Express.js

Harshita Maheshwari

Introduction

Fast, unopinionated, minimalist web framework for Node.js

What is Express??

- Express.js is a very popular web application framework built to create Node.js Web based applications.
- It provides an integrated environment to facilitate rapid development of Node based Web applications.
- Express framework is based on Connect middleware engine and used Jade html template framework for HTML templating.
- Developed by TJ Holowaychuk in Nov 2010

Why use Express??

- Express lets you build single page, multi-page, and hybrid web and mobile applications. Other common backend use is to provide an API for a client (whether web or mobile).
- It comes with a default template engine, Jade which helps to facilitate the flow of data into a website structure and does support other template engines.
- It supports MVC (Model-View-Controller), a very common architecture to design web applications.
- It is cross-platform and is not limited to any particular operating system.
- It leverages upon Node.js single threaded and asynchronous model.

Advantages

- 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

Install Express

To install Express.js, first, you need to create a project directory and create a package.json file which will be holding the project dependencies. Below is the code to perform the same:

```
npm init
```

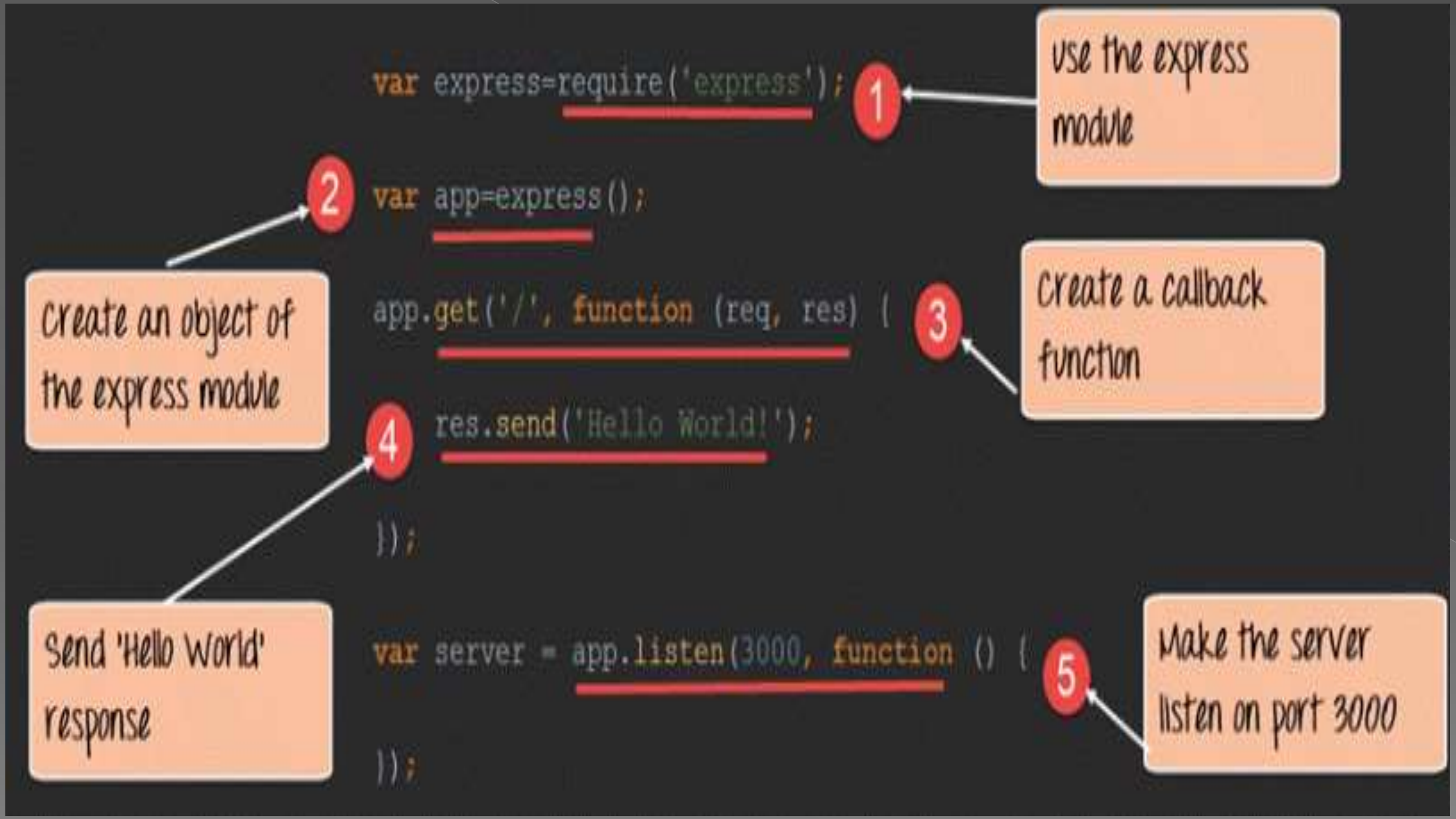
Now, you can install the express.js package in your system. To install it globally, you can use the below command:

```
npm install -g express
```

Or, if you want to install it locally into your project folder, you need to execute the below command:

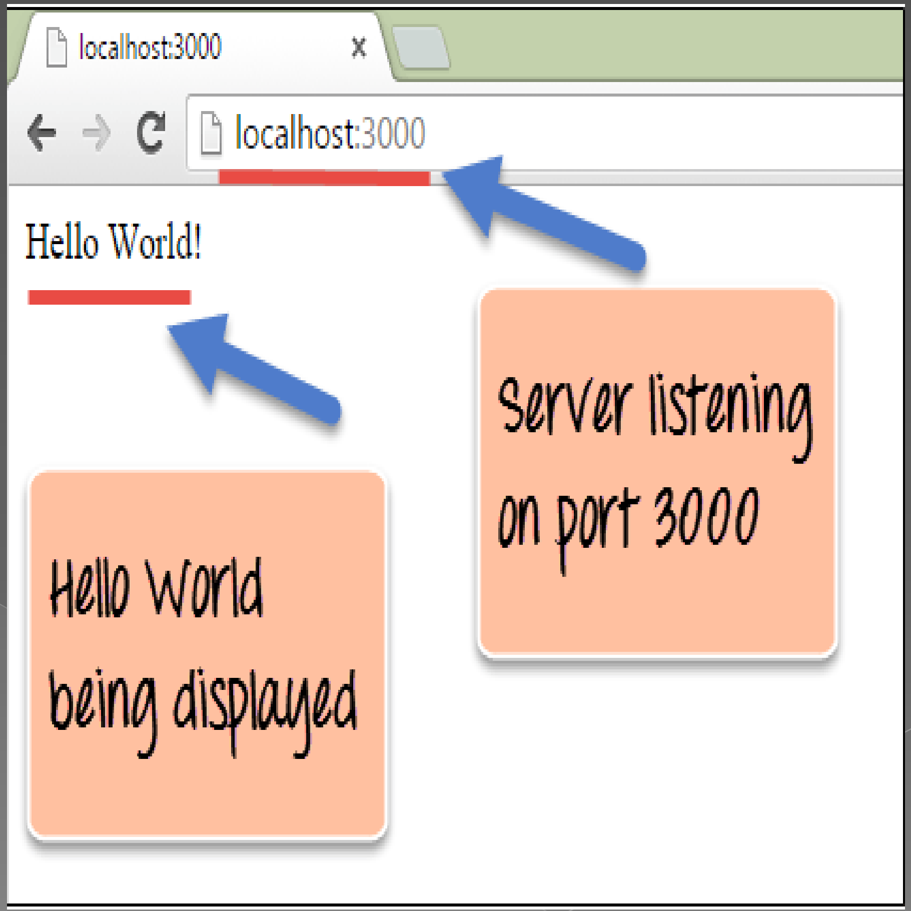
```
npm install express --save
```

Simple Server on Express.js



```
var express=require('express');
var app=express();
app.get('/', function (req, res) {
    res.send('Hello World!');
});
var server = app.listen(3000, function () {
});
```

1
Use the express module
2
Create an object of the express module
3
Create a callback function
4
Send 'Hello World' response
5
Make the server listen on port 3000



Express.js Request Object

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

Properties	Description
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.
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 {}.
req.path	It contains the path part of the request URL
req.query	An object containing a property for each query string parameter in the route.
req.route	The currently-matched route, a string.
req.cookies	When we use cookie-parser middleware, this property is an object that contains cookies sent by the request.

