Modules, Routes and HTML Template Rendering

What is a Module

In Node.js, a module (or package) is a self-contained piece of code that exports values, objects, or functions for other parts of your application to use. Modules in Node.js help you organize your code into reusable, modular pieces and make it easier to manage dependencies between different parts of your application.

A module in Node.js is created by creating a separate JavaScript file that contains the code you want to export. You can then use the module.exports or exports object to define the values, objects, or functions that your module should export. Here is a simple example:

```
// greet.js
exports.greet = function(name) {
  return "Hello, " + name + "!";
};
```

To use the greet module in another file, you can use the require function to load the module:

```
// app.js
var greet = require('./greet');
console.log(greet.greet("barbarik"));
```

The require function returns the value of module.exports from the specified module, which you can then use in your code. In this example, greet.greet("barbarik") would output "Hello, barbarik!".

Modules in Node.js are a crucial part of writing scalable and maintainable applications, and are one of the key features that sets Node.js apart from other JavaScript environments.

ExpressJS is also a module which we use to create a powerful backend server, it is much convenient and powerful than the basic server which we create using http module

Recap - initializing Node Project

As we have learnt previously (ref: Creating First NodeJS Server PgNo. 3) that to initialize a node project we have to run npm init command, and then we have to provide the values we can also run npm init -y command to initialize the project with the default values, ex :

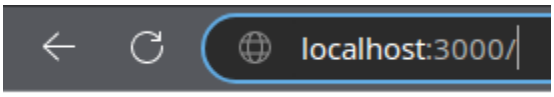
Express JS Server in action

We have previously discussed the code for creating a basic ExpressJS Server, save it in index.js file and spin the server using npm start command :

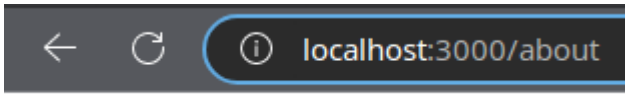
```
const express = require("express");
const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.status(200).send("Welcome to my website!");
/* By the status function we are setting the status code of the response that is to be sent to 200 */
});
```

```
app.get("/about", (req, res) => {
  res.send("This is the About page.");
});

app.get("/contact", (req, res) => {
  res.send("This is the Contact page.");
});

app.listen(port, () => {
  console.log("Server is listening on port ${port}");
});
```

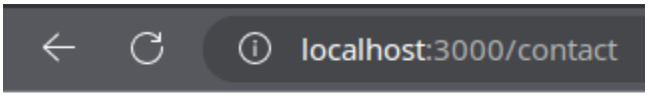
How the site looks at different routes :



Welcome to my website!



This is the About page.



This is the Contact page.

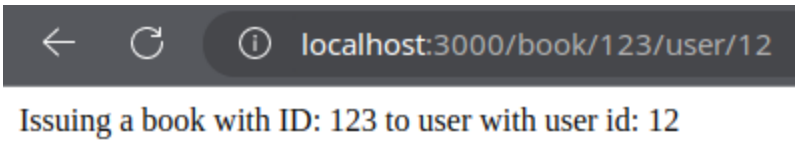
Route Parameters

In ExpressJS, route parameters are named values that you place in the URL which can be used to capture values from the URL. They are like placeholders that can be used to capture values from the URL, and can be used to process data in a specific way based on the values passed in the URL.
Here's an example to help illustrate this concept. Let's say we have a website for a library, and we want to issue a book(with book id: bookID) to a user with id : userID. We could create a route with a parameter to capture the book's ID:

Request URL: http://localhost:3000/book/123/user/12
req.params: { "userID": "12", "bookID": "123" }

```
app.get("/book/:bookID/user/:userID", function (req, res) {
  res.send("Issuing a book with ID: " + req.params.bookID+" to user with user id : "+req.params.userID);
});
```

In this example, the :id portion of the URL is the route parameter. When a client hits /book/123/user/12, for example, the value 123 would be captured as the bookID parameter and value 12 would be captured as the userID parameter, and the response sent to the user would be "Issuing a book with ID: 123 to user with user id : 12".



Route parameters are very useful for creating dynamic, data-driven applications. By capturing values from the URL, you can use them to retrieve information from a database, or to perform some other type of processing based on the data passed in the URL.

HTML Template Rendering

The sendFile() Method

The sendFile method in Express.js is a way to send a file from the server to a client (usually a web browser) in response to an HTTP request. This method is very useful for serving static files like images, videos, documents, etc.

