Build a CRUD app using Node, Express and MongoDB

2/03/2019

Presented by: Hassan Yakefujiang,

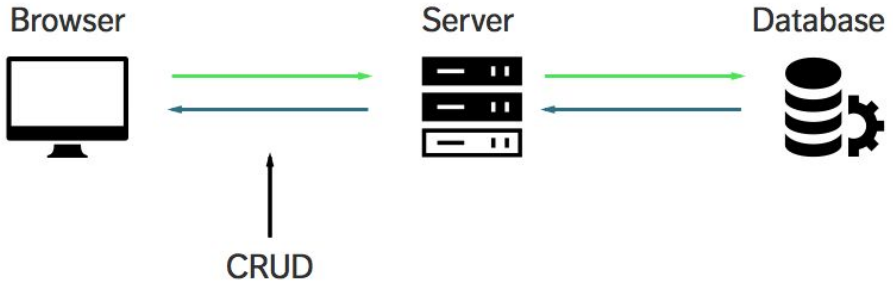
Web Development at VT

Tech Stack

- Node.js
  - JavaScript run-time
- Express
  - Framework for building web applications on top of Node.js
- MongoDB
  - A database (a place to store information)

What is CRUD?

- CRUD (Create, Read, Update, Delete)
- What each operation does:
  - Create (POST) - Make something
  - Read (GET) - Get something
  - Update (PUT) - Change something
  - Delete (DELETE) - Remove something



What we're building?

- A simple list application that allows you to keep track of things within a list (e.g. Todo List)
- Demo

Let’s get started!

1. Go into a folder where you want to work at
2. Open up your command line and run the following code:

Create a node project using the following command:

- Just hit enter through everything that you find confusing.
- You can put your name for the “author” field.

Running Node for the first time in your life

- The simplest way to use node is to run the “node” command, and specify a path to a file.
- Let’s create a file called “server.js” to run node with:

```
Hassan@DESKTOP-HASSAN98 MINGW64 ~/Desktop/quotes_app
$ touch server.js
```

When we execute the server.js file, we want to make sure it's running properly.
To do so, simply write a console.log statement in server.js:

server.js

```
1   console.log('May Node be with you');
```

- Now, run "node server.js" in your command line
- You should see the statement logged:

```
Hassan@DESKTOP-HASSAN98 MINGW64 ~/myWorkshop/WDVT/Node/quotes_app
$ node server.js
May Node be with you
```

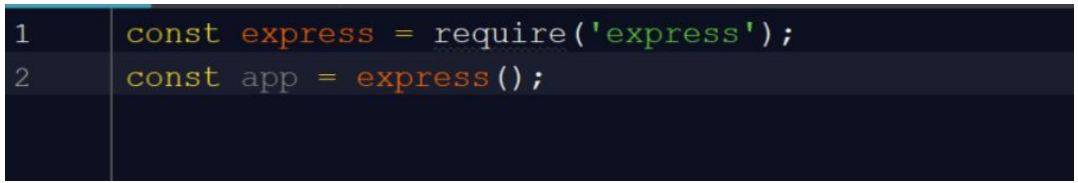
Using Express

- We first have to install Express before we can use it in our application.
- Installing Express is pretty easy.
- All we have to do is run an install command with Node package manager (npm), which comes bundled with Node.

```
Hassan@DESKTOP-HASSAN98 MINGW64 ~/myWorkshop/WDVT/Node/quotes_app
$ npm install express --save
```

- Once you're done, you should see that npm has saved Express as a dependency in package.json.

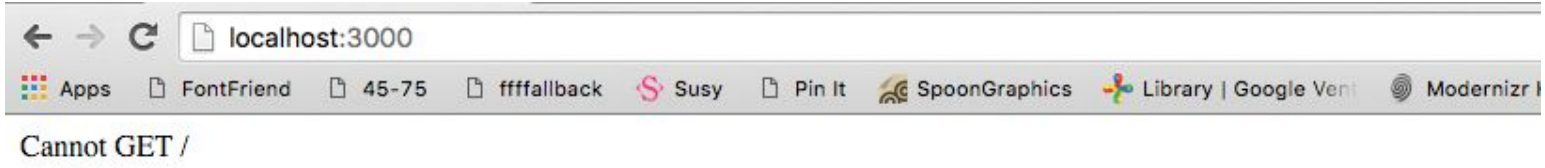
- Next, we use express in "server.js" by requiring it.



- The first thing we want to do is to create a server where browsers can connect to.
- We can do so with the help of a listen method provided by Express:

```
4   const port = 3000;
5
6   app.listen(port, function() {
7     console.log(`Server running on ${port}`)
8   });
```

- Now, run node server.js and navigate to localhost:3000 on your browser.
- You should see a message that says “cannot get /”.



CRUD – READ

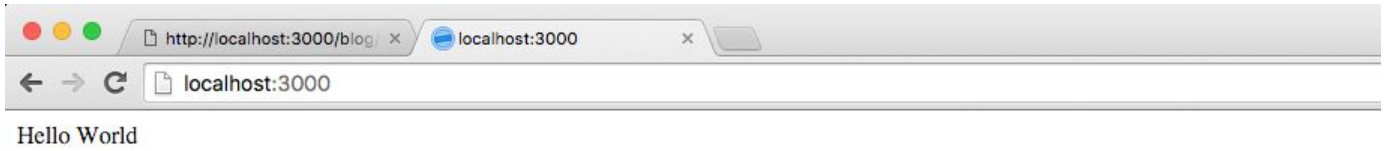
- The READ operation is performed by browsers whenever you visit a webpage.
- Under the hood, browsers send a GET request to the server to perform a READ operation.
- The reason we see the “cannot get /” error is because we have yet to send anything back to the browser from our server.

- In Express, we handle a GET request with the get method:

- I’m going to start writing in ES6 code and show you how to convert to ES6 along the way as well.
- First off, I’m replacing function() with an ES6 arrow function.
- The below code is the same as the previous code:

```
10  app.get('/', (req, res) => {
11      res.send('Hello World')
12  });
```

- Now, restart your server by doing the following:
  - Stop the current server by hitting CTRL + C in the command line.
  - Run node server.js again.



- Great. Let’s change our app so we serve an index.html page back to the browser instead.
- To do so, we use the “sendFile” method that’s provided by the “res” object.

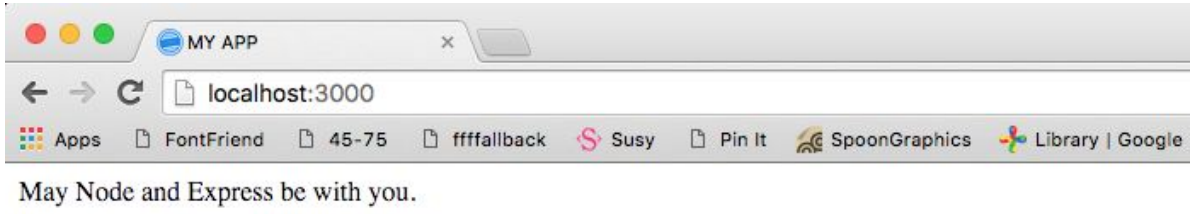
```
 9
10  app.get('/', (req, res) => {
11      res.sendFile( path: __dirname + '/index.html')
12  });
```

- Now create a simple HTML file named “index.html”:

```
Hassan@DESKTOP-HASSAN98 MINGW64
$ touch index.html
```

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <title>MY APP</title>
6   </head>
7   <body>
8       May Node and Express be with you.
9   </body>
10  </html>
```

- Restart your server and refresh your browser.
- You should be able to see the results of your HTML file now.



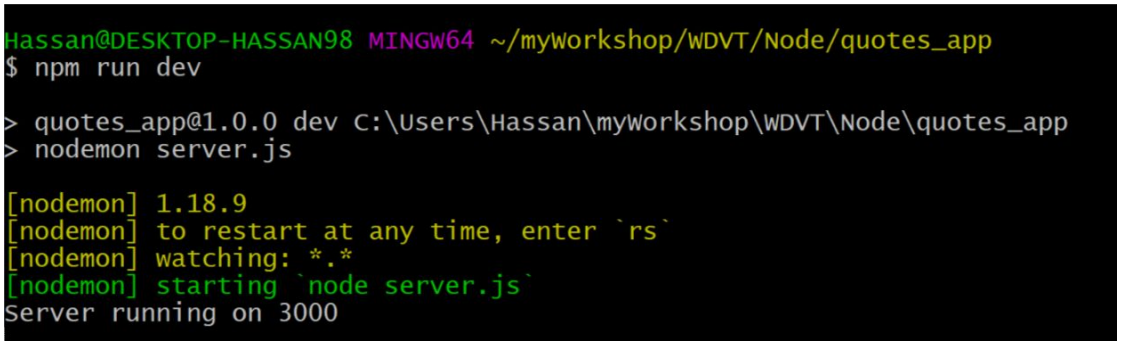
- **This is how Express handles a GET request (READ operation) in a nutshell.**

- At this point, you probably have realized that you need to restart your server whenever you make a change to server.js.

```
Hassan@DESKTOP-HASSAN98 MINGW64 ~/myWorkshop/WDVT/Node/quotes_app
$ npm install nodemon --save-dev
```

- Let’s create a script key so we can run “nodemon”:

```
  "name": "quotes_app",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "dev": "nodemon server.js"
  },
```



```
Hassan@DESKTOP-HASSAN98 MINGW64 ~/myWorkshop/WDVT/Node/quotes_app
$ npm run dev

> quotes_app@1.0.0 dev C:\Users\Hassan\myWorkshop\WDVT\Node\quotes_app
> nodemon server.js

[nodemon] 1.18.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
Server running on 3000
```

CRUD – CREATE

- The CREATE operation is only performed by the browser if a POST request is sent to the server.
- This POST request can be triggered either with JavaScript or through a <form> element.
- Let’s add a form in “index.html”:

```
7   <body>
8     May Node and Express be with you.
9
10    <form action="/quotes" method="POST">
11      <input type="text" placeholder="name" name="name">
12      <input type="text" placeholder="quote" name="quote">
13      <button type="submit">Submit</button>
14    </form>
15
16  </body>
```

- The **action** attribute tells the browser where to navigate to in our Express app.
- In this case, we're navigating to /quotes.
- The **method** attribute tells the browser what request to send. In this case, it's a POST request.
- Let's write our post request:

```
13
14    app.post('/quotes', (req, res) => {
15      console.log('Hellooooooooooooooooo!')
16    });
```

- Now go to your browser, then enter something into your form element.
- You should be able to see "Hellooooooooooooooooo!" in your command line:

```
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
Server running on 3000
Hellooooooooooooooooo!
```

- Great, we know that Express is handling the form for us right now.
- The next question is, how do we get the input values with Express?
- Turns out, Express doesn't handle reading data from the <form> element on it's own.
- We have to add another package called **body-parser** to gain this functionality.

```
Hassan@DESKTOP-HASSAN98 MINGW64 ~/myWorkshop/WDVT/Node/quotes_app
$ npm install body-parser --save
```

- Express allows us to add middleware like body-parser to our application with the use method:

```
3   const bodyParser = require('body-parser');
4
5   // Body parser Middleware
6   app.use(bodyParser.urlencoded({extended: false}));
7   app.use(bodyParser.json());
```

- Now, you should be able to see everything in the form field within the req.body object.
- Try doing a console.log and see what it is!

```
22
23    app.post('/quotes', (req, res) => {
24      console.log(req.body);
25    });
```

- You should be able to get an object similar to the following in your command line:

```
{ name: 'Yoda',
  quote: 'Train yourself to let go of everything you fear to lose' }
```

- Hmmm. Master Yoda has spoken! Let’s make sure we remember Yoda’s words.
- It’s important. We want to be able to retrieve it the next time we load our index page.

- Enter the database, **MongoDB**.