



FORMATION FULL STACK

הרווחה, משרד העבודה
והשירותים החברתיים



Node.js

By Charles Cohen



Charles Cohen
Senior Full Stack Developer
charlesc@edandweb.co.il
050-4449764



Programme du cours Node.js

1. Introduction to Node.js
 - What is Node and what is it not ?
 - Node.js Features ?
 - Our first Node.js script: Hello World
 - Building a web server in Node.js
 - Debugging node applications
2. Building your Stack
 - Pulling in other libraries
 - Building custom libraries
 - A-synchronicity and callbacks
 - Blocking vs. non-blocking I/O
 - Working within the event loop
3. Modular JavaScript with Node.js
 - Writing Modular JavaScript with Node.js
 - Core Modules
 - Installing Packages
 - Publishing packages
4. Avoiding common pitfalls with Async.js
 - Introducing the Asynchronous problem
 - Async.js Library to the rescue
 - Collections
 - Flow Controllers
5. Working with the file system
 - Files manipulations
 - Folder manipulations
 - Putting the file-system module together Async.js
6. Building Web applications with the Express Framework
 - Introduction to Express, installation and basic setup
 - Application configuration
 - Routing
 - Views and Templating options
 - Persistence with Cookies, In-Memory Sessions and session-stores
 - Social Authentication with Passport.js
7. Connecting MySQL Server
 - Database connection
 - A-synchronicity Queries from node.js



Cours 5

Express Framework



Plan du module Node.js

Cours	Date	Cours
1	Lun. 20/05	<ul style="list-style-type: none">• Introduction to Node.js (1)
2	Mer. 29/05	<ul style="list-style-type: none">• Building your Stack (2)• Command Line• File System• Arrow Functions• Modular JavaScript with Node.js
3	Mer. 05/06	<ul style="list-style-type: none">• Asynchronous JS• Consuming API – HTTP requests
4	Lun. 10/06	<ul style="list-style-type: none">• Consuming API – HTTP requests• Exercises – Notes & Weather
5	Lun. 17/06	<ul style="list-style-type: none">• Express framework – templates with Handlebars• Building a get API entry point based on previous exercices (get coordinates and weather from mapbox.com and darksky.net) and returning data as JSON• Implement a search address box that consuming the GET API
6	Lun. 24/06	<ul style="list-style-type: none">• Promise, await, async, Async module ...• Avoiding common pitfalls with Async.js
7	Lun. 01/07	<ul style="list-style-type: none">• Working with MongoDB 1/2
8	Lun 08/07	<ul style="list-style-type: none">• Working with MongoDB 1/2



Environnement de travail

Editeur de code:

- Visual Studio Code
 - Extensions:

Liens Utiles:

- Node.js: <https://nodejs.org/>
- Moteur V8 Javascript: <https://v8.dev/>
- Express : <http://expressjs.com>



Questions/Sujets à enrichir

- I/O Node.js
- Moteur JS Edge, Firefox ?
Firefox: [GECKO engine written in C++](#)
Edge: [Originally built with Microsoft's own EdgeHTML and Chakra engines, Edge is currently being rebuilt as a Chromium-based browser,\[10\]\[11\] using the Blink and V8 engines, based upon WebKit. As part of this big change, Microsoft intends to add support for Windows 7, 8, 8.1, and macOS.\[12\]](#)
- Sleep Node.js ?



Résumé du cours 4

- Utilisation de l'API mapbox: récupération de la longitude et latitude avec une adresse
- Combinaison des fonctions getCoordinates et getGeocode



Résumé du cours 3

- Correction de l'exercice sur les notes
- Correction de l'exercice sur les méthodes (function au sein d'un objet) avec la forme arrow et utilisation de filter
- Code synchrone: execution ligne par ligne du code
- Code asynchrone: execution avec callback asynchrone (event loop and callback Queue). Exemple avec SetTimeout et setInterval.
- Utilisation de la librairie request pour envoyer des requêtes GET sur des API. Paramètre json true
- Utilisation de l'API darksky.net (currently.temperature, currently.precipProbability) et options celsius et francais.



