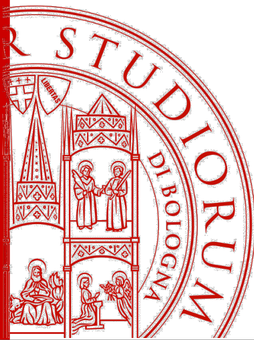
# Introduzione a Express.js

*Angelo Di Iorio*

*Università di Bologna*



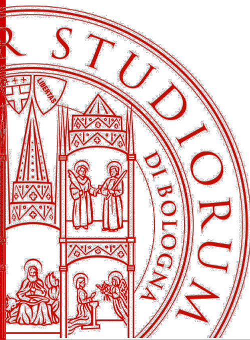
IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



# Applicazioni server-side

- La parte server-side di un'applicazione Web può essere scritta in diversi linguaggi di programmazione e diversi framework, che ne semplificano scritta, manutenzione e deploy
- Ogni soluzione ha le sue specificità ma tutte le applicazioni server-side svolgono alcune operazioni:
  1. Lettura richiesta HTTP, parametri e query string
  2. Autenticazione (se richiesta)
  3. Persistenza dati
  4. Processing dati
  5. Generazione risposta HTTP
     - File statici: immagini, video, audio, fogli di stile, ecc. già disponibili sul server
     - HTML, XML: contenuti pronti per la visualizzazione
     - JSON, XML, CSV, ecc.: dati da processare e visualizzare client-side

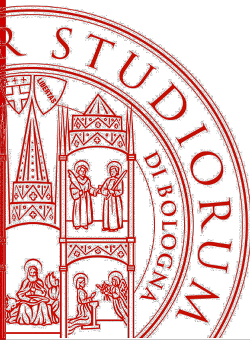

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



# Express.js

- Express è un framework server-side per Node.js
  - Open-source, licenza MIT
  - molto usato e molto supportato dalla comunità
  - molto semplice e molto espandibile con plugin (*middleware*)
- Implementa le funzioni principali di un framework server-side e gestisce richieste e risposte HTTP:
  - routing: associa la richiesta alla funzione che la gestisce
  - lettura richiesta (parametri, query string, header, ecc.)
  - sessioni e autenticazione utenti (tramite middleware)
  - costruzione della risposta (status code, body, headers, ecc.)

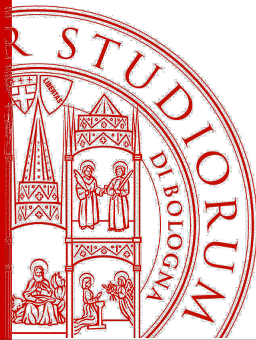

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI




# Basic server

```
express = require("express")
app = express()

docs_handler = function(request, response){
       var docs = {server : "express"}
       response.json(docs);
}

app.get("/docs", docs_handler);

app.listen(8099, function(){
     console.log("\nExpress is running!!! \n")
})
```

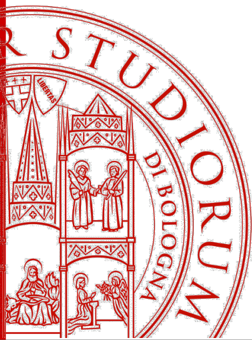

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



# Callback e oggetti res e req

```
processaRichiesta = function(request, response){

        // legge i dati dall'oggetto request
        var name = request.params.name;

        // produce il risultato
        // ad esempio collegandosi a un database
        …

        //scrive il risultato nell'oggetto response
        //in formato JSON
        response.json(docs);
}
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



# Routing

Express fornisce una semplice interfaccia per fare routing:

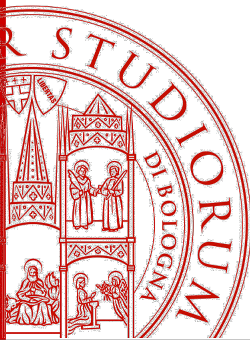
```
app.method(path, function(request, response) { ... })
```

- **method** è uno dei metodi HTTP (*GET*, POST, PUT, ecc.)
- **path** è il *local path* dell'URI richiesto al server
- **function(req, res)** è un handler da eseguire quando viene richiesto il path. I due parametri contengono gli object della richiesta HTTP (uri, intestazioni, parametri, dati, ecc.) e della risposta HTTP in via di restituzione (intestazioni, dati, ecc.)

Ogni route può gestire diversi handler sullo stesso path.

