Sri Lanka Institute of Information Technology

Distributed Systems

Assignment 1- SE3030

Group Details : Y3S1.20(WD) – **Group_28**

Registration Number	Name
IT21219320	Ayeshmantha S.K.S
IT21827662	Dissanayka S.D
IT21256266	Kumara B.D.A.N
IT21220760	Dias D.D.K.S

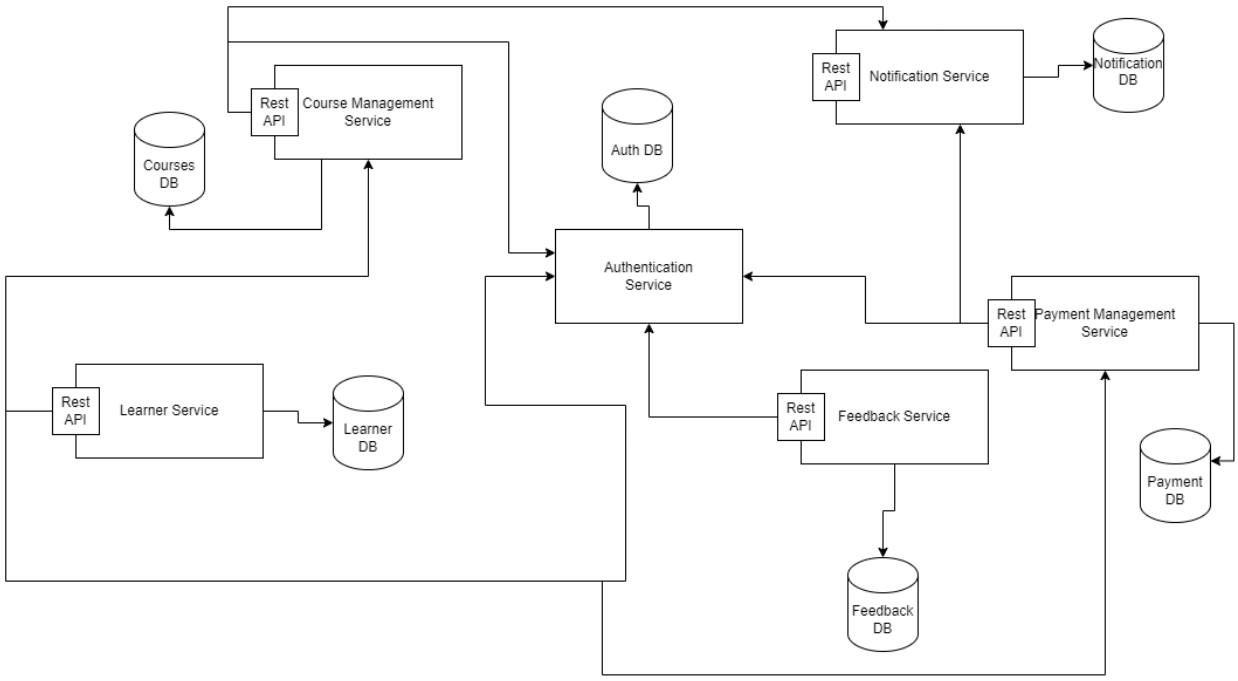
Table of Contents

Project Scope

This application was developed under 6 main services.

1. Authentication service
2. Course management service
3. Enrollment service
4. Payment service
5. Notification service
6. Feedback service

Architectural Diagram



Service interfaces

Authentication service (Base URL: http://localhost:8082)

1. Register new user: POST /v1/new
2. Login: POST /v1/login
3. Get OTP code: POST /forgotPassword /verifyMail/{email}
4. Verify OTP: POST /forgotPassword/ /verifyOtp/{otp}/{email}
5. Change password: PATCH /forgotPassword/changePassword/{email}
6. Get all users: GET v1/users/all
7. Update user by email: PATCH v1/user/{email}
8. Update user password: PATCH v1/user/userProfile/updatePassword/{email}
9. Delete user by id: DELETE v1/user/{id}

Course management service (Base URL: http://localhost:5000)

1. Register a new Course: POST /api/course/create
2. Retrieve All Courses: GET /api/course/all
3. Retrieve All Courses: GET /api/course/instructor/all
4. Retrieve Courses By Id: GET /api/course/one/:id
5. Update Course Details: PUT /api/course/update
6. Approve/Reject Course: PATCH /api/course/approve/:id/:approve
7. Delete Course: DELETE /api/course/delete/:id
8. Add a new content category to course: POST /api/course/content/create

Learner service (Base URL: http://localhost:8000)

1. Enroll to a new course: POST / enrollment/enroll
2. Retreive enrolled courses by userEmail: GET / enrollment/enrolledCourses/:userEmail
3. Cancell enrollment DELETE / enrollment/cancel
4. Retreive all enrolled your by course id GET / enrollment/usersByCourse/:courseId
5. Add completed task by learner POST /progress/tracking
6. Retreive learner progress by course id GET /progress/tracking/:userEmail/:courseId

Payment service (Base URL: http://localhost:3001)

1. Redirect customers to checkout page: GET /
2. Redirect customers to payment success page and enroll to the purchased course: GET /return
3. Redirect customers to payment failure page and send a notification: GET /cancel
4. PayHere notifies the payment status to the server: POST /notify

Notification service (Base URL: http://localhost:4000)

1. Send notification: POST /send-notification
2. Send OTP: POST/send-email

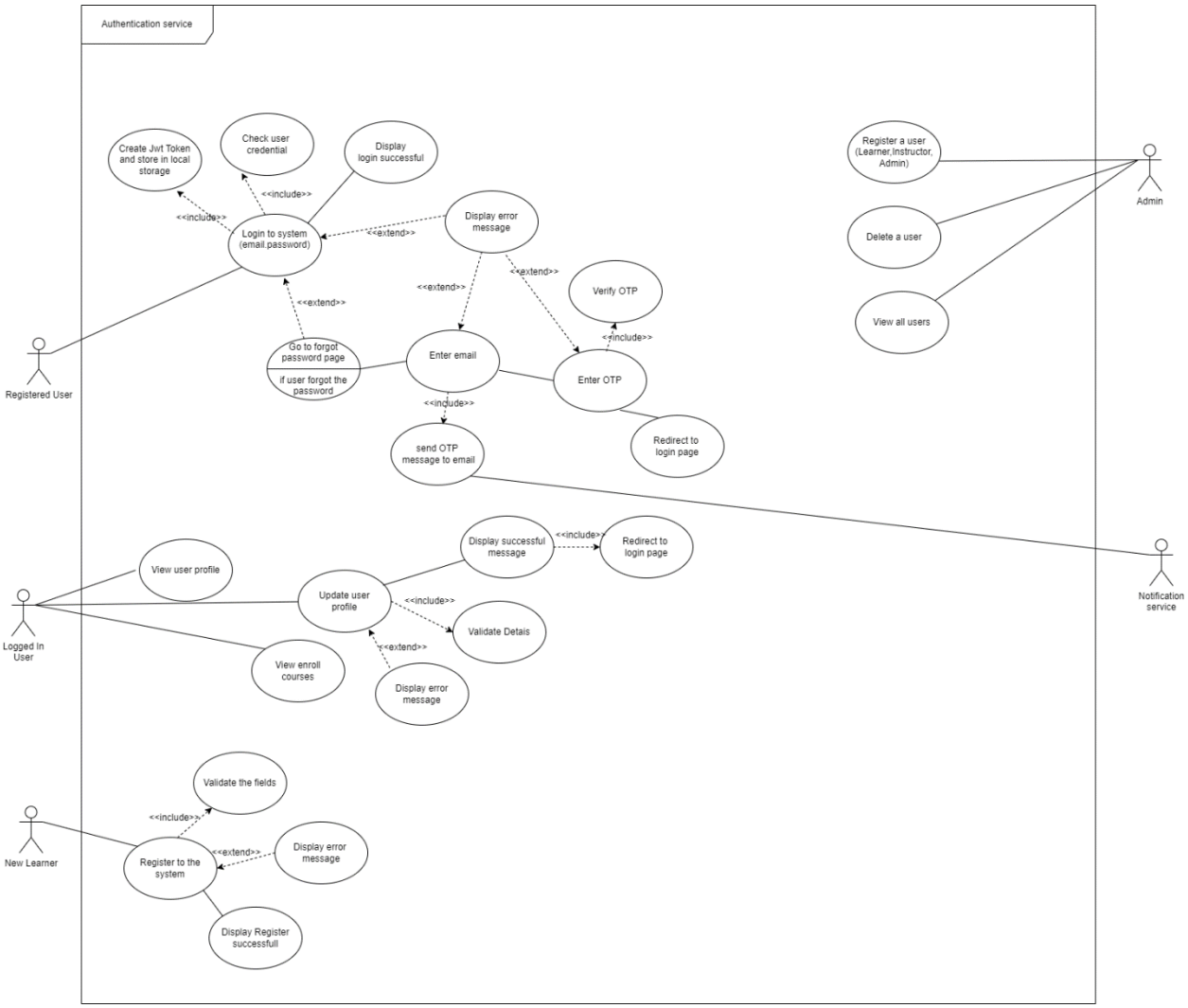
Feedback service (Base URL: http://localhost:5080)

1. Add feedback by learner POST /feedback/add
2. Delete feedback by feedback id DELETE /feedback/:id
3. Update feedback by feedback id PUT /feedback/:id
4. Retreive feedback by course id GET /feedback/course/:id

Authentication service

This microservice, utilizing Spring Boot and MySQL, facilitates user sign-in, registration, and profile management while incorporating JSON Web Token (JWT) for enhanced security. When users log in, their credentials are validated, and upon successful authentication, they receive a JWT token, securely stored to enable seamless access. During registration, stringent checks ensure each user possesses a unique email address to prevent duplicates. Users can update their profiles with ease, with the system ensuring data accuracy through validation checks. Administrators wield additional capabilities to efficiently manage all users within the system. With the integration of Spring Security and JWT, the service upholds robust security standards, safeguarding user data from potential threats.

Use case diagram for Authentication service

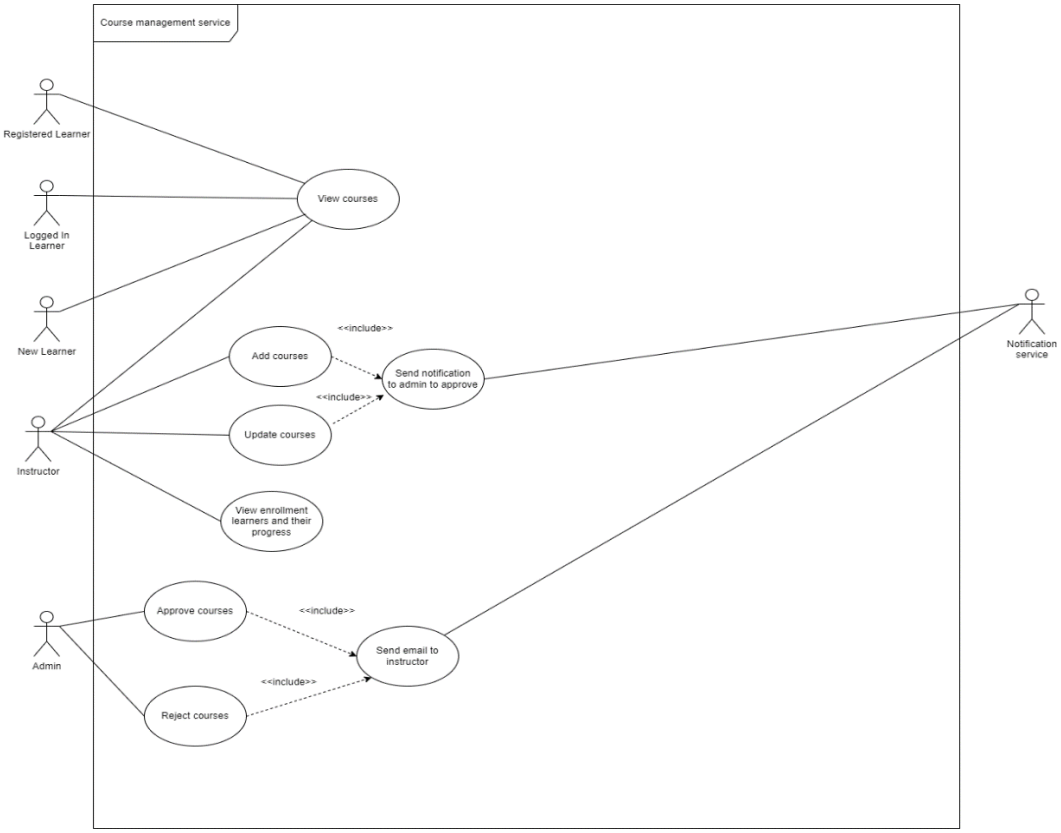


Course management service

The Course Management Microservice, developed with Node.js and MySQL, streamlines course creation, categorization, and oversight. Instructors can effortlessly add courses, assign categories, and upload content while seamlessly integrating with the user authentication microservice for secure access and learner microservice for enrollment and progress tracking. Courses undergo an approval process by administrators before public release, ensuring quality. Instructors receive notifications regarding course approval status via integration with the notification service. Additionally, the microservice incorporates a middle layer for security, featuring three access levels: Level 1 protects routes to allow access only for logged-in users, Level 2 restricts access to admins and instructors, and Level 3 grants access exclusively to administrators. Furthermore, it

includes functionality to decode the JWT token generated by the authentication microservice, ensuring secure and authorized access to course management features. Overall, the microservice offers a user-friendly platform for instructors to manage courses effectively while providing learners with a seamless learning experience.

Use case diagram for Course management service

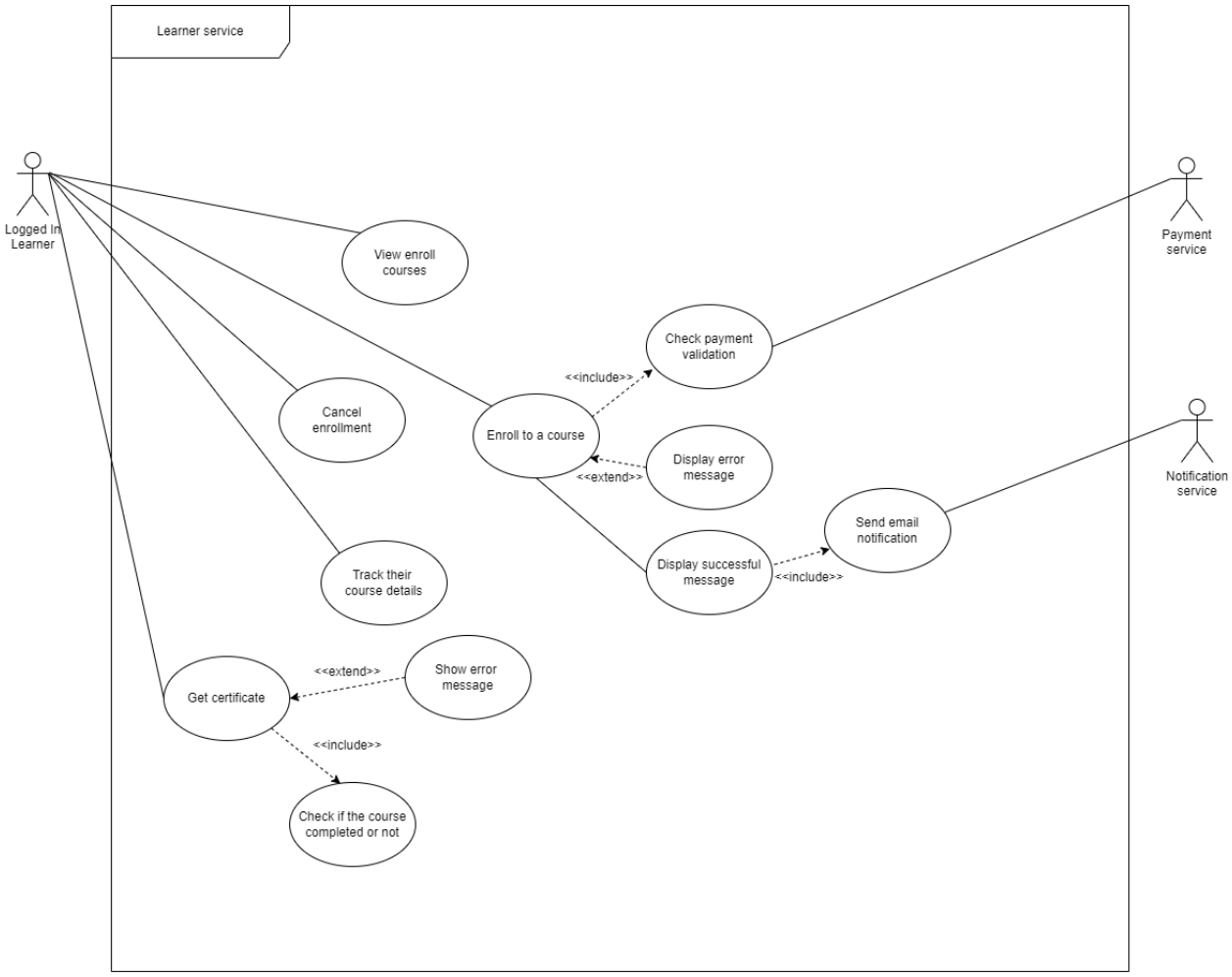


Learner service

The Learning Management System (LMS) Microservice, built on MongoDB, Node.js, Express.js, and React.js, empowers learners with a comprehensive platform for course exploration, enrollment, and progression tracking. Learners can effortlessly browse available courses, enroll in their chosen ones, and seamlessly access course materials.Integration with the user authentication microservice ensures secure access, safeguarding learner data and activities. Additionally, seamless integration with the course management service enables learners to access detailed course information, including descriptions, instructors, and schedules.Furthermore, the feedback and rating service allows learners to provide valuable insights on courses they've completed, enhancing transparency and aiding future learners in their decision-making process. Learners can view

feedback and ratings left by others, fostering a supportive learning community.Moreover, the payment service facilitates secure transactions for course enrollment, ensuring a smooth and reliable experience for learners. With transparent payment processing, learners can enroll in courses hassle-free, focusing on their educational journey. Summarizing, Microservice offers a user-centric approach, providing learners with a seamless and enriching learning experience while prioritizing security, accessibility, and functionality.

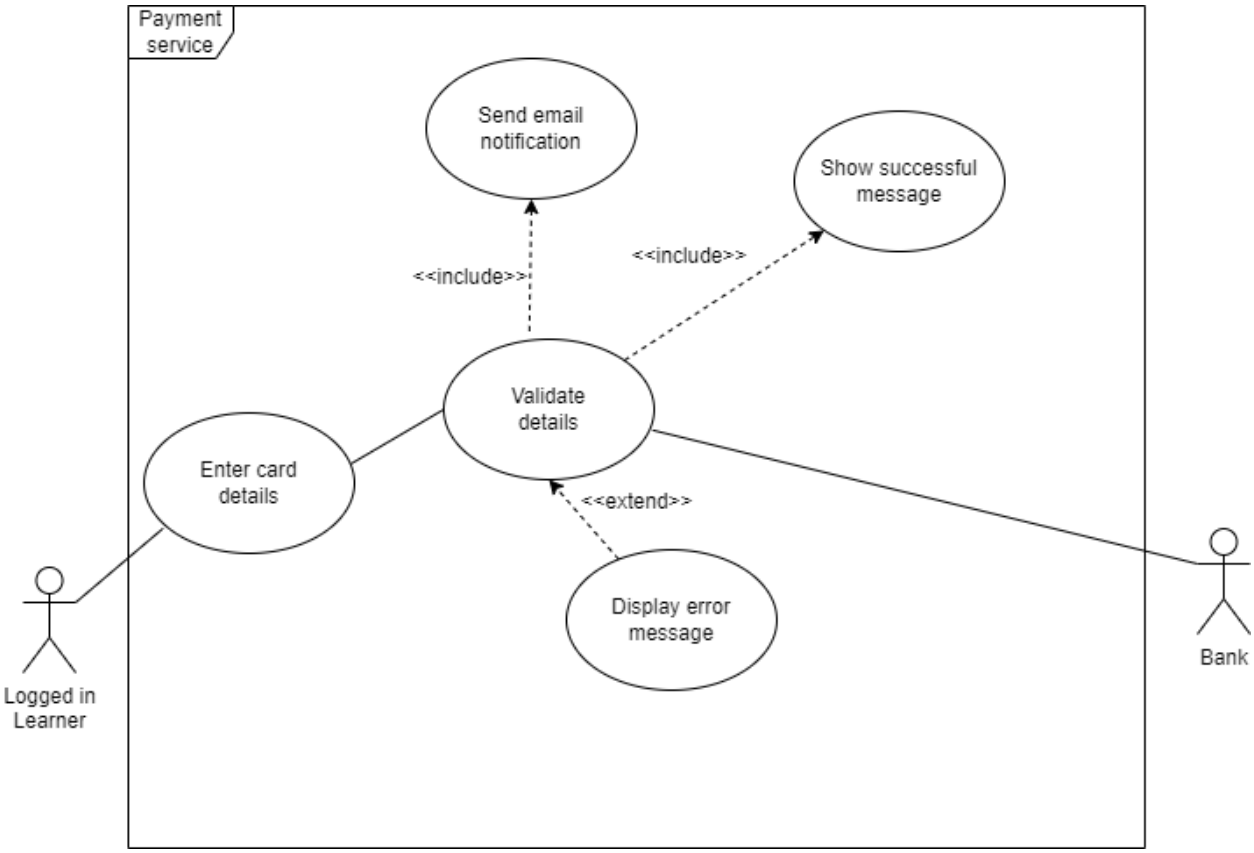
Use case diagram for Learner service



Payment service

The Payment Microservice for the Study Storm Project is meticulously engineered with Node.js, Express.js, MongoDB, and EJS, serving as the backbone of the platform's financial operations. Seamlessly integrating with the Pay Here payment gateway, it optimizes transaction processes and facilitates student enrollments with efficiency. Leveraging the robust capabilities of Node.js, Express.js, and MongoDB, the microservice ensures swift and secure payment processing, complemented by dynamic user interfaces powered by EJS. Upholding the highest security standards, advanced authentication and authorization mechanisms are implemented, aligning with Pay Here's stringent protocols. In essence, the Payment Microservice embodies technological excellence, enabling Study Storm to deliver seamless financial transactions while prioritizing data security and user experience.

