Introduction to Node.js: Using Server-Side JavaScript

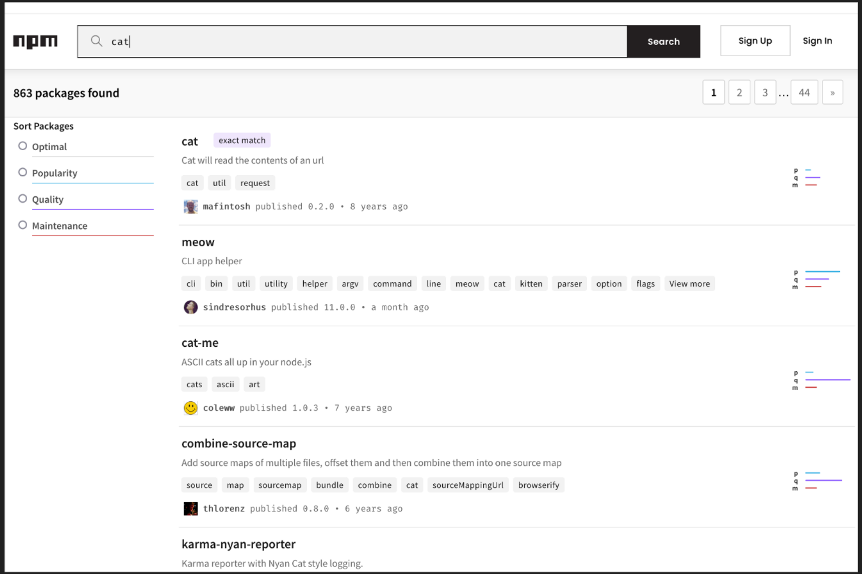
Nathalie Acosta '25

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


npm
cat
Search
Sign Up
Sign In

863 packages found
1 2 3 … 44 »

Sort Packages
Optimal
Popularity
Quality
Maintenance

cat exact match
Cat will read the contents of an url
cat util request
mafintosh published 0.2.0 • 8 years ago

meow
CLI app helper
cli bin util utility helper argv command line meow cat kitten parser option flags View more
sindresorhus published 11.0.0 • a month ago

cat-me
ASCII cats all up in your node.js
cats ascii art
coleww published 1.0.3 • 7 years ago

combine-source-map
Add source maps of multiple files, offset them and then combine them into one source map
source map sourcemap bundle combine cat sourceMappingUrl browserify
thlorenz published 0.8.0 • 6 years ago

karma-nyan-reporter
Karma reporter with Nyan Cat style logging.

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

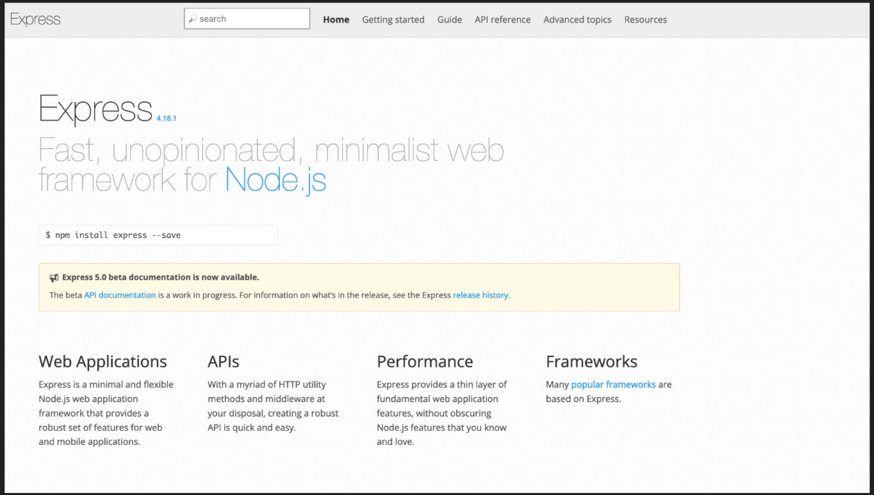
- Let’s test that it’s working. Type the following:

```
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



Express 4.18.1

Fast, unopinionated, minimalist web framework for Node.js

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

Express 5.0 beta documentation is now available.

The beta API documentation is a work in progress. For information on what's in the release, see the Express release history.

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.

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/
- 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:

```
const express = require(“express”);
```

Creating a simple server with Express, part 2

- Now to get the actual web server going, we'll have to type the following code:

```
const app = express();

const port = 3000;

app.listen(port, () => {

console.log(`CS50 app listening on port ${port}`);

});
```

Calls the express function and puts the new Express app inside the variable app

Defines the port to listen on

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:

```
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
- Run `npm install pug`
- Write the code:

Sets the templating/view engine for Express as Pug (there are others so it’s important to specify). This is like Flask but Javascript!

```
app.set('view engine', 'pug')
```

