res) => {
    console.log(req.files, req.body);
    res.send('ok');
  },
);
```

6.3 Router 객체로 라우터 분리하기

1. express.Router

- app.js가 길어지는 것을 막을 수 있음
  - userRouter의 get은 /user와 /가 합쳐져서 GET /user/가 됨

routes/index.js

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

const router = express.Router();

// GET / 라우터
router.get('/', (req, res) => {
  res.send('Hello, Express');
});

module.exports = router;
```

routes/user.js

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

const router = express.Router();

// GET /user 라우터
router.get('/', (req, res) => {
  res.send('Hello, User');
});

module.exports = router;
```

app.js

```
...
const path = require('path');

dotenv.config();
const indexRouter = require('./routes');
const userRouter = require('./routes/user');
...
  name: 'session-cookie',
}));

app.use('/', indexRouter);
app.use('/user', userRouter);

app.use((req, res, next) => {
  res.status(404).send('Not Found');
});

app.use((err, req, res, next) => {
...
```

2. 라우트 매개변수

» :id를 넣으면 req.params.id로 받을 수 있음

- 동적으로 변하는 부분을 라우트 매개변수로 만듦

```
router.get('/user/:id', function(req, res) {
  console.log(req.params, req.query);
});
```

- 일반 라우터보다 뒤에 위치해야 함

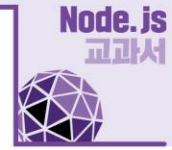
```
router.get('/user/:id', function(req, res) {
  console.log('얘만 실행됩니다.');
});
router.get('/user/like', function(req, res) {
  console.log('전혀 실행되지 않습니다.');
});
```

- /users/123?limit=5&skip=10 주소 요청인 경우

콘솔

```
{ id: '123' } { limit: '5', skip: '10' }
```

3. 404 미들웨어



» 요청과 일치하는 라우터가 없는 경우를 대비해 404 라우터를 만들기

```
app.use((req, res, next) => {
  res.status(404).send('Not Found');
});
```

- 이게 없으면 단순히 Cannot GET 주소 라는 문자열이 표시됨

4. 라우터 그룹화하기

» 주소는 같지만 메서드가 다른 코드가 있을 때

```
router.get('/abc', (req, res) => {
  res.send('GET /abc');
});
router.post('/abc', (req, res) => {
  res.send('POST /abc');
});
```

» router.route로 묶음

```
router.route('/abc')
  .get((req, res) => {
    res.send('GET /abc');
  })
  .post((req, res) => {
    res.send('POST /abc');
  });
```

6.4 req, res 객체 살펴보기

1. req

- **req.app**: req 객체를 통해 app 객체에 접근할 수 있습니다. req.app.get('port')와 같은 식으로 사용할 수 있습니다.
- **req.body**: body-parser 미들웨어가 만드는 요청의 본문을 해석한 객체입니다.
- **req.cookies**: cookie-parser 미들웨어가 만드는 요청의 쿠키를 해석한 객체입니다.
- **req.ip**: 요청의 ip 주소가 담겨 있습니다.
- **req.params**: 라우트 매개변수에 대한 정보가 담긴 객체입니다.
- **req.query**: 쿼리스트링에 대한 정보가 담긴 객체입니다.
- **req.signedCookies**: 서명된 쿠키들은 req.cookies 대신 여기에 담겨 있습니다.
- **req.get(헤더 이름)**: 헤더의 값을 가져오고 싶을 때 사용하는 메서드입니다

2. res

- **res.app**: req.app처럼 res 객체를 통해 app 객체에 접근할 수 있습니다.
- **res.cookie(키, 값, 옵션)**: 쿠키를 설정하는 메서드입니다.
- **res.clearCookie(키, 값, 옵션)**: 쿠키를 제거하는 메서드입니다.
- **res.end()**: 데이터 없이 응답을 보냅니다.
- **res.json(JSON)**: JSON 형식의 응답을 보냅니다.
- **res.redirect(주소)**: 리다이렉트할 주소와 함께 응답을 보냅니다.
- **res.render(뷰, 데이터)**: 다음 절에서 다룰 템플릿 엔진을 렌더링해서 응답할 때 사용하는 메서드입니다.
- **res.send(데이터)**: 데이터와 함께 응답을 보냅니다. 데이터는 문자열일 수도 있고H TML일 수도 있으며, 버퍼일 수도 있고 객체나 배열일 수도 있습니다.
- **res.sendFile(경로)**: 경로에 위치한 파일을 응답합니다.
- **res.set(헤더, 값)**: 응답의 헤더를 설정합니다.
- **res.status(코드)**: 응답 시의 HTTP 상태 코드를 지정합니다.

3. 기타

메서드 체이닝을 지원함

