**Lab 5: Microservices**

# Objective:

To develop bookstore APIs based on Microservices based on the services planned during class. The services to be developed are:

1. Book services
2. Authentication Services
3. Order Services

# Scope of Lab Activity

Focus on building API using Microservices way

* Book services: Revision of how to build an API
* Authentication Services: Passport and JWT for authentication
* Order Services: Using SQL as a database

## 1. Create Book Service (Revision)

1) Create API for CRUD books

2) Define the Mongoose model for Book

3) Create the routes for CRUD

4) Call the Mongoose Operation for CRUD

**Book**

1. Create the **L5 microservice folder** and **book-service subfolder** in L5 microservice folder.
2. Open the cmd terminal from the book-service subfolder and run the following command:
   1. npm init
   2. npm install express body-parser mongoose
3. Create **index.js in the book-service folder**. Copy and paste the boiler code into index.js.

Note: port is set to 5000.

|  |
| --- |
| //import and setting up middleware  var express = require('express'); //call express  var app = express(); //define our app using express  var bodyParser = require('body-parser');  // Middleware  app.use(bodyParser.urlencoded({extended:true}));  app.use(bodyParser.json());  var port = process.env.PORT || 5000 //set our port  //Setting route and path  var router = express.Router()  router.get('/', (req,res)=>{  res.json({message: 'Book service works!!!'})  })  app.use('/api',router);  app.listen(port); // create a server that browsers can connect to  console.log("Magic happened at port "+port); |

1. At cmd run index.js: nodemon index.js
2. Create **book.js**. Copy and paste the book model into book.js.

|  |
| --- |
| const mongoose = require('mongoose');  const BookSchema = new mongoose.Schema({  name:String,  description:String,  price:Number,  isbn:String,  authors:[String],  coverUrl:String  })  module.exports = mongoose.model('Book',BookSchema); |

1. Add the following code in **index.js**.

Note: Replace your mongodb connection string at mongoose.connect()

|  |
| --- |
| const mongoose = require('mongoose');  const Book = require('./book');  mongoose.connect('mongodb+srv://lizawatisalahuddin:XgISAfVg1kGW7dqJ@cluster0.g3rij.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0')  //Create a book  router.post('/books', async (req, res) => {  const book = new Book(req.body);  try {  await book.save();  res.status(201).json(book);  } catch (error) {  res.status(500).json({ error: 'Failed to create book', details: error.message });  }  });  // Read all books  router.get('/books', (req,res)=>{  Book.find().then(function (books){  res.json({message:'OK', data:books})  })  .catch(function (err){  res.json({error:"message "+err})  })  });  //Read one book  router.get('/books/:id', async (req, res) => {  try {  const book = await Book.findById(req.params.id);  if (!book) {  return res.status(404).json({ error: 'Book not found' });  }  res.status(200).json(book);  } catch (error) {  res.status(500).json({ error: 'Failed to fetch book', details: error.message });  }  });  // Update  router.put('/books/:id', (req,res)=>{    });  // Delete  router.delete('/books/:id', (req,res)=>{    }); |

Arrangement some of the codes:

A screen shot of a computer code

Description automatically generated

1. Test at Postman and check at MongoDB. Use the following data to create a book.

|  |
| --- |
| {      "name":"REST API Development with Node.js",      "description":"This book is for beginner to build API using Node.js",      "price":59.90,      "isbn":"80123-123e",      "authors":["Lizawati Salahuddin", "Hafizi Husin"],      "coverUrl":"https://books.google.com.my/books/about/REST\_API\_Development\_with\_Node\_js.html?id=PsNlDwAAQBAJ&source=kp\_cover&redir\_esc=y"  } |

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## 2. Create Authentication Service

1) Create Authentication Service

2) Create a User model

3) Create a login and register function

4) Encrypt password using bcrypt

5) Check if a user is logged in API

**Authentication**

1. Create an **auth-service subfolder** in L5 microservice folder.
2. Open the cmd terminal from the book-service subfolder and run the following command:
   1. npm init
   2. npm install express mongoose bcrypt-node passport passport-jwt jsonwebtoken
3. Copy the following files from **Lab 4** into the auth-service subfolder:
   1. auth.js
   2. config.js
   3. index.js
   4. user.js
4. Modify **index.js** by:
   1. deleting:
      1. const Restaurant = require('./restaurant')”
      2. post endpoint to create a restaurant
   2. add:
      1. app.use(auth.initialize());

A close-up of text

Description automatically generated

* + 1. create a new endpoint that accepts the token and verifies it. If the token is valid, return the associated user data.

|  |
| --- |
| // \*\*Verify Token Route\*\*  app.post('/api/verify-token', async (req, res) => {  const token = req.body.token;  if (!token) {  return res.status(400).json({ error: 'No token provided' });  }  // Verify the token using the secret key  jwt.verify(token, config.secret, async (err, decoded) => {  if (err) {  return res.status(401).json({ error: 'Invalid token' });  }  try {  // Find the user by ID stored in the token payload  const user = await User.findById(decoded.\_id);  if (!user) {  return res.status(404).json({ error: 'User not found' });  }  // Token is valid and user is found  return res.status(200).json({ user: user });  } catch (err) {  return res.status(500).json({ error: 'An error occurred', details: err.message });  }  });  }); |

A screenshot of a computer code

Description automatically generated

1. Run at cmd: nodemon index.js
2. Test at Postman to register a user and login.
3. Check at MongoDB.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## 3. Create Order Service

1. Create Order Service
2. Create Order Model (use Sequelize)

<https://sequelize.org/master/>

1. Define routes
2. Create Create, Read, Update and Delete Function

**Order**

1. Create **order-service subfolder** in L5 microservice folder.
2. Open the cmd terminal from the order-service subfolder and run the following command:
   1. npm init
   2. npm install express sequelize mysql2 axios dotenv
   3. npm install -g sequelize-cli
   * sequelize: an Object-Relational Mapping (ORM) library for working with relational databases (e.g., MySQL, PostgreSQL) in JavaScript. Sequelize is a popular Object-Relational Mapping (ORM) library for Node.js. It helps developers interact with relational databases using JavaScript objects instead of raw SQL queries.
   * mysql2: driver for connecting Node.js applications to MySQL databases.
   * axios: used for calling external APIs or services in your application.
   * dotenv: A module to load environment variables from a .env file into process.env
   * sequelize-cli: a command-line tool provided by Sequelize for managing database-related tasks such as generating models, migrations, and seeders; running migrations to update the database schema; seeding data into the database; and managing database configurations.
3. Create the Sequelize model for Order. Start by initializing Sequelize at the cmd terminal (This will create the necessary directories for models, migrations, and configurations):

npx sequelize init

A screen shot of a computer

Description automatically generated

1. Edit the "password": "", "database": "orderdb", in **config.json** (inside folder config).

A screenshot of a computer screen

Description automatically generated

1. Create orderdb database at **phpMyAdmin**.

At XAMPP control panel 🡪 Apache 🡪 Start

MySQL 🡪 Start 🡪 Admin

A screenshot of a computer

Description automatically generated

phpMyAdmin page will open in your internet browser.

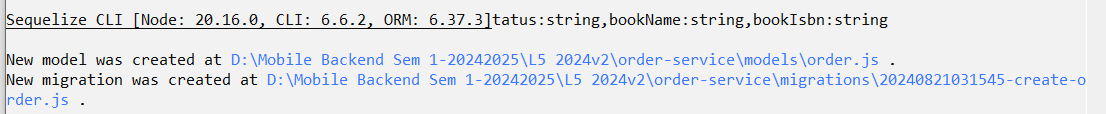
Click on SQL 🡪 copy paste: CREATE DATABASE orderdb 🡪 click Go to create orderdb database:

A screenshot of a computer

Description automatically generated

1. Run the command at the cmd terminal:

npx sequelize model:generate --name Order --attributes orderId:integer,userId:string,bookId:string,quantity:integer,status:string,bookName:string,bookIsbn:string



This will generate a migration and a model file. You can modify the model further in **models/order.js**.

1. Modify **order.js** in the **models folder** by deleting all the codes, then copy and paste the following codes:

|  |
| --- |
| // models/order.js  module.exports = (sequelize, DataTypes) => {  const Order = sequelize.define('Order', {  orderId: {  type: DataTypes.INTEGER, // Set the data type to INTEGER  autoIncrement: true, // Enable auto-increment  primaryKey: true, // Set it as the primary key  allowNull: false, // Disallow null values  },  userId: {  type: DataTypes.STRING,  allowNull: false,  },  bookId: {  type: DataTypes.STRING,  allowNull: false,  },  quantity: {  type: DataTypes.INTEGER,  allowNull: false,  },  status: {  type: DataTypes.STRING,  allowNull: false,  },  bookName: {  type: DataTypes.STRING,  allowNull: false,  },  bookIsbn: {  type: DataTypes.STRING,  allowNull: false,  },  }, {  tableName: 'orders',  timestamps: true, // Keep timestamps if needed  createdAt: 'createdAt', // Ensure Sequelize handles these fields properly  updatedAt: 'updatedAt',  });  // Synchronize the models with the database  sequelize.sync({ force: false }).then(() => {  console.log('Orders table created/updated successfully without an id column');  }).catch(error => {  console.error('Error creating/updating the orders table:', error);  });  return Order;  }; |

1. Replace all the codes in the generated migration file in the **migrations folder** with the following codes to define the schema:

|  |
| --- |
| 'use strict';  module.exports = {  up: async (queryInterface, Sequelize) => {  await queryInterface.createTable('orders', {  orderId: {  type: Sequelize.INTEGER,  autoIncrement: true,  primaryKey: true,  allowNull: false,  },  userId: {  type: Sequelize.STRING,  allowNull: false,  },  bookId: {  type: Sequelize.STRING,  allowNull: false,  },  quantity: {  type: Sequelize.INTEGER,  allowNull: false,  },  status: {  type: Sequelize.STRING,  allowNull: false,  },  bookName: {  type: Sequelize.STRING,  allowNull: false,  },  bookIsbn: {  type: Sequelize.STRING,  allowNull: false,  },  createdAt: {  type: Sequelize.DATE,  allowNull: false,  },  updatedAt: {  type: Sequelize.DATE,  allowNull: false,  }  });  },  down: async (queryInterface, Sequelize) => {  await queryInterface.dropTable('orders');  }  }; |

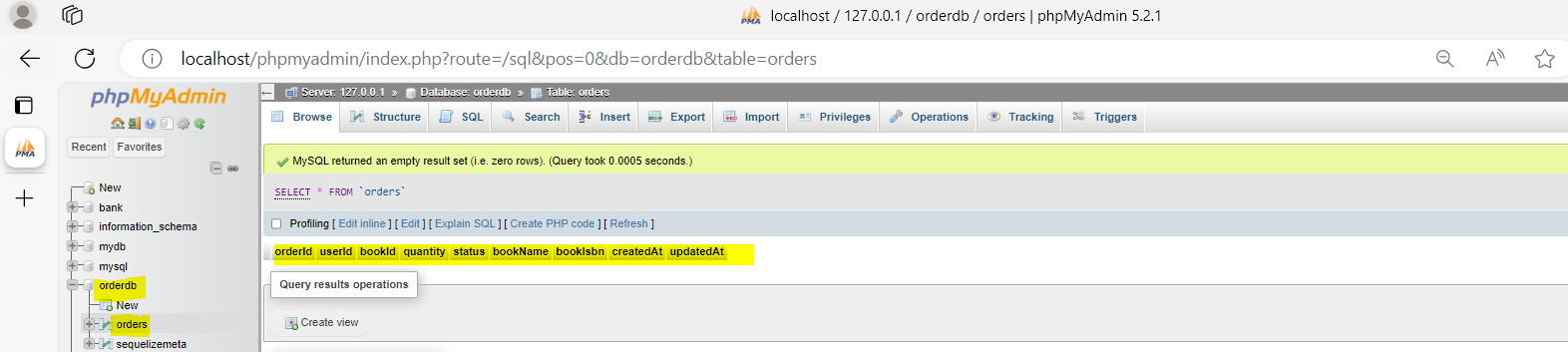
1. Migrations and Models with MySQL at cmd terminal:

npx sequelize-cli db:migrate

A screenshot of a computer code

Description automatically generated

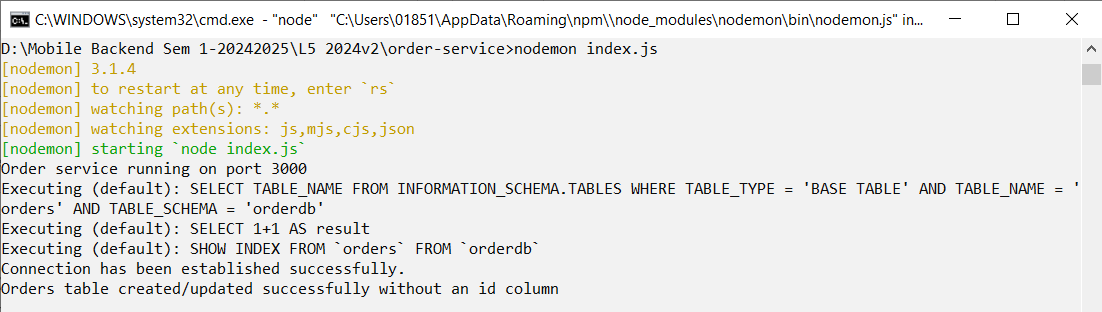
1. Check at phpMyAdmin. Table orders is created.



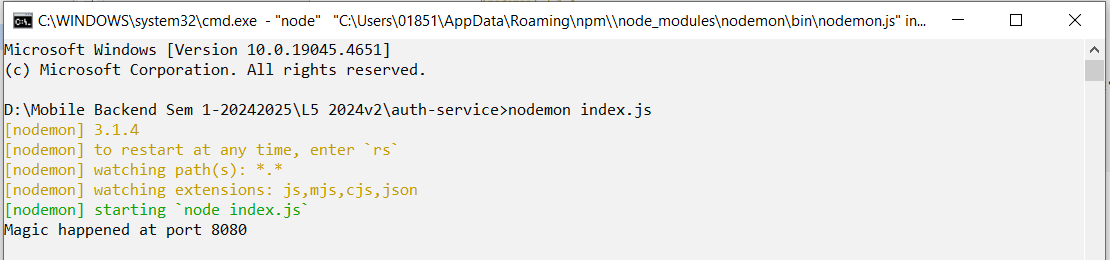
1. Create **index.js in order-service folder** to define the routes and integrate the authentication and book services. Copy and paste the following codes into index.js:

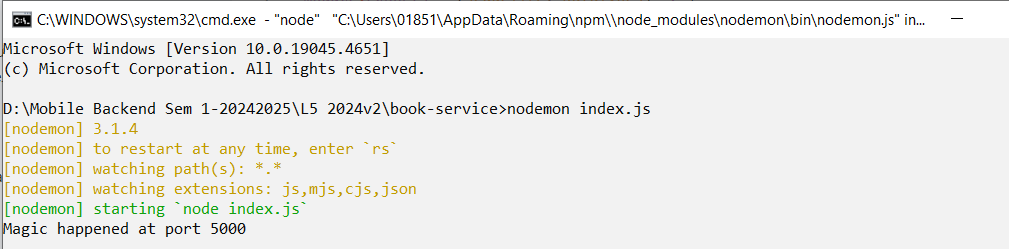
|  |
| --- |
| const express = require('express');  const axios = require('axios');  const { Order } = require('./models'); // Sequelize Order model  const auth = require('../auth-service/auth')(); /\* JWT-based authentication middleware \*/  const dotenv = require('dotenv');  dotenv.config(); // Load environment variables  const app = express();  app.use(express.json());  const { Sequelize } = require('sequelize');  // Create a Sequelize instance with your MySQL database details  const sequelize = new Sequelize('orderdb', 'root', null, {  host: 'localhost', // or your DB host  dialect: 'mysql', // specify MySQL as the dialect  pool: {  max: 5,  min: 0,  acquire: 30000, // Connection acquisition timeout  idle: 10000 // Idle connection timeout  }  });  // Function to check if MySQL connection is successful  const checkConnection = async () => {  try {  // Attempt to connect to the database  await sequelize.authenticate();  console.log('Connection has been established successfully.');  } catch (error) {  // If there is an error, log it  console.error('Unable to connect to the database:', error);  }  };  // Call the function to check the connection  checkConnection();  // Middleware to verify JWT by sending a request to auth-service  const verifyToken = async (req, res, next) => {  const token = req.headers.authorization?.split(' ')[1]; // Extract token  if (!token) {  return res.status(401).json({ error: 'Unauthorized: No token provided' });  }  try {  const response = await axios.post('http://localhost:8080/api/verify-token', {  token: token  });  req.user = response.data.user; // Assuming auth-service returns the user  next();  } catch (error) {  console.error('Error verifying token:', error.response?.data || error.message);  return res.status(401).json({ error: 'Unauthorized: Token verification failed' });  }  };  // Create Order  app.post('/orders', verifyToken, async (req, res) => {  console.log("Order creation request received:", req.body);  // Check if bookId is provided  if (!req.body.bookId) {  return res.status(400).json({ error: 'bookId is required' });  }  const bookId = req.body.bookId;  try {  const bookServiceUrl = 'http://localhost:5000/api'; /\* Replace with your book service URL \*/  const response = await axios.get(`${bookServiceUrl}/books/${bookId}`);  console.log("Book Service Response:", response.data);  if (response.status !== 200) {  return res.status(404).json({ error: 'Book not found' });  }  const book = response.data;  const order = await Order.create({  userId: req.user.\_id,  bookId: bookId,  quantity: req.body.quantity,  status: 'pending',  bookName: book.name,  bookIsbn: book.isbn,  createdAt: new Date(),  });  console.log("Order Created Successfully:", order);  res.status(201).json(order);  } catch (error) {  console.error("Order Creation Error:", error);  res.status(500).json({ error: 'Failed to create order', details: error.message });  }  });  // Get All Orders for the Authenticated User  app.get('/orders', verifyToken, async (req, res) => {  try {  console.log('Authenticated user:', req.user); // Debugging  const orders = await Order.findAll({ where: { userId: req.user.\_id } });  console.log('Fetched orders:', orders); // Debugging  res.status(200).json(orders);  } catch (error) {  console.error('Error fetching orders:', error); // Debugging  res.status(500).json({ error: 'Failed to fetch orders', details: error.message });  }  });  const PORT = process.env.PORT || 3000;  app.listen(PORT, () => {  console.log(`Order service running on port ${PORT}`);  }); |

1. Run order service at cmd: nodemon index.js

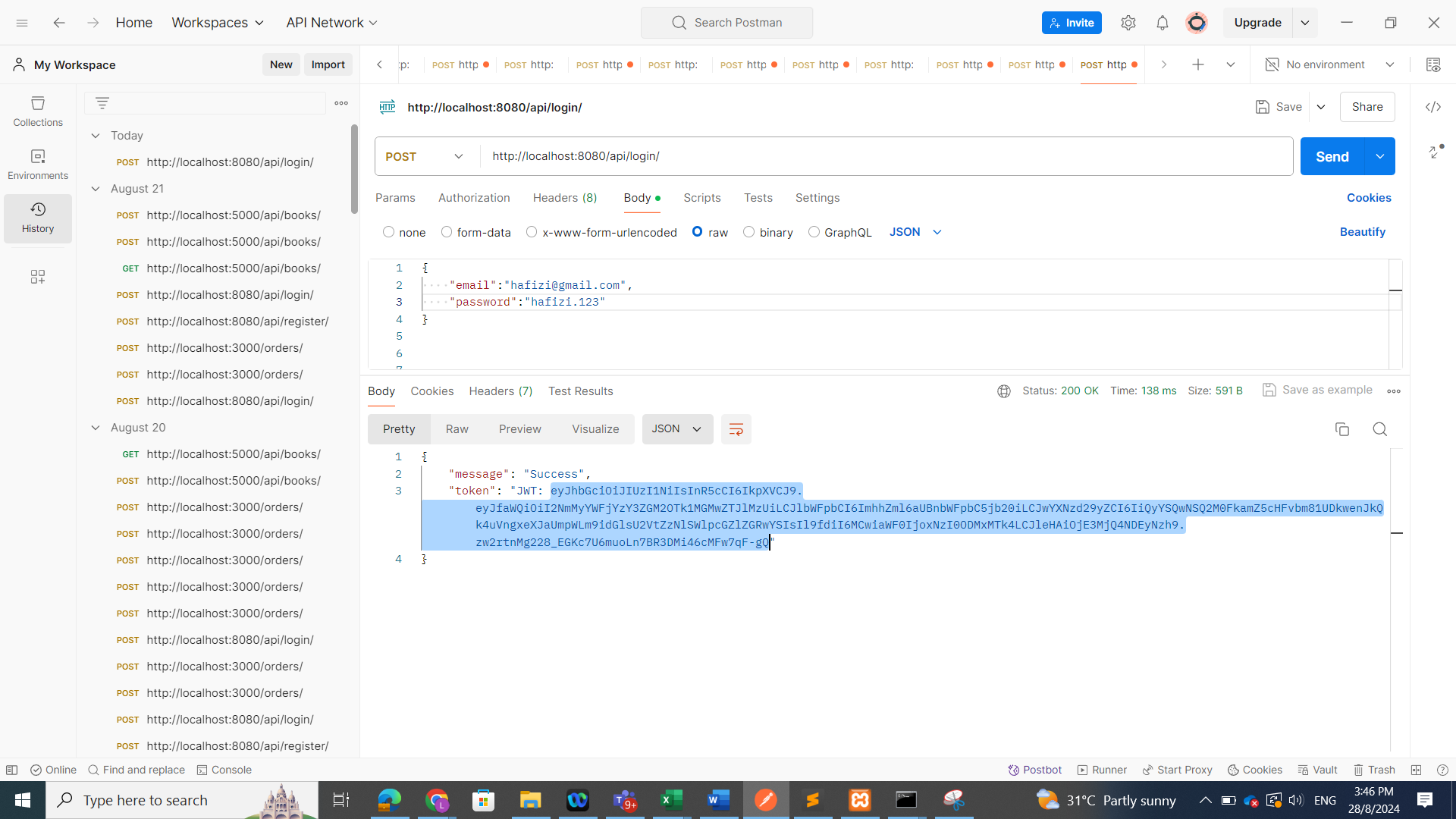


Please also make sure the authentication service and book service are also running at cmd.

