RESTFul and Express.js

Pramod Bavchikar

2023

Contents

What is REST

- **Re**presentational **S**tate **T**ransfer.
- A Web API (or Web Service) conforming to the REST architectural style is a REST API.
- The following principles encourage RESTful applications to be simple, lightweight, and fast:

    1. Use HTTP methods explicitly.
    2. Be stateless.
    3. Expose directory structure-like URIs.
    4. Transfer XML, JavaScript Object Notation (JSON), or both.

HTTP Methods

- To **create** a resource on the server, use **POST**.
- To retrieve a resource, use **GET**.
- To change the state of a resource or to update it, use **PUT**.
- To remove or delete a resource, use **DELETE**.

Before REST

GET /adduser
POST /createUser

After REST

GET /user
POST /user
PUT /user
DELETE /user

HTTP Status Codes

- Informational responses (100–199)
- Successful responses (200–299)
- Redirection messages (300–399)
- Client error responses (400–499)
- Server error responses (500–599)

Commonly Used Status Codes

200 – Success
201 – Created
301 – Redirect
400 – Bad Request
401 – Unauthorized
403 – Forbidden
404 – Not Found
500 – Internal Server Error
503 – Service Unavailable

Commonly Used Status Codes

Request

POST /user **HTTP 1.1**

Accept: application/json
Authorization: <token>
{
  name: ‘somename’
}

Response

HTTP 1.1 200 OK

Accept: application/json
Server: nginx
Age: 2323
Connection: keep-alive
{
  status: ‘success’,
  name: ‘somename’,
}

Frameworks

- Frameworks are software that are developed and used by developers to build applications.

- Since they are often built, tested, and optimized by several experienced software engineers and programmers, software frameworks are versatile, robust, and efficient.

Commonly Used Framework

Express.js
Koa.js
Nest.js
Molecular.js
Socket.io

Express.js

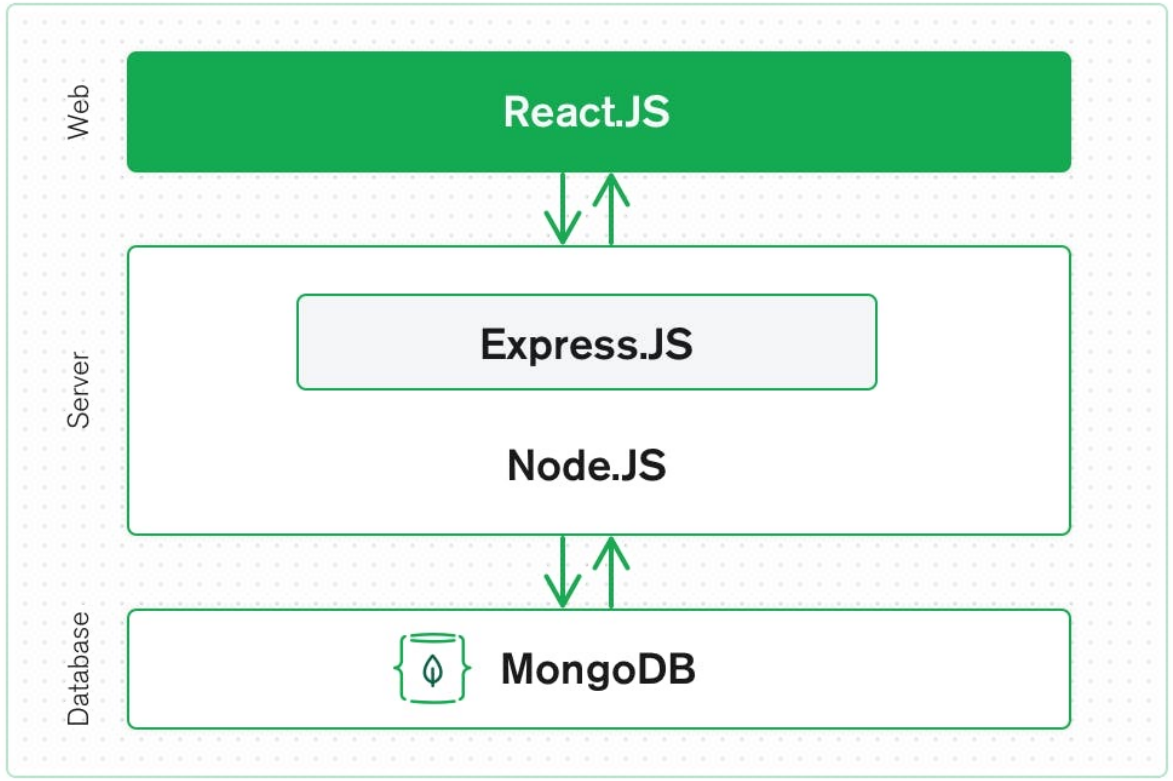
- Fast, unopinionated, minimalist web framework for Node.js

