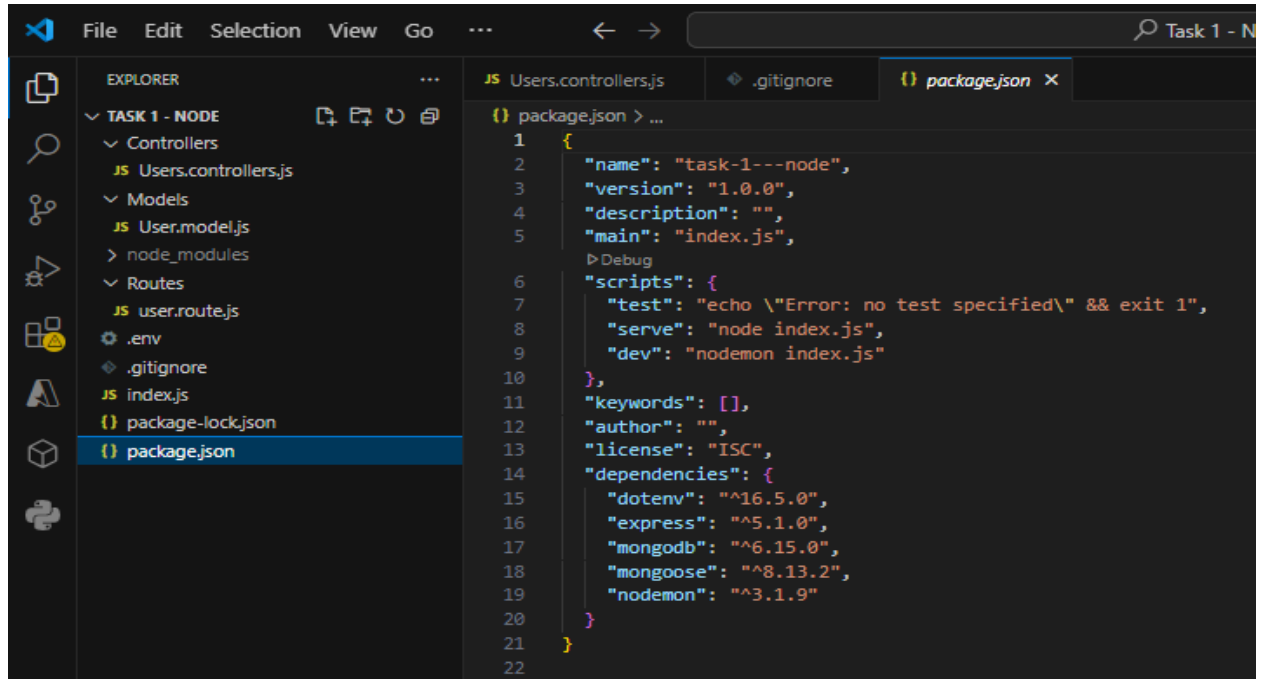


NODE JS - TASK 1

To build Node.js CRUD Application, Testing using Postman and Pushing code to the Github Repository.

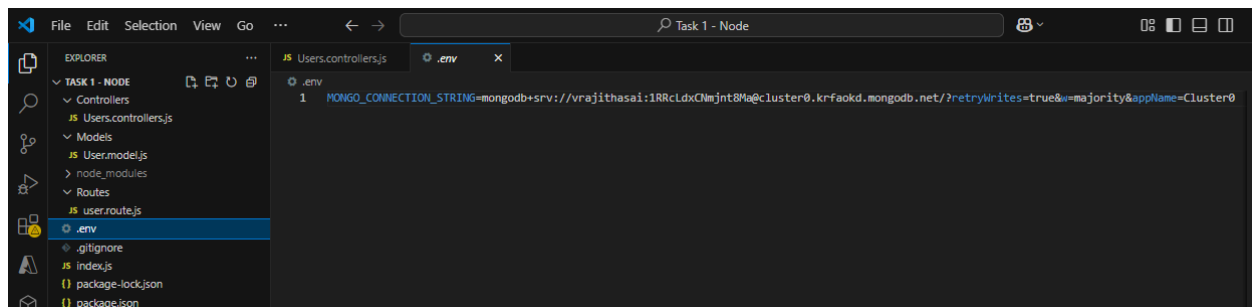
To build Node.js CRUD Application - USER DETAILS APPLICATION

- Initially downloading the necessary packages and checking in package.json if the dependencies are updated correctly.



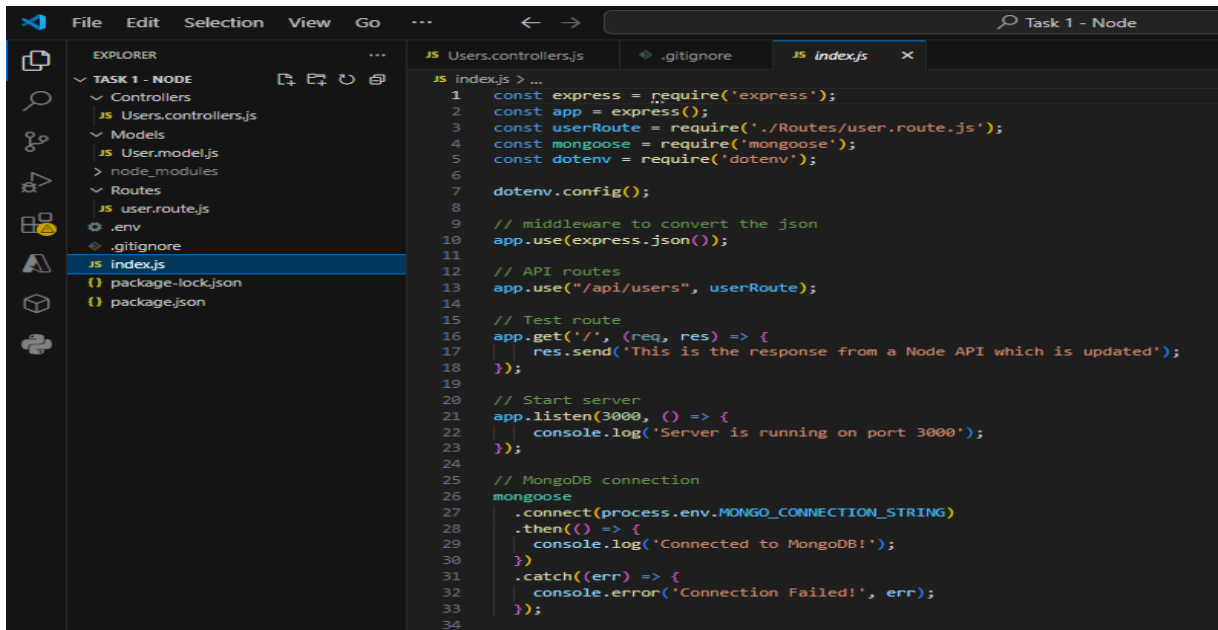
```
1 {
2   "name": "task-1---node",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1",
8     "serve": "node index.js",
9     "dev": "nodemon index.js"
10  },
11  "keywords": [],
12  "author": "",
13  "license": "ISC",
14  "dependencies": {
15    "dotenv": "^16.5.0",
16    "express": "^5.1.0",
17    "mongodb": "^6.15.0",
18    "mongoose": "^8.13.2",
19    "nodemon": "^3.1.9"
20  }
21 }
```