```
res
  .status(201)
  .cookie('test', 'test')
  .redirect('/admin');
```

응답은 한 번만 보내야 함

Warning 응답을 여러 번 보내는 경우

하나의 요청에 대한 응답은 한 번만 보내야 합니다. 두 번 이상 보내면 다음과 같은 에러가 발생합니다.

콘솔

```
Error: Can't set headers after they are sent.
    at validateHeader (_http_outgoing.js:489:11)
    at ServerResponse.setHeader (_http_outgoing.js:496:3)
...
```

이와 같은 에러를 보았다면 라우터에서 res 객체의 응답 메서드가 두 번 이상 사용되지 않았는지 점검해보아야 합니다.

6.5 템플릿 엔진 사용하기

1. 템플릿 엔진

- HTML의 정적인 단점을 개선
  - 반복문, 조건문, 변수 등을 사용할 수 있음
  - 동적인 페이지 작성 가능
  - PHP, JSP와 유사함

2. Pug(구 Jade)

- 문법이 Ruby와 비슷해 코드 양이 많이 줄어듦
  - HTML과 많이 달라 호불호가 갈림
  - 익스프레스에 app.set으로 퍼그 연결

그림 6-11 퍼그 로고

app.js

```
...
app.set('port', process.env.PORT || 3000);
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

app.use(morgan('dev'));
...
```

3. Pug – HTML 표현

퍼그	HTML

doctype html
html
  head
  title= title
  link(rel='stylesheet', href='/stylesheets/style.css')
	<!DOCTYPE html>
	<html>
	  <head>
	  <title>익스프레스</title>
	  <link rel="stylesheet" href="/style.css" />
	  </head>
	</html>

퍼그	HTML
#login-button .post-image span#highlight p.hidden.full	

퍼그	HTML
p Welcome to Express button(type='submit') 전송	Welcome to Express <button type="submit">전송</button>

4. Pug – HTML 표현

퍼그	HTML

p
  | 안녕하세요.
  | 여러 줄을 입력합니다.
  br
  | 태그도 중간에 넣을 수 있습니다.
	
	  안녕하세요. 여러 줄을 입력합니다.
	
	  태그도 중간에 넣을 수 있습니다.

퍼그	HTML

style.
  h1 {
  font-size: 30px;
  }
script.
  const message = 'Pug';
  alert(message);
	<style>
	  h1 {
	  font-size: 30px;
	  }
	</style>
	<script>
	  const message = 'Pug';
	  alert(message);
	</script>

5. Pug - 변수

» res.render에서 두 번째 인수 객체에 Pug 변수를 넣음

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

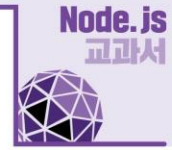
- res.locals 객체에 넣는 것도 가능(미들웨어간 공유됨)