Résumé du cours 2

- Utilisation de la librairie chalk pour coloriser les messages dans la console
- Découverte du packet nodemon (lancement en continu d'un script)
- Installation d'un module en mode globale `npm install -g nodemon` (pas d'impact sur package.json de notre projet)
- Ligne de commande (`process.argv`), affichage argument (`process.argv [2]`)-
- Yargs: ligne de commande: `yargs.command`, `command`, `describe`, `handler`, `builder`, `describe`, `demandOption`, `type`
- Gestion des erreurs: `try { } catch (e) {}`
- `JSON.parse`, `JSON.stringify`
- `writeFileSync`, `readFileSync`
- Arrow functions: `(x) => { }`



Résumé du cours 1

- Node.js is a Javascript **runtime** built on [Chrome's V8](#) Javascript engine.
- Node.js uses an **event-driven, non-blocking**
- Node.js' package ecosystem, **npm**, is the largest ecosystem of open source libraries
- Node.js => JS Code => V8 (C++) => Result
- Le nom de l'objet global est **global** et l'objet équivalent à Document dans le browser est **process**
- Pour utiliser la librairie des fichiers (FileSystem) on utilisera la commande `const fs = require('fs');`
- Pour écrire on utilisera la méthode `WriteFileSync`
- Système de modules de Node.js (FileSystem ...), [API: https://nodejs.org/dist/latest-v10.x/docs/api/](https://nodejs.org/dist/latest-v10.x/docs/api/)
- Nos scripts `require('./utils.js');` ➔ `module.exports`
- `npm init`
- Packets npm (`npm install validator`)
- Projet existant: `npm install`



Fin de l'application Weather

TP

En récupérant le résultat de la longitude et de la latitude, appeler la fonction forecast et obtenez la météo pour l'adresse demandé.

Essayez votre script sur différentes adresses.

Utiliser async et await

```
add(1, 4, (sum) => {  
    console.log(sum);  
});
```

Ecrire une fonction `add` pour que l'appelle à cette fonction ci-dessus retourne la somme après 2 secondes.

Pour cela, il faudra analyser la structure de la fonction (nombre de paramètre ...). Le 3^{ème} paramètre est une fonction de callback c'est celle-ci qu'il devra être appelé après les 2 secondes.

Utiliser `setTimeout` avec un timer de 2s pour que cette fonction puisse afficher la somme après 2s.

```
const add = (a, b, callback) => {  
  setTimeout(() => {  
    callback(a + b)  
  }, 2000);  
};  
  
add(1, 4, (sum) => {  
  console.log(sum);  
});
```

Express Framework

<http://expressjs.com>

Express 4.16.4

Fast, unopinionated, minimalist
web framework for **Node.js**

```
$ npm install express --save
```

Web Applications

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Frameworks

Many popular frameworks are based on Express.

APIs

With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

Installation:

`npm install express`

Getting started:

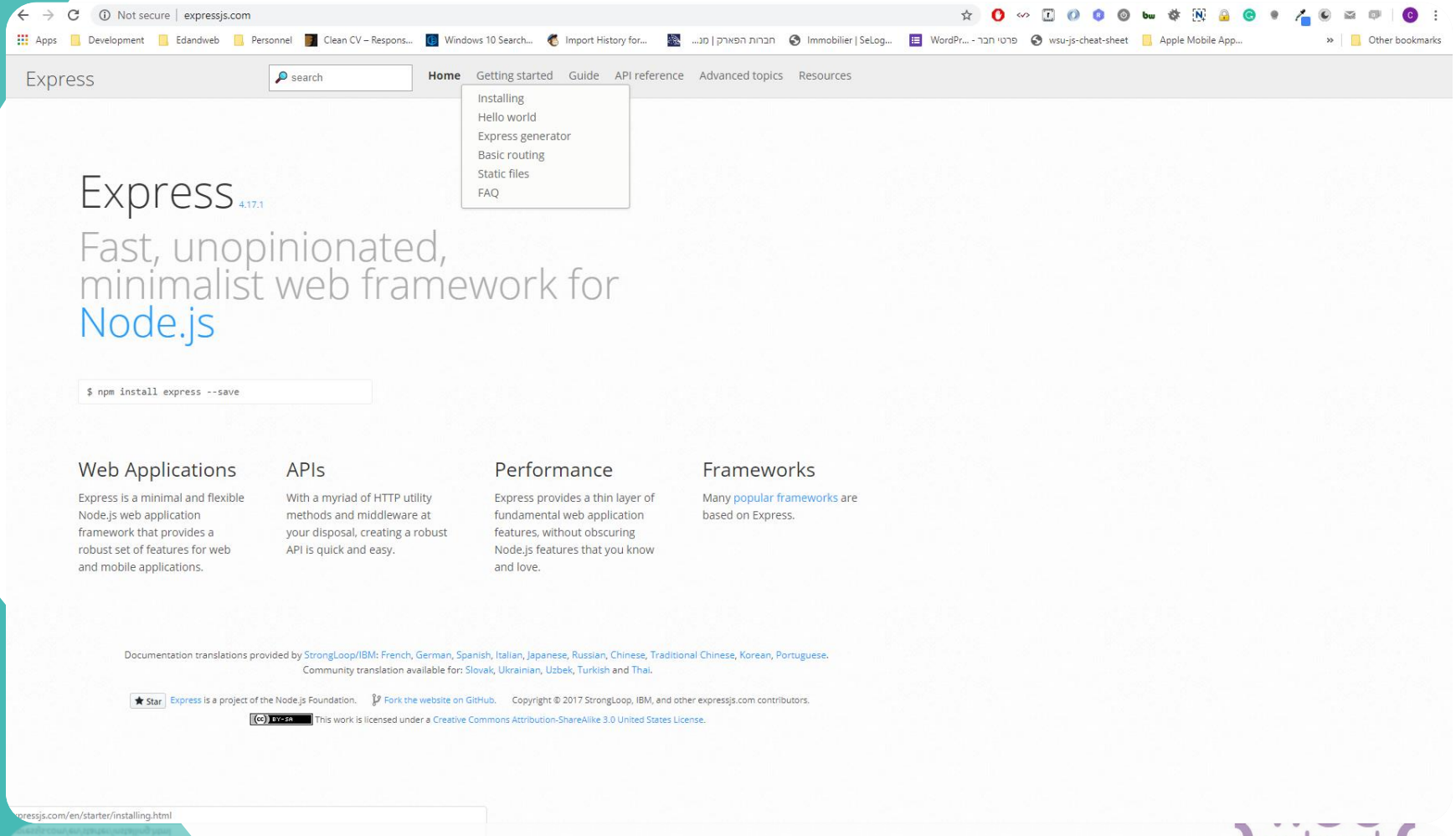
<http://expressjs.com/en/starter/installing.html>

API Docs:

<http://expressjs.com/en/guide/routing.html>



Express Guides



The screenshot shows the Express.js website homepage. The browser's address bar displays 'Not secure | expressjs.com'. The navigation bar includes links for Home, Getting started, Guide, API reference, Advanced topics, and Resources. A dropdown menu for the 'Guide' link lists: Installing, Hello world, Express generator, Basic routing, Static files, and FAQ. The main heading reads 'Express 4.17.1' followed by the tagline 'Fast, unopinionated, minimalist web framework for Node.js'. Below this is a terminal-style code block: '\$ npm install express --save'. The page is divided into four columns: 'Web Applications' (describing Express as a minimal and flexible Node.js web application framework), 'APIs' (describing it as a myriad of HTTP utility methods and middleware), 'Performance' (describing it as a thin layer of fundamental web application features), and 'Frameworks' (stating that many popular frameworks are based on Express). At the bottom, there is a footer with translation information, a GitHub star icon, and a Creative Commons license notice.

Express 4.17.1
Fast, unopinionated, minimalist web framework for Node.js

```
$ npm install express --save
```

Web Applications
Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

APIs
With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance
Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

Frameworks
Many popular frameworks are based on Express.

Documentation translations provided by StrongLoop/IBM: French, German, Spanish, Italian, Japanese, Russian, Chinese, Traditional Chinese, Korean, Portuguese.
Community translation available for: Slovak, Ukrainian, Uzbek, Turkish and Thai.

★ Star Express is a project of the Node.js Foundation. Fork the website on GitHub. Copyright © 2017 StrongLoop, IBM, and other expressjs.com contributors.
This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 United States License.

Création première routes

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

const app = express();

app.get('/', (req, res) => {
  res.send('Welcome to the homepage!');
});

