<WA/>
2025

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

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

The Protocol of the Web

HTTP

HTTP protocol

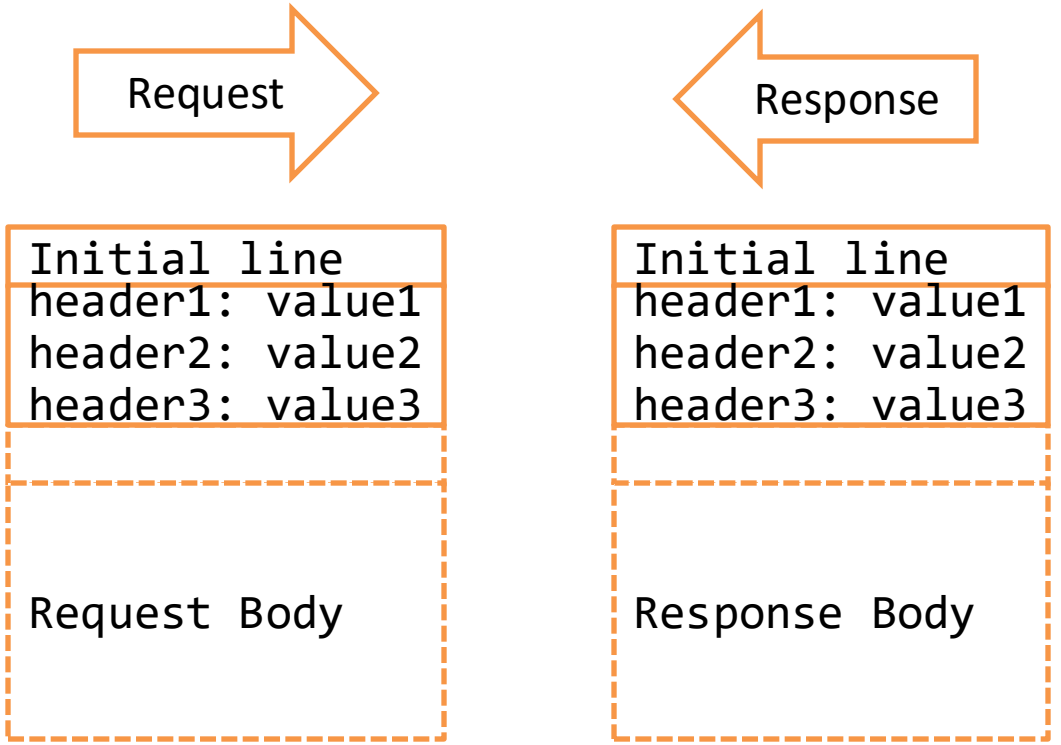
https://tools.ietf.org/html/rfc7230
https://tools.ietf.org/html/rfc7231

```
GET / HTTP/1.1
Host: elite.polito.it
User-Agent: Mozilla/5.0
Accept: text/html,appl
Accept-Language: it-I
Accept-Encoding: gzi
Cookie: __utma=1885
Connection: keep-a
```

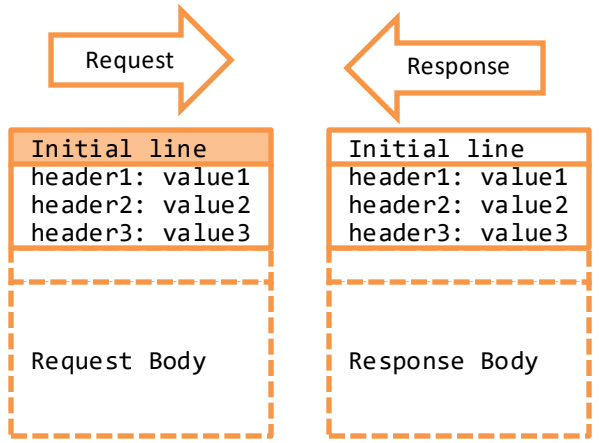
```
HTTP/1.0 200 OK
Cache-Control: no-store, no-cache, must-revalidate,
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 08 Apr 2016 13:36:24 GMT
Expires: Mon, 1 Jan 2020 00:00:00 GMT
Keep-Alive: timeout=15, max=100
Last-Modified: Wed, 08 Apr 2016 13:36:24 GMT
Pragma: no-cache
Server: Apache/2.4.6 (Linux/SUSE)
Transfer-Encoding: chunked
X-Powered-By: PHP/5.6.30
p3p: CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM«

<!DOCTYPE html>
<html>
<head>
. . . . . .
```

HTTP Messages

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



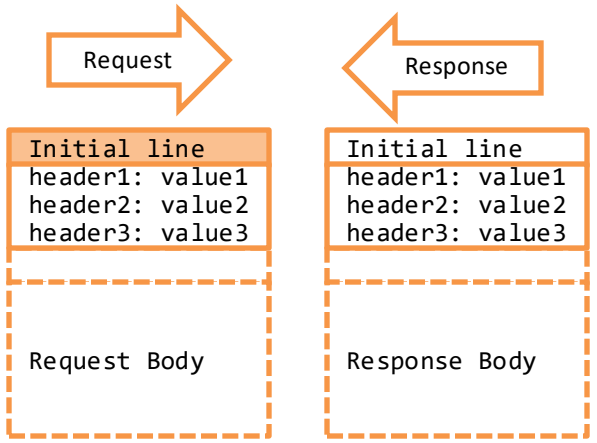
Request – Initial Line



- 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`

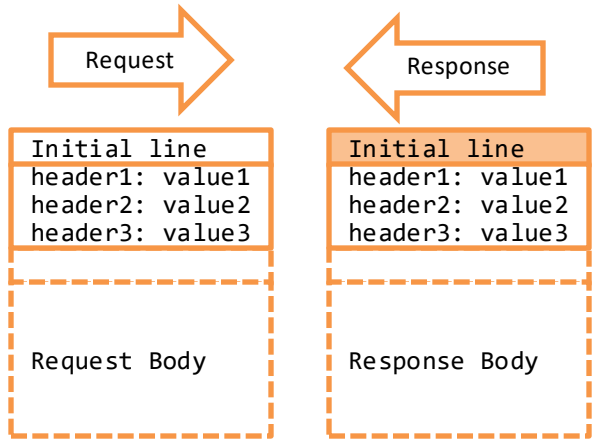
HTTP Methods



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

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

Response – Initial Line

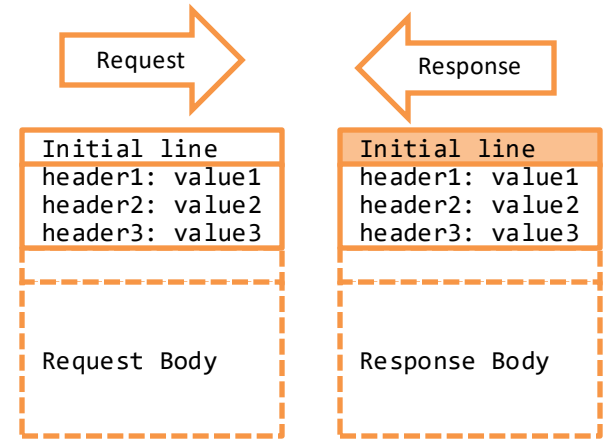


- A status line
- 3 parts separated by spaces:
  - The HTTP version
  - The response status code
  - An English phrase describing the status code
- Example:
  - `HTTP/1.0 200 OK`
  - `HTTP/1.0 404 Not Found`

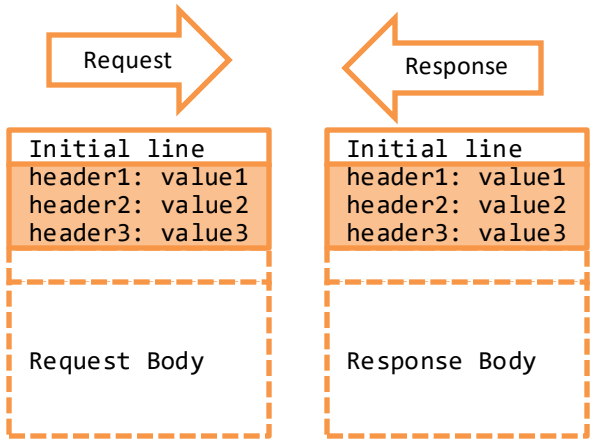
Response Status Codes

- 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



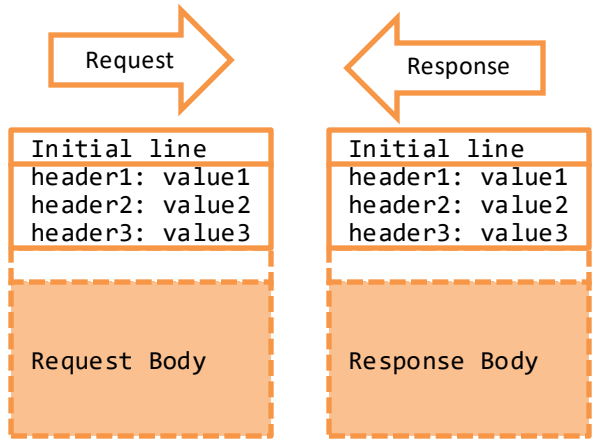
Header Lines



- 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

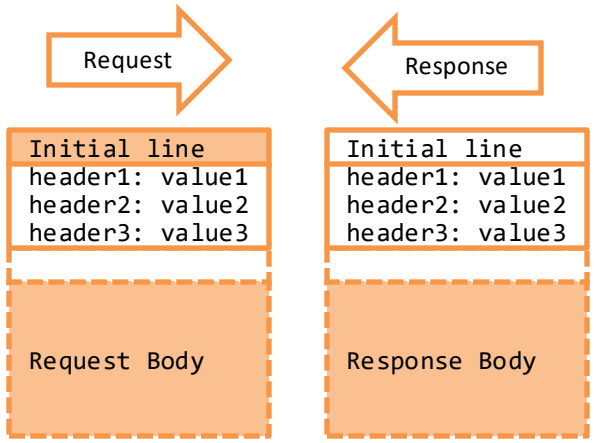
Message Body



- 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

Body In Different HTTP Methods



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

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
```

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 (e.g. by pressing CTRL+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

- `sudo npm install -g nodemon`
- `nodemon index.js`

Very important: For this course, nodemon is the only package that can be installed globally (with –g). All other packages must be installed locally to the project, thus listed in the project's `package.json` file

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 `cookie-parser` 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 `cookie-parser` middleware)
.xhr	true if the request is an XMLHttpRequest

`res` (Response object)

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 representation as the response body.

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

Generate an HTTP 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

Terminating the server

For this course, for the exam, only use the port specified in the template

- The server can be stopped by pressing CTRL+C in the terminal
- **Note**: ONLY ONE TCP server at a time can run on a specified host and port
- If another server is launched on the same port, an error will arise

```
Error: listen EADDRINUSE: address already in use :::3000
```

- Either use another port, or terminate the other server (look for the PID)

- How to terminate a server in Linux if the terminal is lost

```
$ netstat -atpn
tcp  0  0 0.0.0.0:3000  0.0.0.0:* LISTEN 3622/node
$ kill 3622    # PID of the process to terminate
```

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
- **To insert** a middleware on a *specific* route
  - `app.method(path, middlewareCallback, (req,res)=>{})`
- **To register** a middleware for many routes:
  - `app.use(middlewareCallback)  // all routes`
  - `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 folder 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 folder 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
```

https://expressjs.com/en/starter/static-files.html

Interpreting Request Parameters

Request method	Parameters	Values available in	Middleware required
GET	URL-encoded `/search?filter=flagged`	`req.query` `req.query.filter`	`none`

POST / PUT	FORM-encoded in the request body	`req.body`
		`req.body.user`
		`req.body.pass`
			`express.urlencoded()`

POST / PUT	JSON stored in the request body `{ "user": "fc", "pass": "123" }`		`express.json()`

Paths

Path type	Example
Simple paths (String prefix)	`app.get('/abcd', (req, res, next)=> {`

Path Pattern (Regular expressions)	`app.get('/abc?d', (req, res, next)=> {`
	`app.get('/ab+cd', (req, res, next)=> {`
	`app.get('/ab\*cd', (req, res, next)=> {`
	`app.get('/a(bc)?d', (req, res, next)=> {`

JS Regexp object	`app.get(/\/abc|\/xyz/, (req, res, next)=> {`
Array (more than one path)	`app.get(['/abcd', '/xyza', /\/lmn|\/pqr/],` `(req, res, next)=> {`

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'));`

Other Middlewares

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

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