**Tutorial: 1**

**1. Design html page to get below given output using bootstrap css.**

Index.html :

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<title>Tutorial 01</title>

<link rel="stylesheet" type="text/css" href="css/bootstrap.css">

<link rel="stylesheet" type="text/css" href="style/style.css">

</head>

<body>

<div class="container">

<div class="row">

<div class="col-lg-6 bder colA" ></div>

<div class="col-lg-6 bder colA" ></div>

</div>

<div class="row">

<div class="col-lg-4 bder colB " ></div>

<div class="col-lg-4 bder colB" ></div>

<div class="col-lg-4 bder colB" ></div>

</div>

<div class="row">

<div class="col-lg-8 bder colC" style="background-color: green" ></div>

<div class="col-lg-4 bder colC" style="background-color: #17a2b8"></div>

</div>

<div class="row">

<div class="col-lg-3 bder colD" ></div>

<div class="col-lg-3 bder colD" ></div>

<div class="col-lg-3 bder colD" ></div>

<div class="col-lg-3 bder colD" ></div>

</div>

<div class="row">

<div class="col-lg-12 bder colE"></div>

</div>

</div>

</body>

</html>

Style.html :

.container{

align-items: center;

height: 300px;

width: 710px;

}

.bder{

border: 3px solid white;

}

.colA{

background-color: blue;

height: 40px;

}

colB{

background-color: yellow;

height: 65px;

}

.colC{

height: 65px;

}

.colD{

background-color: orange;

height: 65px;

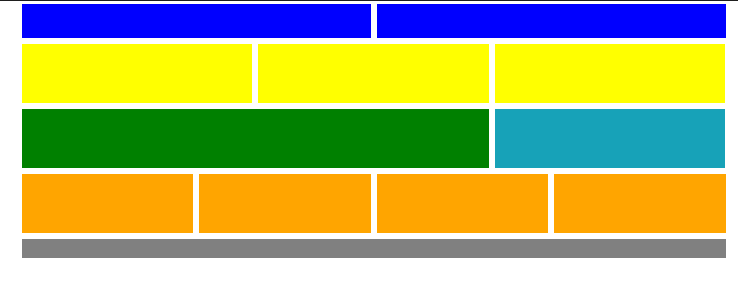
}

.colE{

background-color: gray;

height: 25px;

}



**2.  Use a Bootstrap class to style the table properly and get the following output (with padding and horizontal dividers).**

1. **Add zebra-stripes to the table.**
2. **Add borders on all sides of the table and cells.**
3. **Enable a hover state on table rows.**
4. **Make the table more compact by cutting cell padding in half.**
5. **Use contextual classes to add the following:**

* **Green color to the table row containing John.**
* **Red color to the table row containing Mary.**
* **Orange color to the last table row.**

index.html:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<title>Tutorial 01</title>

<link rel="stylesheet" type="text/css" href="css/bootstrap.css">

<link rel="stylesheet" type="text/css" href="style/style.css">

</head>

<body>

<div class="container">

<table class="table table-hover table-bordered text-white">

<tr class="font-weight-bold text-dark">

<td>Firstname</td>

<td>Lastname</td>

<td>Email</td>

</tr>

<tr class="bg-success">

<td>Jhon</td>

<td>Doe</td>

<td>jhon@example.com</td>

</tr>

<tr class="bg-danger">

<td>Marry</td>

<td>Moe</td>

<td>mary@example.com</td>

</tr>

<tr class="bg-warning">

<td>July</td>

<td>Duddly</td>

<td>july@example.com</td>

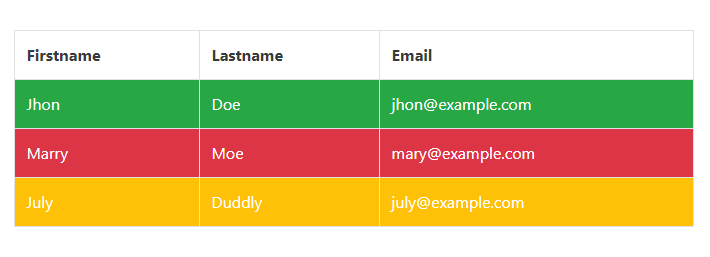
</tr>

</table>

</div>

</body>

</html>



**3. Bootstrapping with Buttons.**

**a.      Use a Bootstrap class to style the button properly with a red color.**

**b.      Change the size of the buttons in the following order: large, medium, small and xsmall.**

**c.       Make the button span the entire width of the parent element.**

**d.      Use a Bootstrap class to disable the button.**

index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<title>Tutorial 01</title>

<link rel="stylesheet" type="text/css" href="css/bootstrap.css">

<link rel="stylesheet" type="text/css" href="style/style.css">

</head>

<body>

<div class="container">

<button class="btn btn-danger">Add Me</button>

<button class="btn btn-primary btn-lg">Add Me</button>

<button class="btn btn-primary btn-md">Add Me</button>

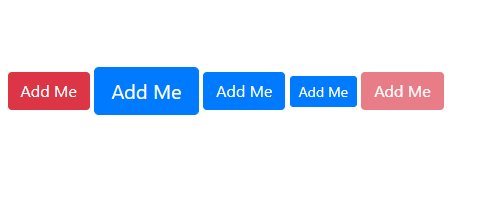
<button class="btn btn-primary btn-sm">Add Me</button>

<button class="btn btn-danger disabled">Add Me</button>

</div>

</body>

</html>



**4. Style the below given html form using bootstrap to get the output shown below.**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="UTF-8">**

**<title>Exercise #6: Simple form</title>**

**</head>**

**<body>**

**<form action="#">**

**<div>**

**<label for="first\_name">First name:</label>**

**<input type="text" name="first\_name" id="first\_name"/>**

**</div>**

**<div>**

**<label for="last\_name">Last name:</label>**

**<input type="text" name="last\_name" id="last\_name"/>**

**</div>**

**<div>**

**<label><input type="radio" name="gender" value="male"/>male</label>**

**<label><input type="radio" name="gender" value="female"/>female</label>**

**</div>**

**<div>**

**<label for="birth\_date">Date of birth:</label>**

**<input type="date" name="birth\_date" id="birth\_date"/>**

**</div>**

**<input type="submit" value="Add"/>**

**</form>**

**</body>**

**</html>**

index.html:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<title>Tutorial 01</title>

<link rel="stylesheet" type="text/css" href="css/bootstrap.css">

<link rel="stylesheet" type="text/css" href="style/style.css">

</head>

<body>

<div class="container">

<form action="#" class="text-dark font-weight-bold">

<div class="form-group">

<label for="first\_name">First name:</label>

<input type="text" name="first\_name" id="first\_name" class="form-control">

</div>

<div class="form-group">

<label for="last\_name">Last name:</label>

<input type="text" name="last\_name" id="last\_name" class="form-control">

</div>

<div class="form-group">

<label><input type="radio" name="gender" value="male" class="mr-1">male</label>

<label><input type="radio" name="gender" value="female" class="mr-1">female</label>

</div>

<div class="form-group">

<label for="birth\_date">Date of birth:</label>

<input type="date" name="birth\_date" id="birth\_date" class="form-control">

</div>

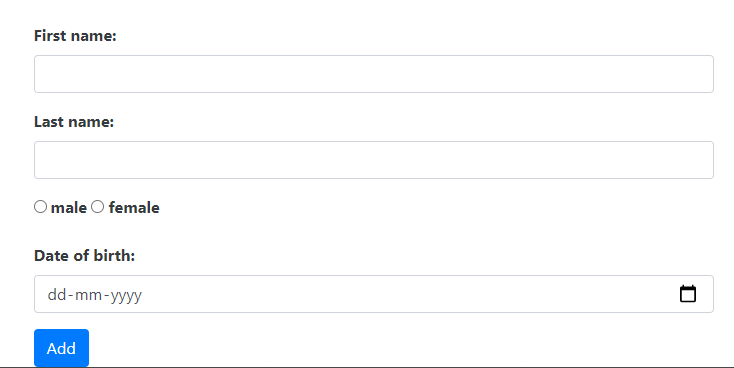
<input type="submit" value="Add" class="btn btn-primary">

</form>

</div>

</body>

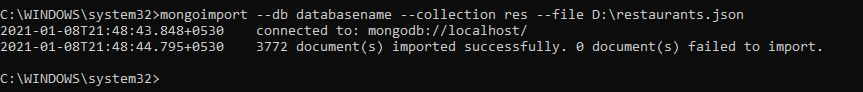
</html>



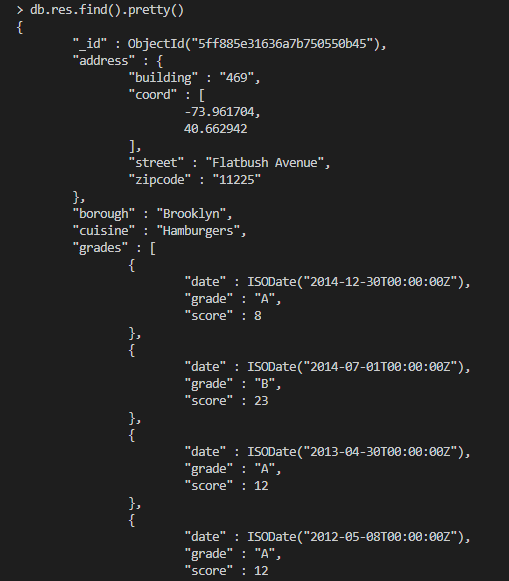
**Tutorial 02**

Import  [restaurants.json](https://www.google.com/url?q=https://canvas.rku.ac.in/courses/686/files/24880/download?wrap%3D1&sa=D&ust=1610120580475000&usg=AOvVaw2AxKxR38_urbasnk8vxj03) file using below command.

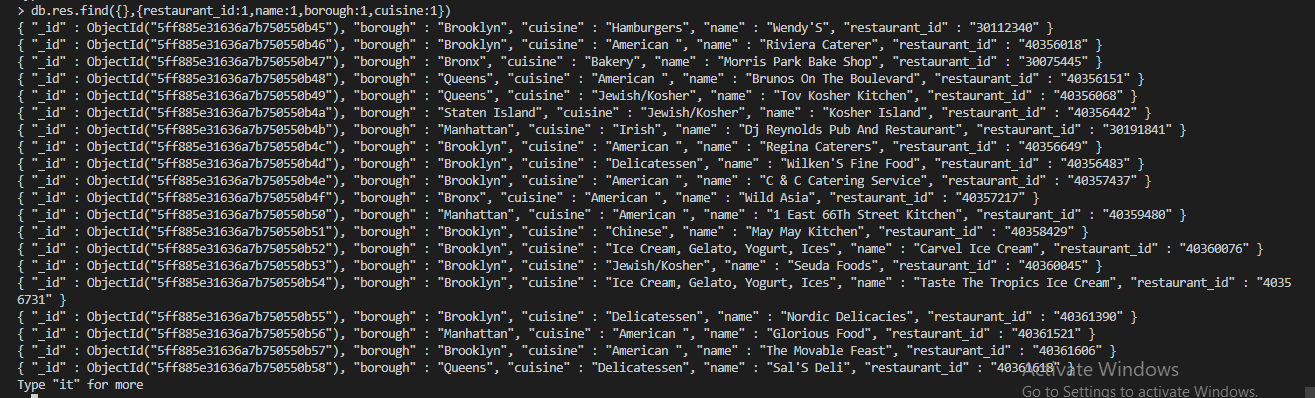
mongoimport --db databasename --collection res --file D:\restaurants.jso



1. Write a MongoDB query to display all the documents in the collection restaurants.



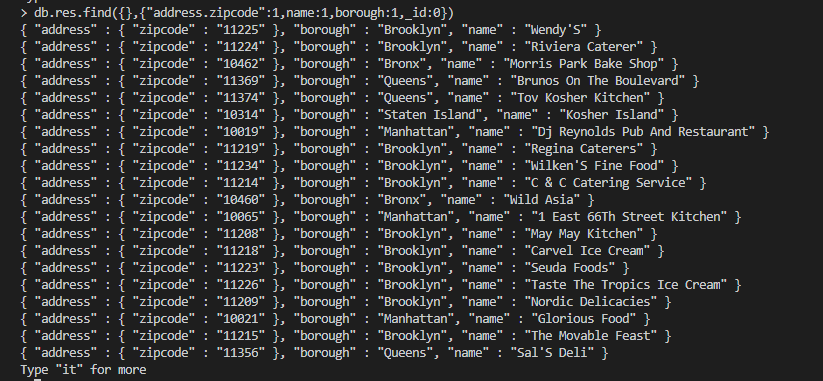
2. Write a MongoDB query to display the fields restaurant\_id, name, borough and cuisine for all the documents in the collection restaurant.



3. Write a MongoDB query to display the fields restaurant\_id, name, borough and cuisine, but exclude the field \_id for all the documents in the collection restaurant.



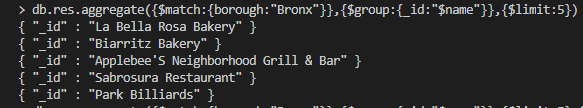
4.  Write a MongoDB query to display the fields restaurant\_id, name, borough and zip code, but exclude the field \_id for all the documents in the collection restaurant.



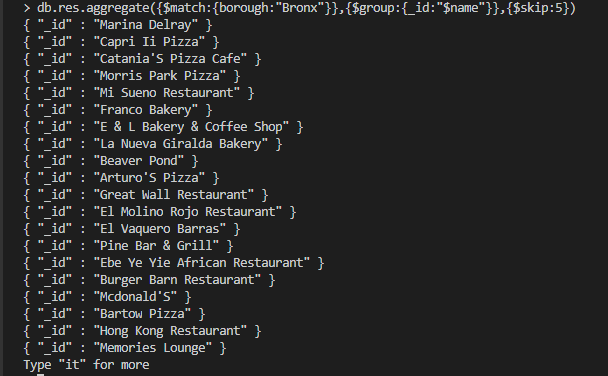
5. Write a MongoDB query to display all the restaurant which is in the borough Bronx.



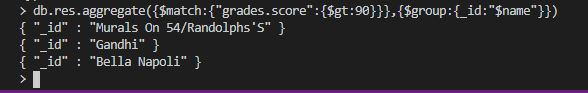
6. Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.



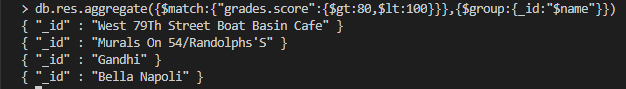
7. Write a MongoDB query to display the next 5 restaurants after skipping first 5 which are in the borough Bronx



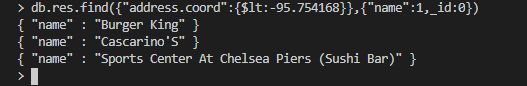
8. Write a MongoDB query to find the restaurants who achieved a score more than 90.



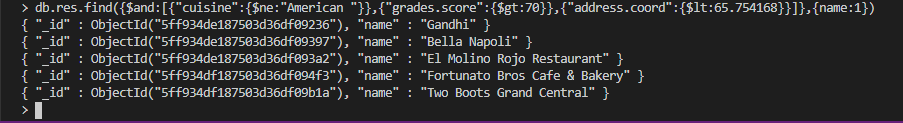
9. Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100.



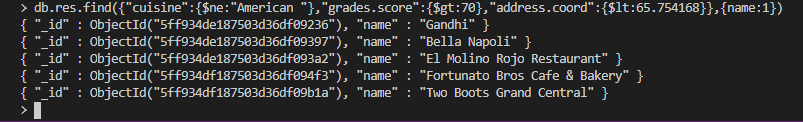
10. Write a MongoDB query to find the restaurants which locate in latitude value less than -95.754168.



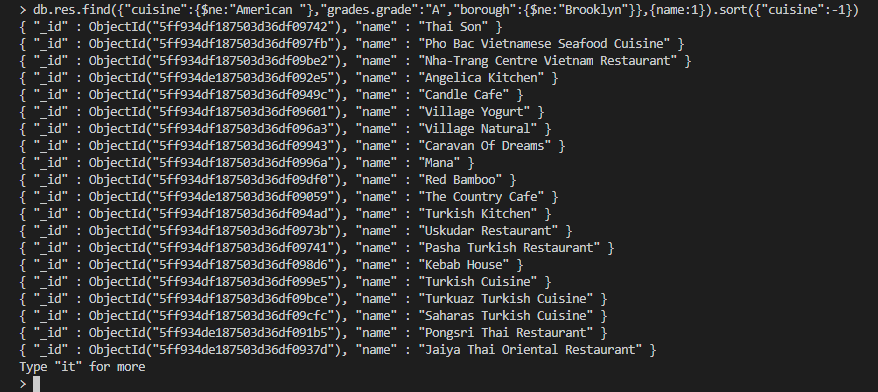
11. Write a MongoDB query to find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168.



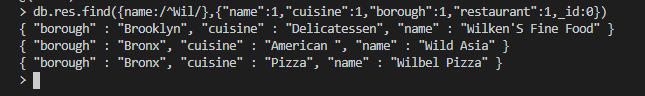
12. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American' and achieved a score more than 70 and located in the longitude less than -65.754168. Note: Do this query without using $and operator.



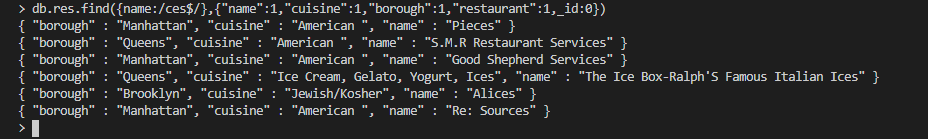
13. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American ' and achieved a grade point 'A' not belongs to the borough Brooklyn. The document must be displayed according to the cuisine in descending order.



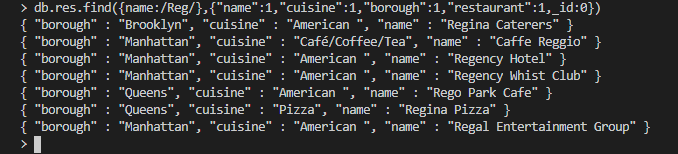
14.  Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Wil' as first three letters for its name.



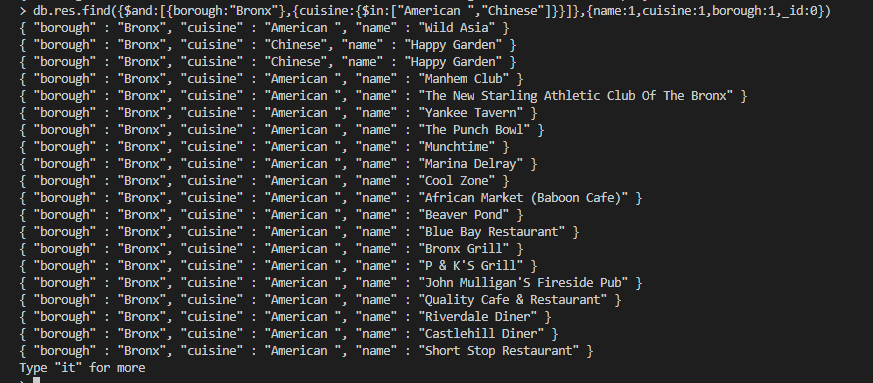
15. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'ces' as last three letters for its name.



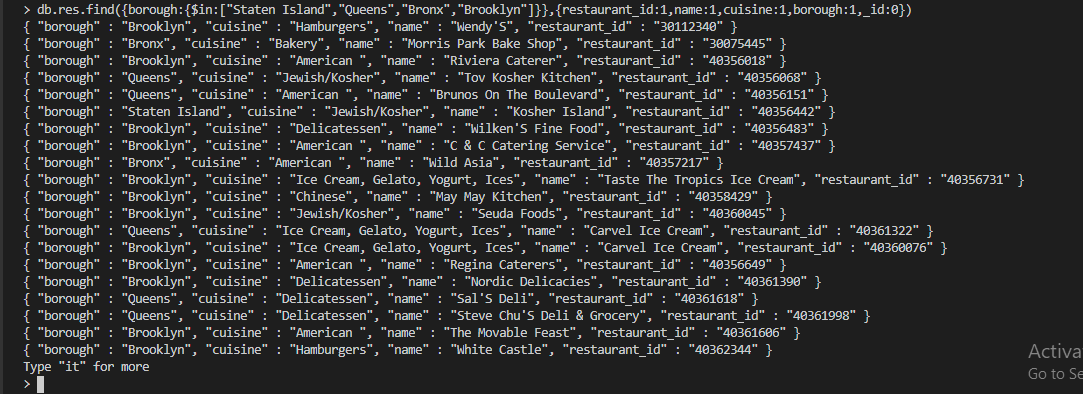
16. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Reg' as three letters somewhere in its name.



17.  Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish



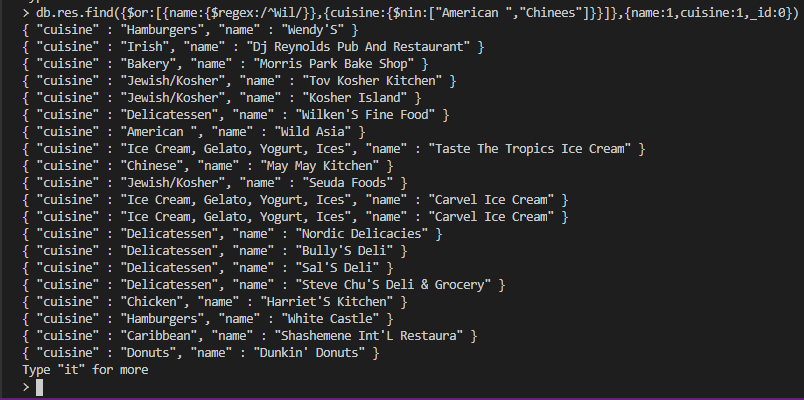
18.  Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which belong to the borough Staten Island or Queens or Bronx or Brooklyn.



19. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronx or Brooklyn.



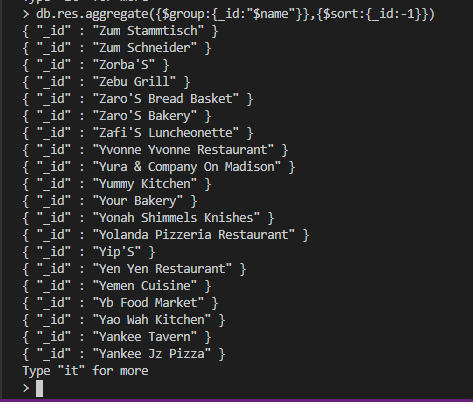
20.  Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinees' or restaurant's name begins with letter 'Wil'.



21. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.



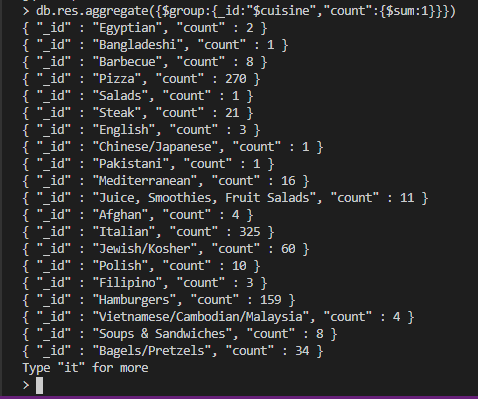
22. Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.



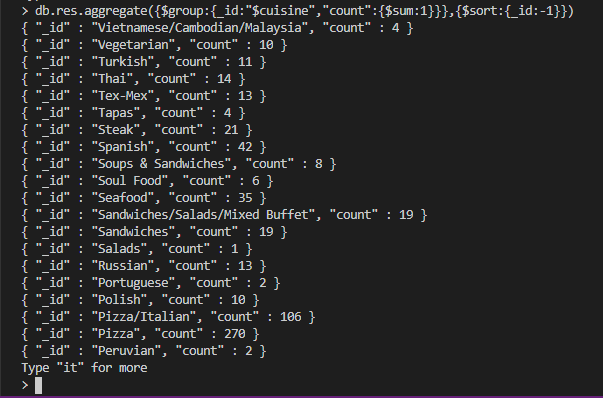
23. Write a MongoDB query to arrange the name of the cuisine in ascending order and for that same cuisine borough should be in descending order



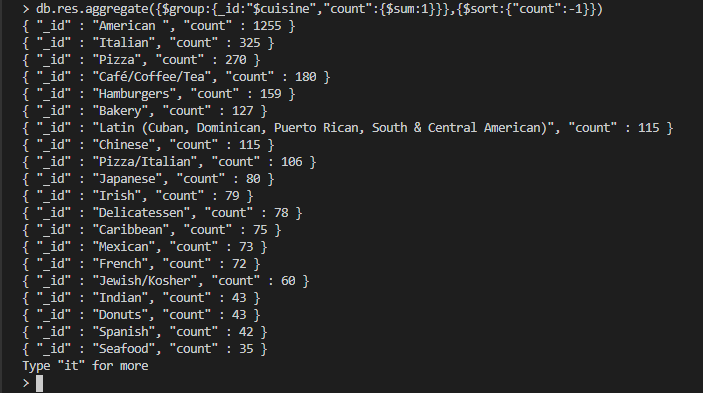
24. Find out how many times each cuisine is offered at various restaurants.



