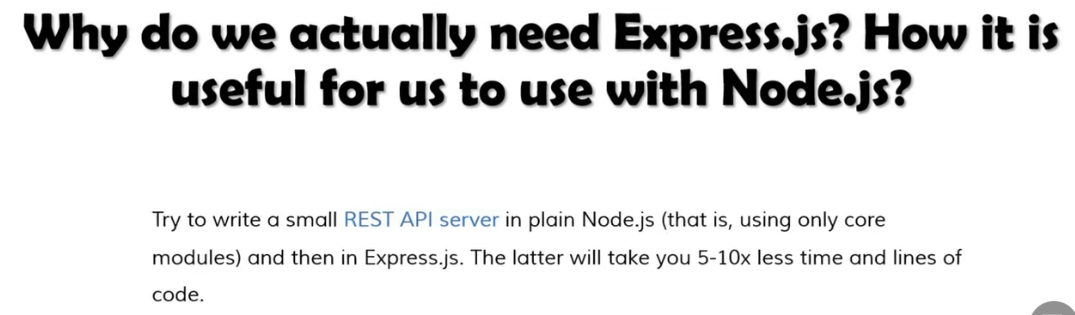
**EXPRESS.js**

[**http://expressjs.com/**](http://expressjs.com/)

It is a most popular framework of nodejs. It is a web application framework that provides you with a simple API to build websites, web apps and back ends.



**Installing Express.js**

[**http://expressjs.com/en/starter/installing.html**](http://expressjs.com/en/starter/installing.html)

[**https://www.npmjs.com/package/express**](https://www.npmjs.com/package/express)

1. Create a folder to hold your application and add a file into it like **app.js**
2. Under this folder install npm as: D:\Learning\expresslearning> **npm init (note that npm file name donot contain capital letters).** Give any package name like **expapp** and rest as it is. After this, **package.json** file is created
3. Now, install **Express** in the directory and save it in the dependencies list as: D:\Learning\expresslearning> **npm i express**

**express():** Creates an Express application. The express() function is a top-level function exported by the express module.

const express = require('express') **//when you use express module, it will you a function**

const app = express() **//with the help of app, we create express application. This app variable contain various properties and methods**

# **EXPRESS JS ROUTING**

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

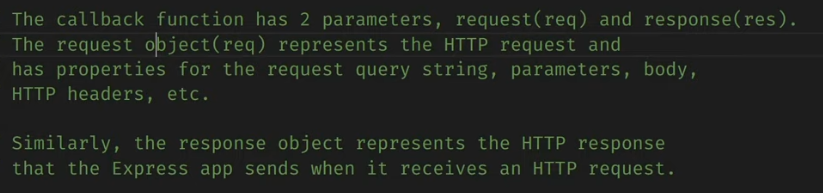
**get:** to read the data

**post:** to create data

**put:** to update the data

**delete:** to delete the data

**app.get(route, callback\_function) //callback function have two parameters i.e. req and res**



**Example**: First Program to print Hello World

const express = require('express'); **//express module**

const app = express();

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

    res.send('Hello Guys');

});

**app.get**('/about', (req, res) => {

    res.send('Aayush Guys');

});

**app.get**('/contact, (req, res) => {

    res.send(‘Contact to Aayush’);

});

app.listen(8000, () => {

    console.log("Listening");

});

# **HOW TO SEND HTML & JSON DATA AS A RESPONSE USING EXPRESS JS**

You can send the JSON response by using **res.json()** method. It accepts an object and converts it into JSON before sending it as a response.

**HOW TO SEND HTML DATA**

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

    res.send('<h1Hello Guys</h1>'); **//send single HTML element**

});

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

    res.write('<h1Hello Guys</h1>'); **//send multiple HTML element with res.write**

res.write('<h1Hello Guys, I am Back</h1>');

res.send(); **//end of sending data**

});

**HOW TO SEND JSON DATA (First Method): res.send()**

**app.get**('/temperature', (req, res) => {

    res.send([{ **//send json data by passing js object**

id: 1, **//pass array of objects**

name: “aayush”

},

{

Id: 2,

Name: “Rihan”

}]);

});

**OUTPUT:**



**HOW TO SEND JSON DATA (Second Method): res.json()**

**app.get**('/temperature', (req, res) => {

**res.json**([{ **//send json data by passing js object**

id: 1, **//pass array of objects**

name: “aayush”

},

{

Id: 2,

Name: “Rihan”

}]);

});

**NOTE:** The methods are identical when an object or array is passed, but res.json() will also convert non-objects, such as null and undefined, which are not valid json

# **SERVE HTML CSS & JS FILES IN EXPRESS JS | MIDDLEWARE IN EXPRESS JS**

<http://expressjs.com/en/starter/static-files.html>

To serve static files such as images, CSS files, and JavaScript files, use the **express.static** built-in middleware function in Express. The function signature is:

**express.static(root, [options])**

The root argument specifies the root directory from which to serve static assets.

