

E-LEARNING PLATFORM

A PROJECT REPORT

Submitted By

Kajal Punia

(University Roll No- 2000290140055)

Niharika Baliyan

(University Roll No- 2000290140058)

Aparna Singh

(University Roll No- 2000290140025)

**Submitted in partial fulfillment of the
Requirements for the Degree of**

MASTER OF COMPUTER APPLICATIONS

**Under the Supervision of
Dr. Arun Kr. Tripathi
Professor**



Submitted to

**DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206**

(JAN 2022)

CERTIFICATE

Certified that **Kajal Punia (Enrolment No-200029014005740), Niharika Baliyan (Enrolment No-200029014005743), Aparna Singh(Enrolment No-200029014005710)** have carried out the project work having “**Title of Report ELP**” for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself /herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date: 13/01/2022

Kajal Punia (University Roll No- 2000290140055)

Niharika Baliyan (University Roll No- 2000290140058)

Aparna Singh(University Roll No-2000290140025)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

13/01/2022

Dr. Arun Kr. Tripathi
Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Signature of Internal Examiner

Signature of Internal Examiner

Dr. Ajay Shrivastava
Head, Department of Computer Applications
KIET Group of Institutions, Ghaziabad

ABSTRACT

E-learning fulfills the thirst of knowledge and offers online content that can be delivered for the learner at anywhere and anytime through a wide range of e-learning solutions while compared with traditional learning systems. It also provides rapid access to specific knowledge and information. With the rapid growth of voluminous information sources and the time constraint the learning methodology has changed. Learners obtain knowledge through e-Learning systems rather than manually teaching and learning.

Through this project we intend to provide an e-learning platform where instructors can create courses and students can enroll themselves in various courses and learn without any boundations. ELP aims to provide education anytime, anywhere and for everyone. It provides added functionality to choose between text or video content.

ELP is designed while keeping in mind the main requirement of the college students-MOOC courses. Now, instead of looking for courses here and there, students can just enroll themselves in the courses designed by their own trusted faculty members, that too free of cost.

ACKNOWLEDGEMENTS

Success in life is never attained single handedly. My deepest gratitude goes to my thesis supervisor, **Dr. Arun Kr. Tripathi** for his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Ajay Kumar Shrivastava, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot in many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kinds of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Kajal Punia

Niharika Baliyan

Aparna Singh

TABLE OF CONTENTS

Certificate	i
Abstract	ii
Acknowledgements	iii
Table of Contents	iv
List of Abbreviations	7
List of Tables	7
List of Figures	7
	8-9
1 Introduction	1-16
1.1 Description	8
1.2 Project Scope	8
1.3 Project Definitions	9
1.4 Project Features	9
2 Feasibility Study	9-10
1.1 Technical Feasibility	9
1.2 Economical Feasibility	10
1.2 Behavioral Feasibility Study	10
3 Database Used	10
4 UML Diagrams	11-21
4.1 Flowchart	11-12
4.2 Use Case Diagram	13-14
4.3 Communication Diagram	15-16
4.4 Activity Diagram	17-18
4.5 Sequence Diagram	19-20
4.6 ER Diagram	20-21
5 System Features (Functional Requirements)	22-23
6 Non Functional Requirements	24
7 User Interface(Screenshots)	24-29

8	Coding	29-32
9	Testing	32-33

List of Abbreviations

Abbreviation	Full form
ELP	e-Learning Platform
e-learning	Electronic Learning
UC	Use Case
Approx.	Approximation

List of Tables

Table No.	Description
1	Test Case Suite

LIST OF FIGURES

Figure Number	Figure Name	Page Number
Fig 3.1.	Flow Chart	
Fig 3.2.	Use Case Diagram	
Fig 3.3.	Communication Diagram	

CHAPTER-1

INTRODUCTION

1. Overview

The ELP has been developed to overcome the challenges of manual learning and teaching system also to promote independent learning. This platform is to reduce hardships faced by existing systems and is designed for the particular need of a college to use the functionalities or operations in a smooth manner.

This platform is easy to use for instructors and students i.e. no formal knowledge is needed to use the platform thus by this all it proves it is user-friendly. This platform reduces errors while entering the data by providing error messages as a user enters invalid data. ELP can lead to secure, fast and error free system. It will help a college to manage or utilize resources and time. This platform allows students to study at their own place. Students can view the course from this platform and instructor can upload and update the course.

This ELP is a space or portal filled with educational content for those courses which are offered by the affiliated university.

1.1 Project description

ELP lets users register with valid email and login with the registered email id. There are two types of users here i.e., Student and Instructor. By default all the users are students, the role of instructor is given at the backend manually. Students can view the courses, enroll in them and then study. Instructors can create courses, add lessons, edit content, view the number of enrolled students and enroll in different courses. ELP transcends the traditional classroom learning to e-learning and makes it available to the students anywhere anytime. With the help of ELP, the MOOC courses will be simplified as well, now students can just enroll in the courses created by their own faculty members and gain extra knowledge. It will also facilitate the faculty members to wrap up their study materials in one place and make it available to the students.

1.2 Project Scope

The scope of the system is to have a high-tech environment. That means by using the e-learning platform, the users can learn new subjects by enrolling into the various courses. This system will add some features in their learning environment that can make their study interesting. That will help the community use the technology in effective ways:

- a. Can have all the courses at one place.
- b. Self paced study.
- c. Manage and organize the course content easily.

1.3 Project Definition

User:

- i. User is supposed to login using its ID and Password. This will allow only a single user to login from a particular device.
- ii. When a user forgets his/her password then he/she can generate a code on his/her email using it the new password can be generated.
- iii. A user can enroll into different courses and study .
- iv. A user can logout from the website whenever done with the study.

Instructor:

- i. The Instructor can create the course, edit the course and publish the course.
- ii. The Instructor can see the number of users enrolled in a particular course.

Admin:

- i. Admin is the main developer who can give a user the role of instructor.

1.4 Hardware and Software used in the project

- i. VsCode
- ii. mongoDB
- iii. Ram 8GB
- iv. i5 Processor

1.5 Project Features

- i. It deals with the virtual learning system which helps the users to study a course at their respective places without going outside. The user has to enroll into the course using their email Id. If a user has forgotten their password, they can generate their new password using a code which is sent to their email.
- ii. It deals with the different courses where a user can view the study material. A user can register them into multiple courses.
- iii. It deals with the instructor that can create a course, edit the course and publish it. Instructor can view the users that are enrolled into the different courses and take a look over the users and courses .
- iv. It deals with the Admin which is the main developer who can give a user the role of instructor.

CHAPTER-2

FEASIBILITY STUDY

2. Feasibility Study

A feasibility analysis is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. It tells us whether a project is worth the investment—in some cases, a project may not be doable. There can be many reasons for this, including requiring too many resources, which not only prevents those resources from performing other tasks but also may cost more than an organization would earn back by taking on a project that isn't profitable.

A well-designed study should offer a historical background of the business or project, such as a description of the product or service, accounting statements, details of operations and management, marketing research and policies, financial data, legal requirements, and tax obligations. Generally, such studies precede technical development and project implementation.

2.1 Technical feasibility

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the technical team is capable of converting the ideas into working systems. Technical feasibility also involves the evaluation of the hardware, software, and other technical requirements of the proposed system.

The organization is able to purchase storage web services and the technology that is being used in this project can be maintained by different developers present inside the organization therefore, this project is technically feasible.

2.2 Behavioral Feasibility

It evaluates and estimates the users' attitude or behavior towards the development of new system. It helps in determining if the system requires special effort to educate, retrain, transfer and how well the users will be able to adapt to it. Since the students are

already aware of the concept of online education and the faculty members are aware of online platforms like Moodle, this project is operationally feasible.

2.3 Economical Feasibility

In the Economic Feasibility study, the cost and benefit of the project are analyzed. Which means, under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. After that it is analyzed whether project will be beneficial in terms of finance for organization or not.

CHAPTER-3

DATABASE

3. Database Used

A properly designed database provides you with access to up-to-date, accurate information. Because a correct design is essential to achieving your goals in working with a database, investing the time required to learn the principles of good design makes sense. In the end, you are much more likely to end up with a database that meets your needs and can easily accommodate change.

The database used here is MongoDB 5.0. MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and functions which is the equivalent of relational database tables.

For further storage purposes, Amazon's S3 services are being used to store the video lectures and photos.

MongoDB is built on a scale-out architecture that has become popular with developers of all kinds for developing scalable applications with evolving data schemas.

As a document database, MongoDB makes it easy for developers to store structured or unstructured data. It uses a JSON-like format to store documents. This format directly maps to native objects in most modern programming languages, making it a natural choice for developers, as they don't need to think about normalizing data. MongoDB can also handle high volume and can scale both vertically or horizontally to accommodate large data loads.

MongoDB was built for people building internet and business applications who need to evolve quickly and scale elegantly. Companies and development teams of all sizes use MongoDB for a wide variety of reasons.

