## **Npm init:**

Interactively create a package.json file. Package JSON is like cover of your book. It contains all the information about your project.

This will ask you a bunch of questions, and then write a package.json for you.

* For a Package Name, no uppercase and special characters.

**Npm start**: This runs an arbitrary command specified in the package's "**start**" property of its "**scripts**" object. If no "start" property is specified on the "scripts" object, it will run node **server.js**.

**Npm install**: This command installs a package, and any packages that it depends on.

**Express Module**

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 is used for validate users, do some pre and post works after req and res.

**Create Server Using Express:**

var express = require('express');

var app = express();

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

res.send('hello World');

});

app.listen(3000, function(){

console.log("Check port 3000");

});

**Routing:**

**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.

Route definition takes the following structure:

app.METHOD(PATH, HANDLER)

Where:

* app is an instance of express.
* METHOD is an HTTP request method, in lowercase.
* PATH is a path on the server.
* HANDLER is the function executed when the route is matched.

The following examples illustrate defining simple routes.

Respond with Hello World! on the homepage:

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

Respond to POST request on the root route (/), the application’s home page:

app.post('/', function (req, res) {  
 res.send('Got a POST request')  
})

Respond to a PUT request to the /user route:

app.put('/user', function (req, res) {  
 res.send('Got a PUT request at /user')  
})

Respond to a DELETE request to the /user route:

app.delete('/user', function (req, res) {  
 res.send('Got a DELETE request at /user')  
})

### **Route parameters**

Route parameters are named URL segments that are used to capture the values specified at their position in the URL. The captured values are populated in the req.params object, with the name of the route parameter specified in the path as their respective keys.

*Route path: /users/:userId/books/:bookId  
Request URL: http://localhost:3000/users/34/books/8989  
req.params: { "userId": "34", "bookId": "8989" }*

To define routes with route parameters, simply specify the route parameters in the path of the route as shown below.

app.get('/users/:userId/books/:bookId', function (req, res) {  
 res.send(req.params)  
})

## **Middleware in Express JS**

Middleware is the application that connects front-end and back-end.

**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.

Middleware functions can perform the following tasks:

* Execute any code.
* Make changes to the request and the response objects.
* End the request-response cycle.
* Call the next middleware in the stack.

var express = require('express');  
var app = express();  
  
//Simple request time logger  
app.use(function(req, res, next){  
 console.log(new Date(), req.method, req.url);  
   
 next();  
});  
  
app.listen(3000);

The above middleware is called for every request on the server. So after every request, we will get the following message in the console.

**Restrict Middleware for a Specific (and its subroutes) Routes:**

var express = require('express');  
var app = express();  
  
//Middleware function to log request protocol  
app.use('/about', function(req, res, next){  
 console.log("A request for things received at " + Date.now());  
 next();  
});  
  
// Route handler that sends the response  
app.get('/about', function(req, res){  
 res.send('Things');  
});  
  
app.listen(3000);

**The order of middleware**

The order of middleware loading is important: middleware functions that are loaded first are also executed first.

If middleware is loaded after the route to the root path, the request never reaches it and the app doesn’t work, because the route handler of the root path terminates the request-response cycle.

The middleware function simply prints a message, then passes on the request to the next middleware function in the stack by calling the next() function.

## **Request**

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

## 

## **Request Properties and Methods**

### **req.app**

This property holds a reference to the instance of the Express application that is using the middleware.

### **req.body**

Contains key-value pairs of data submitted in the request body. By default, it is undefined, and is populated when you use body-parsing middleware such as [**body-parser**](https://www.npmjs.org/package/body-parser) and [**multer**](https://www.npmjs.org/package/multer).

### 

var app = require('express')();  
var bodyParser = require('body-parser');  
var multer = require('multer'); // v1.0.5  
var upload = multer(); // for parsing multipart/form-data  
  
app.use(bodyParser.json()); // for parsing application/json  
app.use(bodyParser.urlencoded({ extended: true })); // for parsing application/x-www-form-urlencoded  
  
app.post('/profile', upload.array(), function (req, res, next) {  
 console.log(req.body);  
 res.json(req.body);  
});

### **req.hostname**

Contains the hostname derived from the Host HTTP header.

### **req.ip**

Contains the remote IP address of the request.

### **req.method**

Contains a string corresponding to the HTTP method of the request: GET, POST, PUT, and so on.

### **req.originalUrl**

app.use('/admin', function(req, res, next) { // GET 'http://www.example.com/admin/new'  
 console.log(req.originalUrl); // '/admin/new'  
 console.log(req.baseUrl); // '/admin'  
 console.log(req.path); // '/new'  
 next();  
});

### **Req.params**

This property is an object containing properties mapped to the named route “parameters”. For example, if you have the route /user/:name, then the “name” property is available as req.params.name. This object defaults to {}.

### **req.protocol**

Contains the request protocol string: either http or (for TLS requests) https.

**req.query**

This property is an object containing a property for each query string parameter in the route. If there is no query string, it is the empty object, {}.

// GET /search?q=tobi+ferret  
req.query.q  
// => "tobi ferret"  
  
// GET /shoes?order=desc&shoe[color]=blue&shoe[type]=converse  
req.query.order  
// => "desc"  
  
req.query.shoe.color  
// => "blue"  
  
**req.route**

Contains the detailed object of the routing

### **req.secure**

A Boolean property that is true if a TLS connection is established.

https => true

**To Handle Mismatched Routing:**

If none of your routes match the request, you'll get a "Cannot GET <your-request-route>" message as response. This message be replaced by a 404 not found page using this simple route −

**//Other routes here  
app.get('\*', function(req, res){  
 res.send('Sorry, this is an invalid URL.');  
});  
app.listen(3000);**

## 

## 

## 

## 

## 

## 

## 

## **Response methods**

The methods on the response object (res) in the following table can send a response to the client, and terminate the request-response cycle. If none of these methods are called from a route handler, the client request will be left hanging.

|  |  |
| --- | --- |
| **Method** | **Description** |
| res.download() | Prompt a file to be downloaded. |
| res.end() | End the response process. |
| res.json() | Send a JSON response. |
| res.redirect() | Redirect a request. |
| res.render() | Render a view template. |
| res.send() | Send a response of various types.  res.send(new Buffer('whoop')); res.send({ some: 'json' }); res.send('<p>some html</p>'); |
| res.sendFile() | Send a file as an octet stream. |
| res.sendStatus() | Set the response status code and send its string representation as the response body. |