- Mongodb Connectivity [.env is used to store the string that was taken from mongodb]



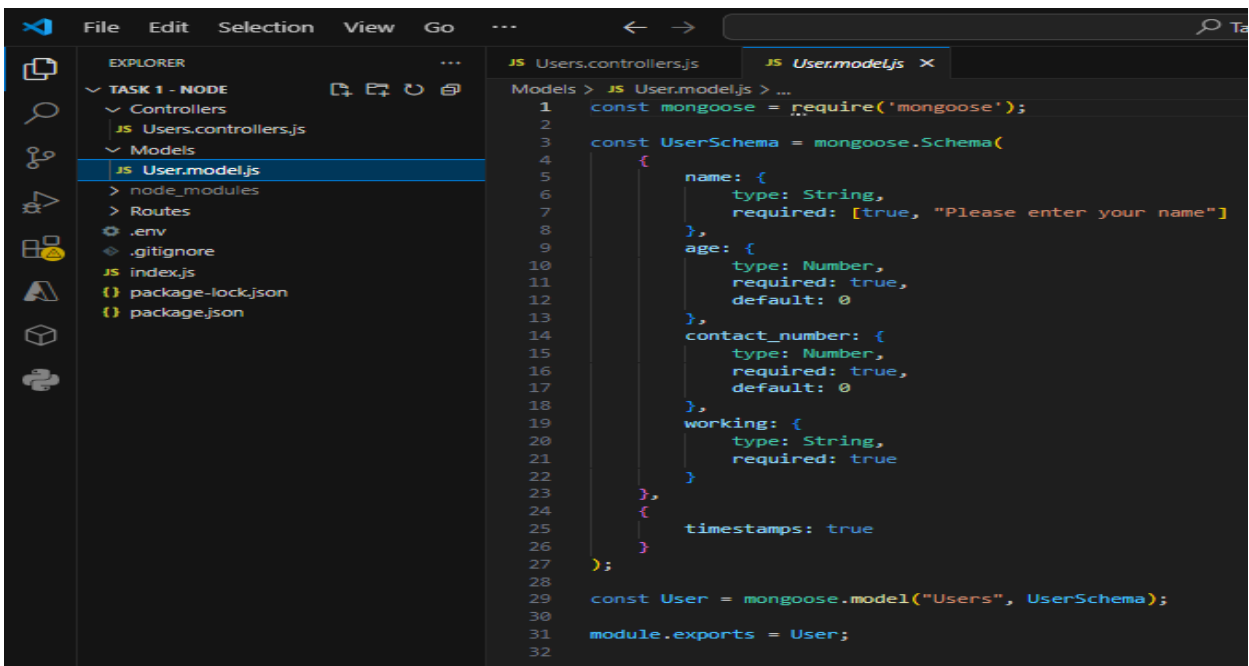
```
1 MONGO_CONNECTION_STRING=mongodb+srv://vrajithasai:1RRcLdxQmJnt8Ma@cluster0.krfaokd.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0
```

- The whole connectivity could be seen in index.js
index.js contains the details of,
Routes (API, Test)
To start the server,
Mongo db connectivity



