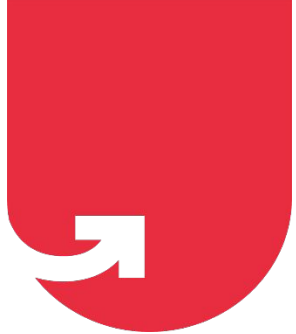
upGrad

#RahoAmbitious



Full Stack Software Development

upGrad

Course: Server-Side Development Using Node.js, Express.js and MongoDB

Lecture on: Building Web Server in Node.js

Instructor: Rocky Jagtiani

In the Previous Class...

- You were introduced to Node.js and understood the reason why it is widely used in building full stack web applications.
- You installed Node.js on your systems and analysed how it works.
- You learnt the difference between modules and packages.
- You also understood the 'Node Package Manager' and its applications.
- You were also oriented with a few terms such as, 'Express.js' and 'MongoDB'.

Poll 1 (15 Sec)

State whether true or false:

Node is a runtime environment for executing JavaScript code

1. True
2. False

Poll 1 (Answer)

State whether true or false:

Node is a runtime environment for executing JavaScript code

1. **True**
2. False

Poll 2 (15 Sec)

To load libraries of code and modules in Node.js, we use which Node.js method?

1. export()
2. import()
3. require()
4. modul()

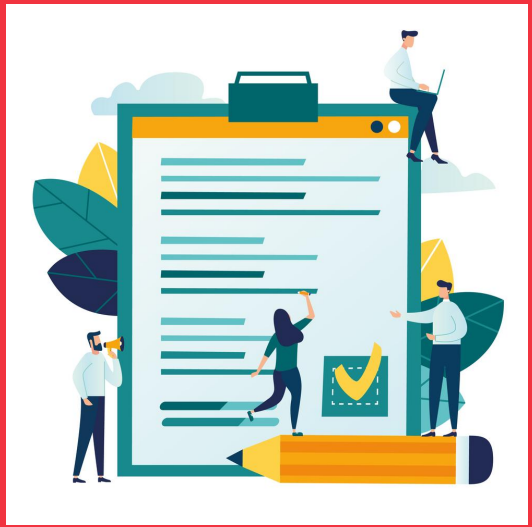
Poll 2 (Answer)

To load libraries of code and modules in Node.js, we use which Node.js method?

1. export()
2. import()
3. **require()**
4. modul()

Today's Agenda

- Web servers
- HTTP methods
- HTTP response codes
- Request-response cycle
- Callbacks in Node.js
- Handling incoming data
- Routes
- Project work



Web Servers

Web Servers

upGrad

- Most Node.js applications are built on web servers
- They allow the loading of images and HTML web pages to the users of your app
- A web server is a software designed to respond to requests over the internet by loading or processing data
- Web servers follow **hypertext transfer protocol (HTTP)**, a standardised system that is globally observed to view web pages and send data over the internet

We can adopt Secure *HTTP* (**HTTPS**), in which transmission of data is encrypted.

HTTP Methods

- The most commonly used HTTP methods are as follows:
  - GET:
    This method is used to extract data from the given URL and it does not modify the data
  - POST:
    This method is used to send user data to the server. It can also be used to create a new entity
  - PUT:
    This method is used to either create a new entity or update an already existing entity
  - DELETE:
    This method is used to remove all the resources provided by the given URL

GET	POST
The GET method is used to fetch data from a specified resource	The POST method is used to send data to a server, such that a resource is created or updated
The parameters passed along with the GET method stay as part of the URL, and hence, they are stored in the browser history	The parameters passed along with the POST method are not a part of the URL, and hence, they are not stored in the browser history
It should not be used in case you are looking for security and want to share sensitive data	It can be used while sharing sensitive data
GET requests have restriction on length	POST requests do not have any length restriction

Poll 3 (15 Sec)

The POST method stores data in the browser history.

Is the above statement true or false?

1. True
2. False

Poll 3 (Answer)

The POST method stores data in the browser history.

Is the above statement true or false?

