PRACTICAL NO .1

AIM :- Write a program to implement MongoDB data models (Employee collection)

❖ INTRODUCTION :-

MongoDB, a leading NoSQL database, offers flexibility in data modelling, allowing developers to design schemas tailored to their application requirements. This program explores implementing MongoDB data models using the Node.js driver. Through basic CRUD operations, we'll demonstrate creating, reading, updating, and deleting data while showcasing diverse data modelling techniques within MongoDB.

- **Create (C):** Adding new data or records to the database. In MongoDB, this is typically done using methods like insertOne() to add a single document or insertMany() to add multiple documents at once.

- **Read (R):** Retrieving data or records from the database. In MongoDB, you use the find() method to retrieve documents from a collection. You can apply filters and projections to narrow down the results.

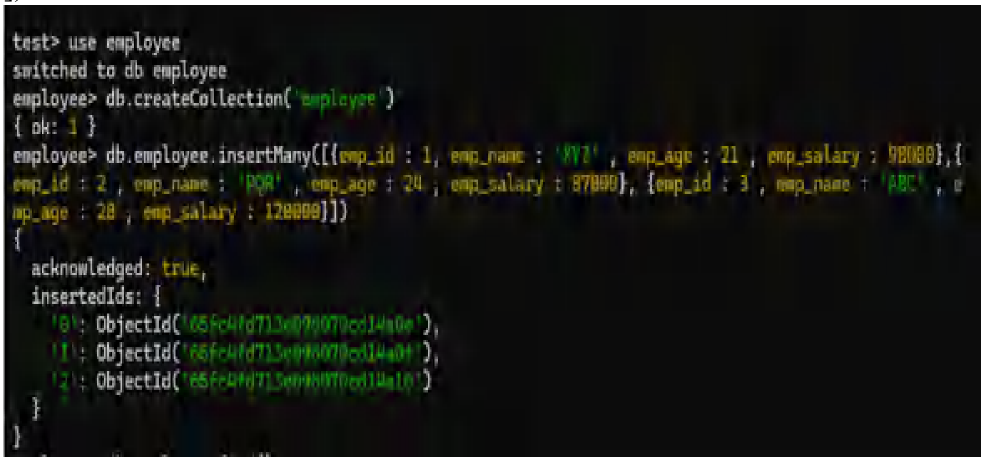
- **Update (U):** Modifying existing data or records in the database. In MongoDB, you use methods like updateOne() to update a single document or updateMany() to update multiple documents. You can specify which fields to modify using update operators like $set.

- **Delete (D):** Removing data or records from the database. In MongoDB, you use methods like deleteOne() to remove a single document or deleteMany() to remove multiple documents. You specify criteria to identify the documents to delete.

❖ **COMMANDS :-**

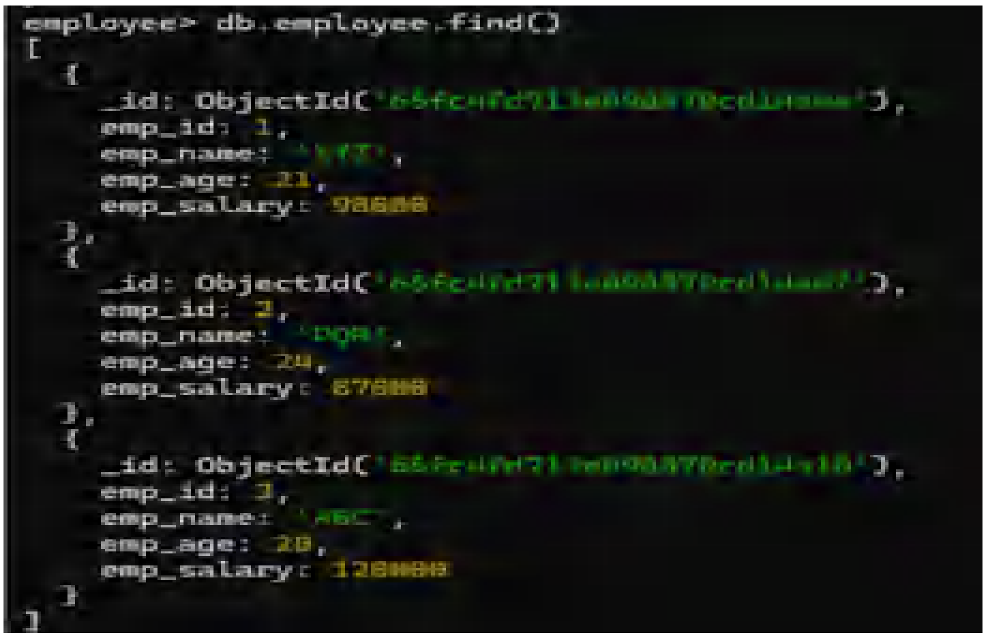
- **Create & Use database :**
  test> use employee
  employee> db.createCollection('employee')

- **Insert :**
  db.employee.insertOne({ emp_id: 1, emp_name: 'XYZ', emp_age: 21, emp_salary: 98000 });
  db.employee.insertMany([
    { emp_id: 2, emp_name: 'PQR', emp_age: 24, emp_salary: 87000 },
    { emp_id: 3, emp_name: 'ABC', emp_age: 28, emp_salary: 120000 }
  ])

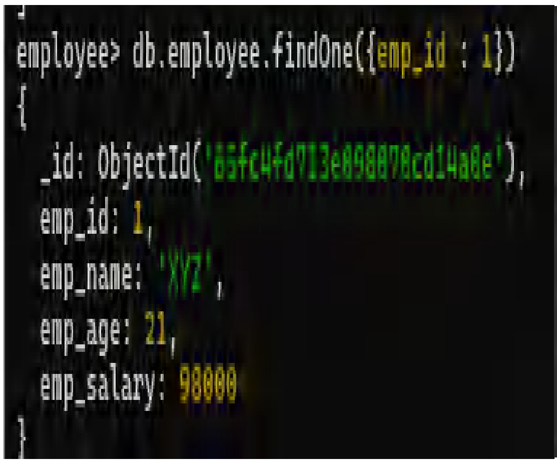


```
test> use employee
switched to db employee
employee> db.createCollection('employee')
{ ok: 1 }
employee> db.employee.insertMany([{emp_id : 1, emp_name : 'XYZ' , emp_age : 21 , emp_salary : 98000},{
emp_id : 2 , emp_name : 'PQR' , emp_age : 24 , emp_salary : 87000}, {emp_id : 3 , emp_name : 'ABC' , e
mp_age : 28 , emp_salary : 120000}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('[illegible]'),
    '1': ObjectId('[illegible]'),
    '2': ObjectId('[illegible]')
  }
}
```

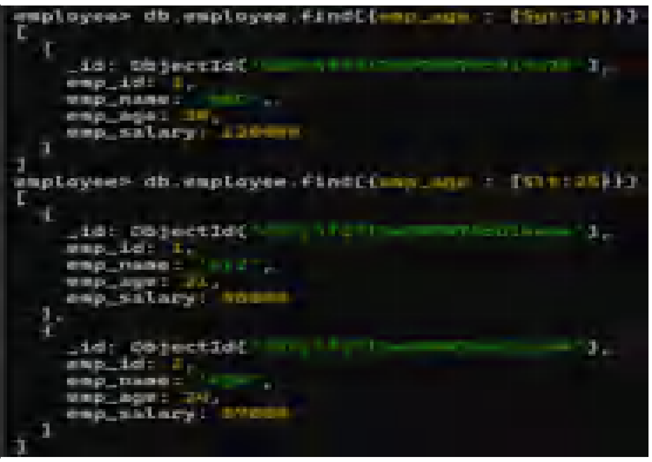
- **Find() :**
  employee> db.employee.find()
  employee> db.employee.findOne({emp_id : 1})
  employee> db.employee.find().sort({emp_age : 1})
  employee> db.employee.find().sort({emp_age : -1})
  employee> db.employee.find({emp_age : {$gt:25}})
  employee> db.employee.find({emp_age : {$lt:25}})
  employee> db.employee.find().pretty()



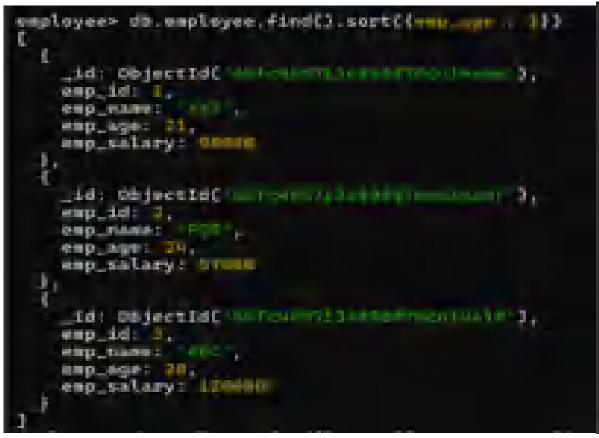
```
employee> db.employee.find()
[
  {
    _id: ObjectId('[illegible]'),
    emp_id: 1,
    emp_name: 'XYZ',
    emp_age: 21,
    emp_salary: 98000
  },
  {
    _id: ObjectId('[illegible]'),
    emp_id: 2,
    emp_name: 'PQR',
    emp_age: 24,
    emp_salary: 87000
  },
  {
    _id: ObjectId('[illegible]'),
    emp_id: 3,
    emp_name: 'ABC',
    emp_age: 28,
    emp_salary: 120000
  }
]
```



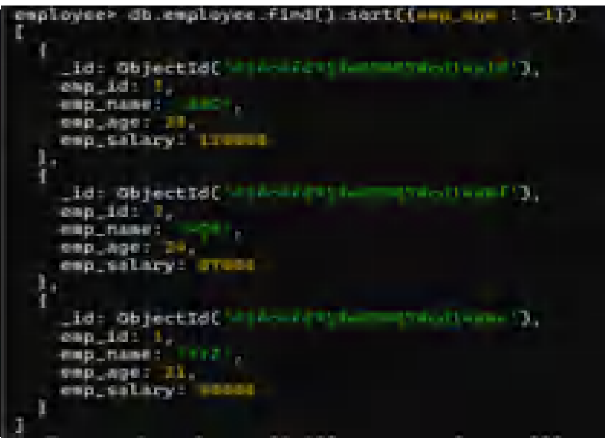
```
employee> db.employee.findOne({emp_id : 1})
{
  _id: ObjectId('65fc4fd713e898078cd14a8e'),
  emp_id: 1,
  emp_name: 'XYZ',
  emp_age: 21,
  emp_salary: 98000
}
```



```
employee> db.employee.find({emp_age : {$gt:25}})
[
  {
    _id: ObjectId('[illegible]'),
    emp_id: 3,
    emp_name: 'ABC',
    emp_age: 28,
    emp_salary: 120000
  }
]
employee> db.employee.find({emp_age : {$lt:25}})
[
  {
    _id: ObjectId('[illegible]'),
    emp_id: 1,
    emp_name: 'XYZ',
    emp_age: 21,
    emp_salary: 98000
  },
  {
    _id: ObjectId('[illegible]'),
    emp_id: 2,
    emp_name: 'PQR',
    emp_age: 24,
    emp_salary: 97000
  }
]
```



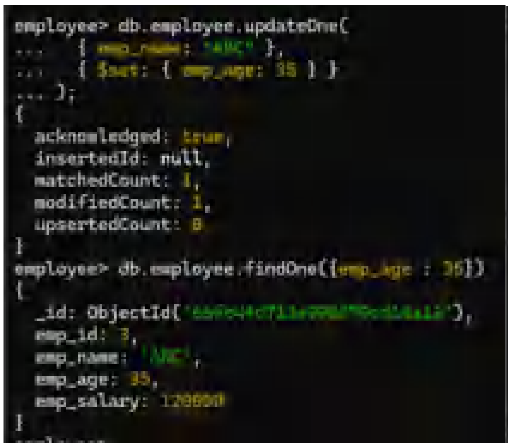
```
employee> db.employee.find().sort({emp_age : 1})
[
  {
    _id: ObjectId('[illegible]'),
    emp_id: 1,
    emp_name: 'XYZ',
    emp_age: 21,
    emp_salary: 98000
  },
  {
    _id: ObjectId('[illegible]'),
    emp_id: 2,
    emp_name: 'PQR',
    emp_age: 24,
    emp_salary: 97000
  },
  {
    _id: ObjectId('[illegible]'),
    emp_id: 3,
    emp_name: 'ABC',
    emp_age: 28,
    emp_salary: 120000
  }
]
```



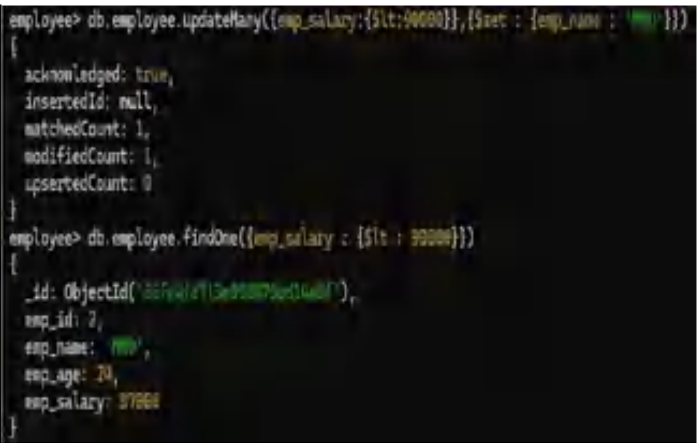
```
employee> db.employee.find().sort({emp_age : -1})
[
  {
    _id: ObjectId('[illegible]'),
    emp_id: 3,
    emp_name: 'ABC',
    emp_age: 28,
    emp_salary: 120000
  },
  {
    _id: ObjectId('[illegible]'),
    emp_id: 2,
    emp_name: 'PQR',
    emp_age: 24,
    emp_salary: 97000
  },
  {
    _id: ObjectId('[illegible]'),
    emp_id: 1,
    emp_name: 'XYZ',
    emp_age: 21,
    emp_salary: 98000
  }
]
```

- **<u>updateOne() & updateMany() :</u>**

```
db.employee.updateOne(
  { emp_name: "ABC" }, // Filter condition
  { $set: { emp_age: 35 } } // Update operation
)
db.employee.updateMany(
  { emp_salary: { $lt: 90000 } }, // Filter condition: Employees with salary less than 90000
  { $set: { emp_name: 'MNO' } } // Update operation: Set emp_name to 'MNO'
)
```



```
employee> db.employee.updateOne(
...     { emp_name: "ABC" },
...     { $set: { emp_age: 35 } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
employee> db.employee.findOne({emp_age : 35})
{
  _id: ObjectId('[illegible]'),
  emp_id: 3,
  emp_name: 'ABC',
  emp_age: 35,
  emp_salary: 120000
}
```



```
employee> db.employee.updateMany({emp_salary:{$lt:90000}},{$set : {emp_name : 'MNO'}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
employee> db.employee.findOne({emp_salary : {$lt : 99000}})
{
  _id: ObjectId('[illegible]'),
  emp_id: 2,
  emp_name: 'MNO',
  emp_age: 24,
  emp_salary: 97000
}
```

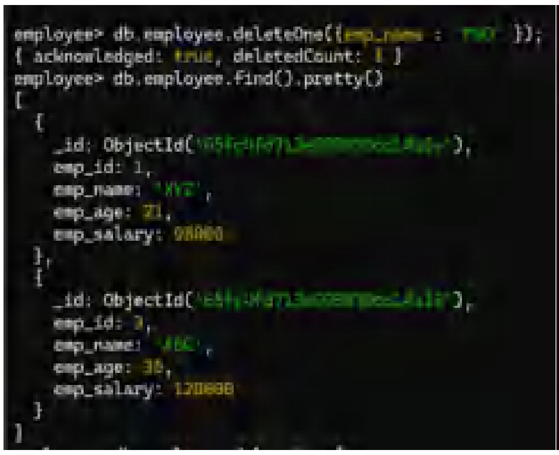
- **<u>deleteOne() & deleteMany() :</u>**

  employee> db.employee.deleteOne({emp_name : 'MNO' })

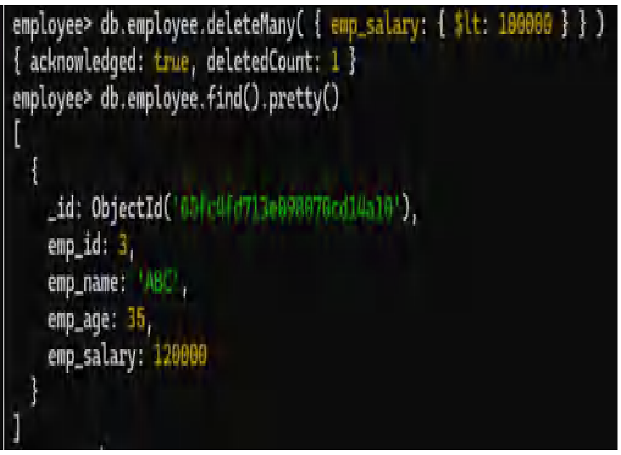
  db.employee.deleteMany(

     { emp_salary: { $lt: 10000 } }

)



```
employee> db.employee.deleteOne({emp_name : 'MNO' });
{ acknowledged: true, deletedCount: 1 }
employee> db.employee.find().pretty()
[
  {
    _id: ObjectId('[illegible]'),
    emp_id: 1,
    emp_name: 'XYZ',
    emp_age: 21,
    emp_salary: 98000
  },
  {
    _id: ObjectId('[illegible]'),
    emp_id: 3,
    emp_name: 'ABC',
    emp_age: 35,
    emp_salary: 120000
  }
]
```



```
employee> db.employee.deleteMany( { emp_salary: { $lt: 100000 } } )
{ acknowledged: true, deletedCount: 1 }
employee> db.employee.find().pretty()
[
  {
    _id: ObjectId('65fc4fd713e098970cd14a10'),
    emp_id: 3,
    emp_name: 'ABC',
    emp_age: 35,
    emp_salary: 120000
  }
]
```

PRACTICAL NO .2

AIM :- **Create collection student with field student id, name, marks, status, class.**

1. **Find all the document of students.**
2. **Find list of all students who are passed.**
3. **Find all the students of Computer Science.**
4. **Update status to 'Fail' of all the students with marks less than 40**

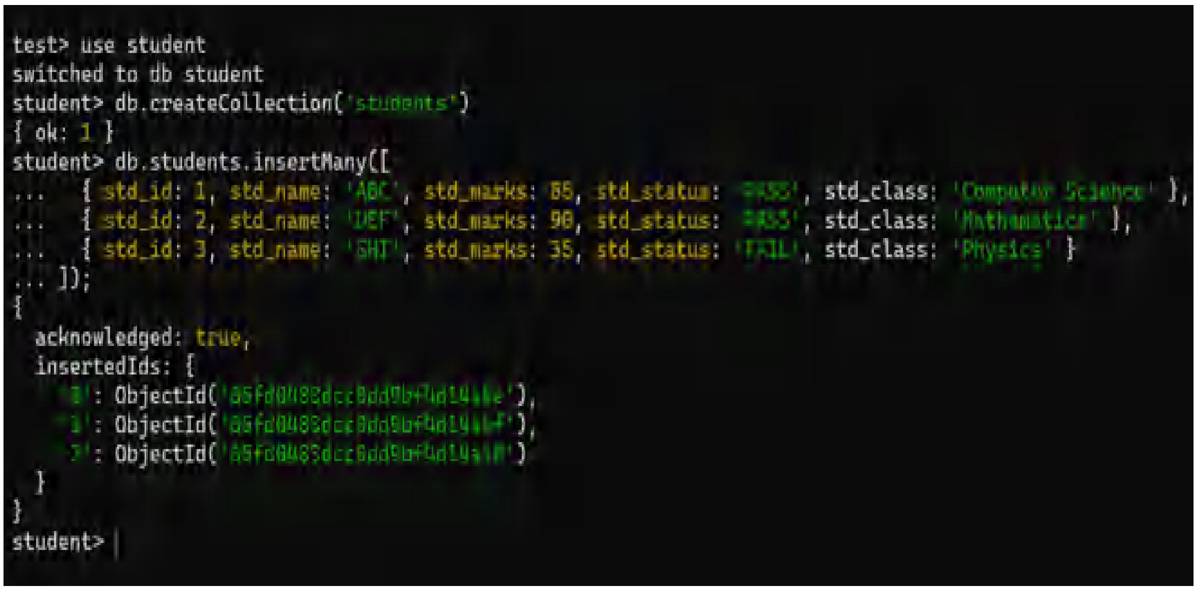
❖ INTRODUCTION :-

In this program, we'll explore how to implement the Students collection in MongoDB using the Node.js driver. We'll demonstrate how to perform CRUD operations to create, read, update, and delete student data, as well as showcase various data modeling techniques to effectively manage student information within MongoDB. Let's dive into the implementation of the Students collection and harness the power of MongoDB for educational applications.

- **Find all documents of students:** Using find() without filter conditions displays all student documents, providing a comprehensive view of student data.
- **Find list of passed students**: With find() and a filter for 'Pass' status, retrieve documents of students who passed their examinations, facilitating identification of successful students.
- **Find all Computer Science students:** By applying find() with a filter for 'Computer Science' class, retrieve documents specifically for students enrolled in that program, offering insights tailored to their studies.
- **Update status to 'Fail' for students with marks < 40:** Leveraging updateMany(), we'll change the status to 'Fail' for students with marks below 40, aiding in identifying students needing additional support.

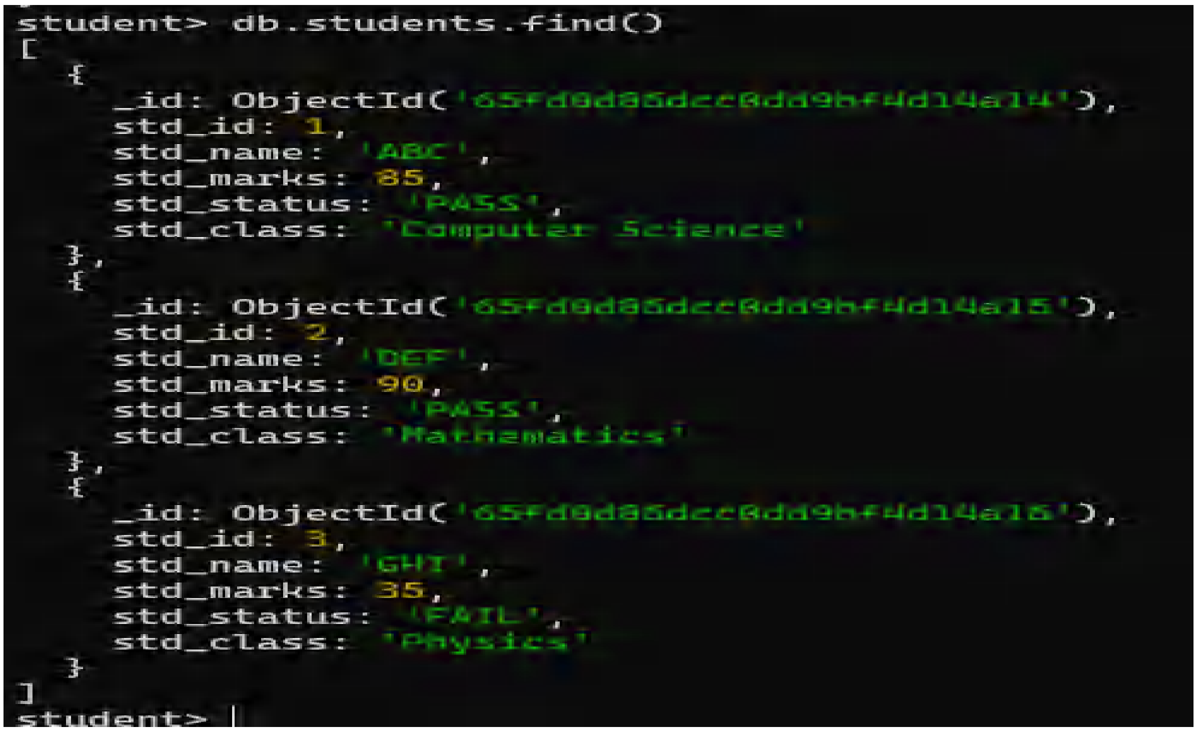
❖ **<u>COMMANDS :-</u>**

- **<u>Create and use database :</u>**
  test> use student
  student> db.createCollection('students')
  db.students.insertMany([
    { std_id: 1, std_name: 'ABC', std_marks: 85, std_status: 'PASS', std_class: 'Computer Science' },
    { std_id: 2, std_name: 'DEF', std_marks: 90, std_status: 'PASS', std_class: 'Mathematics' },
    { std_id: 3, std_name: 'GHI', std_marks: 35, std_status: 'FAIL', std_class: 'Physics' }
  ]);

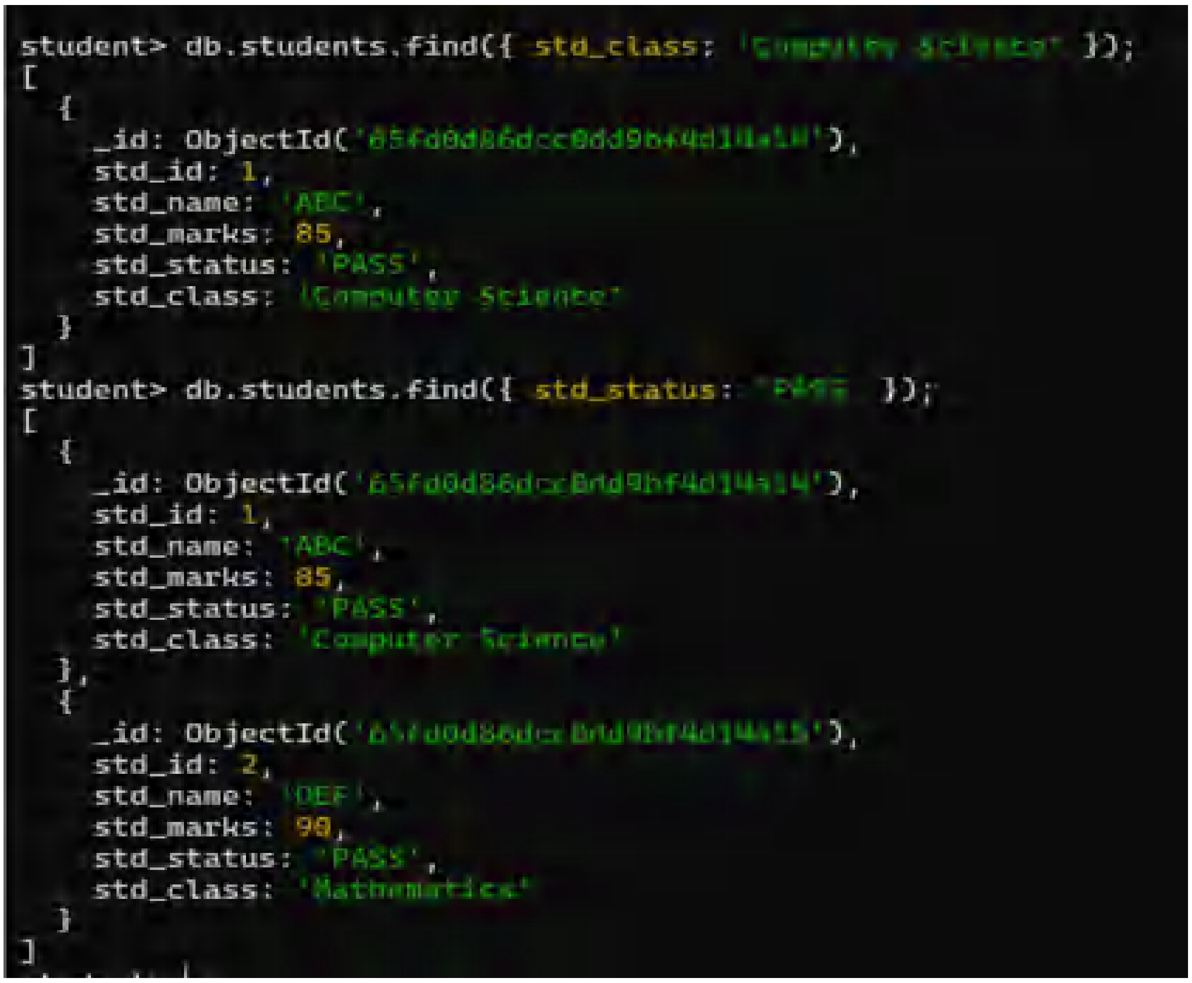


```
test> use student
switched to db student
student> db.createCollection('students')
{ ok: 1 }
student> db.students.insertMany([
...   { std_id: 1, std_name: 'ABC', std_marks: 85, std_status: 'PASS', std_class: 'Computer Science' },
...   { std_id: 2, std_name: 'DEF', std_marks: 90, std_status: 'PASS', std_class: 'Mathematics' },
...   { std_id: 3, std_name: 'GHI', std_marks: 35, std_status: 'FAIL', std_class: 'Physics' }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65fd0488dcc0dd9bf4d14aae'),
    '1': ObjectId('65fd0488dcc0dd9bf4d14aaf'),
    '2': ObjectId('65fd0485dcc0dd9bf4d14ab0')
  }
}
student> |
```

- **<u>Find() :</u>**
  db.students.find()
  db.students.find({ std_class: 'Computer Science' });
  db.students.find({ std_status: ‘PASS’ });