Features of express.js

- Routing
- Middleware
- Static File Serving
- Error Handling
- Very first framework for node.js

Tech Stacks of express.js

- MERN
- MEAN
- PERN
- PEAN



Express.js

- Create a new project and install express

```
npm init -y
npm i express
```

- Create an Express application

```
// app.js
const express = require(“express");
const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.send("Hello World!");
});

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`);
});
```

- Running an Express application

```
$ node app.js
```

Express.js Routing

- Handling http client requests for a particular endpoint

Syntax

app.METHOD(PATH, HANDLER)

- Example route with route parameters included

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
})
```

- Use the express.Router class to create modular, mountable route handlers.

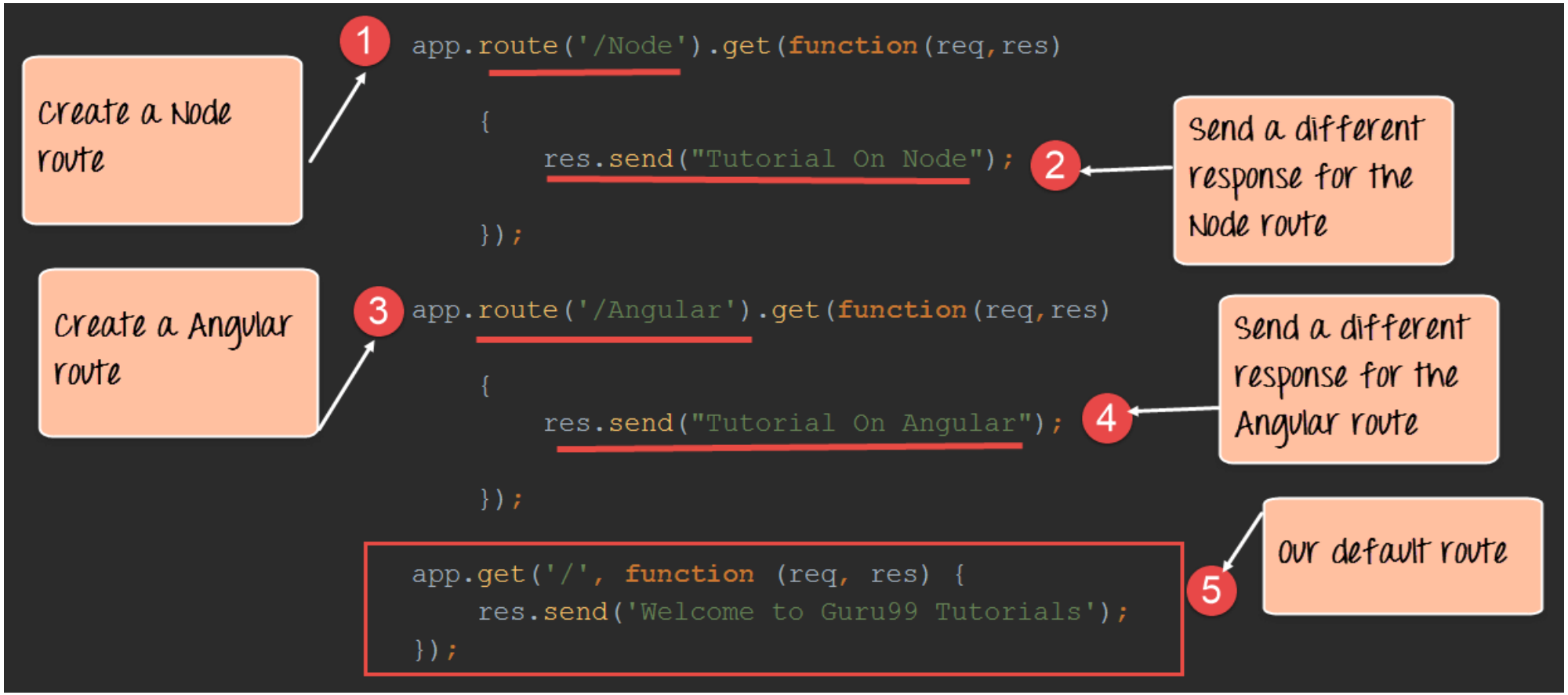
Route methods: - get, post, put, delete, all

Route paths can also be defined using regular expressions

- You can provide multiple callback functions that behave like middleware to handle a request.

```
app.get('/example/b', function (req, res, next) {
  console.log('the response will be sent by the next function ...')
  next()
}, function (req, res) {
  res.send('Hello from B!')
})
```

Express.js Routing



Express.js Routing

Example router module that
- loads a middleware function in it,
- defines some routes,
- and mounts the router module on a path in the main app.

Example router module that
- loads a middleware function in it,
- defines some routes,
- and mounts the router module on a path in the main app.

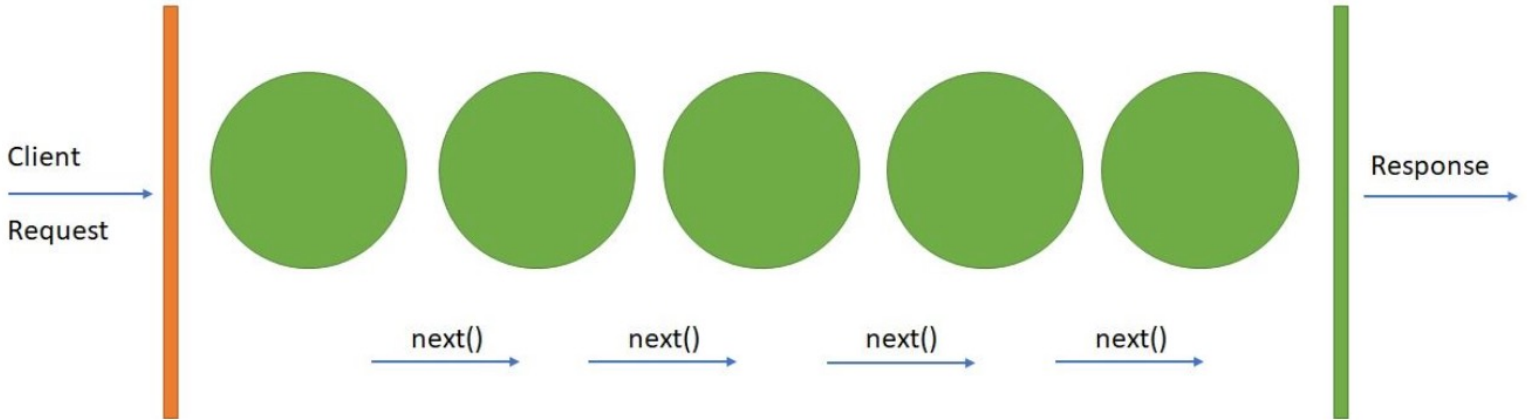
```
// bird.js
var express = require('express')
var router = express.Router()
// middleware that is specific to this router
router.use(function timeLog (req, res, next) {
  console.log('Time: ', Date.now())
  next()
})
// define the about route
router.get('/', function (req, res) {
  res.send('About birds')
})

module.exports = router
```

Middleware

- Middleware is software that provides common services and capabilities to applications outside of what's offered by the operating system.

These are the all commonly handled by middleware.

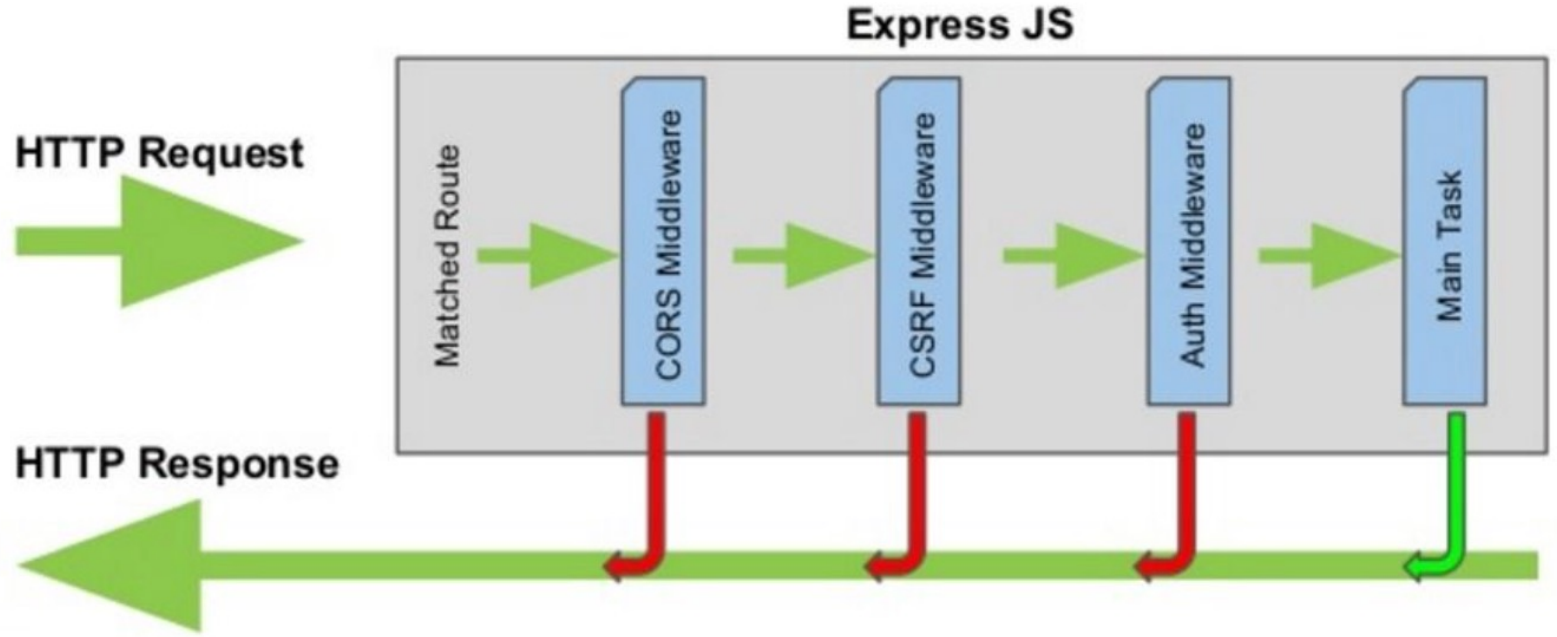
- Data management
- Application services
- Messaging
- Authentication
- and API management



Middleware

- Middleware functions - Functions that have access to
  - the request object (req),
  - the response object (res), and
  - the next function in the application's request-response cycle.
- Next() is a middleware function that calls for the control of another middleware once the code is completed.

- Middleware functions can perform the following tasks:
  - Execute any code - logging request parameters
  - Make changes to the request and the response objects
  - End the request-response cycle.
  - Call the next middleware 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.*



How to use Middleware

- Applying a middleware

```
var myLogger = function (req, res, next) {
  console.log('LOGGED')
  next()
}

app.use(myLogger)

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

Types of Middleware

Application-level middleware

```
app.use('/user/:id', function (req, res, next) {
  console.log('Request Type:', req.method)
  next()
})
```

Router-level middleware

```
router.use('/user/:id', function (req, res, next) {
  console.log('Request URL:', req.originalUrl)
  next()
})
```

Error-handling middleware

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

- Error-handling middleware always takes four arguments. You must provide four arguments to identify it as an error-handling middleware function.
- Apart from it there can be some Built-in middleware or Third-party middleware

Error Handling

- Express comes with a default error handler so you don't need to write your own to get started.
- Synchronous errors are caught by error handler automatically.
- Asynchronous callback errors must be passed to `next()` to make it catchable by default error handler.
- If function does not return any data next() can be passed as callback as shown in code below
- Promise rejections or errors thrown by asynchronous function should also be passed to next(). Express5 handles this category of error also using default handler.

```
router.get('/cb-no-data-2', [
    (req, res, next)=> {
        fs.writeFile('/inaccessible-path', 'data', next)
    },
    (req, res)=> {
        res.send('OK')
    }
]);
```

Error Handling

Custom Error Handler syntax - except error-handling functions have four arguments instead of three: (err, req, res, next)

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

Error-handling middleware is defined at last, after other app.use() and routes calls. There can be multiple handlers

Express API

Express has a small API that includes:

* express() - top-level function exported by the express module that creates Express application
* Built-in middleware function
    - express.json([options])
    - express.static(root, [options])
    - express.Router([options])
    - express.urlencoded([options])
* **Application** - The app object conventionally denotes the Express application
* **Request** - The req object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.
* **Response** - The res object represents the HTTP response that an Express app sends when it gets an HTTP request.
* **Router** - A router object is an isolated instance of middleware and routes. You can think of it as a “mini-application,” capable only of performing middleware and routing functions. Every Express application has a built-in app router.

Express API – Request Object

Properties

req.app
req.originalUrl
req.baseUrl
req.host
req.hostname
req.subdomains
req.cookies
req.body
req.query
req.params
req.path
req.method
req.protocol
req.ip

Methods

req.accepts()
req.acceptsCharsets()
req.acceptsEncodings()
req.AcceptsLanguages()
req.get()
req.is()

Express API – Response Methods

Response methods

The methods on the response object (`res`) in the following table can send a response to the client, and terminate the request-response cycle. If none of these methods are called from a route handler, the client request will be left hanging.

Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view 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.