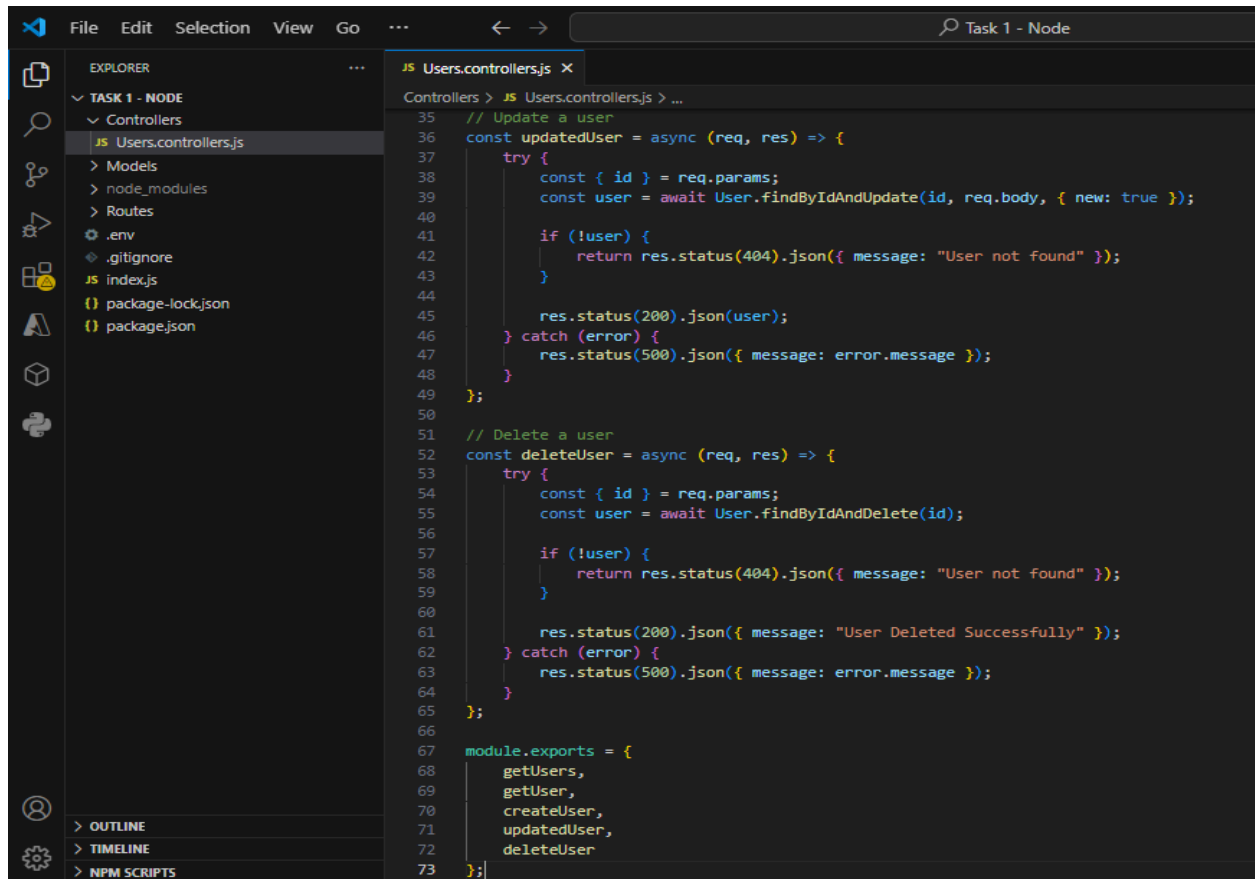
```
1 const express = require('express');
2 const app = express();
3 const userRoute = require('./Routes/user.route.js');
4 const mongoose = require('mongoose');
5 const dotenv = require('dotenv');
6
7 dotenv.config();
8
9 // middleware to convert the json
10 app.use(express.json());
11
12 // API routes
13 app.use("/api/users", userRoute);
14
15 // Test route
16 app.get('/', (req, res) => {
17   res.send('This is the response from a Node API which is updated');
18 });
19
20 // Start server
21 app.listen(3000, () => {
22   console.log('Server is running on port 3000');
23 });
24
25 // MongoDB connection
26 mongoose
27   .connect(process.env.MONGO_CONNECTION_STRING)
28   .then(() => {
29     console.log('Connected to MongoDB!');
30   })
31   .catch((err) => {
32     console.error('Connection Failed!', err);
33   });
34
```

- Creating Schema [USER SCHEMA - name, age, contact_number, working]



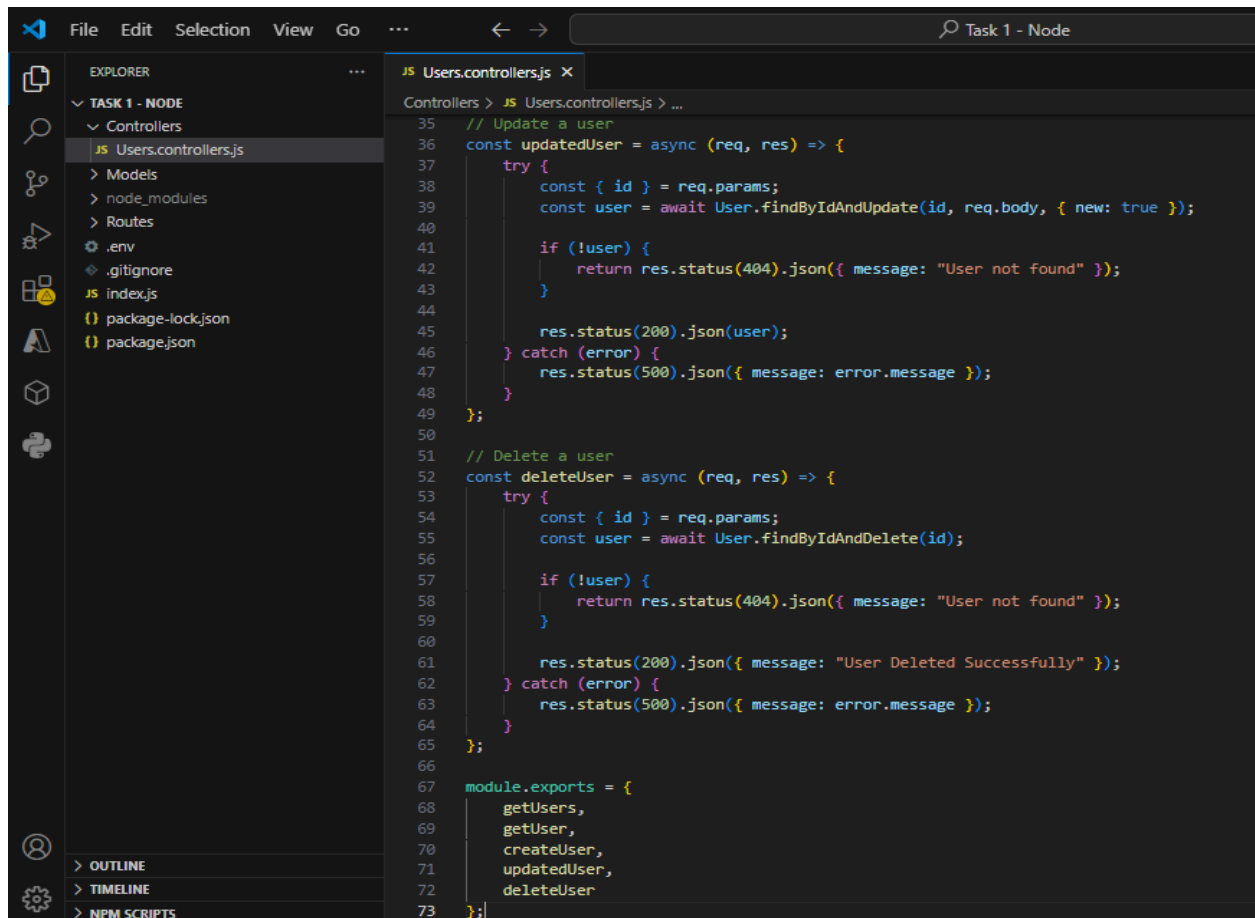
```
1 const mongoose = require('mongoose');
2
3 const UserSchema = mongoose.Schema(
4   {
5     name: {
6       type: String,
7       required: [true, "Please enter your name"]
8     },
9     age: {
10      type: Number,
11      required: true,
12      default: 0
13    },
14    contact_number: {
15      type: Number,
16      required: true,
17      default: 0
18    },
19    working: {
20      type: String,
21      required: true
22    }
23  },
24  {
25    timestamps: true
26  }
27 );
28
29 const User = mongoose.model("Users", UserSchema);
30
31 module.exports = User;
32
```

- Put all the APIs [GET, POST, PUT, DEL]
The details has to be put in the controllers which means it contains functioning or working of each API.



VS Code interface showing the initial state of `Users.controllers.js`. The Explorer sidebar shows the project structure for `TASK 1 - NODE`, including `Controllers`, `Models`, `node_modules`, `Routes`, `.env`, `.gitignore`, `index.js`, `package-lock.json`, and `package.json`. The main editor displays the content of `Users.controllers.js`, which includes functions for updating and deleting users, but the `deleteUser` function is not yet exported in the `module.exports` object.

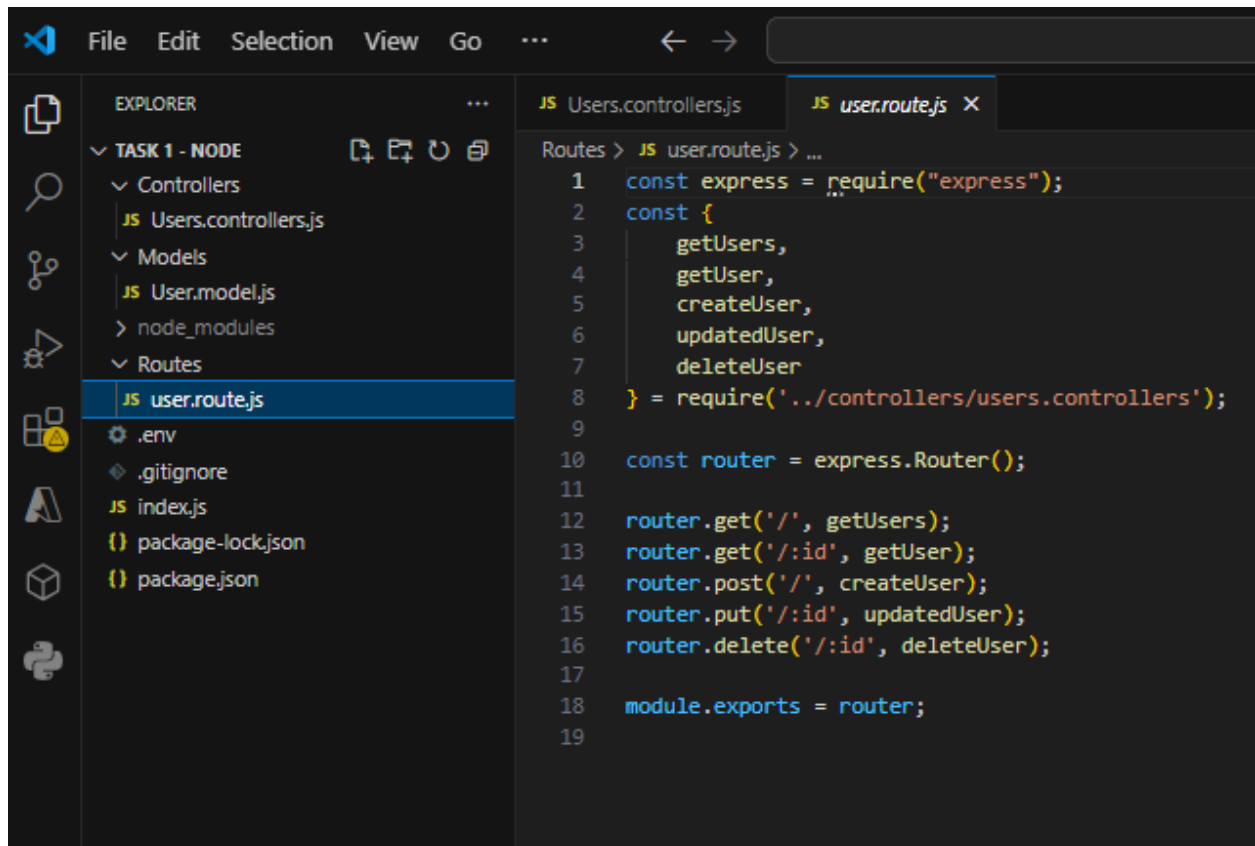
```
35 // Update a user
36 const updatedUser = async (req, res) => {
37   try {
38     const { id } = req.params;
39     const user = await User.findByIdAndUpdate(id, req.body, { new: true });
40
41     if (!user) {
42       return res.status(404).json({ message: "User not found" });
43     }
44
45     res.status(200).json(user);
46   } catch (error) {
47     res.status(500).json({ message: error.message });
48   }
49 };
50
51 // Delete a user
52 const deleteUser = async (req, res) => {
53   try {
54     const { id } = req.params;
55     const user = await User.findByIdAndDelete(id);
56
57     if (!user) {
58       return res.status(404).json({ message: "User not found" });
59     }
60
61     res.status(200).json({ message: "User Deleted Successfully" });
62   } catch (error) {
63     res.status(500).json({ message: error.message });
64   }
65 };
66
67 module.exports = {
68   getUsers,
69   getUser,
70   createUser,
71   updateUser,
72   deleteUser
73 };
```



VS Code interface showing the updated state of `Users.controllers.js`. The `deleteUser` function has been added to the `module.exports` object, making it available for use in other parts of the application.

```
35 // Update a user
36 const updatedUser = async (req, res) => {
37   try {
38     const { id } = req.params;
39     const user = await User.findByIdAndUpdate(id, req.body, { new: true });
40
41     if (!user) {
42       return res.status(404).json({ message: "User not found" });
43     }
44
45     res.status(200).json(user);
46   } catch (error) {
47     res.status(500).json({ message: error.message });
48   }
49 };
50
51 // Delete a user
52 const deleteUser = async (req, res) => {
53   try {
54     const { id } = req.params;
55     const user = await User.findByIdAndDelete(id);
56
57     if (!user) {
58       return res.status(404).json({ message: "User not found" });
59     }
60
61     res.status(200).json({ message: "User Deleted Successfully" });
62   } catch (error) {
63     res.status(500).json({ message: error.message });
64   }
65 };
66
67 module.exports = {
68   getUsers,
69   getUser,
70   createUser,
71   updateUser,
72   deleteUser
73 };
```

