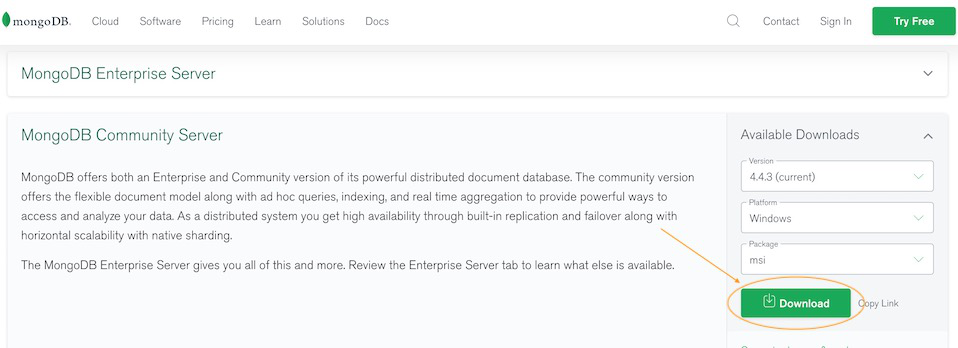
**EXPERIMENT NUMBER – 1**

**AIM:** Installation, configuration, and connection establishment of MongoDB

**DESCRIPTION:**

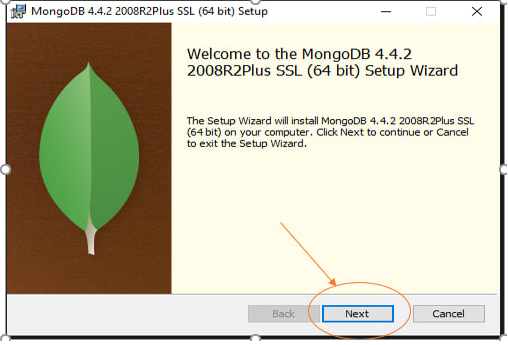
MongoDB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with that data very efficiently. It is categorized under the NoSQL (Not only SQL) database because the storage and retrieval of data in MongoDB are not in the form of tables.

**PROCEDURE:**Step-1: Go to MongoDB Download Center to download MongoDB Community Server.

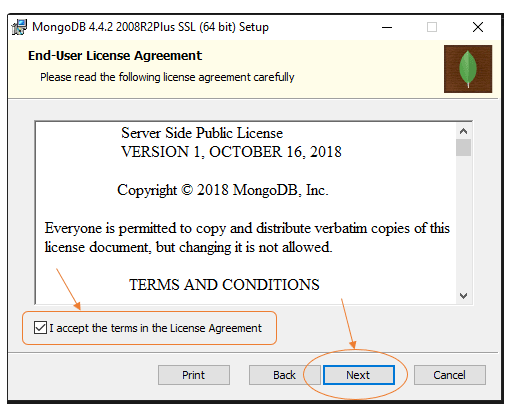
Here, we can select any version, Windows, and package according to our requirement. For Windows, we need to choose:

* Version: 4.2.2
* OS: WindowsOS
* Package: msi

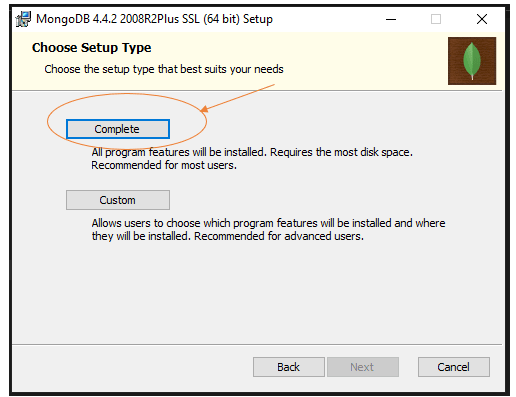
Step-2: When the download is complete open the msi file and click the next button in the startup screen:



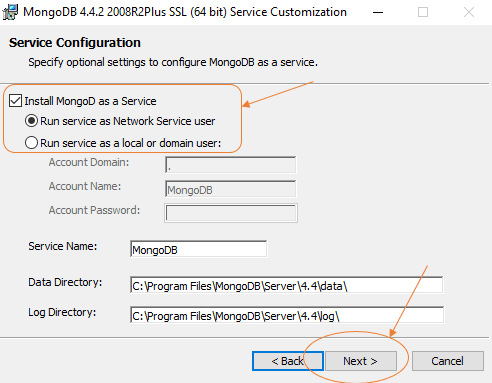
\Step-3: Now accept the End-User License Agreement and click the next button:



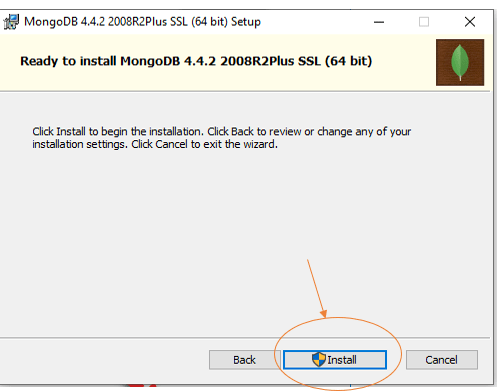
Step-4: Now select the complete option to install all the program features. Here, if we can want to install only selected program features and want to select the location of the installation, then use the Custom option:



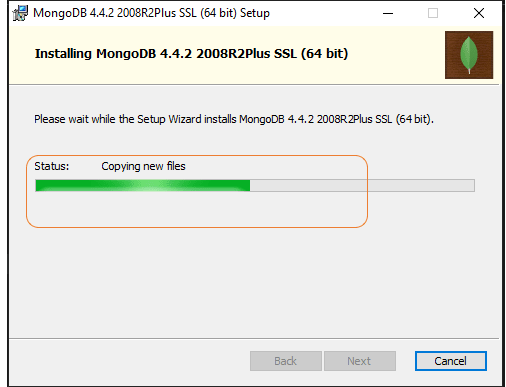
Step-5: Select “Run service as Network Service user” and copy the path of the directory. Click Next:



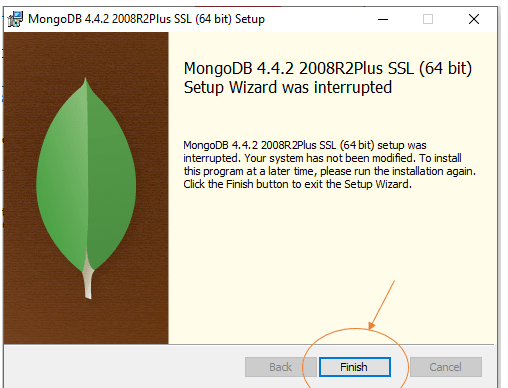
Step-6: Click the Install button to start the installation process:

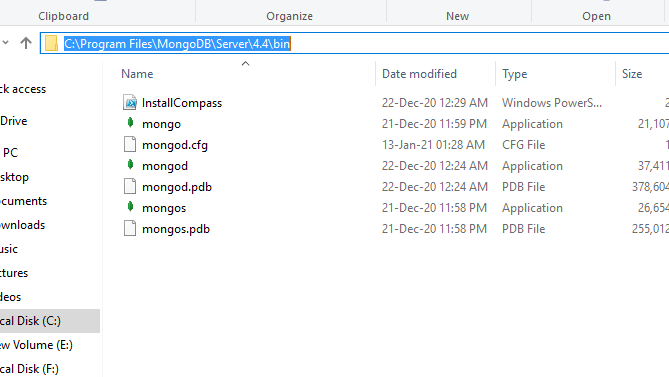


Step-7: After clicking on the install button installation of MongoDB begins:

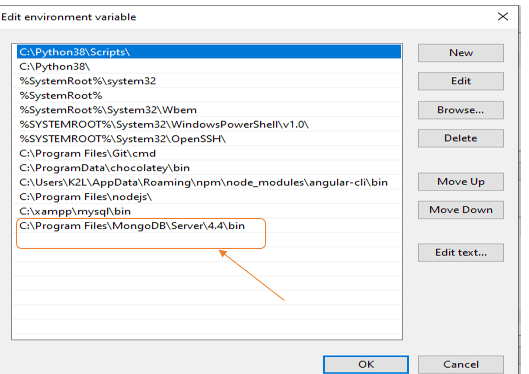


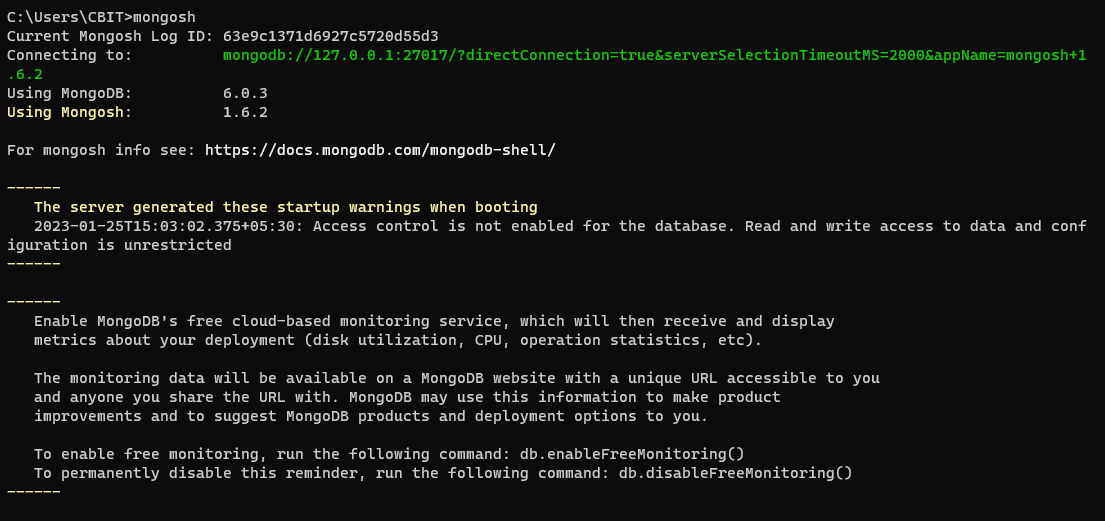
Step-8: Now click the Finish Button to complete the installation process:



Step-9: Now we go to location where MongoDB installed in step-5 in our system and copy the bin path:

Step-10: Now, to create an environment variable open system properties << Environment Variable << System variable << path << Edit Environment variable and paste the copied link to your environment system and click Ok:



Step-11: Now enter the mongosh in the terminal

**CONCLUSION:**

We have successfully installed, configured and have successfully established the connection of MongoDB.

**EXPERIMENT NUMBER – 2**

**AIM:** CRUD Operations on MongoDB

**DESCRIPTION:**

**1. Create Operations:**

The 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. We can perform, create operations using the following methods provided by the MongoDB:

**i. db.createCollection():** It is used to create an empty collection

**ii. db.collection.insertOne():** It is used to insert a single document in the collection

**iii. db.collection.insertMany():** It is used to insert a multiple documents in the 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. We can perform read operation using the following method provided by the MongoDB:

**i. db.collection.find():** It is used to retrieve documents from the collection.

**3. Update Operations:**

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

**i. db.collection.updateOne():** It is used to update a single document in the collection that satisfy the given criteria.

**ii. db.collection.updateMany():** It is used to update multiple documents in the collection that satisfy the given criteria.

**4. Delete Operations:**

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

**i. db.collection.deleteOne():** It is used to delete a single document from the collection that satisfy the given criteria

**ii. db.collection.deleteMany():** It is used to delete multiple documents from the collection that satisfy the given criteria.

**QUERIES AND OUTPUTS:  
1. Use:**

**Description:** This command will create a new database if it doesn’t exist, otherwise it will return the existing database.

**Syntax:** use DATABASE\_NAME

**Query:** use mongo\_practice012  
**Output:**



**2. db.createCollection():**  
**Description:** This command creates a new collection. This method is used primarily for creating new collections that use specific options

**Syntax:** db.createCollection(name, option)

**Query:** db.createCollection(“movies”)  
**Output:**



**3. db.collection.insertOne():**  
**Description:** This command inserts a single document into a collection

**Syntax:** db.collection.insertOne(

<document>,

{

writeConcern: <document>

}

)

**Query:** db.movies.insertOne({

title : "Fight Club",

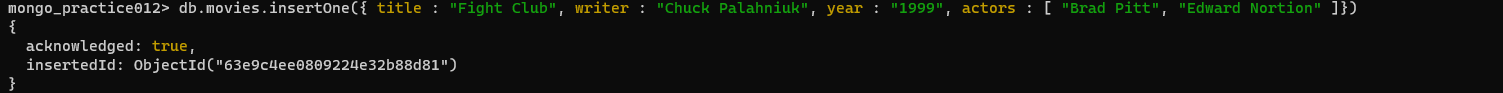
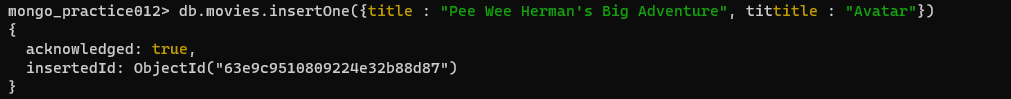
writer : "Chuck Palahniuk",

year : "1999",

actors : [

"Brad Pitt",

"Edward Norton"],

})  
**Output:**

**4. db.collection.insertMany():**  
**Description:** This command inserts multiple documents into a collection

**Syntax:** db.collection.insertMany(

[ <document 1> , <document 2>, ... ],

{

writeConcern: <document>,

ordered: <boolean>

}

)

**Query:** db.mongo\_practice.insertMany([

{

title : "The Hobbit: The Desolation of Smaug",

writer : "J.R.R. Tolkein",

year : 2013,

franchise : "The Hobbit",

},

{

title : "The Hobbit: The Battle of the Five Armies",

writer : "J.R.R. Tolkein",

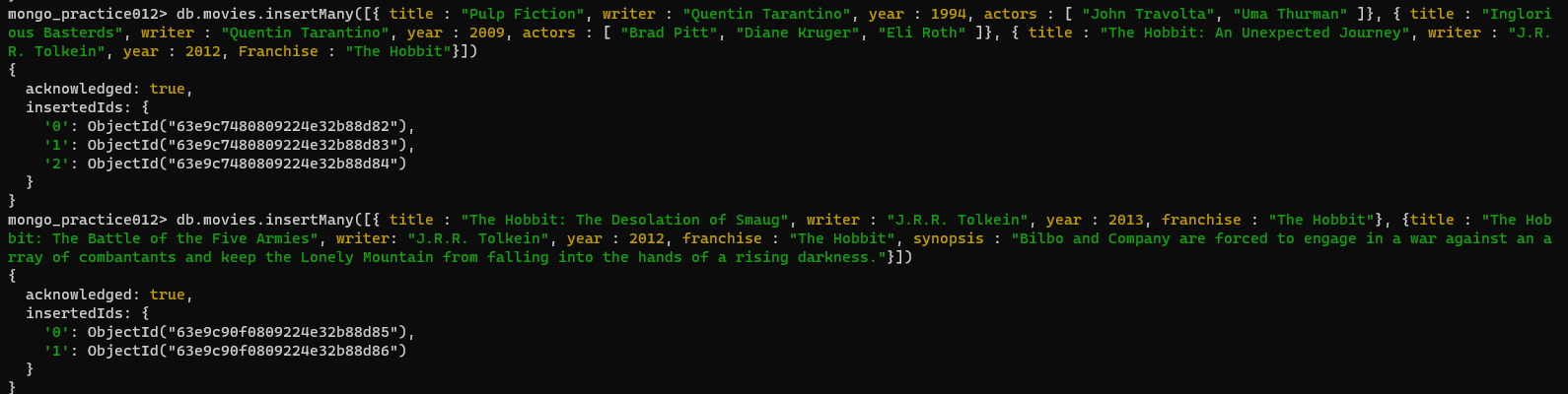
year : 2012,

franchise : "The Hobbit",

synopsis : "Bilbo and Company are forced to engage in a war against an array of combatants and keep the Lonely Mountain from falling into the hands of a rising darkness.",

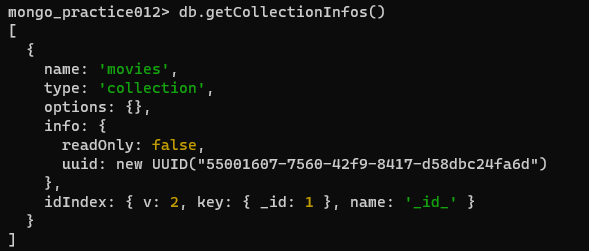
},

],

)  
**Output:**

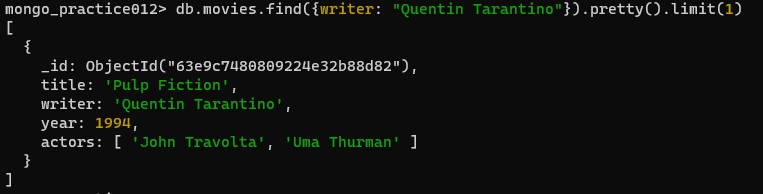
**5. Write a query to display all the documents**  
**Description:** In this query, getcollectionInfos() command can be used. This command returns an array of documents with collection or view information, such as name and options, for the current database

**Syntax:** db.getCollectionInfos(filter, nameOnly, authorizedCollections)

**Query:** db.getCollectionInfos()  
**Output:**

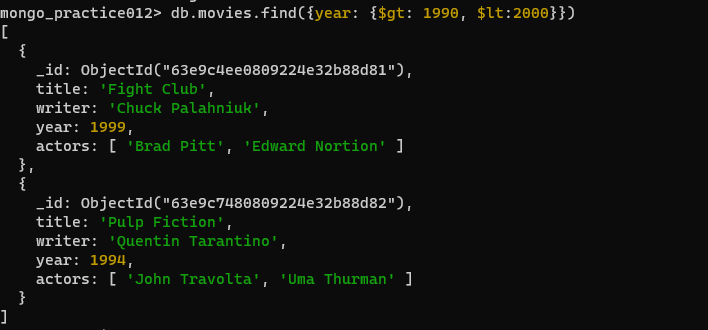
**6. Write a query to get all documents with writer set to "Quentin Tarantino" and restrict the result set to one object  
Description:** In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

**Syntax:** db.collection.find(query, projection, options)

**Query:** db.movies.find({writer:"Quentin Tarantino"}).pretty().limit(1)  
**Output:**

**7. Write a query to get all movies released in the 1990s**

**Description:** In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

**Syntax:** db.collection.find(query, projection, options)  
**Query:** db.movies.find({year:{$gt:"1990", $lt:"2000"}})  
**Output:**

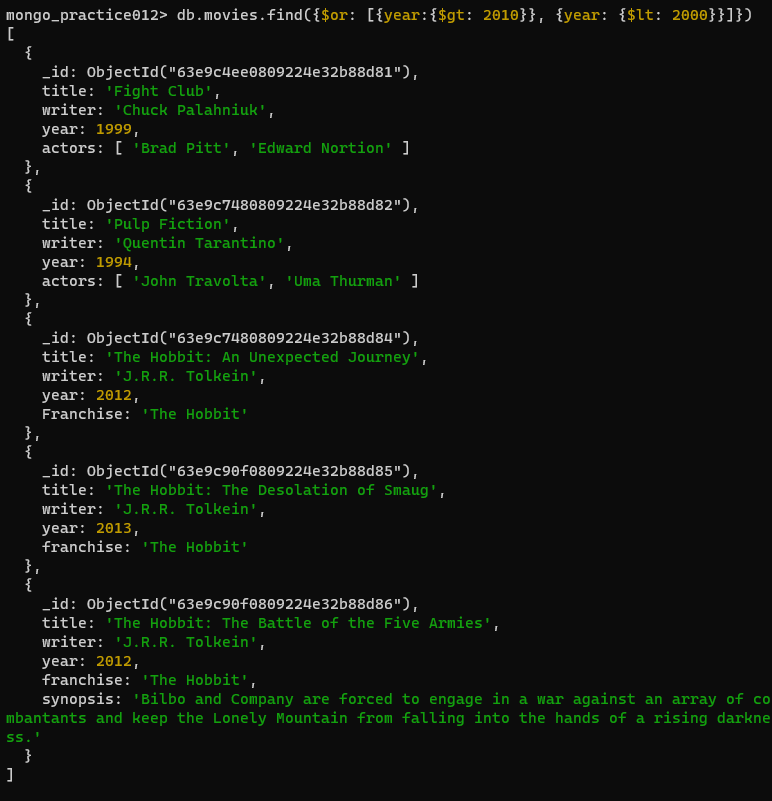
**8. Write a query to get all movies released before this year 2000 or after 2010**

**Description:** In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

**Syntax:** db.collection.find(query, projection, options)

**Query:** db.movies.find({$or:[{year:{$gt:"2010"}},{year: {$lt:"2000"}}]})

**Output:**



**9. Write a query to find the objected for movie “Pulp Fiction”, add an actor named “Samuel L. Jackson” and display the result.**

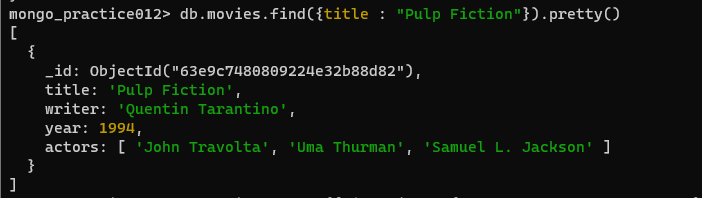
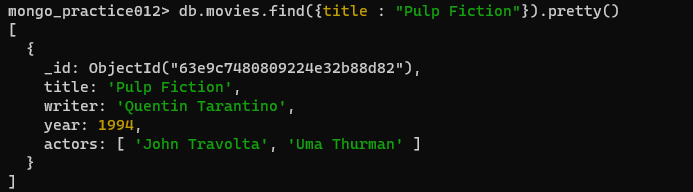
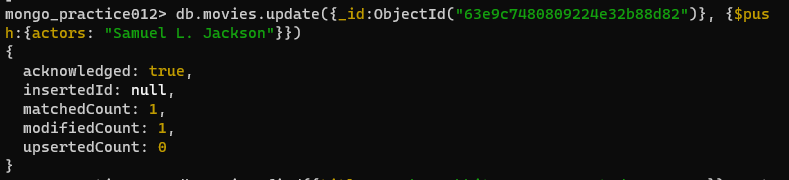
**Description:** In this query db.collection.update() command can be used. This command modifies an existing fields of an existing document or documents or replace an existing document entirely, depending on the update parameter

**Syntax:** db.collection.update(query, update, options)

**Query:**

db.movies.find({field:"value"}).pretty()

db.movies.update({\_id:ObjectId(" ")}, {$push:{actors:"Samuel L. Jackson"}})

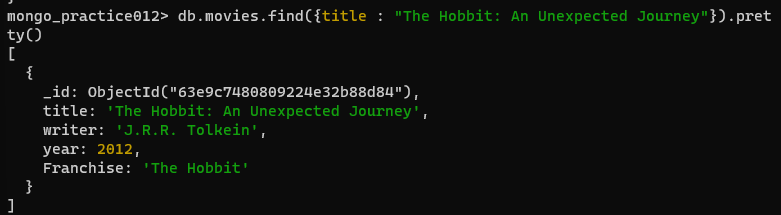
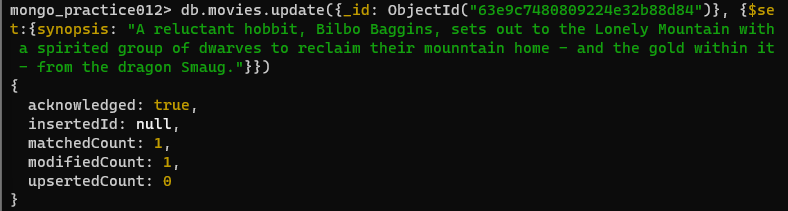
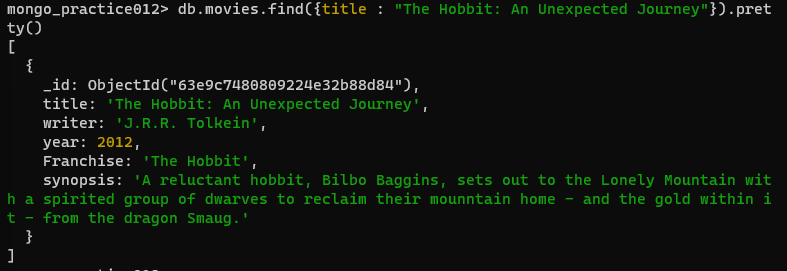
**Output:**

**10. Write a query add a synopsis to "The Hobbit: An Unexpected Journey" : "A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."**

**Description:** In this query db.collection.update() command can be used. This command modifies an existing fields of an existing document or documents or replace an existing document entirely, depending on the update parameter

**Syntax:** db.collection.update(query, update, options)

**Query:** db.movies.update({\_id:ObjectId("5c9f98e5e5c2dfe9b3729bfe")}, {$set:{synopsis:"A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."}})

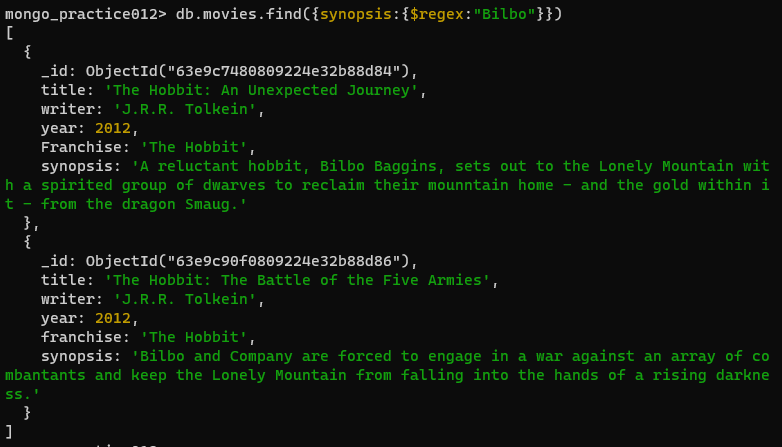
**Output:**

**11. Write a query to find all movies that have a synopsis that contains the word “Bilbo”**

**Description:** In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

**Syntax:** db.collection.find(query, projection, options)

**Query:** db.movies.find({synopsis:{$regex: “Bilbo”}})

**Output:**

**12. Write a query to find all movies that have a synopsis that contains the word “Gandalf”**

**Description:** In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

**Syntax:** db.collection.find(query, projection, options)

**Query:** db.movies.find({synopsis: {$regex: “Gandalf”}})

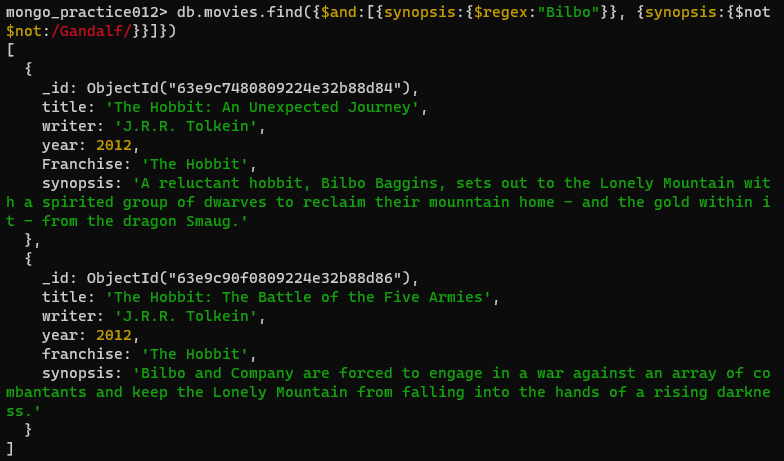
**Output:**

**13. Write a query to find all movies that have a synopsis that contains the word “Bilbo” and not the word “Gandalf”**

**Description:** In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

**Syntax:** db.collection.find(query, projection, options)

**Query:** db.movies.find({$and:[{synopsis:{$regex:"Bilbo"}}, {synopsis:{$not:/Gandalf/}}]})

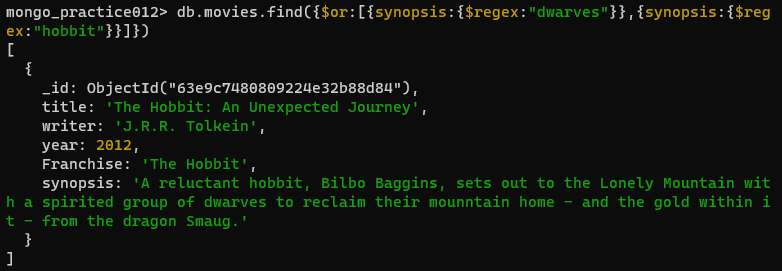
**Output:**

**14. Write a query to find all movies that a synopsis that contains the word “dwarves” or “hobbit”**

**Description:** In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

**Syntax:** db.collection.find(query, projection, options)

**Queries:** db.movies.find({$or:[{synopsis:{$regex:"dwarves"}}, {synopsis:{$regex:"hobbit"}}]})

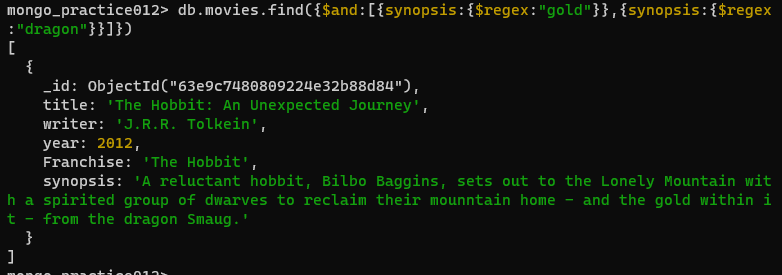
**Output:**

**15. Write a query to find all movies that have a synopsis that contains the word “gold” and “dragon”**

**Description:** In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

**Syntax:** db.collection.find(query, projection, options)

**Query:** db.movies.find({$and:[{synopsis:{$regex:"gold"}}, {synopsis:{$regex:"dragon"}}]})

**Output:**

**16. Write a query to delete the movie “Fee Wee Herman’s Big Adventure”**

**Description:** In this query, db.collectiondeleteOne() command is used. This command is used to delete at most a single document that matches a specified filter.

**Syntax:**

|  |
| --- |
| db.collection.deleteOne( |
| <filter>, |
| { |
| writeConcern: <document>, |
| collation: <document>, |
| hint: <document|string> // Available starting in MongoDB 4.4 |
| } |
| ) |

**Query:** db.movies.deleteOne({\_id: ObjectId(“ “)})

**Output:**

**17. Write a query to delete the movie “Avatar”**

**Description:** In this query, db.collectiondeleteOne() command is used. This command is used to delete at most a single document that matches a specified filter.

**Syntax:**

|  |
| --- |
| db.collection.deleteOne( |
| <filter>, |
| { |
| writeConcern: <document>, |
| collation: <document>, |
| hint: <document|string> // Available starting in MongoDB 4.4 |
| } |
| ) |

**Query:** db.movies.deleteOne({\_id: ObjectId(“ “)})

**Output:**

**CONCLUSION:**

We have successfully understood and executed the CRUD Operations on MongoDB

**EXPERIMENT NUMBER – 3**

**AIM:** Aggregation Operations in MongoDB

**DESCRIPTION:**

Aggregation is a way of processing a large number of documents in a collection by means of passing them through different stages.

The stages make up what is known as a pipeline. The stages in a pipeline can filter, sort, group, reshape and modify documents that pass through the pipeline.

One of the most common use cases of Aggregation is to calculate aggregate values for groups of documents.

This is similar to the basic aggregation available in SQL with the GROUP BY clause and COUNT, SUM and AVG functions.

To perform aggregation operations, we can use:

Aggregation pipelines,which are the preferred method for performing aggregations.

Single purpose aggregation methods,which are simple but lack the capabilities of an aggregation pipeline.

MongoDB Aggregation Pipelines:The pipeline consists of the following stages

Input-->$match-->$group-->$sort-->ouput

$match stage – filters those documents we need to work with, those that fit our needs

$group stage – does the aggregation job

$sort stage – sorts the resulting documents the way we require (ascending or descending)

Syntax:

db.collectionName.aggregate(pipeline, options)

collectionName – is the name of a collection,

pipeline – is an array that contains the aggregation stages,

options – optional parameters for the aggregation

Example:

pipeline = [

{ $match : { … } },

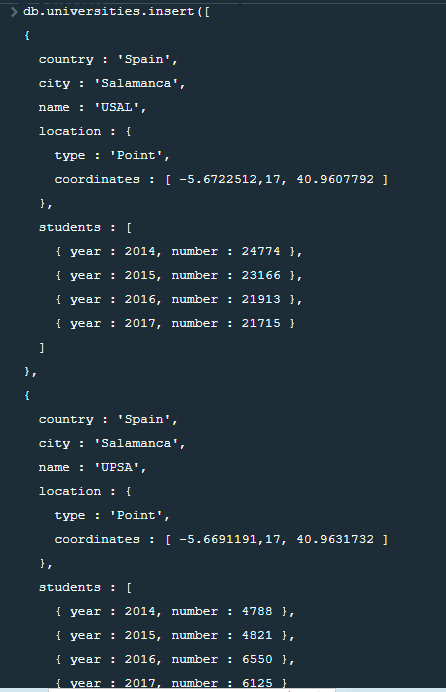
{ $group : { … } },

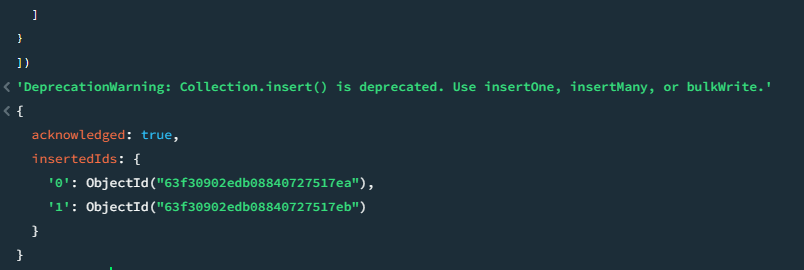
{ $sort : { … } }

]

**QUERIES AND OUTPUT:  
1. Create the Database Universities:**







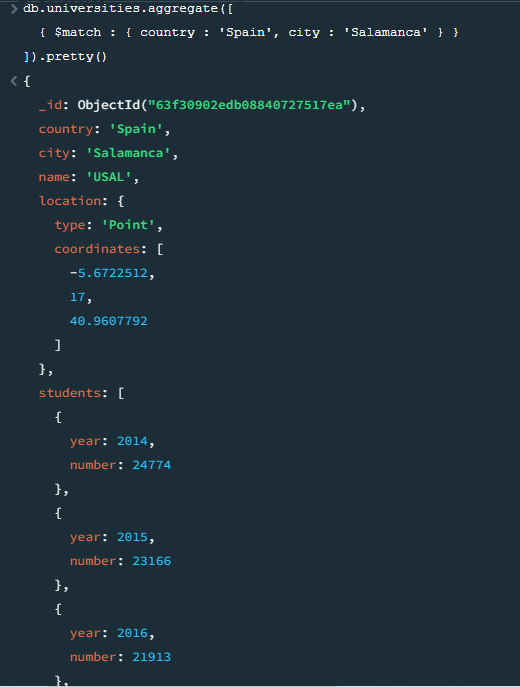


**2. MongoDB $match:**

**Description:** This command filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.

**Syntax:** { $match: { <query> } }

**Query:** db.universities.aggregate([ { $match : { country : ‘Spain’, city : ‘Salamanca’ } } ] ).pretty()

**Output:**

**3. MongoDB $project:**  
**Description:** This command passes along the documents with the requested fields to the next stage in the pipeline. The specified fields can be existing fields from the input documents or newly computed fields.

**Syntax:** { $project: { <specification(s)> } }

**Query:** db.universities.aggregate([ { $project : { \_id : 0, country : 1, city : 1, name : 1 } } ] ).pretty()

**Output:**



**4. MongoDB $group:**  
**Description:** The $group stage separates documents into groups according to a “group key”. The output is one document for each unique group key  
**Syntax:**

{

$group:

{

\_id: <expression>, // Group key

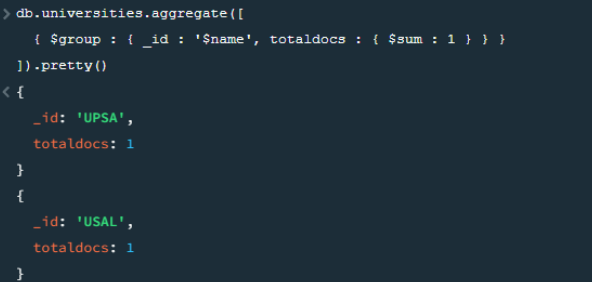
<field1>: { <accumulator1> : <expression1> },

...

}

}  
**Query:** db.universities.aggregate([ { $group : { \_id : ‘$name’, totaldocs : { $sum : 1 } } } ]).pretty()

**Output:**



**CONCLUSION:**

We have successfully understood and executed the Aggregation Operations on MongoDB

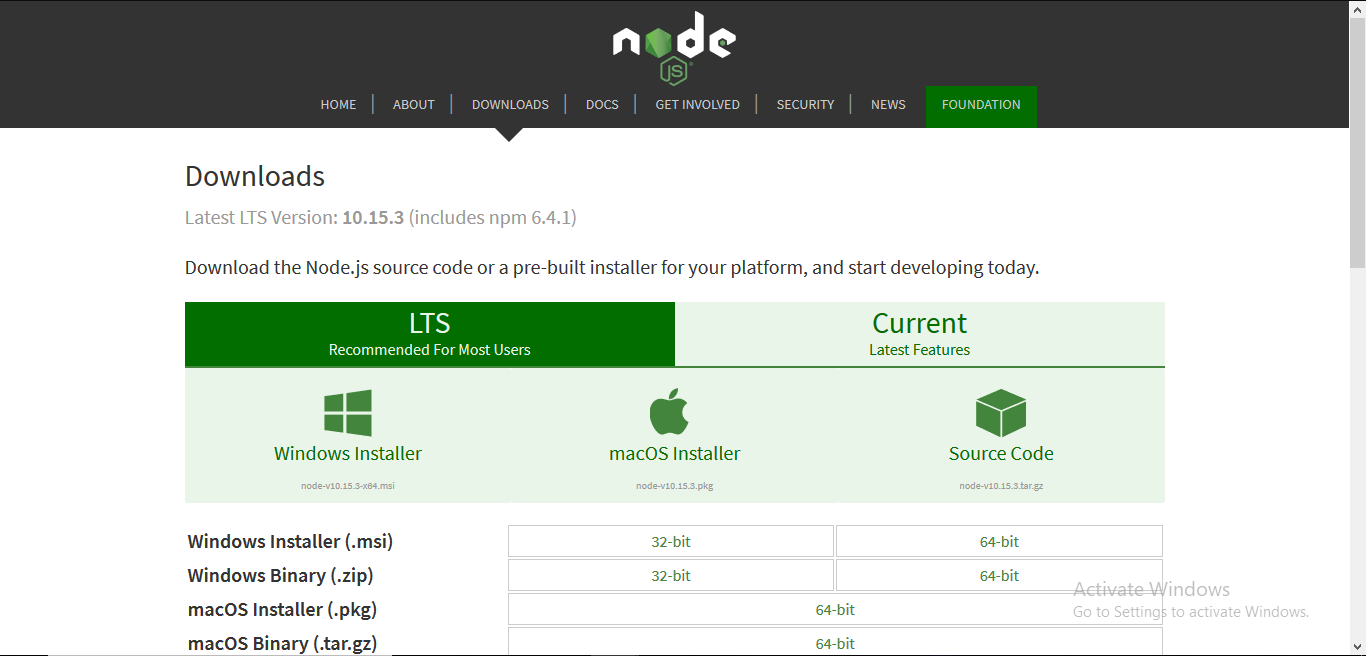
**EXPERIMENT NUMBER – 4**

**AIM:** Installation of Node.js on Windows

**DESCRIPTION:**

Node.js is a very powerful JavaScript-based platform built on Google Chrome's JavaScript V8 Engine. It is used to develop I/O intensive web applications like video streaming sites, single-page applications, and other web applications. Node.js is open source, completely free, and used by thousands of developers around the world.

**PROCEDURE:**  
**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.

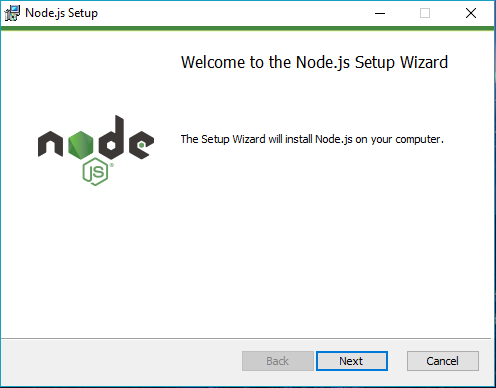
**Step-2:** Running the Node.js installer.

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

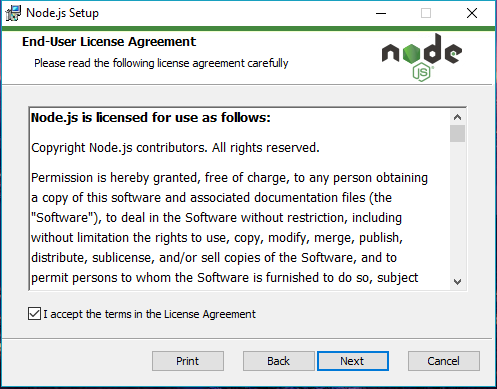


Select “Next”

* After clicking “Next”, End-User License Agreement (EULA) will open

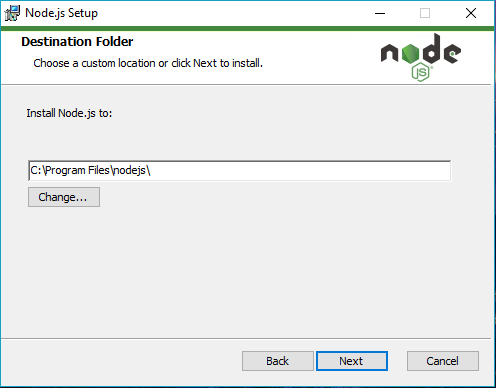
Check “I accept the terms in the License Agreement”

Select “Next”



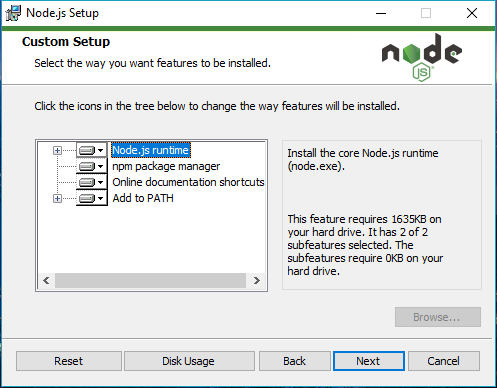
* Destination Folder

Set the Destination Folder where you want to install Node.js and Select “Next”



* Custom Setup

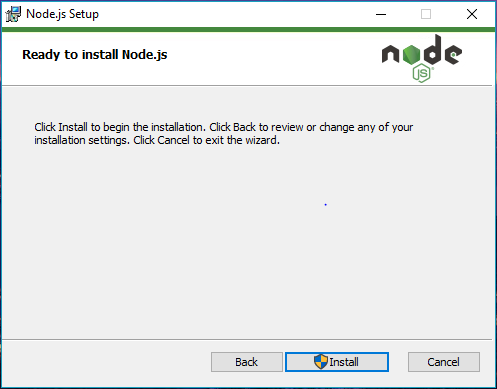
Select “Next”



* Ready to Install Node.js

The installer may prompt you to “install tools for native modules.”

Select “Install”

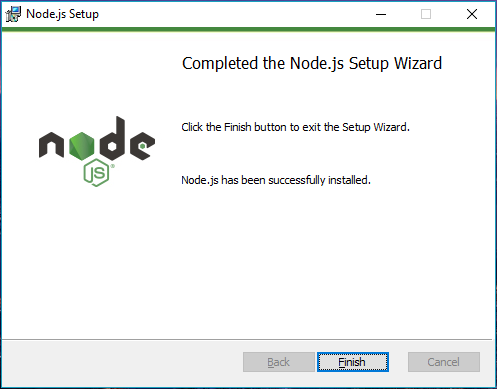


* Installing Node.js

Do not close or cancel the installer until the install is complete

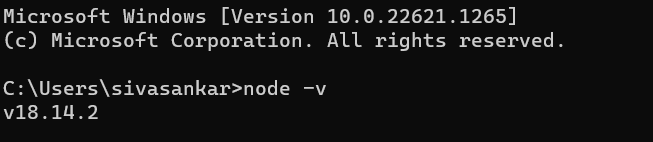
* Complete the Node.js Setup Wizard

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 :-



**CONCLUSION:**

We have successfully installed Node.js on Windows.

**EXPERIMENT NUMBER – 5**

**AIM:** To build a react code to create a simple search filter functionality to display a filtered list based on the search query entered by the user.  
  
**DESCRIPTION:**  
ReactJS is one of the most popular JavaScript front-end libraries which has a strong foundation and a large community. ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front-end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

The following are the steps to create a Search filter using ReactJS:

1. Declare React states for search input values.

2. Create HTML input text for entering search term and update state in onChange function

3. Add Array.filter() on list of items with search term value.

**PROGRAM:**  
**App.js:**

import React, { useState } from "react";

import "./App.css";

function App() {

const list = [

"Banana",

"Apple",

"Orange",

"Mango",

"Pineapple",

"Watermelon"

];

const [filterList, setFilterList] = useState(list);

const handleSearch = (event) => {

if (event.target.value === "") {

setFilterList(list);

return;

}

const filteredValues = list.filter(

(item) =>

item.toLowerCase().indexOf(event.target.value.toLowerCase()) !== -1

);

setFilterList(filteredValues);

};

return (

<div className="app">

<div>

Search: <input name="query" type="text" onChange={handleSearch} />

</div>

{filterList &&

filterList.map((item, index) => (

<div key={index}>{item}</div> //Display each item

))}

</div>

);

}

export default App;

**App.css:**

.app {

font-family: sans-serif;

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

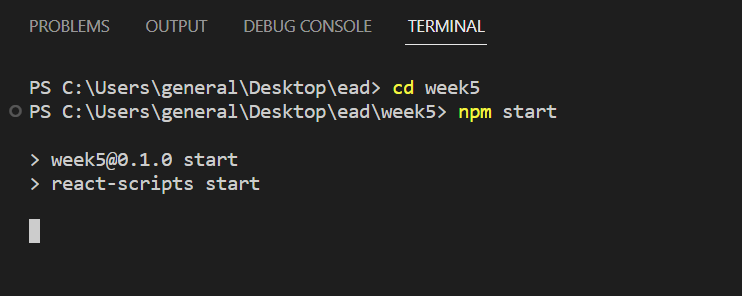
gap: 20px;

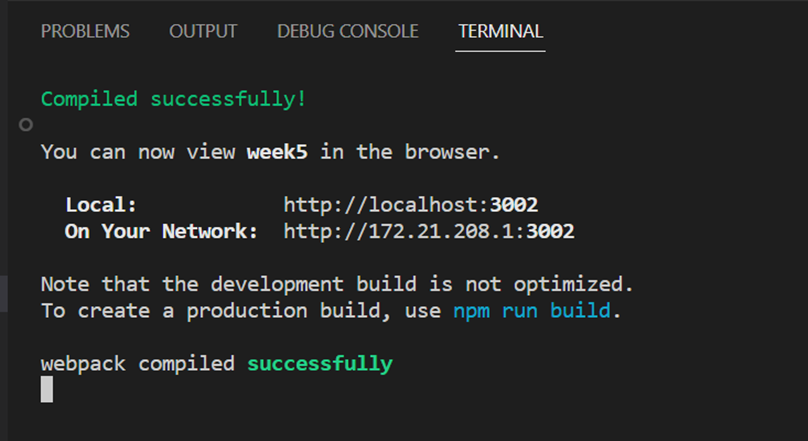
height: 100vh;

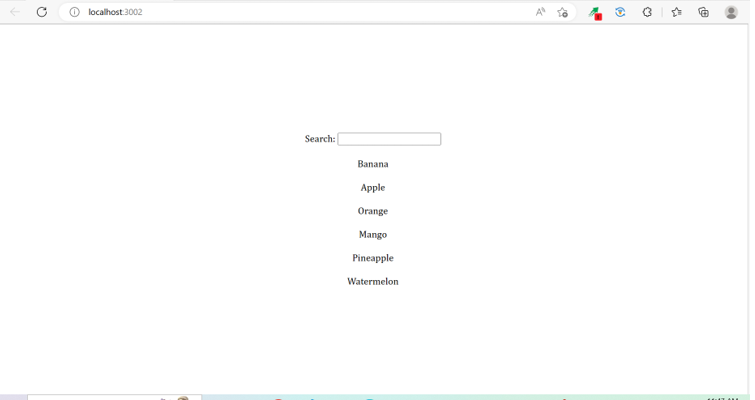
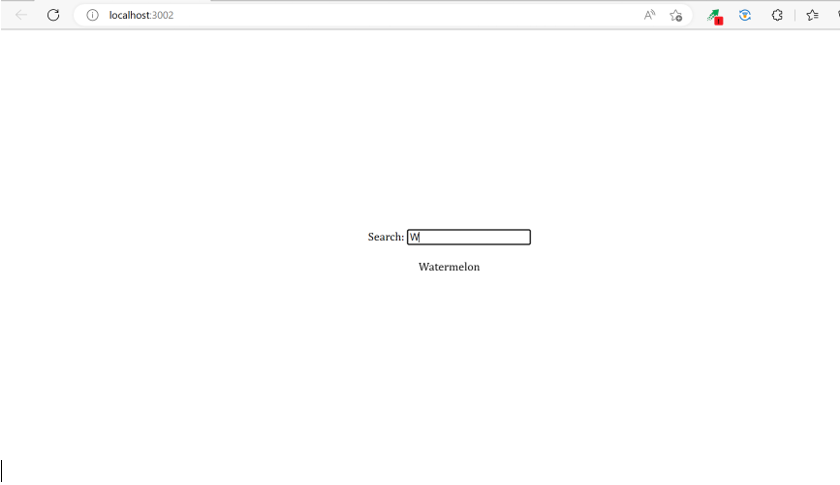
font-family: Cambria, Cochin, Georgia, Times, "Times New Roman", serif;

}

**OUTPUT:**







**CONCLUSION:**

We have successfully built a react code to create a simple search filter functionality to display a filtered list based on the search query entered by the user.

**EXPERIMENT NUMBER – 6**

**AIM:** To build a react code to create a simple Guess the number game

**DESCRIPTION:**  
ReactJS is one of the most popular JavaScript front-end libraries which has a strong foundation and a large community. ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front-end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

The following are the steps to create a Guess the number using ReactJS:

1. Create 2 components App and Result

2. App components contain all the logic, it is stateful and the Result component only shows the appropriate message on the page according to the user’s guess.

3. The App component has a controlled input field in which the user allowed to type and guess the number.

4. There is a default prop we set to the App component ‘secret’ that holds the required secret number and it is generated randomly.

5. The app component passes the value of the input field and the secret number to the Result component.

6. The result component accordingly show the appropriate message that the guessed number is high or low or correct.

**PROGRAM:**  
**index.js:**

import React from 'react'

import ReactDOM from 'react-dom'

import App from './App'

ReactDOM.render(<App />, document.querySelector('#root'))

**App.js:**

import React, { Component } from 'react'

import Result from './Result'

class App extends Component{

  static defaultProps =  {

    secret : Math.floor(Math.random() \* 20) + 1

  }

  constructor(props){

    super(props)

    this.state = { term : '' }

    this.handleChange = this.handleChange.bind(this)

  }

  handleChange(event){

    this.setState({

      [event.target.name] : event.target.value

    })

  }

  render(){

    return (

      <div>

        <div>

          <label htmlFor='term'>

            Guess Number between 1 to 20

          </label>

        </div>

        <input

          id='term'

          type='text'

          name='term'

          value={this.state.term}

          onChange={this.handleChange}

        />

        <Result term={this.state.term}

            secretNum={this.props.secret} />

      </div>

    )

  }

}

export default App

**Result.js:**

import React from 'react'

const Result = ({ term , secretNum }) => {

  let result;

**if**(term){

**if**(secretNum > term){

      result = 'You guessed Lower'

    }

**else** **if**(secretNum < term){

      result ='You guessed Higher'

    }

**else**{

      result ='Yippee, guessed it!'

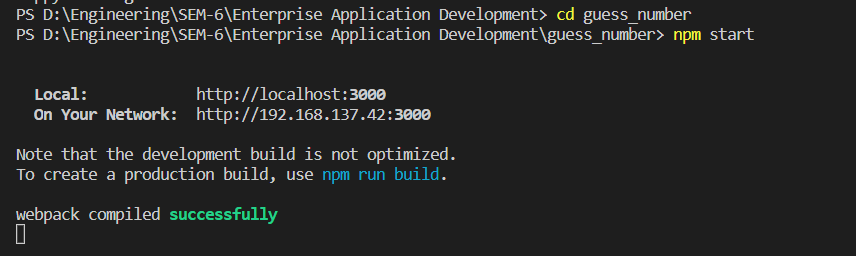
    }

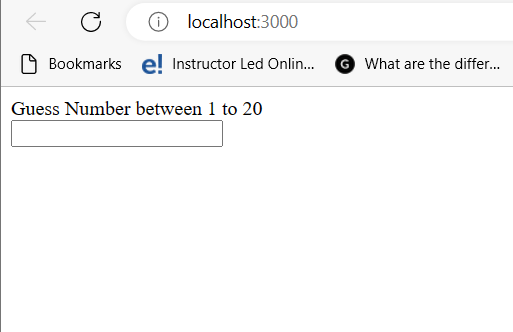
  }

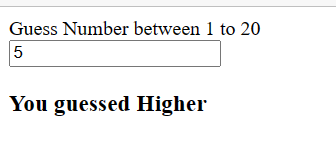
**return** <h3>{result}</h3>

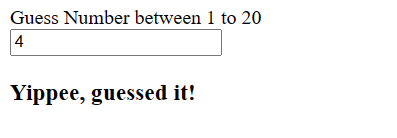
}

export **default** Result

  
**OUTPUT:**







**CONCLUSION:**

We have successfully built a react code to create a simple Guess the number game.

**EXPERIMENT NUMBER – 7**

**AIM:** To Create Express application

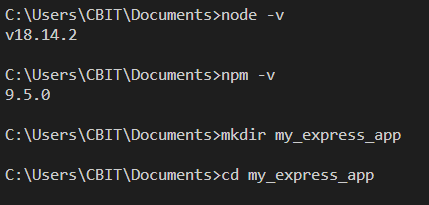
**DESCRIPTION:**

Express is a fast, assertive, essential and moderate web framework of Node.js. You can assume express as a layer built on the top of the Node.js that helps manage a server and routes. It provides a robust set of features to develop web and mobile applications.

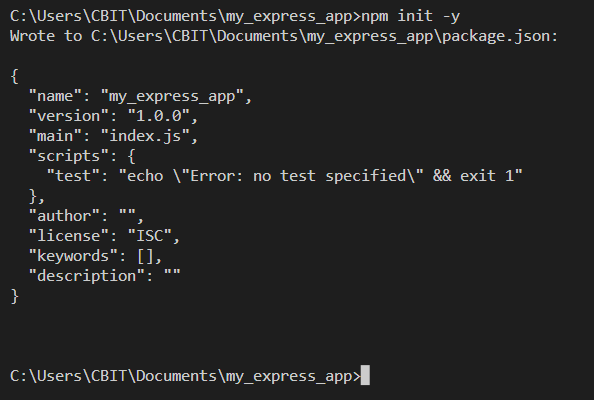
**Steps to create an express application:**

1. Install Node.js and npm: You'll need Node.js and npm (Node Package Manager) installed on your machine to use Express. You can download them from the official website: https://nodejs.org/en/download/

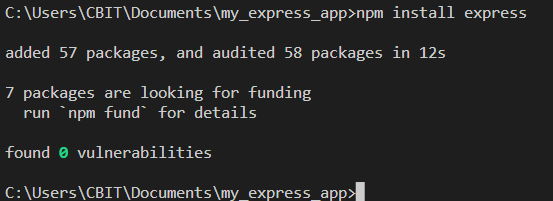
2. Create a new directory for your Express application: Open up your terminal and create a new directory for your application, e.g. mkdir my-express-app. Navigate to the directory using cd my-express-app.



3. Initialize your project: Run npm init in your terminal and follow the prompts to initialize your project. This will create a package.json file in your project directory.



4. Install Express: Run npm install express in your terminal to install Express and add it to your project's dependencies.



5. Create a new file for your server: Create a new file in your project directory, e.g. server.js. This will be the entry point for your application.

6. Set up your Express application: In server.js, require Express and create aninstance of it: const express = require('express'); const app = express();

7. Define routes: You can define routes using the app.get(), app.post(), app.put(), app.delete(), and other methods. For example, here's how you can define a basic route that sends a message to the client: app.get('/', (req, res) => { res.send('Hello, World!'); });

8. Start your server: Finally, start your server by listening on a port. For example, you can listen on port 3000 like this: app.listen(3000, () => { console.log('Server is running on port 3000'); });

**PROGRAM:**

const express = require('express');

const app = express();

app.get('/', (req, res) => {

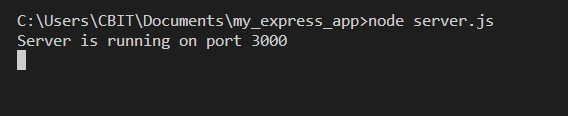
res.send('Hello, World!');

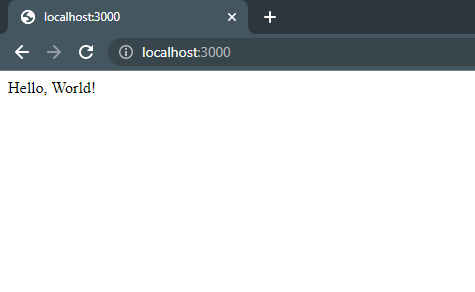
});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

**OUTPUT:**

****

**CONCLUSION:**

We have successfully created an express application.

**EXPERIMENT NUMBER – 8**

**AIM:** To Access Data Using Nodejs

**DESCRIPTION:**

Mongoose Mongoose is a popular object modeling library for MongoDB. It provides a schema-based solution to model your application data and includes features like validation, query building, and middleware.

Step-1 Create a project folder.

Step-2 install mongoose.

Step-3 Create .js file

Step-4 run node file.js

**PROCEDURE:**

**1)Connecting mongoose to mongodb and creating blog post**

const mongoose = require('mongoose');

main().catch(err => console.log(err));

async function main() {

await mongoose.connect('mongodb://127.0.0.1:27017/mydb');

const postschema = new mongoose.Schema({

title: String,

content: String,

date: { type: Date, default: Date.now },

});

const Post = mongoose.model('Post', postschema);

const post = new Post({

title: 'My first blog post',

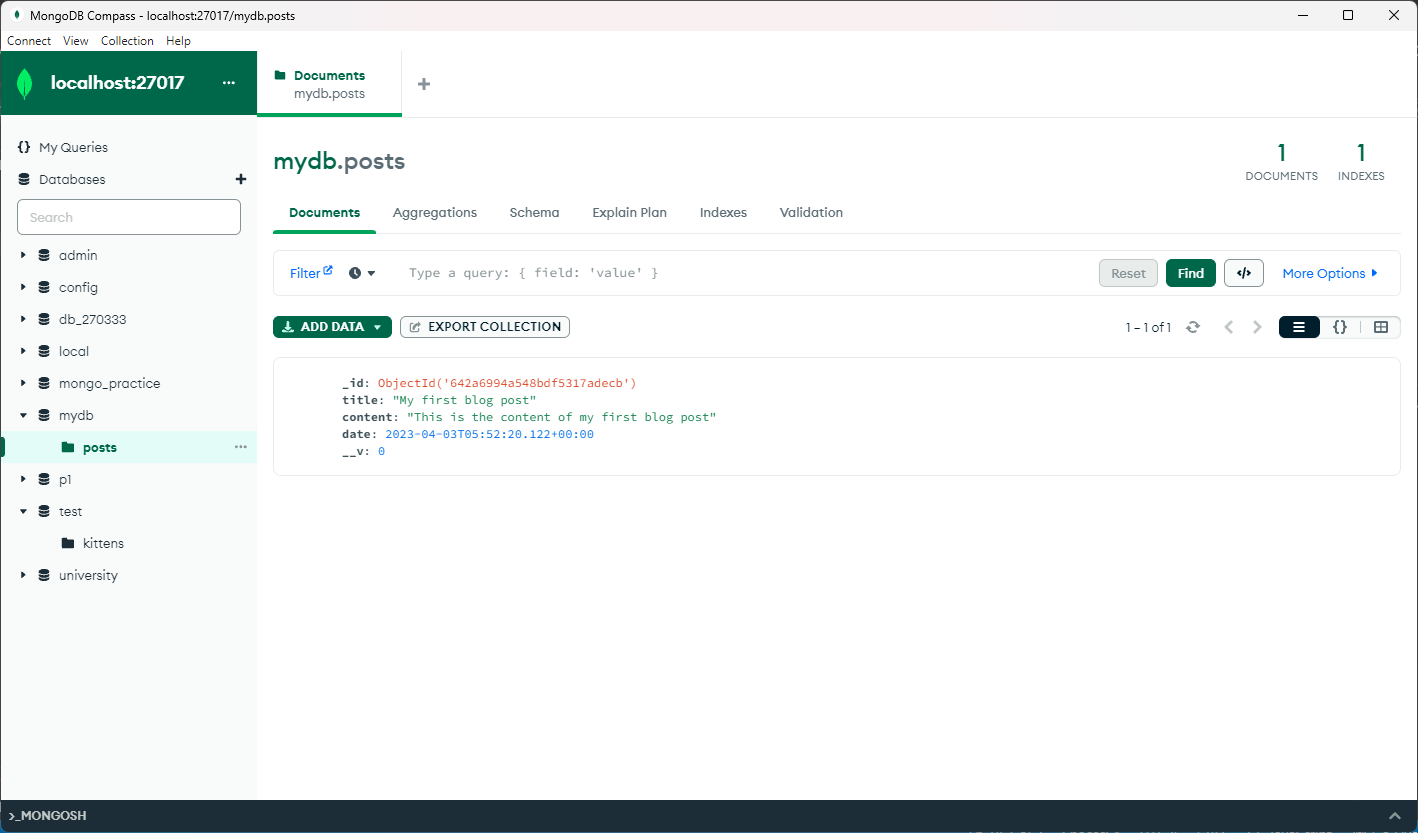
content: 'This is the content of my first blog post',

});

await post.save();

}

**Output:**

****

1. **Find**:

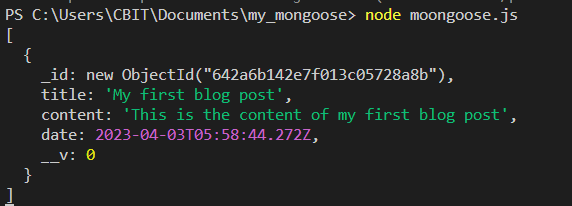
**Description**: The find() function is used to find particular data from the MongoDB database.

**Syntax:** Model.find()

const p = await Post.find();

   console.log(p);

**Output:**

****

1. **Findbyid:**

**Description:** In MongoDB, all documents are unique because the **\_id** field or *path* that MongoDB uses to automatically create a new document.

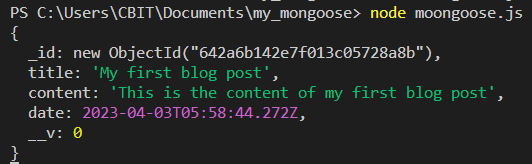
**Syntax:** Model.findById(id)

const id = "642a6b142e7f013c05728a8b"

const l = await Post.findById(id)

console.log(l)

**Output:**

****

1. **Findbyidandupdate:**

**Description:** Function is used to find a matching document, updates it according to the update arg, passing any options, and returns the found document (if any) to the callback.

**Syntax:** [Model.findByIdAndUpdate()](https://mongoosejs.com/docs/api/model.html#Model.findByIdAndUpdate())

const id = "642a6b142e7f013c05728a8b"

const h= await Post.findByIdAndUpdate(id,

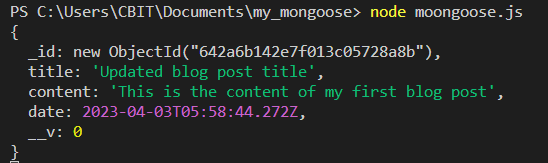
{ title: 'Updated blog post title' },

{ new: true },

)

console.log(h)

**Output:**

****

1. **Findbyanddelete:**

**Description:** It is used to find a matching document, removes it, and passing the found document (if any) to the callback.

**Syntax:** Model.findByIdAndDelete(id)

const id = "642a6b142e7f013c05728a8b"

const m= await Post.findByIdAndDelete(id)

console.log(m)

**Output:**

**IMG_256**

**CONCLUSION:**

We have successfully accessed the data using nodejs.

**EXPERIMENT NUMBER – 9**

**AIM:** To Create a form to edit the data using Angular2.

**DESCRIPTION:**

Angular is a front-end framework which is used to create web applications. It uses typescript by default for creating logics and methods for a class but the browser doesn’t know typescript. Here webpack comes in picture, webpack is used to compile these typescript files to JavaScript. In addition, there are so many configuration files you will need to run an angular project on your computer.  
Angular CLI is a tool that does all these things for you in some simple commands. Angular CLI uses webpack behind to do all this process.

**PROCEDURE:**

1. Set up a new Angular 2 project using the Angular CLI.

Install angular cli

npm install -g @angular/cli

Create new project by this command, Choose yes for routing option and, CSS

ng new myNewApp

Go to your project directory

cd myNewApp

Run server and see your application in action

ng serve

2. Create a new component for your form using the Angular CLI

ng generate component edit-form

3. In your component's HTML file (edit-form.component.html), create a form

using the Angular ngForm directive:

<form #editForm="ngForm" (ngSubmit)="onSubmit()">

<label for="name">Name:</label>

<input type="text" id="name" name="name" [(ngModel)]="user.name" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" [(ngModel)]="user.email"

required>

<button type="submit" [disabled]="editForm.invalid">Submit</button>

</form>

This form has two input fields for the user's name and email, and a submit

button. The ngModel directive is used to bind the form fields to a user object in

your component's TypeScript file.

4. In your component's TypeScript file (edit-form.component.ts), define the user

object and write a function to handle form submissions:

import { Component } from '@angular/core';

@Component({

selector: 'app-edit-form',

templateUrl: './edit-form.component.html',

styleUrls: ['./edit-form.component.css']

})

export class EditFormComponent {

user = {

name: 'John Doe',

email: 'johndoe@example.com'

};

onSubmit() {

console.log('User data submitted: ', this.user);

}

}

In this example, the user object is pre-populated with some sample data. When

the form is submitted, the onSubmit() function logs the user object to the

console.

5. Add your form component to your application by adding it to your

app.module.ts file:

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

import { EditFormComponent } from './edit-form/edit-form.component';@NgModule({

declarations: [

AppComponent,

EditFormComponent

],

imports: [

BrowserModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

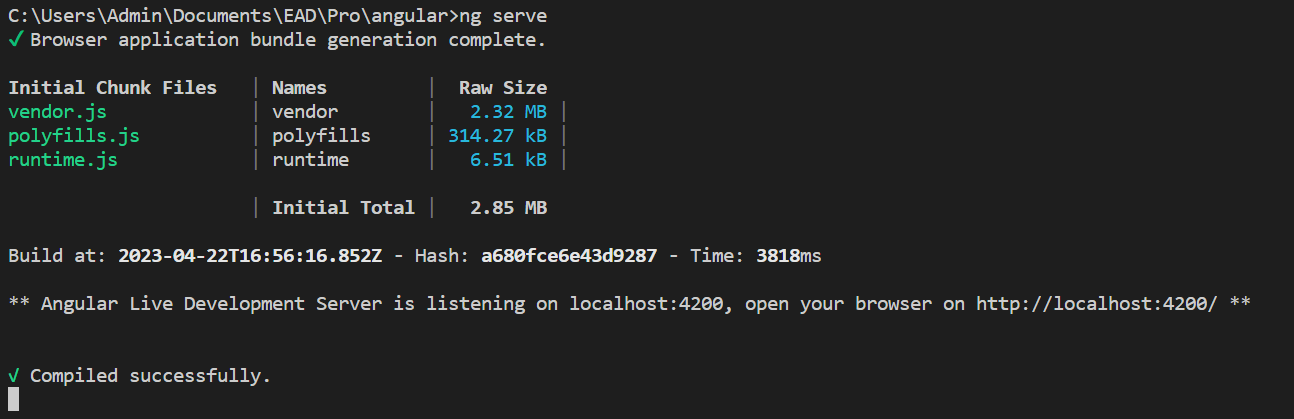
export class AppModule { }

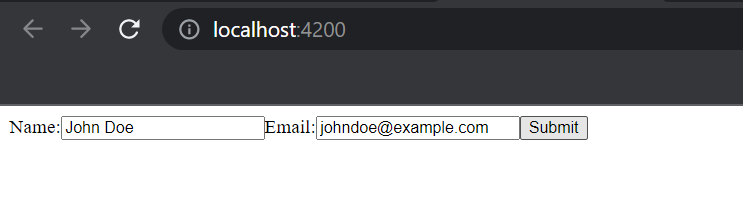
6. Run your application using the Angular CLI:

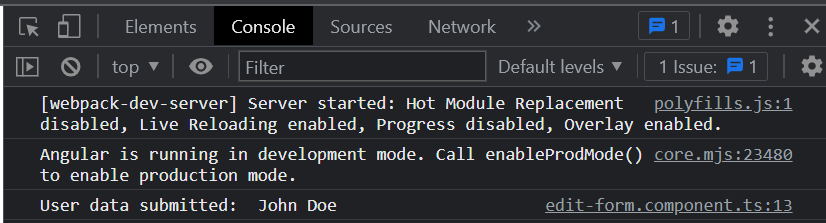
ng serve

This should start a development server and open your application in a browser.

You should now be able to see your form and submit data to it.

**OUTPUT:**





**CONCLUSION:**

We have successfully Created a form to edit the data using Angular2.