# Assignment 11

# Roll No 71 T14

**Aim :** Create a web application that performs CRUD operations (database connectivity)

**Code:**

In backend Files:

Index.js

```
const express = require("express");

var cors = require('cors')

// DATACONNECT
// Import the MongoDB driver
const mongoose = require('mongoose');

// Connect to the MongoDB database
mongoose.connect('mongodb://0.0.0.0:27017/Newdb' );
//specify database for unique email

// Listen for the 'open' event to know when the connection is successful
mongoose.connection.once('open', () => {
  console.log('MongoDB database connection established successfully');
});

// Listen for the 'error' event to know when the connection fails
mongoose.connection.on('error', (err) => {
  console.log('MongoDB database connection failed:', err);
});




const app = express()
const port = 5000


//middleware
app.use(cors())
app.use(express.json());  //orelse will return undefined on req.body

//Available routes
app.use('/api/auth', require('./routes/auth'))
```

```
app.use('/api/notes', require('./routes/notes'))

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Notify listening on port ${port}`)
})
```

**In notes.js**

```
const express = require("express")
const router = express.Router()
const fetchuser = require("../middleware/fetchuser")
const Notes = require('../models/Notes')

const {body,validationResult}= require("express-validator")

const User = require("../models/Notes");
//

// ROUTE1: Get all notes: GET "/api/notes/fetchallnotes"    requires AUTH
login
router.get('/fetchallnotes',fetchuser, async (req,res)=>{

    try {
        const notes = await Notes.find({user : req.user.id});
        res.json(notes);
    } catch (error) {
        console.log(error.message)
        res.status(500).send("Internal Server Error Occured");
    }

})

//ROUTE2 : Add notes : POST 'api/notes/addnote' Requires Auth Login
router.post('/addnote',fetchuser,[
    body('title').isLength({min : 3}),
    body('description' , 'Description length Min 5').isLength({min : 5})
],  async (req,res)=>{

    // If there are errors, return bad request & errors
    const errors = validationResult(req);
    if (!errors.isEmpty()) { //error not empty
      return res.status(400).json({errors:errors.array()});
    // return res.send("ERR") -- my code
```

```
    }


    try {

        const {title,description,tag} = req.body;

        const note = new Notes({
            title,description,tag,user : req.user.id
        })

        const savednote = await note.save();

        // above 3 lines or
        // const note = await Notes.create({
        //     title,description,tag,user : req.user.id
        // })

        res.json(note)

    } catch (error) {
        console.log(error.message)
        res.status(500).send("Internal Server Error Occured");
    }

})

//ROUTE3 : Update existing note : PUT 'api/notes/updatenote:id' Requires Auth
Login
router.put('/updatenote/:id',fetchuser,async (req,res)=>{

    try {
        const {title,description,tag} = req.body;

    //create newnote object
    const newnote = {};
    if(title) {newnote.title = title};
    if(description) {newnote.description = description};
    if(tag) {newnote.tag = tag};

    //Find note to be updated
    let note = await Notes.findById(req.params.id); // params.id -> from url
    if(!note) {return res.status(404).send("Not Found")}

    if(note.user.toString() !== req.user.id){
        return res.status(401).send("Not Alllowed");
    }
```

