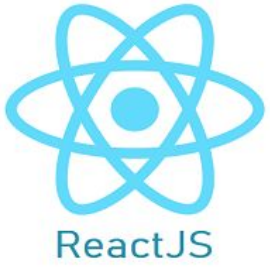
MERN: UNIT-4 Node.js, Express.js and MongoDB

Prepared By,
M.Gouthamm, Asst.Prof, CSE, MRUH

Topics Covered

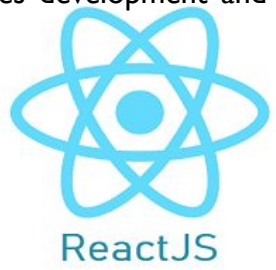
1. **Node.js**
2. **Express.js**
3. **RESTful API with Express.js**
4. **MongoDB**



Node.js

What is Node.js?

- Node.js is a server-side platform built on **Google Chrome's JavaScript Engine** (V8 Engine).
- Node.js was developed by **Ryan Dahl in 2009** and its latest version is v0.10.36.
- Node.js is a platform built on **Chrome's JavaScript runtime** for easily building fast and scalable network applications.
- Node.js uses an **event-driven, non-blocking I/O model** that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.
- Node.js is an open source, **cross-platform runtime environment and library that is used for running web applications outside the client's browser.**
- It is used for **server-side programming,** and primarily deployed for non-blocking, event-driven servers, such as traditional web sites and back-end API services, but was originally designed with real-time, push-based architectures in mind. Every browser has its own version of a JS engine, and node.js is built on Google Chrome's V8 JavaScript engine. Sounds a bit complicated, right?
- In simple terms, what this means is that **entire sites can be run using a unified 'stack'**, which makes development and maintenance quick and easy, allowing you to focus on meeting the business goals of the project.

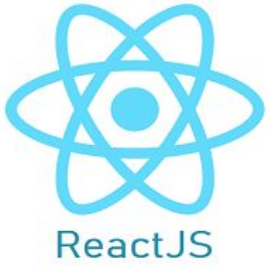


Node.js

- An important thing to understand about Node.js is that it is actually neither a framework or a library - as with traditional application software -, but JavaScript a runtime environment.

Node.js = Runtime Environment + JavaScript Library

- A runtime environment (sometimes shortened to RTE) contains Web API's that a developer can access to build a code, and a JavaScript engine that parses that code. This makes it lightweight, flexible and easy to deploy, all features that will help to optimize and speed up your application project.

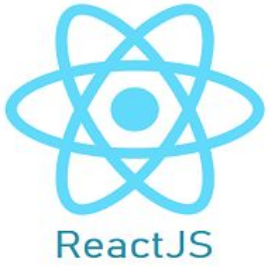


Node.js

Features of Node.js

Following are some of the important features that make Node.js the first choice of software architects.

- **Asynchronous and Event Driven –** All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
- **Very Fast –** Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- **Single Threaded but Highly Scalable –** Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- **No Buffering –** Node.js applications never buffer any data. These applications simply output the data in chunks.



Node.js

Who uses Node.js

Node.js is widely used by developers and organizations across various industries for a wide range of applications.

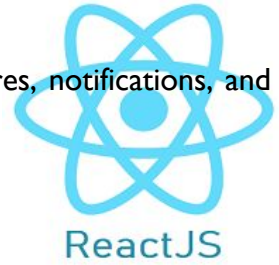
Here are some examples of who uses Node.js:

1. Technology Companies: Many technology companies, both large and small, leverage Node.js for building web applications, APIs, and backend services. This includes companies like **Netflix, Uber, LinkedIn, PayPal, eBay, and Microsoft.**

2. Startups: Node.js is popular among startups due to its scalability, performance, and developer productivity. Startups often use Node.js to rapidly develop and deploy innovative web applications and services.

3. E-commerce Platforms: E-commerce platforms use Node.js to power their websites, handle real-time data updates, and manage inventory and order processing. Examples include **Walmart, Etsy, and Shopify.**

4. Social Media Platforms: Social media platforms utilize Node.js for building real-time messaging features, notifications, and social networking functionalities. Examples include **Twitter, Pinterest, and Snapchat.**



Node.js

5. Content Management Systems (CMS): CMS platforms use Node.js for building custom themes, plugins, and extensions, as well as for managing content and user interactions. Examples include **WordPress and Ghost.**

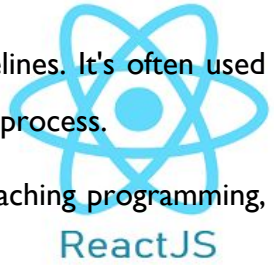
6. Gaming Industry: Node.js is increasingly used in the gaming industry for building multiplayer games, game servers, and real-time gaming experiences. Examples include **Roblox and Ubisoft.**

7. IoT (Internet of Things): Node.js is used in IoT applications for handling device communication, data processing, and remote monitoring. It's well-suited for IoT projects due to its lightweight nature and event-driven architecture.

6. Cloud Computing: Cloud service providers and platforms leverage Node.js for building cloud-native applications, serverless functions, and APIs. Examples include **AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions.**

7. DevOps: DevOps teams use Node.js for building automation scripts, monitoring tools, and CI/CD pipelines. It's often used alongside other technologies like **Docker and Kubernetes** to streamline the development and deployment process.

8. Education and Research: Node.js is used in educational institutions and research organizations for teaching programming, conducting experiments, and building educational software.



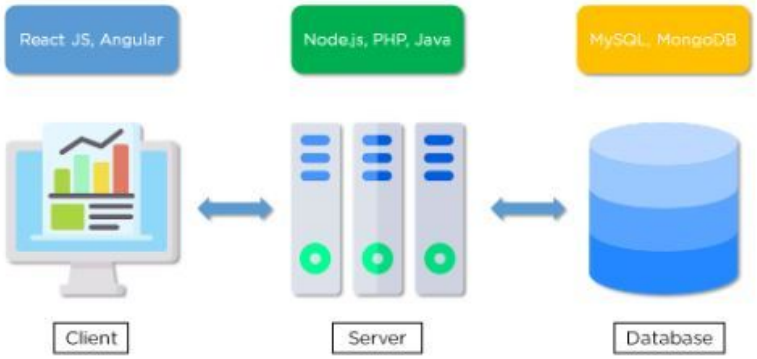
Node.js

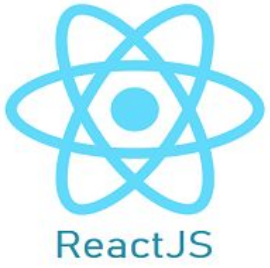
Node.js Web Application Architecture

- Node.js is a **JavaScript-based platform** that is mainly used to create **I/O-intensive web applications** such as chat apps, multimedia streaming sites, etc. It is built on **Google Chrome's** V8 JavaScript engine.
- A web application, as you may already know, is a program that runs on a server and is rendered by a client browser, using the internet to access all the resources of that application.

A typical web application consists of the following components:

- **Client:** A client refers to the user who interacts with the server by sending out requests.
- **Server:** The server is in charge of receiving client requests, performing appropriate tasks, and returning results to the clients. It serves as a bridge between the front-end and the stored data, allowing clients to perform operations on the data.
- **Database:** A database is where a web application's data is stored. Depending on the client's request, the data can be created, modified, and deleted.





Node.js

Client

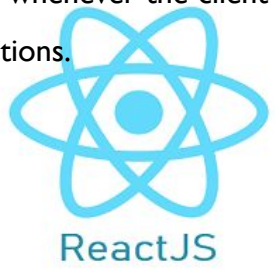
- The user interacts with the front-end part of a web application. The front-end is usually developed using languages like **HTML and CSS styles**, along with extensive usage of **JavaScript-based frameworks like ReactJS and Angular**, which help with application design.

Server

- The server is responsible for **taking the client requests, performing the required tasks, and sending responses back to the clients**. It acts as a **middleware** between the front-end and stored data to enable operations on the data by a client. **Node.js, PHP, and Java are the most popular technologies** in use to develop and maintain a web server.

Database

- The database **stores the data for a web application.** The data can be **created, updated, and deleted** whenever the client requests. **MySQL and MongoDB** are among the most popular databases used to store data for web applications.

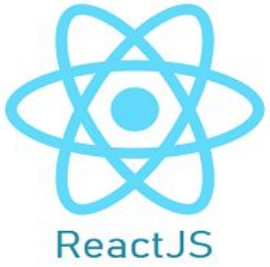


Node.js

- **Node.js Server Architecture:** To manage several concurrent clients, Node.js employs a **"Single Threaded Event Loop" design**. The JavaScript **event-based model and the JavaScript callback mechanism** are employed in the Node.js Processing Model.

It employs two fundamental concepts:

- **Asynchronous model**
- **Non-blocking of I/O operations**



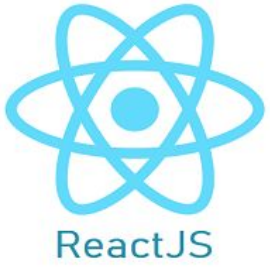
Node.js

These features enhance the scalability, performance, and throughput of Node.js web applications.

Components of the Node.js Architecture:

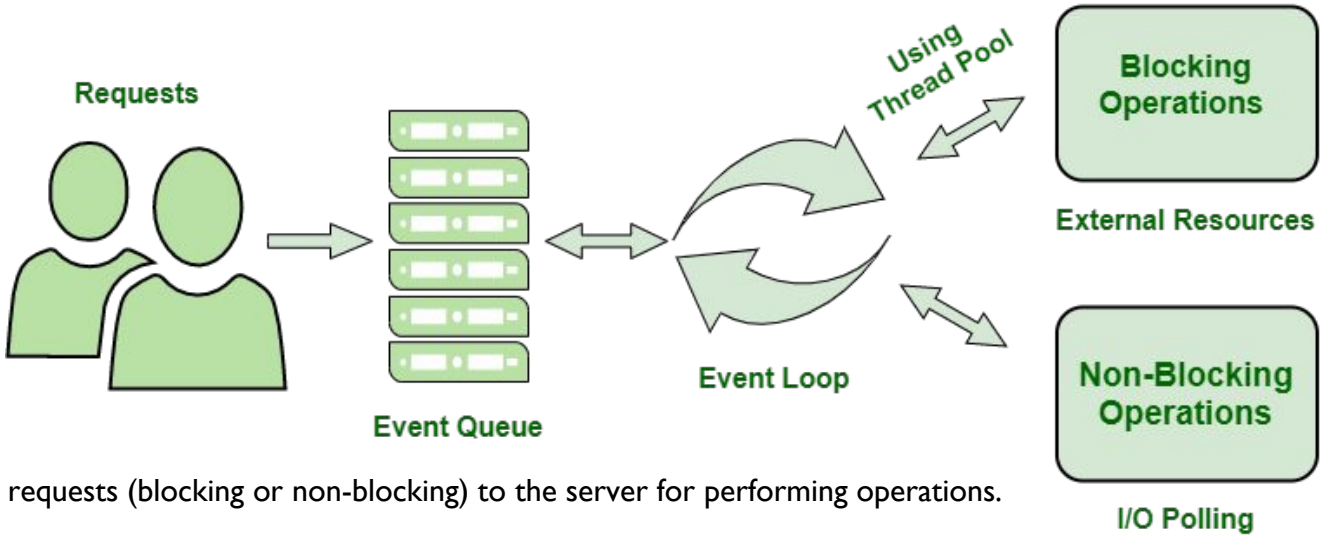
1. **Requests:** Depending on the actions that a user needs to perform, the requests to the server can be either blocking (complex) or non-blocking (simple).
2. **Node.js Server:** The Node.js server accepts user requests, processes them, and returns results to the users.
3. **Event Queue:** The main use of Event Queue is to store the incoming client requests and pass them sequentially to the Event Loop.
4. **Thread Pool:** The Thread pool in a Node.js server contains the threads that are available for performing operations required to process requests.
5. **Event Loop:** Event Loop receives requests from the Event Queue and sends out the responses to the clients.
6. **External Resources:** In order to handle blocking client requests, external resources are used.

   They can be of any type (computation, storage, etc).



Node.js

Workflow of Nodejs Server Architecture :

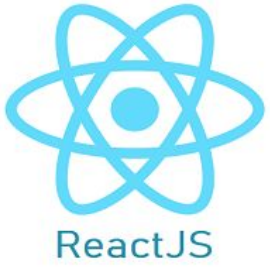


- Users send requests (blocking or non-blocking) to the server for performing operations.
- The requests enter the Event Queue first at the server-side.
- The Event queue passes the requests sequentially to the event loop. The event loop checks the nature of the request (blocking or non-blocking).
- Event Loop processes the non-blocking requests which do not require external resources and returns the responses to the corresponding clients
- For blocking requests, a single thread is assigned to the process for completing the task by using external resources.
- After the completion of the operation, the request is redirected to the Event Loop which delivers the response back to the client.

Node.js

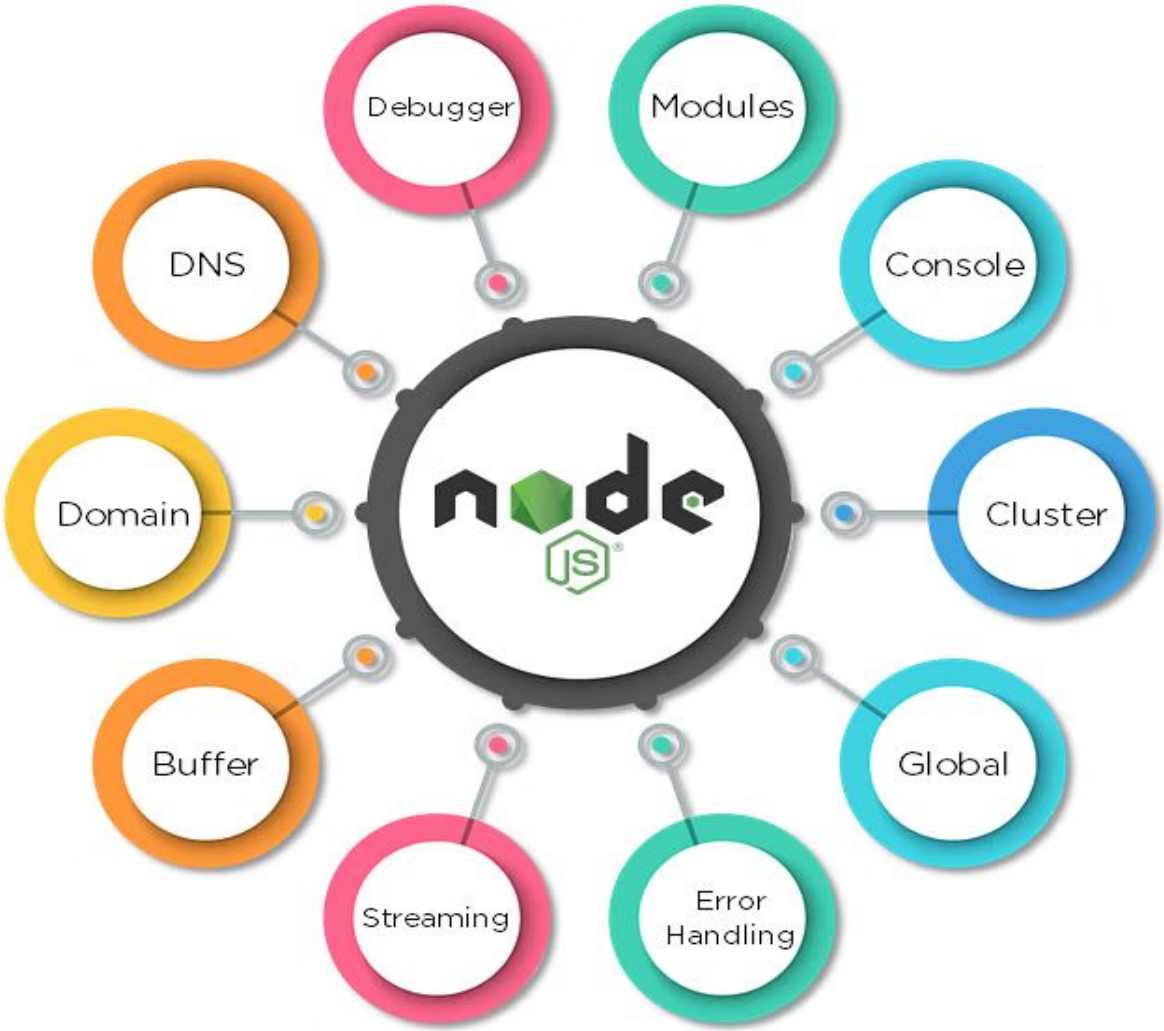
Advantages:

- The Node.js server can efficiently handle a high number of requests by employing the use of Event Queue and Thread Pool.
- There is no need to establish multiple threads because Event Loop processes all requests one at a time, therefore a single thread is sufficient.
- The entire process of serving requests to a Node.js server consumes less memory and server resources since the requests are handled one at a time.



Node.js

Parts of Node.js

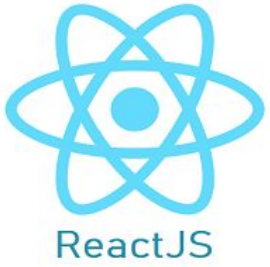


ReactJS

Node.js

What is JavaScript?

- JavaScript is the programming language that **underpins (gives support to)** Node.js. It is a high-level and multi-paradigm language characterized by **'curly bracket' syntax, dynamic typing, prototype-based object orientations and first class functions.**
- These features mean JavaScript can **convert a static webpage to an interactive one,** adding features such as search boxes, embedded videos or news feed refresh tools. These are features that aim to improve user experience by encouraging intuitive engagement.



Node.js

Node.js modules

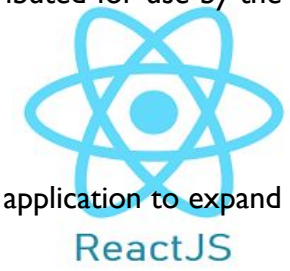
- Node.js features multiple 'modules' that are kept within individual contexts so they do not interfere with other modules or pollute the global scope of node.js. This is crucial for open source softwares.
- A module in Node.js is a functionality - either simple or complex - which is organized into JavaScript files and reusable throughout the Node.js application.

There are three types of module within Node.js: **Core Modules, Local Modules and Third Party Modules.**

1. **Core Modules** include the basic, bare-bones functionalities of Node.js. They load automatically when a node process starts and are part of Node.js's binary distribution.

2. **Local Modules** are modules that are created within the Node.js application. They include different and additional functionalities in separate files and folders to the core functionality package. Local modules can also be packaged and distributed for use by the wider Node.js community.

3. **A third party module** is an existing code written by a third party, and can be imported into your Node.js application to expand or add different features and functions.



Node.js

Node Package Manager(npm)

- npm is the **package manager** for node. The npm Registry is a public collection of packages of open-source code for Node.js, front-end web apps, mobile apps, robots, routers, and countless other needs of the JavaScript community. **npm allows us to access all these packages and install them locally.**

How to use npm?

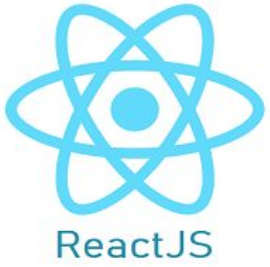
There are two ways to install a package using npm: globally and locally.

- **Globally:** This method is generally used to install development tools and CLI based packages. To install a package globally, use the following code.

```
npm install -g <package-name>
```

- **Locally:** This method is generally used to install frameworks and libraries. A locally installed package can be used only within the directory it is installed. To install a package locally, use the same command as above without the -g flag.

```
npm install <package-name>
```



Node.js

How to create a basic Node.js Application?

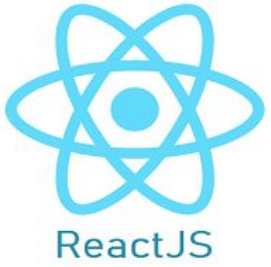
- To create a basic Hello World application in Node.js, save the following single line JavaScript as hello.js file.

```
console.log("Hello World");
```

- Open a powershell (or command prompt) terminal in the folder in which hello.js file is present, and enter the following command –

```
PS D:\nodejs> node hello.js
Hello World
```

- The "Hello World" message is displayed in the terminal.



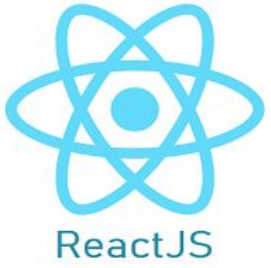
Node.js

To create a "Hello, World!" web application using Node.js, save the following code as hello.js:

Example: hello.js

```
const http = require('node:http');

const hostname = '127.0.0.1';
const port = 4000;

const server = http.createServer((req, res) => {

 // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/html
   res.writeHead(200, {'Content-Type': 'text/html'});

   // Send the response body as "Hello World"
   res.end('<h2 style="text-align: center;">Hello World</h2>');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Cmd: node hello.js



Node.js

How to Install Third-Party Packages in Node.js?

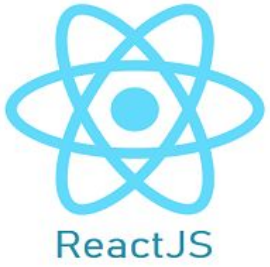
- Node packages are of three types - Core, Local and Third-Party packages.

- Popular third-party modules include: Mongoose, Multer, Body Parser, JSON Web Token etc.

- To install a third-party package, we need to use NPM package manager.

- The following command installs Mongoose package – a MongoDB object modelling tool −

```
npm install mongoose
```

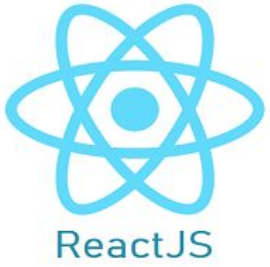


Node.js

- In Node.js, "module" and "package" refer to different concepts, and they play distinct roles in the Node.js ecosystem.

Module:

- A module in Node.js is a reusable piece of code that encapsulates related functionality.
- It can be a single file or a collection of files organized in a directory.
- Modules help in organizing code into smaller, manageable pieces, promoting modularity and code reuse.
- To make functions, variables, or objects from a module available in another module, you use the **exports object or module.exports.**
- Modules are used to structure and break down large applications into smaller, more maintainable parts.



Node.js

Exporting from Modules:

- You can export values from a module using module.exports or exports.

Example// math.js

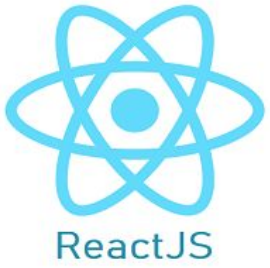
```
function add(a, b) {
    return a + b;
}
module.exports = { add };
```

Importing Modules:

- You can import modules using the require function.

Example:

```
const { add } = require('./math');
console.log(add(2, 3)); // Output: 5
```



Node.js

Package:

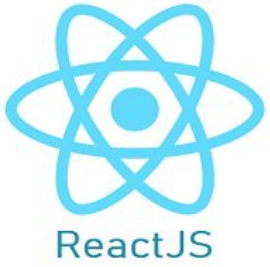
- A package in Node.js is a way of organizing related modules into a directory structure.
- It contains a package.json file, which includes metadata about the package (such as name, version, dependencies, etc.).
- Packages can be published to the npm (Node Package Manager) registry, allowing others to easily install and use them in their projects.
- npm is the default package manager for Node.js, and it simplifies the process of installing, managing, and sharing packages.

Example of a simple package (my-package):

```
my-package/
    ├── package.json
    ├── myModule.js
    └── app.js
```

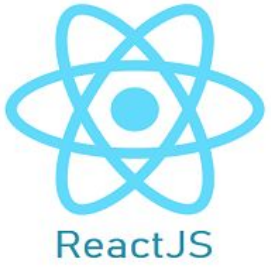
package.json:

```
{
        "name": "my-package",
        "version": "1.0.0",
        "main": "app.js",
        "dependencies": {
            // dependencies go here
        }   }
```



Express.js

- Express provides a minimal interface to build our applications. It provides us the tools that are required to build our app. It is flexible as there are **numerous modules available on npm**, which can be directly plugged into Express.
- Express was developed by **TJ Holowaychuk and is maintained by the Node.js foundation** and numerous open source contributors.
- Express is a small framework that sits on top of Node.js's web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middle ware and routing; it adds helpful utilities to Node.js's HTTP objects; it facilitates the rendering of dynamic HTTP objects.
- Express is a part of MEAN stack, a full stack JavaScript solution used in building fast, robust, and maintainable production web applications.
- MongoDB(Database)
- ExpressJS(Web Framework)
- AngularJS(Front-end Framework)
- NodeJS(Application Server)

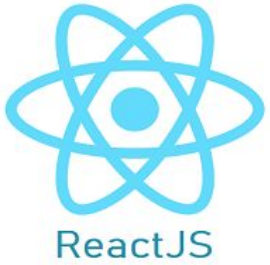


Express.js

- Node.js has a built-in web server. The createServer() method in its http module launches an asynchronous http server.
- It is possible to develop a web application with core Node.js features. However, all the low level manipulations of HTTP request and responses have to be tediously handled.
- The web application frameworks take care of these common tasks, allowing the developer to concentrate on the business logic of the application.
- A web framework such as Express.js is a set of utilities that facilitates rapid, robust and scalable web applications.

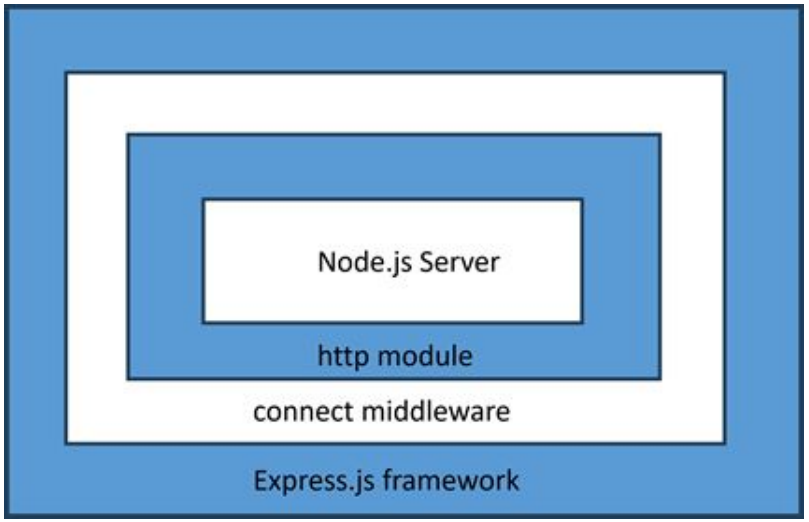
Following are some of the core features of Express framework −

- Allows to set up middlewares to respond to HTTP Requests.
- Defines a routing table which is used to perform different actions based on HTTP Method and URL.
- Allows to dynamically render HTML Pages based on passing arguments to templates.



Express.js

- The Express.js is built on top of the connect middleware, which in turn is based on http, one of the core modules of Node.js API.



Installing Express on Windows

- Assuming that you have installed node.js on your system, the following steps should be followed to install express on your Windows:

Express.js

- Whenever we create a project using npm, we need to provide a package.json file, which has all the details about our project. npm makes it easy for us to set up this file. Let us set up our development project.

Step 1: Start your terminal/cmd, create a new folder named hello-world and cd (create directory) into it:

$ mkdir BackEndProject > $ cd BackEndProject

Step 2: Now to create the package.json file using npm, use the following code.

npm init

It will ask you for the following information.

Just keep pressing enter, and enter your name at the "author name" field.

Step 3: Now we have our package.json file set up, we will further install Express.

To install Express and add it to our package.json file, use the following command:

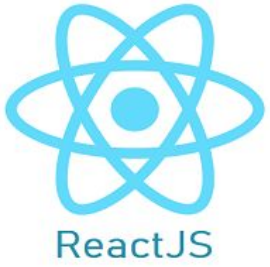
```
npm install --save express
```

Tip: The --save flag can be replaced by the -S flag. This flag ensures that Express is added as a dependency to our package.json file. This has an advantage, the next time we need to install all the dependencies of our project we can just run the command npm install and it will find the dependencies in this file and install them for us.

That's all we need to start development using the Express framework. To make our development process a lot easier, we will install a tool from npm, nodemon. This tool restarts our server as soon as we make a change in any of our files, otherwise we need to restart the server manually after each file modification. To install nodemon, use the following command:

```
npm install -g nodemon
```

You can now start working on Express

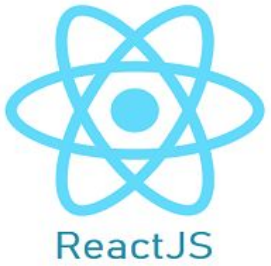


ExpressJS – Hello World

- We have set up the development, now it is time to start developing our first app using Express.
- Create a new file called **index.js** and type the following in it. we will write the whole express code in that file.
- This will be in our folder structure. Now Inside index.js, **Import express with require keyword** and create an **app by calling the express()** function provided by the express framework.
- Set the **port for our local application, 8000** is the default but you can choose any according to the availability of ports.
- Call the **listen() function**, It requires path and callback as an argument.
- It starts listening to the connection on the specified path, the default host is localhost, and our default path for the local machine is the **localhost:8000**, here 8000 is the port which we have set earlier.
- The callback function gets executed either on the successful start of the server or due to an error.

Example:

```
const express = require('express');
const app = express();
const PORT = 8000;
app.listen(PORT, (error) =>{
    if(!error)
      console.log("Server is Successfully Running,
                  and App is listening on port "+ PORT)
    else
      console.log("Error occurred, server can't start", error);
    }
);
```



Save the file, go to your terminal and type the following.

ExpressJS – Hello World

Step to run the application:

- Now as we have created a server we can successfully start running it to see it's working, write this command in your terminal to start the express server.

  node index.js

  This will start the server. To test this app, open your browser and go to http://localhost:8000

- Now with all of this, we have created and run the server successfully, if your server is not starting then there may be some error, try to analyze and read that error and resolve it accordingly.
- Finally, after a successful run if you try to open the URL **(localhost:8000)** on the browser it will **show you cannot GET /** because we have not configured any route on this application yet.

Step 4: Now we will set all the routes for our application.

- Routes are the endpoints of the server, which are configured on our backend server and whenever someone tries to access those endpoints they respond accordingly to their definition at the backend.
- If you're a beginner you can consider route as a function that gets called when someone requests the special path associated with that function and return the expected value as a response.
- We can create **routes for HTTP methods like get, post, put**, and so on.

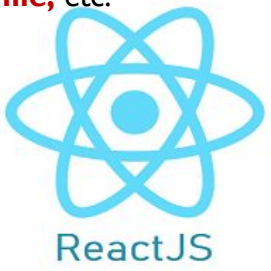
ExpressJS – Hello World

Syntax: The basic syntax of these types of routes looks like this, the given function will execute when the path and the request method resemble.

app.anyMethod(path, function)

Example: Setting up a basic get request route on the root URL ('/' path) of the server.

1. With **app.get()** we are configuring our first route, it requires **two arguments first one is the path and, the second one is a function** that will be executed when anyone requests this path with GET method. The express provides the **request and response object as a parameter** to all such types of functions.
2. The **req** is a giant object which will be **received from the user** and **res** is an object which **will be sent to the user** after the function finishes execution.
3. Later we are calling status() method it takes an HTTP status code as an argument and when the response is returned, the status will be sent along.
4. Finally, we are returning the **response to the user.** The **send()** method takes a **string, object, array, or buffer** as an argument and is used to send the data object back to the client as an HTTP response, also there are **lots of types of response** in express like **res.json() which is used to send JSON object, res.sendFile() which is used to send a file,** etc.

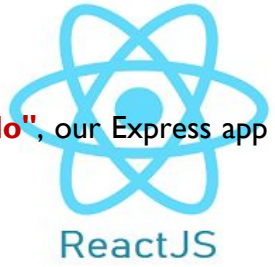


ExpressJS – Hello World

Example: Updated Code including route

```
const express = require('express');

const app = express();
const PORT = 8000;

app.get('/', (req, res)=>{
      res.status(200);
      res.send("Hello World, I'm Responding from Server……");
});

app.listen(PORT, (error) =>{
      if(!error)
            console.log("Server is Successfully Running,
                              and App is listening on port "+ PORT)
      else
            console.log("Error occurred, server can't start", error);
      }
);
```

- If we run our application and go to **localhost:8000/hello**, the server receives a get request at route **"/hello"**, our Express app executes the callback function attached to this route and sends "Hello World!" as the response



ExpressJS – Hello World

How the App Works?

- The first line imports Express in our file, we have access to it through the variable Express. We use it to create an application and assign it to var app.

app.get(route, callback)

- This function tells what to do when a get request at the given route is called. The callback function has **2 parameters, request(req) and response(res).** The request object(req) represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, etc. Similarly, the response object represents the HTTP response that the Express app sends when it receives an HTTP request.

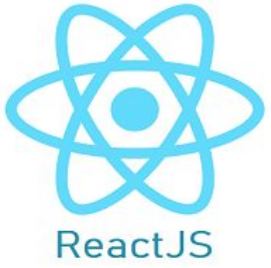
res.send()

- This function takes an object as input and it sends this to the requesting client. Here we are sending the string "Hello World!".

app.listen(port, [host], [backlog], [callback])

- This function binds and listens for connections on the specified host and port. Port is the only required parameter here

Argument	Description
port	A port number on which the server should accept incoming requests.
host	Name of the domain. You need to set it when you deploy your apps to the cloud.
backlog	The maximum number of queued pending connections. The default is 511.
callback	An asynchronous function that is called when the server starts listening for requests.



ExpressJS – Routing

- Web frameworks provide resources such as HTML pages, scripts, images, etc. at different routes.

The following function is used to define routes in an Express application:

```
app.method(path, handler)
```

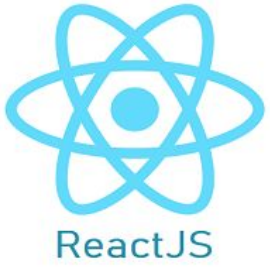
- This METHOD can be applied to any one of the HTTP verbs – **get, set, put, delete.** An alternate method also exists, which executes independent of the request type.
- Path is **the route** at which the request will run.
- Handler is **a callback function that executes** when a matching request type is found on the relevant route.

Then you can define routes like this:

```
app.get('/someUri', function (req, res, next) {})
```

- That structure works for all HTTP methods, and expects a path as the first argument, and a handler for that path, which receives the request and response objects. So, for the basic HTTP methods, these are the routes

```
// GET www.domain.com/myPath
app.get('/myPath', function (req, res, next) {})
// POST www.domain.com/myPath
app.post('/myPath', function (req, res, next) {})
// PUT www.domain.com/myPath
app.put('/myPath', function (req, res, next) {})
// DELETE www.domain.com/myPath
app.delete('/myPath', function (req, res, next) {})
```



ExpressJS – Routing

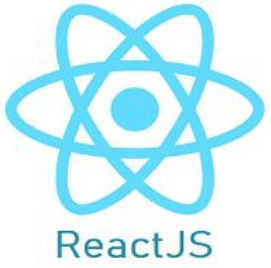
- You can chain your route definitions for a single path

```
app.route('/myPath')
  .get(function (req, res, next) {})
  .post(function (req, res, next) {})
  .put(function (req, res, next) {})
```

- You can also add functions to any HTTP method. They will run before the final callback and take the parameters (req, res, next) as arguments.

// GET www.domain.com/myPath

```
app.get('/myPath', myFunction, function (req, res, next) {})
```

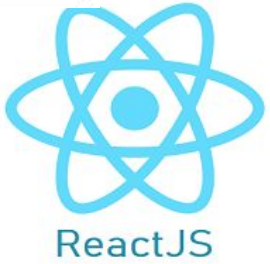


ExpressJS – Http Methods

- The HTTP method is supplied in the request and specifies the operation that the client has requested.

The following table lists the most used HTTP methods

Method	Description
GET	The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
POST	The POST method requests that the server accept the data enclosed in the request as a new object/entity of the resource identified by the URI.
PUT	The PUT method requests that the server accept the data enclosed in the request as a modification to existing object identified by the URI. If it does not exist then the PUT method should create one.
DELETE	The DELETE method requests that the server delete the specified resource.

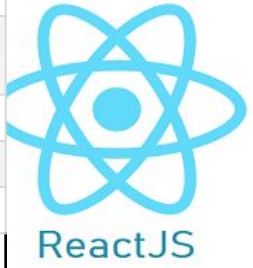


ExpressJS – Http Status Codes

Code	Constant	Reason Phrase
100	CONTINUE	Continue
101	SWITCHING_PROTOCOLS	Switching Protocols
102	PROCESSING	Processing
103	EARLY_HINTS	Early Hints
200	OK	OK
201	CREATED	Created
202	ACCEPTED	Accepted
203	NON_AUTHORITATIVE_INFORMATION	Non Authoritative Information
204	NO_CONTENT	No Content
205	RESET_CONTENT	Reset Content
206	PARTIAL_CONTENT	Partial Content
207	MULTI_STATUS	Multi-Status
300	MULTIPLE_CHOICES	Multiple Choices

ExpressJS – Http Status Codes

301	MOVED_PERMANENTLY	Moved Permanently
302	MOVED_TEMPORARILY	Moved Temporarily
303	SEE_OTHER	See Other
304	NOT_MODIFIED	Not Modified
305	USE_PROXY	Use Proxy
307	TEMPORARY_REDIRECT	Temporary Redirect
308	PERMANENT_REDIRECT	Permanent Redirect
400	BAD_REQUEST	Bad Request
401	UNAUTHORIZED	Unauthorized
402	PAYMENT_REQUIRED	Payment Required
403	FORBIDDEN	Forbidden
404	NOT_FOUND	Not Found
405	METHOD_NOT_ALLOWED	Method Not Allowed
406	NOT_ACCEPTABLE	Not Acceptable
407	PROXY_AUTHENTICATION_REQUIRED	Proxy Authentication Required
408	REQUEST_TIMEOUT	Request Timeout
409	CONFLICT	Conflict
410	GONE	Gone

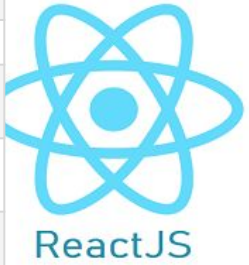


ExpressJS – Http Status Codes

411	LENGTH_REQUIRED	Length Required
412	PRECONDITION_FAILED	Precondition Failed
413	REQUEST_TOO_LONG	Request Entity Too Large
414	REQUEST_URI_TOO_LONG	Request-URI Too Long
415	UNSUPPORTED_MEDIA_TYPE	Unsupported Media Type
416	REQUESTED_RANGE_NOT_SATISFIABLE	Requested Range Not Satisfiable
417	EXPECTATION_FAILED	Expectation Failed
418	IM_A_TEAPOT	I'm a teapot
419	INSUFFICIENT_SPACE_ON_RESOURCE	Insufficient Space on Resource
420	METHOD_FAILURE	Method Failure

ExpressJS – Http Status Codes

421	MISDIRECTED_REQUEST	Misdirected Request
422	UNPROCESSABLE_ENTITY	Unprocessable Entity
423	LOCKED	Locked
424	FAILED_DEPENDENCY	Failed Dependency
426	UPGRADE_REQUIRED	Upgrade Required
428	PRECONDITION_REQUIRED	Precondition Required
429	TOO_MANY_REQUESTS	Too Many Requests
431	REQUEST_HEADER_FIELDS_TOO_LARGE	Request Header Fields Too Large
451	UNAVAILABLE_FOR_LEGAL_REASONS	Unavailable For Legal Reasons
500	INTERNAL_SERVER_ERROR	Internal Server Error
501	NOT_IMPLEMENTED	Not Implemented
502	BAD_GATEWAY	Bad Gateway
503	SERVICE_UNAVAILABLE	Service Unavailable
504	GATEWAY_TIMEOUT	Gateway Timeout
505	HTTP_VERSION_NOT_SUPPORTED	HTTP Version Not Supported
507	INSUFFICIENT_STORAGE	Insufficient Storage
511	NETWORK_AUTHENTICATION_REQUIRED	Network Authentication Required



ExpressJS - RESTFul APIs

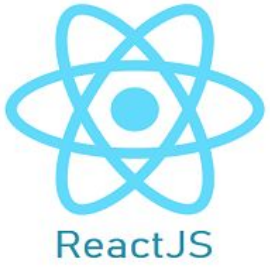
- Restful API is very popular and commonly used to create APIs for web-based applications.
- Express is a back-end web application framework of node js, and with the help of express, we can create an API very easily.

Introduction

- APIs encompass a collection of definitions and protocols that play a crucial role in application development and integration. They serve as the bridge between different software components, enabling seamless data exchange between an information provider (server) and a user.

- APIs define the data and functionalities accessible to clients making requests to the producer, which in turn returns the appropriate responses.

- By utilizing an API, programs can communicate, retrieve information, and execute specific functions. APIs empower users to interact with a system, enabling them to achieve their desired outcomes.



ExpressJS - RESTFul APIs

Below are several reasons highlighting the importance of integrating APIs:

1. **Simplifying Resource and Information Sharing:** APIs facilitate the streamlined sharing of resources and information between different components or services.

2. **Access Control and Authentication:** APIs enable you to control and manage access rights, ensuring that only authorized users can interact with specific resources through proper authentication mechanisms.

3. **Enhancing Safety and Control:** By utilizing APIs, you can enforce security measures and maintain control over the data and functionality exposed, protecting against unauthorized access and ensuring data integrity.

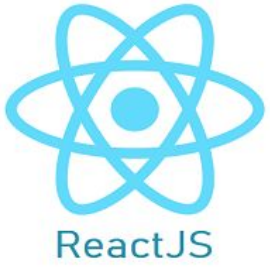
4. **Abstracting Software Specifics:** APIs abstract away the underlying complexities of the software implementation, allowing users to interact with services without needing in-depth knowledge of their internal workings.

5. **Enabling Consistent Communication:** APIs provide a standardized communication interface, allowing different services or systems, even if they employ different technologies, to interact seamlessly and exchange data reliably.

ExpressJS - RESTFul APIs

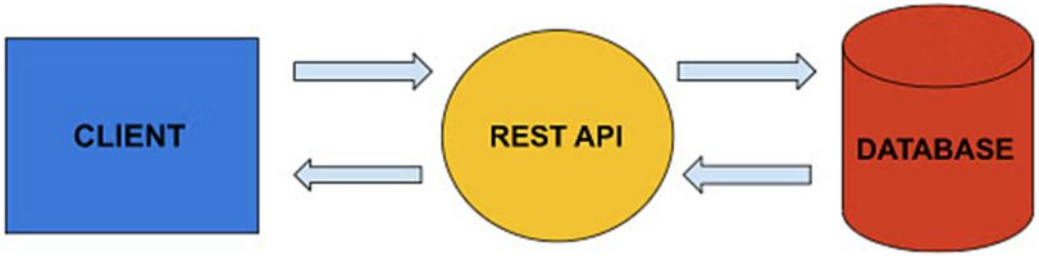
What Is REST API?

- REST (Representational state transfer) is a popular architecture that is used to building and communicating with web services.
- It typically mandates resources on the web are represented in a text format (like JSON, HTML, or XML) and can be accessed or modified by a predetermined set of operations.
- API (Application Programming Interface is a code that allows two software programs to communicate with each other.
- REST API is a software component that facilitates communication between two applications over the internet, enabling interaction across various devices. It acts as the accepted protocol for data exchange in online services.



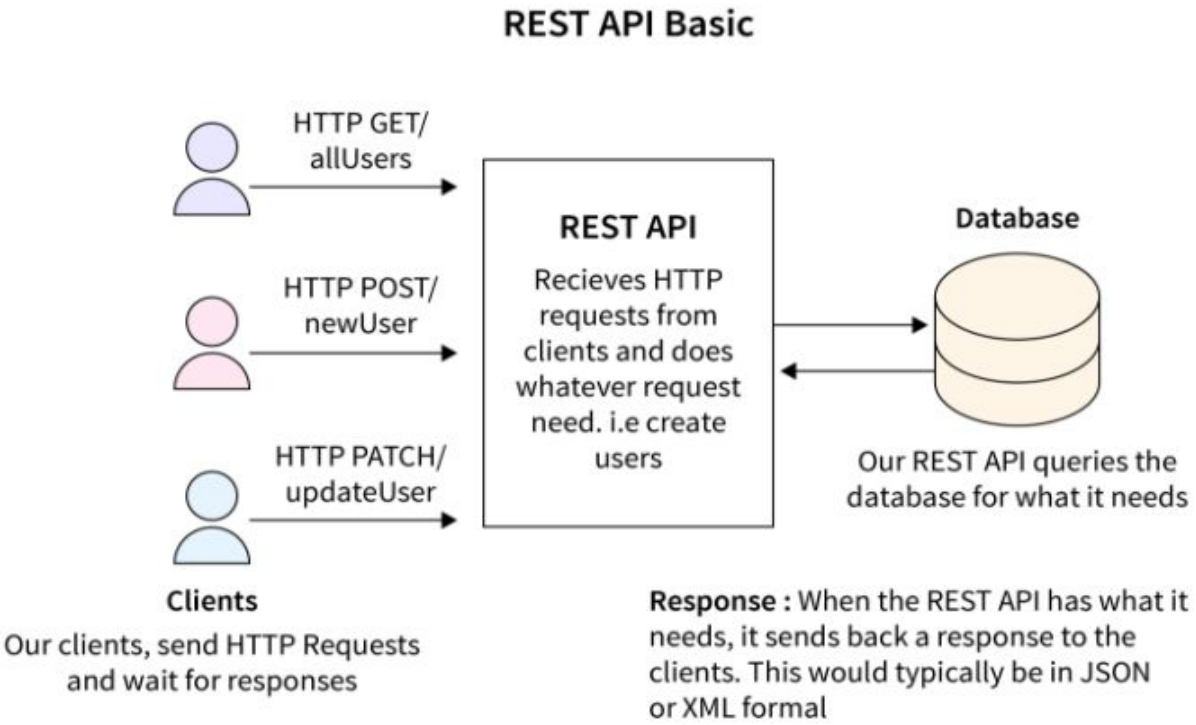
ExpressJS - RESTFul APIs

- REST API is the standard way to send and receive data for web services.
- A client sends a **req which first goes to the rest API** and then to the database to **get or put the data** after that, it will again go to the **rest API and then to the client.**
- Using an API is just like using a website in a browser, but instead of clicking on buttons, we **write code to req data from the server.** It's incredibly adaptable and can handle multiple types of requests.



ReactJS

ExpressJS - RESTFul APIs



Typical HTTP Verbs:

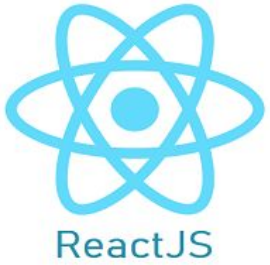
GET - Read from database

POST - Create a new record in the database

PUT - Update/Replace row in Database

DELETE - Delete from the database

PATCH - Update/Modify row in Database



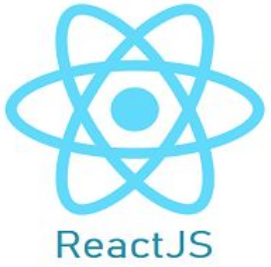
ExpressJS - RESTFul APIs

- The beautiful thing about APIs is that they are adaptable; they can handle a wide range of requests and work in a variety of contexts.
- Images, movies, and text are examples of valuable resources in the realm of REST. As a result, when you access a certain URL and send a request to the server, you are effectively requesting it to provide you with the data you require.

Let's explore this more in-depth,

When you request by accessing a URL, it consists of four important components:

1. **The Endpoint:** This is the URL structure that starts with the root endpoint followed by additional path elements.
2. **The Method:** It chooses which of the five action types you wish to carry out on the server: GET, POST, PUT, PATCH, or DELETE.
3. **The Headers:** These fulfil several functions, including authentication and providing details about the request body's contents. To include specified HTTP headers, use the -H or --header option.
4. **The Data or Body:** With POST, PUT, PATCH, or DELETE requests, you can provide the server with this data by using the -d or --data option.

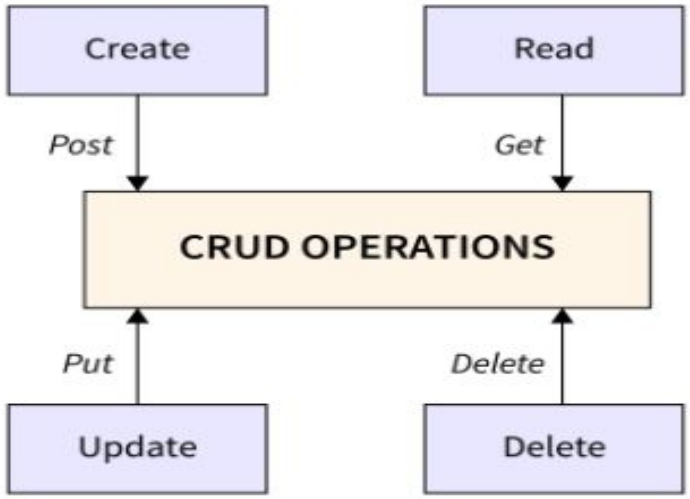


ExpressJS - RESTFul APIs

HTTP Request Types

- HTTP Requests are simply messages that are sent by the client to do some tasks on the server

1. **GET -** Get command is used to request data from the server, but mainly this method is used to read data.
2. **PATCH -** This command is used to update, change or replace the data.
3. **POST -** The post method is used to create new or to edit already existing data.
4. **Delete -** This delete command is used to delete the data completely from the server.

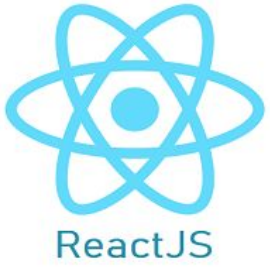


ReactJS

ExpressJS - RESTFul APIs

- When the server responds to your requests, it sends the data back to you in one of the following formats, depending on the server configuration and the type of request made:

1. **HTML**
2. **JSON**
3. **XLT**
4. **PHP**
5. **Python**
6. **plain text**



ExpressJS - RESTFul APIs

Principles of REST

Well, there are six ground principles laid down by Dr. Fielding who was the one to define the REST API design in 2000. Below are the six guiding principles of REST:

1. Stateless

Requests sent from a client to the server contains all the necessary information that is required to completely understand it. It can be a part of the URI, query-string parameters, body, or even headers. The URI is used for uniquely identifying the resource and the body holds the state of the requesting resource. Once the processing is done by the server, an appropriate response is sent back to the client through headers, status or response body.

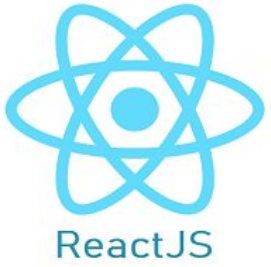
2. Client-Server

It has a uniform interface that separates the clients from the servers. Separating the concerns helps in improving the user interface's portability across multiple platforms as well as enhance the scalability of the server components.

3. Uniform Interface

To obtain the uniformity throughout the application, REST has defined four interface constraints which are:

1. Resource identification
2. Resource Manipulation using representations
3. Self-descriptive messages
4. Hypermedia as the engine of application state



ExpressJS - RESTFul APIs

4. Cacheable

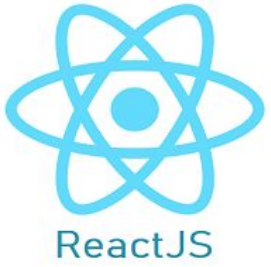
In order to provide a better performance, the applications are often made cacheable. It is done by labeling the response from the server as cacheable or non-cacheable either implicitly or explicitly. If the response is defined as cacheable, then the client cache can reuse the response data for equivalent responses in the future. It also helps in preventing the reuse of the stale data.

5. Layered system

The layered system architecture allows an application to be more stable by limiting component behavior. This architecture enables load balancing and provides shared caches for promoting scalability. The layered architecture also helps in enhancing the application's security as components in each layer cannot interact beyond the next immediate layer they are in.

6. Code on demand

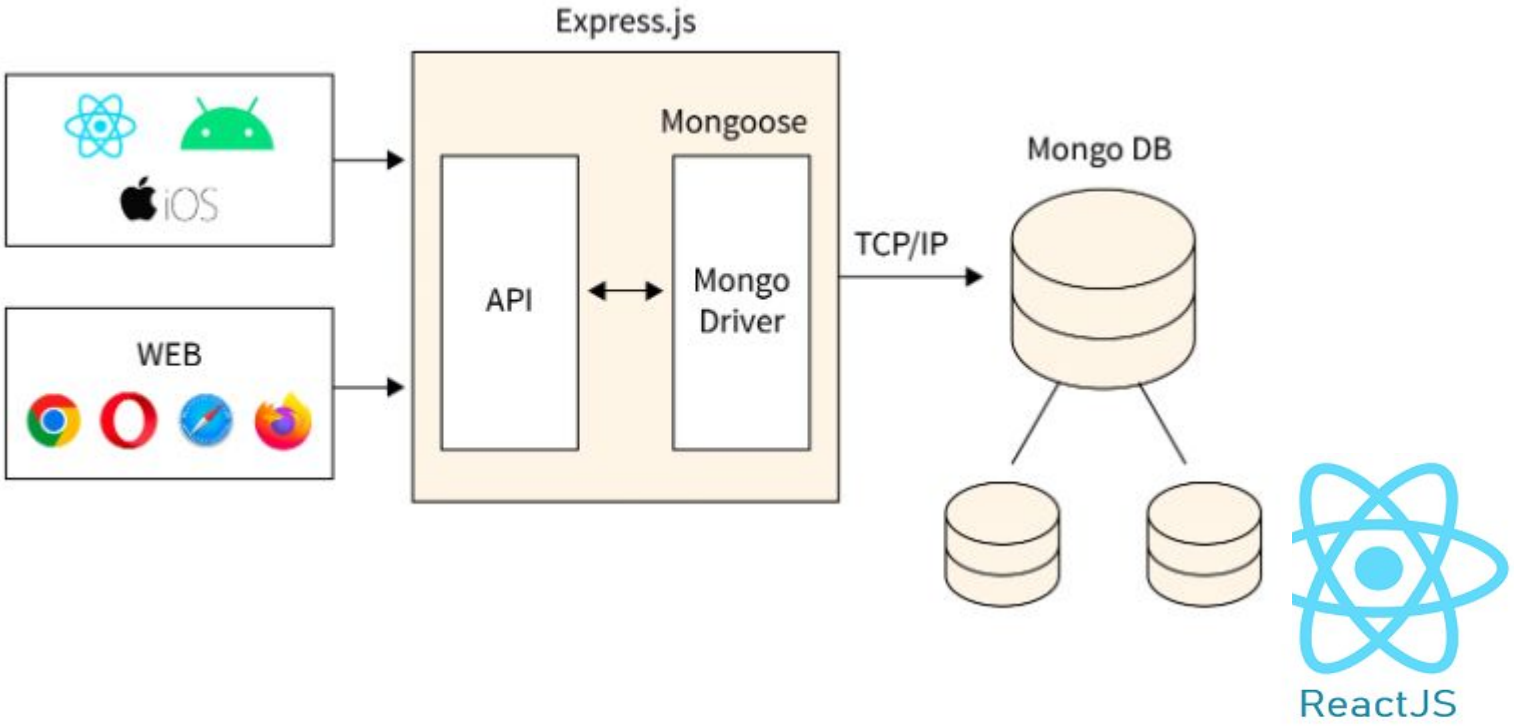
Code on Demand is an optional constraint and is used the least. It permits a clients code or applets to be downloaded and extended via the interface to be used within the application. In essence, it simplifies the clients by creating a smart application which doesn't rely on its own code structure.



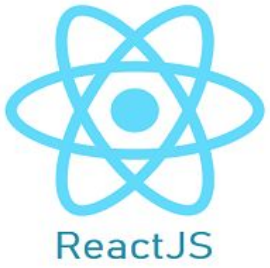
ExpressJS - RESTFul APIs

API Using Express and Its Architecture

- A request is sent by the client in the form of a JSON file, and with the help of an HTTP request which **gets, patch, post and delete**, it will go to the server first then, the server sends back the response to the client in the form of a message to tell what happened to your request.



MongoDB



End Of UNIT-4