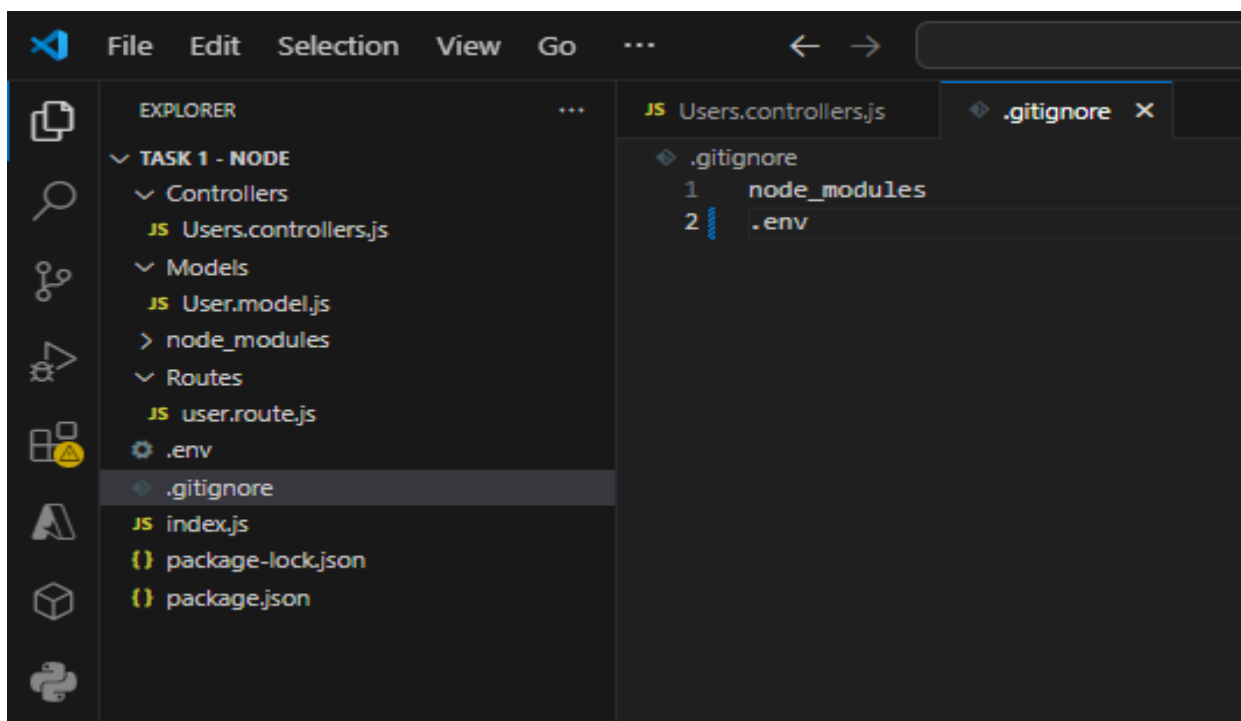
- And now connect all files through Routes



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure under 'TASK 1 - NODE'. It includes folders for 'Controllers', 'Models', and 'Routes'. The 'Routes' folder is expanded, showing 'user.route.js' selected. The main editor area displays the code for 'user.route.js'. The code imports 'express' and 'Users.controllers', creates an Express router, and defines routes for GET, POST, PUT, and DELETE operations. The router is then exported as 'module.exports'.

```
1 const express = require("express");
2 const {
3   getUsers,
4   getUser,
5   createUser,
6   updateUser,
7   deleteUser
8 } = require('../controllers/users.controllers');
9
10 const router = express.Router();
11
12 router.get('/', getUsers);
13 router.get('/:id', getUser);
14 router.post('/', createUser);
15 router.put('/:id', updateUser);
16 router.delete('/:id', deleteUser);
17
18 module.exports = router;
```

- .gitignore file is used - if any files are mentioned in this file, that particular file will not be pushed to github [node_modules - contains more files, .env - contains string which is used for connectivity that has username and password]



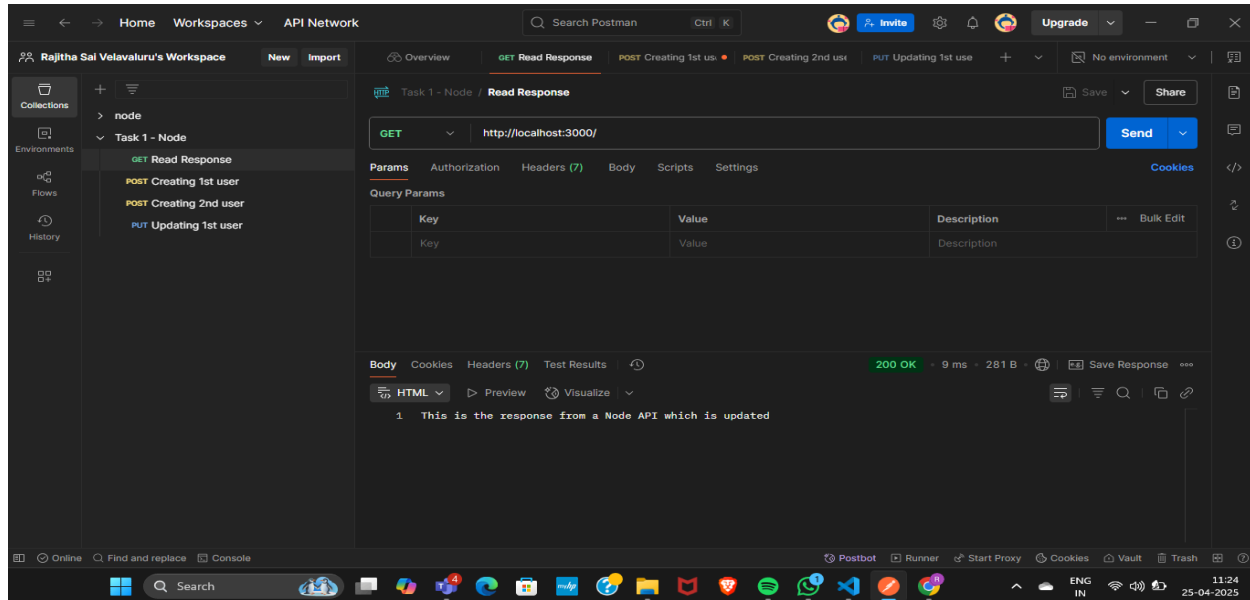
The screenshot shows the Visual Studio Code interface with the '.gitignore' file open in the main editor. The Explorer sidebar on the left shows the same project structure as the previous screenshot, with '.gitignore' selected. The '.gitignore' file contains two entries: 'node_modules' and '.env', which are used to prevent these files from being committed to the repository.

```
1 node_modules
2 .env
```