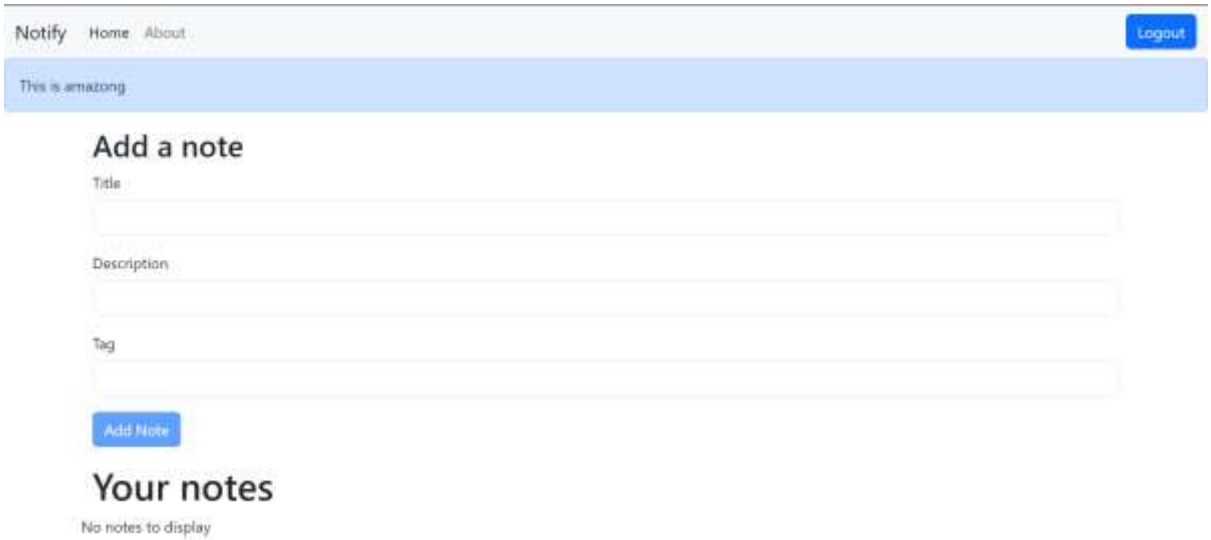
```
    note = await Notes.findByIdAndUpdate(req.params.id,{$set : newnote}, {new
: true})
    // Usually when you perform update operations in mongoose, it returns the
previous state of the document (before it was updated) and not the updated
one. By setting "new" to true in the third argument of the object in
"findByIdAndUpdate()", we tell mongoose to return the updated state of the
object instead of its default behaviour
    res.json(note)
    } catch (error) {
        console.log(error.message)
        res.status(500).send("Internal Server Error Occured");
    }


})


//ROUTE4 : Delete note : DELETE 'api/notes/deletenote:id' Requires Auth Login
router.delete('/deletenote/:id',fetchuser,async (req,res)=>{

    try {

    //Find note to be deleted
    let note = await Notes.findById(req.params.id); // params.id -> from url

    if(!note) {return res.status(404).send("Not Found")}

    //Allow only if user owns this note
    if(note.user.toString() !== req.user.id){
        return res.status(401).send("Not Alllowed");
    }

    note = await Notes.findByIdAndDelete(req.params.id)
    // Usually when you perform update operations in mongoose, it returns the
previous state of the document (before it was updated) and not the updated
one. By setting "new" to true in the third argument of the object in
"findByIdAndUpdate()", we tell mongoose to return the updated state of the
object instead of its default behaviour
    // res.json(note)
    res.json({"Success" : "Note deleted", note : note})


    }
    catch (error) {
        console.log(error.message)
        res.status(500).send("Internal Server Error Occured");
    }
```

```
})

module.exports = router
```