```
app.get('/', function (req, res) {
  res.send('Richiesta GET');
});
```

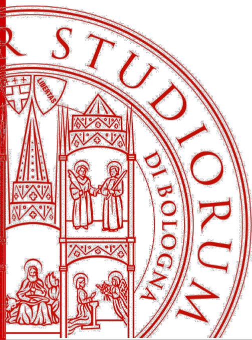

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



# Route paths

- I *route paths* possono essere stringhe o espressioni regolari
- Express.js controlla l'URL della richiesta e se individua un "match" invoca l'handler opportuno
- Esempi
  - `*ab*` : tutti i percorsi che contengono `ab`
  - `abc?de`: `abcde` oppure `abde`
  - `abc+`: `abc`, `abcc`, `abccc`e così via (almeno una c finale)
  - `store$`: path che termina con `store`

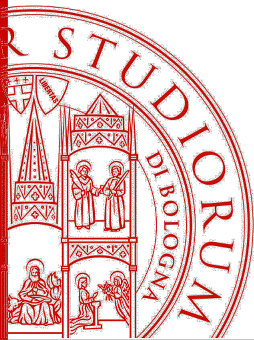

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI




# Route parameters

- I *route paths* possono contenere parametri, ossia frammenti del path a cui è associato un nome che può essere usato per recuperare il valore corrispondente
- Route path: `/clients/:cId/products/:pId`
- Request URL: `/clients/34/products/12`
- `req.params` conterrà `{ "cId": "34", "pId": "12" }`
- `req.params.cId` conterrà `34`
- `req.params.pId` conterrà `12`


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

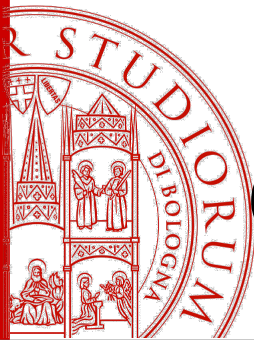


# Esempio routing REST

```
app.get("/clients/", getListaClienti);

app.post("/clients/", aggiungiCliente);

app.put("/clients/:id", creaModificaCliente);

app.get("/clients/:id", getCliente);

app.delete("/clients/:id", cancellaCliente);
```

Nota: l'esempio mostra solo il routing, da completare con le callback


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



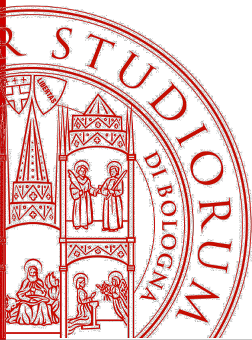

# Oggetti Request e Response

- Le classi `Request` e `Response` rappresentano le richieste e le risposte HTTP ed **espongono i metodi per accedere a tutte le loro informazioni**

- Per le richieste:
  - body, query string, parametri, cookie, header, ecc.
  - https://expressjs.com/en/4x/api.html#res

- Per le risposte:
  - Metodi per spedire lo status code e la risposta in diversi formati
  - Metodi per aggiungere gli header
  - https://expressjs.com/en/4x/api.html#req


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

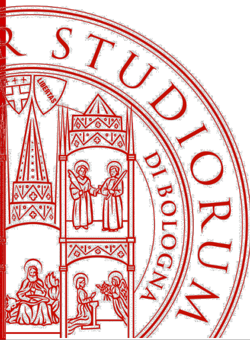


# Spedire una risposta

| 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. |

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


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



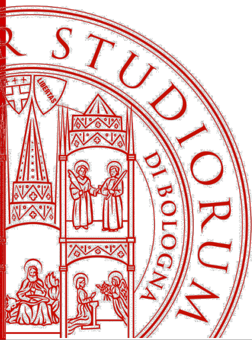
# Middleware in Express

- Express ha pochissime funzionalità proprie (routing), ma utilizza un grande numero di librerie middleware personalizzabili in uno stack di servizi progressivamente più complessi.
- Per aggiungere un middleware allo stack:

```
app.use(<middleware>)
```

- **Un'applicazione Express è allora essenzialmente una sequenza di chiamate a funzioni di middleware tra la richiesta e la risposta.**
- Il middleware può:
  - accedere agli oggetti di richiesta e risposta
  - cambiarli ed eseguire del codice di modifica
  - chiamare la prossima funzione del middleware (`next()` )
  - uscire dal ciclo e mandare la risposta.


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

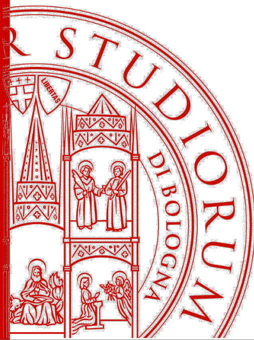


# File statici in Express.js

- Express.js può essere usato anche per restituire file statici memorizzati sul server
- Questa funzionalità si realizza aggiungendo un apposito middleware, che associa ad un path la directory in cui recuperare i file
- E' possibile specificare path e directory diverse

```
express = require("express")
app = express()

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


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI




# Accedere ai dati di un POST

- Appositi middleware sono usati anche per processare dati spediti via POST
- La libreria bodyparser, ad esempio, permette di accedere al corpo dei dati spediti dal POST nel body della richiesta.
- Questi middleware si occupano di recuperare e elaborare i dati e li convertono in oggetti accedibili dall'applicazione.
- I dati del POST di un form si accedono come:

```
<request>.body.<post_variable>
```

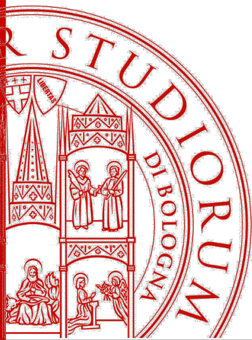
- Come negli altri casi, è necessario include il modulo bodyparser e aggiungere il middleware:

```
bodyparser = require('body-parser');
app.use(bodyparser.urlencoded({ extended: true }));
```

- Lo stesso middleware può essere istanziato per processare dati in JSON:

```
app.use(bodyparser.json())
```

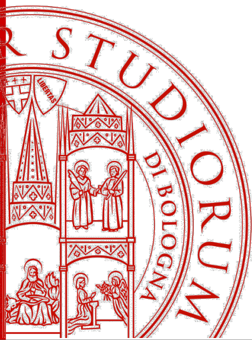

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



# POST data

```
var express = require("express")
var bodyParser = require('body-parser');

app = express()

app.use(bodyparser.urlencoded({ extended: true }));

app.post('/login',function(request,response){
  console.log("Username:" + request.body.user)
  console.log("Password:" + request.body.pwd)

 // Continue and do authentication….
});
```

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

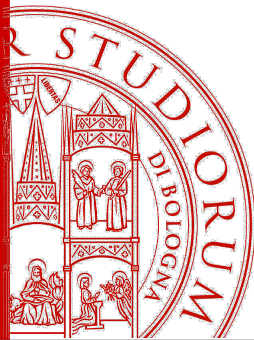


# Rounting modulare

- La class `express.Router` permette di creare *route handlers modulari* e che possono essere combinati
- Anche in questo caso si aggiungere un middleware responsabile di gestire il routing per un insieme di path

```
var express = require('express')
var router = express.Router()

router.get('/', function (req, res) {
  res.send("TODO. Elenco utenti")})

router.get('/:name/', function (req, res) {
 res.send("TODO. Utente " + req.params.name)})

module.exports = router
```

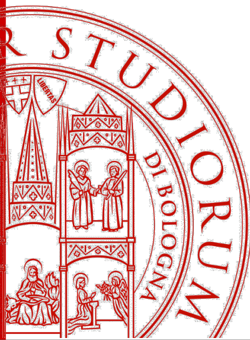
```
usersRouter = require('./usersRouter.js');
app.use("/users/", usersRouter);
```



# Express.js e autenticazione

- In Express.js l'autenticazione è realizzata con appositi middleware
- Come negli altri casi, devono essere installati, inclusi (con `require`) e aggiunti alla propria applicazione (con `use`)
- Uno dei più usati è Passport.js:
  - **http://passportjs.org/**
  - Flessibile e modulare
  - Supporta diverse strategie di autenticazione tra cui HTTP basic e digest, e JWT
- Per aggiungere il supporto a JWT invece si usa il pacchetto **`express-jwt`**:
  - **https://www.npmjs.com/package/express-jwt**


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



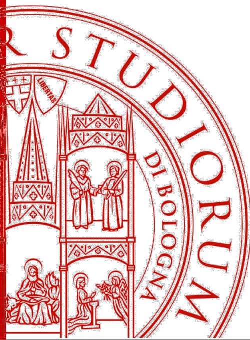
# Express.js e CORS

- In Express.js si usa il middleware *cors* per aggiungere gli header `Origin` e `Access-Control-Allow-Origin` e supportare CORS
- Necessario includere il modulo e aggiungere il middleware
- Permette di abilitare le successive richieste Ajax su tutti i domìni o su domìni specifici e di specificare altre opzioni attraverso oggetti JS passati in input

```
cors = require('cors');
app.use(cors())
app.options('*', cors())
```

- Pacchetto npm: `https://www.npmjs.com/package/cors`


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI



# Conclusioni

- Oggi abbiamo visto le caratteristiche principali di Express.js
- Il framework include anche un motore di template per generare viste HTML a partire da contenuti testuali
- Molte altre funzionalità possono essere aggiunte tramite middleware
- Esempi d'uso di questi middleware nella documentazione ufficiale:https://expressjs.com/en/starter/examples.html


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI