Sri Lanka Institute of Information Technology

SE3020 – Distributed Systems

Assignment

S2 - SE - WE - 03

IT21377358	Hanshani S. G. H. S.	it21377358@my.sliit.lk
IT21377280	Rajapaksha C. S.	it21377280@my.sliit.lk
IT21378270	Wimaladharma T. H. Y. B.	it21378270@my.sliit.lk
IT21355196	Kalpajith K. L. S.	It21355196@my.sliit.lk

Table of Contents

1. Introduction

"EduWave", an educational learning platform designed for motivated learners to empower their education. A wide range of courses are designed to fulfill learners' needs and goals. Using the web interfaces, students can browse the available courses, signup for the courses and monitor their progress. "EduWave" consisting of four main services:

- **User Service**
- **Course Service**
- **Learner Service**
- **Enroll Service**

These services collectively form the core of our platform, facilitating the creation, management, and delivery of courses, as well as empowering learners to enroll in courses and track their progress effortlessly.

In addition, a Gateway Service is integrated for the payment gateways, enabling secure and convenient transactions for course enrollments.

The architecture of "EduWave" is based on the concepts of Microservices, Docker, and Kubernetes. The NodeJS-built backend is further developed on this architecture by utilizing its event-driven, asynchronous design to provide scalable, high-performance solutions. Microservices form the foundation of the backend, allowing to decompose the system into loosely linked services, each in charge of handling a particular set of functions.

The user-friendly interactivity of the user interface (Web client interface) is built using ReactJS. The System thoroughly emphasis on security and authentication, to uniquely identify and authenticate users, generating OTP verification, ensuring the integrity. The system maintains the permissions and privileges by adhering to Role-based access (as Admin, Student, Instructor). The services are built with interoperability and RESTful principles, allowing for smooth communication and platform-wide integration.

2. High-level Architecture

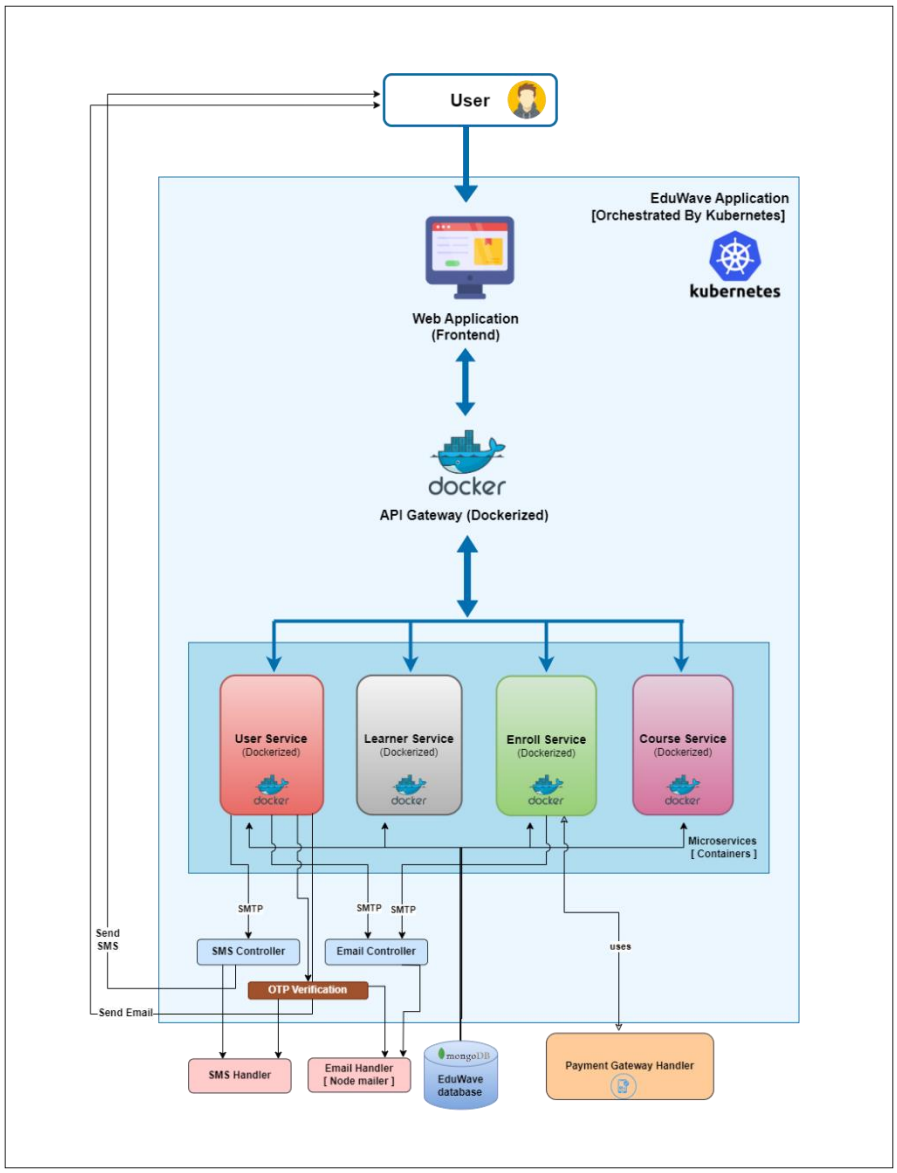


Figure 1. High-level Architecture

In the system's architecture ensures a responsive design that is readable on Web devices which is built using React. The user service, course service, enrollment service, and learner service are the four separate services which make up the backend of a Node.js microservices architecture. These services are all dockerized independently, and their corresponding Docker files are visible. These services enable communication between the frontend and backend by being connected to an API gateway. Furthermore, the API gateway service is exposed and dockerized. To ensure data

scalability, the primary four services store data in distinct tables housed in a MongoDB database called "EduWave." To perform payment-related functions safely, the enrollment service includes a payment gateway, namely Payhere sandbox. To send SMS notifications, the user service makes use of an SMS controller that communicates with an SMS handler. Moreover, email functionality is employed in both the user service and enrollment service, leveraging the Node Mailer dependency during the registration process of the user service, email verification is carried out by sending an OTP (One-Time Password) to verify the user's email address. In the payment process of the Enrollment service an email is generated as a receipt of the completion of the payment.

3. Service Interface

1. User Service

The user service plays a pivotal service in the educational platform where it handles the user registration, login and role-based access control. The most important steps in the User service registration process is creating and sending a one-time password (OTP) for verification. After successfully login into the system the users get directed to the respective dashboards based on their roles Admin, Student, or Instructor. Admin has the authority to manage the users, approving or denying the access. The user service is encapsulated within a docker container, and the service is operated on a unique port number(4001). It follows a structured folder organization, with the distinct components controllers, models, routes, and the index file for streamlined development and maintenance.

2. Enroll Service

The Enroll service is the process of students enrolling for the courses. The user-friendly interface, students can enroll themselves in multiple courses after successfully completing the payment process. The PayHere payment gateway is integrated, "**Payhere”** has turned out to be the perfect payment solution, for students’ demand, convenience, and security. The platform stores and manages the course enrollment data in the EduWave MongoDB. Furthermore, by encapsulating the service within docker containers, it enhances the agility, scalability, and reliability of the enroll service for the users facilitating a streamlined experience within the system. With a meticulously organized folder structure, the Enroll service ensures clarity and efficiency in development, with

distinct directories for controllers, models, routes, and an index file where the operation is conducted in a unique port number 4003.

3. Course Service

The creation, administration, and delivery of courses to students are greatly aided by the Course Management Service in the educational platform assignment. Administrators and course instructors are able to enhance and improve the learning process with the tools provided in this section. For educators, the platform offers an extensive range of features, such as the ability to add, update, and delete course content such as lecture notes, videos, and quizzes. Administrators are given the ability to approve course content using this service. By following these guidelines, the Course Management Service creates an atmosphere that is favorable to efficient instruction and learning, enabling teachers to provide top-notch course materials while administrators keep command and supervision over the platform's functions. Considering the course service, the folder structures are appropriately managed, and the action is carried out independently on port 4004.

4. Learner Service

The Learner Service is an essential component dedicated to enhancing the learning experience for users. Its primary function is to facilitate to access the course and track the progression of the completion of the course for learners. In collaboration with its backend, the frontend components of the service are separately dockerized. This service is operated using the Port number 4002. This service allows users to browse through the catalog of courses available on the platform, providing detailed information about each course, including descriptions, instructors, and prerequisites. During the process, the Learner Service enables learners to track their progress within each course, monitoring completed modules and overall course completion status. This feature empowers learners to manage their learning journey effectively and stay motivated towards achieving their educational goals.

4. Authentication / Security Mechanism

1. JWT Authentication

JSON Web Token is an open standard used to share information between two parties securely, a client and a server. Tokens originate using secret information related to the user's identity and

authorization to access services. By authenticating incoming requests, this helps prevent unauthorized access to privileged resources.

2. **Bcrypt password encryption**

When a password needs to be viewed, it is decrypted using a decrypt token after being encrypted and stored in the database. By rendering the user's password unintelligible while it is kept in the database, this ensures its security.

3. **Registration and Login Form Validation**

It is not permitted to register more than once using the same email address. All users are automatically granted student privileges upon registration. Administrators can change a user's role in the system to give them administrator or teacher access as needed. A list of all users and their permissions is visible to administrators. Secure user authentication is achieved by enforcing password validation during login.

4. **Email Verification**

The users must verify their email addresses as an essential step in the registration process in order to receive access to the site. To confirm the user's identity, an OTP number is emailed to the user's email.

5. **Field Validation**

Only legitimate, correctly formatted data—such as emails and passwords with a minimum of eight characters—is permitted thanks to validation methods. Only numeric input is accepted for course enrollment IDs; all fields are necessary. By implementing tight validation guidelines, this stops security risks and data corruption.

5. Orchestrating and Deployment Strategy (Docker + Kubernetes)

1. **Dockerization**

To create Docker images of the services, first the following Docker files had to be created.

User Service

```
FROM node:20.11.1

# Set the working directory in the container to /app
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install the application dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose port 4001 for the application
EXPOSE 4001

# Start the application
CMD [ "node", "index.js" ]
```

Learner Service

```
FROM node:20.11.1

# Set the working directory in the container to /app
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install the application dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose port 4002 for the application
EXPOSE 4002

# Start the application
CMD [ "node", "index.js" ]
```

Enroll Service

```
FROM node:20.11.1

# Set the working directory in the container to /app
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install the application dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose port 4003 for the application
EXPOSE 4003

# Start the application
CMD [ "node", "index.js" ]
```

Course Service

```
FROM node:20.11.1

# Set the working directory in the container to /app
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install the application dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose port 4004 for the application
EXPOSE 4004

# Start the application
CMD [ "node", "index.js" ]
```

API Gateway

```
FROM node:20.11.1

# Set the working directory in the container to /app
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install the application dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose port 4000 for the application
EXPOSE 4000

# Start the application
CMD [ "node", "index.js" ]
```

.dockerignore files were added to the services.

```
.idea
node_modules
npm-debug.log
Jenkinsfile
```

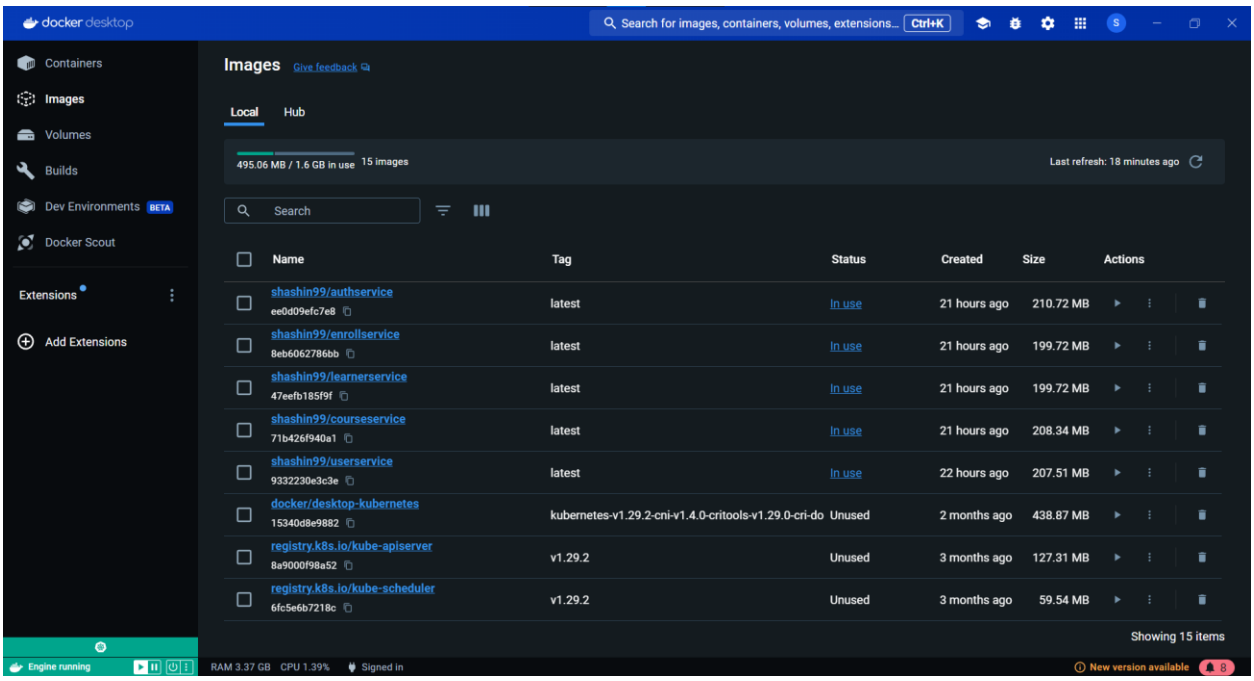
Then, in order to shorten the time needed to create the docker images for the services and API gateway, a docker-compose.yaml file was created.

```yaml
version: '3'
services:
  apiGateway:
    image: shashin99/apigateway:latest
    ports:
      - "4000:4000"
    command: node index.js

  userService:
    image: shashin99/userservice:latest
    ports:
      - "4001:4001"
    command: node index.js

  learnerService:
    image: shashin99/learnerservice:latest
    ports:
      - "4002:4002"
    command: node index.js

  enrollService:
    image: shashin99/enrollservice:latest
    ports:
      - "4003:4003"
    command: node index.js

  courseService:
    image: shashin99/courseservice:latest
    ports:
      - "4004:4004"
    command: node index.js

```

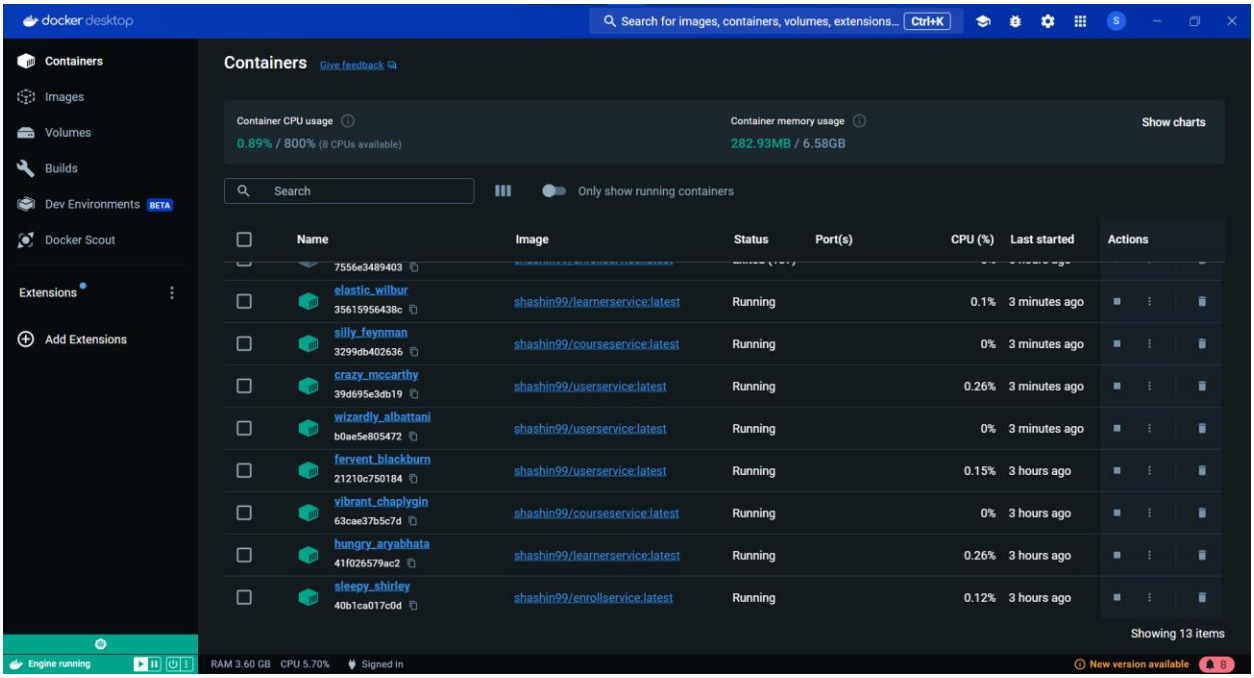
After that, the docker-compose.yaml file was used to generate the images and containers using the following command.

docker compose up -d

docker Images



docker Containers



2. Orchestration of the system using Kubernetes

The ***kubectl*** command was accessible following Kubernetes installation.

Using the following command, first confirm that Kubernetes is pointing towards docker-desktop.

Since it wasn't, modify the configuration using the second command.

kubectl config get-contexts

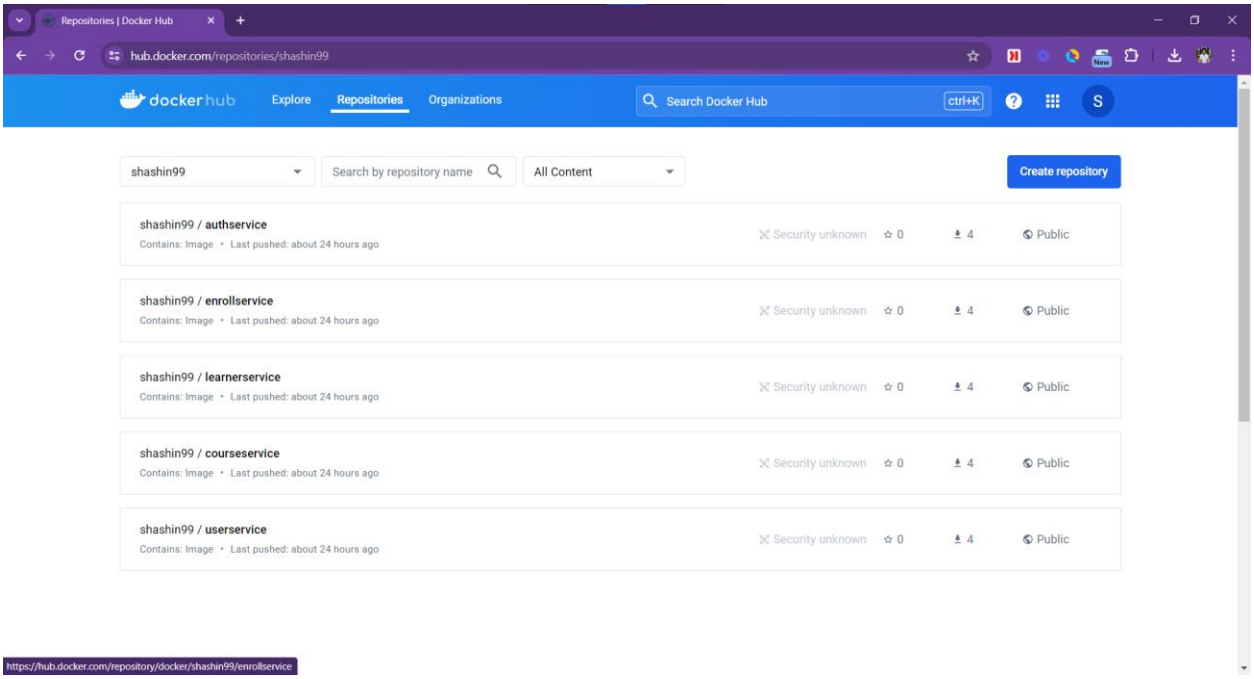
kubectl config use-context docker-desktop

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    COMMENTS

PS D:\SLIIT - Projects\Github\DS Assignment\Educational_Platform_for_Online_Learning_Microservices\Project\backend\UserService> kubectl config get-contexts

CURRENT   NAME             CLUSTER          AUTHINFO         NAMESPACE
*         docker-desktop   docker-desktop   docker-desktop
```

As indicated below, push the locally generated Docker images to the Docker Hub

```
REPOSITORY                          TAG                                                                IMAGE ID       CREATED
        SIZE
shashin99/authservice               latest                                                             ee0d09efc7e8   24 hours
 ago    211MB
shashin99/enrollservice             latest                                                             8eb6062786bb   25 hours
 ago    200MB
shashin99/learnerservice            latest                                                             47eefb185f9f   25 hours
 ago    200MB
shashin99/courseservice             latest                                                             71b426f940a1   25 hours
 ago    208MB
shashin99/userservice               latest                                                             9332230e3c3e   26 hours
 ago    208MB
docker/desktop-kubernetes           kubernetes-v1.29.2-cni-v1.4.0-critools-v1.29.0-cri-dockerd-v0.3.11-1-debian   15340d8e9882   2 months
 ago    439MB
```

Tagged the Docker images that required pushing to the Docker Hub, and following forced them to access the Docker Hub



After that, the files service.yaml and deployment.yaml were written in order to create the pods and services and map them using the designated port numbers.

Here shown the sample of one

```yaml
# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: user-service
  namespace: ds
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 4001
  selector:
    app: user-service
```

```yaml
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user-service
  namespace: ds
spec:
  replicas: 1
  selector:
    matchLabels:
      app: user-service
  template:
    metadata:
      labels:
        app: user-service
    spec:
      containers:
        - name: my-app
          image: shashin99/userservice:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 4001
      imagePullSecrets:
        - name: regcred
```

Create the namespace using following command.

kubectl create namespace ds

Apply the deployment.ymal file and service.ymal file

kubectl apply -f deployment.yaml
kubectl apply -f service.yaml

List the deployments

kubectl get deployments -n ds

```
PS D:\SLIIT - Projects\Github\DS Assignment\Educational_Platform_for_Online_Learning_Microservices\Project> kubectl get deployments -n ds
>>
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
user-service   1/1     1            1           36m
PS D:\SLIIT - Projects\Github\DS Assignment\Educational_Platform_for_Online_Learning_Microservices\Project>
```

Describe a specific deployment (user-service)

kubectl describe deployment user-service -n ds

```
PS D:\SLIIT - Projects\Github\DS Assignment\Educational_Platform_for_Online_Learning_Microservices\Project\backend\UserService> kubectl describe deployment user-service -n ds
>>
Name:                   user-service
Namespace:              ds
CreationTimestamp:      Mon, 13 May 2024 22:20:40 +0530
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=user-service
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=user-service
  Containers:
   my-app:
    Image:        shashin99/userservice:latest
    Port:         4001/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   user-service-774cf4b9d (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  33m   deployment-controller  Scaled up replica set user-service-774cf4b9d to 1
PS D:\SLIIT - Projects\Github\DS Assignment\Educational_Platform_for_Online_Learning_Microservices\Project\backend\UserService>
```

use ***kubectl get pods*** to list the pods and their images

kubectl get pods -n ds

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS                                   powershell

PS D:\SLIIT - Projects\Github\DS Assignment\Educational_Platform_for_Online_Learning_Microservices\Project> kubectl get pods -n ds
>>
NAME                            READY   STATUS    RESTARTS   AGE
user-service-774cf4b9d-6622p    1/1     Running   0          37m
PS D:\SLIIT - Projects\Github\DS Assignment\Educational_Platform_for_Online_Learning_Microservices\Project>
```

6. GitHub Repository Link:

https://github.com/Shashin99/Educational_Platform_for_Online_Learning_Microservices.git

7. YouTube Video Link:

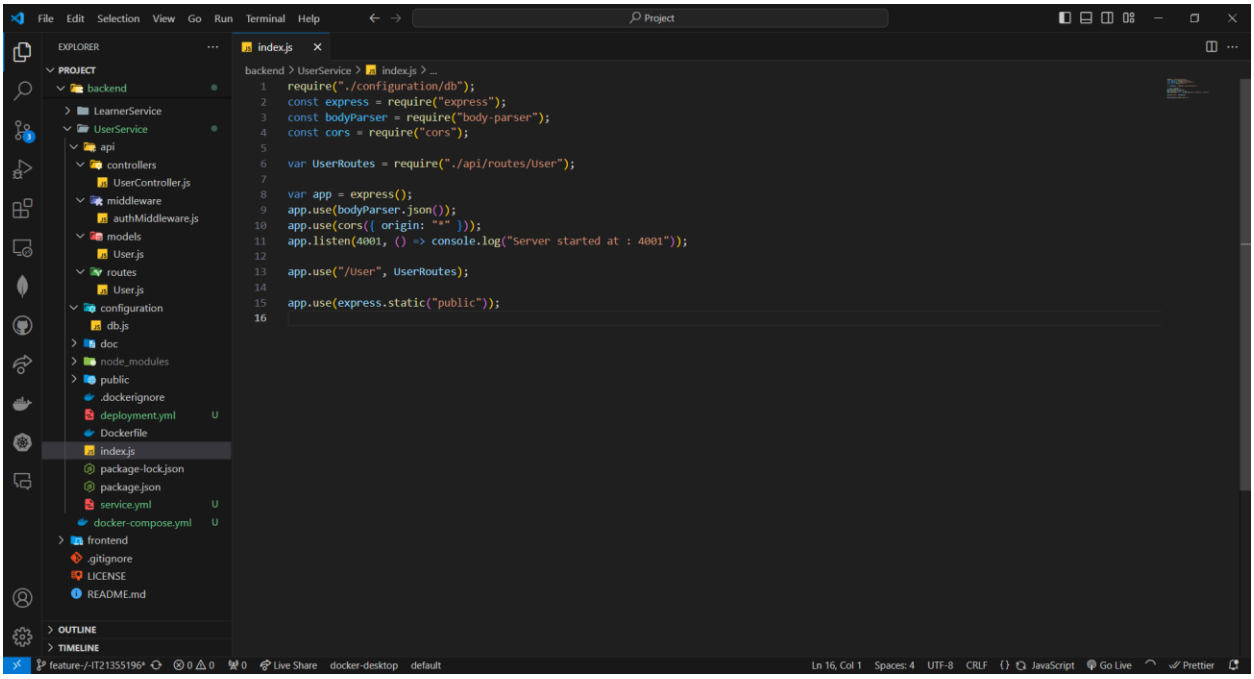
https://www.youtube.com/channel/UCwgDaMx__oyJl9EA1PmtM_Q

8. Individual Contribution

No	Registration Number	Name	Contribution
1	IT21377358	Hanshani S. G. H. S.	• Enroll Service Implementation • Enroll Management Frontend Implementation. • Payment, SMS and Email Function Implementation • Microservices Initiation • Documentation
2	IT21377280	Rajapaksha C. S.	• Course Service Implementation • Course Management Frontend Implementation. • Microservices Initiation • Documentation
3	IT21378270	Wimaladharma T. H. Y. B.	• Learner Service Implementation • Learner Management Frontend Implementation. • Microservices Initiation • Documentation
4	IT21355196	Kalpajith K. L. S.	• User Service Implementation & Email Function Implementation • User Management Frontend Implementation. • Microservices Initiation & API Gateway Implementation. • Docker and Kubernetes Implementation and Deployment • Documentation

Appendix

Folder Structure



Authmiddleware.js

```
const jwt = require("jsonwebtoken");

function verifyToken(req, res, next) {
    const token = req.header("Authorization");
    console.log(token);
    if (!token) return res.status(401).json({ error: "Access denied" });
    try {
        const decoded = jwt.verify(token, "my_key");
        req.userId = decoded.userId;
        req.userType = decoded.userType;
        next();
    } catch (error) {
        res.status(401).json({ error: "Invalid token" });
    }
}

module.exports = verifyToken;
```

```javascript
const jwt = require("jsonwebtoken");

function verifyToken(req, res, next) {
    const token = req.header("Authorization");
    console.log(token);
    if (!token) return res.status(401).json({ error: "Access denied" });
    try {
        const decoded = jwt.verify(token, "my_key");
        req.userId = decoded.userId;
        req.userType = decoded.userType;
        next();
    } catch (error) {
        res.status(401).json({ error: "Invalid token" });
    }
}

module.exports = verifyToken;
```

db.js

```javascript
const mongoose = require("mongoose");

mongoose.set("strictQuery", false);

mongoose.connect(

"mongodb+srv://Shashin:Shashin@cluster0.s3wm8gy.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0",
    { useNewUrlParser: true, useUnifiedTopology: true },
    (err) => {
        if (!err) {
            console.log("connection success!");
        } else {
            console.log("connection fail!" + JSON.stringify(err, undefined, 2));
        }
    }
);
```

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

mongoose.set("strictQuery", false);

mongoose.connect(
    "mongodb+srv://Shashin:Shashin@cluster0.s3wm8gy.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0",
    { useNewUrlParser: true, useUnifiedTopology: true },
    (err) => {
        if (!err) {
            console.log("connection success!");
        } else {
            console.log("connection fail!" + JSON.stringify(err, undefined, 2));
        }
    }
);
```

User Service

Model

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

var User = mongoose.model("User", {
    fname: {
        type: String,
        required: true,
    },

    lname: {
        type: String,
        required: true,
    },

    email: {
        type: String,
        required: true,
        unique: true,
    },

    privilege: {
        type: String,
        enum: ["admin", "instructor", "student"],
        required: true,
    },

    contact_no: {
        type: String,
        required: true,
    },
```

```
    access: {
        type: Boolean,
        required: true,
    },

    active: {
        type: Boolean,
        required: true,
    },

    otp: {
        type: String,
        required: true,
    },

    password: {
        type: String,
        required: true,
    },
});

module.exports = { User };
```

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

var User = mongoose.model("User", {
    fname: {
        type: String,
        required: true,
    },

    lname: {
        type: String,
        required: true,
    },

    email: {
        type: String,
        required: true,
        unique: true,
    },

    privilege: {
        type: String,
```

```
        enum: ["admin", "instructor", "student"],
        required: true,
    },

    contact_no: {
        type: String,
        required: true,
    },

    access: {
        type: Boolean,
        required: true,
    },

    active: {
        type: Boolean,
        required: true,
    },

    otp: {
        type: String,
        required: true,
    },

    password: {
        type: String,
        required: true,
    },
});

module.exports = { User };
```

Controller

```
const express = require("express");
var ObjectID = require("mongoose").Types.ObjectId;
var md5 = require("md5");
var { User } = require("../models/User");
const nodemailer = require("nodemailer");

exports.getAll = async (req, res) => {
   User.find((err, docs) => {
      if (!err) {
```

```
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
    });
};

exports.getId = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    User.findById(req.params.id, (err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
    });
};

exports.login = async (req, res) => {
    let email = req.body.email;
    let password = req.body.password;
    let userFound = await User.findOne({ email: email });
    if (userFound) {
        if (md5(password) == userFound.password) {
            if (userFound.active == true) {
                if (userFound.privilege == "admin") {
                    res.send(
                        JSON.stringify({
                            err: "success",
                            res: "admin",
                            email: userFound.email,
                            id: userFound._id,
                        })
                    );
                } else {
                    res.send(
                        JSON.stringify({
                            err: "success",
                            res: "user",
                            email: userFound.email,
                            id: userFound._id,
                        })
                    );
```

```
            }
        } else {
            res.status(200).send(
                JSON.stringify({ err: "activation fail" })
            );
        }
    } else {
        res.status(200).send(JSON.stringify({ err: "Incorrect Password" }));
    }
} else {
    res.status(200).send(JSON.stringify({ err: "User not found" }));
}
};

exports.newUser = async (req, res) => {
    var code = generate(6);
    var newRecord = new User({
        fname: req.body.fname,
        lname: req.body.lname,
        nic: req.body.nic,
        email: req.body.email,
        contact_no: req.body.contact_no,
        password: md5(req.body.password),
        access: false,
        active: false,
        otp: code,
        privilege: "student",
    });

    newRecord.save((err, docs) => {
        if (!err) {
            email(req.body.email, code);
            console.log(docs);
            email_with_subject(
                req.body.email,
                "REGISTRATION",
                `Hi, ${req.body.fname} ${req.body.lname} YOUR ACCOUNT HAS BEEN
SUCCESSFULLY  CREATED!`
            );
            res.status(200).send({ data: "success" });
        } else {
            console.log(err);
            if (err["keyPattern"]["email"] == 1) {
                console.log(err["keyPattern"]["email"]);
                res.status(200).send({ err: "email" });
            } else if (err["keyPattern"]["nic"] == 1) {
```

```
            console.log(err["keyPattern"]["nic"]);
            res.status(200).send({ err: "nic" });
        } else {
            res.status(err);
        }
    }
  });
};

exports.editUser = async (req, res) => {
  if (!ObjectID.isValid(req.params.id)) {
    return res.status(400).send(req.params.id);
  }

  var updateRecords = {
    privilege: req.body.privilege,
    active: req.body.active,
  };

  User.findByIdAndUpdate(
    req.params.id,
    { $set: updateRecords },
    { new: true },
    (err, docs) => {
      if (!err) {
        res.send(docs);
      } else {
        console.log(JSON.stringify(err, undefined, 2));
      }
    }
  );
};

exports.deleteUser = async (req, res) => {
  if (!ObjectID.isValid(req.params.id)) {
    return res.status(400).send(req.params.id);
  }

  User.findByIdAndRemove(req.params.id, (err, docs) => {
    if (!err) {
      res.send(docs);
    } else {
      res.send(err);
    }
  });
};
```

```
function email(email_address, code) {
    var transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: "shashinleo@gmail.com",
            pass: "sebrvbojstnliwbz",
        },
    });

    var mailOption = {
        from: "shashinleo@gmail.com",
        to: email_address,
        subject: "OTP Code",
        text: code,
    };

    transporter.sendMail(mailOption, function (error, info) {
        if (error) {
            res.send(error);
        } else {
            console.log("Message sent: %s", info.response);
            res.send(info.response);
        }
    });
}

function email_with_subject(email_address, subject, code) {
    var transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: "shashinleo@gmail.com",
            pass: "sebrvbojstnliwbz",
        },
    });

    var mailOption = {
        from: "shashinleo@gmail.com",
        to: email_address,
        subject: subject,
        text: code,
    };

    transporter.sendMail(mailOption, function (error, info) {
        if (error) {
            res.send(error);
```

```
        } else {
            console.log("Message sent: %s", info.response);
            res.send(info.response);
        }
    });
}

function generate(n) {
    var add = 1,
        max = 12 - add;
    if (n > max) {
        return generate(max) + generate(n - max);
    }
    max = Math.pow(10, n + add);
    var min = max / 10;
    var number = Math.floor(Math.random() * (max - min + 1)) + min;
    return ("" + number).substring(add);
}
```

```
const express = require("express");
var ObjectID = require("mongoose").Types.ObjectId;
var md5 = require("md5");
var { User } = require("../models/User");
const nodemailer = require("nodemailer");

exports.getAll = async (req, res) => {
    User.find((err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
    });
};

exports.getId = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    User.findById(req.params.id, (err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
```

```
    });
};

exports.login = async (req, res) => {
    let email = req.body.email;
    let password = req.body.password;
    let userFound = await User.findOne({ email: email });
    if (userFound) {
        if (md5(password) == userFound.password) {
            if (userFound.active == true) {
                if (userFound.privilege == "admin") {
                    res.send(
                        JSON.stringify({
                            err: "success",
                            res: "admin",
                            email: userFound.email,
                            id: userFound._id,
                        })
                    );
                } else {
                    res.send(
                        JSON.stringify({
                            err: "success",
                            res: "user",
                            email: userFound.email,
                            id: userFound._id,
                        })
                    );
                }
            } else {
                res.status(200).send(
                    JSON.stringify({ err: "activation fail" })
                );
            }
        } else {
            res.status(200).send(JSON.stringify({ err: "Incorrect Password" }));
        }
    } else {
        res.status(200).send(JSON.stringify({ err: "User not found" }));
    }
};

exports.newUser = async (req, res) => {
    var code = generate(6);
    var newRecord = new User({
```

```
        fname: req.body.fname,
        lname: req.body.lname,
        nic: req.body.nic,
        email: req.body.email,
        contact_no: req.body.contact_no,
        password: md5(req.body.password),
        access: false,
        active: false,
        otp: code,
        privilege: "student",
    });

    newRecord.save((err, docs) => {
        if (!err) {
            email(req.body.email, code);
            console.log(docs);
            email_with_subject(
                req.body.email,
                "REGISTRATION",
                `Hi, ${req.body.fname} ${req.body.lname} YOUR ACCOUNT HAS BEEN
SUCCESSFULLY CREATED!`
            );
            res.status(200).send({ data: "success" });
        } else {
            console.log(err);
            if (err["keyPattern"]["email"] == 1) {
                console.log(err["keyPattern"]["email"]);
                res.status(200).send({ err: "email" });
            } else if (err["keyPattern"]["nic"] == 1) {
                console.log(err["keyPattern"]["nic"]);
                res.status(200).send({ err: "nic" });
            } else {
                res.status(err);
            }
        }
    });
};

exports.editUser = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    var updateRecords = {
        privilege: req.body.privilege,
```

```
        active: req.body.active,
    };

    User.findByIdAndUpdate(
        req.params.id,
        { $set: updateRecords },
        { new: true },
        (err, docs) => {
            if (!err) {
                res.send(docs);
            } else {
                console.log(JSON.stringify(err, undefined, 2));
            }
        }
    );
};

exports.deleteUser = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    User.findByIdAndRemove(req.params.id, (err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            res.send(err);
        }
    });
};

function email(email_address, code) {
    var transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: "shashinleo@gmail.com",
            pass: "sebrvbojstnliwbz",
        },
    });

    var mailOption = {
        from: "shashinleo@gmail.com",
        to: email_address,
        subject: "OTP Code",
        text: code,
```

```
    };

    transporter.sendMail(mailOption, function (error, info) {
        if (error) {
            res.send(error);
        } else {
            console.log("Message sent: %s", info.response);
            res.send(info.response);
        }
    });
}

function email_with_subject(email_address, subject, code) {
    var transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: "shashinleo@gmail.com",
            pass: "sebrvbojstnliwbz",
        },
    });

    var mailOption = {
        from: "shashinleo@gmail.com",
        to: email_address,
        subject: subject,
        text: code,
    };

    transporter.sendMail(mailOption, function (error, info) {
        if (error) {
            res.send(error);
        } else {
            console.log("Message sent: %s", info.response);
            res.send(info.response);
        }
    });
}

function generate(n) {
    var add = 1,
        max = 12 - add;
    if (n > max) {
        return generate(max) + generate(n - max);
    }
    max = Math.pow(10, n + add);
```

```
    var min = max / 10;
    var number = Math.floor(Math.random() * (max - min + 1)) + min;
    return ("" + number).substring(add);
}
```

Route

```
const express = require("express");
const verifyToken = require("../middleware/authMiddleware");

const {
    getAll,
    getId,
    login,
    newUser,
    editUser,
    deleteUser,
    otp,
} = require("../controllers/UserController");

const router = express.Router();

router.get("/", verifyToken, getAll);
router.get("/:id", verifyToken, getId);
router.post("/", newUser);
router.put("/:id", verifyToken, editUser);
router.delete("/:id", verifyToken, deleteUser);
router.post("/login", login);
router.post("/otp", otp);

module.exports = router;
```

```
const express = require("express");
const verifyToken = require("../middleware/authMiddleware");

const {
    getAll,
    getId,
    login,
    newUser,
    editUser,
    deleteUser,
    otp,
} = require("../controllers/UserController");
```

```
const router = express.Router();

router.get("/", verifyToken, getAll);
router.get("/:id", verifyToken, getId);
router.post("/", newUser);
router.put("/:id", verifyToken, editUser);
router.delete("/:id", verifyToken, deleteUser);
router.post("/login", login);
router.post("/otp", otp);

module.exports = router;
```

index.js

require("./configuration/db");
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

var UserRoutes = require("./api/routes/User");

var app = express();
app.use(bodyParser.json());
app.use(cors({ origin: "*" }));
app.listen(4001, () => console.log("Server started at : 4001"));

app.use("/User", UserRoutes);

app.use(express.static("public"));

```
require("./configuration/db");
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

var UserRoutes = require("./api/routes/User");

var app = express();
app.use(bodyParser.json());
app.use(cors({ origin: "*" }));
app.listen(4001, () => console.log("Server started at : 4001"));

app.use("/User", UserRoutes);
```

```
app.use(express.static("public"));
```

Learner Service

Model

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

var Learner = mongoose.model("Learner", {
    course_id: {
        type: String,
        required: true,
    },

    user_id: {
        type: String,
        required: true,
    },

    note1: {
        type: Boolean,
        required: false,
    },

    note2: {
        type: Boolean,
        default: false,
    },

    note3: {
        type: Boolean,
        default: false,
    },

    note4: {
        type: Boolean,
        default: false,
    },

    note5: {
        type: Boolean,
        default: false,
    },
```

```
});

module.exports = { Learner };
```

Controller

```
const express = require("express");
var ObjectID = require("mongoose").Types.ObjectId;
var { Learner } = require("../models/Learner");
var { Enroll } = require("../models/Enroll");
const nodemailer = require("nodemailer");

exports.getAll = async (req, res) => {
    Learner.find((err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
    });
};

exports.getId = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    Learner.findById(req.params.id, (err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
    });
};

exports.newLearner = async (req, res) => {
    let lFound = await Learner.findOne({
        course_id: req.body.course_id,
        user_id: req.body.user_id,
    });
    if (lFound) {
        if (req.body.note1 == true) {
```

```
            var updateRecords = {
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note1: req.body.note1,
            };
        } else if (req.body.note2 == true) {
            var updateRecords = {
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note2: req.body.note2,
            };
        } else if (req.body.note3 == true) {
            var updateRecords = {
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note3: req.body.note3,
            };
        } else if (req.body.note4 == true) {
            var updateRecords = {
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note4: req.body.note4,
            };
        } else {
            var updateRecords = {
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note5: req.body.note5,
            };
        }
        console.log(lFound._id.toHexString());
        Learner.findByIdAndUpdate(
            lFound._id.toHexString(),
            { $set: updateRecords },
            { new: true },
            async (err, docs) => {
                if (!err) {
                    let lFound1 = await Learner.findOne({
                        course_id: req.body.course_id,
                        user_id: req.body.user_id,
                    });
                    if (lFound1) {
                        if (
                            lFound1.note1 == true &&
                            lFound1.note2 == true &&
```

```
                            lFound1.note3 == true &&
                            lFound1.note4 == true &&
                            lFound1.note5 == true
                        ) {
                            let eFound = await Enroll.findOne({
                                course_id: req.body.course_id,
                                user_id: req.body.user_id,
                            });
                            var updateRecords = {
                                status: "completed",
                            };
                            Enroll.findByIdAndUpdate(
                                eFound._id,
                                { $set: updateRecords },
                                { new: true },
                                (err, docs) => {
                                    if (!err) {
                                        email_with_subject(
                                            req.body.email,
                                            "Course Enrollment",
                                            "You have successfully enroll to the
course."
                                        );
                                        res.send(docs);
                                    } else {
                                        console.log(
                                            JSON.stringify(err, undefined, 2)
                                        );
                                    }
                                }
                            );
                        }
                    }
                } else {
                    console.log(JSON.stringify(err, undefined, 2));
                }
            }
        );
    } else {
        if (req.body.note1 == true) {
            var newRecord = new Learner({
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note1: req.body.note1,
            });
```

```
        } else if (req.body.note2 == true) {
            var newRecord = new Learner({
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note2: req.body.note2,
            });
        } else if (req.body.note3 == true) {
            var newRecord = new Learner({
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note3: req.body.note3,
            });
        } else if (req.body.note4 == true) {
            var newRecord = new Learner({
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note4: req.body.note4,
            });
        } else {
            var newRecord = new Learner({
                course_id: req.body.course_id,
                user_id: req.body.user_id,
                note5: req.body.note5,
            });
        }
        newRecord.save((err, docs) => {
            if (!err) {
                console.log(docs);
                res.status(200).send({ data: "success" });
            } else {
                res.status(err);
            }
        });
    }
};

exports.editLearner = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    var updateRecords = {
        course_id: req.body.course_id,
        user_id: req.body.user_id,
        note1: req.body.note1,
```

```
        note2: req.body.note2,
        note3: req.body.note3,
        note4: req.body.note4,
        note5: req.body.note5,
    };

    Learner.findByIdAndUpdate(
        lFound._id.toHexString(),
        { $set: updateRecords },
        { new: true },
        (err, docs) => {
            if (!err) {
                res.send(docs);
            } else {
                console.log(JSON.stringify(err, undefined, 2));
            }
        }
    );
};

exports.deleteLearner = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    Learner.findByIdAndRemove(req.params.id, (err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            res.send(err);
        }
    });
};

function email_with_subject(email_address, subject, code) {
    var transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: "shashinleo@gmail.com",
            pass: "sebrvbojstnliwbz",
        },
    });

    var mailOption = {
        from: "shashinleo@gmail.com",
```

```
        to: email_address,
        subject: subject,
        text: code,
    };

    transporter.sendMail(mailOption, function (error, info) {
        if (error) {
            res.send(error);
        } else {
            console.log("Message sent: %s", info.response);
            res.send(info.response);
        }
    });
}
```

Route

```
const express = require("express");
const verifyToken = require("../middleware/authMiddleware");

const {
    getAll,
    getId,
    newLearner,
    editLearner,
    deleteLearner,
} = require("../controllers/LearnerController");

const router = express.Router();

router.get("/", verifyToken, getAll);
router.get("/:id", verifyToken, getId);
router.post("/", newLearner);
router.put("/:id", verifyToken, editLearner);
router.delete("/:id", verifyToken, deleteLearner);

module.exports = router;
```

index.js

```
require("./configuration/db.js");
```

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

var LearnerRoutes = require("./api/routes/Learner.js");

var app = express();
app.use(bodyParser.json());
app.use(cors({ origin: "*" }));
app.listen(4002, () => console.log("Server started at : 4002"));

app.use("/Learner", LearnerRoutes);

app.use(express.static("public"));
```

Enroll Service

Model

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

var Enroll = mongoose.model("Enroll", {
    course_id: {
        type: String,
        required: true,
    },

    user_id: {
        type: String,
        required: true,
    },

    date: {
        type: String,
        required: true,
    },

    time: {
        type: String,
        required: true,
    },

    status: {
```

```
        type: String,
        enum: ["completed", "inprogress"],
        required: true,
    },
});

module.exports = { Enroll };
```

Controller

```
const express = require("express");
var ObjectID = require("mongoose").Types.ObjectId;
var { Enroll } = require("../models/Enroll");
const nodemailer = require("nodemailer");

exports.getAll = async (req, res) => {
    Enroll.find((err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
    });
};

exports.getId = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    Enroll.findById(req.params.id, (err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
    });
};

exports.userId = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }
```

```
    await Enroll.aggregate([
        { $addFields: { courseid: { $toObjectId: "$course_id" } } },
        { $addFields: { user: { $toObjectId: "$user_id" } } },
        {
            $lookup: {
                from: "courses",
                localField: "courseid",
                foreignField: "_id",
                as: "course_details",
            },
        },
        { $unwind: "$course_details" },
        { $match: { user: ObjectID(req.params.id) } },
    ])
        .then((result) => {
            res.send(result);
        })
        .catch((error) => {
            res.send(error);
        });
};

exports.newEnroll = async (req, res) => {
    let enrollFound = await Enroll.findOne({
        course_id: req.body.course_id,
        user_id: req.body.user_id,
    });
    if (enrollFound) {
        res.status(200).send({ data: "error" });
    } else {
        var newRecord = new Enroll({
            course_id: req.body.course_id,
            user_id: req.body.user_id,
            date: req.body.date,
            time: req.body.time,
            status: "inprogress",
        });

        newRecord.save((err, docs) => {
            if (!err) {
                console.log(docs);
                res.status(200).send({ data: "success" });
            } else {
                res.status(err);
```

```
            }
        });
    }
};

exports.editEnroll = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    var updateRecords = {
        course_id: req.body.course_id,
        user_id: req.body.user_id,
        date: req.body.date,
        time: req.body.time,
        status: req.body.status,
    };

    Enroll.findByIdAndUpdate(
        req.params.id,
        { $set: updateRecords },
        { new: true },
        (err, docs) => {
            if (!err) {
                email_with_subject(req.body.email, "Subject", "Body");
                res.send(docs);
            } else {
                console.log(JSON.stringify(err, undefined, 2));
            }
        }
    );
};

exports.deleteEnroll = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    Enroll.findByIdAndRemove(req.params.id, (err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            res.send(err);
        }
    });
```

```
};

function email_with_subject(email_address, subject, code) {
    var transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: "shashinleo@gmail.com",
            pass: "sebrvbojstnliwbz",
        },
    });

    var mailOption = {
        from: "shashinleo@gmail.com",
        to: email_address,
        subject: subject,
        text: code,
    };

    transporter.sendMail(mailOption, function (error, info) {
        if (error) {
            res.send(error);
        } else {
            console.log("Message sent: %s", info.response);
            res.send(info.response);
        }
    });
}
```

Route

```
const express = require("express");
const verifyToken = require("../middleware/authMiddleware");

const {
    getAll,
    getId,
    newEnroll,
    editEnroll,
    deleteEnroll,
} = require("../controllers/EnrollController");

const router = express.Router();

router.get("/", verifyToken, getAll);
```

```
router.get("/:id", verifyToken, getId);
router.post("/", verifyToken, newEnroll);
router.put("/:id", verifyToken, editEnroll);
router.delete("/:id", verifyToken, deleteEnroll);

module.exports = router;
```

index.js

```
require("./configuration/db.js");
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

var EnrollRoutes = require("./api/routes/Enroll.js");

var app = express();
app.use(bodyParser.json());
app.use(cors({ origin: "*" }));
app.listen(4003, () => console.log("Server started at : 4003"));

app.use("/Enroll", EnrollRoutes);

app.use(express.static("public"));
```

Course Service

Model

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

var Course = mongoose.model("Course", {
    note1: {
        type: String,
        required: true,
    },

    note2: {
        type: String,
        default: "",
    },
```

```
    note3: {
        type: String,
        default: "",
    },

    note4: {
        type: String,
        default: "",
    },

    note5: {
        type: String,
        default: "",
    },

    video_link: {
        type: String,
        required: true,
    },

    quiz_details: {
        type: String,
        required: true,
    },
});

module.exports = { Course };
```

Controller

```
const express = require("express");
var ObjectID = require("mongoose").Types.ObjectId;
var { Course } = require("../models/Course");
var multer = require("multer");
var uniqid = require("uniqid");

exports.getAll = async (req, res) => {
    Course.find((err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
    });
```

```
};

exports.getId = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    Course.findById(req.params.id, (err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            console.log(JSON.stringify(err, undefined, 2));
        }
    });
};

exports.newCourse = async (req, res) => {
    var newRecord = new Course({
        name: req.body.name,
        note1: req.body.note1,
        note2: req.body.note2,
        note3: req.body.note3,
        note4: req.body.note4,
        note5: req.body.note5,
        price: req.body.price,
        video_link: req.body.video_link,
        quiz_details: req.body.quiz_details,
    });

    newRecord.save((err, docs) => {
        if (!err) {
            console.log(docs);
            res.status(200).send({ data: "success" });
        } else {
            res.status(err);
        }
    });
};

exports.editCourse = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    var updateRecords = {
```

```
        name: req.body.name,
        note1: req.body.note1,
        note2: req.body.note2,
        note3: req.body.note3,
        note4: req.body.note4,
        note5: req.body.note5,
        price: req.body.price,
        video_link: req.body.video_link,
        quiz_details: req.body.quiz_details,
    };

    Course.findByIdAndUpdate(
        req.params.id,
        { $set: updateRecords },
        { new: true },
        (err, docs) => {
            if (!err) {
                res.send(docs);
            } else {
                console.log(JSON.stringify(err, undefined, 2));
            }
        }
    );
};

exports.deleteCourse = async (req, res) => {
    if (!ObjectID.isValid(req.params.id)) {
        return res.status(400).send(req.params.id);
    }

    Course.findByIdAndRemove(req.params.id, (err, docs) => {
        if (!err) {
            res.send(docs);
        } else {
            res.send(err);
        }
    });
};

var storage = multer.diskStorage({
    destination: function (req, file, cb) {
        cb(null, "public");
    },
    filename: function (req, file, cb) {
        cb(null, uniqid() + "-" + file.originalname);
```

```
    },
});

var upload = multer({ storage: storage }).single("file");

exports.uploadFile = async (req, res) => {
    upload(req, res, function (err) {
        if (err instanceof multer.MulterError) {
            return res.status(500).json(err);
        } else if (err) {
            return res.status(500).json(err);
        }
        return res.status(200).send(req.file);
    });
};
```

Route

```
const express = require("express");
const verifyToken = require("../middleware/authMiddleware");

const {
    getAll,
    getId,
    newCourse,
    editCourse,
    deleteCourse,
    uploadFile,
} = require("../controllers/CourseController");

const router = express.Router();

router.get("/", verifyToken, getAll);
router.get("/:id", verifyToken, getId);
router.post("/", verifyToken, newCourse);
router.post("/upload", verifyToken, uploadFile);
router.put("/:id", verifyToken, editCourse);
router.delete("/:id", verifyToken, deleteCourse);

module.exports = router;
```

index.js

```
require("./configuration/db.js");
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

var CourseRoutes = require("./api/routes/Course.js");

var app = express();
app.use(bodyParser.json());
app.use(cors({ origin: "*" }));
app.listen(4004, () => console.log("Server started at : 4004"));

app.use("/Course", CourseRoutes);

app.use(express.static("public"));
```

API Gateway

index.js

```
const express = require("express");
const cors = require("cors");
const expressSession = require("express-session");
const proxy = require("express-http-proxy");
const bodyParser = require("body-parser");

const app = express();

// Middleware
app.use(
    cors({
        origin: "http://localhost:3000",
        credentials: true,
    })
);
app.use(express.json());
app.set("trust proxy", 1);

const sessSettings = expressSession({
    secret: "oursecret",
    resave: true,
    saveUninitialized: true,
    cookie: {
```

```
        secure: false, // For development, you can set it to true in production
if using HTTPS
        maxAge: 360000,
    },
});
app.use(sessSettings);

// Proxy middleware
app.use("/apiuser", proxy("http://localhost:4001"));
app.use("/apilearner", proxy("http://localhost:4002"));
app.use("/apienroll", proxy("http://localhost:4003"));
app.use("/apicourse", proxy("http://localhost:4004"));

app.get("/", (req, res) => {
    res.status(200).json({ message: "Gateway Server is running!" });
});
app.listen(4000, "0.0.0.0", () => console.log("Server started at : 4000"));
```