CHAPTER-4

UML DIAGRAMS

4.1 Flow Chart

A flowchart is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. Flowcharts use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence.

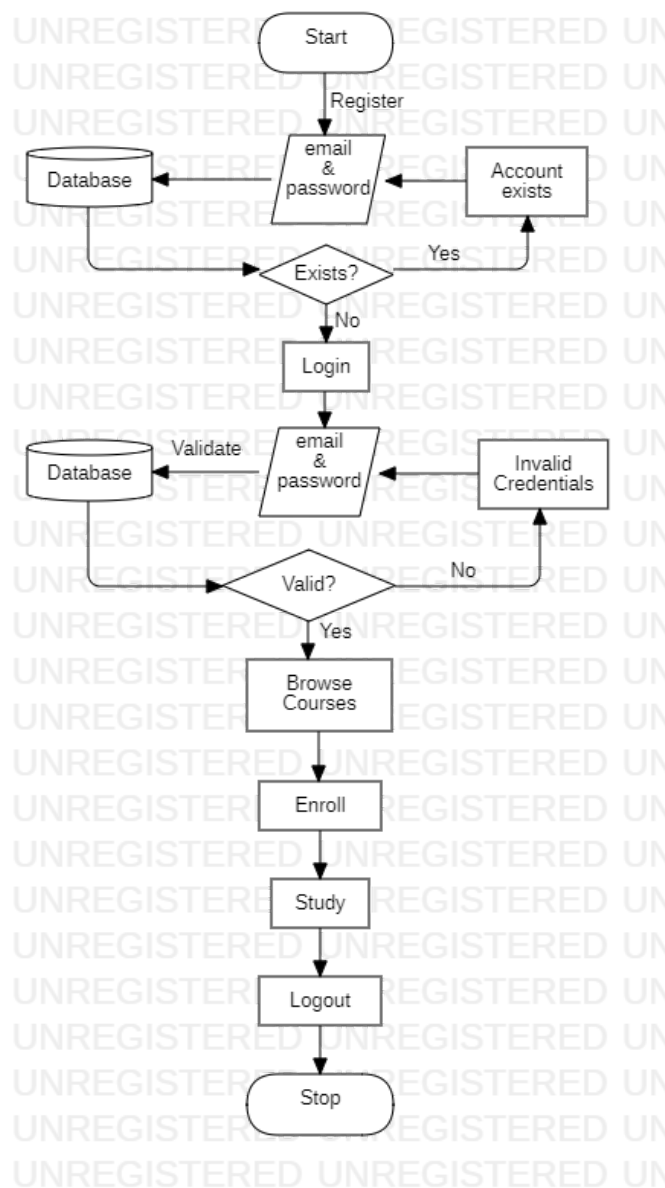


Fig 4.1 Flow Chart

4.2 Use Case Diagram

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

The purpose of a use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and State chart) also have the same purpose. We will look into some specific purpose, which will distinguish it from the other four diagrams.

When the initial task is complete, use case diagrams are modeled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows –

- i. Used to gather the requirements of a system.
- ii. Used to get an outside view of a system.
- iii. Identify the external and internal factors influencing the system.
- iv. Show the interaction among the requirements are actors.

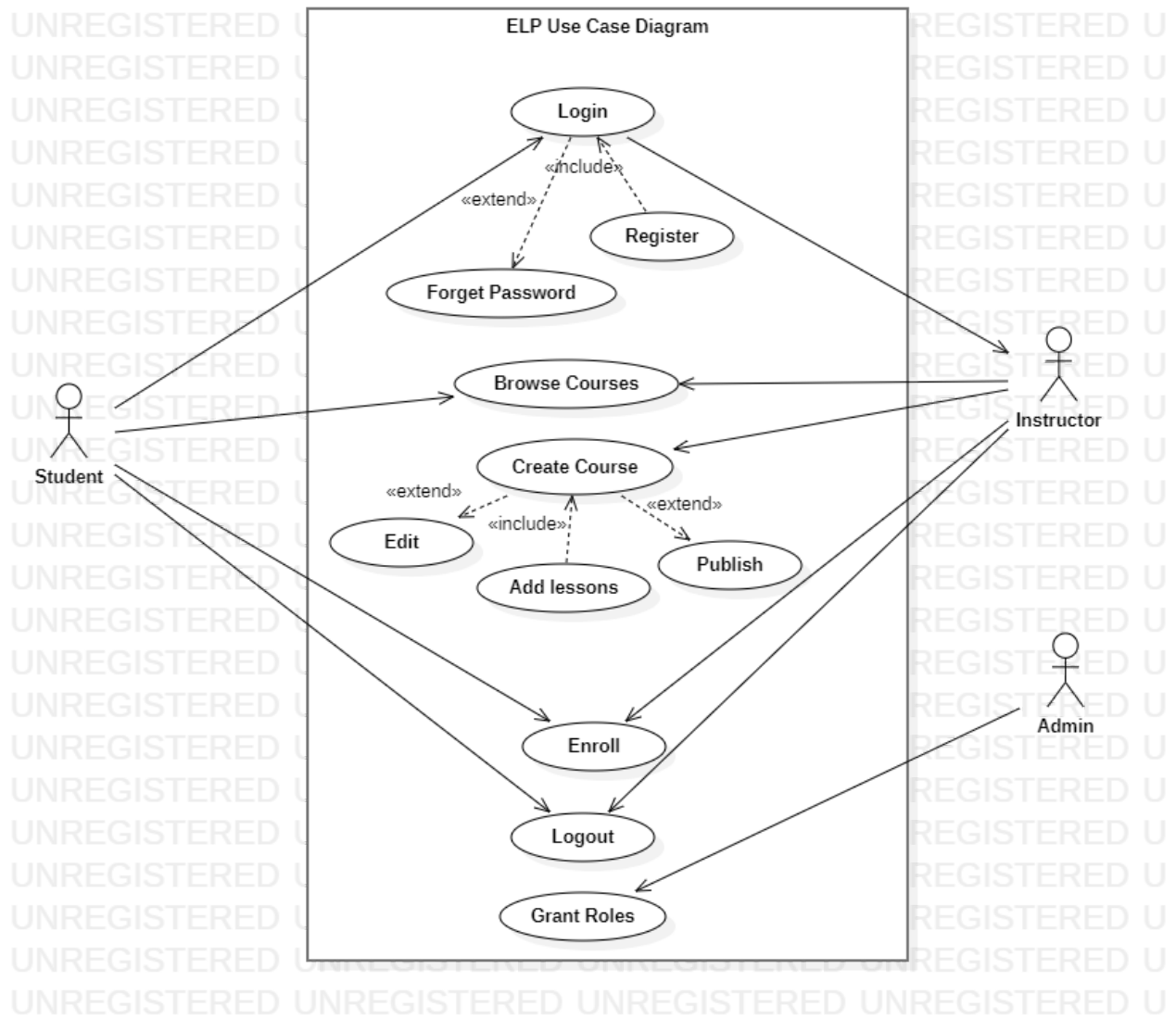


Fig 4.2 Use Case Diagram

4.3 Communication Diagram

This communication diagram shows the interactions between the objects or roles associated with lifelines and the messages that pass between lifelines. Communication diagram for ELP is shown in the figure below.

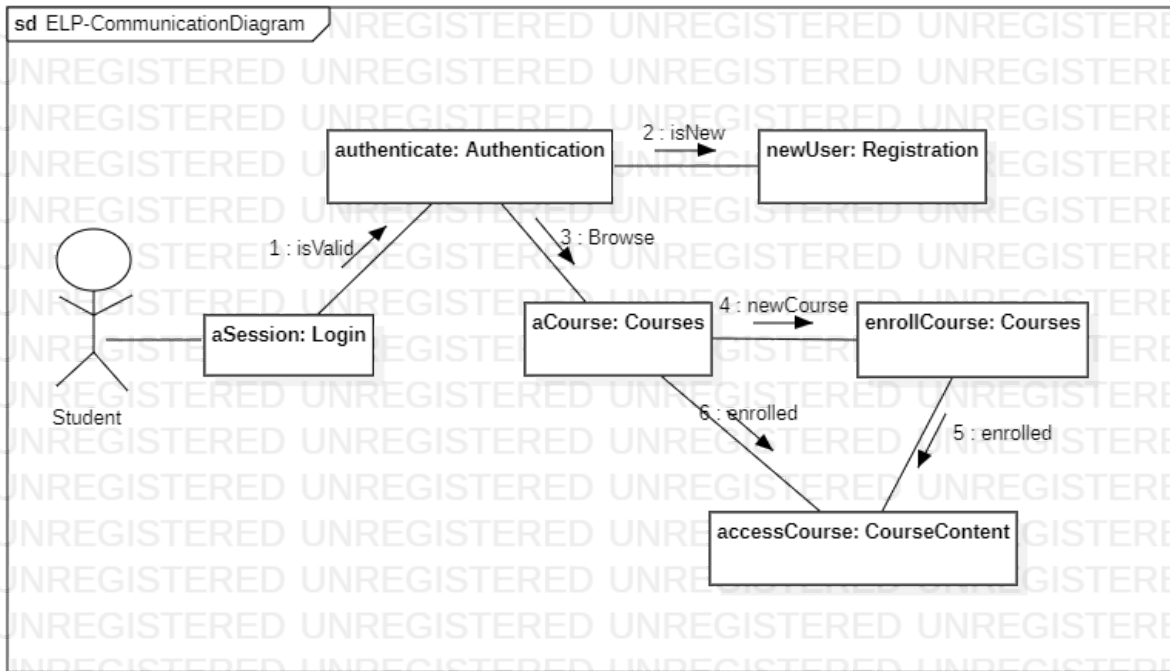


Fig 4.3 Communication Diagram

4.4 Activity Diagram

The activity diagram is used to demonstrate the flow of control within the system rather than the implementation. It models the concurrent and sequential activities.