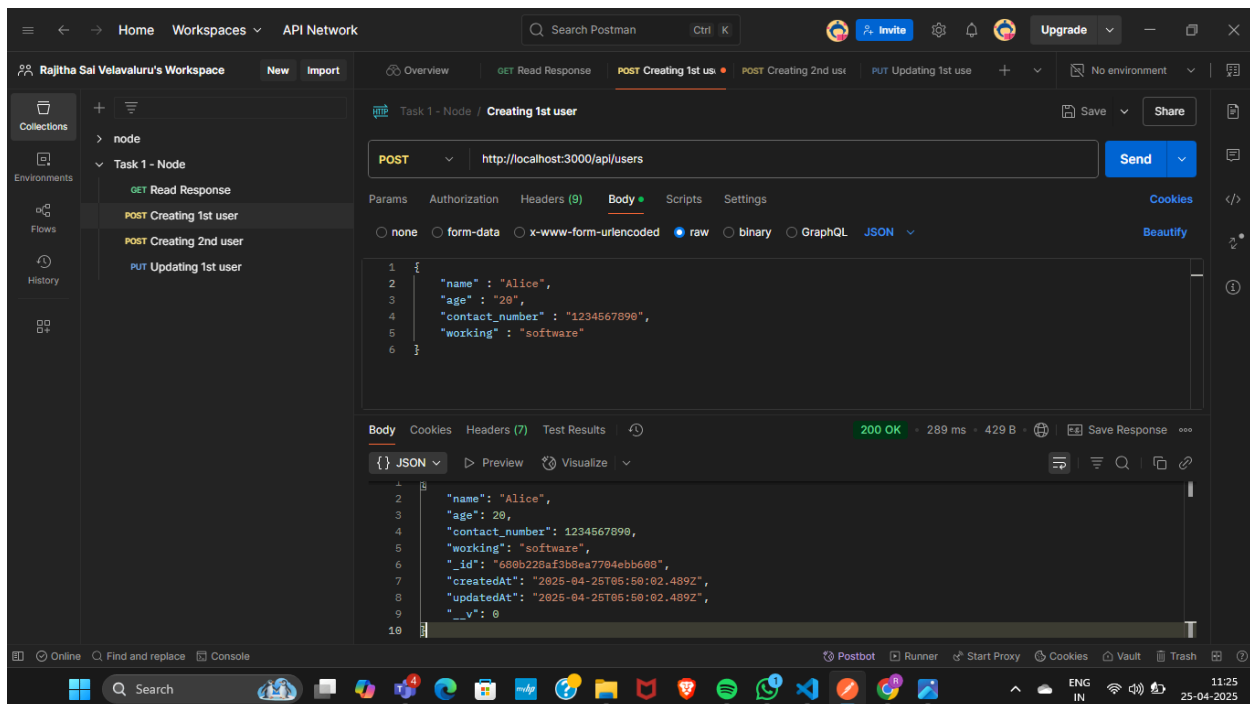
Testing APIs using POSTMAN

Testing includes Creating, Reading, Updating and Deleting of the user

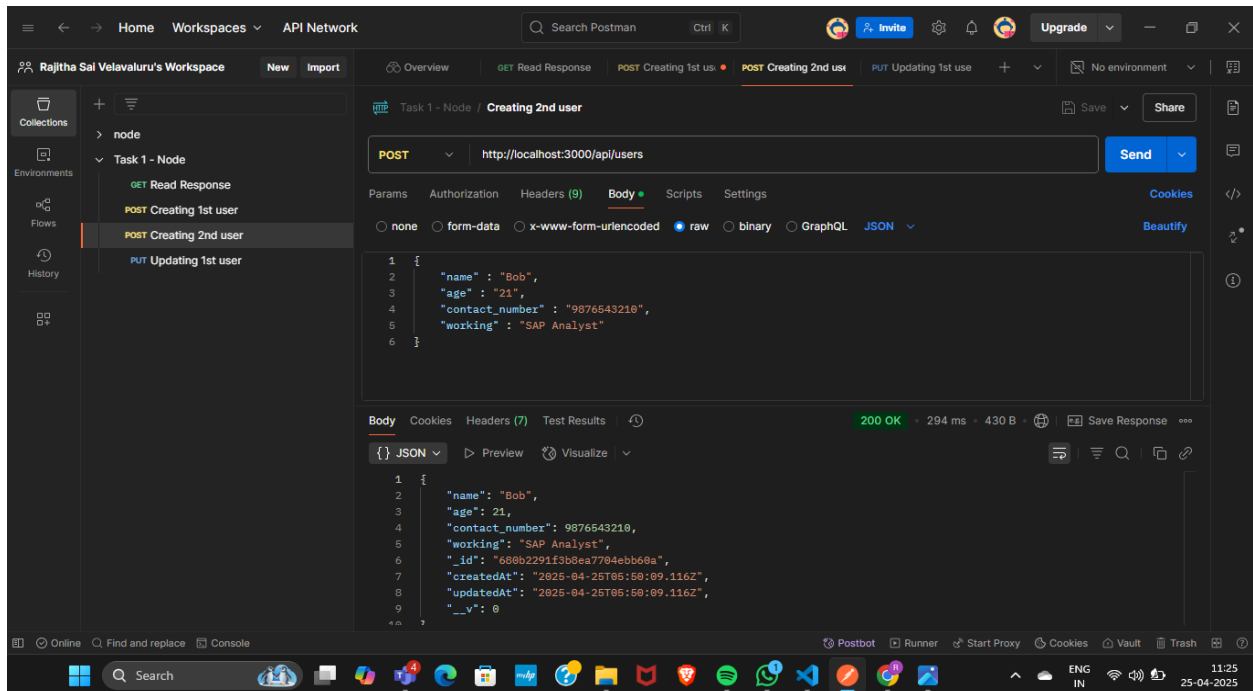
- To check with the response GET method is used



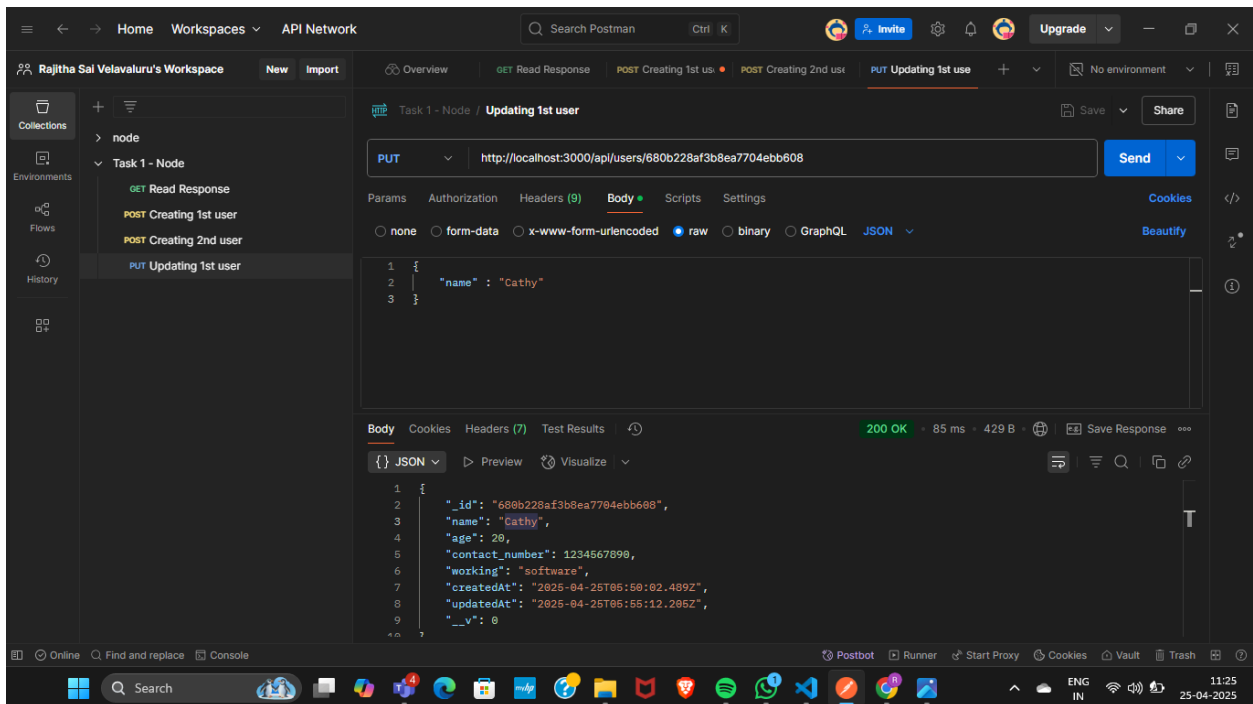
- To post the user details POST method is used [creating 2 users]
1st User - Alice



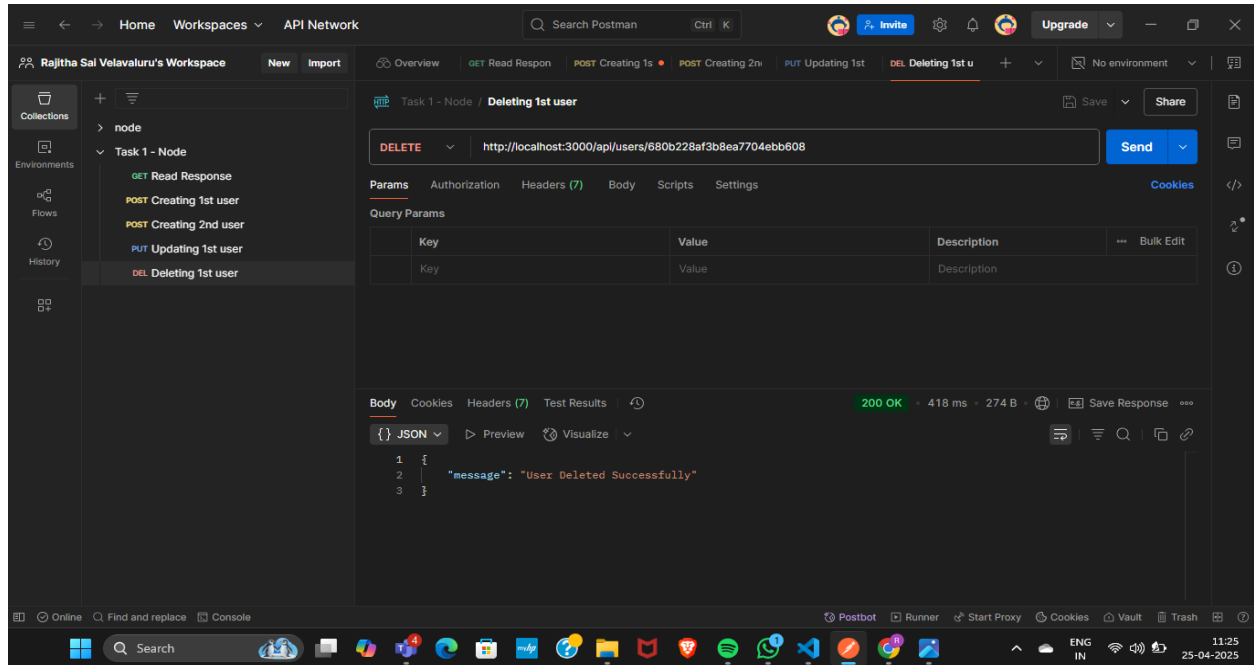
2nd User - Bob



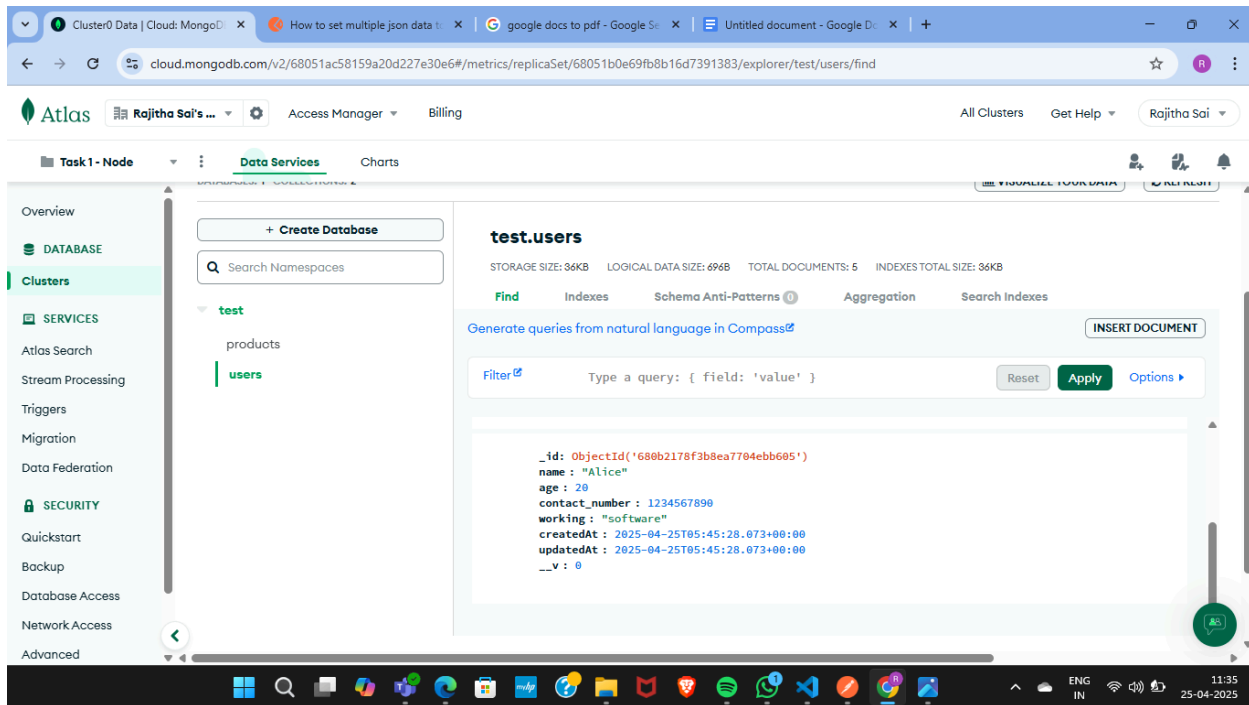
- To update the user details PUT method is user [1st user's name is updated as Cathy using the id]

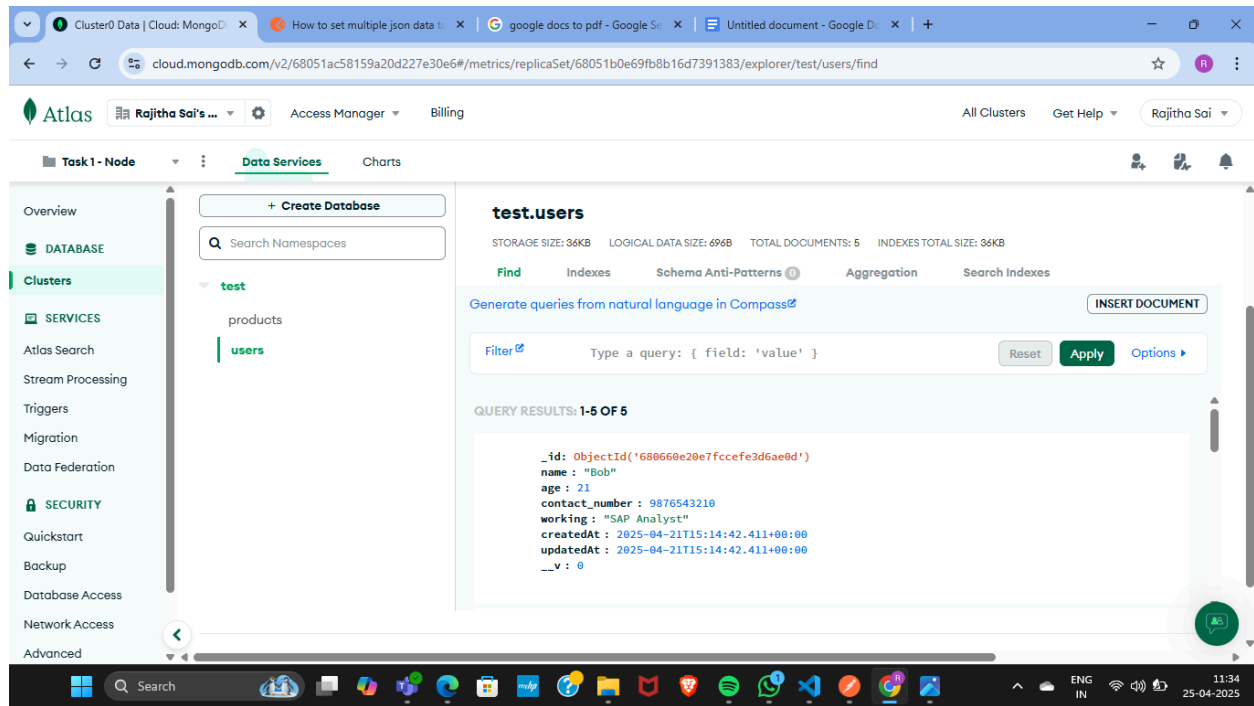


- To delete the user DEL method is used [deleting the 1st user using the id]



User Details can be seen in Mongodb





Pushing code into Github Repository

Using git commands the codes are pushed into the Repository

Repository Link : <https://github.com/rajithasaivelavaluru/task-1-node-CRUD>