Tutorial: Using MongoDB CRUD Operations in a Programming Application

Dr.Nadeem Qazi

In one of the previous tutorials, you learned how to perform CRUD (Create, Read, Update, Delete) operations in MongoDB directly from the command prompt. While these commands are essential, they are not sufficient for developing applications. In real-world applications, MongoDB commands must be integrated into a program, enabling seamless interaction with the database.

This tutorial will guide you through implementing MongoDB CRUD operations in a programming application using **Visual Studio Code (VS Code)**. By the end of this tutorial, you will be able to integrate MongoDB commands into a Node.js application, enabling dynamic database interactions.

Before you begin, ensure you have the following installed:

1. **Node.js** (v14 or later)
2. **MongoDB** (running locally or a cloud instance like MongoDB Atlas)
3. **Visual Studio Code**
4. **MongoDB Node.js Driver**

Learning Objectives

By the end of this tutorial, you will learn how to:

- Connect to a MongoDB database using **Mongoose**.
- Create a new database.
- Create a collection in the database.
- Insert one or multiple documents into a collection programmatically.

Prerequisites

Before you begin, ensure the following components are installed on your system:

1. **MongoDB**
2. **Mongoose** (to be installed within your project).

Step 1: Setting Up the Project (Visual Studio Code Users Only)

If you are using **Visual Studio Code**, follow this step. Otherwise, skip to Step 2.

1. Create a new directory on your system, for example, MongoWeek, or use a name of your choice.
2. Open your terminal or command prompt, navigate to the directory, and run the following command to initialize a Node.js project:

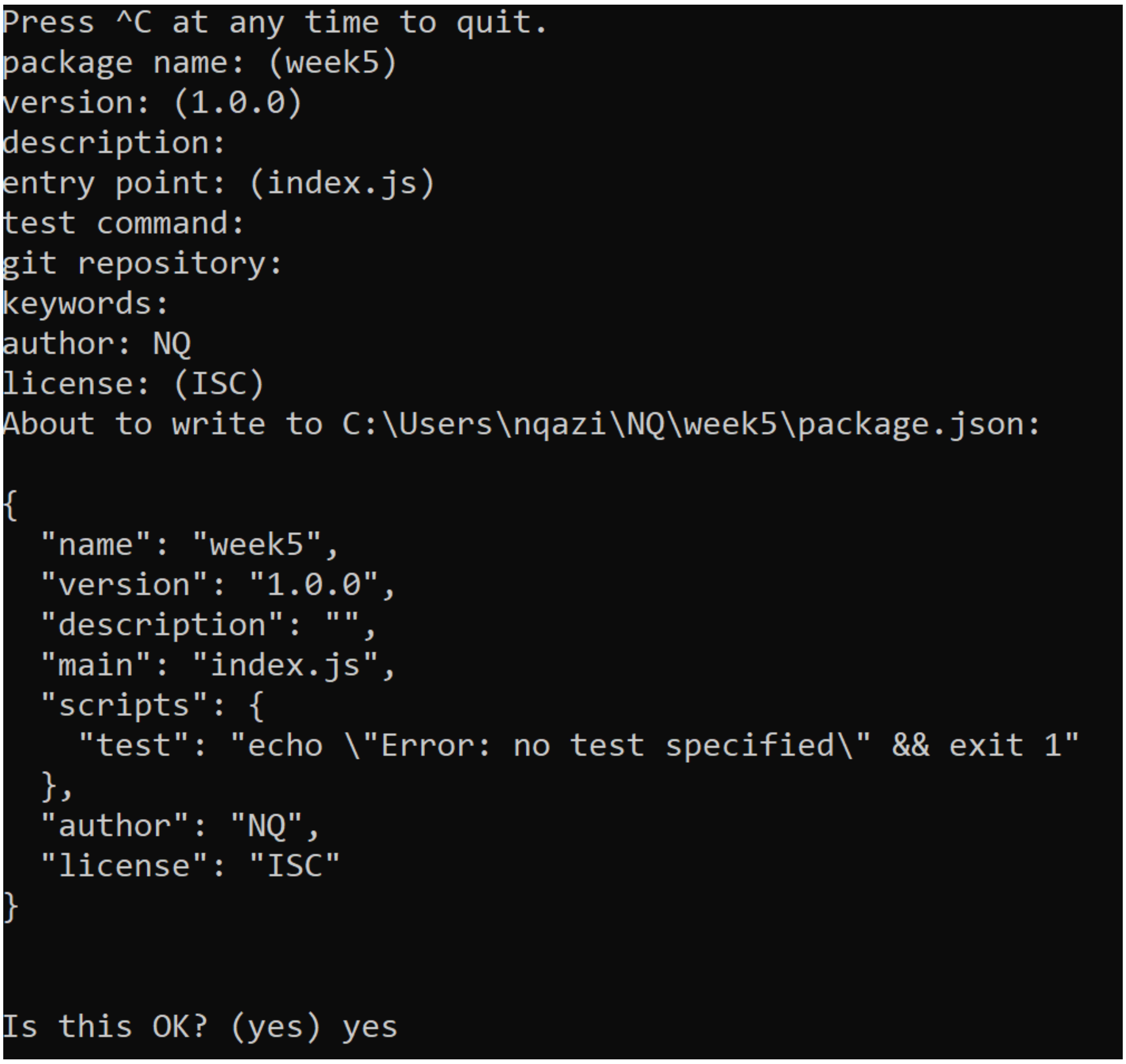
```
npm init
```

You will see a screen similar to the following:

```
This utility will walk you through creating a package.json file.
It only covers the most common items and tries to guess sensible defaults.

Press ^C at any time to quit.
package name: (MongoWeek)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
```

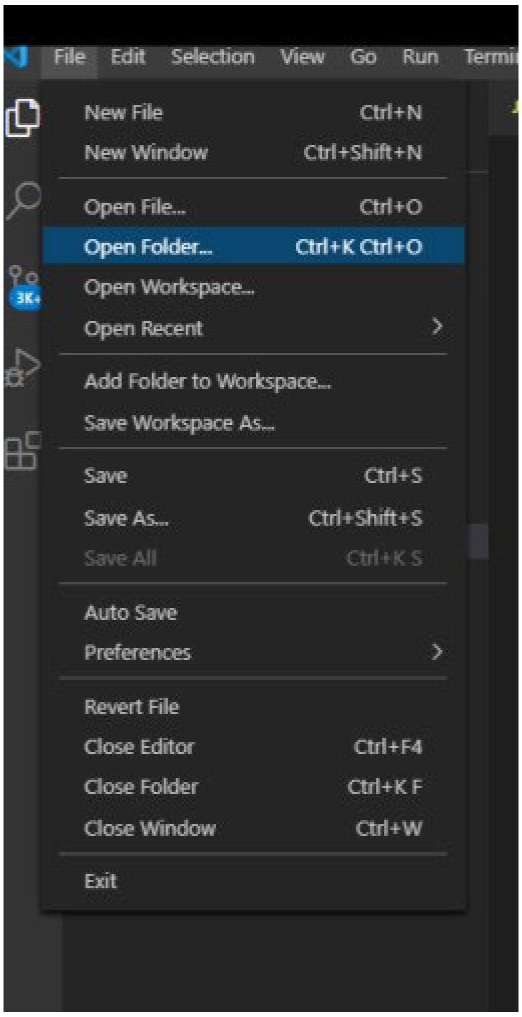
Press enter and default values. This will create a package.json file on your newly created dir Week8. Finally you will see this screen write yes and enter

Press **Enter** to accept the default values, or customize them as needed. This will create a `package.json` file in your project folder.

