



# Express.js

- Express.js is a web framework for Node.js.
- It is a fast, robust and asynchronous in nature.
- Express is a fast, assertive, essential and moderate web framework of Node.js.
- 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.

# Features

- It can be used to design single-page, multi-page and hybrid web applications.
- It allows to setup middlewares to respond to HTTP Requests.
- It defines a routing table which is used to perform different actions based on HTTP method and URL.
- It allows to dynamically render HTML Pages based on passing arguments to templates.

# Why use Express

- Ultra fast I/O
- Asynchronous and single threaded
- MVC like structure
- Robust API makes routing easy

# 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.

- Express is a web framework for Node.js.
- Using Express you can build web applications, REST APIs, frameworks quickly and efficiently.
- First, create a new folder and initialize it with a blank **package.json** file using the command below.
- `npm init --y`

# Install Express.js

- To install the express framework globally to create web application using Node terminal.
- Use the following command to install express framework globally.
  - `npm install -g express`

# Installing Express

- To install the latest and stable version Express in your project, run the following command.
- `npm install --save express`
- Or
- `npm install express --save`
- The above command install express in node\_module directory and create a directory named express inside the node\_module.



- install some other important modules along with express.
- **body-parser:** This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.

`npm install body-parser --save`

- **cookie-parser:** It is used to parse Cookie header and populate req.cookies with an object keyed by the cookie names.
- `npm install cookie-parser --save`
- **multer:** This is a node.js middleware for handling multipart/form-data.

app.js

# Display Hello world

```
const express = require('express');  
const app = express();  
app.get('/', (req, res) => {  
  res.send("Hello TE IT");  
});  
app.listen(process.env.port || 3000);  
console.log('Web Server is listening at port  
' + (process.env.port || 3000));
```

Run on cmd as node app.js

*Navigate your browser to **localhost:3000** to view the response of the web server.*

- `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(re s)***.
- 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.
- `app.listen(port, [host], [backlog], [callback])`
- This function binds and listens for connections on the specified host and port.
- Port is the only required parameter here.

# Express Router

- ***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.

- The app object includes `get()`, `post()`, `put()` and `delete()` methods to define routes for HTTP GET, POST, PUT and DELETE requests respectively.

# 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')  
})
```



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

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

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

```
  console.log("Got a GET request for the  
  homepage");
```

```
  res.send('Welcome to TEIT !');
```

```
})
```

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

```
  console.log("Got a POST request for the  
  homepage");
```

```
  res.send('I am Impossible! '); })
```

```
app.delete('/del_student', function (req, res) {
```

```
  console.log("Got a DELETE request for  
  /del_student");
```

```
app.get('/enrolled_student', function (req, res) {  
  console.log("Got a GET request for  
  /enrolled_student");  
  res.send('I am an enrolled student.');
```

```
})  
// This responds a GET request for abcd, abxcd,  
  ab123cd, and so on
```

```
app.get('/ab*cd', function(req, res) {  
  console.log("Got a GET request for /ab*cd");  
  res.send('Pattern Matched.');
```

*see the result generated by server on the local host <http://127.0.0.1:8000>*

```
})  
var server = app.listen(8000, function () {  
  var host = server.address().address
```

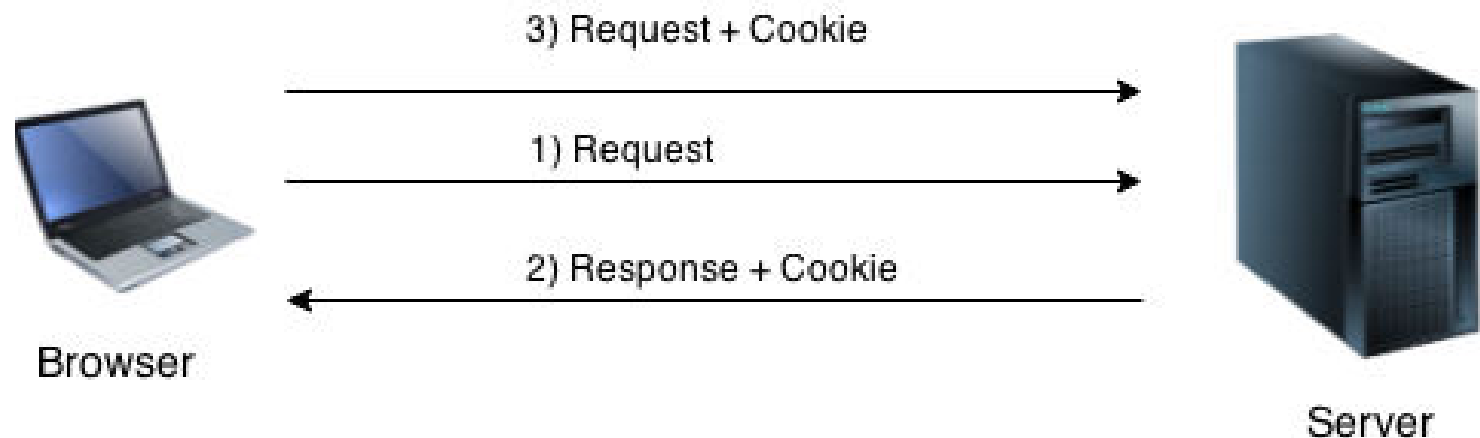
# Note:

- The Command Prompt will be updated after one successful response.
- can see the different pages by changing routes.
- *local host* <http://127.0.0.1:8000>
- **http://127.0.0.1:8000/enrolled\_student**

Next route **http://127.0.0.1:8000/abcd**

# Express.js Cookies Management

- 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.



# Install cookie

- To acquire cookie abilities in Express.js.
- So, install cookie-parser middleware through npm by using the following command:
- `npm install cookie-parser --save`

# To use cookies with Express...

1) We need the cookie-parser middleware

```
npm install --save cookie-parser
```

middleware which parses cookies attached to the client request object

2) We will require the cookie-parser

```
var cookieParser = require('cookie-parser');  
app.use(cookieParser());
```

3) Define a new route in your Express app

```
var express = require('express');  
var app = express();  
app.get('/', function(req, res){  
  res.cookie('name', 'express').send('cookie set');  
});  
app.listen(3000);
```

Sets the name of the cookie as "express"

cookiep.js

# Example

```
var express = require('express');  
var cookieParser = require('cookie-parser');  
var app = express();  
app.use(cookieParser());  
app.get('/cookieset',function(req, res){  
  res.cookie('cookie_name', 'cookie_value');  
  res.cookie('company', 'XIE');  
  res.cookie('name', 'TEIT');  
  res.status(200).send('Cookie is set');  
});
```

```
app.get('/cookieget', function(req, res) {  
  res.status(200).send(req.cookies);  
});  
  
app.get('/', function (req, res) {  
  res.status(200).send('Welcome to Express Cookie  
  Concept!');  
});  
  
var server = app.listen(8000, function () {  
  var host = server.address().address;  
  var port = server.address().port;  
  console.log('Example app listening at  
  http://%s:%s', host, port);  
});
```

Open the page <http://127.0.0.1:8000/> on your browser  
Now open <http://127.0.0.1:8000/cookieset> to set the cookie:  
Now open <http://127.0.0.1:8000/cookieget> to get the cookie:



# Adding cookies with Expiration time

- Pass an object with property 'expire' set to the time when you want it to expire.

```
res.cookie(name, 'value', {expire: 360000 + Date.now()});
```

Expires after 360000 ms from the time it is set

- Use 'maxAge' property.

```
res.cookie(name, 'value', {maxAge: 360000});
```

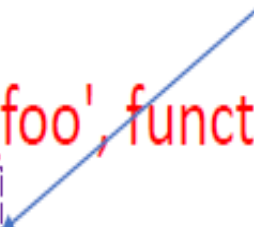
Expires after 360000 ms from the time it is set

# Deleting the existing cookies

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

```
app.get('/clear_cookie_foo', function(req, res){  
  res.clearCookie('foo');  
  res.send('cookie foo cleared');  
});
```

Clear a cookie named 'foo'



```
app.listen(3000);
```

# Session

- HTTP is stateless; in order to associate a request to any other request, you need a way to store user data between HTTP requests.
- Cookies and URL parameters are both suitable ways to transport data between the client and the server.
- But they are both readable and on the client side.
- Sessions solve exactly this problem.
- You assign the client an ID and it makes all further requests using that ID.

# Managing sessions with Express...

- To install express session

`npm install --save express-session`

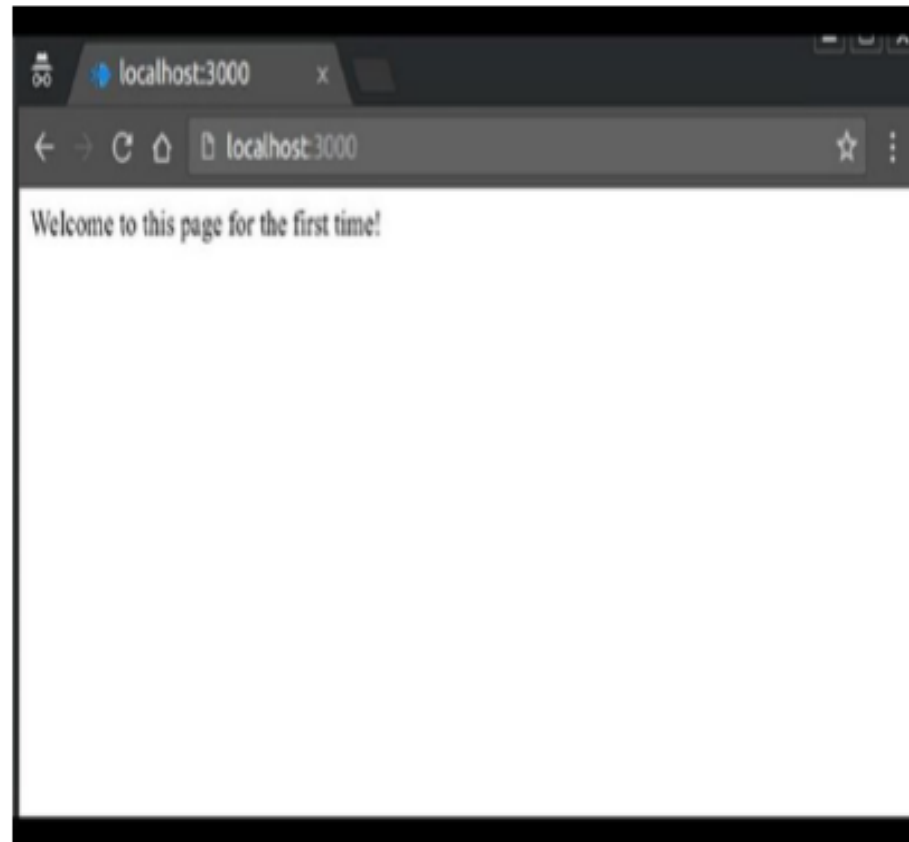
- The session middleware handles all things like creating the session, setting the session cookie and creating the session object in req object.
- Whenever we make a request from the same client again, we will have their session information stored with us.

# example

```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');
var app = express();
app.use(cookieParser());
app.use(session({secret: "Shh, its a secret!"}));
app.get('/', function(req, res){
  if(req.session.page_views){
    req.session.page_views++;
    res.send("You visited this page " +
      req.session.page_views + " times");
  }
}
```

```
else {  
    req.session.page_views = 1;  
    res.send("Welcome to this page for the  
first time!");  
}  
});  
app.listen(3000);
```

# Output



Note: The page counter will increase next time you visit the page

# Authentication

- Process in which the credentials provided are compared to those on file in a database of authorized users' information on a local operating system or within an authentication server.
- If the credentials match, the process is completed and the user is granted authorization for access.
- Passport is authentication middleware for Node.js.

`$ npm install passport-local`



# Express.js Middleware

- Express.js Middleware are different types of functions that are invoked by the Express.js routing layer before the final request handler.
- Middleware appears in the middle between an initial request and final intended route.
- Middleware is commonly used to perform tasks like body parsing for URL-encoded or JSON requests, cookie parsing for basic cookie handling, or even building JavaScript modules on the fly.

# What is a Middleware function

- Middleware functions are the functions that access to the request and response object (req, res) in request-response cycle.
- A middleware function can perform the following tasks:
- It can execute any code.
- It can make changes to the request and the response objects.
- It can end the request-response cycle.
- It can call the next middleware function in the stack.

- Following is a list of possibly used middleware in Express.js app:
- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

# Example

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

app.get('/', function(req, res) {
  res.send('Welcome TEIT ... Middleware here!');
});

app.get('/help', function(req, res) {
  res.send('How can I help You?');
});

var server = app.listen(8000, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
})
```

http://127.0.0.1:8000/

http://127.0.0.1:8000/help

# Express.js Scaffolding

- Scaffolding is a technique that is supported by some MVC frameworks.
- It is mainly supported by the following frameworks:
- Ruby on Rails, OutSystems Platform, Express Framework, Play framework, Django, MonoRail, Brail, Symfony, Laravel, CodeIgniter, Yii, CakePHP, Phalcon PHP, Model-Glue, PRADO, Grails, Catalyst, Seam Framework, Spring Roo, ASP.NET etc.
- 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).

