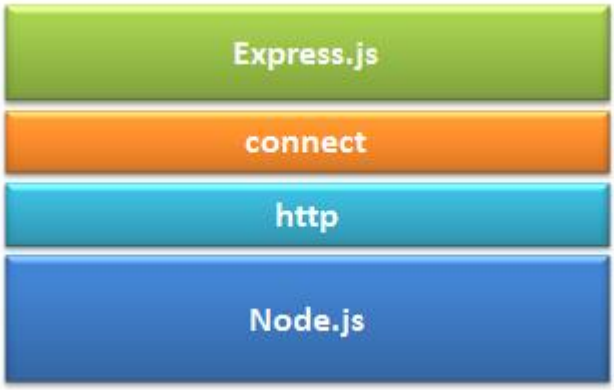
Express Document

Express js:

Express is a fast, assertive, essential and moderate web framework of Node.js. You can assume express as a layer built on the top of the Node.js that helps manage a server and routes. It provides a robust set of features to develop web and mobile applications.



Let's see some of the core features of Express framework:

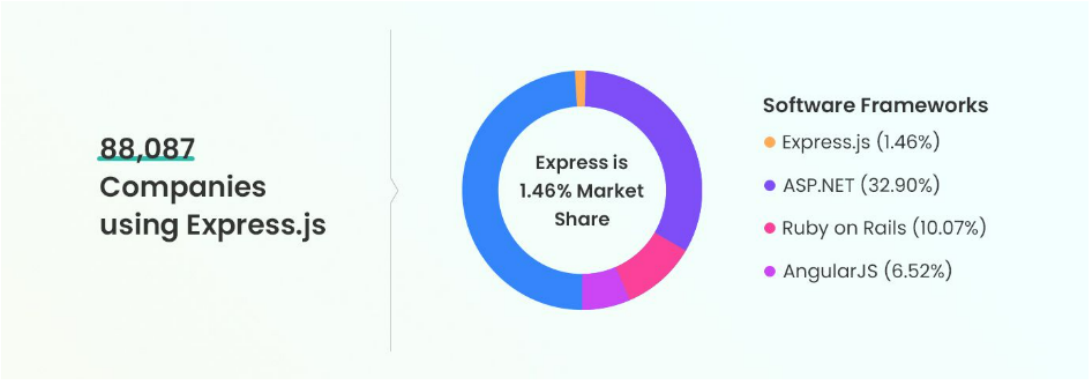
- It can be used to design single-page, multi-page and hybrid web applications.
- It allows to setup middleware's to respond to HTTP Requests.
- It defines a routing table which is used to perform different actions based on HTTP method and URL.
- Htttps://www.sample.co.in/home
- It allows to dynamically render HTML Pages based on passing arguments to templates.

Advantages of Express.js

- Makes Node.js web application development fast and easy.
- Easy to configure and customize.
- Allows you to define routes of your application based on HTTP methods and URLs.
- Includes various middleware modules, which you can use to perform additional tasks on request and response.

- Easy to integrate with different template engines like Jade, Vash, EJS etc.
- Allows you to define an error handling middleware.
- Easy to serve static files and resources of your application.
- Allows you to create REST API server.
- Easy to connect with databases such as MongoDB, Redis, MySQL

Industry-leading Brands That Are Using ExpressJS



According to Enlyft, 88,087 companies are using ExpressJS in their tech stacks, including:

- Twitter
- Accenture
- Kevin
- Paytm
- PayPal
- Uber
- IBM

Install Express:

npm install express --save

or

npm install -g express

The above command install express in node_module directory and create a directory named express inside the node_module.

Basic Express App:

Adding packages to program.

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

```
var app = express();

// define routes here..

var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```

In the above example, we imported Express.js module using require () function. The express module returns a function. This function returns an object, which can be used to configure Express application (app in the above example).

The app object includes methods for routing HTTP requests, configuring middleware, rendering HTML views and registering a template engine.

The app.listen () funct~ion creates the Node.js web server at the specified host and port. It is identical to Node's http.Server.listen () method.

Run the above example using node app.js command and point your browser to http://localhost:5000. It will display Cannot GET / because we have not configured any routes yet.

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Welcome to MEAN Satck DEMO');
});
 app.listen(8000);
```

You should install some other important modules along with express. Following is the list:

- **body-parser:** This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.
- **cookie-parser:** It is used to parse Cookie header and populate req.cookies with an object keyed by the cookie names.
- **multer:** This is a node.js middleware for handling multipart/form-data.

npm install body-parser --save

npm install cookie-parser --save

npm install multer --save

or in one line

npm install express body-parser cookie-parser multer –save

Express.js Request Object:

Express.js Request and Response objects are the parameters of the callback function which is used in Express applications.