Working with Pug, part 1

- 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
```

Working with Pug, part 2

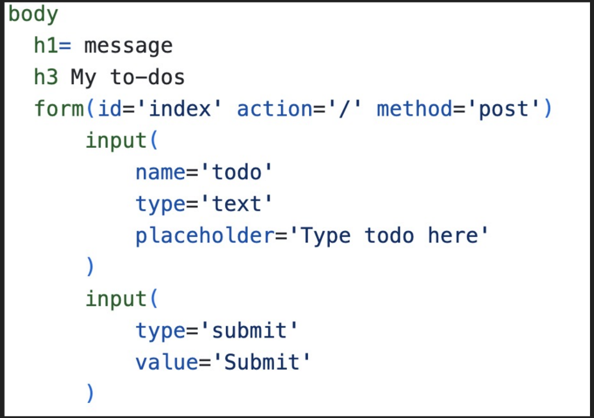
- Now we can give information **from our Express/Node server directly to our homepage**! Here's the code:

```
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:

```
app.use(express.urlencoded({ extended :
true}));

app.use(express.json());
```



```
body
  h1= message
  h3 My to-dos
  form(id='index' action='/' method='post')
      input(
          name='todo'
          type='text'
          placeholder='Type todo here'
      )
      input(
          type='submit'
          value='Submit'
      )
```

Receive the request from the backend

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

```
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`
    - Documentation at https://github.com/TryGhost/node-sqlite3/wiki/API
- Run `sqlite3 todos.db`

```
CREATE TABLE todo(
name TEXT NOT NULL);
```

Store our request inside the database

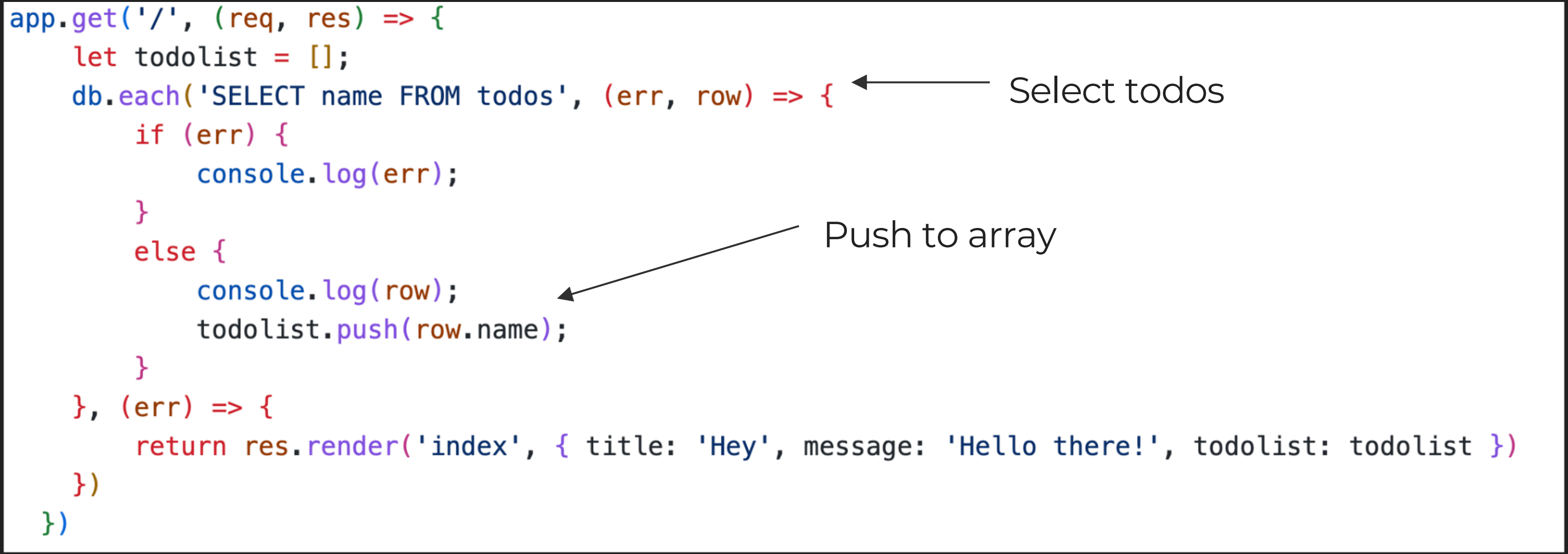
- First, have to **connect to our database**

```
const sqlite3 = require('sqlite3').verbose();

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

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

```
let todo = req.body.todo;

db.run("INSERT INTO todos(name) VALUES(?)", todo);
```

Send a response back to the frontend



```
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 })
    })
  })
```

Select todos

Push to array

Render index with the todolist variable

And finally, access the response from the frontend!

```
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