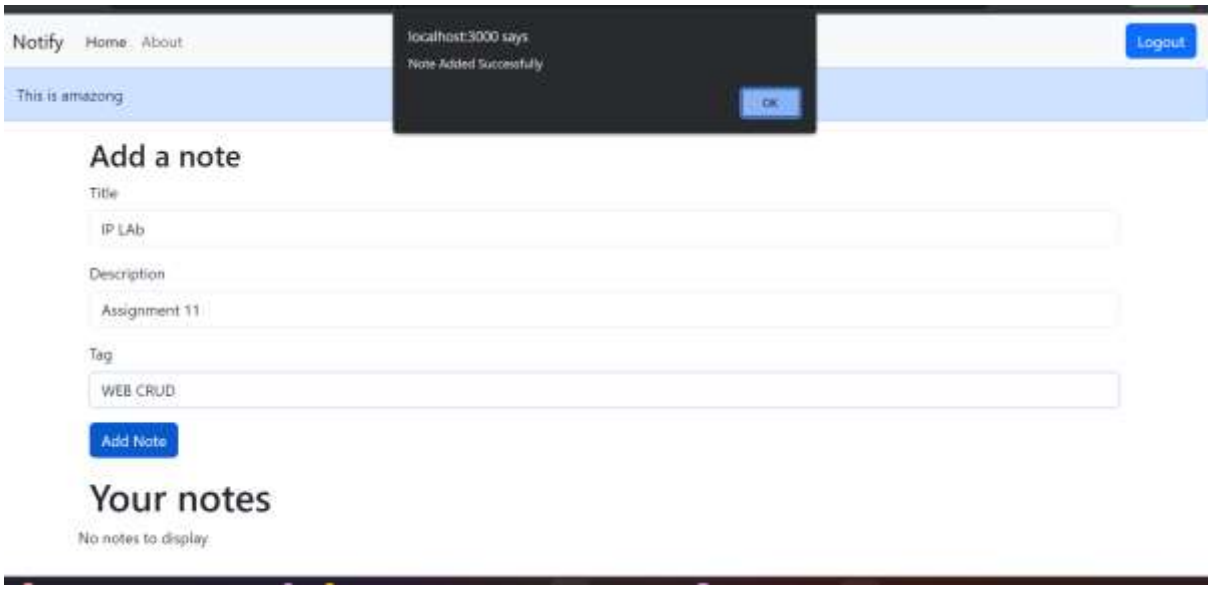
**Notes.js Schema:**

```
const mongoose = require("mongoose")
const { Schema } = mongoose;
const {body,validationResult}= require("express-validator")


const NotesSchema = new Schema({
    user :{
        //works as foreign key
        type : mongoose.Schema.Types.ObjectId,
        ref : 'user'
    },
    title : {
        type:String,
        required : true
    },
    description :{
        type:String,
        required : true,
    },
    tag : {
        type: String,
        default:"General"
    },
    date : {
        type: Date,
        default : Date.now
    },
});



module.exports = mongoose.model('notes',NotesSchema) ;
```