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

```
app.get('/', function (req, res) {
  res.send('Welcome to MEAN Satck DEMO');
});
```

Express.js Request Object Properties:

Index	Properties	Description
1.	req.app	This is used to hold a reference to the instance of the express application that is using the middleware.
2.	req.baseurl	It specifies the URL path on which a router instance was mounted.
3.	req.body	It 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.
4.	req.cookies	When we use cookie-parser middleware, this property is an object that contains cookies sent by the request.
5.	req.fresh	It specifies that the request is "fresh." it is the opposite

		of req.stale.
6.	req.hostname	It contains the hostname from the "host" http header.
7.	req.ip	It specifies the remote IP address of the request.
8.	req.ips	When the trust proxy setting is true, this property contains an array of IP addresses specified in the ?x-forwarded-for? request header.
9.	req.originalurl	This property is much like req.url; however, it retains the original request URL, allowing you to rewrite req.url freely for internal routing purposes.
10.	req.params	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 {}.
11.	req.path	It contains the path part of the request URL.
12.	req.protocol	The request protocol string, "http" or "https" when requested with TLS.
13.	req.query	An object containing a property for each query string parameter in the route.
14.	req.route	The currently-matched route, a string.
15.	req.secure	A Boolean that is true if a TLS connection is established.
16.	req.signedcookies	When using cookie-parser middleware, this property contains signed cookies sent by the request, unsigned and ready for use.
17.	req.stale	It indicates whether the request is "stale," and is the opposite of req.fresh.
18.	req.subdomains	It represents an array of subdomains in the domain name of the request.

19.	req.xhr	A Boolean value that is true if the request's "x-requested-with" header field is "xmlhttprequest", indicating that the request was issued by a client library such as jQuery

Request Object Methods:

req.accepts (types)

This method is used to check whether the specified content types are acceptable, based on the request's Accept HTTP header field

```
req.accepts('html');
```

req.get(field)

This method returns the specified HTTP request header field.

```
req.get('Content-Type');
```

req.is(type)

This method returns true if the incoming request's "Content-Type" HTTP header field matches the MIME type specified by the type parameter.

```
req.is('html');
```

req.param(name [, defaultValue])

```
req.param('name')
```

Express.js GET Request

GET and POST both are two common HTTP requests used for building REST API's. GET requests are used to send only limited amount of data because data is sent into header while POST requests are used to send large amount of data because data is sent in the body.

Express.js GET Method

Fetch data in JSON format:

Get method facilitates you to send only limited amount of data because data is sent in the header. It is not secure because data is visible in URL bar.

Refer Two Examples:

Static files are files that clients download as they are from the server. Create a new directory, public. Express, by default does not allow you to serve static files. You need to enable it using the following built-in middleware.

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

Express.js POST Request

Post method facilitates you to send large amount of data because data is send in the body. Post method is secure because data is not visible in URL bar but it is not used as popularly as GET method. On the otherhand GET method is more efficient and used more than POST.

Fetch data in JSON format

Refer the file for Output:

Serving Static Files in Express.js

Express.js is a great framework for creating APIs, but it can also easily be used to serve static files, whether they're images, CSS, HTML, JavaScript or whatever else you need to host. Express makes serving static files ridiculously easy!

It is easy to serve static files using built-in middleware in Express.js called express.static. Using express.static() method, you can server static resources directly by specifying the folder name where you have stored your static resources.

Creating Our Express Server:

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

app.get('/', (req, res) => {
    res.send('An static files demo');
});

app.listen(3000, () => console.log(' app listening on port 3000!'));
```

Now we need to make some changes so that we'll be serving the static files from our public directory. Let's tell Express to serve users files from our public directory. We do this by adding the following line to our code:

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

Let's look at what this does: when we call app.use(), we're telling Express to use a piece of middleware. Middleware is a slightly more complicated topic that we're not going to go into here, but in short, a middleware function is a function that Express passes requests through before it sends them to our routing functions, such as the app.get('/') route above. We tell Express to use the express.static middleware. express.static is a piece of middleware that comes built into Express, it's purpose is to try to find and return the static file requested. The parameter we pass to the express.static function is the name of the directory we want Express to serve files from, in our case it's public.

Putting it All Together

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

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

app.get('/', (req, res) => {
    res.send('An static files demo');
});

app.listen(3000, () => console.log(' app listening on port 3000!'));
```

Running our app:

Open your web browser and navigate to localhost: 3000, you should see the message An Static Files Demo.

