



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 6

Promise, await, async, Async module

...



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	Building your Stack (2) <ul style="list-style-type: none"> 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. 01/07	<ul style="list-style-type: none"> Promise, await, async, Async module ... Avoiding common pitfalls with Async.js
7	Lun 08/07	<ul style="list-style-type: none"> MongoDB ½
8	Lun 15/07	<ul style="list-style-type: none"> MongoDB 2/2
9	Lun 22/07	<ul style="list-style-type: none"> Working with MongoDB 1/2
10	Lun 15/07	<ul style="list-style-type: none"> Working with MongoDB 2/2



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 5

- Retour sur les framework express et sa configuration (voir résumé cours 4)
- Utilisation des blocks pour étendre un template avec Handlebars dans base.hbs (`{{> @partial-block}}`) et dans index.hbs (`{{#> base}}Content{{/base}}`)
- Utilisation des boucles dans un template:
`{{#each peoples}}`
`Nom: {{this.name}}, Age: {{this.age}}, Country: {{this.country}}`
`{{/each}}`
- Création d'un point d'entrée d'API GET basé sur les API mapbox.com et darksky.net
- Réalisation d'un formulaire de recherche faisant appel à l'API GET pour retourner la météo en fonction de l'adresse tapée dans le navigateur



Résumé du cours 4

- Utilisation de l'API mapbox: récupération de la longitude et latitude avec une adresse
- Exercice écrire une fonction asynchrone pour gérer `add(1, 4, (sum) => {console.log})` avec un `setTimeout` de 2s
- Combinaison des fonctions `getCoordinates` et `getGeocode`
- Découverte du framework express `const app = express();` et définition d'une première route (`app.get`)
- Express adapte les Headers suivant le type de data renvoyé (ex: si objet js alors JSON)
- Servir des fichiers statiques: `app.use(express.static(publicDirectoryPath));`
- Dossiers publiques contenant tous les fichiers (JS, CSS ...) publiques
- Utilisation de hbs (handlebars pour Node.js) pour gérer les templates:

```
app.set('view engine', 'hbs');  
app.set('views', pathTemplates);  
res.render('index', {title: 'La vie est belle.'});  
{{title}}
```

Structurer des templates (header, footer ...): `hbs.registerPartials(pathPartials);`
`{{>header}}`



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
- Debugger avec Node.js `node inspect app.js` et aller dans chrome et taper `chrome://inspect/` et verifier dans configure que les adresses `localhost:9229` et `127.0.0.1:9229` sont bien renseignées.
- 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`



Working with the Event loop

<https://snyk.io/blog/nodejs-how-even-quick-async-functions-can-block-the-event-loop-starve-io/>

snyk

Test

Features ▾

Vulnerability DB

Pricing

Docs

Company ▾

Schedule a Demo

Log in

Sign Up

First, let's block the Event-Loop!

Note:

- Examples below use Node 10.9.0 on an Ubuntu 18.04 VM with 4 cores, running on a MacBook Pro 2017
- You can play with the code below by cloning <https://github.com/michael-go/node-async-block>

So let's start with a simple Express server:

```
1 const express = require('express');
2
3 const PID = process.pid;
4
5 function log(msg) {
6   console.log(`[${PID}]`, new Date(), msg);
7 }
8
9 const app = express();
10
11 app.get('/healthcheck', function healthcheck(req, res) {
12   log('they check my health');
13   res.send('all good!\n');
14 });
15
16 const PORT = process.env.PORT || 1337;
17 let server = app.listen(PORT, () => log('server listening on :' + PORT));
```

and now let's add a nasty event-loop blocking endpoint:

```
1 const crypto = require('crypto');
2
3 function randomString() {
4   function randomBytes(n) {
5     const crypto = require('crypto');
6     return crypto.randomBytes(n).toString('hex');
```

and now let's add a nasty event-loop blocking endpoint:

```
1 const crypto = require('crypto');
2
3 function randomString() {
4   function randomBytes(n) {
5     const crypto = require('crypto');
6     return crypto.randomBytes(n).toString('hex');
```



{web}
school}

Publishing your own package

<https://hackernoon.com/publish-your-own-npm-package-946b19df577e>

<https://docs.npmjs.com/packages-and-modules/contributing-packages-to-the-registry>

The screenshot shows the Hacker Noon website interface. At the top, there's a navigation bar with the Hacker Noon logo and links for LATEST, TOP, CRYPTO, 2.0, COMMUNITY, POD, and GET PUBLISHED. Below the navigation bar, the article title 'Hacker Noon' is displayed, followed by the tagline 'how hackers start their afternoons.' and a 'Follow' button. The article content begins with the command `npm login` in a code block. The text explains that entering a username and password will store credentials for future publishes. The first step is to run `npm publish`. The article then explains that publishing a package to the NPM registry takes less than a minute, and provides a link to check the package: `https://www.npmjs.com/~(username)/(package-name)`. It also mentions that version changes require updating the version number and republishing. The article concludes by reminding users to use `npm version patch`, `npm version minor`, and `npm version major` for automatic updates based on semantic versioning. At the bottom of the article, there are two promotional banners for Hacker Noon, each with a 'GET UPDATES' button.

1. Create an account in npmjs.com.

Sign in [Get started](#)

HACKERNOON LATEST TOP CRYPTO 2.0 COMMUNITY POD GET PUBLISHED

Hacker Noon
how hackers start their afternoons.

[Follow](#)

688

`npm login`

Enter your username and password. This will store the credentials so you don't have to enter it for every publish.

1. Now to publish, run

`npm publish`

This will publish your package to NPM registry. Once publish completes (in less than a minute), you can go check your package in the link [https://www.npmjs.com/~\(username\)/\(package-name\)](https://www.npmjs.com/~(username)/(package-name)).

If you want to make changes to your package, you have to change the version number and publish again.

Remember to use npm commands `npm version patch`, `npm version minor` and `npm version major` to update the version automatically rather than manually updating them. These commands are based on [semantic versioning](#).

Never miss a story from **Hacker Noon**, when you sign up for Medium. [Learn more](#) [GET UPDATES](#)

Never miss a story from **Hacker Noon**, when you sign up for Medium. [Learn more](#) [GET UPDATES](#)



Les promises

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise

Définition (MDN):

L'interface Promise représente un intermédiaire (proxy) vers une valeur qui n'est pas nécessairement connue au moment de sa création. Cela permet d'associer des gestionnaires au succès éventuel d'une action asynchrone et à la raison d'une erreur. Ainsi, des méthodes asynchrones renvoient des valeurs comme les méthodes synchrones, la seule différence est que la valeur retournée par la méthode asynchrone est une promesse (d'avoir une valeur plus tard).

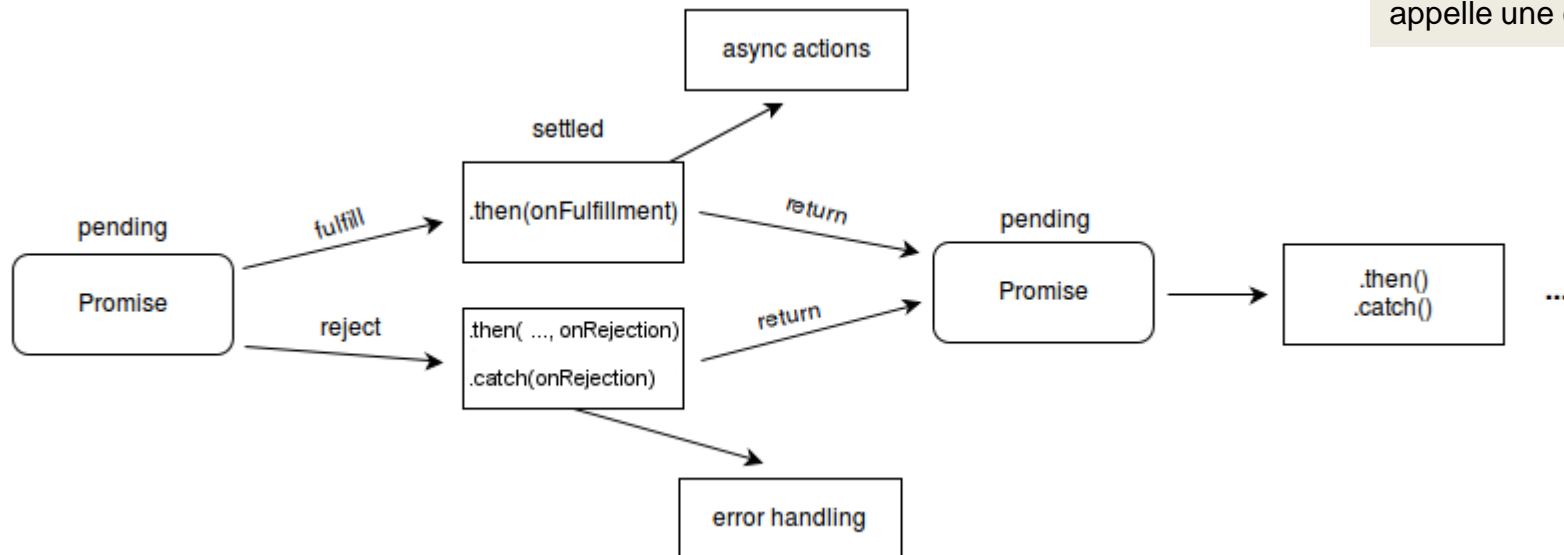
Une Promise est dans un de ces états :

- **pending (en attente)** : état initial, la promesse n'est ni remplie, ni rompue ;
- **fulfilled (rempli/tenue)** : l'opération a réussi ;
- **rejected (rejeté/rompue)** : l'opération a échoué ;
- **settled (réglé/acquittée)** : la promesse est tenue ou rompue mais elle n'est plus en

On pourra utiliser les méthodes **.then** et **.catch** sur le résultat de la promesse:

- fulfilled => **.then**
- rejected => **.catch**

Les méthodes `Promise.prototype.then()` et `Promise.prototype.catch()` renvoient des promesses et peuvent ainsi être chaînées. C'est ce qu'on appelle une composition.



Promise: Example

```
var promise1 = new Promise(function(resolve, reject) {  
    setTimeout(function() {  
        resolve('foo');  
    }, 300);  
});  
promise1.then(function(value) {  
    console.log(value);  
    // expected output: "foo"  
});  
  
console.log(promise1);  
// expected output: [object Promise]
```



Transformons notre code avec des promises (weather and geocodes)

TP

1. Pour chacun des modules (se trouvant dans utils) nous allons transformer le code pour que le code retourne des promises
2. Utiliser `.then` et `.catch` pour résoudre les promises

Les opérateurs Async et Await

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/async_function

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/await>

Async:

La déclaration `async function` définit une fonction asynchrone qui renvoie un objet `AsyncFunction`. Une fonction asynchrone est une fonction qui s'exécute de façon asynchrone grâce à la boucle d'évènement en utilisant une promesse (`Promise`) comme valeur de retour.

On peut également définir des fonctions asynchrones grâce au constructeur `AsyncFunction` et via une expression de fonction asynchrone.

Await:

L'opérateur `await` permet d'attendre la résolution d'une promesse (`Promise`). Il ne peut être utilisé qu'au sein d'une fonction asynchrone (définie avec l'instruction `async function`).

```
function resolveAfter2Seconds() {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve('resolved');
        }, 2000);
    });
}

async function asyncCall() {
    console.log('calling');
    var result = await resolveAfter2Seconds();
    console.log(result);
    // expected output: 'resolved'
}

asyncCall();
```



Transformons notre code avec les opérateurs Async et Await

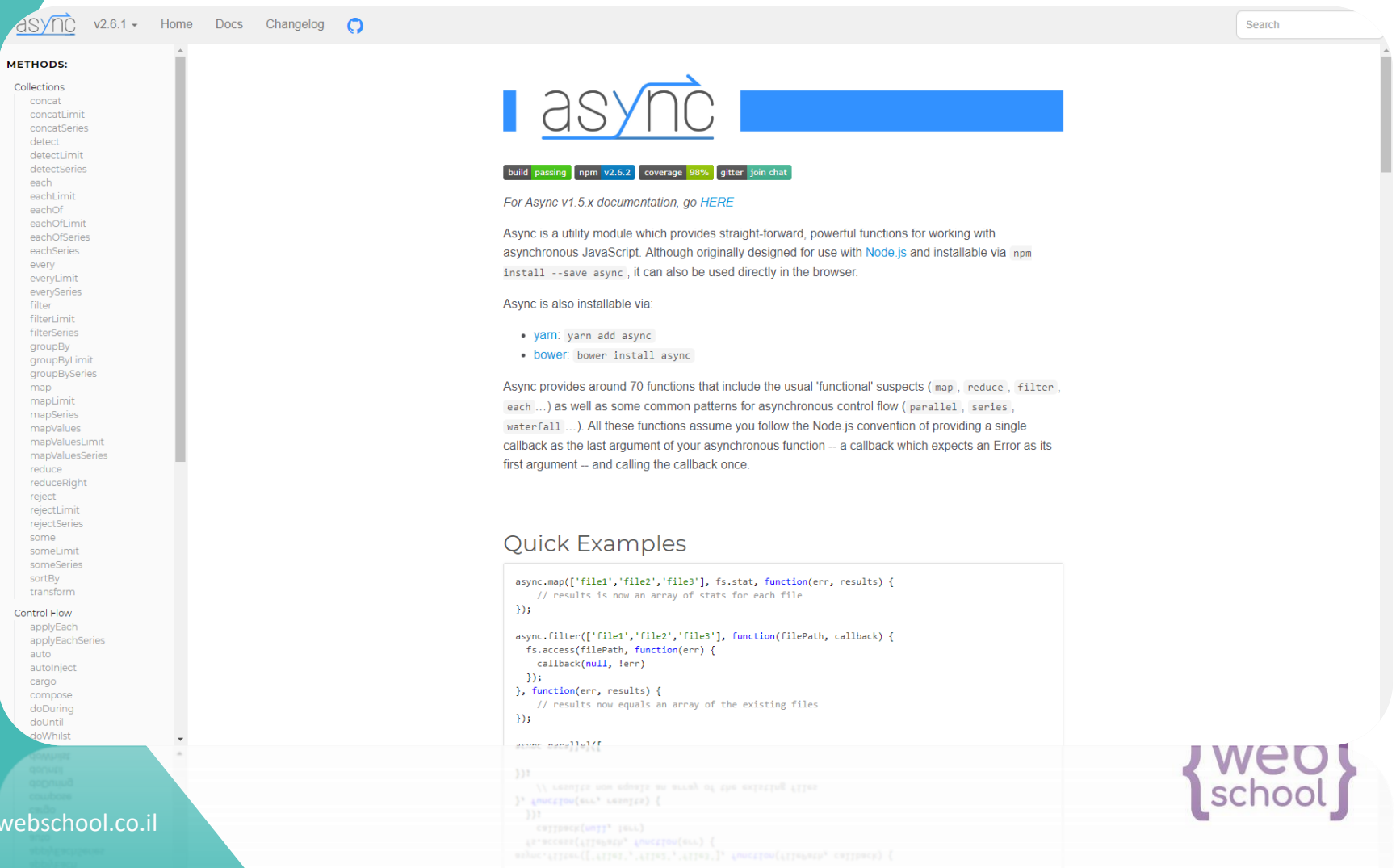
TP

1. Remplacer les méthodes `.then` et `.catch` en utilisant `async` et `await`

Async.js

<https://caolan.github.io/async/>

<https://medium.com/velotio-perspectives/understanding-node-js-async-flows-parallel-serial-waterfall-and-queues-6f9c4badbc17>



The screenshot shows the Async.js documentation page. The top navigation bar includes the Async.js logo, version 2.6.1, and links for Home, Docs, Changelog, and GitHub. A search bar is located on the right. The left sidebar lists various methods under 'Collections' and 'Control Flow'. The main content area features the Async.js logo, build status (build passing, npm v2.6.2, coverage 98%, gitter, join chat), and a link to the v1.5.x documentation. The text describes Async as a utility module for asynchronous JavaScript, installable via npm or directly in the browser. It also lists installation methods using yarn and bower. The 'Quick Examples' section shows code snippets for using Async.map, Async.filter, and Async.parallel.

METHODS:

- Collections**
 - concat
 - concatLimit
 - concatSeries
 - detect
 - detectLimit
 - detectSeries
 - each
 - eachLimit
 - eachOf
 - eachOfLimit
 - eachOfSeries
 - eachSeries
 - every
 - everyLimit
 - everySeries
 - filter
 - filterLimit
 - filterSeries
 - groupBy
 - groupByLimit
 - groupBySeries
 - map
 - mapLimit
 - mapSeries
 - mapValues
 - mapValuesLimit
 - mapValuesSeries
 - reduce
 - reduceRight
 - reject
 - rejectLimit
 - rejectSeries
 - some
 - someLimit
 - someSeries
 - sortBy
 - transform
- Control Flow**
 - applyEach
 - applyEachSeries
 - auto
 - autoInject
 - cargo
 - compose
 - doDuring
 - doUntil
 - doWhilst

async

build passing npm v2.6.2 coverage 98% gitter join chat

For Async v1.5.x documentation, go [HERE](#)

Async is a utility module which provides straight-forward, powerful functions for working with asynchronous JavaScript. Although originally designed for use with [Node.js](#) and installable via `npm install --save async`, it can also be used directly in the browser.

Async is also installable via:

- yarn: `yarn add async`
- bower: `bower install async`

Async provides around 70 functions that include the usual 'functional' suspects (`map`, `reduce`, `filter`, `each` ...) as well as some common patterns for asynchronous control flow (`parallel`, `series`, `waterfall` ...). All these functions assume you follow the Node.js convention of providing a single callback as the last argument of your asynchronous function -- a callback which expects an `Error` as its first argument -- and calling the callback once.

Quick Examples

```
async.map(['file1','file2','file3'], fs.stat, function(err, results) {
  // results is now an array of stats for each file
});

async.filter(['file1','file2','file3'], function(filePath, callback) {
  fs.access(filePath, function(err) {
    callback(null, !err)
  });
}, function(err, results) {
  // results now equals an array of the existing files
});

async.parallel([
  // ...
]);
```

Connect to mysql

<https://www.npmjs.com/package/mysql>

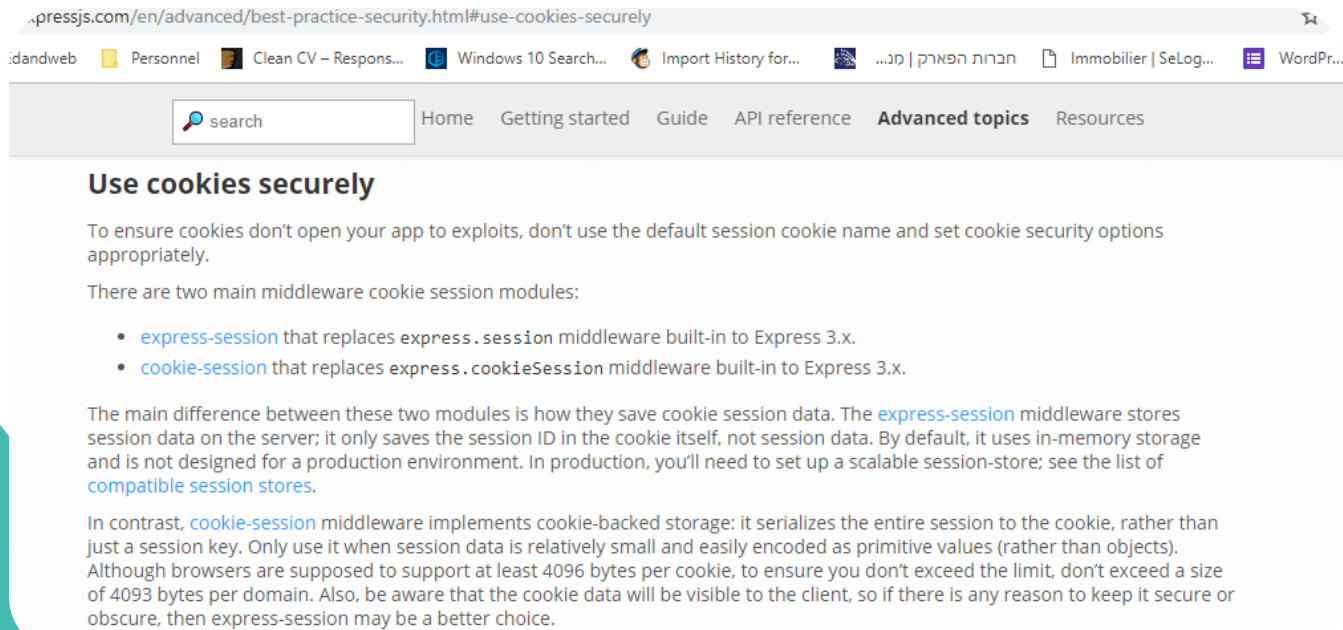
```
npm install mysql
```

```
var mysql = require('mysql');
var connection = mysql.createConnection({
  host : 'localhost',
  user : 'me',
  password : 'secret',
  database : 'my_db'
});
connection.connect();
connection.query('SELECT 1 + 1 AS solution', function (error, results, fields) {
  if (error) throw error;
  console.log('The solution is: ', results[0].solution);
});
connection.end();
```



Express

- Exemple complet pour la mise en place de session avec express:
<https://appdividend.com/2018/02/05/express-session-tutorial-example-scratch/>
- Exemple complet authentication:
<https://www.sitepoint.com/local-authentication-using-passport-node-js/>
- La gestion des sessions :
<https://stackoverflow.com/questions/8749907/what-is-a-good-session-store-for-a-single-host-node-js-production-app>



The screenshot shows the Express.js documentation page for 'Use cookies securely'. The page is in English and is part of the 'Advanced topics' section. It discusses how to ensure cookies don't open your app to exploits by not using the default session cookie name and setting cookie security options appropriately. It mentions two main middleware modules: `express-session` and `cookie-session`. `express-session` stores session data on the server, while `cookie-session` serializes the entire session to the cookie. The page also notes that browsers are supposed to support at least 4096 bytes per cookie, but it's better to keep it secure or obscure.

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.



Fin

