

<WA1/>  
<AW1/>  
2021

# express

## A look at the server side

Fulvio Corno

Luigi De Russis

Enrico Masala



# Goal

- Implement a (simple, minimal) web server
  - In JavaScript
  - For hosting static contents
  - For hosting dynamic APIs
  - Supporting persistence in a Database

Express<sup>4.17.1</sup>  
Fast, unopinionated,  
minimalist web framework for  
Node.js

<https://expressjs.com/>  
<https://github.com/expressjs/express>

The Protocol of the Web

# HTTP

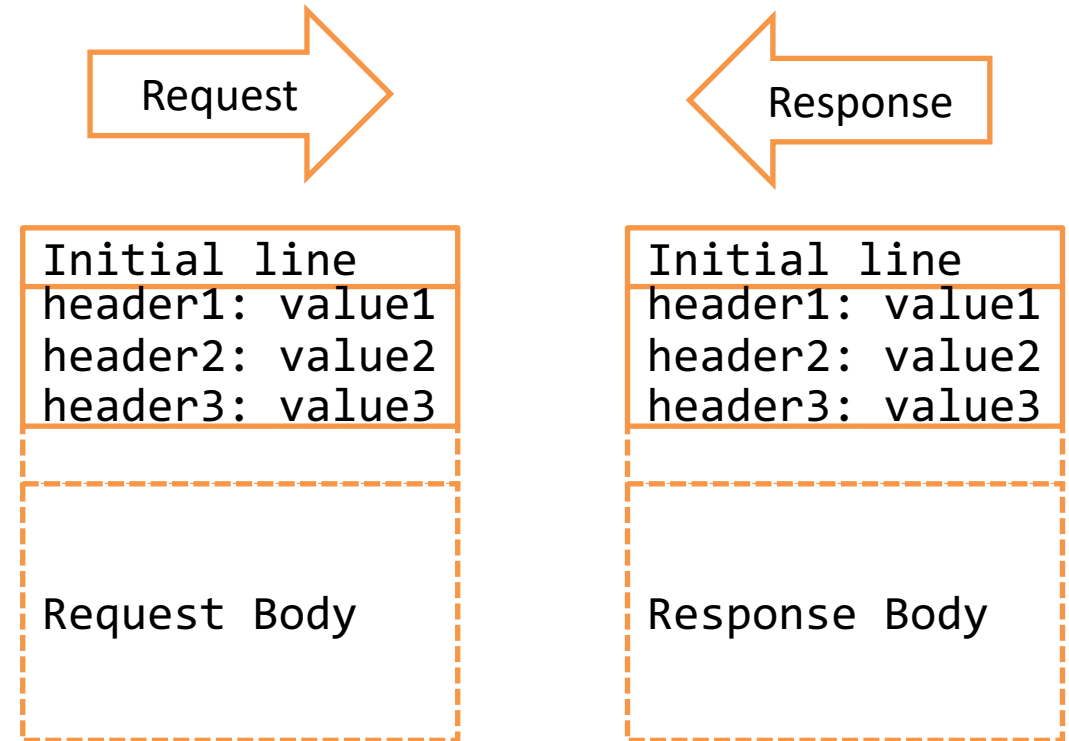
<https://tools.ietf.org/html/rfc7230>  
<https://tools.ietf.org/html/rfc7231>

Connection: keep-alive

• • • • •

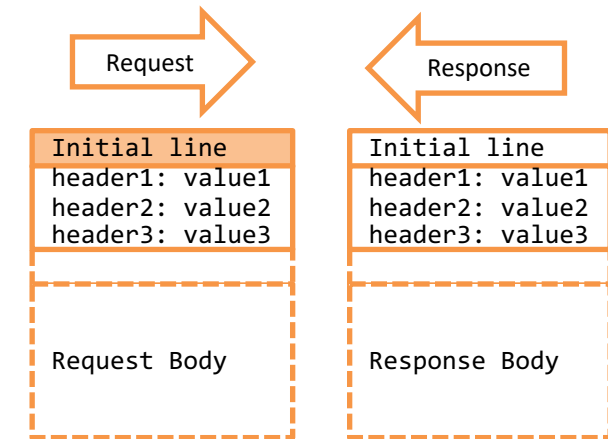
# HTTP Messages

- An **initial** line
- Zero or more **header** lines
- A **blank** line (CRLF)
- An optional message **body**



