upGrad

#RahoAmbitious

Full Stack Software Development

upGrad

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

Lecture on: Introduction to Node.js, Express.js and MongoDB

Instructor: Rocky Jagtiani

MERN Stack


React
Express JS
node
JS
mongoDB
client-side/frontend
server-side/backend
database

Poll 1 (15 Sec)

React is used for _____.

1. Client-side communication
2. Server-side communication

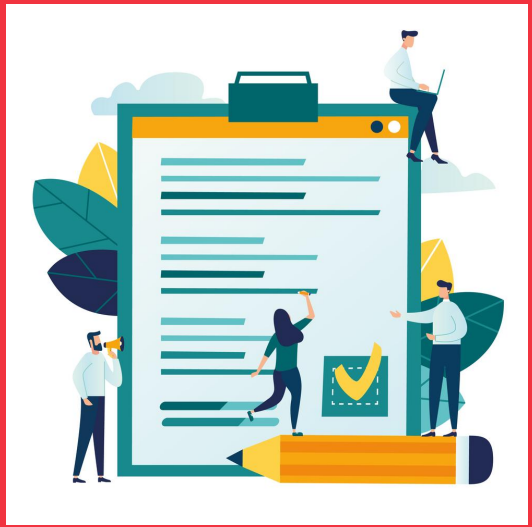
Poll 1 (Answer)

React is used for _____.

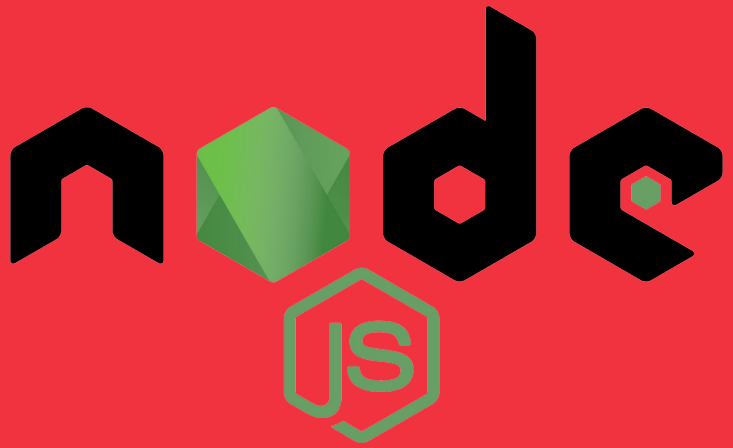
1. **Client-side communication**
2. Server-side communication

Today's Agenda

- Introduction to Node.js
- REPL
- Running JavaScript File With Node.js
- Modules and Packages
- Node Package Manager
- Introduction to Express.js
- Introduction to Database
- Introduction to MongoDB



Introduction to Node.js



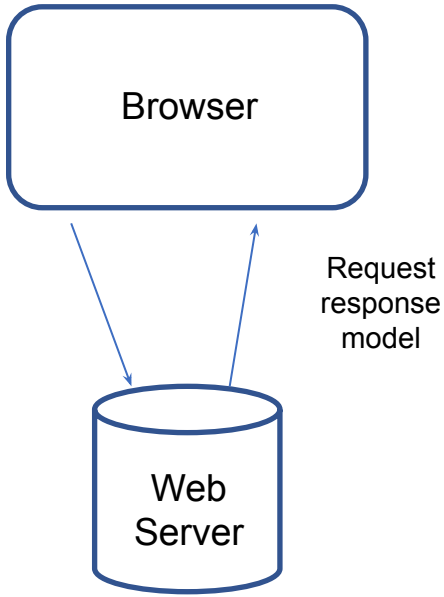
Client-Server Interaction

upGrad

Web development can broadly be broken into two categories:

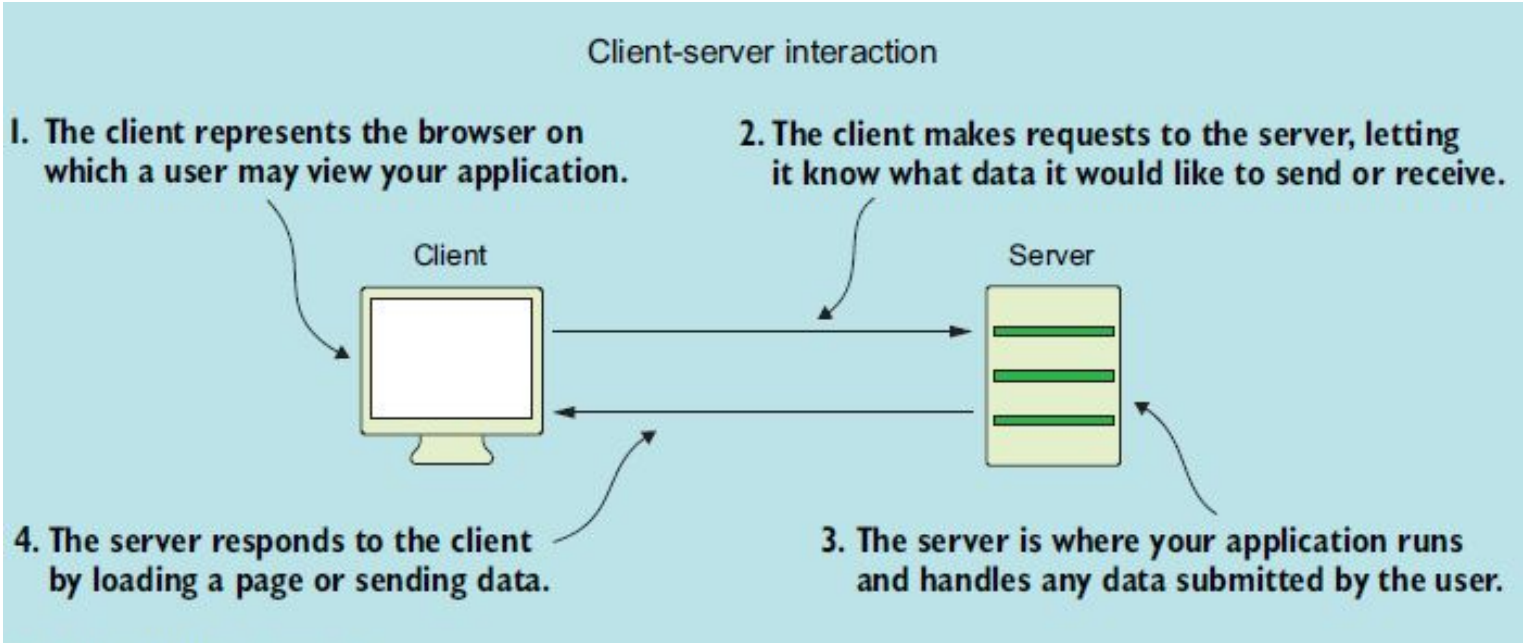
- ***Client-side*** (front-end): refers to the code you write that results in something the user sees in his web browser. Client-side code typically includes JavaScript used to animate the user experience when a web page is loaded