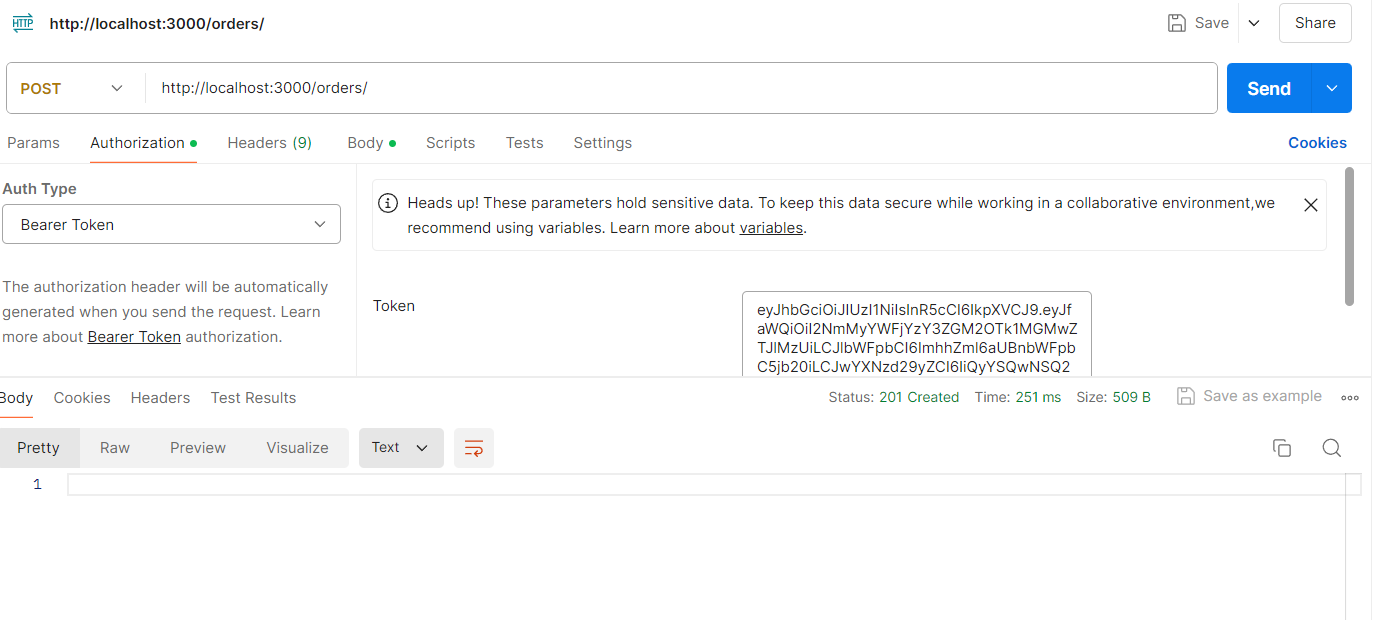




1. Test at Postman to login to get the JWT token. Copy the JWT token.



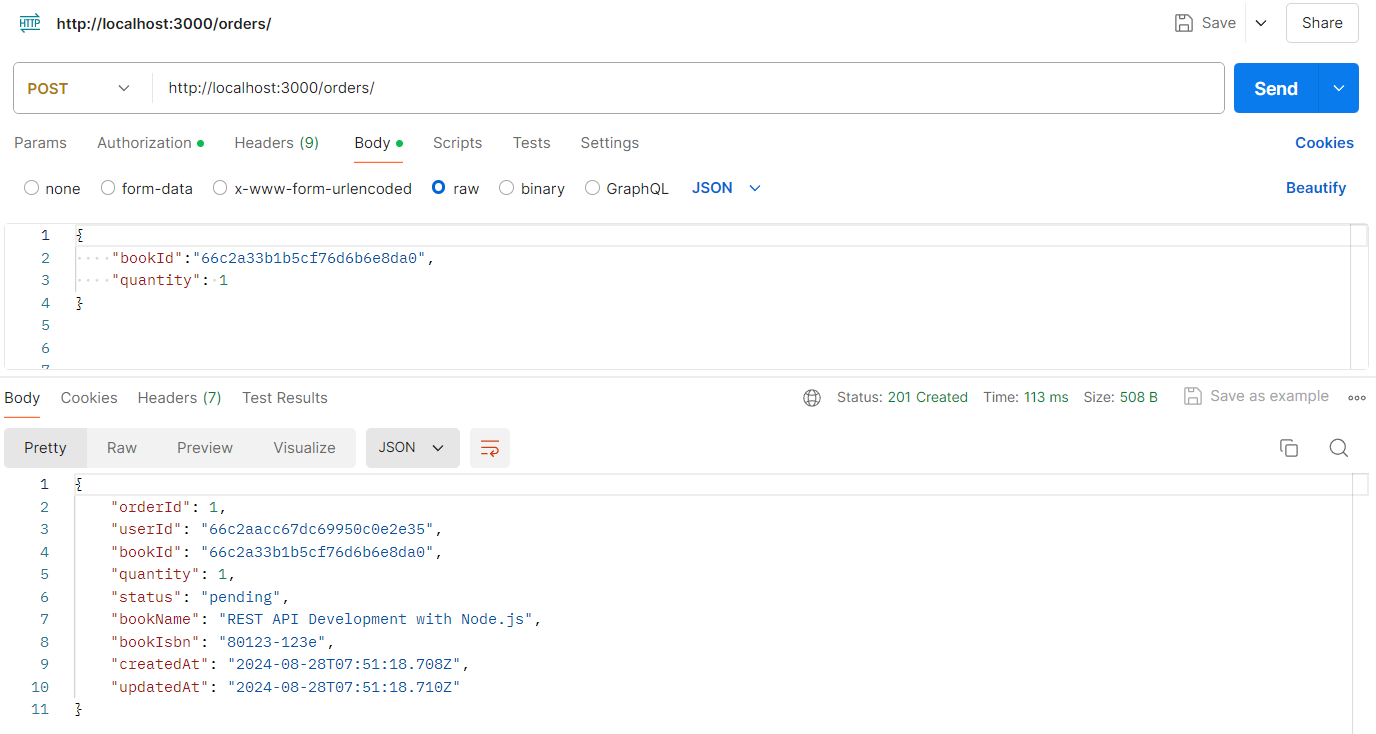
1. Test at Postman to post order.
   1. Click on the Authorization menu, select Auth Type: Brearer Token, and paste the copied JWT token (step 13) at Token



* 1. Click on the Body 🡪 raw 🡪 JSON 🡪 enter the bookID (copy paste \_id from books records in MongoDB) and quantity. Example: A screenshot of a computer

     Description automatically generated

copy \_id from books in MongoDB and paste to Postman



1. Check the order detail is inserted in MySQL.

A screenshot of a computer

Description automatically generated

1. Please upload all the services in .zip file to ULearn.