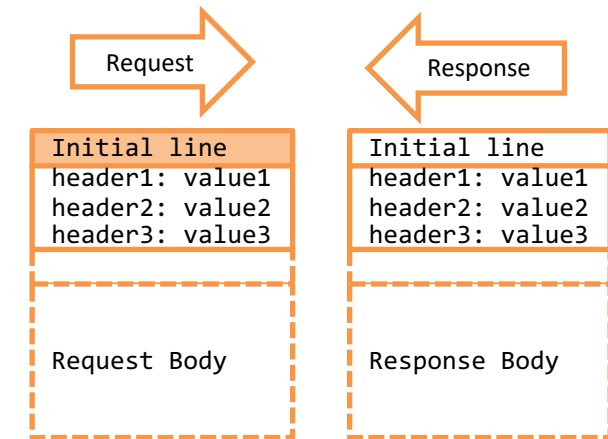
# Request – Initial Line

- A **request** initial line has three parts separated by white spaces:
  - **Method** name
  - Local **path** of the requested resource
  - Version of HTTP being used
- GET /path/to/file/index.html HTTP/1.0



# HTTP Methods

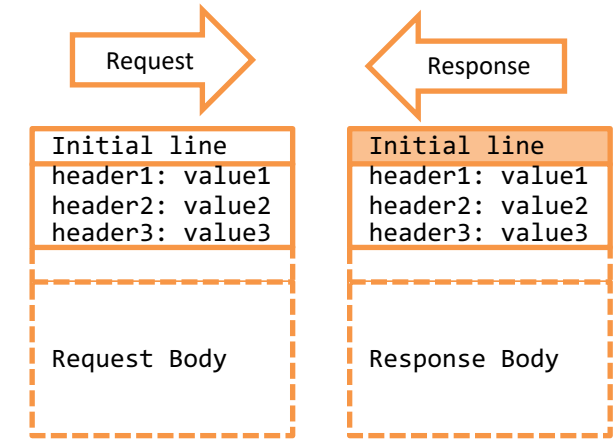
|                |   |
|----------------|---|
| <b>GET</b>     | Requests a representation of the specified resource. Should only retrieve data.                           |
| <b>HEAD</b>    | Asks for a response identical to GET, but without the response body                                       |
| <b>POST</b>    | Submit an entity to the specified resource, often causing a change in state or side effects on the server |
| <b>PUT</b>     | Replaces current representations of the target resource with the request payload                          |
| <b>DELETE</b>  | Deletes the specified resource  |
| <b>TRACE</b>   | Message loop-back test along the path to the target resource  |
| <b>OPTIONS</b> | Describe the communication options for the target resource  |
| <b>CONNECT</b> | Establish a tunnel to the server identified by the target resource  |
| <b>PATCH</b>   | Apply partial modifications to a resource   |



<https://tools.ietf.org/html/rfc7231#section-4.3>

# Response – Initial Line

- A status line
- 3 parts separated by spaces:
  - The HTTP version
  - The response **status code**
  - An English phrase describing the status code
- Example:
  - **HTTP/1.0 200 OK**
  - **HTTP/1.0 404 Not Found**