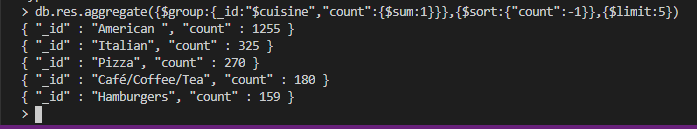
25. Find out how many times each cuisine is offered at various restaurants in descending order.



26. Which cuisine is highly offered among all restaurants?



27. Find out the top 5 highly offered cuisines among all restaurants?



**Tutorial: 3**

Que: 1) Create and Emit a custom event that checks whether the age of the person is greater than 18 or not depending on the date of birth passed to the event.

Code:

var events = require('events')

var em = new events.EventEmitter();

//create Event using On method

em.on('testEvent',(data)=>{

    if(data > 18){

        console.log("person Age is Greater than 18");

    }

    else{

        console.log("Person Age is less than 18");

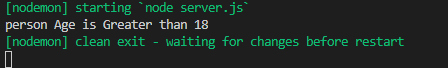
    }

})

//fire the Eevent using emit method

em.emit('testEvent',25)

Output:



Que: 2) Create a node script that gets the parameters using the GET method from the form.html file and log it on the console.

form.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <form action="http://localhost:3000/login" method="GET">

        name:<input type="text" name="fname">

        <input type="submit" name="login" value="login">

    </form>

</body>

</html>

Server.js

var http = require('http')

var url = require('url')

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

      res.writeHead(200,{'Contect-Type':'text/html'});

      var q = url.parse(req.url,true).query;

      console.log("Name is " + q.fname);

 }).listen(3000);

Output:





Que: 3) Create a node script that gets the parameters using POST method from the form.html file and log it on console.

Code:

Form.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <form action="http://localhost:3000" method="POST">

        name:<input type="text" name="fname">

        <input type="submit" name="login" value="login">

    </form>

</body>

</html>

Server.js

var http = require('http')

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

        var data1 = "";

        req.on('data',function(chank){

            data1=chank

            console.log(data1.toString());

        })

})

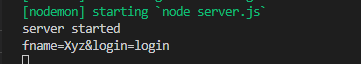
server.listen(3000,()=>{

    console.log('server started');

})

Output:





Que: 4) Create mongodb for students and nodejs that connects to mongodb using mongojs module. (install monojs and nodemon packages).

[Student document must contain: s\_id, s\_name, s\_branch, s\_city, s\_mobilenos, s\_add]

var express = require('express');

var mongojs = require('mongojs');

var bodyparser = require('body-parser')

const {ObjectId} = require('mongodb');

var dbname = 'student';

var collections = ['stud'];

var db = mongojs(dbname, collections);

var app = express();

app.use(bodyparser.urlencoded({extended:true}));

app.use(express.json());

app.listen(3000, () => {

    console.log('server started on 3000');

})

Que: 5) Modify the above created script to insert static data in student db.

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

    var s\_id =req.body.s\_id;

    var name = req.body.name;

    var branch = req.body.branch;

    var city = req.body.city;

    var mobileNo = req.body.mobileNo;

    var add = req.body.add;

    db.stud.insertMany({"s\_id":s\_id,"s\_name":name,"s\_branch":branch,"s\_city":city,"s\_mobileNo":mobileNo,"s\_add":add},(err,data)=>{

        if(!err){

             console.log(data);

             console.log("This data is inserted Successfully");

        }else{

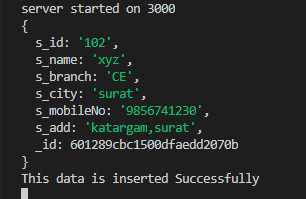
             console(err)

        }

    });

});

Output:



Que: 6) Create a nodejs that fetches all the documents from student db and logs it on the console.



Que: 7) Modify the program 6, in order to save records from the form.html file.

Code:

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <form action="http://localhost:3000/user" method="POST">

        id:<input type="text" name="s\_id"><br><br>

        name: <input type="text" name="name"><br><br>

        branch: <input type="text" name="branch"><br><br>

        city: <input type="text" name="city"><br><br>

        mobileNo: <input type="text" name="mobileNo"><br><br>

        Add: <input type="text" name="add"><br><br>

       <input type="submit" name="login" value="login">

    </form>

</body>

</html>

Server.js

var express = require('express');

var mongojs = require('mongojs');

var bodyparser = require('body-parser')

var dbname = 'student';

var collections = ['stud'];

var db = mongojs(dbname, collections);

var app = express();

app.use(bodyparser.urlencoded({extended:true}));

app.use(express.json());

app.listen(3000, () => {

    console.log('server started on 3000');

})

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

    var s\_id =req.body.s\_id;

    var name = req.body.name;

    var branch = req.body.branch;

    var city = req.body.city;

    var mobileNo = req.body.mobileNo;

    var add = req.body.add;

    db.stud.insertMany({"s\_id":s\_id,"s\_name":name,"s\_branch":branch,"s\_city":city,"s\_mobileNo":mobileNo,"s\_add":add},(err,data)=>{

        if(!err){

             console.log(data);

             console.log("This data is inserted Successfully");

        }else{

             console(err)

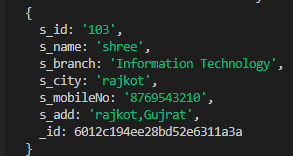
        }

    });

});

Output:





Que: 8) Create a nodejs script to update student records based on the student id.

Code:

app.put('/user/:id',(req,res)=>{

    var id = req.params.id;

    var s\_id =req.body.s\_id;

    var name = req.body.name;

    var branch = req.body.branch;

    var city = req.body.city;

    var mobileNo = req.body.mobileNo;

    var add = req.body.add;

    db.stud.update({\_id:ObjectId(id)},{$set:{"s\_id":s\_id,"s\_name":name,"s\_branch":branch,"s\_city":city,"s\_mobileNo":mobileNo,"s\_add":add}},(err,data)=>{

         if(!err){

            console.log(data);

         }

         else{

             console.log(err);

         }

    })

})

Output:



Que: 9) Create a noejs script to delete student records based on student id.

Code:

app.delete('/user/:id',(req,res)=>{

    var id = req.params.id;

    db.stud.remove({\_id:ObjectId(id)},(err,data)=>{

        if(!err){

            console.log(data);

        }else{

            console.log(err);

        }

    })

})

Output:



**Tutorial 4**

Que: 2  Create an arrow function that calculates the sum of n natural numbers. n is passed as a parameter.

//Qu1

var sum = (n:number)=>{ var sum = 0;

    for(var i=0;i<=n;i++){

    sum = sum+i;

    }

    console.log(n\*(n+1)/2);

    console.log("ans is -->> "+sum);

    }

    sum(10);

Output:



Que:3 Create three arrow functions that demonstrate usage of default parameter, optional parameter and rest parameter.

var Std\_details = (id:number,name:string,Dept:string="CE",email\_id?:string)=>{

    console.log("Id:",id);

    console.log("Name:",name);

    console.log("Dept:",Dept);

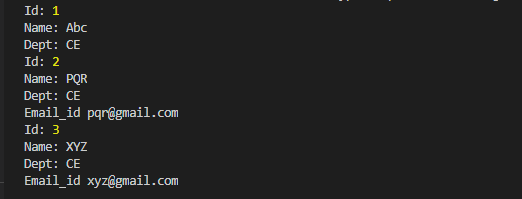
    if(email\_id!=undefined){ console.log("Email\_id",email\_id);

    }

    }

    Std\_details(1,"Abc"); Std\_details(2,"PQR","CE","pqr@gmail.com"); Std\_details(3,"XYZ","CE","xyz@gmail.com");

Output:



Que:4 Create an interface called student with name, city, branch properties and display method. Also, create an object to utilize the student interface.

interface   Student

{ S\_name:string,

    city:string,

    branch:string,

    display: ()=>any

}

var info:Student = {

S\_name:"ABc", city:"XYZ", branch:"CE",

display: ():string=>{ return " Hii hello"

}

}

console.log(info.display()); console.log(info.S\_name); console.log(info.branch); console.log(info.city);

Output:



Que: 5 Demonstrate the usage of single level and multiple inheritance of interface.

interface Person{ P\_name:string, age:number,

}

interface   Student extends Person { Branch:string

}

var info:Student = {

    P\_name:"ABC", age:18, Branch:"IT"

    }

    console.log(info.P\_name); console.log(info.Branch); console.log(info.age);

Output:



interface Person{

    G\_name:string,

}

interface G\_Parents{

    G\_name:string,

}

interface   Parents extends Person{

    P\_name:string

}

interface Child extends Parents,G\_Parents{ L\_name:string;

Child\_name:string;

}

var info:Child = {

G\_name:"XYZ", P\_name:"PQR", Child\_name:"MNO", L\_name:"ABC",

}

console.log(info.Child\_name+" "+info.P\_name+" "+info.G\_name+" "+info.L\_name);

Output:



Que:6 Define a class Clock with three private integer data members hour, min and sec. Define a no

argument constructor to initialize time value to 12:00:00. Define a three argument

constructor to initialize the time.

Define a methods to

  a. Increment time to the next second.

  b. Display the time value.

  c. Return the hour (getHour():number)

  d. Return the minute (getMinute():number)

  e. Return the seconds (getSeconds():number)

class Clock

{

private hours:number;

private min:number;

private sec:number;

constructor(hours:number, minutes:number,seconds:number)

{

this.hours = hours, this.min= minutes, this.sec = seconds

}

inc():void

{

this.sec++;

console.log("New time is " +this.hours + ":" +this.min + ":"

+this.sec);

}

display():void

{

console.log("time is "+this.hours+":"+this.min+":"+this.sec);

}

gethour():number

{

return this.hours;

}

getmin():number

{

return this.min;

}

getsec():number

{

return this.sec;

}

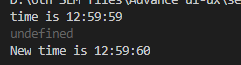
}

var obj = new Clock(12,59,59)

 console.log(obj.display())

 obj.inc()

Output:



Que:7 The employee interface for a company contains employee code, name, designation and basic pay. The employee is given a house rent allowance (HRA) of 10% of the basic pay and dearness allowance (DA) of 45% of the basic pay. The total pay of the employee is calculated as Basic Pay + HRA + DA. Write a class to define the details of the employee. Write a constructor to assign the required initial values. Add a method to calculate HRA, DA and total pay and print them. Create objects for three different employees and calculate HRA, DA and total pay.

interface Employee{

    E\_code:number

    E\_name:string

    E\_designation:string

    E\_basic\_pay:number

}

class emp   implements Employee{

    E\_code:number

    E\_name:string

    E\_designation:string

    E\_basic\_pay:number

constructor(E\_code:number,E\_name:string,E\_designation:string,E\_basic\_pay:number)

    {

    this.E\_code= E\_code

    this.E\_name = E\_name

    this.E\_designation= E\_designation

    this.E\_basic\_pay= E\_basic\_pay

    }

    Total\_pay(E\_basic\_pay:number):void{ this.E\_basic\_pay= E\_basic\_pay+E\_basic\_pay \*

    (20/100.00)+E\_basic\_pay \* (40/100.00); console.log(this.E\_basic\_pay)

    }

    display():string{

    return this.E\_code+" "+this.E\_name+" "+this.E\_designation;

    }

    }

    var obj1    = new emp(1,"ABC","XYZ",30000)

    console.log(obj1.display())

    obj1.Total\_pay(12000)

Output:



**Tutorial 05**

Create a component that displays data the from the live API created using node (refer prev tutorials).

Make sure to use following things:

* ngFor
* bootstrap (any)
* Pagination
* Sorting
* Searching
* Service
* Interface

App.module.ts:

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

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

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

import {HttpClientModule} from '@angular/common/http';

import {NgxPaginationModule} from 'ngx-pagination';

import {Ng2SearchPipeModule} from 'ng2-search-filter';

import { AppRoutingModule } from './app-routing.module';

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

import { UserComponent } from './user/user.component';

import { NavComponent } from './nav/nav.component';

import { HomeComponent } from './home/home.component';

import { APIComponent } from './api/api.component';

@NgModule({

  declarations: [

    AppComponent,

    UserComponent,

    NavComponent,

    HomeComponent,

    APIComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule,

    HttpClientModule,

    NgxPaginationModule,

    Ng2SearchPipeModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

Student-todo-service.ts:

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

import {HttpClient} from '@angular/common/http';

import {Todo} from './api/student'

import { from } from 'rxjs';

@Injectable({

  providedIn: 'root'

})

export class StudentTodoApiService {

  url = "https://jsonplaceholder.typicode.com/todos";

  constructor(private \_http:HttpClient) { }

  getData(){

    return this.\_http.get<Todo[]>(this.url);

  }

}

Interface – studnet.ts:

export interface Todo{

    UserId:number,

    Id:number,

    title:string,

    completed:boolean

}

Api.component.ts:

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

import { StudentTodoApiService } from '../student-todo-api.service';

@Component({

  selector: 'app-api',

  templateUrl: './api.component.html',

  styleUrls: ['./api.component.css']

})

export class APIComponent implements OnInit {

  todos = [];

  p=1

  title=""

  constructor(private \_studenttodoapi:StudentTodoApiService) { }

  search(){

    if(this.title!=""){

        this.todos = this.todos.filter(res=>{

          return res.title.match(this.title);

        })

    }

    else{

      this.ngOnInit()

    }

  }

  ngOnInit(): void {

    this.\_studenttodoapi.getData().subscribe((response)=>{

      this.todos = response;

      console.log(this.todos);

    })

  }

}

Api.component.html:

<div class="container">

    <form class="form-inline mt-3 d-flex justify-content-end">

        <input class="form-control mr-sm-2" type="text" name="title" [(ngModel)]="title" (ngModelChange)="search()" placeholder="Search" aria-label="Search">

        <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>

      </form>

    <div class="d-flex justify-content-center">

        <table class="table  m-4">

            <thead class="text-secondary">

                <th>UserId</th>

                <th>Id</th>

                <th>Title</th>

                <th>Complete</th>

            </thead>

            <tbody>

                <tr \*ngFor="let todo of todos | paginate:{itemsPerPage:8,currentPage:p}">

                    <td>{{todo.userId}}</td>

                    <td>{{todo.id}}</td>

                    <td>{{todo.title}}</td>

                    <td>{{todo.completed}}</td>

                </tr>

            </tbody>

        </table>

    </div>

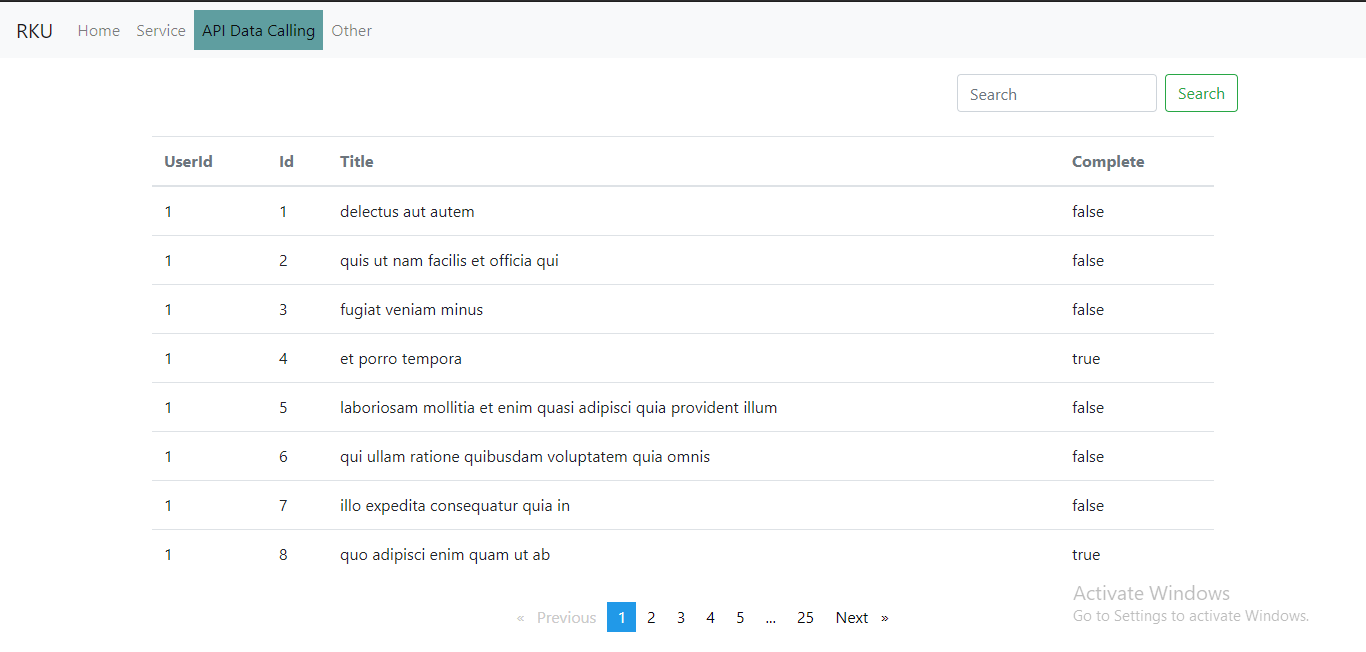
    <div class="d-flex justify-content-center">

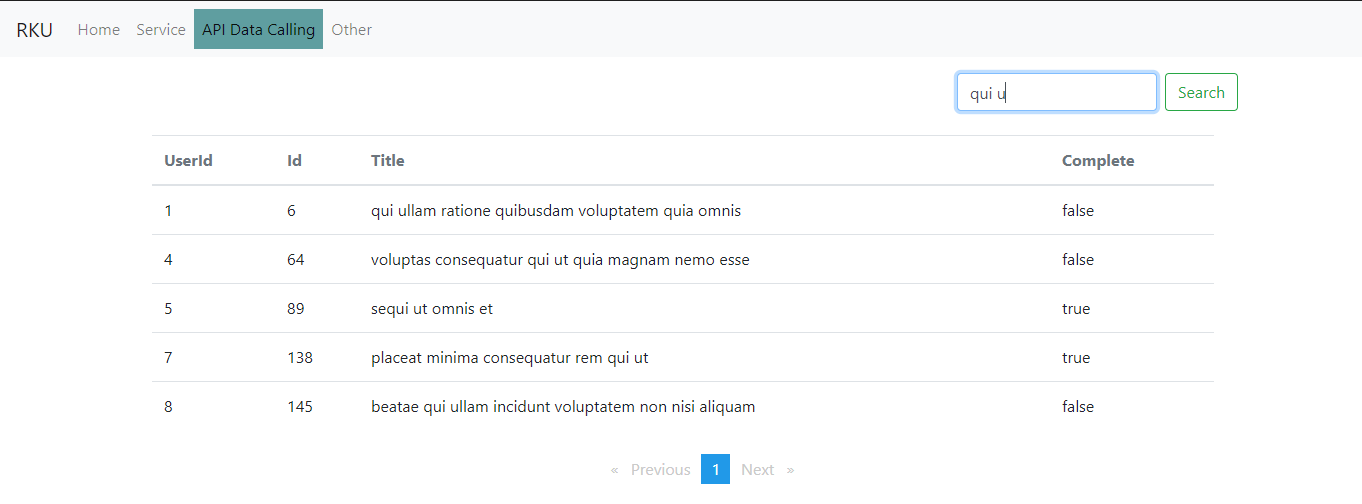
        <pagination-controls (pageChange)="p=$event" responsive="true" ></pagination-controls>

    </div>

</div>

Output:





**Tutorial 6**

Connect you Backend (NODEJS) with the front-end and utilize the post API for adding records using angular form with appropriate validations.

Book.api.component.html:

<h1 class="d-flex justify-content-center">Book Registration</h1>

<div class="container d-flex justify-content-center">

    <form #bookForm="ngForm" class="m-5" (ngSubmit)="onSubmit(bookForm)">

        <label>Book Name :</label><br>

        <input type="text" #bookname="ngModel" required name="bookname" [(ngModel)]="book.bookname"><br>

        <div class="mt-2 alert alert-danger" \*ngIf="bookname.invalid && (bookname.touched || bookname.dirty)">

            bookname is must required

        </div>

        <label>Qauntity</label><br>

        <input type="number" #bookqty required name="bookqty" [(ngModel)]="book.bookqty"><br>

        <div class="mt-2 alert alert-danger" \*ngIf="bookqty.invalid && (bookqty.touched || bookqty.dirty)">

            book Qauntity is must required

        </div>

        <button class="btn btn-primary m-1" type="submit" [disabled]="bookForm.invalid">Submit</button>

    </form>

</div>

Book.api.component.ts:

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

import { BookService } from './book.service';

@Component({

  selector: 'app-book-api',

  templateUrl: './book-api.component.html',

  styleUrls: ['./book-api.component.css']

})

export class BookApiComponent implements OnInit {

  book = {

    bookname :"",

    bookqty :""

  }

  constructor(private \_books:BookService) { }

  ngOnInit(): void {

  }

  onSubmit(bookForm){

      console.log(bookForm.value.bookname);

      console.log(bookForm.value.bookqty);

      this.\_books.addBook(bookForm.value).subscribe(res=>{

        console.log(res);

      })

  }

}

Book.service.ts:

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

import {HttpClient} from '@angular/common/http';

import { Observable } from 'rxjs';

import { book } from './book';

@Injectable({

  providedIn: 'root'

})

export class BookService {

  constructor(private \_http:HttpClient) { }

  url = "http://localhost:3000/api/books"

  addBook(data:book):Observable<book>{

      console.log(data);

      return this.\_http.post<book>(this.url,data);

  }

}

Book.ts

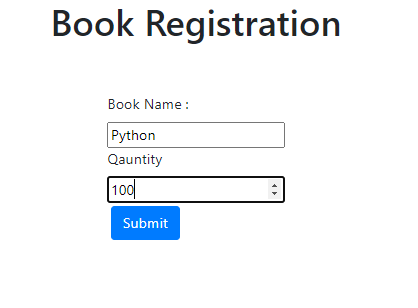
export interface book {

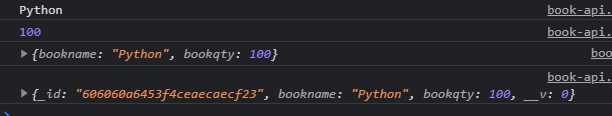
    bookname : string,

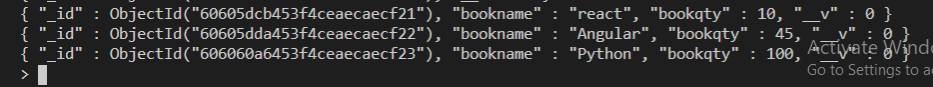
    bookqty : number

}

Output:







**Tutorial 07**

Create an editable table in order to perform crud operations. Make sure it is in-line editable and pagination is provided. Also, try to utilize "sweet-alert" to ask for the confirmation before deleting the record.

Book-api.component.html:

<div class="container">

    <h1 class="d-flex justify-content-center m-3 text-secondary">My Book Aplication</h1>

    <div class="d-flex justify-content-center">

        <table class="table  m-4">

            <thead class="text-secondary">

                <th>Id</th>

                <th>Book Name</th>

                <th>Qauntity</th>

                <th>Action</th>

            </thead>

            <tbody>

                <tr \*ngFor="let book of books">

                    <td>{{book.\_id}}</td>

                    <td>

                        <div \*ngIf="book.isEdit">

                            <input type="text" [(ngModel)]="book.bookname">

                        </div>

                        <div \*ngIf="!book.isEdit">

                            {{book.bookname}}

                        </div>

                    </td>

                    <td>

                        <div \*ngIf="book.isEdit">

                             <input type="text" [(ngModel)]="book.bookqty">

                        </div>

                        <div \*ngIf="!book.isEdit">

                            {{book.bookqty}}

                        </div>

                    </td>

                    <td>

                        <div \*ngIf="book.isEdit">

                            <button class="btn btn-warning text-white" (click)="update(book)">Update</button>

                            <button class="btn btn-secondary m-1" (click)="cancle(book)">Cancle</button>

                        </div>

                        <div \*ngIf="!book.isEdit">

                            <button class="btn btn-success" (click)="edit(book)">Edit</button>

                            <button class="btn btn-danger m-1" (click)="delete(book)">Delete</button>

                        </div>

                    </td>

                </tr>

            </tbody>

        </table>

    </div>

</div>

Book-api.component.ts :

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

import { book } from './book';

import { BookService } from './book.service';

import Swal from 'sweetalert2';

@Component({

  selector: 'app-book-api',

  templateUrl: './book-api.component.html',

  styleUrls: ['./book-api.component.css']

})

export class BookApiComponent implements OnInit {

  book = {

    bookname :"",

    bookqty :""

  }

  books = [];

  constructor(private \_books:BookService) { }

  ngOnInit(): void {

    this.\_books.getData().subscribe((response:book[])=>{

        this.books = response;

        this.books.forEach((element)=>{

          element['isEdit'] = false;

        })

        console.log(this.books);

    })

  }

  edit(book){

    book.isEdit = true;

  }

  cancle(book){

    book.isEdit = false;

  }

  update(book){

    this.\_books.update(book).subscribe(data=>{

      console.log(data);

    })

    book.isEdit = false;

  }

  delete(book){

    Swal.fire({

      title: 'Are you sure?',

      text: 'You will not be able to recover this imaginary file!',

      icon: 'warning',

      showCancelButton: true,

      confirmButtonText: 'Yes, delete it!',

      cancelButtonText: 'No, keep it'

    }).then((result) => {

      if (result.value) {

        Swal.fire(

          'Deleted!',

          'Your imaginary file has been deleted.',

          'success'

        )

        this.\_books.delete(book).subscribe(data=>{

          console.log(data);

        })

        this.ngOnInit()

      } else if (result.dismiss === Swal.DismissReason.cancel) {

        Swal.fire(

          'Cancelled',

          'Your imaginary file is safe :)',

          'error'

        )

      }

    })

  }

  onSubmit(bookForm){

      console.log(bookForm.value.bookname);

      console.log(bookForm.value.bookqty);

      this.\_books.addBook(bookForm.value).subscribe(res=>{

        console.log(res);

      })

  }

}

Book.ts:

export interface book {

    \_id : string,

    bookname : string,

    bookqty : number

}

Book.service.ts

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

import {HttpClient} from '@angular/common/http';

import { Observable } from 'rxjs';

import { book } from './book';

@Injectable({

  providedIn: 'root'

})

export class BookService {

  constructor(private \_http:HttpClient) { }

  url = "http://localhost:3000/api/books/"

  getData(){

    return this.\_http.get<book[]>(this.url);

  }

  addBook(data:book):Observable<book>{

      console.log(data);

      return this.\_http.post<book>(this.url,data);

  }

  update(data:book):Observable<book>{

    return this.\_http.patch<book>(this.url+data.\_id , data)

  }

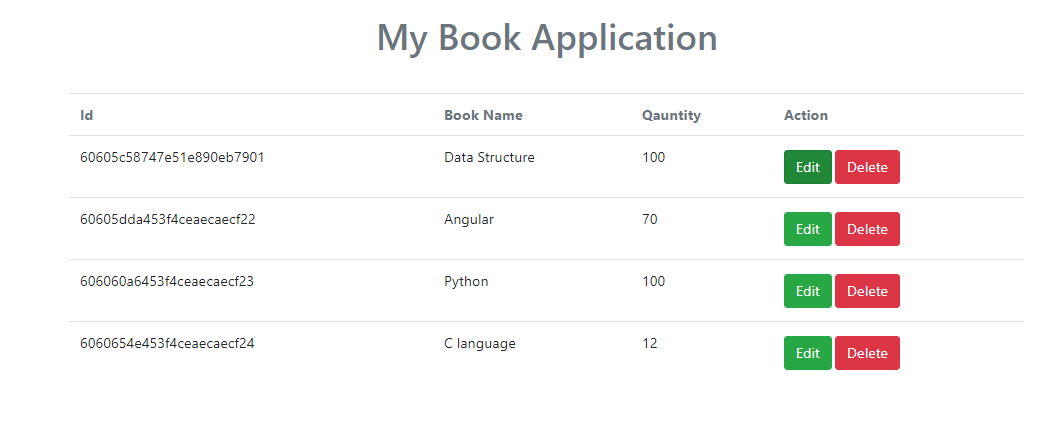
  delete(data:book):Observable<book>{

   return this.\_http.delete<book>(this.url+data.\_id)

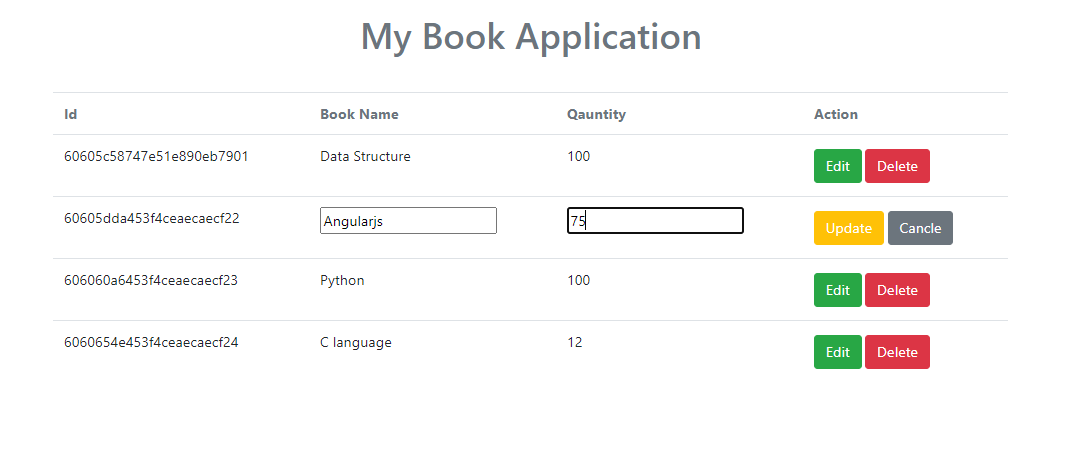
  }

}

Output:

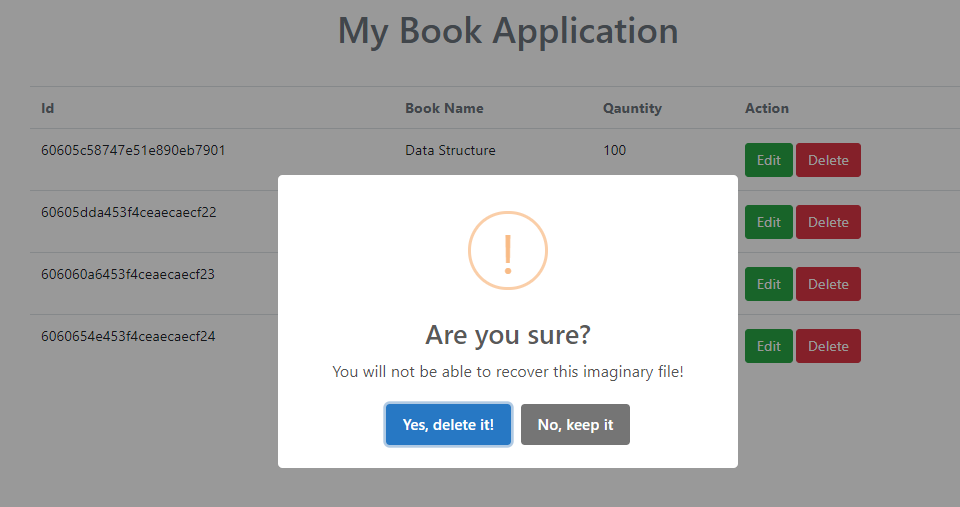


Update Output:





Delete Output:





**Tutorial 8**

Auth.service.ts:

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

import {HttpClient} from '@angular/common/http';

import { Router } from '@angular/router';

import { user } from './user';

import { Observable } from 'rxjs';

import { book } from './book/book';

@Injectable({

  providedIn: 'root'

})

export class AuthService {

  url = "http://localhost:3000/api"

  bookurl = "http://localhost:5000/api/books/"

  constructor(private \_http:HttpClient , private \_router:Router) { }

  register(user:user):Observable<any>{

    return this.\_http.post<any>(this.url+"/registration",user);

  }

  login(user:user){

    return this.\_http.post<any>(this.url+"/login",user)

    .subscribe((res:any)=>{

        localStorage.setItem('token',res.token)

        this.\_router.navigate(['/book']);

    });

   }

   isLoggedIn():boolean{

      let token = localStorage.getItem('token');

      return (token) !== null ? true : false

   }

   logout(){

     if(localStorage.removeItem('token')==null){

       this.\_router.navigate(['/login'])

     }

   }

   getBooks():Observable<any>{

     return this.\_http.get<book[]>(this.bookurl)

   }

   getAccessToken(){

     return localStorage.getItem('token')

   }

}

Auth.guard.ts:

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

import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router } from '@angular/router';

import { Observable } from 'rxjs';

import { AuthService } from './auth.service';

@Injectable({

  providedIn: 'root'

})

export class AuthGuard implements CanActivate {

  constructor(private \_auth:AuthService,private \_router:Router){}

  canActivate():boolean{

      if(this.\_auth.isLoggedIn()!== true){

        window.alert("Access not allowed please login!!")

        this.\_router.navigate(['/login'])

      }

      return true;

  }

}

App.module.ts:

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

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

import {HttpClientModule} from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';

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

import { LoginComponent } from './login/login.component';

import { RegistrationComponent } from './registration/registration.component';

import { BookComponent } from './book/book.component';

import { AuthGuard } from './auth.guard';

import { AuthService } from './auth.service';

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

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

@NgModule({

  declarations: [

    AppComponent,

    LoginComponent,

    RegistrationComponent,

    BookComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    HttpClientModule,

    FormsModule,

    ReactiveFormsModule

  ],

  providers: [AuthGuard,AuthService],

  bootstrap: [AppComponent]

})

export class AppModule { }

app.routing.ts:

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

import { RouterModule, Routes } from '@angular/router';

import { AuthGuard } from './auth.guard';

import { BookComponent } from './book/book.component';

import { LoginComponent } from './login/login.component';

import { RegistrationComponent } from './registration/registration.component';

const routes: Routes = [

  {path:'',redirectTo:'/login',pathMatch:'full'},

  {path:'login',component:LoginComponent},

  {path: 'registration',component:RegistrationComponent},

  {path:'book',component:BookComponent,canActivate:[AuthGuard]}

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

app.component.html:

<nav class="navbar navbar-expand-lg navbar-light bg-light">

  <a class="navbar-brand" href="#">Shree</a>

  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

    <span class="navbar-toggler-icon"></span>

  </button>

  <div class="collapse navbar-collapse" id="navbarSupportedContent">

    <ul class="navbar-nav ml-auto">

      <li class="nav-item">

        <a class="nav-link" \*ngIf="!this.\_auth.isLoggedIn()" routerLink="login" routerLinkActive="active">Login</a>

      </li>

      <li class="nav-item">

          <a class="nav-link" \*ngIf="!this.\_auth.isLoggedIn()" routerLink="registration" routerLinkActive="active">Registration</a>

      </li>

      <li class="nav-item">

        <a class="nav-link" routerLink="book">Book</a>

      </li>

      <li class="nav-item">

        <button class=" btn btn-danger text-white" \*ngIf="this.\_auth.isLoggedIn()" (click)="logout()" >Logout</button>

      </li>

    </ul>

  </div>

</nav>

<router-outlet></router-outlet>

App.component.ts:

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

import { AuthService } from './auth.service';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'myapp';

  constructor(public \_auth:AuthService){}

  logout(){

    this.\_auth.logout();

  }

}

User.ts:

export interface user{

    \_id: String,

    uname : String,

    pswd : String

}

Registration.component.html:

<div class="container col-4 mt-4">

    <form [formGroup]="signup" (ngSubmit)="onSubmit()">

        <div class="form-group">

          <label>Username</label>

          <input type="text" formControlName="uname" class="form-control" placeholder="Enter email">

          <div class="alert alert-danger col-8 mt-2" \*ngIf="signup.controls.uname.invalid && (signup.controls.uname.touched || signup.controls.uname.dirty)">

            <div \*ngIf="signup.controls.uname.errors.required">

                username is compulsary

            </div>

            <div \*ngIf="signup.controls.uname.errors.minlength">

                Minimum length should be 3

            </div>

           </div>

        </div>

        <div class="form-group">

          <label for="exampleInputPassword1">Password</label>

          <input type="password" formControlName="pswd" class="form-control" placeholder="Password">

          <div class="alert alert-danger col-8 mt-2" \*ngIf="signup.get('pswd').invalid && (signup.get('pswd').touched || signup.get('pswd').dirty)">

            <div \*ngIf="signup.controls.pswd.errors.required">

                Passwd is compulsary

            </div>

            <div \*ngIf="signup.controls.pswd.errors.minlength">

                Minimum length should be 3

            </div>

        </div>

        </div>

        <button type="submit" class="btn btn-primary" [disabled]="signup.invalid">Submit</button>

      </form>

</div>

Registration.component.ts:

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

import { FormControl, FormGroup, Validators } from '@angular/forms';

import { Router } from '@angular/router';

import { AuthService } from '../auth.service';

@Component({

  selector: 'app-registration',

  templateUrl: './registration.component.html',

  styleUrls: ['./registration.component.css']

})

export class RegistrationComponent implements OnInit {

  signup:FormGroup;

  constructor(private \_authService:AuthService,private \_router:Router) { }

  username = {

    uname : "",

    pswd : ""

  }

  ngOnInit(): void {

    this.signup = new FormGroup({

      uname:new FormControl(null,[Validators.required,Validators.minLength(3)]),

      pswd:new FormControl(null,[Validators.required,Validators.minLength(3)])

    })

  }

  onSubmit(){

    this.\_authService.register(this.signup.value).subscribe(res=>{

      console.log(res);

      this.signup.reset();

      this.\_router.navigate(['/login']);

    })

  }

}

Login.component.html:

<div class="container col-4 mt-4">

    <form [formGroup]="login" (ngSubmit)="onLoginUser()">

        <div class="form-group">

          <label>Username</label>

          <input type="text" class="form-control" formControlName="uname"   placeholder="Enter email">

          <div class="alert alert-danger col-8 mt-2" \*ngIf="login.controls.uname.invalid && (login.controls.uname.touched || login.controls.uname.dirty)">

            <div \*ngIf="login.controls.uname.errors.required">

                username is compulsary

            </div>

          </div>

        </div>

        <div class="form-group">

          <label>Password</label>

          <input type="password" class="form-control" formControlName="pswd" placeholder="Password">

          <div class="alert alert-danger col-8 mt-2" \*ngIf="login.controls.pswd.invalid && (login.controls.pswd.touched || login.controls.pswd.dirty)">

            <div \*ngIf="login.controls.pswd.errors.required">

                Password is compulsary

            </div>

          </div>

        </div>

        <button type="submit" class="btn btn-primary" [disabled]="login.invalid">Submit</button>

      </form>

</div>

Login. Component.ts:

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

import { FormControl, FormGroup, Validators } from '@angular/forms';

import { Router } from '@angular/router';

import { AuthService } from '../auth.service';

@Component({

  selector: 'app-login',

  templateUrl: './login.component.html',

  styleUrls: ['./login.component.css']

})

export class LoginComponent implements OnInit {

  login : FormGroup;

  constructor(private \_auth:AuthService,private \_router:Router) { }

  ngOnInit(): void {

    this.login = new FormGroup({

      uname:new FormControl(null,[Validators.required]),

      pswd:new FormControl(null,[Validators.required])

    })

 }

 onLoginUser(){

   console.log(this.login.value);

   this.\_auth.login(this.login.value);

 }

}

Book.ts:

export interface book {

    \_id : string,

    bookname : string,

    bookqty : number

}

Book.component.html:

<div class="container">

    <h1 class="d-flex justify-content-center m-3 text-secondary">My Book App/lication</h1>

    <div class="d-flex justify-content-center">

        <table class="table  m-4">

            <thead class="text-secondary">

                <th>Id</th>

                <th>Book Name</th>

                <th>Qauntity</th>

                <th>Action</th>

            </thead>

            <tbody>

                <tr \*ngFor="let book of books">

                    <td>{{book.\_id}}</td>

                    <td>

                        <div \*ngIf="book.isEdit">

                            <input type="text" [(ngModel)]="book.bookname">

                        </div>

                        <div \*ngIf="!book.isEdit">

                            {{book.name}}

                        </div>

                    </td>

                    <td>

                        <div \*ngIf="book.isEdit">

                             <input type="text" [(ngModel)]="book.bookqty">

                        </div>

                        <div \*ngIf="!book.isEdit">

                            {{book.qty}}

                        </div>

                    </td>

                    <td>

                        <div \*ngIf="book.isEdit">

                            <button class="btn btn-warning text-white" (click)="update(book)">Update</button>

                            <button class="btn btn-secondary m-1" (click)="cancle(book)">Cancle</button>

                        </div>

                        <div \*ngIf="!book.isEdit">

                            <button class="btn btn-success" (click)="edit(book)">Edit</button>

                            <button class="btn btn-danger m-1" (click)="delete(book)">Delete</button>

                        </div>

                    </td>

                </tr>

            </tbody>

        </table>

    </div>

</div>

Book.component.ts:

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

import { AuthService } from '../auth.service';

import { book } from './book';

@Component({

  selector: 'app-book',

  templateUrl: './book.component.html',

  styleUrls: ['./book.component.css']

})

export class BookComponent implements OnInit {

  constructor(private \_auth:AuthService) { }

  book = {

    bookname :"",

    bookqty :""

  }

  books = [];

  ngOnInit(): void {

    this.\_auth.getBooks().subscribe((response:book[])=>{

      this.books = response;

      console.log(this.books);

  })

  }

}

Output:

