ExpressJS is a web application framework that provides you with a simple API to build websites, web apps and back ends.

**Step 1** − Start your terminal/cmd, create a new folder named hello-world and cd (create directory) into it –

MKDIR TESTFOLDER

**Step 2** − Now to create the package.json file using npm, use the following code.

npm init

**Step 3** − Now we have our package.json file set up, we will further install Express. To install Express and add it to our package.json file, use the following command −

npm install --save express

You can now start working on Express.

We have set up the development, now it is time to start developing our first app using Express. Create a new file called **index.js** and type the following in it.

var express = require('express');

var app = express();

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

res.send("Hello world!");

});

app.listen(3000);

Save the file, go to your terminal and type the following.

Node index.js

This will start the server. To test this app, open your browser and go to **http://localhost:3000** and a message will be displayed as in the following screenshot.

How the App Works?

The first line imports Express in our file, we have access to it through the variable Express. We use it to create an application and assign it to var app.

### app.get(route, callback)

This function tells what to do when a **get** request at the given route is called. The callback function has 2 parameters, ***request(req)*** and ***response(res)***. The request **object(req)** represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, etc. Similarly, the response object represents the HTTP response that the Express app sends when it receives an HTTP request.

### res.send()

This function takes an object as input and it sends this to the requesting client. Here we are sending the string *"Hello World!"*.

**Routing in Express**

Web frameworks provide resources such as HTML pages, scripts, images, etc. at different routes.

The following function is used to define routes in an Express application −

## app.method(path, handler)

This METHOD can be applied to any one of the HTTP verbs – get, set, put, delete. An alternate method also exists, which executes independent of the request type.

Path is the route at which the request will run.

Handler is a callback function that executes when a matching request type is found on the relevant route. For example,

var express = require('express');

var app = express();

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

res.send("Hello World!");

});

app.listen(3000);

If we run our application and go to **localhost:3000/hello**

We can also have multiple different methods at the same route. For example,

var express = require('express');

var app = express();

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

res.send("Hello World!");

});

app.post('/hello', function(req, res){

res.send("You just called the post method at '/hello'!\n");

});

app.listen(3000);

A special method, ***all***, is provided by Express to handle all types of http methods at a particular route using the same function. To use this method, try the following.

app.all('/test', function(req, res){

res.send("HTTP method doesn't have any effect on this route!");

});

## Routers

Defining routes like above is very tedious to maintain. To separate the routes from our main **index.js** file, we will use **Express.Router**. Create a new file called **things.js** and type the following in it.

var express = require('express');

var router = express.Router();

router.get('/', function(req, res){

res.send('GET route on things.');

});

router.post('/', function(req, res){

res.send('POST route on things.');

});

//export this router to use in our index.js

module.exports = router;

Now to use this router in our **index.js**, type in the following before the **app.listen** function call.

var express = require('Express');

var app = express();

var things = require('./things.js');

//both index.js and things.js should be in same directory

app.use('/things', things);

app.listen(3000);

The ***app.use*** function call on route **'/things'** attaches the **things** router with this route. Now whatever requests our app gets at the '/things', will be handled by our things.js router. The **'/'** route in things.js is actually a subroute of '/things'. Visit localhost:3000/things/ and you will see the following output.

We can now define routes, but those are static or fixed. To use the dynamic routes, we SHOULD provide different types of routes. Using dynamic routes allows us to pass parameters and process based on them.

Here is an example of a dynamic route −

var express = require('express');

var app = express();

app.get('/:id', function(req, res){

res.send('The id you specified is ' + req.params.id);

});

app.listen(3000);

To test this go to [**http://localhost:3000/123**](http://localhost:3000/123)

You can replace '123' in the URL with anything else and the change will reflect in the response. A more complex example of the above is –

var express = require('express');

var app = express();

app.get('/things/:name/:id', function(req, res) {

res.send('id: ' + req.params.id + ' and name: ' + req.params.name);

});

app.listen(3000);

[**http://localhost:3000/things/tutorialspoint/12345**](http://localhost:3000/things/tutorialspoint/12345).

Middleware

Middleware functions are functions that have access to the **request object (req)**, the **response object (res)**, and the next middleware function in the application’s request-response cycle.

## Third Party Middleware

### 1. body-parser

## This is used to parse the body of requests which have payloads attached to them. To mount body parser, we need to install it using npm install --save body-parser

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

//To parse URL encoded data

app.use(bodyParser.urlencoded({ extended: false }))

//To parse json data

app.use(bodyParser.json())

### cookie-parser

It parses *Cookie* header and populate req.cookies with an object keyed by cookie names. To mount cookie parser, we need to install it using npm install --save cookie-parser

var cookieParser = require('cookie-parser');

app.use(cookieParser())