```
router.get('/', function(req, res, next) {
  res.locals.title = 'Express';
  res.render('index');
});
```

- =이나 #{}으로 변수 렌더링 가능(= 뒤에는 자바스크립트 문법 사용 가능)

퍼그	HTML
h1= title p Welcome to #{title} button(class=title, type='submit') 전송 input(placeholder=title + ' 연습')	Express Welcome to Express <button class="Express" type="submit">전송</button> <input placeholder="Express 연습" />

6. Pug – 파일 내 변수



» 퍼그 파일 안에서 변수 선언 가능

- - 뒤에 자바스크립트 사용

퍼그	HTML
- const node = 'Node.js' - const js = 'Javascript' p # {node}와 # {js}	Node.js와 Javascript

- 변수 값을 이스케이프 하지 않을 수도 있음(자동 이스케이프)

퍼그	HTML
p= '이스케이프' p!= '이스케이프하지 않음'	이스케이프 이스케이프하지 않음

7. Pug – 반복문

» for in이나 each in으로 반복문 돌릴 수 있음

퍼그	HTML

ul
  each fruit in ['사과', '배', '오렌지', '바나나', '복숭아']
  li= fruit
	
	   - 사과
	   - 배
	   - 오렌지
	   - 바나나
	   - 복숭아

- 값과 인덱스 가져올 수 있음

퍼그	HTML

ul
  each fruit, index in ['사과', '배', '오렌지', '바나나', '복숭아']
  li= (index + 1) + '번째 ' + fruit
	
	    1번째 사과
	    2번째 배
	    3번째 오렌지
	    4번째 바나나
	    5번째 복숭아

8. Pug – 조건문

» if else if else문, case when문 사용 가능

퍼그	HTML

if isLoggedIn
  div 로그인 되었습니다.
else
  div 로그인이 필요합니다.
	<!-- isLoggedIn이 true일 때 -->
	 로그인 되었습니다.
	<!-- isLoggedIn이 false일 때 -->
	 로그인이 필요합니다.

퍼그	HTML

case fruit
  when 'apple'
  p 사과입니다.
  when 'banana'
  p 바나나입니다.
  when 'orange'
  p 오렌지입니다.
  default
  p 사과도 바나나도 오렌지도 아닙니다.
	<!-- fruit이 apple일 때 -->
	 사과입니다.
	<!-- fruit이 banana일 때 -->
	 바나나입니다.
	<!-- fruit이 orange일 때 -->
	 오렌지입니다.
	<!-- 기본값 -->
	 사과도 바나나도 오렌지도 아닙니다.

9. Pug – include

» 퍼그 파일에 다른 퍼그 파일을 넣을 수 있음

- 헤더, 푸터, 내비게이션 등의 공통 부분을 따로 관리할 수 있어 편리
- include로 파일 경로 지정

header.pug

```
header
  a(href='/') Home
  a(href='/about') About
```

footer.pug

```
footer
  div 푸터입니다
```

main.pug

```
include header
main
  h1 메인 파일
  p 다른 파일을 include할 수 있습니다.
include footer
```

HTML

```
<header>
  <a href="/">Home</a>
  <a href="/about">About</a>
</header>
<main>
  <h1>메인 파일</h1>
  <p>다른 파일을 include할 수 있습니다.</p>
</main>
<footer>
  <div>푸터입니다.</div>
</footer>
```

10. Pug – extends와 block

레이아웃을 정할 수 있음

- 공통되는 레이아웃을 따로 관리할 수 있어 좋음, include와도 같이 사용

퍼그

layout.pug

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/style.
css')
    block style
  body
    header 헤더입니다.
    block content
    footer 푸터입니다.
    block script
```

body.pug

```
extends layout

block content
  main
    p 내용입니다.

block script
  script(src="/main.js")
```

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Express</title>
    <link rel="stylesheet" href="/style.css"
/>
  </head>
  <body>
    <header>헤더입니다.</header>
    <main>
      <p>내용입니다.</p>
    </main>
    <footer>푸터입니다.</footer>
    <script src="/main.js"></script>
  </body>
</html>
```

11. 넌적스

» Pug의 문법에 적응되지 않는다면 넌적스를 사용하면 좋음

- Pug를 지우고 Nunjucks 설치
- 확장자는 html 또는 njk(view engine을 njk로)

콘솔

```
$ npm i nunjucks
```

view engine을 퍼그 대신 넌적스로 교체합니다.

app.js

```
...
const path = require('path');
const nunjucks = require('nunjucks');

dotenv.config();
const indexRouter = require('./routes');
const userRouter = require('./routes/user');

const app = express();
app.set('port', process.env.PORT || 3000);
app.set('view engine', 'html');

nunjucks.configure('views', {
  express: app,
  watch: true,
});

