



# FORMATION FULL STACK

הרווחה, משרד העבודה  
והשירותים החברתיים



## Node.js

By Charles Cohen



Charles Cohen  
Senior Full Stack Developer  
[charlesc@edandweb.co.il](mailto: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 4

Consuming API  
Express Framework



# Plan du module Node.js

Cours	Date	Cours
1	Lun. 20/05	<ul style="list-style-type: none"><li>• Introduction to Node.js (1)</li></ul>
2	Mer. 29/05	<ul style="list-style-type: none"><li>• Building your Stack (2)</li><li>• Command Line</li><li>• File System</li><li>• Arrow Functions</li><li>• Modular JavaScript with Node.js</li></ul>
3	Mer. 05/06	<ul style="list-style-type: none"><li>• Asynchronous JS</li><li>• Consuming API – HTTP requests</li></ul>
4	Lun. 10/06	<ul style="list-style-type: none"><li>• Consuming API – HTTP requests</li><li>• Building Web applications with the Express Framework</li></ul>
5	Lun. 17/06	<ul style="list-style-type: none"><li>• Express framework – templates with Handlebars</li><li>• 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</li><li>• Implement a search address box that consuming the GET API</li></ul>
6	Lun. 24/06	<ul style="list-style-type: none"><li>• Promise, await, async, Async module ...</li><li>• Avoiding common pitfalls with Async.js</li></ul>
7	Lun. 01/07	<ul style="list-style-type: none"><li>• Working with MongoDB 1/2</li></ul>
8	Lun 08/07	<ul style="list-style-type: none"><li>• Working with MongoDB 1/2</li></ul>



# 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 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`



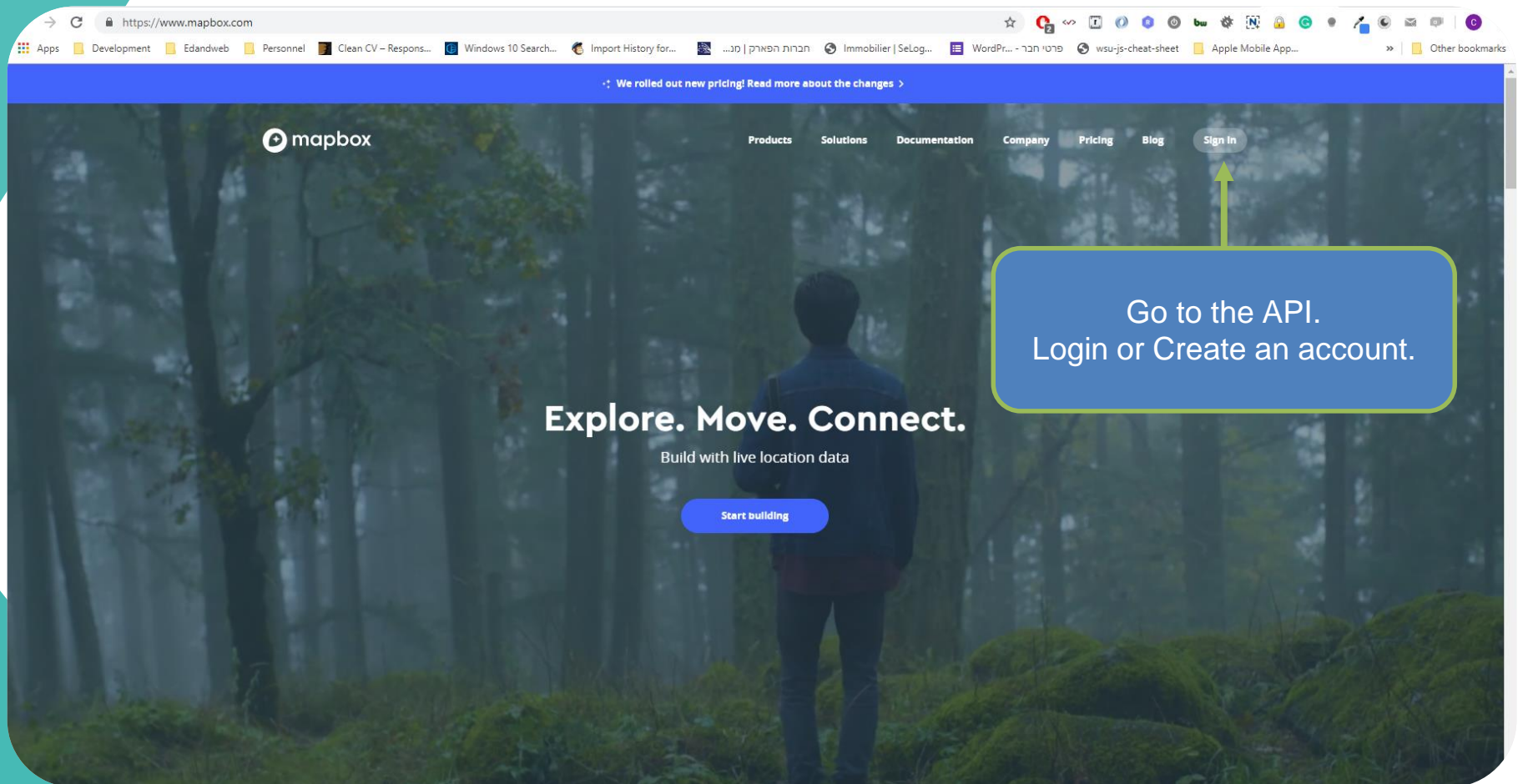
# Fin de l'application de notes

Ecrire les méthodes:

- Read (title)
- List
- Update (title, newTitle, newBody)

Vous pouvez créer une fonction de recherche qui retournera l'index.

# Mapbox.com



# Mapbox.com

## *Dashboard*

The screenshot shows the Mapbox.com dashboard for a user named 'edandweb'. The browser address bar displays 'https://account.mapbox.com'. A blue banner at the top contains a message: 'We rolled out new pricing so your invoice may look a little different. Read more about the changes >'. The navigation bar includes the Mapbox logo, the user's account name 'edandweb', and links for 'Dashboard', 'Tokens', 'Statistics', 'Invoices', and 'Settings'. A green arrow points from a callout box to the 'Tokens' link. The callout box, which has a green border, contains the text 'Go to the API documentation'. The main content area is divided into several sections: a 'Welcome, edandweb!' message; a 'Start by designing a map >' section with an illustration of a map on a tablet; a section for installing the Maps SDK for 'JS Web', 'iOS', 'Android', and 'Unity'; an 'Access tokens' section with a '+ Create a token' button and explanatory text; and a 'Plan' section with a 'View billing' button. On the right side, there is a 'Tools & resources' section with links to 'Integrate Mapbox', 'Design in Mapbox Studio', and 'Documentation'. The bottom right corner features the 'webschool' logo.

→ ↻ https://account.mapbox.com

Apps Development Edandweb Personnel Clean CV – Respons... Windows 10 Search... Import History for... חברות הפארק | Immo... WordPr... פורטי חבר - wsu-js-cheat-sheet Apple Mobile App... » | Other bookmarks

⚠ We rolled out new pricing so your invoice may look a little different. Read more about the changes >

mapbox | Account Dashboard Tokens Statistics Invoices Settings

Welcome, edandweb!

Go to the API documentation

Start by designing a map >

Or install the Maps SDK:

- JS Web
- iOS
- Android
- Unity

Plan View billing

Pay-as-you-go

Upgrade to new API pricing

Current billing cycle usage

No usage information to display.

Learn more about our new product offerings and how you can get started for free >

Tools & resources

- Integrate Mapbox
- Design in Mapbox Studio
- Documentation

Access tokens + Create a token

You need an API access token to configure Mapbox GL JS, Mobile, and Mapbox web services like routing and geocoding. Read more about API access tokens in our documentation.

Geocoding Read more about API access tokens in our documentation.

You need an API access token to configure Mapbox GL JS, Mobile, and Mapbox web services like routing and geocoding.

Access tokens + Create a token

Integrate Mapbox Studio

Design in Mapbox Studio

Documentation

Documentation

Design in Mapbox Studio

Design in Mapbox Studio

webschool

# Mapbox.com

## API Documentation

### API Documentation

#### Introduction

Reading this documentation  
Access tokens and token scopes  
API versioning  
Rate limit headers  
Rate limits  
HTTPS and CORS  
Coordinate format  
Date and time format  
Pagination  
High DPI images  
SDK and library support

#### Maps service

#### Navigation service

#### Search service

#### Accounts service

#### Mapbox API changelog

## Introduction

The Mapbox web services APIs allow you to programmatically access Mapbox tools and services. You can use these APIs to retrieve information about your account, upload and change resources, use core Mapbox tools, and more.

Mapbox APIs are divided into four distinct services: **Maps**, **Directions**, **Geocoding**, and **Accounts**. Each of these services has its own overview page in this documentation. These overview pages are divided into the individual APIs that make up the service. The documentation for each API is structured by *endpoints*. An endpoint is a specific method within an API that performs one action and is located at a specific URL.

#### Maps service

Overview of the Mapbox Maps service APIs.



#### Navigation service

Overview of the Mapbox Navigation service APIs.



#### Search service

Overview of the Mapbox Search service APIs.



#### Accounts service

Overview of the Mapbox Accounts service APIs.



The Mapbox APIs described in this documentation are subject to Mapbox's [Terms of Service](#).

## Reading this documentation

Each API endpoint in this documentation is described using several parts:

- **The HTTP method.** Includes GET, POST, PUT, PATCH, DELETE.
- **The base path.** All URLs referenced in the documentation have the base path `https://api.mapbox.com`. This base path goes before the endpoint path.
- **The page base.** All URLs referenced in the documentation have the page base `https://docs.mapbox.com`.
- **The HTTP method.** Includes GET, POST, PUT, PATCH, DELETE.

We'll use the Search service



## API Documentation

Introduction



Maps service



Navigation service



## Search service

## Geocoding

Endpoints

Data types

Forward geocoding

Reverse geocoding

Batch geocoding

Geocoding response object

Language coverage

Point of interest category coverage

Geocoding restrictions and limits

Accounts service



Mapbox API changelog



## Search service

The Mapbox Search service is composed of the following APIs:

- [Geocoding API](#)

## Geocoding

The Mapbox Geocoding API does two things: *forward geocoding* and *reverse geocoding*.

Forward geocoding converts location text into geographic coordinates, turning `2 Lincoln Memorial Circle NW` into `-77.050,38.889`.

Reverse geocoding turns geographic coordinates into place names, turning `-77.050, 38.889` into `2 Lincoln Memorial Circle NW`. These location names can vary in specificity, from individual addresses to states and countries that contain the given coordinates.

For more background information on the Mapbox Geocoding API and how it works, see the [How geocoding works guide](#).

## Endpoints

The geocoding API includes two different endpoints: `mapbox.places` and `mapbox.places-permanent`.

### mapbox.places

The `mapbox.places` endpoint is accessible to all geocoding customers. Requests to this endpoint must be triggered by user activity. Any results must be displayed on a Mapbox map and cannot be stored permanently, as described in Mapbox's [terms of service](#).

### mapbox.places-permanent

The `mapbox.places-permanent` endpoint is accessible to all geocoding customers. Requests to this endpoint must be triggered by user activity. Any results must be displayed on a Mapbox map and cannot be stored permanently, as described in Mapbox's [terms of service](#).

### mapbox.places

The `mapbox.places` endpoint is accessible to all geocoding customers. Requests to this endpoint must be triggered by user activity. Any results must be displayed on a Mapbox map and cannot be stored permanently, as described in Mapbox's [terms of service](#).

### mapbox.places



## API Documentation

Introduction

Maps service

Navigation service

### Search service

#### Geocoding

Endpoints

#### Data types

Forward geocoding

Reverse geocoding

Batch geocoding

Geocoding response object

Language coverage

Point of interest category coverage

Geocoding restrictions and limits

Accounts service

Mapbox API changelog

## Forward geocoding

```
GET /geocoding/v5/{endpoint}/{search_text}.json
```

The **forward geocoding** query type allows you to look up a single location by name and returns its geographic coordinates.

Try forward geocoding in the [Search Playground](#).

Required parameters	Description
<code>endpoint</code>	One of <code>mapbox.places</code> or <code>mapbox.places-permanent</code> , as described in the <a href="#">Endpoints section</a> .
<code>search_text</code>	The feature you're trying to look up. This could be an address, a point of interest name, a city name, etc. When searching for points of interest, it can also be a category name (for example, "coffee shop"). For information on categories, see the <a href="#">Point of interest category coverage section</a> . The search text should be expressed as a URL-encoded UTF-8 string, and must not contain the semicolon character (either raw or URL-encoded). Your search text, once decoded, must consist of at most 20 words and numbers separated by spacing and punctuation, and at most 256 characters.

You can further refine the results of a forward geocoding query with the following optional parameters:

Optional parameters	Description
<code>autocomplete</code>	Specify whether to return autocomplete results ( <code>true</code> , default) or not ( <code>false</code> ). When autocomplete is enabled, results will be included that start with the requested string, rather than just responses that match it exactly. For example, a query for <code>India</code> might return both <code>India</code> and <code>Indiana</code> with autocomplete enabled, but only <code>India</code> if it's disabled.
<code>bbox</code>	Limit results to only those contained within the supplied bounding box. Bounding boxes should be supplied as four numbers separated by commas, in <code>minLon,minLat,maxLon,maxLat</code> order. The bounding box cannot cross the 180th meridian.
<code>prox</code>	Proximity to a specific location. The value should be a distance in meters, ranging from 0 to 10000. The results will be sorted by proximity to the specified location.
<code>types</code>	A list of feature types to filter the results. The types are defined in the <a href="#">Point of interest category coverage section</a> .





## API Documentation

### Introduction

### Maps service

### Navigation service

### Search service

#### Geocoding

Endpoints

Data types

#### Forward geocoding

Reverse geocoding

Batch geocoding

Geocoding response object

Language coverage

Point of interest category coverage

Geocoding restrictions and limits

### Accounts service

### Mapbox API changelog

## Example request: Forward geocoding

```
# A basic forward geocoding request
# Find Los Angeles

$ curl "https://api.mapbox.com/geocoding/v5/mapbox.places/Los%20Angeles.json?
access_token=pk.eyJ1Ijo1ZWRhbmR3ZWl1IjoiY2p1ODB3bm1uM0o4djN5cGZ6ejJzbnh4NCJ9.2La2amf6m8YVPVwb5YkviA"

# Find a town called 'Chester' in a specific region
# Add the proximity parameter with local coordinates
# This ensures the town of Chester, New Jersey is in the results

$ curl "https://api.mapbox.com/geocoding/v5/mapbox.places/chester.json?
proximity=-74.70850,40.78375&access_token=pk.eyJ1Ijo1ZWRhbmR3ZWl1IjoiY2p1ODB3bm1uM0o4djN5cGZ6ejJzbnh4NCJ9.2L
```

### Endpoint support

Mapbox wrapper libraries help you integrate Mapbox APIs into your existing application. The following SDKs support this endpoint:

- [Mapbox CLI SDK](#)
- [MapboxDirections.swift](#) (Objective-C and Swift)
- [Mapbox Java SDK](#)
- [Mapbox JavaScript SDK](#)
- [Mapbox Python SDK](#)
- [Mapbox React Geocoding Plugin](#)
- [Mapbox Ruby SDK](#)

See the SDK documentation for details and examples of how to use the relevant methods to query this endpoint.

## Response: Forward geocoding

The API response for a forward geocoding query returns a GeoJSON feature collection in Mapbox Geocoding Response format. For more details on how a response from the Geocoding API is formatted, see the [Geocoding API response object section](#).

## Reverse geocoding

Reverse geocoding

Reverse geocoding

Reverse geocoding: For more details on how a response from the Geocoding API is formatted, see the [Geocoding API response object section](#). Reverse geocoding: For more details on how a response from the Geocoding API is formatted, see the [Geocoding API response object section](#).



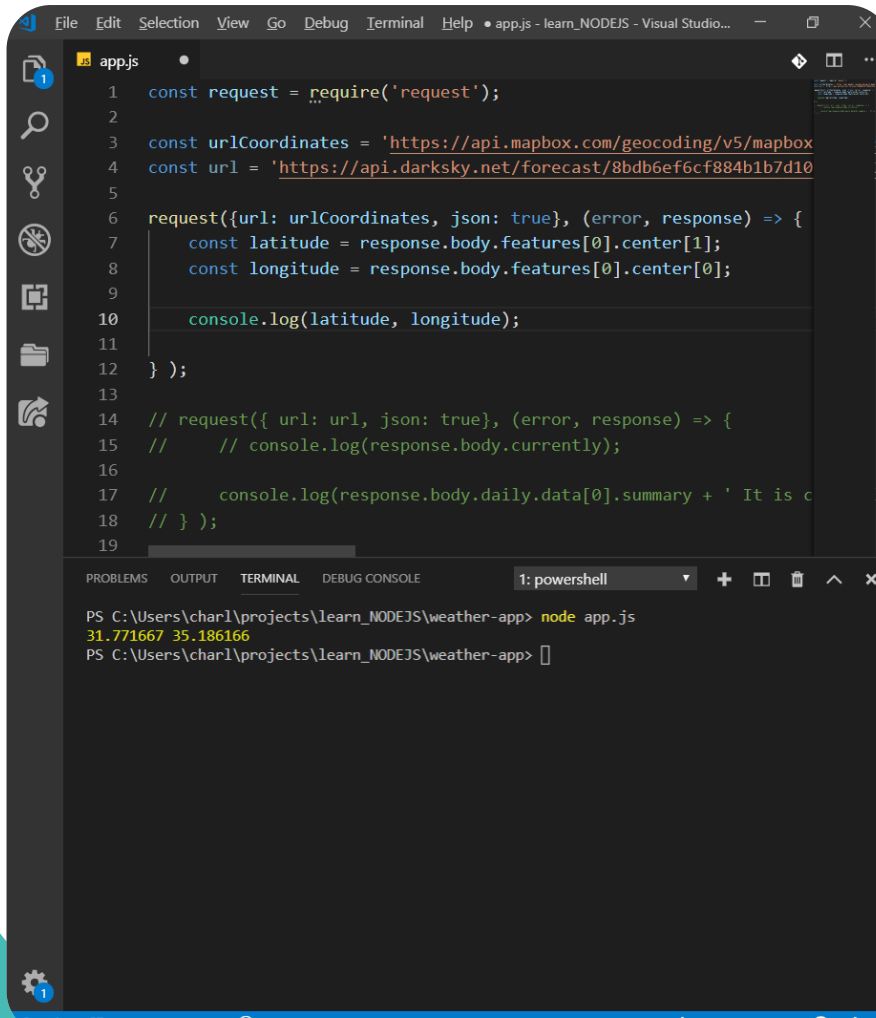
# API Geocode – longitude , latitude

Mapbox Inc. [US] | https://api.mapbox.com/geocoding/v5/mapbox.places/Los%20Angeles.json?access\_token=pk.eyJ1IjoizWRhbmR3ZWlLCjhljoiY2p1ODB3bm1uMHo4d...  
Apps Development Edandweb Personnel Clean CV – Respons... Windows 10 Search... Import History for... חברות הפאנק | מנ... Immobilier | SeLog... WordPr... פרטי חבר - wsu-js-cheat-sheet Apple Mobile App... Other bookmarks

```
{
  type: "FeatureCollection",
  query: [
    "los",
    "angeles"
  ],
  features: [
    {
      id: "place.7397503093427640",
      type: "Feature",
      place_type: [
        "place"
      ],
      relevance: 1,
      properties: {
        wikidata: "Q65"
      },
      text: "Los Angeles",
      place_name: "Los Angeles, California, United States",
      bbox: [
        -118.521456965901,
        33.9018913203336,
        -118.121305008073,
        34.161440999758
      ],
      center: [
        -118.2439,
        34.0544
      ],
      geometry: {
        type: "Point",
        coordinates: [
          -118.2439,
          34.0544
        ]
      },
      context: [
        {
          id: "region.11319063928738010",
          short_code: "US-CA",
          wikidata: "Q99",
          text: "California"
        },
        {
          id: "country.9053006287256050",
          short_code: "us",
          wikidata: "Q30",
          text: "United States"
        }
      ]
    }
  ]
}
```



# Créons la fonction récupérant la longitude et la latitude



```
1 const request = require('request');
2
3 const urlCoordinates = 'https://api.mapbox.com/geocoding/v5/mapbox
4 const url = 'https://api.darksky.net/forecast/8bdb6ef6cf884b1b7d10
5
6 request({url: urlCoordinates, json: true}, (error, response) => {
7   const latitude = response.body.features[0].center[1];
8   const longitude = response.body.features[0].center[0];
9
10  console.log(latitude, longitude);
11
12 } );
13
14 // request({ url: url, json: true}, (error, response) => {
15 //   // console.log(response.body.currently);
16
17 //   console.log(response.body.daily.data[0].summary + ' It is c
18 // } );
19
```

TERMINAL

```
PS C:\Users\charl\projects\learn_NODEJS\weather-app> node app.js
31.771667 35.186166
PS C:\Users\charl\projects\learn_NODEJS\weather-app>
```

## TP

Inspirez vous de la fonction forecast pour construire fonction geocode.

Gérer les erreurs:

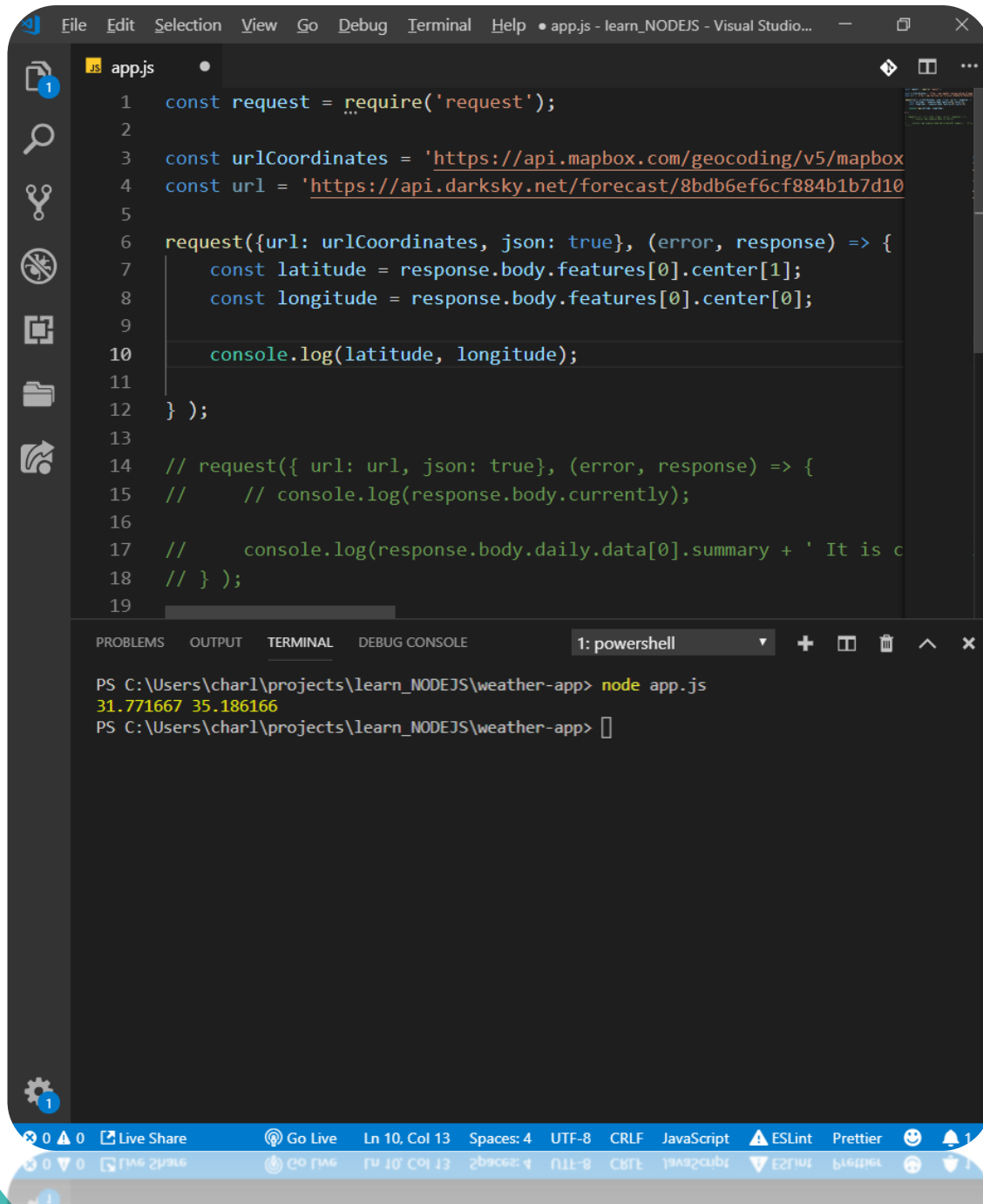
- Erreur low level (network ...)
- Problème de coordonnées

```
1 const request = require('request');
2 const geocode = require('./utils/geocode.js');
3 const forecast = require('./utils/forecast.js');
4
5
6
7 geocode.getCoordinates('Jerusalem', (error, data) => {
8   if (error) {
9     console.log('Error occurred: ' + error);
10   } else {
11     console.log(data);
12   }
13 });
14
15
16 forecast.getWeather(-75.7088, 44.1545, (error, data) => {
17   if (error) {
18     console.log('Error', error)
19   } else {
20     // console.log('Data', data)
21     console.log(data.daily.data[0].summary + ' It is currently ' + data.currently.temperature + ' degrees out. There is a ' + data.currently.precipProbability + '% chance of rain.');
22   }
23 });
24
25
26
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

1: powershell

PS C:\Users\charl\projects\learn\_NODEJS\weather-app> node app.js  
{ latitude: 31.771667, longitude: 35.186166 }  
Brumeux toute la journée. It is currently -43.61 degrees out. There is a 0.07% chance of rain.  
5  
PS C:\Users\charl\projects\learn\_NODEJS\weather-app>



```
1  const request = require('request');
2
3  const urlCoordinates = 'https://api.mapbox.com/geocoding/v5/mapbox
4  const url = 'https://api.darksky.net/forecast/8bdb6ef6cf884b1b7d10
5
6  request({url: urlCoordinates, json: true}, (error, response) => {
7    const latitude = response.body.features[0].center[1];
8    const longitude = response.body.features[0].center[0];
9
10   console.log(latitude, longitude);
11
12 } );
13
14 // request({ url: url, json: true}, (error, response) => {
15 //   // console.log(response.body.currently);
16
17 //   console.log(response.body.daily.data[0].summary + ' It is c
18 // } );
19
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: powershell

```
PS C:\Users\charl\projects\learn_NODEJS\weather-app> node app.js
31.771667 35.186166
PS C:\Users\charl\projects\learn_NODEJS\weather-app>
```

```
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<sup>ème</sup> 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);  
});
```

# Appeler la fonction forecast avec le résultat de la fonction geocode

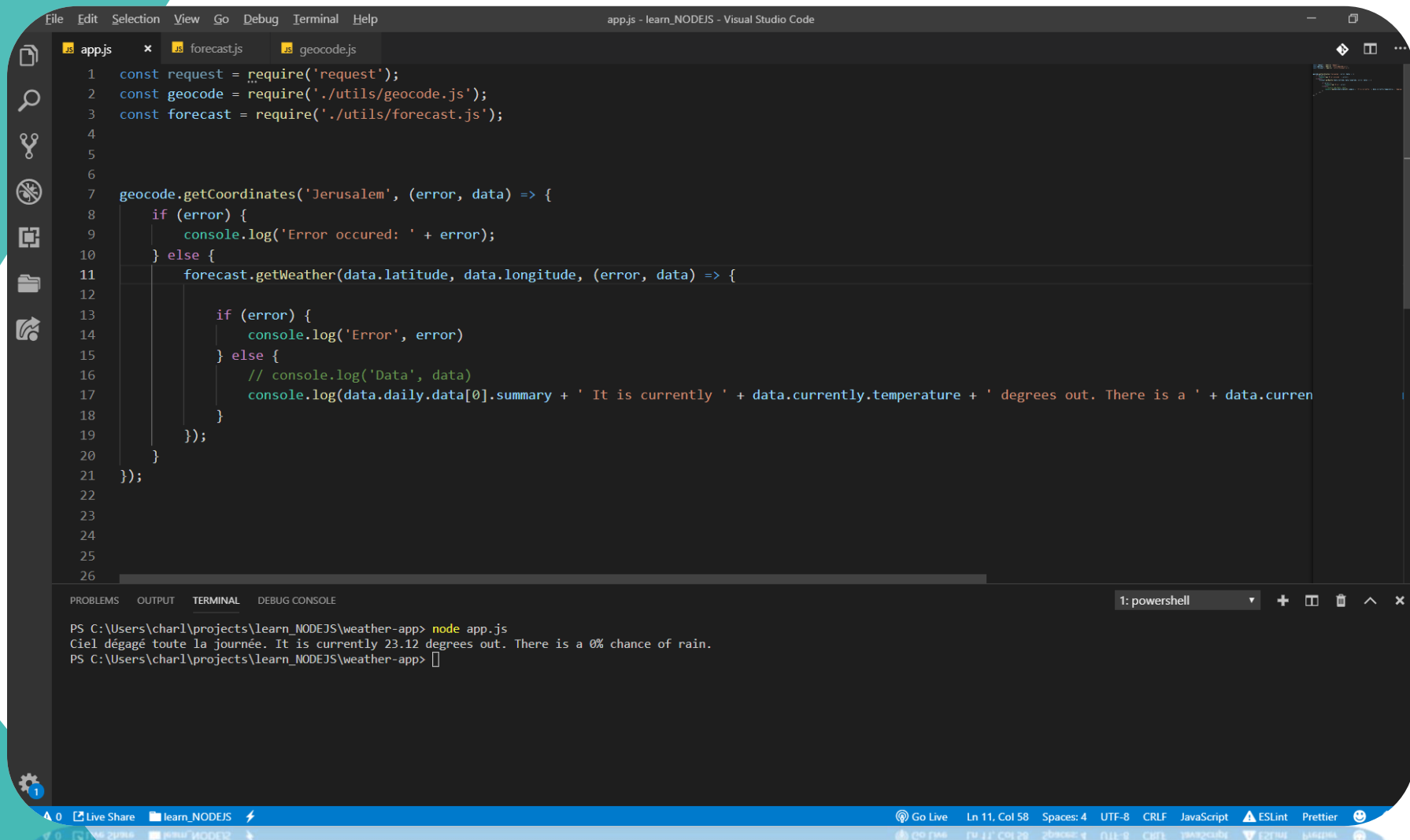
TP

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





```
File Edit Selection View Go Debug Terminal Help
app.js - learn_NODEJS - Visual Studio Code
app.js forecast.js geocode.js
1 const request = require('request');
2 const geocode = require('./utils/geocode.js');
3 const forecast = require('./utils/forecast.js');
4
5
6
7 geocode.getCoordinates('Jerusalem', (error, data) => {
8   if (error) {
9     console.log('Error occurred: ' + error);
10   } else {
11     forecast.getWeather(data.latitude, data.longitude, (error, data) => {
12
13       if (error) {
14         console.log('Error', error)
15       } else {
16         // console.log('Data', data)
17         console.log(data.daily.data[0].summary + ' It is currently ' + data.currently.temperature + ' degrees out. There is a ' + data.currently.precipProbability + '% chance of rain.');
18       }
19     });
20   }
21 });
22
23
24
25
26
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
1: powershell
PS C:\Users\charl\projects\learn_NODEJS\weather-app> node app.js
Ciel dégagé toute la journée. It is currently 23.12 degrees out. There is a 0% chance of rain.
PS C:\Users\charl\projects\learn_NODEJS\weather-app>
```

# 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>



# Création première routes

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

const app = express();

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

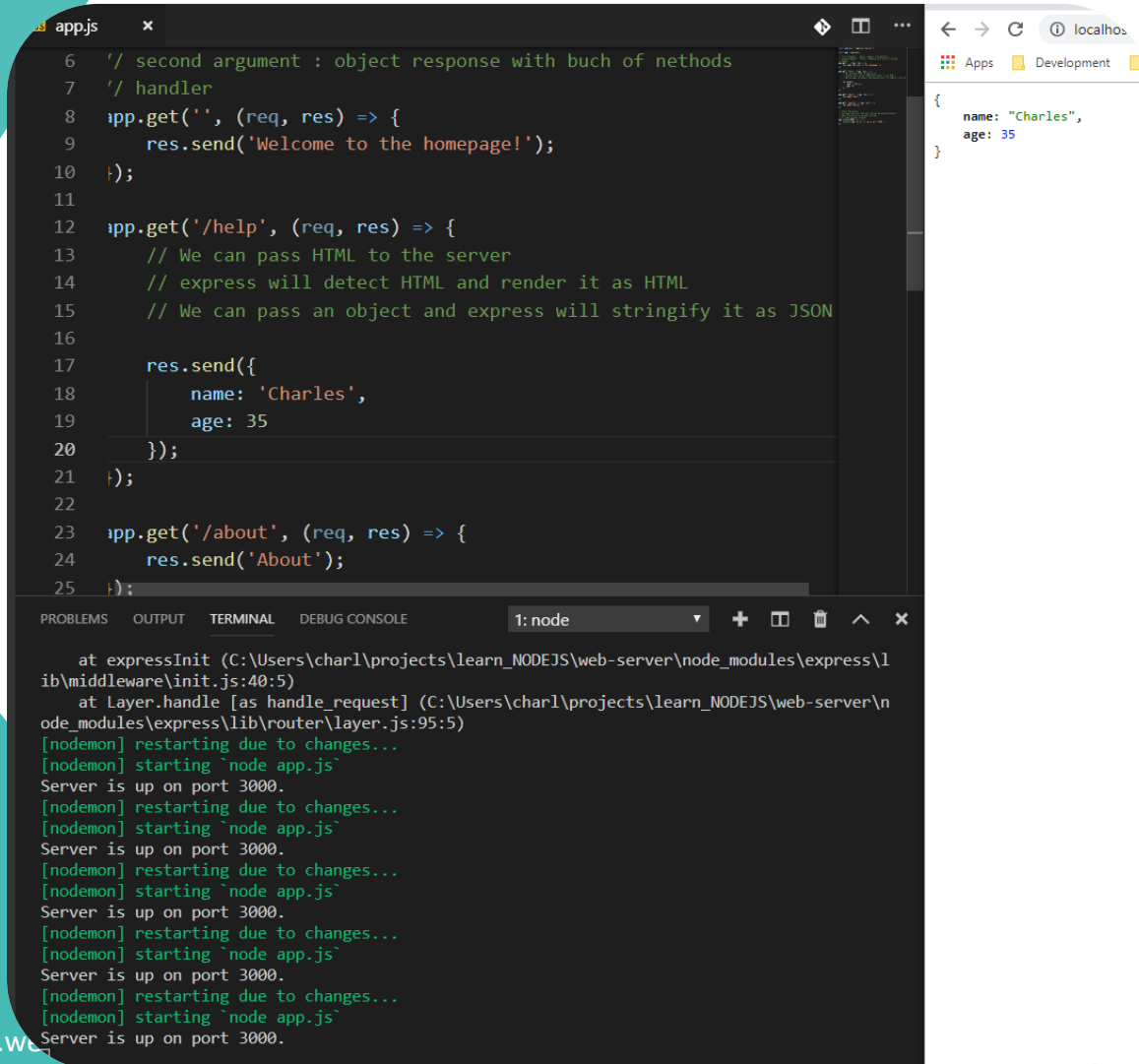
app.listen(3000, () => {
  console.log('Server is up on port 3000.');
```

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



The screenshot shows a VS Code editor with a file named `app.js`. The code defines an Express application with three routes: a root route, a `/help` route, and an `/about` route. The `/help` route sends a JSON object with `name: 'Charles'` and `age: 35`. The `/about` route sends the string `'About'`. The terminal at the bottom shows the server starting on port 3000 and restarting multiple times due to changes in the code.

```
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: 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](https://nodejs.org/docs/latest/api/modules.html#modules_dirname)

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

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

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

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

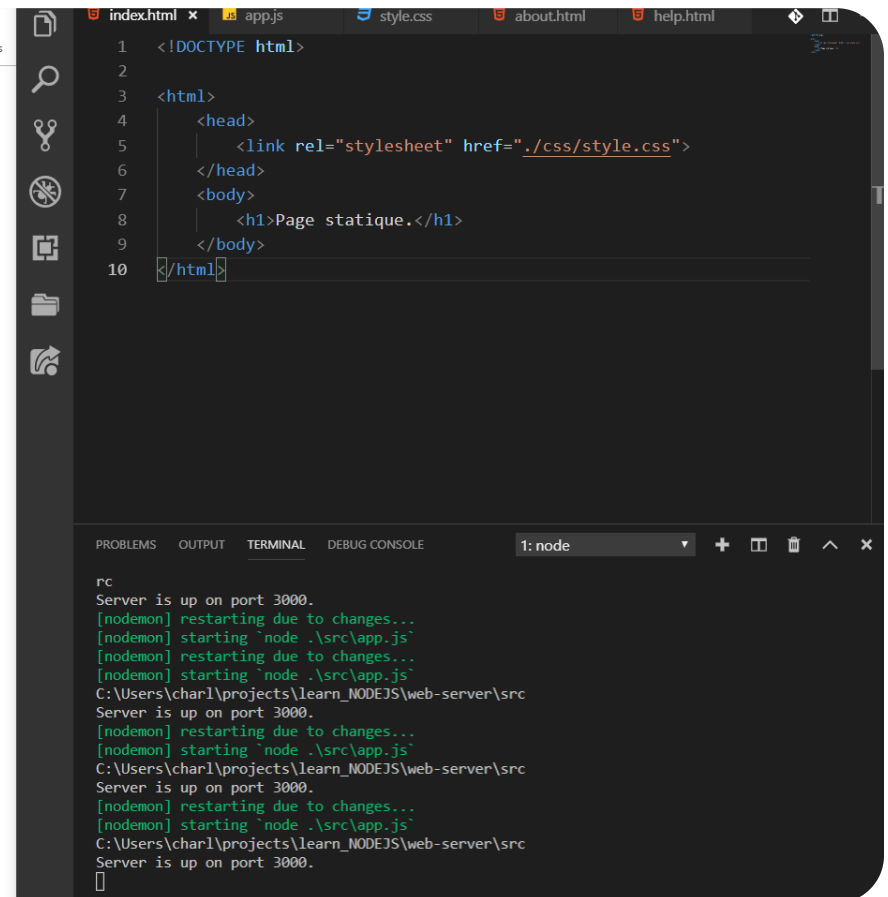
## TP

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.
[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.
```

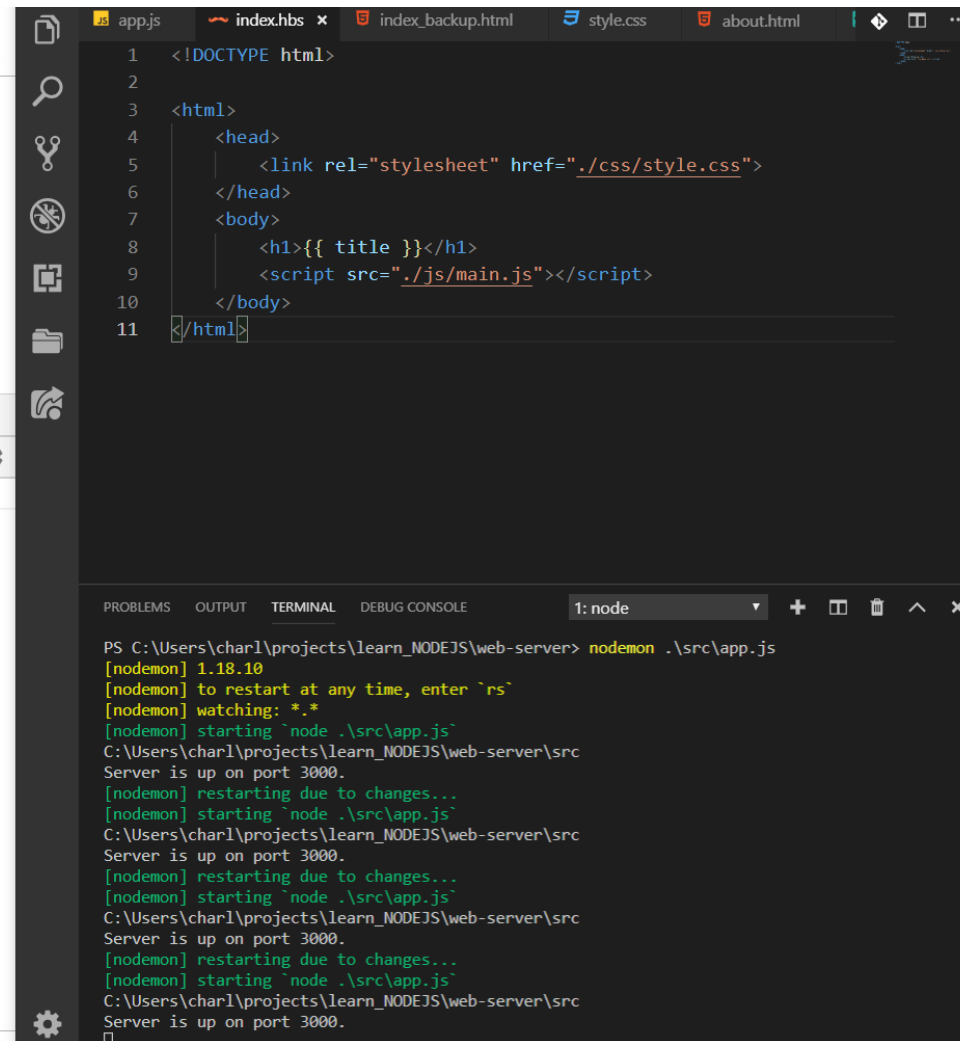
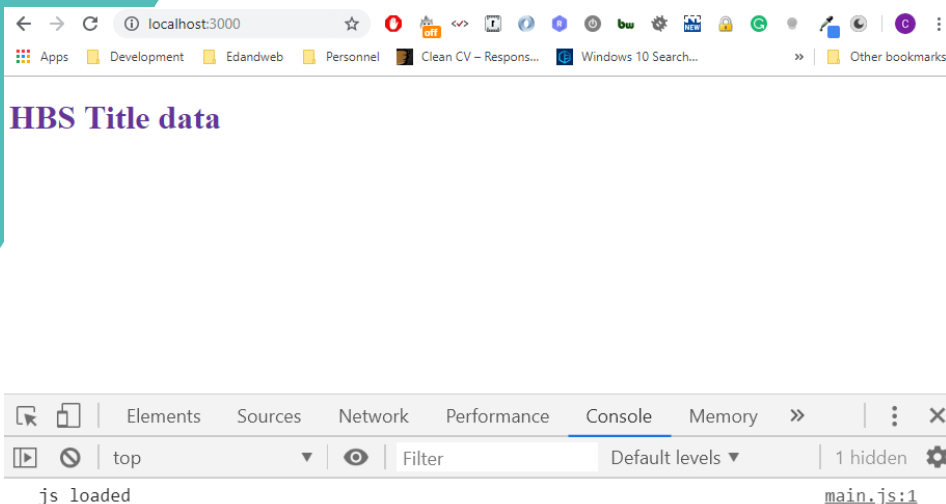


# Importer un script JS et une photo

## TP

1. Créer un script main.js dans un dossier js et importer le script dans la
2. Page index.html
3. Télécharger un 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 de 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 de la route utiliser la fonction render qui prend en paramètre le nom du template et les data (optionnel objet).

Si on souhaite changer le nom du répertoire des templates:

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

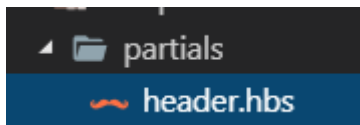


# Créer des blocs de templates partials

```
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}}
```

# Résumé