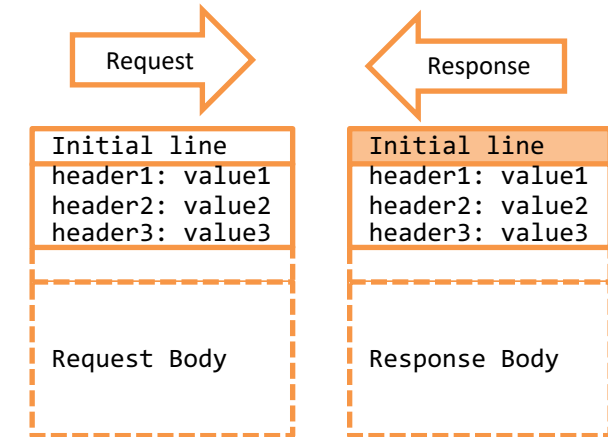




# Response Status Codes

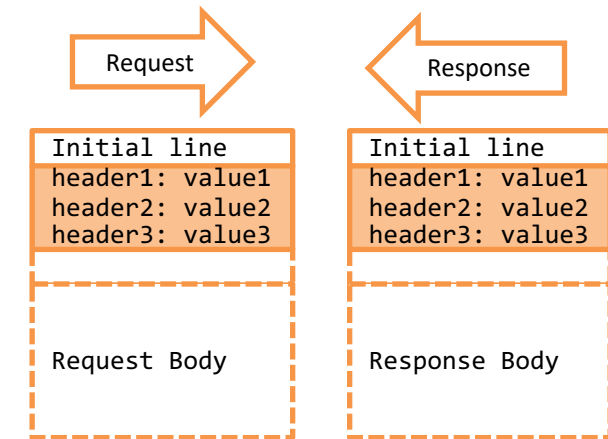
- 1xx – Informational
- 2xx – Success
- 3xx – Redirection
- 4xx – Client Error
- 5xx – Server Error

- 100 Continue
- 101 Switching Protocols
- **200 OK**
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 300 Multiple Choices
- **301 Moved Permanently**
- 302 Found
- 303 See Other
- 305 Use Proxy
- **307 Temporary Redirect**
- 400 Bad Request
- 402 Payment Required
- 403 Forbidden
- **404 Not Found**
- 405 Method Not Allowed
- 406 Not Acceptable
- 408 Request Timeout
- 410 Gone
- 411 Length Required
- 413 Payload Too Large
- 414 URI Too Long
- 415 Unsupported Media Type
- 417 Expectation Failed
- 426 Upgrade Required
- **500 Internal Server Error**
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout
- 505 HTTP Version Not Supported



# Header Lines

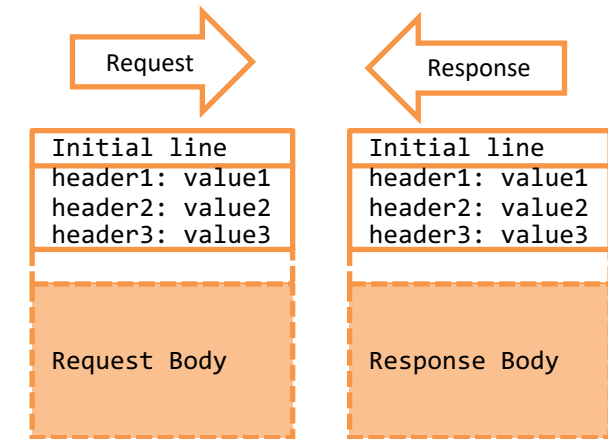
- Information about the request/response
- Information about the object sent in the message body
- One line per header
- Header-Name: header-value
- HTTP/1.1 defines 46 headers. Only 1 is mandatory in all requests:
  - Host



<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

# Message Body

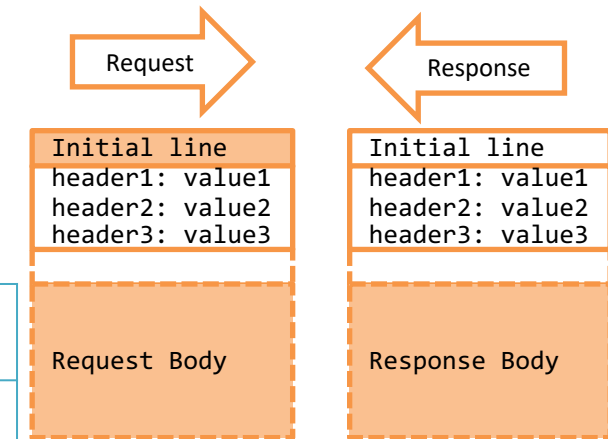
- Data sent after the header lines
  - **Request:** data entered in a form, a file to upload, ...
  - **Response:** the resource returned to the client
    - Images
    - text/plain, text/html
    - ...
- **Content-Type** (header) indicates the media type of the resource
  - Content-Type: `text/html; charset=UTF-8`
  - Content-Type: `application/json`
  - Content-Type: `multipart/form-data; boundary=something`
  - Content-Type: `application/x-www-form-urlencoded`
- **Content-Encoding:** the compression (e.g., gzip) applied to the body



[https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics\\_of\\_HTTP/MIME\\_types](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types)

# Body In Different HTTP Methods

| Method | Request Body                       | Response Body                      | Idempotent | HTML Forms |
|--------|------------------------------------|------------------------------------|------------|------------|
| GET    | No                                 | Yes: resource content              | Yes        | Yes        |
| HEAD   | No                                 | No                                 | Yes        | No         |
| POST   | Yes: form data or application data | May (usually modification results) | No         | Yes        |
| PUT    | Yes: application data              | May (usually modification results) | Yes        | No         |
| DELETE | May                                | May                                | Yes        | No         |



<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>



The Express Handbook, Flavio Copes

<https://flaviocopes.com/page/express-handbook/>

A simple and easy to use HTTP and Application server

# EXPRESS

# Web Frameworks in Node

- Node already contains a 'http' module to activate a web server
  - Low-level, not very friendly
- Several other frameworks were developed
- **Express** is one of the most popular, and quite easy to use

```
npm init  
npm install express  
node index.js
```

|              |      |        |   |
|--------------|------|--------|---|
| ✓ Express    | Star | 48,190 | i |
| ✓ koa.js     | Star | 28,930 | i |
| ✓ Lad        | Star | 1,711  | i |
| ✓ fastify    | Star | 14,131 | i |
| ✓ hapi       | Star | 12,285 | i |
| ✓ total.js   | Star | 4,058  | i |
| ✓ flatiron   | Star | 1,346  | i |
| ✓ locomotive | Star | 879    | i |
| ✓ diet.js    | Star | 381    | i |
| ✓ Flicker.js | Star | 18     | i |
| ✓ ZinkyJS    | Star | 27     | i |

<http://nodeframework.com/>

# Running the Express Server



- `node index.js`
- Will start the server application with the specified file
- Until the application crashes, or is interrupted by the user (^C)
- If you modify a file, it must be stopped and restarted.

- Useful Tip: **nodemon**
- **nodemon** executes a script with **node**, and monitors any changes of the JS files
- **node** is automatically restarted if a file is modified
- **npm install -g nodemon**
- `nodemon index.js`

# First Steps With Express

- Calling `express()` creates an application object `app`
- `app.listen()` starts the server on the specified port (3000)
- Incoming HTTP requests are routed to a callback according to
  - **path**, e.g., `'/'`
  - **method**, e.g., `get`
- Callback receives Request and Response objects (**req**, **res**)

```
// Import package
const express = require('express') ;
// Create application
const app = express() ;

// Define routes and web pages
app.get('/', (req, res) =>
    res.send('Hello World!')) ;

// Activate server
app.listen(3000, () =>
    console.log('Server ready')) ;
```



# Routing

- `app.method(path, handler);`
  - **app**: the express instance
  - **method**: an HTTP Request method (get, post, put, delete, ...)
    - `app.all()` catches all request types
  - **path**: a path on the server
    - Matched with the path in the HTTP Request Message
  - **handler**: callback executed when the route is matched

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

# Handler Callbacks

```
function (req, res) { ... }
```

## req (Request object)

| Property       | Description  |
|----------------|--|
| .app           | holds a reference to the Express app object  |
| .baseUrl       | the base path on which the app responds  |
| .body          | contains the data submitted in the request body (must be parsed and populated manually before you can access it) |
| .cookies       | contains the cookies sent by the request (needs the <code>cookie-parser</code> middleware)                       |
| .hostname      | the server hostname  |
| .ip            | the server IP  |
| .method        | the HTTP method used   |
| .params        | the route named parameters   |
| .path          | the URL path   |
| .protocol      | the request protocol   |
| .query         | an object containing all the query strings used in the request   |
| .secure        | true if the request is secure (uses HTTPS)   |
| .signedCookies | contains the signed cookies sent by the request (needs the <code>cookie-parser</code> middleware)                |
| .xhr           | true if the request is an <a href="#">XMLHttpRequest</a>   |

## res (Response object)

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

<https://expressjs.com/en/guide/routing.html>

# Generate a Response

- `res.send('something')` sets the response body and returns it to the browser
- `res.end()` sends an empty response
- `res.status()` sets the response status code
  - `res.status(200).send()`
  - `res.status(404).end()`
- `res.json()` sends an object by serializing it into JSON
  - `res.json({a:3, b:7})`
- `res.download()` prompts the user to download (not display) the resource

# Redirects

- `res.redirect('/go-there')`

# Extending express With 'Middlewares'

- **Middleware:** a function that is called for every request
- `function(req, res, next)`
  - Receives (req, res), may process and modify them
  - Calls `next()` to activate the next middleware function
- **Insert** a middleware on a specific route
  - `app.method(path, middlewareCallback, (req, res)=>{})`
- **Register** a middleware with
  - `app.use(middlewareCallback)`
  - `app.use(path, middlewareCallback)`  
// handles requests in the specified path, only

# Serving Static Requests

- Middleware: `express.static(root, [options])`
- All files under the root are served automatically
  - No need to register `app.get` handlers per each file

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

Serves files from `./public` as:

`http://localhost:3000/images/kitten.jpg`

`http://localhost:3000/css/style.css`

`http://localhost:3000/js/app.js`

`http://localhost:3000/images/bg.png`

`http://localhost:3000/hello.html`

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

Serves files from `./public` as:

`http://localhost:3000/static/images/kitten.jpg`

`http://localhost:3000/static/css/style.css`

`http://localhost:3000/static/js/app.js`

`http://localhost:3000/static/images/bg.png`

`http://localhost:3000/static/hello.html`

# Interpreting Request Parameters

| Request method | Parameters  | Values available in  | Middleware requested              |
|----------------|---|--|-----------------------------------|
| GET            | URL-encoded<br><code>/login?user=fc&amp;pass=123</code>                         | <code>req.query</code><br><code>req.query.user</code><br><code>req.query.pass</code> | none                              |
| POST / PUT     | FORM-encoded in the request body  | <code>req.body</code><br><code>req.body.user</code><br><code>req.body.pass</code>    | <code>express.urlencoded()</code> |
| POST / PUT     | JSON stored in the request body<br><code>{ "user": "fc", "pass": "123" }</code> | <code>req.body</code><br><code>req.body.user</code><br><code>req.body.pass</code>    | <code>express.json()</code>       |

# Paths

| Path type                          | Example   |
|------------------------------------|---|
| Simple paths (String prefix)       | <code>app.get('/abcd', (req, res, next)=&gt; {</code>   |
| Path Pattern (Regular expressions) | <code>app.get('/abc?d', (req, res, next)=&gt; {<br/>app.get('/ab+cd', (req, res, next)=&gt; {<br/>app.get('/ab\*cd', (req, res, next)=&gt; {<br/>app.get('/a(bc)?d', (req, res, next)=&gt; {</code> |
| JS Regexp object                   | <code>app.get(/\//abc \//xyz/, (req, res, next)=&gt; {</code>   |
| Array (more than one path)         | <code>app.get(['/abcd', '/xyza', /\//lmn \//pqr/],<br/>      (req, res, next)=&gt; {</code>   |

<https://expressjs.com/en/4x/api.html#path-examples>



# Parametric Paths

- A Path may contain one or more *parametric segments*:
  - Using the `:id` syntax
  - Free matching segments
  - Bound to an identifier
  - Available in `req.params`
- May specify a matching regexp
  - `/user/:userId([0-9]+)`

```
app.get('/users/:userId/books/:bookId', (req, res) => {  
  res.send(req.params)  
});
```

*Request URL:*

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

*Results in:*

`req.params.userId == "34"`

`req.params.bookId == "8989"`

<https://expressjs.com/en/guide/routing.html#route-parameters>

# Logging

- By default, express does not log the received requests
- For debugging purposes, it is useful to activate a logging middleware
- Example: **morgan**
  - <https://github.com/expressjs/morgan> (npm install morgan)
  - `const morgan = require('morgan');`
  - `app.use(morgan('dev'));`

# Validating Input


- <https://express-validator.github.io/docs/>
  - npm install express-validator
- Declarative validator for query parameters

```
app.post('/user', [ // additional (2nd) parameter in app.post to pre-process request
  check('username').isEmail(), // username must be an email
  check('password').isLength({ min: 5 }) // password must be at least 5 chars long
], (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).json({ errors: errors.array() });
  }
  . . . Process request
});
```

<https://github.com/validatorjs/validator.js#validators>

# Other Middlewares

| Middleware module  | Description  |
|--|--|
| <a href="#">body-parser</a>  | Parse HTTP request body. See also: <a href="#">body</a> , <a href="#">co-body</a> , and <a href="#">raw-body</a> .         |
| <a href="#">compression</a>  | Compress HTTP responses.   |
| <a href="#">connect-redis</a>  | Generate unique request ID.  |
| <a href="#">cookie-parser</a>  | Parse cookie header and populate <code>req.cookies</code> . See also <a href="#">cookies</a> and <a href="#">keygrip</a> . |
| <a href="#">cookie-session</a>   | Establish cookie-based sessions.   |
|  <a href="#">cors</a>   | Enable cross-origin resource sharing (CORS) with various options.  |
| <a href="#">csurf</a>  | Protect from CSRF exploits.  |
| <a href="#">errorhandler</a>   | Development error-handling/debugging.  |
| <a href="#">method-override</a>  | Override HTTP methods using header.  |
|  <a href="#">morgan</a> | HTTP request logger.   |
| <a href="#">multer</a>   | Handle multi-part form data.   |
| <a href="#">response-time</a>  | Record HTTP response time.   |
| <a href="#">serve-favicon</a>  | Serve a favicon.   |
| <a href="#">serve-index</a>  | Serve directory listing for a given path.  |
| <a href="#">serve-static</a>   | Serve static files.  |
| <a href="#">session</a>  | Establish server-based sessions (development only).  |
| <a href="#">timeout</a>  | Set a timeout period for HTTP request processing.  |
| <a href="#">vhost</a>  | Create virtual domains.  |

| Middleware module  | Description  |
|--|--|
| <a href="#">cls-rtracer</a>  | Middleware for CLS-based request id generation. An out-of-the-box solution for adding request ids into your logs.  |
| <a href="#">connect-image-optimus</a>  | Optimize image serving. Switches images to <code>.webp</code> or <code>.jxx</code> , if possible.  |
| <a href="#">express-debug</a>  | Development tool that adds information about template variables (locals), current session, and so on.  |
| <a href="#">express-partial-response</a>   | Filters out parts of JSON responses based on the <code>fields</code> query-string; by using Google API's Partial Response.                                     |
| <a href="#">express-simple-cdn</a>   | Use a CDN for static assets, with multiple host support.   |
| <a href="#">express-slash</a>  | Handles routes with and without trailing slashes.  |
| <a href="#">express-stormpath</a>  | User storage, authentication, authorization, SSO, and data security.   |
| <a href="#">express-uncapitalize</a>   | Redirects HTTP requests containing uppercase to a canonical lowercase form.  |
| <a href="#">helmet</a>   | Helps secure your apps by setting various HTTP headers.  |
| <a href="#">join-io</a>  | Joins files on the fly to reduce the requests count.   |
|  <a href="#">passport</a> | Authentication using "strategies" such as OAuth, OpenID and many others. See <a href="http://passportjs.org/">http://passportjs.org/</a> for more information. |
| <a href="#">static-expiry</a>  | Fingerprint URLs or caching headers for static assets.   |
| <a href="#">view-helpers</a>   | Common helper methods for views.   |
| <a href="#">sriracha-admin</a>   | Dynamically generate an admin site for Mongoose.   |

<https://expressjs.com/en/resources/middleware.html>

# License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for [commercial purposes](#).
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
  - **No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

