Exercise

- Use browser first
- Use **postman** for testing your api requests

```
const people = [
{ id: 1, name: 'john' },
{ id: 2, name: 'peter' },
{ id: 3, name: 'susan' },
{ id: 4, name: 'anna' },
{ id: 5, name: 'emma' },
]
```

Step-by-step crud operations nodejs+express

```
const express= require('express')
const app=express()
let{people}= require('./data')


//static assets
app.use(express.static('./public-methods'))
//parse from data
app.use(express.urlencoded({extended:false}))
//parse json
app.use(express.json())
```

Step-by-step crud operations nodejs+express

```
//read data (GET)
app.get('/api/people',(req,res)=>{
res.status(200).json({success: true, data:people})
})

//the below line help with fetching request body
// (POST)
app.post('/login',(req,res)=>{
    console.log("body: ", req.body)
    const {name}= req.body
    if(name){
        return res.status(200).send(`welcome ${name}`)
    }
    res.status(401).send('please provide data')
})
```

Step-by-step crud operations nodejs+express

```
//POST for javascript
app.post('/api/people',(req,res)=>{
    console.log(req.query)
    const {name}= req.query
    console.log(name)
    if(!name){
        return res
        .status(400)
        .json({success:false, msg:'please provide name'})
        }
    res.status(201).json({success:true,person:name})
})
```

Step-by-step crud operations nodejs+express

```
//Update data (PUT)
app.put('/api/people/:id',(req,res)=>{
    const {id}=req.params
    const {name}=req.body
    console.log(id,name)
    const person=people.find((person)=>person.id===Number(id))
    if(!person){
        return res.status(404).json({success:false, msg:`no person found
    ${id}`})
    }
    const newPeople = people.map((person)=>{
        if(person.id ===Number(id)){
            person.name=name
            }
        return person
        })
    res.status(200).json({success:true,data:newPeople})

})
```

Step-by-step crud operations nodejs+express

```
//Delete data (DELETE)
app.delete('/api/people/:id',(req,res)=>{
    const {id} = req.params;
    const person= people.find((person)=> person.id===Number(id))

    if(!person){
        return res.status(404).json({success:false, msg:`no person found
        ${id}`})

    }
    const newPeople = people.filter((person)=>person.id != Number(id))
    return res.status(200).json({success:true, data:newPeople})
})
```

Node js + Mongodb

Lets start with creating our Mongodb cluster

- Go to mongodb website
- Create an account
- Create a new project (use free shared option)
- Allow access from any ip 0.0.0.0/0 (for now)
- Add admin access details (username and password)
- Cluster0 created
- Create a database – name it AppDB
- (optional) create a collection – products
- Choose connect using vscode (copy the path)

notes

- Mongodb ←→ server ←→ client
- Server side must handle connection with mongodb and requests from clients
- Public and views (for general purposes – view index.html, 404.html)
- Middleware (built in , custom, 3rd party middleware's)
  - Example: express.static, express.json / logger /
- Cookie-parser
- CORS (options)
- dotenv (allows access to environment variables) --> .env file creation
- Mongoose (for mongodb connections)

See the big picture of our server folder structure

- Config
  - dbConn.js *(connect to mongoDB using mongoose)*
  - corsOptions.js
- Controllers
  - usersControllers.js *(this will be used to control all crud operations for user creations)*
- Models
  - User.js *(the structure details of our user collection)*
- Routes
  - Root.js
  - userRoutes.js
- Public
  - *Anything that will be shared in the route path (.css, logos)*
- Views
  - Index.html, 404.html
- .env *(save important details)*
- server.js
- .gitignore
- Package.json

Step 1: set up server.js

```
require('dotenv').config()
const express= require('express')
const app= express();
const path=require('path’)
const {logger}= require('./middleware/logger')
const mongoose = require('mongoose’)

const PORT = process.env.PORT || 3501//grap the one in the cloud server
app.use(logger)
app.use(express.json())
app.use('/', express.static(path.join(__dirname,'public')))
    app.use('/', (req,res)=>{
        res.sendFile(path.join(__dirname,'views','index.html'))
    })
app.listen(PORT, ()=>{
console.log(`server running on port ${PORT}`)
})
```

Step 2: set up CORS in config folder

```
const allowedOrigins=[
'http://localhost:3000'
]

const corsOptions = {
    origin: (origin, callback)=>{
        if(allowedOrigins.indexOf(origin)!== -1 || !origin){
            callback(null,true)
        }else{
            callback(new Error('not allowed by CORS'))
        }
    },
        credentials: true,
        optionsSuccessStatus: 200
    }

module.exports = corsOptions
```

Step 2.1: set up CORS in server.js

```
const cors= require('cors')
const corsOptions = require('./config/corsOptions')

app.use(cors(corsOptions))
```

CORS ➔ (“Cross-Origin Resource Sharing.”)
allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.

Step 3: set up .env file

```
NODE_ENV= WebDevTesting
DATABASE_URI=
<username>:<password>@cluster0.l1c4zc8.mongodb.net/ServerDB?retryWrites=true&
w=majority


//in server.js
console.log(process.env.NODE_ENV)
```

<username> and <password> must be secured

Step 4: set up dbConn.js in config folder

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

const connectDB = async()=>{
   try{
      await mongoose.connect(process.env.DATABASE_URI)
   }catch (err){
      console.log(err)
   }
   }

module.exports = connectDB
```

Step 4.1: import dbConn in server.js

```
const connectDB= require('./config/dbConn’)
connectDB()

//listener
mongoose.connection.once('open',()=>{
     console.log('connected to mongo’)
     app.listen(PORT, ()=>{
          console.log(`server running on port ${PORT}`)
   })
})
mongoose.connection.on('error',err=>{
console.log(err)
})
```

Step 4.2: create User.js inside models

```
const mongoose = require('mongoose')
const userSchema = new mongoose.Schema({
    username:{
        type:String,
        required:true
        },
    password:{
        type:String,
        required:true
        },
    roles: [{
        type:String,
        default:"user"
        }],
    active:{
        type:Boolean,
        default:true
        }
    })
    module.exports= mongoose.model('User',userSchema)
```

Step 5: create userRoutes.js in routes

```
const express= require('express')
const router= express.Router()
const userController = require('../controllers/usersControllers')
router.route('/')
.get(userController.getAllUsers)
.post(userController.createNewUser)
.patch(userController.updateUser)
.delete(userController.deleteUser)


module.exports= router
```

In server.js

```
app.use('/users',require('./routes/userRoutes'))
```

Step 5.1: setup usersControllers.js in controllers

```
const User = require('../models/User')

const asyncHandler = require('express-async-handler')
const bcrypt = require('bcrypt’)
```

For any GET METHOD handler

```
const getAllUsers= asyncHandler(async(req,res)=>{

    const users= await User.find().select('-password').lean()
    if(!users?.length){
        return res.status(400).json({message: 'no users found'})
    }
    res.json(users)
})
```

Step 5.2: setup usersControllers.js in controllers

POST METHOD handler

```
const createNewUser= asyncHandler(async(req,res)=>{
    const {username, password, roles} = req.body
    //handle empty body

    const duplicate = await User.findOne({username}).lean().exec()
    if(duplicate){
        return res.status(400).json({message:"duplicate username"})
    }
    //hash password
    const hashed = await bcrypt.hash(password, 10)
    //new document to be added to the collection
    const userObj = {username,'password': hashed, roles}

    const user = await User.create(userObj) //add to User collection
    if(user){
        res.status(201).json({message:"user created"})
    }else{
        res.status(400).json({message:'invalid user data'})
    }
})
```

Step 5.2: setup usersControllers.js in controllers

PATCH METHOD handler

```
const updateUser= asyncHandler(async(req,res)=>{
    const {id,username,roles, active, password}= req.body
    //check for empty body first
    const user = await User.findById(id).exec()
    if(!user){
        return res.status(400).json({message:'user not found'})
    }
    user.username=username
    user.roles= roles
    user.active = active
    if(password){
        user.password = await bcrypt.hash(password, 10)
    }
    const updatedUser = await user.save()
    res.json({message:'updated user done!'})
})
```

Step 5.2: setup usersControllers.js in controllers

DELETE METHOD handler

```
const deleteUser= asyncHandler(async(req,res)=>{
    const {id}= req.body
    if(!id){
        return res.status(400).json({message:'user id required'})
    }
    //if you are deleting a user => all their data in other collections
    //must be deleted as well! or restrict this delete

    const user = await User.findById(id).exec()

    if(!user){
        return res.status(400).json({message:'user not found'})

    }
    const result = await user.deleteOne()
    res.json({message:`user ${result._id} deleted `})
})
```