E-LEARNING MANAGEMENT SYSTEM

A Project Report

Submitted to the

MANONMANIAM SUNDARANAR UNIVERSITY

In partial fulfillment for award of the degree of

BACHELOR OF COMPUTER APPLICATIONS

BY

SUBIN A	20213211401228
SIVARAM S L	20213211401226
MERLIN SAM E	20213211401211
SUBIN JOSUVA G	20213211401229

Under the guidance of

Mrs. V.THANGA RUBITHA M.C.A. , M.Phil

Assistant Professor, Department of Computer Applications

St. Jerome's College (Arts & Science)

Anandhanadarkudy, Nagercoil – 629201.

DEPARTMENT OF COMPUTER APPLICATIONS

ST. JEROME'S COLLEGE (Arts and Science)

ANANDHANADARKUDY

APRIL-2024

ST.JEROME'S COLLEGE

ANANDHANADARKUDY

DEPARTMENT OF COMPUTER APPLICATIONS

<u>CERTIFICATE</u>

This is to certify that the project entitled **"E-LEARNING MANAGEMENT SYSTEM"** is a bonafide record of the work done

BY

SUBIN A	20213211401228
SIVARAM S L	20213211401226
MERLIN SAM E	20213211401211
SUBIN JOSUVA G	20213211401229

of the department of Computer Applications, St.Jerome's College, A.N.Kudy during the academic year 2023-2024 for the partial fulfillment of the requirements for the award of Bachelor's Degree in Computer Applications is a record of original project work submitted to **Manonmaniam Sundaranar University, Tirunelveli,** under the guidance of **Mrs. V.THANGA RUBITHA M.C.A. , M.Phil,** Assistant Professor, Department of Computer Applications.

Signature of the Guide **Head of the Department**

Submitted for the viva-voce examination held at Department of Computer Applications, St.Jerome's college, Anandhanadarkudy on_______________.

External Examiner 1 External Examier 2

CERTIFICATE

This is to certify that the project entitled **“E-LEARNING MANAGEMENT SYSTEM”** is a bonafiderecord of the work done

BY

SUBIN A	20213211401228
SIVARAM S L	20213211401226
MERLIN SAM E	20213211401211
SUBIN JOSUVA G	20213211401229

of the department of Computer Applications, St.Jerome’s College, Anandhanadarkudy during the academic year 2023-2024 for the partial fulfillment of requirement of Bachelor’s Degree in Computer Applications, submitted to Manonmaniam Sundaranar University, Tirunelveli is a record of original work done under my supervision and guidance and the project has not formed the basis for the award of any Degree/Diploma or other similar title of any candidates of any university.

Signature of the Guide

Declaration

We hereby declare that this project entitled "**E-Learning**" is an authentic record of study carried out by us during the academic year 2023-2024 under the guidance of **Mrs. V.THANGA RUBITHA M.C.A, M.Phil,** Assistant professor, Department of Computer Applications in partial fulillment of the requirements for award of Bachelor’s degree in Computer Applications in Manonmaniam Sundaranar University and that no part of it has been previously presented for any degree of diploma in any university.

By,

Subin A

Sivaram S.L

Merlin Sam E

Subin Josuva G

Acknowlegement

In the accomplishment of this project successfully, many people have bestowed upon us their blessing and the heart pledged support, this time we are utilizing all the people who have be concerned with this project.

First of all, we would like to thank God the almighty for being able to complete this project success. Then we would like to thank our beloved correspondent Rev**. Fr.L.Sundar, Crs.,** for his full support. Then we extend our heartfelt thanks to our college principal **Dr.S.Rajesh** who supported us from beginning .Then we extend our thanks to our Head of the department **Dr.J.Jeslin Suji**, who encourages us to do this project successfully. Then we extend our thanks to our project guide **Mrs. V.THANGA RUBITHA M.C.A. , M.Phil,** Asst. Professor, Department of Computer Applications and all the staff of our department whose valuable guidance has been the one that helped uspatch this project and make it full proof success. The suggestions and their instructions have saved me asthe major contributor towards the completion of the project.

Then we would like to thank our parents and friends who have helped us with their valuable suggestions and guidance which have been very helpful in various phases of the completion of the project.

Last but not the least we would like to thank our classmates and department students who have helped a lot.

CONTENTS

1.Project Details

1.1 Instructor module

1.2 Student module

2. System Analysis

2.1 Existing System

2.2 Proposed System

3. System Requirements

3.1 Hardware specification

3.2 Software specification

4. System Environment

4.1 Javascript

4.1.1 Reactjs

4.1.2 Nodejs

4.1.3 Expressjs

4.2 Mongodb

4.3 Postman

5. System Design

5.1 Introduction

5.2 Class Diagram

5.3 UML Diagram

6. Database Design

6.1 Collections

7. System Testing

7.1 Unit Testing

7.2 Integration Testing

8 Source Code

9. Output Layout

10. Conclusion

11. Future Enhancement

12. Bibliography

Abstract

The E-Learning Platform represents a groundbreaking initiative poised to redefine the landscape of education in the digital age. Crafted as a dynamic online hub, it serves as a beacon of accessibility and innovation, offering students an expansive marketplace where they can embark on a journey of discovery across a vast spectrum of courses. From foundational subjects to advanced skill development, the platform caters to learners of all backgrounds and interests, fostering a culture of continuous growth and exploration.
At its core, this platform empowers instructors to become architects of knowledge, providing them with the tools and resources to create, publish, and manage their courses with unparalleled ease and flexibility. With a global audience at their fingertips, educators can transcend geographical boundaries and share their expertise on a scale never before imagined, enriching the educational experience for learners around the world.

Embedded within the platform's architecture is a robust framework for secure payment processing, ensuring seamless transactions and peace of mind for both students and instructors alike. Furthermore, stringent user authentication mechanisms lay the foundation for a safe and engaging learning environment, where collaboration and interaction thrive.

In essence, the E-Learning Platform is not just a platform—it's a catalyst for change, a testament to the transformative power of technology in education. By embracing accessibility, convenience, and the boundless possibilities of digital learning, it heralds a new era of educational excellence, where knowledge knows no limits and the pursuit of learning knows no bounds.

In addition to its role as a facilitator of traditional coursework, the E-Learning Platform champions the ethos of lifelong learning, empowering individuals to pursue their passions and interests at their own pace and on their own terms. Through a diverse range of supplementary resources, such as webinars, tutorials, and interactive forums, users can delve deeper into their areas of interest, expand their horizons, and cultivate new skills beyond the confines of traditional academia.

Furthermore, the platform serves as a catalyst for professional development, offering a gateway to career advancement and personal growth. Whether seeking to acquire new skills for career progression or exploring new avenues of personal enrichment, users have access to a wealth of opportunities to enhance their knowledge and expertise, positioning themselves for success in an ever-evolving global marketplace.

Project module

1.1 Instructor Module:

1. **Course Management:**
    - **Create Course:** Instructors can create new courses, set prices, and upload course materials.
    - **Publish Course:** Instructors can make their courses available for students to enroll.
    - **Manage Content:** Instructors can upload videos, documents, and quizzes for their courses.
2. **Student Interaction:**
    - **Respond to Messages:** Instructors can answer student questions and provide support through messaging.
    - **Join Chat Rooms:** Instructors can engage with students in course-specific chat rooms.

1.2 Student Module:

1. **Course Catalog:**
    - **Browse Courses:** Students can explore available courses and add them to their cart.
    - **Enroll and Purchase:** Students can enroll in courses and make purchases securely.
2. **Learning Experience:**
    - **Access Course Content:** Students can access course materials, including videos and assignments.
    - **Community Interaction:** Students can participate in chat rooms to discuss course topics with peers.
3. **Support and Feedback:**
    - **Message Instructors:** Students can ask questions and provide feedback to instructors.

2. System Analysis

2.1 Existing System:

In the Existing system, while instructors possess the ability to create courses, a significant limitation exists in terms of student interaction. Notably, students face hurdles in addressing queries or seeking clarification directly from instructors or the broader community.

Disadvantages of Existing System:

1. **Limited Student Interaction:**
    - Students encounter obstacles in directly addressing queries or seeking clarification from instructors, leading to potential gaps in understanding course material.
    - The absence of a structured platform for student engagement deprives them of opportunities to interact with peers, exchange ideas, and collaborate on course-related topics.

2.2 Proposed System:

1. **Instructor-Student Messaging Feature:**
    - By integrating a dedicated messaging feature, students gain the ability to initiate direct communication with instructors. This empowers them to pose questions, seek guidance, and receive timely feedback, thereby enhancing their learning journey.
2. **Community Chat Integration:**
    - The proposed system will incorporate a robust community chat functionality, serving as a virtual forum for students to engage in discussions, share insights, and collaborate with peers. This interactive platform fosters a sense of community within the e-learning environment, facilitating knowledge exchange and collective problem-solving.

3. Requirement Specification

3.1 Hardware Specification:

Client Side:

RAM	512 MB
Hard disk	10 GB
Processor	1.0 GHz

Server Side:

RAM	**1 GB**
Hard disk	**20 GB**
Processor	**2.0 GHz**

3.2 Software Specification:

Client Side:

Web Browser	Google Chrome or any compatible browser
Operating System	Windows or any equivalent OS

Server Side:

Server side Language	Nodejs (Expressjs)
Database Server	Mongodb
Web Browser	Google Chrome or any compatible browser
Operating System	Windows or any equivalent OS

4. System Environment

4.1 Javascript

JavaScript is a dynamic programming language used for client-side web development. It powers interactive web applications by manipulating the DOM, handling events, and enabling asynchronous operations. JavaScript is also used for server-side development with platforms like Node.js. With a rich ecosystem of libraries and frameworks, JavaScript enables developers to create responsive and engaging web experiences efficiently.

4.1.1 Reactjs

ReactJS, created by Facebook, simplifies UI development with its:

- Declarative syntax for clear code.
- Component-based architecture for reusability.
- Virtual DOM for efficient updates.
- JSX syntax for blending HTML with JavaScript.
- React Hooks for state management in functional components.
- Robust ecosystem and supportive community.

4.1.2 Nodejs

Node.js is a runtime environment for executing JavaScript code outside the browser. features include:

- Asynchronous and event-driven architecture for efficient handling of I/O operations.
- Single-threaded, non-blocking nature conducive to building scalable and high-performance applications.
- Extensive ecosystem of packages/modules through npm, enabling rapid development of server-side applications.
- Widely used for building web servers, RESTful APIs, real-time applications, and microservices.

4.1.3 Expressjs

press.js is a minimalist web framework for Node.js. Its features include:

- Simplified routing for handling HTTP requests and defining endpoints.

- Middleware support for processing requests, enabling tasks like logging, authentication, and error handling.
- Templating engines integration for rendering dynamic content.
- Ideal for building APIs, web applications, and server-side applications with Node.js.

4.2 Mongodb

MonoDB is a NoSQL database known for its flexibility and scalability. It stores data in flexible, JSON-like documents, making it easy to work with. MongoDB is widely used for handling large volumes of data, real-time analytics, and powering modern applications requiring flexible data models.

4.3 Postman

Postman is a popular API testing and development tool used by developers worldwide. It simplifies the process of testing APIs by providing a user-friendly interface for sending HTTP requests, inspecting responses, and debugging APIs. With features like automated testing, request history, and collection management, Postman streamlines API development workflows and facilitates collaboration among team members.

5. System Design

5.1 Introduction

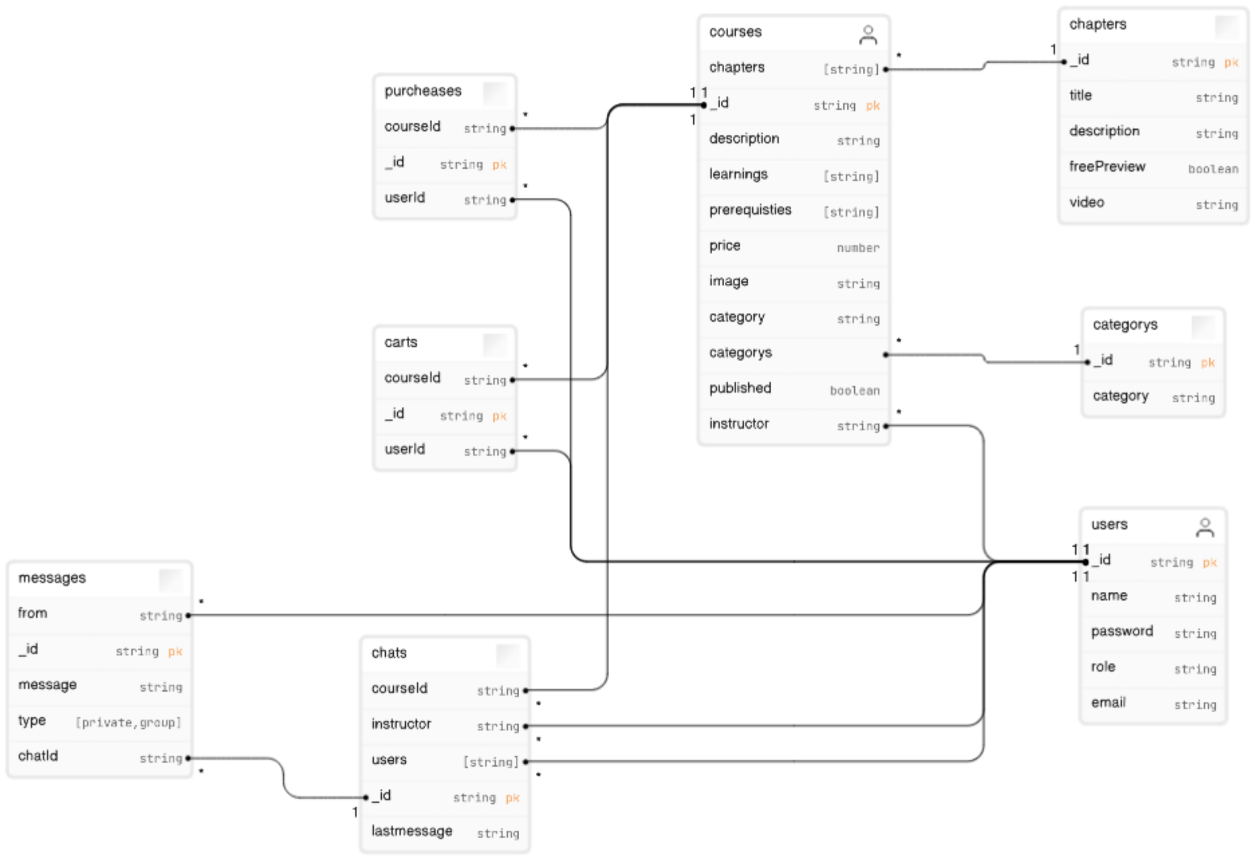
Design is the first step in the development phase for any techniquesand principles for the purpose of defining a device, a process or system in sufficient detail to permit its physical realization.

Once the software requirements have been analyzed and specified the software design involves three technical activities - design, coding, implementation and testing that are required to build and verify thesoftware.The design activities are of main importance in this phase, because in this activity, decisions ultimately affecting the success of the software implementation and its ease of maintenance are made. These decisions have the final bearing upon reliability and maintainability of the system.

Design is the only way to accurately translate the customer’s requirements into finished software or a system. Design is the place where quality is fostered in development. Software design is a process through which requirements are translated into a representation of software. Software design is conducted in two steps. Preliminary design is concerned with the transformation of requirements into data.

5.2 Class Diagram:

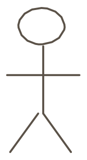
A description of set of objects that share the same attributes operations, relationships, and semantics



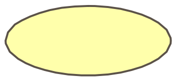
5.3 UML Diagrams:

Actor:

A coherent set of roles that users of use cases play when interacting with the use `cases.

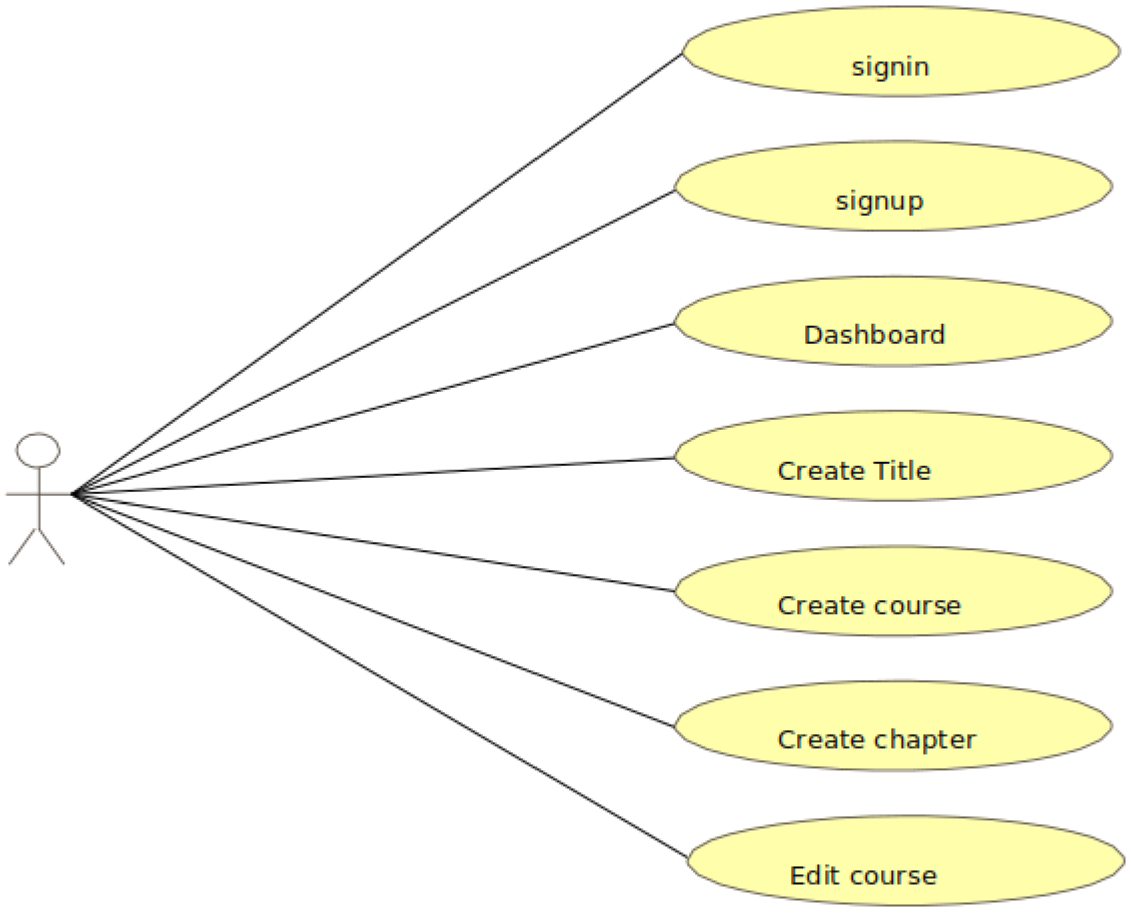


Use case:A description of sequence of actions, including variants, that a system performs that yields an observable result of value of an actor.

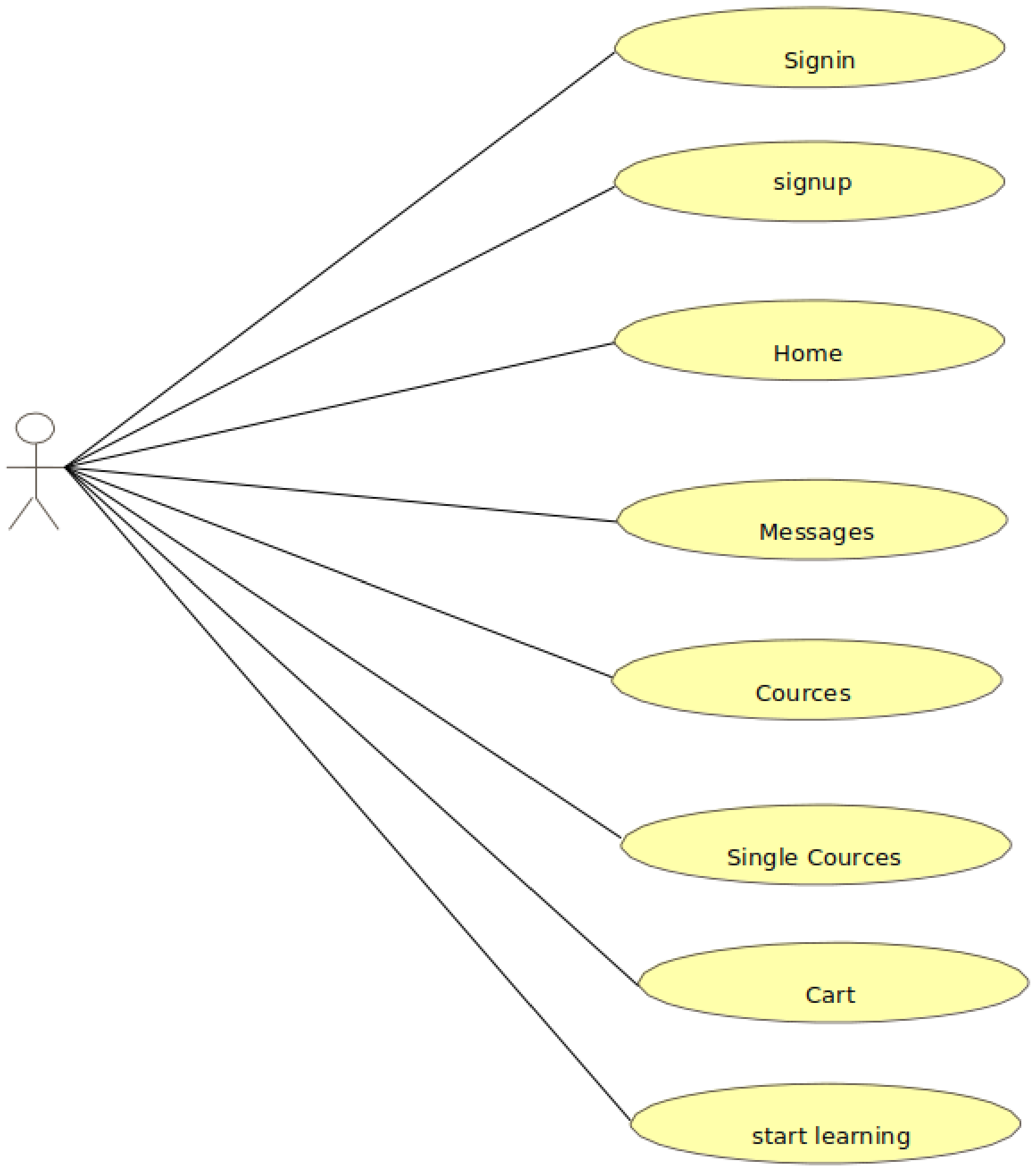


UML stands for Unified Modeling Language. UML is a language for specifying, visualizing and documenting the system. This is the step while developing any product after analysis. The goal from this is to produce a model of the entities involved in the project which later need to be built. The representation of the entities that are to be used in the product being developed need to be designed.

**Instr
ucto
r**



Student


Signin
signup
Home
Messages
Cources
Single Cources
Cart
start learning

6. Database Design

6.1 Collections

```
 const userSchema = new mongoose.Schema(
    {
        name: {
            type: String,
            required: [true, "Please enter your name."],
        },
        email: {
            type: String,
            required: [true, "Please enter an email."],
            unique: true,
            lowercase: true,
            validate: [validator.isEmail, "Please enter a valid email."],
        },

        password: {
            type: String,
            required: [true, "Please enter a password."],
            minlength: 8,
        },
        role: {
            type: String,
            enum: ["user", "instructor"],
            default: "user",
        },
    },
    { timestamps: true },
);
```

```
const CourseSchema = new mongoose.Schema(
  {
    title: {
      type: String,
      required: [true, "Please enter the course title."],
    },
    description: {
      type: String,
      trim:true
    },

    learnings: {
      type: [String]
    },
    prerequisties: {
      type: [String],

    },
    chapters: {
      type:[{type:mongoose.Types.ObjectId, ref:"Chapter"}],

    },
    price: {
      type:Number,
    },
    image:{
      type:String,
      trim:true
    },
        category:{
            type: mongoose.Types.ObjectId,
            ref:"Category"

        },
        instructor: {
            type: mongoose.Types.ObjectId,
            ref:"user"

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

    },
    { timestamps: true },
);

const cartSchema = new mongoose.Schema({
    userId: {
        type: mongoose.Schema.Types.ObjectId,
        ref:"User",
        required: [true, 'Please provide a userId.']
    },
    courseId: {
        type: mongoose.Schema.Types.ObjectId,
        ref:"Course",
        required: [true, 'Please provide a courseid.']
    },



},{timestamps:true})
```

```
const CategoryModel = new mongoose.Schema(
   {
      category: {
         type: String,
         required: [true, "Please enter the category."],
      },



   },
   { timestamps: true },
);

const ChapterSchema = new mongoose.Schema(
   {
      title: {
         type: String,
         required: [true, "Please enter the course title."],
      },
      description: {
         type: String,
         trim:true
      },
      freePreview: {
         type:Boolean,
         default: false,
      },
      video: {
         type:String,
         trim: true,
      }
```

```
    },
    { timestamps: true },
);

const PurcheaseSchema = new mongoose.Schema({
    userId: {
        type: mongoose.Schema.Types.ObjectId,
        ref:"User",
        required: [true, 'Please provide a userId.']
    },
    courseId: {
        type: mongoose.Schema.Types.ObjectId,
        ref:"Course",
        required: [true, 'Please provide a courseid.']
    },



},{timestamps:true})
const ChatSchema = new mongoose.Schema({
    instructor: {
        type: mongoose.Schema.Types.ObjectId,
        ref:"User",
        required: [true, 'Please provide a userId.']
    },
    courseId: {
        type: mongoose.Schema.Types.ObjectId,
        ref:"Course",
        required: [true, 'Please provide a courseid.']
    },
    users: {
       type:[ {type: mongoose.Schema.Types.ObjectId,
        ref:"User"}],
        default:[]
    },
    lastmessage: {
        type:String,
        trim:true,

    }
},{timestamps:true})

const MessageSchema = new mongoose.Schema({
    from: {
        type: mongoose.Schema.Types.ObjectId,
        ref:"User",
        required: [true, 'Please provide a userId.']
    },
    chatId: {
        type: mongoose.Schema.Types.ObjectId,
        ref:"Chat",
        required: [true, 'Please provide a chat id.']
    },
    message: {
        type:String,
        trim:true
    },
    type:{
        type:String,
        enum:["private","group"],
        default:"private"
    }



},{timestamps:true})
```

7. System Testing

Upon completion of each development phase, the system undergoes a rigorous testing regimen to ensure its robustness, reliability, and compliance with requirements. The testing process is iterative, with each stage designed to uncover and rectify potential defects before deployment.

7.1 Unit Testing

Unit testing focuses on isolating individual components or modules within the system and subjecting them to meticulous scrutiny. The primary goal is to verify that each unit performs its intended function accurately and reliably. Here's a more detailed breakdown of unit testing:

- **Test Case Design**: Unit tests are created based on detailed specifications and requirements for each module. Test cases encompass various scenarios, including normal operation, boundary conditions, and exceptional cases.

- **Mocking and Stubs**: In cases where a module relies on external dependencies, such as databases or external APIs, mocks or stubs may be used to simulate these dependencies during testing. This ensures that tests remain focused on the functionality of the unit under test.

- **Code Coverage Analysis**: Test coverage metrics are employed to assess the extent to which the codebase is exercised by unit tests. Adequate code coverage helps identify areas of the code that may require additional testing or refinement.

- **Regression Testing**: As the project evolves, changes to code can inadvertently introduce new defects or impact existing functionality. Regression testing involves re-executing previously successful tests to ensure that recent modifications have not adversely affected the system's behavior.

7.2 Integration Testing

Integration testing evaluates the interactions between different modules or subsystems within the system. It aims to validate the seamless integration of components and ensure that they function harmoniously as a unified whole. Here's a more comprehensive overview of integration testing:

- **Top-Down and Bottom-Up Approaches**: Integration testing can be performed using various strategies, including top-down and bottom-up approaches. Top-down integration involves testing higher-level components first, while bottom-up integration focuses on testing lower-level components before gradually integrating upwards.

- **Interface Compatibility**: Integration tests verify that module interfaces adhere to specified protocols and facilitate seamless communication and data exchange. Compatibility checks ensure that inputs and outputs are correctly interpreted and processed across interconnected modules.

- **Concurrency and Parallelism**: In systems with concurrent or parallel execution, integration testing assesses how modules interact under concurrent conditions. This includes evaluating synchronization mechanisms, data consistency, and deadlock prevention strategies.

- **System Interoperability**: Integration testing extends beyond individual modules to assess interoperability between disparate systems or external interfaces. Compatibility with third-party services, data formats, and communication protocols is verified to ensure seamless integration into the broader ecosystem.

By conducting thorough unit and integration testing, software development teams can identify and address defects early in the development lifecycle, mitigating risks and ensuring the delivery of a high-quality, reliable system to end-users.

8. Source Code

Frontend

client/src/Main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import "./index.css";
import { UserContext } from "@/context/userContext.jsx";
import { ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
ReactDOM.createRoot(document.getElementById("root")).render(
  <UserContext>
    <ToastContainer theme="dark" />
    <App />
  </UserContext>
);
```

client/src/app.jsx

```
import { RouterProvider, createBrowserRouter } from "react-router-dom";
import { ThemeProvider } from "./components/theme-provider";
import RootLayout from "./layouts/RootLayout";
import AuthLayout from "./layouts/AuthLayout";
import MessageLayout from "./layouts/MessageLayout";
import Home from "./pages/Home";
import Cart from "./pages/Cart";
import Courses from "./pages/Courses";
import StartLearning from "./pages/Start-Learning";
import Login from "./pages/Login";
import Signup from "./pages/Signup";
import SingleCourse from "./pages/Single-course";
import WatchVideos from "./pages/Watch-videos";
import Message from "./pages/Messages";
import InstructorHome from "./pages/Instructor-home";
import CreateCourseTitle from "./pages/Instructor-course-title";
import CreateCourse from "./pages/Instructor-create-course";
import CreateChapter from "./pages/Instructor-create-chapter";
import Payment from "./pages/Payment";
import { useEffect } from "react";
import { useUserContext } from "./context/userContext";
import { verify } from "./api/api";
import ConvertInstructor from "./pages/Convert-instructor";
import ProtectedRoute from "./components/ProtectedRoute";
import ProtectInstructorRoute from "./components/ProtectInstructorRoute";
// import ProtectedRoute from "./components/ProtectedRoute";
function App() {
  const { user, setUser } = useUserContext();
  useEffect(() => {
    const verifyUser = async () => {
      console.log("user verifyed from app.jsx");
      try {
        const data = await verify();
        setUser(data);
      } catch (error) {
        console.log(error);
      }
    };
    if (!user) verifyUser();
  }, []);
  const router = createBrowserRouter([
    {
      path: "/",
      element: <RootLayout />,
      children: [
        {
          path: "/",
          element: <Home />,
        },
        {
          path: "/carts",
          element: (
            <ProtectedRoute>
              <Cart />
            </ProtectedRoute>
          ),
        },
        {
          path: "/start-learning",
          element: (
            <ProtectedRoute>
              <StartLearning />
            </ProtectedRoute>
          ),
        },
        {
          path: "/videos/:id",
          element: (
            <ProtectedRoute>
              <WatchVideos />
            </ProtectedRoute>
          ),
        },
        {
          path: "/courses",


          children: [
            {
              path: "",
                  element: <Courses />,
                },
                {
                  path: "payment/:id",
                  element: (
                    <ProtectedRoute>
                      {" "}
                      <Payment />
                    </ProtectedRoute>
                  ),
                },
                {
                  path: ":id",
                  element: <SingleCourse />,
                },
              ],
            },
            {
              path: "/convert-instructor",
              element: (
                <ProtectedRoute>
                  {" "}
                  <ConvertInstructor />
                </ProtectedRoute>
              ),
            },


            {
              path: "/instructor",
              element: (
                <ProtectInstructorRoute>
                  <InstructorHome />
              </ProtectInstructorRoute>
          ),
},
{
    path: "/instructor/payment",
    element: <Payment />,
},
{
    path: "/instructor/create-course-title",
    element: (
        <ProtectInstructorRoute>
            {" "}
            <CreateCourseTitle />
        </ProtectInstructorRoute>
    ),
},
{
    path: "/instructor/create-course/:id",
    element: (
        <ProtectInstructorRoute>
            <CreateCourse />
        </ProtectInstructorRoute>
    ),
},
{
    path: "/instructor/create-chapter/:id",
    element: (
        <ProtectInstructorRoute>
            <CreateChapter />
        </ProtectInstructorRoute>
    ),
},
              {
                  path: "/auth",
                  element: <AuthLayout />,
                  children: [
                      {
                          path: "login",
                          element: <Login />,
                      },
                      {
                          path: "signup",
                          element: <Signup />,
                      },
                  ],
              },
          ],
},
{
    path: "/messages",
    element: <MessageLayout />,
    children: [
        {
            path: "",
            element: (
                <ProtectedRoute>
                    {" "}
                    <Message />
                </ProtectedRoute>
            ),
        },
    ],
},
        ]);
        return (
          <session>
            <ThemeProvider defaultTheme="dark" storageKey="vite-ui-theme">
              <RouterProvider router={router} />
            </ThemeProvider>
          </session>
        );
}
export default App;
```

client/src/pages/login/index.jsx

