NODE JS

## **What is node.js? How node.js works?**

Node.js is an open-source, cross-platform runtime environment for developing server-side applications written in JavaScript. It is made of Google Chrome’s V8 engine written in multiple platforms of C library i.e., C++ and Libuv. Node.js allows us to run JavaScript on the server.

Node follows Single Threaded with Event Loop Model as it is completely event-driven. Node JS Processing model mainly based on JavaScript Event based model with JavaScript call-back mechanism.

The server starts processing event and when there is a blocking IO operation, it does not wait until it completes and instead registers a call-back function. The server then immediately starts to process another event (maybe another request). When the IO operation is finished, that is another kind of event, and the server will process it by executing the call-back as soon as it has time. So, the server never needs to create additional threads or switch between threads.

## **Node.js work flow:**

Requests >> Node.js server >> Event Queue >> Thread Pool >> Event Loop >> External Resources

Clients send requests to the web server to interact the application. Requests can be non-blocking or blocking through Querying, Deleting, Updating etc.,

Node.js retrieves the incoming requests and adds those requests to the Event Queue.

The requests are then passed one-by-one through the Event Loop. It checks if the requests are simple enough to not require any external resources

Event Loop processes simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients.

A single thread from the Thread Pool is assigned to a single complex request. This thread is responsible for completing a particular blocking request by accessing the external resources, such as database, file system, etc.

Once, the task is carried out completely, the response is sent to the Event Loop that in turn sends that response back to the Client

## **Advantages?**

Handling multiple concurrent client requests is fast and easy with the use of Event Queue and Thread Pool, the Node.js server enables efficient handling of a large number of incoming requests.

No need for creating multiple threads Event Loop handles all requests one-by-one, so there is no need to create multiple threads. Instead, a single thread is sufficient to handle a blocking incoming request.

All these advantages contribute to making the servers developed using Node.js much faster and responsive when compared to those developed using other server development technologies.

## **Why node is single threaded?**

Node.js is a single threaded application with event looping for async processing. The biggest advantage of doing async processing on a single thread is, can handle more concurrent client’s requests with ease. Eliminates the need of creating more threads, because of the Event loop.

## **Explain Below:**

**Event Driven:** Events are actions generated by the user like a click. Event-driven programming is a programming paradigm in which the flow of the program is determined by events. An event-driven program performs actions in response to events.

**NPM** stands for Node Package Manager. Online repositories for node.js packages/modules. Command line utility to install packages, do version management and dependency management of Node.js packages.

|  |  |
| --- | --- |
| **package.json** | **package.lock.json** |
| It contains basic information about the project. | It describes the exact tree that was generated to allow subsequent installs to have the identical tree. |
| It is mandatory for every project. | It is automatically generated for those operations where npm modifies either node\_modules tree or package.json. |
| It records important metadata about the project. | It allows future devs to install the same dependencies in the project. |
| It contains information such as name, description, author, script, and dependencies. | It contains the name, dependencies, and locked version of the project. |

angular

## **What is Angular Framework & TypeScript ?**

Angular is a TypeScript-based open-source front-end platform that makes it easy to build applications within web/mobile/desktop. TypeScript is a superset of JavaScript created by Microsoft that adds optional types, classes, async/await, and many other features, and compiles to plain JavaScript. Angular built entirely in TypeScript and used as a primary language.

## **What is Angular Application Workflow?**

angular.json => main.ts => app.module.ts => app.component.ts => app-root => index.html;

angular.json file will contain all the configurations of the app. While building the app, the builder looks at this file to find the entry point of the application. Inside the build section, the main property of the options object defines the entry point of the application (default case is main.ts).

The main.ts file creates a browser environment for the application to run, and, along with this, it also calls a function called bootstrapModule, which bootstraps the application. AppModule is getting bootstrapped by platformBrowserDynamic().bootstrapModule(AppModule) which is existing in main.ts.

The AppModule is declared in the app.module.ts file. This module contains declarations of all the components. This AppModule bootstraps the AppComponent which is defined in app.component.ts file. This file interacts with the webpage and serves data to it. Each component is declared with three properties: 1). Selector - used for accessing the component; 2). Template/TemplateURL - contains HTML of the component; 3). StylesURL - contains component-specific stylesheets

After this, Angular calls the index.html file. This file consequently calls the root component that is app-root. The root component is defined in app.component.ts. The HTML template of the root component is displayed inside the <app-root> tags. This is how every angular application works.

## **Explain Below:**

**Ivy** Compiler is default in Angular 9 onwards which provides huge improvements in application with better debugging, fast testing, Improved type checking, Improved build errors & Improved build times. Ivy has very big focus on tree-shaking (compiler looks at code and figure out exactly which libraries are required in project and eliminates the unused code. Also improves application loading time due to reducing the bundle size.

**Pipes** are simple functions designed to accept an input value, process, and return as an output, a transformed value in a more technical understanding. The parameterized pipe can be created by declaring the pipe name with a colon ( : ) and then the parameter value. If the pipe accepts multiple parameters, separate the values with colons.

**HttpClient:** Most of the Front-end applications communicate with backend services over HTTP protocol using either XMLHttpRequest interface or the fetch() API. Angular provides a simplified client HTTP API known as HttpClient which is based on top of XMLHttpRequest interface. This client is available from @angular/common/http package.

**Dependency Injection** (DI) – Angular makes use of DI to provide required dependencies to new components. Typically, dependencies required by a component are services. A component’s constructor parameters tell Angular about the services that a component requires. So, a dependency injection offers a way to supply fully-formed dependencies required by a new instance of a class.

**AOT** The Angular components and templates cannot be understood by the browser directly. Due to those Angular applications require a compilation process before they can run in a browser. For example, In AOT compilation, both Angular HTML and TypeScript code converted into efficient JavaScript code during the build phase before browser runs it. Faster Rendering, Fewer Ajax Requests, Quick detection of template errors are the advantages of AOT.

**@Input & @Output** Both are used to pass the different types of data form parent to child component and child to parent component. @Input decorator is used to pass data (property binding) from parent to child component (should be annotated with @Input decorator). @Output decorator is used to pass the data from child to parent component. @Output binds a property of EventEmitter class.

**Observables and Promises** help us work with asynchronous functionality in JavaScript. Promises deal with one asynchronous event at a time, while observables handle a sequence of asynchronous events over a period. The first difference is that the Promise is eager, whereas the Observable is lazy.

**RxJS** (Reactive Extensions for JavaScript) is a library for composing asynchronous and callback-functions using Observables.

**Data-binding** is a feature in angular, which provides a way to communicate between the component(Model) and its view(HTML template). Data-binding can be done in two ways, one-way binding and two-way binding. In Angular, data from the component can be inserted inside the HTML template. In one-way binding, any changes in the component will directly reflect inside the HTML template but, vice-versa is not possible. Whereas it is possible in two-way binding.

**String interpolation** is a one-way data-binding technique that outputs the data from TypeScript code to HTML view and it uses the double curly braces {{ }} to display data from the component. Angular automatically runs the expression written inside the curly braces.

Using **property binding**, we can bind the DOM properties of an HTML element to a component's property. Property binding uses the square brackets [ ] syntax.

## **What is lifecycle hooks available?**

Angular calls lifecycle hook methods on directives and components as it creates, changes, and destroys them.

* Constructor: While creating a component/directive by calling new keyword on the class.
* ngOnChanges: Responds when Angular sets/resets data-bound input properties.
* ngOnInit: While initialization of the directive/component after Angular first displays the data-bound properties happens.
* ngDoCheck: Detect & act upon changes that Angular can't or won't detect on its own.
* ngAfterContentInit: Response after Angular projects external content into the component's view.
* ngAfterContentChecked: Response after Angular checks the content projected into the component.
* ngAfterViewInit: Response after Angular initializes the component's views and child views.
* ngAfterViewChecked: Response after Angular checks the component's views and child views.
* ngOnDestroy: Clean-up phase just before Angular destroys the directive/component.

## **Version Changes:**

Ang 1: [Oct-10] In JavaScript, Supports MVC, doesn’t support mobile;

Ang 2: [Sep-16] Complete rewritten, Written in TypeScript, support’s mobile

Ang 3: Skipped

Ang 4: [Mar-17] Intro of HttpClient

Ang 5: [Nov-17] Replaced @angular/http with @angular/common/http library.

Ang 6: [May-18] Available RxJS

Ang 7: [Oct-18] drag and drop interfaces => @angular/cdk/drag-drop

Ang 8: [May-19] Intro of Ivy compiler as opt-in feature; JIT is default upto 8;

Ang 9: [Nov-19] Ivy as default compiler

Ang 10: [Jun-20] TypeScript 3.9,

Ang 11: [Nov-20] Typescript 4.0., Deprecating support of IE9&IE10 and IE Mobile, **Hot Module Replacement** (HMR) helps in replacing the modules without a browser refresh.

Ang 12: [May-21] TypeScript 4.2, New Dev Tools, Nullish Coalescing, Ivy Everywhere.

Ang 13: [Nov-21] Deprecated for IE11, TypeScript 4.4 Rxjsl 7.4 is now as default for new apps.

## **What are Angular Router, Router Outlet, Router Link & Activatedroute ?**

Angular Router is a mechanism in which navigation happens from one view to the next. The RouterOutlet is a directive, and it acts as a placeholder spot in the template where the router should display the component’s output. The RouterLink is a directive on the anchor tags give the router control over those elements. ActivatedRoute contains the information about a route associated with a component loaded (URL & Its params).

## **What are the key components of Angular?**

**Components** are the basic building blocks, which control a part of the UI for an application. A component is defined using the @Component decorator. Every component consists of three parts, a selector: to represent in the UI, a templateUrl: to load the view for the component and a styleUrls: to apply the styles for the component. ng g c test / ng generate component test will generate the test component.

**Module** is a place where we can group components, directives, services, and pipes. Module decides whether the components, directives, etc can be used by other modules, by exporting or hiding these elements. Every module is defined with a @NgModule decorator. By default, modules are of two types: Root Module & Feature Module. Every application can have only one root module whereas, it can have one or more feature modules. A root module imports BrowserModule, whereas a feature module imports CommonModule. ng g m test-module / ng generate module test-module;

**Template** is a HTML view where you can display data by binding controls to properties of an Angular component. We can define it inline using the template property, or we can define the template in a separate HTML file and link to it in the component metadata using the @Component decorator's templateUrl property.

**Services**: The main objective of a service is to share data, functions with different components of an Angular application. A service is defined using a @Injectable decorator. A function defined inside a service can be invoked from any component or directive. ng g s test-service / ng generate service test-service;

**Metadata**: Metadata Provides additional information about the component to the Angular. Angular uses this information to process the class. Metadata is used to decorate a class so that it can configure the expected behaviour of the class. The metadata is represented by decorators. The purpose of these decorators is to accept a metadata object that provides relevant information about the component.

**Class decorators:** e.g. @Component and @NgModule;

**Method decorators:** Used for methods inside classes, e.g. @HostListener

**Parameter decorators:** Used for parameters inside class constructors, e.g. @Inject, Optional

**Property decorators:** Used for properties inside classes, e.g. @Input and @Output;

## **Directive and its types**

Directives are classes that add additional behaviour to elements in Angular applications. They execute whenever the Angular compiler finds them in the DOM. Directive is a class in Angular that is declared with a @Directive decorator.

**Component directives**: Instead of @Directive decorator we use @Component decorator to declare these directives. These directives have a view, a stylesheet and a selector property.

**Structural directives**: These directives are generally used to manipulate DOM elements. Every structural directive has a ‘ \* ’ sign before them. We can apply these directives to any DOM element. Ex: \*ngIf & \*ngFor;

**Attribute Directives**: These directives are used to change the look and behaviour of a DOM element. Let’s understand attribute directives by creating one: ng g directive blue Background;

Javascript

## **Synchronous vs Asynchronous**

Synchronous execution usually refers to code executing in sequence. In synchronous, the program is executed line by line means one line at a time. Execution waits until task finishes to run next line of code.

Asynchronous execution refers to execution that doesn’t run in the sequence. In async the program doesn’t wait for the task to complete and can move on to the next task. Instead of the code reading top to bottom procedurally, async programs may execute different functions at different times based on the order and speed like http requests or file system reads happen.

## **Blocking vs Non-blocking**

Blocking refers to operations that block further execution until that operation finishes while non-blocking refers to code that doesn’t block execution. Blocking methods execute synchronously while non-blocking methods execute asynchronously.

## **Callbacks** A callback is a function passed as an argument into another function, which can then be invoked (called back) inside the outer function to complete other action at a convenient time. The invocation may be immediate (sync callback) or it might happen at a later time (async callback). An async callback may be called when an event happens or when a task completes. It prevents blocking by allowing other code to be executed in the meantime.

## **Explain Below:**

**Middleware** is a function that has the access to the Request, Response, Next() in the application’s request-response cycle. ‘next’ is a parameter which is passed in middleware function to pass control to next middleware. Middleware can handle error handling in Express. This can be done by passing one extra error-handling (err) parameter in middleware function.

**EventEmitter** is a Node.js class that includes all the objects that are basically capable of emitting events. This can be done by attaching named events that are emitted by the object using an eventEmitter.on() function. This object exposes the on and emit methods. **emit** is used to trigger an event. **on** is used to add a callback function that's going to be executed when the event is triggered.

**Input/Output**, I/O refers primarily to the program’s interaction with the system’s disk and network. Examples of I/O operations include reading/writing data from/to a disk, making HTTP requests, and talking to databases.

**Var** keyword is function scoped which can access within a function. **Let** & **Const** both are blocks scoped which can access within innermost block. Const is immutable in JavaScript. You cannot change or reassign its value once it is declared.

**Promise** is a good way to handle asynchronous operations. It is used to find out if the asynchronous operation is successfully completed or not. A promise may have one of three states. *Pending, Fulfilled & Rejected*. The Promise() constructor takes a function as an argument. The function also accepts two functions resolve() and reject(). If the promise returns successfully, the resolve() function is called. And, if an error occurs, the reject() function is called.

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Freeze** | **Seal** | **PreventExtns** |
| ADD | No | No | No |
| InnerADD | Yes | Yes | Yes |
| DELETE | No | No | Yes |
| InnerDELETE | Yes | Yes | Yes |
| READ/Inner | Yes | Yes | Yes |
| EDIT | No | Yes | Yes |
| InnerEDIT | Yes | Yes | Yes |
|  |  |  |  |

**Hoisting** is JavaScript's default behaviour of moving declarations to the top.

A **lexical** scope in JavaScript means, that a variable defined outside a function can be accessible inside of function.

The **THIS** keyword refers to different objects depending on how it is used:

In an object method, **this** refers to the object. Alone, **this** refers to the global object. In a function, **this** refers to the global object. In a function, in strict mode, **this** is undefined. In an event, **this** refers to the element that received the event.

**Arrow functions** allow us to write shorter function syntax. If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword. if we have only one parameter, you can skip the parentheses as well. Note: This works only if the function has only one statement.

A **prototype** allows us to use properties and methods on an object even if the properties and methods do not exist on the current object.

**Prototype:** In JavaScript, every object holds a reference to a superior object from which it inherits its properties. This superior object is called a prototype and this behaviour is referred to as prototypal inheritance. The prototypal inheritance happens via a prototype chain. JavaScript tends to search the availability of the property in the object itself. In case the property is not available in the object, JavaScript moves a step higher in the prototype chain and searches for the property in the object’s prototype. In case the property is available in the prototype, then JavaScript makes use of the property from the prototype; else, it steps up the prototype chain till it is able to find the property.

A **closure** gives you access to an outer function's scope from an inner function. A **closure** is a function that accesses outside variables in inside function’s scope.

function outer(){

let counter = 0;

function inner(){

counter ++;

return counter;

}

return inner;

}

let myF = outer(); console.log(myF());

Express & rest

## **What is ExpressJs and its features ?**

Express Js is a framework for develop web and mobile applications which is open-sourced, light-weight and faster. Following are the features of Express.js:

* It allows middleware to respond HTTP Requests
* It defines routing table which is used to achieve action based on HTTP Method and URL.
* It allows dynamically generate the HTML Pages.
* It follows MVC architecture for web application.

## **What are RESTful Web Services?**

REST (Representational State Transfer) is a stateless client-server architecture used for developing applications that are accessible over the web. It can be defined as the web service that uses HTTP methods for implementing the REST architecture. **Advantages**:

* Platform independent.
* Support different formats such as JSON, XML, HTML, etc.
* Supports different languages and executed on any platform.
* Lightweight, manageable, scalable, and reusable.
* Faster and provide better performance.

## **CORS, Authentication & Authorization & HTTP Status Codes:**

Cross-origin resource sharing (CORS) is a security feature that restricts cross-origin HTTP requests from scripts running in the browser. We need to enable CORS support to access the resources by the HTTP requests.

|  |  |
| --- | --- |
| **Authentication** | **Authorization** |
| It’s a process to find the identity of user by entered credentials. | It’s a process to find the accessing levels on respective user after authentication. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1xx Informational** | **2xx Success** | **3xx Redirection** | **4xx Client Error** | **5xx Server Error** |
| 100 Continue | 200 Ok | 300 Multiple Choices | 400 Bad Request | 500 Internal Server Error |
| 102 Processing | 202 Accepted | 301 Moved Permanently | 401 Unauthorized | 502 Bad Gateway |
|  | 204 No Content | 302 Found | 403 Forbidden | 503 Service Unavailable |
|  |  |  | 404 Not Found | 504 Gateway Timeout |

## **Explain Below:**

**Routing** defines how an application responds over client request to a particular endpoint (URI). Endpoint is specially a path and any one of HTTP request methods (GET, POST, etc).

**Scaffolding** is a tool, which set up all required public directory, add middleware, create separate route file etc (set up skeleton for web application). So that we can directly get started building our application. Yeoman is scaffolding tool built for Node.js.

The **LocalStorage** object allows you to save key/value pairs in the browser. The localStorage object stores data with no expiration date. The data is not deleted when the browser is closed, and are available for future sessions. The **SessionStorage** Object which stores data for one session. The data is deleted when the browser window is closed.

**Cookies** are data, stored in small text files, on your computer. When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user. Cookies were invented to solve the problem "how to remember information about the user". When a user visits a web page, his/her name can be stored in a cookie. Next time the user visits the page, the cookie "remembers" his/her name. Cookies are saved in name-value pairs

database

## **SQL vs NoSQL:**

|  |  |
| --- | --- |
| **SQL** | **NoSQL** |
| SQL is generally used in Relational Database Management system. | NoSQL is used for Non-relational, distributed database system. |
| Structured data can be stored in tables. | Using JSON data, un-structured data can be stored |
| The Schemas are Static. | The Schemas are Dynamic. |
| Schemas are rigid and bound to relationships. | Schemas are non-rigid, they are flexible. |
| Helpful to design complex queries. | No interface to prepare complex query. |

## **What is MongoDB, Advantages & it’s features?**

MongoDB is an open-source NoSQL database written in C++ language. It uses JSON-like documents with optional schemas. It provides easy scalability and is a cross-platform, document-oriented database. MongoDB works on the concept of Collection and Document.

* MongoDB supports field, range-based, string pattern matching type queries. for searching the data in the database
* MongoDB support primary and secondary index on any fields
* MongoDB basically uses JavaScript objects in place of procedures
* MongoDB uses a dynamic database schema
* MongoDB is very easy to scale up or down
* MongoDB has inbuilt support for data partitioning (Sharding).

**Features**:

* **Indexing**: It supports generic secondary indexes and provides unique, compound, geospatial, and full-text indexing capabilities as well.
* **Aggregation**: It provides an aggregation framework based on the concept of data processing pipelines.
* Special collection and index types: It supports time-to-live (TTL) collections for data that should expire at a certain time
* File storage: It supports an easy-to-use protocol for storing large files and file metadata.
* **Sharding**: Sharding is the process of splitting data up across machines.

Security - JWT

## **What is JWT?**

JWT securely transmit trusted information between parties in a compact way. This information can be verified when it’s been digitally signed, it also holds user’s claim info like authorization, so the service provider does not need to access the database. It can be signed or encrypted easily. JWT tokens hold claims that can be encoded as a JSON object and are digitally. JWT is divided into 3 parts:

**Header** is represented as a JSON object which is encoded to a baseURL. **Payload** contains actual data to be transferred using token. It is also called claims. There are 3 types of Payload: Registered, Public, Private. **Signature** is used for verification of the message that was not changed along the way. It is created for using the encoded header, payload, and the algorithm specified in the header. **Advantages**:

* Good Performance: JWT itself contains all information, so we don't have to go to Authorization server to get the user's information to verify whether user is valid or not.
* Portable: Allow to use multiple backends with single access token.
* It is Very Mobile Friendly, because cookies are not required.
* JWT contains expiration date as a claim that can be used to determine when the access token is going to expire.
* It's very secure way to validate the user information, as it's digitally signed.
* It's digitally signed, so if anyone updates it the server will know about it.
* It is most suitable for Microservices Architecture.
* It has other advantages like specifying the expiration time.

## **What is Workflow of JWT?**

* Customers sign in by submitting their credentials to the provider.
* Upon successful authentication, it generates JWT containing user details and privileges for accessing the services and sets the JWT expiry date in payload.
* The server signs and encrypts the JWT if necessary and sends it to the client as a response with credentials to the initial request.
* Based on the expiration set by the server, the customer/client stores the JWT for a restricted or infinite amount of time.
* The client sends this JWT token in the header for all subsequent requests.
* The client authenticates the user with this token. So we don't need the client to send the user name and password to the server during each authentication process, but only once the server sends the client a JWT.