**Building RESTful APIs Using Express**

**1) Introduction**:

A REST API defines a set of functions which developers can perform requests and receive responses via HTTP protocol such as GET and POST.

**2) RESTful Services**:

RESTful Web Services are basically REST Architecture based Web Services. In REST Architecture everything is a resource. RESTful web services are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications.

REST stands for Representational State Transfer. REST is web standards-based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

**HTTP methods**:

Following four HTTP methods are commonly used in REST based architecture.

1. GET − Provides a read only access to a resource.
2. POST − Used to create a new resource.
3. DELETE − Used to remove a resource.
4. PUT − Used to update an existing resource or create a new resource.

**3) Introducing Express**:

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is an open source framework developed and maintained by the Node.js foundation.

Express was developed by TJ Holowaychuk and is maintained by the Node.js foundation and numerous open source contributors.

**Install Express**:

For install express run the following command in console

npm i express

**4) Building Your First Web Server (Using Express)**:

Create a file name index.js in our project root directory. In this file write the following code.

const express = require("express");

const app = express();

app.get("/", (req, res) => {

res.send("Hello World");

});

app.listen(3000, () => console.log("Listening on port 3000..."));

Here, first we have to lode the module "express"

const express = require("express");

Here, require("express") return a function. express() function return an object and by convention we store it in app variable.

const app = express();

Now we have to define a route.

app.get("/", (req, res) => {

res.send("Hello World");

});

* Here, get() method have two parameter "/" and a callback function.
* "/" represent the root of the website
* callback function has two parameters (req, res). req -> Request object have a branch of useful properties that gives us information about incoming request. For more info we have to see Express documentation <http://expressjs.com/> . res -> Response is use to response something to the client.

Finally, we need to listen on a given port.

app.listen(3000, () => console.log("Listening on port 3000..."));

Now go to the console and run "node index.js". Program will run and print " Listening on port 3000..." in console. Go to browser and in the address bar write <http://localhost:3000/> and press enter. In the browser we will see "Hello World"

Now another routes in our application.

const express = require("express");

const app = express();

app.get("/", (req, res) => {

res.send("Hello World");

});

app.get("/api/courses", (req, res) => {

res.send([1, 2, 3, 4, 5]);

});

app.listen(3000, () => console.log("Listening on port 3000..."));

**5) Nodemon**:

Every time we make change in our code, we have to go to the terminal stop the process and start it again. This is not a good practice. For resolve this problem we can use Nodemon.

nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

nodemon does not require any additional changes to your code or method of development. nodemon is a replacement wrapper for node, to use nodemon replace the word node on the command line when executing your script.

**Install**:

We can install nodemon globally to your system path or install nodemon as a development dependency.

**Globally**:

npm i -g nodemon

**Development dependency**:

npm install --save -dev nodemon

Now if we change anything in our code nodemon restart our server automatically.

**6) Environment Variables**:

Working with environment variables is a great way to configure different aspects of our Node.js application. Many cloud hosts (Heroku, Azure, AWS, now.sh, etc.) and Node.js modules use environment variables. Hosts, for example, will set a PORT variable that specifies on which port the server should listen to properly work. Modules might have different behaviors (like logging) depending on the value of NODE\_ENV variable.

When we deploy our application in a hosting environment, the port is dynamically assigning by the hosting environment. In our application we set the value 3000 which is not working in hosting environment. The way to fixed this is by using an environment variable.

An environment variable is basically a variable that is part of this environment in which a process run. It value is set outside the application.

We can define the environment variable by using process object. The process is a global object. Here we use dynamic port or 3000.

const port = process.env.PORT || 3000;

Now we can replace 3000 with our dynamic port.

**app.js**:

const express = require("express");

const app = express();

app.get("/", (req, res) => {

res.send("Hello World!!!");

});

app.get("/api/courses", (req, res) => {

res.send([1, 2, 3, 4, 5]);

});

const port = process.env.PORT || 3000;

app.listen(port, () => console.log(`Listening on port ${port}...`));

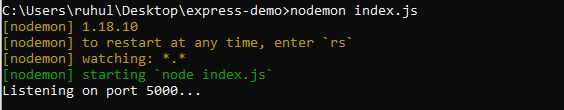
*//Listening on port 3000...*

Now when we run the program, (app.js) we will see the output

*//Listening on port 3000...*

Because we don’t set any environment variable call PORT. Now we set the environment variable PORT as 5000.

Go to console and run the command "set PORT=5000" for set 5000 as port and then run the command "nodemon index.js"



**7) Route Parameters**:

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

Each route can have one or more handler functions, which are executed when the route is matched.

**Example**:

The following code is an example of a very basic route.

var express = require("express");

var app = express();

*// respond with "hello world" when a GET request is made to the homepage*

app.get("/", function(req, res) {

res.send("hello world");

});

Now we implement a route for get a single course.

const express = require("express");

const app = express();

app.get("/", (req, res) => {

res.send("Hello World!!!");

});

app.get("/api/courses", (req, res) => {

res.send([1, 2, 3, 4, 5]);

});

*//implement a route for get a single course*

app.get("/api/courses/:id", (req, res) => {

res.send(req.params.id);

});

const port = process.env.PORT || 3000;

app.listen(port, () => console.log(`Listening on port ${port}...`));

*/\**

*in browser : http://localhost:3000/api/courses/1*

*output: 1*

*\*/*

Also, it is possible to have multiple parameters in a route. Suppose we are building a service for a blog. We have a route like this

**Example**:

const express = require("express");

const app = express();

app.get("/", (req, res) => {

res.send("Hello World!!!");

});

app.get("/api/courses", (req, res) => {

res.send([1, 2, 3, 4, 5]);

});

*//implement a route for get a single course*

app.get("/api/posts/:year/:month", (req, res) => {

res.send(req.params);

});

const port = process.env.PORT || 3000;

app.listen(port, () => console.log(`Listening on port ${port}...`));

*/\**

*in browser : http://localhost:3000/api/posts/2018/1*

*{"year":"2018","month":"1"}*

*\*/*

With express we can also get query string parameter. This are parameters that we can add in the URL after question mark (?). Now we have to read query parameter.

const express = require("express");

const app = express();

app.get("/", (req, res) => {

res.send("Hello World!!!");

});

app.get("/api/courses", (req, res) => {

res.send([1, 2, 3, 4, 5]);

});

*//implement a route for get a single course*

app.get("/api/posts/:year/:month", (req, res) => {

res.send(req.query);

});

const port = process.env.PORT || 3000;

app.listen(port, () => console.log(`Listening on port ${port}...`));

*/\**

*in browser : http://localhost:3000/api/posts/2018/1?sortBy=name*

*{"sortBy":"name"}*

*\*/*

**8) Handling HTTP GET Requests**:

Let’s implement a new endpoint to get a single course from the server.

const express = require("express");

const app = express();

*//define courses array*

const courses = [

{ id: 1, name: "courses\_1" },

{ id: 2, name: "courses\_2" },

{ id: 3, name: "courses\_3" }

];

app.get("/api/courses", (req, res) => {

res.send(courses);

});

*//implement a new endpoint to get a single course from the server*

app.get("/api/courses/:id", (req, res) => {

const course = courses.find(c => c.id === parseFloat(req.params.id));

if (!course) {

res.status(404).send("The course with the given ID was not found.");

}

res.send(course);

});

const port = process.env.PORT || 3000;

app.listen(port, () => console.log(`Listening on port ${port}...`));

*/\**

*in browser: http://localhost:3000/api/courses*

*[{"id":1,"name":"courses\_1"},{"id":2,"name":"courses\_2"},{"id":3,"name":"courses\_3"}]*

*in browser: http://localhost:3000/api/courses/1*

*{"id":1,"name":"courses\_1"}*

*in browser: http://localhost:3000/api/courses/10*

*The course with the given ID was not found.*

*\*/*

**9) Handling HTTP POST Requests**:

Now we are working with http post request. Using an http post request, we create a new course.

app.post("/api/courses/:id", (req, res) => {

});

In this route handler we need to read the course object that should be in the body of the request. Use these properties to create a new course object and then add the course object to our courses array. Now create a new course object.

*//post request*

app.post("/api/courses", (req, res) => {

const course = {

id: courses.length + 1,

name: req.body.name

};

});

Now in order for this line "name: req.body.name" for work, we need to enable parsing of JSON objects in the body of the request. Because by default, this feature is not enabling in express. So on the top after we get the app object, we need to call "express.json()"

const express = require("express");

const app = express();

app.use(express.json());

Here in the line "app.use(express.json())" we are adding a pic of middleware. When we call "express.json()" method, this method return a pic of middleware and then we call "app.use" to use that middleware in the request processing pipeline.

Now back to our new route handler and push the course object to our array. And finally by convention, when we post an object to the server, when the server creates a new object or a new resource, it should be turn that object in the body of the response.

courses.push(course);

res.send(course);

*//post request*

app.post("/api/courses", (req, res) => {

const course = {

id: courses.length + 1,

name: req.body.name

};

courses.push(course);

res.send(course);

});

**10) Calling Endpoints Using Postman**:

To call http services we use a chrome extension call "postman".

**11) Input Validation**:

**12) Handling HTTP PUT Requests**:

**13) Handling HTTP Delete Requests**:

**14) Project- Build the Genres API**:

04 Building RESTful APIs Using Express