Express.js Response Object

The Response object (res) specifies the HTTP response which is sent by an Express app when it gets an HTTP request.

Response Object Methods:-

method	Description
res.end()	End the response process
res.json()	Send a JSON response
res.redirect()	Redirect a request
res.render()	Render a review template
res.send()	Send a response of various types
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

Routing

Routing determine the way in which 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.

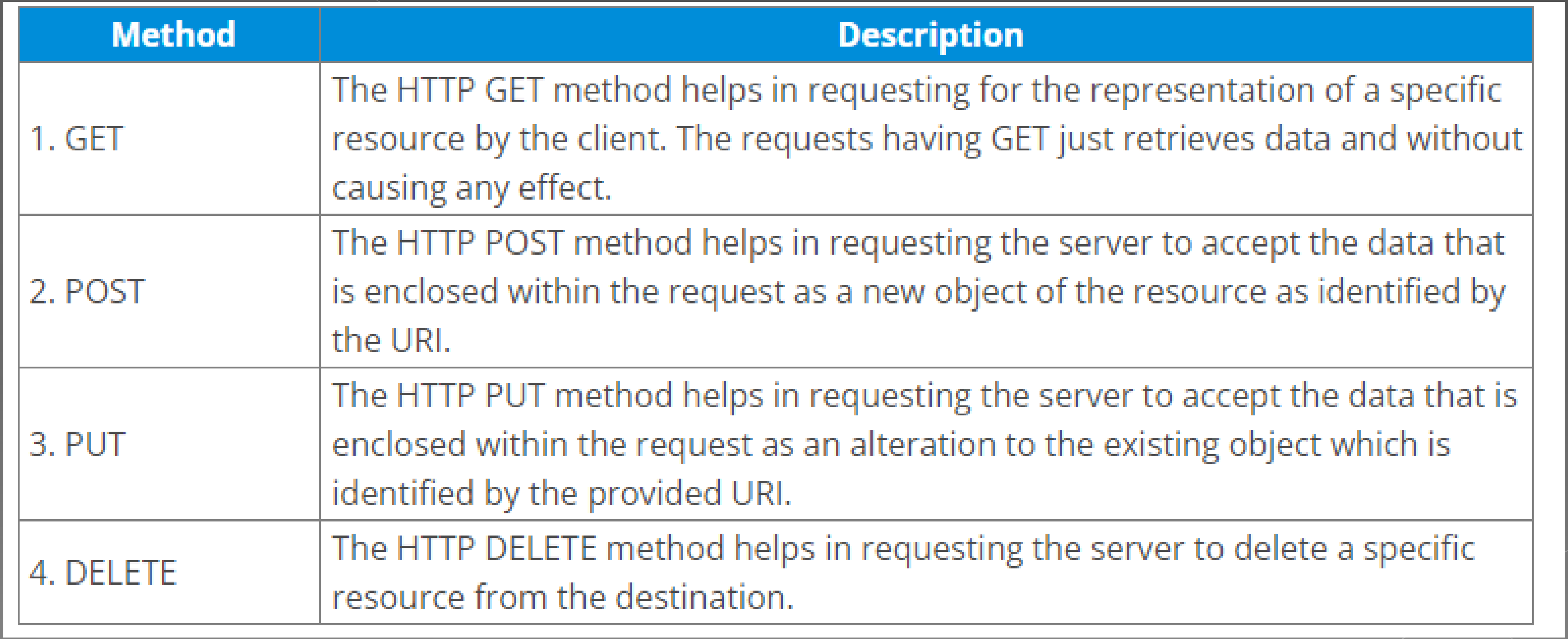
The general syntax for a route is shown below: -

app.METHOD(PATH, HANDLER)

Wherein,

1) app is an instance of the express module
2) METHOD is an HTTP request method (GET, POST, PUT or DELETE)
3) PATH is a path on the server.
4) HANDLER is the function executed when the route is matched.

HTTP Methods



Method	Description
1. GET	The HTTP GET method helps in requesting for the representation of a specific resource by the client. The requests having GET just retrieves data and without causing any effect.
2. POST	The HTTP POST method helps in requesting the server to accept the data that is enclosed within the request as a new object of the resource as identified by the URI.
3. PUT	The HTTP PUT method helps in requesting the server to accept the data that is enclosed within the request as an alteration to the existing object which is identified by the provided URI.
4. DELETE	The HTTP DELETE method helps in requesting the server to delete a specific resource from the destination.

Route paths based on strings

This route path will match requests to the root route, /.

```
app.get('/', function (req, res) {
        res.send('root path'); });
```

This route path will match requests to /about.

```
app.get('/about', function (req, res) {
        res.send('about us page'); });
```

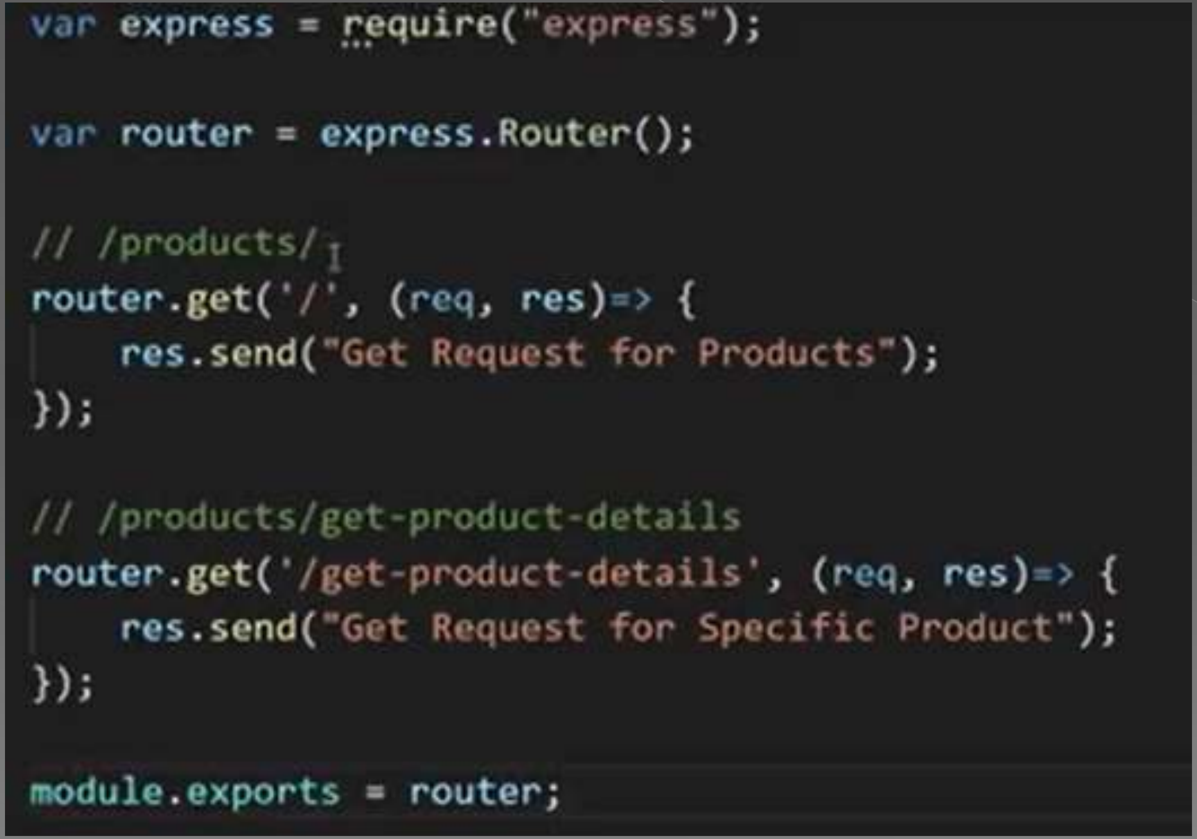
This route path will match requests to /random.text.

```
app.get('/random.text', function (req, res) {
        res.send('random.text file content'); });
```