1. True
2. **False**

HTTP Response Code

HTTP Response Code

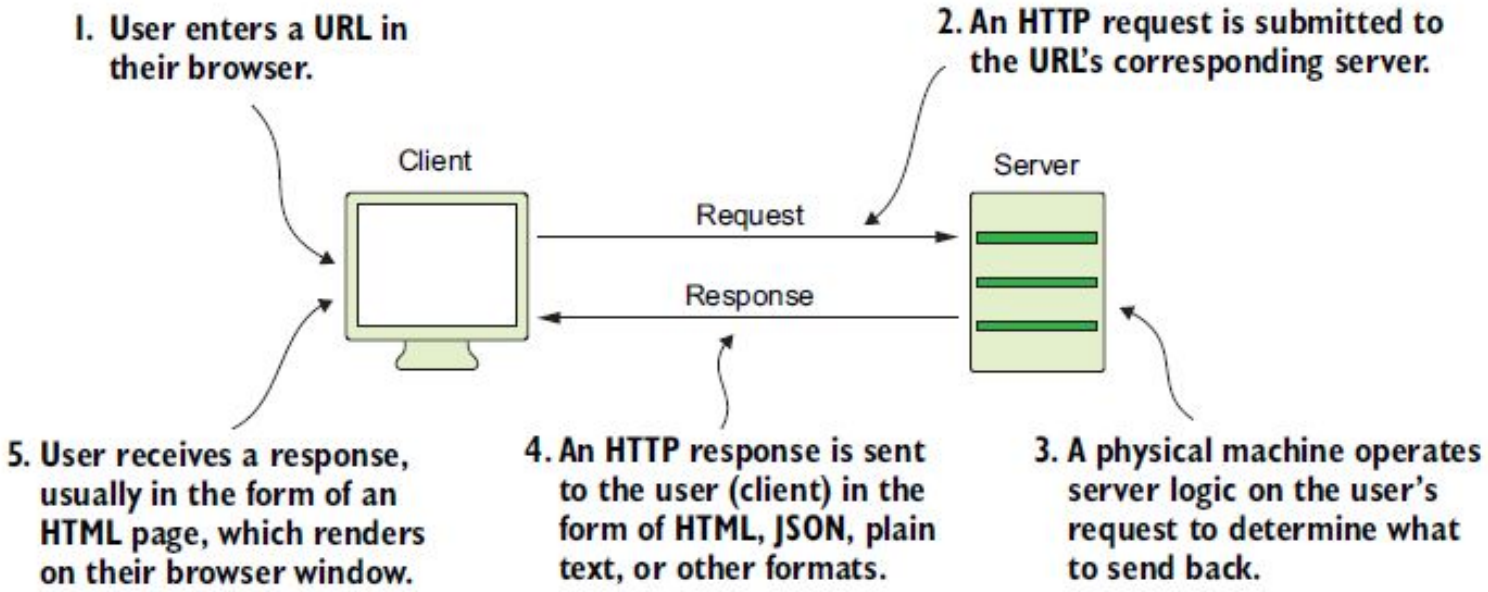
upGrad

Status Code	Description
200 - OK	This status code shows that the request was successfully carried out
301 - Moved Permanently	This status code represents that the data that is being requested is not found at the given address and has been moved to a different address permanently
302 - Moved Temporarily	This status code represents that the data that is being requested is not found at the given address and has been moved to a different address temporarily
403 - Forbidden	This status code represents that the client who is trying to access the data is not authorised for the same
404 - Not Found	This status code represents that the requested server is no longer found at the given address
500 - Internal Server Error	This server response functions as a collection status code for an unexpected server error(s)
503 - Service Unavailable	This server response indicates that the server that is trying to be accessed is currently overloaded

Request-Response Cycle

Request-Response Cycle

upGrad



Hands-on Exercise 1 (10 mins)

upGrad

- Create a directory called **simple_server,** by using the ***mkdir*** command
- Initialise your project by using **npm init**
- When you installed Node.js, the **http** module also was installed. You shall use the **http** module to build the web servers
- You would need a package named **http-status-codes,** to provide constants for use where the HTTP status codes are needed in your application responses

```
run npm i http-status-codes -S
```

- Now, create a new file called **main.js**
- Next, you will type the code given here in **main.js**
- Finally, run ***node main*** command in the terminal

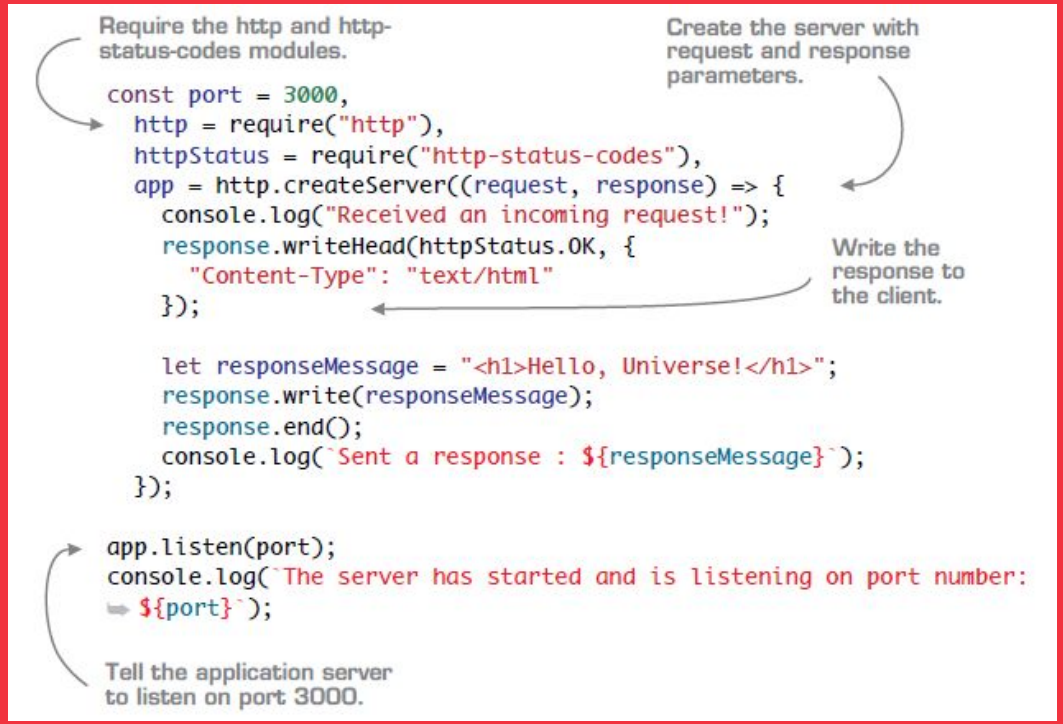
Note*:* 1. Port 3000 is generally used for web servers in development
2. Ports 80 and 443 usually are reserved for HTTP and HTTPS, respectively

upGrad

Hands-on Exercise 1

- The **createServer()** generates a new instance of http.Server, a built-in Node.js class with tools for evaluating HTTP communication. With this newly created server instance, your app is prepared to receive HTTP requests and send HTTP responses

- The argument in **createServer** is a callback function that is invoked whenever some event occurs. When the server is running and your application's root URL (index page) is accessed, an HTTP request event triggers this callback and allows you to run some custom code. In this case, the server returns a simple HTML response

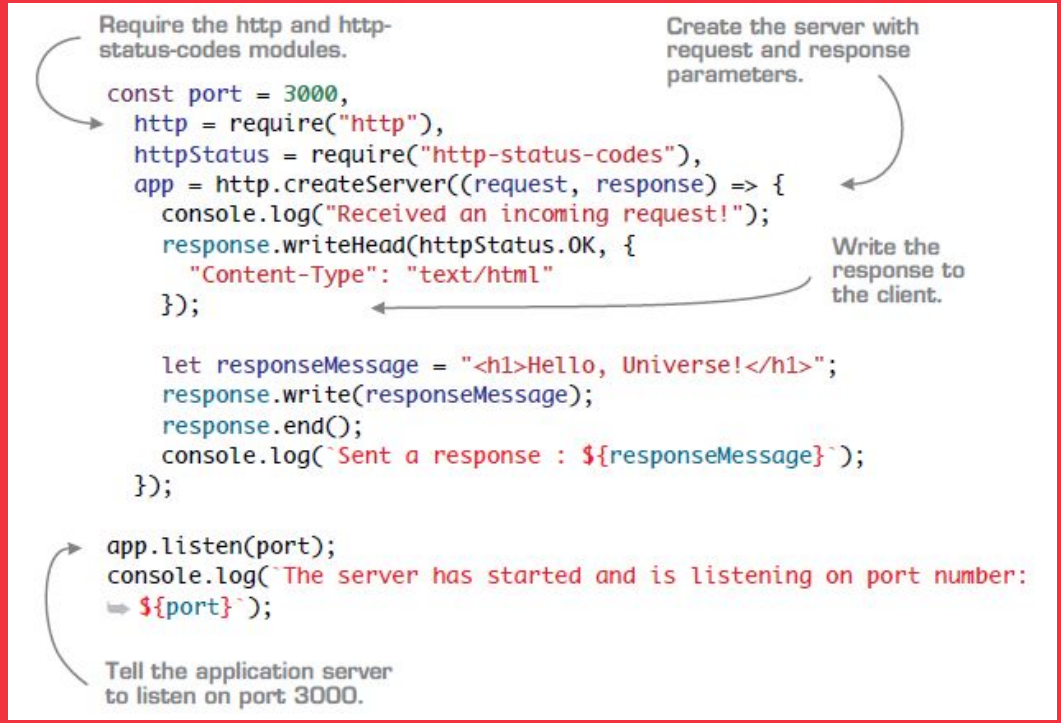



```
const port = 3000,
  http = require("http"),
  httpStatus = require("http-status-codes"),
  app = http.createServer((request, response) => {
    console.log("Received an incoming request!");
    response.writeHead(httpStatus.OK, {
      "Content-Type": "text/html"
    });

    let responseMessage = "<h1>Hello, Universe!</h1>";
    response.write(responseMessage);
    response.end();
    console.log(`Sent a response : ${responseMessage}`);
  });

app.listen(port);
console.log(`The server has started and is listening on port number:
➥ ${port}`);
```

upGrad

Hands-on Exercise

- writeHead() defines some basic properties of the response's HTTP header. HTTP headers contain fields of information that describe the content being transferred in a request or response




```
const port = 3000,
  http = require("http"),
  httpStatus = require("http-status-codes"),
  app = http.createServer((request, response) => {
    console.log("Received an incoming request!");
    response.writeHead(httpStatus.OK, {
      "Content-Type": "text/html"
    });

    let responseMessage = "<h1>Hello, Universe!</h1>";
    response.write(responseMessage);
    response.end();
    console.log(`Sent a response : ${responseMessage}`);
  });

app.listen(port);
console.log(`The server has started and is listening on port number:
➥ ${port}`);
```

Poll 4 (15 Sec)

Which of the following HTTP response codes shows that the request was carried out successfully?

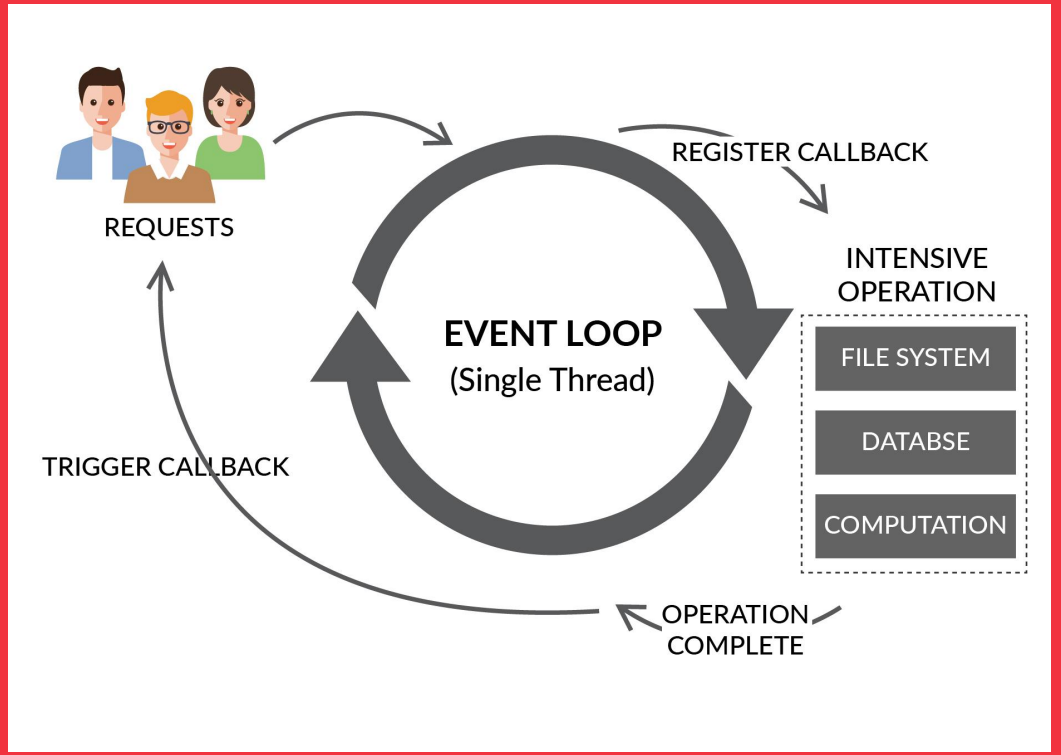
1. 404
2. 403
3. 200
4. 500

Poll 4 (Answer)

Which of the following HTTP response codes shows that the request was carried out successfully?

1. 404
2. 403
3. **200**
4. 500

Callbacks in Node.js



Callbacks in Node.js

upGrad

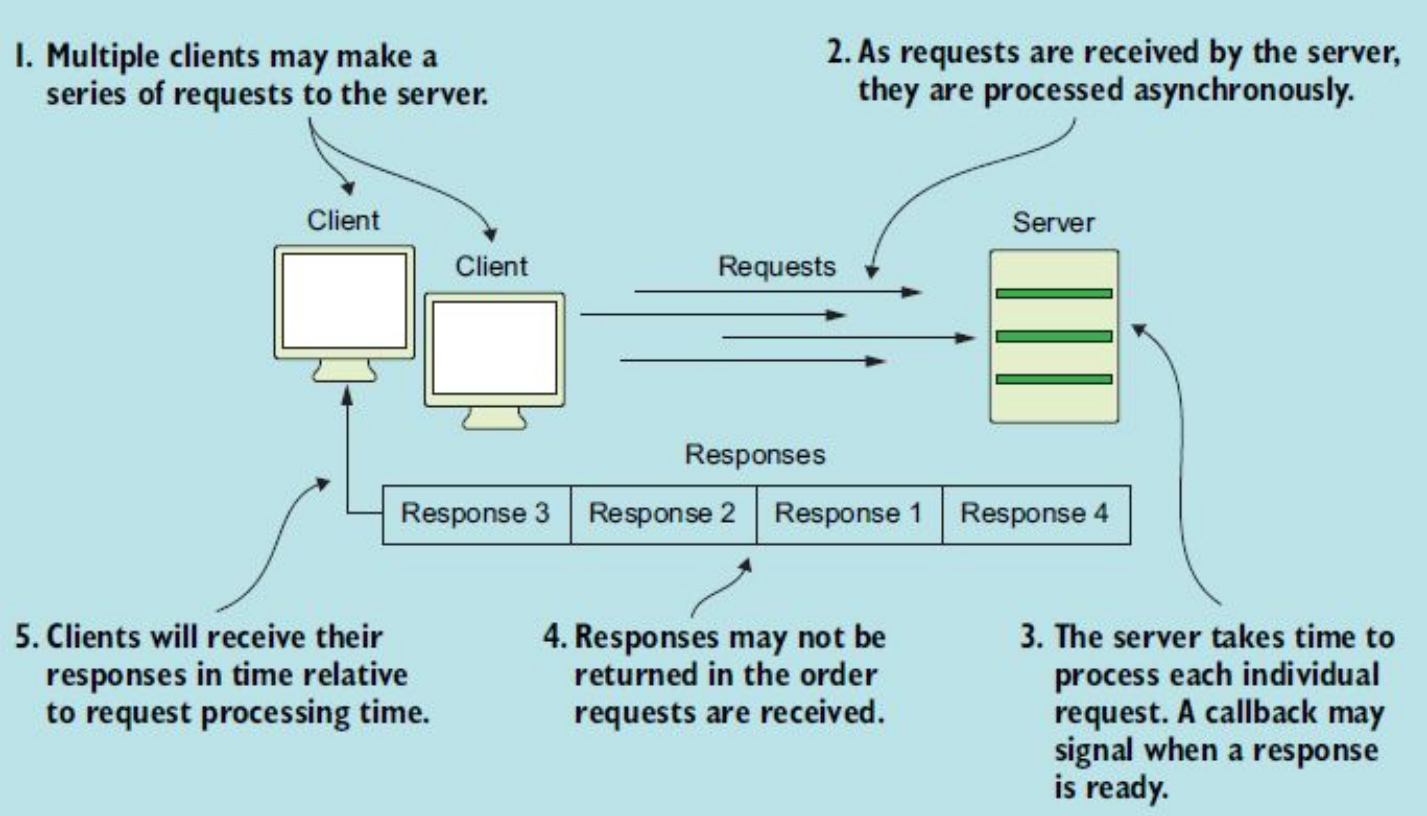
- A callback is an anonymous function (a function without a name) that is to be invoked as soon as another function completes its execution

- Consider the case of virtually depositing a check in your bank account by uploading a picture to your (ICICI) bank's mobile app. So, in this case, a callback is equivalent to receiving a notification after a few hours that the check was verified and deposited. However, in the meantime, you were able to perform other tasks on the bank's app

- In case of the HTTP web server, many requests are received and each request processing takes a different time to complete. Thus, here the callback function intimates the client only when the response is ready

Callbacks in Node.js

upGrad



Poll 5 (15 Sec)

Use **const** instead of **var** to store the HTTP server in your application. This is because the server will continue to listen for communication from clients, and it is important not to re-assign the variable representing the server. Moreover, in ES6, it has become a convention to mark these objects as constants, not re-assignable variables

Is the above statement true or false?

1. True
2. False

Poll 5 (Answer)

Use **const** instead of **var** to store the HTTP server in your application. This is because the server will continue to listen for communication from clients, and it is important not to re-assign the variable representing the server. Moreover, in ES6, it has become a convention to mark these objects as constants, not re-assignable variables

Is the above statement true or false?

1. **True**
2. False

Poll 6 (15 Sec)

When you navigate to ***http://localhost:4500/*** while your server is running, which of the following types of HTTP request are you making?

1. HTTP GET Request
2. HTTP POST Request

Poll 6 (Answer)

When you navigate to ***http://localhost:4500/*** while your server is running, which of the following types of HTTP request are you making?

1. **HTTP GET Request**
2. HTTP POST Request

upGrad

Handling Server Incoming Data

Handling Server Incoming Data

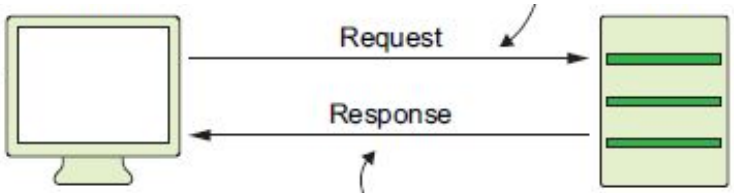
upGrad

POST Request

- The POST method is used to send data to a server, such that a resource is created or updated
- The parameters passed along with the POST method are not a part of the URL; hence, they are not stored in the browser history
- The POST method can be used while sharing sensitive data
- The POST requests do not have any length restriction

Handling Server Incoming Data

upGrad



Each request object has 3 properties:

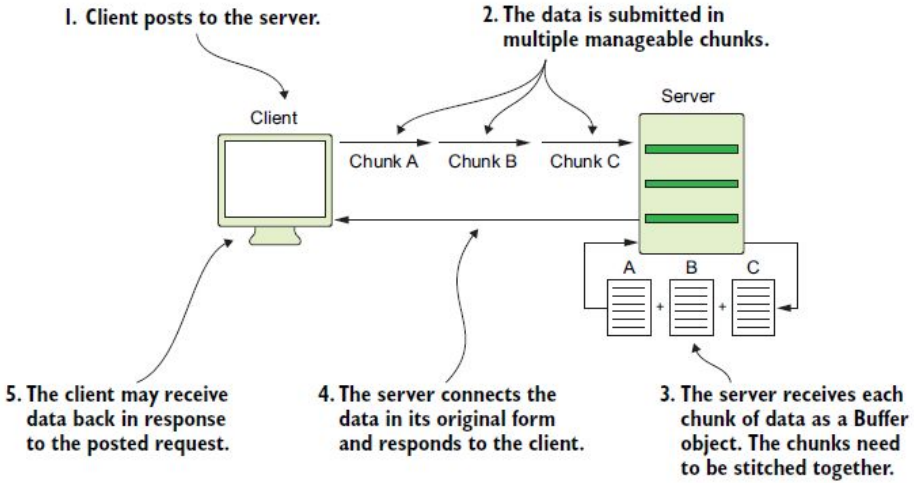
- **request.method**
  Indicates the HTTP method used
- **request.url**
  Indicates requesting URL
- **request.headers**
  Prints header attributes in a JSON form

Handling Server Incoming Data

upGrad

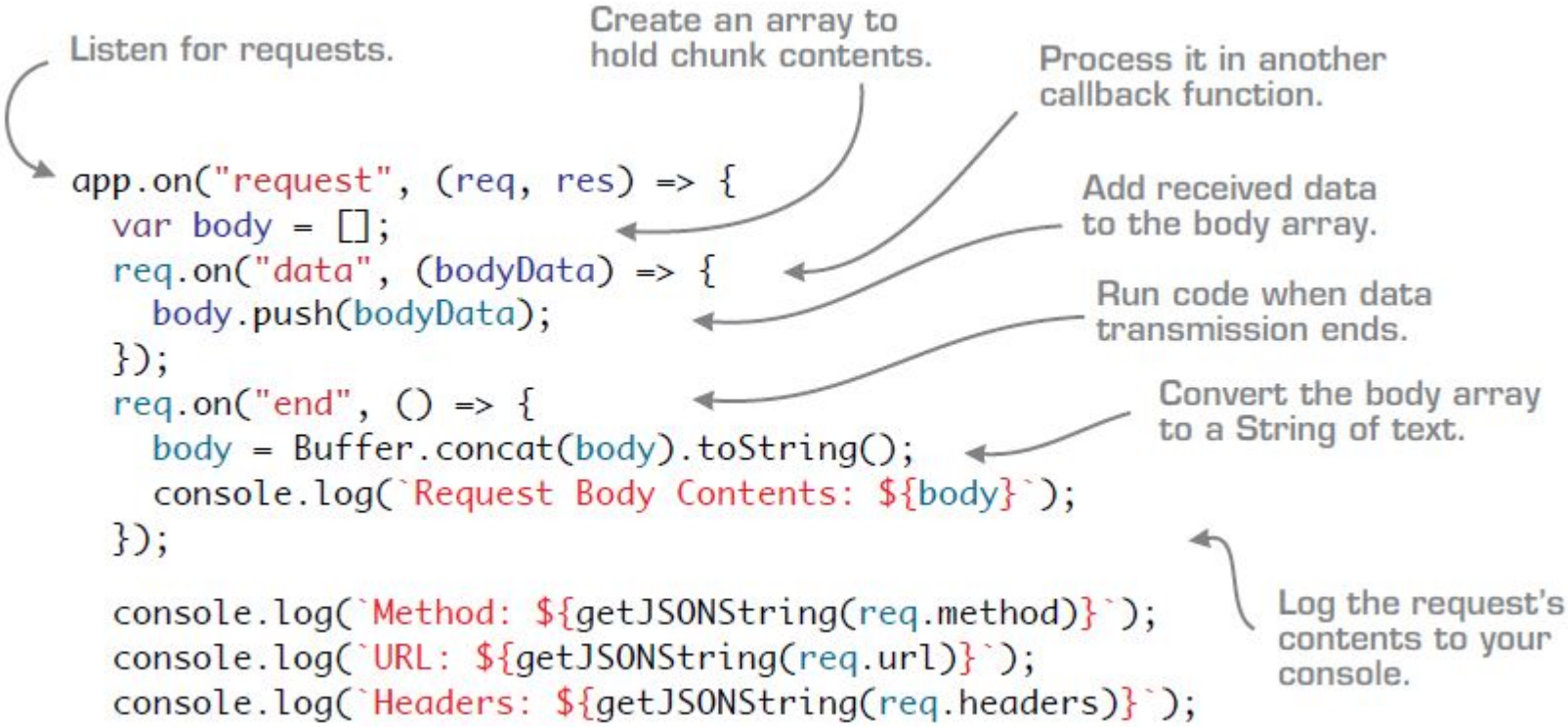
- The web server collects posted data in the form of data chunks and arranges it
- The data chunks allow information to stream in and out of a server

