

Node.js and Express Framework:

Express.js is a minimal and flexible web application framework that provides a robust set of features to develop Node.js based web and mobile applications. Express.js is one of the most popular web frameworks in the Node.js ecosystem. Express.js provides all the features of a modern web framework, such as templating, static file handling, connectivity with SQL and NoSQL databases.

Node.js has a built-in web server. The `createServer()` method in its `http` module launches an asynchronous `http` server. It is possible to develop a web application with core Node.js features. However, all the low level manipulations of HTTP request and responses have to be tediously handled. The web application frameworks take care of these common tasks, allowing the developer to concentrate on the business logic of the application. A web framework such as Express.js is a set of utilities that facilitates rapid, robust and scalable web applications.

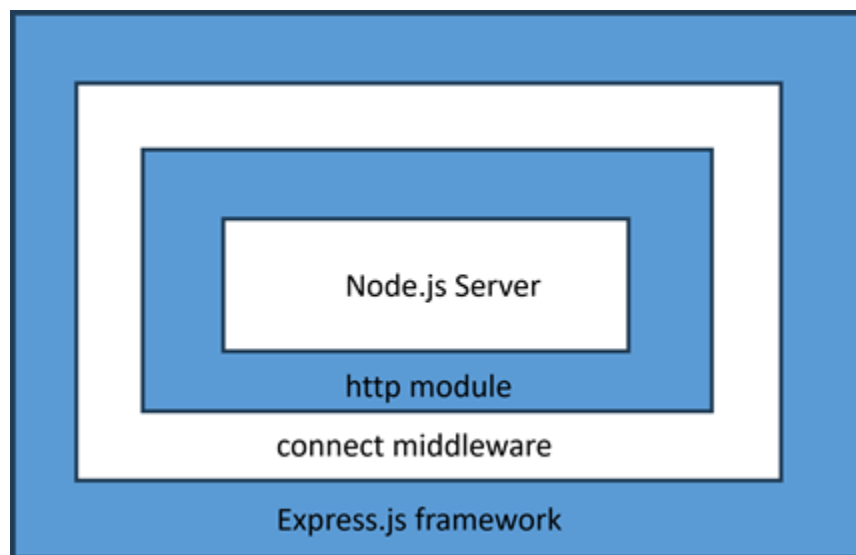
Following are some of the core features of Express framework –

- Allows to set up middlewares to respond to HTTP Requests.

- Defines a routing table which is used to perform different actions based on HTTP Method and URL.

- Allows to dynamically render HTML Pages based on passing arguments to templates.

The Express.js is built on top of the `connect` middleware, which in turn is based on `http`, one of the core modules of Node.js API.



Installing Express

The Express.js package is available on npm package repository. Let us install express package locally in an application folder named ExpressApp.

```
D:\expressApp> npm init
D:\expressApp> npm install express --save
```

The above command saves the installation locally in the node_modules directory and creates a directory express inside node_modules.

Hello world Example

Following is a very basic Express app which starts a server and listens on port 5000 for connection. This app responds with Hello World! for requests to the homepage. For every other path, it will respond with a 404 Not Found.

```
var express = require('express');
var app = express();

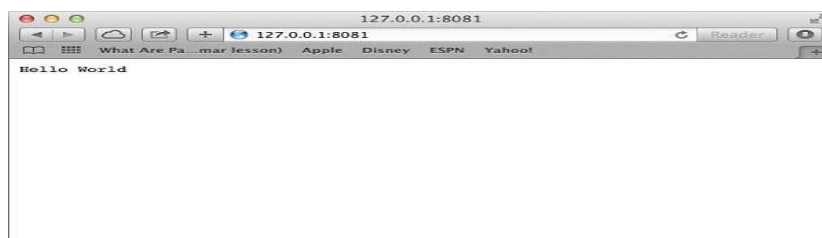
app.get('/', function (req, res) {
  res.send('Hello World');
});

var server = app.listen(5000, function () {
  console.log("Express App running at http://127.0.0.1:5000/");
});
```

Save the above code as index.js and run it from the command-line.

```
D:\expressApp> node index.js
Express App running at http://127.0.0.1:5000/
```

Visit <http://localhost:5000/> in a browser window. It displays the Hello World message.



Application object

An object of the top level express class denotes the application object. It is instantiated by the following statement –

```
var express = require('express');  
var app = express();
```

The Application object handles important tasks such as handling HTTP requests, rendering HTML views, and configuring middleware etc.

The app.listen() method creates the Node.js web server at the specified host and port. It encapsulates the createServer() method in http module of Node.js API.

```
app.listen(port, callback);
```

Basic Routing

The app object handles HTTP requests GET, POST, PUT and DELETE with app.get(), app.post(), app.put() and app.delete() method respectively. The HTTP request and HTTP response objects are provided as arguments to these methods by the NodeJS server. The first parameter to these methods is a string that represents the endpoint of the URL. These methods are asynchronous, and invoke a callback by passing the request and response objects.

GET method

In the above example, we have provided a method that handles the GET request when the client visits '/' endpoint.

```
app.get('/', function (req, res) {  
  res.send('Hello World');  
})
```

Request Object – The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.

Response Object – The response object represents the HTTP response that an Express app sends when it gets an HTTP request. The send() method of the response object formulates the server's response to the client.

You can print request and response objects which provide a lot of information related to HTTP request and response including cookies, sessions, URL, etc.

The response object also has a `sendFile()` method that sends the contents of a given file as the response.

```
res.sendFile(path)
```

Save the following HTML script as `index.html` in the root folder of the express app.

```
<html>
<body>
<h2 style="text-align: center;">Hello World</h2>
</body>
</html>
```

Change the `index.js` file to the code below –

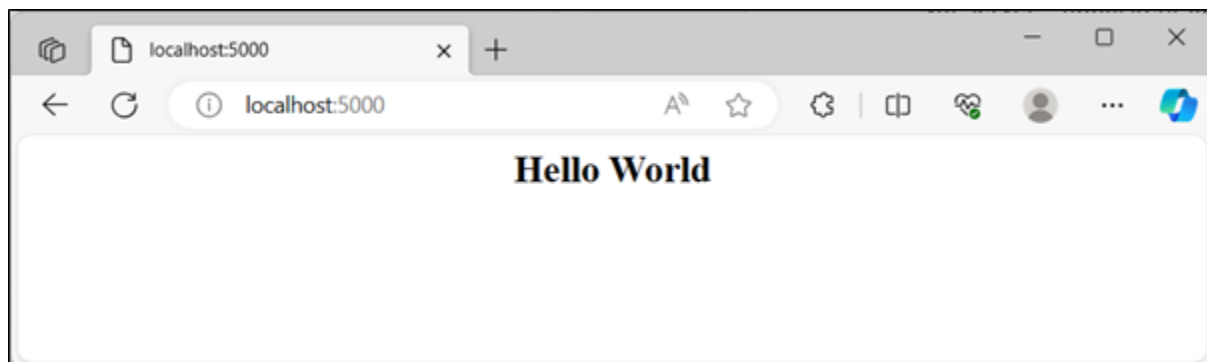
```
var express = require('express');
var app = express();
var path = require('path');

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

var server = app.listen(5000, function () {

  console.log("Express App running at http://127.0.0.1:5000/");
})
```

Run the above program and visit `http://localhost:5000/`, the browser shows Hello World message as below:



Let us use `sendFile()` method to display a HTML form in the `index.html` file.

```
<html>
  <body>

    <form action = "/process_get" method = "GET">
      First Name: <input type = "text" name = "first_name"> <br>
      Last Name: <input type = "text" name = "last_name"> <br>
      <input type = "submit" value = "Submit">
    </form>

  </body>
</html>
```

The above form submits the data to `/process_get` endpoint, with GET method. Hence we need to provide a `app.get()` method that gets called when the form is submitted.

```
app.get('/process_get', function (req, res) {
  // Prepare output in JSON format
  response = {
    first_name:req.query.first_name,
    last_name:req.query.last_name
  };
  console.log(response);
  res.end(JSON.stringify(response));
})
```

The form data is included in the request object. This method retrieves the data from `request.query` array, and sends it as a response to the client.