express.Router

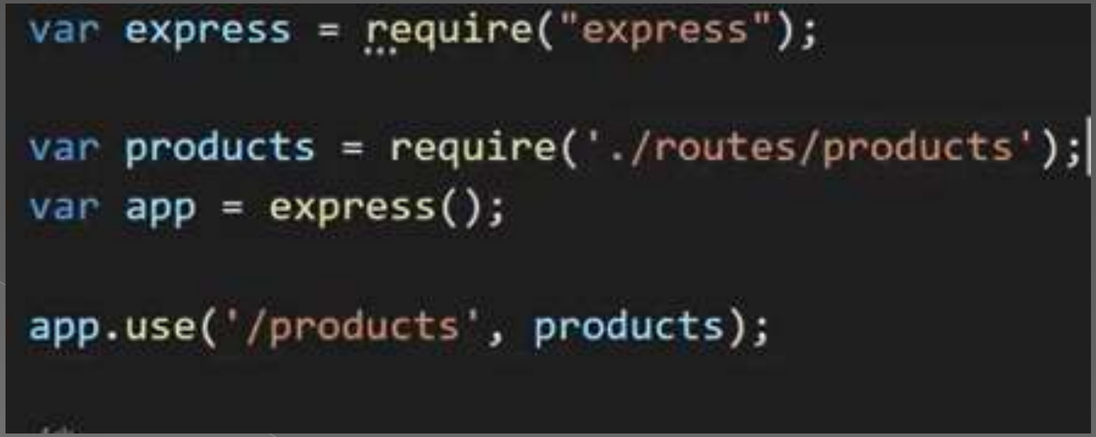
- Use the express.Router class to create modular, mountable route handlers.
- Over the period of time routes grow in size and is extremely difficult to manage.
- Using modular approach using Router we can easily develop, maintain and extend routes.
- We will need to get the Router object and then create routes for the modules.

Create one folder -> routes -→ products.js



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

var router = express.Router();

// /products/
router.get('/', (req, res)=> {
    res.send("Get Request for Products");
});

// /products/get-product-details
router.get('/get-product-details', (req, res)=> {
    res.send("Get Request for Specific Product");
});

module.exports = router;
```

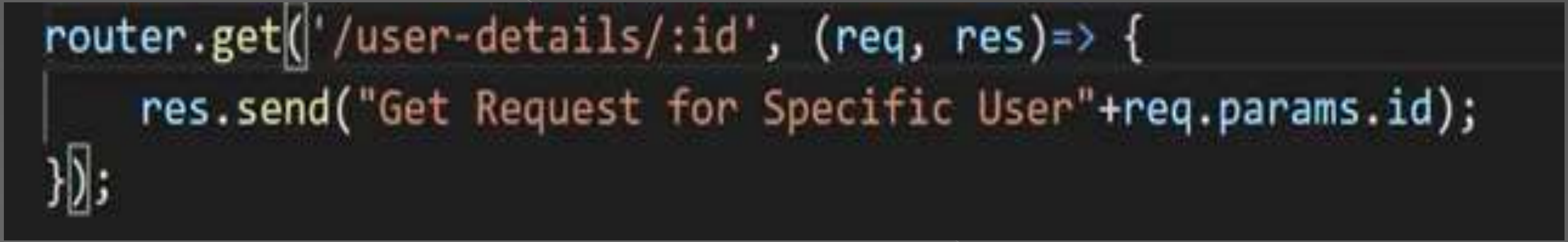
outside the routes folder -> index.js



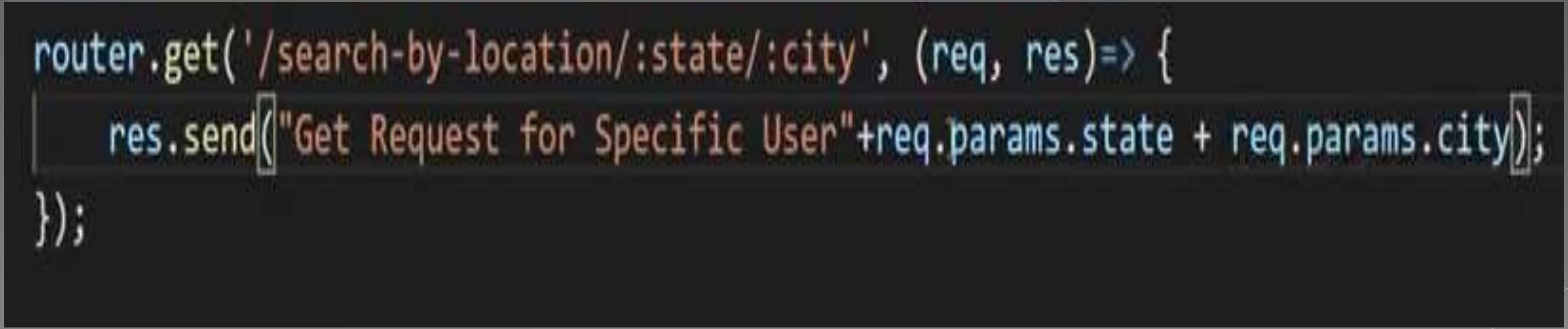
```
var express = require("express");

var products = require('./routes/products');
var app = express();

app.use('/products', products);
```

Dynamic Routes

Express allows us to build URL's dynamically.

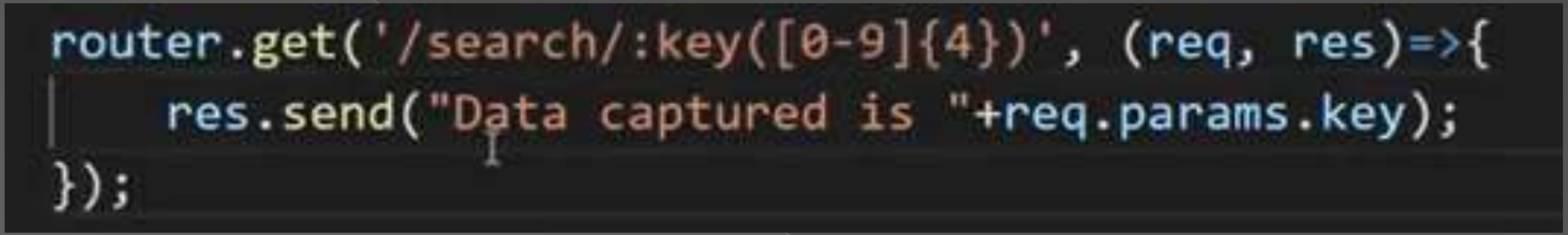


```
router.get('/user-details/:id', (req, res)=> {
    res.send("Get Request for Specific User"+req.params.id);
});
```

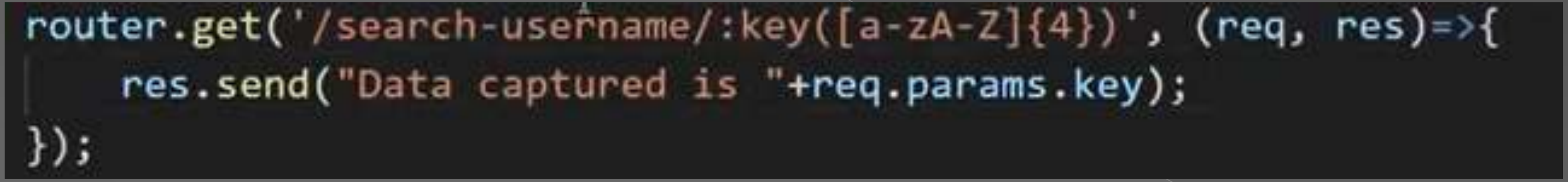


```
router.get('/search-by-location/:state/:city', (req, res)=> {
    res.send("Get Request for Specific User"+req.params.state + req.params.city);
});
```

URL Binding using regex-

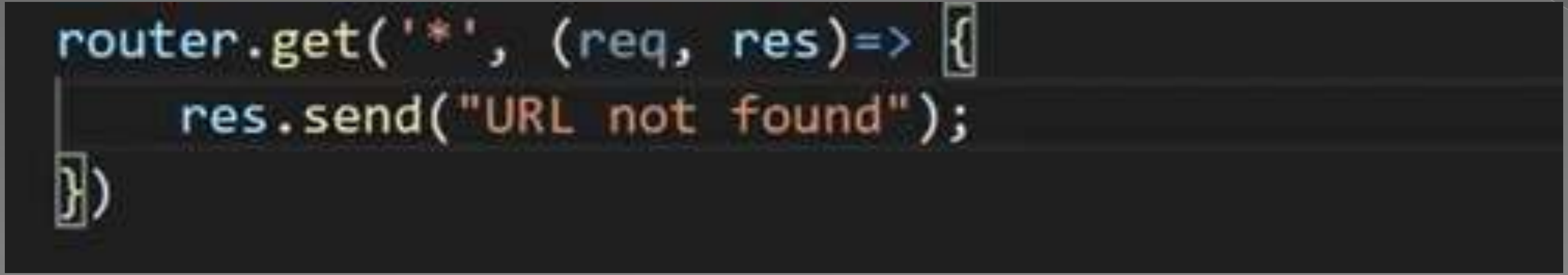


```
router.get('/search/:key([0-9]{4})', (req, res)=>{
    res.send("Data captured is "+req.params.key);
});
```



```
router.get('/search-username/:key([a-zA-Z]{4})', (req, res)=>{
    res.send("Data captured is "+req.params.key);
});
```

Wild Card Route -



```
router.get('*', (req, res)=> {
    res.send("URL not found");
})
```

Middleware

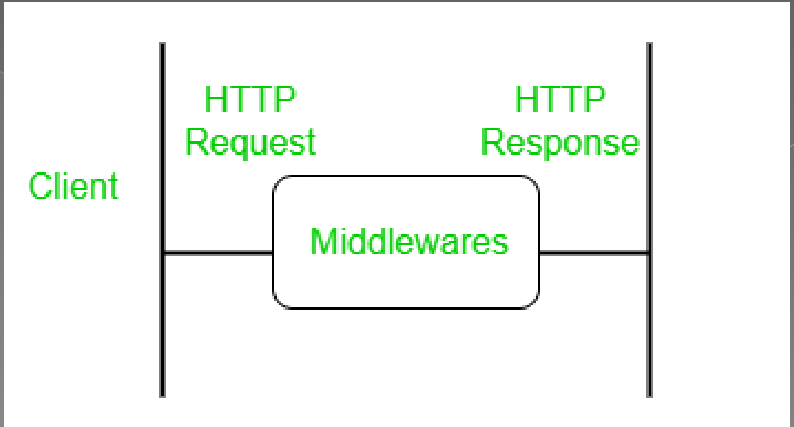
- Middleware are different types of functions that are invoked by the Express.js routing layer before the final request handler. As the name specified, Middleware appears in the middle between an initial request and final intended route. In stack, middleware functions are always invoked in the order in which they are added.

- 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.

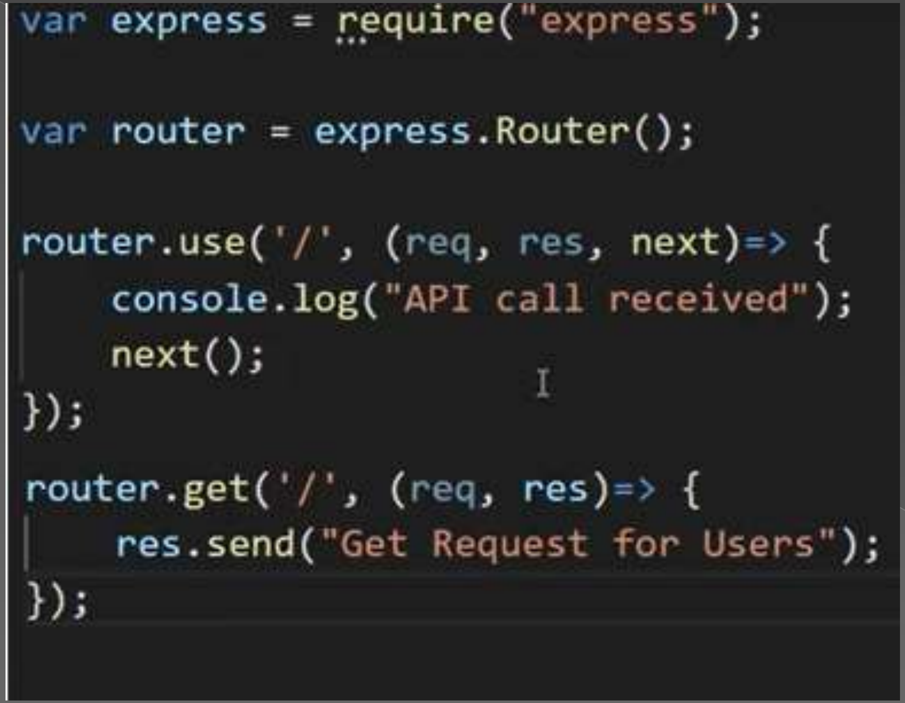
Middleware Function

- middleware functions are the functions which have access to the request and response objects along with the next function present in the application’s request-response cycle.
- Middleware can process request objects multiple times before the server works for that request.
- Middleware can be used to add logging and authentication functionality.
- Middleware improves client-side rendering performance.
- Middleware is used for setting some specific HTTP headers.
- Middleware helps for Optimization and better performance.

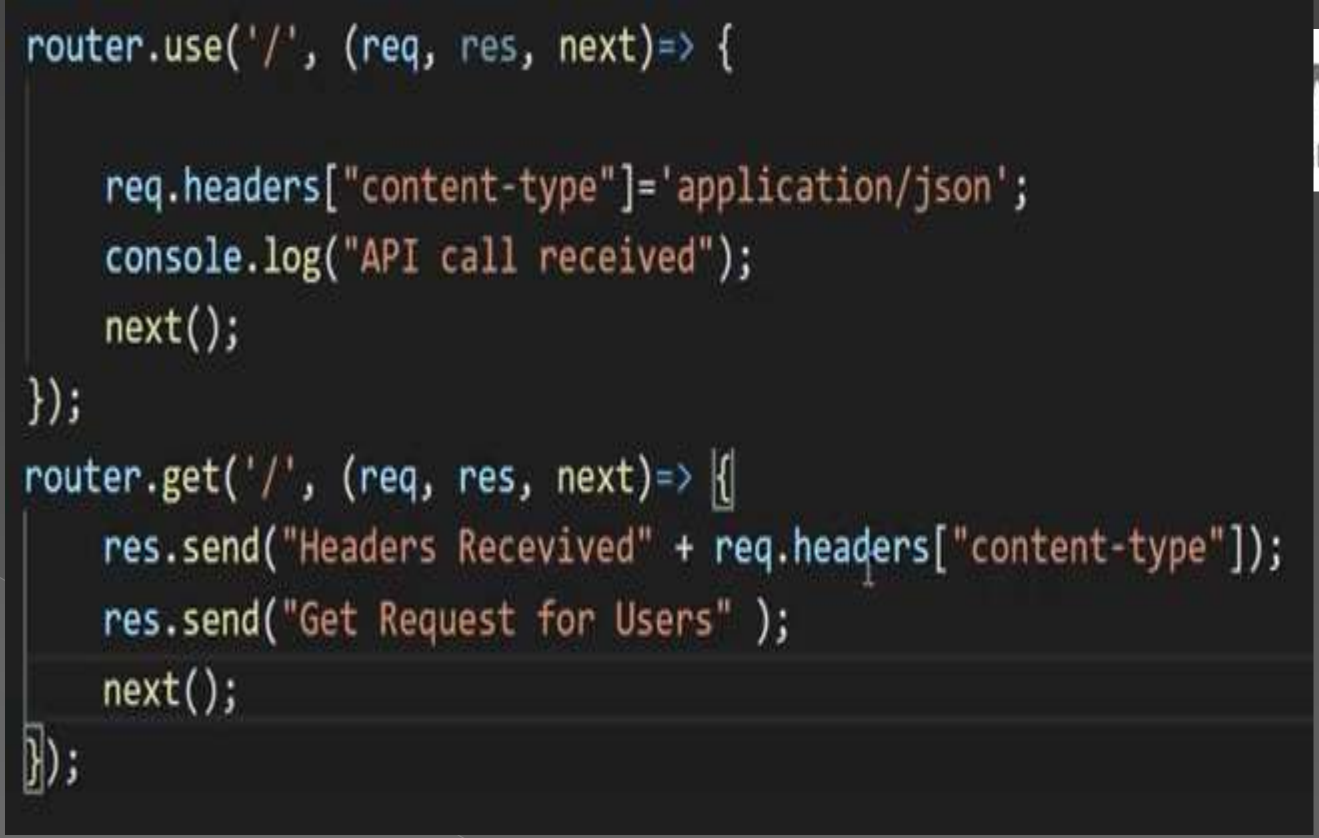
Order of methods is extremely important.



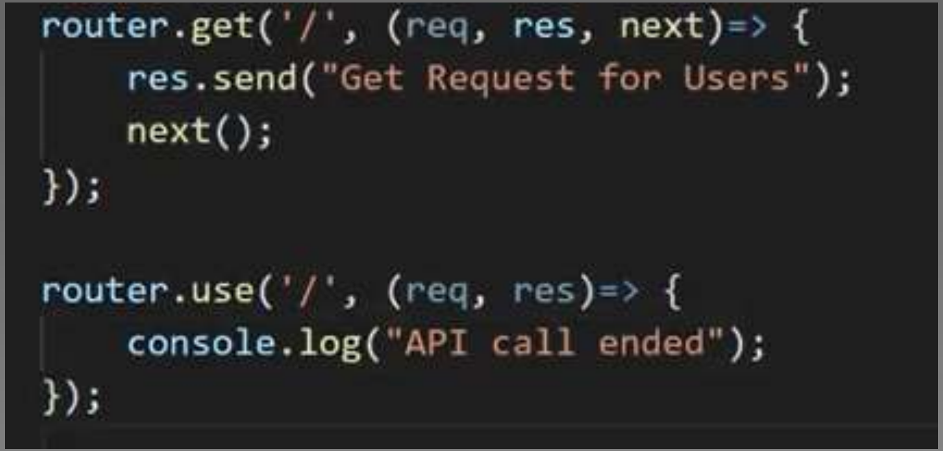
Middleware working



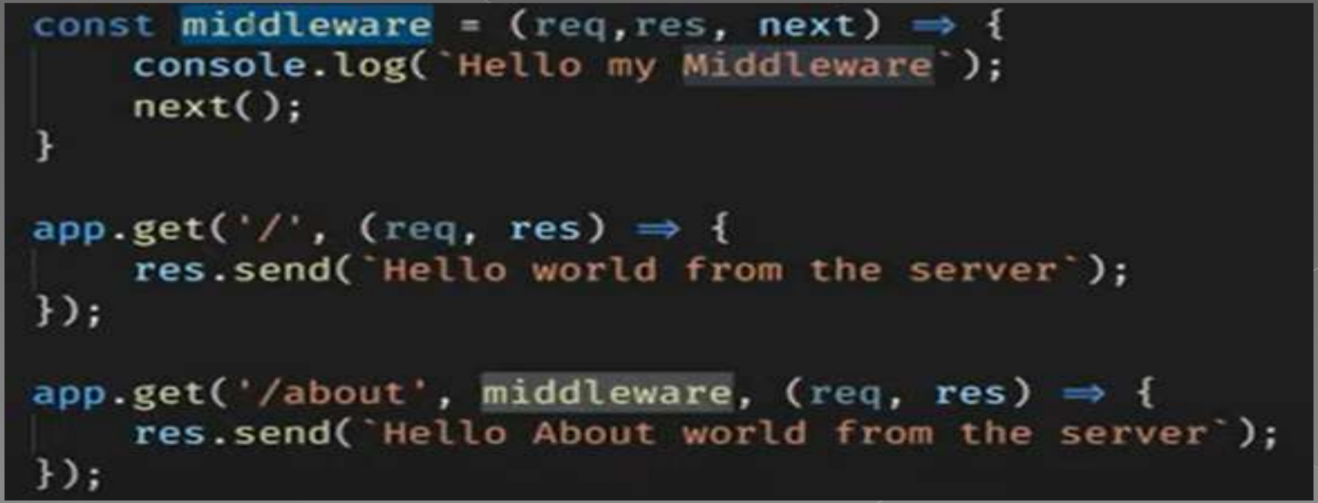
```
var express = require("express");

var router = express.Router();

router.use('/', (req, res, next)=> {
    console.log("API call received");
    next();
});

router.get('/', (req, res)=> {
    res.send("Get Request for Users");
});
```



```
router.use('/', (req, res, next)=> {

    req.headers["content-type"]='application/json';
    console.log("API call received");
    next();
});
router.get('/', (req, res, next)=> {
    res.send("Headers Recevived" + req.headers["content-type"]);
    res.send("Get Request for Users" );
    next();
});
```



```
router.get('/', (req, res, next)=> {
    res.send("Get Request for Users");
    next();
});

router.use('/', (req, res)=> {
    console.log("API call ended");
});
```



```
const middleware = (req,res, next) => {
    console.log(`Hello my Middleware`);
    next();
}

app.get('/', (req, res) => {
    res.send(`Hello world from the server`);
});

app.get('/about', middleware, (req, res) => {
    res.send(`Hello About world from the server`);
});
```

Express Generator

Application generator tool to quickly create an application skeleton.

Easily get standard application shell for quick and rapid prototyping.

How to use ->

Install Globally -> npm i –g express-generator

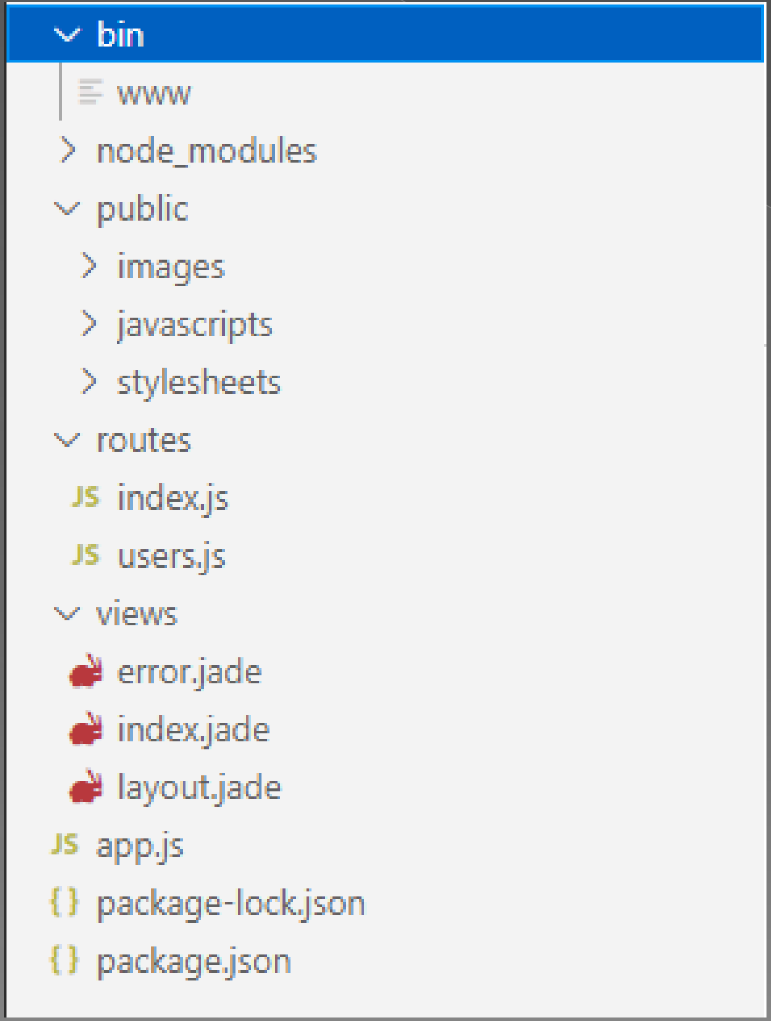
Install locally -> npm i express-generator

How to create project using express-generator

- express projectname
- Go to project folder and install dependencies-> npm install or npm i
- Start the node server -> npm start
- Go to browser and write localhost:3000 in the address bar to execute the

express code

Project Structure



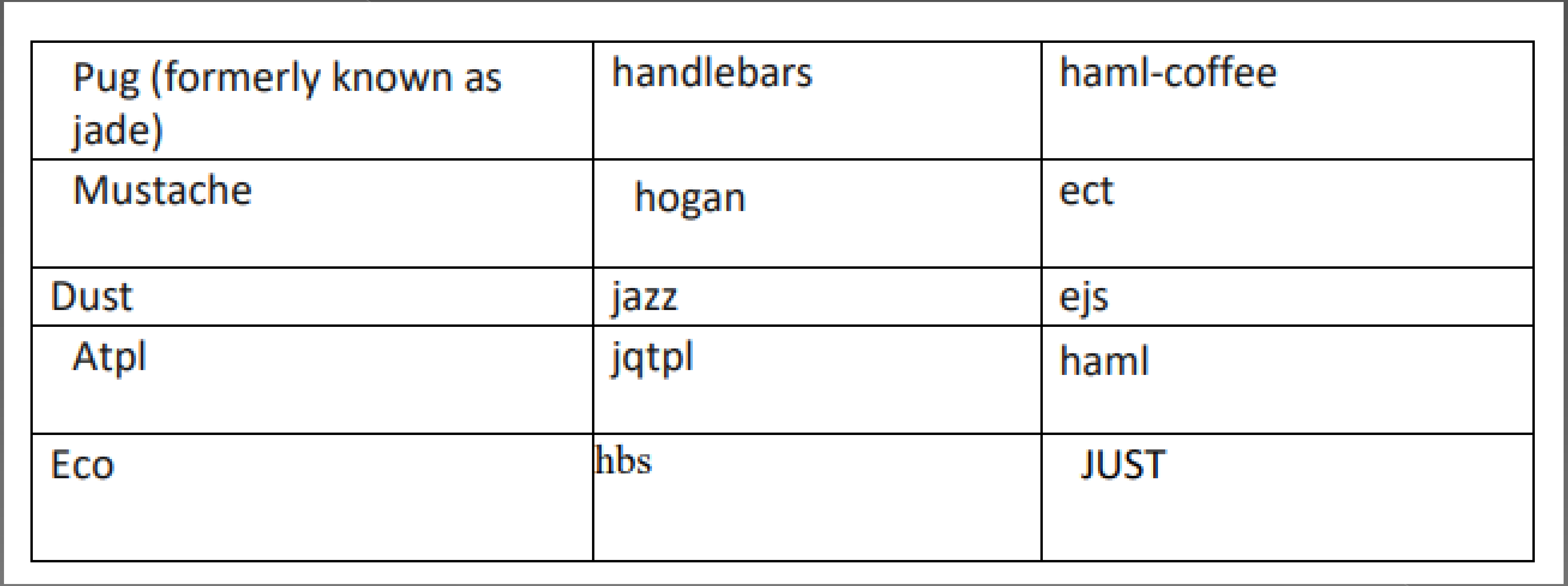
```
bin
  www
node_modules
public
  images
  javascripts
  stylesheets
routes
  index.js
  users.js
views
  error.jade
  index.jade
  layout.jade
app.js
package-lock.json
package.json
```

- app.js:- This file starts your web server. All your set up logic should be in this file.
- Public:- All the public files such as images, javascript files, CSS files should go into this folder.
- Routes:- All your routing-related logic should go into this folder.
- Views:- So this folder contains all your views i.e. HTML/hbs files. Drop this folder if you are building rest API's.

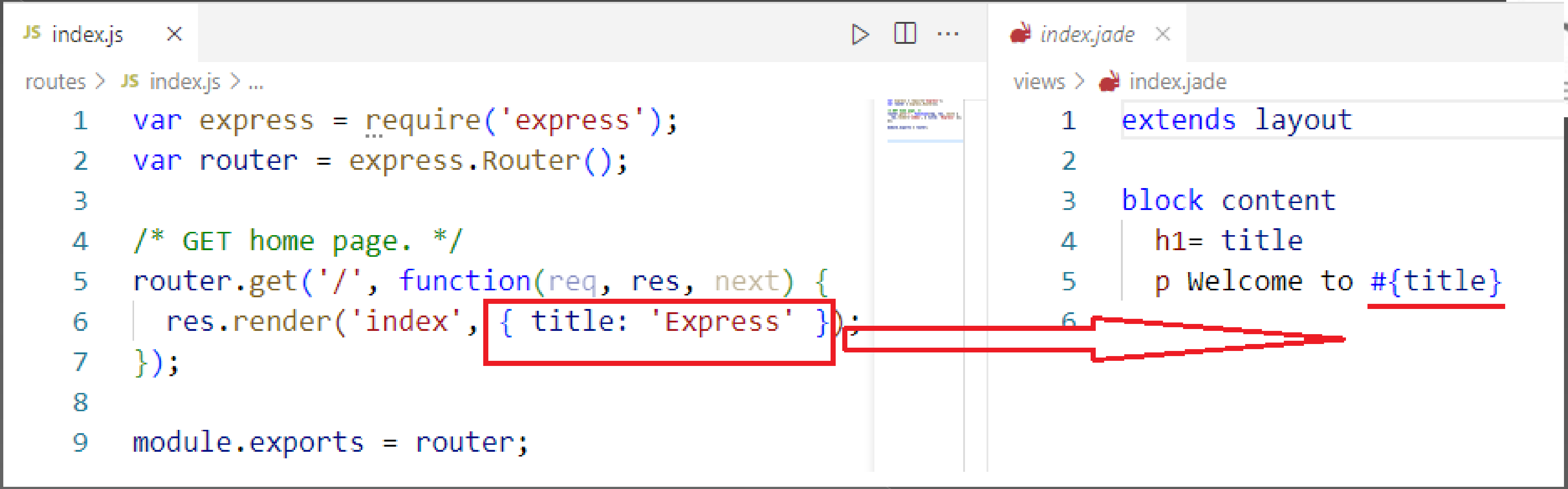
Template Engine

- 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.
- By default - jade

Types of engines

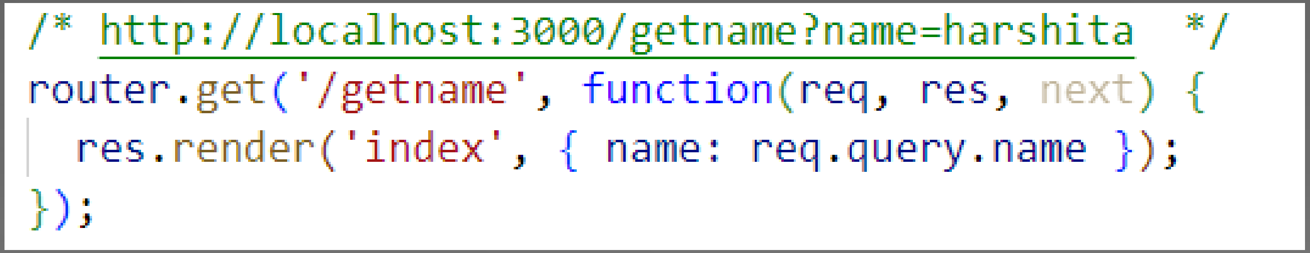


Pug (formerly known as jade)	handlebars	haml-coffee
Mustache	hogan	ect
Dust	jazz	ejs
Atpl	jqtpl	haml
Eco	hbs	JUST

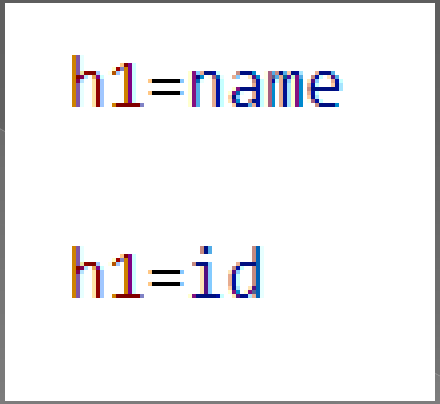


```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

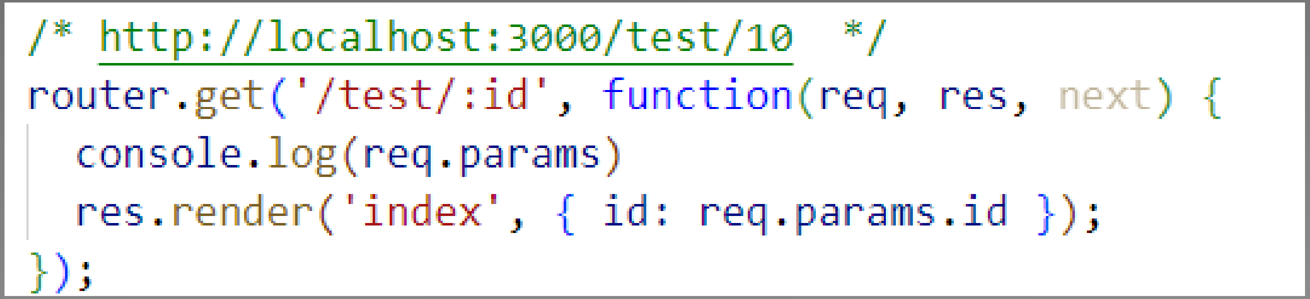
```
extends layout

block content
  h1= title
  p Welcome to #{title}
```



```
/* http://localhost:3000/getname?name=harshita  */
router.get('/getname', function(req, res, next) {
  res.render('index', { name: req.query.name });
});
```



```
h1=name

h1=id
```

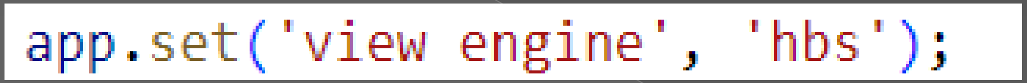


```
/* http://localhost:3000/test/10  */
router.get('/test/:id', function(req, res, next) {
  console.log(req.params)
  res.render('index', { id: req.params.id });
});
```

Infoway Technologies Pvt. Ltd., 3rd Floor Commerce Centre, Rambaug Colony, Paud Road Pune 411038

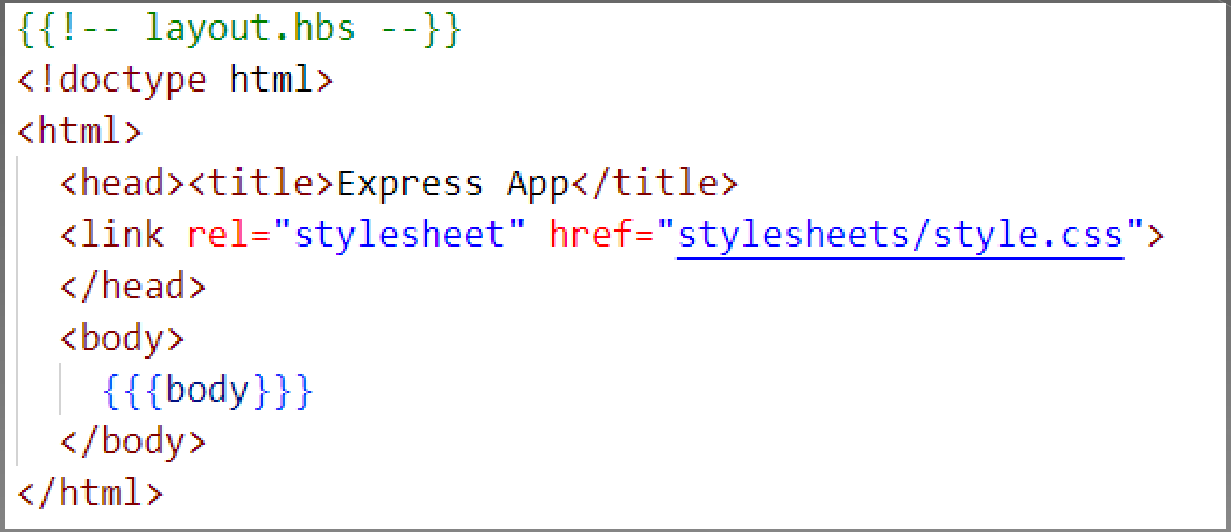
Express handlerbars Templating Engine

- Uninstall jade -> npm uninstall jade –save
- Install hbs -> npm install hbs –save
- In app.js => change view engine from jade to hbs

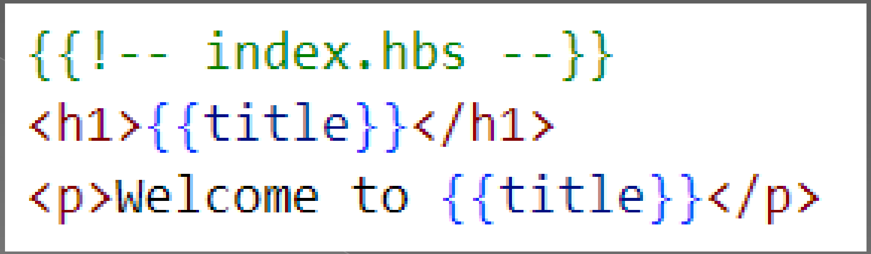


```
app.set('view engine', 'hbs');
```

- Change extension of all view files from jade to hbs and change the files content as well.

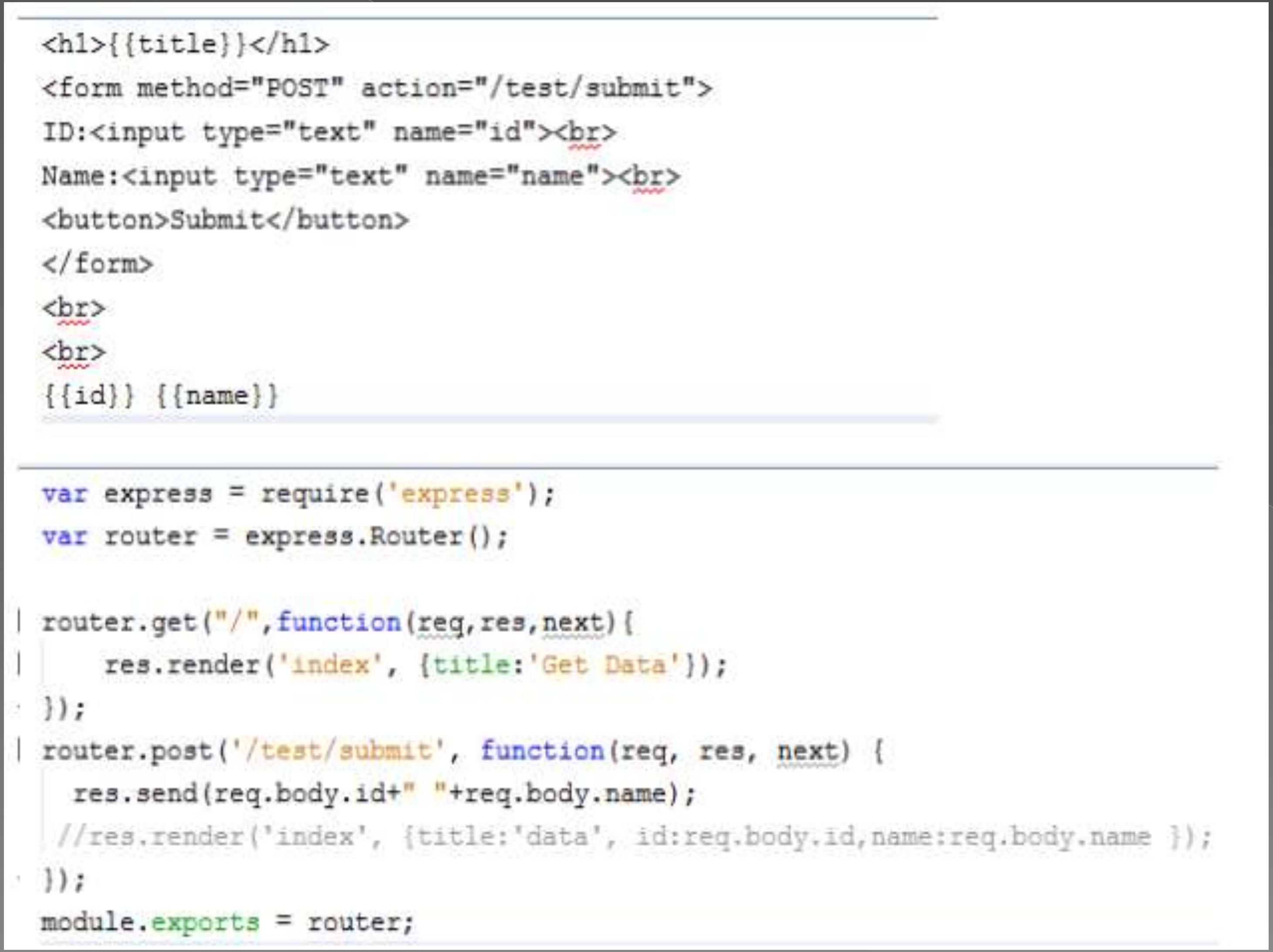


```
{{!-- layout.hbs --}}
<!doctype html>
<html>
  <head><title>Express App</title>
  <link rel="stylesheet" href="stylesheets/style.css">
  </head>
  <body>
    {{{body}}}
  </body>
</html>
```



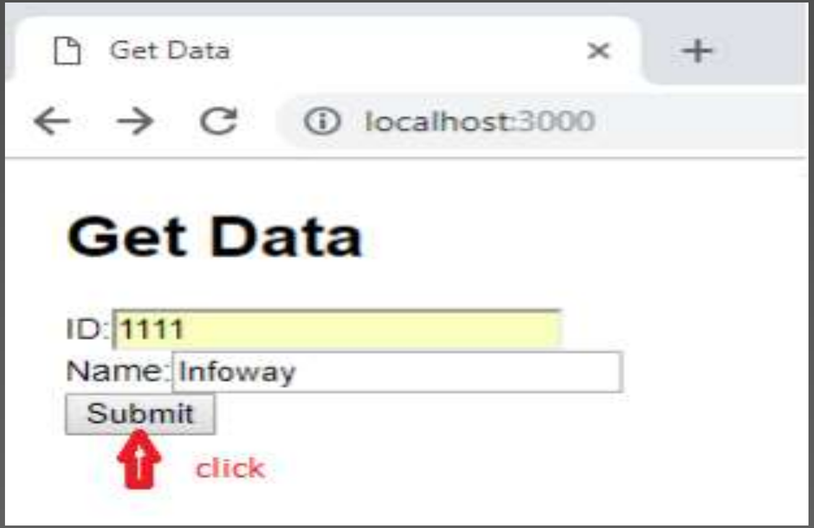
```
{{!-- index.hbs --}}
<h1>{{title}}</h1>
<p>Welcome to {{title}}</p>
```

Handling Form Data



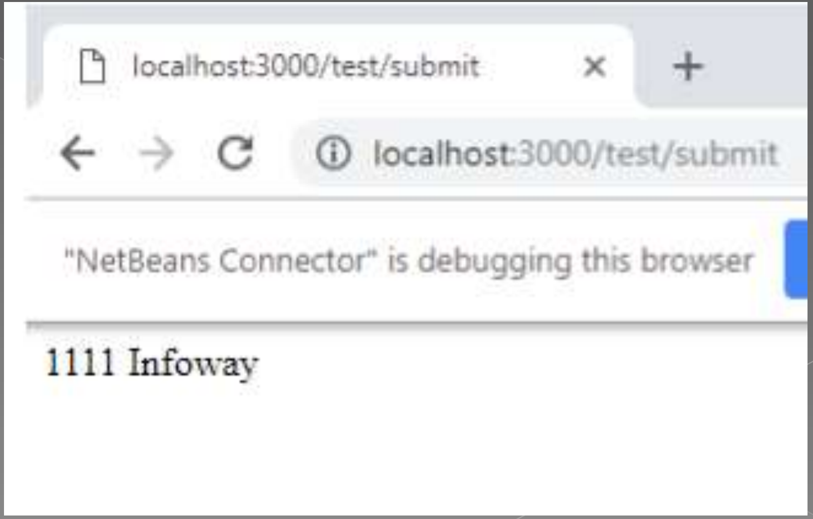
```
<h1>{{title}}</h1>
<form method="POST" action="/test/submit">
ID:<input type="text" name="id"><br>
Name:<input type="text" name="name"><br>
<button>Submit</button>
</form>
<br>
<br>
{{id}} {{name}}
```

```
var express = require('express');
var router = express.Router();

router.get("/",function(req,res,next){
    res.render('index', {title:'Get Data'});
});
router.post('/test/submit', function(req, res, next) {
  res.send(req.body.id+" "+req.body.name);
  //res.render('index', {title:'data', id:req.body.id,name:req.body.name });
});
module.exports = router;
```



Get Data

ID: 1111
Name: Infoway



"NetBeans Connector" is debugging this browser

```
1111 Infoway
```

Main inconveniencies of native MongoDB Driver

- No data validation
- No casting during inserts
- No encapsulation
- No references (joins)

Mongoose- ODM for Node.js

- Although MongoDB won't impose an structure, applications usually manage data with one. We receive data and need to validate it to ensure what we received is what we need. We may also need to process the data in some way before saving it. This is where Mongoose kicks in.

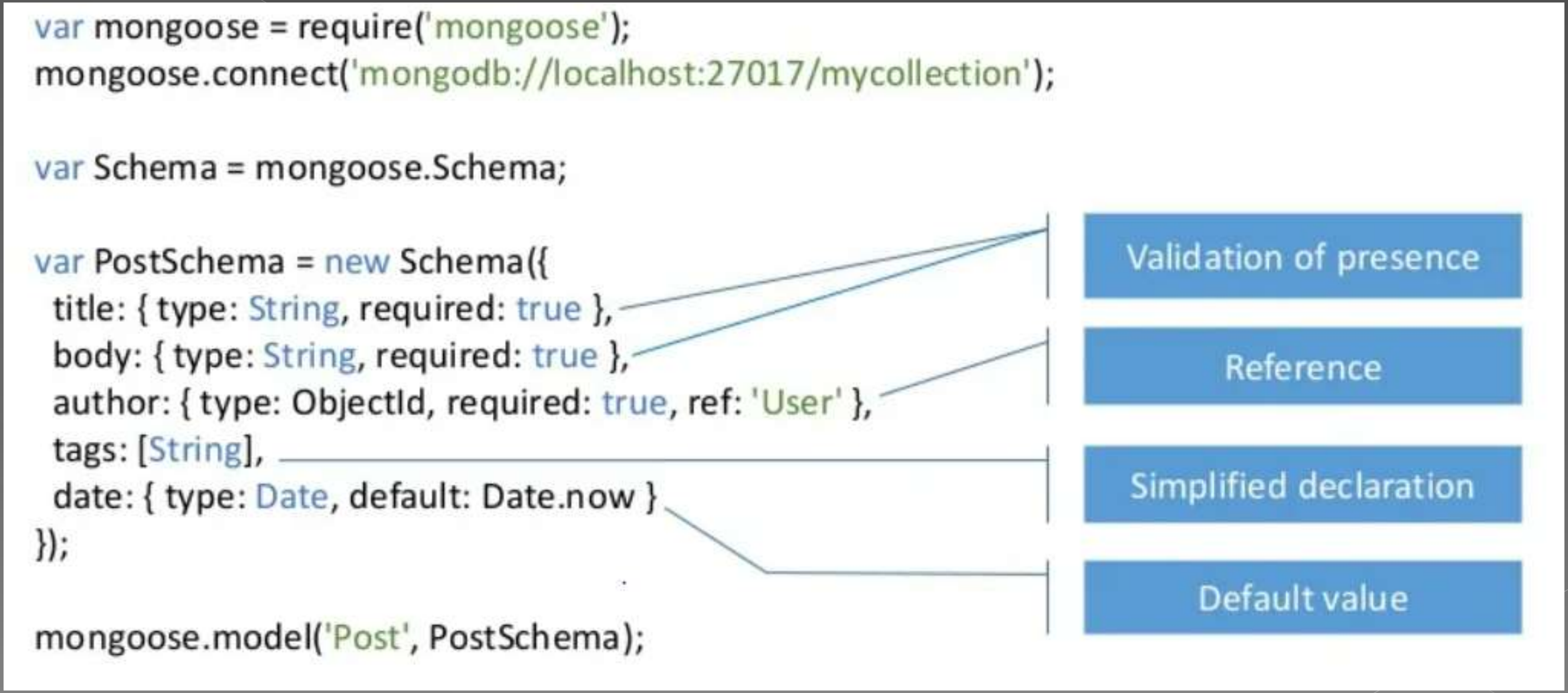
- Mongoose is an NPM package for NodeJS applications. It allows to define schemas for our data to fit into, while also abstracting the access to MongoDB. This way we can ensure all saved documents share a structure and contain required properties.

Connect mongoose with express server

Install mongoose -> npm install mongoose --save

Note:- mongoDB server should be started.

Mongoose setup



```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mycollection');

var Schema = mongoose.Schema;

var PostSchema = new Schema({
  title: { type: String, required: true },
  body: { type: String, required: true },
  author: { type: ObjectId, required: true, ref: 'User' },
  tags: [String],
  date: { type: Date, default: Date.now }
});

mongoose.model('Post', PostSchema);
```

Schema types

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array
- Map
- Schema

required: boolean or function, if true adds a required validator for this property

default: Any or function, sets a default value for the path. If the value is a function, the return value of the function is used as the default.

select: boolean, specifies default projections for queries

validate: function, adds a validator function for this property

get: function, defines a custom getter for this property

set: function, defines a custom setter for this property.

Third Party Middleware

Middleware Module	Description
body-parser	Parse HTTP Request
cookie-parser	Parse cookie header and populate request cookies.
cors	Enable cross-origin resource sharing (CORS) with various options.
errorhandler	Development error-handling /debugging
morgan	HTTP Request logger
multer	Handle multi-part form data
serve-static	Serve static files
session	Establish server based sessions
timeout	Set a timeout period for HTTP request processing

END

infoway TECHNOLOGIES

Thank you !!