For example, use the following code to serve images, CSS files, and JavaScript files in a directory named **public**:

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

Now, you can load the files that are in the public directory:

http://localhost:3000/images/kitten.jpg

http://localhost:3000/css/style.css

http://localhost:3000/js/app.js

http://localhost:3000/images/bg.png

http://localhost:3000/hello.html

Express looks up the files relative to the static directory, so the name of the static directory is not part of the URL. To use multiple static assets directories, call the **express.static** middleware function multiple times:

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

**app.use(express.static('files'))**

Express looks up the files in the order in which you set the static directories with the express.static middleware function.

**Example**:

const path = require('path'); **//use path module to find the absolute path of the directory**

const express = require('express');

const app = express();

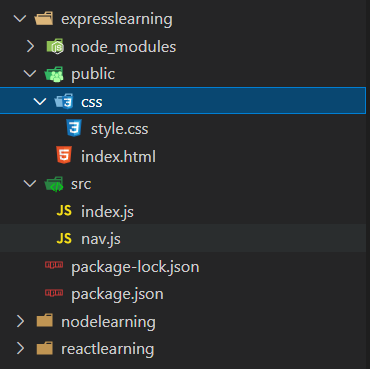
const staticPath = **path.join(\_\_dirname, '../public');**

app.use(express.static(staticPath));

app.listen(8000, () => {

    console.log("Listening");

});



# **TEMPLATE ENGINES (PUG, HBS, EJS) IN NODE JS | ADD DYNAMIC CONTENT IN EXPRESS JS**

A template engine enables you to use static template files in your application. At runtime, the template engine replaces variables in a template file with actual values and transforms the template into an HTML file sent to the client. This approach makes it easier to design an HTML page. Some popular template engines that work with Express are Pug, Handlebars (Mustache), and EJS. The Express application generator uses Jade as its default, but it also supports several others.

<https://www.npmjs.com/package/@andreyvolokitin/handlebars.js>

<https://www.npmjs.com/package/hbs>: hbs is [Express.js](https://expressjs.com/) view engine for [handlebars.js](https://handlebarsjs.com/)

**Installing of HBS**: D:\Learning\expresslearning> **npm i hbs**

To render template files, set the following [application setting properties](http://expressjs.com/en/4x/api.html#app.set), set in app.js in the default app created by the generator:

* views, the directory where the template files are located. Eg: **app.set('views', './views')**. This defaults to the views directory in the application root directory. This directory should be under **src** folder of your project.
* view engine, the template engine to use. For example, to use the hbs template engine: **app.set('view engine', ‘hbs’)**.

**Example**:

const path = require('path');

const express = require('express');

const app = express();

app.set("view engine", "hbs"); **//set view engine**

app.get("/", (req, res) => { **//render index.hbs file under views directory**

    res.render("index");

  });

app.listen(8000, () => {

    console.log("Listening");

});

**Example**: Dynamic Values passing to static page

**index.hbs**

<body>

    <h1>{{name}}</h1>

</body>

**index.js**

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

    res.render("index", {

        name: 'agarwal'

    });

  });

# **CUSTOMIZING THE VIEWS DIRECTORY IN EXPRESS JS**

We have studied that views named folder is to be created for template engine. But we can change the name of that directory also.

Rename your **views** directory to **templates (under src folder):**

const path = require('path');

const express = require('express');

const app = express();

**const templatePath = path.join(\_\_dirname, "./templates");**

app.set("view engine", "hbs");

**app.set('views', templatePath);**

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

    res.render("index", { **//index.hbs under template directory**

        name: 'agarwal'

    });

  });

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

    res.render("about") **//index.hbs under template directory**

  });

app.listen(8000, () => {

    console.log("Listening");

});



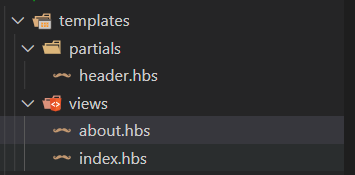
# **USING PARTIALS IN EXPRESS JS**

The Express view system has built-in support for partials and collections, which are “mini” views representing a document fragment. For example rather than iterating in a view to display comments, we could use partial collection:

**partial('comment', { collection: comments });**

**NOTE: If you want to use some component in every page like header and footer, then we use partials in expressjs**

Create a folder under templates which consist of two directory, **views and partials**



**Example: index.js (under src directory)**

const path = require('path');

const express = require('express');

const app = express();

const hbs = require('hbs'); **//need to require hbs module for using partials**

const templatePath = path.join(\_\_dirname, "./templates/views"); **//views folder path**

const partialPath = path.join(\_\_dirname, "./templates/partials"); **//partials folder path**

app.set("view engine", "hbs");

app.set('views', templatePath);

**hbs.registerPartials(partialPath); //register partials for your project**

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

    res.render("index", {

        name: 'agarwal'

    });

  });

  app.get("/about", (req, res) => {

    res.render("about")

  });

app.listen(8000, () => {

    console.log("Listening");

});

