



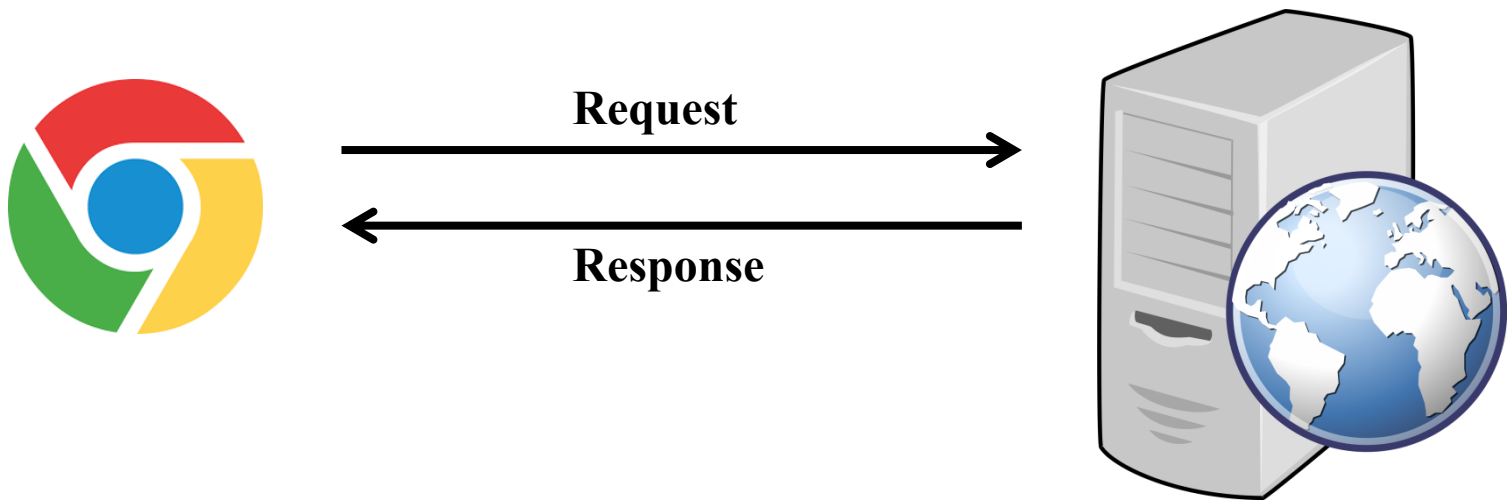
# **Node.js Express**

**Open Source SW Development  
CSE22300**

# Basics

# Request & Response

---



# HTTP Method

---

## GET

- Retrieves a resource
- Guaranteed not to cause side-effect (SAFE)
- Cacheable

## POST

- Creates a new resource
- Unsafe, effect of this verb isn't defined by HTTP

## PUT

- Updates an existing resource
- Used for resource creation when client knows URI
- Can call N times, same thing will always happen (idempotent)

## DELETE

- Removes a resource
- Can call N times, same thing will always happen (idempotent)

# REST

---

- **RESTful API**
  - Uses HTTP requests to **GET, PUT, POST and DELETE** data
  - Uses as a common interface between web services

## REST

GET	/movies	Get list of movies
GET	/movies/:id	Find a movie by its ID
POST	/movies	Create a new movie
PUT	/movies	Update a movie
DELETE	/movies/:id	Delete a movie by ID

# Express

# Express

---

- **Minimal Node.js web application framework**
  - Allows to set up middlewares to respond to HTTP Requests
  - Defines a routing table which is used to perform different actions based on HTTP Method and URL
  - Allows to dynamically render HTML Pages based on passing arguments to templates
- **Install**
  - `npm install express --save`

# Hello World in Express

---

- **Tutorials**

- <http://khuhub.khu.ac.kr/Prof.JinSeongwook/OSS/experiments08>

```
// helloworld in tutorials
var express = require('express');
var app = express();
app.get('/', function (req, res) {
    res.send('Hello World');
})

var server = app.listen(23023, function () {
    var host = server.address().address
    var port = server.address().port
    console.log("Example app listening at http://%s:%s", host, port)
})
```



# Request & Response

---

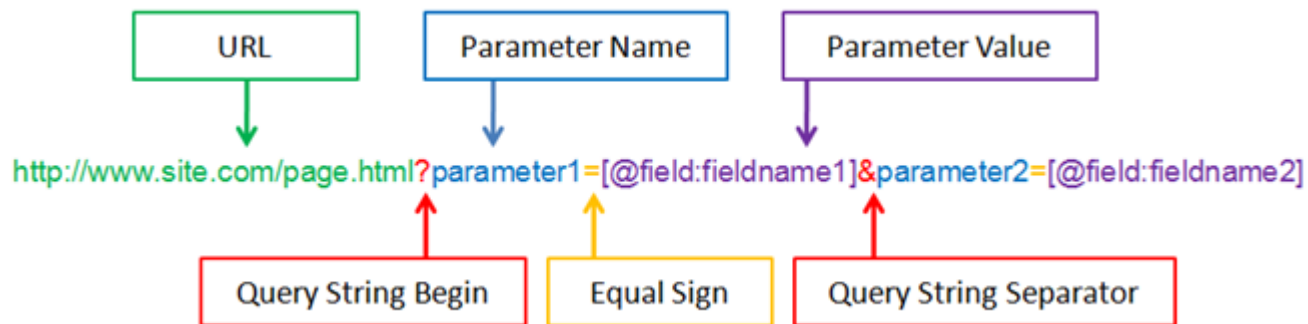
- **Express application uses a callback function whose parameters are request and response objects**

```
app.get('/', function (req, res) {  
  // --  
})
```

- **Request Object**
  - Represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on
- **Response Object**
  - The response object represents the HTTP response that an Express app sends when it gets an HTTP request

# Request Object

- The HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.



method	path	protocol
GET	/tutorials/other/top-20-mysql-best-practices/	HTTP/1.1

```
Host: net.tutsplus.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Pragma: no-cache
Cache-Control: no-cache
```

**HTTP headers** as Name: Value

# Request Properties

---

Properties	Descriptions
req.app	a reference to the instance of the express application
req.baseUrl	The URL path on which a router instance was mounted
req.body	Contains key-value pairs of data submitted in the request body when you use body-parsing middleware
req.cookies	When using cookie-parser middleware, this property is an object that contains cookies
req.ip	The remote IP address of the request.
req.query	An object containing a property for each query string parameter in the route
req.route	The currently-matched route, a string.
req.protocol	The request protocol string, "http" or "https" when requested with TLS.

# Request Method

- **req.accepts(type)**
  - Checks if the specified content types are acceptable, based on the request's Accept HTTP header field

method	path	protocol
GET	/tutorials/other/top-20-mysql-best-practices/	HTTP/1.1

```
Host: net.tutsplus.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Pragma: no-cache
Cache-Control: no-cache
```

HTTP headers as Name: Value

- **req.get(field)**
  - returns the specified HTTP request header field

# Request Method

---

- **req.is(type)**
  - Returns true if the incoming request's "Content-Type" HTTP header field matches the type parameter
- **req.param(name [, defaultValue])**
  - Returns the value of param name when present

```
// ?name=tobi  
req.param('name')  
// => "tobi"
```

# Response Properties

---

Properties	Descriptions
res.app	a reference to the instance of the express application that is using the middleware.
res.headersSent	Boolean property that indicates if the app sent HTTP headers for the response
res.locals	An object that contains response local variables scoped to the request

# Response Method

---

- **res.send([body])**
  - Send the HTTP response
- **res.attachment([filename])**
  - Send a file as an attachment in the HTTP response
- **res.cookie(name, value, [option])**
  - Set cookie name to value
- **res.end([data] [, encoding])**
  - End the response process

# Response Method

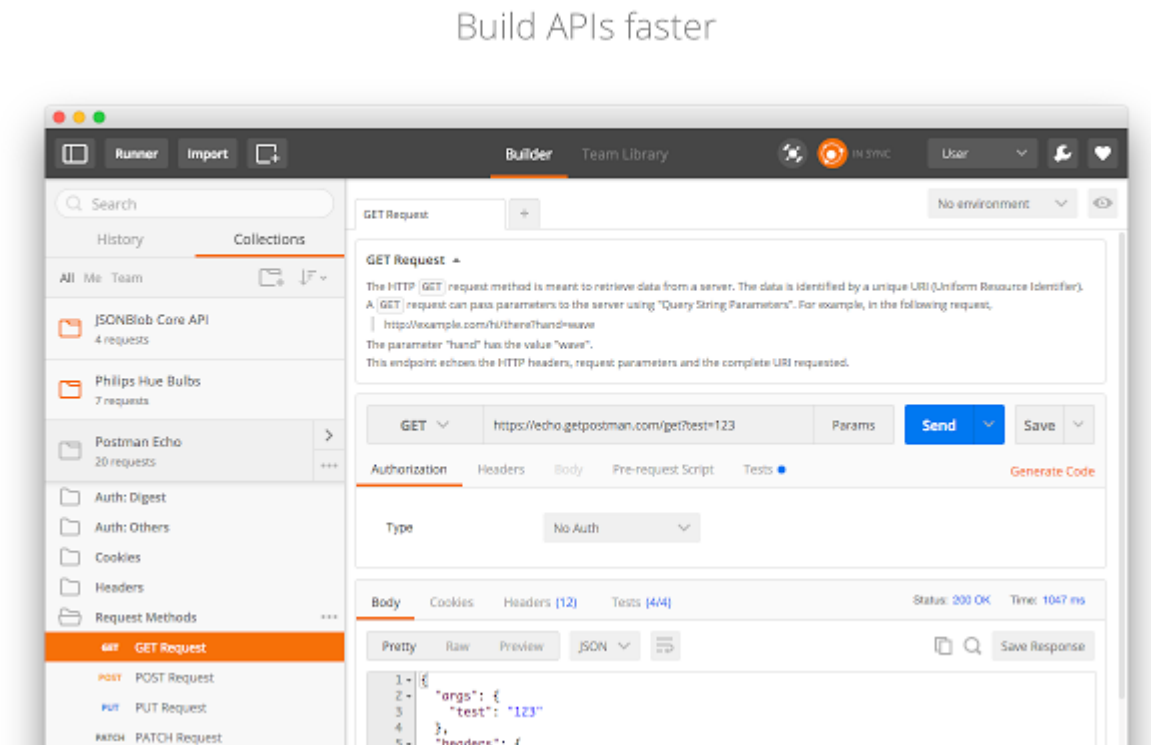
---

- **res.json([body])**
  - Send a JSON response
- **res.render(view [, locals] [, callback])**
  - Render a view and sends the rendered HTML string to the client
- **res.sendStatus(statusCode)**
  - Set the response HTTP status code to `statusCode` and send its string representation as the response body
- **res.set(field [, value])**
  - Set the response's HTTP header field to `value`



# POSTMAN

- The postman application is a request builder that enables easy interaction with an API







# POSTMAN

- Install
  - Chrome Extension
  - Search POSTMAN and Install

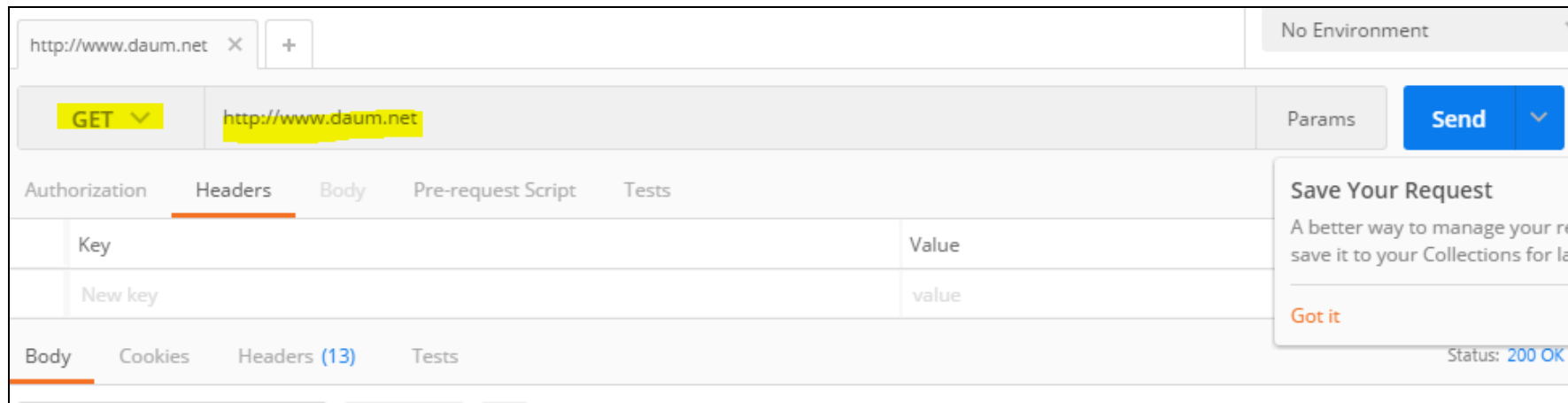
앱

앱 검색결과 더보기

	<div><b>Postman</b></div> <div>www.getpostman.com 제공</div> <div>Supercharge your API workflow with Postman! Build, test, and document your APIs faster. More than a million developers</div>	<div>+ CHROME에 추가</div> <div>개발자 도구</div> <div>★★★★★ (8102)</div> <div>⚡</div>
	<div><b>Galactic Postman</b></div> <div>learnworkplay.net 제공</div> <div>Zip across star systems and galaxies using your puzzle-solving skills to find the shortest routes to each planet.</div>	<div>+ CHROME에 추가</div> <div>퍼즐 및 두뇌게임</div> <div>★★★★☆ (1)</div>
	<div><b>Postal Samurai</b></div> <div>Roa Game...  Android에서 사용 가능 <a href="#">다운로드 &gt;</a></div> <div>Postal Samurai</div>	<div>+ CHROME에 추가</div> <div>아케이드 및 액션 게임</div>

# POSTMAN

- **Install**
  - **Chrome Extension**
  - **Search POSTMAN and Install**
- **API Test**



# Routing

---

- **Routing**
  - Requests are routed to **the code** that handles them
- **Simple Routing**

```
// simplerouting in tutorials
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('hello world');
})

var server = app.listen(23023);
```

# Routing Method

---

- **Routing Method**

- **One of the HTTP methods (GET, POST, PUT, DELETE)**

```
app.get('/', function (req, res) {  
  res.send('GET request to the homepage')  
})  
  
app.post('/', function (req, res) {  
  res.send('POST request to the homepage')  
})
```

- **Special Routing**

- **Handles at a path for all request methods**

```
// speicalrouting in tutorials  
app.all('/', function (req, res, next) {  
  console.log('Accessing the secret section ...')  
  next() // pass control to the next handler  
})
```

# Routing Path

---

- **Routing Path**
  - Define the endpoints at which requests can be made
  - Strings, string pattern, regular expression

```
// routingpath in tutorials
app.get('/about', function (req, res) {
  res.send('about')
})

// The routing path matches requests to /random.text
app.get('/random.text', function (req, res) {
  res.send('random.text')
})

// This route path matches abcd, abxcd, abRANDOMcd, ab123cd, and so on.
app.get('/ab*cd', function (req, res) {
  res.send('ab*cd')
})
```

# Routing Parameter

---

- **Route Parameters**
  - **Route parameters are named URL segments that are used to capture the values**

Route path: /users/:userId/books/:bookId

Request URL:

http://localhost:3000/users/34/books/8989

req.params: { "userId": "34", "bookId": "8989" }

```
// routeparameter in tutorials
```

```
app.get('/users/:userId/books/:bookId', function
```

```
(req, res) {
```

```
  res.send(req.params)
```

```
})
```

# Routing Handler

---

- **Multiple Route Handler**
  - form of a function, an array of function,
  - **next()**

```
// multiplehandler in tutorials
app.get('/example/b', function (req, res, next) {
  console.log('the response will be sent by the next function ...')
  next()
}, function (req, res) {
  res.send('Hello from B!')
})
```

```
var cb0 = function (req, res, next) {
  next()
}
var cb1 = function (req, res, next) {
  res.send('Hello from C!')
}
app.get('/example/c', [cb0, cb1])
```



# Routing Handler

---

- **app.route()**
  - **Create chainable route handlers for a route path**

```
// approute in tutorials
app.route('/book')
  .get(function (req, res) {
    res.send('Get a random book')
  })
  .post(function (req, res) {
    res.send('Add a book')
  })
  .put(function (req, res) {
    res.send('Update the book')
  })
```

# Routing Handler

---

- **express.Router()**
  - Create modular, mountable route handlers

```
// expressrouter in tutorials
var express = require('express')
var router = express.Router()

router.get('/', function (req, res) {
  res.send('Birds home page')
})
router.get('/about', function (req, res) {
  res.send('About birds')
})

module.exports=router;
```

# Routing Handler

---

- **express.Router()**

```
var birds = require('./birds')  
  
// ...  
  
app.use('/birds', birds)
```

- **Router Modular**

# Static Files

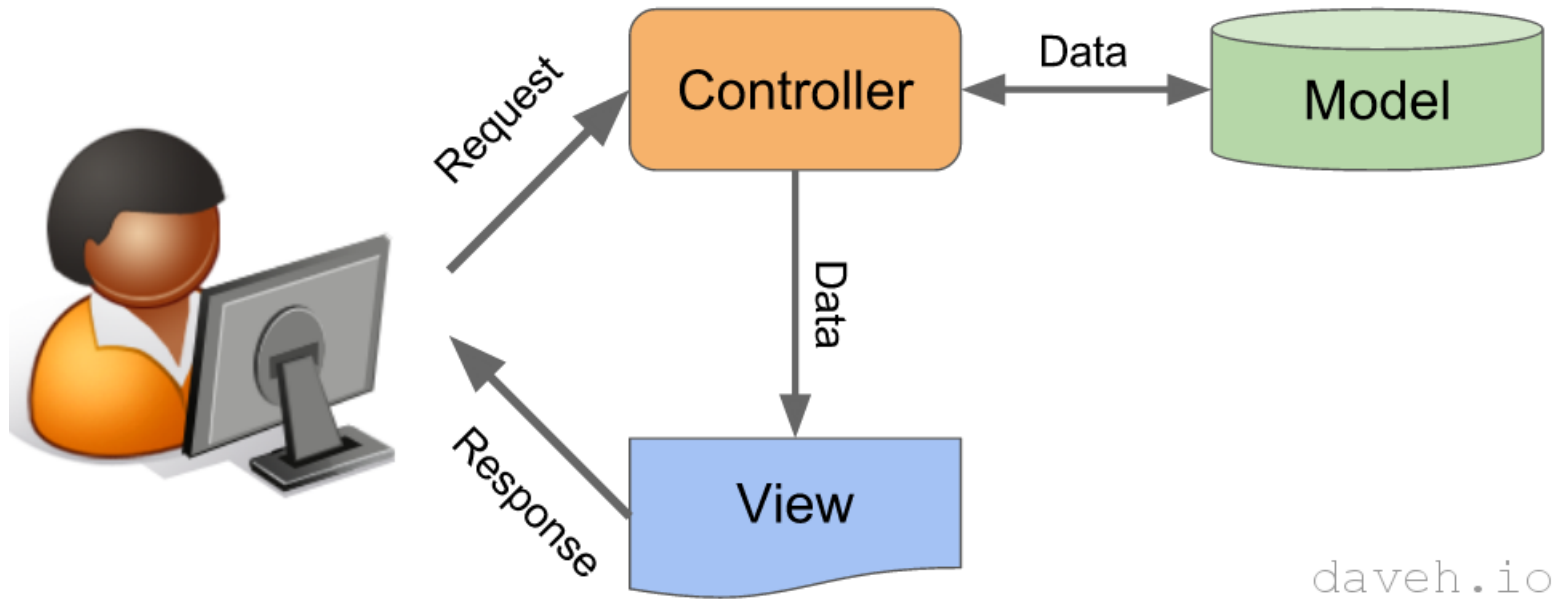
---

- **express.static**
  - serve static files, such as images, CSS, JavaScript, etc
- **app.use(express.static('public'))**
  - if you keep your images, CSS, and JavaScript files in a directory named **public**

```
// staticfiles in tutorials
var express = require('express');
var app = express();
app.use(express.static('public'));
app.get('/', function (req, res) {
  res.send('Hello World');
})
var server = app.listen(8081, function () {
  console.log("Example app listening");
})
```

# MVC

- **Model-View-Controller**



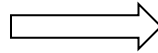
# Template

---

- **EJS**
  - **Embedded JavaScript templates**

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

template : ejs file



```
<h2>MyName</h2>
```

result : html

- **Features**
  - **Control flow with <% %>**
  - **Escaped output with <%= %> (escape function configurable)**
  - **Unescaped raw output with <%- %>**

# Rendering with ejs

---

- **Install EJS**
  - `npm install --save ejs`
- **Setting**
  - `app.set('view engine', 'ejs')`
  - tell our application what view engine are we using
- **Creates index.ejs file in view directory**

```
<html>
<body>
<%= title %>
</body>
</html>
```

# Rendering with ejs

---

- **Rendering code**

```
// render in tutorials
app.get('/', function(req, res) {
  res.render('index', { title: 'The index page!' })
});
```



# Session Management

---

- **HTTP is a *stateless* protocol**
  - HTTP request contains all the information necessary for the server to satisfy the request
- **Cookie**
  - The server sends a bit of information, and the browser stores it for some configurable period of time
  - Security problem (Cookies are not secret from the user)
- **Session**
  - Store only a unique identifier in the cookie and on the server  
**everything else**

# Session Management

---

- **express-session**
  - Creates and manages sessions
- **session(option)**
  - Creates a session middleware

```
var app = express()
var session = require('express-session')
app.use(session({
  secret: 'keyboard cat',
  resave: false,
  saveUninitialized: true,
  cookie: { secure: true }
}))
```

# Session Management

---

- **req.session**
  - **To store or access session data, simply use the request property req.session**

```
// session in tutorials
app.use(session({ secret: 'keyboard cat', cookie: { maxAge: 60000 } }))
app.get('/', function(req, res, next) {
  var sess = req.session
  if (sess.views) {
    sess.views++
    res.setHeader('Content-Type', 'text/html')
    res.write('<p>views: ' + sess.views + '</p>')
    res.write('<p>expires in: ' + (sess.cookie.maxAge / 1000) + 's</p>')
    res.end()
  } else {
    sess.views = 1
    res.end('welcome to the session demo. refresh!')
  }
})
```

# Session Management

---

- **Session.destroy(callback)**
  - Destroys the session and will unset the req.session property. Once complete, the callback will be invoked

```
req.session.destroy(function(err) {  
  // cannot access session here  
})
```

- **Session.save(callback)**
  - Save the session back to the store, replacing the contents on the store with the contents in memory

```
req.session.save(function(err) {  
  // session saved  
})
```