Lesson 10 Demo 03

Demonstrating CRUD Operations

Objective: To demonstrate CRUD operations in an Express.js application for data manipulation and management

Tools Required: Visual Studio, Node.js, Express.js, and, Postman

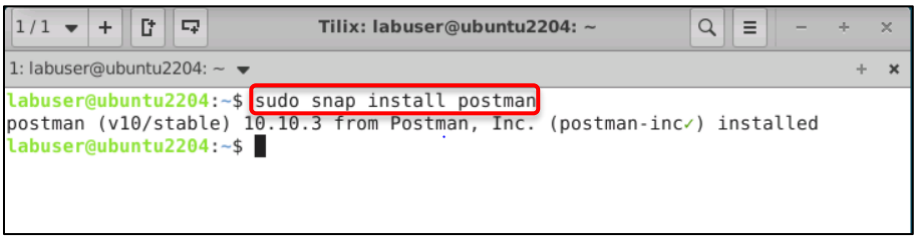
Prerequisites: Knowledge of JavaScript, and Node.js

Steps to be followed:

1. Install prerequisites for performing CRUD operations
2. Analyze the output of CRUD operations

Step 1: Install prerequisites for performing CRUD operations

1.1 Verify the response method by using the Postman application, and execute the following command in the system terminal to install Postman:
sudo snap install postman



1.2 Create a folder named CRUD_operations and open that folder in VS code. Open the terminal and run the following command to start the project:
npm init

1.3 In this project, some packages are required for CRUD operations, and the following lists the names and commands of the packages to be added:

Express: **npm install express**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SQL CONSOLE

labuser@ubuntu2204:~/ExpressJS$ npm install express

up to date, audited 60 packages in 773ms

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
labuser@ubuntu2204:~/ExpressJS$
```

MongoDB: **npm install mongodb**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SQL CONSOLE

labuser@ubuntu2204:~/ExpressJS$ npm install mongodb

added 16 packages, and audited 76 packages in 2s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
labuser@ubuntu2204:~/ExpressJS$
```

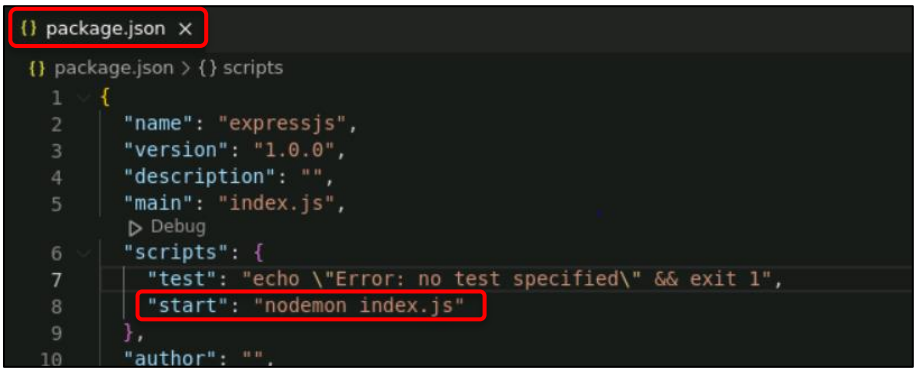
Mongooose: **npm install mongoose**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SQL CONSOLE

labuser@ubuntu2204:~/ExpressJS$ npm install mongoose

added 8 packages, and audited 84 packages in 972ms

8 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
labuser@ubuntu2204:~/ExpressJS$
```

Nodemon: **npm install nodemon –save-dev**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SQL CONSOLE

labuser@ubuntu2204:~/ExpressJS$ npm install nodemon -save-dev

added 32 packages, and audited 116 packages in 3s

11 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
labuser@ubuntu2204:~/ExpressJS$
```

Also, add **"start": "nodemon index.js"** in **package.json** file in **scripts.**

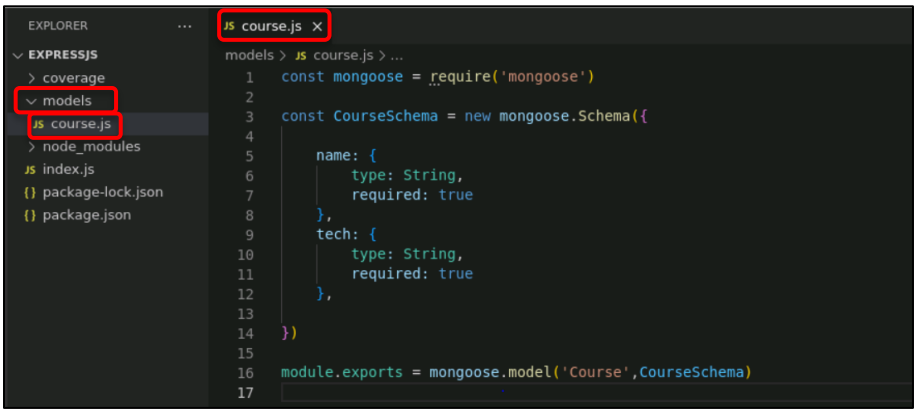


1.4 Create the **index.js** file and add the following code in that file:

```
const express = require('express')
const mongoose = require('mongoose')
const url = 'mongodb://localhost/AlienDBex'

const app = express()

mongoose.connect(url, {useNewUrlParser:true})
const con = mongoose.connection

con.on('open', () => {
  console.log('connected...')
})

app.use(express.json())

const courseRouter = require('./routes/courses')
app.use('/courses',courseRouter)

app.listen(3000, () => {
  console.log('Server started')
})
```

1.5 Create a **models** folder, then inside that folder, generate a **course.js** file and insert the provided code to define a schema for the project model

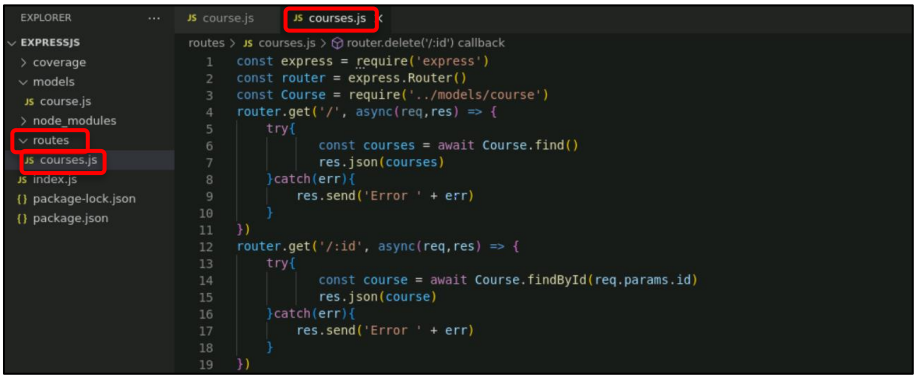
```
const mongoose = require('mongoose')

const CourseSchema = new mongoose.Schema({

   name: {
      type: String,
      required: true
   },
   tech: {
      type: String,
      required: true
   },

})

module.exports = mongoose.model('Course',CourseSchema)
```



1.6 Create a **routes** folder, make a file named **courses.js** inside it, and use the provided code to set up a route for the project

```
const express = require('express')
const router = express.Router()
const Course = require('../models/course')


router.get('/', async(req,res) => {
   try{
        const courses = await Course.find()
        res.json(courses)
   }catch(err){
      res.send('Error ' + err)
   }
})

router.get('/:id', async(req,res) => {
   try{
        const course = await Course.findById(req.params.id)
        res.json(course)
   }catch(err){
      res.send('Error ' + err)
   }
})


router.post('/', async(req,res) => {
   const course = new Course({
      name: req.body.name,
      tech: req.body.tech,
   })

   try{
      const a1 =  await course.save()
      res.json(a1)
   }catch(err){
      res.send('Error')
   }
})

router.patch('/:id',async(req,res)=> {
   try{
      const course = await Course.findById(req.params.id)
      course.name = req.body.name
```

```
        const a1 = await course.save()
        res.json(a1)
    }catch(err){
        res.send('Error')
    }

})

router.delete('/:id',async(req,res)=> {
    try{
        const course = await Course.findById(req.params.id)
        const a1 = await course.remove()
        res.json("Deleted")
    }catch(err){
        res.send('Error')
    }

})

module.exports = router
```



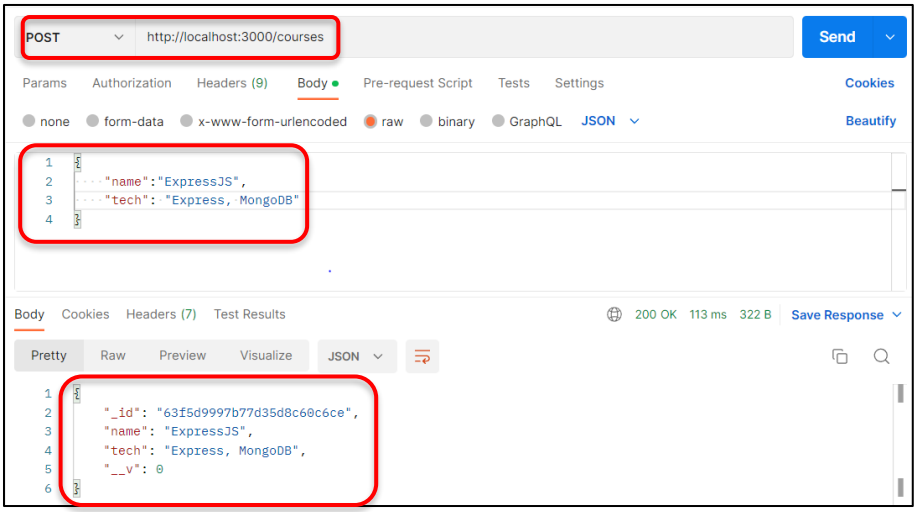
```
JS course.js   JS courses.js ×
routes > JS courses.js > router.patch('/:id') callback
20  router.post('/', async(req,res) => {
21      const course = new Course({
22          name: req.body.name,
23          tech: req.body.tech,
24      })
25      try{const a1 =  await course.save()
26          res.json(a1)
27      }catch(err){
28          res.send('Error')
29      }
30  })
31  router.patch('/:id',async(req,res)=> {
32      try{
33          const course = await Course.findById(req.params.id)
34          course.name = req.body.name
35          const a1 = await course.save()
36          res.json(a1)
37      }catch(err){
38          res.send('Error')
39      }
40  })
41  router.delete('/:id',async(req,res)=> {
42      try{
43          const course = await Course.findById(req.params.id)
44          const a1 = await course.remove()
45          res.json("Deleted")
46      }catch(err){
47          res.send('Error')
48      }
49  })
50  module.exports = router
```

1.7 The project is prepared for CRUD operations; execute the **npm start** command in the terminal

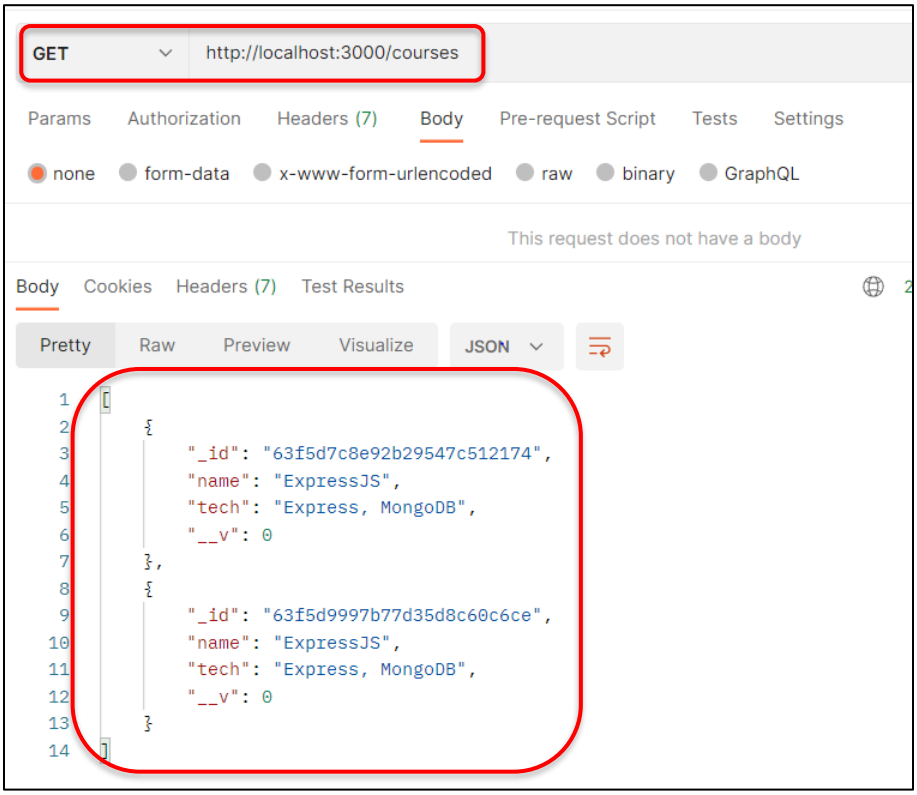
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SQL CONSOLE                    npm + ∨ ⊟ 🗑 ^ X

labuser@ubuntu2204:~/CRUD_oprations$ npm start

> crud_oprations@1.0.0 start
> nodemon index.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
(node:15455) [MONGOOSE] DeprecationWarning: Mongoose: the `strictQuery` option will be switched back to `false` by default in Mongoose 7. Use `mongoose.set(
'strictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server started
```

Step 2: Analyze the output of CRUD operations

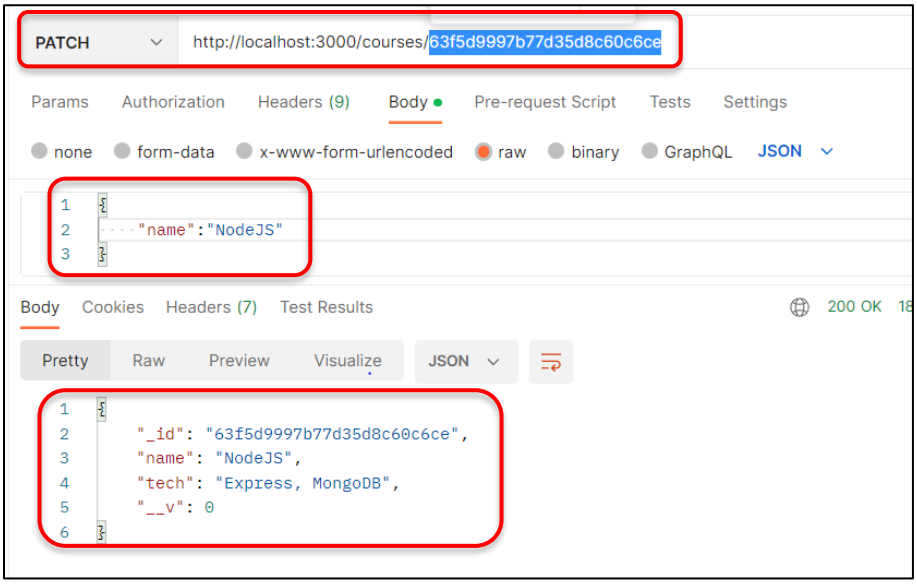
2.1 For the create operation (**C**), use the POST method. In Postman, under the body section, enter {“name”: “ExpressJS”, “tech”: “Express, MongoDB”} and send a POST request to **http://localhost:3000/courses.**



2.2 For the read operation (**R**), use the GET method. In Postman, execute a GET request to **http://localhost:3000/courses.**

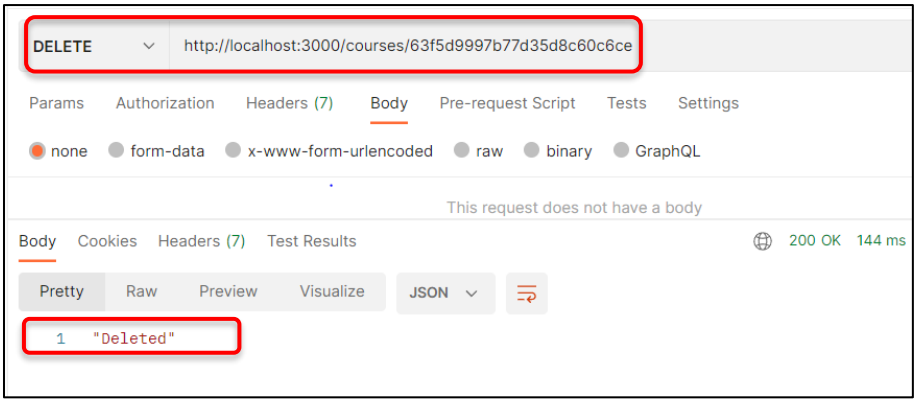


2.3 For the update operation (U), use the PATCH method. In Postman, send a PATCH request to **http://localhost:3000/courses/63f5d9997b77d35d8c60c6ce.**



Note: Here **63f5d9997b77d35d8c60c6ce** is **_id**. Change it according to the ID. To check ID, use the GET method

2.4 For the delete operation (**D**), use the DELETE method. In Postman, send a DELETE request to http://localhost:3000/courses/63f5d9997b77d35d8c60c6ce.



By following these steps, you have successfully implemented CRUD operations in an Express.js application to enable efficient data handling and management.