PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Installation and Configuration of React.

PREREQUISITE: Basic knowledge of software installation.

Theory / Algorithm / Conceptual Description:

React is a popular JavaScript library for building user interfaces, and it's widely used for developing interactive and dynamic web applications. To get started with React, you need to set up your development environment and configure your project correctly. Here, we will explore the installation and configuration of React, providing you with the foundational knowledge to kickstart your React development journey.

Setting Up Your Environment

1. Download and Install Node.js & npm

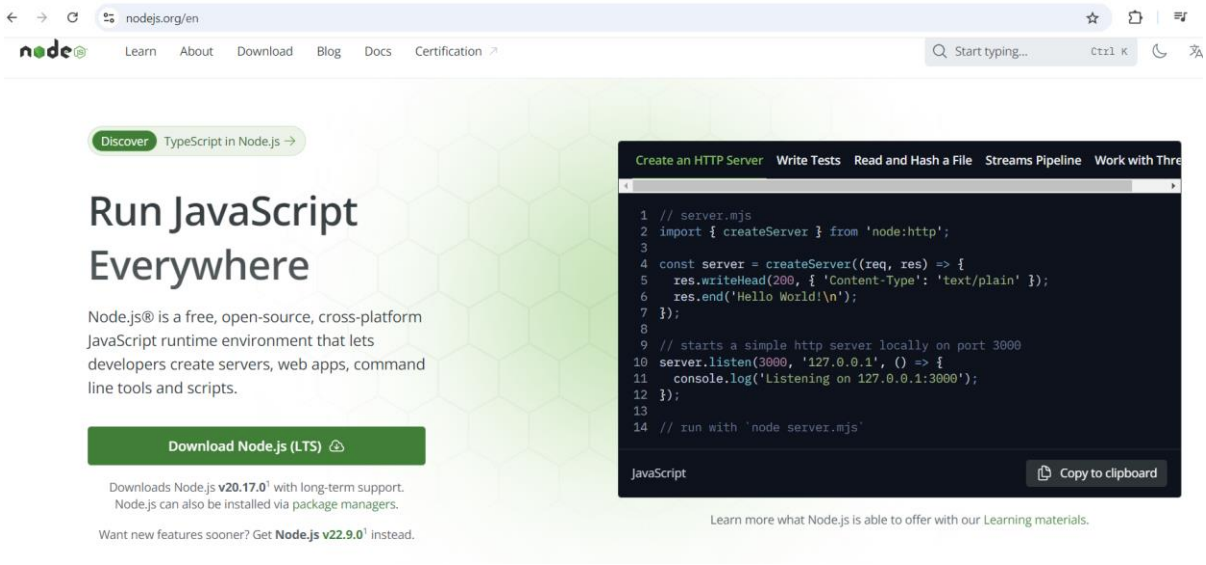
React development relies on Node.js, a JavaScript runtime, and npm (Node Package Manager) for package management.

Node.js should be installed first because React.js is a JavaScript library, and Node.js is a JavaScript runtime environment that allows to run JavaScript on the server side. So when we will writing React, we include JavaScript functions in our React project, and then Node.js helps run this JavaScript code on the web page.

Node.js has various versions. The recommended version is the latest stable version, as it contains major and significant changes. These changes includes bug fixes and security updates, compatibility with your project dependencies, and so on.

To install Node, navigate to the Node.js website. On their webpage, the option to download either the recommended version or the current version, as seen in the image below. After downloading the appropriate version, install it on computer.

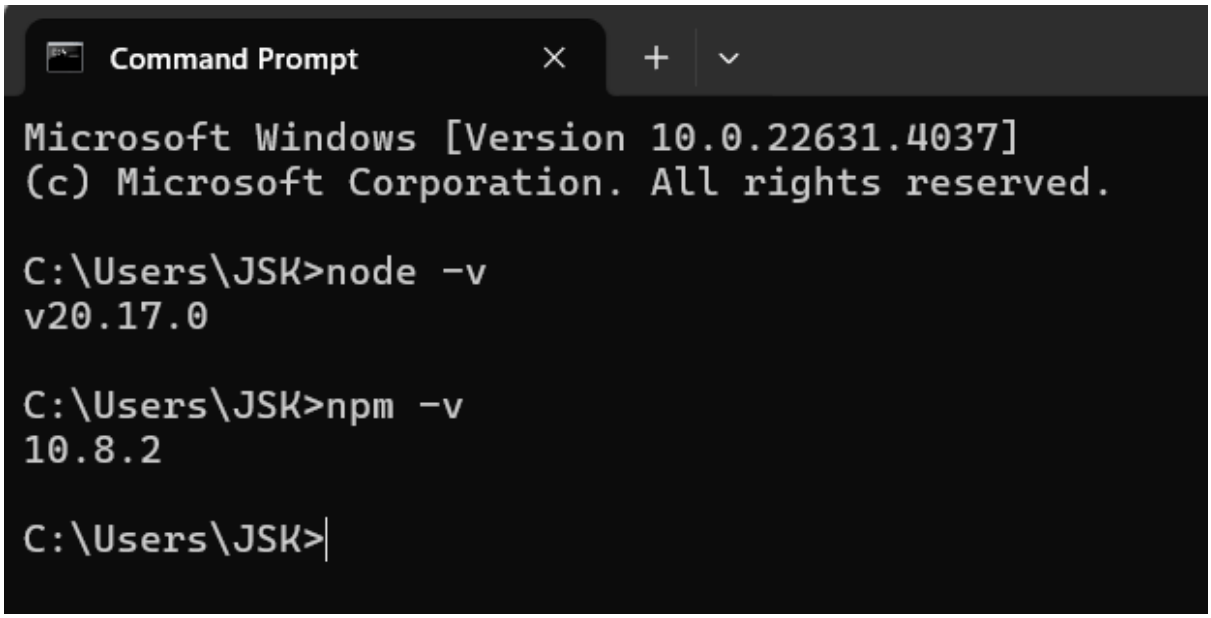
Link: https://nodejs.org/en



To verify that Node.js and npm are properly installed, open terminal or command prompt and run the following commands:

node -v

npm –v



If your Node version is displayed like the above, it means you have successfully installed Node.js on your computer.

2. Install Create-React-App Tool

The next step is to install a tool called create-react-app using NPM. This tool is used to create react applications easily from our system. You can install this at the system level or temporarily at a folder level. We will install it globally by using the following command.

npm install -g create-react-app

```
C:\Users\JSK>node -v
v20.17.0

C:\Users\JSK>npm -v
10.8.2

C:\Users\JSK>npm install -g create-react-app
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache i
f you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerfu
l.
npm warn deprecated fstream-ignore@1.0.5: This package is no longer supported.
npm warn deprecated uid-number@0.0.6: This package is no longer supported.
npm warn deprecated rimraf@2.7.1: Rimraf versions prior to v4 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated fstream@1.0.12: This package is no longer supported.
npm warn deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please
 upgrade asap.

changed 64 packages in 19s

4 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New patch version of npm available! 10.8.2 -> 10.8.3
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.3
npm notice To update run: npm install -g npm@10.8.3
npm notice

C:\Users\JSK>
```

3. Creating a new react project

After create-react-app is installed, we can create our first react application. Let's say I want to create the project or application in D:\react_pract. I will create this folder and let our command prompt point to it by using the change directory command. Let's create a new Project now using the command.

create-react-app exp1

```
C:\Users\JSK>d:

D:\>mkdir react_pract

D:\>cd react_pract

D:\react_pract>create-react-app exp1
```

Remember not to create the project with an upper case character in it. If you encounter any policy restriction error, solve it as follows.

To resolve this issue and be able to run the create-react-app command, you will need to adjust your PowerShell execution policy to allow script execution. Here's how you can do it:

- Open PowerShell as an administrator. To do this, search for "PowerShell" in the Start menu, right-click on "Windows PowerShell," and select "Run as administrator."
- Check your current execution policy by running the following command:
- Get-ExecutionPolicy
- If the execution policy is set to "Restricted," you need to change it. To allow script execution, you can set the execution policy to "RemoteSigned" or "Unrestricted." For example, to set it to "RemoteSigned," run the following command:

  Set-ExecutionPolicy RemoteSigned

4. Running the React Application

Let's do CD to the Project we have created and run it locally on our system using npm start. Launch the browser and visit http://localhost:3000. We can then see our first React Application response in the browser.

cd exp1

npm start

```
Success! Created exp1 at D:\react_pract\exp1
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd exp1
  npm start

Happy hacking!

D:\react_pract>
```

```
D:\react_pract>cd exp1

D:\react_pract\exp1>npm start

> exp1@0.1.0 start
> react-scripts start
```

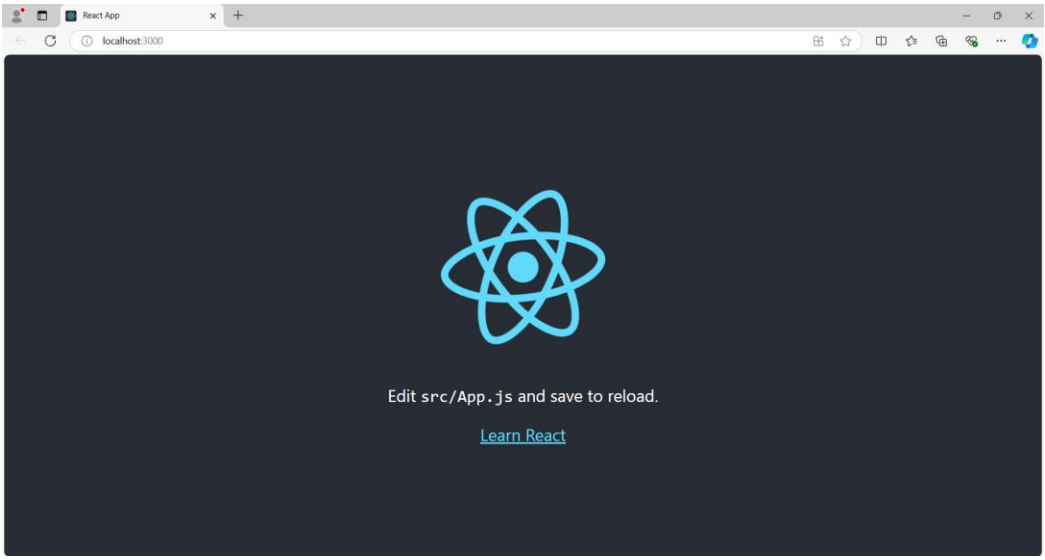
```
Compiled successfully!

You can now view exp1 in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



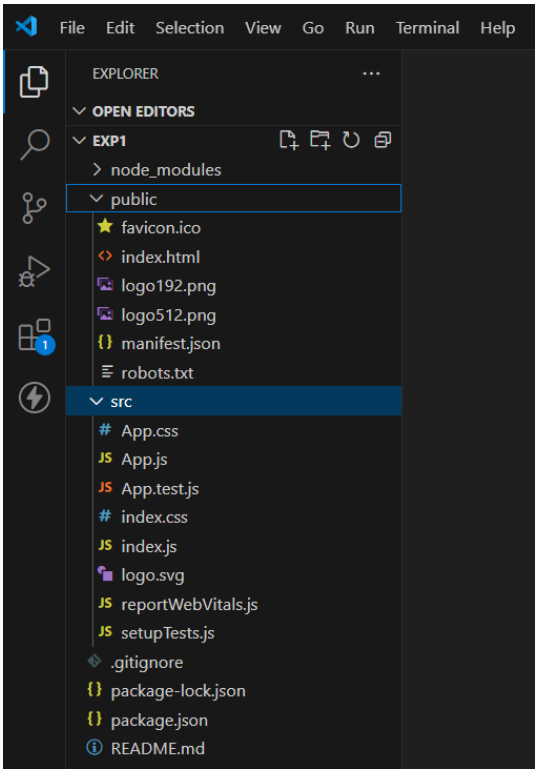
React complete installation in the terminal

We have created a New Project using React and executed the Project. But as a developer, we would be more interested to know about the Project which is created, its structure and we would like to play around with it. So it is time for us to get an Editor. When we think of IDE, we have a variety of choices like Visual Studio Code, React IDE, Sublime Editor, Atom Editor, Webstorm and a few others. We will use the VS Code as our Editor.

5. Install Visual Studio Code

Link: https://code.visualstudio.com/download

Now Start the Visual Studio Code and open the React project in it using *File>Open Folder*. It will be look like this:



CONCLUSION: In this experiment, we have installed React onto our system and made our first react project.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Understanding JSX, Components, Props, State in React

PREREQUISITE: HTML, CSS and JavaScript.

Theory / Algorithm / Conceptual Description:

React is a popular JavaScript library for building user interfaces, and it relies on several fundamental concepts that are essential to understand when working with it. In this experiment, we will explore JSX, components, props, and states in React, providing you with a solid foundation to begin building dynamic web applications.

1. **JSX: JavaScript XML**

JSX, which stands for JavaScript XML, is a syntax extension for JavaScript used in React. It allows you to write HTML-like code within your JavaScript files. JSX is a crucial part of React as it simplifies the process of creating and rendering user interfaces.

Here's a basic example of JSX:

const myElement = <h1> React works with JSX! </h1>;

ReactDOM.render(myElement, document.getElementById ('root'));

In this example, we create a variable element that contains JSX code. It looks like HTML, but it's actually JavaScript. JSX elements can be used to describe the structure of your UI components.

2. **Components**

In React, a component is a reusable building block for your user interface. Components are like custom HTML elements that you can define and reuse throughout your application. React applications are typically composed of multiple components, which makes the code more organized and maintainable.

There are two main types of components in React:

1. **Functional Components:**

These are simple JavaScript functions that return JSX elements. They are commonly used for presentational components.

function Car() {

return <h2> Hi, I am a Car! </h2>;

}

2. **Class Components:**

These are JavaScript classes that extend the React.Component class. They have a render method that returns JSX. Class components are used for components that require state or lifecycle methods.

class Car extends React. Component {

render() { return <h2>Hi, I am a Car!</h2>;

}

}

3. <u>Props</u>

Props, short for properties, are a way to pass data from a parent component to a child component. They make it easy to customize and configure your components. Props are read-only, meaning that a child component cannot modify the props it receives from its parent.

Here's how you pass and use props in a functional component: function Car(props) {

return <h2> I am a {props.color} Car! </h2>;

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Car color="red"/>);

Use an attribute to pass a color to the Car component, and use it in the render() function.

4. **<u>State</u>**

The state is a built-in React object that is used to contain data or information about the component. A component’s state can change over time; whenever it changes, the component re-renders. The change in state can happen as a response to user action or system-generated events and these changes determine the behaviour of the component and how it will render.

class Greetings extends React.Component { state = { name: "World"

};

updateName() {

this.setState({ name: "RCPIT" });

}

render() { return(

```
<div>
{this.state.name}
</div>
)
}
}
```

CONCLUSION: In this experiment, we have understood JSX, components, props, and states in React by making the corresponding codes.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Implementing Forms, Events, Routers, Refs, Keys

PREREQUISITE: HTML, CSS and JavaScript.

Theory / Algorithm / Conceptual Description:

1. Forms in React

Forms are a fundamental part of web applications, allowing users to input data. In React, you can create and manage forms using JSX, similar to regular HTML forms. React offers a controlled component approach, where form elements are controlled by the component's state. This allows for seamless synchronization between the form elements and the component's state.

2. Events in React

React applications are interactive, and to achieve this, you need to work with events. Events in React are similar to regular DOM events but with some differences. In React, events are named using camelCase, and they are passed as props to elements.

React events also have a property called preventDefault() that allows you to prevent the default behaviour of an event, such as preventing a form submission.

3. Routers in React

Routing is crucial for building multi-page web applications. React doesn't include a built-in router, but there are popular third-party libraries like React Router that provide routing capabilities. React Router allows you to define routes and render components based on the current URL.

4. Refs in React

Refs are a way to access and interact with the DOM directly. They allow you to reference a DOM element or a React component instance. Refs are often necessary when you need to imperatively modify the DOM or interact with a child component.

5. Keys in React

Keys are a special attribute in React used to give elements a stable identity during the process of rendering and updating a list of items. Keys help React efficiently update the DOM without unnecessary re-renders.

When rendering a list of items, each item in the list should have a unique key prop.

The key prop should be unique within the list of items. React uses keys to identify which items have changed, been added, or removed, optimizing the rendering process.

Understanding forms, events, routers, refs, and keys in React is essential for building dynamic and interactive web applications. Forms allow for user input, events enable interactivity, routers manage multi-page applications, refs provide access to the DOM, and keys optimize rendering lists.

CONCLUSION: In this experiment, we have seen how to use Forms, Events, Router, Refs, and Keys in React.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Create an application to demonstrate use of Conditional rendering in React JS.

PREREQUISITE: HTML, CSS and JavaScript.

Theory / Algorithm / Conceptual Description:

Conditional Rendering in React

In React, conditional rendering is the process of displaying different content based on certain conditions or states. It allows you to create dynamic user interfaces that can adapt to changes in data and user interactions. In this process, you can use conditional statements to decide what content should be rendered.

Why Conditional Rendering is Necessary in React Applications?

There are several reasons why you might want to use conditional rendering in your React applications:

- **Improved User Experience:** Conditional rendering allows you to create dynamic user interfaces that adapt to changes in data and user interactions. By showing and hiding content based on the user's actions or the application state, you can create a more intuitive and engaging user experience.
- **Improved Performance:** By conditionally rendering content, you can avoid rendering unnecessary components and improve the performance of your application. This is particularly important in larger applications where unnecessary rendering can lead to performance issues.
- **Simplified Code:** Conditional rendering can help you simplify your code and make it more readable. By using conditional statements to decide what content should be rendered, you can avoid duplicating code and create more modular components.
- **Flexibility:** Conditional rendering allows you to create more flexible and customizable components. By rendering different content based on the application state, you can create components that can be used in different contexts and adapt to different user interactions.

Conditional Rendering can be done by the following methods:

1. Conditional Rendering with If Statements

Although JSX doesn't allow traditional if statements directly, conditional rendering can be achieved by using JavaScript if statements outside of the JSX or within a conditional block.

2. Conditional Rendering with Ternary Operator

The ternary operator (condition ? expr1 : expr2) is commonly used for conditional rendering in React. It evaluates the condition and returns expr1 if the condition is true, otherwise expr2.

3. Using Logical && Operator

The logical AND (&&) operator can also be used for conditional rendering in React. It allows for short- circuit evaluation.

Conditional rendering in React is a powerful feature that allows developers to control the appearance of elements or components based on conditions. It enables dynamic and flexible UIs that respond to changes in application state or user interactions.

CONCLUSION: Thus, we have demonstrated how conditional rendering can be achieved in React.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Create an application to demonstrate use of React hooks and JS.

PREREQUISITE: HTML, CSS and JavaScript.

Theory / Algorithm / Conceptual Description:

JavaScript in React:

React is a JavaScript library for building user interfaces. It enables developers to create reusable UI components. Within React, JavaScript plays a pivotal role in managing state, handling lifecycle events, and manipulating the DOM.

useState Hook:

useState is a React Hook used to introduce state management within functional components. It enables functional components to have stateful behavior, previously exclusive to class components. By using useState, you can declare state variables and manage their updates.

- Declaration: *const [state, setState] = useState(initialValue);*
- State Update: *setState(newValue);*
- Functionality: This hook returns a state variable (state) and a function (setState) to update that variable. It preserves the state between re-renders of the component.

useEffect Hook:

useEffect is a Hook that enables performing side effects in functional components. It replaces lifecycle methods (componentDidMount, componentDidUpdate, componentWillUnmount) in class components. It handles side effects like data fetching, subscriptions, DOM manipulations, etc.

- Usage: *useEffect(() => { // side effect code }, [dependencies]);*
- Dependencies: The second argument, an optional array of dependencies, controls when the effect runs. If dependencies change between renders, the effect runs again.
- Cleanup: The effect can return a cleanup function to handle the cleanup of resources when the component unmounts or the dependencies change.

useContext Hook:

useContext is a Hook that allows functional components to consume context that was created by React.createContext(). Context provides a way to pass data through the component tree without having to pass props manually at every level.

- Usage: *const value = useContext(MyContext);*
- Functionality: It accesses the current context value of the provided context and subscribes the component to future changes in that context value.

Benefits of Hooks:

1. Simplicity: Hooks simplify the state and lifecycle management in functional components, reducing boilerplate code.
2. Reusability: They encourage better code reuse, as logic can be encapsulated and shared across components.
3. Improved Readability: Using Hooks can lead to clearer, more organized code, making it easier to understand and maintain.

JavaScript plays a central role in React, and Hooks like useState, useEffect, and useContext enhance functional components by adding state, side effects, and context capabilities. These Hooks significantly contribute to simplifying React development, allowing for more modular, readable, and maintainable code.

CONCLUSION: Thus, we have demonstrated how to use Hooks, especially the useState, useEffect, and useContext Hooks in React.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Implement CRUD operations in MongoDB.

PREREQUISITE: SQL, DBMS.

Theory / Algorithm / Conceptual Description:

Introduction to MongoDB:

MongoDB is a popular NoSQL database that utilizes a flexible, document-based model to store data. It stores data in BSON (Binary JSON) format, offering scalability, flexibility, and high performance. CRUD Operations (Create, Read, Update, and Delete) are the basic set of operations that allow users to interact with the MongoDB server.

To use MongoDB we need to interact with the MongoDB server to perform certain operations like entering new data into the application, reading the application data, updating data into the application and deleting data from the application.

1. **Create (Insert) Operations:**
   Create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database.
   Create operations can be perform using the following methods provided by the MongoDB:

Method	Description
db.collection.insertOne()	It is used to insert a single document in the collection.
db.collection.insertMany()	It is used to insert multiple documents in the collection.
db.createCollection()	It is used to create an empty collection.

2. **Read Operations:**

The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document.

Read operation can perform using the following method provided by the MongoDB:

Method	Description
db.collection.find()	It is used to retrieve documents from the collection.

Comparison operators

Operator	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.

Logical Operators

MongoDB provides different types of logical query operators.

Method	Description
{ $and: [{ condition1 }, { condition2 }, ...	$and operator is used to perform logical AND operation on the array of one or more expressions and select or retrieve only those documents that match all the given expression in the array.
{ $or: [{ condition1 }, { condition2 }, ...	$or operator is used to perform logical OR operation on the array of two or more expressions and select or retrieve only those documents that match at least one of the given expression in the array.
{ field: { $not: { operator: value } } }	#not operator is used to perform logical NOT operation on the specified operator expressions and select or retrieve only those documents that do not match the given operator expression.
{ $nor: [{ condition1 }, { condition2 }, ...	$nor operator is used to perform logical NOR operation on the array of one or more expressions and select or retrieve only those documents that do not match all the given expression in the array

Cursor Methods

The MongoDB cursor is a pointer that references the documents of the collection returned by the find() method.

Method	Description
db.collection_name.find().count()	To get the count of documents we can use the count() method which returns the total number of documents present in the given collection.
db.collection_name.find().limit(<number>)	The limit() method helps to fetch limited records from a collection.
db.collection_name.find().sort(<1 or -1>)	Usually while verifying documents, if the output is in sorted order, either in ascending or descending order, it will be easier. So we use the sort() method to sort the documents. If you want to sort the documents in ascending, then set the value of the field to 1 and in descending, then set -1.

3. **Update Operations:**

Update operations are used to update or modify the existing document in the collection.
Update operations can be perform using the following methods provided by the MongoDB:

Method	Description
db.collection.updateOne()	It is used to update a single document in the collection that satisfy the given criteria.
db.collection.updateMany()	It is used to update multiple documents in the collection that satisfy the given criteria.

4. **Delete Operations:**

Delete operation are used to delete or remove the documents from a collection.
Delete operations can be perform using the following methods provided by the MongoDB:

Method	Description
db.collection.deleteOne()	It is used to delete a single document from the collection that satisfy the given criteria.
db.collection.deleteMany()	It is used to delete multiple documents from the collection that satisfy the given criteria.

CRUD Operations Summary:

- Create: Inserts new documents into a collection.
- Read: Retrieves documents from a collection based on specified conditions.
- Update: Modifies existing documents by updating specific fields or replacing them.
- Delete: Removes documents from a collection based on specified conditions.

CRUD operations form the backbone of database interactions in MongoDB. These operations allow developers to perform essential tasks such as storing, retrieving, updating, and deleting data within collections. Understanding and effectively utilizing CRUD operations in MongoDB are fundamental skills in building robust and efficient applications.

Program

Conclusion:

Thus, we have demonstrated how we can perform CRUD Operations in MongoDB.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Installation and Configuration of Node.js

PREREQUISITE: Basic knowledge of software installation.

Theory / Algorithm / Conceptual Description:

What is Node.js?
Node.js is a server-side JavaScript runtime built on Chrome's V8 JavaScript engine. It enables the execution of JavaScript code outside a web browser, allowing developers to create server-side and networking applications.

Key Features of Node.js:

- Asynchronous and Event-Driven: Node.js uses an event-driven, non-blocking I/O model, making it efficient for handling concurrent operations. This enables high scalability and performance.
- JavaScript Runtime: Node.js allows developers to write server-side code using JavaScript, promoting code reuse, and leveraging the same language for both frontend and backend development.
- Vibrant Ecosystem: It has a rich ecosystem of libraries (npm - Node Package Manager) that provide various modules and packages for extending its functionalities. Single-Threaded with Event Loop: Node.js operates on a single-threaded event loop, efficiently handling multiple simultaneous connections without creating new threads for each request.

Use Cases and Applications:

- Web Servers: Node.js is commonly used to create fast, scalable web servers, handling concurrent connections efficiently.
- API Development: It is well-suited for building RESTful APIs and microservices due to its lightweight and high-speed nature.
- Real-Time Applications: Node.js is used in applications requiring real-time data updates, such as chat applications, gaming servers, and collaborative tools.
- IoT and Networking: Its event-driven architecture makes it suitable for building IoT applications and network-based applications.

Core Concepts of Node.js

1. **Non-Blocking I/O:** Node.js employs non-blocking I/O operations, meaning it doesn't wait for file I/O or network requests to complete. Instead, it continues executing other tasks and notifies the application upon completion using callbacks or Promises.

2. **Event-Driven Architecture:** Node.js operates on an event-driven architecture, utilizing events, emitters, and listeners. It allows developers to create responsive applications that react to various events efficiently.
3. **CommonJS Modules:** Node.js follows the CommonJS module system, allowing developers to organize code into reusable modules. Modules encapsulate functionality, promoting code reusability and maintainability.
4. **npm (Node Package Manager):** npm is the package manager for Node.js, providing access to a vast ecosystem of open-source libraries and tools. It allows developers to easily install, manage, and share packages for their Node.js projects.

Node.js Architecture

- V8 Engine
  Node.js utilizes the V8 JavaScript engine, developed by Google for Chrome. V8 compiles JavaScript into machine code, optimizing performance and executing JavaScript quickly.
- Libuv
  Libuv is a multi-platform support library in Node.js, providing event loops, asynchronous I/O operations, and thread pooling. It handles tasks like file system operations, networking, and concurrency.
- Node.js Modules
  Node.js includes core modules like http, fs (file system), events, and util. Additionally, it allows developers to create custom modules for reusable functionalities.

Procedure

Installing Node On Windows (WINDOWS 10):

One must follow the following steps to install the Node.js on your Windows:

Step-1: Downloading the Node.js '.msi' installer.

The first step to install Node.js on windows is to download the installer. Visit the official Node.js website i.e) https://nodejs.org/en/download/ and download the .msi file according to your system environment (32-bit & 64-bit). An MSI installer will be downloaded on your system.

Step-2: Running the Node.js installer.

Now you need to install the node.js installer on your PC. You need to follow the following steps for the Node.js to be installed:

- Double click on the .msi installer.The Node.js Setup wizard will open.
- Welcome To Node.js Setup Wizard.
- Select "Next"
- After clicking "Next", End-User License Agreement (EULA) will open.
- Check "I accept the terms in the License Agreement"
- Select "Next"
- Destination Folder. Set the Destination Folder where you want to install Node.js & Select "Next"
- Custom Setup. Select "Next"
- Ready to Install Node.js.
- The installer may prompt you to "install tools for native modules". Select "Install"
- Installing Node.js.
- Do not close or cancel the installer until the install is complete
- Complete the Node.js Setup Wizard.
- Click "Finish"

Step 3: Verify that Node.js was properly installed or not.

To check that node.js was completely installed on your system or not, you can run the following command in your command prompt or Windows Powershell and test it:-

If node.js was completely installed on your system, the command prompt will print the version of the node.js installed.

Program

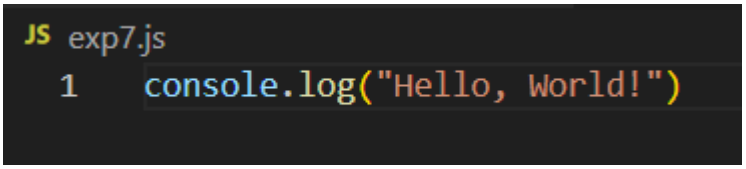
To verify whether Node.js was properly installed on your system, create a file named main.js with the following code.

```
console.log("Hello, World!")
```

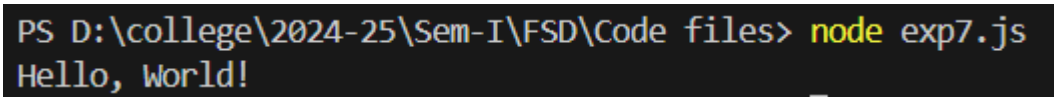
Then in the terminal type the following command to see the output:

node main.js

Input



Output



Conclusion:
Thus, we have installed and configured and made our first program in Node.js.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Implementing Callbacks, Event loops in Node.js

PREREQUISITE: Node JS**.**

Theory / Algorithm / Conceptual Description:

What is Callback?

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So, there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

Example of Non-Blocking Code:

1. Create a text file named input.txt with the following content.
   Tutorials Point is giving self learning content to teach the world in simple and easy way!!!!!
2. Update main.js to have the following code − *var fs = require("fs"); fs.readFile('input.txt', function (err, data) { if (err) return console.error(err); console.log(data.toString());*
   });
   console.log("Program Ended");
3. Now run the main.js to see the result –
   $ node main.js
4. Verify the Output.
   Program Ended
   Tutorials Point is giving self learning contentto teach the world in simple and easy way!!!!
   The program does not wait for file reading and proceeds to print "Program Ended" and at the same time, the program without blocking continues reading the file.

Event Loops

The Event Loop and Emitters are fundamental concepts in Node.js, which make it an efficient and event driven environment for server-side programming. Let's explore these concepts in detail:

Event Loop

The event loop is a critical component of Node.js that enables non-blocking, asynchronous I/O operations. It's responsible for managing the execution of code in response to events, allowing Node.js to efficiently handle multiple concurrent connections without blocking the main thread.

Here is how the event loop works:

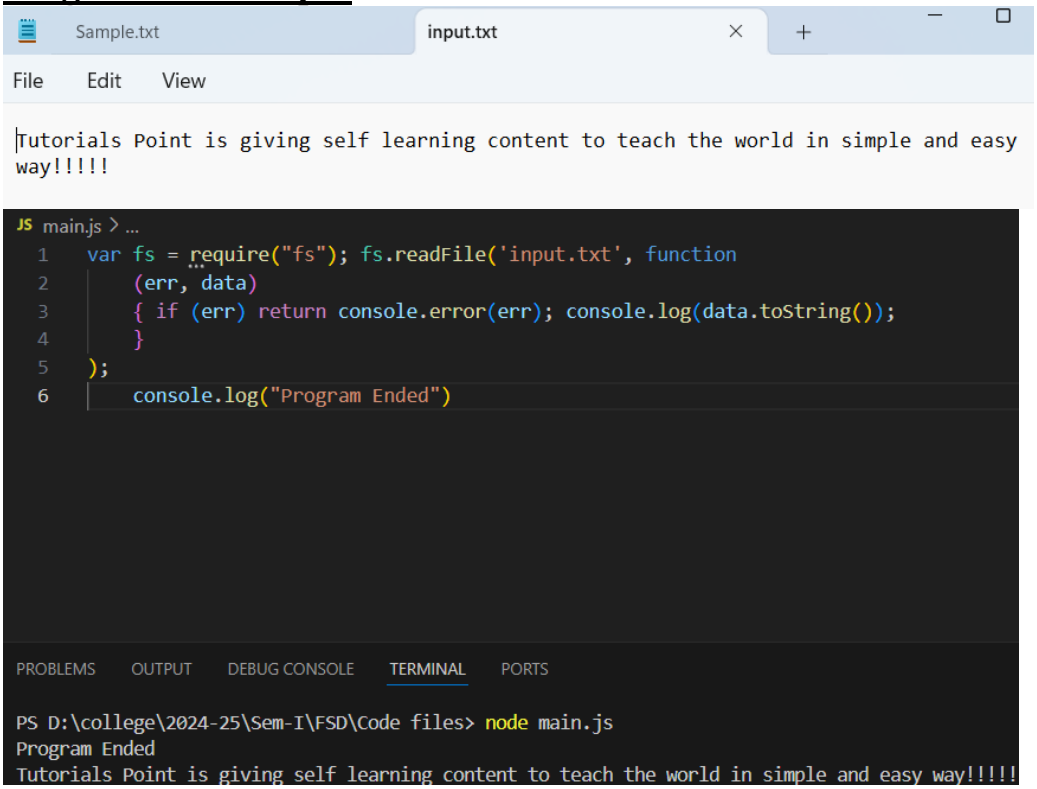
- **Event Queue:** When asynchronous operations (such as reading files, making network requests, or handling user input) are initiated, they are placed in an event queue.
- **Event Loop:** The event loop continually checks the event queue to see if there are any events (such as callbacks or promises) that need to be executed.
- **Non-Blocking:** While waiting for I/O operations to complete, the event loop allows other code to run, ensuring that the application remains responsive and doesn't block.
- **Callbacks:** Callback functions are often used in Node.js to handle events when operations are completed. When an event is ready to be processed, its associated callback function is executed.

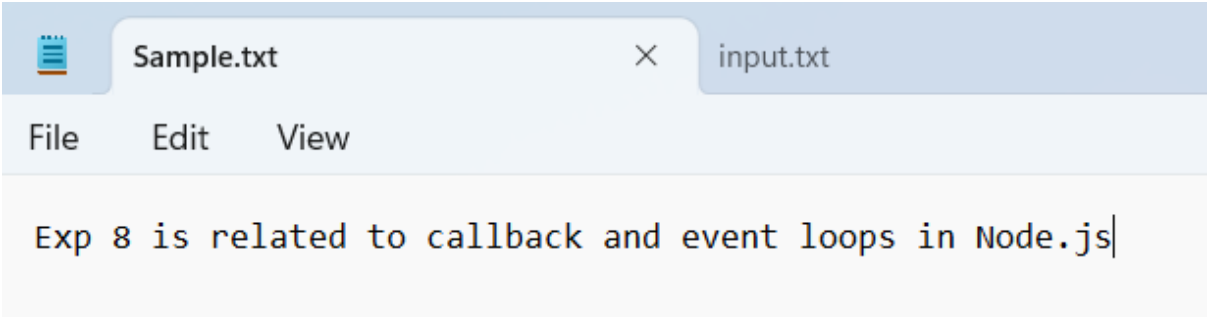
Example of an event loop in Node.js:

const fs = require('fs');
// Asynchronous file read operation fs.readFile('file.txt', 'utf8', (err, data) => { if (err) throw err; console.log(data);
});
console.log('Reading file...');

In this example, the file read operation is asynchronous, and the callback function is executed once the operation is complete. While waiting for the I/O operation to finish, other code (e.g., the console.log) can run without blocking.

Program with Output





App.js > ...

```
1   const fs = require('fs');
2   function readFileCallback(err, data){
3   if(err){
4   console.error('Error reading the file:', err);
5   }
6   else{
7   console.log('File content:', data);
8   }
9   }
10  fs.readFile('sample.txt', 'utf8', readFileCallback);
11  console.log('Reading the file...');
12  console.log('Main Program continues...');
13  console.log('Node.js Program started')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\college\2024-25\Sem-I\FSD\Code files> node App.js
Reading the file...
Main Program continues...
Node.js Program started
File content: Exp 8 is related to callback and event loops in Node.js
```

Conclusion:
Thus, we have demonstrated how we can make use of Callbacks and Event Loops in Node.js.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Create an application to demonstrate various Node.js Events.

PREREQUISITE: Node JS**.**

Theory / Algorithm / Conceptual Description:

The Event Loop and Emitters are fundamental concepts in Node.js, which make it an efficient and event driven environment for server-side programming. Let's explore these concepts in detail:

1. **Event Loop**

   The event loop is a critical component of Node.js that enables non-blocking, asynchronous I/O operations. It's responsible for managing the execution of code in response to events, allowing Node.js to efficiently handle multiple concurrent connections without blocking the main thread.

   Here is how the event loop works:

   - **Event Queue:** When asynchronous operations (such as reading files, making network requests, or handling user input) are initiated, they are placed in an event queue.
   - **Event Loop:** The event loop continually checks the event queue to see if there are any events (such as callbacks or promises) that need to be executed.
   - **Non-Blocking:** While waiting for I/O operations to complete, the event loop allows other code to run, ensuring that the application remains responsive and doesn't block.
   - **Callbacks:** Callback functions are often used in Node.js to handle events when operations are completed. When an event is ready to be processed, its associated callback function is executed.

Example of an event loop in Node.js:

```
const fs = require('fs');
// Asynchronous file read operation fs.readFile('file.txt', 'utf8', (err, data) => { if (err) throw err; console.log(data);
});
console.log('Reading file...');
```

In this example, the file read operation is asynchronous, and the callback function is executed once the operation is complete. While waiting for the I/O operation to finish, other code (e.g., the console.log) can run without blocking.

2. **Emitters:**

   In Node.js, emitters are objects that can emit named events and allow functions

(listeners) to be attached to those events. These emitters are the foundation for building custom event-driven systems. The EventEmitter class in Node.js provides this functionality.

- **Emitting Events:** An emitter can emit events using the emit method, specifying the event name and optional data.
- **Listening for Events**: You can attach listeners to an emitter to execute code when a specific event occurs. Use the on or addListener method to add event listeners.

Example using the EventEmitter:

const EventEmitter = require('events'); class MyEmitter extends EventEmitter {} const myEmitter = new
MyEmitter();
// Adding an event listener myEmitter.on('myEvent', () => { console.log('Event occurred!'); });
// Emitting the event myEmitter.emit('myEvent');

In this example, the myEmitter object emits the 'myEvent' event, and the attached listener responds by logging a message.

Emitters are widely used in Node.js for building custom modules, handling events in web servers, and managing asynchronous operations.

These two concepts, the Event Loop and Emitters, are central to Node.js's ability to handle concurrent, non- blocking operations efficiently. They enable the development of scalable and performant applications, particularly in scenarios where handling many connections or asynchronous I/O operations is essential

Program

emitter.js > ...

```
1   var events = require('events');
2   // Create an eventEmitter object
3   var eventEmitter = new events.EventEmitter();
4   // Create an event handler as follows
5   var connectHandler = function connected() {
6   console.log('connection succesful.');
7   // Fire the data_received event
8   eventEmitter.emit('data_received');
9   }
10  // Bind the connection event with the handler
11  eventEmitter.on('connection', connectHandler);
12  // Bind the data_received event with the anonymous function
13  eventEmitter.on('data_received', function() {
14  console.log('data received succesfully.');
15  });
16  // Fire the connection event
17  eventEmitter.emit('connection');
18  console.log("Program Ended.");
```

Output

```
PS D:\college\2024-25\Sem-I\FSD\Code files> node emitter.js
connection succesful.
data received succesfully.
Program Ended.
```

Conclusion:
Thus, we have demonstrated the usage of Events and Event Emitters in Node.js.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Create an application to demonstrate various Node.js Functions.

PREREQUISITE: Node JS**.**

Theory / Algorithm / Conceptual Description:

Introduction to Functions:
In Node.js, functions are blocks of reusable code designed to perform a specific task. They encapsulate a set of instructions and can accept parameters (inputs) and return values (outputs).
Function Declaration:
function greet(name) { return `Hello, ${name}!`; }

- **'function' Keyword:** Used to declare a function.
- **Function Name:** Identifies the function.
- **Parameters:** Input values passed to the function.
- **Return Statement:** Specifies the value returned by the function.

Callback Functions:
In Node.js, callback functions are crucial for handling asynchronous operations. They are functions passed as arguments to other functions, to be executed once an asynchronous operation completes. For example, when reading a file or making a network request, the callback handles the retrieved data or an error.

Higher-Order Functions:
Higher-order functions are functions that can take other functions as arguments or return them as results. They enable functional programming paradigms, allowing for more concise and expressive code.

Arrow Functions:
Arrow functions provide a concise syntax for defining functions, especially useful for inline functions and maintaining the lexical context of this.
const greet = (name) => `Hello, ${name}!`;

- **Arrow Function Syntax:** A concise way to define functions using the => arrow.
- **Implicit Return:** If the function body is a single expression, it's implicitly returned without using the return keyword.

Default Parameters:
In Node.js, you can define default parameter values for function parameters. If an argument isn't provided, the default value is used.
Rest Parameters:
Rest parameters allow functions to accept an indefinite number of arguments as an array, making it flexible to handle varying numbers of parameters.
Functions in Node.js serve as essential building blocks for structuring code, encapsulating logic, and promoting reusability. Understanding the fundamentals of functions and basic arithmetic operations forms a cornerstone for developing efficient and well-structured Node.js applications. Functions are fundamental in Node.js, enabling asynchronous operations, providing flexibility through higher-order functions, and simplifying syntax with features like arrow functions, default parameters, and rest parameters. Mastering functions in Node.js is crucial for building scalable and efficient applications.

Program

Exp10.js > ...

```
1   const readline = require('readline');
2   const rl = readline.createInterface({
3   input:process.stdin,
4   output:process.stdout,
5   });
6
7   function add(a,b){
8   a=parseInt(a);
9   b=parseInt(b);
10  return a+b;
11  }
12  function subtract(a,b){
13  return a-b;
14  }
15  function multiply(a,b){
16  return a*b;
17  }
18  function divide(a,b){
19  return a/b;
20  }
21
22  function performOperation(){
23  rl.question('Enter first number:',(num1) => {
24  rl.question('Enter second number:',(num2) => {
25  console.log('Sum: ', add(num1,num2));
26  console.log('Difference: ', subtract(num1,num2));
27  console.log('Product: ',multiply(num1,num2));
28  console.log('Quotient: ',divide(num1,num2));
29  rl.close();
30  });
31  });
32  }
33  performOperation();
```

Output

```
PS D:\college\2024-25\Sem-I\FSD\Code files> node Exp10.js
Enter first number:50
Enter second number:20
Sum:  70
Difference:  30
Product:  1000
Quotient:  2.5
```

Conclusion:
Thus, we have demonstrated the use of different functions in Node.js.

PRN: **Roll No.:**

Name: **Batch:**

Date of Performance: ___/___/2024 **Date of Submission:** ___/___/2024

TITLE: Installation and Configuration of MongoDB (Content

PREREQUISITE: Basic knowledge of software installation.

Theory / Algorithm / Conceptual Description:

What is MongoDB?

MongoDB is an open-source, document-oriented NoSQL database management system. It is designed for flexibility, scalability, and performance in handling unstructured or semi-structured data. It was created by a company called 10gen, which is now known as MongoDB, Inc. The company was founded by Eliot Horowitz and Dwight Merriman in 2007. The first version of MongoDB was released in 2009.

Differences between MongoDB & SQL:

SQL Database	NoSQL Database (MongoDB)
Is Relational database	Is Non-relational database
It supports SQL query language	It supports JSON query language
It is generally Table-based structure	It is mostly Collection-based and key-value pair structure
It follows Row-based structure	It follows Document-based structure
It follows Column-based structure	It follows Field-based structure
It supports foreign keys	It does not support foreign keys
It supports triggers	It does not support triggers
Schema is predefined	Schema is dynamic
Not good to use for hierarchical data storage	Good to use for hierarchical data storage
Due to Vertically scalability –user can increase RAM	Due to Horizontally scalability – user can add more servers
Highlights on ACID properties (Atomicity, Consistency, Isolation and Durability)	Highlights on CAP theorem (Consistency, Availability and Partition tolerance)
Tools used are Microsoft SQL Server, PostgreSQL, MySQL, Oracle, etc	Tools used are MongoDB, Cassandra, CouchDB, Bigtable, FlockDB, ArangoDB, etc

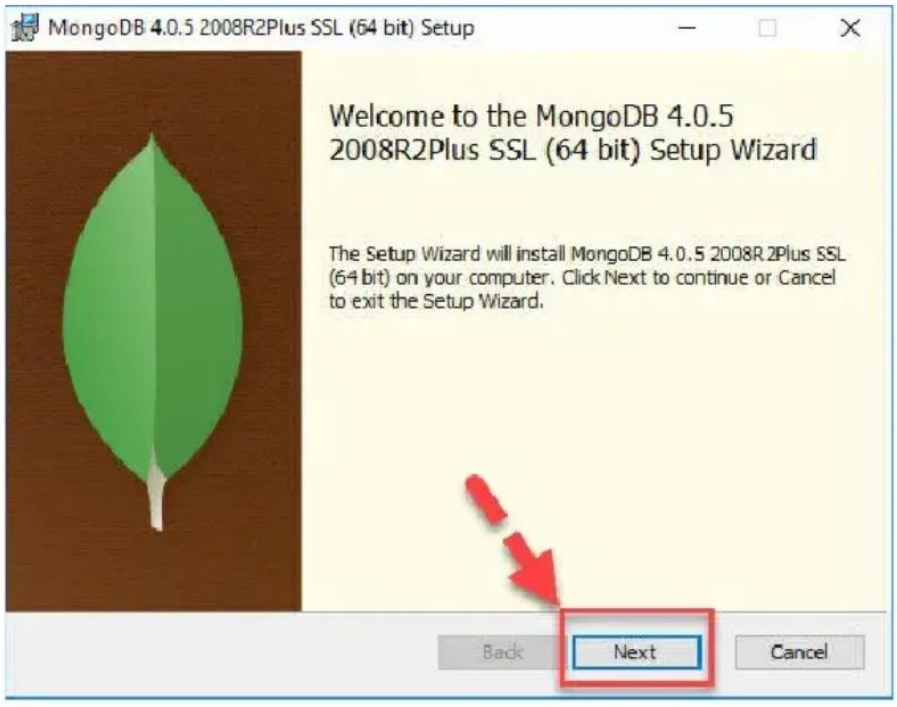
Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements: Ubuntu 14.04, Mongodb Packages Theory: MongoDB is a free and open-source NoSQL document database used commonly in modern web applications. MongoDB works on concept of collection and document.

Installation-
Download & Install MongoDB on Windows

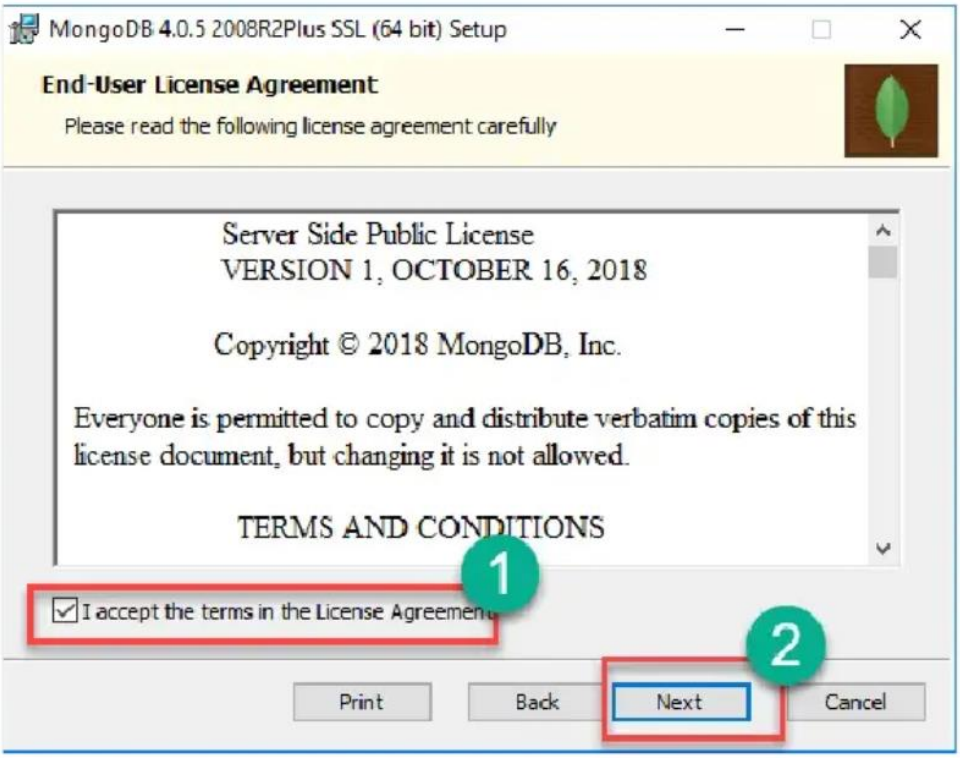
Step 1) Go to **https://www.mongodb.com/try/download/community** and Download MongoDB Community Server. We will install the 64-bit version for Windows.

Step 2) Once download is complete open the msi file. Click Next in the start up screen

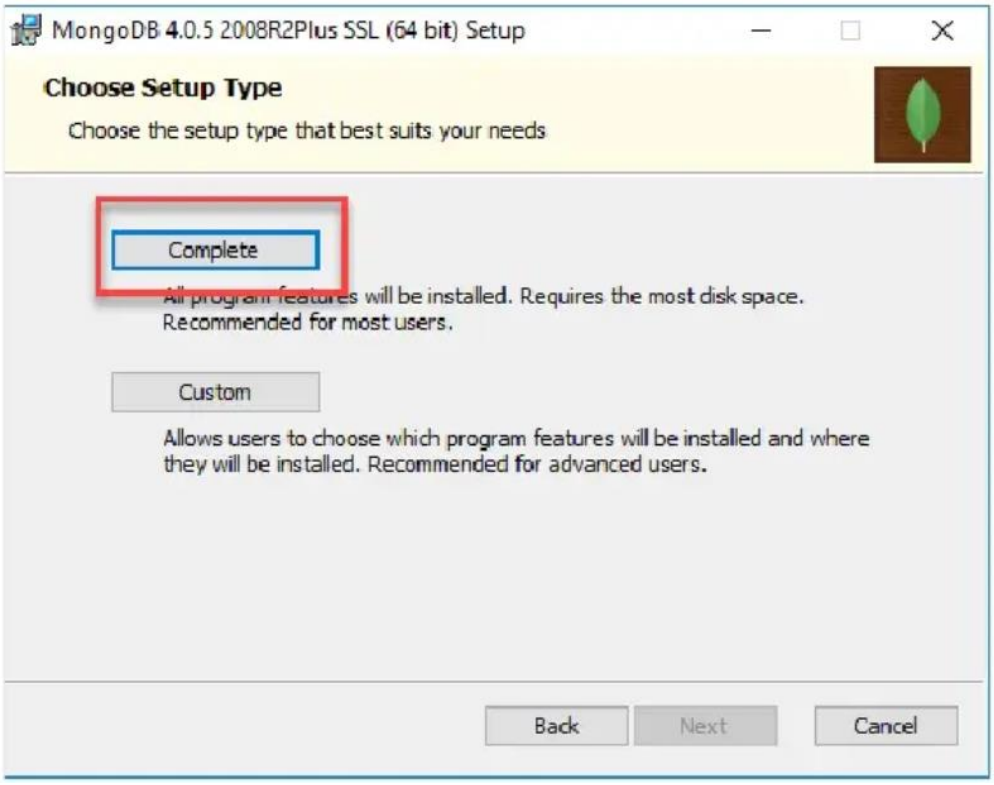


Step 3)

1. Click End-User license agreement.
2. Click Next.

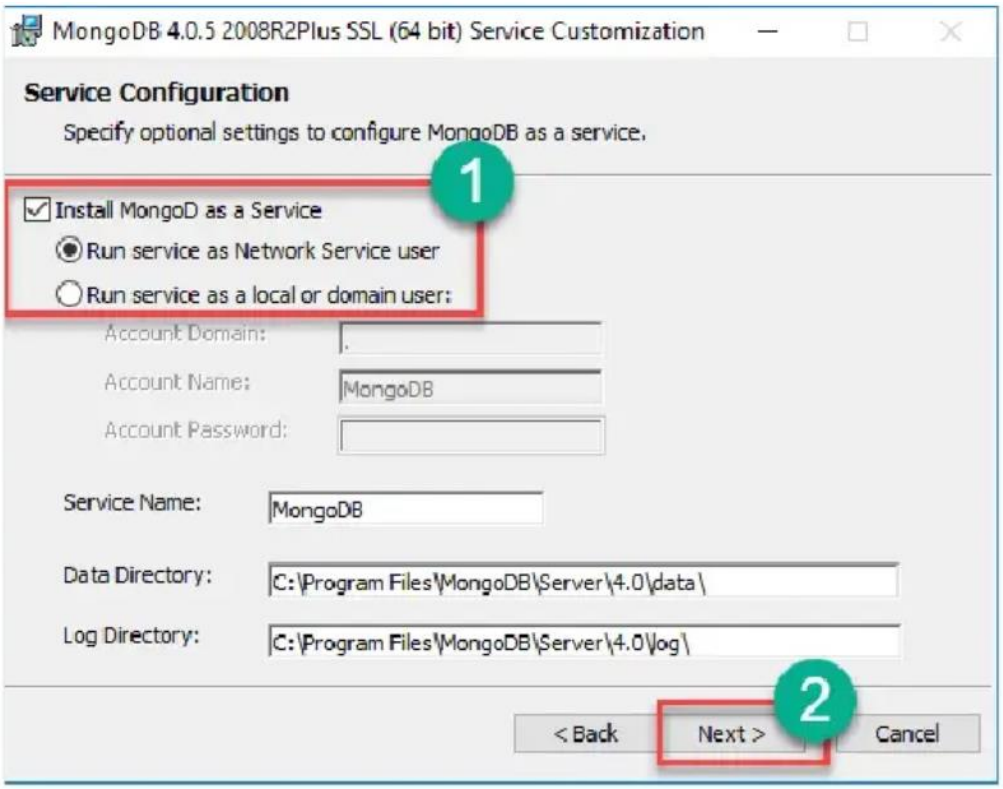


Step 4) Click on the "complete" button to install all of the components. The custom option can be used to install selective components or if you want to change the location of the installation.

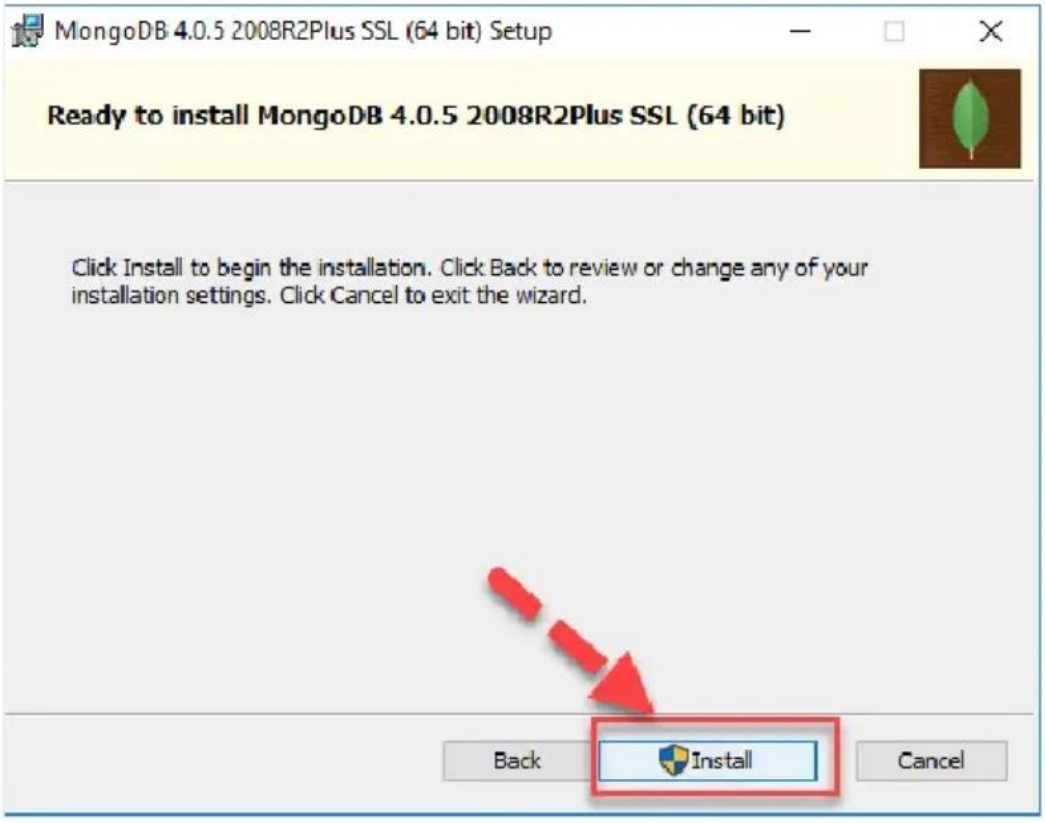


Step 5)

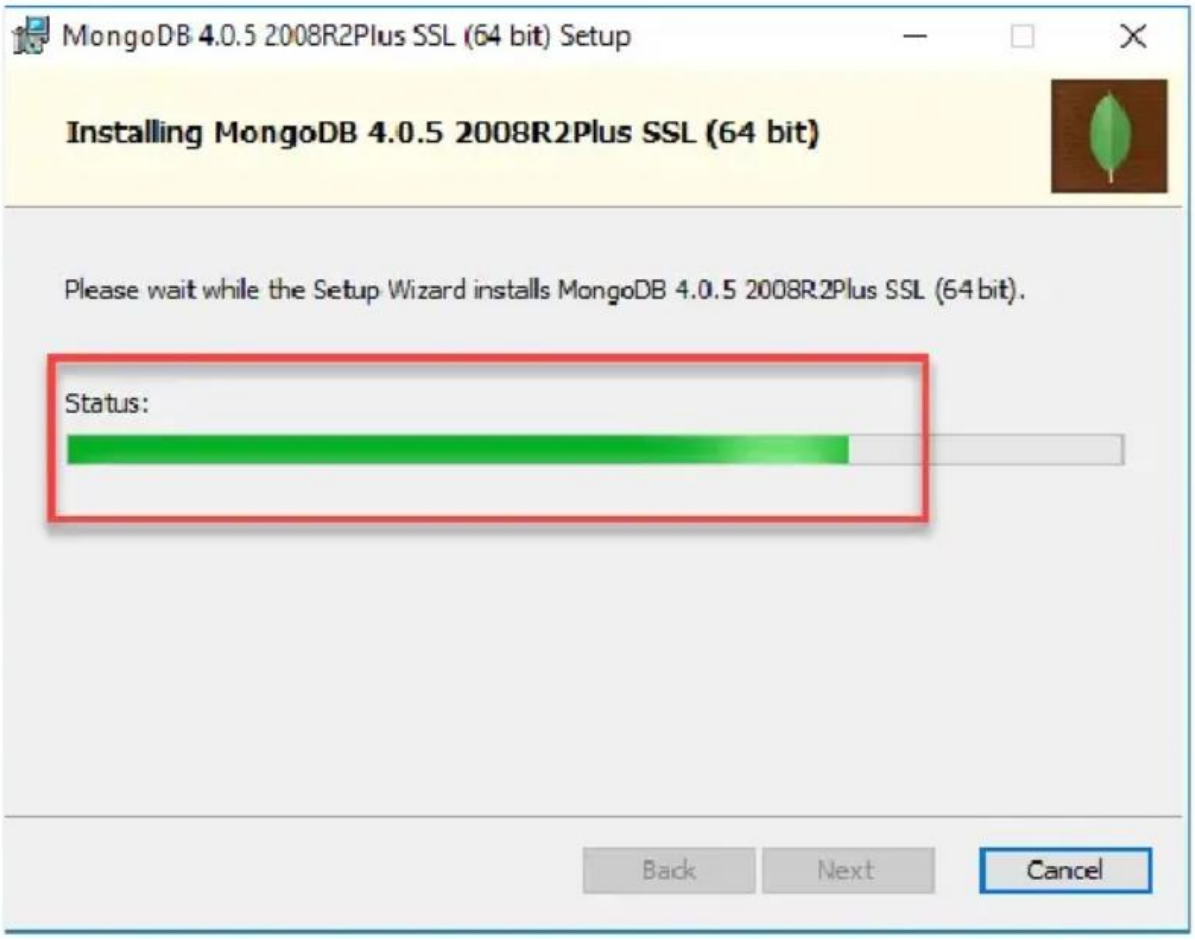
1. Select “Run service as Network Service user”. make a note of the data directory, we’ll need this later.
2. Click Next



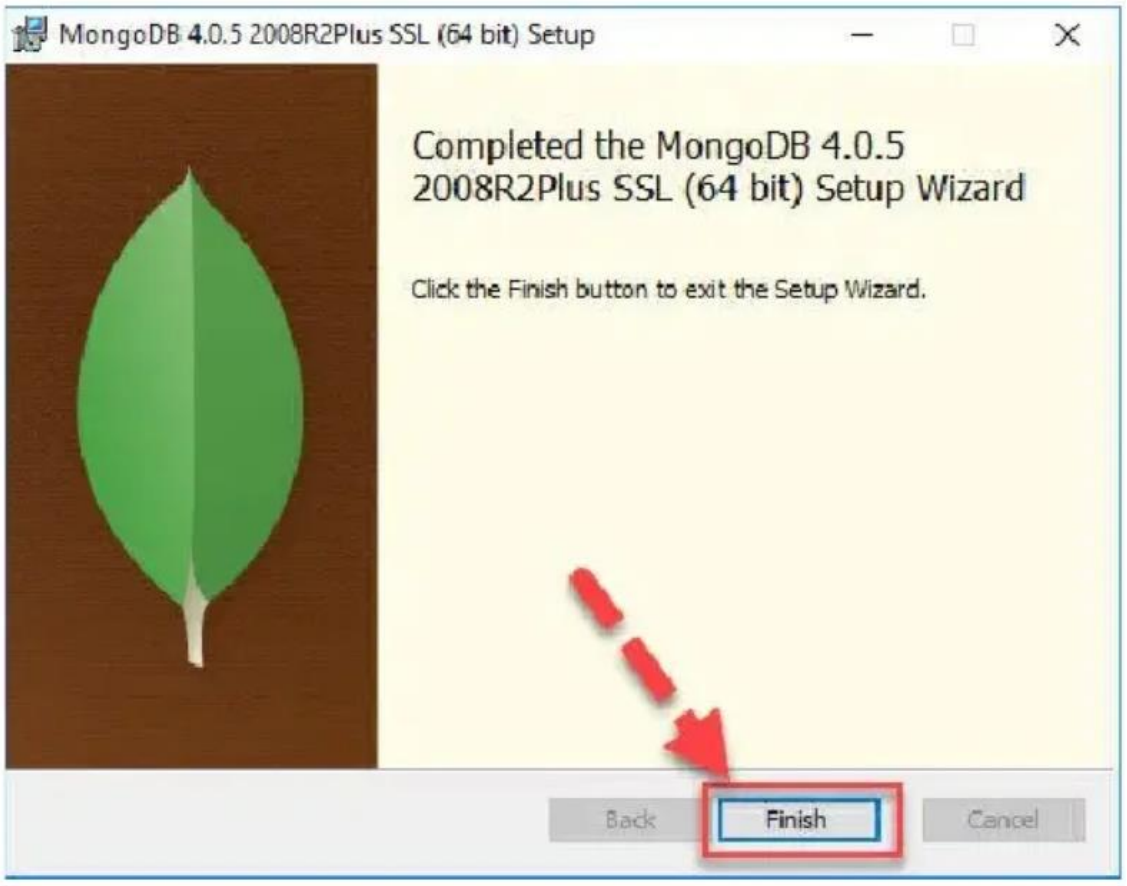
Step 6) Click on the Install button to start the installation.



Step 7) Installation begins. Click Next once completed



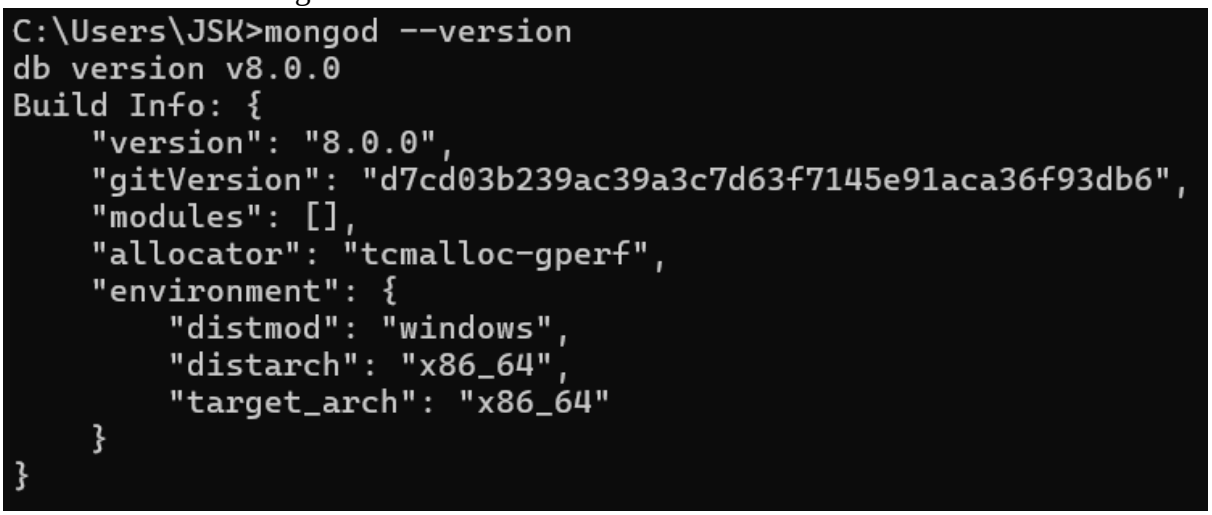
Step 8) Click on the Finish button to complete the installation



Step 9: Verify that MongoDB was properly installed or not.

To check that MongoDB was completely installed on your system or not, you can run the following command in your command prompt test it:-

If MongoDB was completely installed on your system, the command prompt will print the version of the MongoDB installed.



Conclusion:
In this experiment, we have installed and configured MongoDB onto our system.