The complete code for `index.js` is as follows –

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

app.use(express.static('public'));


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

app.get('/process_get', function (req, res) {
```

```
// Prepare output in JSON format
response = {
  first_name:req.query.first_name,
  last_name:req.query.last_name
};
console.log(response);
res.end(JSON.stringify(response));
})

var server = app.listen(5000, function () {
  console.log("Express App running at http://127.0.0.1:5000/");
})
```

Visit <http://localhost:5000/>.



The image shows a simple web form on a light gray background. It contains two text input fields. The first field is preceded by the label 'First Name:'. The second field is preceded by the label 'Last Name:'. Below these two fields is a button labeled 'Submit'.

Now you can enter the First and Last Name and then click submit button to see the result and it should return the following result –

```
{"first_name":"John","last_name":"Paul"}
```

POST method

The HTML form is normally used to submit the data to the server, with its method parameter set to POST, especially when some binary data such as images is to be submitted. So, let us change the method parameter in index.html to POST, and action parameter to "process_POST".

```

<html>
  <body>
    <form action = "/process_POST" method = "POST">
      First Name: <input type = "text" name = "first_name"> <br>
      Last Name: <input type = "text" name = "last_name"> <br>
      <input type = "submit" value = "Submit">
    </form>
  </body>
</html>

```

To handle the POST data, we need to install the body-parser package from npm. Use the following command.

```
npm install body-parser -save
```

This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.

This package is included in the JavaScript code with the following require statement.

```
var bodyParser = require('body-parser');
```

The urlencoded() function creates application/x-www-form-urlencoded parser

```
// Create application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })
```

Add the following app.post() method in the express application code to handle POST data.

```

app.post('/process_post', urlencodedParser, function (req, res) {
  // Prepare output in JSON format
  response = {
    first_name:req.body.first_name,
    last_name:req.body.last_name
  };
  console.log(response);
  res.end(JSON.stringify(response));
})

```

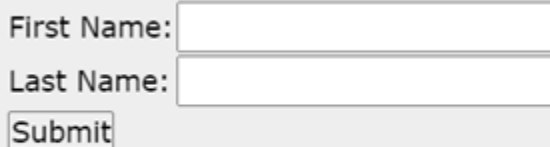
Here is the complete code for index.js file

```
var express = require('express');
var app = express();
var path = require('path');
var bodyParser = require('body-parser');
// Create application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })
app.use(express.static('public'));
app.get('/', function (req, res) {
  res.sendFile(path.join(__dirname, "index.html"));
})

app.get('/process_get', function (req, res) {
  // Prepare output in JSON format
  response = {
    first_name:req.query.first_name,
    last_name:req.query.last_name
  };
  console.log(response);
  res.end(JSON.stringify(response));
})

app.post("/process_post", )
var server = app.listen(5000, function () {
  console.log("Express App running at http://127.0.0.1:5000/");
})
```

Run index.js from command prompt and visit <http://localhost:5000/>.



First Name:

Last Name:

Now you can enter the First and Last Name and then click the submit button to see the following result –

```
{"first_name":"John","last_name":"Paul"}
```


Serving Static Files

Express provides a built-in middleware `express.static` to serve static files, such as images, CSS, JavaScript, etc.

You simply need to pass the name of the directory where you keep your static assets, to the `express.static` middleware to start serving the files directly. For example, if you keep your images, CSS, and JavaScript files in a directory named `public`, you can do this –

```
app.use(express.static('public'));
```

We will keep a few images in `public/images` sub-directory as follows –

```
node_modules
index.js
public/
public/images
public/images/logo.png
```

Let's modify "Hello Word" app to add the functionality to handle static files.

```
var express = require('express');
var app = express();
app.use(express.static('public'));

app.get('/', function (req, res) {
  res.send('Hello World');
});

var server = app.listen(5000, function () {
  console.log("Express App running at http://127.0.0.1:5000/");
});
```

Save the above code in a file named `index.js` and run it with the following command.

```
D:\expressApp> node index.js
```

Now open `http://127.0.0.1:5000/images/logo.png` in any browser and see observe following result.