**index.hbs (under views directory)**

<body>

    {{>header}}

    <h1>Welcome</h1>

</body>

**header.hbs (under partials directory)**

<nav class="navbar navbar-expand-lg navbar-light bg-light">

  <a class="navbar-brand" href="#">Navbar</a>

  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

    <span class="navbar-toggler-icon"></span>

  </button>

  <div class="collapse navbar-collapse" id="navbarSupportedContent">

    <ul class="navbar-nav mr-auto">

      <li class="nav-item active">

        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>

      </li>

      <li class="nav-item">

        <a class="nav-link" href="#">Link</a>

      </li>

      <li class="nav-item dropdown">

        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

          Dropdown

        </a>

        <div class="dropdown-menu" aria-labelledby="navbarDropdown">

          <a class="dropdown-item" href="#">Action</a>

          <a class="dropdown-item" href="#">Another action</a>

          <div class="dropdown-divider"></div>

          <a class="dropdown-item" href="#">Something else here</a>

        </div>

      </li>

      <li class="nav-item">

        <a class="nav-link disabled" href="#">Disabled</a>

      </li>

    </ul>

    <form class="form-inline my-2 my-lg-0">

      <input class="form-control mr-sm-2" type="search" placeholder="Search" aria-label="Search">

      <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>

    </form>

  </div>

</nav>

# **ADD 404 ERROR PAGE IN DYNAMIC WEBSITE USING EXPRESS JS**

**Example: index.js (under src directory)**

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

    res.render("404", {

      errorcomment: 'OOps'

    })

  });

**404.hbs (under views directory)**

<body>

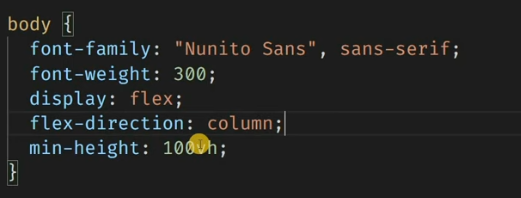
    {{>header}}

    {{errorcomment}}

</body>

# **ADDING RESPONSIVE STYLES TO COMPLETE DYNAMIC WEBSITE USING CSS IN EXPRESS**

**Example: How to fix footer section just at bottom of webpage**





# **NODEJS QUERY STRINGS | URL PARAMETERS IN EXPRESS JS**

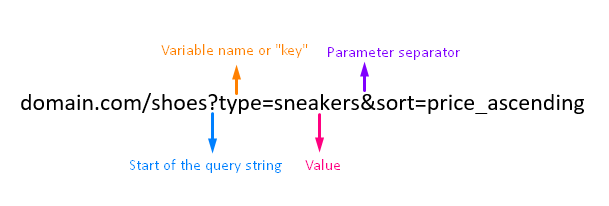
URL Parameters are pieces of information located in the query string of a URL. The query string is the portion of the URL that follows a question mark.

**Example**: [www.thapa.com/course?name=mernstack](http://www.thapa.com/course?name=mernstack)

URL Parameters are pieces of information located in the query string of a URL. The query string is the portion of the URL that follows a question mark. For example:  domain.com/shoes**?type=sneakers**

**Format of URL Parameters**

Upon first look, you might just see a random string of letters, numbers, and symbols, but once you understand the anatomy of a URL parameter, you'll be able to make sense of these long strings of code.



* **Question Mark:** This starts the URL parameter (domain.com/shoes**?**type=sneakers)
* **Ampersand:** This separates parameters when you have multiple in one URL (domain.com/shoes?type=sneakers**&**sort=price\_ascending)
* **Variable Name:** (or "key") is like the title or label of the parameter (domain.com/shoes?**type**=sneakers)
* **Value:** This is the specific value that the key identifies (domain.com/shoes?type=**sneakers**). For grammar nerds, it's like the predicate nominative in a sentence. In the sentence "Bagel is my dog", *Bagel* is like the key/variable name while *dog* is the value.

## Active & Passive URL Parameters

Query strings are used to either change the page content or to track information about a click through the URL. These two main uses indicate two main types of URL parameters: active and passive.

Active URL parameters modify the content on the page. Some examples include:

* Narrowing content (ex: **?type=yellow** to **display only “yellow”** results on a page)
* Reordering content (ex: **?sort=price\_ascending** to **display items in a particular order**)
* Segmenting content (ex: **?page=1, ?page=2, ?page=3** to **break up one large piece of content** into three smaller parts)

Passive URL parameters do not change the content of the page. They are used for tracking and attribution. Some examples include:

* Session IDs (A website’s server assigns a unique number to a specific user for the duration of their website visit.)
* UTM codes (Stands for “urchin tracking module” and helps identify the source, medium, and campaign a website visit came from.)
* Affiliate IDs (Often used by bloggers/influences. The owner of the website with this type of link generates revenue per click.)

**Example: How to display name which is written in URL:**

**http://localhost:8000/about?name=aayush&sports=cricket**

  app.get("/about", (req, res) => {

    res.render("about", {

**name: req.query.name, //request query string**

**sports: req.query.sports**

    })

  });