# Creating a Node.js web app

In this document, we'll be looking at how we can use Node.js (along with some other Node.js modules) to create a simple website. The main things we'll be looking at are:

1. Using npm to install packages.
2. Using Express to deal with site routing (i.e. routing appropriately for different URL paths).
3. Using templates (Pug) to render different views depending on the page route.

This document explains the main points you should be aware of and are based on the following tutorial (with extra commentary):

https://auth0.com/blog/create-a-simple-and-stylish-node-express-app/

Let's go over npm and Express before the example.

## *npm*

> *Resource*
>
> https://nodesource.com/blog/an-absolute-beginners-guide-to-using-npm/

npm is the package manager for Node.js. It is included when you install Node.js and is used to install modules you want to use with your Node.js app. When you initialize your project using npm it creates a ***package.json*** file which contains important info about your project—mainly, the name, version, dependencies and any scripts you want to be able to run. (The scripts are useful to run other commands—especially during the development process.)

### *npm init*

This command (in CLI) will initialize your project and create a ***package.json*** file. You will be prompted for some initial information about your project (e.g. the project name).

### *npm install* <*package_name*>
***(****short form:* ***npm i*** <*package_name*>***)***

This will install a package in your project inside a folder named ***node_modules***. There are a couple flags you can use with your install command.

*`--global (-g)`*

You can install commonly used dev tools globally as well by adding a `-g` flag.

For example, to install Express globally: `npm i -g express`

*`--save`*

This flag adds the module you're installing as a dependency in the ***package.json*** file. This is useful because then you don't need to copy the ***node_modules*** folder (e.g. to your GitHub repo) and whoever

downloads and runs the project can then just get the main files then run `npm install` to download and install all identified dependencies.

*`--save-dev (-D)`*

This is similar to –save but adds the module to devDependencies in ***package.json***. These are dependencies for **developing** the app but not for running (i.e. it's only of use for devs).

## *Express*

Express is a web application framework. As we saw in the *Intro to Node.js concepts* (server example), using Node.js's built-in http module can be a little onerous, especially because you need to deal with handling server codes and so on. It's also a little more difficult when dealing with routing pages and files. This is where Express comes in.

Once you have Express installed, in your Node.js app file (e.g. ***index.js***) you can use Express by including at the top:

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

The `require('express')` is what imports the Express module and then we store the Express app object in an `app` variable. This (`app`) is what we'll be using for our various routing calls and server listening.

**Example:** The following line of code will define what's happening when the "/" (or, index page) of the site gets a GET request (i.e. when the page is loaded because when you load a page your making a GET request of the page).

```
app.get("/", (req, res) => {
  //do something when the index page is loaded (e.g. http://yoursite.com/)
});
```

You can add more routes by adding more `app.get()` lines but with different routes as the first argument (e.g. "`/about`").

## *Pug template engine*

Pug is a templating engine. Express supports many templating engines but Pug is the default so it's easy to start with. To set everything up:

1. Install Pug:
   - `npm i pug`
2. Create a ***views*** folder in the root of your project. This is the folder where your template files will reside (***views*** is a conventional name).
3. In your main app file (usually, ***index.js***) set the views directory using the following line of code:

   ```
   app.set("views", path.join(__dirname, "views"));
   ```

   The `path.join()` is the path to the views folder relative to the root of the app (`__dirname`). **Note the double underscore!** (The `path` module is needed to use `path.join()`.)

4. Set the templating engine to Pug:

```
app.set("view engine", "pug");
```

A template (***.pug*** file) allows you to create a skeleton of your page content. For dynamic values you can pass variables. You can set HTML element classes, ids and attributes as well. For the HTML structure, there are no tags in the traditional HTML sense so the structure is determined by indenting (i.e. an indent means it's a child/is nested).

**For example:**

```
doctype html
html(lang="en")
  head
    meta(charset="utf-8")
    title Home
    link(rel="stylesheet", href="/assets/css/styles.css")
  body
    header#header
      h2#site-name
        a(href="/") Site name
    main#id
      h1.page-title Welcome
      p Some random paragraph
```

There are a few important syntax structures to be aware of.

***<element>*** *<content>*

To define an element and its contents (i.e. content within opening and closing tags), write the element name followed by a space and then the contents.

You can add classes or an id to an element by using the **.** or **#** symbols, respectively.

***<element>****(attr="value")*

To define an element's attributes, add the attributes and their values in parentheses following the element name. **There should be NO space between the element name and the parentheses.** You can add multiple attributes by separating each attribute name-value pair with commas (see template example).

*More about Pug*

- https://pugjs.org/api/getting-started.html (complete syntax reference under ***Language Reference*** in the sidebar)
- https://www.iamtimsmith.com/blog/using-the-pug-templating-engine-part-2-logic
- https://www.iamtimsmith.com/blog/using-the-pug-templating-engine-part-3-layouts

## *Build a Node.js web app with Express and Pug*

Let's try out our first app.

First initialize the project and install Express.

1. Create your project folder.
2. Open your CLI and browse to your project folder.
3. Initialize the project (add whatever values as needed):
   - `npm init`
4. Install the Express module:
   - `npm i express`
5. Create an ***index.js*** file. This will serve as the main file for your Node.js app. This is where you'll import Express and set up your routes.
6. Add the following to the ***index.js*** file:

```
//import required modules
const express = require("express");
const path = require("path");

//set up Express object and port
const app = express();
const port = process.env.PORT || "8888";

//test message
app.get("/", (req, res) => {
  res.status(200).send("Test page");
});

//set up server listening
app.listen(port, () => {
  console.log(`Listening on http://localhost:${port}`);
});
```

   Notice the backticks in the console log. This allows you to use template literals (e.g. `${port}`) to print the variable value directly in the string.

7. Set up Nodemon (technically not necessary, but very useful).
   a. In the terminal: `npm i --save-dev nodemon`
   b. Add a `dev` script to ***package.json***. We can run this with npm run to run nodemon. In the "scripts" key, add:

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

   This creates an alias for the nodemon command, so we can enter `npm run dev` in the CLI to run the script.

> *Nodemon*
>
> Nodemon will watch for changes in the code file identified (in the above line, in ***index.js***). If there are changes, restart the Node.js server. If not using Nodemon, you'll need to stop the server and start it again whenever you change the Node app code.

c. In the CLI, run the dev script to auto-restart the Node.js server upon code changes:

   - `npm run dev`

8. Open the browser and go to *http://localhost:8888*. You should see the text "Test page". (If you have not installed Nodemon, you'll need to run the app with `node index.js`.)
9. Point your routes to your templates.
   a. Install Pug. In your terminal, type:
      - `npm i pug`
   b. Create a ***views*** folder in the root of your project directory.
   c. Set up the Express object to know where your *views* folder is located and which templating engine is being used. Add in the ***index.js*** file (above the routing code):

```
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "pug");
```

   d. Create a ***layout.pug*** file in the ***views*** folder. This will be used for the main layout template. (You can use a different name if you want.) Notice that the #{title} will be replaced with a title variable passed from ***index.js***.

```
block variables
doctype html
html
  head
    meta(charset="utf-8")
    title #{title} | Test website
  body
    header#header
      div.header-content
        h2#site-name
          a(href="#") Test site
    main#main
      block layout-content
    footer#footer
      p &copy; Copyright HTTP5211, 2022.
```

   e. Notice that ***layout.pug*** only contains **common** styles, but for the main content (or other content variable by page), there is a `block layout-content`. A block named "layout-content" can be defined in other template files. This is how we can change the content for each "page". Create an ***index.pug*** template file in the ***views*** folder. This will hold our main content for the home page.

```
extends layout

block layout-content
  div.content
    h1.page-title About us
    p Some content about us...
    p A random paragraph
```

The `extends layout` will extend the ***layout.pug*** file (notice that `extends` uses the filename minus the ***.pug***). The `block layout-content` (notice the same name as in ***layout.pug***) defines the content for this particular block for the **index** view.

f. Add the route in ***index.js*** (you can remove the `res.status()` stuff):

```
app.get("/", (req, res) => {
  res.render("index", { title: "Home" });
});
```

The `res.render()` line is where the magic happens. It renders an index view (looks for the matching template name in the ***views*** folder minus ***.pug***) and passes along a variable named "title" (used in the ***layout.pug*** template file).

g. Refresh in the browser (or restart the app if Nodemon is not installed). You should now see the content as defined in the templates.

10. Add CSS or JS files (client-side scripts).
    a. Create a folder for holding static files (e.g. CSS, JS, image files). Typically, this is named ***public***, so we'll use the same name, though it can be any name you want.
    b. Set up your path to the ***public*** folder in your Node app to be used for static files. In ***index.js*** add:

```
app.use(express.static(path.join(__dirname, "public")));
```

    c. Add a CSS file in the ***public*** folder (as a best practice, organize your files in the folder). Try creating a ***styles.css*** file in ***public/css/***.
    d. Link to the CSS file in ***layout.pug*** by adding the following within head:

```
link(rel="stylesheet", href="/css/styles.css")
```

Notice that the "***public***" is not part of the href. The "/" starts from ***public*** for static files.

    e. Add some styling and refresh in the browser. You should see your changes.
    f. To add more "pages", you can create another template file for another page and add a route (`app.get()` with `res.render()`) in ***index.js***.

*Browsersync*

When developing, even with Nodemon, you'll need to refresh in the browser to see coding changes because Nodemon only restarts the server. To auto-refresh (live sync), you need Browsersync.

Follow the instructions for adding Browsersync from the tutorial linked at the start of this document: https://auth0.com/blog/create-a-simple-and-stylish-node-express-app/#Add-Live-Reload-to-Express-Using-Browsersync

If `browser-sync init` doesn't work after installing (command not found/recognized), try installing globally or adding to your PATH environment variable.

**Consider:** Look up how to deal with server errors (e.g. 404 error) and how to serve up an error page with Express.