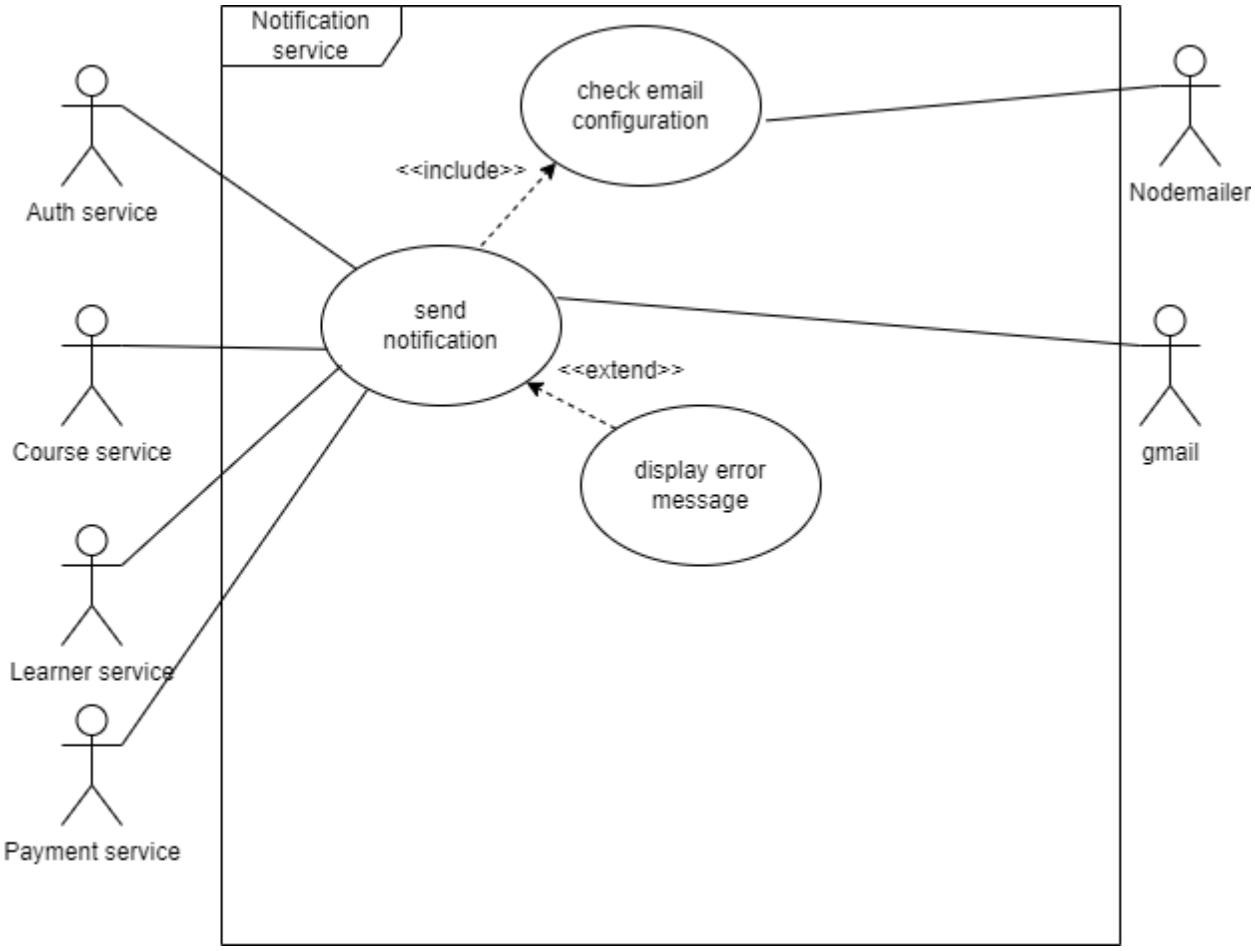
Use case diagram for Payment service



Notification service

This microservice, utilizing Node.js and mongoDB facilitates send notification via email. Nodemailer is used for send email. Used Gmail as a service provider. Whenever the service is requested it check the email configuration and send the response

Use case diagram for Notification service

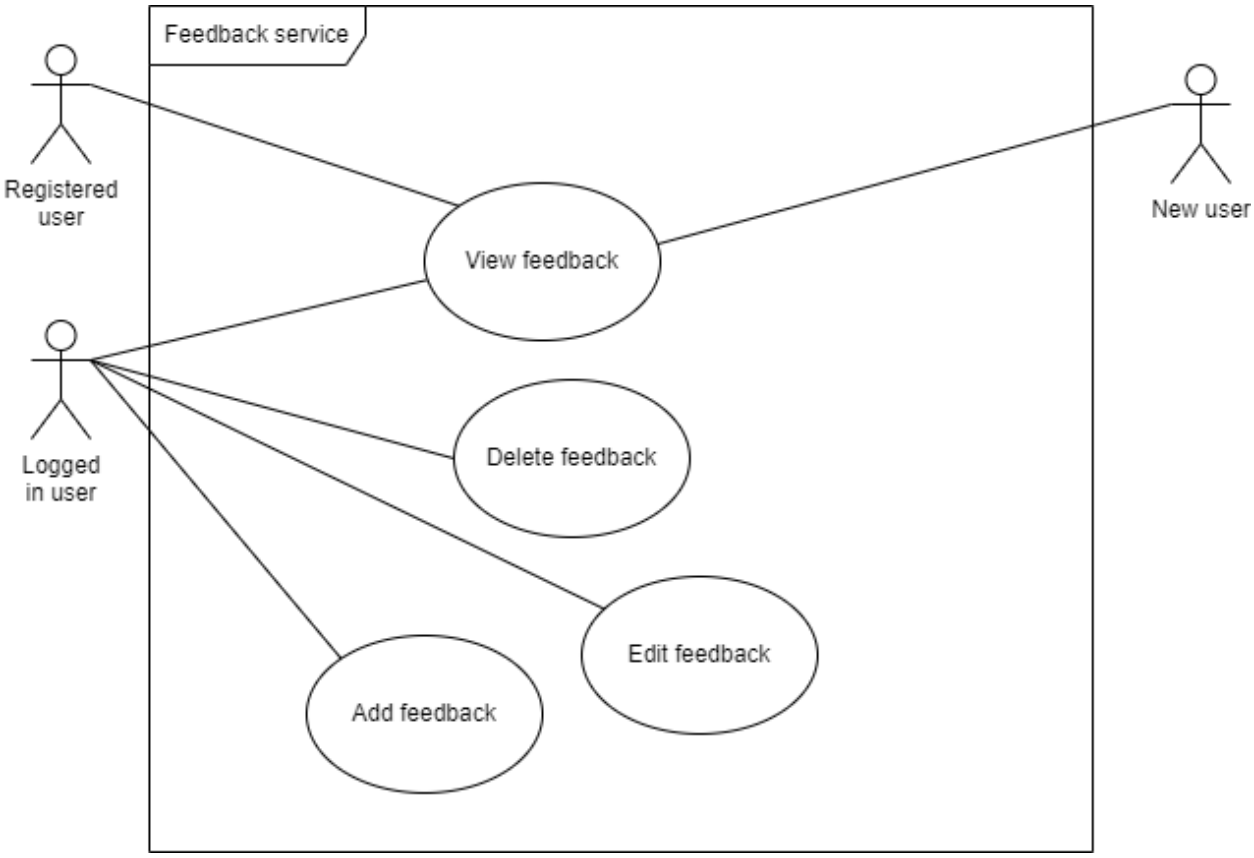


Feedback service

The Feedback Management Microservice, powered by MongoDB, Node.js, Express.js, and React.js, revolutionizes how newcomers and learners engage with course feedback and ratings.Seamlessly integrated with the user authentication microservice, it ensures secure access

for users to manage their feedback effectively. Leveraging the course management service,learners can effortlessly access course details to provide informed feedback. With intuitive interfaces, learners can not only view existing feedback but also contribute their insights and ratings, fostering a collaborative learning environment. Empowering learners to manage their own feedback, this microservice enhances transparency and enriches the learning experience while prioritizing security and seamless integration.

Use case diagram for Feedback service



Appendix

Appendix – Authentication service

IT21256264 – Backend

1. Entity files

- Appendix 01: User.java

```
package com.studyStorm.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String firstName;
    private String lastName;
    private String email;
    private String phoneNumber;
    private String password;
    private String roles;
}
```

- Appendix 01: ForgotPasswod.java

```
package com.studyStorm.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;

import java.util.Date;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Builder
public class ForgotPassword {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer fpid;
```

```
    @Column(nullable = false)
    private Integer otp;

    @Column(nullable = false)
    private Date expirationTime;

    @OneToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;
}
```

2. Data transfer object (DTO) files

- Appendix 02 : AddUserRequest.java

```
package com.studyStorm.dto;

public record AddUserRequest(String firstName,
                             String lastName,
                             String email,
                             String phoneNumber,
                             String password,
                             String confirmPassword,
                             String roles) {
}
```

- Appendix 02 : AuthRequest.java

```
package com.studyStorm.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class AuthRequest {

    private String username ;
    private String password;
}
```

- Appendix 02 : ChangePassword.java

```
package com.studyStorm.dto;

public record ChangePassword(String password, String
confirmPassword) {
}
```

- Appendix 02 : ChangePasswordRequest.java

```
package com.studyStorm.dto;

public record ChangePasswordRequest(String currentPassword,
                                    String newPassword,
                                    String confirmPassword) {
}
```

❖ Appendix 02 : JwtResponse.java

```
package com.studyStorm.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class JwtResponse {
    private String accessToken;
    private String token;
}
```

❖ Appendix 02 : MailBody.java

```
package com.studyStorm.dto;

import lombok.Builder;

@Builder
public record MailBody(String to, String subject, String text) {

}
```

❖ Appendix 02 : UpdateUserRequest.java

```
package com.studyStorm.dto;

public record UpdateUserRequest(String firstName,
                                String lastName,
                                String email,
                                String phoneNumber
                                ) {
}
```

3. Controllers files

❖ Appendix 03 : UserController.java

```
package com.studyStorm.controller;

import com.studyStorm.dto.*;
```

```java
import com.studyStorm.entity.RefreshToken;
import com.studyStorm.entity.User;
import com.studyStorm.service.JwtService;
import com.studyStorm.service.RefreshTokenService;
import com.studyStorm.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenti
cationToken;
import org.springframework.security.core.Authentication;
import
org.springframework.security.core.userdetails.UsernameNotFoundExcept
ion;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin
@RequestMapping("/v1")
public class UserController {

    @Autowired
    private UserService service;
    @Autowired
    private JwtService jwtService;

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private RefreshTokenService refreshTokenService;

    @GetMapping("/home")
    public String welcome() {
        return "Welcome this endpoint is not secure";
    }

//    get all users
//    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    @GetMapping("/users/all")
    public Iterable<User> getAllUsers(){
        return service.getAllUsers();
    }


    @PostMapping("/new")
    public String addNewUser(@RequestBody AddUserRequest userInfo){
        return service.addUser(userInfo);
    }

//    update user by email
    @PatchMapping("/user/{email}")
    public String updateUser(@PathVariable String email,
@RequestBody UpdateUserRequest userInfo){
        return service.updateUser(email, userInfo);
```

```
    }

//    change password by email
    @PatchMapping("/user/userProfile/updatePassword/{email}")
    public String changePassword(@PathVariable String email,
@RequestBody ChangePasswordRequest changePasswordRequest){
        return service.changePassword(email, changePasswordRequest);
    }

//    delete user by id
    @DeleteMapping("/user/{id}")
    public String deleteUser(@PathVariable Integer id){
        return service.deleteUser(id);
    }



    @PostMapping("/auth/admin/users")
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public String addNewUserByAdmin(@RequestBody AddUserRequest
userInfo){
        return service.addUser(userInfo);
    }

    @PostMapping("/login")
    public JwtResponse authenticateAndGetToken(@RequestBody
AuthRequest authRequest) {
        User user = service.findByEmail(authRequest.getUsername());

        Authentication authentication =
authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(authRequest.getUsername()
                , authRequest.getPassword()));
        if (authentication.isAuthenticated()) {
            RefreshToken refreshToken =
refreshTokenService.createRefreshToken(authRequest.getUsername());
            jwtService.generateToken(authRequest.getUsername(),
user.getRoles());
            return JwtResponse.builder()

.accessToken(jwtService.generateToken(authRequest.getUsername(),
user.getRoles()))
                    .token(refreshToken.getToken())
                    .build();
        } else {
            throw new UsernameNotFoundException("invalid user
request !");
        }

    }

//    get user details from token
    @GetMapping("/user")
    public User getUserDetails(@RequestHeader("Authorization")
String token) {
        return
service.findByEmail(jwtService.getEmailFromToken(token));
```

```
        }

        //    get user details from email
        @GetMapping("/user/{email}")
        // @PreAuthorize("hasAuthority('ROLE_INSTRUCTOR')")
        public User getUserDetailsByEmail(@PathVariable String email) {
            return service.findByEmail(email);
        }



        @PostMapping("/refreshToken")
        public JwtResponse refreshToken(@RequestBody RefreshTokenRequest
    refreshTokenRequest) {
            return
    refreshTokenService.findByToken(refreshTokenRequest.getToken())
                    .map(refreshTokenService::verifyExpiration)
                    .map(RefreshToken::getUser)
                    .map(user -> {
                        String accessToken =
    jwtService.generateToken(user.getEmail(), user.getRoles());
                        return JwtResponse.builder()
                                .accessToken(accessToken)

    .token(refreshTokenRequest.getToken())
                                .build();
                    }).orElseThrow(() -> new RuntimeException("Invalid
    refresh Token"));
        }


    }
```

❖ Appendix 03 : ForgotPasswordController.java

```
❖  package com.studyStorm.controller;

    import com.studyStorm.dto.ChangePassword;
    import com.studyStorm.dto.MailBody;
    import com.studyStorm.entity.ForgotPassword;
    import com.studyStorm.entity.User;
    import com.studyStorm.repository.ForgotPasswordRepository;
    import com.studyStorm.repository.UserRepository;
    import com.studyStorm.service.EmailService;
    import org.springframework.http.HttpStatus;
    import org.springframework.http.ResponseEntity;
    import org.springframework.security.crypto.password.PasswordEncoder;
    import org.springframework.web.bind.annotation.*;

    import java.time.Instant;
    import java.util.Date;
    import java.util.Objects;
```

```java
import java.util.Random;

@RestController
@CrossOrigin
@RequestMapping("/forgotPassword")
public class ForgotPasswordController {

    private final UserRepository userRepository;
    private final EmailService emailService;
    private final ForgotPasswordRepository forgotPasswordRepository;
    private final PasswordEncoder passwordEncoder;

    public ForgotPasswordController(UserRepository userRepository,
EmailService emailService, ForgotPasswordRepository
forgotPasswordRepository, PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.emailService = emailService;
        this.forgotPasswordRepository = forgotPasswordRepository;

        this.passwordEncoder = passwordEncoder;
    }

//    send mail for email verification
    @PostMapping("/verifyMail/{email}")
    public ResponseEntity<String> verifyMail(@PathVariable String
email) {
        User user = userRepository.findByEmail(email).orElseThrow(()
-> new RuntimeException("User not found"));

        int otp = otpGenerator();
        MailBody mailBody = MailBody.builder()
                .to(email)
                .text("This is the OTP for the request :" + otp)
                .subject("OTP for forgot password")
                .build();

        ForgotPassword forgotPassword = ForgotPassword.builder()
                .otp(otp)
                .expirationTime(new
Date(System.currentTimeMillis()+70*1000)) // 70 seconds
                .user(user)
                .build();

        emailService.sendSimpleMessage(mailBody);
        forgotPasswordRepository.save(forgotPassword);
        return ResponseEntity.ok("Mail sent successfully");

    }

    @PostMapping("/verifyOtp/{otp}/{email}")
    public ResponseEntity<String> verifyOtp(@PathVariable Integer
otp, @PathVariable String email) {
       User user = userRepository.findByEmail(email)
               .orElseThrow(() -> new RuntimeException("Please enter
valid email"));

         ForgotPassword forgotPassword =
```

```
    forgotPasswordRepository.findByOptAndUser(otp, user)
                    .orElseThrow(() -> new RuntimeException("Please
    enter valid otp"));


    if(forgotPassword.getExpirationTime().before(Date.from(Instant.now()
    ))) {

    forgotPasswordRepository.deleteById(forgotPassword.getFpid());
                    return new ResponseEntity<>("OTP expired",
    HttpStatus.EXPECTATION_FAILED);
                }

                return ResponseEntity.ok("OTP verified successfully");

        }

        @PostMapping("/changePassword/{email}")
        public ResponseEntity<String> changePassword(@RequestBody
    ChangePassword changePassword,
                                                     @PathVariable
    String email) {
            if (!Objects.equals(changePassword.password(),
    changePassword.confirmPassword())) {
                return new ResponseEntity<>("Password and confirm
    password should be same", HttpStatus.BAD_REQUEST);
            }

            String encodedPassword =
    passwordEncoder.encode(changePassword.password());
            userRepository.updatePassword(email, encodedPassword);
            return ResponseEntity.ok("Password changed successfully");

        }


        private Integer otpGenerator() {
            Random random = new Random();
            return random.nextInt(100_000, 999_999);
        }

    }
```

Course Managemenr Service

```
// @desc    Register a new Course
// route    POST /api/course/create
// @access  Private - Auth Lvl 3
```

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

        const {
            name,
            desc,
            subject,
            language,
            type,
            level,
            duration,
            skills,
            start_date,
            price,
        } = req.body;

        const thumbnail = req.file?.path;

        try {
            if(!name){
                throw new Error('Course Name is required');
            } else if(!desc){
                throw new Error('Course Description is required');
            } else if(!subject){
                throw new Error('Course Subject is required');
            } else if(!start_date){
                throw new Error('Course Start Date is required');
            } else if(!duration){
                throw new Error('Course Duration is required');
            }
        } catch (error) {
            if (req.file) {
                fs.unlinkSync(req.file.path);
            }
            return res.status(400).json({ message: error.message });
        }

        var course = await Courses.findOne({ where: { name } });

        if(course){
            if (req.file) {
                fs.unlinkSync(req.file.path);
            }
            return res.status(400).json({ message: 'Course Already Exists' });
```

```
        }

        course = await Courses.create({
            name,
            desc,
            subject,
            language,
            type,
            level,
            duration,
            skills,
            start_date,
            price,
            thumbnail,
            approved: null,
            created_by: req.email
        });

        if(course){
            return res.status(201).json({ message: 'Course Created Successfully',
payload: course });

        }else{
            if (req.file) {
                fs.unlinkSync(req.file.path);
            }
            return res.status(400).json({ message: 'Course Creation Unsuccessful'
});
        }

    } catch (error) {
        if (req.file) {
            fs.unlinkSync(req.file.path);
        }
        return res.status(500).json({ message: error.message })
    }

});


// @desc    Retrieve All Courses
// route    GET /api/course/all
```

```
// @access  Public
const getCourseList = asyncHandler(async (req, res) => {

    try {

        const page = parseInt(req.query.page)-1;
        const rows = parseInt(req.query.rows);
        let approved = req.query.approved || 1

        if( approved == 'null'){
            approved = null
        }

        let courses = await Courses.findAndCountAll({ where:{approved}, offset:
page, limit: rows})

        if(courses.count <= 0){
            return res.status(404).json({ message: 'No Courses Available!' });

        }

        return res.status(200).json({ message: 'Courses Retreived Successfully',
payload: courses });

    } catch (error) {

        return res.status(500).json({ message: error.message })

    }

});



// @desc    Retrieve All Courses By Instructor
// route    GET /api/course/instructor/all
// @access  Public
const getCourseListByInstructor = asyncHandler(async (req, res) => {

    try {

        const page = parseInt(req.query.page)-1;
        const rows = parseInt(req.query.rows);
        let approved = req.query.approved || true
```

```
        const email = req.email

        if( approved == 'null'){
            approved = null
        }


        let   courses   =   await   Courses.findAndCountAll({   where:{approved,
created_by:email}, offset: page, limit: rows})

        if(courses.count <= 0){
            return res.status(404).json({ message: 'No Courses Available!' });

        }

        return res.status(200).json({ message: 'Courses Retreived Successfully',
payload: courses });

    } catch (error) {

        return res.status(500).json({ message: error.message })

    }

});


// @desc    Retrieve Courses By Id
// route    GET /api/course/one/:id
// @access  Public
const getCourseById = asyncHandler(async (req, res) => {

    try {

        const id = parseInt(req.params.id);

        if(isNaN(id)){
            return res.status(400).json({ message: 'Invalid Course Id!' });

        }

        let course = await Courses.findByPk(id)
```

```
        if(!course){
            return res.status(404).json({ message: 'Course Not Found!' });
        }

        course.skills = course.skills.split(',');

        return res.status(200).json({ message: 'Course Retreived Successfully',
payload: course });

    } catch (error) {

        return res.status(500).json({ message: error.message })


    }

});


// @desc    Update Course Details
// route    PUT /api/course/update
// @access  Private - Auth Lvl 3
const updateCourse = asyncHandler(async (req, res) => {

    try {

        const {
            course_id,
            name,
            desc,
            subject,
            language,
            type,
            level,
            duration,
            skills,
            start_date,
            price,
        } = req.body;

        const thumbnail = req.file?.path;

        let course = await Courses.findByPk(course_id)
```

```
        if (!course) {
            if (req.file) {
                fs.unlinkSync(req.file.path);
            }
            return res.status(404).json({ message: 'Course Not Found!' })
        }

        let oldThumbnail = course.thumbnail;

        course = await Courses.update(
            {
                name: name || course.name,
                desc: desc || course.desc,
                subject: subject || course.subject,
                language: language || course.language,
                type: type || course.type,
                skills: skills || null,
                level: level || course.level,
                start_date: start_date || course.start_date,
                duration: duration || course.duration,
                price: price || course.price,
                thumbnail: thumbnail || null,
                approved: null
            },
            {
              where: {
                course_id: course_id,
              },
            },
        );

        if (oldThumbnail) {
            fs.unlinkSync(oldThumbnail);
        }
        return res.status(200).json({ message: 'Course Updated Successfully' });

    } catch (error) {
        if (req.file) {
            fs.unlinkSync(req.file.path);
        }
        return res.status(500).json({ message: error.message })
    }
});
```

```
// @desc    Approve Course
// route    PATCH /api/course/approve/:id/:approve
// @access  Private - Auth Lvl 2
const approveCourse = asyncHandler(async (req, res) => {

    try {

        const id = parseInt(req.params.id);
        const approve = req.params.approve;
        console.log(approve);

        if(isNaN(id)){
            return res.status(400).json({ message: 'Invalid Course Id!' });
        }

        let course = await Courses.findByPk(id)

        if (!course) {
            return res.status(404).json({ message: 'Course Not Found!' })
        }
        let courseDetails = course;

        course = await Courses.update(
            {
                approved: approve
            },
            {
              where: {
                course_id: id,
              },
            },
        );

        let errMail = `Dear Instructor,<br />The Course You created titled ${courseDetails.name} has been rejected! Please re-check and verify the content and re submit for approval.`
        let succMail = `Dear Instructor,<br />The Course You created titled ${courseDetails.name} has been approved!`

        let mailData = {
            "notifications": [
```

```
                {
                    "email": courseDetails.created_by,
                    "message": approve ? succMail : errMail
                },
            ]
        }
        sendMail(mailData)


        return res.status(200).json({ message: `Course ${approve ? 'Rejected' :
'Approved'} Successfully` });

    } catch (error) {
        return res.status(500).json({ message: error.message })
    }
});


// @desc    Delete Course
// route    DELETE /api/course/delete/:id
// @access  Private - Auth Lvl 3
const deleteCourse = asyncHandler(async (req, res) => {

    try {

        const id = parseInt(req.params.id);

        let course = await Courses.findByPk(id);
        if(!course){
            return res.status(404).json({ message: "Course Not Found!" })
        }

        let thumbnail = course.thumbnail

        course = await Courses.destroy({
            where: {
                course_id: id,
            },
        });


        if(thumbnail){
            fs.unlinkSync(thumbnail);
```

```
        }
        return res.status(200).json({ message: 'Course Deleted Successfully' });


    } catch (error) {

        return res.status(500).json({ message: error.message })


    }
});
```