The activity diagram helps in envisioning the workflow from one activity to another. It puts emphasis on the condition of flow and the order in which it occurs. The flow can be sequential, branched, or concurrent, and to deal with such kinds of flows, the activity diagram has come up with a fork, join, etc.

It is also termed as an object-oriented flowchart. It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram.

Why Activity Diagram?

An event is created as an activity diagram encompassing a group of nodes associated with edges. To model the behavior of activities, they can be attached to any modeling element. It can model use cases, classes, interfaces, components, and collaborations.

It mainly models processes and workflows. It envisions the dynamic behavior of the system as well as constructs a runnable system that incorporates forward and reverse engineering. It is the same as that of a flowchart but not exactly a flowchart itself. It is used to depict the flow between several activities.

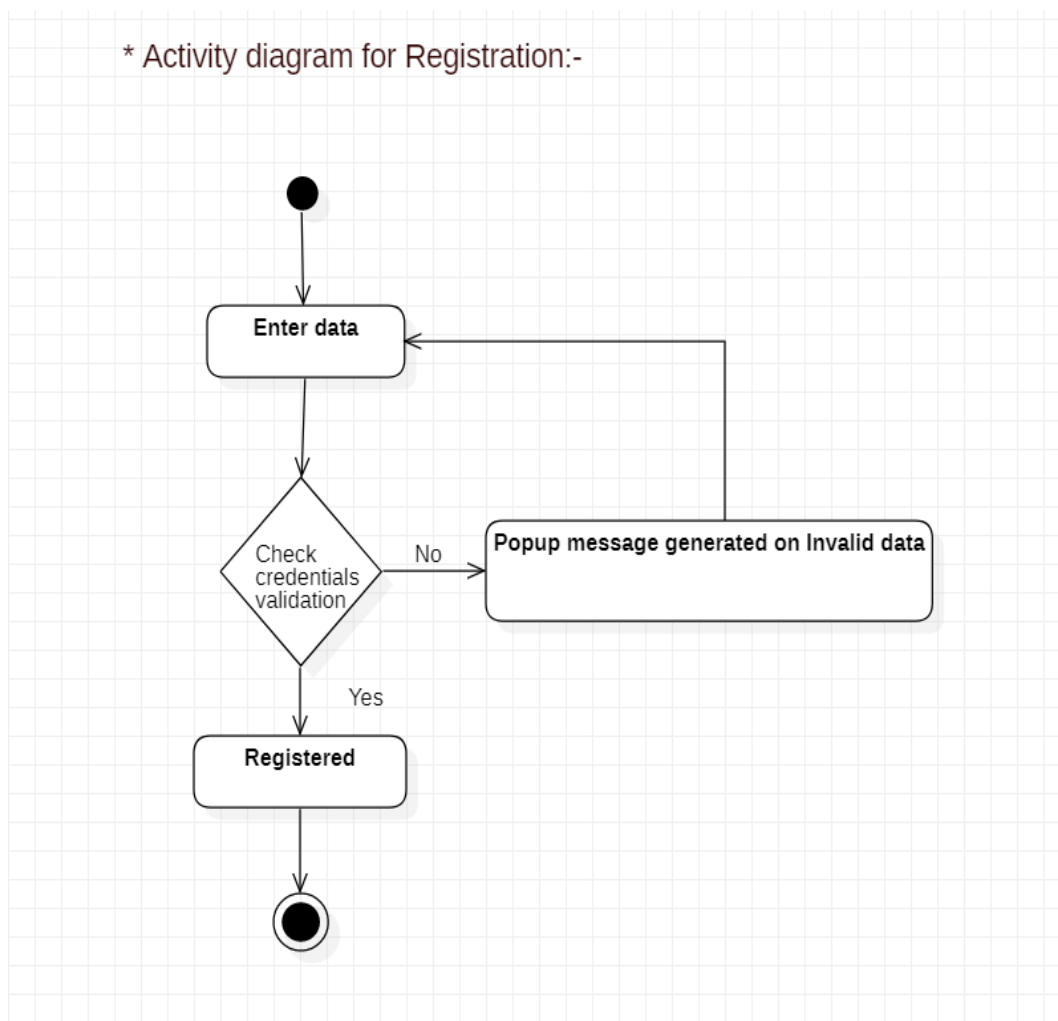


Fig 4.4.1 Activity Diagram for Registration

* Activity diagram for Login:-

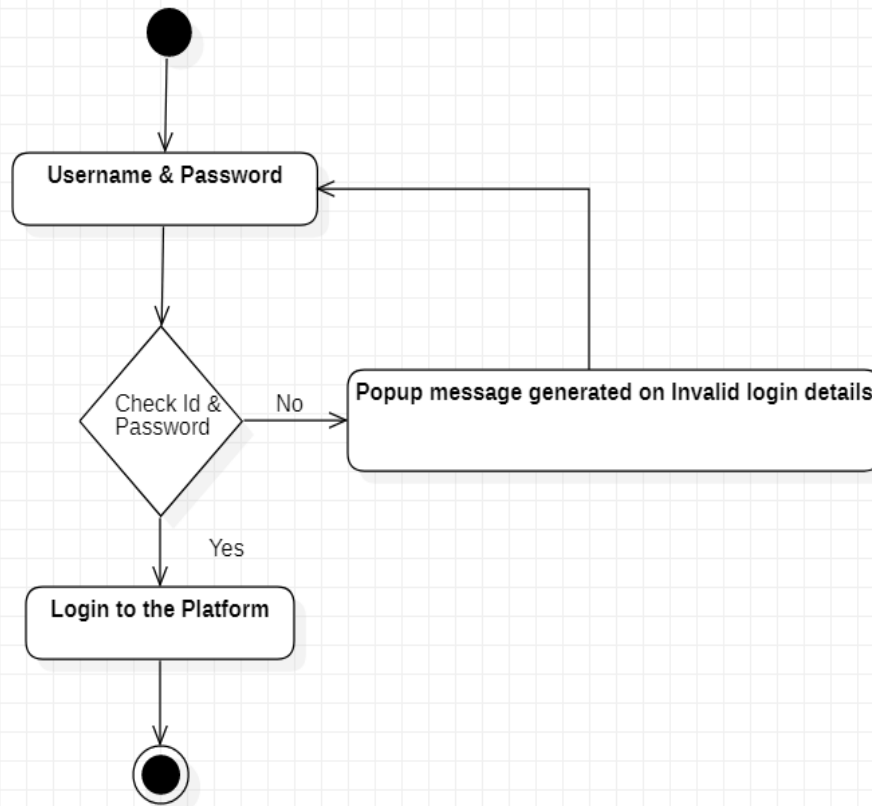


Fig 4.4.2 Activity Diagram for Login

4.5 Sequence Diagram

It represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time.

