**Tutorial – 1**

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

A picture containing icon

Description automatically generated

**Code:**

<!DOCTYPE html>

<html lang="en">

<head>

  <title>Chetan Kochiyaniya</title>

  <meta name="viewport" content="width=device-width, initial-scale=1">

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

    <script src="js/bootstrap.min.js" ></script>

    <!--

        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">

        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" ></script>

    -->

</head>

<body>

    <div class="container-fluid">

      <div class="row g-2 mb-1">

        <div class="col-6">

          <div class="p-3 border bg-primary"></div>

        </div>

        <div class="col-6">

          <div class="p-3 border bg-primary"></div>

        </div>

      </div>

      <div class="row g-2 mb-1">

        <div class="col-4">

          <div class="p-4 border bg-warning"></div>

        </div>

        <div class="col-4">

          <div class="p-4 border bg-warning"></div>

        </div>

        <div class="col-4">

          <div class="p-4 border bg-warning"></div>

        </div>

      </div>

      <div class="row g-2 mb-1">

        <div class="col-8">

          <div class="p-4 border bg-success"></div>

        </div>

        <div class="col-4">

          <div class="p-4 border bg-info"></div>

        </div>

      </div>

      <div class="row g-2 mb-1">

        <div class="col-3">

          <div class="p-4 border bg-danger"></div>

        </div>

        <div class="col-3">

          <div class="p-4 border bg-danger"></div>

        </div>

        <div class="col-3">

          <div class="p-4 border bg-danger"></div>

        </div>

        <div class="col-3">

          <div class="p-4 border bg-danger"></div>

        </div>

      </div>

      <div class="row g-2">

        <div class="col-12">

          <div class="p-2 border bg-secondary"></div>

        </div>

      </div>

    </div>

</body>

</html>

**Output:**

**Chart

Description automatically generated**

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

Graphical user interface, text, application, email

Description automatically generated

**Code:**

<!DOCTYPE html>

<html lang="en">

<head>

  <title>Chetan Kochiyaniya</title>

  <meta name="viewport" content="width=device-width, initial-scale=1">

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

    <script src="js/bootstrap.min.js" ></script>

</head>

<body>

  <div class="container mt-3">

       <table class="table table-striped table-bordered table-hover table-sm">

          <thead>

            <tr>

              <th>Firstname</th>

              <th>Lastname</th>

              <th>Email</th>

            </tr>

          </thead>

          <tbody>

            <tr class="table-success">

              <td>John</td>

              <td>Doe</td>

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

            </tr>

            <tr class="table-danger">

              <td>Mary</td>

              <td>Moe</td>

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

            </tr>

            <tr class="table-warning">

              <td>July</td>

              <td>Dooley</td>

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

            </tr>

          </tbody>

        </table>

      </div>

</body>

</html>

1 Add zebra-stripes to the table. **[table-striped]**

**Output:**

Background pattern

Description automatically generated with medium confidence

2 Add borders on all sides of the table and cells**. [table-bordered]**

**Output:**

Table

Description automatically generated with low confidence

3 Enable a hover state on table rows**. [ table-hover]**

**Output:**

Graphical user interface, application

Description automatically generated with medium confidence

4 Make the table more compact by cutting cell padding in half**. [ table-sm]**

**Output:**

**Graphical user interface, text, application

Description automatically generated**

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.

**Output:**

Graphical user interface, application

Description automatically generated

1. Bootstrapping with Buttons.

Code: <!DOCTYPE html>

<head>

  <meta charset="UTF-8">

  <title>Buttons</title>

  <meta name="viewport" content="width=device-width, initial-scale=1">

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

    <script src="js/bootstrap.min.js" ></script>

</head>

<body>

  <div class="container mt-5">

        <button type="button" class="btn btn-danger">Red</button>

        <br><br>

        <button type="button" class="btn btn-primary btn-lg">Large button</button>

        <button type="button" class="btn btn-primary btn-md">Medium</button>

        <button type="button" class="btn btn-primary btn-sm">Small</button>

        <br><br>

        <div class="d-grid">

            <button type="button" class="btn btn-primary btn-block">Full-Width Button</button>

          </div>

        <br><br>

        <button type="button" class="btn btn-primary active">Active Primary</button>

        <button type="button" class="btn btn-primary" disabled>Disabled Primary</button>

    </div>

</body>

</html>

**Output:**

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

Icon

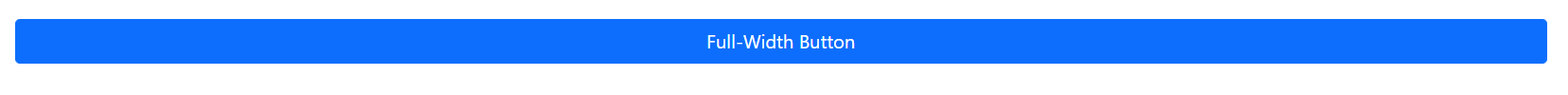
Description automatically generated with medium confidence

- Change the size of the buttons in the following order: large, medium and small.

A picture containing icon

Description automatically generated

- Make the button span the entire width of the parent element.



- Use a Bootstrap class to disable the button.

Graphical user interface, text

Description automatically generated

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>

Graphical user interface, text, application, email, Teams

Description automatically generated

**Code:**

<!DOCTYPE html>

<head>

  <meta charset="UTF-8">

  <title>Exercise #1: Simple form</title>

  <meta name="viewport" content="width=device-width, initial-scale=1">

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

    <script src="js/bootstrap.min.js" ></script>

</head>

<body>

  <div class="container col-4 mt-5">

    <form action="#">

      <div class="mb-3 mt-3" >

          <label for="first\_name" class="form-label fw-bold">First name:</label>

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

      </div>

      <div class="mb-3">

          <label for="last\_name" class="form-label fw-bold">Last name:</label>

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

      </div>

      <div class="mb-3">

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

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

      </div>

      <div class="mb-3">

          <label for="birth\_date" class="form-label fw-bold">Date of birth:</label>

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

      </div>

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

    </form>

  </div>

</body>

</html>

**Output:**

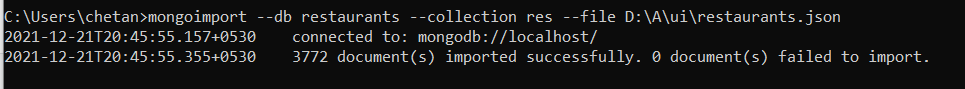
Graphical user interface, text, application, email

Description automatically generated

# Tutorial-02

Import restaurant.json file using below command.

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



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

Text

Description automatically generated

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

Text

Description automatically generated

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.

Text

Description automatically generated

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.

Text

Description automatically generated

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

Text

Description automatically generated

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

Text

Description automatically generated

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

Text

Description automatically generated

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

Text

Description automatically generated

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

Text

Description automatically generated

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

Text

Description automatically generated

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.

A picture containing text, screenshot, screen

Description automatically generated

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.

A picture containing text, screenshot, screen

Description automatically generated

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.

Text

Description automatically generated

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.

Text

Description automatically generated

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.

A computer screen capture

Description automatically generated with medium confidence

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.

Text

Description automatically generated

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

Text

Description automatically generated

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 Bronxor Brooklyn.

Text

Description automatically generated

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 Bronxor Brooklyn.

A screenshot of a computer

Description automatically generated with medium confidence

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'.

Text

Description automatically generated

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

Text

Description automatically generated

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

Text

Description automatically generated

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

A computer screen capture

Description automatically generated with low confidence

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

Text

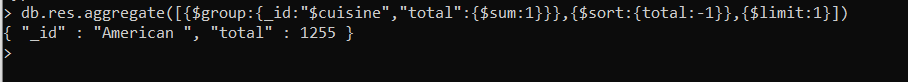
Description automatically generated

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

A computer screen capture

Description automatically generated with medium confidence

26. Which cuisine is highly offered among all restaurants?



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

Text

Description automatically generated

# Tutorial-03

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.

var events = require('events')

var em = new events.EventEmitter();

em.on('event', function(data){

    function cal\_age(date) {

        var d\_m = Date.now() - date.getTime();

        var a\_d = new Date(d\_m);

        return Math.abs(a\_d.getUTCFullYear() - 1970);

    }

    age = cal\_age(new Date(2019, 07, 20));

    if(age >= 18){

        console.log('Age is > 18')

    }

    else if(age < 18 && age > 0){

        console.log('Age is < 18')

    }

    else{

        console.log('Please Enter Valid Age...!!')

    }

});

em.emit('event');

Output:



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 http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>Get Method</title>

</head>

<body>

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

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

        Password: <input type="password" name="pwd"><br><br>

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

    </form>

</body>

</html>

From.js

var http = require('http');

var url = require('url')

var querystring = require('querystring');

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

    var query = url.parse(req.url).query;

    var Name = querystring.parse(query)['uname'];

    var Password = querystring.parse(query)['pwd'];

    console.log(Name)

    console.log(Password)

}).listen(3000);

Output:

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

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

From.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>Get Method</title>

</head>

<body>

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

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

        Password: <input type="password" name="pwd"><br><br>

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

    </form>

</body>

</html>

Form.js

var http = require('http');

var querystring = require('querystring');

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

    var data1 = "";

    req.on('data',function(chunk){

        console.log(chunk.toString());

        data1 += chunk;

    })

    req.on('end',function(){

        Name = querystring.parse(data1)["uname"];

        Password = querystring.parse(data1)["pwd"];

        console.log(Name)

        console.log(Password)

    })

}).listen(3000);

Output:

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

4. Create  mongodb for students and  nodejs that connects to mongodb using mongoose module.(install mongoose and nodemon packages).

var express = require('express')

var mongoose = require('mongoose')

mongoose.connect('mongodb://localhost:27017/students').then(()=>{

    console.log("DB Connected");

    var app = express()

    app.listen(3000,()=>{

        console.log("Server Started");

    })

})

# Tutorial-04

**Product.js**

var mongoose = require('mongoose');

var productSchema = mongoose.Schema({

    name:String,

    price:Number,

    likes:Number

})

module.exports = mongoose.model("products",productSchema)

**Index.js**

var mongoose = require('mongoose')

var express = require('express')

var route = require('./routes')

var bodyParser =require('body-parser')

mongoose.connect('mongodb+srv://ckochiyaniya:Test12345@cluster0.33siq.mongodb.net/products?retryWrites=true&w=majority').then(()=>{

    console.log('connected')

    app = express();

    app.use(bodyParser.urlencoded({extended:false}))

    app.use('/api',route)

    app.get('/', (req,res)=>{

        res.sendFile('index.html',{root:\_\_dirname})

    })

    app.listen((process.env.PORT||3000),()=>{

        console.log('server started')

    })

}).catch((e)=>{

    console.log(e.toString())

})

1. Create a NodeAPI to insert records into products database using mongoose library.

//to add the data

router.post("/products",async(req,res)=>{

    const product = new Product({

        name:req.body.name,

        price:req.body.price,

        likes:req.body.likes

    })

    await product.save((err,msg)=>{

        if(err){

            res.status(500).json({

                "error":err

            })

        }

        else{

            res.status(200).json({

                "My-message":msg

            })

        }

    })

})

1. Create a NodeAPI to get products from the database using mongoose library.

//to fetch data

router.get('/products',async(req,res)=>{

    const product = await Product.find()

    res.send(product)

})

1. Create a NodeAPI to update products from the database using mongoose library.

// api for updating data

router.patch('/products/:id',async (req,res)=>{

    const product = await Product.findOne({\_id:req.params.id})

    product.name = req.body.name

    product.price=req.body.price

    product.likes = req.body.likes

    await product.save((err,msg)=>{

        if(err){

            res.status(500).json({

                error:err

            })

        }

        else{

            res.status(200).json({

                msg:msg

            })

        }

    })

})

1. Create a NodeAPI to delete products from the database using mongoose library.

router.delete('/products/:id', async (req, res) => {   // delete by id

   try{

    const \_id = req.params.id;

    const deleteItem = await Product.findByIdAndDelete(\_id);

    res.send(deleteItem);

   }

    catch (e){

        res.send(e);

    }

})

# Tutorial-05

1)Create a database of your choice with 1 collections on the mongo cluster.

2)Create a NodeAPI to perform CRUD operations on the above created collections for the data stored on mlab cluster.

3)Host the same project on Heroku and need to present the same.