# Express.js Scaffold

- An Express.js scaffold supports candy and more web projects based on Node.js.
- Install scaffold
- Execute the following command to install scaffold.
- `npm install express-scaffold`

# Scaffolding using Yeoman

- install Yeoman  
`npm install -g yeoman`
- To install 'generator-Express-simple' generator  
`npm install -g generator-express-simple`
- To use this generator  
`yo express-simple test-app`

# Express application generator

- The Express-generator package is a utility that provides a command-line tool to use to scaffold your project - ie create boilerplate folder structure, files and code.
- Use the application generator tool, **express-generator**, to quickly create an application skeleton.
- Run the application generator with the npx command (available in Node.js 8.2.0).
- **\$ npx express-generator**
- Install the application generator as a global npm package and then launch it:
- **\$ npm install -g express-generator**
- **\$ express myapp**



# Express Generator

- This creates a new Express project called myapp, which is then placed inside of the myapp directory:
- **cd myapp**
- **npm install**
- Running npm install installs all dependencies for the project.
- By default, the express-generator includes several packages that are commonly used with an Express server.

- \$ express -h

```
$ express -h
```

```
Usage: express [options] [dir]
```

```
Options:
```

-h, --help	output usage information
--version	output the version number
-e, --ejs	add ejs engine support
--hbs	add handlebars engine support
--pug	add pug engine support
-H, --hogan	add hogan.js engine support
--no-view	generate without view engine
-v, --view <engine>	add view <engine> support (ejs hbs hjs jade pug twig vash) (defaults to jade)
-c, --css <engine>	add stylesheet <engine> support (less stylus compass sass) (defaults to plain css)
--git	add .gitignore
-f, --force	force on non-empty directory

- The generator CLI takes half a dozen arguments, but the two most useful ones are the following:
- **-v** . This lets you select a view engine to install. The default is jade. Although this still works, it has been deprecated and you should always specify an alternative engine.
- **-c** . By default, the generator creates a very basic CSS file for you, but selecting a CSS engine will configure your new app with middleware to compile any of the above options.

- The app will be created in a folder named *myapp* in the current working directory and the view engine will be set to Pug:
- `$ express --view=pug myapp`

```
create : myapp
create : myapp/package.json
create : myapp/app.js
create : myapp/public
create : myapp/public/javascripts
create : myapp/public/images
create : myapp/routes
create : myapp/routes/index.js
create : myapp/routes/users.js
create : myapp/public/stylesheets
create : myapp/public/stylesheets/style.css
create : myapp/views
create : myapp/views/index.pug
create : myapp/views/layout.pug
create : myapp/views/error.pug
create : myapp/bin
create : myapp/bin/www
```

- On MacOS or Linux, run the app with this command:
- `$ DEBUG=myapp:* npm start`
- On Windows Command Prompt, use this command:
- `> set DEBUG=myapp:* & npm start`
- Then load `http://localhost:3000/` in your browser to access the app.

# Directory structure

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

7 directories, 9 files

- **bin**
- The bin folder contains the executable file that starts your app. It starts the server (on port 3000, if no alternative is supplied) and sets up some basic error handling.
- **public**
- The public folder is one of the important ones: *everything* in this folder is accessible to people connecting to your application. In this folder, you'll put JavaScript, CSS, images, and other assets that people need when they load your website.
- **routes**
- The routes folder is where you'll put your router files.
- The generator creates two files, index.js and users.js, which serve as examples of how to separate out your application's route configuration.
- Usually, you'll have a different file here for each major route on your website.
- So you might have files called blog.js, home.js, and/or about.js in this folder.

- **views**
- The views folder is where you have the files used by your templating engine. The generator will configure Express to look in here for a matching view when you call the render method.
- Outside of these folders, there's one file that you should know well.
- **app.js**
- The app.js file is special, because it sets up your Express application and glues all of the different parts together.
- **express.json**. You might notice that there are two lines for parsing the body of incoming HTTP requests. The first line handles when JSON is sent via a POST request and it puts this data in request.body.
- **express.urlencoded**. The second line parses query string data in the URL (e.g. /profile?id=5) and puts this in request.query.
- **cookieParser**. This takes all the cookies the client sends and puts them in request.cookies. It also allows you to modify them before sending them back to the client, by changing response.cookies.
- **express.static**. This middleware serves static assets from your public folder. If you wanted to rename or move the public folder, you can change the path here.



SUCCESS  
is dependent on effort.

— Stephen Covey

ALL THE BEST