The Sequence diagram for the logout module is depicted in figure 4.5

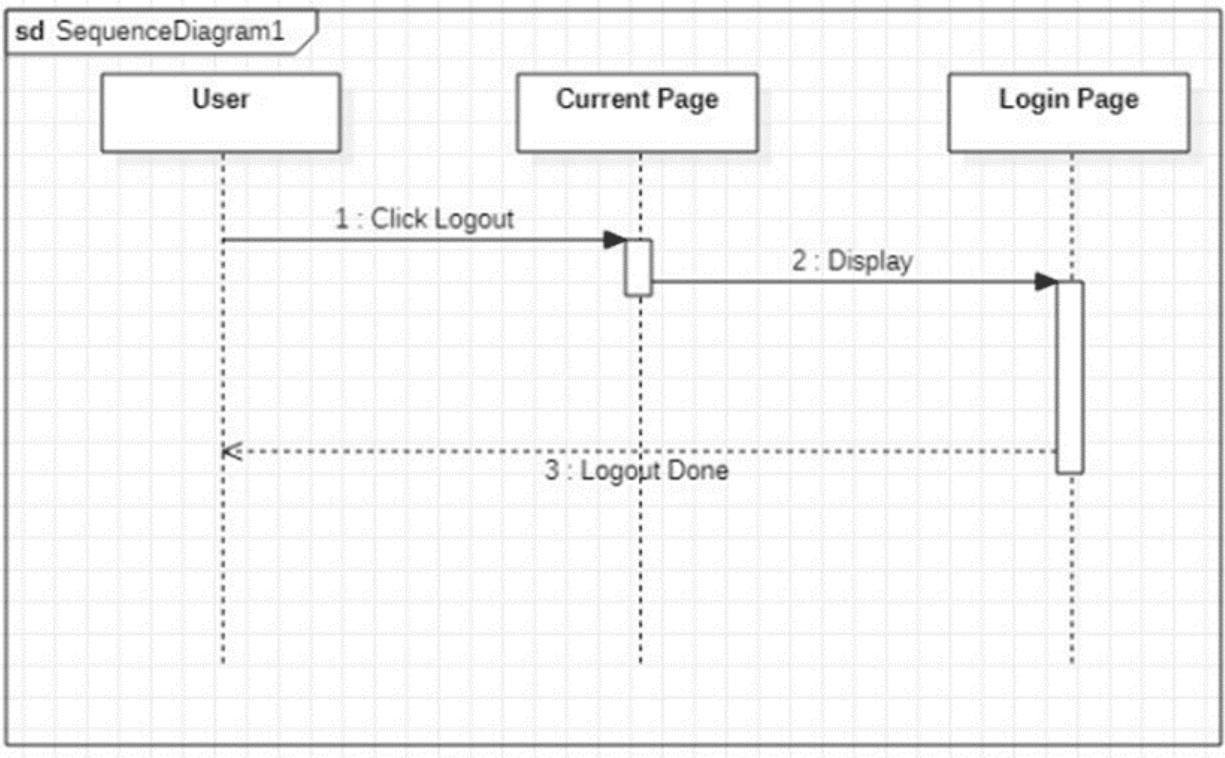


Figure 3.5 Sequence Diagram for Logout Module

4.6 E-R Diagram

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs. ER diagrams are related to data structure diagrams (DSDs), which focus on the relationships of elements within entities instead of relationships between entities themselves. ER diagrams also are often used in conjunction with data flow diagrams (DFDs), which map out the flow of information for processes or systems.

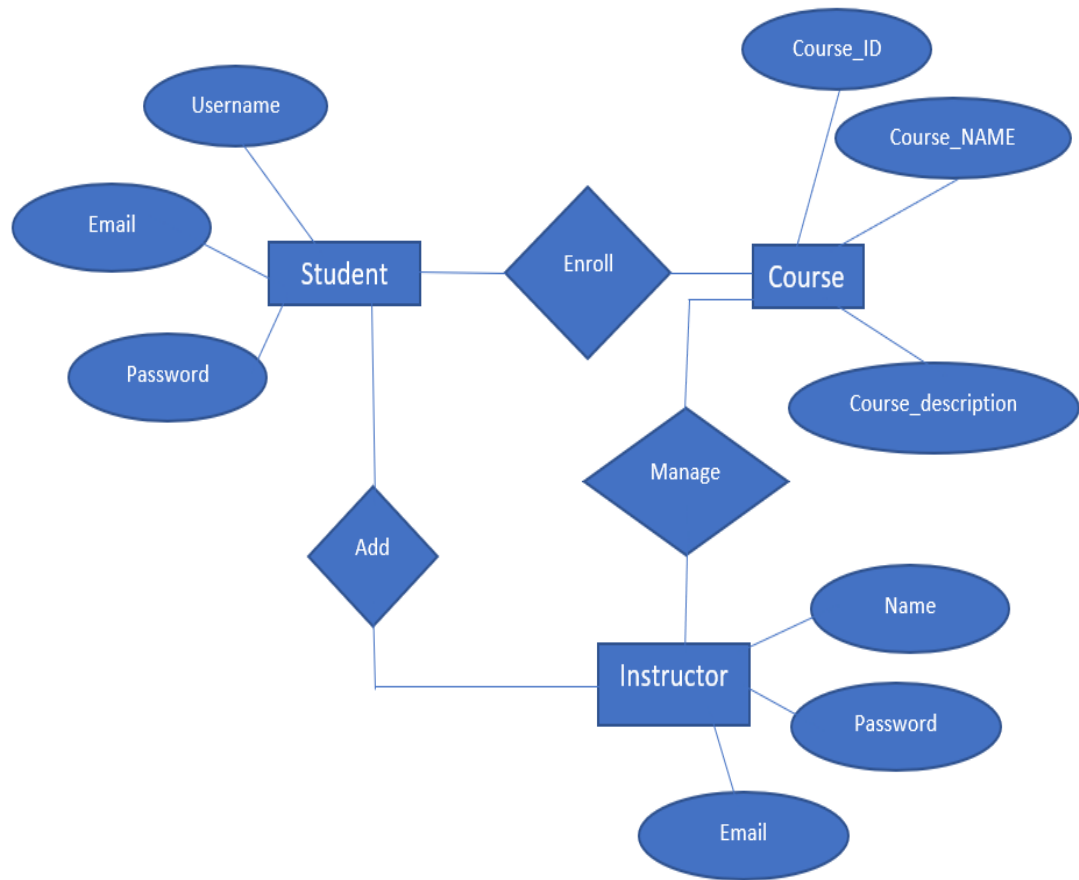


Fig 4.6 E-R Diagram

CHAPTER 5

SYSTEM FEATURES

5.1 Functional Requirements

5.1.1 Registration Page

REQ-1: The user should be able to view and click on login link

REQ-2: The user should be able to enter the username and Password.

REQ-3: There should be only one account per email else and error pop-up should appear.

5.1.2 Login/Signup Page

REQ-1: The user should be able to view and click on login link

REQ-2: The user should be able to enter and validate the username and Password.

REQ-3: There should be only one account per email else and error pop-up should appear.

5.1.3 View Course

REQ-1: The user should be able to view and click on the Course link.

REQ-2: The user should be able to enroll for a particular course.

REQ-3: The user should be able to view course details by clicking on a particular course.

5.2 User Interfaces

i. Login Display:

This is the login page for User & Instructor. This interface is designed to get the users and instructor register into the courses.

ii. Error message view:

If the user data is not identified with the device, the user has to check whether he has

registered using the email which is already in use . This means, this user does not have a record in the system.

iii. **Welcome View page:**

This page means, the device accepts the user credentials and welcomes them into the enrollment process. So, on the screen, it will show the details of all the courses.

iv. **Create a New Course:**

This page will appear at the instructor's side. This page will let the instructor create a new course.

v. **Edit and Publish a Course:**

This page will appear at the instructor's side. Using this page, the instructor can edit the contents of the course and publish the course to the user.

vi. **Add the Instructors:**

This page will appear at the admin's side. Using this page, the admin can assign the instructors to each course.

CHAPTER-6

NON-FUNCTIONAL REQUIREMENTS

6. Non-functional Requirements

6.1. Security Requirements

The Current System Security:

The current system, which is a website, has its policy on its site page. The current system builds upon a username and password access. Students and faculty can access his/her account through their page, and they can control it.

The system now has its own policy and security; however, the new feature we will add to the system will need some security requirements to the system. The new feature in the system will add some values to the current policy to maintain the security in the right way. It also provides proof of compliance.

The new policy in the system will deal with the security in many cases. The security will have more components on the system in a high control panel. The plan is to secure the outsider and insider community of misused the system (e.g. identification theft). Strong security is part of the policy's purpose.

User Access

Inside the community, there are students and instructors who are going to use the system. The main actor in the entity is users. Users will use the system by logging into the system and access the courses. Instructors can access the system through login and can access the course and users information. Instructor role is to add, edit and publish the courses. Furthermore, Admin can add the instructors into the system.

Threats to the system security

This system may face many threats. Sometimes, it comes from a community insider. This could be someone who discloses the data from the database where it is located.

Levels of security:

Operating system: the security in this case will be in the same level of the website & application security.

Network: it is part of the current system security.

The data management system:

- i. Students can access their courses by login.
- ii. Instructor access websites would be the same as we have now, and they will control the courses and users .
- iii. The purpose of the admin is to make an instructor.

Level of access

People level:

- 1) Users (students).
- 2) Instructor (Control on Courses and users).
- 3) Admin (add and control the Instructor).

Reference Monitor

The authorization and the access control present in the security matrix below:

Instructor can login and create, edit, publish the “Courses”

Admin can assign the users the role of “Instructor”.

Users are able to Enroll into the “Courses” and view the ‘Courses’.

