

EXPRESS JS INTERVIEW QUESTIONS FOR BEGINNERS TO EXPERIENCED

1) What Is Express Js?

Express JS is an application framework that is light-weighted node JS. Variety of versatile, helpful and vital options are provided by this JavaScript framework for the event of mobile additionally as internet applications with the assistance of node JS.



2) What Type Of Web Application Can Built Using Express Js?

you can build single-page, multi-page, and hybrid web applications.

3) What Are Core Features Of Express Framework?

- **Middleware** : Set up middleware in order to respond to HTTP/RESTful Requests.
- **Routing** : It is possible to defines a routing table in order to perform different HTTP operations.
- **Templates** : Dynamically renders HTML Pages based on passing arguments to templates.
- **High Performance** : Express prepare a thin layer, therefore, the performance is adequate.

- **Database Support** : Express supports RDBMS as well as NoSQL databases.
- **MVC Support** : Organize the web application into an MVC architecture.
- Manages everything from routes to rendering view and performing HTTP request.

4) Why I Should Use Express Js?

Express 3.x is a light-weight web application framework to help organize your web application into an MVC architecture on the server side.

5) How To Install Express.js?

Assuming you've already installed [Node.js](#), create a directory to hold your application, and make that your working directory.

```
$ mkdir myapp  
$ cd myapp
```

Use the `npm init` command to create a `package.json` file for your application. For more information on how `package.json` works, see [Specifics of npm's package.json handling](#).

```
$ npm init
```

This command prompts you for a number of things, such as the name and version of your application. For now, you can simply hit RETURN to accept the defaults for most of them, with the following exception:

```
entry point: (index.js)
```

Enter `app.js`, or whatever you want the name of the main file to be. If you want it to be `index.js`, hit RETURN to accept the suggested default file name.

Now install Express in the `myapp` directory and save it in the dependencies list. For example:

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

To install Express temporarily and not add it to the dependencies list:

```
$ npm install express --no-save
```

6) How to setup an Express.js App?

We can follow the below step by step approach to set up an Application using ExpressJS Framework.

- Create a folder with the same name as Project name.

- Inside the folder create a file called **package.json**.

```
2  "name": "npm_smart_grocery",
3  "version": "1.0.0",
4  "description": "a sample smart grocery manager ",
5  "main": "index.js",
6  "": {
7    "ajv": "^4.9.0",
8    "async": "^1.4.2",
9    "body-parser": "^1.13.3",
10   "cloudant": "^1.4.0",
11   "dotenv": "^2.0.0",
12   "express": "^4.13.3",
13   "express-session": "^1.11.3",
14   "memory-cache": "^0.1.4",
15   "moment": "2.10.6",
16   "passport": "^0.3.2",
17   "path-exists": "^3.0.0",
18   "r-script": "0.0.3",
19   "rio": "^2.4.1",
20   "rox": "0.0.1-13",
21   "superagent": "^1.3.0",
22   "twitter": "^1.4.0",
23   "underscore": "^1.8.3",
24   "v8": "^0.1.0",
25   "winston": "^2.1.1",
26   "winston-daily-rotate-file": "^1.0.1"
27 },
28 "devDependencies": {},
29 "scripts": {
```

- Open command prompt on the project folder and run following command.

```
1 npm install
```

This will install all the libraries defined in **package.json** inside **dependencies{}** and the libraries are installed in **node_modules** folder.

- Create a file called **server.js**.

```
var express = require('express');
var bodyParser = require('body-parser');
var session = require('express-session')
```

```
//var Project = require('./schema').Project;
var http = require('http');
var https = require('https');
```

```
/* Initialize express app */
var app = express();
```

```
var host = process.env.APP_HOST || 'localhost';
var port = process.env.APP_PORT || '3000';
app.use(bodyParser.json());
app.use(session({
  rolling: true,
  saveUninitialized: true,
  resave: true,
  secret: config.SESSION_SECRET,
  cookie: {
    maxAge: 36000000,
    httpOnly: false
  }
}));
```

```
app.use(express.static(__dirname + '/app'));
var index = require('./routes/index');
app.use('/', index);
```

```
/* Start server */
app.listen(app.get('port'), function() {
  logger.info('Express server listening on http://' + app.get('host') + ':' + app.get('port'));
});
```

- Create a folder called 'routes' inside the project folder.
- Create a file inside 'routes' folder called **index.js**.

```
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;
```

- Create a folder called 'app' inside the project folder and create a file inside 'app' folder called 'index.html'.



- Create a index.html file.

```
1 <h1> Hello World!</h1>
```

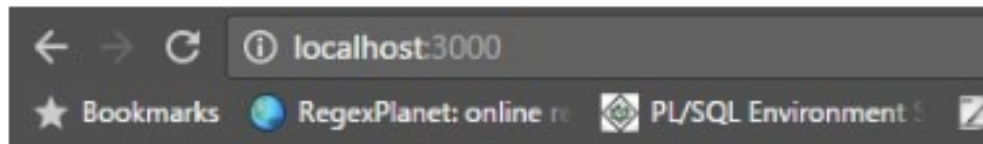
- Open command prompt on the project folder and run following command.

```
1 node server.js
```

The output would be

```
1 2017-02-01T21:31:15.889Z - info: Express server listening on http://localhost:3000
```

- Now open a browser with link <http://localhost:3000>, the output would be:



Hello World!

7) How to create an Http server using Express?

Express is best for developing web application using Node.js. You can create an http server using express as given below:

```
var express = require("express");
var app = express();
app.get( "/", function (req, res) {
  res.write("Hello, Express");
  res.end();
});
var port = process.env.port || 1305;
app.listen(port);
console.log("Server is running at http://localhost:" + port);
```

8) How To Get Variables In Express.js In Get Method?

Answer :

```
var express = require('express');
var app = express();
app.get('/', function(req, res){
  /* req have all the values **/
  res.send('id: ' + req.query.id);
});
app.listen(3000);
```

9) How To Get Post A Query In Express.js?

Answer : `var bodyParser = require('body-parser')`
`app.use(bodyParser.json()); // to support JSON-encoded`
`app.use(bodyParser.urlencoded({ // to support URL-encoded`
 `extended: true`
`}));`

10) How To Output Pretty Html In Express.js?

```
app.set('view options', { pretty: true });
```

11) How To Get The Full Url In Express?

```
var port = req.app.settings.port || cfg.port;
```

```
res.locals.requested_url = req.protocol + '://' + req.host + ( port == 80 || port == 443 ? '' : ':' + port ) + req.path;
```

12) How to enable debugging in express app?

On UNIX operating system the command would be as follows:

- `$ DEBUG=express:* node index.js`

On Windows the command would be:

- `set DEBUG=express:* & node index.js`

From Webstorm IDE

- `C:\Program Files (x86)\JetBrains\WebStorm 2016.2.4\bin\runnerw.exe" "C:\Program Files\nodejs\node.exe" -debug-brk=61081 -expose_debug_as=v8debug E:\Development\nodejd\library\bin\www`

13) How To Do 404 Errors?

```
Answer :  app.get('*', function(req, res){  
    res.send('what???', 404);  
});
```

14) How to implement File uploading and downloading with Express?

Below we have explained the process to upload as well as download a file with ExpressJS App.

Upload File in Express.js:

- Install formidable.

```
1 npm install --save formidable
```

- Add the following code in server.js in order to upload a file.

```
1 var formidable = require('formidable');
2 app.post('/', function (req, res){
3     var form = new formidable.IncomingForm();
4
5     form.parse(req);
6
7     form.on('fileBegin', function (name, file){
8         file.path = __dirname + '/uploads/' + file.name;
9     });
10
11    form.on('file', function (name, file){
12        console.log('Uploaded Successfully! ' + file.name);
13    });
14    res.sendFile(__dirname + '/index.html');
15 });
```

- Update the index.html as following:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Simple File Upload Example</title>
5 </head>
6 <body>
7 <form action="/" enctype="multipart/form-data" method="post">
8     <input type="file" name="upload" multiple>
9     <input type="submit" value="Upload">
10 </form>
11 </body>
12 </html>
```

- On Browser run '<http://localhost:3000>'.

Download File in Express.js:

- Add the following code in server.js.

```
1 var router = express.Router();  
2 // ...  
3 router.get('/:id/download', function (req, res, next) {  
4   var filePath = "/my/file/path/...";  
5   var fileName = "samplefile.pdf";  
6   res.download(filePath, fileName);  
7 });
```

15) What Is The Parameter “next” Used For In Express?

It passes management to a consecutive matching route. OR a operate to pass management to 1 of the following route handlers. The argument could also be omitted, however, is beneficial in cases wherever you have got a series of handlers and you'd wish to pass management to 1 of the following route handlers, and skip this one.

```
app.get('/user details/:id?', function(req, res, next));
```

Req and Res – It represents the request and response objects

Next – It passes management to a consecutive matching route.

16) What Function Arguments Are Available To Express.js Route Handlers?

The arguments available to an Express.js route handler function are:

- **req** - the request object.
- **res** - the response object.
- **next (optional)** - a function to pass control to one of the subsequent route handlers.

The third argument may be omitted, but is useful in cases where you have a chain of handlers and you would like to pass control to one of the subsequent route handlers, and

skip the current one.

17) How To Config Properties In Express Application?

In an ExpressJS Application, we can config properties in following two ways:

1. With Process.ENV:

- Create a file with name '.env' inside the project folder.
- Add all the properties in '.env' file.

```
cloudant_username=sampleusername  
cloudant_password=#####  
cloudnt_databasename=smart_gocery  
env.APP_HOST=localhost  
env.APP_PORT=9080  
recipe_design_doc=recipes  
recipe_index_name=searchall
```

- In server.js any of the properties can be used as:

```
1 var host = process.env.APP_HOST  
2 app.set('host', host);  
3 logger.info('Express server listening on http://' + app.get('host'));
```

2. With RequireJs:

- Create a file called 'config.json' inside a folder called 'config' inside the project folder.
- Add config properties in config.json.

```
1 {  
2   "env": "development",  
3   "apiurl": "http://localhost:9080/api/v1/"  
4 }
```

- Use require to access the config.json file.

```
1 var config = require('./config/config.json');
```

18) How To Allow Cors In Expressjs? Explain With An Example?

In order to allow CORS in Express.js, add the following code in server.js:

```
1 app.all('*', function(req, res, next) {  
2   res.set('Access-Control-Allow-Origin', '*');  
3   res.set('Access-Control-Allow-Methods', 'GET, POST, DELETE, PUT');  
4   res.set('Access-Control-Allow-Headers', 'X-Requested-With, Content-Type');  
5   if ('OPTIONS' == req.method) return res.send(200);  
6   next();  
7 });
```

19) How To Redirect 404 Errors To A Page In Expressjs?

In server.js add the following code to redirect 404 errors back to a page in our ExpressJS App:

```
1 /* Define fallback route */  
2 app.use(function(req, res, next) {  
3   res.status(404).json({errorCode: 404, errorMsg: "route not found"});  
4 });
```

20) Explain Error Handling In Express.js Using An Example?

From Express 4.0 Error handling is much easier. Create an express.js application and as there is no built-in middleware like errorhandler in express 4.0, therefore, the middleware need to be either installed or need to create a custom one. The steps are as following:

1. Create a Middleware:

- Create a middleware as following:

```
1 // error handler
2 app.use(function(err, req, res, next) {
3   // set locals, only providing error in development
4   res.locals.message = err.message;
5   res.locals.error = req.app.get('env') === 'development' ? err : {};
6
7   // render the error page
8   res.status(err.status || 500);
9   res.render('error');
10 });
```

2. Install Error Handler Middleware:

- Install errorhandler.

```
1 npm install errorhandler --save
```

- Create a variable.

```
1 var errorHandler = require('errorhandler')
```

- Use the middleware as following:

```
1 if (process.env.NODE_ENV === 'development') {
2   // only use in development
3   app.use(errorHandler({log: errorNotification}))
4 }
5
6 function errorNotification(err, str, req) {
7   var title = 'Error in ' + req.method + ' ' + req.url
8
9   notifier.notify({
10     title: title,
11     message: str
12   })
13 }
```

21) How To Implement Jwt Authentication In Express App ? Explain With Example?

- Create a folder called 'keys' inside project folder and install some dependencies as following.

```
1 Npm install jsonwebtoken -save
```

- Add the login router routes/index.js.

```
router.post('/login, function(req, res) {
  // find the user
  User.findOne({
    name: req.body.username
  }, function(err, res) {
    if (err) throw err;
    if (!res) {
      res.json({ success: false, message: Login failed.' });
    } else if (res) {

      // check if password matches
      if (res.password !== req.body.password) {
        res.json({ success: false, message: Login failed. Wrong password.' });
      } else {
        var token = jwt.sign(res, app.get('superSecret'), {
          expiresInMinutes: 1600
        });
        // return the information including token as JSON
        res.json({
          success: true,
          message: 'Valid token!',
          token: token
        });
      }
    }
  });
});
```

- Use the token in application.

```
jwt = require("express-jwt");
app.use(function(req, res, next) {
  var token = req.body.token || req.query.token || req.headers['x-access-token'];
  if (token) {
    jwt.verify(token, app.get('superSecret'), function(err, decoded) {
      if (err) {
        return res.json({ success: false, message: 'Invalid token.' });
      } else {
        req.decoded = decoded;
        next();
      }
    });
  } else {
    return res.status(403).send({
      success: false,
      message: 'No token given.'
    });
  }
});
```

22) How Should I Structure My Application?

There is no definitive answer to this question. The answer depends on the scale of your application and the team that is involved. To be as flexible as possible, Express makes no assumptions in terms of structure. Routes and other application-specific logic can live in as many files as you wish, in any directory structure you prefer. View the following examples for inspiration:

- Route listings.
- Route map.
- MVC style controllers.

Also, there are third-party extensions for Express, which simplify some of these patterns:

- Resourceful routing.

23) How Do I Define Models?

Express has no notion of a database. This concept is left up to third-party Node modules, allowing you to interface with nearly any database.

24) How Can I Authenticate Users?

Authentication is another opinionated area that Express does not venture into. You may use any authentication scheme you wish.

25) What is template engine?

A template engine allows us to create and render an HTML template with minimal code. At runtime, a template engine executes expressions and replaces variables with their actual values in a template file. In this way, it transforms the template into an HTML file and sent to it the client.

26) What template engines you can use with express?

The popular template engines which you can use with Express are Pug, Handlebars, Mustache, and EJS. The Express application generator uses Pug as its default template engine. creating an example with Node.js and the Express.js and will be using EJS view engine to manage our HTML code.

- Install Express generator package globally (g stand for global in command):

```
npm install express-generator -g
```

- Open a terminal and move to your directory where you want your code to reside and type. This command will create skeleton of our application and will set ejs as our default view engine:

```
express --view=ejs mywebsite
```

Next step is to install all dependencies listed in mywebsite/package.json file. Move to your myapp directory and type:

```
npm install
```

- Open mywebsite/app.js file, you will see:


```
var express = require('express');
var path = require('path');
var logger = require('morgan');
var index = require('./routes/index');
var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

// set path for static assets
app.use(express.static(path.join(__dirname, 'public')));

// routes
app.use('/', index);
```

- So this is place where we will be defining all our website routes i.e. routes/index file. Open your routes/index.js file and replace your route entry and we are passing page and menuId variable to index view file:

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

New
/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', {page:'Home', menuId:'home'});
});
```

- Now open your views/index.ejs file and replace your code with following:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <% include partials/head %>
</head>
<body>
  <% include partials/menu %>

  <div class="container-fluid bg-3 text-center">
    <h3><%= page %></h3><br>
    <div class="row">
      <div class="col-sm-4">
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
</p>

      </div>
```

```
      <div class="col-sm-4">
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
</p>
      </div>
      <div class="col-sm-4">
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
</p>
      </div>
    </div>
  </div>

</body>
<% include partials/script %>
</html>
```

- Create partials directory in mywebsite/views directory and create following files in partials directory:

head.ejs

```

<title>Static Website | <%= page %></title>
<meta charset="utf-8">
<meta name="active-menu" content="<%= menuId %>">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.
min.css">
<link rel="stylesheet" href="./stylesheets/style.css">

```

menu.ejs

```

<nav class="navbar navbar-default">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target="#myNavbar">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">LOGO</a>
    </div>
    <div class="collapse navbar-collapse" id="myNavbar">
      <ul class="nav navbar-nav navbar-right">
        <li id="home"><a href="/">HOME</a></li>
        <li id="about"><a href="/about">ABOUT US</a></li>
        <li id="contact"><a href="/contact">CONTACT US</a></li>
      </ul>
    </div>
  </div>
</nav>

```

script.ejs

```

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.j
s"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.mi
n.js"></script>
<script src="./javascripts/menu.js"></script>

```

- If you have noticed we have separated our header, topbar menu and master scripts(javascript which need to be load in all pages) code. And all the partials files are included in views/index.ejs file.

```
<% include partials/head %>
<% include partials/menu %>
<% include partials/script %>
```

- Create javascripts and stylesheets directory inside mywebsite/public directory and create new file menu.js inside javascript directory and style.css inside stylesheets directory, I have created these 2 files just to show how we can load external CSS and js file in our app.

menu.js

```
$(document).ready(function(){
    var element = $('meta[name="active-menu"]').attr('content');
    $('#'+ element).addClass('active');
});
```

style.css

```
.bg-3 {
  background-color: #ffffff;
  color: #555555;
}
.container-fluid {
  padding-top: 70px;
  padding-bottom: 70px;
}
.navbar {
  padding-top: 15px;
  padding-bottom: 15px;
  border: 0;
  border-radius: 0;
  margin-bottom: 0;
  font-size: 12px;
  letter-spacing: 5px;
}
.navbar-nav li a:hover {
  color: #1abc9c !important;
}

.active>a {
  color: #1abc9c !important;;
}
```

- Now we are all set to run our node server, type command:

```
nodemon start
```

- Now tune to URL <http://localhost:3000>, our updated home page will be loaded similarly, you can create multiple static views like we did for index page:



27) Which Template Engines Does Express Support?

Express supports any template engine that conforms with the (path, locals, callback) signature.

28) How Do I Render Plain Html?

There's no need to "render" HTML with the `res.render()` function. If you have a specific file, use the `res.sendFile()` function. If you are serving many assets from a directory, use the `express.static()` middleware function.

29) What is Scaffolding in Express JS?

Scaffolding is creating the skeleton structure of application. There are 2 way to do this:

- Express application generator
- Yeoman

1. Express application generator:

Use express-generator to quickly create an application skeleton.

```
code source
1. npm install express-generator -g
2. express myTestExpressApp
```

The above command will create your project named - myTestExpressApp with following the files/folders in project.

1. **bin:** The **bin** folder has one file called **www** is the main configuration file of our app.
2. **public:** Public folder contains JavaScript, CSS, images etc.
3. **routes:** Routes folder contains routing files.
4. **views:** View folder contains the view files of the application.
5. **app.js:** The app.js file is the main file of the application.
6. **package.json:** package.json file is the manifest file. It contains all metadata of the project such as the packages used in the app (called dependencies) etc.

Install all the dependencies mentioned in the package.json file:

```
code source
1. cd myTestExpressApp
2. npm install
```

30) What is routing and how routing works in Express.js?

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 Syntax:

```
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.

Example:

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

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

app.post('/hello', function(req, res){
  res.send("You just called the post method at '/hello'!\n");
});

app.listen(3000);
```

31) What is router in Express.js?

- Defining routes like above is very tedious to maintain. To separate the routes from our main **index.js** file, we will use **Express.Router**. Create a new file called **things.js** and type the following in it.

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

router.get('/', function(req, res){
  res.send('GET route on things.');
```

```
});
router.post('/', function(req, res){
  res.send('POST route on things.');
```

```
});

//export this router to use in our index.js
module.exports = router;
```

- Now to use this router in our **index.js**, type in the following before

the **app.listen** function call.

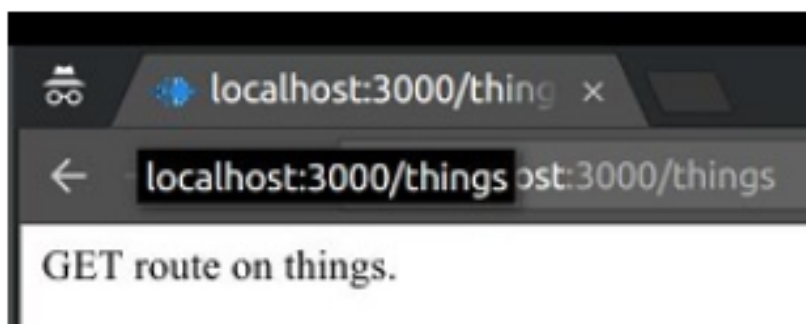
```
var express = require('Express');
var app = express();

var things = require('./things.js');

//both index.js and things.js should be in same directory
app.use('/things', things);

app.listen(3000);
```

- The **app.use** function call on route **'/things'** attaches the **things** router with this route. Now whatever requests our app gets at the **'/things'**, will be handled by our things.js router. The **'/'** route in things.js is actually a subroute of **'/things'**. Visit **localhost:3000/things/** and you will see the following output.



- Routers are very helpful in separating concerns and keep relevant portions of our code together. They help in building maintainable code. You should define your routes relating to an entity in a single file and include it using the above method in your **index.js** file.

32) Dynamic routing and how it works in Express.js?

When someone pass parameters in URL i.e. Parametrized URL, this routing phenomenon is called dynamic routing. In example: id is a parameters, which can be different for different calls.

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

app.get('/article/:id', function(req, res){
    res.render('article' + req.params.id);
})
```

33) Serving static files in Express?

```
-----
#Example:
app.use(express.static('public'))
app.use('/static', express.static(path.join(__dirname, 'public')))
```

34) What is middleware in Express Js?

A function that is invoked by the Express routing layer before the final request handler.

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 function in the stack.

If the current middleware function does not end the request-response cycle, it must call `next()` to pass control to the next middleware function. Otherwise, the request will be left hanging. An Express application can use the following types of middleware:

1. Application-level middleware : This kind of middleware method is bind to the app Object using `app.use()` method. Now whenever you request any subroute of `'/things'`, only then it will log the time.

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

//Middleware function to log request protocol
app.use('/things', function(req, res, next){
  console.log("A request for things received at " + Date.now());
  next();
});

// Route handler that sends the response
app.get('/things', function(req, res){
  res.send('Things');
});

app.listen(3000);
```

2. Router-level middleware : Router-level middleware works in the same way as application-level middleware, except it is bound to an instance of `express.Router()`.

```
var router = express.Router()
```

3. Error-handling middleware : Error-handling middleware always takes **four** arguments. You must provide four arguments to identify it as an error-handling middleware function. Even if you don't need to use the next object, you must specify it to maintain the signature. Otherwise, the next object will be interpreted as regular middleware and will fail to handle errors.

```
app.use(function (err, req, res, next) {
  console.error(err.stack)
  res.status(500).send('Something broke!')
})
```

4. Built-in middleware : Starting with version 4.x, Express no longer depends on **Connect**. Express has the following built-in middleware functions:

- **express.static** serves static assets such as HTML files, images, and so on.

- **express.json** parses incoming requests with JSON payloads. **NOTE: Available with Express 4.16.0+**
- **express.urlencoded** parses incoming requests with URL-encoded payloads. **NOTE: Available with Express 4.16.0+**

5. Third-party middleware : A list of Third party middleware for Express is available here. Following are some of the most commonly used middleware; we will also learn how to use/mount these.

- body-parser is used to parse the body of requests which have payloads attached to them.

```
$ npm install body-parser
```

```
var express = require('express')
var bodyParser = require('body-parser')

var app = express()

// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }))

// parse application/json
app.use(bodyParser.json())
```

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

```
$ npm install cookie-parser
```

```
var express = require('express')
var cookieParser = require('cookie-parser')

var app = express()
app.use(cookieParser())
```

- express-session creates a session middleware with the given options. We will discuss its usage in the Sessions section.

```
$ npm install express-session
```

```
var session = require('express-session')
```

```
var app = express()
app.set('trust proxy', 1) // trust first proxy
app.use(session({
  secret: 'keyboard cat',
  resave: false,
  saveUninitialized: true,
  cookie: { secure: true }
}))
```

35) Database integration in Express Js?

Adding the capability to connect databases to Express apps is just a matter of loading an appropriate Node.js driver for the database in your app. Express Js supports many RDBMS & NoSQL database like:

MySQL

- Installation:

```
$ npm install mysql
```

- Example:

```
var mysql      = require('mysql');
var connection = mysql.createConnection({
  host        : 'localhost',
  user        : 'me',
  password    : 'secret',
  database    : 'my_db'
});

connection.connect();

connection.query('SELECT 1 + 1 AS solution', function (error, results, fields) {
  if (error) throw error;
  console.log('The solution is: ', results[0].solution);
});

connection.end();
```

MongoDB

- Installation:

```
npm install mongodb --save
```

- Example:

Insert a Document

Add to **app.js** the following function which uses the **insertMany** method collection.

```
const insertDocuments = function(db, callback) {  
  // Get the documents collection  
  const collection = db.collection('documents');  
  // Insert some documents  
  collection.insertMany([  
    {a : 1}, {a : 2}, {a : 3}  
  ], function(err, result) {  
    assert.equal(err, null);  
    assert.equal(3, result.result.n);  
    assert.equal(3, result.ops.length);  
    console.log("Inserted 3 documents into the collection");  
    callback(result);  
  });  
}
```

Update a document

The following operation updates a document in the **documents** collection.

```
const updateDocument = function(db, callback) {  
  // Get the documents collection  
  const collection = db.collection('documents');  
  // Update document where a is 2, set b equal to 1  
  collection.updateOne({ a : 2 }  
    , { $set: { b : 1 } }, function(err, result) {  
    assert.equal(err, null);  
    assert.equal(1, result.result.n);  
    console.log("Updated the document with the field a equal to 2");  
    callback(result);  
  });  
}
```

Remove a document

Remove the document where the field **a** is equal to **3**.

```
const removeDocument = function(db, callback) {  
  // Get the documents collection  
  const collection = db.collection('documents');  
  // Delete document where a is 3  
  collection.deleteOne({ a : 3 }, function(err, result) {  
    assert.equal(err, null);  
    assert.equal(1, result.result.n);  
    console.log("Removed the document with the field a equal to 3");  
    callback(result);  
  });  
}
```

Find All Documents

Add a query that returns all the documents.

```
const findDocuments = function(db, callback) {  
  // Get the documents collection  
  const collection = db.collection('documents');  
  // Find some documents  
  collection.find({}).toArray(function(err, docs) {  
    assert.equal(err, null);  
    console.log("Found the following records");  
    console.log(docs)  
    callback(docs);  
  });  
}
```


Add the new method to the **MongoClient.connect** callback function.

```
const MongoClient = require('mongodb').MongoClient;
const assert = require('assert');

// Connection URL
const url = 'mongodb://localhost:27017';

// Database Name
const dbName = 'myproject';

// Use connect method to connect to the server
MongoClient.connect(url, function(err, client) {
  assert.equal(null, err);
  console.log("Connected successfully to server");

  const db = client.db(dbName);

  insertDocuments(db, function() {
    updateDocument(db, function() {
      removeDocument(db, function() {
        client.close();
      });
    });
  });
});
```

36) How to setting up connection with mongoose in Express.js?

- Now that you have installed Mongo, let us install Mongoose, the same way we have been installing our other node packages:

```
npm install --save mongoose
```

- Before we start using mongoose, we have to create a database using the Mongo shell. To create a new database, open your terminal and enter "mongo". A Mongo shell will start, enter the the code given below. A new database will be created for you. Whenever you open up the mongo shell, it will default to "test" db and you will have

to change to your database using the same command as given below:

```
use my_db
```

- To use Mongoose, we will require it in our **index.js** file and then connect to the **mongodb** service running on **mongodb://localhost**:

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

- Now our app is connected to our database, let us create a new Model. This model will act as a collection in our database. To create a new Model, use the code given below before defining any route. The code defines the schema for a person and is used to create a Mongoose Model **Person**.

```
var personSchema = mongoose.Schema({  
  name: String,  
  age: Number,  
  nationality: String  
});  
var Person = mongoose.model("Person", personSchema);
```

37) Explain Saving Documents in mongodb using mongoose in Express.js?

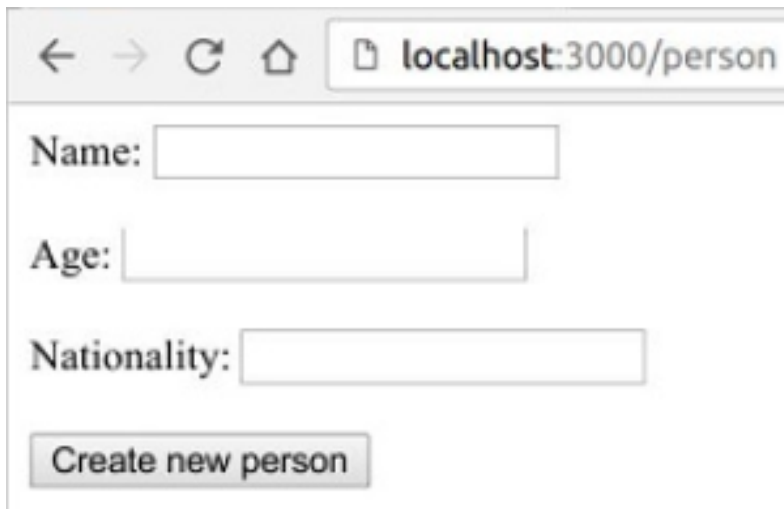
- Now, we will create a new html form; this form will help you get the details of a person and save it to our database. To create the form, create a new view file called **person.pug** in views directory with the following content:

```
html
head
  title Person
  body
    form(action = "/person", method = "POST")
    div
      label(for = "name") Name:
      input(name = "name")
    br
    div
      label(for = "age") Age:
      input(name = "age")
    br
    div
      label(for = "nationality") Nationality:
      input(name = "nationality")
    br
    button(type = "submit") Create new person
```

- Also add a **new get route** in **index.js** to render this document:

```
app.get('/person', function(req, res){
  res.render('person');
});
```

- Go to "**localhost:3000/person**" to check if the form is displaying the correct output. Note that this is just the UI, it is not working yet. The following screenshot shows how the form is displayed:



A screenshot of a web browser window. The address bar shows 'localhost:3000/person'. The page contains a form with three input fields: 'Name:', 'Age:', and 'Nationality:'. Below these fields is a button labeled 'Create new person'.

- We will now define a post route handler at **'/person'** which will handle this request. In the code, if we receive any empty field or do not receive any field, we will send an error response. But if we receive a well-formed document, then we create a **newPerson** document from Person model and save it to our DB using the **newPerson.save()** function. This is defined in Mongoose and accepts a callback as argument. This callback has 2 arguments – error and response. These arguments will render the **show_message** view.

```
app.post('/person', function(req, res){
  var personInfo = req.body; //Get the parsed information

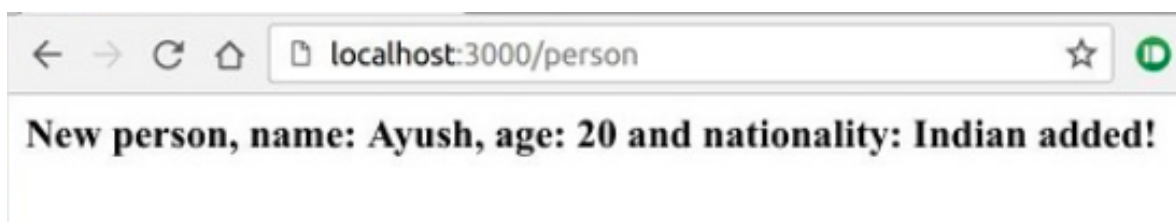
  if(!personInfo.name || !personInfo.age || !personInfo.nationality){
    res.render('show_message', {
      message: "Sorry, you provided wrong info", type: "error"});
  } else {
    var newPerson = new Person({
      name: personInfo.name,
      age: personInfo.age,
      nationality: personInfo.nationality
    });
  }
});
```

```
newPerson.save(function(err, Person){
  if(err)
    res.render('show_message', {message: "Database error", type:
"error"});
  else
    res.render('show_message', {
      message: "New person added", type: "success", person:
personInfo});
  });
});
```

- To show the response from this route, we will also need to create a **show_message** view. Create a new view with the following code:

```
html
  head
    title Person
  body
    if(type == "error")
      h3(style = "color:red") #{message}
    else
      h3 New person,
        name: #{person.name},
        age: #{person.age} and
        nationality: #{person.nationality} added!
```

- We will receive the following response on successfully submitting the **form(show_message.pug)**:



38) Explain Retrieving Documents in mongodb using mongoose in Express.js?

Mongoose provides a lot of functions for retrieving documents, we will focus on 3 of those. All these functions also take a callback as the last parameter, and just like the save function,

their arguments are error and response. The three functions are as follows.

Model.find(conditions, callback):

- This function finds all the documents matching the fields in conditions object. Same operators used in Mongo also work in mongoose. For example

```
Person.find(function(err, response){
  console.log(response);
});
```

- This will fetch all the documents from the person's collection and all documents where field name is "Ayush" and age is 20.

```
Person.find({name: "Ayush", age: 20},
  function(err, response){
    console.log(response);
  });
```

- We can also provide projection we need, i.e., the fields we need. For example, if we want only the **names** of people whose **nationality** is "Indian", we use:

```
Person.find({nationality: "Indian"}, "name", function(err, response){
  console.log(response);
});
```

- Let us now create a route to view all people records:

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

var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/my_db');

var personSchema = mongoose.Schema({
  name: String,
  age: Number,
  nationality: String
});
```

```
var Person = mongoose.model("Person", personSchema);

app.get('/people', function(req, res){
  Person.find(function(err, response){
    res.json(response);
  });
});

app.listen(3000);
```

Model.findOne(conditions, callback):

- This function always fetches a single, most relevant document. It has the same exact arguments as **Model.find()**.