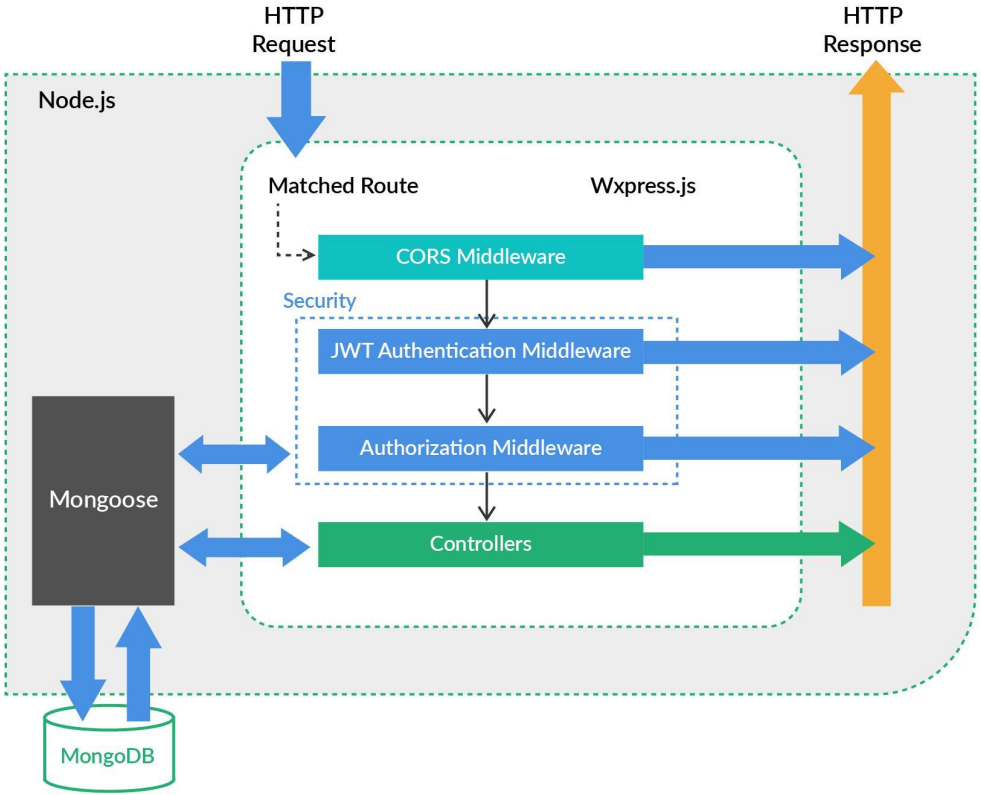
- ***Server-side*** (back-end): refers to code used for application logic (how data is organised and saved to the database). Server-side code is responsible for authenticating users on a login page, running scheduled tasks, and even ensuring that the client-side code reaches the client



Client-Server Interaction

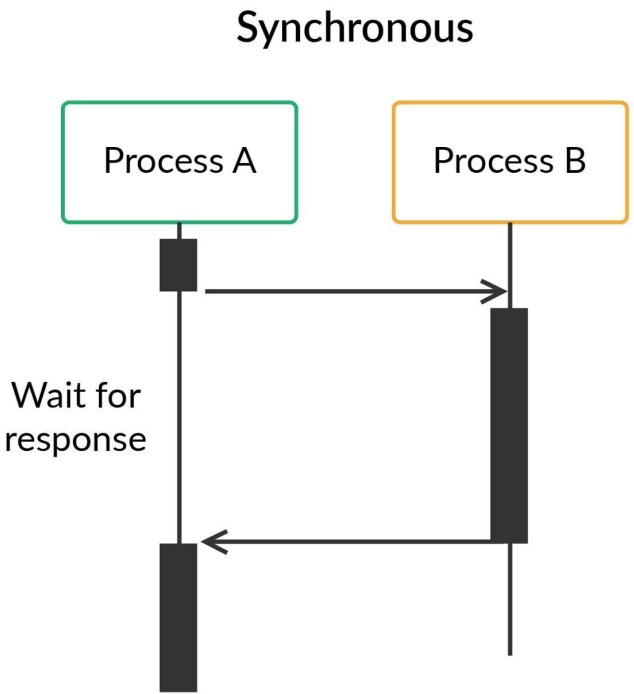
upGrad

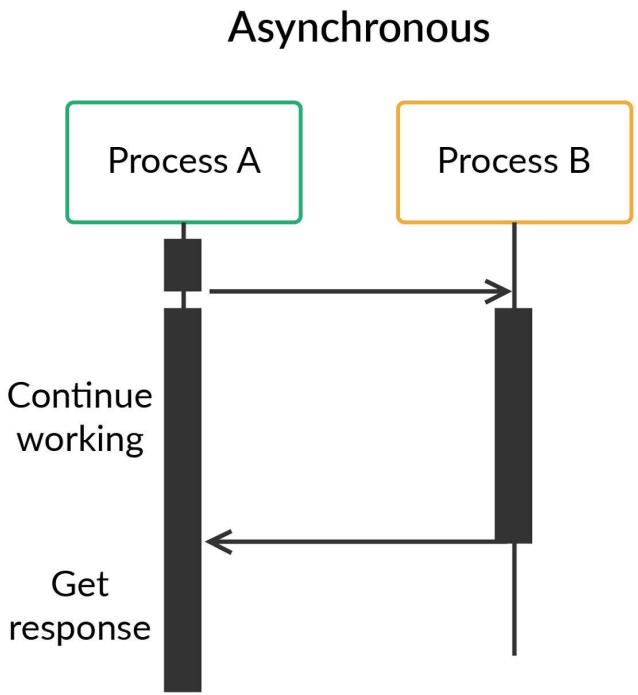



HTTP
Request
HTTP
Response
Node.js
Matched Route
Wxpress.js
CORS Middleware
Security
JWT Authentication Middleware
Authorization Middleware
Controllers
Mongoose
MongoDB

Synchronous Vs Asynchronous

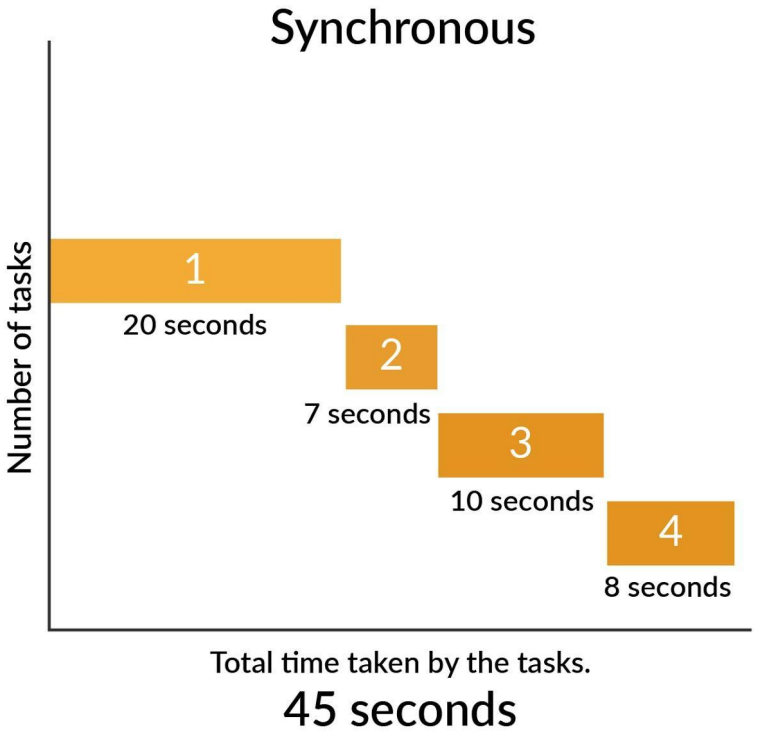
upGrad

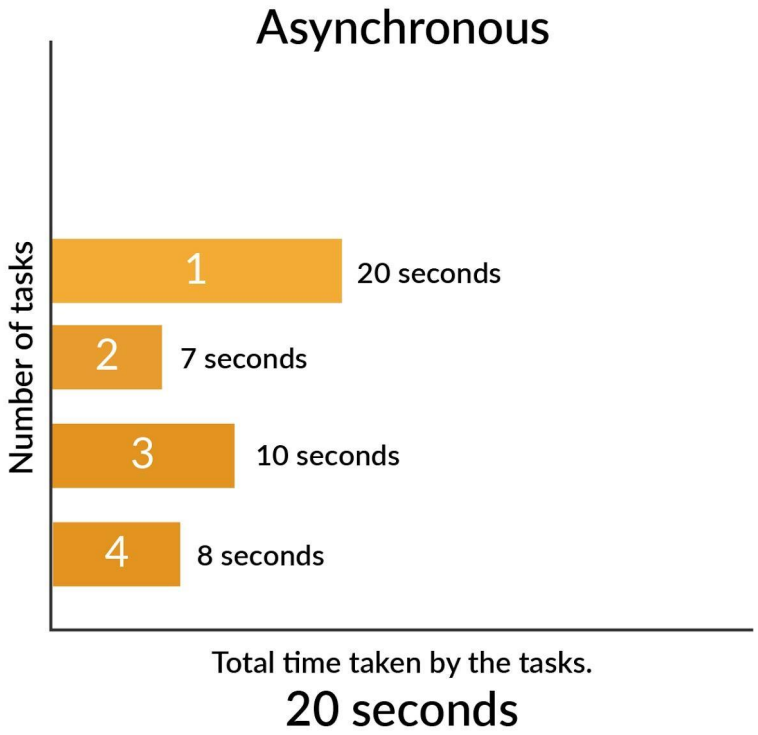




Synchronous Vs Asynchronous

upGrad





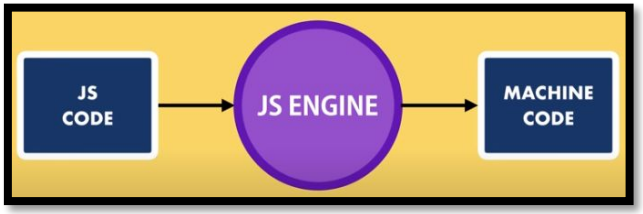
Introduction to Node.js

upGrad

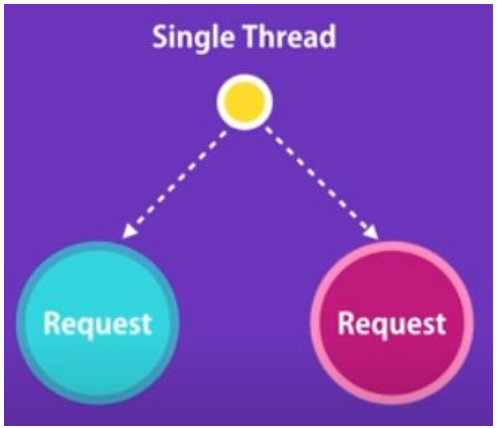
The official definition of Node.js is, '***Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine***'

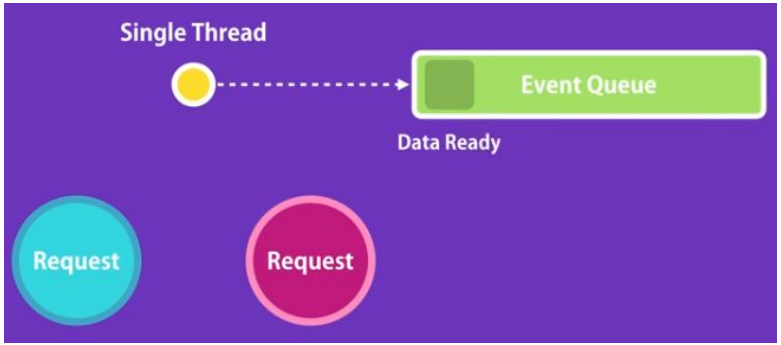
Let's try to break the definition for better understanding:

- ***Node.js® is a JavaScript runtime:*** It simply means an environment in which the JavaScript code runs when executed
- ***built on Chrome's V8 JavaScript engine:***

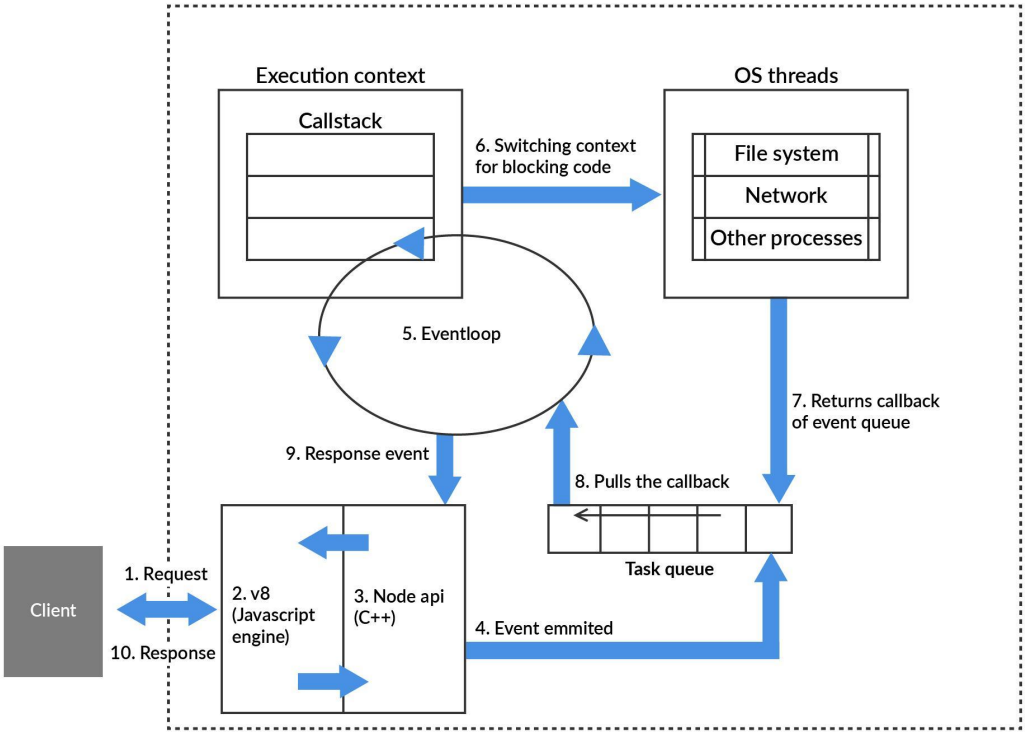


Let's understand the asynchronous behaviour of Node.js with a simple example





Let's look into the runtime environment of JavaScript in detail



JavaScript's runtime environment consists of the following components:

1. **Memory Heap:** It is a data structure that contains the memory allocated to all the variables and functions used in a program.

2. **Execution/Call Stack:** It is a data structure that tells where in the program you are.

3. **Web APIs Container:** It contains all the long-running tasks such as event listeners, HTTP/AJAX requests and HTML Timer APIs. Each long-running task must ideally have a callback associated with it, which must be invoked after completing the long-running task. When an event is triggered, or the long-running task finishes its execution, the associated callback is sent to the callback queue.

4. **Callback Queue:** The callback queue stores all the callback functions in the order in which they are added. It follows the traditional approach of *FIFO* (First-In-First-Out). The first callback added to the queue is the first one that will be executed. When the execution (or call) stack is empty, the callback is sent to the stack to be executed

5. Event Loop: This loop constantly monitors the execution (or call) stack and the callback queue. If the callback queue consists of some code to be executed, but the stack is not empty, the event loop waits for it to get empty. Whenever it finds the execution stack empty (and the callback queue contains a callback method), it sends the callback at the beginning of the queue to the stack's top. When a callback is pushed to the stack, it is then executed. In another scenario, when both the execution stack and the queue are empty, the event loop waits for some callback to be added to the callback queue

Observe the behaviour of the code shown below

```
const foo = () => {
   console.log('Hey');
   bar();
}
const bar = () => {
   console.log('Hello');
   setTimeout(() => {
      console.log('I am executed at last!');
   }, 2000);
   baz();
}
const baz = () => {
   console.log('Hi');
}
foo();
```

The output is shown below

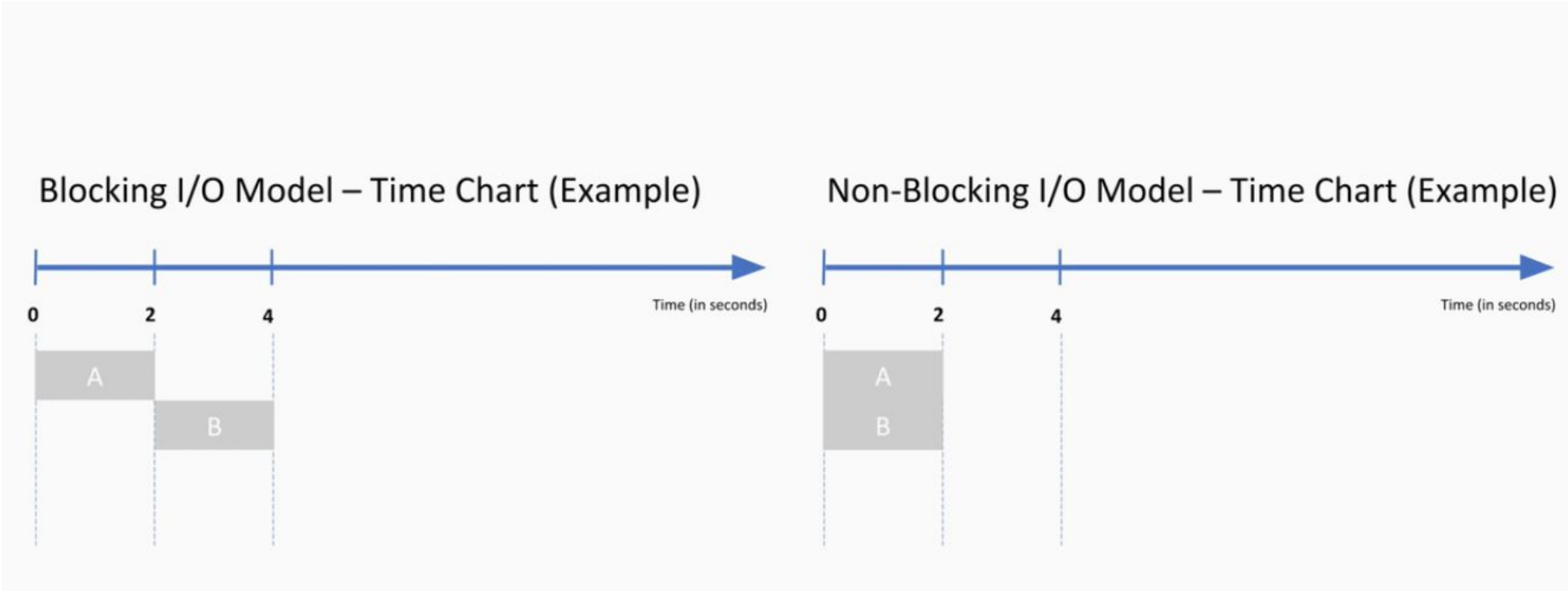
```
Hey
Hello
Hi
I am executed at last!
```

- In the previous example, Web APIs container, callback queue, and event loop components in the JavaScript runtime assume that JavaScript can run code asynchronously

- However, this is just an illusion. In reality, JavaScript is synchronous because it can execute only one function at a time, which is at the top of the execution stack

- The illusion occurs because the long-running tasks are sent to the Web APIs container in parallel, but their callbacks are executed only when the stack is empty

Blocking vs Non-Blocking I/O Model



upGrad

Blocking I/O Model

```
const foo = () => setTimeout(() => console.log('Inside foo!'), 2000);
const bar = () => setTimeout(() => console.log('Inside bar!'), 4000);

foo();
bar();


//Output:
At 2s:
Inside foo!
At 6s:
Inside bar!
```

Non-Blocking I/O Model

```
const foo = () => setTimeout(() => console.log('Inside foo!'), 2000);
const bar = () => setTimeout(() => console.log('Inside bar!'), 4000);

foo();
bar();


//Output:
At 2s:
Inside foo!
At 4s:
Inside bar!
```

1. Node is used to make highly scalable, data-intensive real-time apps.

2. It uses **non-blocking asynchronous** calls (one waiter serves many tables).

3. A **single thread** handles multiple requests (The waiter in the example can be considered as a thread).

4. As node uses a single thread or asynchronous architecture, it requires less hardware on the server.

5. Node is ideal for I/O-intensive apps (not for CPU intensive apps like image processing).
   Example: lots of database read-write request
   Node uses an event queue to monitor when the data or response is ready.

Pointers:

- ✔ Node **is not** a programming language
- ✔ Node **is not** a framework
- ✔ Node is a runtime environment for executing JavaScript code

Poll 2 (30 Sec)

Node uses:

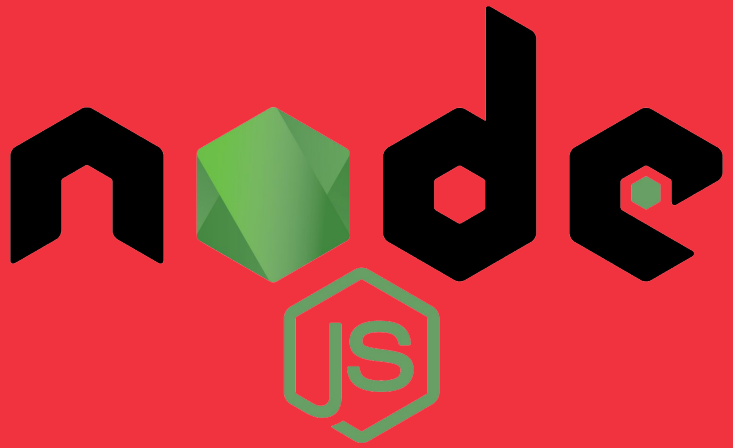
1. Blocking or asynchronous calls
2. Non-blocking or synchronous calls
3. Blocking or synchronous calls
4. Non-blocking or asynchronous calls

Poll 2 (Answer)

Node uses:

1. Blocking or asynchronous calls
2. Non-blocking or synchronous calls
3. Blocking or synchronous calls
4. **Non-blocking or asynchronous calls**

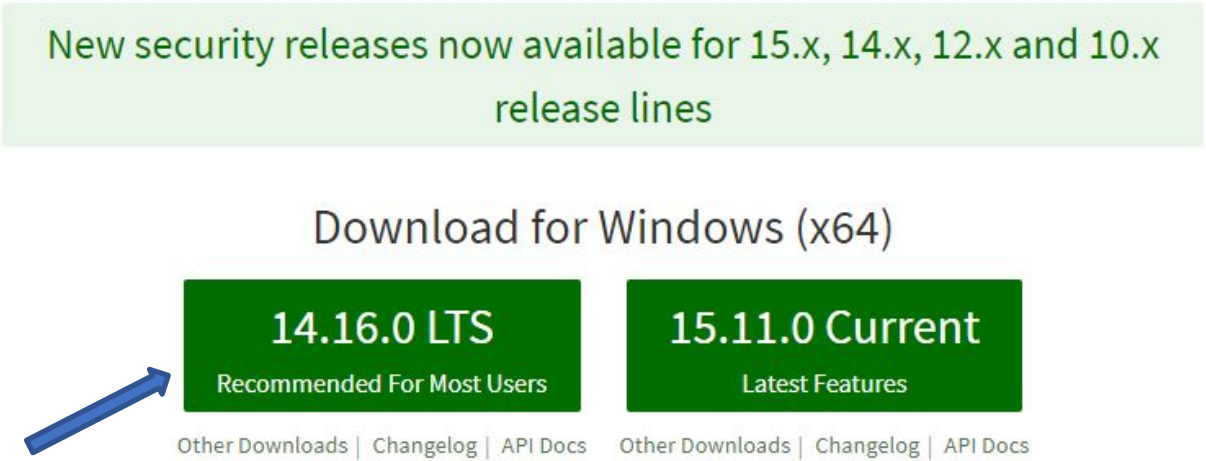
Installing Node.js



Installing Node.js

upGrad

1. Go to Node.js main site, https://nodejs.org



2. Follow the instructions and prompts to download the installer for the latest version of Node.js

Because Node.js is platform-independent, you can download and install it on your Mac, Windows, or Linux computer and expect full functionality

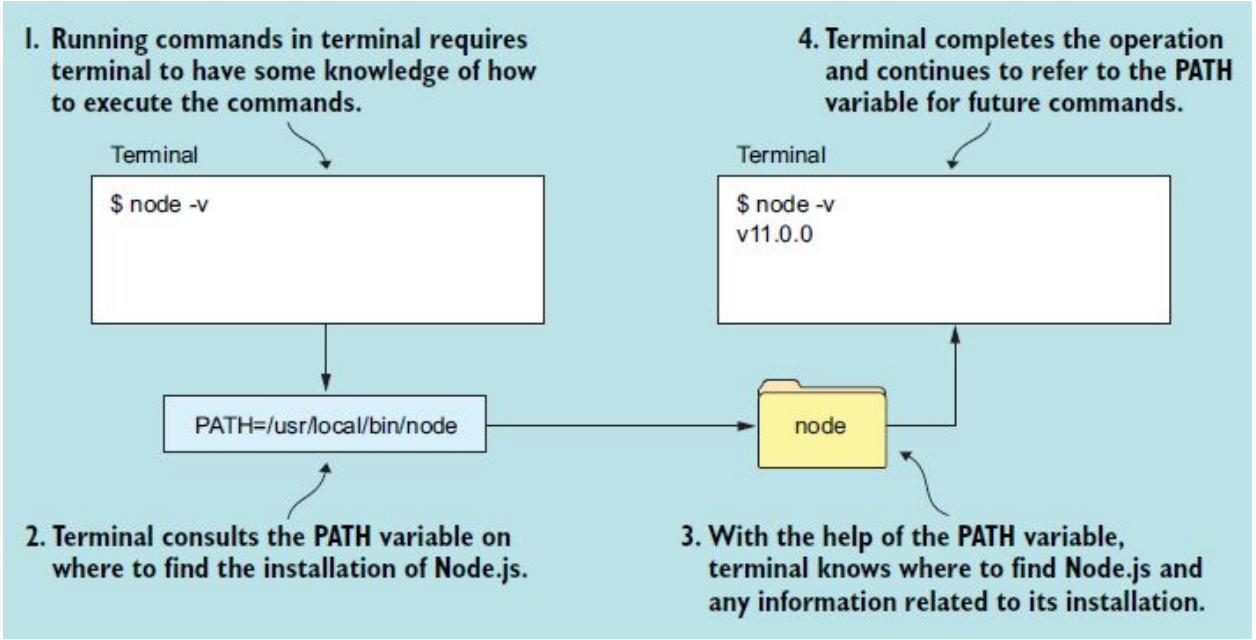
When you install Node.js, you also get **npm**, the Node.js ecosystem of external libraries

Installing Node.js

upGrad

1. You will be working mostly in your computer's **terminal**

1. **PATH** is an environmental variable, i.e., a variable that can be set to influence the behavior of operations on your machine. Your computer's PATH variable specifies where to find directories and executable files needed to perform operations on your system



Tip: Preferably use Visual Studio Code IDE (https://code.visualstudio.com/)

Read-Evaluate-Print-Loop (REPL)

Read-Evaluate-Print-Loop (REPL)

upGrad

The interactive Node.js shell is called a **Read-Evaluate-Print-Loop (REPL)** shell

This shell is a space in which you can write pure JavaScript and evaluate your code in the terminal window in real-time. Within the window, your written code is read and evaluated by Node.js, and results are printed back on the console

You can get to REPL by just typing in **'node'** keyword in your terminal window

```
C:\SuvenNodeApps\NodeProjects> node
Welcome to Node.js v12.17.0.
Type ".help" for more information.
> 3+3
6
> 3/0
Infinity
> console.log("hello everyone at Upgrad !! from Rocky Sir")
hello everyone at Upgrad !! from Rocky Sir
undefined
> let name = "Suven"
undefined
> console.log(name)
Suven
undefined
```

Refer repl.js

Read-Evaluate-Print-Loop (REPL)

upGrad

- REPL (read, evaluate, print, loop) is an interactive environment where you can read input from the user, evaluate it, print the result(s) of the evaluation and loop these three commands until termination

- REPL lets you see the results of your input code immediately without going through the compilation phase in the cycle

```
code -> compile -> execute
```

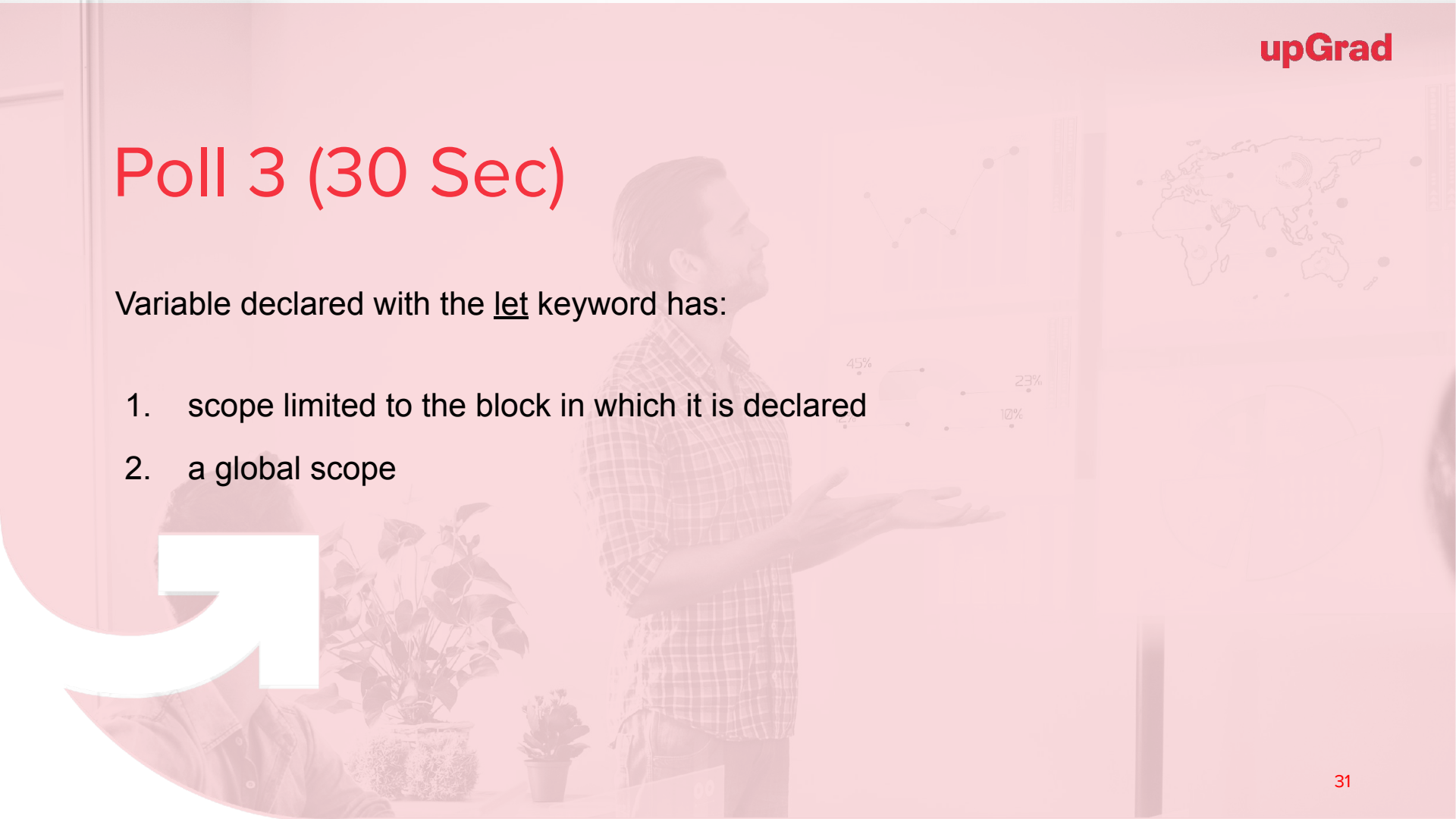
  Thus, you can directly execute the code without compiling it

- REPL for Node.js comes out-of-the-box with Node.js while installing the latter

Poll 3 (30 Sec)

Variable declared with the let keyword has:

1. scope limited to the block in which it is declared
2. a global scope



Poll 3 (Answer)

Variable declared with the let keyword has:

1. **scope limited to the block in which it is declared**
2. a global scope

Poll 4 (45 Sec)

What are three ways in which you could exit the REPL environment?

To exit your Node.js REPL environment, you can type:

(choose all 3 correct options)

1. type .exit() on the node terminal
2. press Ctrl-C twice
3. press Ctrl-D twice
4. type .exit on the node terminal

Poll 4 (Answer)

What are three ways in which you could exit the REPL environment?

To exit your Node.js REPL environment, you can:

(choose all 3 correct options)

1. type .exit() on the node terminal
2. **press Ctrl-C twice**
3. **press Ctrl-D twice**
4. **type .exit on the node terminal**

Running JavaScript File With Node.js

Running JavaScript File With Node.js

upGrad

Follow the steps mentioned below:

1. Open your text editor to a new window
1. Type the following code in that empty file: ***console.log("Hello, Universe!");***
1. Save this file as ***hello.js*** in ***UpgradNodeApps*** under ***C:***
1. Open a new terminal window
1. Navigate to file path
1. Run your JavaScript file by entering the node keyword followed by the file's name

Try Example 1 | Try Example 2

Useful Tip: Write **'use strict'** at start of the program before writing any line of code

Importance of strict mode

Use of strict mode helps discover:

- Accidentally created global variables: *You will not be able to create a variable without the var, let, or const keywords*
- Assigning variables that cannot be assigned: *You can't use undefined as a variable name*
- Using non-unique function parameter names or property names in an object literal: *You need to choose names that do not repeat within the same scope when assigning values*

Poll 5 (60 Sec)

Type the lines of code shown below and save the Javascript file as ***printNumbers.js***

```
let printNumbers = arr => {
arr.forEach(num => console.log(num));
};
printNumbers([10,'asd',20,45.56])
```

Next, on node terminal, type

```
> node printNumbers.js
```

What will be the output?

1. 10,'asd',20,45.56
2. 10
   asd
   20
   45.56
1. NumberFormatException is raised
2. 10
   NumberFormatException is raised

Poll 5 (Answer)

Type the lines of code shown below and save the Javascript file as ***printNumbers.js***

```
let printNumbers = arr => {
arr.forEach(num => console.log(num));
};
printNumbers([10,'asd',20,45.56])
```

Next, on node terminal, type

```
> node printNumbers.js
```

What will be the output?

1. 10,'asd',20,45.56
2. **10**
   asd
   20
   45.56
1. NumberFormatException is raised
2. 10
   NumberFormatException is raised

upGrad

Modules and Packages

About Modules, Packages and Dependencies

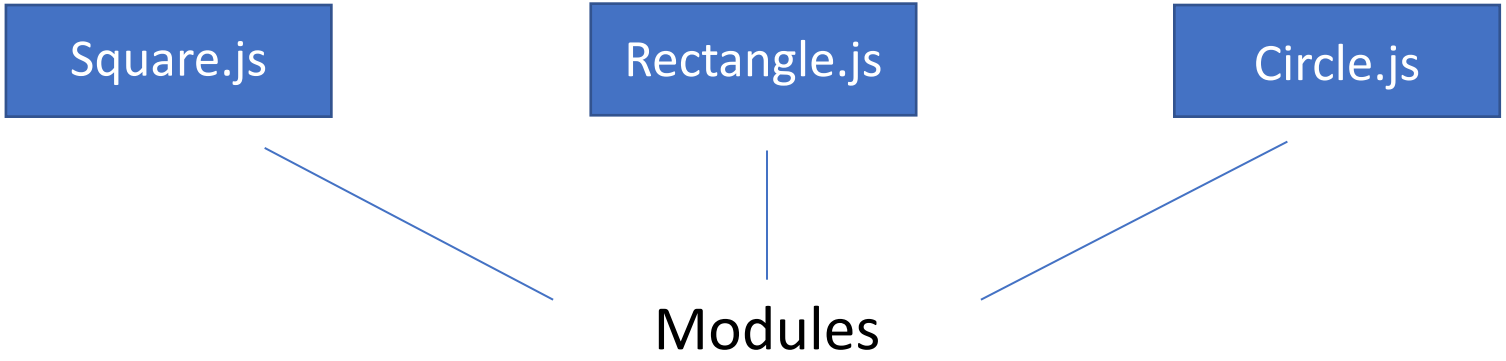
upGrad

The following terms will often be heard while working with Node.js:

- ***Modules*** are individual JavaScript files containing code that pertains to a single concept, functionality, or library
- ***Packages*** may contain multiple modules or a single module, which are used to group files offering relevant tools
- ***Dependencies*** are Node.js modules used by an application or another module

Modules

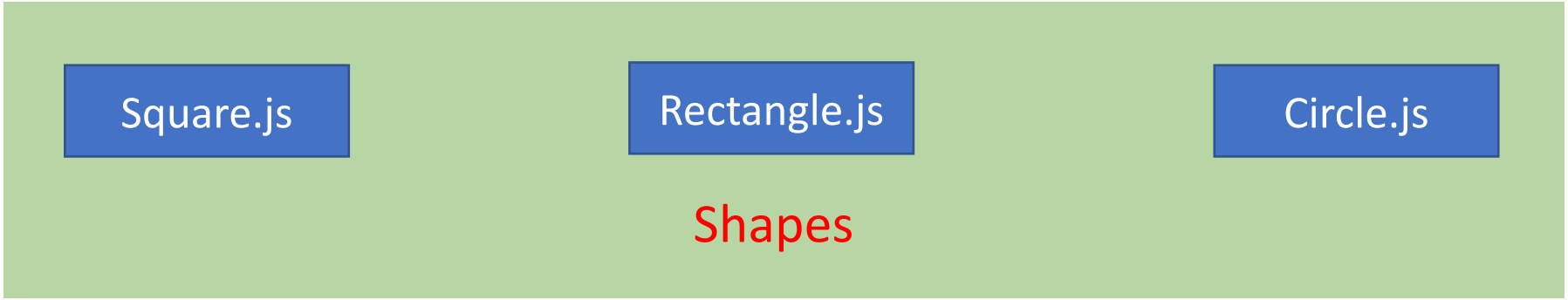
upGrad



A module corresponds to the entire code that you write in a single file

Packages

upGrad



Package

A package is two or more modules grouped together

Poll 6 (15 Sec)

Are you a fan of superheroes?
Which of the following can be a package?

1. Iron Man, Hulk, Captain America, Thor
2. Superman, Batman, Wonder Woman, Flash
3. Both of the above
4. None of the above

Poll 6 (Answer)

Are you a fan of superheroes?
Which of the following can be a package?

1. Iron Man, Hulk, Captain America, Thor
2. Superman, Batman, Wonder Woman, Flash
3. **Both of the above**
4. None of the above

Modules and Packages

upGrad

Suppose you want to build an application to help people share food recipes and learn from one another. Let's use Node.js to build this web application and start by verifying the ZIP codes of the users'. **So, do you think you would need a tool for checking ZIP codes for the application?**

No, you can simply use ***Node Package Manager(npm)*** to install Node.js ***packages*** ***(libraries of code others have written that add specific features to your application)*** where a package for verifying locations based on ZIP codes is already available

A Node.js application is made up of many JavaScript files. For your application to stay organised and efficient, these files need to have access to one another's contents when necessary

Each JavaScript file or folder containing a code library is called a module

Modules: Exports and Require

upGrad

- To access some function or object across javascript files, define the object using ***exports*** module

```
exports.messages = [
    "A change of environment can be a good thing!",
    "You will make it!",
    "Just run with the code!"
];
```

- ***exports*** is a property of the module object (*module is both the name of the code files in Node.js and one of its global objects*)

  Note: *exports is shorthand for module.exports*

Modules: Exports and Require

upGrad

- To import ***exports.object*** in another file, use ***require()*** module

```
const messageModule = require("./messages");
messageModule.messages.forEach ( m =>
        console.log(m)
);
```

To load libraries of code and modules in Node.js, use require()

The **module.require** function resides on the global module object

Try Example 3

upGrad

Hands-on Exercise 1 (10 mins)

Create a calculator application which does the following:

- Create a folder ***Example 4***
- In the folder Example 4, create a ***calculator.js*** file having following functionality
  - Add two numbers. The add function could be defined as shown below

```
exports.addNum = (x, y) => { return x + y; };
```

Note: *Here addNum function is assigned to exports module, so that its visible in other modules or javascript files*

  - Similarly, define three more functions to *subtract, multiply and divide.*

- Define another file ***main.js*** having following functionality
  - import or require the calculator module
  - Call all the four functions

Refer Example 4

Poll 7 (30 Sec)

What object is used to make functions or variables within one module available to others? (choose 2 correct options)

1. module.exports
2. exports
3. module.require
4. require

Poll 7 (Answer)

What object is used to make functions or variables within one module available to others? (choose 2 correct options)

1. **module.exports**
2. **exports**
3. module.require
4. require

Node Package Manager

Node Package Manager

upGrad

- ✔ With your installation of Node.js, you get npm, a package manager for Node.js
- ✔ npm is responsible for managing the external packages in your application
- ✔ Throughout application development, you use npm to install, remove, and modify these packages
- ✔ Entering **npm -l** in your terminal brings up a list of npm commands with brief explanations

npm init
Initialises a Node.js application and creates a package.json file

npm install <package>
Installs a Node.js package

npm docs <package>
Opens the likely documentation page (web page) for your specified package

npm install <package> --save
npm install <package> --S
installs the package as a dependency for your application.

npm install <package> --global
npm install <package> --g
installs the package globally, accessible to all modules.

npm uninstall <package>
uninstalls the given package.

Poll 6 (60 Sec)

The --global or -g flag installs a package for use as a command-line tool globally on your computer. The package can be accessible to other projects, not exclusively to the one you are working on

1) True

2) False

Poll 6 (Answer)

The --global or -g flag installs a package for use as a command-line tool globally on your computer. The package can be accessible to other projects, not exclusively to the one you are working on

1) True

2) False

Initialising a Node.js Application

upGrad

- Every Node.js application or module contains a **package.json** file to define the properties of that project
- package.json resides at the root level of the project
- Typically, this file is where you specify the version of your current release, the name of your application and the main application file
- To get started, create a folder say **node_project**, navigate to your project directory in the terminal and use the **npm init** command to initialise your application

Poll 7 (30 Sec)

Which of the following statements holds true?
(More than one option can be true)

1. **npm init** initialises a Node.js app and prompts you to create a package.json file
2. **npm init** initialises a Node.js app and prompts you to create a package-lock.json file
3. **npm install <packageName> --save** saves the package as a dependency for current project
4. **npm install <packageName> --save** saves the package globally for all projects to access

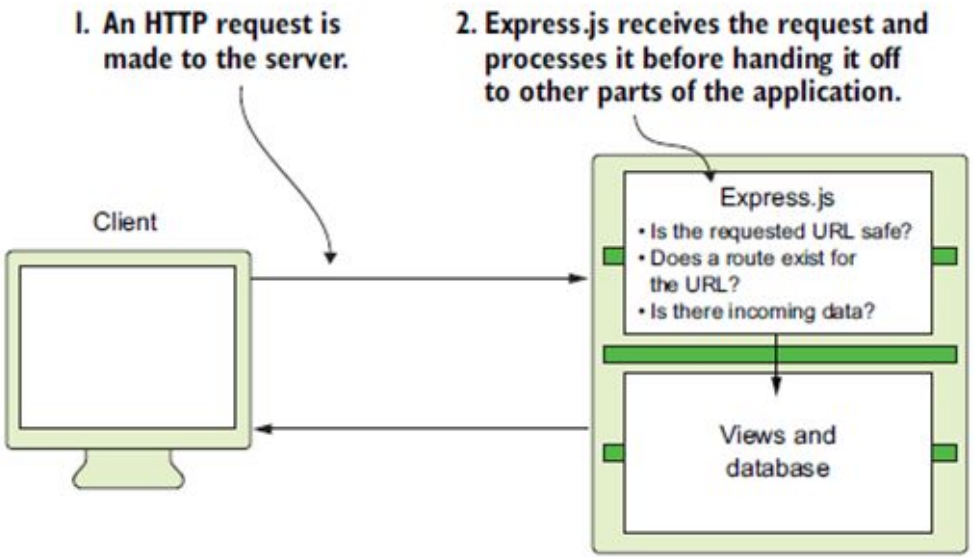
Poll 7 (Answer)

Which of the following statements holds true?
(More than one option can be true)

1. **npm init** initialises a Node.js app and prompts you to create a package.json file
2. **npm init** initialises a Node.js app and prompts you to create a package-lock.json file.
3. **npm install <packageName> --save** saves the package as a dependency for current project
4. **npm install <packageName> --save** saves the package globally for all projects to access

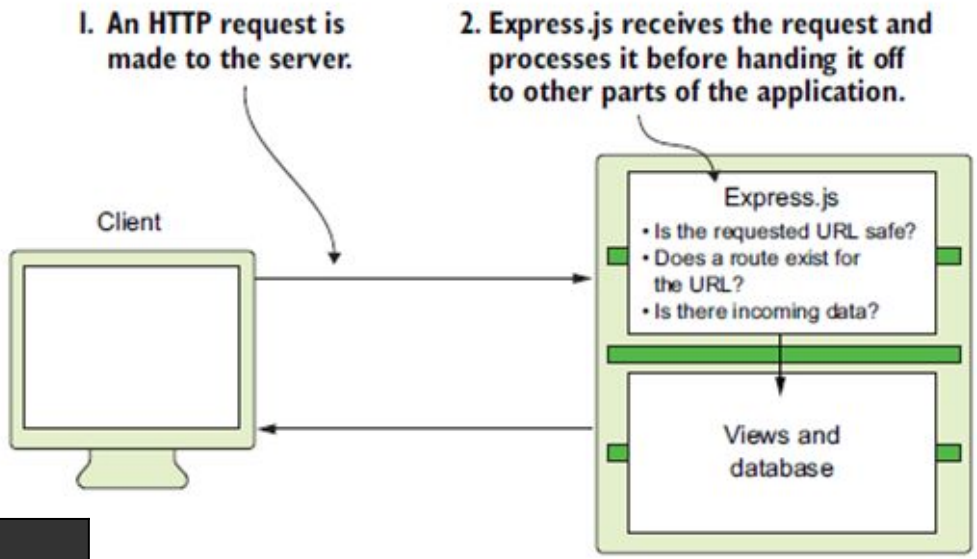
Introduction to Express.js

- A framework makes it easier to build a consistent web application. A web framework is merely a library with a predefined structure
- Express.js is a node.js framework
- Express.js operates through functions known as middleware because they sit between HTTP interaction on the web and the Node.js platform



- *(Routing means assigning functions to respond to users' requests)* Express routers operate as **middleware** (*meaning they have access to both the request and response objects. Hence codes written using Express is shorter and more readable than plain Node.js*)

- Routing in Express.js follows a basic format



app.VERB('path', callback...);

Where VERB is any of the GET, POST, PUT, and DELETE verbs.

Poll 8 (30 Sec)

State whether true or false:

Express.js is a JavaScript framework

1. True
2. False

Poll 8 (Answer)

State whether true or false:

Express.js is a JavaScript framework

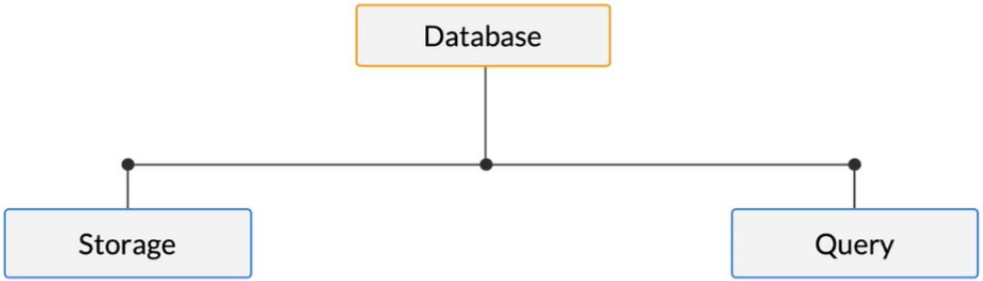
1. True
2. **False**

upGrad

Introduction to Database

Why do you need database?

- Think about the Zomato or Swiggy application. *How do you navigate through the website and the different types of data on it? Also, how is all the data inter-related and relevant when you place an order or add a review on the website?*



Queries

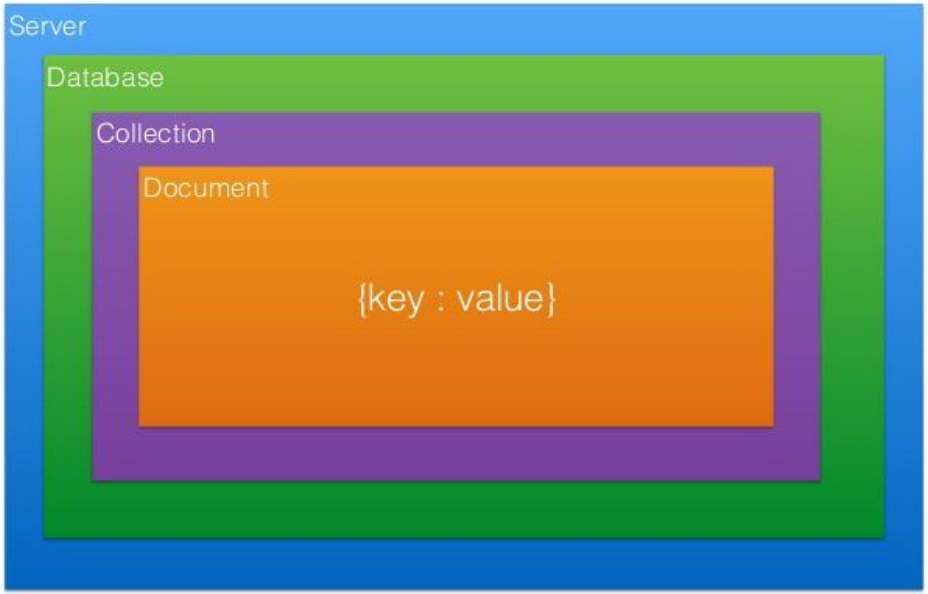

1
Create
Creating/adding/inserting a new record.
Example : Zomato adds a new restaurant on its website
2
Read
Retrieving data.
Example: User can see all the restaurants listed with Zomato.
3
Update
Changing an existing record.
Example: User updates his/her location on Zomato
4
Delete
Deleting an existing record.
Example: Zomato deletes a previously existing restaurant on its website.

upGrad

SQL	NoSQL
Stores information in form of tables with fixed schema	NoSQL databases follow a dynamic schema
SQL databases are generally scaled vertically by increasing the CPU and RAM to increase the power of the existing database	NoSQL databases are more suitable for horizontal scaling, where you keep adding more servers to improve performance
SQL databases are designed to handle advanced querying requirements	NoSQL databases are utilised to handle complex data and they can be scaled as per requirement

NoSQL Database (MongoDB)

Document Oriented Storage



1. MongoDB is a document database
2. Each document database is a set of collections
3. 'Collection' is simply a collection of documents or records
4. Each document is composed of field and value pairs
5. MongoDB documents are similar to JSON objects
6. You would install Compass, which is the official GUI for MongoDB (later in the respective session of MongoDB)

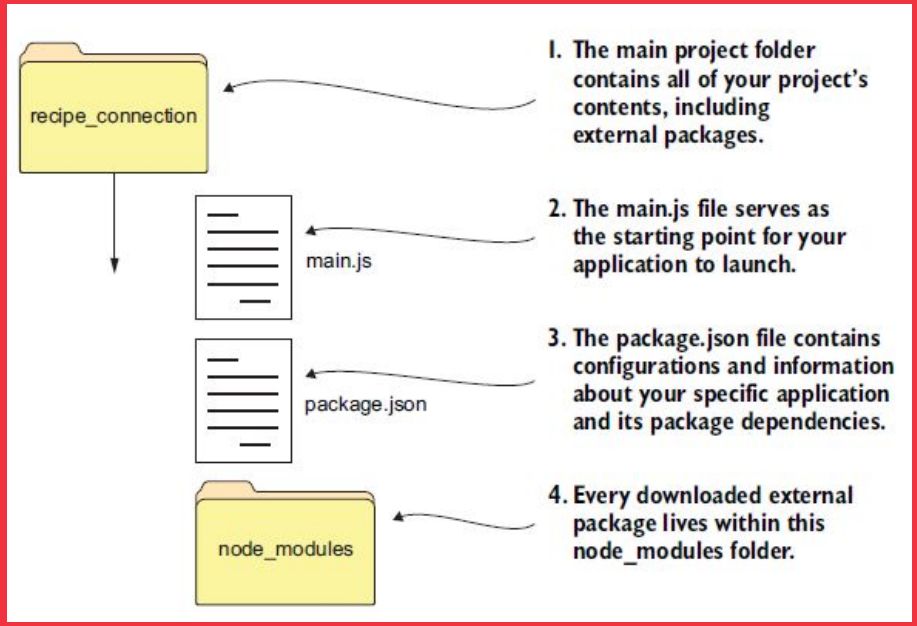
```
{
  name: "sue",          <-- field: value
  age: 26,              <-- field: value
  status: "A",          <-- field: value
  groups: [ "news", "sports" ]   <-- field: value
}
```

upGrad

Hands-on Exercise (10 mins)

1. Create a folder called recipe_connection
2. navigate to your project directory in terminal, and use the **npm init** command to initialise your application.
3. For now, be sure to: **enter your name**, use **main.js** as the **entry point** and press enter to accept all the default options.
4. install cities package **npm install cities --save**
5. Test this new package by adding the lines of code given below to main.js

```
const cities = require("cities");
var myCity = cities.zip_lookup("10016");
console.log(myCity);
```



Refer recipe_connection

Homework

1. Create a Node.js app name it as myFirstApp. Now, create two JavaScript files, *holidays.js* and *printHolidays.js*
    a. *holidays.js* should contain a list of bank holidays in 2021
    b. *printHolidays.js* should print the list of holidays

Hint : use exports module. Do npm init after making the myFirstApp

2. Create a personal calculator which does the following:
    a. calculates percentage of given quantity
    b. finds the profit given selling price and cost price of a product
    c. finds the interest amount on a fixed deposit for given principal, rate and period (use simple interest formula)

upGrad

Doubt Clearance (5 mins)

Key Takeaways

1. You understood the definition of Node.js by breaking it down into parts
2. You learnt to run the JavaScript files with Node.js
3. You understood the difference between modules and packages
4. You saw that node package manager is responsible for managing the external packages in your application
5. You also became familiar with the concept of NoSQL

Tasks to complete after today's session

Homework
MCQs
Coding Questions
Course Project - Checkpoint 1

In the next class, we will discuss...

- Introduction to web servers
- Understand the different HTTP methods and HTTP response codes
- Understand the request-response cycle
- Learn how web servers handle the incoming data
- Understand the architecture of the project

upGrad

#RahoAmbitious

Thank You!