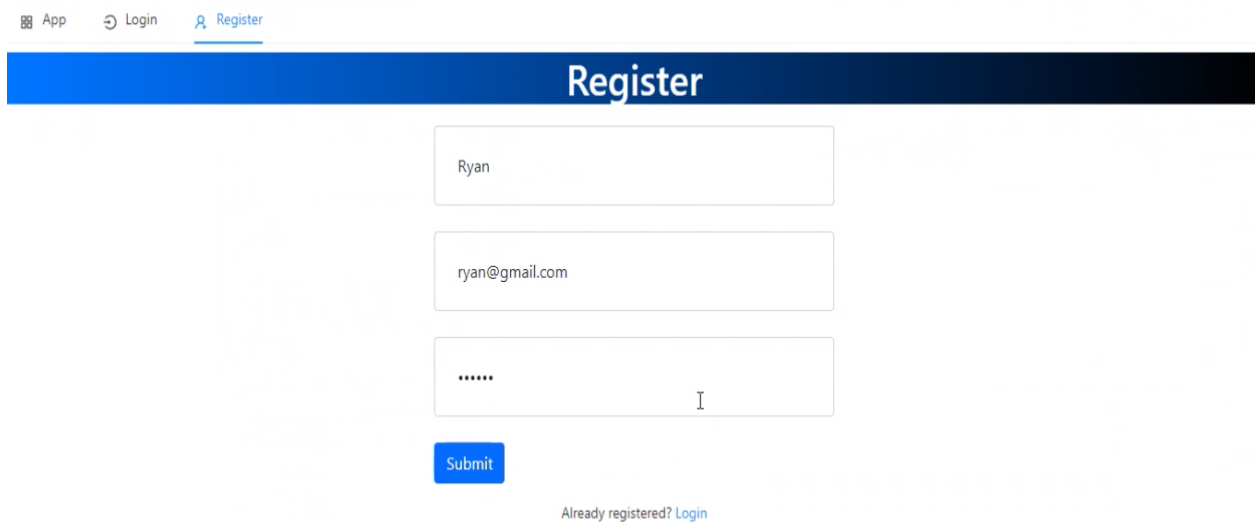
CHAPTER-7

USER INTERFACE

7. User Interface Screenshots

7.1 Registration Page

Through the registration page a user can sign up himself/herself. It will be possible only if the user is not already registered i.e. one email address per user.



The screenshot shows a web application interface for registration. At the top, there is a navigation bar with links for 'App', 'Login', and 'Register'. The 'Register' link is highlighted. Below the navigation bar is a large blue header with the word 'Register' in white. The main content area contains three input fields: the first is for a name, with 'Ryan' entered; the second is for an email address, with 'ryan@gmail.com' entered; and the third is for a password, with six dots indicating masked characters. Below these fields is a blue 'Submit' button. At the bottom, there is a link that says 'Already registered? Login'.

Fig 7.1

7.2 Login Page

This page enables the registered users to login and if the entered email address is not registered then it shows an error and asks the user to register first.

App

Login

Register

Login

instructor@gmail.com

.....

Submit

Not yet registered? Register

Forgot password

7.3 Course Enroll Page

This page lets the user enroll in different courses. To enroll in a course the user must login first.

App

Login

Register

Computer Network

Created by Instructor2

Free



Login To Enroll

5 Lessons

1

Introduction

2

Topologies

3

Communication components

4


Administration of Computer Network

7.4 User Dashboard (Screenshot)


App Create Course Instructor Instructor2

Dashboard


User dashboard




Computer Network
5 lessons
By Instructor2



HTML FOR DEV update1.0
6 lessons
By Instructor2






7.5 Instructor Dashboard (Screenshot)

App Create Course Instructor Instructor2


Dashboard

Course Create


Instructor Dashboard




HTML FOR DEV update1.0
6 Lessons
Your course is live in the marketplace





Computer Network
5 Lessons
Your course is live in the marketplace



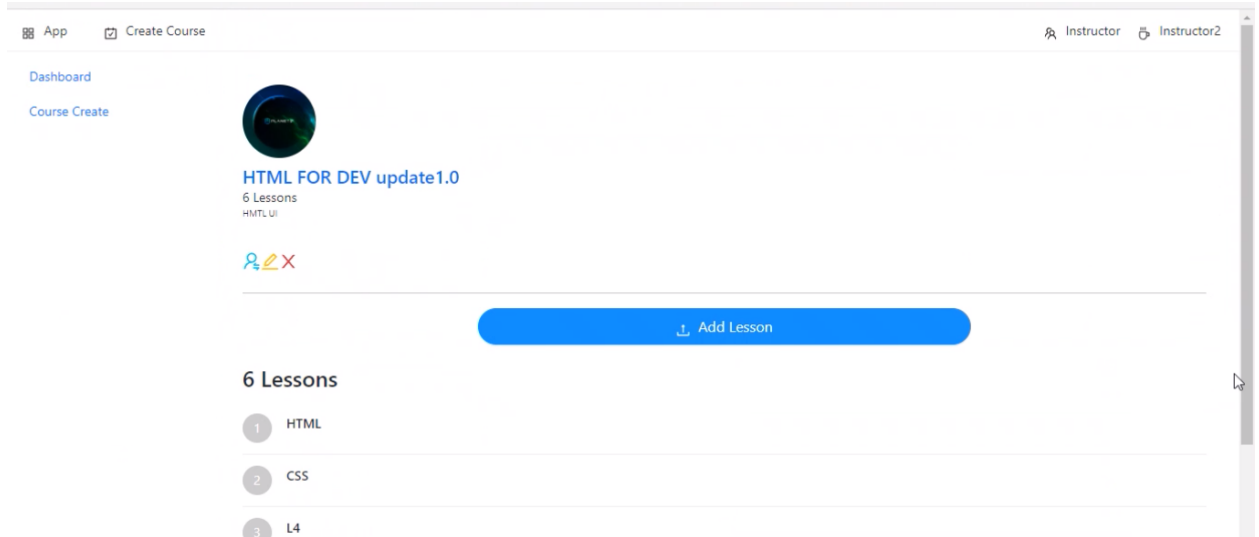
Cloud Computing







7.6 Edit Course



7.7 Create New Course Page (Screenshot)

App
Create Course

Instructor
Instructor2

Dashboard
Course Create

Create Course

Name

Free

Category

Upload Image

Save & Continue

```


{
  "name": "",

```

7.8 Courses Page (Screenshot)


App
Create Course

Instructor
Instructor2




Web development
by Instructor2

Free




Cloud Computing
by Instructor2


Free




Computer Network
by Instructor2

Free







8. CODING

8.1 User Navbar

```
import { useState, useEffect } from "react";
import Link from "next/link";

const UserNav = () => {
  const [current, setCurrent] = useState("");

  useEffect(() => {
    process.browser && setCurrent(window.location.pathname);
  }, [process.browser && window.location.pathname]);

  return (
    <div className="nav flex-column nav-pills">
      <Link href="/user">
        <a className={`nav-link ${current === "/user" && "active"}}`>
          Dashboard
        </a>
      </Link>
    </div>
  );
};

export default UserNav;
```

8.2 Instructor Navbar

```
import { useState, useEffect } from "react";
import Link from "next/link";

const InstructorNav = () => {
  const [current, setCurrent] = useState("");

  useEffect(() => {
    process.browser && setCurrent(window.location.pathname);
  }, [process.browser && window.location.pathname]);

  return (
```

```

    <div className="nav flex-column nav-pills">
      <Link href="/instructor">
        <a className={`nav-link ${current === "/instructor" &&
"active"}}`>
          Dashboard
        </a>
      </Link>
      <Link href="/instructor/course/create">
        <a
          className={`nav-link ${
            current === "/instructor/course/create" && "active"
          }}`>
          Course Create
        </a>
      </Link>
    </div>
  );
};

export default InstructorNav;

```

8.3 User Routes

```

import { useEffect, useState } from "react";
import axios from "axios";
import { useRouter } from "next/router";
import { SyncOutlined } from "@ant-design/icons";
import UserNav from "../nav/UserNav";
const UserRoute = ({ children }) => {
  // state
  const [ok, setOk] = useState(false);
  // router
  const router = useRouter();

  useEffect(() => {
    fetchUser();
  }, []);

```



```

const fetchUser = async () => {
  try {
    const { data } = await axios.get("/api/current-user");
    // console.log(data);
    if (data.ok) setOk(true);
  } catch (err) {
    console.log(err);
    setOk(false);
    router.push("/login");
  }
};

return (
  <>
    {!ok ? (
      <SyncOutlined
        spin
        className="d-flex justify-content-center display-1 text-primary
p-5"
      />
    ) : (
      <div className="container-fluid">
        <div className="row">
          <div className="col-md-2">
            <UserNav />
          </div>
          <div className="col-md-10">{children}</div>
        </div>
      </div>
    )}
  </>
);
};

export default UserRoute;

```

8.4 auth.js

```
import express from "express";
```

```

import { requireSignin } from "../middlewares";

const router = express.Router();

import {
  register,
  login,
  logout,
  currentUser,
  forgotPassword,
  resetPassword,
} from "../controllers/auth";

router.post("/register", register);
router.post("/login", login);
router.get("/logout", logout);
router.get("/current-user", requireSignin, currentUser);
router.post("/forgot-password", forgotPassword);
router.post("/reset-password", resetPassword);

module.exports = router;

```

8.5 courses.js

```

import express from "express";
import formidable from "express-formidable";

const router = express.Router();

// middleware
import { isEnrolled, isInstructor, requireSignin } from "../middlewares";

// controllers
import {
  uploadImage,
  removeImage,
  create,
  read,
  uploadVideo,
  removeVideo,
  addLesson,

```

```

    update,
    removeLesson,
    updateLesson,
    publishCourse,
    unpublishCourse,
    courses,
    checkEnrollment,
    freeEnrollment,
    userCourses,
    markCompleted,
    listCompleted,
    markIncomplete,
} from "../controllers/course";

router.get("/courses", courses);

router.post("/course/upload-image", uploadImage);
router.post("/course/remove-image", removeImage);

router.post("/course", requireSignin, isInstructor, create);
router.put("/course/:slug", requireSignin, isInstructor, update);
router.get("/course/:slug", read);
router.post(
  "/course/video-upload/:instructorId",
  requireSignin,
  formidable(),
  uploadVideo
);
router.post("/course/video-remove/:instructorId", requireSignin,
removeVideo);

router.put("/course/publish/:courseId", requireSignin, publishCourse);
router.put("/course/unpublish/:courseId", requireSignin, unpublishCourse);

router.post("/course/lesson/:slug/:instructorId", requireSignin,
addLesson);
router.put("/course/:slug/:lessonId", requireSignin, removeLesson);
router.put("/course/lesson/:slug/:instructorId", requireSignin,
updateLesson);

```

```

router.get("/check-enrollment/:courseId", requireSignin, checkEnrollment);
router.post("/free-enrollment/:courseId", requireSignin, freeEnrollment);

router.get("/user-courses", requireSignin, userCourses);
router.get("/user/course/:slug", requireSignin, isEnrolled, read);

router.post("/mark-completed", requireSignin, markCompleted);
router.post("/list-completed", requireSignin, listCompleted);
router.post("/mark-incomplete", requireSignin, markIncomplete);

module.exports = router;

```

8.6 instructor.js

```

import express from "express";

const router = express.Router();

// middleware
import { requireSignin } from "../middlewares";

// controllers
import {
  currentInstructor,
  instructorCourses,
  studentCount,
} from "../controllers/instructor";

router.get("/current-instructor", requireSignin, currentInstructor);
router.get("/instructor-courses", requireSignin, instructorCourses);

router.post("/instructor/student-count", requireSignin, studentCount);

module.exports = router;

```

8.7 utils auth.js

```

import bcrypt from "bcrypt";

export const hashPassword = (password) => {
  return new Promise((resolve, reject) => {
    bcrypt.genSalt(12, (err, salt) => {
      if (err) {
        reject(err);
      }
      bcrypt.hash(password, salt, (err, hash) => {
        if (err) {
          reject(err);
        }
        resolve(hash);
      });
    });
  });
};

export const comparePassword = (password, hashed) => {
  return bcrypt.compare(password, hashed);
};

```

8.8 Middleware index.js

```

import expressJwt from "express-jwt";
import User from "../models/user";
import Course from "../models/course";

export const requireSignin = expressJwt({
  getToken: (req, res) => req.cookies.token,
  secret: process.env.JWT_SECRET,
  algorithms: ["HS256"],
});

export const isInstructor = async (req, res, next) => {
  try {
    const user = await User.findById(req.user._id).exec();
    if (!user.role.includes("Instructor")) {
      return res.sendStatus(403);
    } else {

```

```

        next();
    }
} catch (err) {
    console.log(err);
}
};

export const isEnrolled = async (req, res, next) => {
    try {
        const user = await User.findById(req.user._id).exec();
        const course = await Course.findOne({ slug: req.params.slug }).exec();

        // check if course id is found in user courses array
        let ids = [];
        for (let i = 0; i < user.courses.length; i++) {
            ids.push(user.courses[i].toString());
        }

        if (!ids.includes(course._id.toString())) {
            res.sendStatus(403);
        } else {
            next();
        }
    } catch (err) {
        console.log(err);
    }
};

```

8.9 controllers auth.js

```

import User from "../models/user";
import { hashPassword, comparePassword } from "../utils/auth";
import jwt from "jsonwebtoken";
import { nanoid } from "nanoid";
import AWS from "aws-sdk";

const awsConfig = {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,

```

```

    region: process.env.AWS_REGION,
    apiVersion: process.env.AWS_API_VERSION,
  };

const SES = new AWS.SES(awsConfig);

export const register = async (req, res) => {
  try {
    // console.log(req.body);
    const { name, email, password } = req.body;
    // validation
    if (!name) return res.status(400).send("Name is required");
    if (!password || password.length < 6) {
      return res
        .status(400)
        .send("Password is required and should be min 6 characters long");
    }
    let userExist = await User.findOne({ email }).exec();
    if (userExist) return res.status(400).send("Email is taken");

    // hash password
    const hashedPassword = await hashPassword(password);

    // register
    const user = new User({
      name,
      email,
      password: hashedPassword,
    });
    await user.save();
    // console.log("saved user", user);
    return res.json({ ok: true });
  } catch (err) {
    console.log(err);
    return res.status(400).send("Error. Try again.");
  }
};

export const login = async (req, res) => {

```

```

try {
  // console.log(req.body);
  const { email, password } = req.body;
  // check if our db has user with that email
  const user = await User.findOne({ email }).exec();
  if (!user) return res.status(400).send("No user found");
  // check password
  const match = await comparePassword(password, user.password);
  if (!match) return res.status(400).send("Wrong password");

  // create signed jwt
  const token = jwt.sign({ _id: user._id }, process.env.JWT_SECRET, {
    expiresIn: "30d",
  });
  // return user and token to client, exclude hashed password
  user.password = undefined;
  // send token in cookie
  res.cookie("token", token, {
    httpOnly: true,
    // secure: true, // only works on https
  });
  // send user as json response
  res.json(user);
} catch (err) {
  console.log(err);
  return res.status(400).send("Error. Try again.");
}
};

export const logout = async (req, res) => {
  try {
    res.clearCookie("token");
    return res.json({ message: "Signout success" });
  } catch (err) {
    console.log(err);
  }
};

export const currentUser = async (req, res) => {

```



```

    try {
        const user = await
User.findById(req.user._id).select("-password").exec();
        console.log("CURRENT_USER", user);
        return res.json({ ok: true });
    } catch (err) {
        console.log(err);
    }
};

export const forgotPassword = async (req, res) => {
    try {
        const { email } = req.body;
        // console.log(email);
        const shortCode = nanoid(6).toUpperCase();
        const user = await User.findOneAndUpdate(
            { email },
            { passwordResetCode: shortCode }
        );
        if (!user) return res.status(400).send("User not found");

        // prepare for email
        const params = {
            Source: process.env.EMAIL_FROM,
            Destination: {
                ToAddresses: [email],
            },
            Message: {
                Body: {
                    Html: {
                        Charset: "UTF-8",
                        Data: `
<html>
    <h1>Reset password</h1>
    <p>User this code to reset your password</p>
    <h2 style="color:red;">${shortCode}</h2>
    <i>edemy.com</i>
</html>
`
                    },
                },
            },
        };
    }
};

```

```

    },
  },
  Subject: {
    Charset: "UTF-8",
    Data: "Reset Password",
  },
},
};

const emailSent = SES.sendEmail(params).promise();
emailSent
  .then((data) => {
    console.log(data);
    res.json({ ok: true });
  })
  .catch((err) => {
    console.log(err);
  });
} catch (err) {
  console.log(err);
}
};

export const resetPassword = async (req, res) => {
  try {
    const { email, code, newPassword } = req.body;
    // console.table({ email, code, newPassword });
    const hashedPassword = await hashPassword(newPassword);

    const user = User.findOneAndUpdate(
      {
        email,
        passwordResetCode: code,
      },
      {
        password: hashedPassword,
        passwordResetCode: "",
      }
    ).exec();
  }
};

```

```

    res.json({ ok: true });
  } catch (err) {
    console.log(err);
    return res.status(400).send("Error! Try again.");
  }
};

```

8.10 controllers courses

```

import AWS from "aws-sdk";
import { nanoid } from "nanoid";
import Course from "../models/course";
import Completed from "../models/completed";
import slugify from "slugify";
import { readFileSync } from "fs";
import User from "../models/user";

const awsConfig = {
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  region: process.env.AWS_REGION,
  apiVersion: process.env.AWS_API_VERSION,
};

const S3 = new AWS.S3(awsConfig);

export const uploadImage = async (req, res) => {
  // console.log(req.body);
  try {
    const { image } = req.body;
    if (!image) return res.status(400).send("No image");

    // prepare the image
    const base64Data = new Buffer.from(
      image.replace(/^data:image\/\w+;base64/, ""),
      "base64"
    );

    const type = image.split(";")[0].split("/")[1];
  }
};

