JavaScript Node.js

Web applications

Url: https://github.com/obiSerra/laboratorio-nodejs

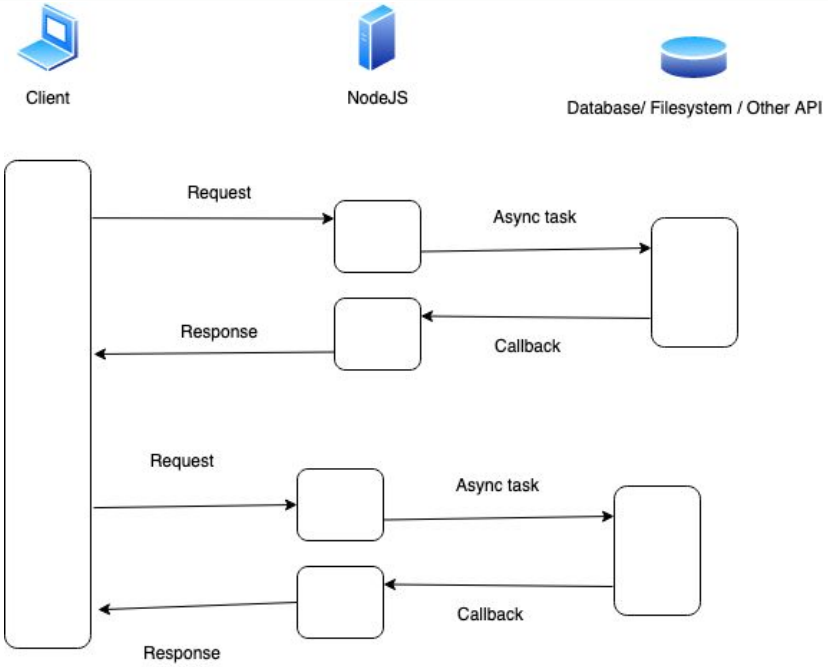
Riassunto della lezione precedente

- Async programming
- Callbacks
- Events
- Promise
- Async/Await

Webserver con Node

- Esistono diverse librerie e framework per creare un web server con Node, quella piu' utulizzata e' *Express (https://expressjs.com/)*
- Uno dei vantaggi principali di Express e' la sua semplicita' di utilizzo
- L'architettura di Express si basa su una serie di *middleware* che si "passano" la *request* fino a quando uno di questi non invia la risposta

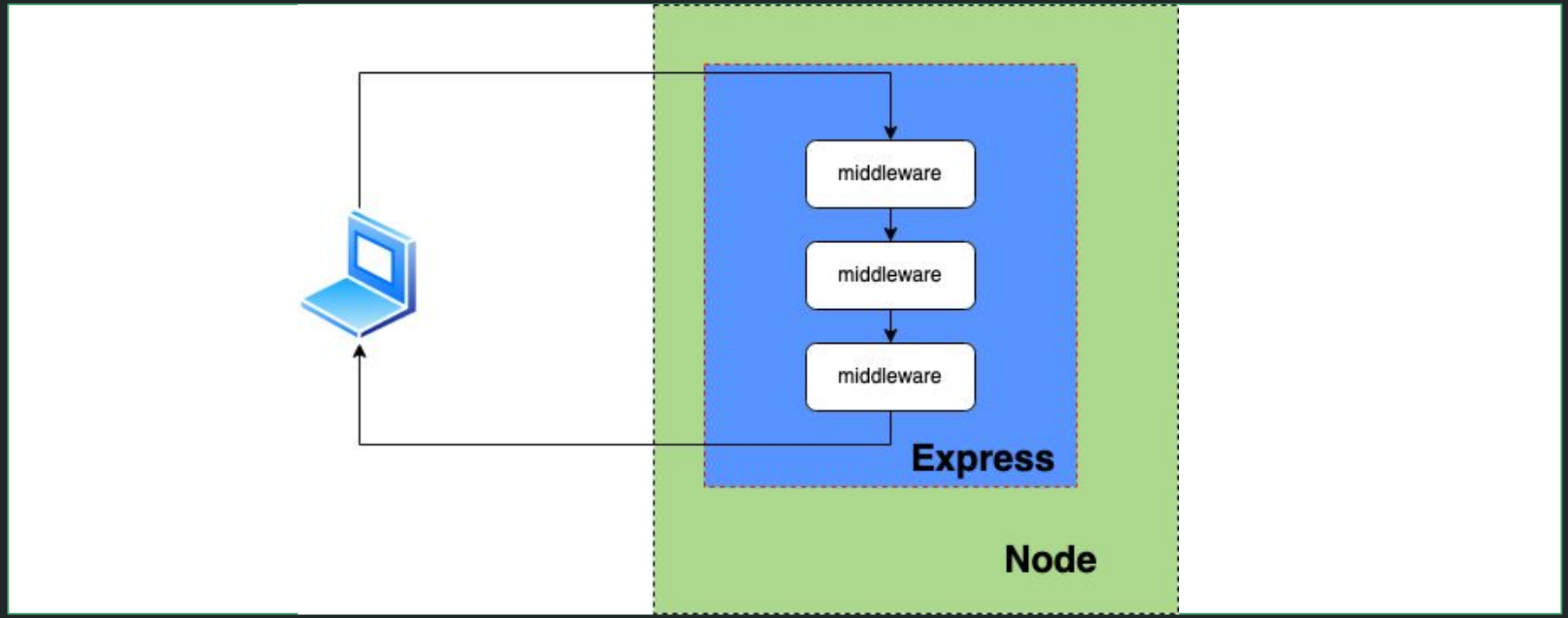
NodeJs as Web Server



Hello World Express

- Creiamo un semplice progetto con Express e un solo file server.js
- Tutti quelli dichiarati con *app.use*, *app.get* ... sono middleware
- *req* contiene la request, *res* contiene la risposta da inviare; *next* serve per passare la chiamata al *middleware* successivo
- Proviamo ad aggiungere un semplice middleware di log

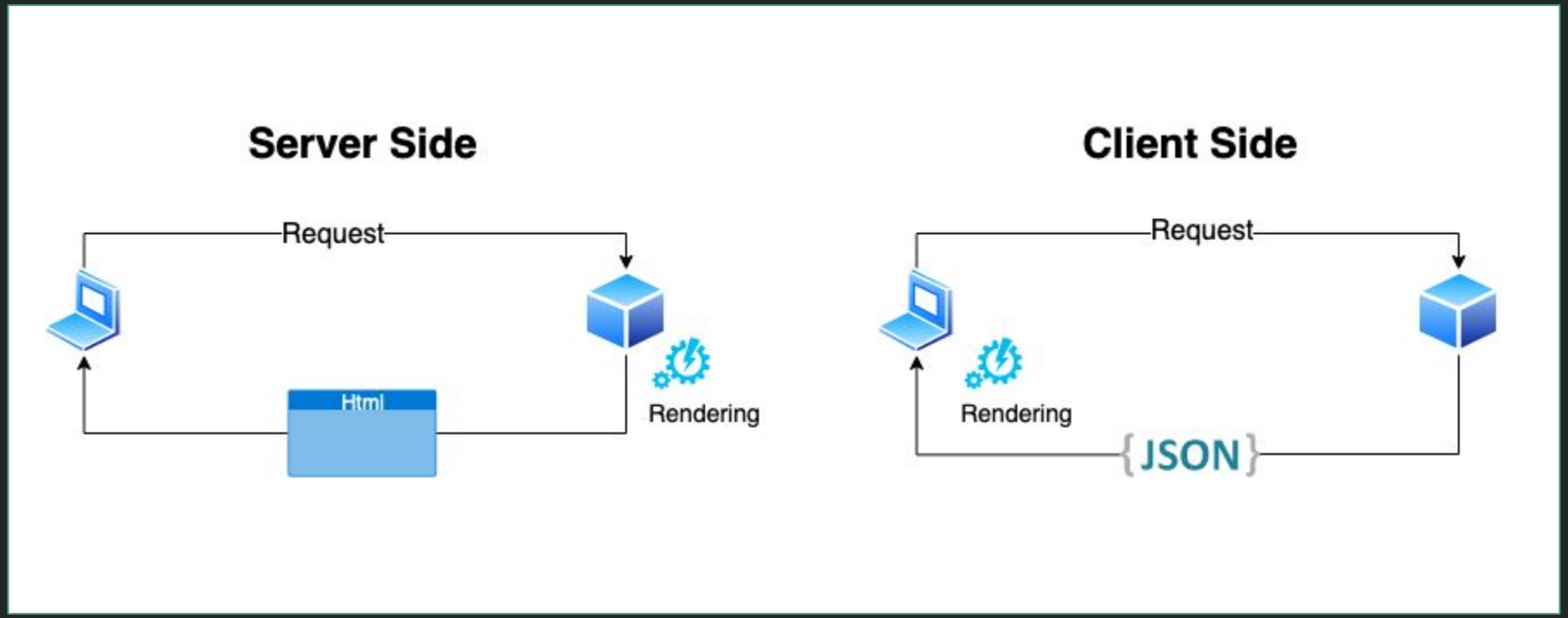
Express



Server Side Rendering vs API

- Un server puo’ con una pagina Html o con una struttura dati (generalmente un JSON o un XML)
- Se il server “costruisce” e restituisce una pagina, parliamo di Server Side Rendering
- Se il server restituisce solamente dei dati, parliamo di API
- Esistono diversi tipi di API: REST e’ il piu’ comune, ma ci anche SOAP e GraphQL

Server Side vs Client Side Rendering



Il nostro primo (vero) web server

- Creiamo un nuovo progetto
- Installiamo express con *$ npm install express --save*
- Per fare un semplice server con Express bastano poche righe di JS
- Se invece vogliamo fare qualcosa di piu' complicato possiamo usare un *generator*
- Lanciamo *$ npx express-generator*
- Possiamo andare a vedere i file che sono stati generati
- Lanciando *$ npm install* e *$ npm start* possiamo far partire il server

Routing in Express WIP

- *ref lezione-3/hello-world-exp/routes/index.js*
- Routing delle risorse statiche con *app.use(express.static(...))*
- Gli endpoint vengono dichiarati con *router.get router.post*
- E’ possibile spezzare le route in sotto-route
- I path parameters vanno dichiarati nella route
- I query parameters invece non fanno parte della route e possono essere direttamente letti

Template Engine

- Express supporta diversi template engine
- Un template engine trasforma un DSL in Html
- Jade e’ il template engine di default
- E’ basato su un struttura gerarchica ti template

Troppa teoria

- aggiungiamo 2 pagine:
  - /users/ che restituisce la lista degli utenti (finta)
  - /users/:id che restituisce lo user con quell'id (dalla finta lista di utenti)
- entrambe le pagine saranno visibili solamente se l'utente avra' il corretto token nell'url (?token=123)
- entrambe le pagine utilizzeranno Jade per renderizzare i risultati

REST API

- Express permette anche di sviluppare API REST
- Installiamo Postman (https://www.postman.com/) per facilitare lo sviluppo
- Creiamo una nuova applicazione Express con *$ npx express-generator --no-view*
- Possiamo cancellare i file che non ci servono
- Per effettuare risposte in JSON, ci basta user *res.json()*

REST API 2

- Creiamo una nuova collection in Postman
- Creiamo una semplice GET request
- Aggiungiamo un *query parameter*
- Creiamo una semplice POST

Url: https://github.com/obiSerra/laboratorio-nodejs

Login

- Creiamo una nuova API per l’autenticazione PUT */login*
- Questo endpoint aspettera’ un body json con email e password e verifichera’ la validita’ dei dati
- Possiamo usare il pacchetto *crypto* per creare un *digest* della password da salvare da confrontare con la password fornita durante il login
- Usiamo il pacchetto npm *dotenv* (https://www.npmjs.com/package/dotenv) per gestire le configurazioni

Autenticazione con JWT

- JSON Web Token (https://jwt.io/) e' un modo sicuro e comodo per gestire l'autenticazione degli utenti
- Si basa sulla generazione di un Token firmato , contenente tutte le informazioni necessarie all'autenticazione, che verra' poi validato dal server
- *jsonwebtoken* e' un buon pacchetto per gestire i JWT in Node
- Aggiungiamo la generazione e la verifica di un token JWT
- Usando un middleware, possiamo facilmente applicare l'autenticazione a diverse route

Database

- Installiamo MongoDb (https://docs.mongodb.com/manual/administration/install-community/) e NoSqlBooster (https://www.nosqlbooster.com/downloads/)
- MongoDb e' NoSql Database
- Semplice da usare, gestisce dati in formato JSON
- Non essendo un DB relazionale non e' molto performante per alcune tipologie di query
- Dopo aver installato Mongodb possiamo creare una nuova directory per il db e lanciare *$ mongod --dbpath db*

MongoDb e Node

- Per prima cosa dobbiamo installare i driver per Node *$ npm install mongodb --save*
- Creiamo un modulo per gestire le connessioni con il DB
- Adesso possiamo spostare la gestione degli utenti nel DB