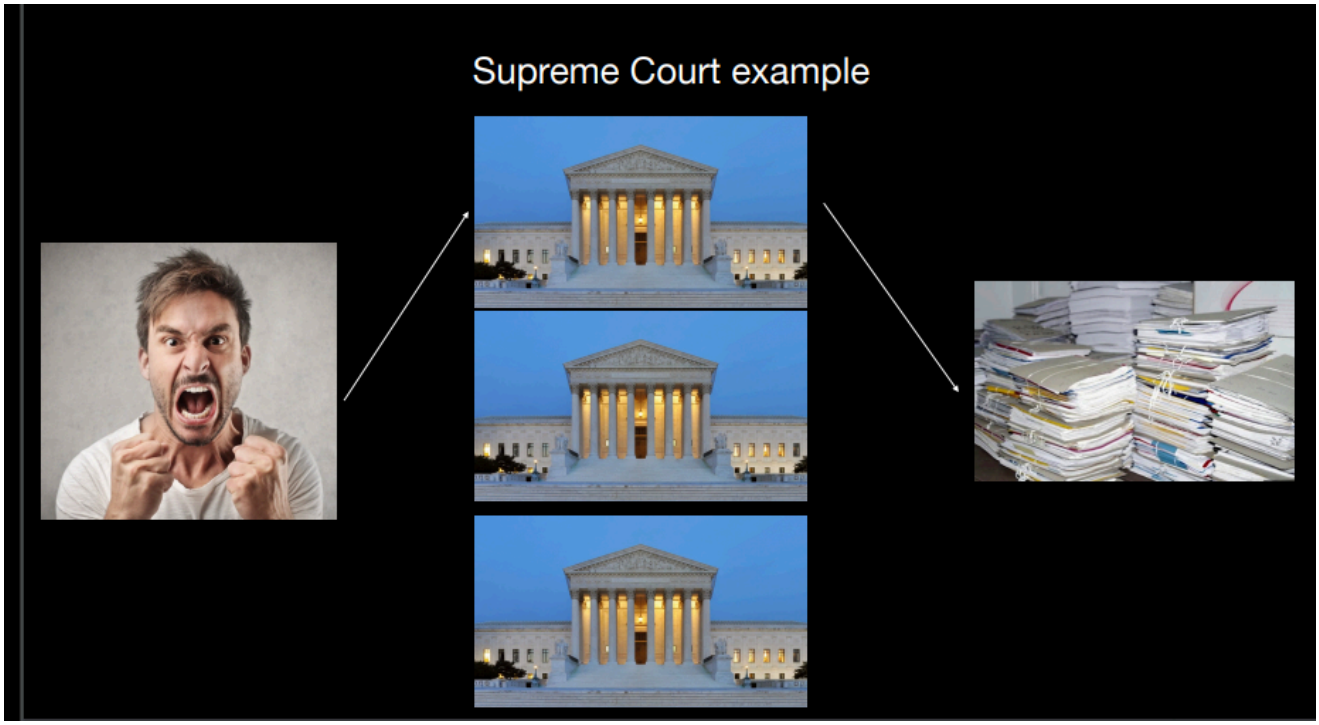
MongoDB deep dive

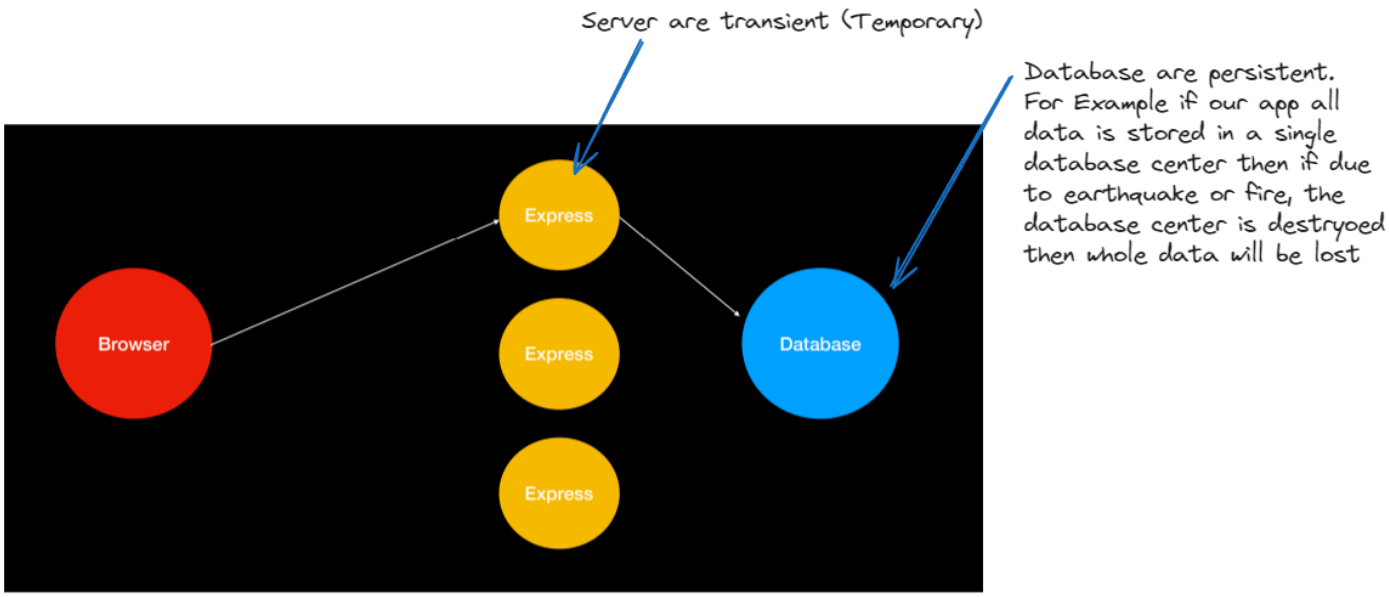
Understanding CRUD, mongoose and building an end to end authenticated app

Lets see and example:



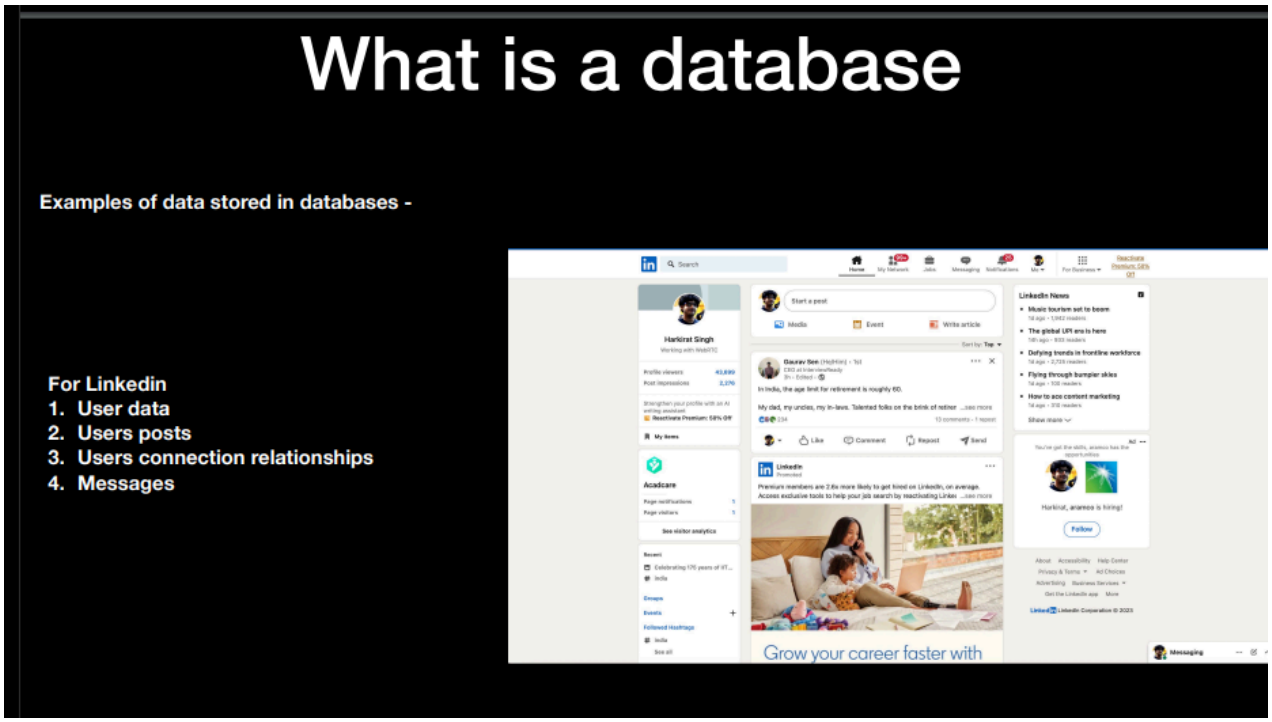
What is database?

It is place where data is stored persistently

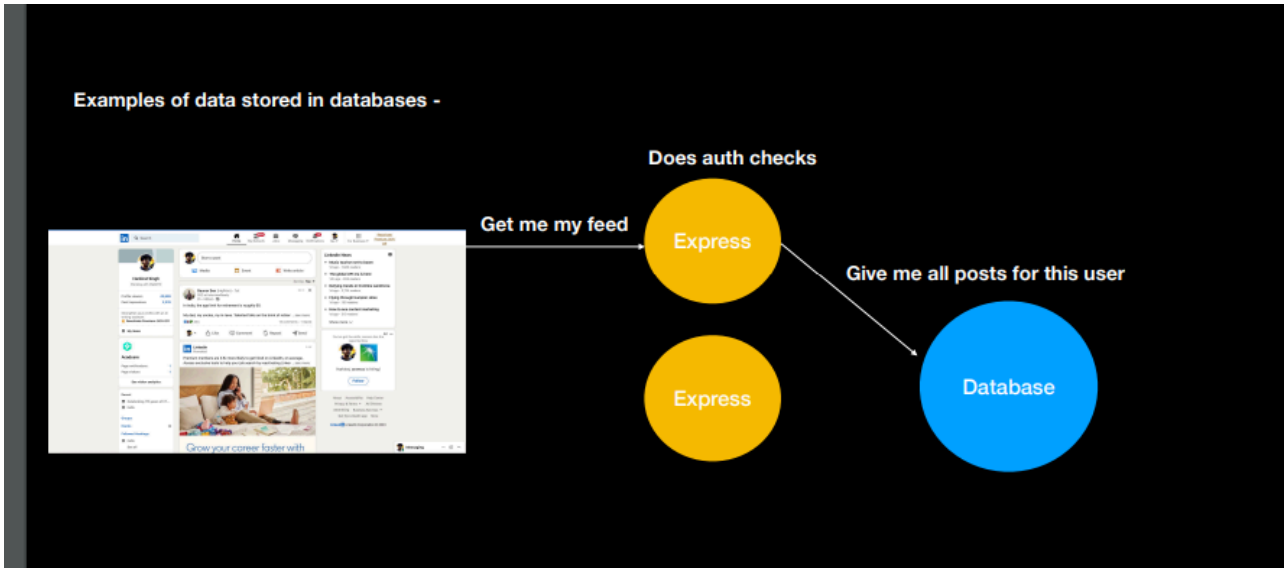


Servers autoscale.

Lets see an example of LinkedIn \

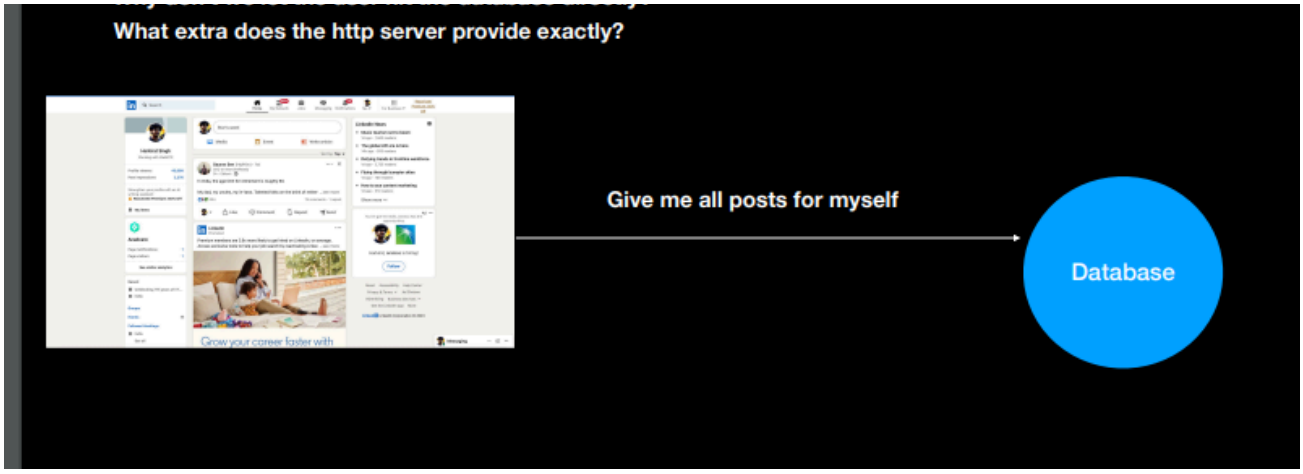


How does the request usually go by:



Why don’t we let the user hit the database directly?

What extra does the http server provide exactly?



1. Database were created using protocols that browsers don’t understand
2. **Databases don’t have granual (**database give access to everything or nothing, Browser can ask for any data , Express server has access to all the database , hence it need to restrict the access of database (by

browser) auth check **) access as a first-class citizen. Very hard to do user-specific access in them**

3. Some databases (firebase) let you get rid of the HTTP server and try their best to provide granola access. (hence its not compulsory to always have an HTTP Server)

Databases usually allow access to 4 primitives

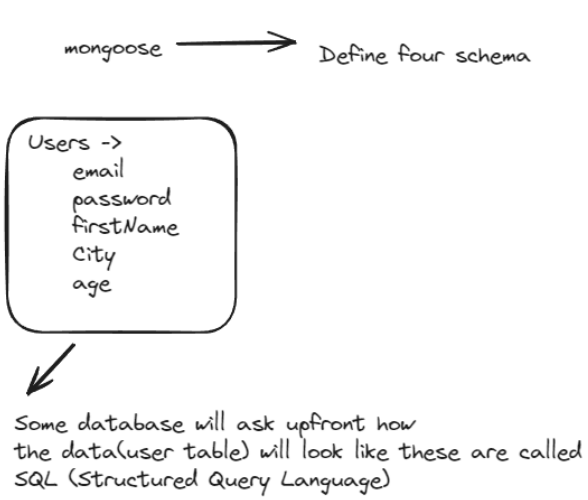
1. Create Data
2. Read Data
3. Update Data
4. Delete Data

Popularly known as **CRUD**

ODM : Library to talk to database : examples mongoose and Prisma

Let's see the API for the mongoose library
Eventually, we'll be using prisma (which is the industry standard way of doing this)

In mongoose, first you have to define the schema
This sound counter intuitive since mongodb is schemaless?
That is true, but mongoose makes you define schema for thing like autocompletions/ Validating data before it goes in DB to make sure you're doing things right Schemaless DBs can be very dangerous, using schemas in mongo makes it slightly less Dangerous.



mongoDB says send me anything you
want . We can store two document
of different schema completely in database
But SQL does'nt alllow

mongoose library says first tell me what is
your schema , one we tell them then we can insert
all the data
So basically if we add the data from our express
server then we can to follow schema strictly which
we send to mongoose library
But we still can insert random data from GUI of
the MongoDB library.

Generally in our application we need strict schema and mongoose make our application more strict

Similar to TS and JS . TS provide us type safety which give us long term benefit.

Defining Schema:

```
const UserSchema = new mongoose.Schema({
    email:String,
    password:String,
    purchasedCourses: [{
        type: mongoose.Schema.Types.ObjectId,
        // ObjectID : mongoDB way to give randomised ID
        ref:"Course"
    }]
})
const CourseSchema = new mongoose.Schema({
    title : String,
    price : 5999
})
```

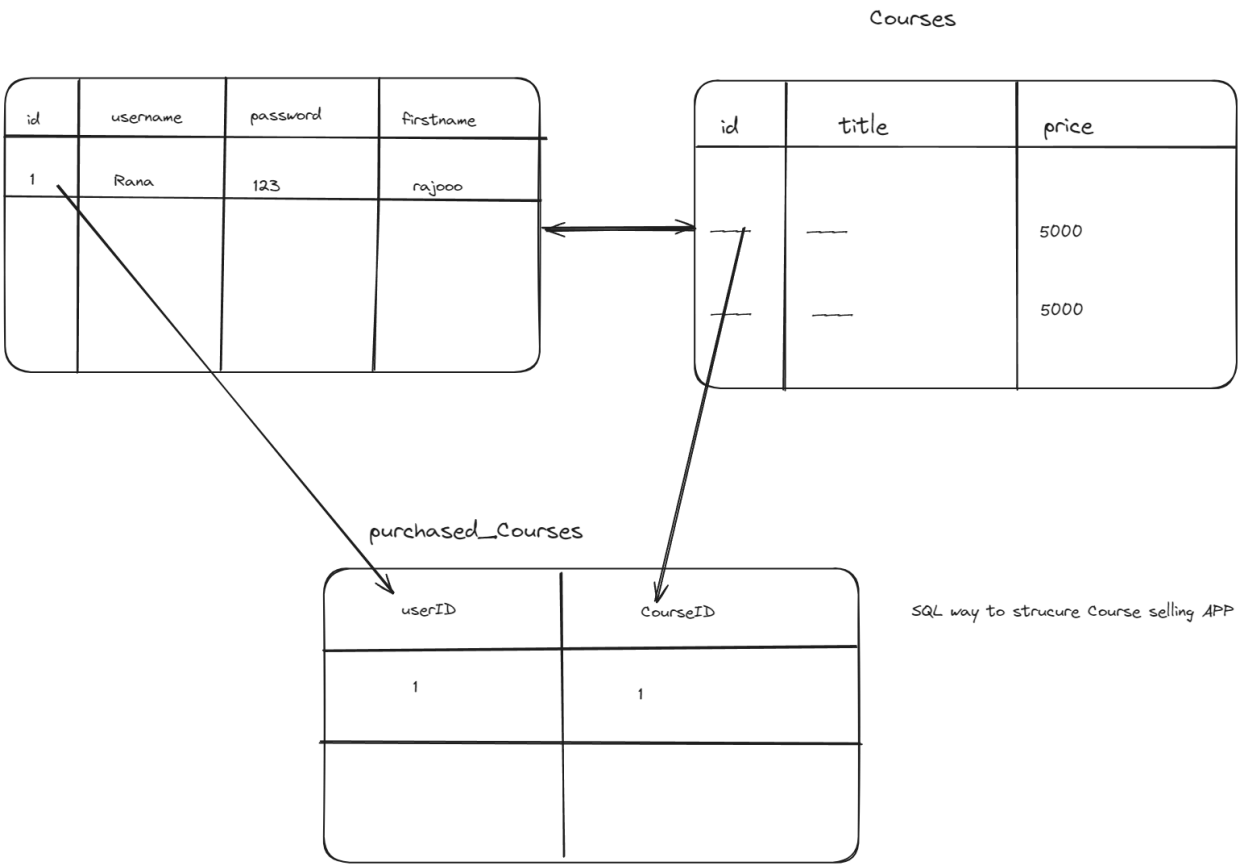
```
const User = mongoose.model('User',UserSchema);
const Course = mongoose.model('Course',CourseSchema);
```

MongoDB allow us to create complex objects For example:

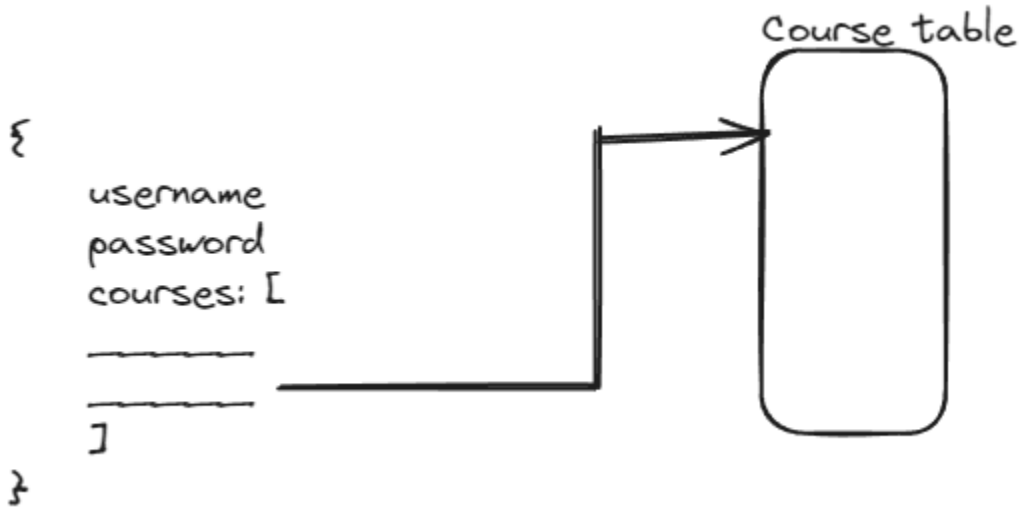
```
{
    'id': {
        'Sold' : 'ascasr123'
    },
    "name": "Thapu",
    "courses": [
        "0 to 0 Football fighting course",
        "Unborn kids boxing tutorials"
    ]
}
```

In SQL we have table of Username, age, profession etc

The way to create relationship in MongoDB is by just creating array of things.



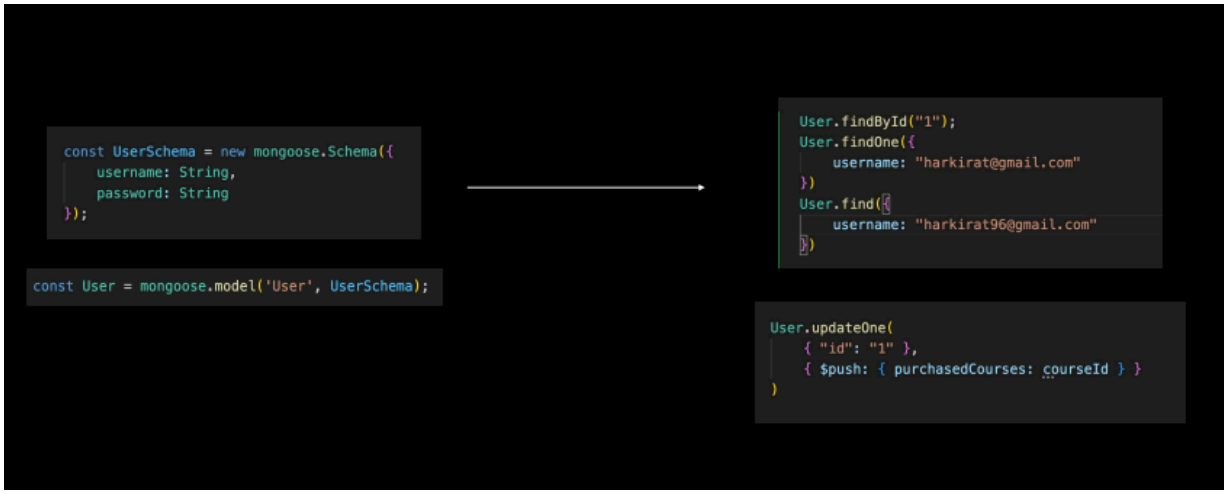
Another way(No SQL) of doing thing can be:



Create:

Entry to the UserTable

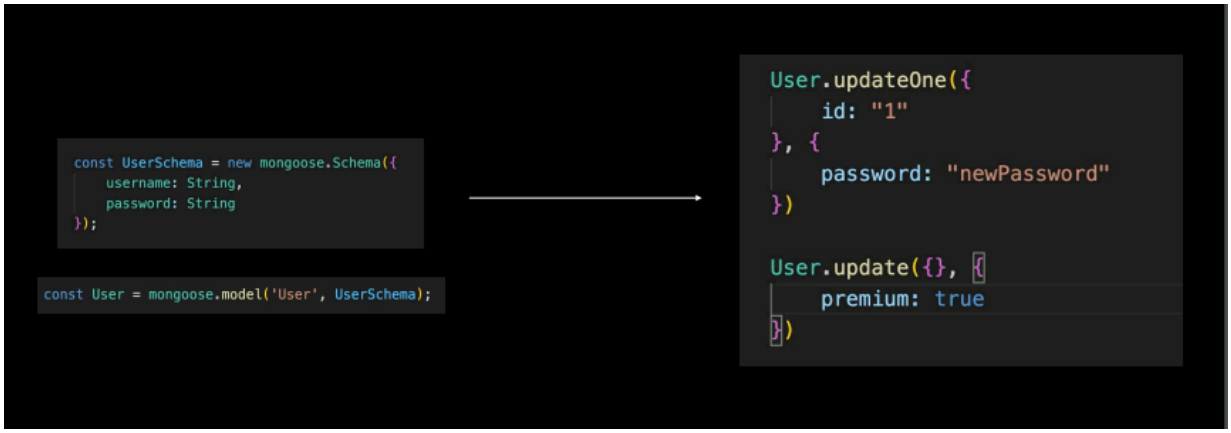
Read:



.findOne() : find a single entry

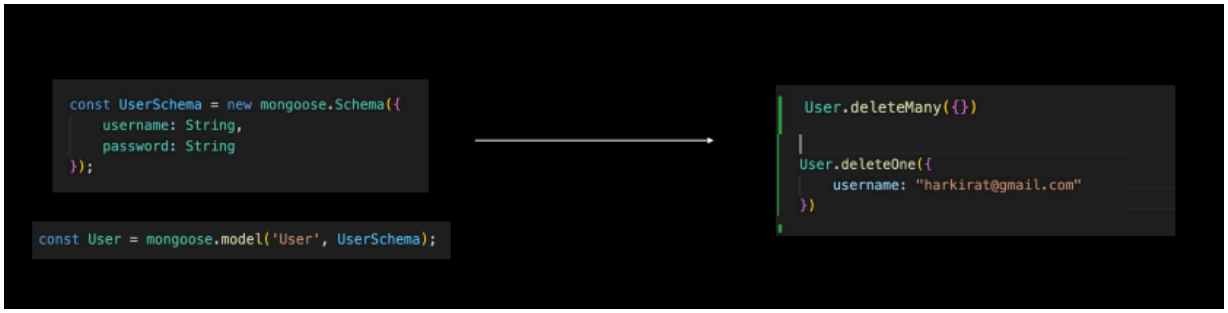
Syntax of pushing new course

Update:



Giving free access to our courses

Delete:

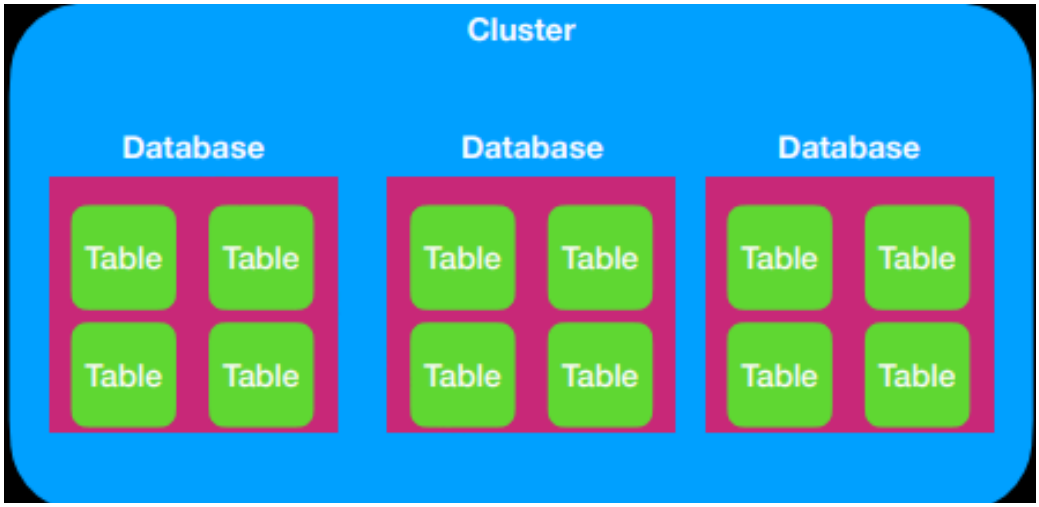


User.deleteMany()

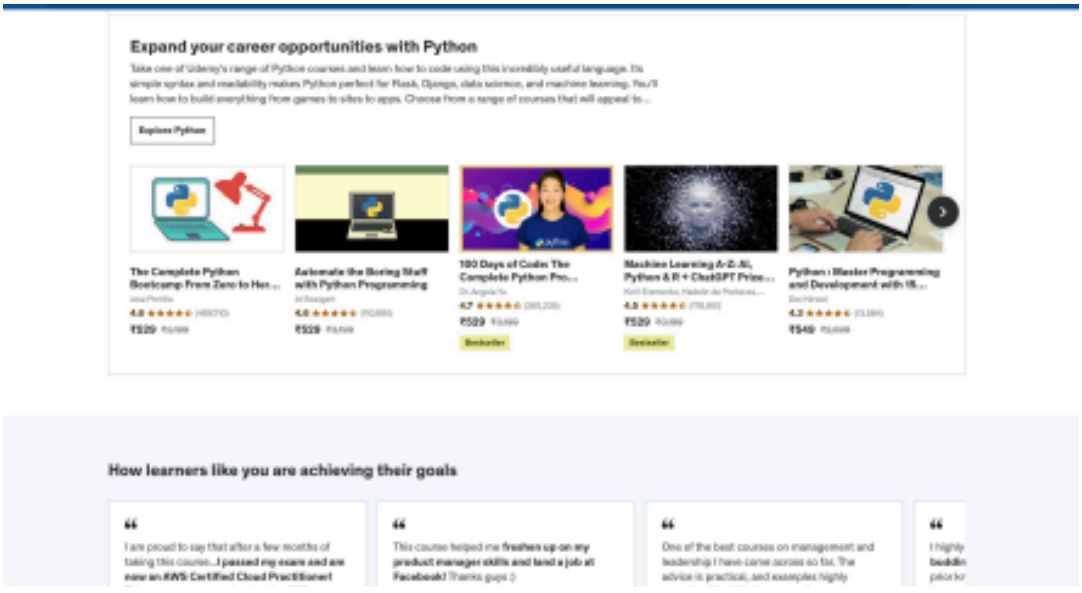
Deleting everything from the user table

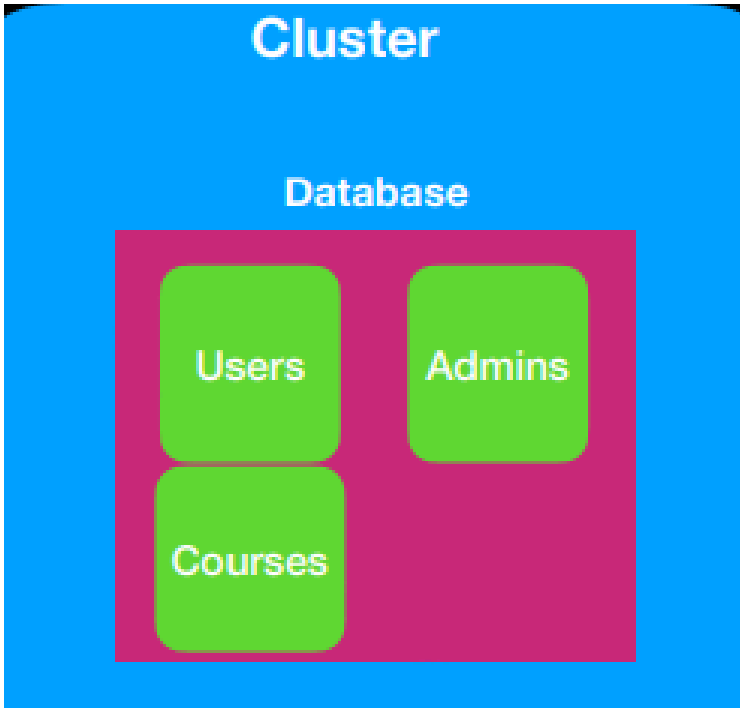
3 Jargons to know in Databases

1. Cluster
2. Database
3. Table



Lets see what would this mean for a simple course App:





User Table:

```
Users
Id | Email                | Password | Name    | Age
1  | harkirat@gmail.com   | 123123   | harkirat | 20
2  | raman@gmail.com      | kirat123 | harkirat | 22
```

Courses Table:

```
Courses
id  | Title       | Description       | Price
1.  | Full stack  | Learn Full stack  | 5000
2.  | Web3        | Learn Web3.       | 3999
```

Purchases Table:

Purchases

user_id	course_id	timestamp	payment_ref
1.	1	02/12/2024.	pay_123123
2.	1	02/12/2024.	pay_331213

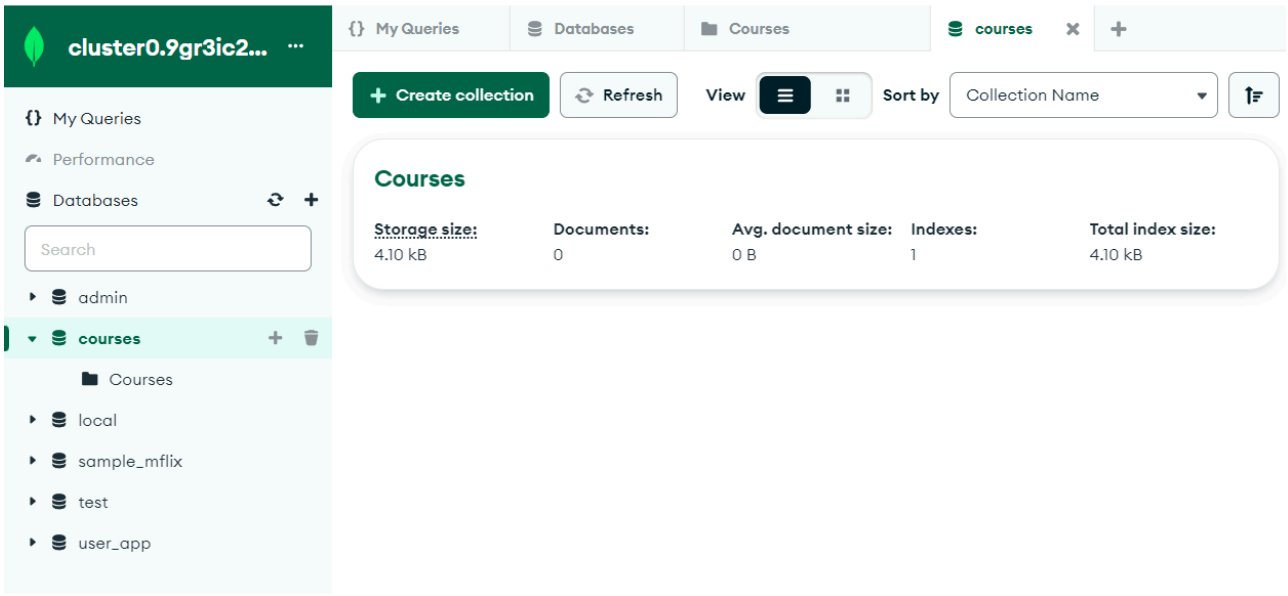
ASSIGNMENT

```
## Create a course selling website

### Description
You need to implement a course selling app. Make sure you setup your own
mongodb instance before starting.
mongodb+srv://admin:<password>@cluster0.9gr3ic2.mongodb.net/

It needs to support two types of users -
1. Admins
2. Users

Admins are allowed to sign up, create courses.
Users are allowed to sign up, view courses, purchase courses.
This in the real world would translate to an app like udemy.

This one doesn't use authentication the right way. We will learn how to do
that in the next assignment.
For this one, in every authenticated requests, you need to send the
username and password in the headers (and not the jwt).
This is the reason why this assignment doesn't have a sign in route.

You need to use mongodb to store all the data persistently.

## Routes
### Admin Routes:
- POST /admin/signup
```

```
  Description: Creates a new admin account.
  Input Body: { username: 'admin', password: 'pass' }
  Output: { message: 'Admin created successfully' }
- POST /admin/courses
  Description: Creates a new course.
  Input: Headers: { 'username': 'username', 'password': 'password' },
Body: { title: 'course title', description: 'course description', price:
100, imageLink: 'https://linktoimage.com' }
  Output: { message: 'Course created successfully', courseId: "new course
id" }
- GET /admin/courses
  Description: Returns all the courses.
  Input: Headers: { 'username': 'username', 'password': 'password' }
  Output: { courses: [ { id: 1, title: 'course title', description:
'course description', price: 100, imageLink: 'https://linktoimage.com',
published: true }, ... ] }

### User routes
- POST /users/signup
  Description: Creates a new user account.
  Input: { username: 'user', password: 'pass' }
  Output: { message: 'User created successfully' }
- GET /users/courses
  Description: Lists all the courses.
  Input: Headers: { 'username': 'username', 'password': 'password' }
  Output: { courses: [ { id: 1, title: 'course title', description:
'course description', price: 100, imageLink: 'https://linktoimage.com',
published: true }, ... ] }
- POST /users/courses/:courseId
  Description: Purchases a course. courseId in the URL path should be
replaced with the ID of the course to be purchased.
  Input: Headers: { 'username': 'username', 'password': 'password' }
  Output: { message: 'Course purchased successfully' }
- GET /users/purchasedCourses
  Description: Lists all the courses purchased by the user.
  Input: Headers: { 'username': 'username', 'password': 'password' }
  Output: { purchasedCourses: [ { id: 1, title: 'course title',
description: 'course description', price: 100, imageLink:
'https://linktoimage.com', published: true }, ... ] }
```

Goal of sign-in is to send token or jwt

We sign in to MongoDB , and connect our cluster to the MongoDBCompass
Then we will create a new data base by the name of courses and inside it we will have a table/colllection by the name of Courses.



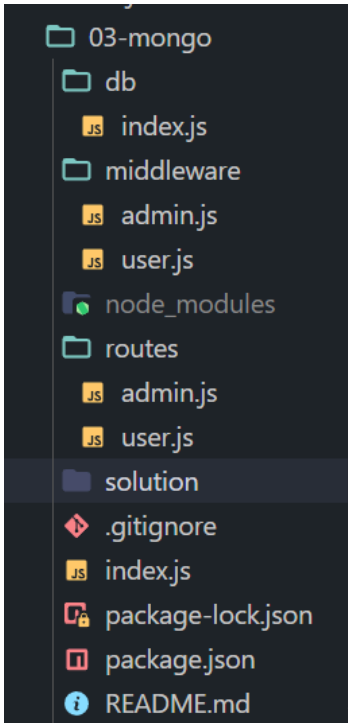
Starting Code:

index.js

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const adminRouter = require("./routes/admin")
const userRouter = require("./routes/user");

// Middleware for parsing request bodies
app.use(bodyParser.json());
// all the admin go here
app.use("/admin", adminRouter)
app.use("/user", userRouter)
const PORT = 3000;
```

```
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

File Structure



db > index.js

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

// Connect to MongoDB
mongoose.connect('your-mongodb-url');

// Define schemas
const AdminSchema = new mongoose.Schema({
    // Schema definition here
});

const UserSchema = new mongoose.Schema({
    // Schema definition here
});
```

```
const CourseSchema = new mongoose.Schema({
    // Schema definition here
});

// tables required
const Admin = mongoose.model('Admin', AdminSchema);
const User = mongoose.model('User', UserSchema);
const Course = mongoose.model('Course', CourseSchema);

module.exports = {
    Admin,
    User,
    Course
}
```

middleware > admin.js

```
// Middleware for handling auth
function adminMiddleware(req, res, next) {
    // Implement admin auth logic
    // You need to check the headers and validate the admin from the admin
DB. Check readme for the exact headers to be expected
}
// does the validation logic

module.exports = adminMiddleware;
```

middleware > user.js

```
function userMiddleware(req, res, next) {
    // Implement user auth logic
    // You need to check the headers and validate the user from the user
DB. Check readme for the exact headers to be expected
}

module.exports = userMiddleware;
```

routes > admin.js

```
const { Router } = require("express");
```

```
const adminMiddleware = require("../middleware/admin");
const router = Router();

// doesn not mean this handle the /signup endpoint
// But it handles /admin/signup
// Because in the main index.js file we have used app.use()
// Admin Routes
router.post('/signup', (req, res) => {
    // Implement admin signup logic
});
// protected by adminMiddleware
// any request for /admin/courses will reach here
router.post('/courses', adminMiddleware, (req, res) => {
    // Implement course creation logic
});
router.get('/courses', adminMiddleware, (req, res) => {
    // Implement fetching all courses logic
});
module.exports = router;
```

routes > user.js

```
const { Router } = require("express");
const router = Router();
const userMiddleware = require("../middleware/user");
// User Routes
// one way to structure the express app
router.post('/signup', (req, res) => {
    // Implement user signup logic
});
router.get('/courses', (req, res) => {
    // Implement listing all courses logic
});
router.post('/courses/:courseId', userMiddleware, (req, res) => {
    // Implement course purchase logic
});
router.get('/purchasedCourses', userMiddleware, (req, res) => {
    // Implement fetching purchased courses logic
});
module.exports = router
```

To run this application

- **npm install**
- **node index.js**

Lets first define Our database schema.

db/index.js

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

// Connect to MongoDB
mongoose.connect('mongodb+srv://admin:KmCihXn0l1podXRj@cluster0.9gr3ic2.mo
ngodb.net/');

// Define schemas
const AdminSchema = new mongoose.Schema({
    // Schema definition here
    username: String,
    password: String
});

const UserSchema = new mongoose.Schema({
    // Schema definition here
    username: String,
    password: String,
    purchasedCourses: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Course'
        // it refer to course table
    }]
});

const CourseSchema = new mongoose.Schema({
    // course Schema definition here
    title: String,
    description: String,
    imageLink: String,
    price:Number
});
```

```
// tables required
const Admin = mongoose.model('Admin', AdminSchema);
const User = mongoose.model('User', UserSchema);
const Course = mongoose.model('Course', CourseSchema);

module.exports = {
    Admin,
    User,
    Course
}
```

middleware/admin.js

```
const {Admin} = require("../db");

// Middleware for handling auth
function adminMiddleware(req, res, next) {
    // Implement admin auth logic
    // You need to check the headers and validate the admin from the admin
DB.
    // Check readme for the exact headers to be expected
    const username = req.headers.username; //nishu123@gmail.com
    const password = req.headers.password;// Hakunamata
    // I have to check does this username actually exist in the admin
database
    // Step 1: I need to get the admin model
    Admin.findOne({
        username: username,
        password: password
    })//we can use async await syntax
    .then(function(value){
        if(value){ // if value exist then
            next();
             //If there exists a user with the above username and password
inside the database then we call next or basically control reaches next,
(inside routes/admin.js)
        }else{
            // couldn't found single user
            // possibility that username was matched but the user had
different password
```

```
            res.status(403).json({
                msg: 'User doesnt exist'
            })
            // only able to access if they had their sign up
        }
    })
}
// does the validation logic

module.exports = adminMiddleware;
```

middleware/user.js

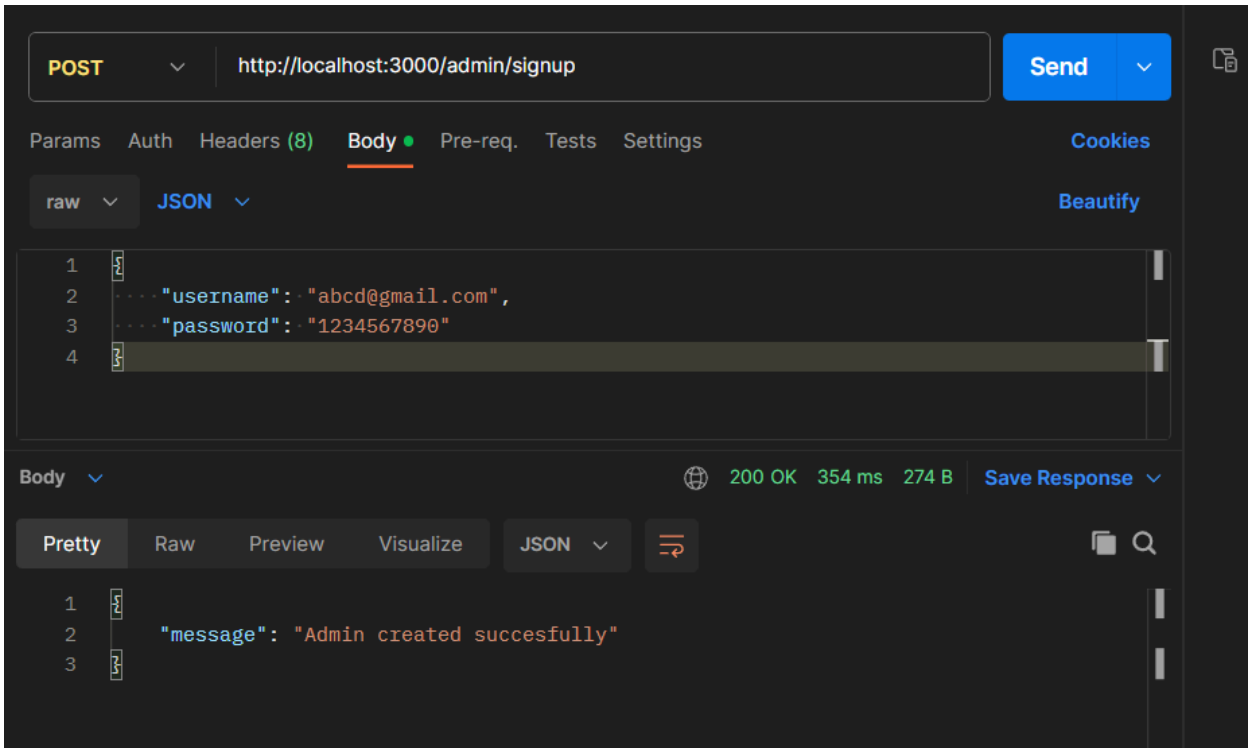
```
const {  User } = require("../db");

function userMiddleware(req, res, next) {
    // Implement user auth logic
    // You need to check the headers and validate the user from the user
DB. Check readme for the exact headers to be expected
    const username = req.headers.username;
    const password = req.headers.password;

    User.findOne({
        username:username,
        password: password
    })
    .then(function(value){
        if(value){
            next();
        }else{
            res.status(403).json({
                msg:"User doesn't exist"
            })
        }
    })
}

module.exports = userMiddleware;
```

routes/admin.js

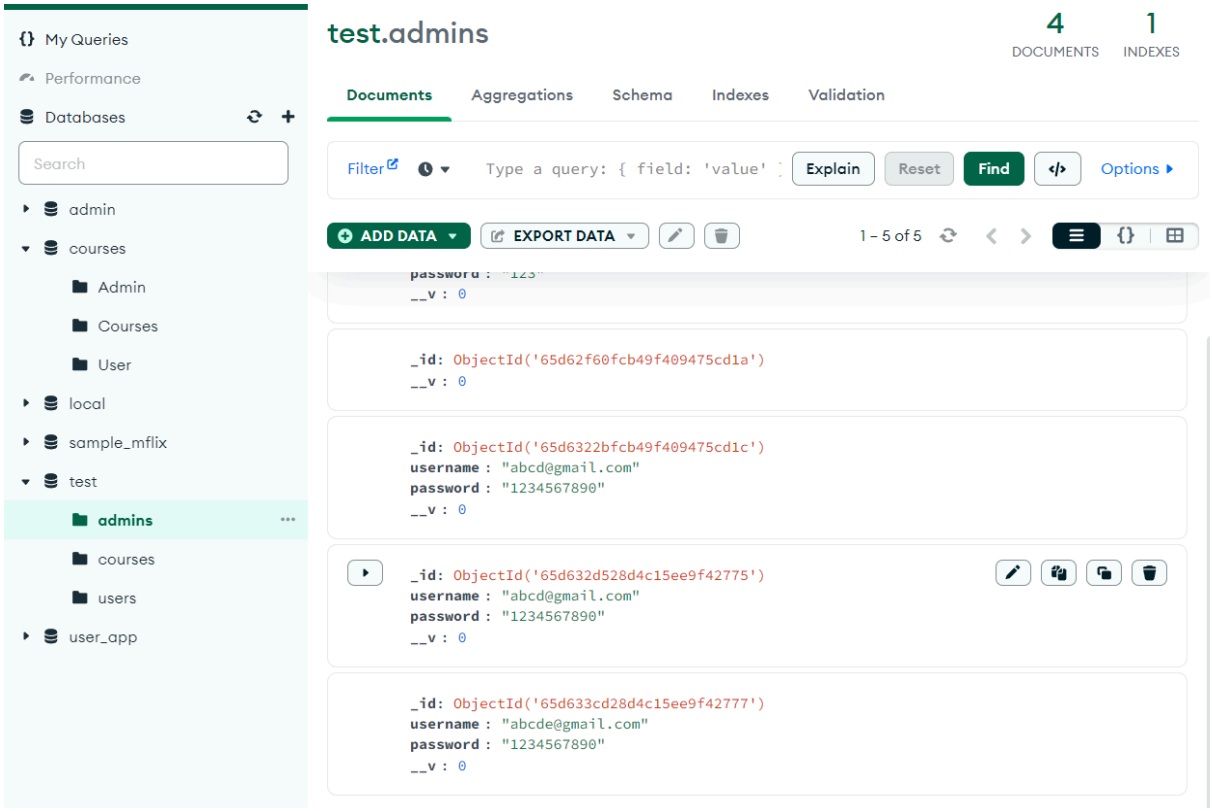
```
const { Router } = require("express");
const adminMiddleware = require("../middleware/admin");
const { Admin } = require("../db");
const router = Router();

// doesn not mean this handle the /signup endpoint
// But it handles /admin/signup
// Because in the main index.js file we have used app.use()
// Admin Routes
router.post('/signup',async (req, res) => {
    // Implement admin signup logic
    // Readme file must be read
    const username = req.body.username;
    const password = req.body.password;
    // check if a user with this username already exist , it it exist then
we must stop this request

    await Admin.create({
        username: username,
        password: password
    })
    res.json({
        message:"Admin created succesfully"
    })
    // .then(function(){
    //     res.json({
    //         message: 'Admin created successfully'
    //     })
    // })
    // .then() ideally we must await so that there was no network issue
etc.
    // .catch(function(){
// to print the error
    // })
    // if they do not exist then we must insert something inside the
database.
});

// protected by adminMiddleware
```

```
// any request for /admin/courses will reach here
router.post('/courses', adminMiddleware, (req, res) => {
    // not accessible until the sign up is done
    // Implement course creation logic
});

router.get('/courses', adminMiddleware, (req, res) => {
    // Implement fetching all courses logic
});

module.exports = router;
```

Testing it on POSTMAN



How can we verify where our data is stored



It automatically created test with three collection/table

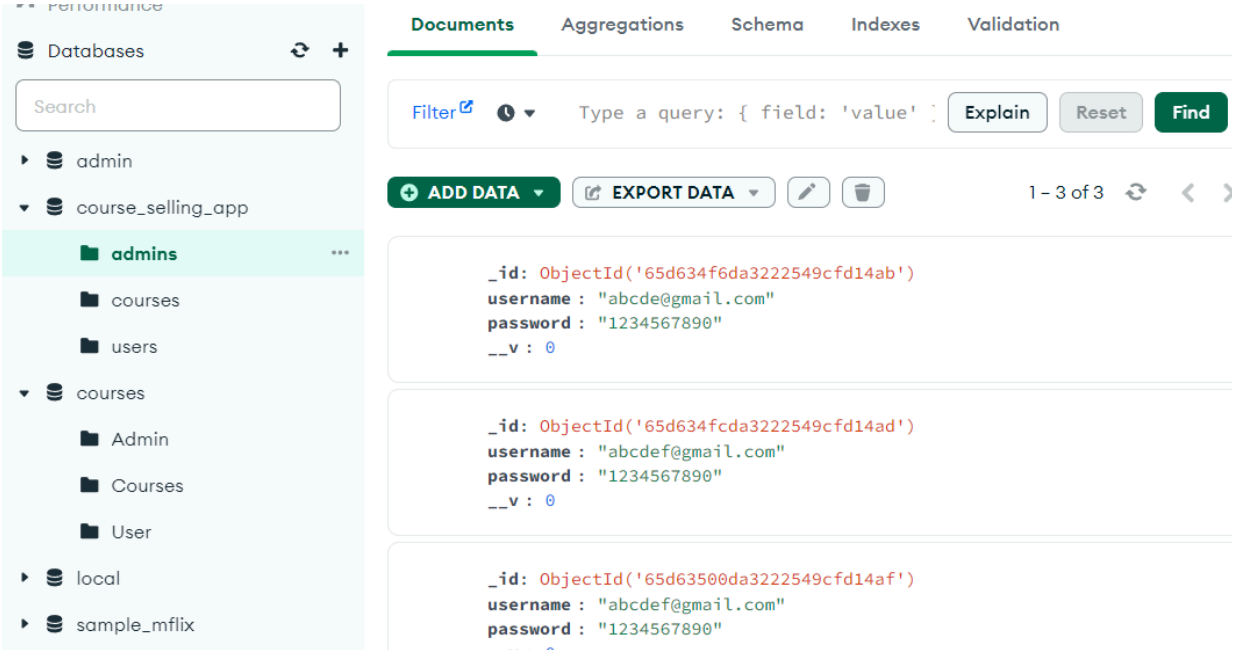
(it is default mongoDB)

If we want to choose Course DB we can do it by changing url

```
mongoose.connect('mongodb+srv://admin:KmCihXn0l1podXRj@cluster0.9gr3ic2.mongodb.net/course_selling_app');
```

Now it will be stored in the course_selling_app database

Now again send a POST request from POSTMAN



Now we will add course adding endpoint

```
router.post('/courses', adminMiddleware,async (req, res) => {
    // not accessible until the sign up is done
    // Implement course creation logic
    // need username and password
    const title = req.body.title;
    const description = req.body.description;
    const imageLink = req.body.imageLink;
    const price = req.body.price;
    // zod should be used here
    const newCourse = await Course.create({
        title,
        // title:title
        // both same if key and value are same
        description,
        imageLink,
        price
    })
    res.json({
        // in mongoDB any data we entered is given some id randomly
        // now to know the courseId of just enter data in database we need
to await
```

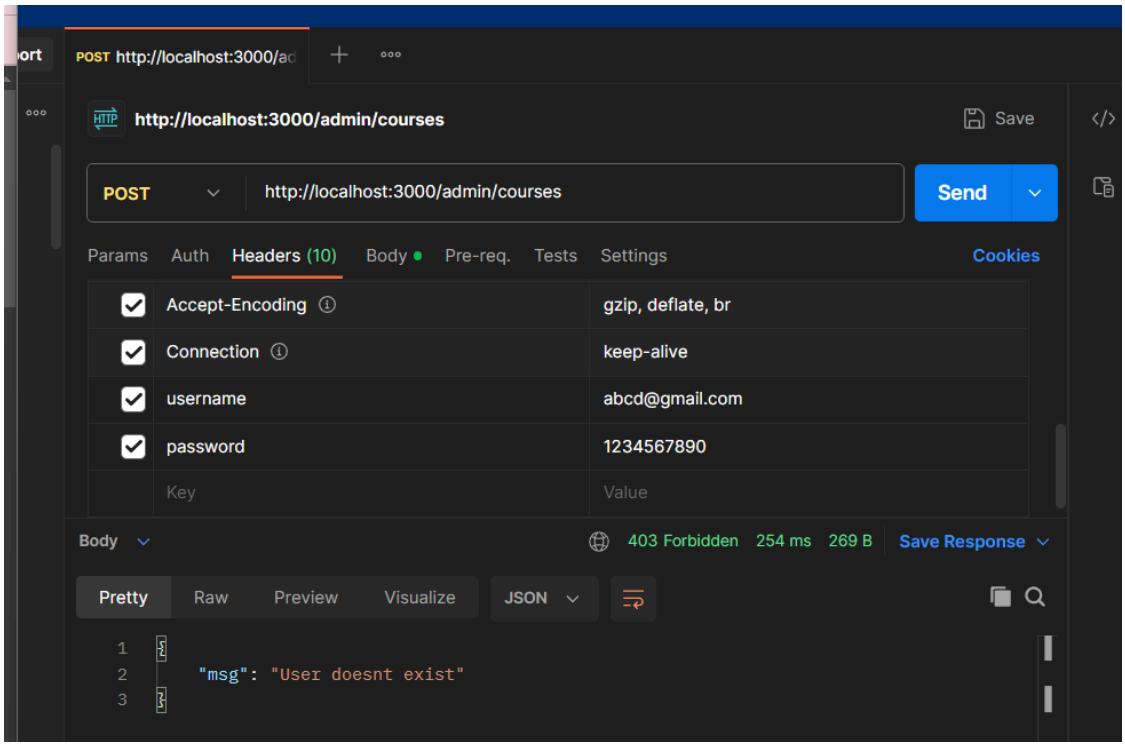
```
        message:"Course created succesfully",
        courseId: newCourse._id
    })
});
```

Lets check it on POSTMAN

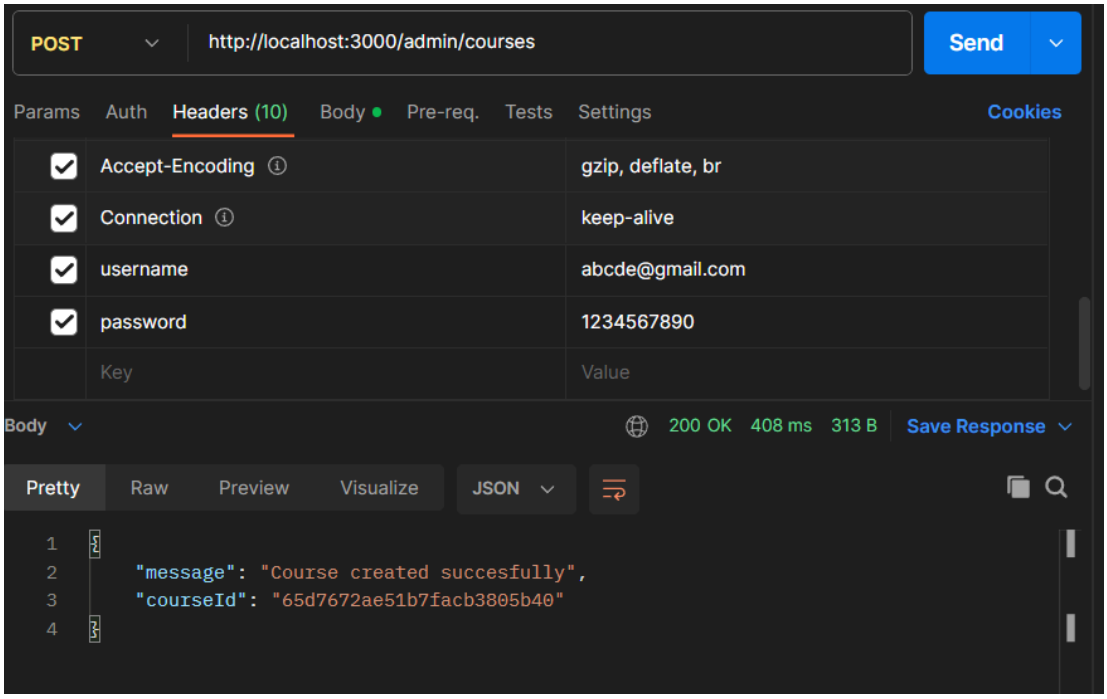
In headers we pass the username and password.

And in body we send title, description, imageLink, price

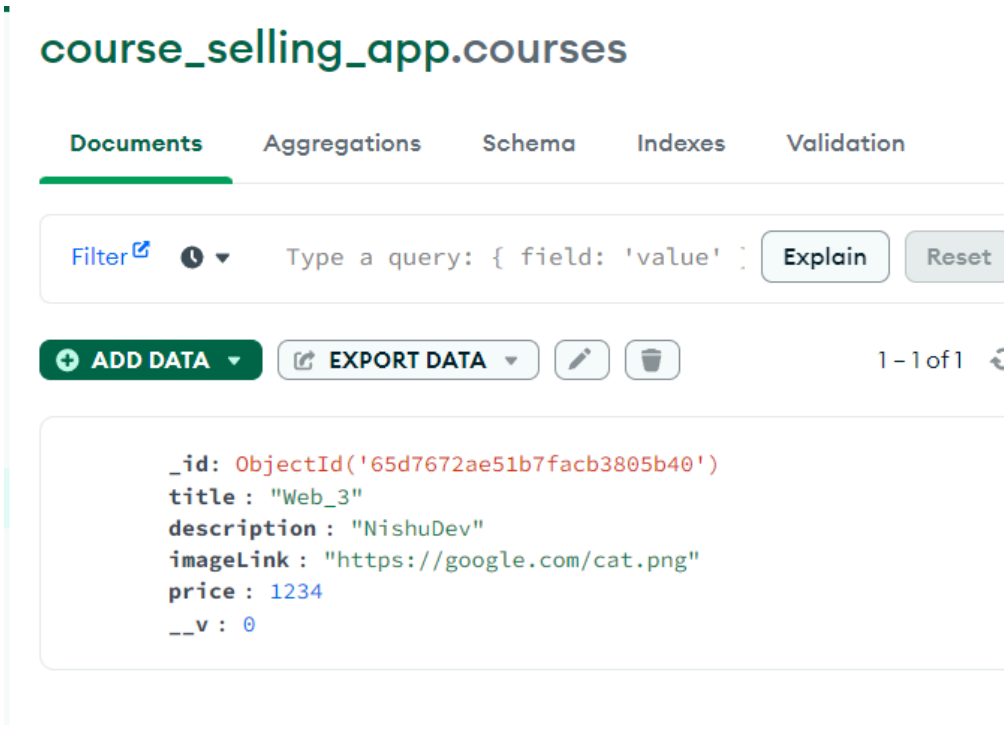
Case 1: If we enter the wrong user (user not in database admin)



Case 2: User do exist



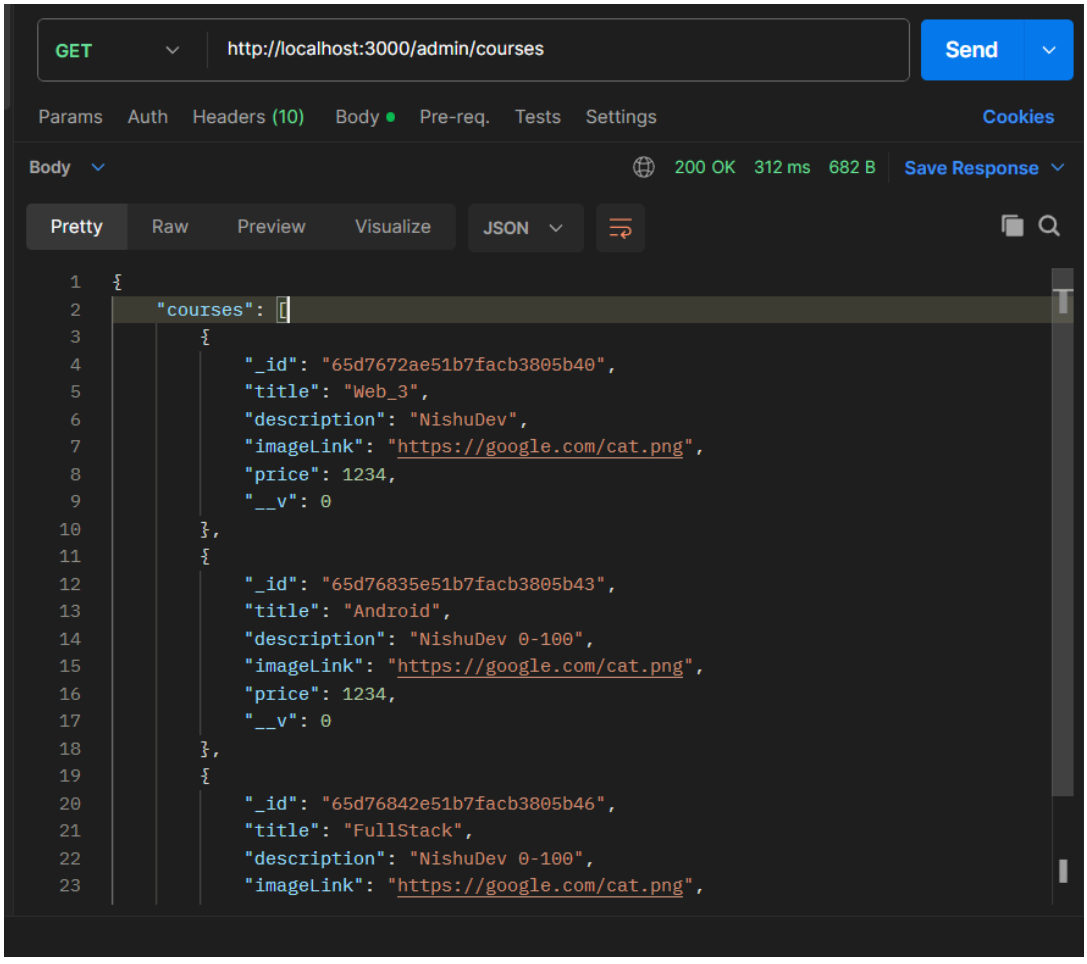
Now let's check our database in MongoDBcompass:



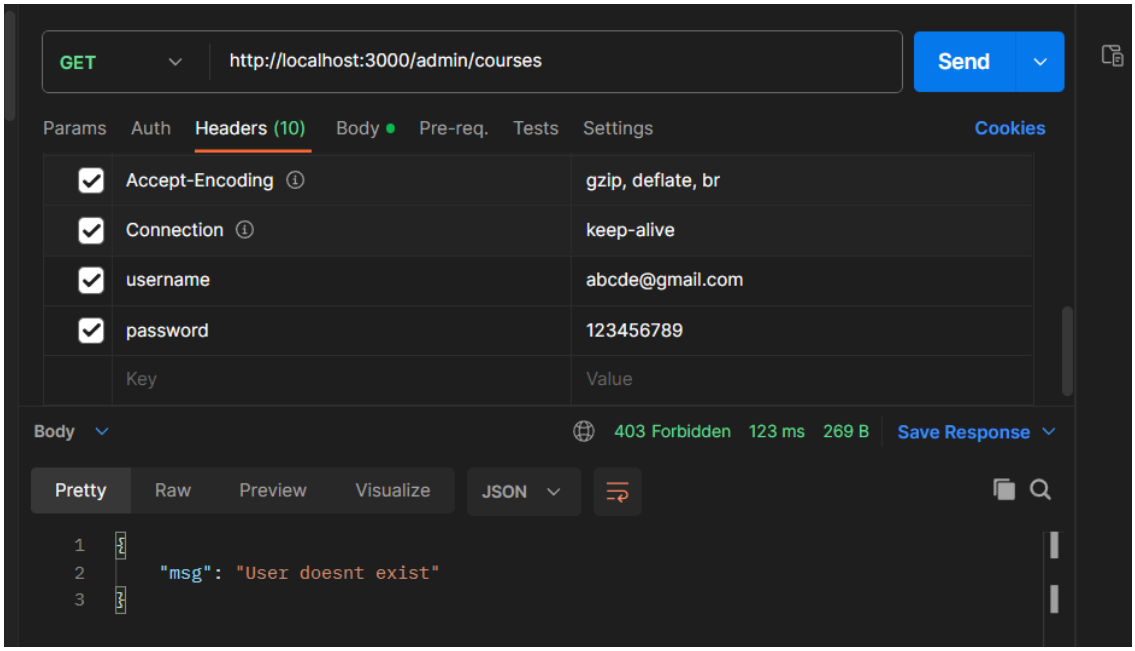
Lets work on last endpoint which return admin all its courses

```
router.get('/courses', adminMiddleware, (req, res) => {
    // Implement fetching all courses logic
```

```
    Course.find({})
    .then(function(response){
        res.json({
            courses: response
        })
    })
});

// router.get('/courses',adminMiddleware, async(req,res) => {
//     const response = await Course.find({})
//     res.json({
//         courses: response
//     })
// })
```



Let say we send the wrong user



Authentication in real world take the help of jwt.

route/admin.js

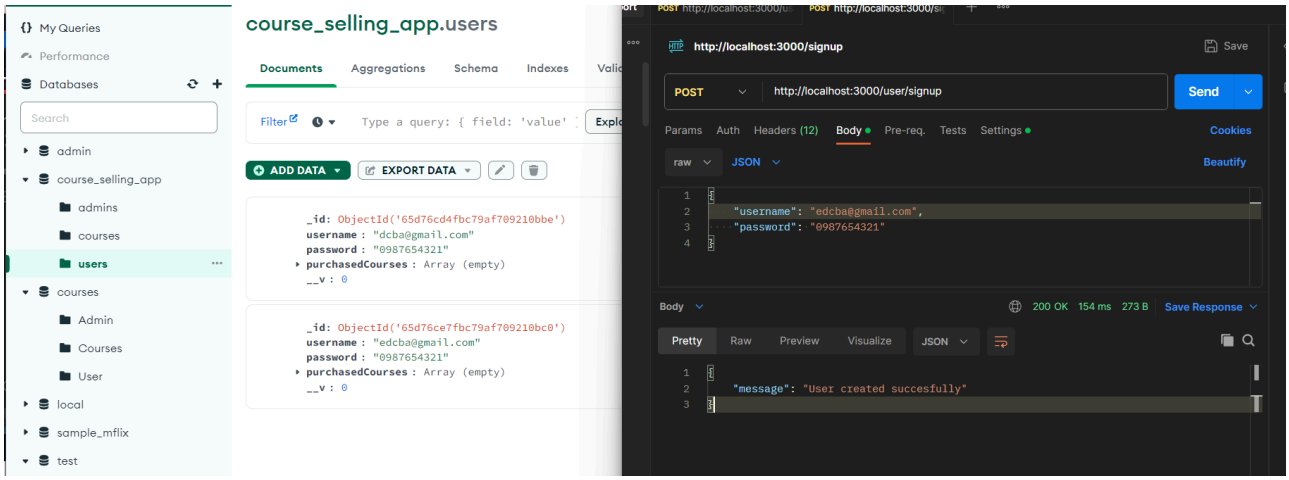
```
const { Router } = require("express");
const adminMiddleware = require("../middleware/admin");
const { Admin, Course } = require("../db");
const router = Router();

router.post('/signup',async (req, res) => {
    // Implement admin signup logic
    // Readme file must be read
    const username = req.body.username;
    const password = req.body.password;
    // check if a user with this username already exist , it it exist then
we must stop this request

    await Admin.create({
        username: username,
        password: password
    })
    res.json({
        message:"Admin created succesfully"
```

```
    })

});
router.post('/courses', adminMiddleware,async (req, res) => {
    // not accessible until the sign up is done
    // Implement course creation logic
    // need username and password
    const title = req.body.title;
    const description = req.body.description;
    const imageLink = req.body.imageLink;
    const price = req.body.price;
    // zod should be used here
    const newCourse = await Course.create({
        title,
        // title:title
        // both same if key and value are same
        description,
        imageLink,
        price
    })
    res.json({
        // in mongoDB any data we entered is given some id randomly
        // now to know the courseId of just enter data in database we need
to await
        message:"Course created succesfully",
        courseId: newCourse._id
    })
});

router.get('/courses', adminMiddleware, (req, res) => {
    // Implement fetching all courses logic
    Course.find({})
    .then(function(response){
        res.json({
            courses: response
        })
    })
});

module.exports = router;
```

Now working on the user endpoints

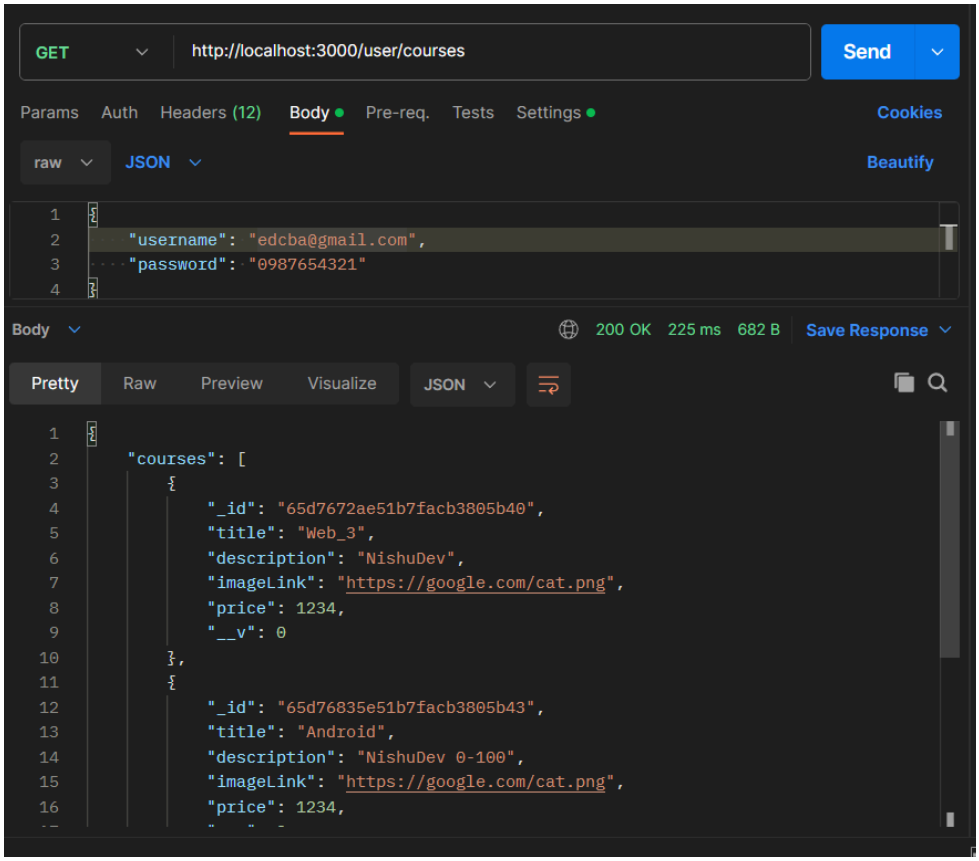
- **signup endpoint**

```
// User Routes
// one way to structure the express app

router.post('/signup', async(req, res) => {
    // Implement user signup logic
    // Implement admin signup logic
    // Readme file must be read
    const username = req.body.username;
    const password = req.body.password;
    // check if a user with this username already exist , it it exist then
we must stop this request

    await User.create({
        username: username,
        password: password
    })
    res.json({
        message:"User created succesfully"
    })
});
```



Endpoint displaying all the courses published to the user.

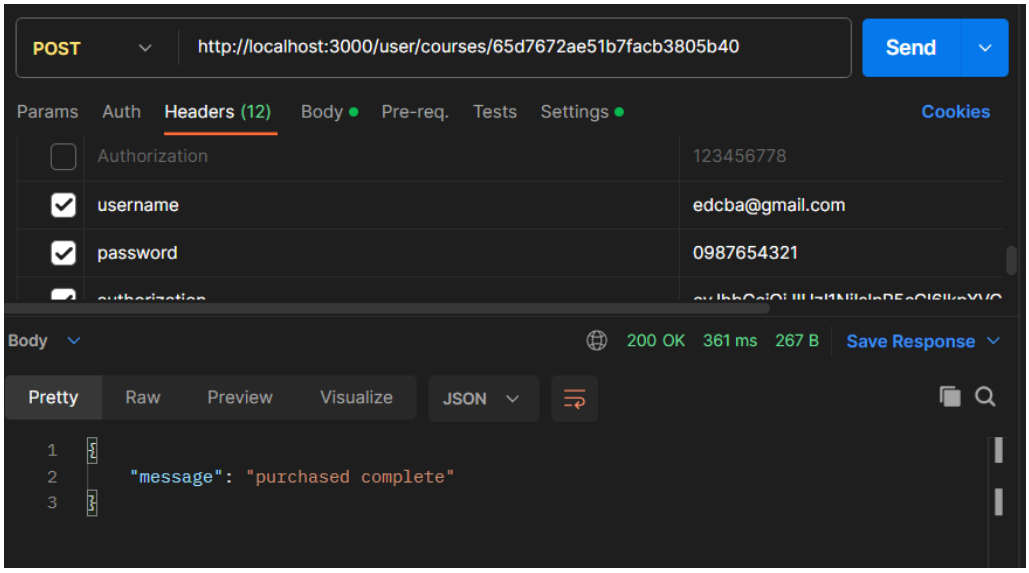
```
router.get('/courses', async(req, res) => {
    // Implement listing all courses logic
    // this can be open endpoint , listing all the courses present just like in udemy
    // so the question may arise why we created exactly same endpoint in the admin , it is because in the application like udemy there is a header called ispublished: yes/no which are only visible to the admins and list shown to the user is filtered.
    const response = await Course.find({

    });
    res.json({
        courses:response
    })
});
```

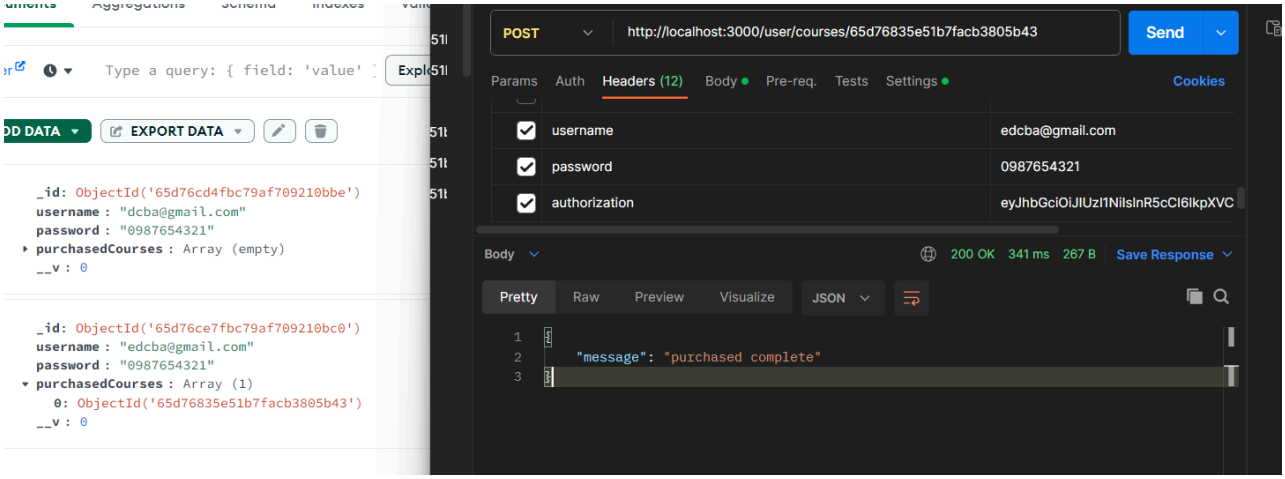


purchased course endpoint

In the real-world project we hit the razor pay API then the course is purchased else not.

```
router.post('/courses/:courseId', userMiddleware,async (req, res) => {
    // Implement course purchase logic
    // We don't have a purchase table
    // get back the course id which user want to buy
    // we need to extract courseId from the url
    const courseId = req.params.courseId;
    const username = req.headers.username;
    // input validatiion using zod
    // enough to update the user table in mongoDB
    await User.updateOne({
        username: username
    },{
        "$push":{
            purchasedCourses: courseId
        }
        // WRONG
        // purchasedCourses:{
        //     "$push":courseId
        // }
        // if we see error we can log it
    })
    res.json({
        message: "purchased complete"
    })
});
```



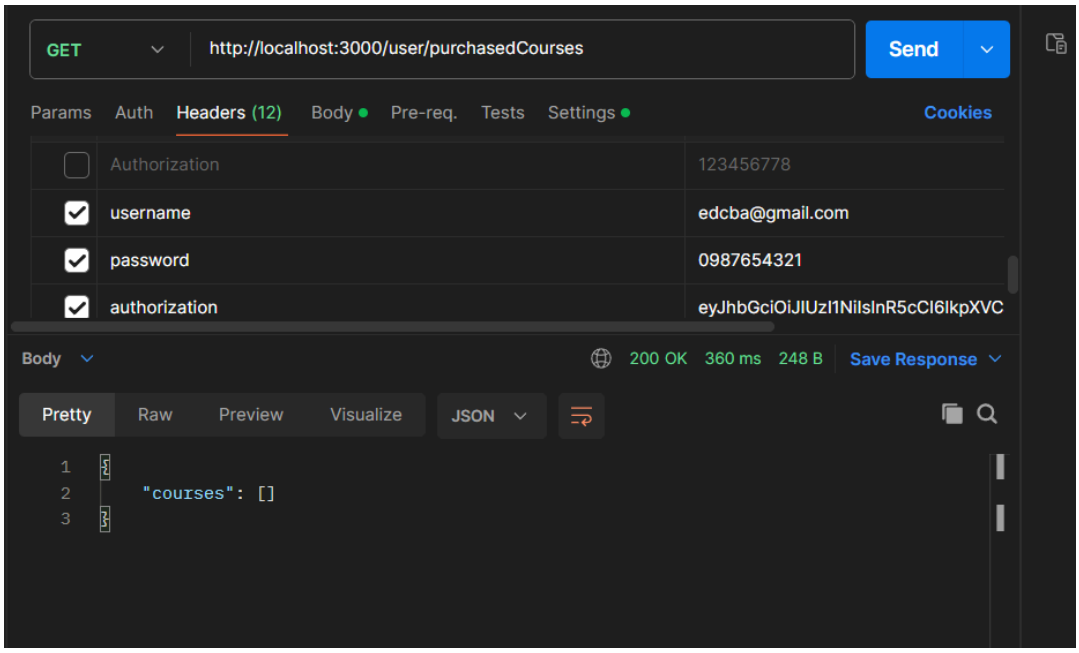


Now lets work on last endpoint of the users showing their purchased courses.

```
router.get('/purchasedCourses', userMiddleware,async (req, res) => {
    // Implement fetching purchased courses logic
    // This endpoint is used so that user can see their purchased courses in the udemy dashboard.
    // get purchased id from the user table
    const user = await User.findOne({
        username: req.headers.username
```

```
    })
    // console.log(user.purchasedCourses)
    // we get an array of purchasedCoursesId
    const courses = await Course.find({
        courseId: {
            "$in": user.purchasedCourses
            // how to do find on mongoose on array
        }
    })
    res.json({
        courses: courses
    })
});
```

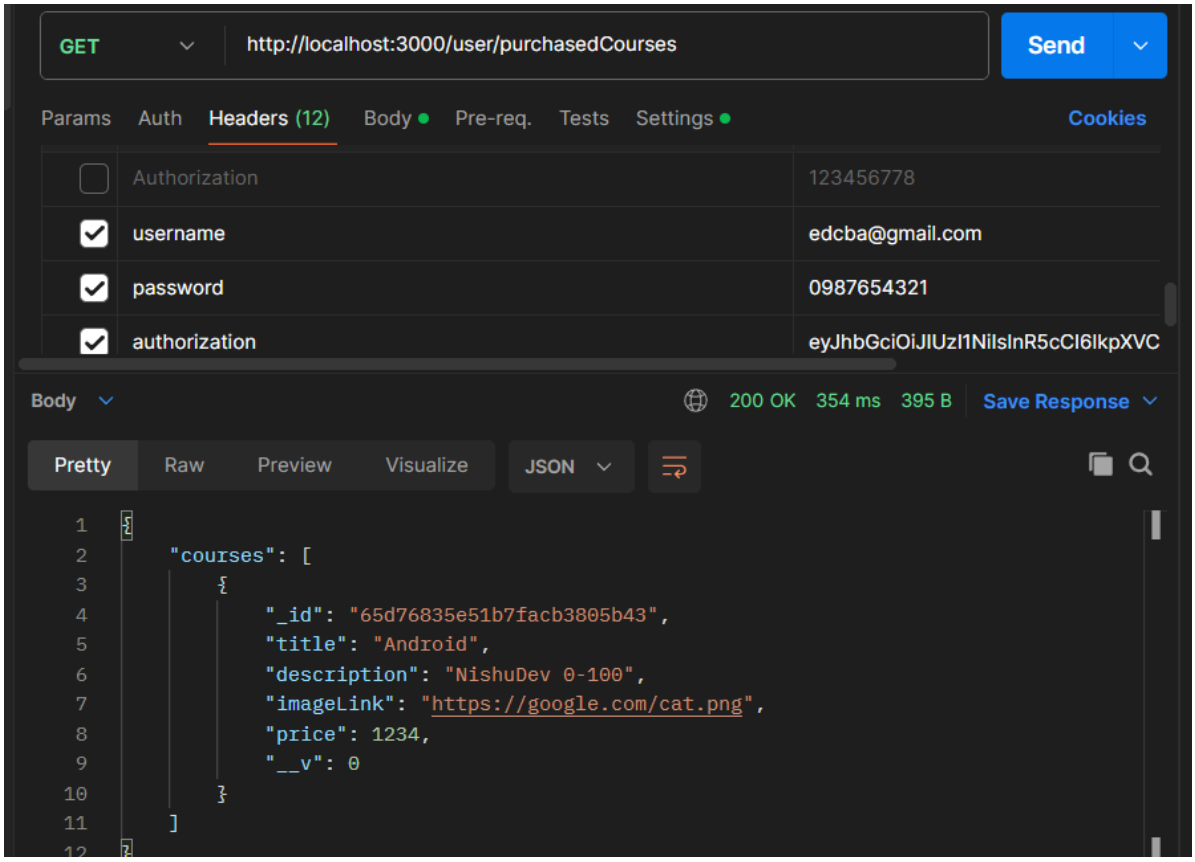
When we check on the POSTMAN there is some error lets see



We see an empty purchasedCourse array of the given user.

```
    const courses = await Course.find({
        _id: {
            "$in": user.purchasedCourses
            // how to do find on mongoose on array
        }
    })
```

Now lets again test it on POSTMAN:



routes/user.js

```
const { Router } = require("express");
const router = Router();
const userMiddleware = require("../middleware/user");
const { User,Course } = require("../db");

// User Routes
// one way to structure the express app

router.post('/signup', async(req, res) => {
    // Implement user signup logic
    // Implement admin signup logic
    // Readme file must be read
    const username = req.body.username;
    const password = req.body.password;
```

```
    // check if a user with this username already exist , it it exist then
we must stop this request

    await User.create({
        username: username,
        password: password
    })
    res.json({
        message:"User created succesfully"
    })
});

router.get('/courses', async(req, res) => {
    // Implement listing all courses logic
    // this can be open endpoint , listing all the courses present just
like in udemy
    // so the question may arise why we created exactly same endpoint in
the admin , it is because in the application like udemy there is a header
called ispublished: yes/no which are only visible to the admins and list
shown to the user is filtered.
    const response = await Course.find({});
    res.json({
        courses:response
    })
});

router.post('/courses/:courseId', userMiddleware,async (req, res) => {
    // Implement course purchase logic
    // We don't have a purchase table
    // get back the course id which user want to buy
    // we need to extract courseId from the url
    const courseId = req.params.courseId;
    const username = req.headers.username;
    // input validatiion using zod
    // enough to update the user table in mongoDB
    await User.updateOne({
        username: username
    },{
        "$push":{
            purchasedCourses: courseId
```

```
        }
        // WRONG
        // purchasedCourses:{
        //     "$push":courseId
        // }
        // if we see error we can log it
    })
    res.json({
        message: "purchased complete"
    })
});

router.get('/purchasedCourses', userMiddleware,async (req, res) => {
    // Implement fetching purchased courses logic
    // This endpoint is used so that user can see their purchased courses
in the udemy dashboard.
    // get purchased id from the user table
    const user = await User.findOne({
        username: req.headers.username
    })
    // console.log(user.purchasedCourses)
    // we get an array of purchasedCoursesId
    const courses = await Course.find({
        _id: {
            "$in": user.purchasedCourses
            // how to do find on mongoose on array
        }
    })
    res.json({
        courses: courses
    })
});

module.exports = router
```

Bearer tells the type of token

When sending the authorization header.

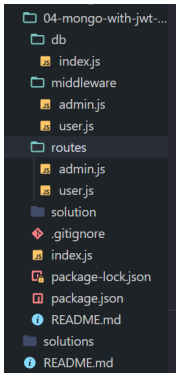
Assignment 2:

```
## Create a course-selling website

### Description

Same as the last assignment but you need to use jwts for authentication.
We have introduced the signgin endpoints for both users and admins.
For this one, in every authenticated requests, you need to send the jwt in
headers (Authorization : "Bearer <actual token>").
You need to use mongodb to store all the data persistently.

## Routes

### Admin Routes:

- POST /admin/signup
  Description: Creates a new admin account.
  Input Body: { username: 'admin', password: 'pass' }
  Output: { message: 'Admin created successfully' }
- POST /admin/signin
  Description: Logs in an admin account.
  Input Body: { username: 'admin', password: 'pass' }
  Output: { token: 'your-token' }
- POST /admin/courses
  Description: Creates a new course.
  Input: Headers: { 'Authorization': 'Bearer <your-token>' }, Body: {
title: 'course title', description: 'course description', price: 100,
imageLink: 'https://linktoimage.com' }
  Output: { message: 'Course created successfully', courseId: "new course
id" }
- GET /admin/courses
  Description: Returns all the courses.
  Input: Headers: { 'Authorization': 'Bearer <your-token>' }
  Output: { courses: [ { id: 1, title: 'course title', description:
'course description', price: 100, imageLink: 'https://linktoimage.com',
published: true }, ... ] }

### User routes
```

```
- POST /users/signup
  Description: Creates a new user account.
  Input: { username: 'user', password: 'pass' }
  Output: { message: 'User created successfully' }
- POST /users/signin
  Description: Logs in a user account.
  Input: { username: 'user', password: 'pass' }
  Output: { token: 'your-token' }
- GET /users/courses
  Description: Lists all the courses.
  Input: Headers: { 'Authorization': 'Bearer <your-token>' }
  Output: { courses: [ { id: 1, title: 'course title', description:
'course description', price: 100, imageLink: 'https://linktoimage.com',
published: true }, ... ] }
- POST /users/courses/:courseId
  Description: Purchases a course. courseId in the URL path should be
replaced with the ID of the course to be purchased.
  Input: Headers: { 'Authorization': 'Bearer <your-token>' }
  Output: { message: 'Course purchased successfully' }
- GET /users/purchasedCourses
  Description: Lists all the courses purchased by the user.
  Input: Headers: { 'Authorization': 'Bearer <your-token>' }
  Output: { purchasedCourses: [ { id: 1, title: 'course title',
description: 'course description', price: 100, imageLink:
'https://linktoimage.com', published: true }, ... ] }
```

File Structure

Download from the assignment Week-3 , assignment 4



db/index.js

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

// Connect to MongoDB
mongoose.connect('mongodb+srv://admin:KmCihXn0l1podXRj@cluster0.9gr3ic2.mo
ngodb.net/course_selling_app');

// Define schemas
const AdminSchema = new mongoose.Schema({
    // Schema definition here
    username: String,
    password: String
});

const UserSchema = new mongoose.Schema({
    // Schema definition here
    username: String,
    password: String,
    purchasedCourses: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Course'
        // it refer to course table
    }]
});

const CourseSchema = new mongoose.Schema({
    // course Schema definition here
    title: String,
    description: String,
    imageLink: String,
    price:Number
});

// tables required
const Admin = mongoose.model('Admin', AdminSchema);
const User = mongoose.model('User', UserSchema);
const Course = mongoose.model('Course', CourseSchema);

module.exports = {
    Admin,
```

```
    User,
    Course
}
```

Now we will work on middleware

Lets first set up the project

cd on the working directory

npm install

npm install jsonwebtoken

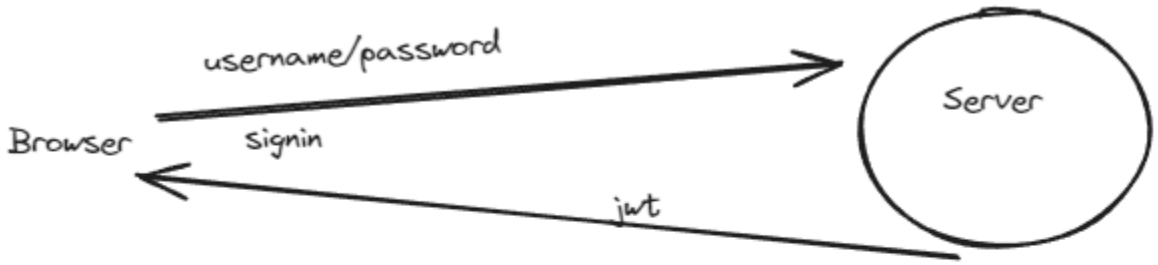
node index.js

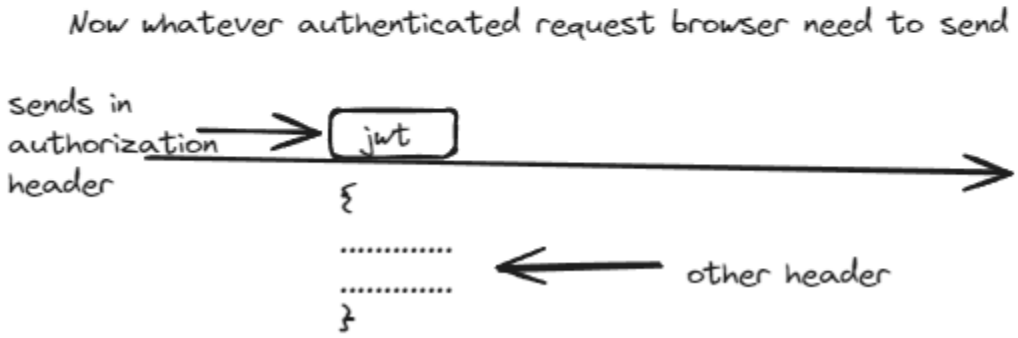
middleware/admin.js

```
const jwt = require("jsonwebtoken")
const secret = require("../index");

// Middleware for handling auth
function adminMiddleware(req, res, next) {
    // Implement admin auth logic
    // You need to check the headers and validate the admin from the admin
DB. Check readme for the exact headers to be expected
    // While accessing headers everything get converted to the lowercase
    const token = req.headers.authorization;
    // we could have seen this by console.log(req.autho..)
    // Bearer asfgsddhfdsafdgdsasdfsaDGDF get back token from string
    const words = token.split(" ");
    const jwtToken = words[1];
    // from here we do the jwt verification
    // there should be some global secret which used to generate the token
and verify the token
    const decodedValue = jwt.verify(jwtToken, secret);
    if( decodedValue.username){
    // where does this username come from??
        next();
    }else{
        res.status(403).json({
            msg:"You are not authmeticated"
        })
```

```
    }
}

module.exports = adminMiddleware;
```

What are we doing





Jwt has encoded version of the username .

Never send password in jwt.

```
if( decodedValue.username){
    // where does this username come from??
        next();
    }
```

Middleware uses this to extract the username from the jwt

middleware/admin.js

```
const jwt = require("jsonwebtoken")
const secret = require("../index");

// Middleware for handling auth
```

```
function adminMiddleware(req, res, next) {
    // Implement admin auth logic
    // You need to check the headers and validate the admin from the admin
DB. Check readme for the exact headers to be expected
    // While accessing headers everything get converted to the lowercase
    const token = req.headers.authorization;
    // we could have seen this by console.log(req.autho..)
    // Bearer asfgsddhfdsafdgdsasdfsaDGDF get back token from string
    const words = token.split(" ");
    const jwtToken = words[1];
    // from here we do the jwt verification
    // there should be some global secret which used to generate the token
and verify the token
    const decodedValue = jwt.verify(jwtToken, secret);
    // why aren't we checking in the database that whether the user exist
in our database or not?
    // This is reason why jwt is powerfull, it save us a database call.
    // It save us 1DB call
    // It checks in inmemory that whether the jwt was created by me or
not.
    // We just have to send jwt once the user sign only one time , this
jwt which was send is enough to verify that user exist or not.
    if( decodedValue.username){
    // where does this username come from??
    // if decoded.username exist then tell them that user exist
        next();
    }else{
        res.status(403).json({
            msg:"You are not authmeticated"
        })
    }
}

module.exports = adminMiddleware;
```

middleware/user.js

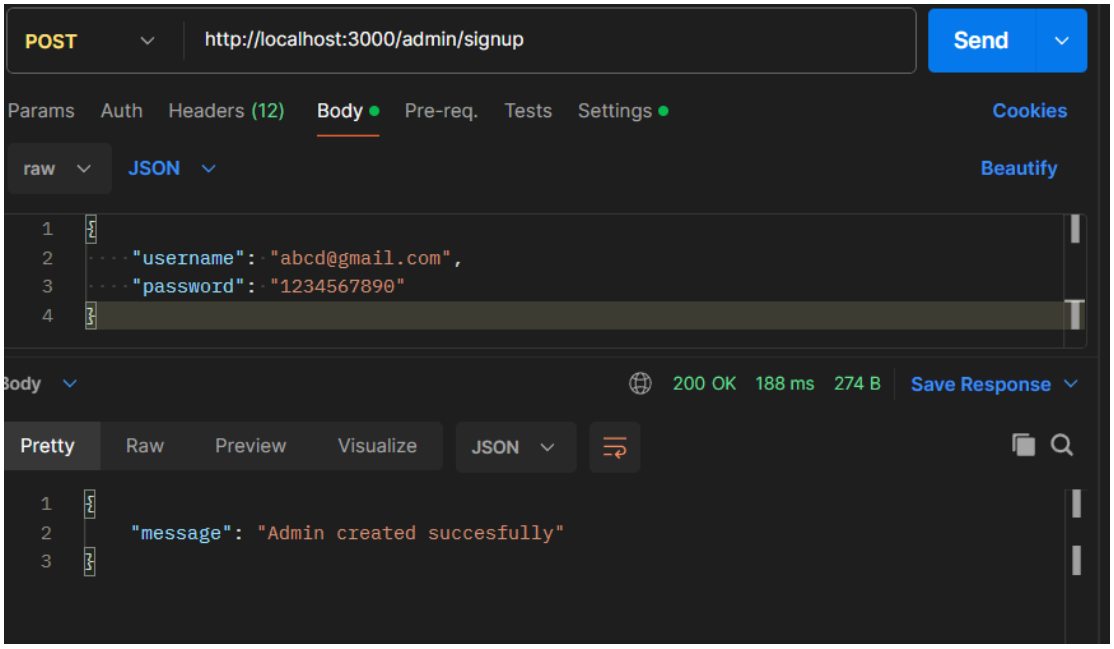
```
function userMiddleware(req, res, next) {
    const token = req.headers.authorization;
    const words = token.split(" ");
```

```
    const jwtToken = words[1];
    const decodedValue = jwt.verify(jwtToken, secret);
    // ideally we should send the type of user
    // username, type: "admin" | "user"
    // and check type in if block (This is called authorization)
    if( decodedValue.username){
        next();
    }else{
        res.status(403).json({
            msg:"You are not authmeticated"
        })
    }
}

module.exports = userMiddleware;
```

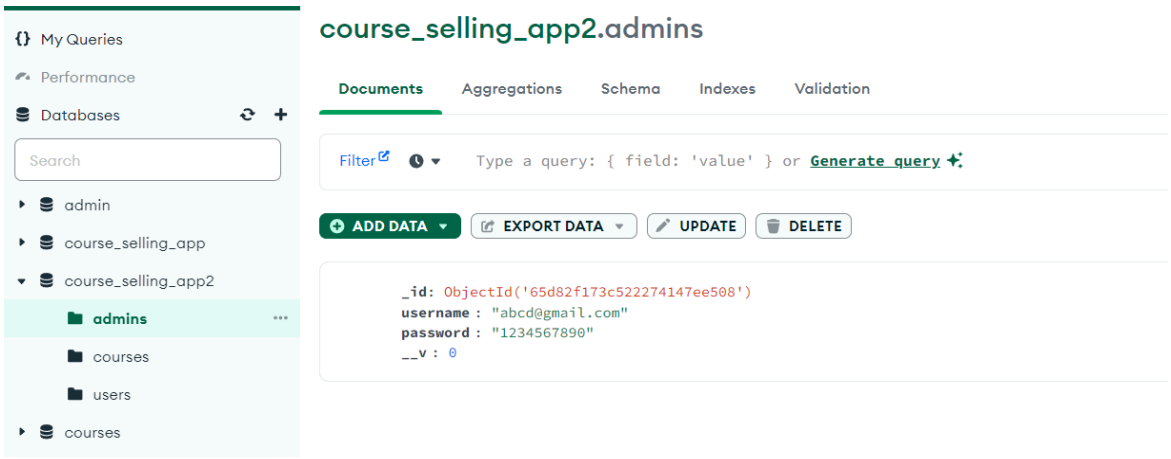
Let work on admin routes

- **First we work on the signup endpoint**

```
// admin routes
router.post('/signup',async (req, res) => {
    // Implement admin signup logic
    // Readme file must be read
    const username = req.body.username;
    const password = req.body.password;
    // check if a user with this username already exist , it it exist then
we must stop this request

    await Admin.create({
        username: username,
        password: password
    })
    res.json({
        message:"Admin created succesfully"
    })
});
```



Database:

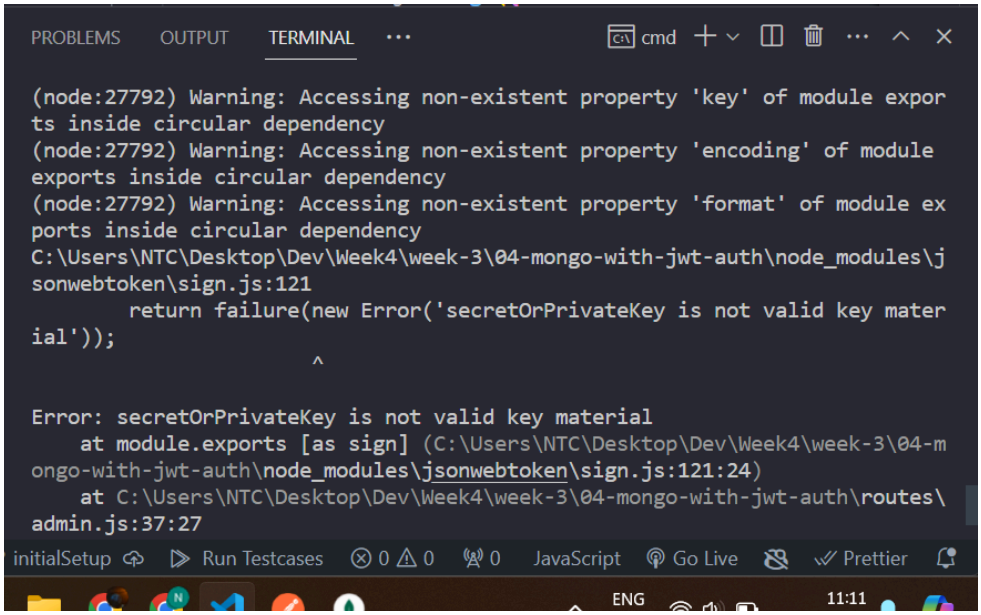


- **signin endpoint**

```
router.post('/signin',async (req, res) => {
    // Implement admin signup logic
    // Input Body: { username: 'admin', password: 'pass' }
    //   Output: { token: 'your-token' }
    const username = req.body.username;
    const password = req.body.password;

    const user = await User.find({
        username,
        password
```

```
    })
    if(user){
        const token = jwt.sign({
            username
        },JWT_SECRET);
        res.json({
            token
        })
    }else{
        res.status(411).json({
            message:"Incorrect email and password"
        })
    }
});
```

Lets see whether it works or not.



We get an error.

Basically, we are creating a circular dependencies

index,js import admin route and admin import index.js

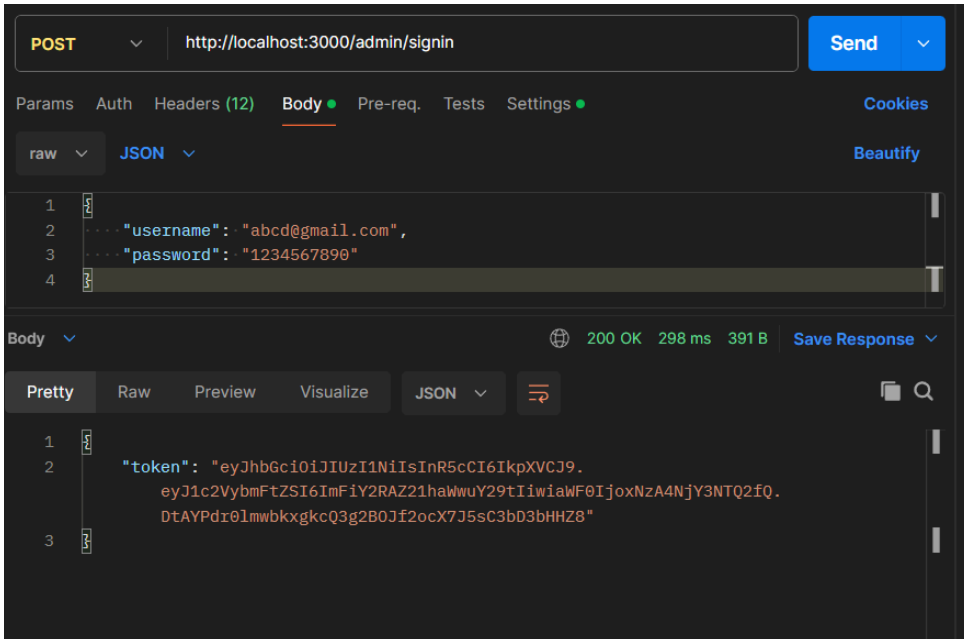
To solve this we will create a config.js file.

```
module.exports ={
    JWT_SECRET :"thapa_server"
```

```
}
// making index.js return nothing
```

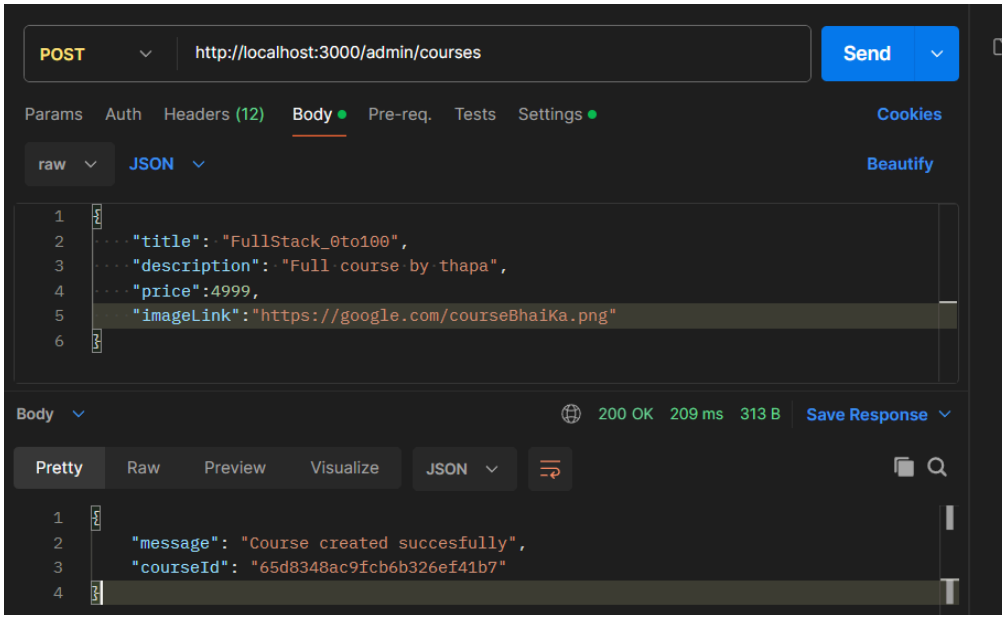
And make changes in rest of file where we import JWT_SECRET see the final solution

Now the signin endpoint works:

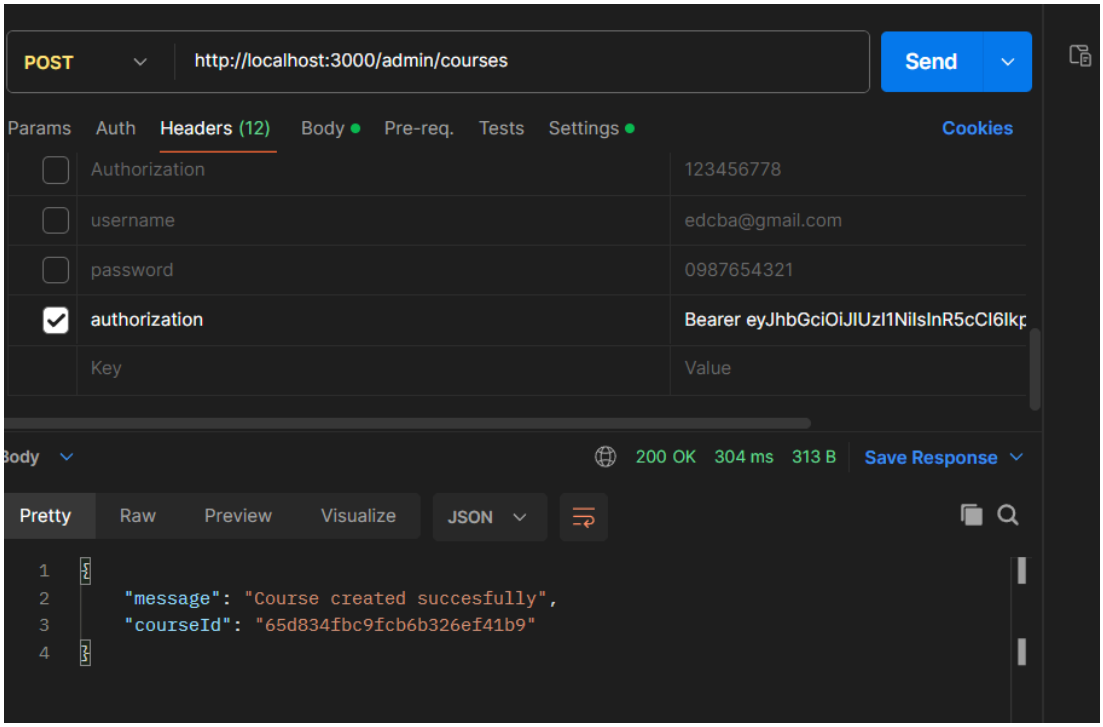


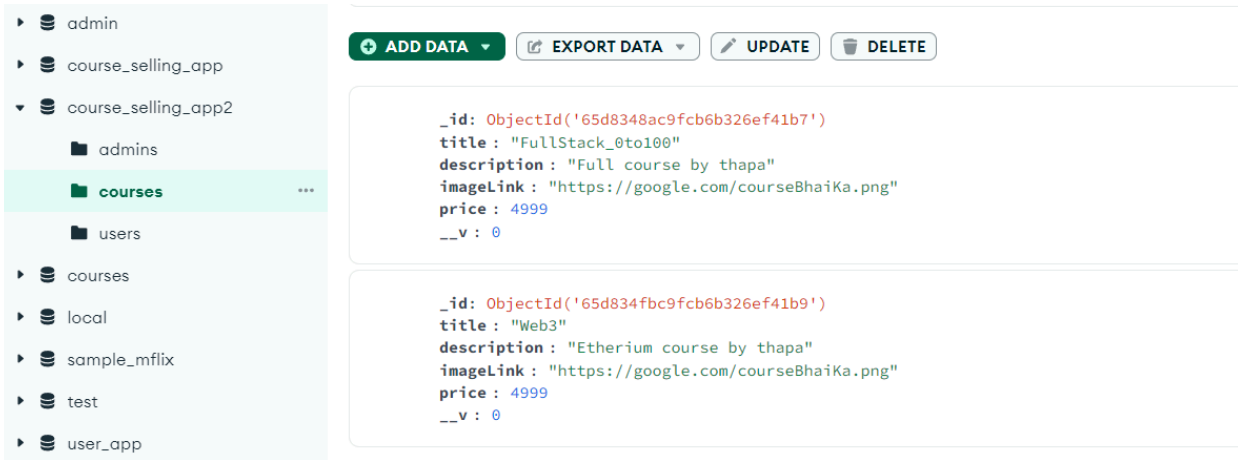
Let implement the course making endpoint

```
router.post('/courses', adminMiddleware,async (req, res) => {
    const title = req.body.title;
    const description = req.body.description;
    const imageLink = req.body.imageLink;
    const price = req.body.price;
    const newCourse = await Course.create({
        title,
        description,
        imageLink,
        price
    })
    res.json({
        message:"Course created succesfully",
        courseId: newCourse._id
    })
});
```



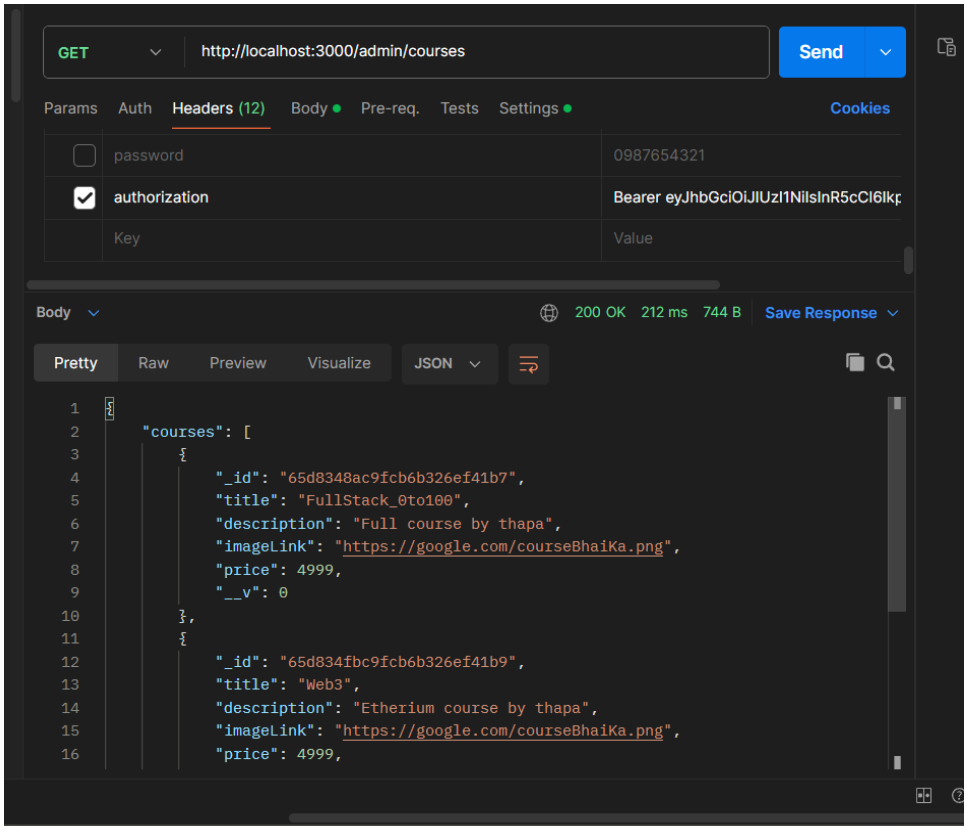
In Headers we pass the Authorization



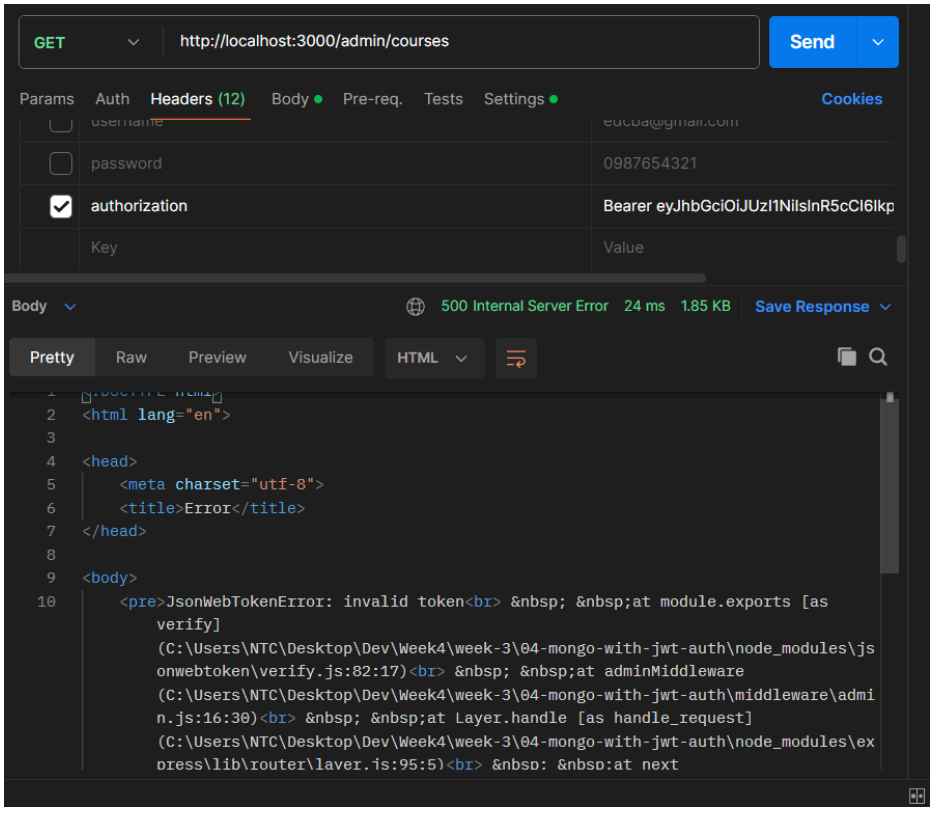


Lets implement the getting the purchased course list endpoint

```
router.get('/courses', adminMiddleware, (req, res) => {
    // Implement fetching all courses logic
    Course.find({})
    .then(function(response){
        res.json({
            courses: response
        })
    })
});
```



Suppose the token is wrong



We can handle it by doing try/ catch in admin middleware

routes/admin.js

```
const {Router} = require("express")
const adminMiddleware = require("../middleware/admin");
const {Admin,Course} = require("../db")
const {JWT_SECRET} = require("../config")
const router = Router();
const jwt = require("jsonwebtoken")

// admin routes
router.post('/signup',async (req, res) => {
    // Implement admin signup logic
    // Readme file must be read
    const username = req.body.username;
    const password = req.body.password;
    // check if a user with this username already exist , it it exist then
we must stop this request

    await Admin.create({
        username: username,
        password: password
    })
    res.json({
        message:"Admin created succesfully"
    })
});

router.post('/signin',async (req, res) => {
    // Implement admin signup logic
    // Input Body: { username: 'admin', password: 'pass' }
    //   Output: { token: 'your-token' }
    const username = req.body.username;
    const password = req.body.password;

    const user = await Admin.find({
        username,
        password
    })
```

```
    if(user){
        const token = jwt.sign({
            username
        },JWT_SECRET);
        res.json({
            token
        })
    }else{
        res.status(411).json({
            message:"Incorrect email and password"
        })
    }
});
router.post('/courses', adminMiddleware,async (req, res) => {
    const title = req.body.title;
    const description = req.body.description;
    const imageLink = req.body.imageLink;
    const price = req.body.price;
    // zod should be used here
    const newCourse = await Course.create({
        title,
        description,
        imageLink,
        price
    })
    res.json({
        message:"Course created succesfully",
        courseId: newCourse._id
    })
});


router.get('/courses', adminMiddleware, (req, res) => {
    // Implement fetching all courses logic
    Course.find({})
    .then(function(response){
        res.json({
            courses: response
        })
    })
```

```
});

module.exports = router;
```

Now implementing User endpoints

Mainly there is one endpoint with some change

Purchasing course endpoint

Here we are not passing username in the headers , because we are passing authorization in the headers . Because if we change the username then , different user will access courses with the same jwt token

Hence better approach is to keep the username in req object

```
if( decodedValue.username){
        req.username = decodedValue.username;
        next();
    }else{
        res.status(403).json({
            msg:"You are not authmeticated"
        })
    }
```

Middleware suppose to do

- End the request
- Forward the request
- Pass data along to the next middleware

```
router.post('/courses/:courseId', userMiddleware, (req, res) => {
    // Implement course purchase logic
    const username = req.username;
    // without user explicilitly sending their username we were able to
get the username
});
```