```

```

// image params
const params = {
  Bucket: "edemybuckets",
  Key: `${nanoid()}.${type}`,
  Body: base64Data,
  ACL: "public-read",
  ContentEncoding: "base64",
  ContentType: `image/${type}`,
};

// upload to s3
S3.upload(params, (err, data) => {
  if (err) {
    console.log(err);
    return res.sendStatus(400);
  }
  console.log(data);
  res.send(data);
});
} catch (err) {
  console.log(err);
}
};

export const removeImage = async (req, res) => {
  try {
    const { image } = req.body;
    // image params
    const params = {
      Bucket: image.Bucket,
      Key: image.Key,
    };

    // send remove request to s3
    S3.deleteObject(params, (err, data) => {
      if (err) {
        console.log(err);
        res.sendStatus(400);
      }
    });
  }
};

```

```

    }
    res.send({ ok: true });
  });
} catch (err) {
  console.log(err);
}
};

export const create = async (req, res) => {
  // console.log("CREATE COURSE", req.body);
  // return;
  try {
    const alreadyExist = await Course.findOne({
      slug: slugify(req.body.name.toLowerCase()),
    });
    if (alreadyExist) return res.status(400).send("Title is taken");

    const course = await new Course({
      slug: slugify(req.body.name),
      instructor: req.user._id,
      ...req.body,
    }).save();

    res.json(course);
  } catch (err) {
    console.log(err);
    return res.status(400).send("Course create failed. Try again.");
  }
};

export const read = async (req, res) => {
  try {
    const course = await Course.findOne({ slug: req.params.slug })
      .populate("instructor", "_id name")
      .exec();
    res.json(course);
  } catch (err) {
    console.log(err);
  }
}

```

```

};

export const uploadVideo = async (req, res) => {
  try {
    // console.log("req.user._id", req.user._id);
    // console.log("req.params.instructorId", req.params.instructorId);
    if (req.user._id !== req.params.instructorId) {
      return res.status(400).send("Unauthorized");
    }

    const { video } = req.files;
    // console.log(video);
    if (!video) return res.status(400).send("No video");

    // video params
    const params = {
      Bucket: "edemybuckets",
      Key: `${nanoid()}.${video.type.split("/") [1]}`,
      Body: readFileSync(video.path),
      ACL: "public-read",
      ContentType: video.type,
    };

    // upload to s3
    S3.upload(params, (err, data) => {
      if (err) {
        console.log(err);
        res.sendStatus(400);
      }
      console.log(data);
      res.send(data);
    });
  } catch (err) {
    console.log(err);
  }
};

export const removeVideo = async (req, res) => {
  try {

```

```

    if (req.user._id !== req.params.instructorId) {
      return res.status(400).send("Unauthorized");
    }

    const { Bucket, Key } = req.body;
    // console.log("VIDEO REMOVE =====> ", req.body);

    // video params
    const params = {
      Bucket,
      Key,
    };

    // upload to s3
    S3.deleteObject(params, (err, data) => {
      if (err) {
        console.log(err);
        res.sendStatus(400);
      }
      console.log(data);
      res.send({ ok: true });
    });
  } catch (err) {
    console.log(err);
  }
};

export const addLesson = async (req, res) => {
  try {
    const { slug, instructorId } = req.params;
    const { title, content, video } = req.body;

    if (req.user._id !== instructorId) {
      return res.status(400).send("Unauthorized");
    }

    const updated = await Course.findOneAndUpdate(
      { slug },
      {

```

```

        $push: { lessons: { title, content, video, slug: slugify(title) }
    },
    },
    { new: true }
)
    .populate("instructor", "_id name")
    .exec();
res.json(updated);
} catch (err) {
    console.log(err);
    return res.status(400).send("Add lesson failed");
}
};

export const update = async (req, res) => {
    try {
        const { slug } = req.params;
        // console.log(slug);
        const course = await Course.findOne({ slug }).exec();
        // console.log("COURSE FOUND => ", course);
        if (req.user._id !== course.instructor._id) {
            return res.status(400).send("Unauthorized");
        }

        const updated = await Course.findOneAndUpdate({ slug }, req.body, {
            new: true,
        }).exec();

        res.json(updated);
    } catch (err) {
        console.log(err);
        return res.status(400).send(err.message);
    }
};

export const removeLesson = async (req, res) => {
    const { slug, lessonId } = req.params;
    const course = await Course.findOne({ slug }).exec();
    if (req.user._id !== course.instructor._id) {

```



```

        return res.status(400).send("Unauthorized");
    }

    const deletedCourse = await Course.findByIdAndUpdate(course._id, {
        $pull: { lessons: { _id: lessonId } },
    }).exec();

    res.json({ ok: true });
};

export const updateLesson = async (req, res) => {
    try {
        const { courseId, lessonId } = req.params;
        const { _id, title, content, video, free_preview } = req.body;
        // find post
        const courseFound = await Course.findById(courseId)
            .select("instructor")
            .exec();
        // is owner?
        if (req.user._id !== courseFound.instructor._id) {
            return res.status(400).send("Unauthorized");
        }

        const updated = await Course.updateOne(
            { "lessons._id": lessonId },
            {
                $set: {
                    "lessons.$.title": title,
                    "lessons.$.content": content,
                    "lessons.$.video": video,
                    "lessons.$.free_preview": free_preview,
                },
            },
        ).exec();
        console.log("updated => ", updated);
        res.json({ ok: true });
    } catch (err) {
        console.log(err);
        return res.status(400).send("Update lesson failed");
    }
}

```

```

    }
  };

export const publishCourse = async (req, res) => {
  try {
    const { courseId } = req.params;
    const courseFound = await Course.findById(courseId)
      .select("instructor")
      .exec();
    if (req.user._id !== courseFound.instructor._id) {
      return res.status(400).send("Unauthorized");
    }

    let course = await Course.findByIdAndUpdate(
      courseId,
      { published: true },
      { new: true }
    ).exec();
    res.json(course);
  } catch (err) {
    console.log(err);
    return res.status(400).send("Publish course failed");
  }
};

```

```

export const unpublishCourse = async (req, res) => {
  try {
    const { courseId } = req.params;
    const courseFound = await Course.findById(courseId)
      .select("instructor")
      .exec();
    if (req.user._id !== courseFound.instructor._id) {
      return res.status(400).send("Unauthorized");
    }

    let course = await Course.findByIdAndUpdate(
      courseId,
      { published: false },
      { new: true }
    ).exec();
    res.json(course);
  } catch (err) {
    console.log(err);
    return res.status(400).send("Unpublish course failed");
  }
};

```

```

    ).exec();
    res.json(course);
  } catch (err) {
    console.log(err);
    return res.status(400).send("Unpublish course failed");
  }
};

export const courses = async (req, res) => {
  const all = await Course.find({ published: true })
    .limit(11)
    .populate("instructor", "_id name")
    .populate("categories", "_id name")
    .exec();
  res.json(all);
};

export const checkEnrollment = async (req, res) => {
  const { courseId } = req.params;
  // find courses of the currently logged in user
  const user = await User.findById(req.user._id).exec();
  // check if course id is found in user courses array
  let ids = [];
  let length = user.courses && user.courses.length;
  for (let i = 0; i < length; i++) {
    ids.push(user.courses[i].toString());
  }
  res.json({
    status: ids.includes(courseId),
    course: await Course.findById(courseId).exec(),
  });
};

export const freeEnrollment = async (req, res) => {
  try {
    // check if course is free or paid
    const course = await Course.findById(req.params.courseId).exec();
    if (course.paid) return;
  }
};

```

```

    const result = await User.findByIdAndUpdate(
      req.user._id,
      {
        $addToSet: { courses: course._id },
      },
      { new: true }
    ).exec();
    console.log(result);
    res.json({
      message: "Congratulations! You have successfully enrolled",
      course,
    });
  } catch (err) {
    console.log("free enrollment err", err);
    return res.status(400).send("Enrollment create failed");
  }
};

export const userCourses = async (req, res) => {
  const user = await User.findById(req.user._id).exec();
  const courses = await Course.find({ _id: { $in: user.courses } })
    .populate("instructor", "_id name")
    .exec();
  res.json(courses);
};

export const markCompleted = async (req, res) => {
  const { courseId, lessonId } = req.body;
  // console.log(courseId, lessonId);
  // find if user with that course is already created
  const existing = await Completed.findOne({
    user: req.user._id,
    course: courseId,
  }).exec();

  if (existing) {
    // update
    const updated = await Completed.findOneAndUpdate(
      {

```

```

        user: req.user._id,
        course: courseId,
      },
      {
        $addToSet: { lessons: lessonId },
      }
    ).exec();
    res.json({ ok: true });
  } else {
    // create
    const created = await new Completed({
      user: req.user._id,
      course: courseId,
      lessons: lessonId,
    }).save();
    res.json({ ok: true });
  }
};

export const listCompleted = async (req, res) => {
  try {
    const list = await Completed.findOne({
      user: req.user._id,
      course: req.body.courseId,
    }).exec();
    list && res.json(list.lessons);
  } catch (err) {
    console.log(err);
  }
};

export const markIncomplete = async (req, res) => {
  try {
    const { courseId, lessonId } = req.body;

    const updated = await Completed.findOneAndUpdate(
      {
        user: req.user._id,
        course: courseId,

```

```

    },
    {
      $pull: { lessons: lessonId },
    }
  ).exec();
  res.json({ ok: true });
} catch (err) {
  console.log(err);
}
};

```

8.11 become-instructor.js

```

import { useContext, useState, us } from "react";
import { Context } from "../../context";
import { Button } from "antd";
import axios from "axios";
import {
  SettingOutlined,
  UserSwitchOutlined,
  LoadingOutlined,
} from "@ant-design/icons";
import { toast } from "react-toastify";
import Router, { useRouter } from "next/router";
import UserRoute from "../../components/routes/UserRoute";

const BecomeInstructor = () => {
  // state
  const [loading, setLoading] = useState(false);
  const router = useRouter();
  const {
    state: { user },
  } = useContext(Context);

  const becomeInstructor = () => {
    setLoading(true);
    axios
      .post("/api/make-instructor")
      .then((res) => {

```

```

        console.log(res);
        router.push("/course/create");
    })
    .catch((err) => {
        console.error(err);
        router.push("/course/create");
        toast("Stripe onboarding failed. Try again.");
        setLoading(false);
    });
};

return (
    <>
        <h1 className="jumbotron text-center square">Become Instructor</h1>

        <div className="container">
            <div className="row">
                <div className="col-md-6 offset-md-3 text-center">
                    <div className="pt-4">
                        <UserSwitchOutlined className="display-1 pb-3" />
                        <br />
                        <h2>Setup payout to publish courses on Edemy</h2>
                        <p className="lead text-warning">
                            Edemy partners with stripe to transfer earnings to your
bank
                            account
                        </p>

                        <Button
                            className="mb-3"
                            type="primary"
                            block
                            shape="round"
                            icon={loading ? <LoadingOutlined /> : <SettingOutlined />}
                            size="large"
                            onClick={becomeInstructor}
                            disabled={
                                (user && user.role && user.role.includes("Instructor"))

```



```

const {
  state: { user },
} = useContext(Context);
// router
const router = useRouter();

// redirect if user is logged in
useEffect(() => {
  if (user !== null) router.push("/");
}, [user]);

const handleSubmit = async (e) => {
  e.preventDefault();
  setLoading(true);
  try {
    const { data } = await axios.post("/api/forgot-password", { email
  });
  setSuccess(true);
  toast("Check your email for the secret code");
  setLoading(false);
} catch (err) {
  setLoading(false);
  toast(err.response.data);
}
};

const handleResetPassword = async (e) => {
  e.preventDefault();
  // console.log(email, code, newPassword);
  // return;
  try {
    setLoading(true);
    const { data } = await axios.post("/api/reset-password", {
      email,
      code,
      newPassword,
    });
    setEmail("");
    setCode("");
  }

```

```

    setNewPassword("");
    setLoading(false);
  } catch (err) {
    setLoading(false);
    toast(err.response.data);
  }
};

return (
  <>
    <h1 className="jumbotron text-center bg-primary square">
      Forgot Password
    </h1>

    <div className="container col-md-4 offset-md-4 pb-5">
      <form onSubmit={success ? handleResetPassword : handleSubmit}>
        <input
          type="email"
          className="form-control mb-4 p-4"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          placeholder="Enter email"
          required
        />
        {success && (
          <>
            <input
              type="text"
              className="form-control mb-4 p-4"
              value={code}
              onChange={(e) => setCode(e.target.value)}
              placeholder="Enter secret code"
              required
            />

            <input
              type="password"
              className="form-control mb-4 p-4"
              value={newPassword}

```

```

        onChange={ (e) => setNewPassword(e.target.value) }
        placeholder="New Password"
        required
      />
    </>
  )}

  <button
    type="submit"
    className="btn btn-primary btn-block p-2"
    disabled={loading || !email}
  >
    {loading ? <SyncOutlined spin /> : "Submit"}
  </button>
</form>
</div>
</>
);
};

export default ForgotPassword;

```

8.13 login.js

```

import { useState, useContext, useEffect } from "react";
import axios from "axios";
import { toast } from "react-toastify";
import { SyncOutlined } from "@ant-design/icons";
import Link from "next/link";
import { Context } from "../context";
import { useRouter } from "next/router";

const Login = () => {
  const [email, setEmail] = useState("instructor@gmail.com");
  const [password, setPassword] = useState("rrrrrrr");
  const [loading, setLoading] = useState(false);

  // state
  const { state, dispatch } = useContext(Context);
  const { user } = state;

```

```

//router
const router = useRouter();

useEffect(() => {
  if (user !== null) {
    router.push("/");
  }
}, [user]);

const handleSubmit = async (e) => {
  e.preventDefault();
  // console.table({ name, email, password });
  try {
    setLoading(true);
    const { data } = await axios.post(`/api/login`, {
      email,
      password,
    });
    dispatch({
      type: "LOGIN",
      payload: data,
    });

    window.localStorage.setItem("user", JSON.stringify(data));

    //redirect
    router.push("/user");
  } catch (err) {
    toast(err.response.data);
    setLoading(false);
  }
};

return (
  <>
    <h1 className="jumbotron text-center bg-primary square">Login</h1>

    <div className="container col-md-4 offset-md-4 pb-5">

```

```

<form onSubmit={handleSubmit}>
  <input
    type="email"
    className="form-control mb-4 p-4"
    value={email}
    onChange={(e) => setEmail(e.target.value)}
    placeholder="Enter email"
    required
  />

  <input
    type="password"
    className="form-control mb-4 p-4"
    value={password}
    onChange={(e) => setPassword(e.target.value)}
    placeholder="Enter password"
    required
  />

  <button
    type="submit"
    className="btn btn-block btn-primary"
    disabled={!email || !password || loading}
  >
    {loading ? <SyncOutlined spin /> : "Submit"}
  </button>
</form>

<p className="text-center p-3">
  Not yet registered?{" "}
  <Link href="/register">
    <a>Register</a>
  </Link>
</p>

<p className="text-center">
  <Link href="/forgot-password">
    <a className="text-danger">Forgot password</a>
  </Link>

```

```

    </p>
  </div>
</>
);
};

export default Login;

```

9 TESTING

Project Name: E-Learning Platform(ELP)
 Team Leader: Kajal Punia
 Execution Date: 08/12/2021

Project ID: SN1
 QA Manager:

Test Cases										
Test Case Id	Component	Priority	Test cases	Description/Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed By
TC01	Registration	p10	TC01.1	To verify that when a user enters an unregistered email address and presses enter, an OTP must be sent to that email and a page should open that will ask the user to enter the OTP(One Time Password).	ELP website must be open.	1. Click on the register option. 2. Once the registration page is launched, enter a valid and unregistered email address. 3. Press enter	An OTP should be sent to the entered email address and next page should be launched where the user needs to enter the OTP.	An OTP sent to the entered email address and next page launched where the user needs to enter the OTP.	Pass	
			TC01.2	To verify that when a user enters an already registered email address then an error message should pop up saying "Account already exists, login instead".	A registered email address and ELP website must be open.	1. Click on the register option. 2. Once the registration page is launched, enter an already registered email address. 3. Press enter	An error message should pop up saying "Account already exists, login instead".	An error message pops up saying "Account already exists, login instead".	Pass	
			TC01.3	To verify that when a user enters an invalid email address then an error message should pop up saying "Enter a valid email address".	ELP website must be open.	1. Click on the register option. 2. Once the registration page is launched, enter an invalid email address. 3. Press enter	An error message should pop up saying "Enter a valid email address".	An error message pops up saying "Enter a valid email address".	Pass	
TC02	Login	p10	TC02.1	To verify that when a user enters a registered email address and presses enter, next page should be launched where the user needs to enter the password.	ELP website must be open.	1. Click on the Login option. 2. Once the Login page is launched, enter a registered email address. 3. Press enter	Next page should be launched where the user needs to enter the password.	Next page launched to enter the password.	Pass	
			TC02.2	To verify that when a user enters an unregistered email address and presses enter, an error message must pop up saying "No such account found, SignUp instead".	ELP website must be open.	1. Click on the Login option. 2. Once the Login page is launched, enter an unregistered email. 3. Press enter	An error message should pop up saying "No such account found, Login instead."	An error message pops up saying "No such account found, Login instead."	Pass	
			TC02.3	To verify that when a user enters an invalid email address and presses enter, an error message must pop up saying "Enter a valid email address".	ELP website must be open.	1. Click on the Login option. 2. Once the Login page is launched, enter an invalid email. 3. Press enter	An error message should pop up saying "Enter a valid email address."	An error message pops up saying "Enter a valid email address."	Pass	

TC03	Search_Bar_Module	P5	TC03.1	To verify that when a user writes a search term and presses enter, search results should be displayed.	ELP website must be open.	1. Write the name of the course or subject in the search bar. 2. Press enter	Search results related to the searched term should be displayed	Search results with 'searched term' keywords are displayed.	Pass	
------	-------------------	----	--------	--	---------------------------	---	---	---	------	--