Menu.js

var mongoose = require('mongoose');

var foodmenuSchema = mongoose.Schema({

    name:String,

    category:String,

    price:Number

})

module.exports = mongoose.model("foodmenu",foodmenuSchema)

Index.js

var mongoose = require('mongoose')

var express = require('express')

var route = require('./routes')

var bodyParser =require('body-parser')

mongoose.connect('mongodb+srv://ckochiyaniya:Test12345@cluster0.33siq.mongodb.net/restaurant?retryWrites=true&w=majority').then(()=>{

    console.log('connected')

    app = express();

    app.use(bodyParser.urlencoded({extended:false}))

    app.use('/api',route)

    app.get('/', (req,res)=>{

        res.sendFile('index.html',{root:\_\_dirname})

    })

    app.listen((process.env.PORT||3000),()=>{

        console.log('server started')

    })

}).catch((e)=>{

    console.log(e.toString())

})

routes.js

var express = require('express');

var router = express.Router();

var Menu = require('./Models/Menu')

//to fetch data

router.get('/foodmenu',async(req,res)=>{

    const foodmenu = await Menu.find()

    res.send(foodmenu)

})

//to add the data

router.post("/foodmenu",async(req,res)=>{

    const foodmenu = new Menu({

        name:req.body.name,

        category:req.body.category,

        price:req.body.price

    })

    await foodmenu.save((err,msg)=>{

        if(err){

            res.status(500).json({

                "error":err

            })

        }

        else{

            res.status(200).json({

                "My-message":msg

            })

        }

    })

})

// api for updating data

router.patch('/foodmenu/:id',async (req,res)=>{

    const foodmenu = await Menu.findOne({\_id:req.params.id})

    foodmenu.name = req.body.name

    foodmenu.category = req.body.category

    foodmenu.price=req.body.price

    await foodmenu.save((err,msg)=>{

        if(err){

            res.status(500).json({

                error:err

            })

        }

        else{

            res.status(200).json({

                msg:msg

            })

        }

    })

})

router.delete('/foodmenu/:id', async (req, res) => {   // delete by id

    const \_id = req.params.id;

    const deleteItem = await Menu.findByIdAndDelete(\_id);

    res.send(deleteItem);

})

module.exports = router;

# Tutorial-06

1.nstall typescript

Ans:

npm i -g typescript

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

Ans:

var sum=(n:number)=>(n\*(n+1)/2)

console.log(sum(5))

Output:

Text

Description automatically generated

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

Ans:

//default parameter

var dpera =(a:number,b:number=1)=>a+b

console.log("\nfunction call dpera(1):")

console.log(dpera(1))

//optional parameter

var opera =(a:number,b?:number)=>{

    console.log("a:",a)

    if(b!=undefined)console.log("b:",b)

}

console.log("\n\nfunction call opera(1):")

opera(1)

console.log("function call opera(1,2):")

opera(1,2)

//rest parameter

var rpera =(...n:number[])=> {

    var sum=0;

    for(let i=0;i<n.length;i++){

        sum +=n[i]

    }

    return sum

}

console.log("\n\nfunction call rpera(1,2,3,4,5):")

console.log(rpera(1,2,3,4,5))

Output:

Text

Description automatically generated

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

Ans:

interface student{

    name:string

    city:string

    branch:string

    display:()=>string

}

var stud : student = <student>{};

stud.name="chetan"

stud.city="botad"

stud.branch="CE"

stud.display=():string=>{

    return stud.name

}

console.log(stud.display())

Output:

Text

Description automatically generated

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

interface Person{

    P\_name:string,

}

interface Student extends Person {

    Branch:string

}

var info:Student = {

    P\_name:"Chetan", Branch:"CE"

    }

    console.log(info.P\_name);

    console.log(info.Branch);

Output:

Text

Description automatically generated

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:



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(10, 49, 49)

obj.display()

obj.inc()

Output:

Graphical user interface, text

Description automatically generated

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 {

    code: number

    name: string

    designation: string

    basic\_pay: number

}

class emp implements Employee {

    code: number

    name: string

    designation: string

    basic\_pay: number

    constructor(code: number, name: string, designation: string, basic\_pay: number) {

        this.code = code

        this.name = name

        this.designation = designation

        this.basic\_pay = basic\_pay

    }

    Total\_pay(basic\_pay: number): void {

        this.basic\_pay = basic\_pay + basic\_pay \*

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

    }

    display(): string {

        return this.code + "\n" + this.name + "\n" + this.designation;

    }

}

var obj1 = new emp(1, "Chetan", "software developer", 100000)

console.log(obj1.display())

obj1.Total\_pay(1000000)

Output:

Text

Description automatically generated

# Tutorial-07

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:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

# Tutorial-08

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:

Graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated

