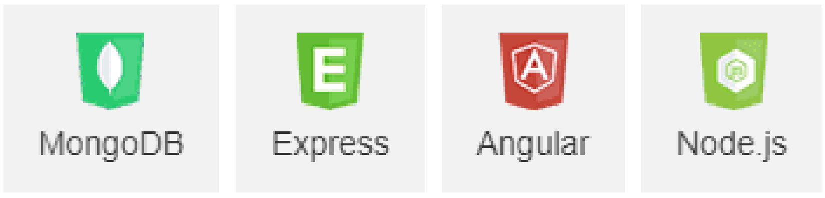
INTRODUCTION

MEAN is a user-friendly full-stack JavaScript framework ideal for building dynamic websites and applications. It is a free and open-source stack designed to supply developers with a quick and organized method for creating rapid prototypes of MEAN-based web applications. One of the main benefits of the MEAN stack is that a single language, JavaScript, runs on every level of the application, making it an efficient and modern approach to web development.



Node.js is a server-side JavaScript execution environment. It's a platform built on Google Chrome's V8 JavaScript runtime. It helps in building highly scalable and concurrent applications rapidly.

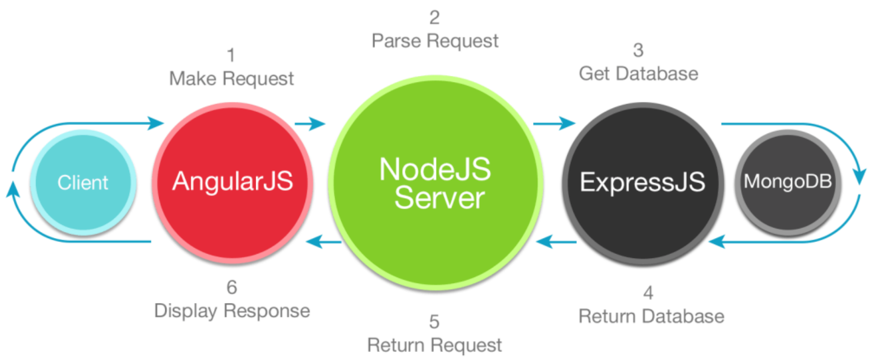
Express is lightweight framework used to build web applications in Node. It provides a number of robust features for building single and multi page web application. Express is inspired by the popular Ruby framework, Sinatra.

MongoDB is a schema less NoSQL database system. MongoDB saves data in binary JSON format which makes it easier to pass data between client and server.

AngularJS is a JavaScript framework developed by Google. It provides some awesome features like the two-way data binding. It's a complete solution for rapid and awesome front-end development.

<u>Why learn MEAN stack?</u>

It's hard to accomplish much on the web without JavaScript, which is the single language that runs the entire MEAN full stack and boasts one of the most active developer communities. Because every part of MEAN programming is written in one language, it allows unique server-side and client-side execution environments. Valued for its versatility in building fast, robust and maintainable production web applications, MEAN is in high demand with numerous startups and employers.



Features of MEAN

Employability
More and more employers are in need of engineers familiar with MEAN Stack and other JavaScript-based technologies.

Simple & Quick
Building websites and applications that revolve around one language, JavaScript, is relatively straightforward.

Adaptability
Due to the versatility of MEAN Stack’s common programming language, JavaScript, it is highly adaptable for a wide range of web applications.

Active Dev Community
MEAN Stack runs on JavaScript, the most common programming language in the world with one of the most active developer communities, making solutions to problems easily accessible.

Ex. No: 1 **Name:**

Date: **Reg no:**

CALCULATOR NODE

AIM:

To create a calculator program using node.js with relevant arithmetic operations.

PROCEDURE:

- Create an app.js that require another file called calculator.js. When we call node app.js should show the result in the console.
- Create a folder for operations and create every operation in separate folders which is needed in the main app.js file.
- It should contain the following files, app.js/sum.js/multiplication.js/subtraction.js/division.js.
- Then install the module moment to show the current time.

CODE:

DIVISION.JS

```
function division(a, b) {
  console.log("The division of " + a + " & " + b + " is : " + a / b);
}
module.exports = division;
```

MULTIPLICATION.JS

```
function multiplication(a, b) {
  console.log("The multiplication of " + a + " & " + b + " is : " + a * b);
}
module.exports = multiplication;
```

SUBTRACTION.JS

```
function subtraction(a, b) {
  console.log("The subtraction of " + a + " & " + b + " is : " + (a - b));
}
module.exports = subtraction;
```

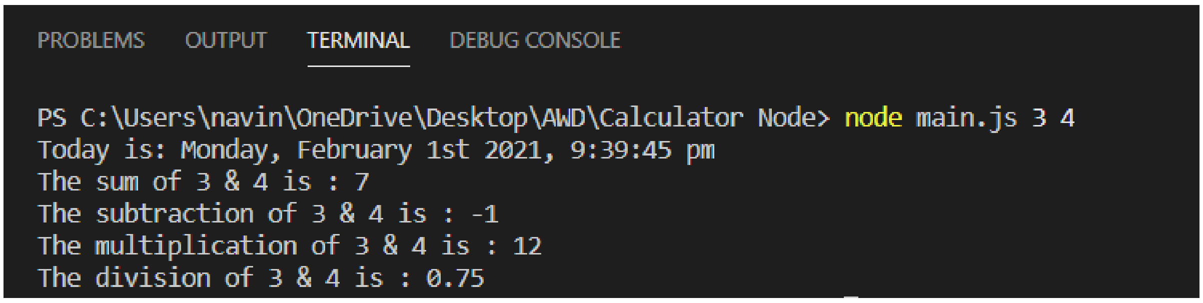
SUM.JS

```
function sum(a, b) {
  console.log("The sum of " + a + " & " + b + " is : " + (a + b));
}
module.exports = sum;
```

MAIN.JS

```
var moment = require("moment");
var sum = require("./operations/sum");
var subs = require("./operations/subtraction");
var mult = require("./operations/multiplication");
var div = require("./operations/division");
var firstOperand = +process.argv[2];
var secondOperand = +process.argv[3];
console.log("Today is: " + moment().format("dddd, MMMM Do YYYY, h:mm:ss a"));
sum(firstOperand, secondOperand);
subs(firstOperand, secondOperand);
mult(firstOperand, secondOperand);
div(firstOperand, secondOperand);
```

OUTPUT:



```
PS C:\Users\navin\OneDrive\Desktop\AWD\Calculator Node> node main.js 3 4
Today is: Monday, February 1st 2021, 9:39:45 pm
The sum of 3 & 4 is : 7
The subtraction of 3 & 4 is : -1
The multiplication of 3 & 4 is : 12
The division of 3 & 4 is : 0.75
```

RESULT:

Thus the program has been completed and executed successfully.

Ex. No: 2 **Name:**

Date: **Reg no:**

CHAIN MIDDLEWARE TO CREATE A TIME SERVER

AIM:

A chain middleware function and the final handler to get the current time with date.

PROCEDURE:

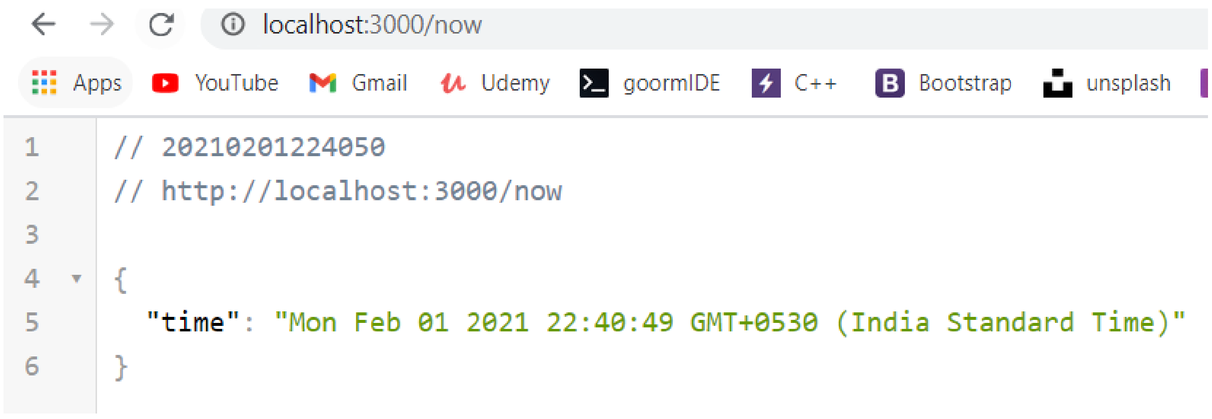
- In the middleware function, we should add current time in req.time key.
- We can use new Date().toString()
- In the handler,respond with JSON object , the structure should be {time: req.time}
- In the route app.get('/now',…) at the url to display the output.

CODE:

MAIN.JS

```
var express = require("express");
var app = express();
// Chaining middleware. A Time server
app.get(
  "/now",
  (req, res, next) => {
    req.time = new Date().toString();
    next();
  },
  (req, res) => {
    res.json({ time: req.time });
  }
);

app.listen(process.env.PORT || 3000);
```

OUTPUT:



```
// 20210201224050
// http://localhost:3000/now

{
  "time": "Mon Feb 01 2021 22:40:49 GMT+0530 (India Standard Time)"
}
```

RESULT:

Thus the program the program has been completed and executed successfully**.**

Ex. No: 3 **Name:**

Date: **Reg no:**

GET ROUTE PARAMETER INPUT FROM THE CLIENT

AIM:

Build an echo server, respond with JSON object to get the input from the user.

PROCEDURE:

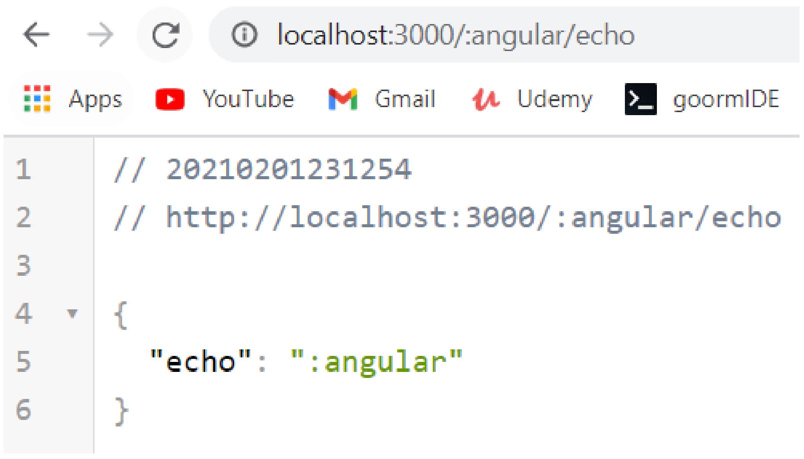
- Create an echo server, mounted at the route GET/:word/echo.
- The structure of the JSON object{echo:word}
- Then test the route in browser's address bar as your-app-path/angular/echo.

CODE:

MAIN.JS

```
var express = require("express");
var app = express();
// Get input from client - Route parameters
app.get("/:word/echo", (req, res) => {
  res.json({ echo: req.params.word });
});
app.listen(process.env.PORT || 3000);
```

OUTPUT:



```
// 20210201231254
// http://localhost:3000/:angular/echo

{
  "echo": ":angular"
}
```

RESULT:

Thus the program has been completed and executed successfully.

Ex. No: 4 **Name:**

Date: **Reg no:**

GET QUERY PARAMETER INPUT FROM THE CLIENT

AIM:

Build an API endpoint, respond with JSON object to get the input from the user.

PROCEDURE:

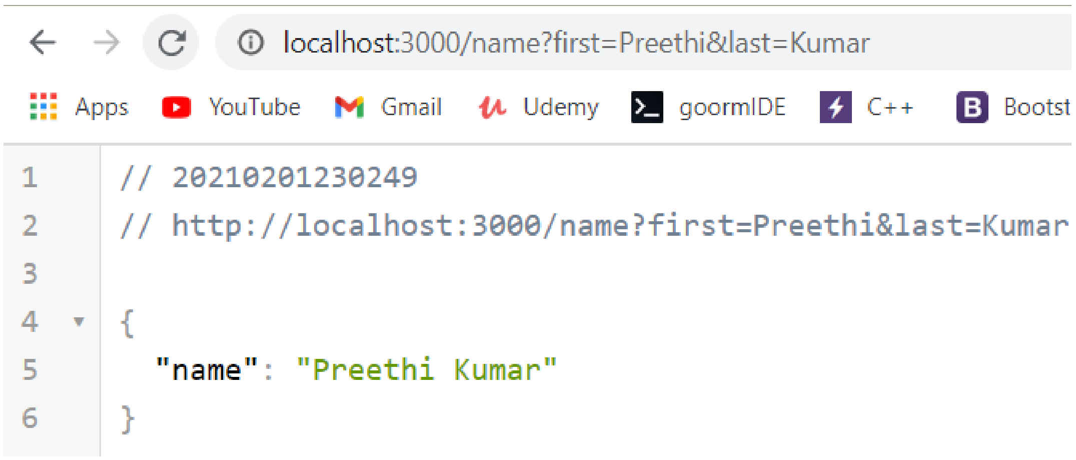
- Create an echo server, mounted at GET /name.
- The structure should be {name:'firstnamelastname'}.
- The first and last name should be encoded in the query string as (?first=firstname&last=lastname).

CODE:

MAIN.JS

```
var express = require("express");
var app = express();

// Get input from client - Query parameters
// /name?first=<firstname>&last=<lastname>
app.route("/name").get((req, res) => {
  res.json({ name: `${req.query.first} ${req.query.last}` });
});

app.listen(process.env.PORT || 3000);
```

OUTPUT:



```
// 20210201230249
// http://localhost:3000/name?first=Preethi&last=Kumar

{
  "name": "Preethi Kumar"
}
```

RESULT:

Thus the program has been completed and executed successfully.

Ex. No: 5 **Name:**

Date: **Reg no:**

SHOPPING LIST API

AIM:

Build a simple application to store a shopping list.

PROCEDURE:

- Create a simple application where we will store a shopping list.
- Create a separate folder as item.js and add operations which is needed in main.js file.
- Application should have following routes,
- GET/items, POST/items, GET/items/id, PATCH/items/id, DELETE/items/id.

CODE:

ITEM.JS

```
class Item {
  constructor(name, price) {
    this.name = name;
    this.price = price;
    this.id = Item.id;
    Item.list.push(this);
    Item.id++;
  }
  static update(id, data) {
    let foundItem = Item.list.find((v) => v.id === id);
    foundItem.name = data.name;
    foundItem.price = data.price;
    return foundItem;
  }
  static find(id) {
    return Item.list.find((v) => v.id === id);
  }
  static remove(id) {
    let foundIdx = Item.list.findIndex((v) => v.id === id);
    Item.list.splice(foundIdx, 1);
  }
}
Item.id = 1;
Item.list = [];
module.exports = Item;
```

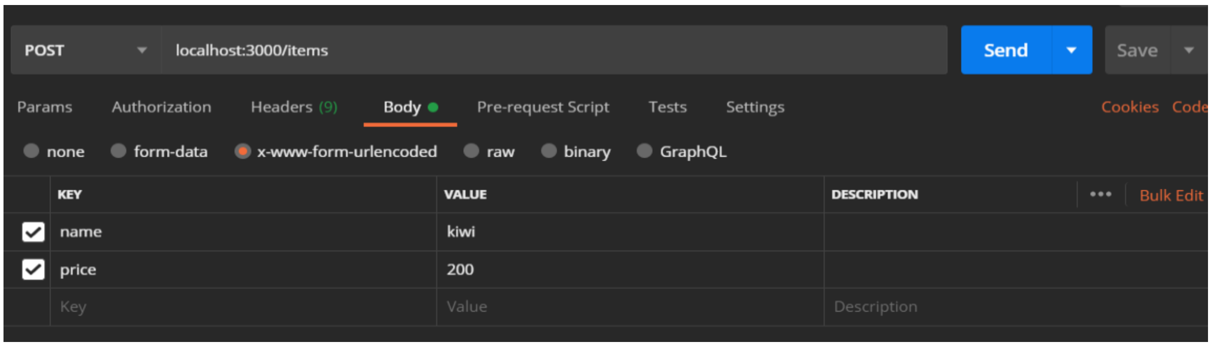
MAIN.JS

```
const express = require("express");
const app = express();
const morgan = require("morgan");
const bodyParser = require("body-parser");

const Item = require("./item");

app.use(morgan("tiny"));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.get("/items", (req, res) => {
  return res.json(Item.list);
});

app.post("/items", (req, res) => {
  let newItem = new Item(req.body.name, req.body.price);
  return res.json(newItem);
});

app.get("/items/:id", (req, res) => {
  let foundItem = Item.find(+req.params.id);
  return res.json(foundItem);
});

app.patch("/items/:id", (req, res) => {
  let foundItem = Item.update(+req.params.id, req.body);
  return res.json(foundItem);
});

app.delete("/items/:id", (req, res) => {
  Item.remove(+req.params.id);
  return res.json("Removed");
});

// catch 404 and forward to error handler
app.use((req, res, next) => {
  var err = new Error("Not Found");
  err.status = 404;
  next(err);
});
```

```
// error handlers

// development error handler
// will print stacktrace
if (app.get("env") === "development") {
  app.use((err, req, res, next) => {
    res.status(err.status || 500);
    res.send({
      message: err.message,
      error: err,
    });
  });
}
// production error handler
// no stacktraces leaked to user
app.use((err, req, res, next) => {
  res.status(err.status || 500);
  res.send({
    message: err.message,
    error: {},
  });
});

app.listen(process.env.PORT || 3000, () => {
  console.log("Server is listening on port 3000");
});
```

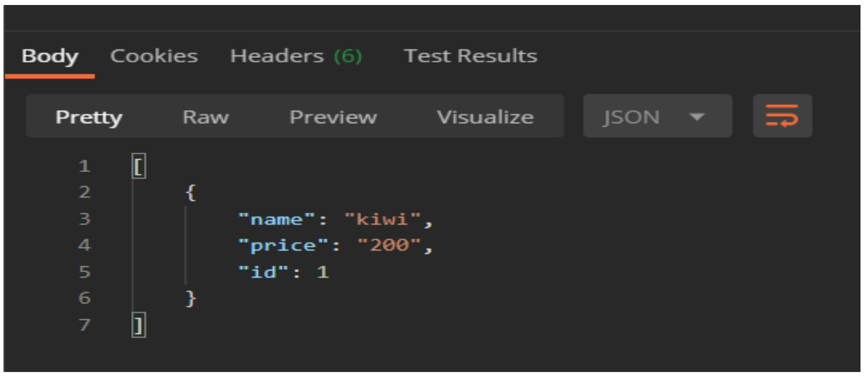
OUTPUT:

POST:



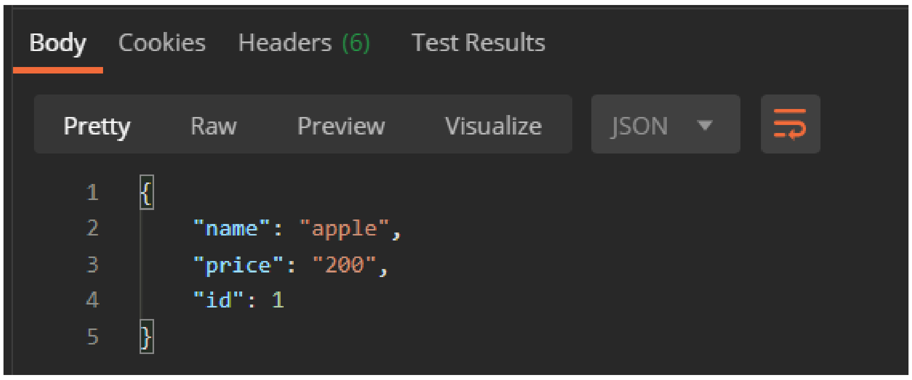
	KEY	VALUE	DESCRIPTION
✓	name	kiwi	
✓	price	200	
	Key	Value	Description

GET:



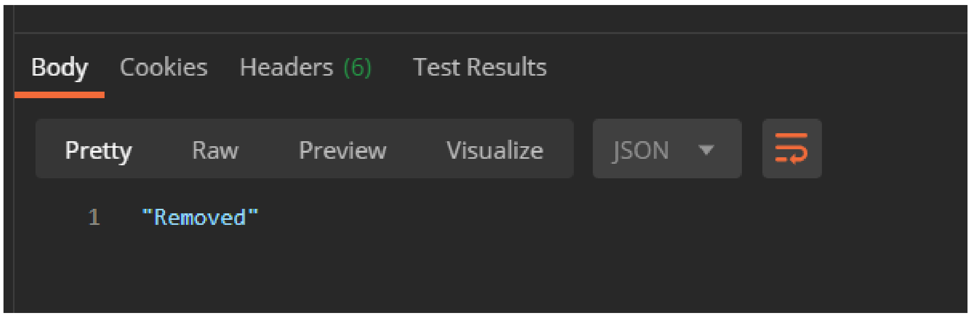
```
[
    {
        "name": "kiwi",
        "price": "200",
        "id": 1
    }
]
```

PATCH:



```
{
    "name": "apple",
    "price": "200",
    "id": 1
}
```

DELETE:



```
"Removed"
```

RESULT:

Thus the program has been completed and executed successfully.

Ex. No: 6 **Name:**

Date: **Reg no:**

Create a Model – Using model.find()

Aim:

To create a schema personSchema and find model using model.find()

Procedure:

- Create a person schema called personSchema.
- Use Mongoose basic schema types.
- Create a model Person from person Schema. Use model.find() to search for the model.

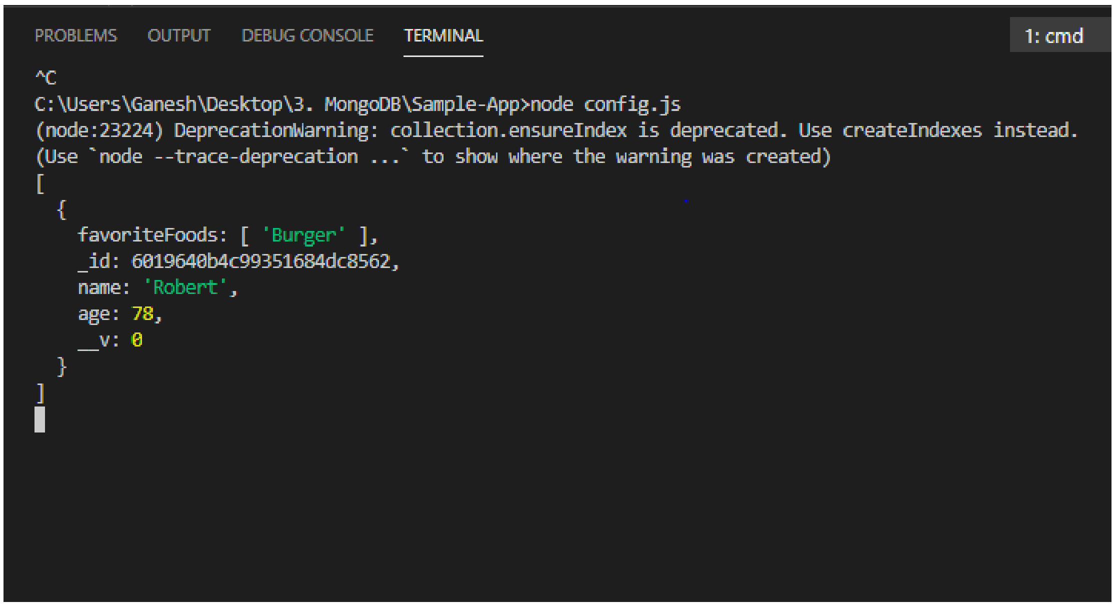
Source Code:

config.js:

```
// Creating schema
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/Sample", {
useNewUrlParser: true,
useUnifiedTopology: true,
});

var Schema = mongoose.Schema;
varPersonSchema = new Schema({
name: { type: String, required: true },
age: Number,
favoriteFoods: [{ type: String, unique: true }],
});
var Person = mongoose.model("Person", PersonSchema);
vararrayOfPeople = [
{ name: "Frankie", age: 74, favoriteFoods: ["Taco"] },
{ name: "Sol", age: 76, favoriteFoods: ["Roast chicken", "Pizza"] },
{ name: "Robert", age: 78, favoriteFoods: ["Burger"] },
];
```

```
// Defining and Using model.find()
varfindPeopleByName = function (personName) {
Person.find({ name: personName }, function (err, personFound) {
if (err) return console.log(err);
done(personFound);
  });
};

findPeopleByName("Robert");
```

Output:



```
^C
C:\Users\Ganesh\Desktop\3. MongoDB\Sample-App>node config.js
(node:23224) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
[
  {
    favoriteFoods: [ 'Burger' ],
    _id: 6019640b4c99351684dc8562,
    name: 'Robert',
    age: 78,
    __v: 0
  }
]
```

Result:

Hence, We have created a Person model in personSchema and used model.find() to find the person “Robert”.

Ex. No: 7 **Name:**

Date: **Reg no:**

Create a Model – using model.fineOne()

Aim:

To create a schema personSchema and find model using model.findOne()

Procedure:

- Create a person schema called personSchema.
- Use Mongoose basic schema types.
- Create a model Person from person Schema. Use model.findOne() to return the Single Matching document from Person model.

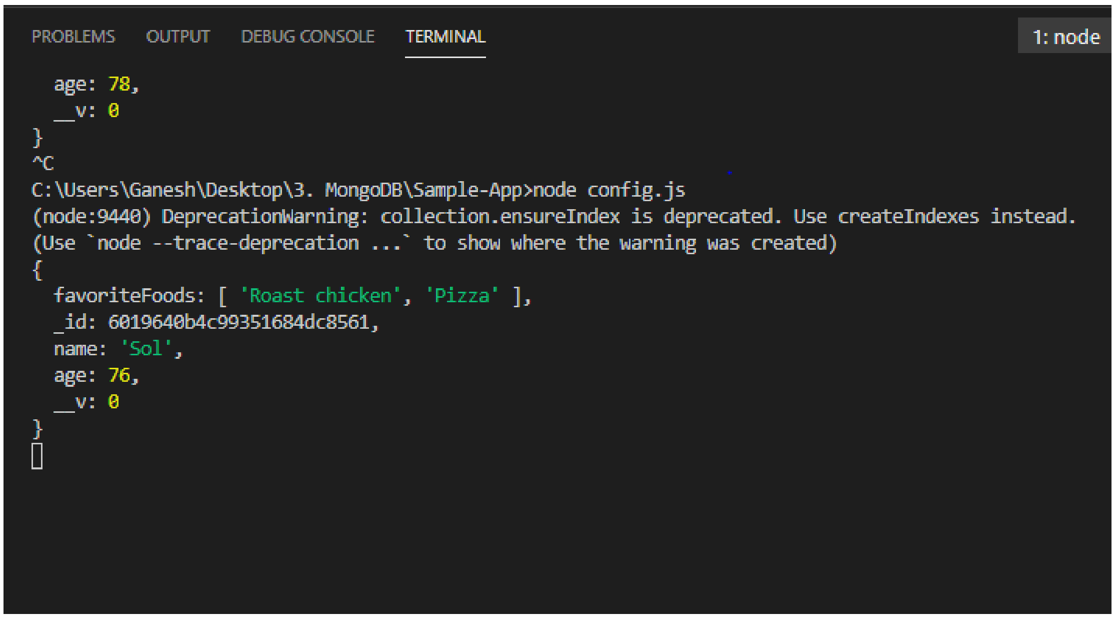
Source Code:

config.js:

```
// Creating schema
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/Sample", {
useNewUrlParser: true,
useUnifiedTopology: true,
});

var Schema = mongoose.Schema;
varPersonSchema = new Schema({
name: { type: String, required: true },
age: Number,
favoriteFoods: [{ type: String, unique: true }],
});

var Person = mongoose.model("Person", PersonSchema);
```

```
vararrayOfPeople = [
{ name: "Frankie", age: 74, favoriteFoods: ["Taco"] },
{ name: "Sol", age: 76, favoriteFoods: ["Roast chicken", "Pizza"] },
{ name: "Robert", age: 78, favoriteFoods: ["Burger"] },
];

// Defining and using model.findOne()

varfindOneByFood = function (food) {
Person.findOne({ favoriteFoods: food }, (err, data) =>
err ?done(err) : done(data)
  );
};

findOneByFood("Pizza");
```

Output:



```
  age: 78,
  __v: 0
}
^C
C:\Users\Ganesh\Desktop\3. MongoDB\Sample-App>node config.js
(node:9440) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
{
  favoriteFoods: [ 'Roast chicken', 'Pizza' ],
  _id: 6019640b4c99351684dc8561,
  name: 'Sol',
  age: 76,
  __v: 0
}
```

Result:

Hence, We have created a Person model in personSchema and used model.findOne() to find the person whose favorite food is "Pizza".

Ex. No: 8 **Name:**

Date: **Reg no:**

Create a Model – Find , Edit , Save

Aim:

To create a schema personSchema and find model using findEditThenSave().

Procedure:

- Create a person schema called personSchema.
- Use Mongoose basic schema types.
- Create a model Person from person Schema. Use findEditThenSave function to find a person by id with parameter personId as key and add “Hamburger” to that person’s favourite foods list and then callback save() to save and update the model.

Source Code:

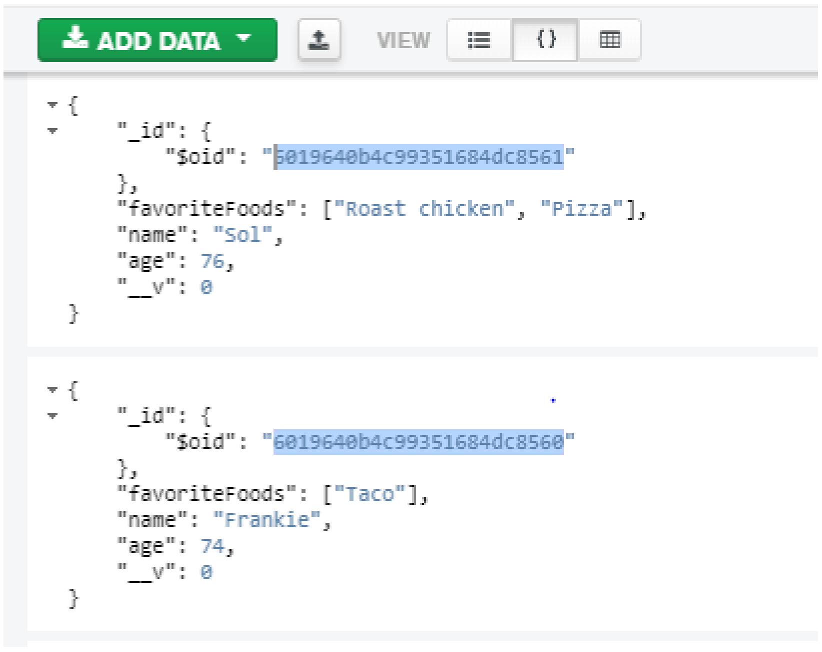
config.js:

```
// Creating schema
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/Sample", {
useNewUrlParser: true,
useUnifiedTopology: true,
});

var Schema = mongoose.Schema;
varPersonSchema = new Schema({
name: { type: String, required: true },
age: Number,
favoriteFoods: [{ type: String, unique: true }],
});
```

```
var Person = mongoose.model("Person", PersonSchema);

vararrayOfPeople = [
{ name: "Frankie", age: 74, favoriteFoods: ["Taco"] },
{ name: "Sol", age: 76, favoriteFoods: ["Roast chicken", "Pizza"] },
{ name: "Robert", age: 78, favoriteFoods: ["Burger"] },
];

// Defining and calling findEditThenSave function
varfindEditThenSave = function (personId) {
varfoodToAdd = "hamburger";
Person.findById({ _id: personId }, function (err, data) {
if (err) {
return done(err);
    } else {
data.favoriteFoods.push(foodToAdd);
data.save((err, data) => (err ? done(err) : done(data)));
    }
  });
};

findEditThenSave("6019640b4c99351684dc8561");
```

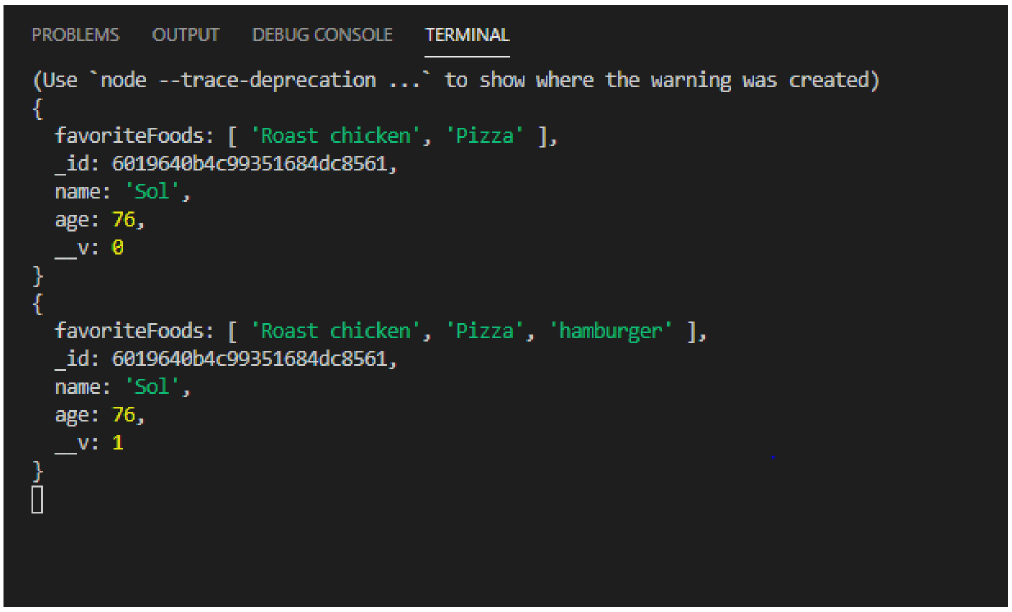
Output:

//Before using findEditThenSave



```
{
    "_id": {
        "$oid": "6019640b4c99351684dc8561"
    },
    "favoriteFoods": ["Roast chicken", "Pizza"],
    "name": "Sol",
    "age": 76,
    "__v": 0
}

{
    "_id": {
        "$oid": "6019640b4c99351684dc8560"
    },
    "favoriteFoods": ["Taco"],
    "name": "Frankie",
    "age": 74,
    "__v": 0
}
```

//After running findEditThenSave



```
(Use `node --trace-deprecation ...` to show where the warning was created)
{
  favoriteFoods: [ 'Roast chicken', 'Pizza' ],
  _id: 6019640b4c99351684dc8561,
  name: 'Sol',
  age: 76,
  __v: 0
}
{
  favoriteFoods: [ 'Roast chicken', 'Pizza', 'hamburger' ],
  _id: 6019640b4c99351684dc8561,
  name: 'Sol',
  age: 76,
  __v: 1
}
```

Result:

Hence, We have created a Person model in personSchema and used findEditThenSave() to find the person using id and update their favorite foods list and save the model.

Ex. No: 9 **Name:**

Date: **Reg no:**

Create a Model – Delete

Aim:

To create a schema personSchema and find model using model.Remove()

Procedure:

- Create a person schema called personSchema.
- Use Mongoose basic schema types.
- Create a model Person from person Schema. Use model.Remove() to search for the model.

Source Code:

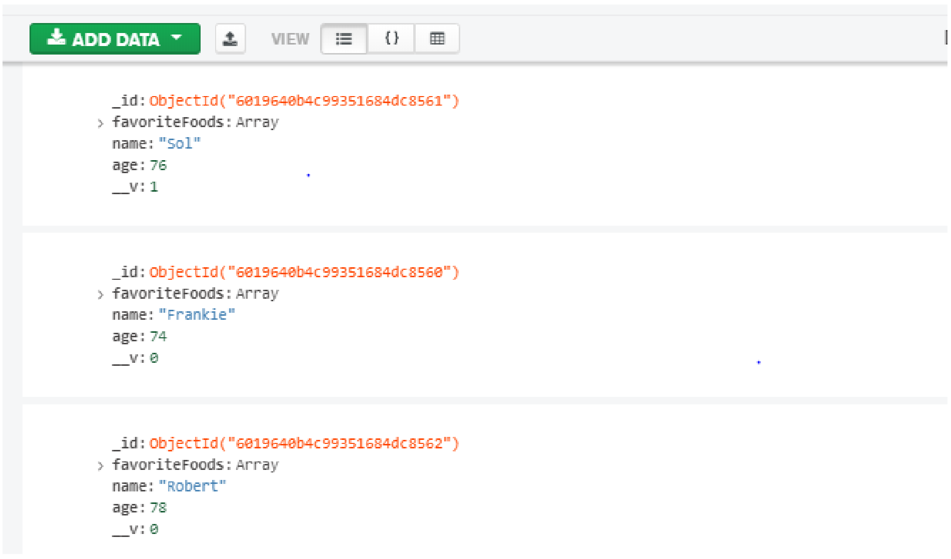
config.js:

```
// Creating schema
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/Sample", {
useNewUrlParser: true,
useUnifiedTopology: true,
});

var Schema = mongoose.Schema;
varPersonSchema = new Schema({
name: { type: String, required: true },
age: Number,
favoriteFoods: [{ type: String, unique: true }],
});
```

```
var Person = mongoose.model("Person", PersonSchema);

vararrayOfPeople = [
{ name: "Frankie", age: 74, favoriteFoods: ["Taco"] },
{ name: "Sol", age: 76, favoriteFoods: ["Roast chicken", "Pizza"] },
{ name: "Robert", age: 78, favoriteFoods: ["Burger"] },
];

varremoveManyPeople = function () {
varnameToRemove = "Robert";
Person.remove({ name: nameToRemove }, function (error, data) {
error ?done(error) : done(data);
  });
};
removeManyPeople("Robert");
```

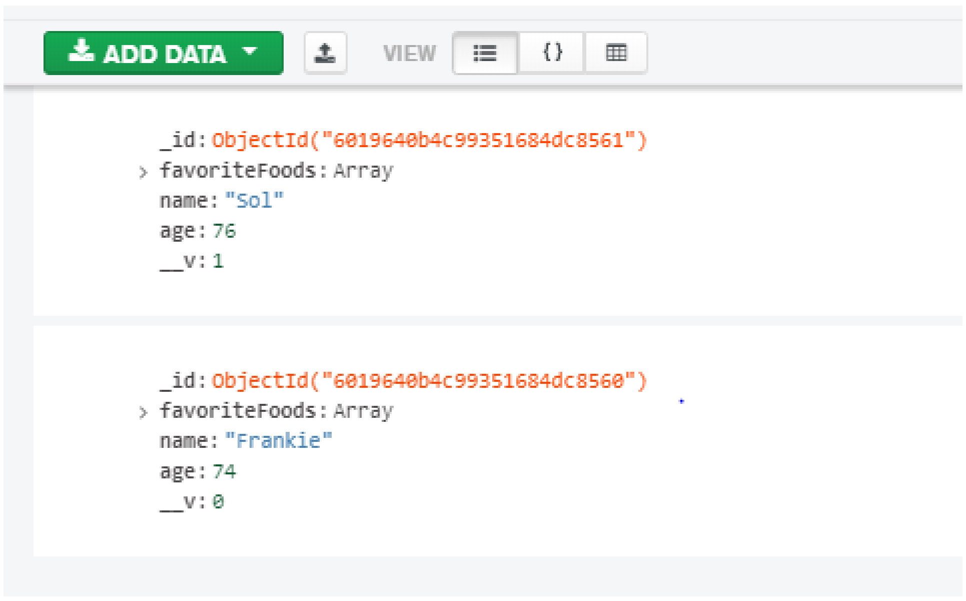
Output:

// Before running model.Remove()



```
_id: ObjectId("6019640b4c99351684dc8561")
> favoriteFoods: Array
name: "Sol"
age: 76
__v: 1

_id: ObjectId("6019640b4c99351684dc8560")
> favoriteFoods: Array
name: "Frankie"
age: 74
__v: 0

_id: ObjectId("6019640b4c99351684dc8562")
> favoriteFoods: Array
name: "Robert"
age: 78
__v: 0
```

//After running model.Remove()



```
_id: ObjectId("6019640b4c99351684dc8561")
> favoriteFoods: Array
name: "Sol"
age: 76
__v: 1

_id: ObjectId("6019640b4c99351684dc8560")
> favoriteFoods: Array
name: "Frankie"
age: 74
__v: 0
```

Result:

Hence, We have created a Person model in personSchema and used model.Remove() to find the person using name and remove the person from the model and save the model.

Ex. No: 10 **Name:**

Date: **Reg no:**

Queries

Aim:

To write a MongoDB query to display all the documents in restaurants collection.

Procedure:

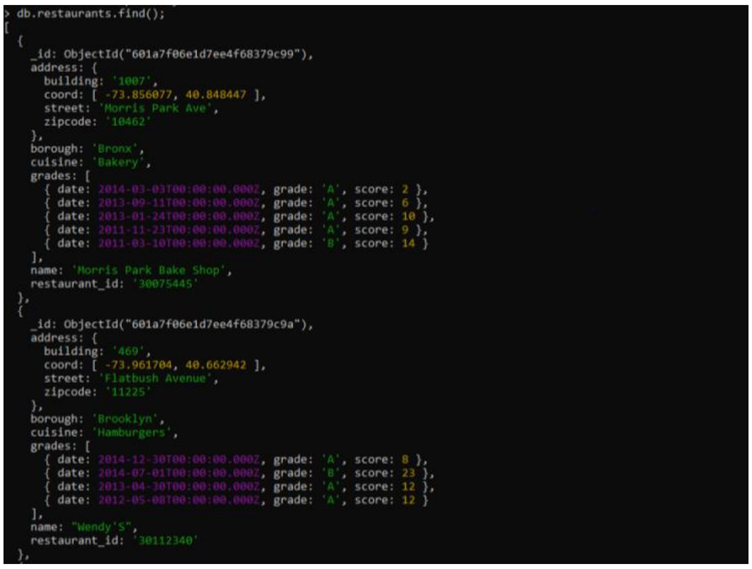
1. Create Hospitality database and collection namely restaurants.
2. Open Mongo shell.
3. Use MongoDB queries to fetch from collections.

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

Query:

db.restaurants.find(); //restaurants is the collection

Output:



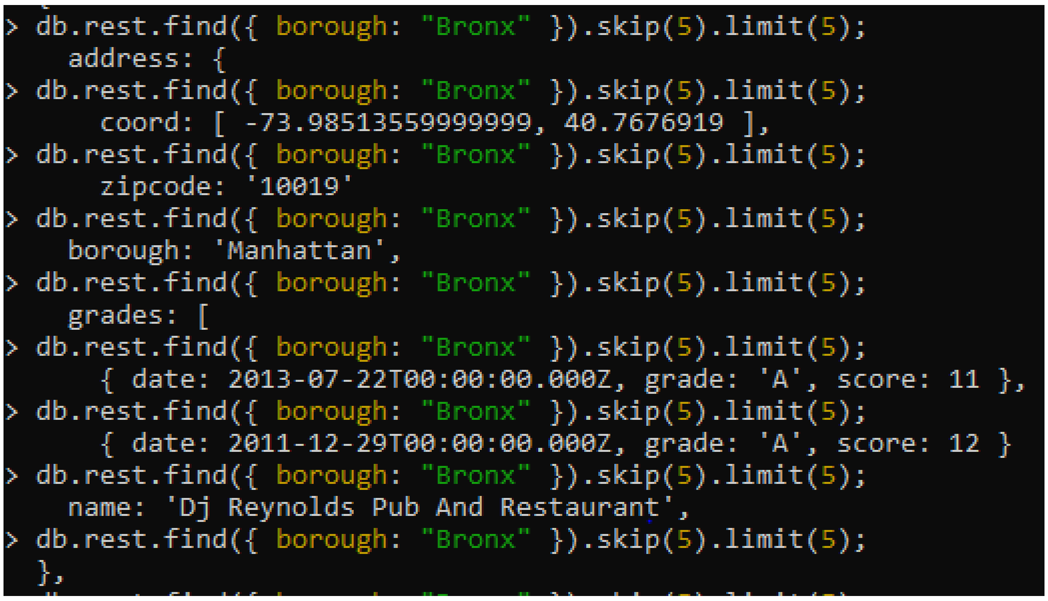
```
> db.restaurants.find();
[
  {
    _id: ObjectId("601a7f06e1d7ee4f68379c99"),
    address: {
      building: '1007',
      coord: [ -73.856077, 40.848447 ],
      street: 'Morris Park Ave',
      zipcode: '10462'
    },
    borough: 'Bronx',
    cuisine: 'Bakery',
    grades: [
      { date: 2014-03-03T00:00:00.000Z, grade: 'A', score: 2 },
      { date: 2013-09-11T00:00:00.000Z, grade: 'A', score: 6 },
      { date: 2013-01-24T00:00:00.000Z, grade: 'A', score: 10 },
      { date: 2011-11-23T00:00:00.000Z, grade: 'A', score: 9 },
      { date: 2011-03-10T00:00:00.000Z, grade: 'B', score: 14 }
    ],
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  },
  {
    _id: ObjectId("601a7f06e1d7ee4f68379c9a"),
    address: {
      building: '469',
      coord: [ -73.961704, 40.662942 ],
      street: 'Flatbush Avenue',
      zipcode: '11225'
    },
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
    grades: [
      { date: 2014-12-30T00:00:00.000Z, grade: 'A', score: 8 },
      { date: 2014-07-01T00:00:00.000Z, grade: 'B', score: 23 },
      { date: 2013-04-30T00:00:00.000Z, grade: 'A', score: 12 },
      { date: 2012-05-08T00:00:00.000Z, grade: 'A', score: 12 }
    ],
    name: "Wendy'S",
    restaurant_id: '30112340'
  },
```

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

Query:

db.rest.find({ borough: "Bronx" }).skip(5).limit(5);

Output:



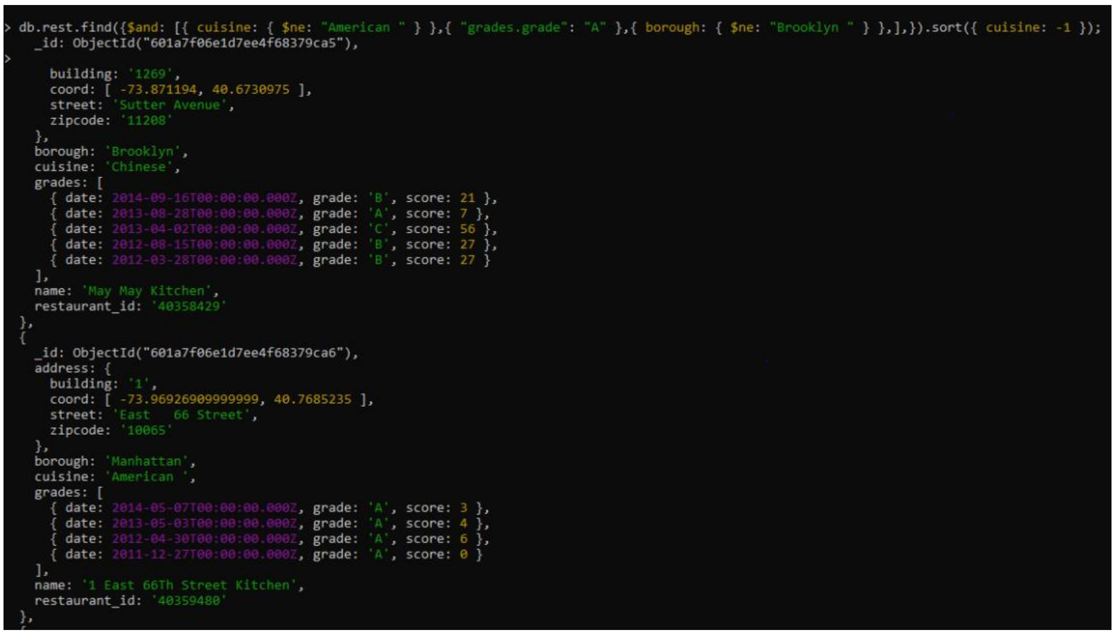
```
> db.rest.find({ borough: "Bronx" }).skip(5).limit(5);
    address: {
> db.rest.find({ borough: "Bronx" }).skip(5).limit(5);
      coord: [ -73.98513559999999, 40.7676919 ],
> db.rest.find({ borough: "Bronx" }).skip(5).limit(5);
      zipcode: '10019'
> db.rest.find({ borough: "Bronx" }).skip(5).limit(5);
    borough: 'Manhattan',
> db.rest.find({ borough: "Bronx" }).skip(5).limit(5);
    grades: [
> db.rest.find({ borough: "Bronx" }).skip(5).limit(5);
      { date: 2013-07-22T00:00:00.000Z, grade: 'A', score: 11 },
> db.rest.find({ borough: "Bronx" }).skip(5).limit(5);
      { date: 2011-12-29T00:00:00.000Z, grade: 'A', score: 12 }
> db.rest.find({ borough: "Bronx" }).skip(5).limit(5);
    name: 'Dj Reynolds Pub And Restaurant',
> db.rest.find({ borough: "Bronx" }).skip(5).limit(5);
  },
```

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

Query:

```
db.rest
  .find({
    $and: [
{ cuisine: { $ne: "American " } },
{ "grades.grade": "A" },
{ borough: { $ne: "Brooklyn " } },
    ],
  })
  .sort({ cuisine: -1 });
```

Output:



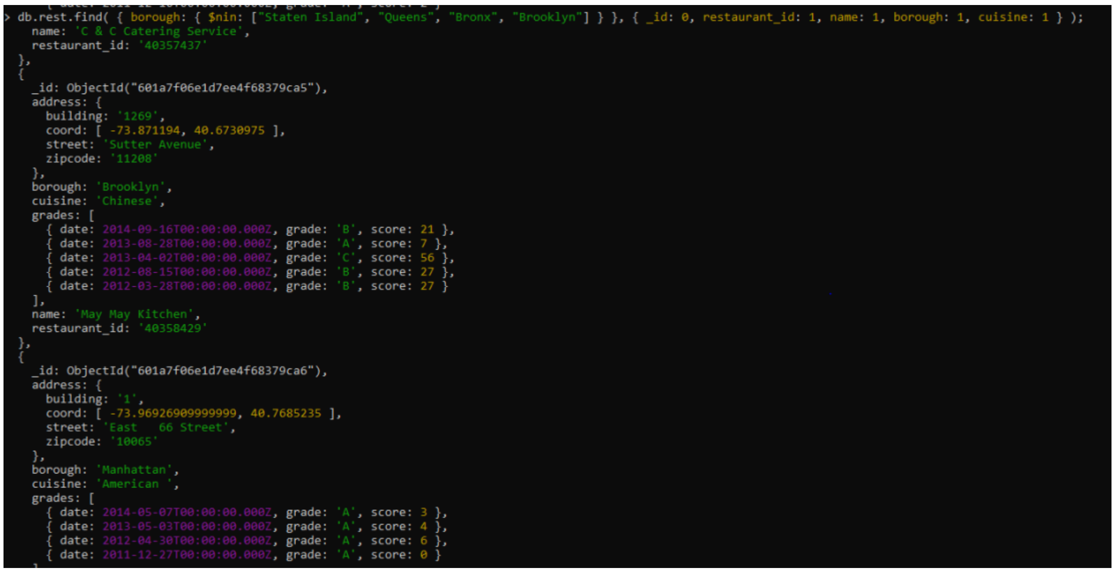
```
> db.rest.find({$and: [{ cuisine: { $ne: "American " } },{ "grades.grade": "A" },{ borough: { $ne: "Brooklyn " } },],}).sort({ cuisine: -1 });
    _id: ObjectId("601a7f06e1d7ee4f68379ca5"),
>
      building: '1269',
      coord: [ -73.871194, 40.6730975 ],
      street: 'Sutter Avenue',
      zipcode: '11208'
    },
    borough: 'Brooklyn',
    cuisine: 'Chinese',
    grades: [
      { date: 2014-09-16T00:00:00.000Z, grade: 'B', score: 21 },
      { date: 2013-08-28T00:00:00.000Z, grade: 'A', score: 7 },
      { date: 2013-04-02T00:00:00.000Z, grade: 'C', score: 56 },
      { date: 2012-08-15T00:00:00.000Z, grade: 'B', score: 27 },
      { date: 2012-03-28T00:00:00.000Z, grade: 'B', score: 27 }
    ],
    name: 'May May Kitchen',
    restaurant_id: '40358429'
  },
  {
    _id: ObjectId("601a7f06e1d7ee4f68379ca6"),
    address: {
      building: '1',
      coord: [ -73.96926909999999, 40.7685235 ],
      street: 'East   66 Street',
      zipcode: '10065'
    },
    borough: 'Manhattan',
    cuisine: 'American ',
    grades: [
      { date: 2014-05-07T00:00:00.000Z, grade: 'A', score: 3 },
      { date: 2013-05-03T00:00:00.000Z, grade: 'A', score: 4 },
      { date: 2012-04-30T00:00:00.000Z, grade: 'A', score: 6 },
      { date: 2011-12-27T00:00:00.000Z, grade: 'A', score: 0 }
    ],
    name: '1 East 66Th Street Kitchen',
    restaurant_id: '40359480'
  },
```

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

Query:

```
db.rest.find(
{ borough: { $nin: ["Staten Island", "Queens", "Bronx", "Brooklyn"] } },
{ _id: 0, restaurant_id: 1, name: 1, borough: 1, cuisine: 1 }
);
```

Output:



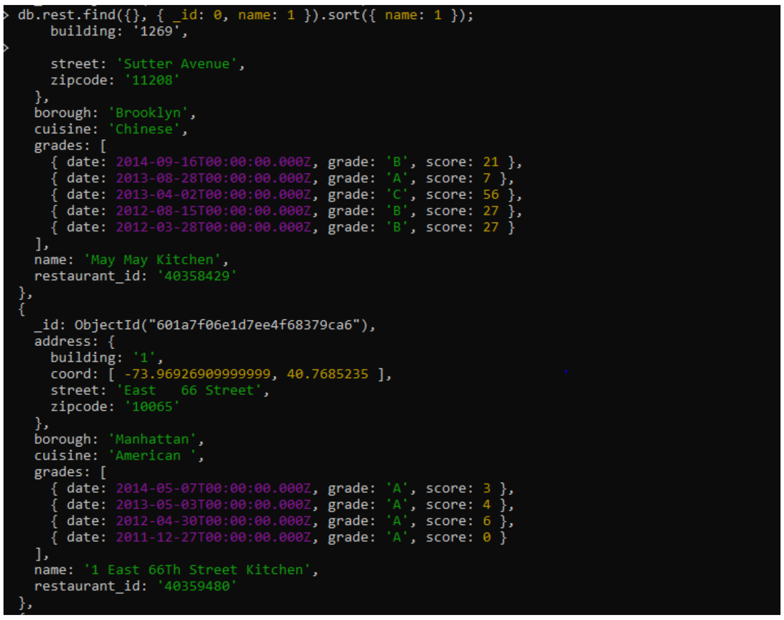
```
> db.rest.find( { borough: { $nin: ["Staten Island", "Queens", "Bronx", "Brooklyn"] } }, { _id: 0, restaurant_id: 1, name: 1, borough: 1, cuisine: 1 } );
    name: 'C & C Catering Service',
    restaurant_id: '40357437'
  },
  {
    _id: ObjectId("601a7f06e1d7ee4f68379ca5"),
    address: {
      building: '1269',
      coord: [ -73.871194, 40.6730975 ],
      street: 'Sutter Avenue',
      zipcode: '11208'
    },
    borough: 'Brooklyn',
    cuisine: 'Chinese',
    grades: [
      { date: 2014-09-16T00:00:00.000Z, grade: 'B', score: 21 },
      { date: 2013-08-28T00:00:00.000Z, grade: 'A', score: 7 },
      { date: 2013-04-02T00:00:00.000Z, grade: 'C', score: 56 },
      { date: 2012-08-15T00:00:00.000Z, grade: 'B', score: 27 },
      { date: 2012-03-28T00:00:00.000Z, grade: 'B', score: 27 }
    ],
    name: 'May May Kitchen',
    restaurant_id: '40358429'
  },
  {
    _id: ObjectId("601a7f06e1d7ee4f68379ca6"),
    address: {
      building: '1',
      coord: [ -73.96926909999999, 40.7685235 ],
      street: 'East   66 Street',
      zipcode: '10065'
    },
    borough: 'Manhattan',
    cuisine: 'American ',
    grades: [
      { date: 2014-05-07T00:00:00.000Z, grade: 'A', score: 3 },
      { date: 2013-05-03T00:00:00.000Z, grade: 'A', score: 4 },
      { date: 2012-04-30T00:00:00.000Z, grade: 'A', score: 6 },
      { date: 2011-12-27T00:00:00.000Z, grade: 'A', score: 0 }
```

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

Query:

db.rest.find({}, { _id: 0, name: 1 }).sort({ name: 1 });

Output:



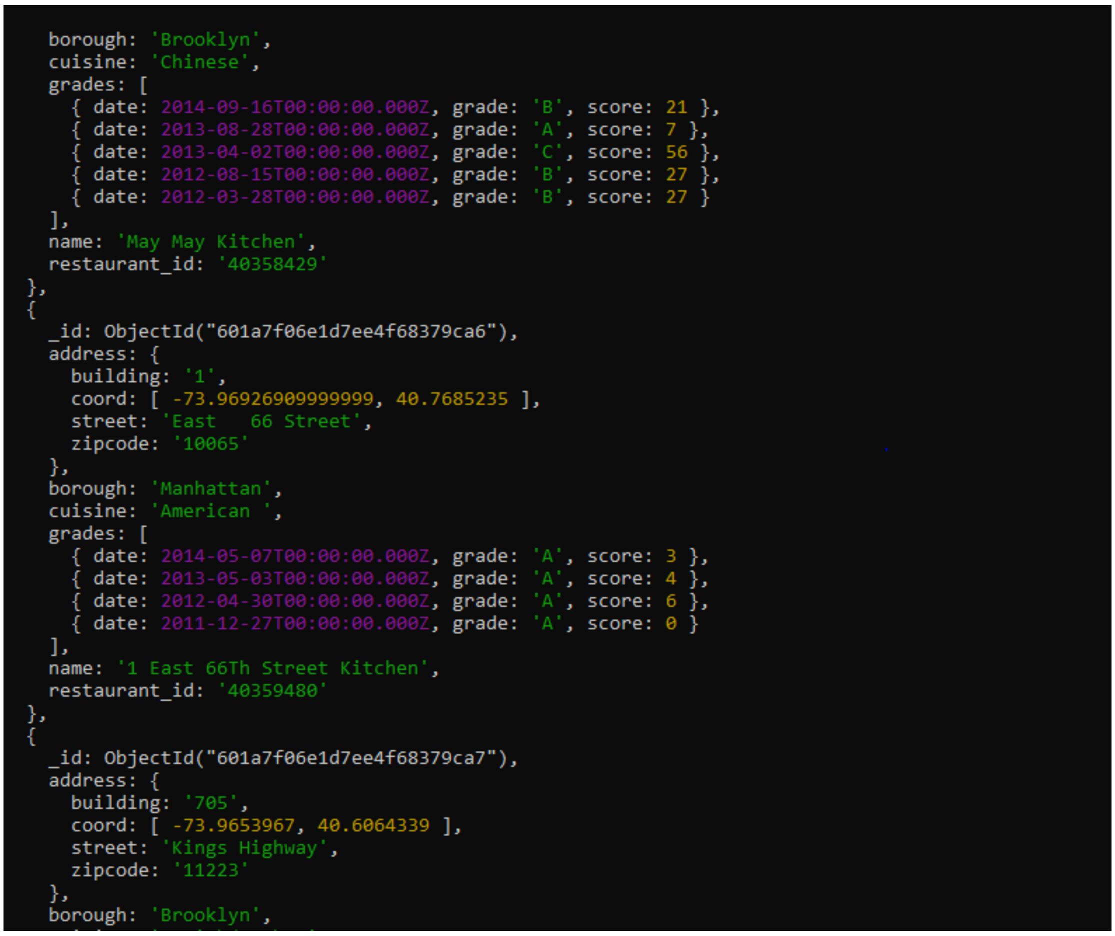
```
> db.rest.find({}, { _id: 0, name: 1 }).sort({ name: 1 });
      building: '1269',
>
      street: 'Sutter Avenue',
      zipcode: '11208'
    },
    borough: 'Brooklyn',
    cuisine: 'Chinese',
    grades: [
      { date: 2014-09-16T00:00:00.000Z, grade: 'B', score: 21 },
      { date: 2013-08-28T00:00:00.000Z, grade: 'A', score: 7 },
      { date: 2013-04-02T00:00:00.000Z, grade: 'C', score: 56 },
      { date: 2012-08-15T00:00:00.000Z, grade: 'B', score: 27 },
      { date: 2012-03-28T00:00:00.000Z, grade: 'B', score: 27 }
    ],
    name: 'May May Kitchen',
    restaurant_id: '40358429'
  },
  {
    _id: ObjectId("601a7f06e1d7ee4f68379ca6"),
    address: {
      building: '1',
      coord: [ -73.96926909999999, 40.7685235 ],
      street: 'East   66 Street',
      zipcode: '10065'
    },
    borough: 'Manhattan',
    cuisine: 'American ',
    grades: [
      { date: 2014-05-07T00:00:00.000Z, grade: 'A', score: 3 },
      { date: 2013-05-03T00:00:00.000Z, grade: 'A', score: 4 },
      { date: 2012-04-30T00:00:00.000Z, grade: 'A', score: 6 },
      { date: 2011-12-27T00:00:00.000Z, grade: 'A', score: 0 }
    ],
    name: '1 East 66Th Street Kitchen',
    restaurant_id: '40359480'
  },
```

6. Write a MongoDB query to find the restaurant name, borough, longitude, and latitude and cuisine for those restaurants which contain 'Mad' as the first three letters of its name.

Query:

```
db.rest.find(


{ name: { $regex: /^Mad.*/ } },

{ _id: 0, name: 1, borough: 1, "address.coord": 1, cuisine: 1 }


);
```

Output:



```
    borough: 'Brooklyn',
    cuisine: 'Chinese',
    grades: [
      { date: 2014-09-16T00:00:00.000Z, grade: 'B', score: 21 },
      { date: 2013-08-28T00:00:00.000Z, grade: 'A', score: 7 },
      { date: 2013-04-02T00:00:00.000Z, grade: 'C', score: 56 },
      { date: 2012-08-15T00:00:00.000Z, grade: 'B', score: 27 },
      { date: 2012-03-28T00:00:00.000Z, grade: 'B', score: 27 }
    ],
    name: 'May May Kitchen',
    restaurant_id: '40358429'
  },
  {
    _id: ObjectId("601a7f06e1d7ee4f68379ca6"),
    address: {
      building: '1',
      coord: [ -73.96926909999999, 40.7685235 ],
      street: 'East   66 Street',
      zipcode: '10065'
    },
    borough: 'Manhattan',
    cuisine: 'American ',
    grades: [
      { date: 2014-05-07T00:00:00.000Z, grade: 'A', score: 3 },
      { date: 2013-05-03T00:00:00.000Z, grade: 'A', score: 4 },
      { date: 2012-04-30T00:00:00.000Z, grade: 'A', score: 6 },
      { date: 2011-12-27T00:00:00.000Z, grade: 'A', score: 0 }
    ],
    name: '1 East 66Th Street Kitchen',
    restaurant_id: '40359480'
  },
  {
    _id: ObjectId("601a7f06e1d7ee4f68379ca7"),
    address: {
      building: '705',
      coord: [ -73.9653967, 40.6064339 ],
      street: 'Kings Highway',
      zipcode: '11223'
    },
    borough: 'Brooklyn',
```

Result:

Hence, We have created a restaurants model in Hospitality and implemented Mongodb queries to fetch collection and its data.

Ex. No: 11 **Name:**

Date: **Reg no:**

Navigation Menu

Aim:

To build a navigation menu that highlights the selected entry.

Procedure:

- Build a navigation menu that highlights the selected entry. The example uses only Angular’s directives and is the simplest app possible using the framework.

Source Code:

App.component.html:

```
<div id="main">
<nav class="{{active}}" (click)="$event.preventDefault()">
<a href="#" class="home" (click)="active='home'">Home</a>
<a href="#" class="projects" (click)="active='projects'">Projects</a>
<a href="#" class="services" (click)="active='services'">Services</a>
<a href="#" class="contact" (click)="active='contact'">Contact</a>
</nav>

<p *ngIf="!active">Please click a menu item</p>
<p *ngIf="active">You chose <b>{{active}}</b></p>
</div>
```

app.component.scss:

```
* {
margin: 0;
padding: 0;
}
```

```
body {
font: 15px/1.3 "Open Sans", sans-serif;
color: #5e5b64;
text-align: center;
}

a,
a:visited {
outline: none;
color: #389dc1;
}

a:hover {
text-decoration: none;
}

section,
footer,
header,
aside,
nav {
display: block;
}

/*-------------------------
The menu
--------------------------*/
#main {
text-align: center;
}
```

```
nav {
display: inline-block;
margin: 60px auto 45px;
background-color: #5597b4;
box-shadow: 0 1px 1px #ccc;
border-radius: 2px;
}

nav a {
display: inline-block;
padding: 18px 30px;
color: #fff !important;
font-weight: bold;
font-size: 16px;
text-decoration: none !important;
line-height: 1;
text-transform: uppercase;
background-color: transparent;

  -webkit-transition: background-color 0.25s;
  -moz-transition: background-color 0.25s;
transition: background-color 0.25s;
}

nav a:first-child {
border-radius: 2px 0 0 2px;
}
nav a:last-child {
border-radius: 0 2px 2px 0;
}
```

```
nav.home .home,
nav.projects .projects,
nav.services .services,
nav.contact .contact {
background-color: #e35885;
}

p {
font-size: 22px;
font-weight: bold;
color: #7d9098;
}

p b {
color: #ffffff;
display: inline-block;
padding: 5px 10px;
background-color: #c4d7e0;
border-radius: 2px;
text-transform: uppercase;
font-size: 18px;
}
```

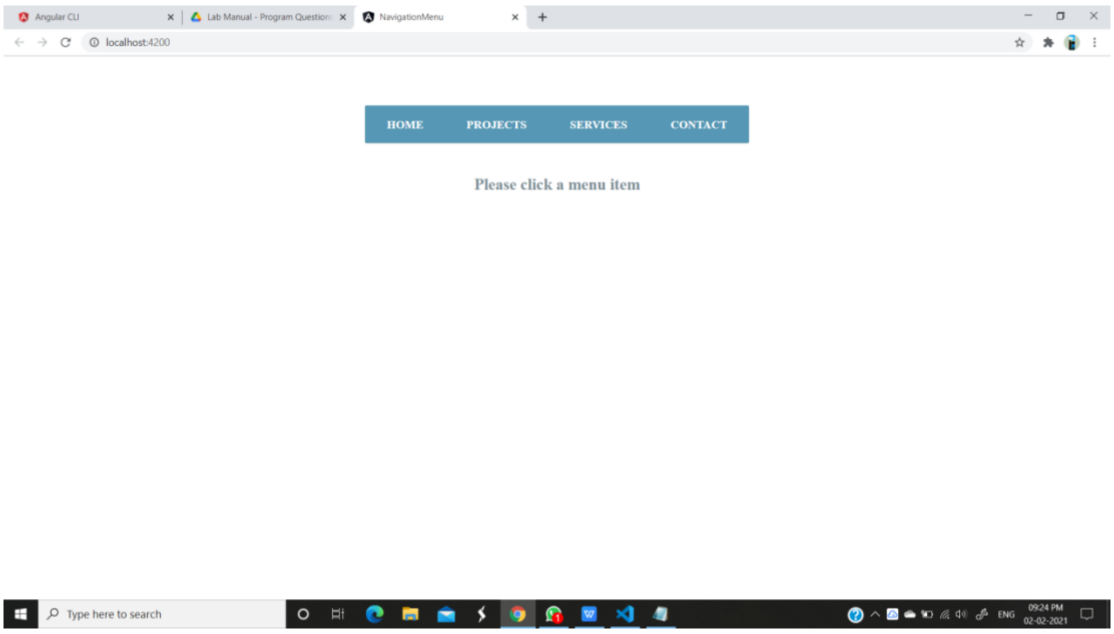
App.component.ts:

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

@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.scss']
})
export class AppComponent {
active: string;
}
```

App.module.ts:

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

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

@NgModule({
declarations: [
AppComponent
  ],
imports: [
BrowserModule
  ],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Output:



Please click a menu item

Result:

Hence, We have built a navigation menu that highlights the selected entry.

Ex. No: 12 **Name:**

Date: **Reg no:**

Inline Editor

Aim:

To create a simple inline editor.

Procedure:

- Create a simple inline editor - clicking a paragraph will show a tooltip with a text field. We will use a controller that will initialize the models and declare two methods for toggling the visibility of the tooltip.

Source Code:

app.component.html:

```
<div id="main" (click)="hideTooltip()">
<div class="tooltip" (click)="$event.stopPropagation()" *ngIf="showtooltip">
<label for="text"></label>
<input name="text" type="text" [(ngModel)]="value" />
</div>

<p (click)="toggleTooltip($event)">{{value}}</p>
</div>
```

app.component.scss:

```
* {
margin: 0;
padding: 0;
}
body {
font: 15px/1.3 "Open Sans", sans-serif;
color: #5e5b64;
text-align: center;
}
```

```
a,
a:visited {
outline: none;
color: #389dc1;
}

a:hover {
text-decoration: none;
}

section,
footer,
header,
aside,
nav {
display: block;
}

/*-------------------------
   The edit tooltip
--------------------------*/

.tooltip {
background-color: #5c9bb7;

background-image: -webkit-linear-gradient(to bottom, #5c9bb7, #5392ad);
background-image: -moz-linear-gradient(to bottom, #5c9bb7, #5392ad);
background-image: linear-gradient(to bottom, #5c9bb7, #5392ad);

box-shadow: 0 1px 1px #ccc;
```

```
border-radius: 3px;
width: 290px;
padding: 10px;

position: absolute;
left: 50%;
margin-left: -150px;
top: 80px;
}

.tooltip:after {
  /* The tip of the tooltip */
content: "";
position: absolute;
border: 6px solid #5190ac;
border-color: #5190ac transparent transparent;
width: 0;
height: 0;
bottom: -12px;
left: 50%;
margin-left: -6px;
}

.tooltip input {
border: none;
width: 100%;
line-height: 34px;
border-radius: 3px;
box-shadow: 0 2px 6px #bbb inset;
text-align: center;
font-size: 16px;
```

```
font-family: inherit;
color: #8d9395;
font-weight: bold;
outline: none;
}

p {
font-size: 22px;
font-weight: bold;
color: #6d8088;
height: 30px;
cursor: default;
text-align: center;
}

p b {
color: #ffffff;
display: inline-block;
padding: 5px 10px;
background-color: #c4d7e0;
border-radius: 2px;
text-transform: uppercase;
font-size: 18px;
}

p:before {
content: "✎";
display: inline-block;
margin-right: 5px;
font-weight: normal;
vertical-align: text-bottom;
```

```
}

#main {
height: 300px;
position: relative;
padding-top: 150px;
}
```

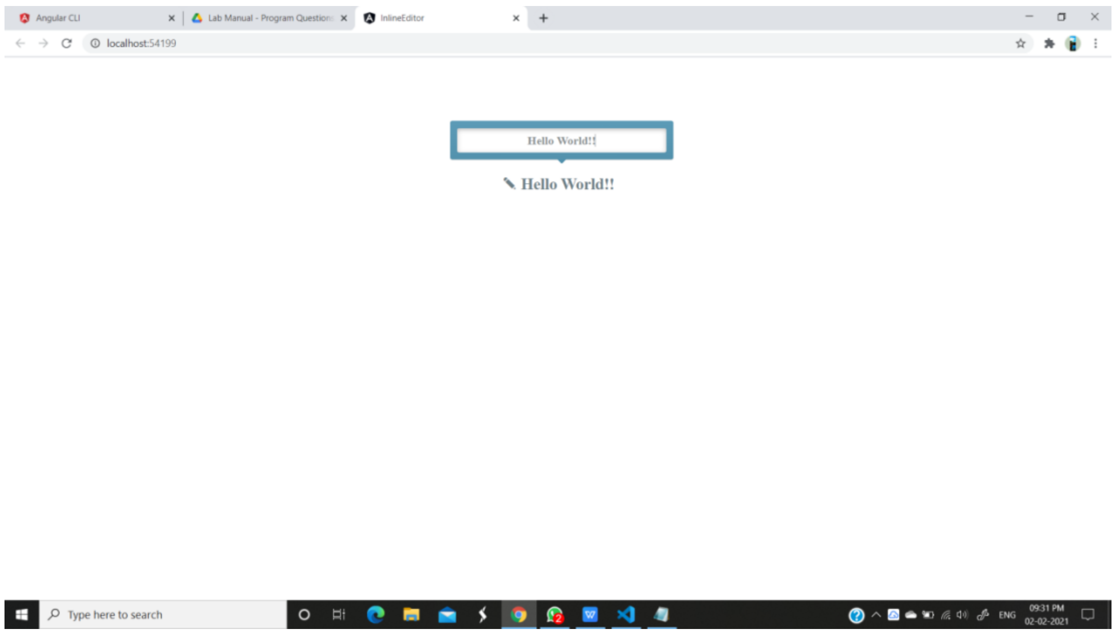
app.component.ts:

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

@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.scss']
})
export class AppComponent {
showtooltip = false;
value = 'Edit me.';

hideTooltip = function () {
this.showtooltip = false;
  }

toggleTooltip = function (e) {
e.stopPropagation();
this.showtooltip= !this.showtooltip;
  }
}
```

app.module.ts:

```
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';

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

@NgModule({
declarations: [
AppComponent
  ],
imports: [
BrowserModule,
FormsModule
  ],

providers: [],
bootstrap: [AppComponent]
})

export class AppModule { }
```

Output:



Hello World!!

Hello World!!

Result:

Hence, We have created a simple inline editor.

Ex. No: 13 **Name:**

Date: **Reg no:**

Order Form

Aim:

To code an order form with a total price updated in real time.

Procedure:

- Code an order form with a total price updated in real time, using another one of angular's useful features - pipes.

Source Code:

app.component.html:

```
<form>

<h1>Services</h1>

<ul>

<li *ngFor="let service of services" (click)="toggleActive(service)"
[class]="{active:service.active}">

    {{service.name}} <span>{{service.price | currency}}</span>

</li>

</ul>

<div class="total">

  Total: <span>{{total() | currency}}</span>

</div>

</form>
```

app.component.scss:

```
@import url(https://fonts.googleapis.com/css?family=Cookie);


* {

margin: 0;

padding: 0;

}
```

```
body {
font: 15px/1.3 "Open Sans", sans-serif;
color: #5e5b64;
text-align: center;
}
a,
a:visited {
outline: none;
color: #389dc1;
}
a:hover {
text-decoration: none;
}
section,
footer,
header,
aside,
nav {
display: block;
}
/*-------------------------
   The order form
--------------------------*/
form {
background-color: #61a1bc;
border-radius: 2px;
box-shadow: 0 1px 1px #ccc;
width: 400px;
padding: 35px 60px;
margin: 50px auto;
}
```

```
form h1 {
color: #fff;
font-size: 64px;
font-family: "Cookie", cursive;
font-weight: normal;
line-height: 1;
text-shadow: 0 3px 0 rgba(0, 0, 0, 0.1);
}

formul {
list-style: none;
color: #fff;
font-size: 20px;
font-weight: bold;
text-align: left;
margin: 20px 0 15px;
}
formul li {
padding: 20px 30px;
background-color: #e35885;
margin-bottom: 8px;
box-shadow: 0 1px 1pxrgba(0, 0, 0, 0.1);
cursor: pointer;
}

formul li span {
float: right;
}
```

```
formulli.active {
background-color: #8ec16d;
}

div.total {
border-top: 1px solid rgba(255, 255, 255, 0.5);
padding: 15px 30px;
font-size: 20px;
font-weight: bold;
text-align: left;
color: #fff;
}

div.total span {
float: right;
}
```

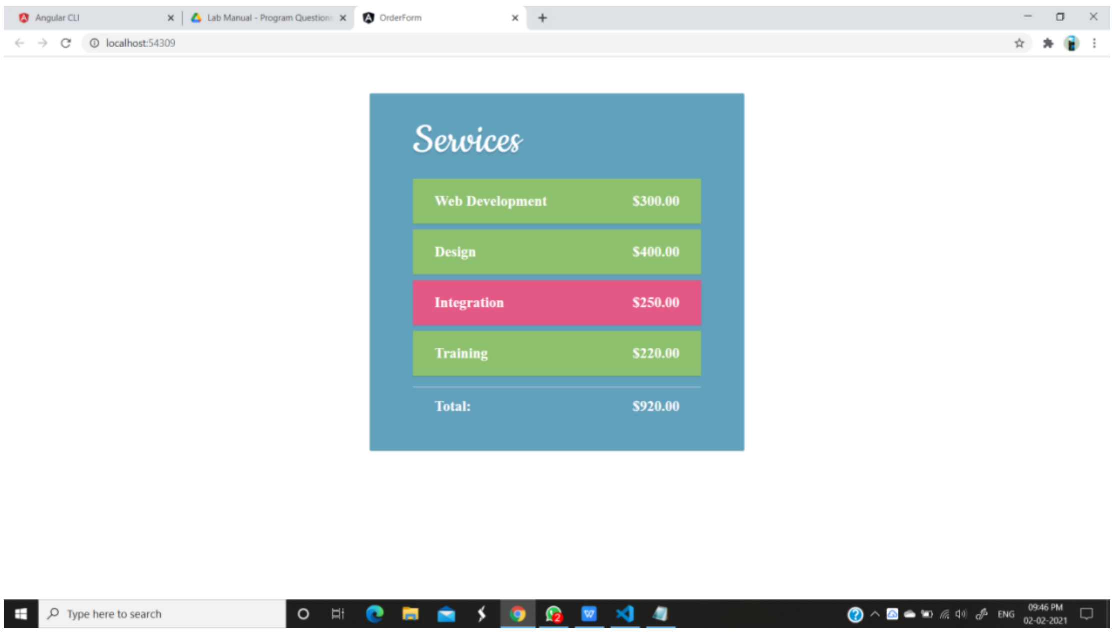
app.component.ts:

```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.scss']
})
export class AppComponent {
services = [
   {
name: 'Web Development',
price: 300,
active: true
   },
    {
name: 'Design',
price: 400,
active: false
    },
    {
name: 'Integration',
price: 250,
active: false
    },
    {
name: 'Training',
price: 220,
active: false
    }
  ];

toggleActive = function (s) {
s.active= !s.active;
  };

total = function () {
let total = 0;

this.services.forEach(element => {
if (element.active) {
total += element.price;
      }
    });
return total;
  }}
```

app.module.ts:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
@NgModule({
declarations: [
AppComponent
  ],
imports: [
BrowserModule
  ],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Output:



```
Services
Web Development    $300.00
Design             $400.00
Integration        $250.00
Training           $220.00
Total:             $920.00
```

Result:

Hence, we were able to code an order form with a total price updated in real time.

Ex. No: 14 **Name:**

Date: **Reg no:**

Instant Search

Aim:

To allow users to filter a list of items by typing into text field.

Procedure:

- This will allow users to filter a list of items by typing into text field. This is another place where Angular shines, and is the perfect use case for writing a custom pipe.

Source Code:

app.component.html:

```
<div>
<div class="bar">
<input type="text" [(ngModel)]="searchString" placeholder="Enter your search terms" />
</div>
<ul>
<li *ngFor="let i of items | searchFor:searchString">
<a [href]="i.url">
<img [src]="i.image" />
</a>
<p>{{i.title}}</p>
</li>
</ul>
</div>
```

app.component.scss:

```
* {
margin: 0;
padding: 0;
}
```

```
body {
font: 15px/1.3 "Open Sans", sans-serif;
color: #5e5b64;
text-align: center;
}
a,
a:visited {
outline: none;
color: #389dc1;
}

a:hover {
text-decoration: none;
}

section,
footer,
header,
aside,
nav {
display: block;
}
/*-------------------------
   The search input
--------------------------*/

.bar {
background-color: #5c9bb7;

background-image: -webkit-linear-gradient(to bottom, #5c9bb7, #5392ad);
background-image: -moz-linear-gradient(to bottom, #5c9bb7, #5392ad);
```

```
background-image: linear-gradient(to bottom, #5c9bb7, #5392ad);


box-shadow: 0 1px 1px #ccc;

border-radius: 2px;

width: 400px;

padding: 14px;

margin: 45px auto 20px;

position: relative;

}

.bar input {

background: #fff no-repeat 13px 13px;

  background-image:
url(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf
8/9hAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAyBpVFh
0WE1MOmNvbS5hZG9iZS54bXAAAAAAADw/eHBhY2tldCBiZWdpbj0i77u/IiBpZD0iV
zVNME1wQ2VoaUh6cmVTek5UY3prYzlkIj8+IDx4OnhtcG1ldGEgeG1sbnM6eD0iYWRv
YmU6bnM6bWV0YS8iIHg6eG1wdGs9IkFkb2JlIFhNUCBDb3JlIDUuMC1jMDYwIDYxLj
EzNDc3NywgMjAxMC8wMi8xMi0xNzozMjowMCAgICAgICAgIj4gPHJkZjpSREYgeG1s
bnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMj
Ij4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IiIgeG1sbnM6eG1wPSJodHRwOi8vb
nMuYWRvYmUuY29tL3hhcC8xLjAvIiB4bWxuczp4bXBNTT0iaHR0cDovL25zLmFkb2Jl
LmNvbS94YXAvMS4wL21tLyIgeG1sbnM6c3RSZWY9Imh0dHA6Ly9ucy5hZG9iZS5jb20
veGFwLzEuMC9zVHlwZS9SZXNvdXJjZVJlZiMiIHhtcDpDcmVhdG9yVG9vbD0iQWRv
YmUgUGhvdG9zaG9wIENTNSBXaW5kb3dzIiB4bXBNTTpJbnN0YW5jZUlEPSJ4bXAua
WlkOkU5NEY0RTlFMTA4NzExRTM5RTEzQkFBQzMyRjkyQzVBIiB4bXBNTTpEb2N1
bWVudElEPSJ4bXAuZGlkOkU5NEY0RTlGMTA4NzExRTM5RTEzQkFBQzMyRjkyQzV
BIj4gPHhtcE1NOkRlcml2ZWRGcm9tIHN0UmVmOmluc3RhbmNlSUQ9InhtcC5paWQ6R
Tk0RjRFOUMxMDg3MTFFMzlFMTNCQUFDMzJGOTJDNUEiIHN0UmVmOmRvY3Vt
ZW50SUQ9InhtcC5kaWQ6RTk0RjRFOUQxMDg3MTFFMzlFMTNCQUFDMzJGOTJDN
UEiLz4gPC9yZGY6RGVzY3JpcHRpb24+IDwvcmRmOlJERj4gPC94OnhtcG1ldGE+IDw/e
HBhY2tldCBlbmQ9InIiPz4DjA/RAAABK0lEQVR42pTSQUdEURjG8dOY0TqmPkGmRcq
YD9CmzZAWJRHVRIa0iFYtM6uofYaiEW2SRJtEi9YxIklp07ZkWswu0v/wnByve7vm5ee
8M+85zz1jbt9Os+WiGkYdYxjCOx5wgFeXUHmtBSzpcCGa+5BJTCjEP+0nKWAT8xqe4A
rPGEEVC1hHEbs2oBwdXkM7mj/JLZrad437sCGHOfUtcziutuYu2v8XUFF/4f6vMK/YgA
H1HxkBYV60AR31gxkBYd6xAeF3VzMCwvzOBpypX8V4yuFRzX2d2gD/l5yjH4fYQEnz
kj4fae5rJulF2sMXVrAsaTWttRFu4Osb+1jEDT71/ZveyhouTch2fINQL9hKefKjuYFfuznX
WzXMTabyrvfyIV3M4vhXgAEAUMs7K0J9UJAAAAAASUVORK5CYII=);


border: none;

width: 100%;
```

```
line-height: 19px;
padding: 11px 0;

border-radius: 2px;
box-shadow: 0 2px 8px #c4c4c4 inset;
text-align: left;
font-size: 14px;
font-family: inherit;
color: #738289;
font-weight: bold;
outline: none;
text-indent: 40px;
}

ul {
list-style: none;
width: 428px;
margin: 0 auto;
text-align: left;
}

ul li {
border-bottom: 1px solid #ddd;
padding: 10px;
overflow: hidden;
}

ul li img {
width: 60px;
height: 60px;
float: left;
```

```
border: none;
}

ul li p {
margin-left: 75px;
font-weight: bold;
padding-top: 12px;
color: #6e7a7f;
}
```

app.component.ts:

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

@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.scss'],
})
export class AppComponent {
searchString;
items = [
   {
url: 'https://tutorialzine.com/2013/07/50-must-have-plugins-for-extending-twitter-bootstrap/',
title: '50 Must-have plugins for extending Twitter Bootstrap',
image: 'https://tutorialzine.com/media/2013/07/featured_4.jpg'
   },
   {
url: 'https://tutorialzine.com/2013/08/simple-registration-system-php-mysql/',
title: 'Making a Super Simple Registration System With PHP and MySQL',
image: 'https://tutorialzine.com/media/2013/08/simple_registration_system.jpg'
   },
    {
url: 'https://tutorialzine.com/2013/08/slideout-footer-css/',
title: 'Create a slide-out footer with this neat z-index trick',
image: 'https://tutorialzine.com/media/2013/08/slide-out-footer.jpg'
    },
    {
url: 'https://tutorialzine.com/2013/06/digital-clock/',
title: 'How to Make a Digital Clock with jQuery and CSS3',
image: 'https://tutorialzine.com/media/2013/06/digital_clock.jpg'
    },
    {
url: 'https://tutorialzine.com/2013/05/diagonal-fade-gallery/',
title: 'Smooth Diagonal Fade Gallery with CSS3 Transitions',
image: 'https://tutorialzine.com/media/2013/05/featured.jpg'
    },
    {
url: 'https://tutorialzine.com/2013/05/mini-ajax-file-upload-form/',
title: 'Mini AJAX File Upload Form',
image: 'https://tutorialzine.com/media/2013/05/ajax-file-upload-form.jpg'
    },
    {
url: 'https://tutorialzine.com/2013/04/services-chooser-backbone-js/',
title: 'Your First Backbone.js App – Service Chooser',
image: 'https://tutorialzine.com/media/2013/04/service_chooser_form.jpg'
    }
  ];
}
```

app.module.ts:

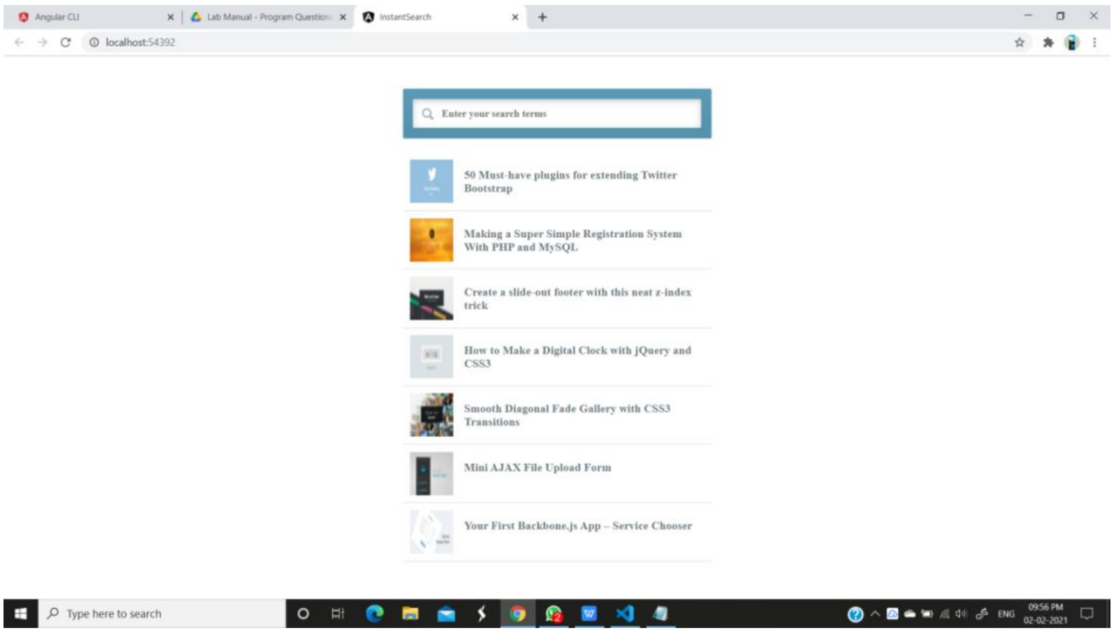
```
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { SearchForPipe } from './search-for.pipe';

@NgModule({
declarations: [
AppComponent,
SearchForPipe
  ],
imports: [
BrowserModule,
FormsModule
  ],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

search-for.pipe.ts:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
name: 'searchFor'
})
export class SearchForPipe implements PipeTransform {
```

```
transform(arr, searchString) {
if (!searchString) {
returnarr;
   }


var result = [];


searchString = searchString.toLowerCase();


   // Using the forEach helper method to loop through the array
arr.forEach(function (item) {
if (item.title.toLowerCase().indexOf(searchString) !== -1) {
result.push(item);
     }
   });


return result;
 }}
```

Output:



Enter your search terms

50 Must-have plugins for extending Twitter Bootstrap

Making a Super Simple Registration System With PHP and MySQL

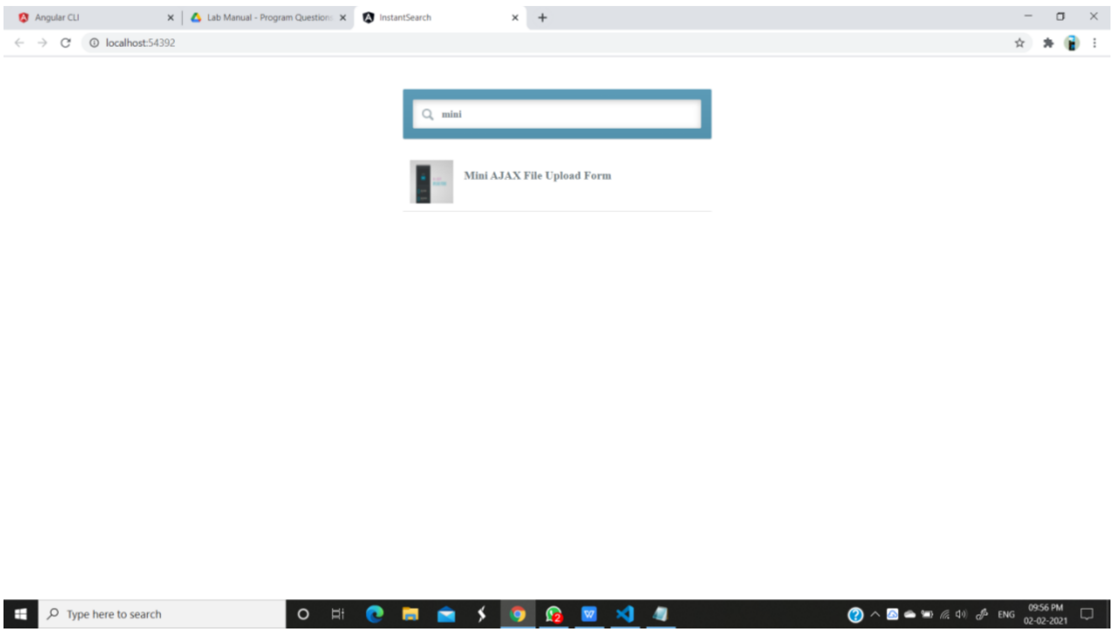
Create a slide-out footer with this neat z-index trick

How to Make a Digital Clock with jQuery and CSS3

Smooth Diagonal Fade Gallery with CSS3 Transitions

Mini AJAX File Upload Form

Your First Backbone.js App – Service Chooser



Mini AJAX File Upload Form

Result:

Hence, we have allowed users to filter a list of items by typing into text field

Ex. No: 15 **Name:**

Date: **Reg no:**

Note App

Aim:

To design a Note app with API and front end.

Procedure:

- Creating Our Server and Installing Required Packages
- Create a MongoDB Database
- Connecting Our App with MongoDB, the URL Encoder, and the Dotenv!
- Creating Our MongoDB Schema
- Setting Up the Routes
- Initializing the Angular App
- Creating components for each page
- Adding services to query the API
- Enabling the search function
- Adding some animations!

Source Code:

Frontend:

app.module.ts:

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

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

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

import {BrowserAnimationsModule} from "@angular/platform-browser/animations";

import {HTTP_INTERCEPTORS, HttpClientModule} from "@angular/common/http";

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

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

import {NotesListComponent} from './pages/notes-list/notes-list.component';

import {MainLayoutComponent} from './pages/main-layout/main-layout.component';

import {NoteCardComponent} from './note-card/note-card.component';

import {NoteDetailsComponent} from './pages/note-details/note-details.component';

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

import {SignupComponent} from './pages/signup/signup.component';

import {WebRequestInterceptor} from "./shared/web-request.interceptor";
```

```
@NgModule({  declarations: [    AppComponent,
NotesListComponent,
MainLayoutComponent,
NoteCardComponent,
NoteDetailsComponent,
LoginComponent,
SignupComponent
  ],  imports: [    BrowserModule,
AppRoutingModule,
FormsModule,
BrowserAnimationsModule,
HttpClientModule
  ],  providers: [
    {      provide: HTTP_INTERCEPTORS,
useClass: WebRequestInterceptor,
      multi: true    }  ],  bootstrap: [AppComponent]
})export class AppModule {
}
```

note-card.component.html:

```
<div class="note-card-container">
<a [routerLink]="link">
<div class="note-card-content">
<h1 class="note-card-title">{{title}}</h1>

<div #bodyText class="note-card-body">
<p>{{body}}</p>
<div #truncator class="fade-out-truncation"></div>
</div></div></a>
<div (click)="onDelete()" class="x-button"></div>
</div>
```

note-card.component.scss:

```
@import "src/styles";
.note-card-container {  position: relative;
background: white;
border-radius: 5px;
box-shadow: 0 2px 15px 2px rgba(black, 0.068);
transition: box-shadow 0.2s ease-out;
&:hover {    cursor: pointer;
box-shadow: 0 0 0 4px rgba(black, 0.068);
    .x-button {
opacity: 1;
transform: scale(1);
transition-delay: 0.35s;
    }  }
  .note-card-content {
padding: 25px;
    .note-card-title {
font-size: 22px;
font-weight: bold;
color: $purple;    }
    .note-card-body {
position: relative;
color: #555555;
      // The maximum height before it is truncated
max-height: 80px;
overflow: hidden;
      .fade-out-truncation {
position: absolute;
pointer-events: none;

bottom: 0;
height: 50px;

width: 100%;

background: linear-gradient(to bottom, rgba(white, 0) 0%, rgba(white, 0.5) 50%, white
100%);

      }   }  }

  .x-button {

position: absolute;

top: 12px;

right: 12px;

height: 34px;

width: 34px;

border-radius: 5px;

background-color: $light-red;

background-image: url("../../assets/delete_icon.svg");

background-repeat: no-repeat;

background-position: center;

    // The button is hidden by default

opacity: 0;

transform: scale(0.1);

transition: opacity 0.2s ease-out, transform 0.2s ease-out;

&:hover {      background-color: darken($light-red, 2%);

}&:active {

background-color: darken($light-red, 4%);

    }  }}
```

note-card.component.ts:

```
import {Component, ElementRef, EventEmitter, Input, OnInit, Output, Renderer2,
ViewChild} from '@angular/core';


@Component({

selector: 'app-note-card',

templateUrl: './note-card.component.html',

styleUrls: ['./note-card.component.scss']
})
export class NoteCardComponent implements OnInit {
@Input() title!: string;
@Input() body!: string;
@Input() link!: string;
@Output('delete') deleteEvent: EventEmitter<void> = new EventEmitter<void>();
@ViewChild('truncator', {static: true}) truncator!: ElementRef<HTMLElement>;
@ViewChild('bodyText', {static: true}) bodyText!: ElementRef<HTMLElement>;
constructor(private renderer: Renderer2) {
  }
ngOnInit(): void {
   // Check if there is an overflow and if not, hide the truncator
let style = window.getComputedStyle(this.bodyText.nativeElement, null);
letviewableHeight = parseInt(style.getPropertyValue("height"), 10);

if (this.bodyText.nativeElement.scrollHeight>viewableHeight) {
     // If there is a text overflow, show the fade out truncator
this.renderer.setStyle(this.truncator.nativeElement, 'display', 'block');
    } else {
     // Else (there is no text overflow), hide the fade out truncator
this.renderer.setStyle(this.truncator.nativeElement, 'display', 'none');
    }  }
onDelete() {
this.deleteEvent.emit();
  }}
```

login.component.html:

```
<div class="main-section">
<div class="main-container-box">
<div class="form-container">
<h1 class="title">Log in</h1>
```

```
<form #loginForm="ngForm" (ngSubmit)="onLogin(loginForm)">
<div class="field">
<label class="label">Email</label>
<div class="control">
<label>
<input  [ngModel]="email"
class="input"
email
minlength="5"
name="email"
placeholder="Ex: johndoe@email.com"
required
type="email"></label></div></div>
<div class="field">
<label class="label">Password</label>
<div class="control">
<label><input
          [ngModel]="password"
class="input"
minlength="8"
name="password"
placeholder="Enter a strong password"
required
type="password"></label></div></div>
<div class="field">
<div class="control">
<button [disabled]="!loginForm.valid" class="button is-primary has-text-white is-fullwidth"
style="margin-top: 28px"
type="submit">Log in</button></div></div></form></div>
<div class="alternate-bar">
<h2>Don't have an account?</h2>
```

```
<button class="button" routerLink="/signup">Sign up</button>
</div>
</div>
</div>
```

login.component.ts:

```
import {Component, OnInit} from '@angular/core';
import {Router} from '@angular/router';
import {NgForm} from '@angular/forms';
import {AuthService} from "../../shared/auth.service";
@Component({
selector: 'app-login',
templateUrl: './login.component.html',
styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {
email!: string;
password!: string;
constructor(private authService: AuthService, private router: Router) {
  }
ngOnInit(): void {
  }
onLogin(form: NgForm) {
this.authService.login(form.value.email, form.value.password).subscribe(()
=>this.router.navigateByUrl("/"))
  }}
```

Backend:

App.js:

```
const express = require('express');
constcookieParser = require('cookie-parser');
const logger = require('morgan');
constcors = require('cors');
constjwt = require('jsonwebtoken');
```

```
const app = express();
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({extended: false}));
app.use(cookieParser());
const Note = require('./models/note.model');
const User = require('./models/user.model');
require('./helpers/db'); // connect to MongoDB
app.use(cors({
exposedHeaders: ['x-access-token', 'x-refresh-token']
}));
// Check whether the request has a valid JWT Access Token
const authenticate = (req, res, next) => {
  // Grab the access token from the request header
constaccessToken = req.header('x-access-token');

  // Verify the JWT
jwt.verify(accessToken, User.getJWTSecret(), (error, decoded) => {
if (error) {
      // there was an error
      // jwt is invalid - DO NOT AUTHENTICATE
res.status(401).send({error});
    } else {
      // JWT is valid
req.userId = decoded._id;
next();
    }
  })
}

// Verify Refresh Token middleware (which will be verifying the session)
```

```
constverifySession = (req, res, next) => {
   // grab the refresh token from the request header
constrefreshToken = req.header('x-refresh-token');

   // grab the _id from the request header
const _id = req.header('_id');

User.findByIdAndToken(_id, refreshToken).then((user) => {
if (!user) {
         // user couldn't be found
returnPromise.reject({
            'error': 'User not found. Make sure that the refresh token and user id are correct.'
         })
      }

      // if the code reaches here, then the user was found
      // therefore the refresh token exists in the database
      // but we still have to check whether or not it has expired

letisSessionValid = false;

user.sessions.forEach((session) => {
if (session.token === refreshToken) {
            // check if the session has expired
if (User.hasRefreshTokenExpired(session.expiresAt) === false) {
               // the refresh token hasn't expired
isSessionValid = true;
            }
         }
      })
if (isSessionValid) {
            // the session is VALID

            // set properties on the request object
req.userId = user._id;
req.userObj = user;
req.refreshToken = refreshToken;

next();
        } else {
            // the session is NOT valid
returnPromise.reject({
                'error': 'Refresh token has expired or the session is invalid'
            })
        }
    }).catch(e => {
res.status(401).send(e);
    })
}
/* ROUTES */
/**
 * Retrieve all notes
 */
app.get('/notes', authenticate, (req, res) => {
Note.find({
        _userId: req.userId
    }).then((notes) => {
res.send(notes);
    }).catch((e) => {
res.status(400).send(e);
    })
```

```
})
/**
 * Retrieves a specific note (by id)
 */
app.get('/notes/:id', authenticate, (req, res) => {
Note.findOne({
        _id: req.params.id,
        _userId: req.userId
    }).then((note) => {
res.send(note);
    }).catch(e => {
res.status(400).send(e);
    })
})
/**
 * Create a new note
 */
app.post('/notes', authenticate, (req, res) => {
letnoteInfo = req.body;
noteInfo._userId = req.userId;

letnewNote = new Note(noteInfo);
newNote.save().then((newNoteDoc) => {
        // the full note document (incl. id) is passed to this callback
res.send(newNoteDoc);
    }).catch((e) => {
res.status(400).send(e);
    })
})

/**
```

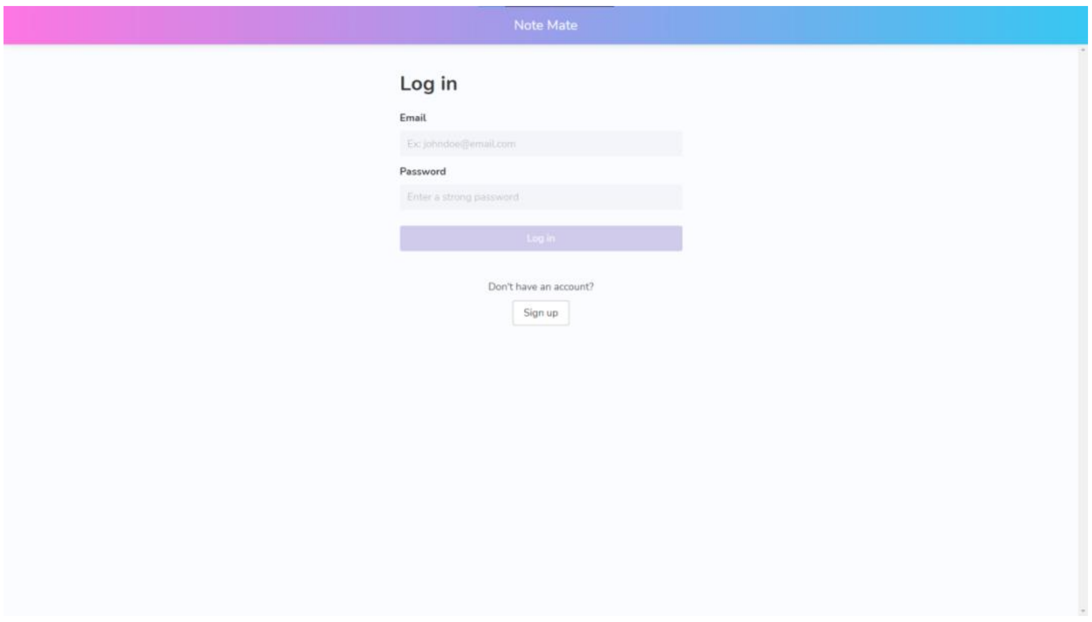
```
 * Update a note
 */
app.patch('/notes/:id', authenticate, (req, res) => {
Note.findOneAndUpdate({
        _id: req.params.id,
        _userId: req.userId
    }, {
        $set: req.body
    }).then(() => {
res.send();
    }).catch((e) => {
res.status(400).send(e);
    })
})

/**
 * Delete a note
 */
app.delete('/notes/:id', authenticate, (req, res) => {
Note.findOneAndRemove({
        _id: req.params.id,
        _userId: req.userId
    }).then((removedNoteDoc) => {
res.send(removedNoteDoc);
    })
})
/* USER ROUTES */

/**
 * Create a user (Sign up)
 */
```

```
app.post('/users', (req, res) => {
constuserInfo = req.body;
constnewUser = new User(userInfo);

newUser.save().then(() => {
res.send(newUser);
    }).catch(e => {
res.status(400).send(e);
    })
})
/**
 * Log in
 */
app.post('/users/login', (req, res) => {
const email = req.body.email;
const password = req.body.password;

User.findByCredentials(email, password).then((user) => {
returnuser.createSession().then((refreshToken) => {
            // Session has been created successfully
            // and the refresh token has been returned
returnuser.generateAccessToken().then((accessToken) => {
                // access token has been generated successfully
                // so now we return an object containing the auth tokens
return {accessToken, refreshToken}
            })
        }).then((authTokens) => {
            // now construct and send the response to the caller
            // with their auth tokens in the response headers
            // and the user object in the response body
res
```

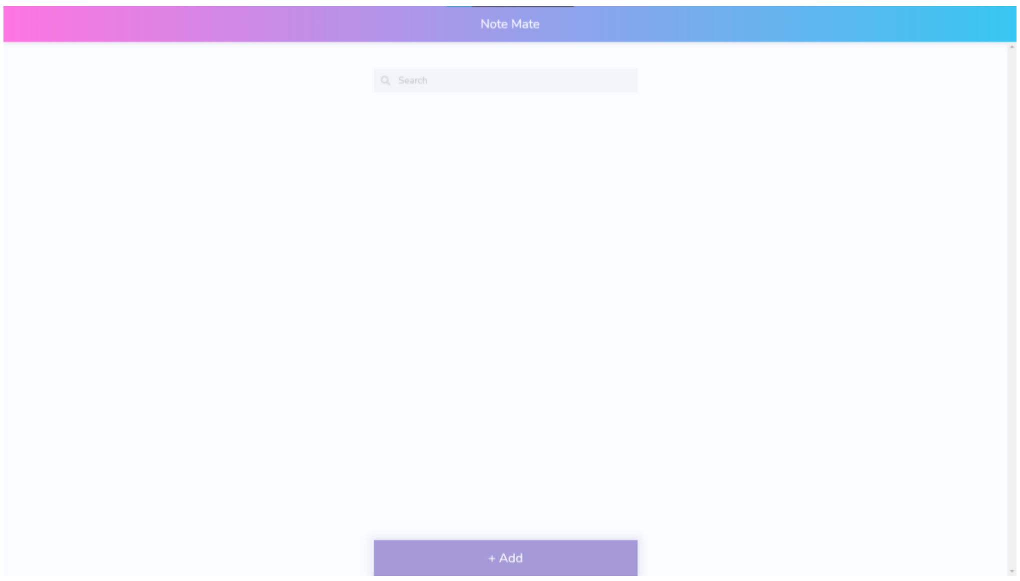
```
                .header('x-refresh-token', authTokens.refreshToken)
                .header('x-access-token', authTokens.accessToken)
                .send(user);
        })
    }).catch(e => {
res.status(400).send(e);
    })
})
/**
 * Generating a fresh access token
 */
app.get('/users/me/access-token', verifySession, (req, res) => {
    // we can use the user object to generate a new access token
    // we have access to the user object because of verifySession
req.userObj.generateAccessToken().then((accessToken) => {
res.header('x-access-token', accessToken).send();
    }).catch((e) => {
res.status(400).send(e);
    })})/** * Update user details
 */
app.patch('/users/me', authenticate, (req, res) => {
let body = req.body;
deletebody.sessions;
User.findById(req.userId).then((userDoc) => {
Object.assign(userDoc, body);
userDoc.save().then(() => {
res.status(200).send();
        })
    }).catch(e => {
res.status(400).send(e);
    })})/** * Logout (Delete a session from the database)
 */app.delete('/users/me/session', verifySession, (req, res) => {
let _id = req.userId;
letrefreshToken = req.refreshToken; // this is the token we have to invalidate
User.findOneAndUpdate({_id}, {
$pull: {
sessions: {
token: refreshToken
            }
        }
    }).then(() => {
res.status(200).send();
    })
}) module.exports = app;
```

Output:

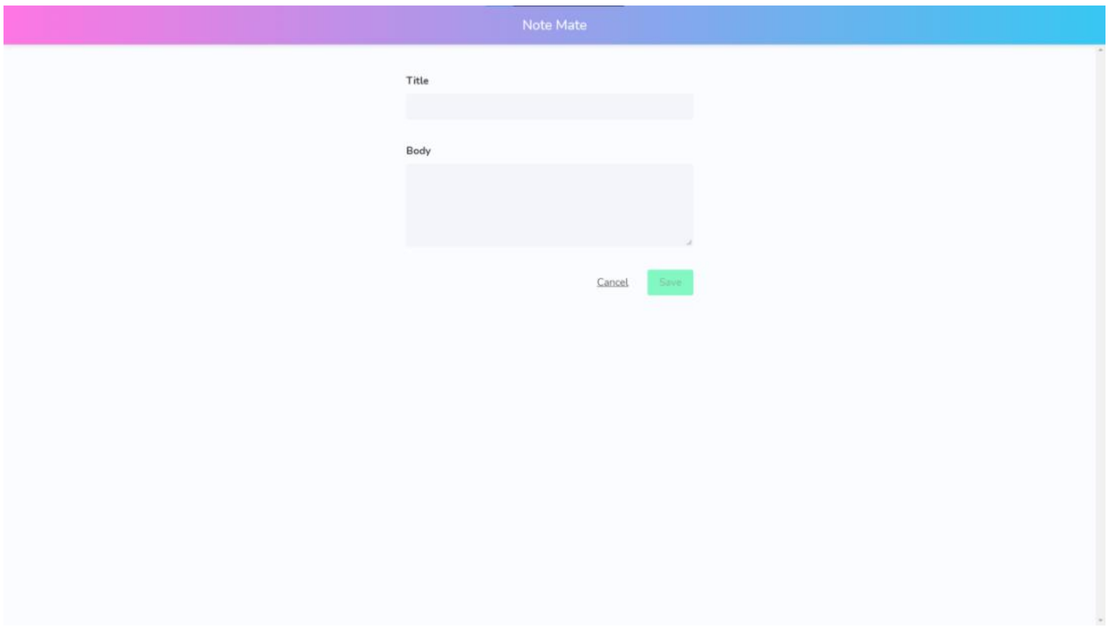
Login:



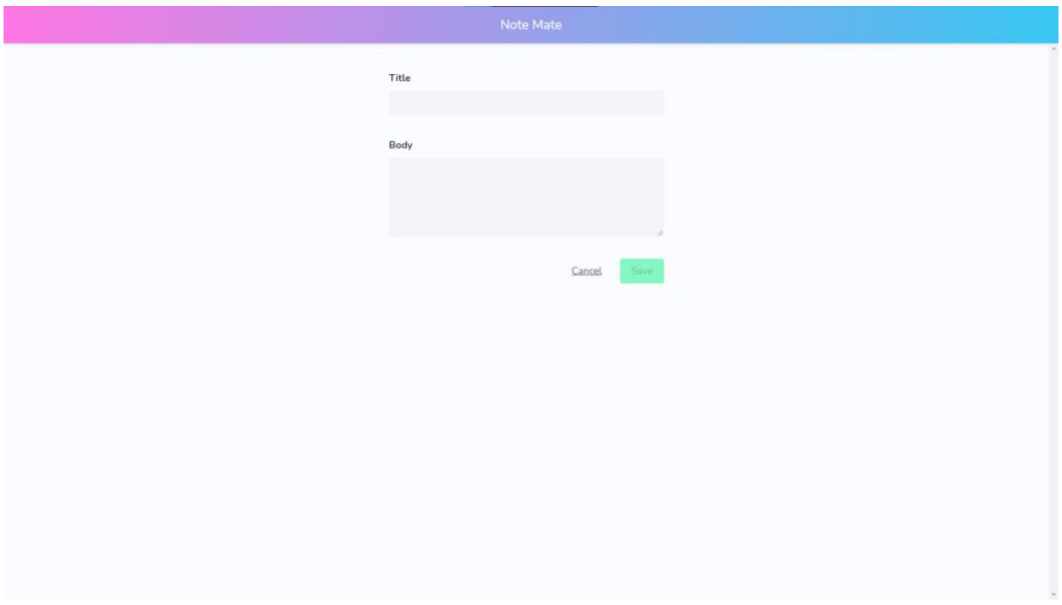
Main Page:



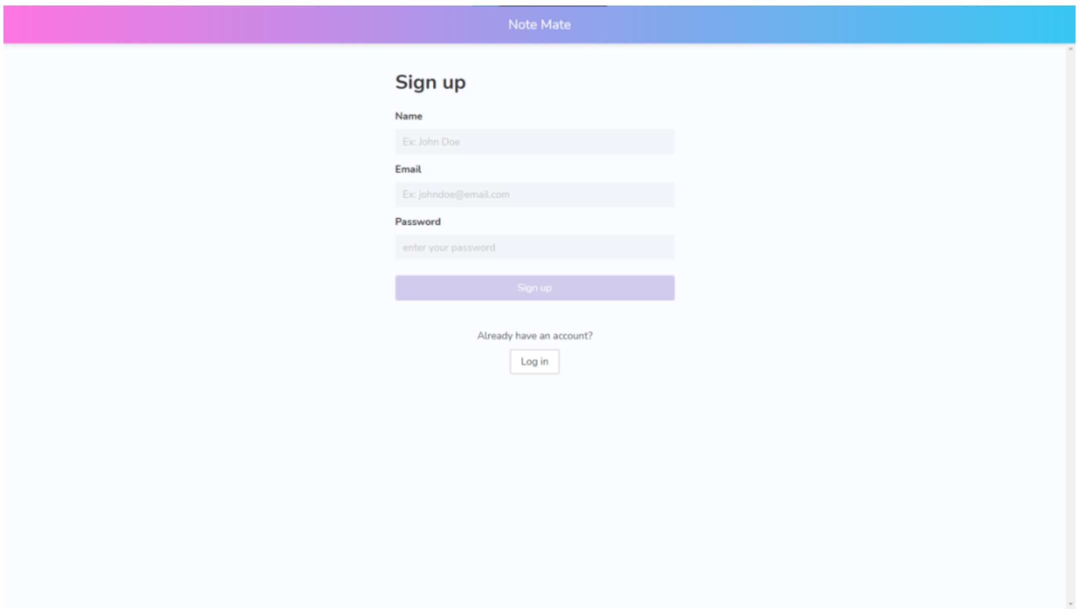
NewNote:



NotesDisplay:



SignUp:



Result:

Hence, we designed a Note app with API and front end.