```
// find one iphone adventures - iphone adventures??
Adventure.findOne({ type: 'iphone' }, function (err, adventure) {});

// same as above
Adventure.findOne({ type: 'iphone' }).exec(function (err, adventure) {});
```

Model.findById(id, callback):

- This function takes in the **_id**(defined by mongo) as the first argument, an optional projection string and a callback to handle the response. For example:

```
Person.findById("507f1f77bcf86cd799439011", function(err, response){
  console.log(response);
});
```

39) Explain Updating Documents in mongodb using mongoose in Express.js?

Mongoose provides 3 functions to update documents. The functions are described below:

Model.update(condition, updates, callback)

- This function takes a conditions and updates an object as input and applies the changes to all the documents matching the conditions in the collection. For example, following code will update the nationality "American" in all Person documents:

```
Person.update({age: 25}, {nationality: "American"}, function(err, response){
  console.log(response);
});
```

Model.findOneAndUpdate(condition, updates, callback)

- It finds one document based on the query and updates that according to the second argument. It also takes a callback as last argument. Let us perform the following example to understand the function:

```
Person.findOneAndUpdate({name: "Ayush"}, {age: 40}, function(err, response) {
  console.log(response);
});
```

Model.findByIdAndUpdate(id, updates, callback)

- This function updates a single document identified by its id. For example:

```
Person.findByIdAndUpdate("507f1f77bcf86cd799439011", {name: "James"},
  function(err, response){
    console.log(response);
  });
```

- Let us now create a route to update people. This will be a **PUT** route with the id as a

parameter and details in the payload:

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

var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/my_db');

var personSchema = mongoose.Schema({
  name: String,
  age: Number,
  nationality: String
});
```

```
var Person = mongoose.model("Person", personSchema);

app.put('/people/:id', function(req, res){
  Person.findByIdAndUpdate(req.params.id, req.body, function(err, response){
    if(err) res.json({message: "Error in updating person with id " +
    req.params.id});
    res.json(response);
  });
});

app.listen(3000);
```

- To test this route, enter the following in your terminal (replace the id with an id from your created **people**). This will update the document associated with the id provided in the route with the below details:

```
curl -X PUT --data "name = James&age = 20&nationality = American"
"http://localhost:3000/people/507f1f77bcf86cd799439011"
```

40) Explain Deleting Documents in mongodb using mongoose in Express.js?

We will see how Mongoose can be used to **Delete** documents. We have 3 functions here, exactly like update.

Model.remove(condition, [callback])

- This function takes a condition object as input and removes all documents matching the conditions. For example, if we need to remove all people aged 20, use the following syntax:

```
Person.remove({age:20});
```

Model.findOneAndRemove(condition, [callback])

- This function removes a **single**, most relevant document according to conditions object. Let us execute the following code to understand the same:

```
Person.findOneAndRemove({name: "Ayush"});
```

Model.findByIdAndRemove(id, [callback])

- This function removes a single document identified by its id. For example:

```
Person.findByIdAndRemove("507f1f77bcf86cd799439011");
```

- Let us now create a route to delete people from our database:

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

var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/my_db');

var personSchema = mongoose.Schema({
  name: String,
  age: Number,
  nationality: String
});
```

```
var Person = mongoose.model("Person", personSchema);

app.delete('/people/:id', function(req, res){
  Person.findByIdAndRemove(req.params.id, function(err, response){
    if(err) res.json({message: "Error in deleting record id " +
req.params.id});
    else res.json({message: "Person with id " + req.params.id + " removed."});
  });
});

app.listen(3000);
```

- To check the output, use the following curl command:

```
curl -X DELETE http://localhost:3000/people/507f1f77bcf86cd799439011
```

- This will remove the person with given id producing the following message:

```
{message: "Person with id 507f1f77bcf86cd799439011 removed."}
```

41) How can we upload or send form data in Express.js?

Forms are an integral part of the web. Almost every website we visit offers us forms that submit or fetch some information for us. To get started with forms, we will first install the *body-parser* (for parsing JSON and url-encoded data) and *multer* (for parsing multipart/form data) middleware. This is the most recommended way to send a request. There are many other ways, but those are irrelevant to cover here, because our Express app will handle all those requests in the same way.

- To install the *body-parser* and *multer*, go to your terminal and use:

```
npm install --save body-parser multer
```

- After importing the body parser and multer, we will use the **body-parser** for parsing json and x-www-form-urlencoded header requests, while we will use **multer** for parsing multipart/form-data. Replace your **index.js** file contents with the following code:

```
var express = require('express');
var bodyParser = require('body-parser');
var multer = require('multer');
var upload = multer();
var app = express();

app.get('/', function(req, res){
  res.render('form');
});
```

```
app.set('view engine', 'pug');
app.set('views', './views');

// for parsing application/json
app.use(bodyParser.json());

// for parsing application/xwww-
app.use(bodyParser.urlencoded({ extended: true }));
//form-urlencoded

// for parsing multipart/form-data
app.use(upload.array());
app.use(express.static('public'));
```

```
app.post('/', function(req, res){
  console.log(req.body);
  res.send("recieved your request!");
});
app.listen(3000);
```

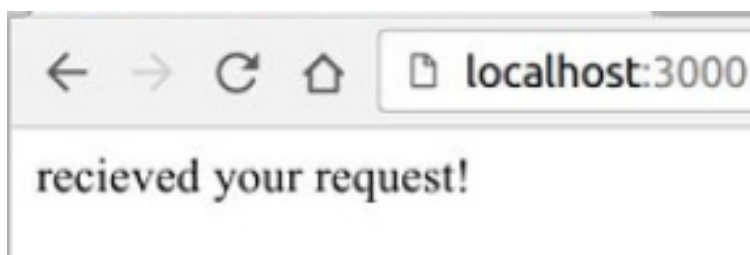
- Let us create an html form to test this out. Create a new view called **form.pug** with the following code:

```
html
  head
    title Form Tester
  body
    form(action = "/", method = "POST")
      div
        label(for = "say") Say:
        input(name = "say" value = "Hi")
      br
      div
        label(for = "to") To:
        input(name = "to" value = "Express forms")
      br
      button(type = "submit") Send my greetings
```

- Run your server using the following:

```
nodemon index.js
```

- Now go to localhost:3000/ and fill the form as you like, and submit it. The following response will be displayed:



- Have a look at your console; it will show you the body of your request as a JavaScript object as in the following screenshot. The **req.body** object contains your parsed request body. To use fields from that object, just use them like normal JS objects.

```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
{ say: 'Hi', to: 'Express forms' }
{ say: 'Hi', to: 'Express forms' }
```

42) What are different methods in REST API?

An API is always needed to create mobile applications, single page applications, use AJAX calls and provide data to clients. RESTful URIs and methods provide us with almost all information we need to process a request. The table given below summarizes how the various verbs should be used and how URIs should be named. We will be creating a movies API towards the end; let us now discuss how it will be structured:

Method	URI	Details	Function
GET	/movies	Safe, cachable	Gets the list of all movies and their details
GET	/movies/1234	Safe, cachable	Gets the details of Movie id 1234
POST	/movies	N/A	Creates a new movie with the details provided. Response contains the URI for this newly created resource.
PUT	/movies/1234	Idempotent	Modifies movie id 1234 <i>createsoneifitdoesn't alreadyexist</i> . Response contains the URI for this newly created resource.
DELETE	/movies/1234	Idempotent	Movie id 1234 should be deleted, if it exists. Response should contain the status of the request.

- Let us now create this API in Express. We will be using JSON as our transport data format as it is easy to work with in JavaScript and has other benefits. Replace your **index.js** file with the **movies.js** file as in the following program:

index.js

```
var express = require('express');
var bodyParser = require('body-parser');
var multer = require('multer');
var upload = multer();

var app = express();

app.use(cookieParser());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(upload.array());
```

```
//Require the Router we defined in movies.js
var movies = require('./movies.js');

//Use the Router on the sub route /movies
app.use('/movies', movies);

app.listen(3000);
```

- Now that we have our application set up, let us concentrate on creating the API. Start by setting up the **movies.js** file. We are not using a database to store the movies but are storing them in memory; so every time the server restarts, the movies added by us will vanish. This can easily be mimicked using a database or a file (using node fs module). Once you import Express then, create a Router and export it using *module.exports*:

```
var express = require('express');
var router = express.Router();
var movies = [
  {id: 101, name: "Fight Club", year: 1999, rating: 8.1},
  {id: 102, name: "Inception", year: 2010, rating: 8.7},
  {id: 103, name: "The Dark Knight", year: 2008, rating: 9},
  {id: 104, name: "12 Angry Men", year: 1957, rating: 8.9}
];
```



```
router.get('/:id([0-9]{3,})', function(req, res){
  var currMovie = movies.filter(function(movie){
    if(movie.id == req.params.id){
      return true;
    }
  });

  if(currMovie.length == 1){
    res.json(currMovie[0])
  } else {
    res.status(404); //Set status to 404 as movie was not found
    res.json({message: "Not Found"});
  }
});
```

```
router.post('/', function(req, res){
  //Check if all fields are provided and are valid:
  if(!req.body.name ||
    !req.body.year.toString().match(/^([0-9]{4})$/g) ||
    !req.body.rating.toString().match(/^([0-9]\.[0-9])$/g)){
    res.status(400);
    res.json({message: "Bad Request"});
  } else {
    var newId = movies[movies.length-1].id+1;
    movies.push({
      id: newId,
      name: req.body.name,
      year: req.body.year,
      rating: req.body.rating
    });
    res.json({message: "New movie created.", location: "/movies/" + newId});
  }
});
```

```
router.put('/:id', function(req, res) {
  //Check if all fields are provided and are valid:
  if(!req.body.name ||
    !req.body.year.toString().match(/^([0-9]{4})$/g) ||
    !req.body.rating.toString().match(/^([0-9]\.[0-9])$/g) ||
    !req.params.id.toString().match(/^([0-9]{3,})$/g)){
    res.status(400);
    res.json({message: "Bad Request"});
  } else {
    //Gets us the index of movie with given id.
    var updateIndex = movies.map(function(movie){
      return movie.id;
    }).indexOf(parseInt(req.params.id));
```



```

    if(updateIndex === -1){
        //Movie not found, create new
        movies.push({
            id: req.params.id,
            name: req.body.name,
            year: req.body.year,
            rating: req.body.rating
        });
        res.json({
            message: "New movie created.", location: "/movies/" +
req.params.id});
    } else {
        //Update existing movie
        movies[updateIndex] = {
            id: req.params.id,
            name: req.body.name,
            year: req.body.year,
            rating: req.body.rating
        };
        res.json({message: "Movie id " + req.params.id + " updated.",
            location: "/movies/" + req.params.id});
    }
}
});

```

```

router.delete('/:id', function(req, res){
    var removeIndex = movies.map(function(movie){
        return movie.id;
    }).indexOf(req.params.id); //Gets us the index of movie with given id.

    if(removeIndex === -1){
        res.json({message: "Not found"});
    } else {
        movies.splice(removeIndex, 1);
        res.send({message: "Movie id " + req.params.id + " removed."});
    }
});
module.exports = router;

```

- This completes our REST API. Now you can create much more complex applications using this simple architectural style and Express.

43) Explain Logging in Express.js? Give a practical example to demonstrate?

With Express 4.0, the application can be generated using express-generator and it includes **morgan** as the logger:

- Create express app using express generator.
- The middleware in app.js is already added.

```
1 var logger = require('morgan');
```

- Create the local middleware.

```
1 var logger = morgan('combined');
```

- Otherwise, If logging is need to be added to a log file.
Add fs to app.js

```
1 var fs = require('fs')
```

Add the file

```
1 var log_file = fs.createWriteStream(path.join(__dirname, log.log'), {flags: 'a'})
```

Create the middleware

```
1 Var logger = morgan('combined', {stream: log_file})
```

- Make sure logging will be enabled only in development environment.

```
1 app.use(logger('dev'));
```

- Now if we run from the browser we can see that every request is being logged.

```
1 GET /dsfsdf 500 387.461 ms - 1144
2 GET /stylesheets/style.css 304 3.383 ms - -
3 GET / 304 40.564 ms - -
4 GET /stylesheets/style.css 304 1.791 ms - -
5 GET /todos 200 1.397 ms - 51
6 GET /todos/new 304 62.912 ms - -
7 GET /stylesheets/style.css 304 0.397 ms - -
```