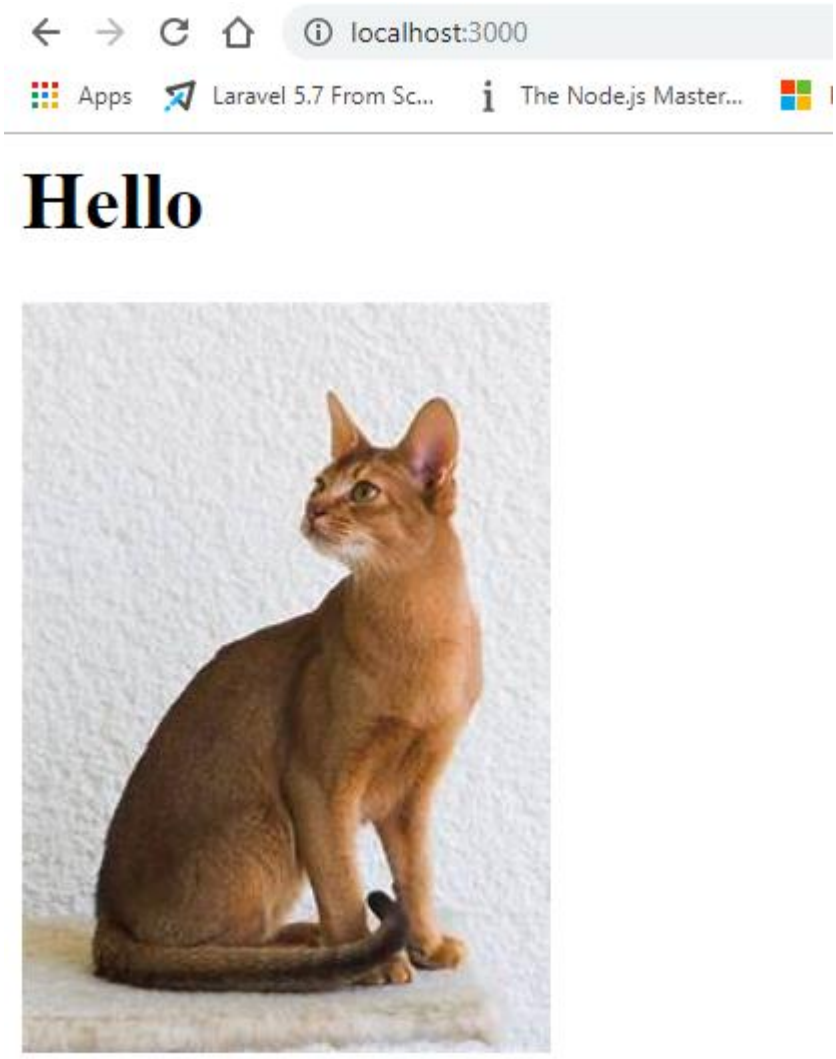
Creating our Web Page:

At this point, we have an Express server set up to serve static files, but we don't have anything for it to serve.

First, create a file called index.html in the public directory. This file will be our home page. You can put whatever you want in it. We're going to make a simple Hello World webpage for simplicity sake.

```
<head>
  <title>Hello World!</title>
</head>
<body>
  <h1>Hello</h1>
  <img src="cat.jpg">
</body>
</html>
```

Now, if you open localhost:3000 in your browser, you should see our webpage:



`

Express Routing:

Routing is made from the word route. It is used to determine the specific behaviour of an application. It specifies how an application responds to a client request to a particular route, URI or path and a specific HTTP request method (GET, POST, etc.). It can handle different types of HTTP requests.

Express.js Cookies:

Cookies are small piece of information i.e. sent from a website and stored in user's web browser when user browses that website. Every time the user loads that website back, the browser sends that stored data back to website or server, to recognize user.

Import cookie-parser into your app.

Cookie-parser parses Cookie header and populate req.cookies with an object keyed by the cookie names.

Scaffolding

Scaffolding is a technique that is supported by some MVC frameworks.

Scaffolding facilitates the programmers to specify how the application data may be used. This specification is used by the frameworks with predefined code templates, to generate the final code that the application can use for CRUD operations (create, read, update and delete database entries).

- **npm install -g express-generator**
- create a folder and use command to create web directory as **express**
- visit the folder to view the content
- to start an application use command
- **npm start**
- if any error then type
- **npm install**
- **npm start**