**Assignment 1**

**1. Write program to generate random numbers which contains string,**

**numbers, special characters and epoch time.**

function generateRandomString()

{

const characters ='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';

let result = ' ';

const charactersLength = characters.length;

for ( let i = 0; i < 5; i++ )

{

result += characters.charAt(Math.floor(Math.random() \* charactersLength));

}

let epochTime = Math.round((new Date()).getTime() / 1000); result += epochTime.toString();

let specialChars = '!@#$%^&\*()\_+-=<>?,./';

let randomIndex = Math.floor(Math.random() \* specialChars.length); let randomSpecialChar = specialChars[randomIndex];

result += randomSpecialChar; return result;

}

let randomString = generateRandomString(); console.log(randomString);

**o/p**:

**OUTPUT:**

PS C:\Users\MCA\Documents\MERN\Node\_Js> node random.js

tVXQc1672372048>

PS C:\Users\MCA\Documents\MERN\Node\_Js>

**2. Write a program to compute the difference in days by accepting two**

**dates.**

let date1 = new Date("06/30/2019");

    let date2 = new Date("07/30/2019");

    let Difference\_In\_Time = date2.getTime() - date1.getTime();

    let Difference\_In\_Days = Difference\_In\_Time / (1000 \* 3600 \* 24);

    console.log("Total number of days between dates  :" + date1 + 'and' + date2 + " is:  " + Difference\_In\_Days);

**OUTPUT:**

PS C:\Users\MCA\Documents\MERN\Node\_Js> node days.js

Total number of days between dates :Sun Jun 30 2019 00:00:00 GMT+0530 (India Standard Time)andTue Jul 30 2019 00:00:00 GMT+0530 (India Standard Time) is: 30PS

**3. Write a program to convert kilometer to miles.**

const readline = require('readline').createInterface({

    input: process.stdin,

    output: process.stdout,

  });

  readline.question(`Enter Number`, kms => {

    let miles = kms \* 0.62137119 ;

       readline.close();

       console.log(miles);

  });

**OUTPUT:**

PS C:\Users\MCA\Documents\MERN\Node\_Js> node KMtoMiles.js

Enter Number 2

1.24274238

**4. Write a program to accept array of animals and list the name of animals**

**that starts with ‘E’**

let names = ["Elephent", "Eagle", "Dog", "cat"];

let eNames = names.filter(name => name.charAt(0) === 'e' || name.charAt(0) === 'E');

console.log(eNames);

**OUTPUT:**

PS C:\Users\MCA\Documents\MERN\Node\_Js> node namefilter.js

[ 'Elephent', 'Eagle' ]

**5. Write a program to find the largest of 3 numbers using functions.**

const readline = require('readline').createInterface({

    input: process.stdin,

    output: process.stdout,

  });

function largestamoung(num1,num2,num3) {

    let largest;

    if(num1 >= num2 && num1 >= num3) {

        largest = num1;

    }

    else if (num2 >= num1 && num2 >= num3) {

        largest = num2;

    }

    else {

        largest = num3;

    }

    return largest;

}

readline.question(`Enter num1 : `, num1 => {

    readline.question(`Enter num2 : `, num2 => {

        readline.question(`Enter num3 : `, num3 => {

    const result= largestamoung(num1,num2,num3);

       readline.close();

       console.log(result);

        });

    });

 });

**OUTPUT:**

PS C:\Users\MCA\Documents\MERN\Node\_Js> node largest\_amoung3.js

Enter num1 : 10

Enter num2 : 2

Enter num3 : 25

25

PS C:\Users\MCA\Documents\MERN\Node\_Js>

**6. Write a program to convert decimal to binary.**

const readline = require('readline').createInterface({

    input: process.stdin,

    output: process.stdout,

  });

function decimalToBinary(number)

{

  let binary = '';

  while (number > 0)

  {

    binary = (number % 2) + binary;

    number = Math.floor(number / 2);

  }

  return binary;

}

readline.question(`Enter Number`, number => {

  const result= decimalToBinary(number);

     readline.close();

     console.log(result);

});

**OUTPUT:**

PS C:\Users\MCA\Documents\MERN\Node\_Js> node binary.js

Enter Number5

101

**7. Write a program to reverse the given string.**

const readline = require('readline').createInterface({

    input: process.stdin,

    output: process.stdout,

  });

function reverseString(str) {

    let newString = "";

    for (let i = str.length - 1; i >= 0; i--) {

        newString += str[i];

    }

    return newString;

}

readline.question(`Enter String`, name => {

     const result= reverseString(name);

        readline.close();

        console.log(result);

  });

**OUTPUT:**

PS C:\Users\MCA\Documents\MERN\Node\_Js> node reversstring.js

Enter String : Mahaning

gninahaM

**8. Write a program to identify occurrence of a character in a given string.**

let givenStr = "Mahaning !!";

let ch = 'n';

let count = 0;

for(let i = 0; i<givenStr.length; i++)

{

    if(givenStr.charAt(i) == ch)

    {

count ++;

}

}

console.log(`Total occurrence : ${count}`);

**OUTPUT**

PS C:\Users\MCA\Documents\MERN\Node\_Js> node charcount.js

Total occurrence : 2

**9. Create a dictionary and retrieve values of integer data type.**

let dict = {

key1: "Hello",

key2: 'A',

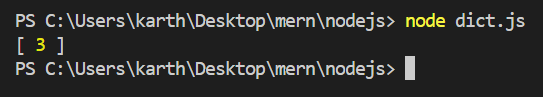
key3: 3

};

let intValues = Object.values(dict).filter(value => typeof value === 'number');

console.log(intValues);

**o/p:**



**10. Program to merge elements of two arrays and sort in descending order.**

function getUniqueAfterMerge(arr1, arr2){

    let arr = arr1.concat(arr2);

    let uniqueArr = [];

    for(let i of arr) {

        if(uniqueArr.indexOf(i) === -1) {

            uniqueArr.push(i);

        }

    }

    console.log('Merged array '+uniqueArr);

    bubbleSort(uniqueArr);

    function bubbleSort(uniqueArr) {

        var done = false;

        while (!done) {

          done = true;

          for (var i = 1; i < uniqueArr.length; i += 1) {

            if (uniqueArr[i - 1] > uniqueArr[i]) {

              done = false;

              var tmp = uniqueArr[i - 1];

              uniqueArr[i - 1] = uniqueArr[i];

              uniqueArr[i] = tmp;

            }

          }

        }

              console.log('Sorted array '+uniqueArr);

      }

}

const array1 = [1, 2, 3];

const array2 = [6, 3, 5]

getUniqueAfterMerge(array1, array2);

**OUTPUT:**

PS C:\Users\MCA\Documents\MERN\Node\_Js> node merge\_and\_sort.js

Merged array 1,2,3,6,5

Sorted array 1,2,3,5,6

**Assignment 2**

**Create React component for listing vehicle information such as manufacturer, model name, year, safety ratings using class based components, React properties and states.**

* **Class based implementation**

**index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import { IssueListState } from './assig1';

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

root.render(

<React.StrictMode>

<IssueListState/>

</React.StrictMode>

);

**IssueAdd.js**

import React from "react";

export class IssueAdd extends React.Component{

constructor(){

super();

this.handleSubmit=this.handleSubmit.bind(this);

}

handleSubmit(e){

e.preventDefault();

const form=document.forms.issueAdd;

const issue={

manufacturer:form.manufacturer.value,model\_name:form.model\_name.value,model\_year:form.model\_year.value,safety\_rate:form.safety\_rate.value

}

// sending data

this.props.createIssue(issue);

form.id.value=""

form.manufacturer.value=""

form.model\_name.value=""

form.model\_year.value=""

form.safety\_rate.value=""

}

render(){

return(

<center>

<form id="issueAdd" onSubmit={this.handleSubmit}>

Manufacturer: <input type="text" name="manufacturer" placeholder=""></input><br></br>

model name:<input type="text" name="model\_name" placeholder=""></input><br></br>

model year:<input type="text" name="model\_year" placeholder=""></input><br></br>

safety rating:<input type="text" name="safety\_rate" placeholder=""></input><br></br>

<button>Add Issue</button>

</form>

</center>

)

}

}

**IssueList.js**

import React from"react"

class IssueView extends React.Component{

render(){

const issue=this.props.issue

return(

<tr>

<td style={this.props.td\_style}>{issue.id}</td>

<td style={this.props.td\_style}>{issue.manufacturer}</td>

<td style={this.props.td\_style}>{issue.model\_name}</td>

<td style={this.props.td\_style}>{issue.model\_year}</td>

<td style={this.props.td\_style}>{issue.safety\_rate}</td>

</tr>

)

}

}

export class IssueList extends React.Component{

render(){

const td\_style={border:'1px solid black',padding:'5px'}

const issues=[

{id:1,manufacturer:"Lexus",model\_name:"S1",model\_year:"2011",safety\_rate:"4"},

{id:2,manufacturer:"Jeep",model\_name:"Compass",model\_year:"2015",safety\_rate:"5"}

]

const issuerow=issues.map(

issue=> <IssueView key={issue.id} td\_style={td\_style} issue={issue}/>)

return(

<table style={{margin: '0 auto'}}>

<thead>

<tr>

<th style={td\_style}>id</th>

<th style={td\_style}>Manufacturer</th>

<th style={td\_style}>model name</th>

<th style={td\_style}>model year</th>

<th style={td\_style}>safety rating</th>

</tr>

/\* dynamic calling below \*/

{issuerow}

</thead>

</table>

)

}

}

**assign1.js**

import React from"react"

import { IssueAdd } from "./IssueAdd";

class IssueViewState extends React.Component{

render(){

const issue=this.props.issue

return(

<tr>

<td style={this.props.td\_style}>{issue.id}</td>

<td style={this.props.td\_style}>{issue.manufacturer}</td>

<td style={this.props.td\_style}>{issue.model\_name}</td>

<td style={this.props.td\_style}>{issue.model\_year}</td>

<td style={this.props.td\_style}>{issue.safety\_rate}</td>

</tr>

)

}

}

export class IssueListState extends React.Component{

constructor(){

super();

this.state={issues:[]}

this.createIssue=this.createIssue.bind(this);

}

createIssue(issue){

issue.id=this.state.issues.length+1;

const newIssueList=this.state.issues.slice();

newIssueList.push(issue);

this.setState({issues: newIssueList});

}

componentDidMount(){

this.loadInitialData();

}

loadInitialData(){

const Initialissues=[

{id:1,manufacturer:"Lexus",model\_name:"S1",model\_year:"2011",safety\_rate:"4"},

{id:2,manufacturer:"Jeep",model\_name:"Compass",model\_year:"2015",safety\_rate:"5"}

]

setTimeout(() =>{this.setState({issues:Initialissues})},5000)

}

render(){

const td\_style={border:'1px solid black',padding:'5px'}

const issuerow=this.state.issues.map(

issue=> <IssueViewState key={issue.id} td\_style={td\_style} issue={issue}/>)

return(

<table style={{margin: '0 auto'}}>

<thead>

<tr>

<th style={td\_style}>id</th>

<th style={td\_style}>Manufacturer</th>

<th style={td\_style}>model name</th>

<th style={td\_style}>model year</th>

<th style={td\_style}>safety rating</th>

</tr>

{issuerow}

<IssueAdd createIssue={this.createIssue}/>

</thead>

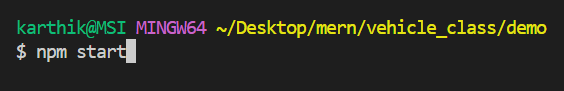
</table>

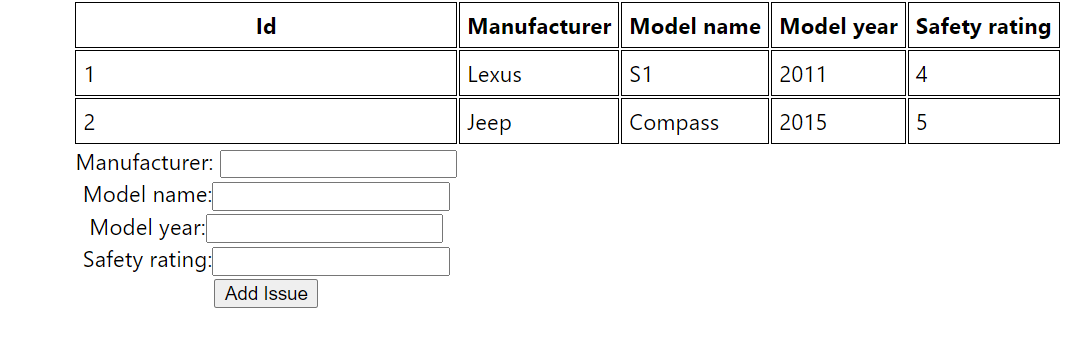
)

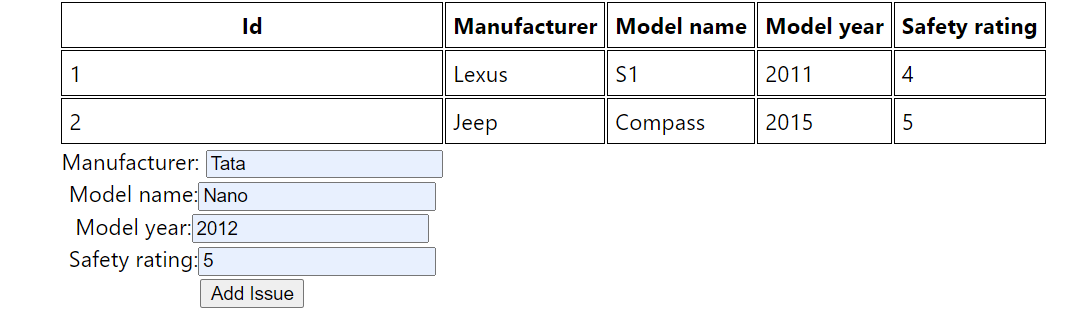
}

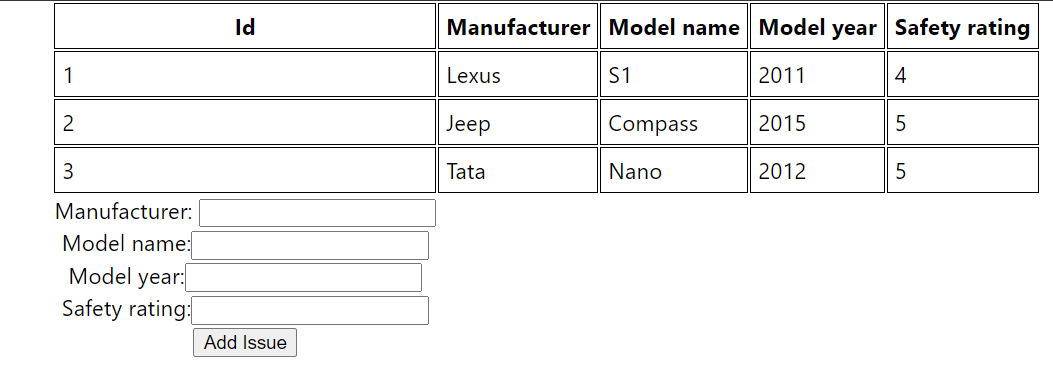
}

Output:









* **Function based implementation**

**assign1.js**

import React, { useState, useEffect } from "react";

import { IssueAdd } from "./issueAdd";

function IssueViewState({ td\_style, issue }) {

return (

<tr>

<td style={td\_style}>{issue.id}</td>

<td style={td\_style}>{issue.manufacturer}</td>

<td style={td\_style}>{issue.model\_name}</td>

<td style={td\_style}>{issue.model\_year}</td>

<td style={td\_style}>{issue.safety\_rate}</td>

</tr>

);

}

export function IssueListState() {

const [issues, setIssues] = useState([]);

useEffect(() => {

loadInitialData();

}, []);

function createIssue(issue) {

issue.id = issues.length + 1;

const newIssueList = [...issues, issue];

setIssues(newIssueList);

}

function loadInitialData() {

const initialIssues = [

{

id: 1,

manufacturer: "Lexus",

model\_name: "S1",

model\_year: "2022",

safety\_rate: "4",

},

{

id: 2,

manufacturer: "Porsche",

model\_name: "911",

model\_year: "2013",

safety\_rate: "4",

},

];

setTimeout(() => {

setIssues(initialIssues);

}, 5000);

}

const td\_style = { border: "1px solid black", padding: "5px" };

const issueRows = issues.map((issue) => (

<IssueViewState key={issue.id} td\_style={td\_style} issue={issue} />

));

return (

<table style={{ margin: "0 auto" }}>

<thead>

<tr>

<th style={td\_style}>Id</th>

<th style={td\_style}>Manufacturer</th>

<th style={td\_style}>Model name</th>

<th style={td\_style}>Model year</th>

<th style={td\_style}>Safety rating</th>

</tr>

</thead>

<tbody>{issueRows}</tbody>

<IssueAdd createIssue={createIssue} />

</table>

);

}

**index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import { IssueListState } from './assig1';

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

root.render(

<React.StrictMode>

<IssueListState/>

</React.StrictMode>

);

**issueAdd.js**

import React from "react";

export function IssueAdd(props) {

const handleSubmit = (e) => {

e.preventDefault();

const form = document.forms.issueAdd;

const issue = {

manufacturer: form.manufacturer.value,

model\_name: form.model\_name.value,

model\_year: form.model\_year.value,

safety\_rate: form.safety\_rate.value,

};

// sending data

props.createIssue(issue);

form.id.value = "";

form.manufacturer.value = "";

form.model\_name.value = "";

form.model\_year.value = "";

form.safety\_rate.value = "";

};

return (

<center>

<form id="issueAdd" onSubmit={handleSubmit}>

Manufacturer:{" "}

<input type="text" name="manufacturer" placeholder=""></input>

<br></br>

Model name:

<input type="text" name="model\_name" placeholder=""></input>

<br></br>

Model year:

<input type="text" name="model\_year" placeholder=""></input>

<br></br>

Safety rating:

<input type="text" name="safety\_rate" placeholder=""></input>

<br></br>

<button>Add Issue</button>

</form>

</center>

);

}

export function IssueViewState(props) {

const issue = props.issue;

return (

<tr>

<td style={props.td\_style}>{issue.id}</td>

<td style={props.td\_style}>{issue.manufacturer}</td>

<td style={props.td\_style}>{issue.model\_name}</td>

<td style={props.td\_style}>{issue.model\_year}</td>

<td style={props.td\_style}>{issue.safety\_rate}</td>

</tr>

);

}

export function IssueListState() {

const [issues, setIssues] = React.useState([]);

const createIssue = (issue) => {

issue.id = issues.length + 1;

const newIssueList = issues.slice();

newIssueList.push(issue);

setIssues(newIssueList);

};

React.useEffect(() => {

loadInitialData();

}, []);

const loadInitialData = () => {

const Initialissues = [

{

id: 1,

manufacturer: "Lexus",

model\_name: "S1",

model\_year: "2022",

safety\_rate: "4",

},

{

id: 2,

manufacturer: "Porsche",

model\_name: "911",

model\_year: "2013",

safety\_rate: "4",

},

];

setTimeout(() => {

setIssues(Initialissues);

}, 5000);

};

const td\_style = { border: "1px solid black", padding: "5px" };

const issueRow = issues.map((issue) => (

<IssueViewState key={issue.id} td\_style={td\_style} issue={issue} />

));

return (

<table style={{ margin: "0 auto" }}>

<thead>

<tr>

<th style={td\_style}>Id</th>

<th style={td\_style}>Manufacturer</th>

<th style={td\_style}>Model name</th>

<th style={td\_style}>Model year</th>

<th style={td\_style}>Safety rating</th>

</tr>

</thead>

<tbody>{issueRow}</tbody>

</table>

)

}

export default IssueAdd;

**issuelist.js**

import React from "react";

function IssueView(props) {

const issue = props.issue;

return (

<tr>

<td style={props.td\_style}>{issue.id}</td>

<td style={props.td\_style}>{issue.manufacturer}</td>

<td style={props.td\_style}>{issue.model\_name}</td>

<td style={props.td\_style}>{issue.model\_year}</td>

<td style={props.td\_style}>{issue.safety\_rate}</td>

</tr>

);

}

function IssueList() {

const td\_style = { border: "1px solid black", padding: "5px" };

const issues = [

{ id: 1, manufacturer: "Lexus", model\_name: "S1", model\_year: "2022", safety\_rate: "4" },

{ id: 2, manufacturer: "Porsche", model\_name: "911", model\_year: "2013", safety\_rate: "4" },

];

const issuerow = issues.map((issue) => (

<IssueView key={issue.id} td\_style={td\_style} issue={issue} />

));

return (

<table style={{ margin: "0 auto" }}>

<thead>

<tr>

<th style={td\_style}>Id</th>

<th style={td\_style}>Manufacturer</th>

<th style={td\_style}>Model name</th>

<th style={td\_style}>Model year</th>

<th style={td\_style}>Safety rating</th>

</tr>

</thead>

<tbody>{issuerow}</tbody>

</table>

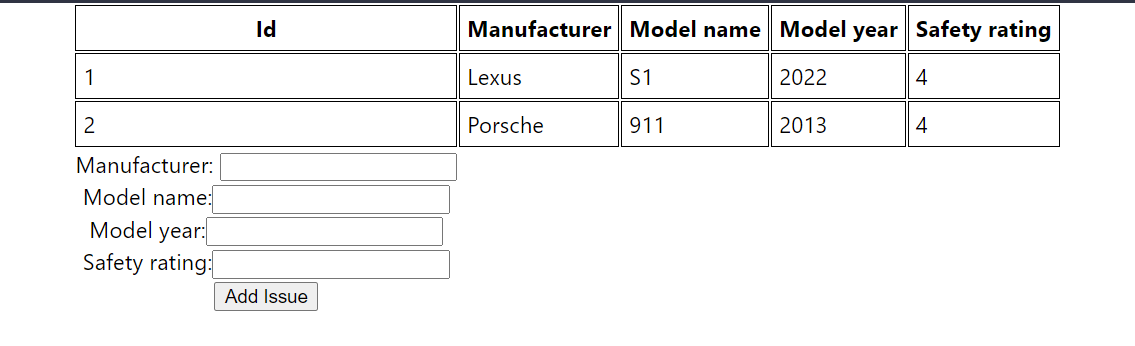
);

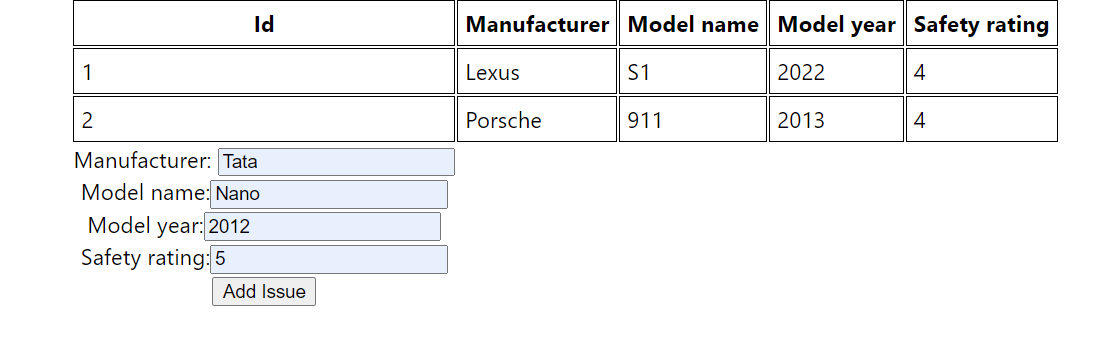
}

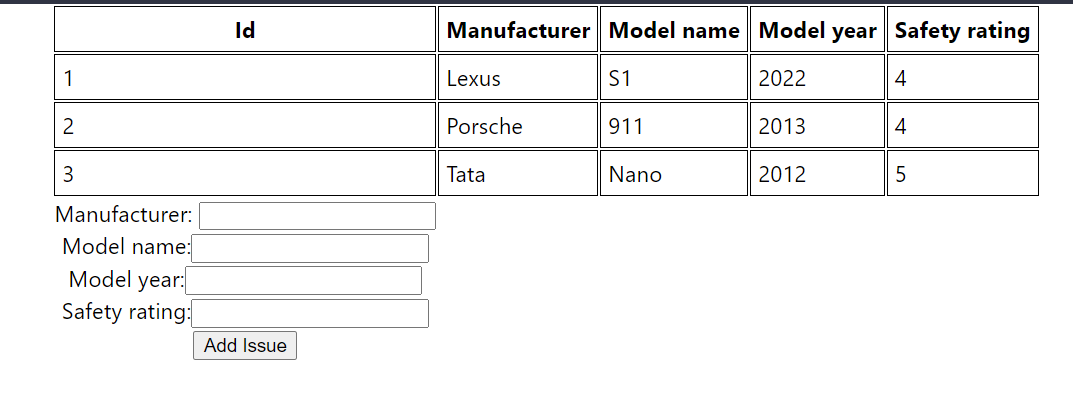
export default IssueList;

**Output:**

**$npm start**







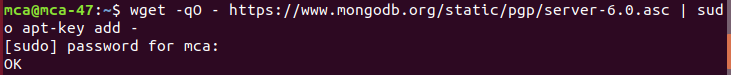
**Assignment 3**

**Write the steps to download and install MongoDB on Linux platform.**

**Install MongoDB Community Edition**

**1. Import the public key used by the package management system.**

wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add –

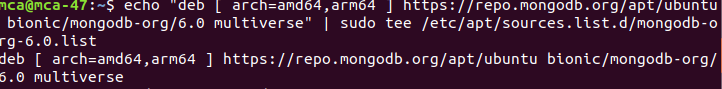


**2. Create a list file for MongoDB.**

Create the /etc/apt/sources.list.d/mongodb-org-6.0.list file for Ubuntu 20.04 (Focal):

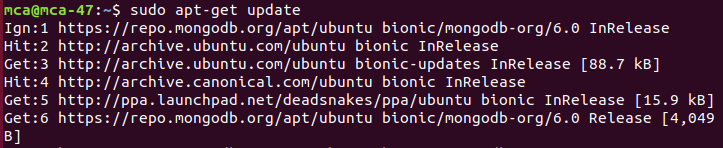
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-

org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list



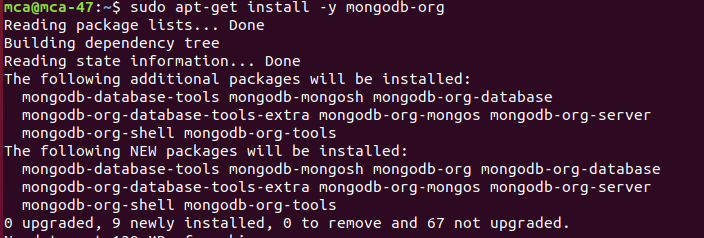
**3. Reload local package database.**

sudo apt-get update



**4. Install the MongoDB packages.**

sudo apt-get install -y mongodb-org



**5. Run MongoDB Community Edition**

If you are unsure which init system your platform uses, run the following command:

ps --no-headers -o comm 1

**6. Start MongoDB**

You can start the mongod process by issuing the following command:

sudo systemctl start mongod

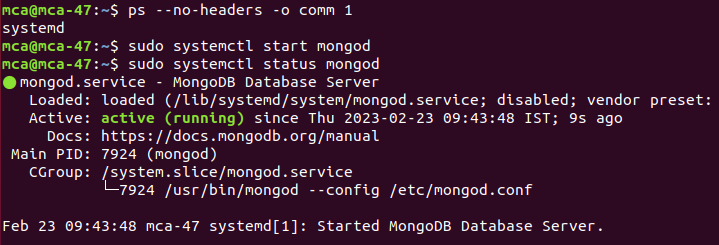
**7. Verify that MongoDB has started successfully.**

sudo systemctl status mongod

You can optionally ensure that MongoDB will start following a system reboot by issuing

the following command:

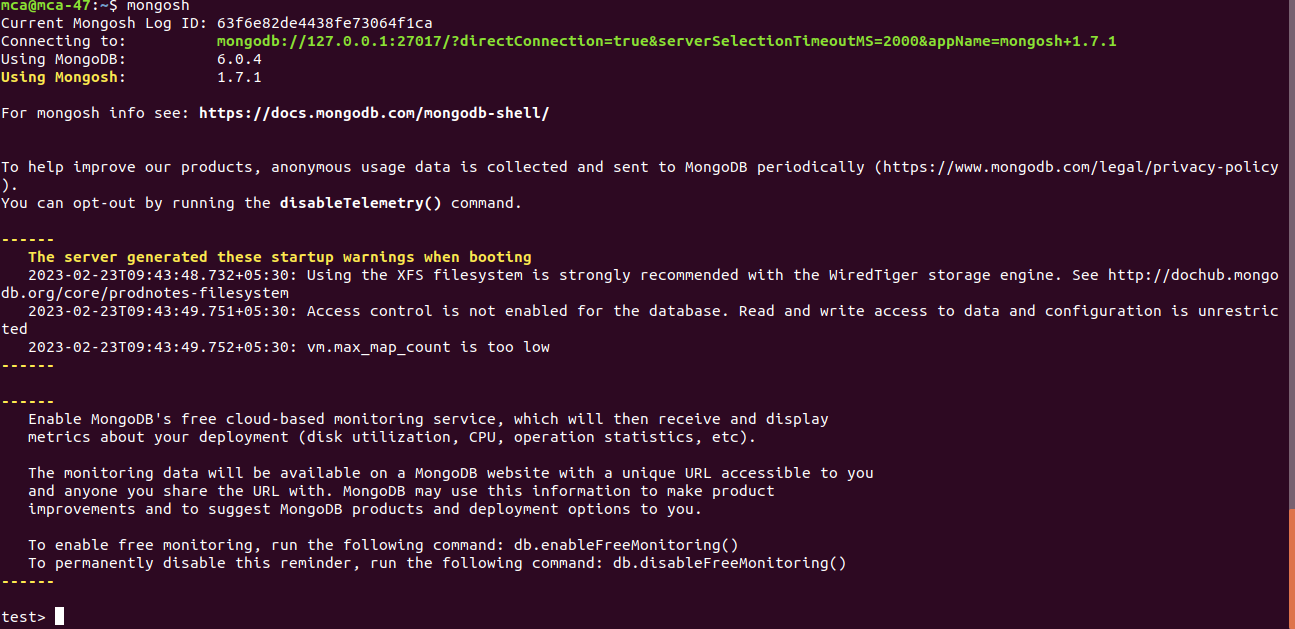
sudo systemctl enable mongod



6.png

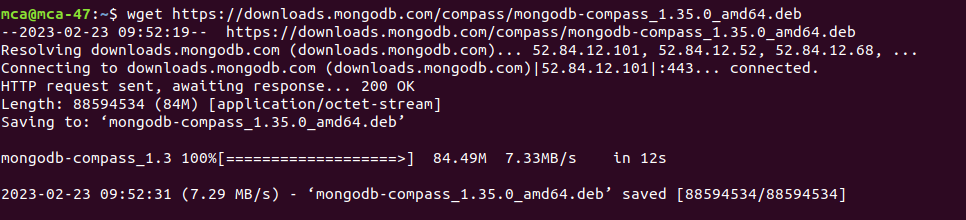
**8. Begin using MongoDB.**

mongosh



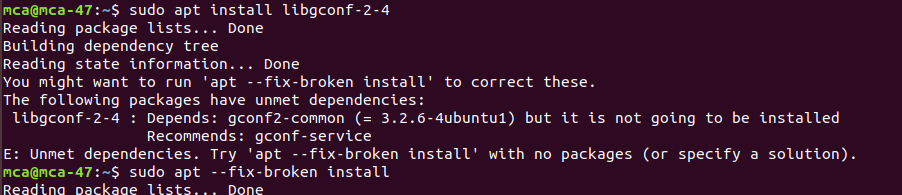
**9. Download MongoDB Compass**

wget <https://downloads.mongodb.com/compass/mongodb-compass_1.35.0_amd64.deb>



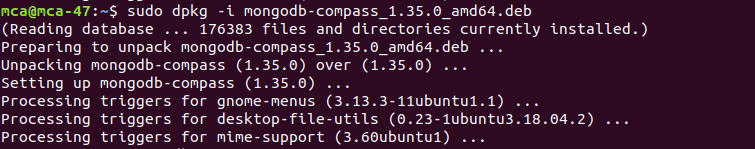
**10. Install GNOME configuration database system(ibgconf-2-4)**

sudo apt install libgconf-2-4



**11. Install MongoDB Compass**

sudo dpkg -i mongodb-compass\_1.35.0\_amd64.deb

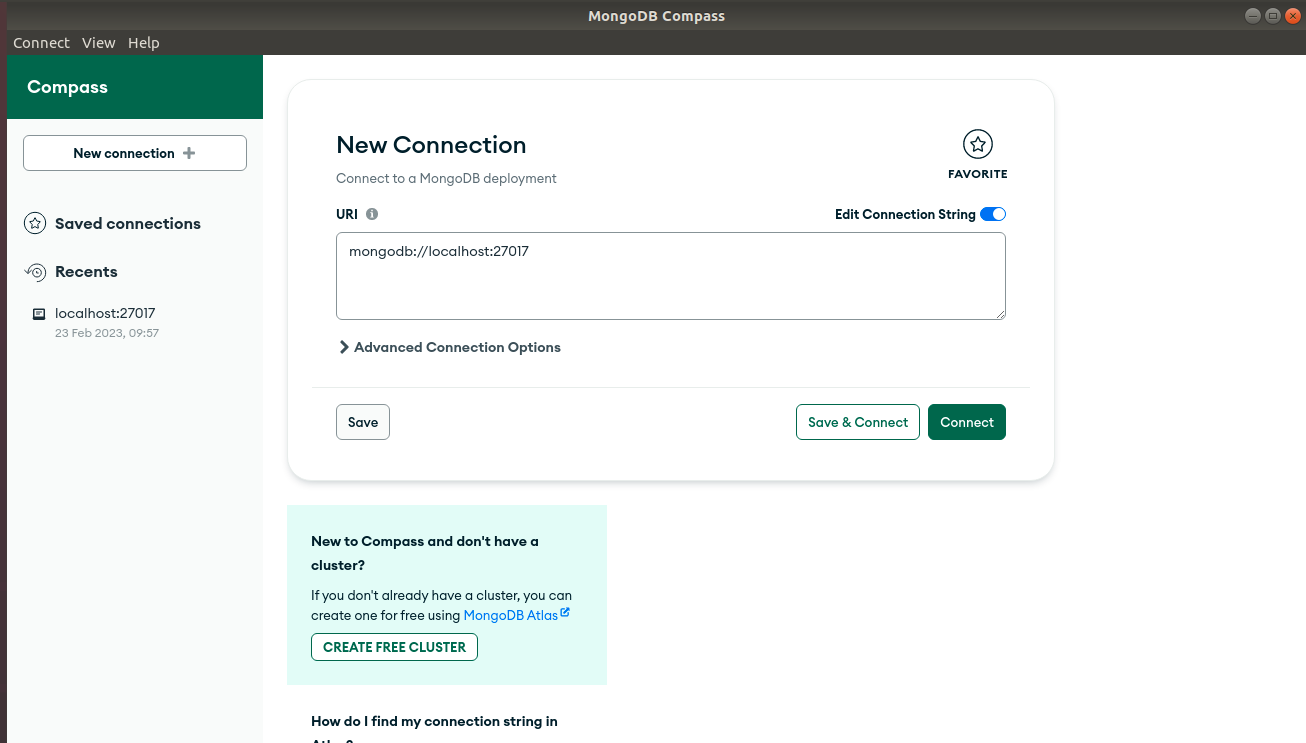


**12. Start MongoDB Compass**

mongodb-compass

C:\Users\karth\Desktop\mongodb_install_ubuntu\compass4.png

**13. Mongodb Compass started**



**Assignment 4**

**Create a MongoDB database with two collections such as student and course with the attributes of student containing name, department, DOB, marks and USN, attributes of course containing course name, course code, course author, course type.**

**Creating a new database**

use school

**Creating collections student and course**

db.createCollection("student")

db.createCollection("course")

# Inserting in student table

db.student.insertOne( { name: "Balrama", dept: "MCA", dob: new ISODate("2000-02-01"), marks: 75, usn: "01fe21mca002" })

db.student.insertOne( { name: "Ishwar", dept: "MCA", dob: new ISODate("2003-02-04"), marks: 85, usn: "01fe21mca003" })

db.student.insertOne( { name: "Ganesha", dept: "MCA", dob: new ISODate("2001-02-25"), marks: 80, usn: "01fe21mca004" })

db.student.insertOne( { name: "Rajendra", dept: "MCA", dob: new ISODate("2003-06-05"), marks: 95, usn: "01fe21mca005" })

db.student.insertOne( { name: "Mayura", dept: "MCA", dob: new ISODate("2002-12-05"), marks: 77, usn: "01fe21mca006" })

db.student.insertMany( [{ name: "Raju", dept: "MCA", dob: new ISODate("2000-02-17"), marks: 66, usn: "01fe21mca007" },

{name: "Ravi", dept: "MCA", dob: new ISODate("2001-09-15"), marks: 57, usn: "01fe21mca008" },

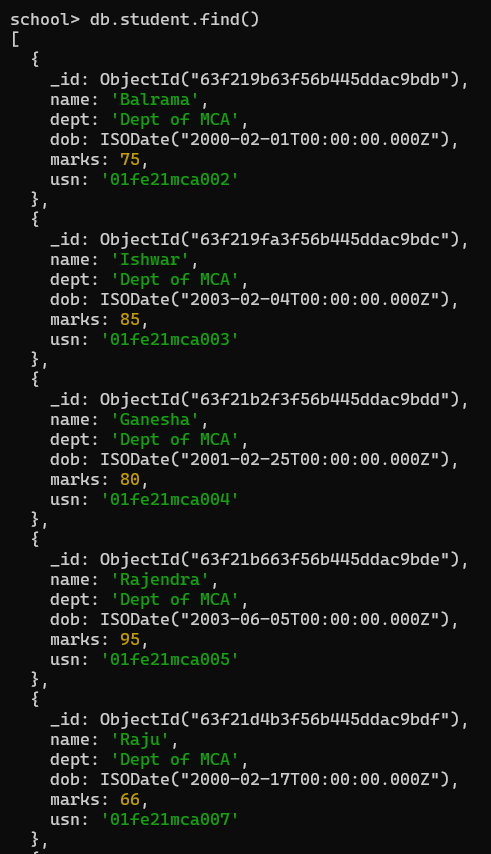
{name: "Raghu", dept: "MCA", dob: new ISODate("2003-08-05"), marks: 67, usn: "01fe21mca009" },

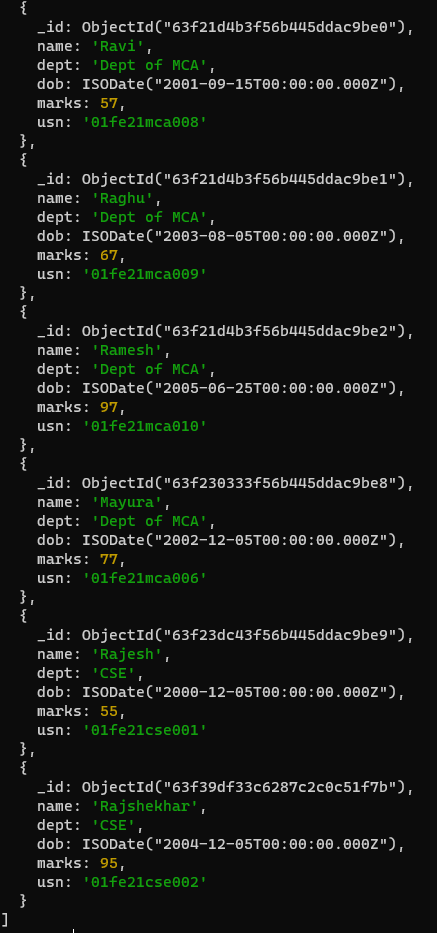
{name: "Ramesh", dept: "MCA", dob: new ISODate("2005-06-29"), marks: 97, usn: "01fe21mca010" }])

db.student.insertOne( { name: "Rajesh", dept: "CSE", dob: new ISODate("2000-12-05"), marks: 55, usn: "01fe21cse001" })

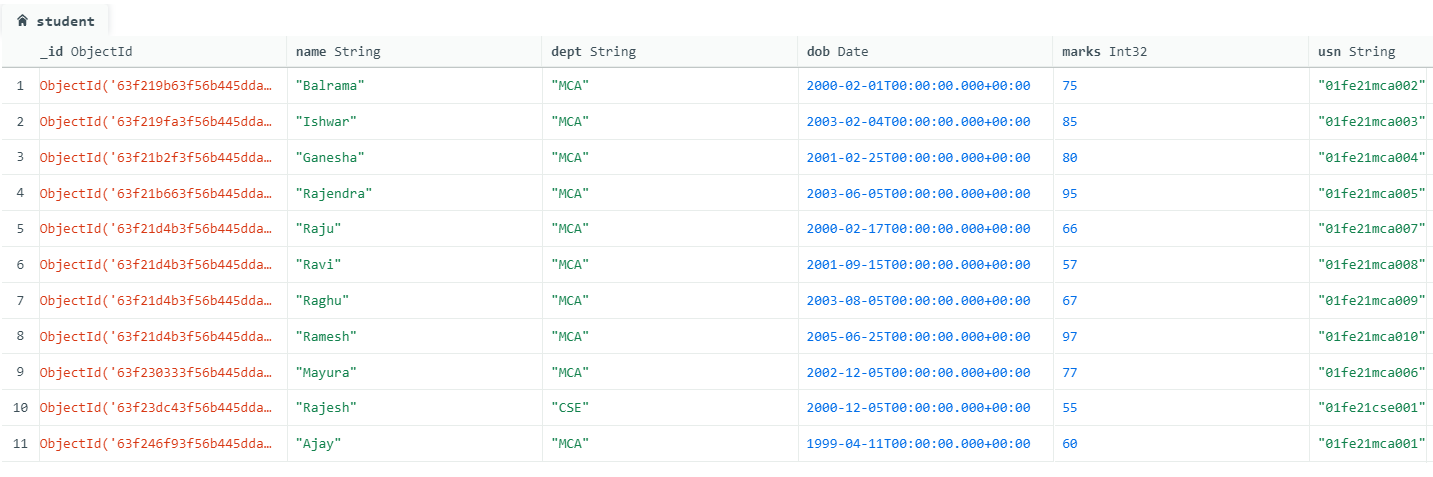
db.student.insertOne({name:"Ajay",dept:"MCA",dob: new ISODate("1999-04-11"), marks:60, usn: "01fe21mca001"})

**Displaying student table**









**Inserting values for course**

db.course.insertMany([{cname:"MERN",code:"b01",author:"Vasan",type:"basic"},

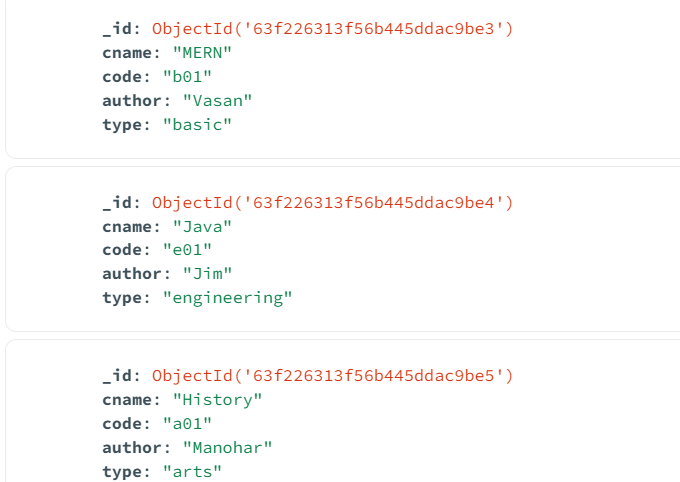
{cname:"Java",code:"e01",author:"Jim",type:"basic"},

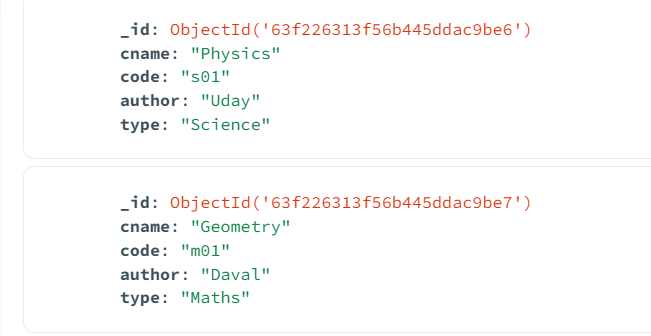
{cname:"History",code:"a01",author:"Manohar",type:"arts"},

{cname:"Physics",code:"s01",author:"Uday",type:"Science"},

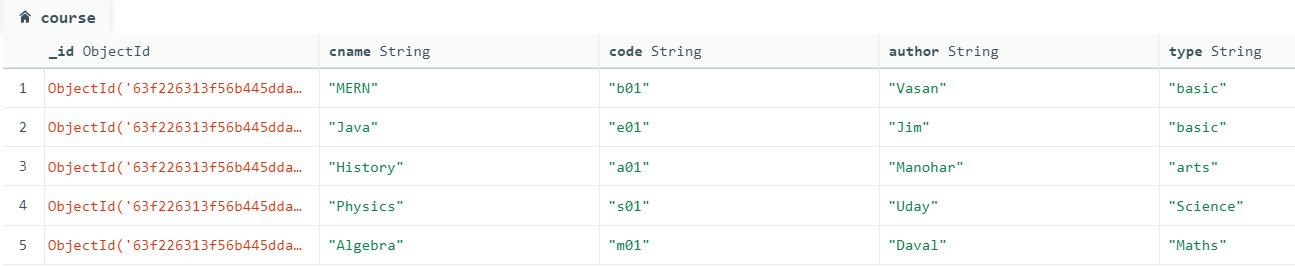
{cname:"Algebra",code:"m01",author:"Daval",type:"Maths"}])

**Displaying course table**









**Queries**

**1. Write Mongodb query to fetch students based on USN**

db.student.find({usn: /mca/})

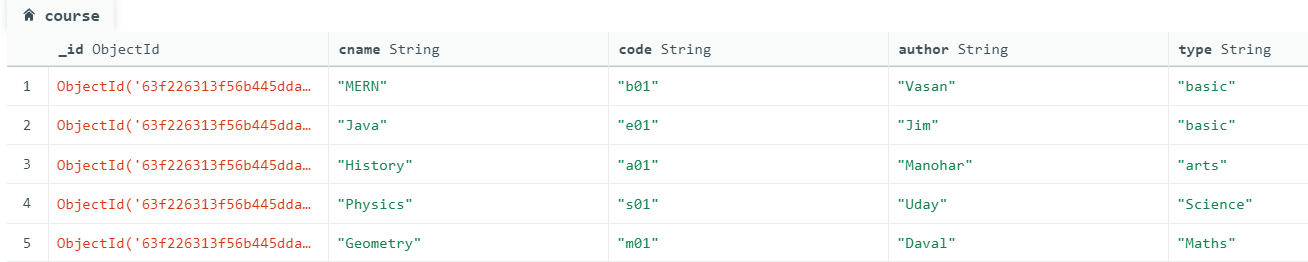




**2. Write Mongodb query to update course name for specific course code.**

db.course.update({code:"m01"},{$set:{cname:"Geometry"}})



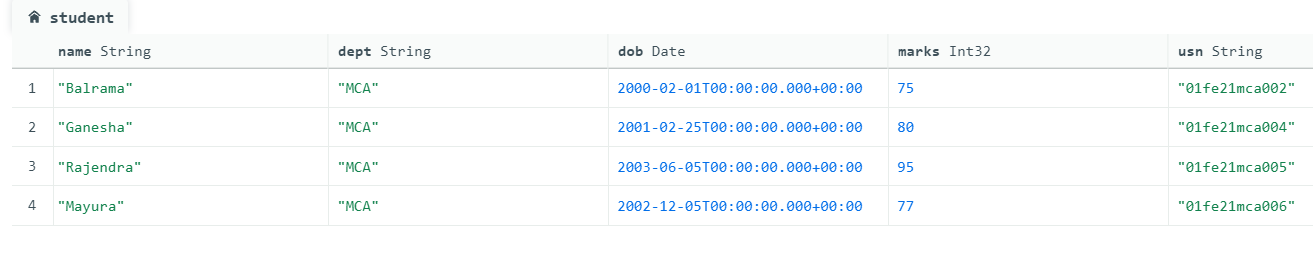


**3. Write Mongodb query to Query to retrieve list of students whose name**

**ends with 'a'.**

db.student.find({name: {$regex: "a$"}})



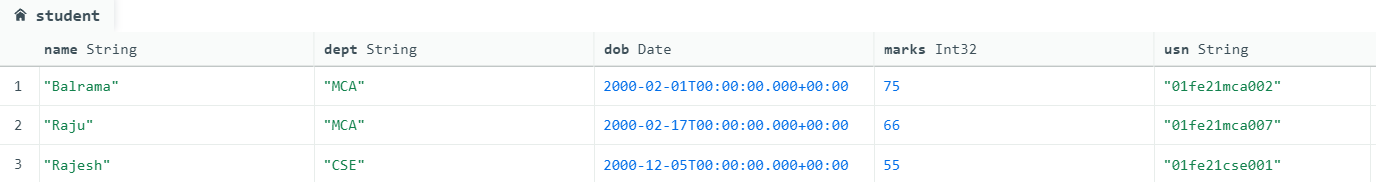


**4. Write Mongodb query to retrieve list of students whose dob is in year**

**2000.**

db.student.find({"dob":{ $gte:ISODate("2000-01-01"), $lt:ISODate("2000-12-31")}})





**5. Write Mongodb query to update students department from MCA to**

**Dept whose USN contains MCA.**

db.student.updateMany({usn: /mca/},{$set:{dept:"Dept of MCA"}})



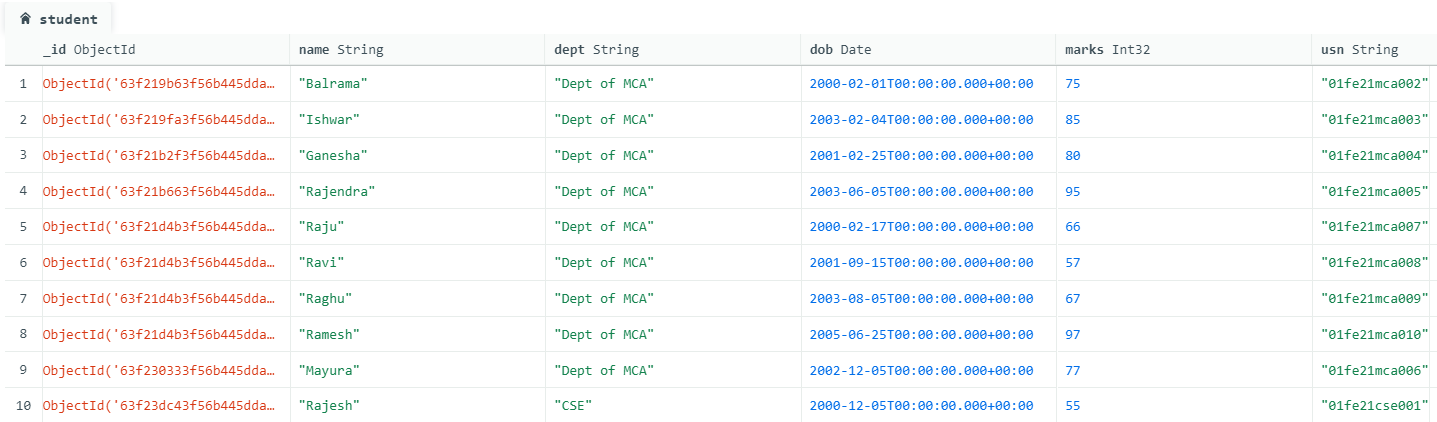


**6. Write Mongodb query to query to delete student information whose**

**DOB is less than 01-01-2000.**

db.student.remove({dob:{$lt:ISODate("2000-01-01")}})



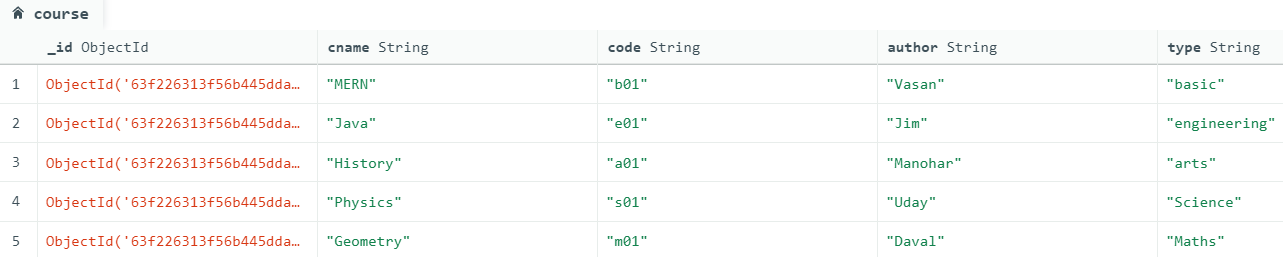


**7. Write Mongodb query to update course type from basic to engineering**

**for courses that contain title as Java.**

db.course.update({cname:"Java"},{$set:{type:"engineering"}})





**8. Retrieve list of students whose dobn is blw 01-01-2000 and 01-01-2005**

db.student.find({"dob":{ $gte:ISODate("2000-01-01"), $lt:ISODate("2005-01-01")}})





\

**Assignment 5**

1. **Create a Mongoose schema for cars which contains following properties**

**Manufacturer, make, model, year of manufacture – validate year of manufacture should be between 2020 and 2024.**

1. **Ensure that same model car should not be present for any of the manufacturers.**
2. **Validate the enumerators for car such as [hatchback, sedan, suv].**
3. **Validate engine number to accept numbers which is of 10 digits.**
4. **Validate chassis number which is alphanumerical and should be between 10-15 digits.**

**config.mongoose.js**

var mongoose = require('mongoose');

var uri = 'mongodb://127.0.0.1:27017/car';

module.exports = function() {

var db = mongoose.connect(uri);

console.log('DB Connection');

require('../students/models/models.car');

// require('../students/models/models.courses');

require('../students/models/models.users');

return db;

};

**controllers.car.js**

var app = require('../../config/config.express');

var Car = require('../models/models.car);

// var Course = require('../models/models.courses');

app.post('/', function(req, res, next) {

var car = new Car(req.body);

car.save(function(err) {

if (err) {

return next(err);

} else {

res.json(car);

}

});

});

**model.car.js**

var mongoose = require('mongoose');

var CarSchema = new mongoose.Schema({

\_id:{type:Number},

manufacturer: { type: String, trim: true, required: true },

make:{type:String,trim: true},

model: {type: String,unique: true, trim: true},

type: { type: String, enum: ['hatchback', 'sedan','suv']},

engine\_num:{type:String, trim: true,unique: true,

validate: [

function(v) {

return /^[0-9]{10}$/.test(v);

}

]

},

//min:10,max:15,

chasis\_num:{type:String, trim: true,unique: true,

validate: [

function(v) {

return /^[A-Za-z0-9]{10,15}$/.test(v); }

]},

manufacture\_year: { type: Date, min: '2020-01-01',max: '2024-01-01' },

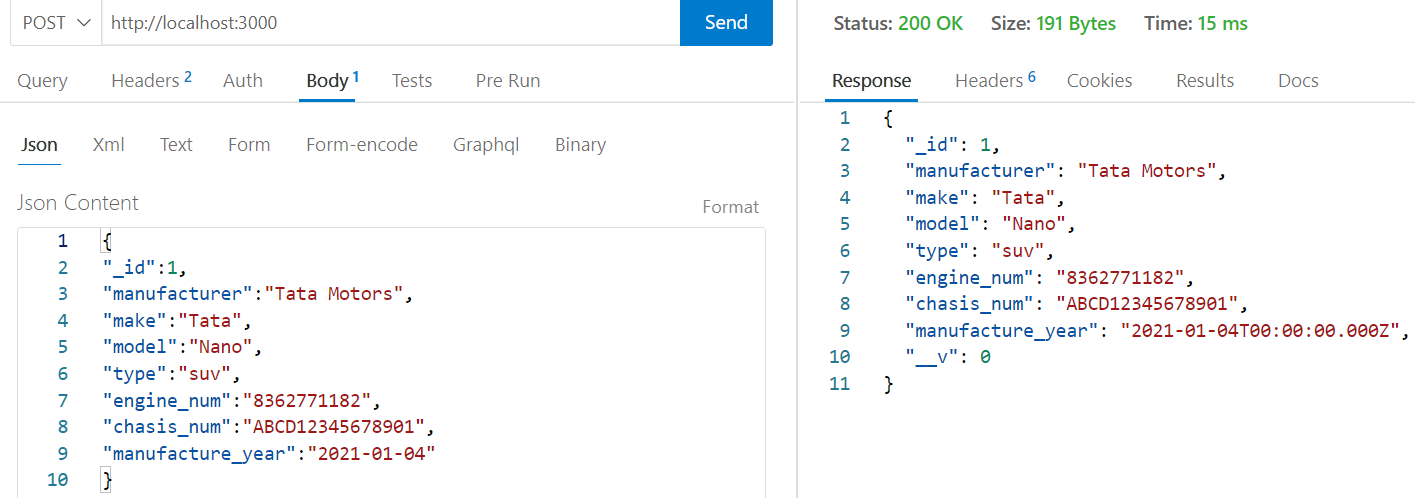
});

var Car = mongoose.model('Car', CarSchema);

module.exports = Car;



Connecting Thunder Client to MongoDB compass



{

"\_id":1,

"manufacturer":"Tata Motors",

"make":"Tata",

"model":"Nano",

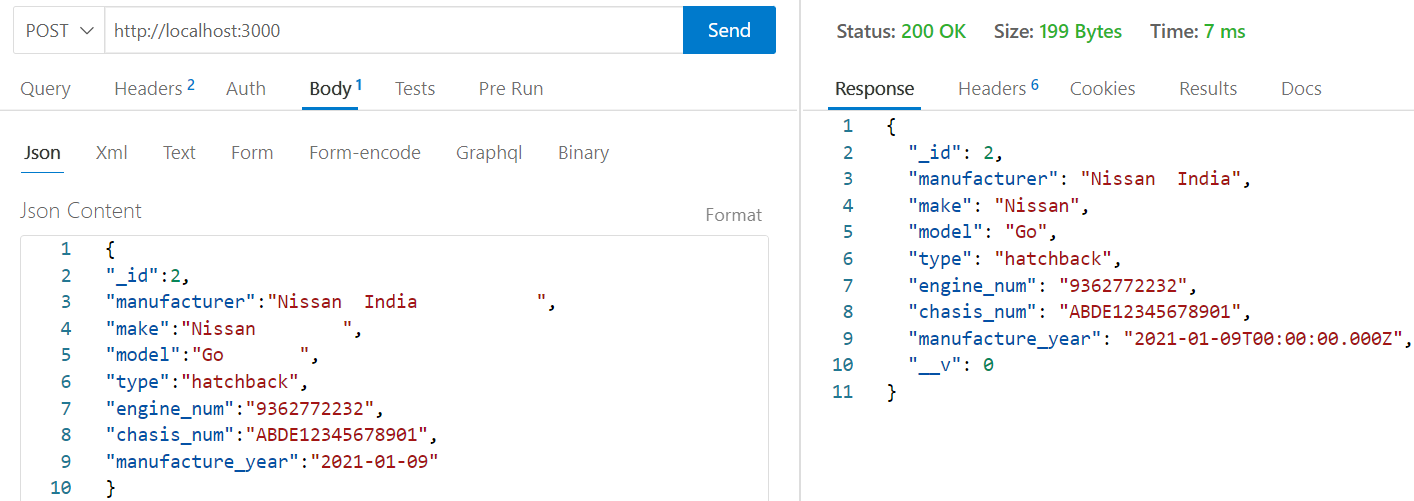
"type":"suv",

"engine\_num":"8362771182",

"chasis\_num":"ABCD12345678901",

"manufacture\_year":"2021-01-04"

}



{

"\_id":2,

"manufacturer":"Nissan India ",

"make":"Nissan ",

"model":"Go ",

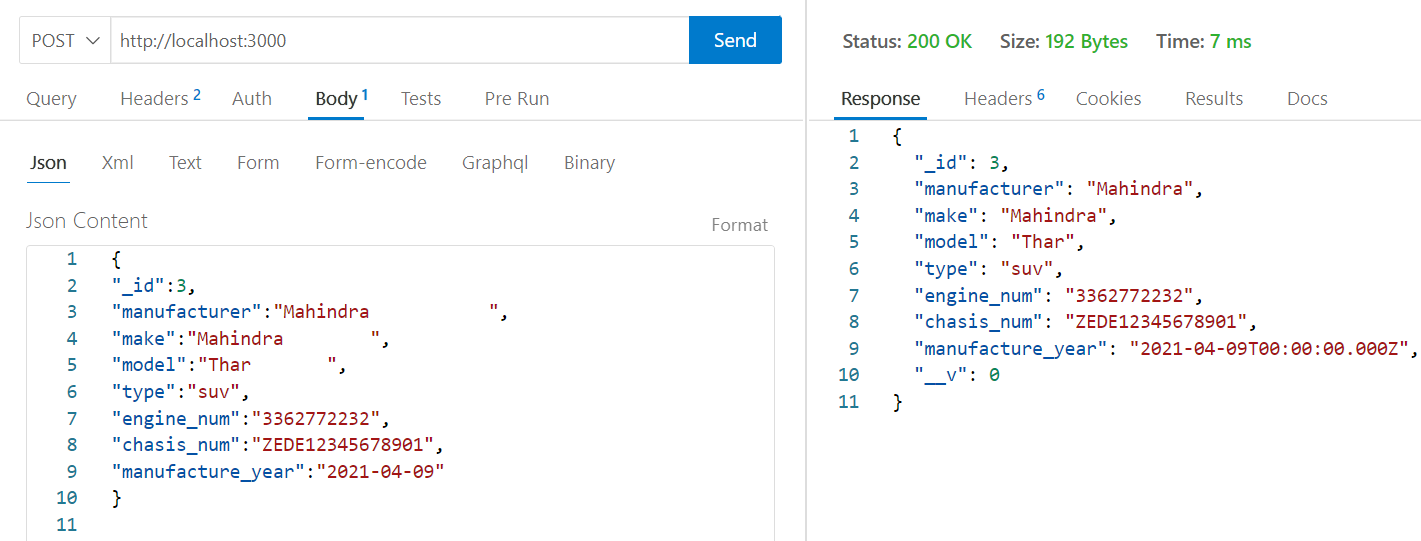
"type":"hatchback",

"engine\_num":"9362772232",

"chasis\_num":"ABDE12345678901",

"manufacture\_year":"2021-01-09"

}



{

"\_id":3,

"manufacturer":"Mahindra ",

"make":"Mahindra ",

"model":"Thar ",

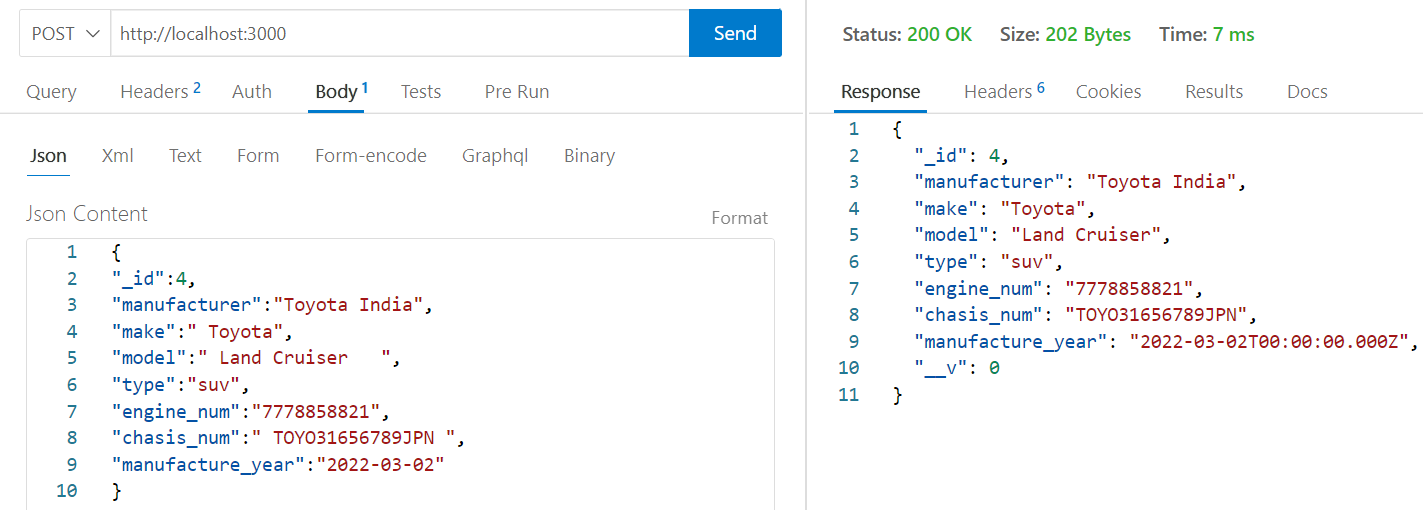
"type":"suv",

"engine\_num":"3362772232",

"chasis\_num":"ZEDE12345678901",

"manufacture\_year":"2021-04-09"

}



{

"\_id":4,

"manufacturer":"Toyota India",

"make":" Toyota",

"model":" Land Cruiser ",

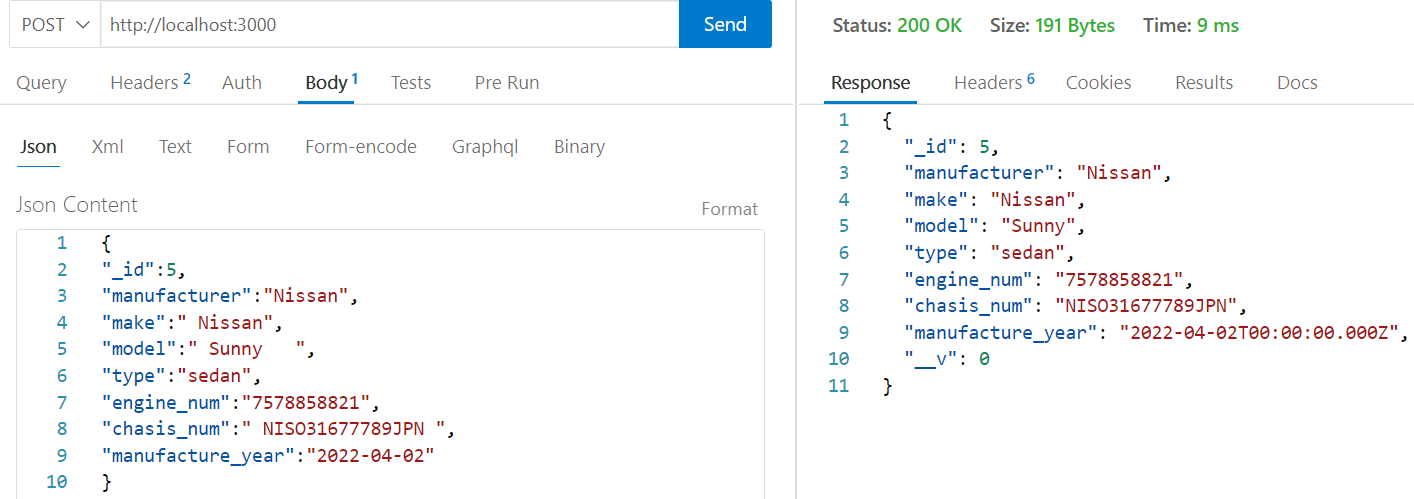
"type":"suv",

"engine\_num":"7778858821",

"chasis\_num":" TOYO31656789JPN ",

"manufacture\_year":"2022-03-02"

}

****

{

"\_id":5,

"manufacturer":"Nissan",

"make":" Nissan",

"model":" Sunny ",

"type":"sedan",

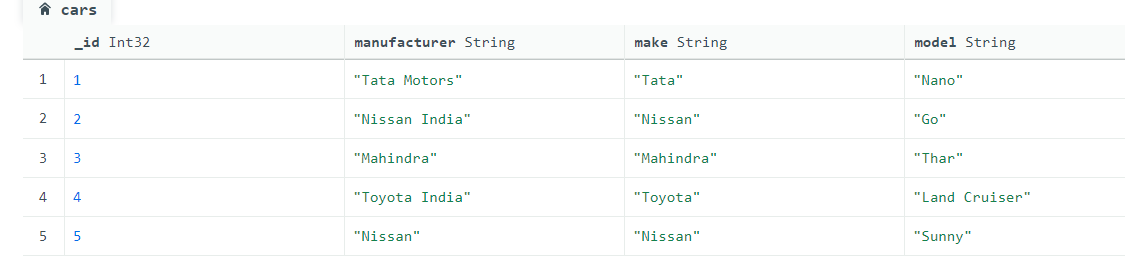
"engine\_num":"7578858821",

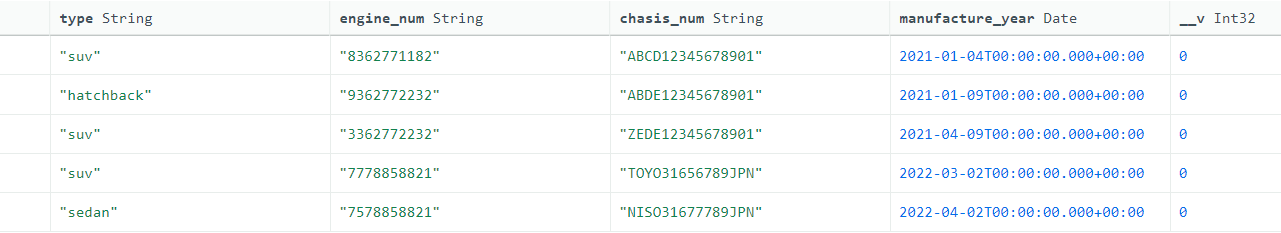
"chasis\_num":" NISO31677789JPN ",

"manufacture\_year":"2022-04-02"

}

**Output in MomgoDB Compass**





**Assignment 6**

**React Router component based on simple react routing and nested routing.**

* **Simple React Router**

**App.js**

import React from "react";

import { BrowserRouter as Router, Switch, Route, Link } from "react-router-dom";

function App() {

return (

<Router>

<div>

<Link to="/">Main page</Link> |

<Link to="/page1">Page 1</Link> |

<Link to="/page2">Page 2</Link>

</div>

<Switch>

<Route path="/page2">

<h1>I'm Page 2</h1>

</Route>

<Route path="/page1">

<h1>I'm Page 1</h1>

</Route>

<Route path="/">

<h1>I'm Main page</h1>

</Route>

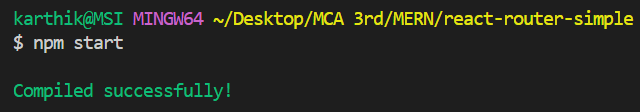
</Switch>

</Router>

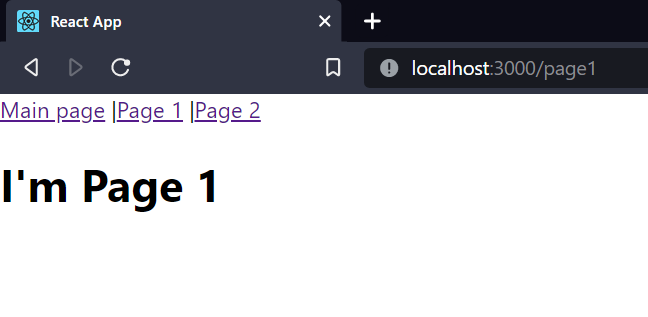
);

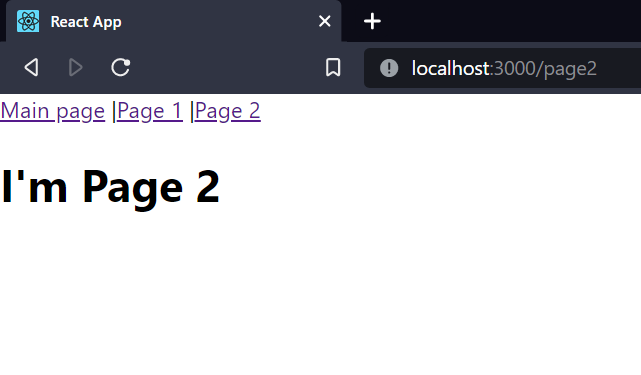
}

export default App;









* **Component based simple react routing**

**Components Folder has files(About.jsx,Gallery.jsx,Home.jsx)**

**About.jsx**

import React from "react";

const About = () => (

<div>

<h1>Hello, I'm About component</h1>

<p>

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English.

</p>

</div>

);

export default About;

**Gallery.jsx**

import React from "react";

import image1 from './images/image1.jpg'

import image2 from './images/image2.jpg'

const Gallery = () => (

<div>

<h1>Hello, I'm Gallery component</h1>

<div>

<img src={image1} alt="image1" height={150}width={150}/>

<img src={image2} alt="image2" height={150} width={150}/>

</div>

</div>

);

export default Gallery;

**Home.jsx**

import React from "react";

const Home = () => <h1>Hello, I'm Home component</h1>;

export default Home;

**App.js**

import React, { Fragment } from "react";

import { BrowserRouter as Router, Switch, Route, Link } from "react-router-dom";

import Home from "./Components/Home";

import Gallery from "./Components/Gallery";

import About from "./Components/About";

function App() {

return (

<Fragment>

<Router>

<div>

<ul>

<li>

<Link to="/">Main page</Link>

</li>

<li>

<Link to="/gallery">Gallery</Link>

</li>

<li>

<Link to="/about">About</Link>

</li>

</ul>

</div>

<Switch>

<Route path="/about">

<About />

</Route>

<Route path="/gallery">

<Gallery />

</Route>

<Route path="/">

<Home />

</Route>

</Switch>

</Router>

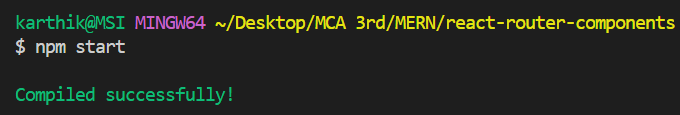
<footer> Hello I'm a little footer</footer>

</Fragment>

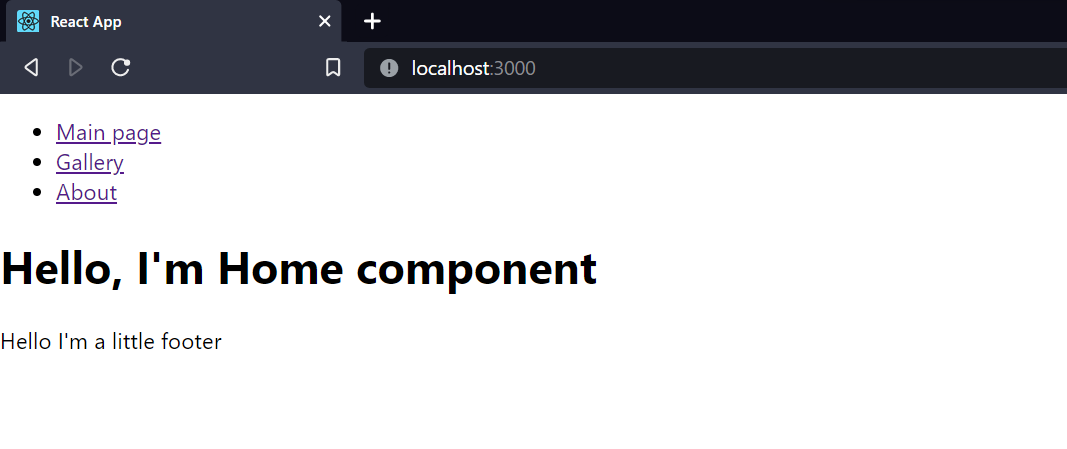
);

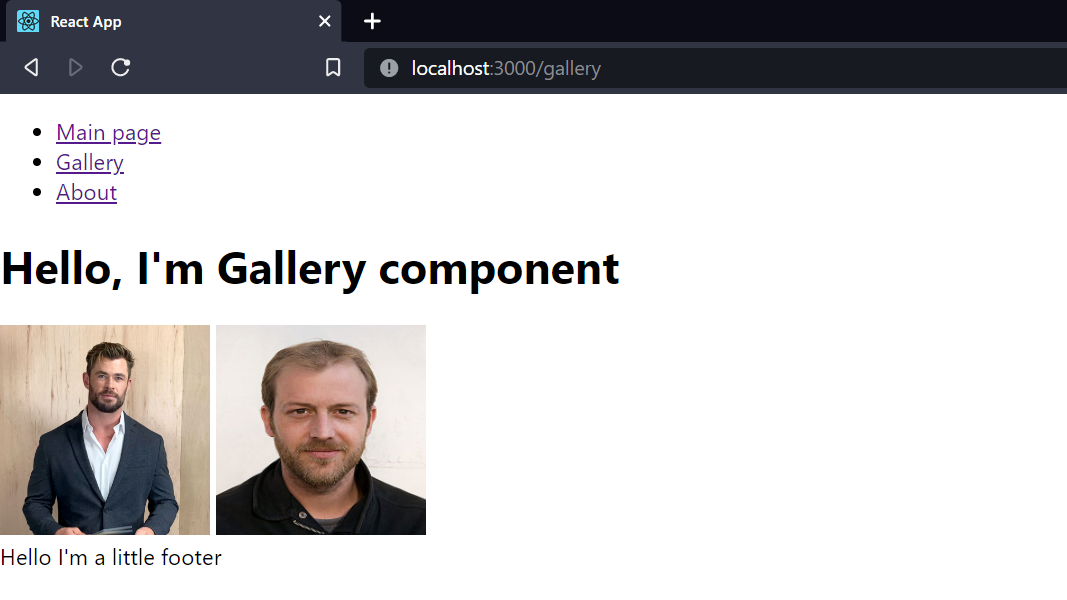
}

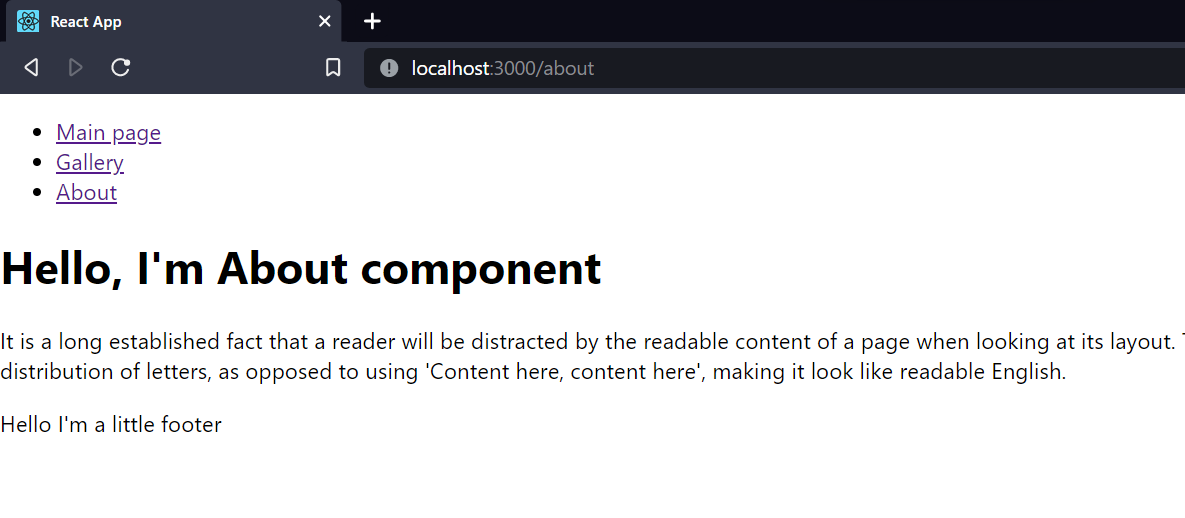
export default App;



**Output:**

****

****

****

* **Nested Routing**

**npx create-react-app react-nested-routes**

**npm install bootstap**

In components folder present in src include files(Dashboard.js,Details.js,Home.js,Details.js)

**Dashboard.js**

import React from 'react'

import { Link, Outlet } from 'react-router-dom'

function Dashboard() {

return (

<div>

<h2>React Nested Routes Example</h2>

<div className="product-nav mb-5">

<Link to="/dashboard/product">Product</Link><br></br>

<Link to="/dashboard/details"> Product Details</Link>

</div>

<Outlet />

</div>

)

}

export default Dashboard

**Details.js**

const Details = () => {

return <div className="page-wrapper">I'm in Product Details</div>

}

export default Details

**Home.js**

import React from 'react'

function Home() {

return (

<div>I'm in Home</div>

)

}

export default Home

**Product.js**

import React from 'react'

function Product() {

return (

<div>I'm in Product</div>

)

}

export default Product

**App.js**

import '../node\_modules/bootstrap/dist/css/bootstrap.min.css'

import Home from './components/Home'

import Dashboard from './components/Dashboard'

import Product from './components/Product'

import Details from './components/Details'

import { BrowserRouter, Routes, Route, Link } from 'react-router-dom'

export default function App() {

return (

<div className="container mt-5 text-center">

<BrowserRouter>

<nav>

<Link className="nav-link" to="/">

Home

</Link>

<Link className="nav-link" to="dashboard">

Dashboard

</Link>

</nav>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/dashboard" element={<Dashboard />}>

<Route path="product" element={<Product />} />

<Route path="details" element={<Details />} />

</Route>

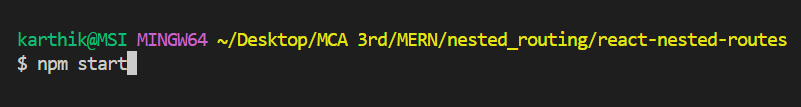
</Routes>

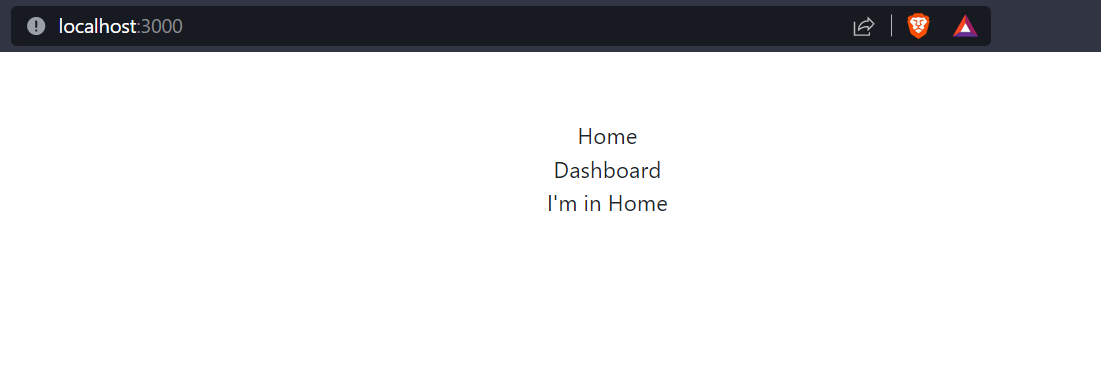
</BrowserRouter>

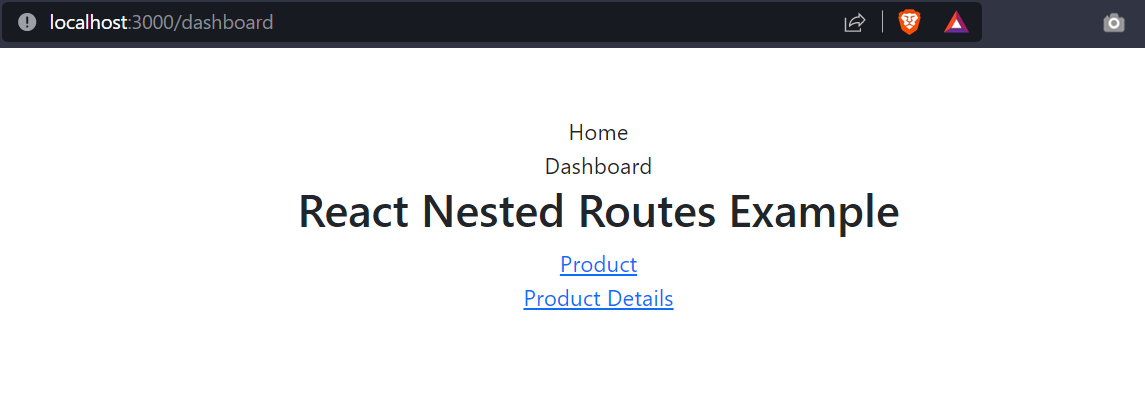
</div>

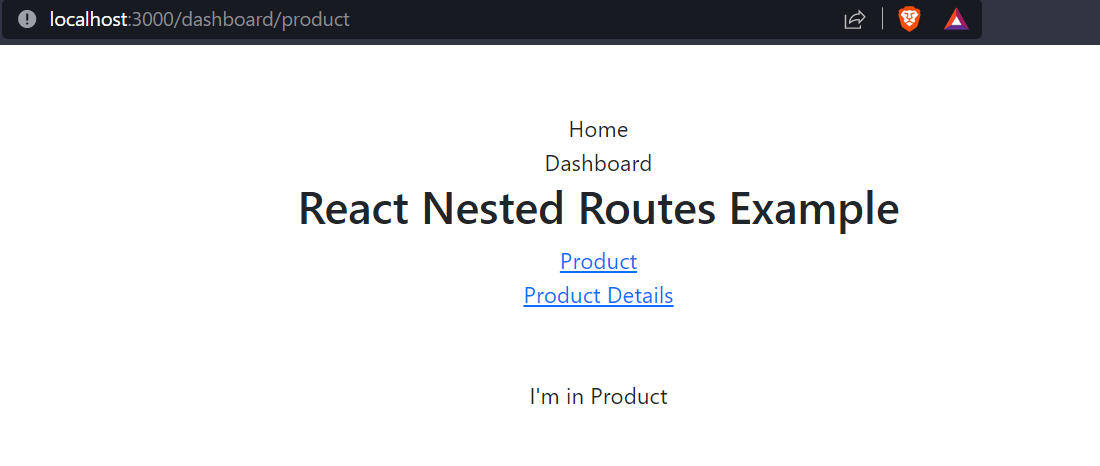
)

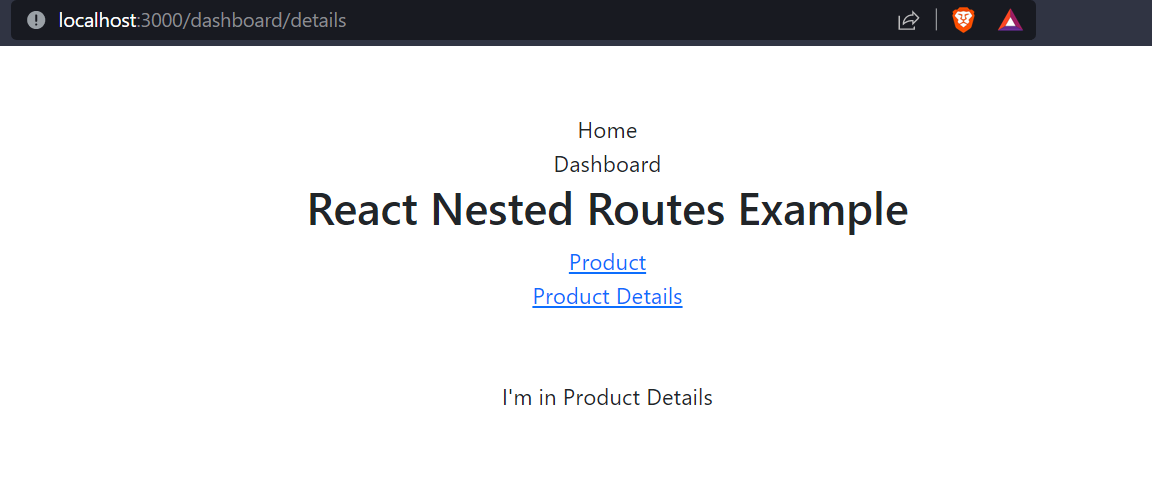
}

****

****

****

****

****

**Assignment 7**

**Create React form and perform validation for student registration page which contains basic information.**

**Form.js**

import { useState } from 'react';

export default function Form() {

// States for registration

const [fname, setFirstName] = useState('');

const [lname, setLastName] = useState('');

const [email, setEmail] = useState('');

const [address, setAddress] = useState('');

const [phone, setPhone] = useState('');

// States for checking the errors

const [submitted, setSubmitted] = useState(false);

const [error, setError] = useState(false);

// Handling the name change

const handleFName = (e) => {

if(e.target.value.match("^[a-zA-Z ]\*$")!=null){

setFirstName(e.target.value);

setSubmitted(false);

}

};

const handleLName = (e) => {

if(e.target.value.match("^[a-zA-Z ]\*$")!=null){

setLastName(e.target.value);

setSubmitted(false);

}

};

// Handling the email change

const handleEmail = (e) => {

setEmail(e.target.value);

setSubmitted(false);

};

// Handling the address change

const handleAddress = (e) => {

if(e.target.value.match("^[a-zA-Z0-9+\_.-]")!=null){

setAddress(e.target.value);

setSubmitted(false);

}

};

// Handling the phone change

const handlePhone = (e) => {

if(e.target.value.match("^[0-9]\*$")!=null){

setPhone(e.target.value);

setSubmitted(false);

}

};

// Handling the form submission

const handleSubmit = (e) => {

e.preventDefault();

if (fname === '' || lname === '' || email===''|| address === '' || phone === '') {

setError(true);

} else {

setSubmitted(true);

setError(false);

}

};

// Showing success message

const successMessage = () => {

return (

<div

className="Success"

style={{

display: submitted ? '' : 'none',

}}>

<h1>{fname} {lname} is successfully registered!!</h1>

</div>

);

};

// Showing error message if error is true

const errorMessage = () => {

return (

<div

className="error"

style={{

display: error ? '' : 'none',

}}>

<h1>Please enter all the fields</h1>

</div>

);

};

return (

<div className="form">

<div>

<h1>Student Registration Form</h1>

</div>

{/\* Calling to the methods \*/}

<div className="messages">

{errorMessage()}

{successMessage()}

</div>

<form>

{/\* Labels and inputs for form data \*/}

<label className="label">First Name</label>&nbsp;&nbsp;

<input onChange={handleFName} className="input"

value={fname} type="text" /><br></br>

<label className="label">Last Name</label>&nbsp;&nbsp;

<input onChange={handleLName} className="input"

value={lname} type="text" /><br></br>

<label className="label">Email</label>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<input onChange={handleEmail} className="input"

value={email} type="email" /><br></br>

<label className="label">Address</label>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<input onChange={handleAddress} className="input"

value={address} type="text" /><br></br>

<label className="label">Phone</label>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<input type="tel" onChange={handlePhone} className="input" maxLength={10}

value={phone}/><br></br><br></br>

<button onClick={handleSubmit} className="btn" type="submit">

Submit

</button>

</form>

</div>

);}

**App.js**

import './App.css';

import Form from "./Form"

function App() {

return (

<div className="App">

<Form />

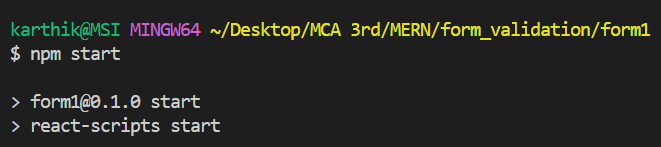
</div>

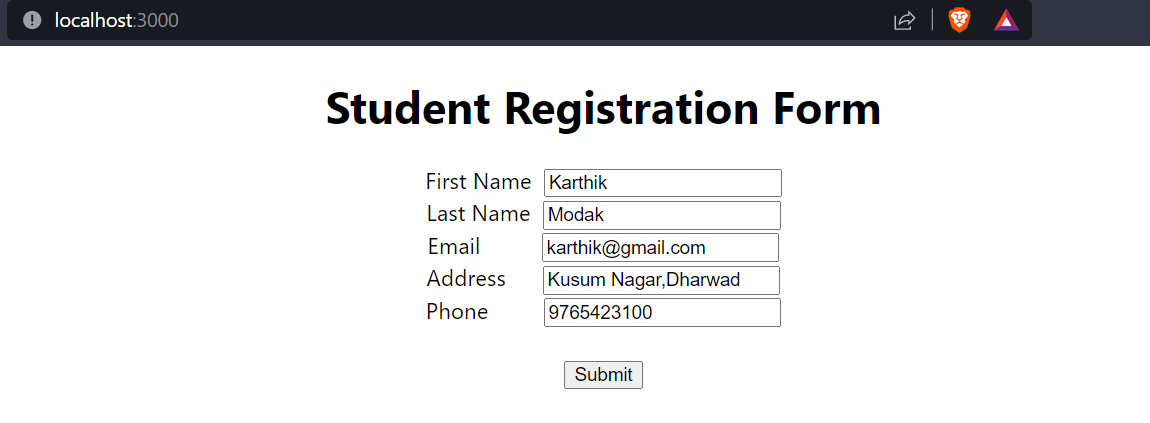
);

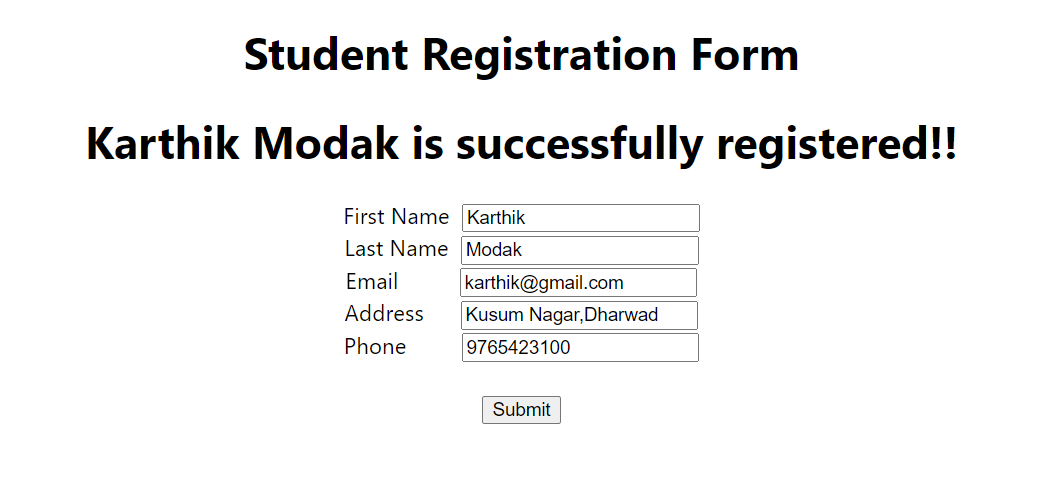
}

export default App;

**Output:**

****

****

****

**Assignment 8**

**GraphQL and its Functionalities**

GraphQL was developed to address just these concerns. As a result, GraphQL is a far more elaborate specification, with the following salient features. GraphQL is a query language for your API, and a server-side runtime for executing queries using a type system you define for your data. GraphQL isn't tied to any specific database or storage engine and is instead backed by your existing code and data.

A GraphQL service is created by defining types and fields on those types, then providing functions for each field on each type.

**Functionalities:**

1. **Field Specification**

Unlike REST APIs, where you have little control on what the server returns as part of an object, in GraphQL, the properties of an object that need to be returned must be specified. Specifying no fields of an object would, in a REST API, return the entire object. In contrast, in a GraphQL query, it is invalid to request nothing.

This lets the client control the amount of data that is transferred over the network, making it more efficient, especially for lighter front-ends such as mobile applications. Further, addition of new capabilities (fields or new APIs) does not require you to introduce a new version of the API set. Given a query, since the shape of the returned data is determined by it, the effect of it is the same, regardless of changes to the API.

A downside to this that there is a bit of a learning curve for the GraphQL query language, which must be used to make any API call. Fortunately, the specification of the language is quite simple and easy to master.

1. **Graph Based**

REST APIs were resource based, whereas GraphQL is graph based. This means that relationships between objects are naturally handled in GraphQL APIs. In the Issue Tracker application, you could think of Issues and Users having a relation: An issue is assigned to a user, and a user has one or more issues assigned to them. When querying for a user’s properties, GraphQL makes it natural to query for some properties associated with all the issues assigned to them as well

1. **Single Endpoint**

GraphQL API servers have a single endpoint in contrast to one endpoint per resource in REST. The name of the resource(s) or field(s) being accessed is supplied as part of the query itself. This makes it possible to use a single query for all the data that is required by a client. Due to the graph-based nature of the query, all related objects can be retrieved as part of a query for one object. Not only that, even unrelated objects can be queried in a single call to the API server. This obviates the need for "aggregation" services whose job was to put together multiple API results into one bundle.

1. **Strongly Typed**

GraphQL is a strongly typed query language. All fields and arguments have a type against which both queries and results can be validated and give descriptive error messages. In addition to types, it is also possible to specify which fields and arguments are required and which others are optional. All this is done using the GraphQL schema language. The advantage of a strongly typed system is that it prevents errors. This is a great thing, considering that APIs are written and consumed by different teams and there is bound to be communication gaps due to this. The type system of GraphQL has its own language for specifying the details of the types that you wish to support in your API. It supports the basic scalar types such as integer and string, objects composed of these basic data types, and custom scalar types and enumerations.

1. **Introspection**

A GraphQL server can be queried for the types it supports. This creates a powerful platform for tools and client software to build atop this information. This includes code-generation utilities in statically typed languages and explorers that let developers test and learn an API set quickly, without grepping the codebase or wrangling with cURL. We will be using one such tool, called the Apollo Playground, to test our APIs before integrating them into the application’s UI.

1. **Libraries**

Parsing and dealing with the type system language (also called the GraphQL Schema Language) as well as the query language is hard to do on your own. Fortunately, there are tools and libraries available in most languages for this purpose.

For JavaScript on the back-end, there is a reference implementation of GraphQL called GraphQL.js. To tie this to Express and enable HTTP requests to be the transport mechanism for the API calls, there is a package called express-graphql.

But these are very basic tools that lack some advanced support such as modularized schemas and seamless handling of custom scalar types. The package graphql-tools and the related apollo-server are built on top of GraphQL.js to add these advanced features. We will be using the advanced packages for the Issue Tracker application in this chapter.

I will cover only those features of GraphQL that are needed for the purpose of the application. For advanced features that you may need in your own specific application, do refer to the complete documentation of GraphQL at https://graphql.org and the tools at https://www.apollographql.com/ docs/graphql-tools/.