Introduction to Node.js: Using Server-Side JavaScript

Nathalie Acosta ‘25
What is Node.js and why should I use it?

- Node.js is a **JavaScript runtime environment** that lets you run JavaScript outside of the browser.
- It’s often used to **build back-end services** (your server).
- **Application Programming Interfaces** (APIs) are a prominent feature of Node.js.
Pros of Node.js

- It’s best to build **highly-scalable, data-intensive, real-time** applications
- Can use **JavaScript** everywhere (client-side and server-side)
- **Large ecosystem** of **open-source** libraries and packages
Let’s build a to do app!
Let’s get started!

- Navigate to https://code.cs50.io/
- Make a new directory called todo
- Type node --version to make sure node is installed
- Create app.js
Run Node

- Let’s test that it’s working. Type the following:

```javascript
console.log("hello CS50!");
```

- Run it in the terminal with `node app.js`
Creating a server

- One of the most popular Node packages is called **Express** (it has 22,000,000 weekly downloads!)

- It is a **minimal web application framework** that helps us set up a server and create APIs
Setting up package.json

- First, let’s set up our `package.json`
- Package.json is a universal file in Node.js that contains metadata about the Node packages installed, the project name and description, and other details
- Run `npm init` to get one!
Creating a simple server with Express, part 1

- Now we can install **NPM (Node Package Manager) packages**!
- Run `npm install express` to install Express
  - You can read more about Express on their website [https://expressjs.com/](https://expressjs.com/)
- This **adds stuff to package.json and package-lock.json** (a more specific version of package.json we will not touch) and also creates the folder full of our installs called `node_modules`
- To let `app.js` know to use the Express module, we have to use:

```javascript
const express = require("express");
```
Now to get the actual web server going, we'll have to type the following code:

```javascript
const app = express(); // Calls the express function and puts the new Express app inside the variable app
const port = 3000; // Defines the port to listen on
app.listen(port, () => {
  console.log(`CS50 app listening on port ${port}`); // Listens for the app on the port and does whatever inside the brackets. () => { *does stuff here* }
});
```
Where is my server?

- Look for the link to your new Express web server under **PORTS** in the terminal (right click and click ports if you hid it before)
- However, **this is annoying** to do every time, so let’s install another great NPM package that helps us in our development environment (no need to re-open each time!)
- Run `npm install nodemon`
- In package.json, **let’s type up a script** we only need to run once!

  "dev": "nodemon app.js"
Routes in Express

- `app.METHOD(PATH, HANDLER)` is the general pattern for how to handle client requests to a particular endpoint (whether this be via GET request, POST request, etc.)
- First things first—we need to set up a GET request for our homepage! Let’s type the following:

  ```javascript
  app.get('/', (req, res) => {
    res.send('Hello World!')
  })
  ```
  Sends the text “Hello World!”
How to serve up files using Express

- We can create html files and use `res.sendFile`, but we want to serve dynamic content.
- So let's use a templating engine! A well known one is **Pug** (it also sounds cute)
  - Their documentation can be found at [https://pugjs.org/api/getting-started.html](https://pugjs.org/api/getting-started.html)
- Run `npm install pug`
- Write the code:
  ```javascript
  app.set('view engine', 'pug')
  ```

This sets the templating/view engine for Express as Pug (there are others so it’s important to specify). This is like Flask but Javascript!
Let’s create a **views** directory with our files, and call our homepage `index.pug`

Pug’s syntax can look a bit strange, but it is easy to familiarize yourself with it

  - (and you don’t have to use Pug if you don’t want to!)

```
html

head

  title= title

body

  h1= message
```
Now we can give information from our Express/Node server directly to our homepage! Here's the code:

```javascript
app.get('/', (req, res) => {
  res.render('index', { title: 'Hey', message: 'Hello there!' })
})
```
Request from the frontend

- First, let’s **build our simple form**
- This is how it looks like in Pug
- We need to make sure Express can read our request in a JSON (**JavaScript Object Notation: a comma-separated key:value list**) format:

```javascript
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
```
Receive the request from the backend

- Now in our backend, we have to get the request that was posted
- Let's confirm it with

  ```javascript
  app.post('/', (req, res) => {
    console.log(req.body);
  })
  res.redirect('/');
  ```

- Hurray! Our frontend is sending a request to our backend. Let's send a response back
Create a SQLite3 database

- Time to store the data inside a SQLite database! Run `npm install sqlite3`

- Run `sqlite3 todos.db`

  ```
  CREATE TABLE todo(
    name TEXT NOT NULL);
  ```
Store our request inside the database

- First, have to **connect to our database**

  ```javascript
  const sqlite3 = require('sqlite3').verbose();
  const db = new sqlite3.Database('./todos.db');
  ```

- Then, we can **insert into our database** (or run any command!)

  ```javascript
  let todo = req.body.todo;
  db.run("INSERT INTO todos(name) VALUES(?)", todo);
  ```
Send a response back to the frontend

```javascript
app.get('/', (req, res) => {
  let todolist = [];
  db.each('SELECT name FROM todos', (err, row) => {
    if (err) {
      console.log(err);
    } else {
      console.log(row);
      todolist.push(row.name);
    }
  }, (err) => {
    return res.render('index', { title: 'Hey', message: 'Hello there!', todolist: todolist })
  })
});
```
And finally, access the response from the frontend!

```python
for todo in todolist
    li = todo

else
    p there are no todos... yet
```
Some final notes

- **You don’t have to use SQLite if you don’t want to**; if you want to learn more, I recommend using MongoDB if you want to publish your website.
- **You don’t have to use Pug if you don’t want to**; if you want to learn more, I recommend implementing a frontend JavaScript library like React or Vue for a more professional touch.
- **Ultimately, there is no best tech stack.** If Node.js doesn’t appeal to you you can implement your final project in Flask or other framework!
Thank you for your attention!
The final version of the code is on my Github: https://github.com/nathalieacosta/todo-final