app.use(morgan('dev'));
...
```

12. 넌적스 - 변수

{{변수}}

넌적스

```
<h1>{{title}}</h1>
<p>Welcome to {{title}}</p>
<button class="{{title}}" type="submit">전송</button>
<input placeholder="{{title}} 연습" />
```

내부 변수 선언 가능 {%set 자바스크립트 구문 }

넌적스	HTML
{% set node = 'Node.js' %} {% set js = 'Javascript' %} {{node}}와 {{js}}	Node.js와 Javascript

넌적스	HTML
{{'이스케이프'}} {{'이스케이프하지 않음' | safe }}	이스케이프 이스케이프하지 않음

13. 넌적스 - 반복문

» {% %} 안에 for in 작성(인덱스는 loop 키워드)

넌적스	HTML
 {% set fruits = ['사과', '배', '오렌지', '바나나', '복숭아'] %} {% for item in fruits %} {{item}} {% endfor %} 	 - 사과 - 배 - 오렌지 - 바나나 - 복숭아

넌적스	HTML
 {% set fruits = ['사과', '배', '오렌지', '바나나', '복숭아'] %} {% for item in fruits %} {{loop.index}}번째 {{item}} {% endfor %} 	 1번째 사과 2번째 배 3번째 오렌지 4번째 바나나 5번째 복숭아

14. 넌적스 - 조건문

» {% if %} 안에 조건문 작성

넌적스	HTML
{% if isLoggedIn %} 로그인 되었습니다. {% else %} 로그인이 필요합니다. {% endif %}	<!-- isLoggedIn이 true일 때 --> 로그인 되었습니다. <!-- isLoggedIn이 false일 때 --> 로그인이 필요합니다.

넌적스	HTML
{% if fruit === 'apple' %} 사과입니다. {% elif fruit === 'banana' %} 바나나입니다. {% elif fruit === 'orange' %} 오렌지입니다. {% else %} 사과도 바나나도 오렌지도 아닙니다. {% endif %}	<!-- fruit이 apple일 때 --> 사과입니다. <!-- fruit이 banana일 때 --> 바나나입니다. <!-- fruit이 orange일 때 --> 오렌지입니다. <!-- 기본값 --> 사과도 바나나도 오렌지도 아닙니다.

15. 넌적스 - include

» 파일이 다른 파일을 불러올 수 있음

- include에 파일 경로 넣어줄 수 있음

넌적스

header.html

```
<header>
  <a href="/">Home</a>
  <a href="/about">About</a>
</header>
```

footer.html

```
<footer>
  <div>푸터입니다.</div>
</footer>
```

main.html

```
{% include "header.html" %}
<main>
  <h1>메인 파일</h1>
  <p>다른 파일을 include할 수 있습니다.</p>
</main>
{% include "footer.html" %}
```

HTML

```
<header>
  <a href="/">Home</a>
  <a href="/about">About</a>
</header>
<main>
  <h1>메인 파일</h1>
  <p>다른 파일을 include할 수 있습니다.</p>
</main>
<footer>
  <div>푸터입니다.</div>
</footer>
```

16. 넌적스 - 레이아웃

» 레이아웃을 정할 수 있음

- 공통되는 레이아웃을 따로 관리할 수 있어 좋음, include와도 같이 사용

넌적스

layout.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
    <link rel="stylesheet" href="/style.
css" />
    {% block style %}
    {% endblock %}
  </head>
  <body>
    <header>헤더입니다.</header>
    {% block content %}
    {% endblock %}
    <footer>푸터입니다.</footer>
    {% block script %}
    {% endblock %}
  </body>
</html>
```

body.html

```
{% extends 'layout.html' %}

{% block content %}
<main>
  <p>내용입니다.</p>
</main>
{% endblock %}

{% block script %}
<script src="/main.js"></script>
{% endblock %}
```

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Express</title>
    <link rel="stylesheet" href="/style.css"
/>
  </head>
  <body>
    <header>헤더입니다.</header>
    <main>
      <p>내용입니다.</p>
    </main>
    <footer>푸터입니다.</footer>
    <script src="/main.js"></script>
  </body>
</html>
```

17. 에러 처리 미들웨어

- 에러 발생 시 템플릿 엔진과 상관없이 템플릿 엔진 변수를 설정하고 error 템플릿을 렌더링함
  - res.locals.변수명으로도 템플릿 엔진 변수 생성 가능
  - process.env.NODE_ENV는 개발환경인지 배포환경인지 구분해주는 속성

app.js

```
...
app.use((req, res, next) => {
  const error =  new Error(`${req.method} ${req.url} 라우터가 없습니다.`);
  error.status = 404;
  next(error);
});

app.use((err, req, res, next) => {
  res.locals.message = err.message;
  res.locals.error = process.env.NODE_ENV !== 'production' ? err : {};
  res.status(err.status || 500);
  res.render('error');
});
...
```

그림 6-12 에러 스택 트레이스