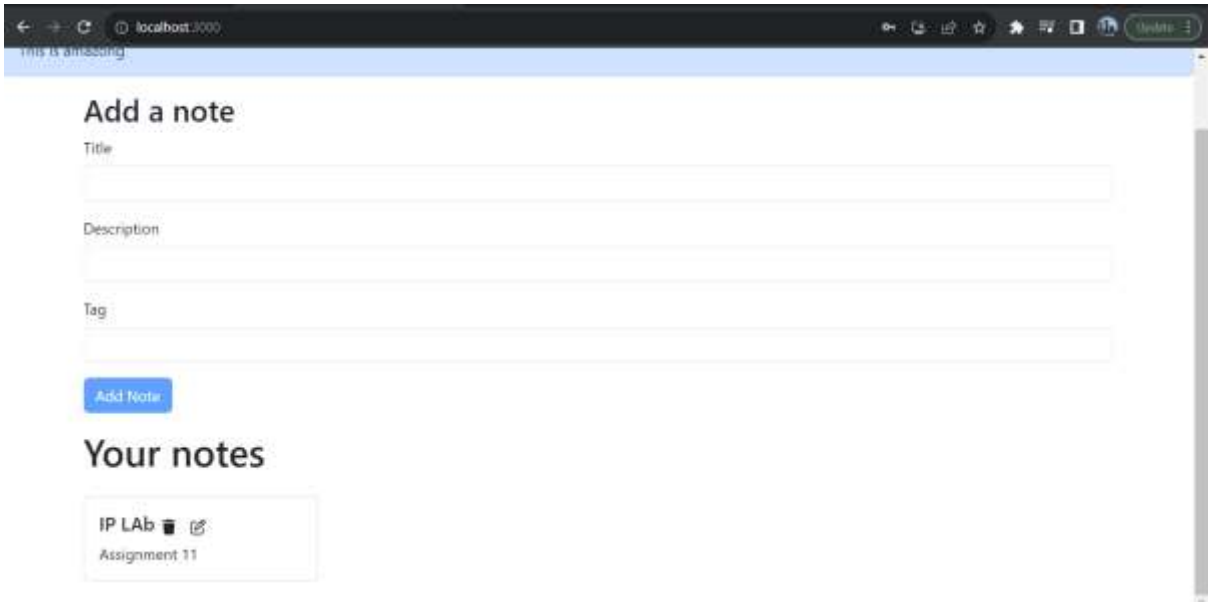
**Output :**



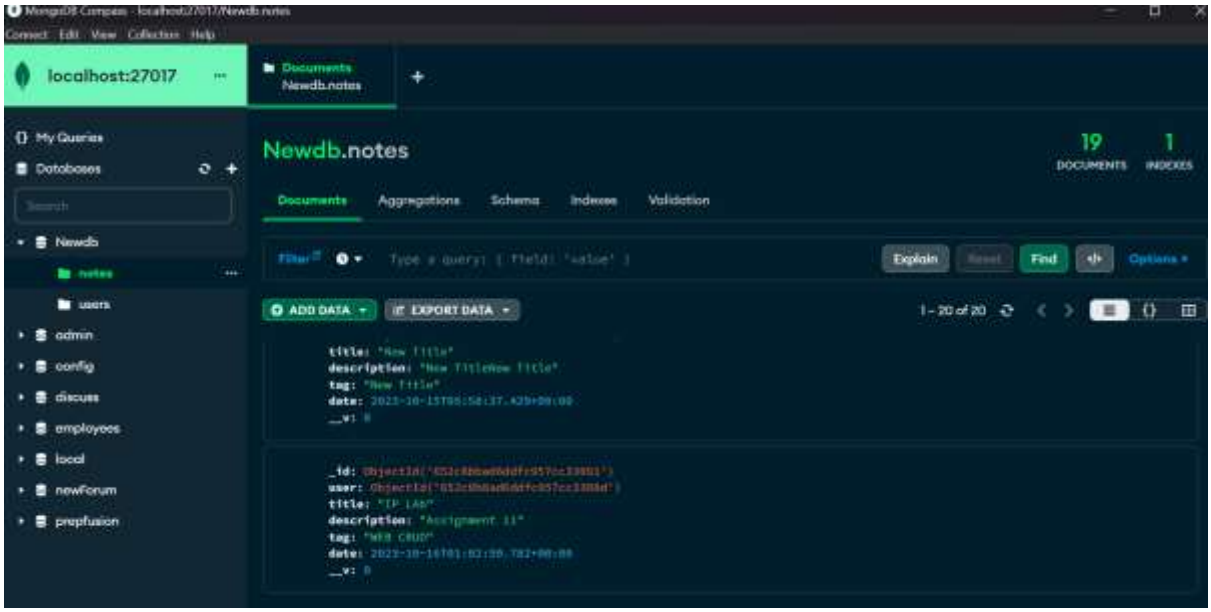

Adding a Note:



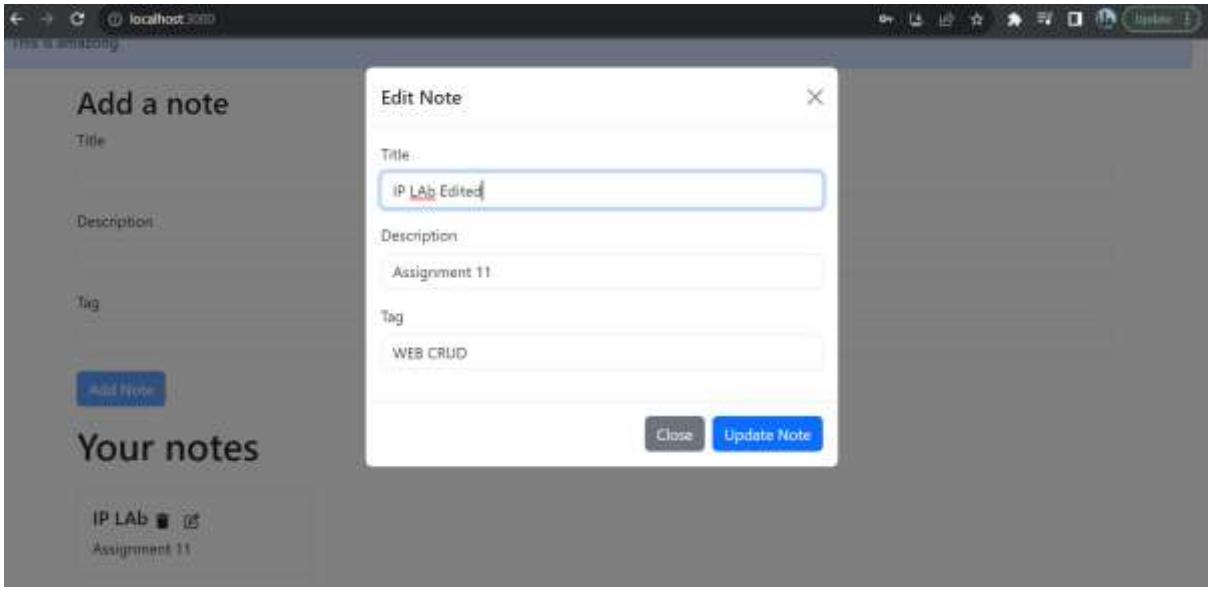
Reflects on Frontend: Fetched from Backend Database



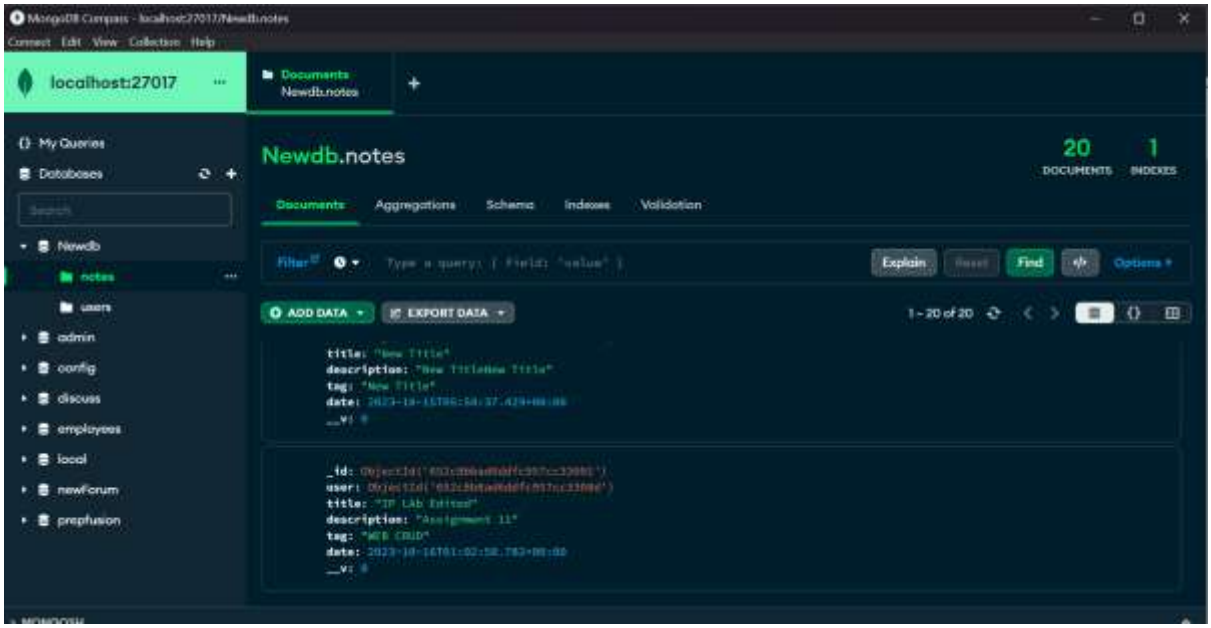

Showed in TheDatabase



Editing the creeated note




Can see in database the note details are edited:



On deleting the note: