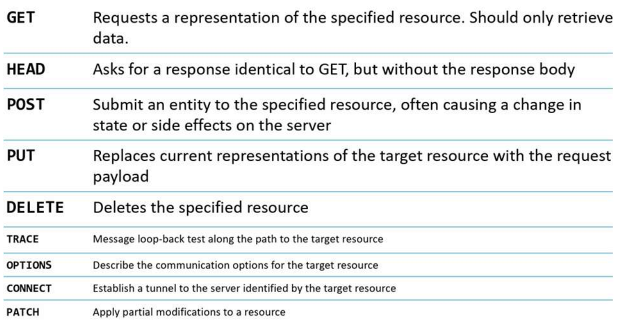
EXPRESS

http

DOMANDA



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

GET → manda richiesta per prendere dei dati

POST → form (es: login)

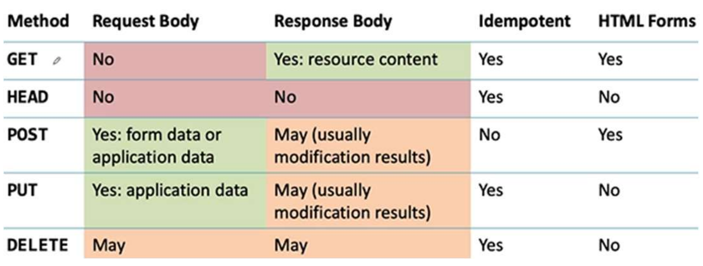
DELETE → manda richiesta per cancellare qualcosa

PUT → aggiorna dati già esistenti

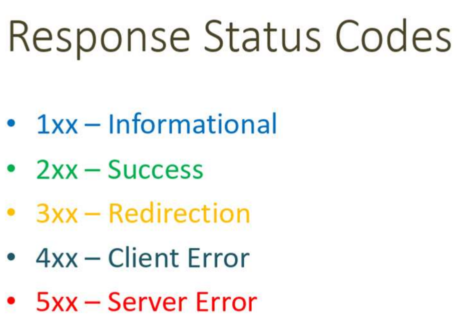
PATCH → aggiornamento parziale di dati già esistenti

RISPOSTA

- Corpo
- Metodi
- Codice di risposta

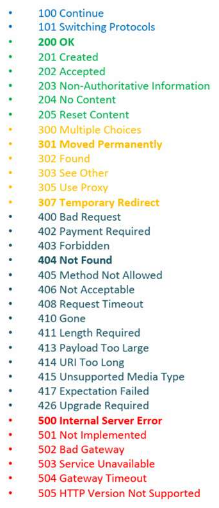


Method	Request Body	Response Body	Idempotent	HTML Forms
GET	No	Yes: resource content	Yes	Yes
HEAD	No	No	Yes	No
POST	Yes: form data or application data	May (usually modification results)	No	Yes
PUT	Yes: application data	May (usually modification results)	Yes	No
DELETE	May	May	Yes	No



Response Status Codes

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

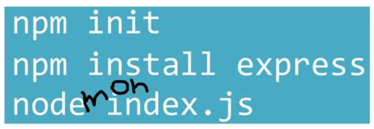


- 100 Continue
- 101 Switching Protocols
- **200 OK**
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 300 Multiple Choices
- **301 Moved Permanently**
- 302 Found
- 303 See Other
- 305 Use Proxy
- **307 Temporary Redirect**
- 400 Bad Request
- 402 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

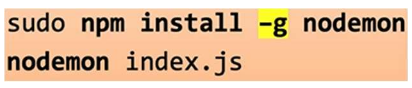
EXPRESS

Framework web in node.

- Una volta lanciato, Il server resta attivo tranne se applicazione crasha o utente fa ctrl C.
- Nel caso in cui si effettua una modifica, bisogna stoppare il server e poi restartarlo. (node ha un modulo che si accorge se un file è stato modificato e nel caso restarta il server)



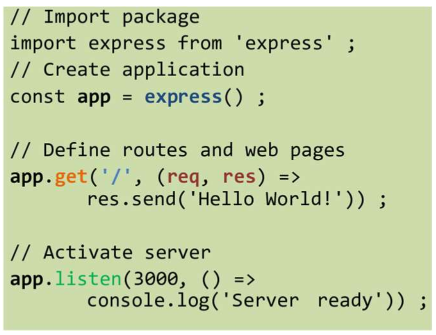
```
npm init
npm install express
nodemon index.js
```



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

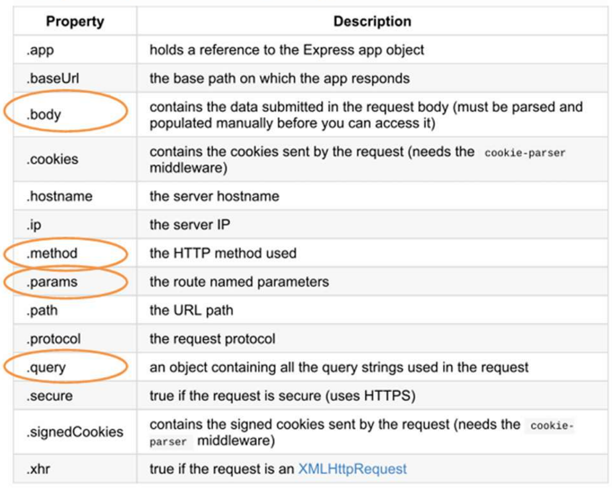
Applicazione express formata da **3 aree**:

- Importa modulo e crea applicazione
  - Import express from ‘express’;
  - Const app =express();
- Configurazione di route, percorsi a cui il server deve fornire una risposta
  - App.get(‘url’, (req, res) => corpo callback)
  - App.method_name(path, handler);
    - Metodi: get, post, put, delete, all, etc
      - all → prova a gestire qualsiasi tipo di richiesta
    - path → path a cui deve rispondere il server
- Attivazione server
  - App.listen(nr_porta, callback)



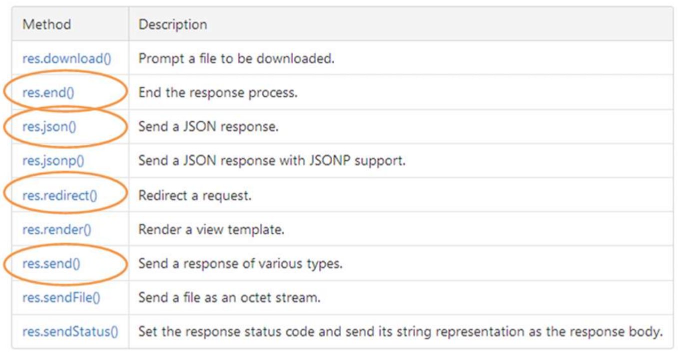
```
// Import package
import express from 'express' ;
// Create application
const app = express() ;

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

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

req (Request object)

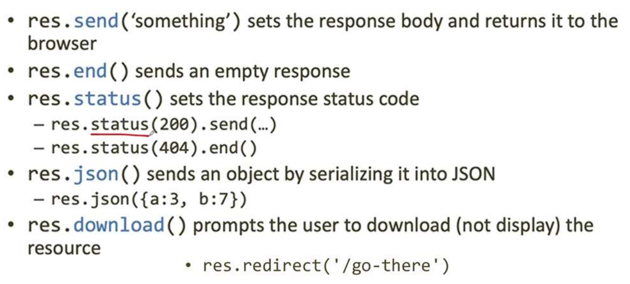


Property	Description
.app	holds a reference to the Express app object
.baseUrl	the base path on which the app responds
.body	contains the data submitted in the request body (must be parsed and populated manually before you can access it)
.cookies	contains the cookies sent by the request (needs the `cookie-parser` 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)



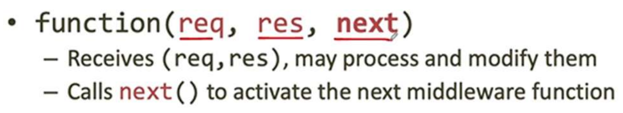
Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
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.



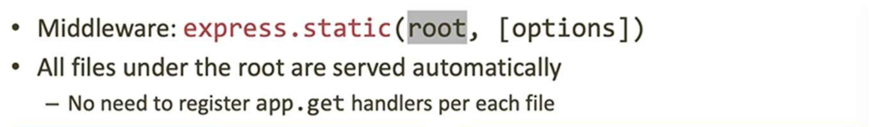
- `res.send`('something') sets the response body and returns it to the browser
- `res.end()` sends an empty response
- `res.status()` sets the response status code
  - `res.status(200).send(…)`
  - `res.status(404).end()`
- `res.json()` sends an object by serializing it into JSON
  - `res.json({a:3,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

MIDDLEWER: funzioni che vengono chiamate ad ogni/certe richiesta che un server riceve.

- app.use(MiddlwareCallback) → attivato per tutte le route
- app.use(path, MiddCallback) → attivato per path specific
- app.method(path, MiddCallback, (req,res) =>{}) → attivato per una specifica route su quel metodo

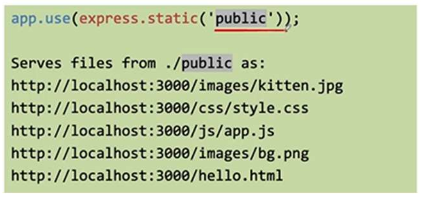


- `function(req, res, next)`
  - Receives `(req,res)`, may process and modify them
  - Calls `next()` to activate the next middleware function

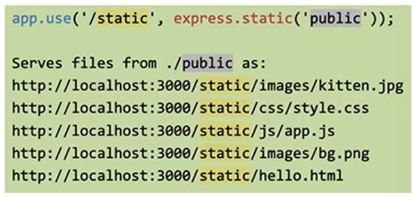


- Middleware: `express.static(root, [options])`
- All files under the root are served automatically
  - No need to register `app.get` handlers per each file

→ per oggetti statici ad esempio

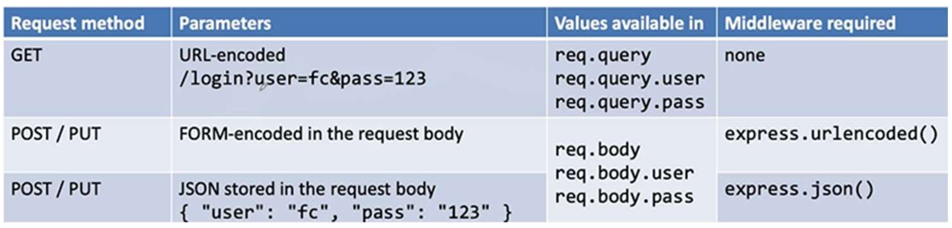


```
app.use(express.static('public'));

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
```



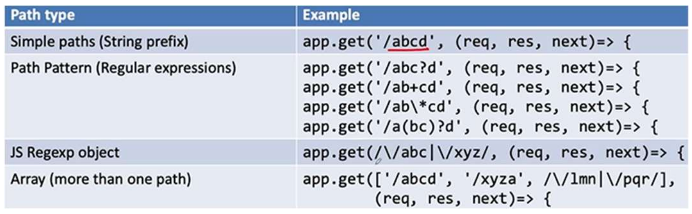
```
app.use('/static', express.static('public'));

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
```

Richieste: usando il middleware json, la stringa viene convertita in una stringa json con le relative proprietà.

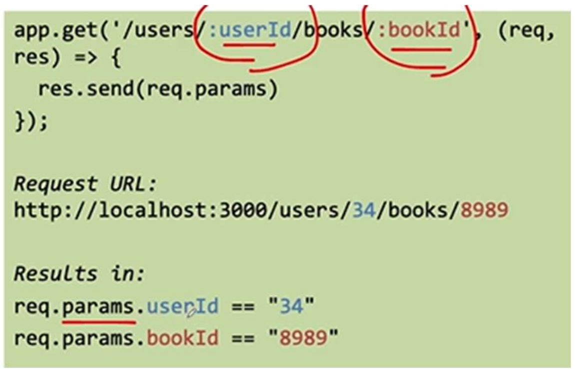


Request method	Parameters	Values available in	Middleware required
GET	URL-encoded `/login?user=fc&pass=123`	`req.query` `req.query.user` `req.query.pass`	none
POST / PUT	FORM-encoded in the request body	`req.body` `req.body.user` `req.body.pass`	`express.urlencoded()`
POST / PUT	JSON stored in the request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

PATH:



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)=> {`



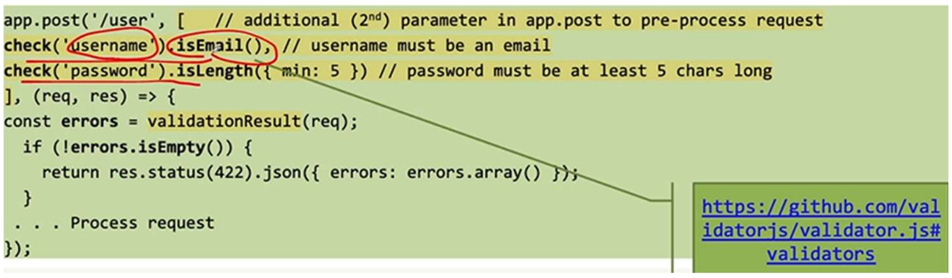
```
app.get('/users/:userId/books/:bookId', (req,
res) => {
  res.send(req.params)
});

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

Results in:
req.params.userId == "34"
req.params.bookId == "8989"
```

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'));`

VALIDATORE DI BODY:

- 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