Note: Instead of waiting for a large set of information to arrive at the server, Node.js allows you to work with parts of that information as it comes via the **ReadableStream** library



Handling Server Incoming Data

upGrad



Poll 7 (15 Sec)

Every submitted form sends its full contents in a single chunk of data.

Is the above statement true or false?

1. True
2. False

Poll 7 (Answer)

Every submitted form sends its full contents in a single chunk of data.

Is the above statement true or false?

1. True
2. **False**

upGrad

Routes

Routes

upGrad

A route is a way of determining how an application should respond to a request made to a specific URL

Example of URLs:

/info	**http:// localhost:3000/info**
/contactUs	**http:// localhost:3000/contactUs**
/	**http:// localhost:3000/**

‘/’ is the home page, index page, or default page

Routes

upGrad

- Mapping of routes to the responses is called **routeResponseMap**
- Suppose a request is made to ***http://localhost:3000/info***, you will check whether the request's URL has a match in **routeResponseMap** and respond with an info page heading
- Suppose a request is made to ***http://localhost:3000/contact***. In this case, you will respond with a contact page heading

```
const routeResponseMap = {
  "/info": "<h1>Info Page</h1>",
  "/contact": "<h1>Contact Us</h1>",
  "/about": "<h1>Learn More About Us.</h1>",
  "/hello": "<h1>Say hello by emailing us here</h1>",
  "/error": "<h1>Sorry the page you are looking for is not here.</h1>"
};

const port = 3000,
  http = require("http"),
  httpStatus = require("http-status-codes"),
  app = http.createServer((req, res) => {
    res.writeHead(200, {
      "Content-Type": "text/html"
    });

    if (routeResponseMap[req.url]) {
      res.end(routeResponseMap[req.url]);
    } else {
```

Define mapping of routes with responses.

Check whether a request route is defined in the map.

Respond with default HTML.

Poll 8 (15 Sec)

With which of the following URLs will you route requests to the home page?

1. /
2. /index.js
3. /home.js
4. /root.js

Poll 8 (Answer)

With which of the following URLs will you route requests to the home page?

1. **/**
2. /index.js
3. /home.js
4. /root.js

Hands-on Exercise 2 (10 mins)

upGrad

- Create a project called **simple_routes**
- Initialise the project by using **npm init**
- Install **http-status-codes**
- In **main.js** file of simple_routes, code the **routeResponseMap** function

```
const routeResponseMap = {
    "/info": "<h1>Info Page</h1>",
    "/contact": "<h1>Contact Us</h1>",
    "/about": "<h1>Learn More About Us.</h1>",
    "/hello": "<h1>Say hello by emailing us here</h1>",
    "/error": "<h1>Sorry the page you are looking for is not here.</h1>"
};
```

- Now, code the **createServer()** function
- Finally, run the project by using **node simple_routes**

Refer : simple_routes

upGrad

Project Work - Checkpoint 1

Checkpoint 1 - Problem Statement

upGrad

In the first checkpoint, you will understand the problem statement and the architecture of our project:

- Suppose you are a MERN Stack Developer and you are allotted a task by upGrad to create a prototype for their website. (Hereafter, we would refer to upGrad as **'Client'**)

- The client wants you to create a simple working model where the following use cases can be experienced:
  - Home page:
    - A course can be searched by its title
    - Courses can also be searched by their category (E.g., front end, backend)
    - Cards of the recently published courses can be seen with their image, title and price
    - Users' login can view all courses without logging in

  - When a course is selected from the home page, the user is directed towards the course details page

Checkpoint 1 - Problem Statement

upGrad

- Course details page:
  - Title of the course
  - Duration of the course (in hours)
  - Course image
  - Introductory video
  - Price of the course
  - Discounted price of the course
  - Major skills of the course
  - Chapters
  - 'Enroll Me' button

- There are two types of login:
  - Admin login
  - User login

Checkpoint 1 - Problem Statement

upGrad

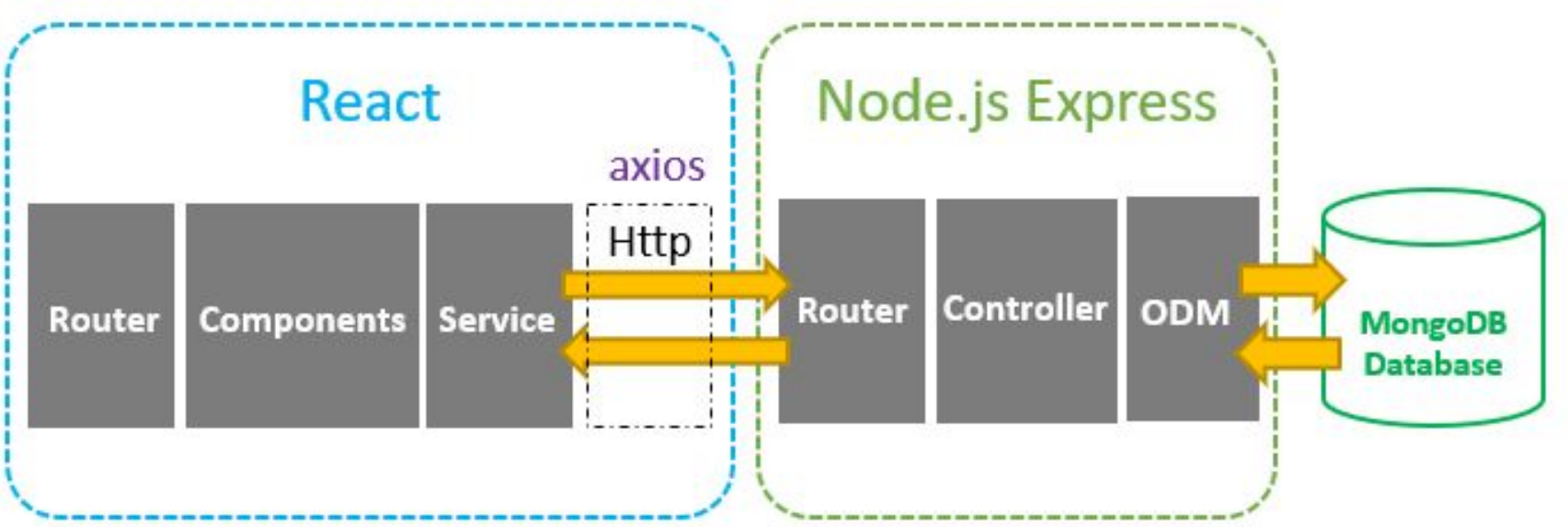
- On **admin** login:
  - Course can be added, deleted and edited

- The client wants you to store the data about courses in MongoDB, the front-end is to be made by using React.js and the APIs are to be created from Node-Express.js

Checkpoint 1 - Project Architecture

upGrad

Our React.js + Node.js + Express.js + MongoDB applications will follow the architecture given below



Checkpoint 1 - Project Architecture

upGrad

Terminologies used in the previous MERN stack architecture diagram:

1. Axios is a library that helps us make the http requests to external resources. In our React applications, we often need to retrieve data from the external APIs so that it can be displayed in our web pages

2. ODM refers to object document mapping. Mongoose is an example of ODM. It helps us to make a schema, which we strictly follow to add data to the database (MongoDB)

Homework

Create a web server to listen on port 3000
The server should be able to handle following routes

1. /aboutus
   respond with a message, "Show a aboutus web page"

2. /contactus
   respond with a message, "Show a aboutus web page"

3. For any other url
   respond with a message, "hello everyone at upgrad"

4. Test all routes with METHOD as GET and POST using SWAGGER or POSTMAN

Solution : simple_routes_home

Doubt Clearance (5 mins)

upGrad

Key Takeaways

- You learnt about web servers and understood why the Node.js applications are built on web servers
- You analysed how server handles the incoming data
- You were introduced to the concept of routes
- Finally, you understood the problem statement and architecture of your project

The following tasks are to be completed after today's session:

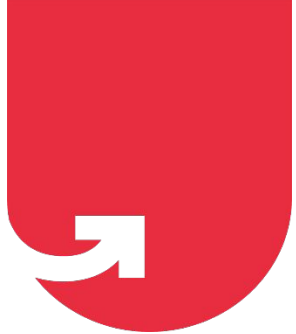
Homework
MCQs
Coding Questions
Course Project - Checkpoint 2

In the next class, we will discuss.....

- Routing the static HTML web pages

upGrad

#RahoAmbitious



Thank you!