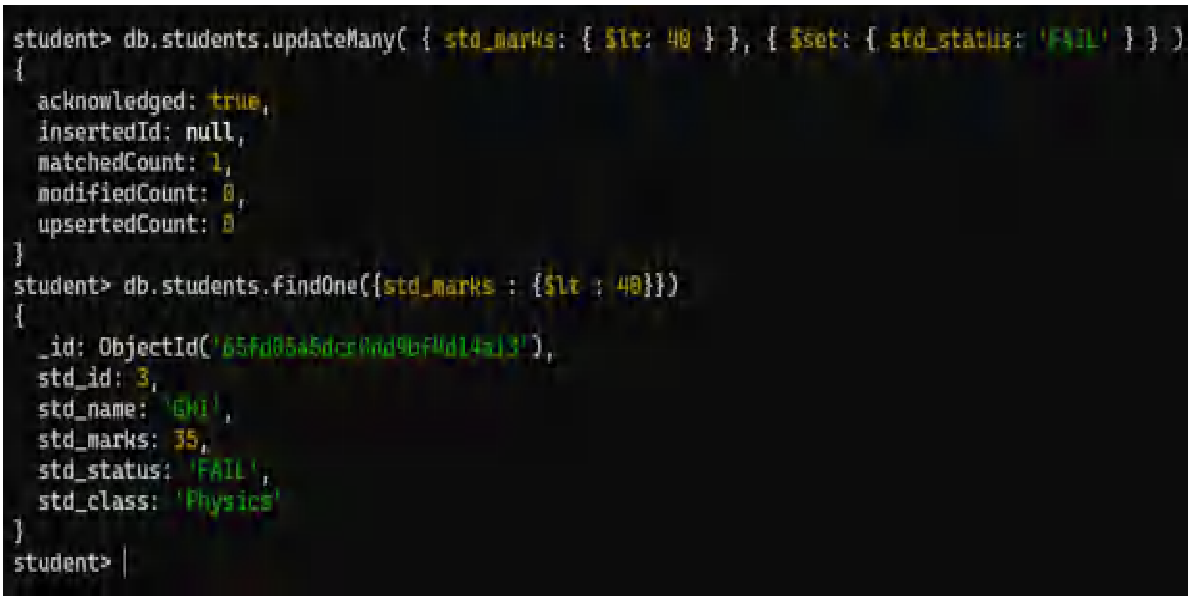


```
student> db.students.find()
[
  {
    _id: ObjectId('65fd9d85dcc0dd9bf4d14a14'),
    std_id: 1,
    std_name: 'ABC',
    std_marks: 85,
    std_status: 'PASS',
    std_class: 'Computer Science'
  },
  {
    _id: ObjectId('65fd9d85dcc0dd9bf4d14a15'),
    std_id: 2,
    std_name: 'DEF',
    std_marks: 90,
    std_status: 'PASS',
    std_class: 'Mathematics'
  },
  {
    _id: ObjectId('65fd9d85dcc0dd9bf4d14a16'),
    std_id: 3,
    std_name: 'GHI',
    std_marks: 35,
    std_status: 'FAIL',
    std_class: 'Physics'
  }
]
student> |
```



```
student> db.students.find({ std_class: 'Computer Science' });
[
  {
    _id: ObjectId('65fd0d86dcc0dd9bf4d14a14'),
    std_id: 1,
    std_name: 'ABC',
    std_marks: 85,
    std_status: 'PASS',
    std_class: 'Computer Science'
  }
]
student> db.students.find({ std_status: 'PASS' });
[
  {
    _id: ObjectId('65fd0d86dcc0dd9bf4d14a14'),
    std_id: 1,
    std_name: 'ABC',
    std_marks: 85,
    std_status: 'PASS',
    std_class: 'Computer Science'
  },
  {
    _id: ObjectId('65fd0d86dcc0dd9bf4d14a15'),
    std_id: 2,
    std_name: 'DEF',
    std_marks: 90,
    std_status: 'PASS',
    std_class: 'Mathematics'
  }
]
```

- **<u>Update:-</u>**
  db.students.updateMany({ std_marks: { $lt: 40 } }, { $set: { std_status: 'FAIL' } })



```
student> db.students.updateMany( { std_marks: { $lt: 40 } }, { $set: { std_status: 'FAIL' } } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
student> db.students.findOne({std_marks : {$lt : 40}})
{
  _id: ObjectId('65fd05a5dcc0dd9bf4d14a13'),
  std_id: 3,
  std_name: 'GHI',
  std_marks: 35,
  std_status: 'FAIL',
  std_class: 'Physics'
}
student> |
```

PRACTICAL NO .3

AIM :- Create Customer collection (id, name, age. contact) with atleast 5 document and perform :

1. **Display all the document from the collection.**
2. **List out the document of the customer with age less than 18**

❖ INTRODUCTION :-

In this program the "Customer" collection stores information about customers, including their ID, name, age, and contact details. This program aims to create a Customer collection with at least five documents. Then, we'll perform two tasks: displaying all documents from the collection and listing documents of customers aged less than 18.

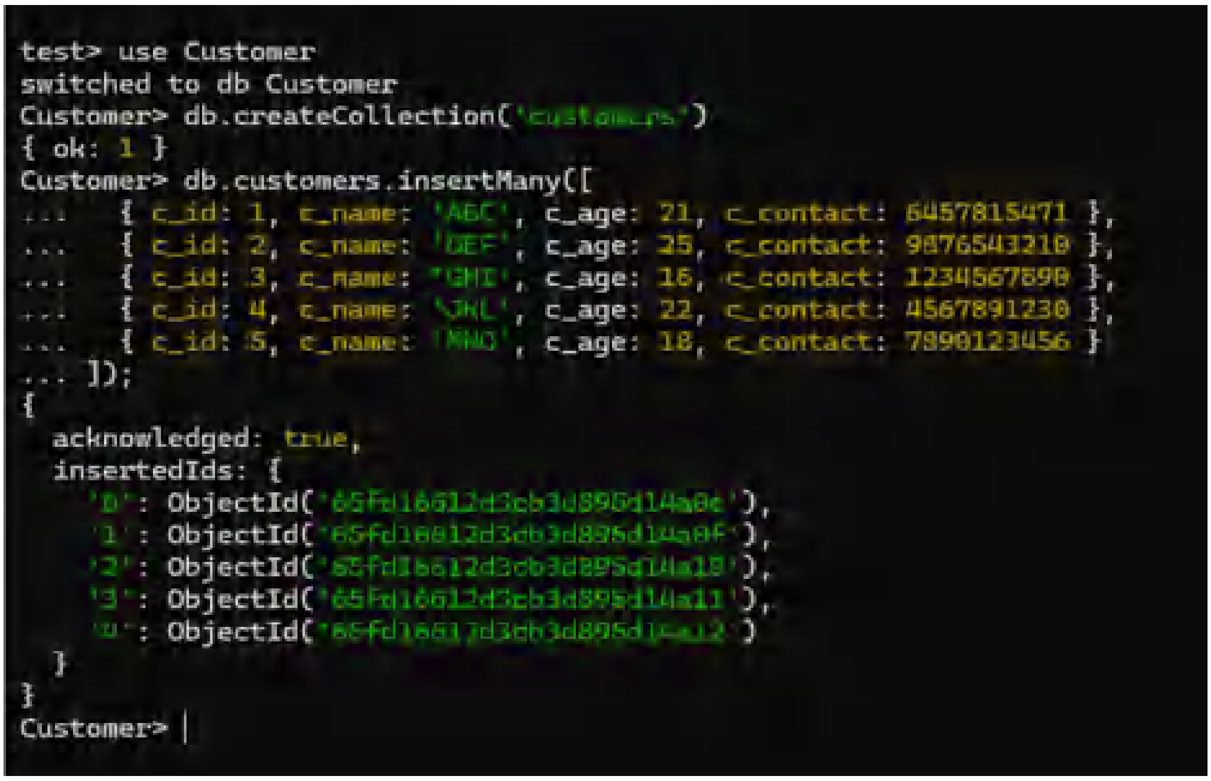
- **insertMany():** We'll use this method to insert multiple documents representing customer records into the Customer collection. Each document will contain fields such as ID, name, age, and contact.

- **find():** We'll use this method to retrieve documents from the Customer collection. For the first task, we'll use it to display all documents in the collection. For the second task, we'll use it to find and display documents where the age of the customers is less than 18.

❖ **<u>COMMANDS :-</u>**

- **<u>Create and Use Customer database :-</u>**

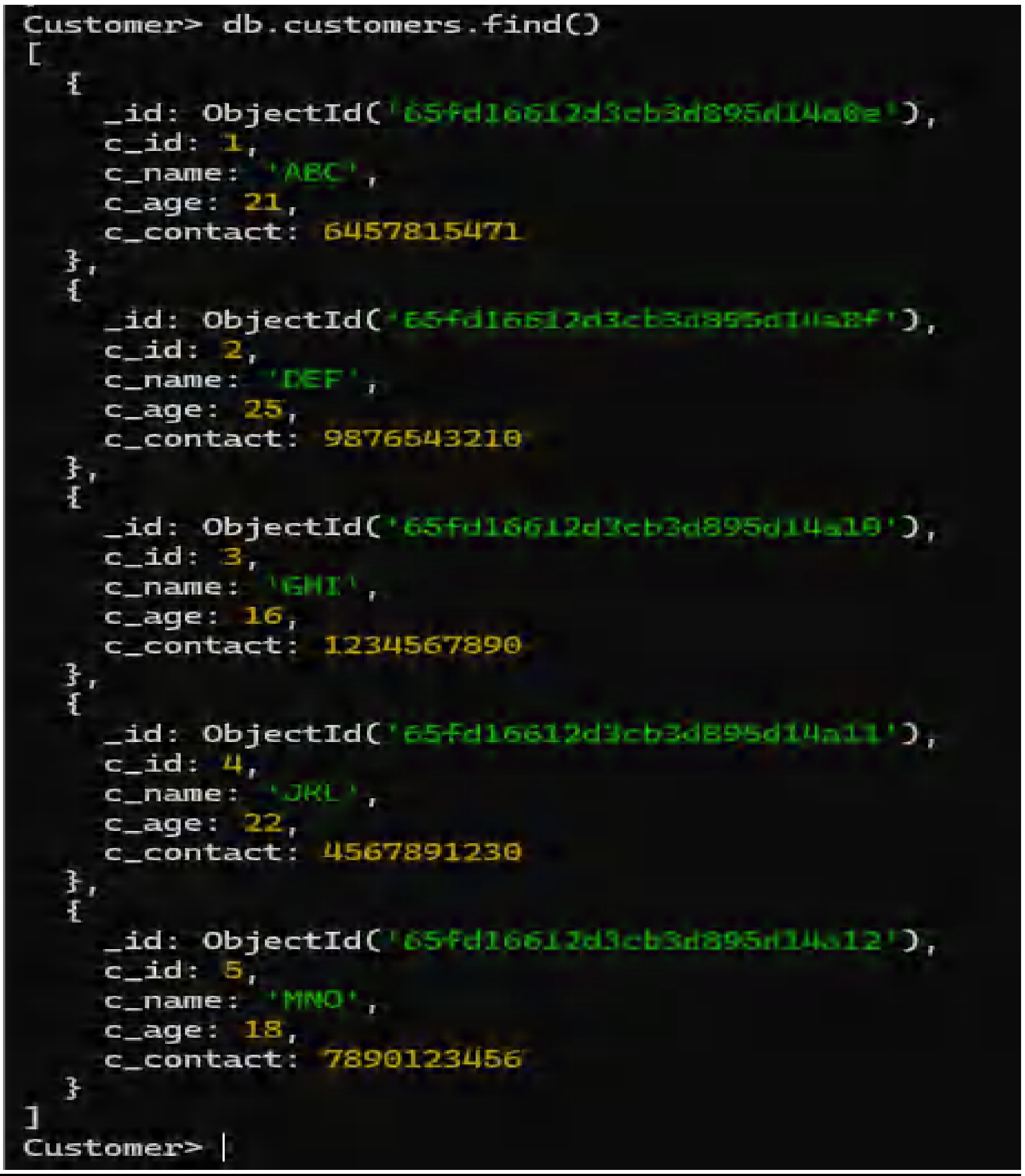
```
test> use Customer
Customer> db.createCollection('customers')
db.customers.insertMany([
  { c_id: 1, c_name: 'ABC', c_age: 21, c_contact: 6457815471 },
  { c_id: 2, c_name: 'DEF', c_age: 25, c_contact: 9876543210 },
  { c_id: 3, c_name: 'GHI', c_age: 16, c_contact: 1234567890 },
  { c_id: 4, c_name: 'JKL', c_age: 22, c_contact: 4567891230 },
  { c_id: 5, c_name: 'MNO', c_age: 18, c_contact: 7890123456 }
]);
```



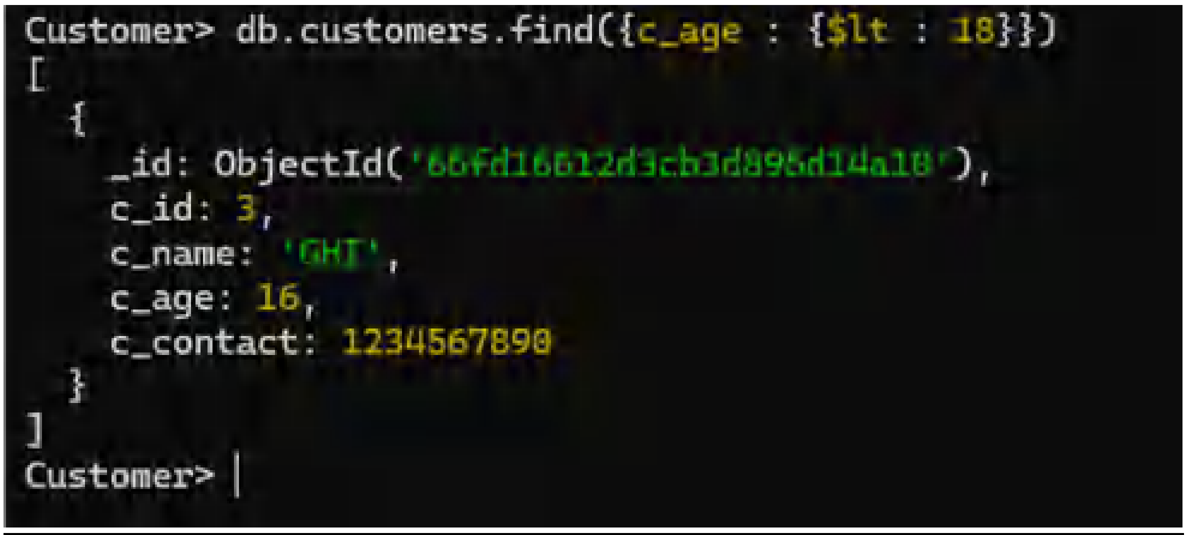
```
test> use Customer
switched to db Customer
Customer> db.createCollection('customers')
{ ok: 1 }
Customer> db.customers.insertMany([
...   { c_id: 1, c_name: 'ABC', c_age: 21, c_contact: 6457815471 },
...   { c_id: 2, c_name: 'DEF', c_age: 25, c_contact: 9876543210 },
...   { c_id: 3, c_name: 'GHI', c_age: 16, c_contact: 1234567890 },
...   { c_id: 4, c_name: 'JKL', c_age: 22, c_contact: 4567891230 },
...   { c_id: 5, c_name: 'MNO', c_age: 18, c_contact: 7890123456 }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65fd16612d3cb3d895d14a0e'),
    '1': ObjectId('65fd16612d3cb3d895d14a0f'),
    '2': ObjectId('65fd16612d3cb3d895d14a10'),
    '3': ObjectId('65fd16612d3cb3d895d14a11'),
    '4': ObjectId('65fd16612d3cb3d895d14a12')
  }
}
Customer> |
```

- **<u>Find() :-</u>**

```
Customer> db.customers.find()

Customer> db.customers.find({c_age : {$lt : 18}})
```



```
Customer> db.customers.find()
[
  {
    _id: ObjectId('65fd16612d3cb3d895d14a0e'),
    c_id: 1,
    c_name: 'ABC',
    c_age: 21,
    c_contact: 6457815471
  },
  {
    _id: ObjectId('65fd16612d3cb3d895d14a0f'),
    c_id: 2,
    c_name: 'DEF',
    c_age: 25,
    c_contact: 9876543210
  },
  {
    _id: ObjectId('65fd16612d3cb3d895d14a10'),
    c_id: 3,
    c_name: 'GHI',
    c_age: 16,
    c_contact: 1234567890
  },
  {
    _id: ObjectId('65fd16612d3cb3d895d14a11'),
    c_id: 4,
    c_name: 'JKL',
    c_age: 22,
    c_contact: 4567891230
  },
  {
    _id: ObjectId('65fd16612d3cb3d895d14a12'),
    c_id: 5,
    c_name: 'MNO',
    c_age: 18,
    c_contact: 7890123456
  }
]
Customer> |
```



```
Customer> db.customers.find({c_age : {$lt : 18}})
[
  {
    _id: ObjectId('65fd16612d3cb3d895d14a10'),
    c_id: 3,
    c_name: 'GHI',
    c_age: 16,
    c_contact: 1234567890
  }
]
Customer> |
```

PRACTICAL NO .4

AIM :- **Create Schema product (product_id, product_name, product_price, product_availability, product_category).**

1. **Create product collection & Insert atleast 5 records.**
2. **Find all the data of product schema.**
3. **Display all the product with price greater than 50.**
4. **Find all products with no. of availability less than 5.**
5. **Find all products with category = 'biscuit'.**
6. **Update all price of product including GST**
7. **Delete all product with category = 'candy'**

❖ INTRODUCTION :-

- The "product" schema encompasses essential attributes such as product ID, name, price, availability, and category, offering a comprehensive representation of product data. Below are the operations conducted on the product collection:

- **Collection Creation & Record Insertion:** Creation of the "product" collection and insertion of a minimum of five records, ensuring a diverse representation of products within the database.

- **Retrieve All Product Data:** Utilization of the find() method to retrieve and present all data stored within the "product" collection, facilitating a comprehensive overview of available products.

- **Display Products with Price > 50:** Identification and display of products with a price exceeding 50, enabling the recognition of higher-priced items within the product catalogue.

- **Find Products with Limited Availability:** Utilization of the find() method to locate products with availability less than 5, aiding in inventory management and highlighting items with restricted availability.

- **Retrieve Products Categorized as 'Biscuit':** Querying the collection to retrieve and present products categorized as 'biscuit', offering insights specific to this product category.

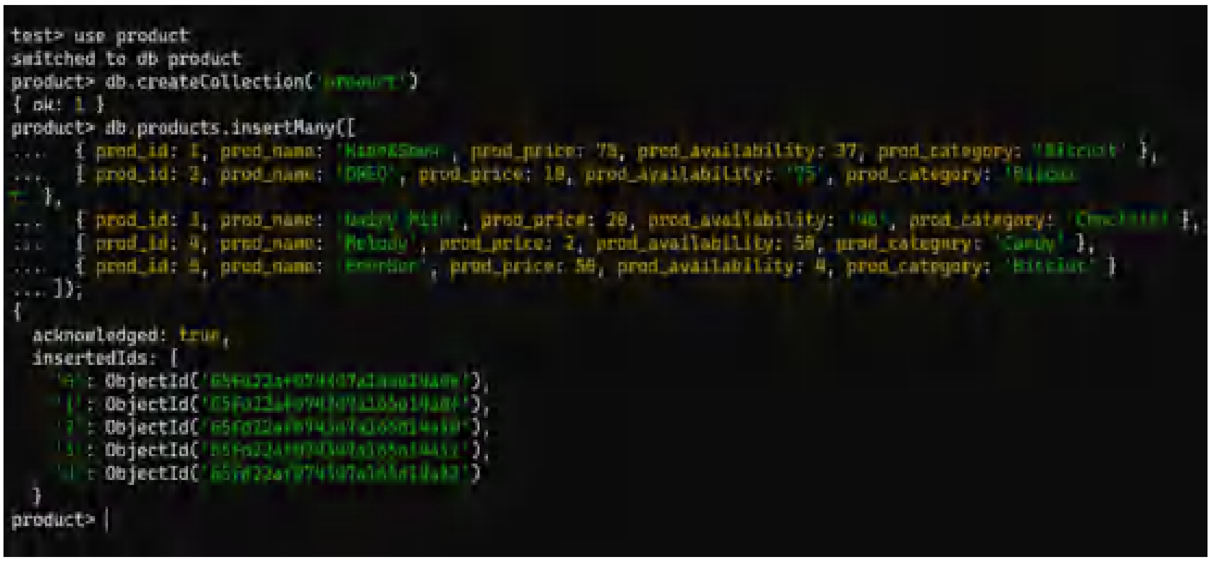
- **Update Product Prices Inclusive of GST:** Execution of an update operation to adjust all product prices to incorporate GST (Goods and Services Tax), ensuring accurate pricing representation across the product range.

- **Delete Products Categorized as 'Candy':** Deletion of all products categorized as 'candy' from the collection, streamlining product management processes and maintaining catalog coherence.

❖ **COMMANDS :-**

- **Create and insert records :-**

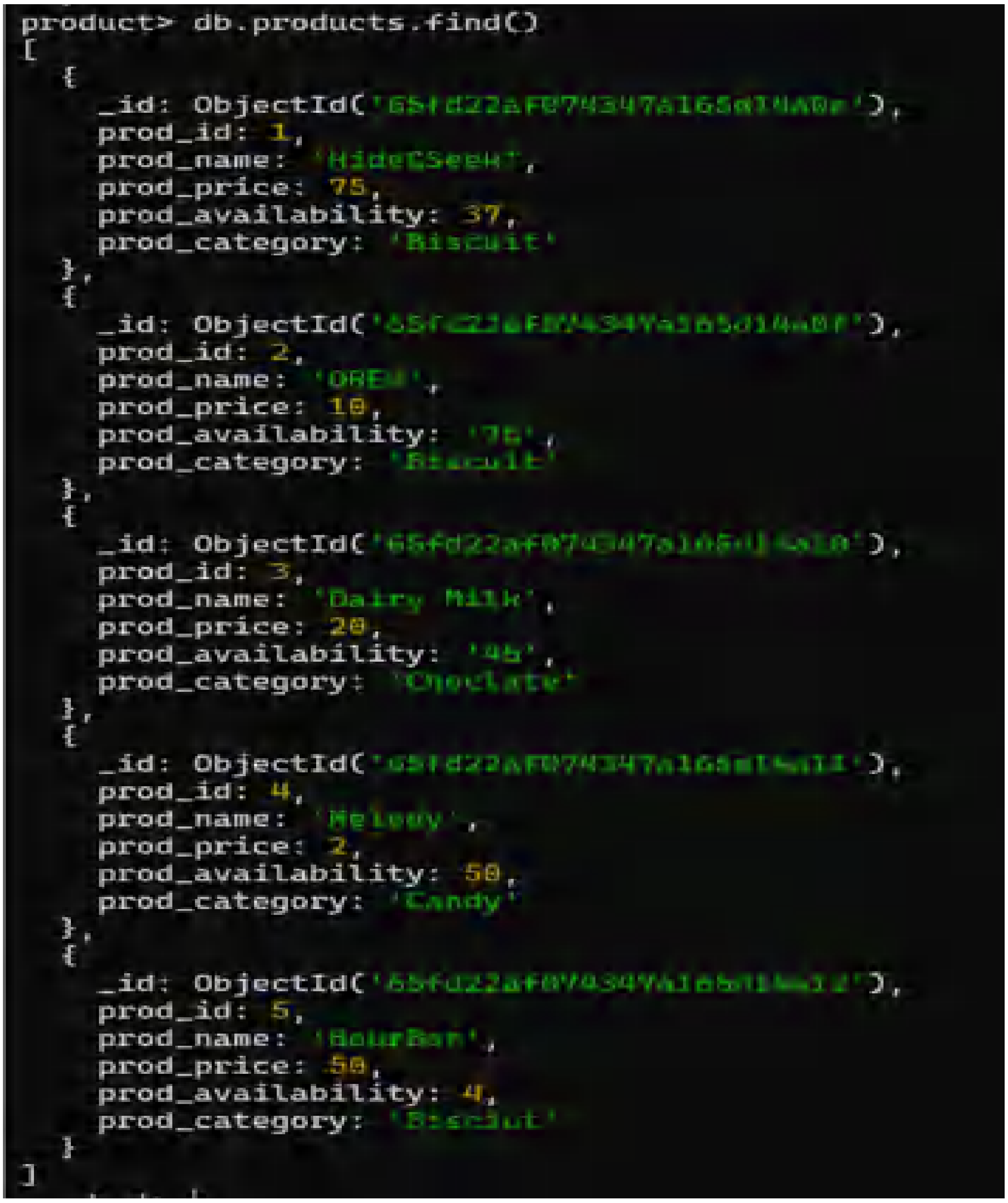
```
test> use product
product> db.createCollection('product')

db.products.insertMany([
  { prod_id: 1, prod_name: 'Hide&Seek', prod_price: 75, prod_availability: 37,
prod_category: 'Biscuit' },
  { prod_id: 2, prod_name: 'OREO', prod_price: 10, prod_availability: '75', prod_category:
'Biscuit' },
  { prod_id: 3, prod_name: 'Dairy Milk', prod_price: 20, prod_availability: '45',
prod_category: 'Choclate' },
  { prod_id: 4, prod_name: 'Melody', prod_price: 2, prod_availability: 50, prod_category:
'Candy' },
  { prod_id: 5, prod_name: 'BourBon', prod_price: 50, prod_availability: 4, prod_category:
'Bisciut' }
]);
```



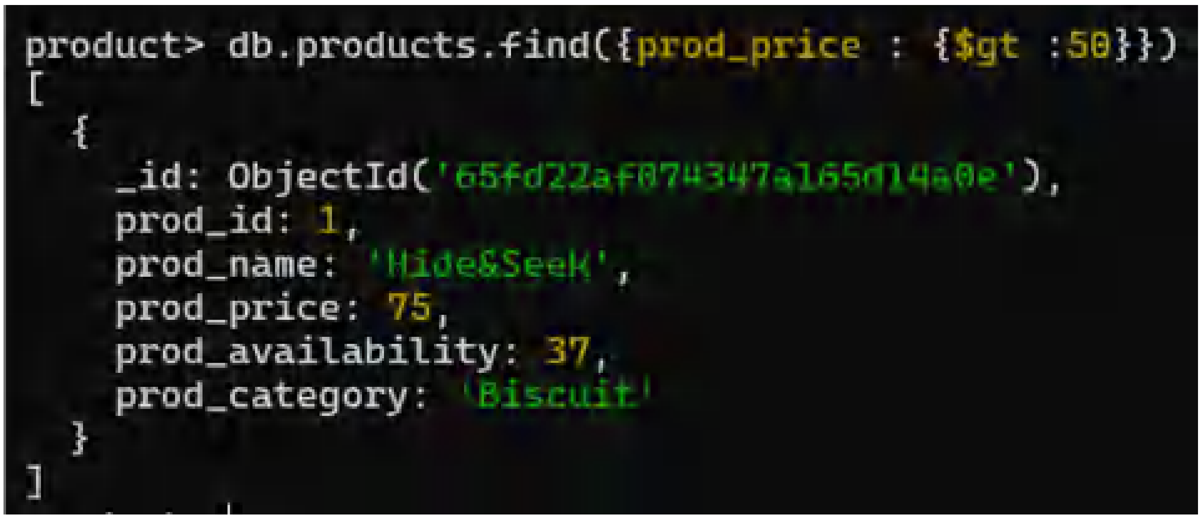
```
test> use product
switched to db product
product> db.createCollection('product')
{ ok: 1 }
product> db.products.insertMany([
...   { prod_id: 1, prod_name: 'Hide&Seek', prod_price: 75, prod_availability: 37, prod_category: 'Biscuit' },
...   { prod_id: 2, prod_name: 'OREO', prod_price: 10, prod_availability: '75', prod_category: 'Biscui
t' },
...   { prod_id: 3, prod_name: 'Dairy Milk', prod_price: 20, prod_availability: '45', prod_category: 'Choclate' },
...   { prod_id: 4, prod_name: 'Melody', prod_price: 2, prod_availability: 50, prod_category: 'Candy' },
...   { prod_id: 5, prod_name: 'BourBon', prod_price: 50, prod_availability: 4, prod_category: 'Bisciut' }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('[illegible]'),
    '1': ObjectId('[illegible]'),
    '2': ObjectId('[illegible]'),
    '3': ObjectId('[illegible]'),
    '4': ObjectId('[illegible]')
  }
}
product> |
```

- **Find () :-**

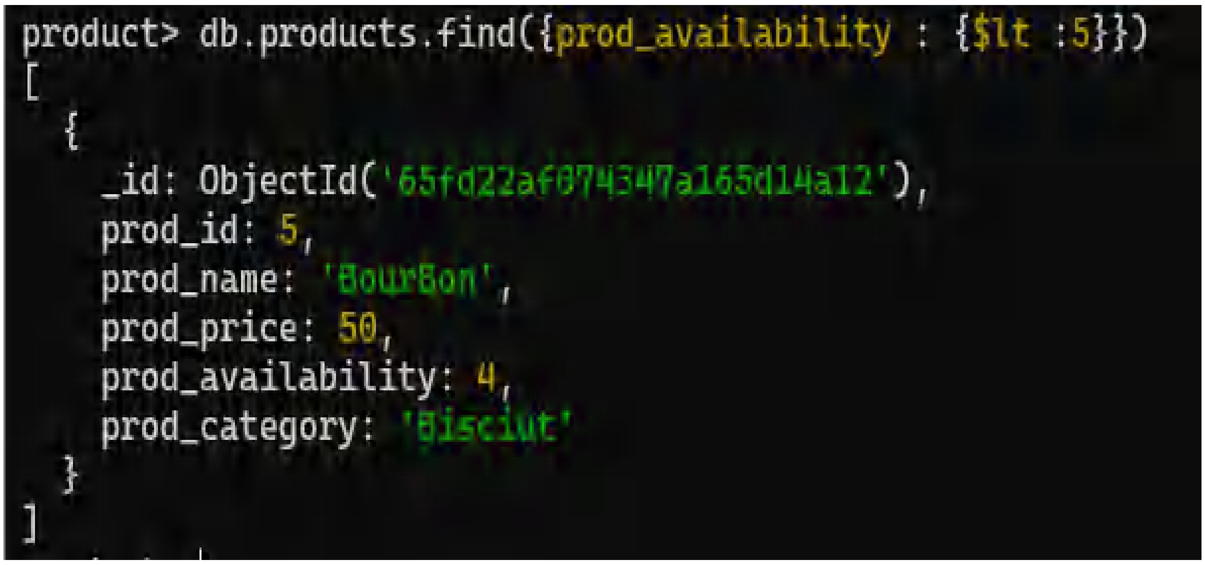
```
product> db.products.find()
product> db.products.find({prod_price : {$gt :50}})
product> db.products.find({prod_availability : {$lt :5}})
product> db.products.find({prod_category : 'Biscuit'})
```



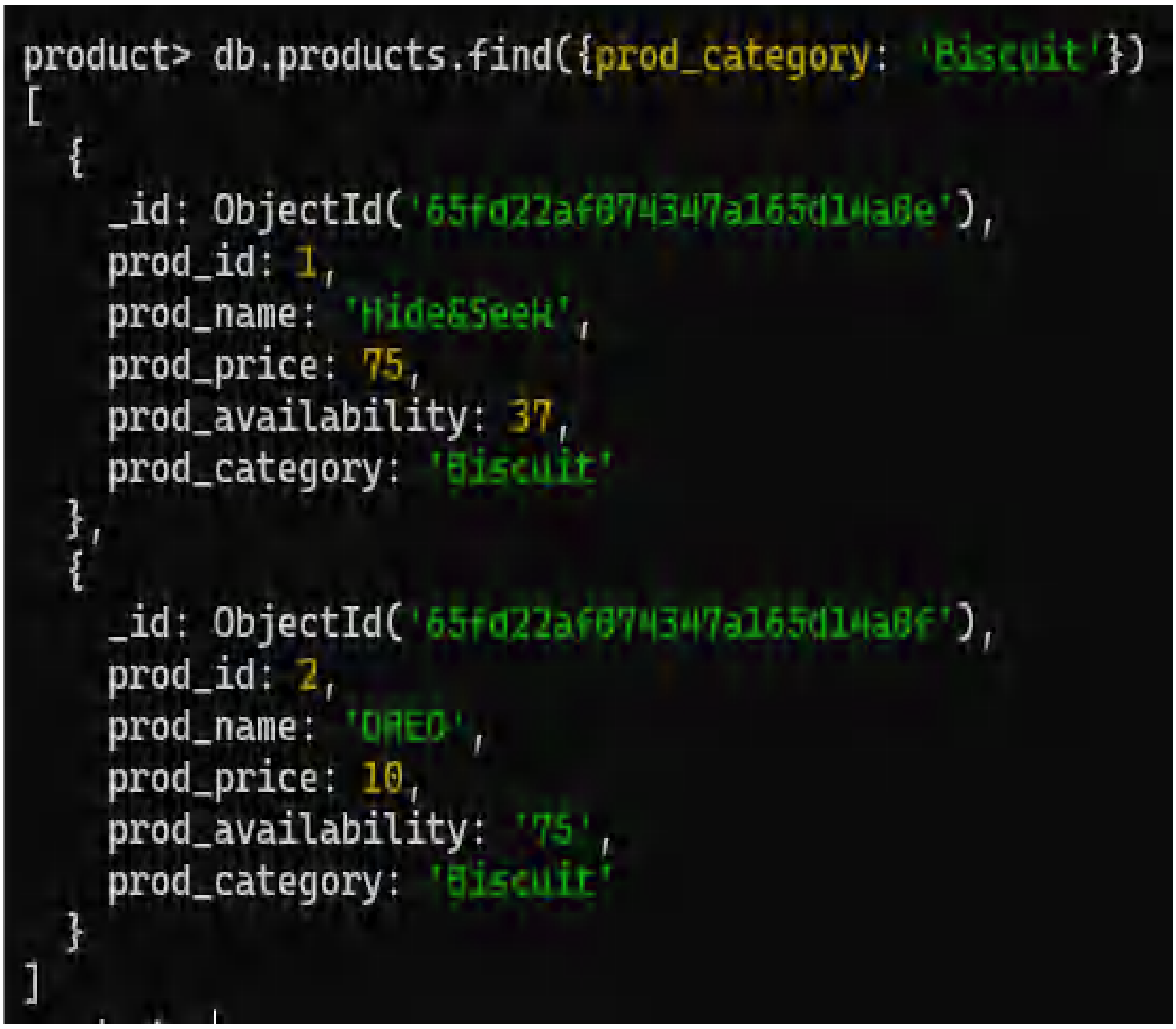
```
product> db.products.find()
[
  {
    _id: ObjectId('65fd22af074347a165d14a0e'),
    prod_id: 1,
    prod_name: 'Hide&Seek',
    prod_price: 75,
    prod_availability: 37,
    prod_category: 'Biscuit'
  },
  {
    _id: ObjectId('65fd22af074347a165d14a0f'),
    prod_id: 2,
    prod_name: 'OREO',
    prod_price: 10,
    prod_availability: '75',
    prod_category: 'Biscuit'
  },
  {
    _id: ObjectId('65fd22af074347a165d14a10'),
    prod_id: 3,
    prod_name: 'Dairy Milk',
    prod_price: 20,
    prod_availability: '45',
    prod_category: 'Choclate'
  },
  {
    _id: ObjectId('65fd22af074347a165d14a11'),
    prod_id: 4,
    prod_name: 'Melody',
    prod_price: 2,
    prod_availability: 50,
    prod_category: 'Candy'
  },
  {
    _id: ObjectId('65fd22af074347a165d14a12'),
    prod_id: 5,
    prod_name: 'BourBon',
    prod_price: 50,
    prod_availability: 4,
    prod_category: 'Biscuit'
  }
]
```



```
product> db.products.find({prod_price : {$gt :50}})
[
  {
    _id: ObjectId('65fd22af074347a165d14a0e'),
    prod_id: 1,
    prod_name: 'Hide&Seek',
    prod_price: 75,
    prod_availability: 37,
    prod_category: 'Biscuit'
  }
]
```



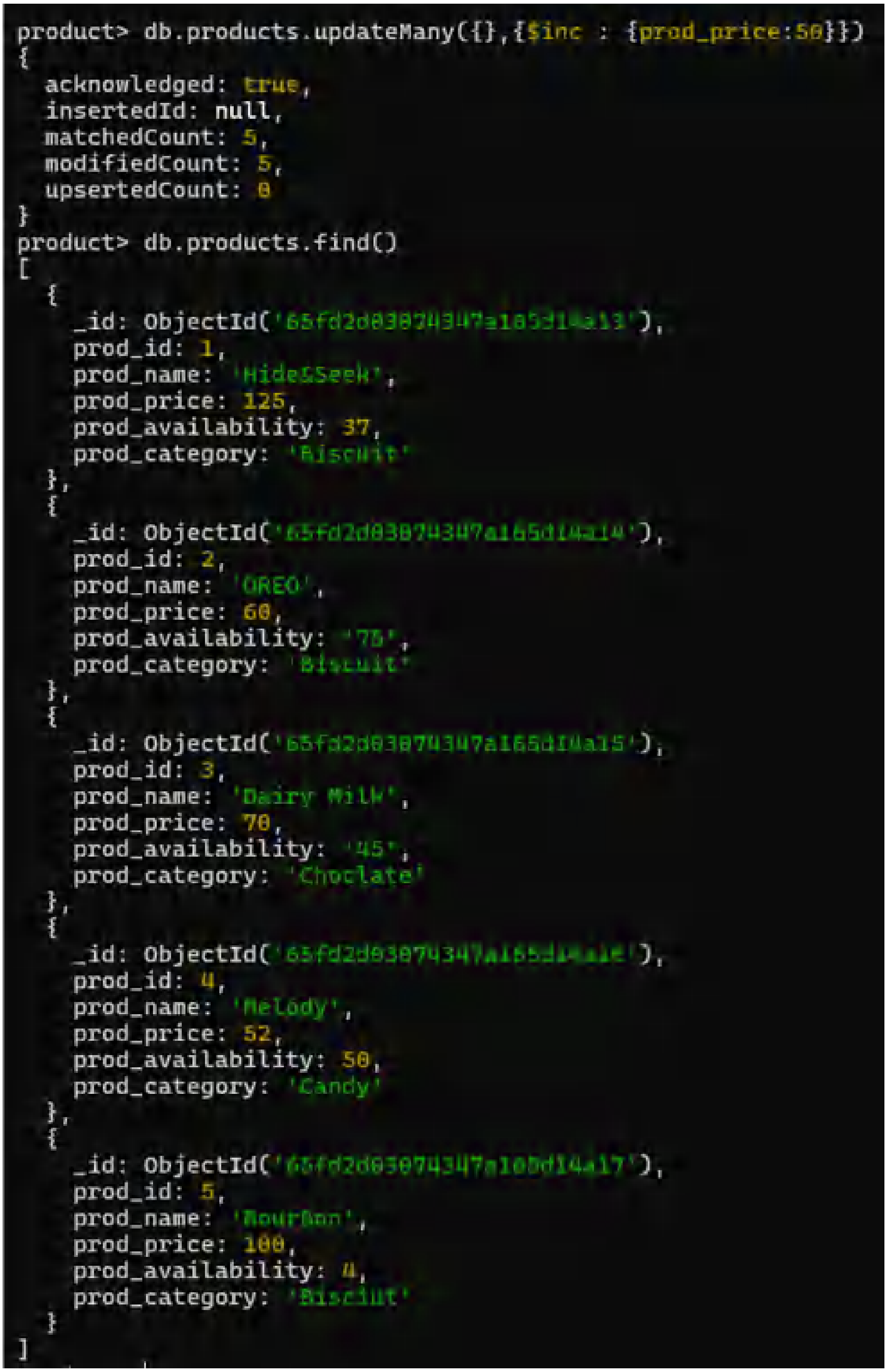
```
product> db.products.find({prod_availability : {$lt :5}})
[
  {
    _id: ObjectId('65fd22af074347a165d14a12'),
    prod_id: 5,
    prod_name: 'BourBon',
    prod_price: 50,
    prod_availability: 4,
    prod_category: 'Biscuit'
  }
]
```



```
product> db.products.find({prod_category: 'Biscuit'})
[
  {
    _id: ObjectId('65fd22af074347a165d14a0e'),
    prod_id: 1,
    prod_name: 'Hide&Seek',
    prod_price: 75,
    prod_availability: 37,
    prod_category: 'Biscuit'
  },
  {
    _id: ObjectId('65fd22af074347a165d14a0f'),
    prod_id: 2,
    prod_name: 'OREO',
    prod_price: 10,
    prod_availability: '75',
    prod_category: 'Biscuit'
  }
]
```

- **Update() :**

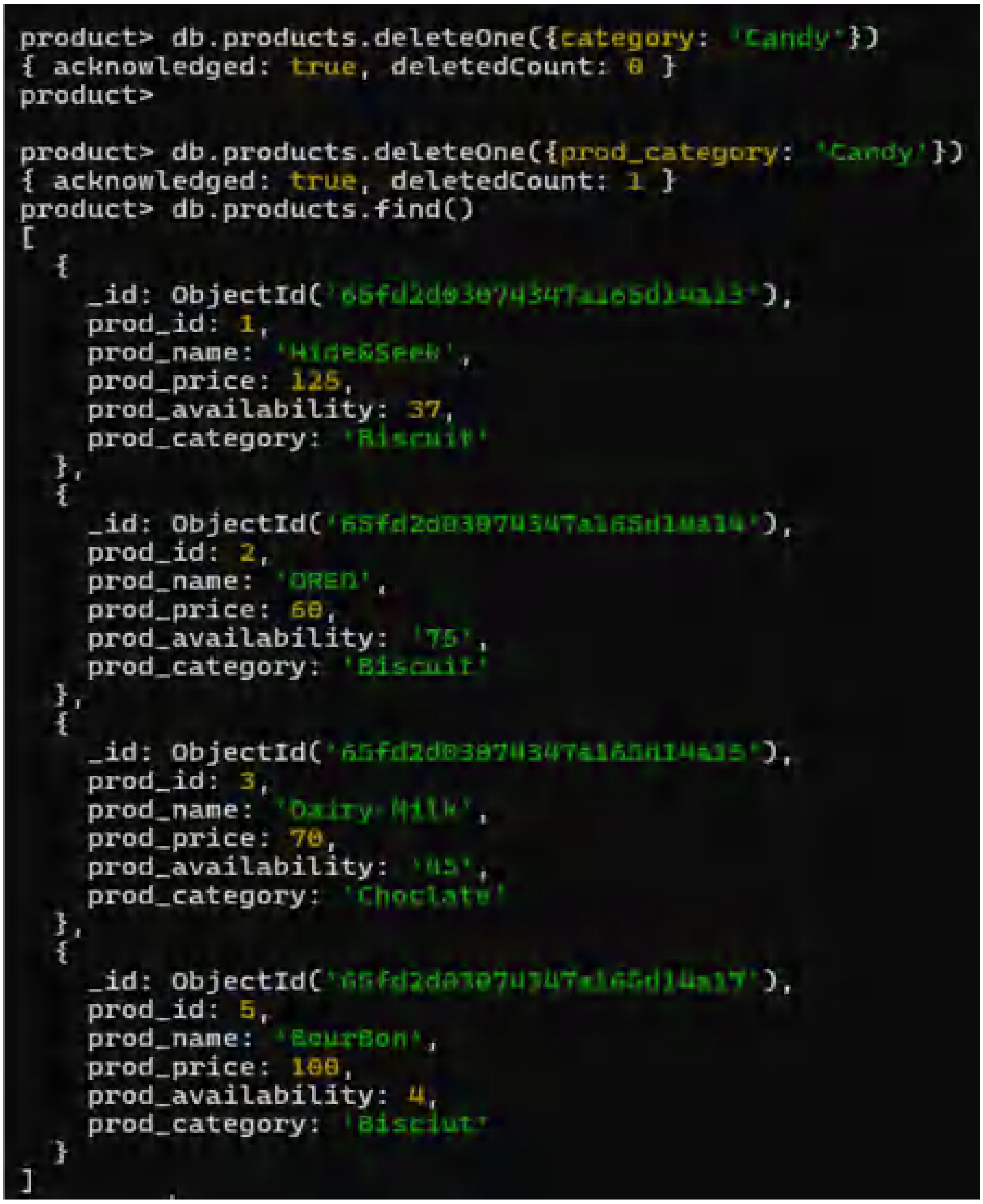
product> db.products.updateMany({},{$inc : {prod_price:50}})



```
product> db.products.updateMany({},{$inc : {prod_price:50}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
product> db.products.find()
[
  {
    _id: ObjectId('65fd2d03874347a165d14a13'),
    prod_id: 1,
    prod_name: 'Hide&Seek',
    prod_price: 125,
    prod_availability: 37,
    prod_category: 'Biscuit'
  },
  {
    _id: ObjectId('65fd2d03874347a165d14a14'),
    prod_id: 2,
    prod_name: 'OREO',
    prod_price: 60,
    prod_availability: '75',
    prod_category: 'Biscuit'
  },
  {
    _id: ObjectId('65fd2d03874347a165d14a15'),
    prod_id: 3,
    prod_name: 'Dairy Milk',
    prod_price: 70,
    prod_availability: '45',
    prod_category: 'Choclate'
  },
  {
    _id: ObjectId('65fd2d03874347a165d14a16'),
    prod_id: 4,
    prod_name: 'Melody',
    prod_price: 52,
    prod_availability: 50,
    prod_category: 'Candy'
  },
  {
    _id: ObjectId('65fd2d03874347a165d14a17'),
    prod_id: 5,
    prod_name: 'Bourbon',
    prod_price: 100,
    prod_availability: 4,
    prod_category: 'Biscuit'
  }
]
```

- **<u>Delete :-</u>**

product> db.products.deleteOne({prod_category: 'Candy'})



```
product> db.products.deleteOne({category: 'Candy'})
{ acknowledged: true, deletedCount: 0 }
product>

product> db.products.deleteOne({prod_category: 'Candy'})
{ acknowledged: true, deletedCount: 1 }
product> db.products.find()
[
  {
    _id: ObjectId('65fd2d03874347a165d14a13'),
    prod_id: 1,
    prod_name: 'Hide&Seek',
    prod_price: 125,
    prod_availability: 37,
    prod_category: 'Biscuit'
  },
  {
    _id: ObjectId('65fd2d03874347a165d14a14'),
    prod_id: 2,
    prod_name: 'OREO',
    prod_price: 60,
    prod_availability: '75',
    prod_category: 'Biscuit'
  },
  {
    _id: ObjectId('65fd2d03874347a165d14a15'),
    prod_id: 3,
    prod_name: 'Dairy Milk',
    prod_price: 70,
    prod_availability: '45',
    prod_category: 'Choclate'
  },
  {
    _id: ObjectId('65fd2d03874347a165d14a17'),
    prod_id: 5,
    prod_name: 'BourBon',
    prod_price: 100,
    prod_availability: 4,
    prod_category: 'Bisciut'
  }
]
```

PRACTICAL NO .5

AIM :- Write a program to perform validation of a form using AngularJS

❖ INTRODUCTION :-

AngularJS, a Google-maintained JavaScript framework, simplifies the process of building dynamic web applications by providing powerful features such as two-way data binding and directives. One common task in web development is form validation, ensuring that users provide valid information before submitting data. AngularJS makes form validation straightforward with its built-in directives and features. In this tutorial, we'll explore AngularJS's built-in validation directives and error handling strategies, empowering you to create robust web forms effortlessly.

- **HTML Structure:**
  <!DOCTYPE html> declaration specifies the document type and version of HTML.
  <head> section contains the document title and includes the AngularJS library via a CDN link.
  Inline styles define font family and color for the heading.

- **AngularJS Initialization:**
  ng-app="" directive bootstraps an AngularJS application, initialized in the entire document.

- **Form Setup:**
  Inside <body>, a heading displays "GeeksforGeeks" and a subheading indicates "AngularJS Form Validation".
  <form> element is defined with the name attribute "form1" to reference the form in AngularJS.

- **Form Fields:**
  Two input fields exist within the form: one for "Name" and another for "Address".
  Each input field has a name attribute for unique identification within the form.
  ng-model directive binds input fields to AngularJS scope variables (username and useraddress).
  The required attribute is added to make both fields mandatory.

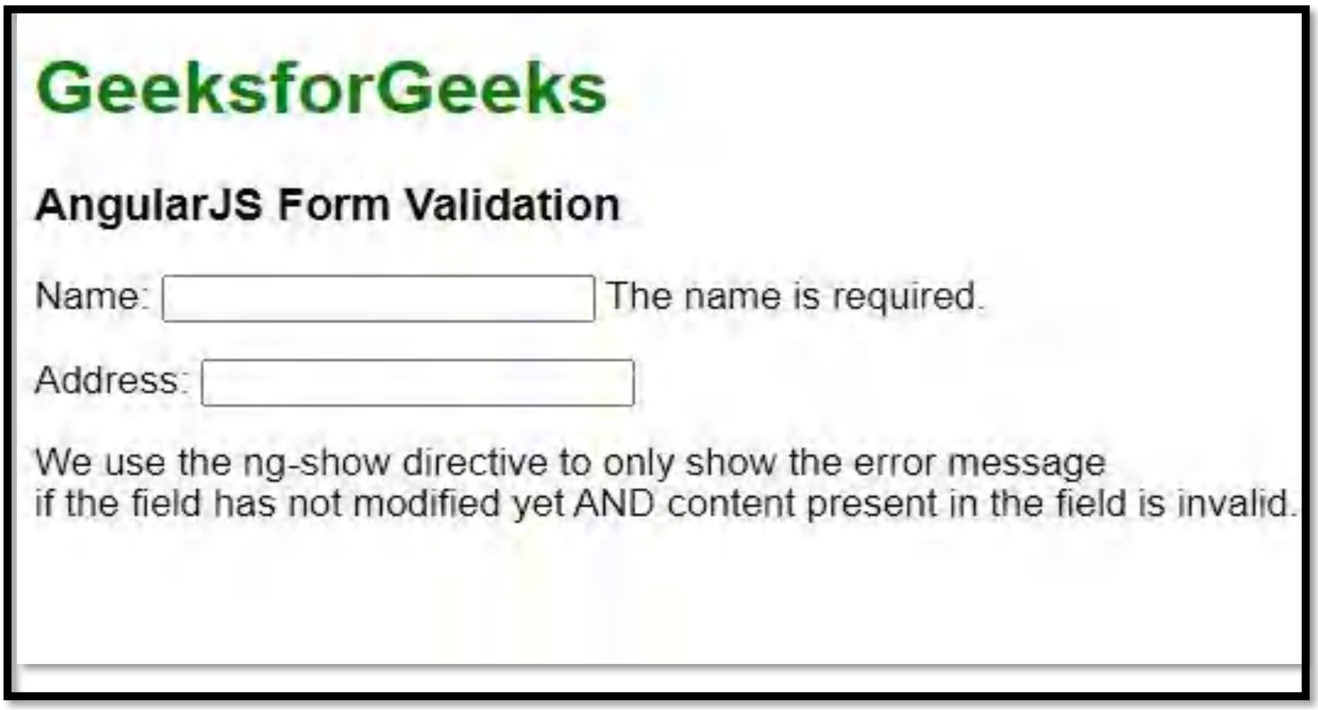
- **Error Display:**
  ng-show directive conditionally displays error messages.
  For the "Name" field, the error message appears if the field is pristine and invalid.

- **Information Message:**
  A paragraph below the form explains how the ng-show directive displays error messages only under specific conditions.

- **CODE :-**

```
<!DOCTYPE html>
<html>

<head>
      <title>AngularJS Form Validation</title>
    <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
    </script>
    <style>
        body {
            font-family: Arial, Helvetica, sans-serif;
        }
        h1 {
            color: green;
        }
    </style>
</head>

<body ng-app="">
    <h1>GeeksforGeeks</h1>
    <h3>AngularJS Form Validation</h3>
    <form name="form1">
        <p>Name:
            <input name="username"
                   ng-model="username" required>
            <span ng-show=
"form1.username.$pristine && form1.username.$invalid">
                The name is required.</span>
        </p>
        <p>Address:
            <input name="useraddress"
                   ng-model="useraddress" required>
        </p>
    </form>
    <p>
        We use the ng-show directive to only
        show the error message <br>if the field
          has not modified yet AND content present
          in the field is invalid.
    </p>
</body>
</html>
```

- **OUTPUT :-**



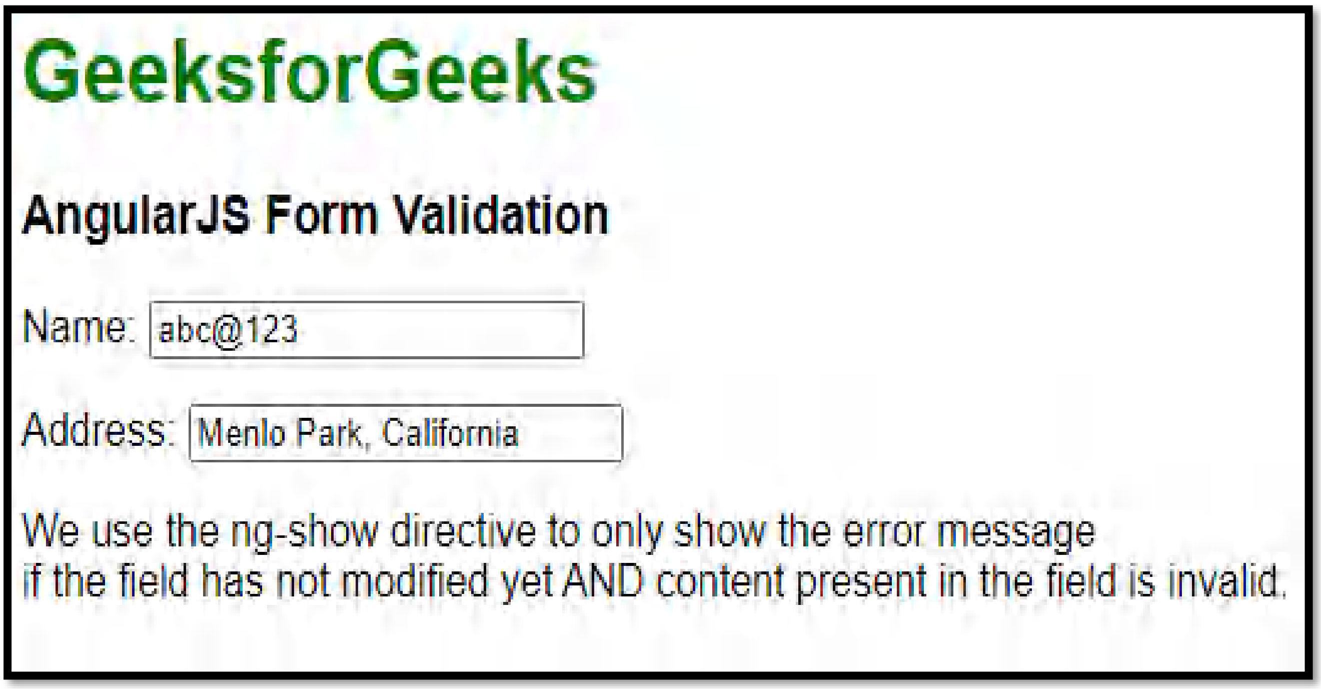
GeksforGeeks

AngularJS Form Validation

Name: [ ] The name is required.

Address: [ ]

We use the ng-show directive to only show the error message
if the field has not modified yet AND content present in the field is invalid.



GeeksforGeeks

AngularJS Form Validation

Name: abc@123

Address: Menlo Park, California

We use the ng-show directive to only show the error message
if the field has not modified yet AND content present in the field is invalid.

PRACTICAL NO .6

AIM :- **Write a program to create and implement modules and controllers in Angular JS**

❖ INTRODUCTION :-

- **Modules in AngularJS:**
  Definition:
  Modules are containers for different parts of an AngularJS application, such as controllers, services, directives, etc.

  Key Features:
  Provide a way to organize and manage application components.
  Support dependency injection.
  Facilitate modularity and code encapsulation.

- **Controllers in AngularJS:**
  Definition:
  Controllers are responsible for handling application logic and data binding in AngularJS.

  Key Features:
  Bind data to the view using the scope object.
  Encapsulate business logic and user interactions.
  Promote separation of concerns within the application.

- **Benefits:**
  Modularity: Modules help in organizing code and promoting code reuse.
  Testability: Controllers encapsulate logic, making it easier to test individual components.
  Code Organization: Provides a structured way to organize code, improving readability and maintainability.

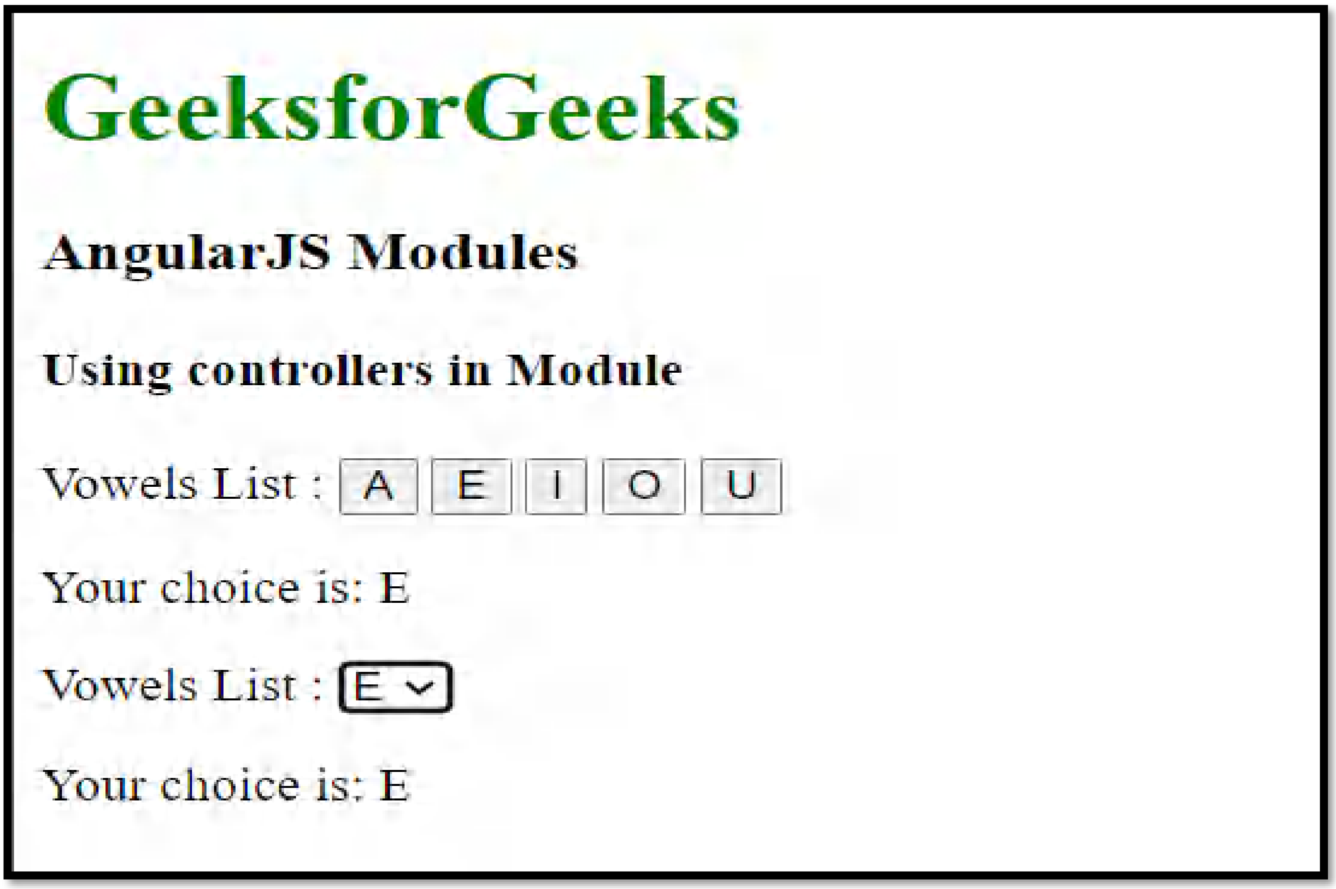
- **CODE :-**
  - **DemoController.js :**

```
var app = angular.module('DemoApp',[]);
app.controller('DemoController', function ($scope) {
    $scope.list = ['A', 'E', 'I', 'O', 'U'];
    $scope.choice = 'Your choice is: GeeksforGeeks';
    $scope.ch = function (choice) {
        $scope.choice = "Your choice is: " + choice;
    };
    $scope.c = function () {
        $scope.choice = "Your choice is: " + $scope.mychoice;
    };
});
```

  - **Index.html:-**

```
<!DOCTYPE html>
<html>
<head>
    <title>
        Modules and Controllers in Files
    </title>
    <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
    </script>
    <script src="DemoApp.js"></script>
    <script src="DemoController.js"></script>
</head>
<body ng-app="DemoApp">
    <h1 style="color:green">GeeksforGeeks</h1>
    <h3>AngularJS Modules</h3>
    <h4>
        Using controllers in Module
    </h4>
    <div ng-app="DemoApp"
         ng-controller="DemoController">

            Vowels List :
        <button ng-click="ch('A')">A</button>
        <button ng-click="ch('E')">E</button>
        <button ng-click="ch('I')">I</button>
        <button ng-click="ch('O')">O</button>
        <button ng-click="ch('U')">U</button>
        <p>{{ choice }}</p>

        Vowels List :
        <select ng-options="option for option in list"
                ng-model="mychoice"
                ng-change="c()">
        </select>
        <p>{{ choice }}</p>
    </div>
</body>
</html>
```

- **OUTPUT :-**



GeeksforGeeks

AngularJS Modules

Using controllers in Module

Vowels List : A E I O U

Your choice is: E

Vowels List : E

Your choice is: E

PRACTICAL NO .7

AIM :- Write a program to perform Email Validation using AngularJS

- **INTRODUCTION :-**

  Email validation is a crucial aspect of web development, ensuring that users provide accurate email addresses when submitting forms. AngularJS, a powerful JavaScript framework maintained by Google, simplifies the process of email validation through its built-in directives and features.

  In this tutorial, we will delve into AngularJS's email validation capabilities by creating a simple program. We'll design a web form that includes an email input field and utilize AngularJS directives to validate the entered email address. Through this example, we'll explore how AngularJS streamlines the validation process and enhances user experience by providing immediate feedback on invalid inputs.

  - **Tags Used:**
    <form>: Defines the form element.
    <input>: Defines input fields.
    <span>: Used to display error messages.

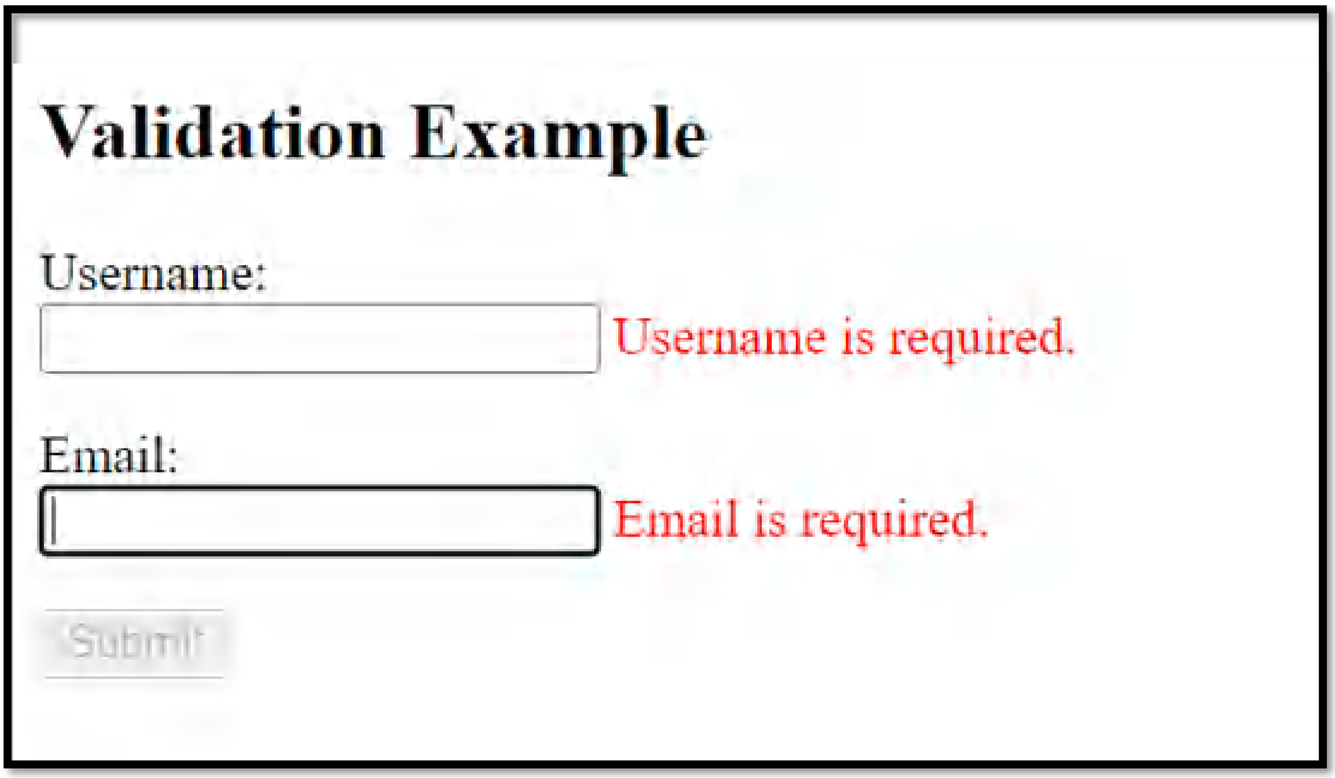
  - **AngularJS Directives:**
    ng-app="myApp": Defines the AngularJS application module.
    ng-controller="validateCtrl": Associates the controller with the form.
    ng-model: Binds input fields to AngularJS scope variables.
    ng-show: Conditionally shows error messages based on validation status.
    ng-disabled: Disables the submit button based on form validity.

  - **Function Name:**
    validateCtrl: The AngularJS controller function responsible for initializing scope variables and handling form logic.

- **CODE :-**

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<h2>Validation Example</h2>

<form  ng-app="myApp"  ng-controller="validateCtrl"
name="myForm" novalidate>

<p>Username:<br>
  <input type="text" name="user" ng-model="user" required>
  <span style="color:red" ng-show="myForm.user.$dirty && myForm.user.$invalid">
  <span ng-show="myForm.user.$error.required">Username is required.</span>
  </span>
</p>

<p>Email:<br>
  <input type="email" name="email" ng-model="email" required>
  <span style="color:red" ng-show="myForm.email.$dirty && myForm.email.$invalid">
  <span ng-show="myForm.email.$error.required">Email is required.</span>
  <span ng-show="myForm.email.$error.email">Invalid email address.</span>
  </span>
</p>

<p>
  <input type="submit"
  ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||
  myForm.email.$dirty && myForm.email.$invalid">
</p>

</form>

<script>
var app = angular.module('myApp', []);
app.controller('validateCtrl', function($scope) {
  $scope.user = 'John Doe';
  $scope.email = 'john.doe@gmail.com';
});
</script>

</body>
</html>
```

- **OUTPUT :-**



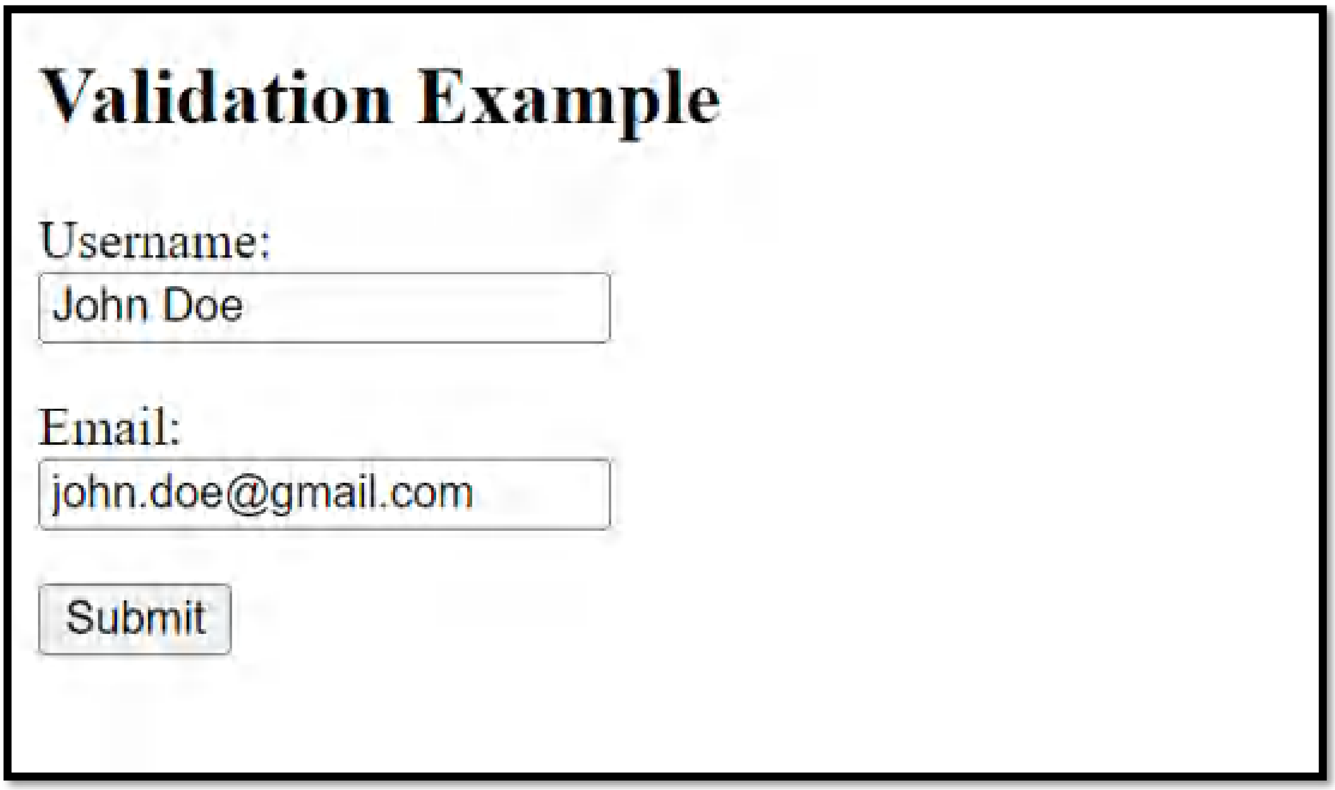
Validation Example

Username:

Username is required.

Email:

Email is required.



Validation Example

Username:

John Doe

Email:

john.doe@gmail.com

PRACTICAL NO .8

AIM :- Write a program to implement Error Handling in Angular JS

- **INTRODUCTION :-**

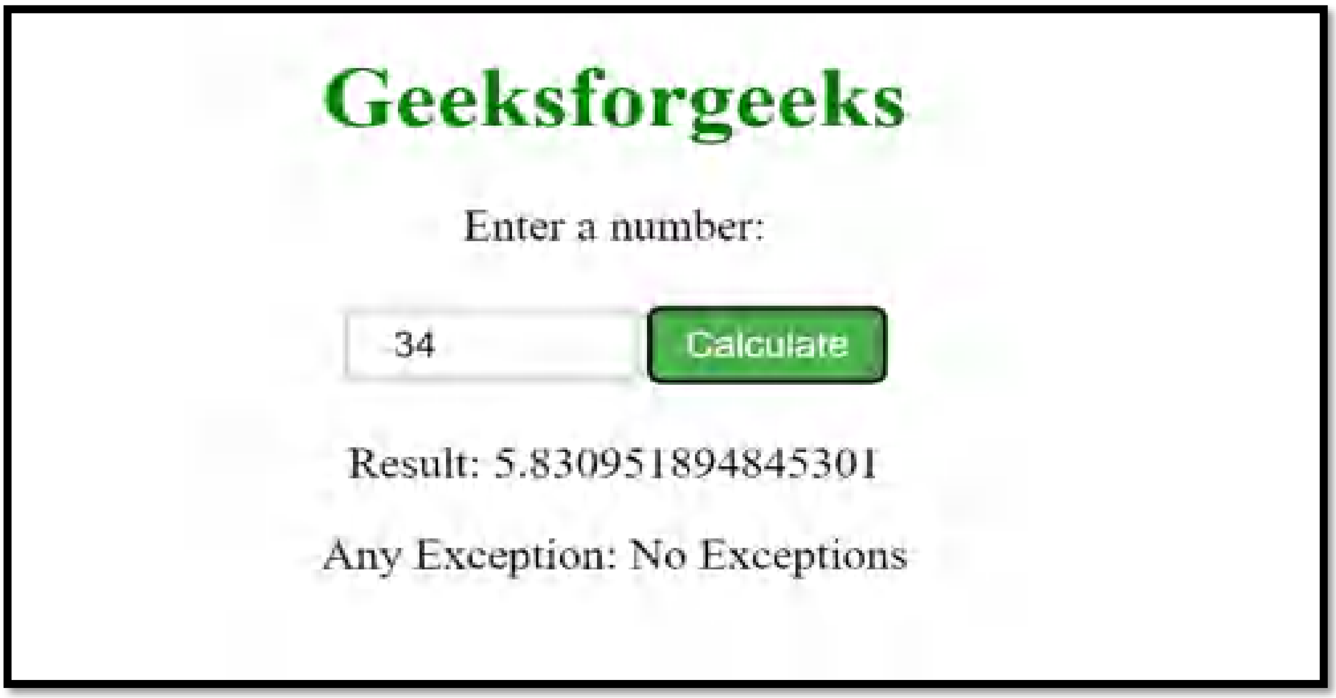
Error handling is a crucial aspect of any application development, including AngularJS. It involves identifying, reporting, and managing errors that occur during the execution of an application. Proper error handling ensures that users receive informative error messages and helps developers diagnose and fix issues quickly. In AngularJS, error handling can be implemented using various techniques and mechanisms provided by the framework.

- The AngularJS framework is utilized to create an application with a single controller (myCtrl) defined within the myApp module.
- The controller contains a function (calculate()) triggered by a button click. This function attempts to calculate the square root of a user-inputted number.
- Inside the calculate() function, there's a try-catch block to handle potential exceptions. If the user enters a negative number, an exception is thrown with an appropriate error message.
- The $exceptionHandler service provided by AngularJS is used to log and handle exceptions globally within the application.
- The HTML template includes an input field for entering a number, a button to trigger the calculation, and placeholders to display the result and any exceptions that occur during the process.

- **CODE :-**

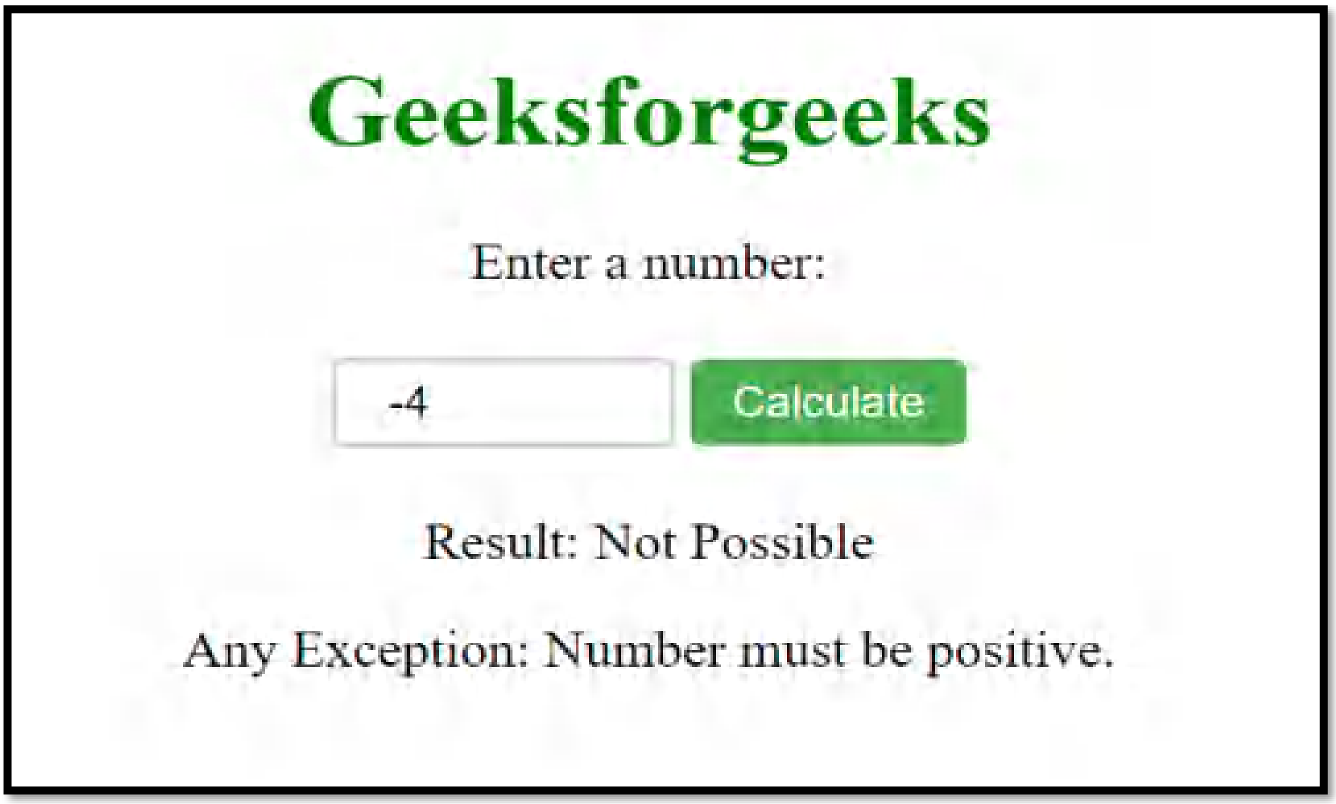
```
<!DOCTYPE html>
<html>
<head>
  <title>Exception Handling in AngularJS</title>
  <style>
    input {
      width: 100px;
      padding: 5px 15px;
      margin: 5px 0;
      box-sizing: border-box;
      border: 2px solid #ccc;
      border-radius: 4px;
    }
    button {
      width: 80px;
      background-color: #4caf50;
      color: white;
      padding: 6px 12px;
      margin: 5px 0;
      border: none;
      border-radius: 4px;
      cursor: pointer;
    }
    h1 {
      color: green;
    }
  </style>
</head>
<body ng-app="myApp">
  <center>
    <h1> Geeksforgeeks</h1>
  <div ng-controller="myCtrl">
    <p>Enter a number:</p>
    <input type="number" ng-model="num" />
    <button ng-click="calculate()">Calculate</button>
    <p>Result: {{result}}</p>
    <p>Any Exception: {{errorMessage}}</p>
  </div>
  </center>
  <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
  </script>
  <script>
    angular.module('myApp', [])
      .controller('myCtrl', function ($scope, $exceptionHandler) {
        $scope.calculate = function () {
          try {
            if ($scope.num < 0) {
              $scope.result = "Not Possible";
              throw new Error("Number must be positive.");
            }
            $scope.result = Math.sqrt($scope.num);
            $scope.errorMessage = "No Exceptions";
          } catch (e) {
            $scope.errorMessage = e.message;
            $exceptionHandler(e);
          }
        }
      });
  </script>
</body>
</html>
```

- **OUTPUT:-**



Result: 5.830951894845301

Any Exception: No Exceptions



Result: Not Possible

Any Exception: Number must be positive.

PRACTICAL NO .9

AIM :- Create a simple HTML —Hello World‖ Project using AngularJS Framework and applyng-controller, ng-model and expressions

- **INTRODUCTION :-**

  Angular CLI (Command Line Interface) is a powerful tool provided by the Angular team for simplifying the process of creating, building, and maintaining Angular applications. It offers a set of commands to streamline various tasks such as generating components, services, modules, etc., as well as managing dependencies and running development servers..

  - **Installing Angular CLI:**

    To install Angular CLI globally on your system, you can use npm (Node Package Manager), which comes bundled with Node.js. Open your terminal or command prompt and run the following command:

    >npm install -g @angular/cli

    This command installs Angular CLI globally on your system, allowing you to use it from any directory in your terminal.

  - **Creating a New Angular App:**

    After installing Angular CLI, you can create a new Angular application by running the following command:

    >ng new my-app

    ng new my-app: This command creates a new Angular application named my-app in the current directory. Angular CLI sets up the project structure, installs necessary dependencies, and generates initial files required for an Angular application.

- **Set-ExecutionPolicy command :**

  `Set-ExecutionPolicy`: Command used to set the execution policy.

  `-Scope CurrentUser`: Specifies that the execution policy applies only to the current user.

  `-ExecutionPolicy RemoteSigned`: Sets the execution policy to

- **COMMANDS :-**

  C:\Users\wwwjs>npm install -g @angular/cli

  C:\Users\wwwjs>ng new my-app

  PS E:\angular> Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned

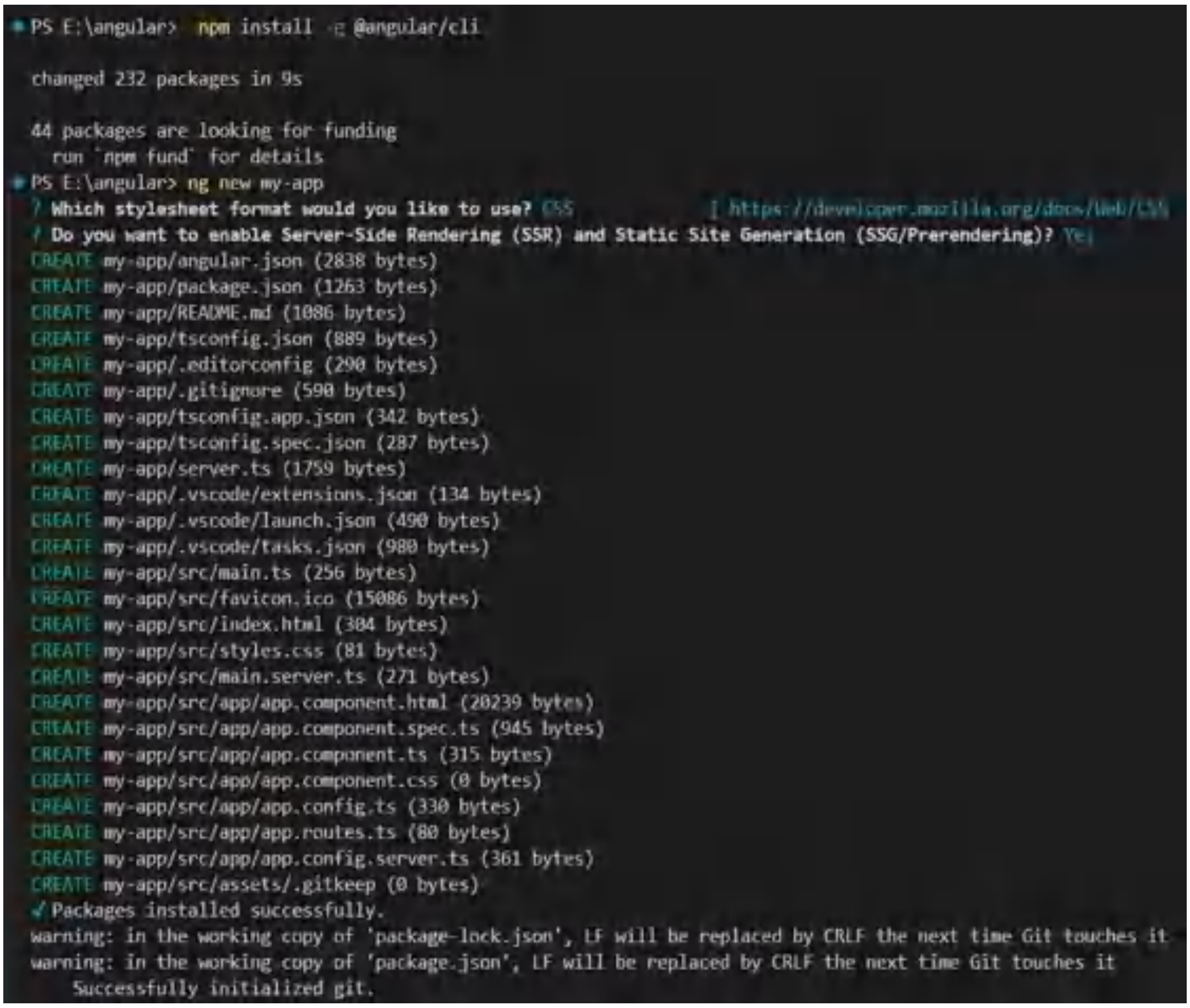
  >> cd my-app

  PS E:\angular\my-app> ng serve

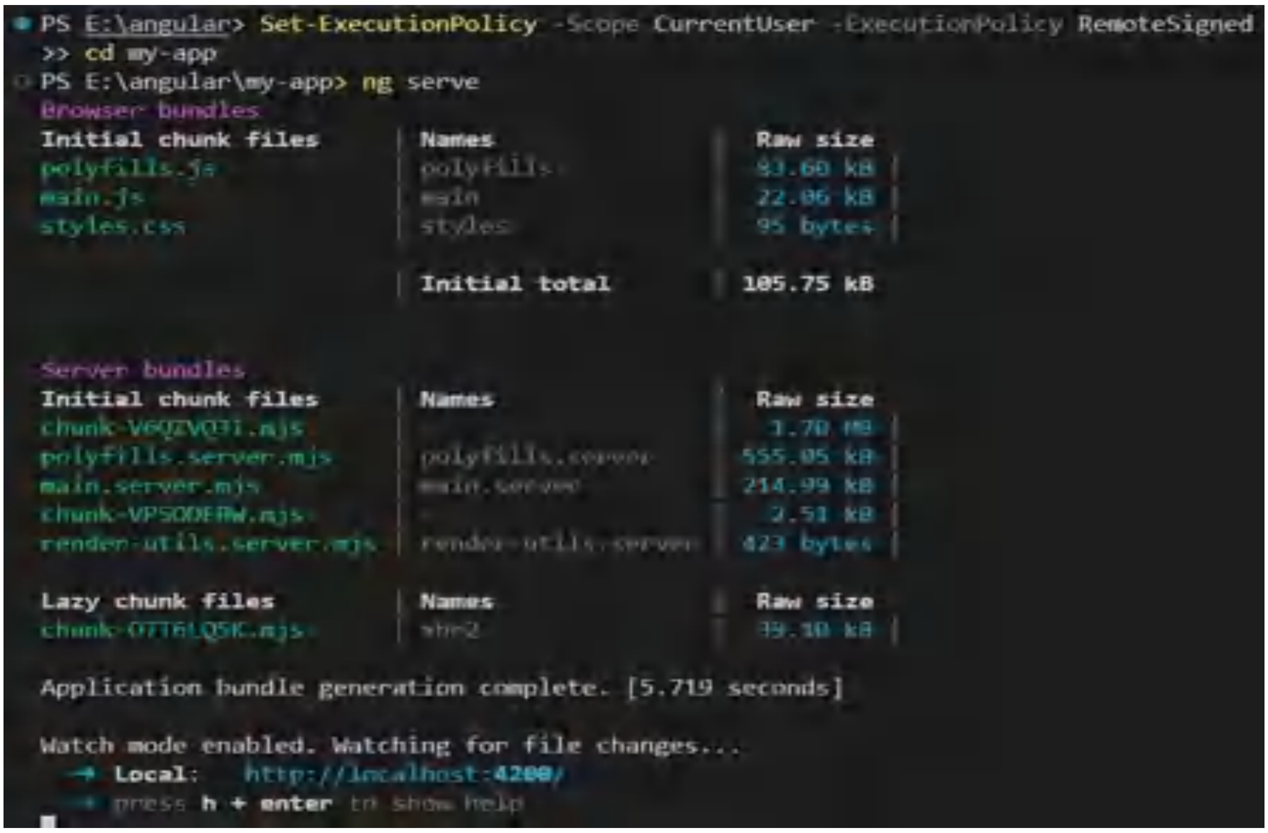
  -- That's it! You should now be able to see your Angular application in your web browser at http://localhost:4200/.

  Watch mode enabled. Watching for file changes...
  ➜ Local: http://localhost:4200/
  ➜ press h + enter to show help

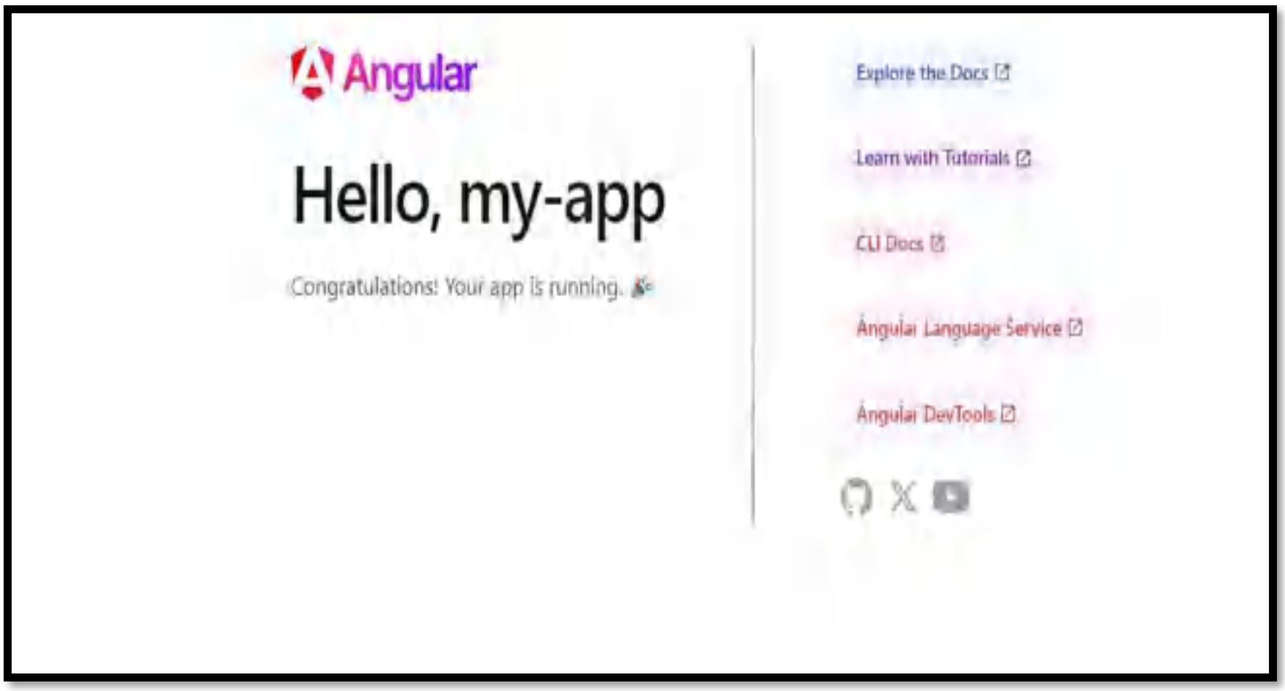
- **OUTPUT:-**



```
PS E:\angular>  npm install -g @angular/cli

changed 232 packages in 9s

44 packages are looking for funding
  run `npm fund` for details
PS E:\angular> ng new my-app
? Which stylesheet format would you like to use? CSS             [ https://developer.mozilla.org/docs/Web/CSS
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? Yes
CREATE my-app/angular.json (2838 bytes)
CREATE my-app/package.json (1263 bytes)
CREATE my-app/README.md (1086 bytes)
CREATE my-app/tsconfig.json (889 bytes)
CREATE my-app/.editorconfig (290 bytes)
CREATE my-app/.gitignore (590 bytes)
CREATE my-app/tsconfig.app.json (342 bytes)
CREATE my-app/tsconfig.spec.json (287 bytes)
CREATE my-app/server.ts (1759 bytes)
CREATE my-app/.vscode/extensions.json (134 bytes)
CREATE my-app/.vscode/launch.json (490 bytes)
CREATE my-app/.vscode/tasks.json (980 bytes)
CREATE my-app/src/main.ts (256 bytes)
CREATE my-app/src/favicon.ico (15086 bytes)
CREATE my-app/src/index.html (304 bytes)
CREATE my-app/src/styles.css (81 bytes)
CREATE my-app/src/main.server.ts (271 bytes)
CREATE my-app/src/app/app.component.html (20239 bytes)
CREATE my-app/src/app/app.component.spec.ts (945 bytes)
CREATE my-app/src/app/app.component.ts (315 bytes)
CREATE my-app/src/app/app.component.css (0 bytes)
CREATE my-app/src/app/app.config.ts (330 bytes)
CREATE my-app/src/app/app.routes.ts (80 bytes)
CREATE my-app/src/app/app.config.server.ts (361 bytes)
CREATE my-app/src/assets/.gitkeep (0 bytes)
✓ Packages installed successfully.
warning: in the working copy of 'package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package.json', LF will be replaced by CRLF the next time Git touches it
    Successfully initialized git.
```



```
PS E:\angular> Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
>> cd my-app
PS E:\angular\my-app> ng serve
Browser bundles
Initial chunk files        | Names                   |  Raw size
polyfills.js               | polyfills               |  83.60 kB |
main.js                    | main                    |  22.06 kB |
styles.css                 | styles                  |  95 bytes |

                           | Initial total           | 105.75 kB


Server bundles
Initial chunk files        | Names                   |  Raw size
chunk-V6QZVQ3I.mjs         | -                       |   1.70 MB |
polyfills.server.mjs       | polyfills.server        | 555.05 kB |
main.server.mjs            | main.server             | 214.99 kB |
chunk-VPSODERW.mjs         | -                       |   2.51 kB |
render-utils.server.mjs    | render-utils.server     | 423 bytes |

Lazy chunk files           | Names                   |  Raw size
chunk-O7T6LQ5K.mjs         | xhr2                    |  39.10 kB |

Application bundle generation complete. [5.719 seconds]

Watch mode enabled. Watching for file changes...
  ➜  Local:   http://localhost:4200/
  ➜  press h + enter to show help
```



Hello, my-app

Congratulations! Your app is running. 🎉