웹 시스템 설계
Web System Design

19. Express JS

References

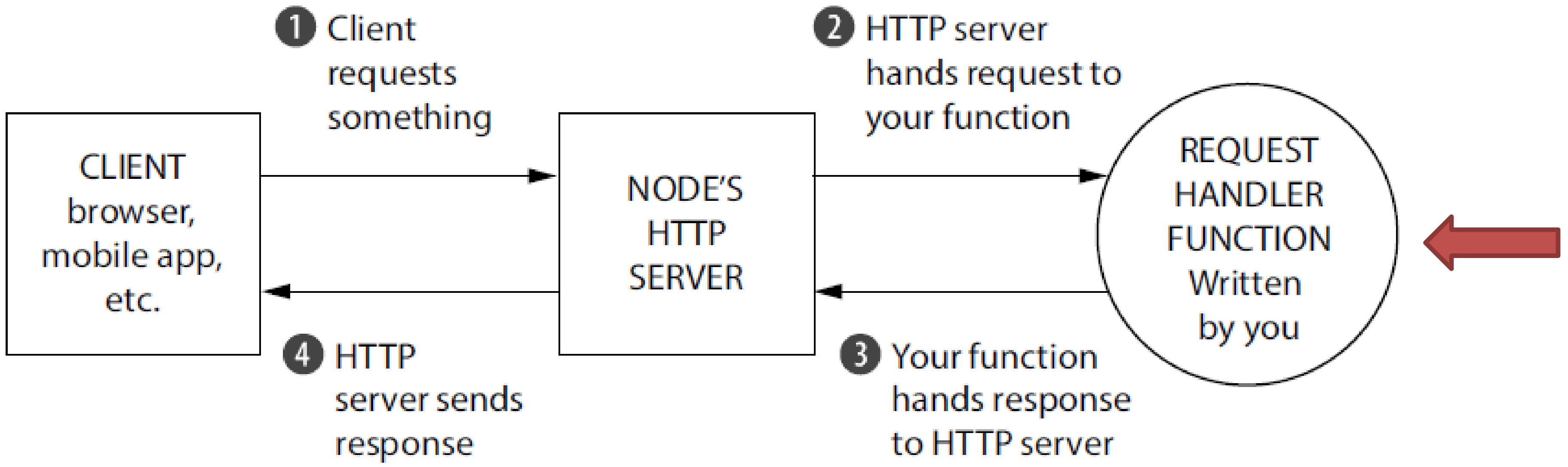
- Full Stack JavaScript Development with MEAN by Adam Bretz and Colin J. Ihrig
- ExpressJS.com Guide http://expressjs.com/ko or http://expressjs.com

Express JS Overview

Overview

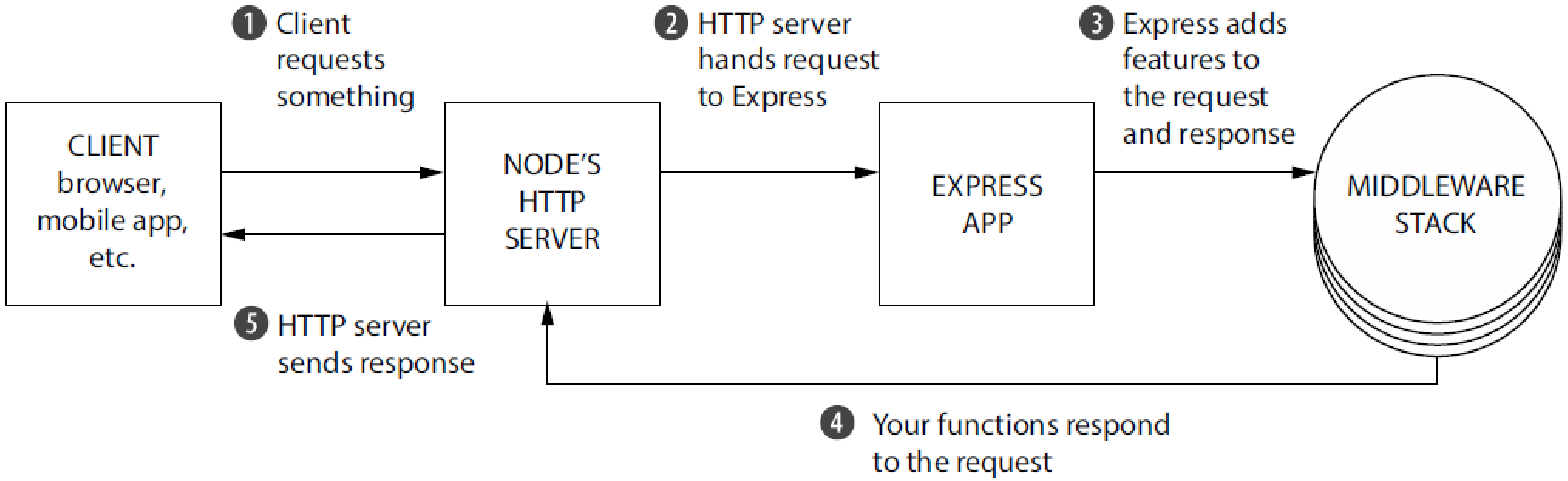
- Express is a Node module that provides a thin web application framework around the core Node modules
  - Request routing
  - Static file server
  - View engine integration

Working with Node



❶ Client requests something
❷ HTTP server hands request to your function
❸ Your function hands response to HTTP server
❹ HTTP server sends response
CLIENT browser, mobile app, etc.
NODE'S HTTP SERVER
REQUEST HANDLER FUNCTION Written by you

Working with Express


1 Client requests something
2 HTTP server hands request to Express
3 Express adds features to the request and response
CLIENT browser, mobile app, etc.
NODE'S HTTP SERVER
EXPRESS APP
MIDDLEWARE STACK
5 HTTP server sends response
4 Your functions respond to the request

Express JS Introduction

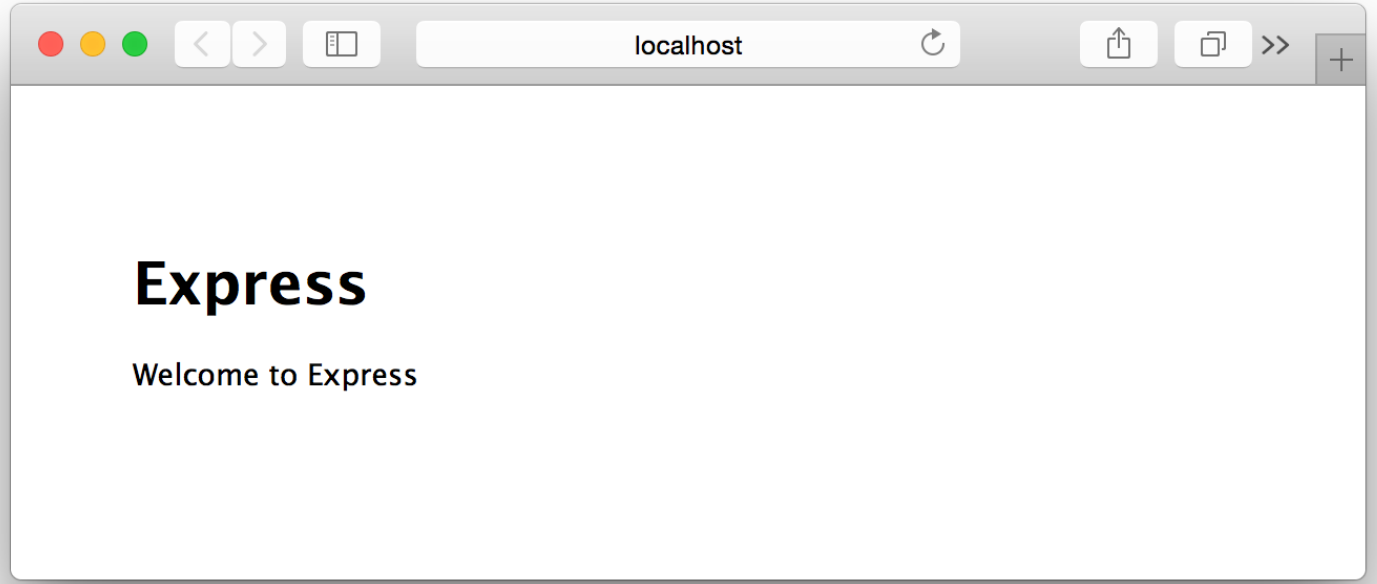
Install Express and Express-generator

```
npm install express --save
npm install express-generator –g

express myFirstApp



npm install // to install dependencies

npm start // start the express app
```

```
/myFirstApp
  | /bin
      | www
  | /public
      | /images
      | /javascripts
      | /stylesheets
  | /routes
      | index.js
      | users.js
  | /views
      | index.pug
  | app.js
  | package.json
```

Generate Express Application Skeleton

❖ Use express-generator


localhost

Express

Welcome to Express

Creating Express App.