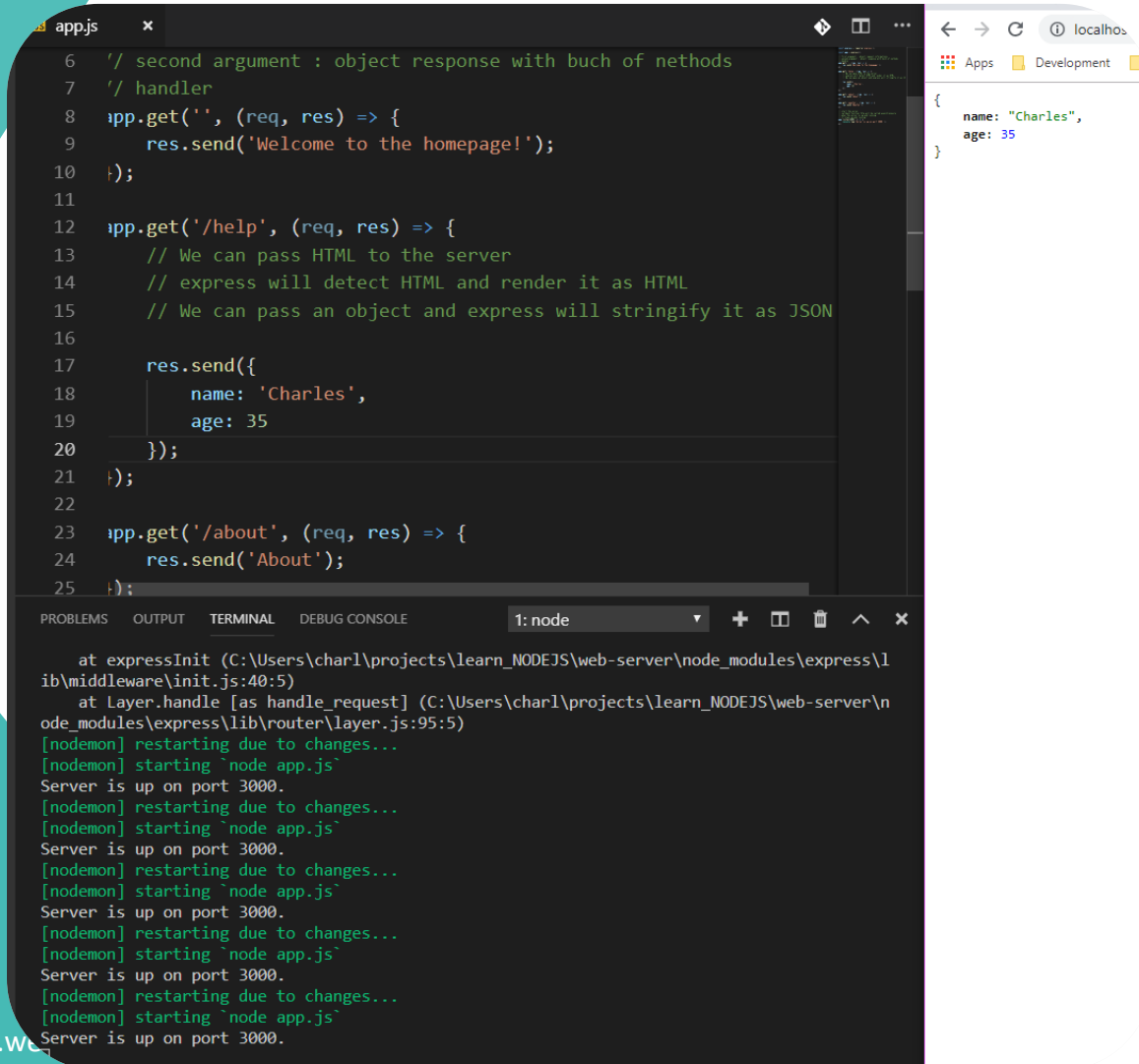
app.listen(3000, () => {
  console.log('Server is up on port 3000.');
```

Pour initialiser le framework:

1. Initialisation de npm
2. npm install express
3. Création d'un dossier src
4. Création d'un script app.js
5. Import de la librairie
6. Création d'une application express()
7. Définition d'une route:
<http://expressjs.com/en/guide/routing.html>
8. On écoute sur un port particulier et on définit une fonction au moment où le serveur est lancé.



Express - retour JSON



```
app.js
6 // second argument : object response with buch of methods
7 // handler
8 app.get('', (req, res) => {
9   res.send('Welcome to the homepage!');
10 });
11
12 app.get('/help', (req, res) => {
13   // We can pass HTML to the server
14   // express will detect HTML and render it as HTML
15   // We can pass an object and express will stringify it as JSON
16
17   res.send({
18     name: 'Charles',
19     age: 35
20   });
21 });
22
23 app.get('/about', (req, res) => {
24   res.send('About');
25 });
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: node
at expressInit (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\express\lib\middleware\init.js:40:5)
at Layer.handle [as handle_request] (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\express\lib\router\layer.js:95:5)
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Server is up on port 3000.
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Server is up on port 3000.
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Server is up on port 3000.
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Server is up on port 3000.
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Server is up on port 3000.
```

Suivant les données retournées par la fonction `send` express adaptera la sortie générée.

Si nous avons du HTML alors la page affichée sera du HTML.

Si nous retournons un objet alors automatiquement express retournera un JSON.



Créer 3 routes: about, help et weather

TP

Créer 2 routes:

1. /about : retourne un titre html h1 avec le nom About
2. /help: retourne un titre html h1 avec le nom Help
3. /weather: retourne un objet json avec 2 propriétés forecast (phrase météo) et location (ville)

Serve static files

(Servir les fichiers statiques)

Pour servir les fichiers statiques, nous devons spécifier à express un chemin **absolu** vers le dossier des fichiers statiques.

Créons un répertoire public où nous mettrons nos fichiers statiques exposés par le serveur WEB.

Pour obtenir le chemin absolu nous utiliserons `__dirname` :

https://nodejs.org/docs/latest/api/modules.html#modules_dirname

```
const pathPublic = path.join(__dirname, '../public');
```

Ensuite nous utiliserons la méthode `use` de notre application:

```
app.use(express.static(pathPublic));
```

Maintenant créons un fichier `index.html` et plaçons-le dans le dossier public.

Visitez l'URL root de notre projet.

Que se passe-t-il ?



Remplaçons les routes about et help par un fichier statique

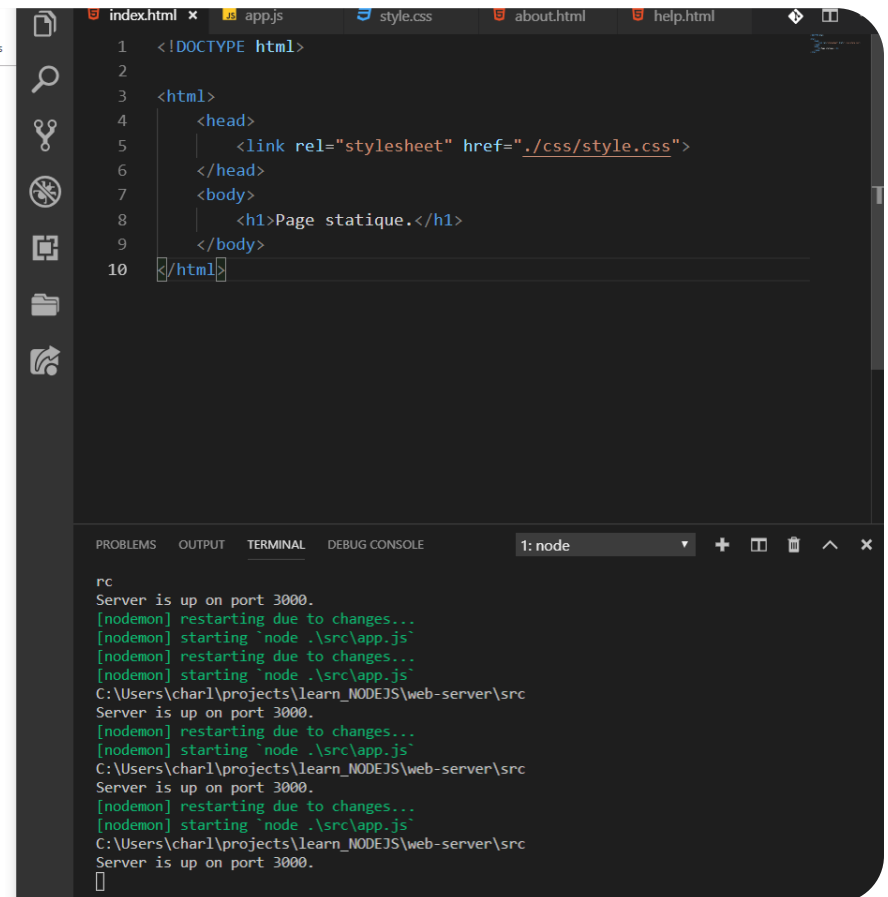
TP

1. Remplacer les 2 routes /about et /help par des fichier HTML.
2. Tester les routes.
3. Supprimer les routes de nodejs
4. Re tester les routes.

Serve CSS and JS files

Important: mettre ce qui doit être publique dans le dossier publique et rien d'autre.
./ fait référence au dossier publique

Page statique.



The screenshot shows a web browser on the left and a code editor on the right. The browser displays a simple static page with the text "Page statique." The code editor shows the source code of the page, which includes a link to a CSS file and the text "Page statique."

```
1 <!DOCTYPE html>
2
3 <html>
4   <head>
5     <link rel="stylesheet" href="./css/style.css">
6   </head>
7   <body>
8     <h1>Page statique.</h1>
9   </body>
10 </html>
```

The terminal window at the bottom of the code editor shows the output of the Node.js server, indicating that the server is up on port 3000 and restarting due to changes.

```
rc
Server is up on port 3000.
[nodemon] restarting due to changes...
[nodemon] starting `node .\src\app.js`
[nodemon] restarting due to changes...
[nodemon] starting `node .\src\app.js`
C:\Users\charl\projects\learn_NODEJS\web-server\src
Server is up on port 3000.
[nodemon] restarting due to changes...
[nodemon] starting `node .\src\app.js`
C:\Users\charl\projects\learn_NODEJS\web-server\src
Server is up on port 3000.
[nodemon] restarting due to changes...
[nodemon] starting `node .\src\app.js`
C:\Users\charl\projects\learn_NODEJS\web-server\src
Server is up on port 3000.
```

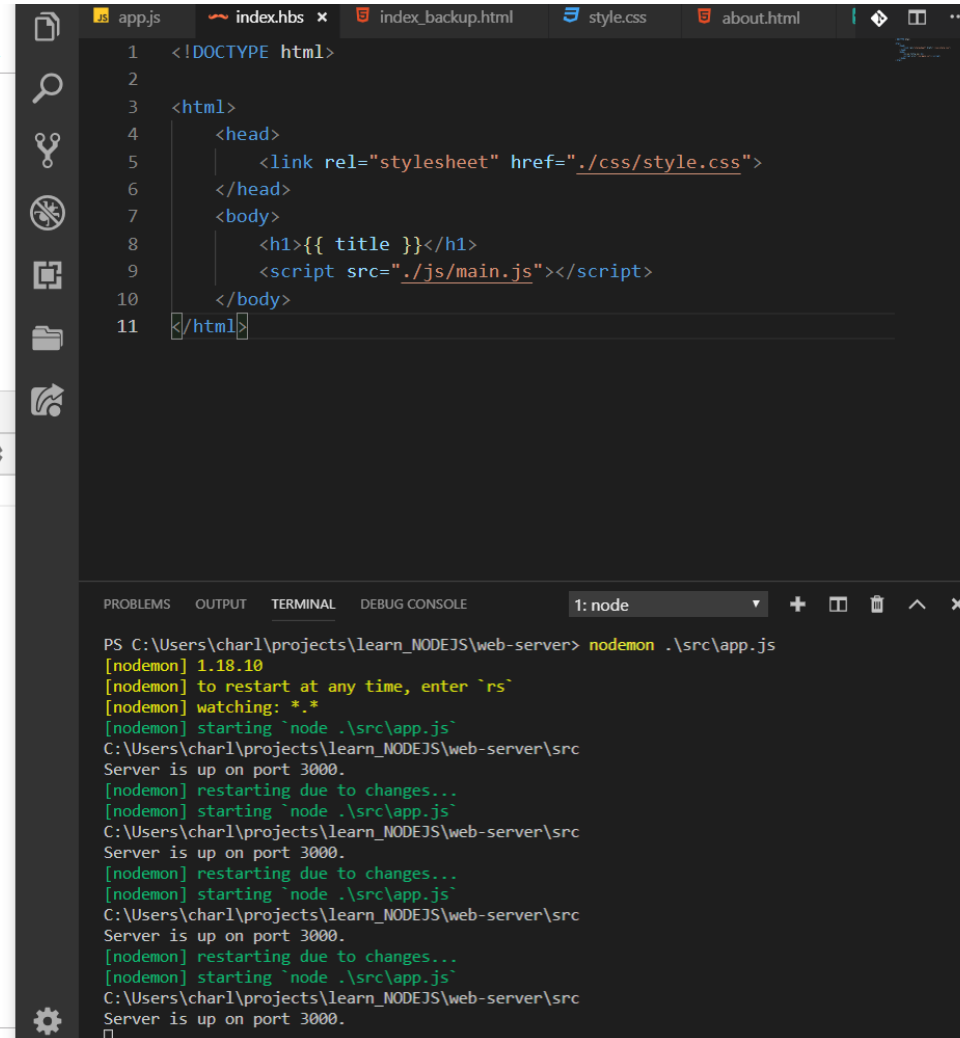
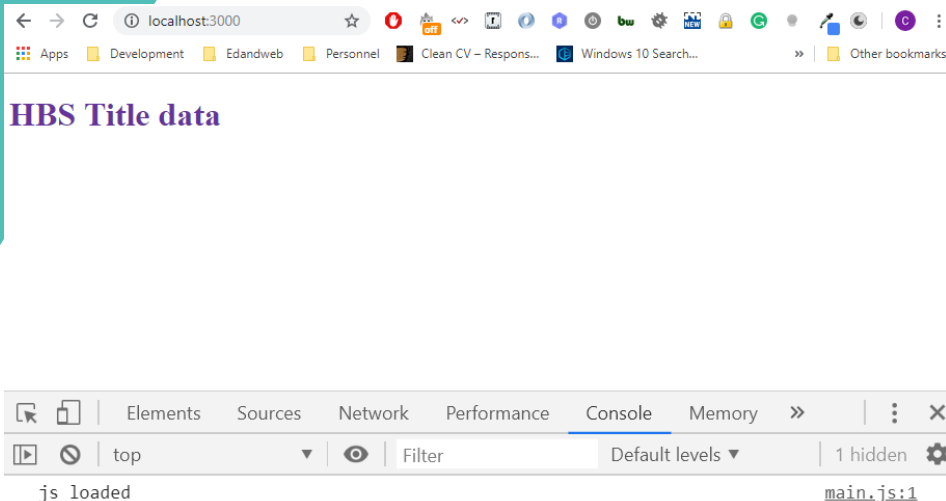
[school]

Importer un script JS et une photo

TP

1. Créer un script main.js dans un dossier js
2. Importer le script dans la page index.html
3. Télécharger une photo et placer la dans un dossier img.
4. Insérer la photo dans la page about.html et ajouter la feuille de style.
5. Ajouter une règle css (ex: largeur de 250px.)

Templates in Node.js with hbs (Handlebars for Node.js)



Utilisation des templates

Créer un dossier nommer views.

Créer un fichier nommé index.hbs dans ce dossier.

Au niveau du controller de la route (celle ou vous souhaitez implémenter le rendu du template) utiliser la fonction render (à partir de la variable response) qui prend en paramètre le nom du template et les data (optionnel objet).

Ex: `res.render('template_name', {data: data})`

Si on souhaite changer le nom du répertoire des templates:

```
const viewsPath = path.join(__dirname, '../templates/views');
```

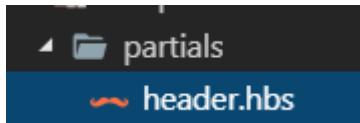


Créer des blocs de templates partials

```
const hbs = require('hbs');
```

```
const partialsPath = path.join(__dirname, '../templates/partials');
```

```
hbs.registerPartials(partialsPath);
```



```
<h1>{{title}}</h1>

<div>
  <a href="/">Weather</a>
  <a href="/about">About</a>
  <a href="/help">Help</a>
</div>
```

{{> @partial-block}}

```
{{>header}}
```



Etendre un template

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="/css/styles.css" />
  </head>
  <body>
    {{>header}}
    {{> @partial-block}}
    {{>footer}}
    <script src="/js/main.js"></script>
  </body>
</html>
```

```
{{#> base}}
<form method="POST" id="searchWeatherForm" action="" name="searchWeather">
  <input type="text" value="" name="destination">
  <input type="submit" value="Search">
</form>
<div id="meteo"></div>
{{/base}}
```

Afficher les données provenant d'objets ou d'Array

```
app.get('/about', (req, res) => {  
  const peoplesData = {  
    person1: {name: 'Charles Cohen', age: 35, country: 'Israel'},  
    person2: {name: 'David Cohen', age: 38, country: 'France'},  
    person3: {name: 'Dan Azoulay', age: 30, country: 'United States'}  
  };  
  res.render('about', {title: 'About', peoples: peoplesData});  
});
```

```
{{#each peoples}}  
  {{!-- {{this}} --}}  
  <li>Nom: {{this.name}}, Age: {{this.age}}, Country: {{this.country}} </li>  
{{/each}}
```

Error: C:\Users\charl\projects\learn_NODEJS\web-server\templates\views\index.hbs: The partial header could not be found
 at Object.invokePartial (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:175:32)
 at Object.invokePartialWrapper [as invokePartial] (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:175:32)
 at Object.eval [as main] (eval at createFunctionContext (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:175:32))
 at ret (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:178:12)
 at ret (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\compiler\compiler.js:526:21)
 at C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\hbs\lib\hbs.js:63:19
 at FSReqWrap.readFileAfterClose [as oncomplete] (internal/fs/read_file_context.js:53:3)

Refused to load the font 'cURL' because it violates the following Content Security Policy directive: "default-src 'self'". Note that 'font-src' was not explicitly set, so 'default-src' is used as a fallback.

Refused to load the image 'data:image/png;base64,iVBORw0KGgoAAAANSUHEugA localhost/:1 AABgAAAAAYCAYAAADgdz34AAAKQWIDQ...wTyZvw7vyxILHKN7tUJ/SzB1PFa/Seqxi/E6gGnufjb8tvOL5bY3Z6B JoAAAAASUVORK5CYII=' because it violates the following Content Security Policy directive: "default-src 'self'". Note that 'img-src' was not explicitly set, so 'default-src' is used as a fallback.

EXPLORER

app.js header.hbs index.hbs help.hbs

OPEN EDITORS

- app.js web-server/src
- header.hbs web-server/...
- index.hbs web-server/...
- help.hbs web-server/...

LEARN_NODEJS

- docs a preparer
- hello-app
- node_modules
- notes-app
- weather-app
- web-server
 - node_modules
 - public
 - css
 - style.css
 - img
 - me.jpg
 - js
 - main.js
 - about.html
 - help.html
 - index_backup.html
 - src
 - app.js
 - app1.js
 - templates
 - partials
 - header.hbs
 - views
 - help.hbs
 - index.hbs
 - package-lock.json
 - package.json

```

1 <!DOCTYPE html>
2
3 <html>
4   <head>
5     <link rel="stylesheet" href="/css/style.css">
6   </head>
7   <body>
8     {{>header}}
9     <script src="/js/main.js"></script>
10  </body>
11 </html>
  
```

PROBLEMS TERMINAL ... 1: node

```

web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:
281:11)
    at Object.invokePartialWrapper [as invokePartial] (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:68:39)
    at Object.eval [as main] (eval at createFunctionContext (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:175:32))
    at ret (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:178:12)
    at ret (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\compiler\javascript-compiler.js:257:23), <anonymous>:6:28)
    at main (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:175:32)
    at ret (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\runtime.js:178:12)
    at ret (C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\handlebars\dist\cjs\handlebars\compiler\compiler.js:526:21)
    at C:\Users\charl\projects\learn_NODEJS\web-server\node_modules\hbs\lib\hbs.js:63:19
    at FSReqWrap.readFileAfterClose [as oncomplete] (internal/fs/read_file_context.js:53:3)
  
```



Solution pour prise en compte changement dans les templates

```
nodemon .\src\app.js -e js,hbs
```

Listen for changes in extension



Créer un footer (partials)

TP

1. Créer un footer contenant un paragraphe:
2. Créé par {{name}}
3. Importer le footer dans vos templates.
4. Créer un nouveau partial menu et mettez les liens dans ce fichiers.
5. Importer le menu depuis votre header et votre footer.

Gérer les erreurs 404

Routes spécifique

```
app.get('/help/*', (req, res) => {  
    res.send('Help article not  
found.');
```

Toutes les routes

```
// need to be the last  
// express will search in the public folder  
// next on each routes definitions  
// finally will find  
app.get('*', (req, res) => {  
    res.send('404 Page Not found.');
```



Gérer les erreurs 404

TP

1. Créer une page 404 avec handlebars pour toutes les requetes.
2. Créer une page 404 pour les routes en dessous de help /help/*
3. Pour chaque vue définir passer en argument un message d'erreur en fonction de la vue.

Intégrer les fonctionnalités des fonctions forecast et geocode dans notre application

TP

1. Copier le dossier utils
2. Appeler avec la route weather les fonctions geocode et weather

Créer un search form qui retournera la meteo pour une adresse

Exercice maison

1. Créer un search form qui recevra une adresse pour ensuite appeler l'API de weather et retournera la météo pour cette adresse.

```
fetch('http://google.com').then((response) => {  
    response.json().then((data) => {  
        console.log(data);  
    });  
});
```

A voir

Cookies, Sessions

Exemple complet: <https://appdividend.com/2018/02/05/express-session-tutorial-example-scratch/>

A lire: <https://stackoverflow.com/questions/8749907/what-is-a-good-session-store-for-a-single-host-node-js-production-app>

```
var express = require('express')
var cookieParser = require('cookie-parser')
var app = express()
app.use(cookieParser())
```

expressjs.com/en/advanced/best-practice-security.html#use-cookies-securely

landweb Personnel Clean CV – Respons... Windows 10 Search... Import History for... חברות הפארק | מנ... Immobilier | SeLog... WordPr...



search

Home Getting started Guide API reference **Advanced topics** Resources

Use cookies securely

To ensure cookies don't open your app to exploits, don't use the default session cookie name and set cookie security options appropriately.

There are two main middleware cookie session modules:

- [express-session](#) that replaces `express.session` middleware built-in to Express 3.x.
- [cookie-session](#) that replaces `express.cookieSession` middleware built-in to Express 3.x.

The main difference between these two modules is how they save cookie session data. The [express-session](#) middleware stores session data on the server; it only saves the session ID in the cookie itself, not session data. By default, it uses in-memory storage and is not designed for a production environment. In production, you'll need to set up a scalable session-store; see the list of [compatible session stores](#).

In contrast, [cookie-session](#) middleware implements cookie-backed storage: it serializes the entire session to the cookie, rather than just a session key. Only use it when session data is relatively small and easily encoded as primitive values (rather than objects). Although browsers are supposed to support at least 4096 bytes per cookie, to ensure you don't exceed the limit, don't exceed a size of 4093 bytes per domain. Also, be aware that the cookie data will be visible to the client, so if there is any reason to keep it secure or obscure, then [express-session](#) may be a better choice.



Résumé