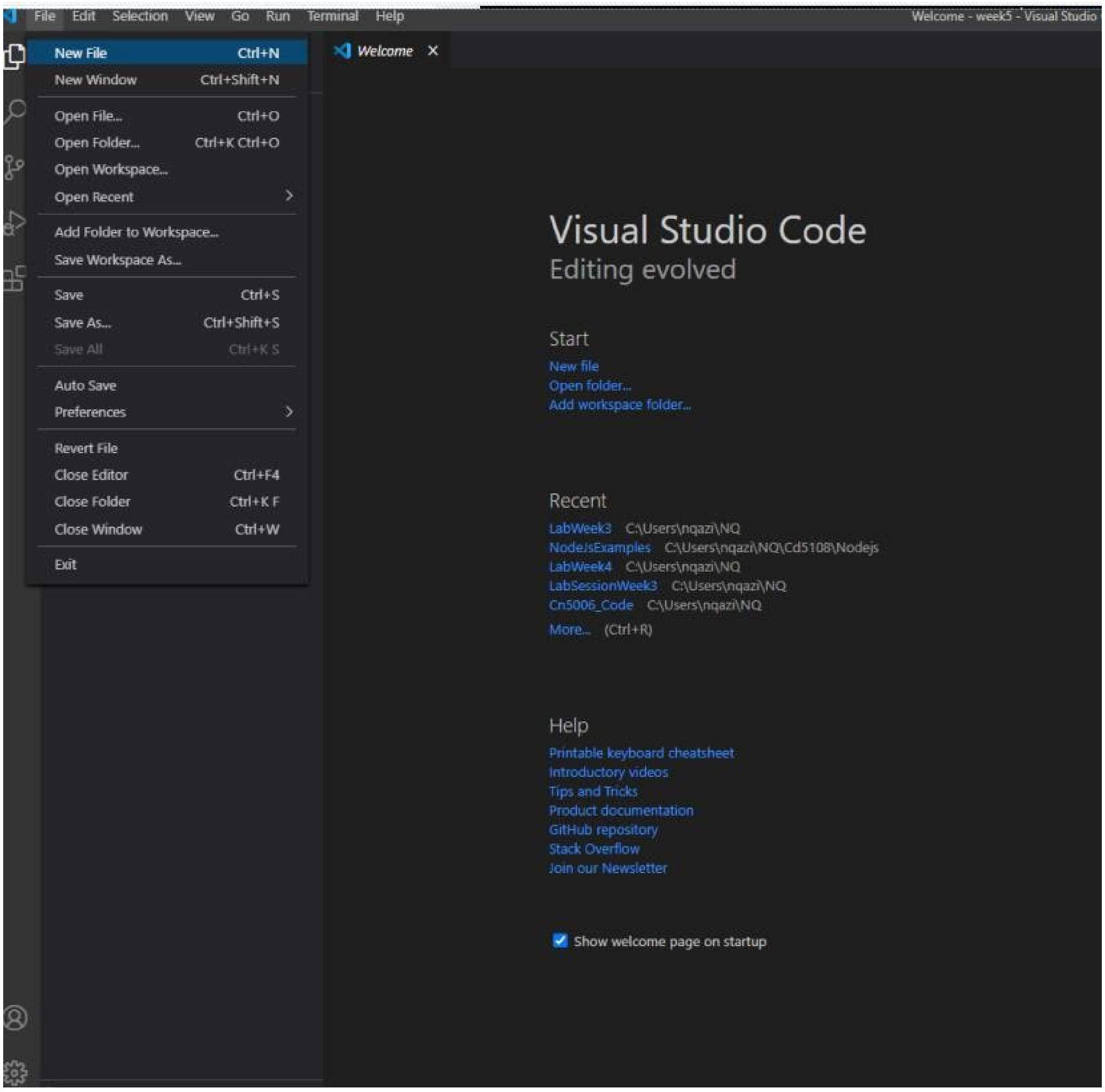


```
Press ^C at any time to quit.
package name: (week5)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author: NQ
license: (ISC)
About to write to C:\Users\nqazi\NQ\week5\package.json:

{
  "name": "week5",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "NQ",
  "license": "ISC"
}


Is this OK? (yes) yes
```

Now start Visual Studio code and change the folder you just created. You can change to your folder in this case MongoWeek folder by clicking the open folder menu in Visual Studio code as shown in the figure below:



After changing to the folder to MongoWeek or the folder you created , click the file then new as shown in the figure below



Save this file as index.js

Now we're ready to dive into coding! I understand that programming can sometimes feel challenging, so I've kept the code in this tutorial as simple and concise as possible. Hopefully, this approach will make it easier for you to follow and understand.

Steps to Connect MongoDB Programmatically

In order to perform this tutorial you will be doing following steps

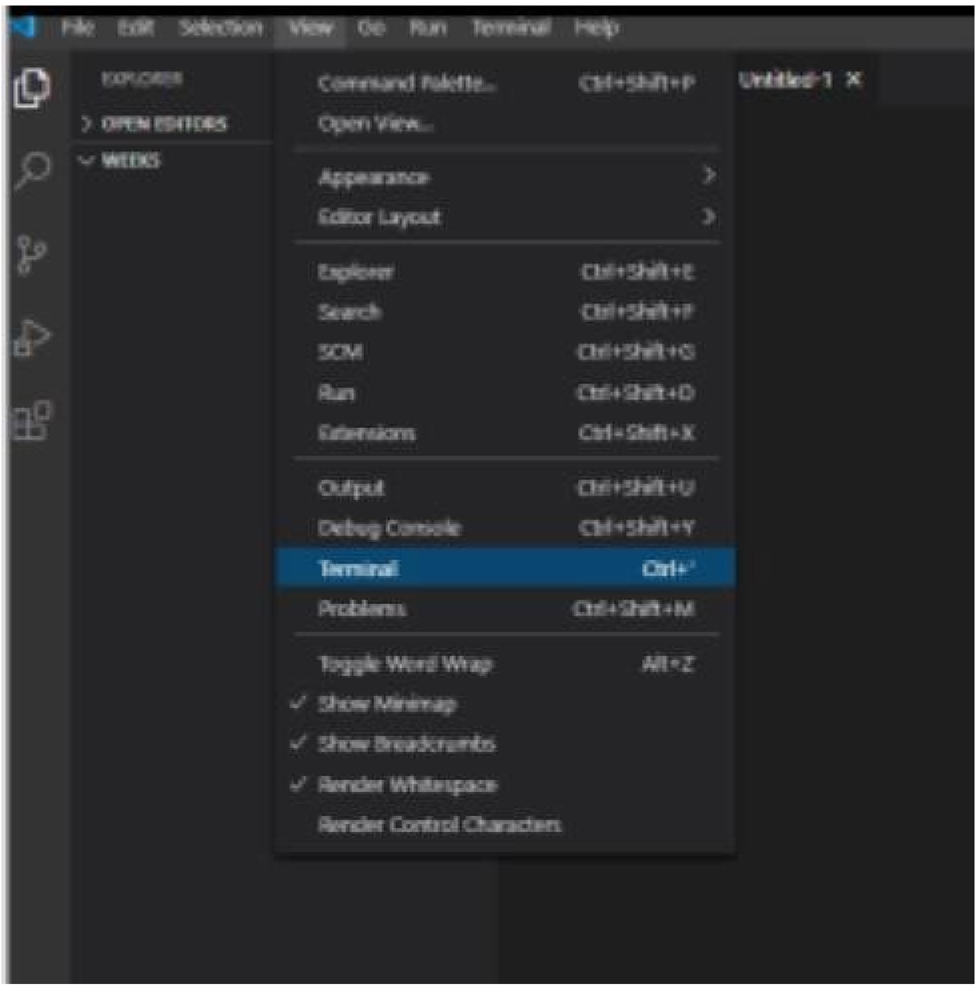
1. Mongoose: Steps required
    1. Install Mongoose (only for first time)
    2. Import mongoose
    3. Connect with Mongodb
        1. URL= localhost:portnumber/databasename mongodb port is 27017

    2. Use of callback functions (on and once)to handle error and connection

4. Create Schema
5. Create Model
6. Create or use existing   collection
    i. Create documents
    ii. Update documents
    iii. Delete documents
    iv. Query documents

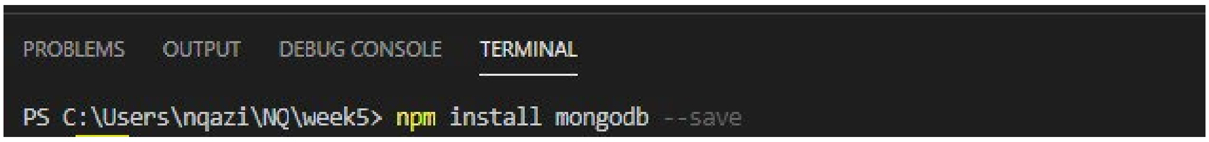
Step 1. Installation of mongoose.

Click view and then click terminal as shown in the figure



A terminal window will be opened write following in the terminal window

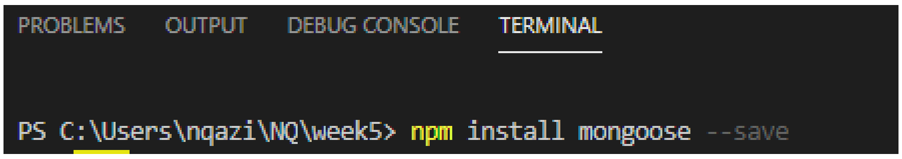
npm install mongodb –save



```
PS C:\Users\nqazi\NQ\week5> npm install mongodb --save
```

Then in the same terminal window type the following

command:	npm install mongoose –save



```
PS C:\Users\nqazi\NQ\week5> npm install mongoose --save
```

With Mongoose and MongoDB successfully installed, we're now ready to begin coding in index.js.Write the first command:

To include Mongoose in your program, use the following code:, to understand the code please watch the video or read week8lecture.pdf

Task 1: connect with mongoDB and add a single document

Add following code in index.js and save

```
mongoose  =  require('mongoose');
//app  =  express();
const  MONGO_URI  =  'mongodb://localhost:27017/Week8';
mongoose.connect(MONGO_URI,  {useUnifiedTopology:  true,useNewUrlParser:  true})
; const  db  =  mongoose.connection;

db.on('error',  function(err)

{console.log("Error  occured  during  connection"+err)
}
);

db.once('connected',  function()    {
console.log(`Connected  to  ${MONGO_URI}`);
});

// creating the scheme
const  PersonScheme  =  new mongoose.Schema({ name: {
type:  String, required:  true
},
age:  Number, Gender:String, Salary:Number
});

//  creating  model  named  as  modelname  with  collection  named  as  personCo
llection
const  person_doc  =  mongoose.model('modelname',  PersonScheme,'personCollectio
n');
// creating a single document
const  doc1  =  new  person_doc({  name:  'Jacky',age:362,Gender:"Male",Salary:3
456 }

);
//  adding  one  document  in  the  collection

doc1
    .save()
    .then((doc1) => {
        console.log("New Article Has been Added Into Your DataBase.",doc1);
    })
    .catch((err) => {
        console.error(err);
    });
```

Save this code and on the terminal write **<u>Node (with small n) index.js</u>**

You will see the number of documents displayed in the terminal.

Now, open MongoDB Compass after running the program. You should see a database named week8, and within it, a collection called personCollection.

Check the number of documents in it . there should be one document name jack

Now in the code above change the value of the name to some other name such as

```
const doc1 = new person_doc({ name:
"Yousuf",age:44,Gender:"Male",Salary:3456
      }
```

save the file again and compile it again using node index.js to run the programme .
check again the week8 database and collection person collection

how many records do you see:

repeat the same procedure with a different name age and salary and see the number of documents added to the collection.

Task 1: In your portfolio for this week, explain the purpose of doc1.save and `.then` in the code. It is used to handle promises, ensuring that the database connection or any asynchronous operation is successfully completed before executing the subsequent logic. This helps manage flow control and handle results or errors effectively.

Task 2: Adding multiple documents

Now in the index.js append the following code

```
manypersons=[{  name:  'Simon',age:42,Gender:"Male",Salary:3456 }
   ,{  name:  'Neesha',age:23,Gender:"Female",Salary:1000  }
   ,{  name:  'Mary',age:27,Gender:"Female",Salary:5402   },
   {  name:  'Mike',age:40,Gender:"Male",Salary:4519   }
   ]


   person_doc.insertMany(manypersons).then(function(){
       console.log("Data inserted")  // Success
   }).catch(function(error){
       console.log(error)      // Failure
   });
```

Save the index.js and run it using node index.js

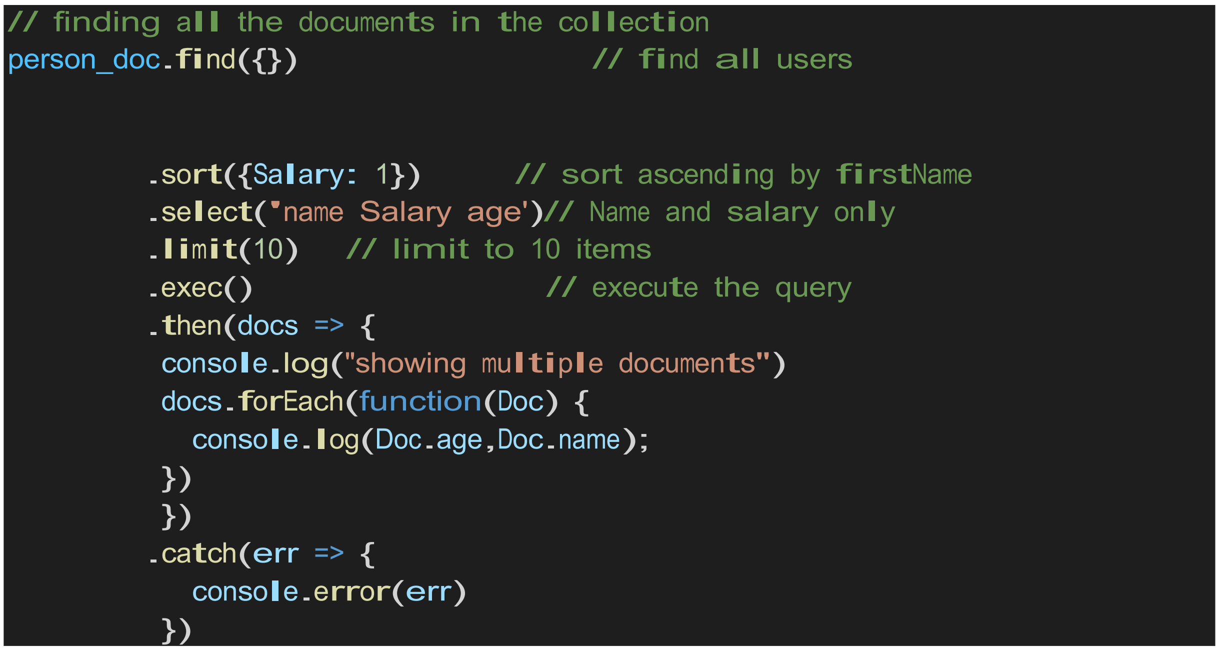
Check the number of documents created in mongodb.

Identify which command is used for creating multiple documents and explain it when you write your portfolio

Tasking 3 fetching the data use of find:

i) Return all the documents from the collection without any filtering criteria and limit the records to 5.

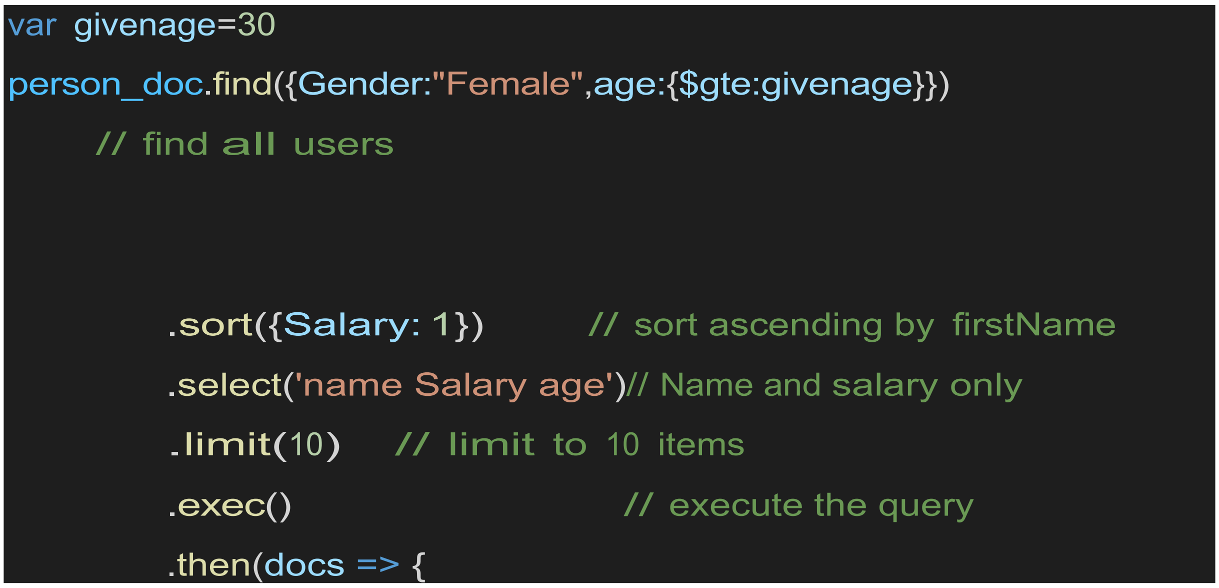
ii) Return the documents from collection with filtering criteria
Add following code in index.js to return all the records



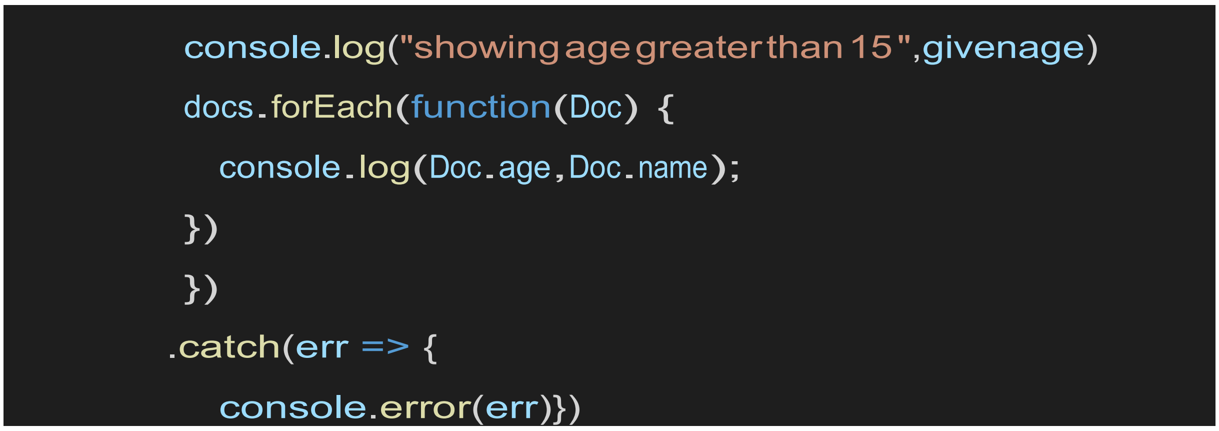
```
// finding all the documents in the collection
person_doc.find({})                    // find all users


         .sort({Salary: 1})      // sort ascending by firstName
         .select('name Salary age')// Name and salary only
         .limit(10)   // limit to 10 items
         .exec()                      // execute the query
         .then(docs => {
          console.log("showing multiple documents")
          docs.forEach(function(Doc) {
            console.log(Doc.age,Doc.name);
          })
          })
         .catch(err => {
            console.error(err)
          })
```

Save the index.js and run the code using **node index.js.** Check how many documents are returned. Take the screenshot of the output.

Task 4 find command with filtering criteria. Run the find command s for which gender =Female and age is greater than some given number.



```
var givenage=30
person_doc.find({Gender:"Female",age:{$gte:givenage}})
      // find all users


          .sort({Salary: 1})        // sort ascending by firstName
          .select('name Salary age')// Name and salary only
          .limit(10)     // limit to 10 items
          .exec()                         // execute the query
          .then(docs => {
```

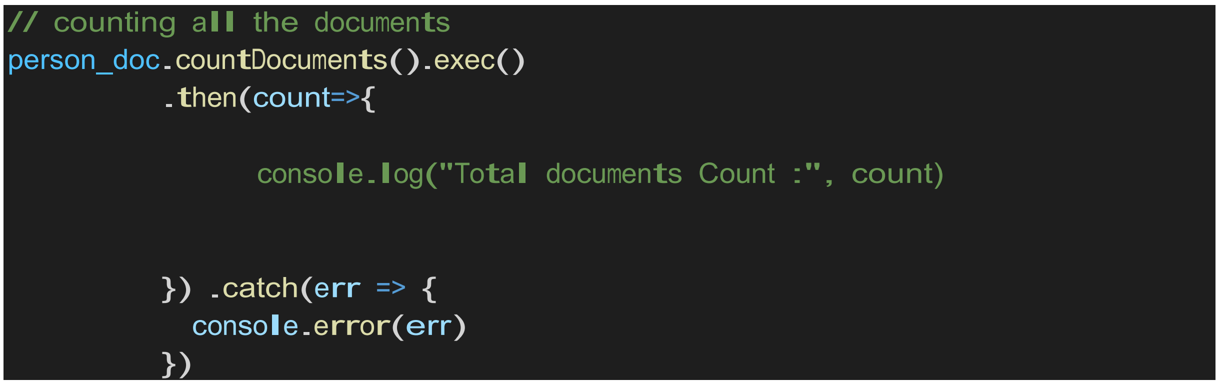


```
          console.log("showing age greater than 15",givenage)
          docs.forEach(function(Doc) {
            console.log(Doc.age,Doc.name);
          })
          })
         .catch(err => {
            console.error(err)})
```

save the file index.js and run again the programme using **<u>node index.js</u> <u>change the value of the givenage and observe the change in the number of records.</u>**

Tasks 5 : write a query to return the total number of documents in the collection.

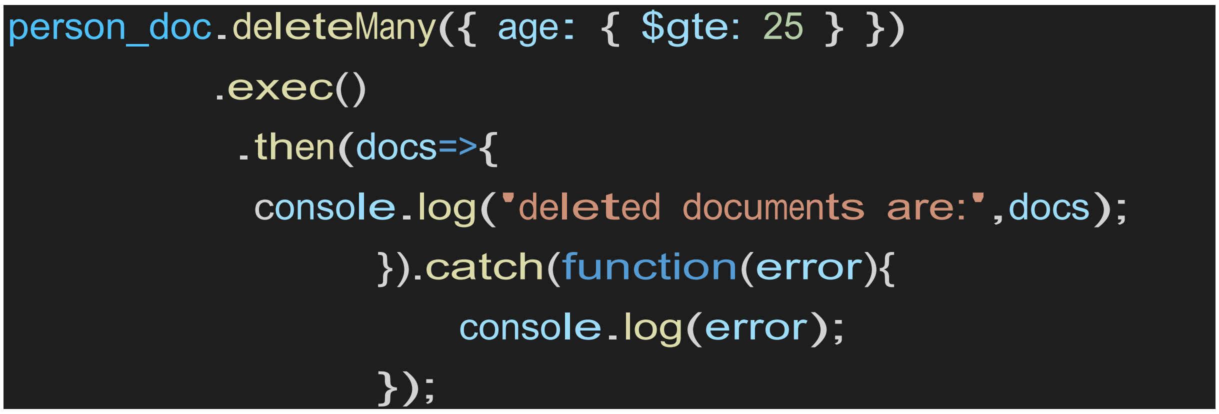
<u>Add the following code in index.js save it and run using node.index.js</u>



```
// counting all the documents
person_doc.countDocuments().exec()
         .then(count=>{

               console.log("Total documents Count :", count)


         }) .catch(err => {
           console.error(err)
         })
```

Task 6 Delete the documents for a given criteria

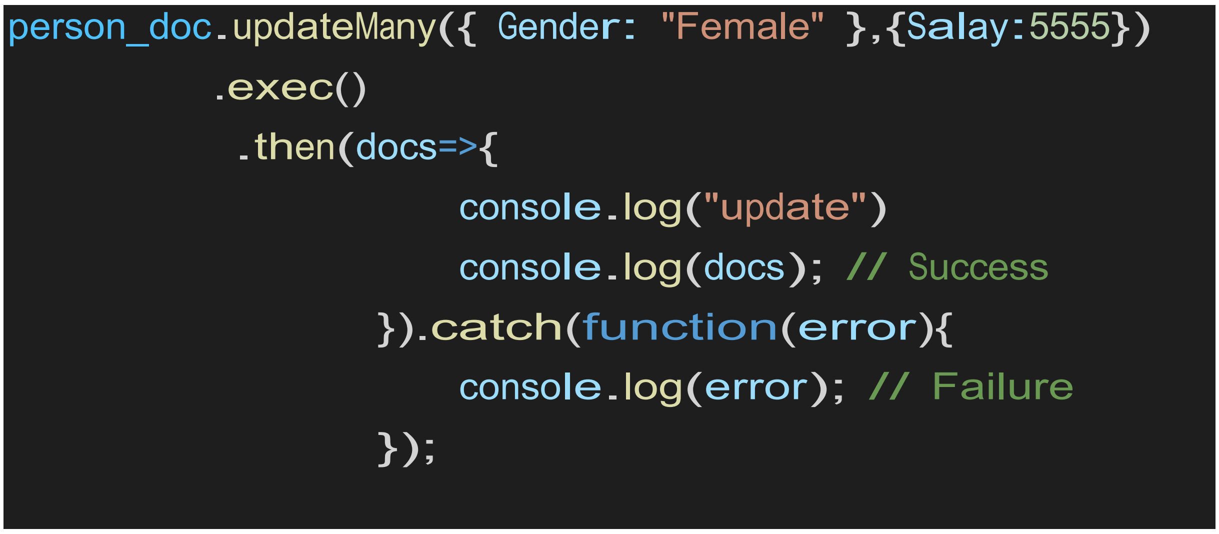
For this we will use deleteMany and filtering criteria is delete all records for which age is greater than 25. Add the following code in index.js ,save it and run it using **node index.js**



```
person_doc.deleteMany({ age: { $gte: 25 } })
            .exec()
             .then(docs=>{
              console.log('deleted documents are:',docs);
                     }).catch(function(error){
                         console.log(error);
                     });
```

Observe what is the number of the deleted documents displayed in the console i.e. terminal window.

Task 7 update all document of which the gender is female and set their salary to 5555 Addfollowing code in index.js save and run it



```
person_doc.updateMany({ Gender: "Female" },{Salay:5555})
          .exec()
           .then(docs=>{
                      console.log("update")
                      console.log(docs); // Success
                  }).catch(function(error){
                      console.log(error); // Failure
                  });
```

Observe how many records are updated.

What to write your portfolio

Include all the provided code as part of your exercise, and share the link to your code in your portfolio. Additionally, explain the functions used, such as `save`, `find`, etc., with detailed descriptions. Enhance your portfolio by including explanations along with relevant screenshots to illustrate the functionality