- (Express) Generator creates for us.
  - Creating several folders and files, giving developers a reasonable and predictable project structure to work in.
  - Also creates a `package.json` file with some dependencies pre-filled in

- `app.js` is the main entry point for the application
  - where logic for the web server resides.
- `public` folder was created and seeded with subfolders for images, JavaScript files, and style sheets.
- `routes` folder are several files for declaring and attaching routes to the Express app.
  - Since a complete web server there could be thousands of routes, making the app.js file completely unmaintainable. It would also be difficult to work in a team because that file would constantly be changing.

Simple express 'Hello World'

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

NODE

```
const http = require('http');
const server = http.createServer((req, res) => {
  res.end('Hello World\n');
});
server.listen(3000, '127.0.0.1', () => {
  console.log(`Server running at http://localhost:3000/`);
});
```

Routing

❖ *Routing* refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

- Each route can have one or more handler functions, which are executed when the route is matched.
- Route definition takes the following structure:

```
app.METHOD(PATH, HANDLER)
```

- app is an instance of express.
- METHOD is an HTTP request method, in lowercase.
- PATH is a path on the server.
- HANDLER is the function executed when the route is matched.

Middleware

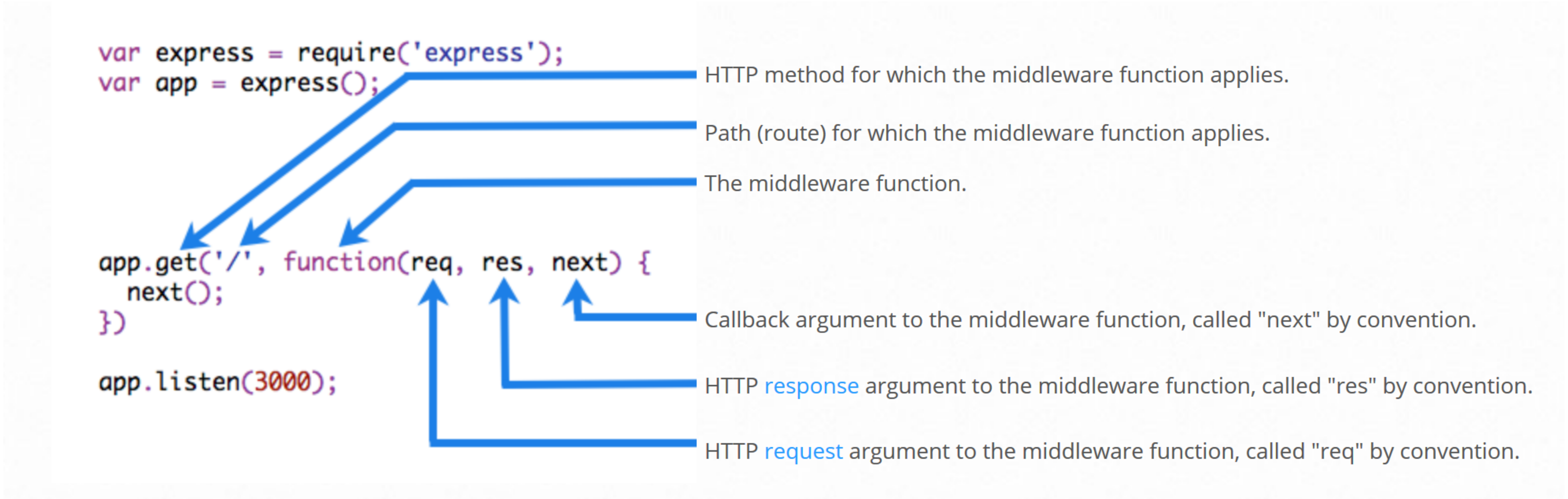
- ❖ 미들웨어 함수는, 요청 오브젝트(req), 응답 오브젝트 (res), 그리고 다음의 미들웨어 함수 대한 액세스 권한을 갖는 함수
  - 다음의 미들웨어 함수는 일반적으로 next라는 이름의 변수로 표시됨
  - 현재의 미들웨어가 request-response 주기를 마무리하지 않는다면, 반드시 next()를 call 해야 함.
- ❖ App 이 처리해야 하는 logic을 여러 middleware 들로 정의하여, 연속하여 수행함 (decompose one large request handler into separate middleware functions.)

- ❖ Middleware의 종류

```
▪ Application-level middleware      // bind to app
▪ Router-level middleware           // bind to router
▪ Error-handling middleware         // app.use(function(err, req, res, next)
▪ Built-in middleware               // express.static
▪ Third-party middleware (ex. cookie-parser)
```

Application Level Middleware of Express (expressjs.com)

- ❖ Bind application-level middleware to an instance of the app object by using the `app.use(`*`middleware`*`)` and `app.METHOD(`*`path, middleware`*`)` functions
  - `METHOD` is the HTTP method of the request that the middleware function handles (such as `GET, PUT, or POST`) in lowercase



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

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

app.listen(3000);
```

HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

HTTP request argument to the middleware function, called "req" by convention.

Application Level Middleware of Express (example) – app_order.js

```
const express = require('express');
const app = express();
// no binding, always executed
app.use((req, res, next) => {
        console.log(`Request is ${req.method} ${req.path}`);
        next();
});
app.get('/', (req, res) => {
        res.end('Hello World!');
});
app.listen(8000, (err) => {
        console.log('Server is running at 8000');
});
```

Order of Middleware – middleware_test.js, middleware_order.js

❖ the order in which they are written/included in your file is the order in which they are executed (the route matches)

```
var express = require('express');
var app = express();
//First middleware before response is sent
app.use(function(req, res, next){
   console.log("Start");
   next();
});
//Route handler
app.get('/', function(req, res, next){
   res.send(“Hello World!");
   console.log("Middle");
   next();
});
app.use('/', function(req, res){
   console.log('End');
}); app.listen(3000);
```

Express Web Server with three routes

Params property employeeId

```
var express = require('express');
var app = express();
// Route one
app.get('/teams/:teamName/employees/:employeeId', function (req, res, next) {
        console.log('teamName = ' + req.params.teamName);
        console.log('employeeId = ' + req.params.employeeId);
        res.send('path one');
});
// Route two
app.get('/teams/:teamName/employees', function (req, res, next) {
        console.log('setting content type');
        res.set('Content-Type', 'application/json');
        res.locals.data = 100 ;
        next();
}, function (req, res, next) {
        console.log('teamName = ' + req.params.teamName);
        console.log(res.locals.data);
        res.send('path two');
});
```

Express Web Server with three routes

```
// Route three
app.get(/^\/groups\/(\w+)\/(\d+)$/, function (req, res, next) {
        console.log('groupname = ' + req.params[0]);
        console.log('groupId = ' + req.params[1]);
        res.send('path three');
});

var server = app.listen(1337, function() {
        console.log('Server started on port 1337');
});
```

Route matching with regular expression

- Route three illustrates using regular expression parameters.

- `(/^\/groups\/(\w+)\/(\d+)$/`
  - `\w+ : matches any word character`
  - `\d+ : matches a digit(equal to [0-9])`
  - `^ : matches beginning of the input`
  - `$ : matches end of input`

- Reference: `https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide/정규식`

Route: Response methods

- The methods on the response object (res) in the following table can send a response to the client, and terminate the request-response cycle.
- If none of these methods are called from a route handler, the client request will be left hanging.

Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile()	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representati on as the response body.

Router Application Middleware of Express

❖ **`var router = express.Router();`**

- **`express.Router`:** Use the `express.Router` class to create modular, mountable route handlers. A Router instance is a complete middleware and routing system("mini-app").

```
var express = require('express');
var router = express.Router();
// middleware that is specific to this router
router.use(function timeLog (req, res, next) {
  console.log('Time: ', Date.now());
  next();
})
// define the home page route
router.get('/', function (req, res) {
  res.send('Birds home page');
})
// define the about route
router.get('/about', function (req, res) {
  res.send('About birds');
})
module.exports = router;
```

```
var birds = require('./birds')

// ...

app.use('/birds', birds)
```