Seminar 6

Module, dependențe, REST & Express.js

❖ Package.json

Package.json este:

- Un fișier JSON care se află în directorul rădăcină al proiectului tău;
- Locul central pentru a configura și descrie cum să interacționeze și să ruleze aplicația;
- Utilizat de către CLI-ul npm (și yarn) pentru a identifica proiectul și pentru a înțelege cum să gestioneze dependențele proiectului;
- Fișierul care îi permite CLI-ul npm să pornească proiectul, să ruleze scripturi, să instaleze dependențele, să publice în registrele NPM și să îndeplinească multe alte sarcini utile;

Un exemplu de package.json:

```
{
      "name": "my-project",
      "version": "1.5.0",
      "description": "Express server project using compression",
      "main": "src/index.js",
      "scripts": {
          "start": "node index.js",
          "dev": "nodemon",
          "lint": "eslint **/*.js"
      },
      "dependencies": {
          "express": "^4.16.4",
          "compression": "~1.7.4"
      },
      "devDependencies": {
          "eslint": "^5.16.0",
          "nodemon": "^1.18.11"
      },
      "repository": {
          "type": "git",
          "url": "https://github.com/osiolabs/example.git"
      },
      "author": "Jon Church",
      "contributors": [{
          "name": "Amber Matz",
          "email": "example@example.com",
          "url": "https://www.osiolabs.com/#team"
      }],
      "keywords": ["server", "osiolabs", "express", "compression"]
  }
```

Pași necesari pentru crearea unui fișier package.json:

1. Deschidem directorul rădăcină al proiectului în editorul de cod sau în terminalul liniei de comandă.
2. Putem să creăm fișierul manual cu numele **‘package.json’** sau să rulăm în terminal comanda **‘npm init’**.
3. Dacă preferăm să folosim comanda ‘npm init’, atunci această comandă ne va solicita o serie de întrebări pentru a configura fișierul package.json. Putem răspunde la aceste întrebări pentru a furniza detalii despre proiectul, cum ar fi numele, versiunea, descrierea, punctul de intrare, comanda de testare și altele.

Pentru mai multe detalii legate de ce este un package.json: https://heynode.com/tutorial/what-packagejson/

❖ Module

Modulele reprezintă unități de cod care încapsulează funcționalități și variabile legate de un anumit aspect al unei aplicații sau proiect. Ele permit organizarea și separarea codului în părți mai mici și mai gestionabile. Modulele au devenit o parte esențială a dezvoltării JavaScript modernă și oferă o serie de avantaje:

1. **Încapsulare:** Modulele permit încapsularea datelor și funcționalității, astfel încât să poți ascunde detalii interne și să expui doar ceea ce este necesar pentru a fi utilizat în afara modulului. Acest lucru promovează principiul informației private și securității.
2. **Reutilizare:** Poți utiliza modulele în mai multe locuri ale aplicației sau în diferite proiecte. Acest lucru duce la reutilizarea codului și economisirea timpului de dezvoltare.
3. **Mentenanță ușoară:** Modulele permit dezvoltatorilor să se concentreze pe părți specifice ale aplicației fără a fi nevoie să se ocupe de întregul cod. Acest lucru face mentenanța mai ușoară și reduce riscul de erori.

În JavaScript, există două sisteme majore de module:

1. **CommonJS:** Acest sistem este utilizat în principal în medii server-side, precum Node.js. Acesta folosește sintaxa **‘require’** pentru a importa module și **‘module.exports’** sau **‘exports’** pentru a exporta funcționalitate.
2. **ES Modules (ESM):** Acest sistem este standardizat prin specificația ECMAScript 2015 (ES6) și este utilizat în principal în mediile browser. Sintaxa constă în utilizarea **‘import’** pentru a importa module și **‘export’** pentru a exporta funcționalitate.

Specificați în interiorul fișierului package.json steag-ul:

```
{
    "type": "module"
}
```

De reținut: Dacă alegem să lucrăm cu ES Modules, trebuie să avem grijă să specificăm în mod explicit extensia atunci când importăm un fișier.

❖ NPM (Node Package Manager)

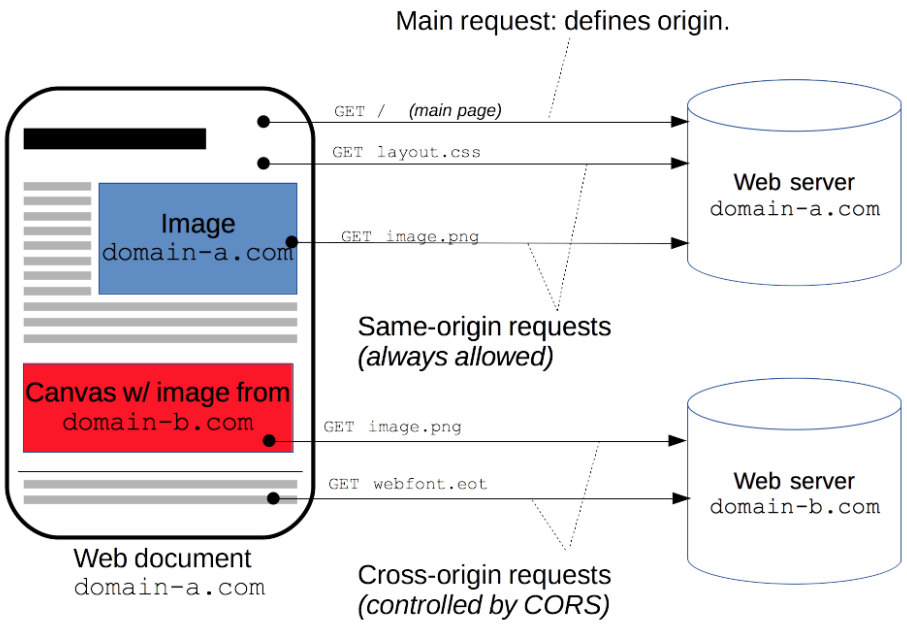
NPM (Node Package Manager) este un instrument software folosit pentru gestionarea pachetelor JavaScript în ecosistemul Node.js. Acesta îndeplinește mai multe roluri esențiale în dezvoltarea JavaScript, inclusiv:

1. **Gestionarea dependențelor:** NPM permite dezvoltatorilor să definească, să instaleze și să actualizeze dependențele proiectelor lor. Aceste dependențe pot fi biblioteci sau module JavaScript dezvoltate de alții și utilizate pentru a extinde funcționalitatea aplicațiilor.
2. **Publicarea și distribuirea pachetelor:** Dezvoltatorii pot publica propriile pachete sau module JavaScript în Registrul NPM (NPM Registry) pentru a împărtăși cu ceilalți și pentru a face pachetele disponibile pentru a fi utilizate în alte proiecte. Astfel, NPM facilitează partajarea codului sursă și colaborarea între dezvoltatori.
3. **Gestionarea scripturilor:** NPM permite definirea și rularea scripturilor personalizate asociate cu proiectul. Aceste scripturi pot include comenzi pentru testare, construire, pornirea serverelor, sau orice alte acțiuni necesare în dezvoltarea și gestionarea proiectului.
4. **Rezolvarea și gestionarea versiunilor:** NPM se ocupă de gestionarea versiunilor pentru pachetele JavaScript, asigurându-se că proiectele utilizează versiuni compatibile ale dependențelor lor. Acest lucru contribuie la menținerea stabilității și consistenței proiectelor.

❖ CORS (Cross-Origin Resource Sharing)

Este o tehnologie web care permite aplicațiilor web să facă cereri de resurse (de exemplu, fișiere, date sau servicii) către un alt domeniu (o altă locație a serverului) decât cel de pe care provine pagina web.

De obicei, paginile web sunt restricționate să facă cereri doar către resursele din același domeniu din motive de securitate. CORS îmbunătățește această restricție pentru a permite comunicația între domenii diferite.



Câteva aspecte importante despre CORS:

1. **Politica de Same-Origin:** Fără CORS, într-un browser web, o pagină web care se încarcă de pe un domeniu (de exemplu, example.com) nu poate face cereri la un alt domeniu (de exemplu, api.example-api.com) din motive de securitate, conform politicii de "same-origin".
2. **Cereri cu CORS:** Atunci când un server web dorește să permită paginii web de pe un alt domeniu să facă cereri către el, trebuie să specifice în antetul HTTP că permite CORS. Acest lucru se face prin adăugarea unor antete precum Access-Control-Allow-Origin pentru a specifica domeniile permise.
3. **Cereri prefligate:** Anumite cereri, cum ar fi cele care implică cereri AJAX cu metode HTTP complexe (de exemplu, PUT sau DELETE) sau cu capete personalizate, necesită o cerere prealabilă (cerere de opțiuni prefligate - "preflight request") pentru a verifica dacă serverul acceptă cererile CORS. Serverul va răspunde cu un antet Access-Control-Allow-Origin pentru a indica permisiunile.
4. **Securitate:** Utilizarea CORS este esențială pentru a preveni atacuri cum ar fi Cross-Site Request Forgery (CSRF) și Cross-Site Scripting (XSS). Prin configurarea corectă a politicilor CORS, serverele pot permite doar cereri de la domenii de încredere.
5. **Exemplu de utilizare:** CORS este des utilizat în dezvoltarea aplicațiilor web pentru a face cereri AJAX sau fetch la API-uri de la un domeniu diferit sau pentru a include resurse, cum ar fi fonturi sau imagini, din alte surse.

❖ REST

Un **API REST** (cunoscut și sub denumirea de API RESTful) este o interfață de programare a aplicației (API sau web API) care se conformează restricțiilor stilului arhitectural REST și permite interacțiunea cu servicii web RESTful.

REST reprezintă transfer de date reprezentativ, deoarece transferă o reprezentare a stării resursei către solicitant sau punct final.

REST folosește protocolul HTTP pentru a accesa și utiliza datele. Acele date pot fi folosite pentru a obține (GET), actualiza (PUT), crea (POST) și șterge (DELETE) tipuri de date, ceea ce se referă la operațiile de citire, actualizare, creare și ștergere a resurselor.

Pentru ca un API să fie considerat RESTful, trebuie să se conformeze acestor criterii:

1. O arhitectură client-server compusă din clienți, servere și resurse, cu cereri gestionate prin HTTP.
2. Comunicare client-server în care să nu se stocheze informații despre client între cereri GET, unde fiecare cerere este separată și fără legătură.
3. Date cacheabile care optimizează interacțiunile client-server. Practic, să avem posibilitatea de a stoca temporar datele pe client sau pe server pentru a îmbunătăți eficiența și performanța comunicării dintre client și server.
4. O interfață uniformă între componente astfel încât informațiile să fie transferate într-o formă standard. Acest principiu se referă la faptul că în arhitectura REST, toate resursele sunt accesate și modificate folosind un set comun de operații standard, indiferent de resursa specifică. Prin folosirea unui set uniform de operații, REST face ca interacțiunile dintre componente să fie consistente și previzibile, ceea ce facilitează dezvoltarea și utilizarea serviciilor web.

Pentru mai multe detalii legate despre ce ste un API RESTful:
https://www.redhat.com/en/topics/api/what-is-a-rest-api

❖ Node.js

Node.js este un mediu de execuție open-source, server-side, care permite dezvoltatorilor să ruleze cod JavaScript pe server. Acesta este conceput pentru a fi eficient și scalabil, ceea ce îl face potrivit pentru construirea de aplicații de rețea și servere web. Node.js este construit pe motorul JavaScript V8, același motor care alimentează browserul Google Chrome, ceea ce înseamnă că poate executa cod JavaScript foarte rapid.

```
// Load HTTP module
const http = require("http");

const hostname = "127.0.0.1";
const port = 8000;

// Create HTTP server
const server = http.createServer((req, res) => {

   // Set the response HTTP header with HTTP status and Content type
   res.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body "Hello World"
   res.end('Hello World\n');
});

// Prints a log once the server starts listening
server.listen(port, hostname, () => {
   console.log(`Server running at http://${hostname}:${port}/`);
})
```

Pentru crearea unei comenzi custom prin care să folosim node.js, mergem în package.json și adăugăm următoarea linie:

```
  "scripts": {
    "dev": "node index.js"
  },
```

Astfel, de fiecare dată când o să dorim să pornim serverul, nu o să mai rulăm ‘node index.js’ ci o să scriem ‘npm run dev’ prin care o să folosim npm pentru a porni serverul.

❖ Express.js

Express.js este un framework web pentru **Node.js**, conceput pentru a simplifica dezvoltarea aplicațiilor web și API-urilor. Express.js oferă o serie de funcționalități și un set de instrumente pentru a construi cu ușurință aplicații web și servicii web rapide și eficiente cu Node.js.

Câteva caracteristici și funcționalități cheie ale Express.js:

1. **Routing simplificat:** Express.js facilitează definirea și gestionarea rutelor pentru aplicații. Putem defini rute pentru diferite URL-uri și metode HTTP, ceea ce face gestionarea cererilor și a resurselor foarte convenabilă.
2. **Middleware:** Express.js utilizează conceptul de middleware pentru a procesa cererile HTTP. Acest lucru permite adăugarea de funcționalități intermediare pentru a manipula cererile înainte de a ajunge la rutele finale. Middleware-ul poate fi utilizat pentru autentificare, logare, manipularea datelor din cerere și multe altele.
3. **Integrare cu baze de date:** ne putem integra cu diferite baze de date, precum MongoDB, MySQL, PostgreSQL, prin intermediul modulelor și bibliotecilor disponibile. Acest lucru facilitează interacțiunea cu bazele de date în aplicații web.
4. **Extensibilitate:** Express.js este un framework extensibil, ceea ce înseamnă că puteți adăuga funcționalități suplimentare prin intermediul modulelor terțe sau personalizate.

Exemplu de server:

```
const express = require('express');
const app = express();

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000);
```

Req.params vs req.query

1. **Req.params:** Acest obiect este folosit pentru a accesa valorile parametrilor de rută din URL. Parametrii de rută sunt părți dintr-un URL care sunt definite ca variabile și care de obicei încep cu caracterul **‘:’**, și sunt specificate în definirea rutei în Express.js. De exemplu, în ruta **‘/utilizator/:id’**, **’:id’** este un parametru de rută. Pentru a accesa valoarea acestui parametru în codul Express, trebuie să utilizăm req.params:

```
app.get('/utilizator/:id', (req, res) => {
  const userId = req.params.id; // Accesează valoarea parametrului de rută "id"
  res.send(`ID utilizator: ${userId}`);
});
```

2. **Req.query:** Acest obiect este folosit pentru a accesa datele primite dintr-o cerere HTTP ca parametri de interogare (query parameters). Parametrii de interogare sunt perechi cheie-valoare care apar în URL-ul cererii după caracterul **'?'**. De exemplu, în URL-ul **'/resursa?pagina=1&limita=10'**, pagina și limita sunt parametri de interogare. Pentru a accesa aceste parametri în codul Express, utilizăm req.query:

```
app.get('/resursa', (req, res) => {
  const pagina = req.query.pagina; // Accesează valoarea parametrului de
interogare "pagina"
  const limita = req.query.limita; // Accesează valoarea parametrului de
interogare "limita"
  res.send(`Pagina: ${pagina}, Limita: ${limita}`);
});
```

❖ Nodemon

Nodemon este un utilitar pentru dezvoltatori în limbajul Node.js care facilitează dezvoltarea aplicațiilor. Nodemon monitorizează fișierele dintr-un proiect Node.js și repornește automat aplicația atunci când detectează modificări în codul sursă. Aceasta îmbunătățește semnificativ fluxul de lucru al dezvoltatorilor, deoarece nu mai este necesară repornirea manuală a serverului sau a aplicației Node.js de fiecare dată când se fac modificări în cod.

Comandă instalare nodemon: **npm install nodemon**

Următorul pas este să mergem din nou în package.json și să modificăm din 'node index.js' în 'nodemon index.js':

```
"scripts": {
    "dev": "nodemon index.js"
  },
```

Când o să pornim serverul, o să primim acest mesaj în consolă care se va actualiza de fiecare dată când o să modificăm ceva în cod:

```
[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
```