

Read HTML Form Data Using GET and POST Method in Node.js



Souvik Paul

Follow

Oct 7, 2020 · 6 min read

Hello world, welcome to the blog. In this blog we'll discuss how to create a server application using Node.js that will read data from HTML form and will perform basic operations like insert, read, update and delete on SQLite3 database using GET and POST requests. We'll also add security features by using modules like **Helmet** and **Express-rate-limit**.




Photo by [Sai Kiran Anagani](#) on [Unsplash](#)

You can also find the entire project in my GitHub repository —

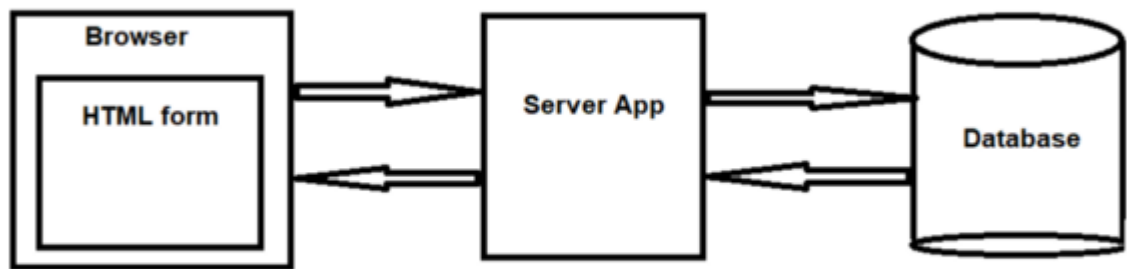
souvik-pl/crudApp_htmlForm

It is a server application developed using Node.js that will read data from HTML form and will perform basic operations...

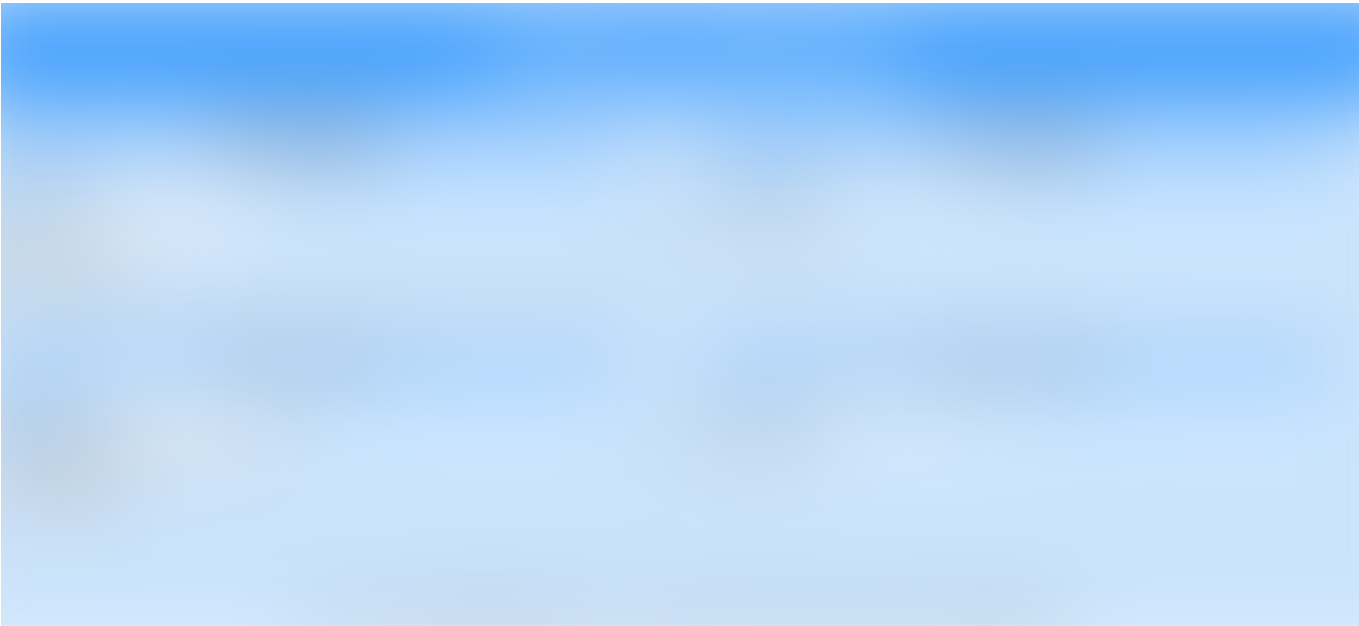
github.com



The basic outline of our project and the HTML form is shown below -



project outline



HTML form

Prerequisites -

1. Knowledge of — HTML, CSS, Node.js, and SQLite3
2. Node.js and SQLite3 should be installed on your local machine.

So, now we are ready to initiate our project. let's dive in.

Step 1: First of all, open the terminal and create one directory which will be dedicated for our project. Navigate into that directory and run `npm init` . Then you will be asked to enter various details about your application, which will be stored as a json file named 'Package.json'. Then you have to run following commands —

`npm install express --save` to install the 'express' module,

`npm install sqlite3 --save` to install 'sqlite3' module, and

`npm install body-parser --save` to install 'body-parser' module

`npm install helmet --save` to install 'helmet' module.

`npm install express-rate-limit --save` to install 'express-rate-limit' module.

Note- helmet is a Node.js module that helps in securing 'express' applications by setting various HTTP headers. It helps in mitigating cross-site scripting attacks, misissued SSL certificates etc.

Express-rate-limit module is a middleware for Express which is used to limit repeated requests to public APIs and/or endpoints such as password reset. By limiting the number of requests to the server, we can prevent the **Denial-of-Service (DoS) attack**. It is the type of attack in which the server is flooded with repeated requests making it unavailable to its intended users and ultimately shutting it down.

Step 2: Now let's start writing HTML and CSS code. We will keep our HTML and CSS files in a directory named 'public', which is itself present in our project directory. In the HTML file, we'll define the head section first.

```
<!DOCTYPE html>
<html lang = "en">
<head>
  <meta charset = "UTF-8">
  <link rel = "stylesheet" href="style.css">
  <title> My Form </title>
</head>
```

Then we'll write the body section. The following code will enable user to add a new employee to the database.

```
<body>
  <header>
    <h1>Employee Database</h1>
  </header>
  <form action="/add" method="POST">
    <fieldset>
      <h3>Add new employee</h3>
      <label>Employee ID</label>
      <input type = "text" id = 'empID' name="id" required>
      <br><br>
      <label>Name</label>
      <input type="text" id = "name" name="name" required>
      <br><br>
      <button type = "reset">Reset</button>
      <button type = "submit">Submit</button>
    </fieldset>
  </form>
```

The code below will enable user to view any employee based on their ID number.

```
<form action="/view" method="POST">
  <fieldset>
    <h3>View an employee</h3>
    <label>Employee ID</label>
    <input type="text" id="empID" name="id" required>
    <br><br>
    <button type = "reset">Reset</button>
    <button type = "submit">Submit</button>
    <br><br><br>
  </fieldset>
</form>
```

The following two code snippets are for updating and deleting employees from the database.

```
<form action="/update" method="POST">
  <fieldset>
    <h3>Update an employee</h3>
    <label>Employee ID</label>
    <input type="text" id="empID" name="id" required>
    <br><br>
    <label>New Name</label>
    <input type="text" id="name" name="name" required>
    <br><br>
    <button type="reset">Reset</button>
    <button type="submit">Submit</button>
  </fieldset>
</form>

<form action="/delete" method="POST">
  <fieldset>
    <h3>Delete an employee</h3>
    <label>Employee ID</label>
    <input type="text" id="empID" name="id" required>
    <br><br>
    <button type="reset">Reset</button>
    <button type="submit">Submit</button>
    <br><br><br>
  </fieldset>
</form>
```

At last we'll include a footer which will tell the user how to close the database connection.

```
<footer>
  <hr>
  <h4>Please enter 'http://localhost:3000/close' to close the
  database connection before closing this window</h4>
</footer>
</body>
</html>
```

Step 3: Now we'll write the CSS codes for styling our webpage.

Step 4: Since, we have designed our front end, now we'll build our back-end server application in Node.js. We'll create a file called 'app.js' in our main project directory and import all the necessary modules.

Notice that the execution mode is set to verbose to produce long stack traces.

In the above code, we have created an instance of express, named 'app' and we have also created a database named 'employee' in the 'database' directory which is present in our current directory.

windowMs is the timeframe for which requests are checked/remembered.

max is the maximum number of connections during **windowMs** milliseconds before sending a 429 response.

bodyParser.urlencoded() returns middleware that only parses `urlencoded` bodies and only looks at requests where the `Content-Type` header matches the `type` option.

express.static() is used to serve static files in 'express'.

Then, we'll create a table named 'emp' in the database having two columns- 'id' and 'name' using the following code -

The above code will make sure that 'emp' table won't be created again and again whenever we run the application.

Step 3: Now, it is the time to write the codes for listening to the GET and POST requests made by the browser.

When the user enters `http://localhost:3000` in the browser's address bar, following code will take care of this GET request and will send the HTML form as a response.

INSERT





To insert a new employee into the 'emp' table, the user is required to fill this part of the form.

And the following code will take care of this POST request -

In the above code, the `serialize()` method puts the execution mode into serialized mode. It means that only one statement can execute at a time. Other statements will wait in a queue until all the previous statements are executed.

READ



To view an employee from the 'emp' table, user is required to enter the Employee ID in this part of the form.

The following code will take care of this POST request.

In the above code, the `each()` method executes an SQL query with specified parameters and calls a callback for every row in the result set.

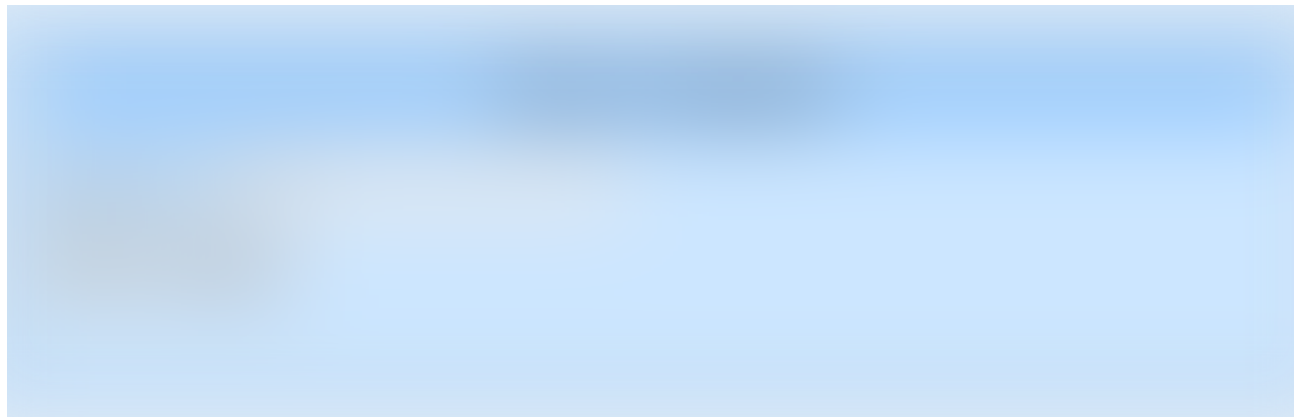
UPDATE



To update an existing employee, the user is required to enter the existing Employee ID and the New Name.

This POST request will be handled by the following code -

DELETE



To delete an existing employee from the 'emp' table, user is required to enter the Employee ID and this POST request will be handled by the following code-

Step 4: Now, we'll write code for closing the database connection.

The above code will function when the user enters `http://localhost:3000/close` in the address bar of the browser.

Step 5: Now, we need to make our server application listen to all the requests made by the browser, which will be achieved by the following command-

Step 6: Now that we have written all the codes for our server application, we'll save it and go back to terminal to run this using the command `node app.js` . The following message will be displayed in the console -

```
Server listening on port: 3000
```

So, now our server is up and running. We will open the browser and enter `http://localhost:3000` to start doing the CRUD operations.

Step 7: Then, open the **Network** tab by clicking on **Inspect Element**. Click on **localhost** and you will notice some additional set of headers in the response, which are set by the **helmet** module.

Summary

If you have completed above steps, then you have created a secured server application which can read data from HTML form and perform insert, view, update and delete operations.

References:

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

A complete documentation for using express module.

<https://www.npmjs.com/package/sqlite3>

A complete documentation for using sqlite3 module.

<https://www.npmjs.com/package/body-parser>

A complete documentation for using body-parser module.

<https://www.npmjs.com/package/express-rate-limit>

A complete documentation for using express-rate-limit module.

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

A complete documentation for using helmet module.

Sign up for Top 10 Stories

By The Startup

Get smarter at building your thing. Subscribe to receive The Startup's top 10 most read stories — delivered straight into your inbox, once a week. [Take a look.](#)



Get this newsletter

Emails will be sent to duggempudisrinu520@gmail.com.
[Not you?](#)

Nodejs

Sqlite3

HTML

Expressjs

Helmet



[About](#) [Help](#) [Legal](#)

Get the Medium app



Download on the
App Store



GET IT ON
Google Play