Here's an example of how you could use the sendFile method to send an html file:
Create a folder by the name of views (ref: mvc architecture) which will contain different pages that we will be displaying in the browser. The folder name could be anything but we are following mvc convention.

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

app.get("/", (req, res) => {
  res.sendFile(path.join(__dirname + '/views/index.html'))
});

app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

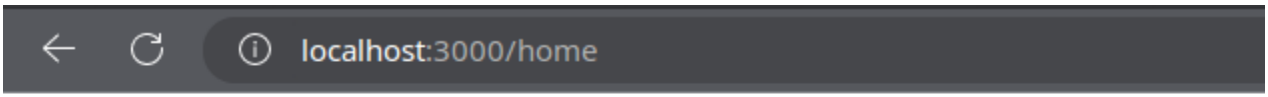
In this example, when a client makes a GET request to the /home endpoint, the server will respond by sending the index.html file located in the same directory as the script. The path.join method is used to construct the full file path, which ensures that the file can be found on any operating system.
Note that __dirname is a global variable in Node.js that holds the absolute path of the directory containing the script that is currently being executed. It provides a way to access the file system and make use of files and directories relative to the script. The value of __dirname is determined at runtime and remains constant during the lifetime of a script.

Let's say you have the file index.html (which has html for the home page) saved in your project folder. When you run this code and navigate to http://localhost:3000/home in your web browser, the home page should be displayed.

Add the following code to the index.js file

```
const path = require('path');

app.get("/", (req, res) => {
   res.sendFile(path.join(__dirname + '/views/index.html'))
})
```

After spinning the server and hit the /home route :



Welcome to website. This is the Home Page

By the way, the index. html file contains the following lines :

index.js | index.html | about.html | contact.html

index.html > html > body > span

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>Home</title>
9  </head>
10
11 <body>
12     <span>
13         Welcome to website. This is the <b>Home</b> Page
14     </span>
15 </body>
16
17 </html>
```

We can render different html pages based upon the route entered by the user. Lets first create the about page and the contact page like :

index.js | index.html | about.html | contact.html

about.html > html

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>About</title>
9  </head>
10
11 <body>
12     <span>
13         Welcome to website. This is the <b>About</b> Page
14     </span>
15 </body>
16
17 </html>
```

index.js | index.html | about.html | contact.html

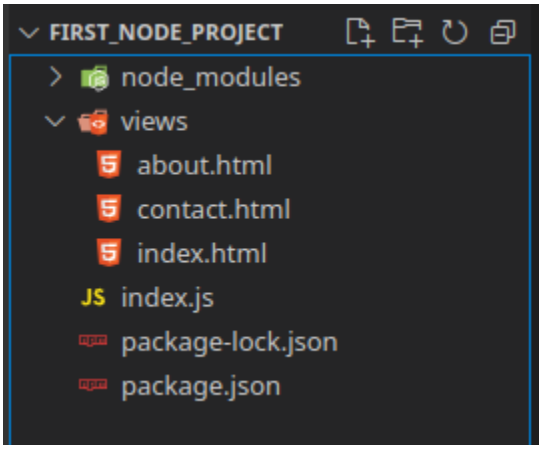
contact.html > html > body > span

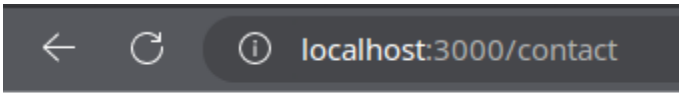
```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>Contact</title>
9  </head>
10
11 <body>
12     <span>
13         Welcome to website. This is the <b>Contact</b> Page
14     </span>
15 </body>
16
17 </html>
```

Then render the html pages by setting the routes as follows :

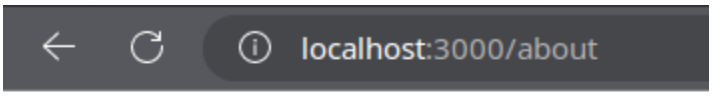
```
0 references
10 app.get("/", (req, res) => {
11     res.sendFile(path.join(__dirname + '/views/index.html'))
12 });
13
0 references
14 app.get("/about", (req, res) => {
15     res.sendFile(path.join(__dirname + '/views/about.html'))
16 })
0 references
17 app.get("/contact", (req, res) => {
18     res.sendFile(path.join(__dirname + '/views/contact.html'))
19 })
20
```

The Folder Structure looks something like this in the VSCode :





Welcome to website. This is the **Contact** Page



Welcome to website. This is the **About** Page

We can use css js scripts and functionality and usability to the website

You should try experimenting with these to learn more about it.