```
import React, { useState } from "react";
import { Input } from "@/components/ui/input";
import {
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Button } from "@/components/ui/button";
import { Link, useNavigate } from "react-router-dom";
import { useUserContext } from "@/context/userContext";
import { toast } from "react-toastify";
import { signin } from "@/api/api";
const Login = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const { setUser } = useUserContext();
  const navigate = useNavigate();
  const handleSubmit = async (e) => {
    e.preventDefault();
    console.log("working fine");
    if ((!name, !email || !password)) return toast("Fill all the field");
    else if (password.length < 8)
      return toast("Password must have minimum 8 characters.");
    let id;
    try {
      id = toast.loading("Please wait...");
      const userData = await signin({ email, password });
      toast.update(id, {
        render: "All is good",
        type: "success",
        isLoading: false,
        autoClose: true,
        closeOnClick: true,
      });
      setUser(userData);
      navigate("/");
    } catch (error) {
      toast.update(id, {
        render: "Try again later",
        type: "error",
        isLoading: false,
        autoClose: true,
        closeOnClick: true,
      });
      console.log(error);
    }
  };

  return (
    <div className="h-full  w-full grid place-items-center">
      <div>
        <CardHeader>
```

```
          <CardTitle>Login</CardTitle>
        </CardHeader>
        <CardContent>
          <form className="space-y-5 w-[400px]">
            <Input
              type="email"
              placeholder="Email"
              onChange={(e) => setEmail(e.target.value)}
            />
            <Input
              type="password"
              placeholder="Password"
              onChange={(e) => setPassword(e.target.value)}
            />
          </form>
        </CardContent>
        <CardFooter className="flex flex-col items-start">
          <Button onClick={handleSubmit}>Login</Button>

          <CardDescription className="mt-5">
            Don't have an account{" "}
            <Link to="/auth/signup" className="underline">
              create account
            </Link>
          </CardDescription>
        </CardFooter>
      </div>
    </div>
  );
};

export default Login;
```

client/src/pages/signup/index.jsx

```
import React, { useState } from "react";
import { Input } from "@/components/ui/input";
import {
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Button } from "@/components/ui/button";
import { Link, useNavigate } from "react-router-dom";
import { toast } from "react-toastify";

import { signup } from "../../api/api";
import { useUserContext } from "@/context/userContext";

const Signup = () => {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  const { setUser } = useUserContext();
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();

    console.log("working fine");
    if ((!name, !email || !password)) return toast("Fill all the field");
    else if (password.length < 8)
      return toast("Password must have minimum 8 characters.");
    let id;
    try {
      id = toast.loading("Please wait...");
      const userData = await signup({ name, email, password });
      toast.update(id, {
          render: "All is good",
          type: "success",
          isLoading: false,
          autoClose: true,
          closeOnClick: true,
      });
      setUser(userData);
      navigate("/");
    } catch (error) {
      toast.update(id, {
          render: "Try again later",
          type: "error",
          isLoading: false,
          autoClose: true,
          closeOnClick: true,
      });
      console.log(error);
    }
  };

  return (
    <div className="h-full  w-full grid place-items-center">
      <div>
          <CardHeader>
              <CardTitle>Signup</CardTitle>
          </CardHeader>
          <CardContent>
              <form className="space-y-5 w-[400px]">
                  <Input
                      type="text"
                        placeholder="User name"
                        onChange={(e) => setName(e.target.value)}
                    />
                    <Input
                        type="email"
                        placeholder="Email"
                        onChange={(e) => setEmail(e.target.value)}
                    />
                    <Input
                        type="password"
                        placeholder="Password"
                        onChange={(e) => setPassword(e.target.value)}
                    />
                </form>
            </CardContent>
            <CardFooter className="flex flex-col items-start">
                <Button onClick={handleSubmit}>Signup</Button>
                <CardDescription className="mt-5">
                    Alredy have an account{" "}
                    <Link to="/auth/login" className="underline">
                        login
                    </Link>
                </CardDescription>
            </CardFooter>
        </div>
    </div>
  );
};

export default Signup;
```

client/src/pages/home/index.jsx

```
import React, { useEffect, useState } from "react";
import HeroBanner from "./HeroBanner";
import Features from "./Features";
import PopularCourses from "./PopularCourses";
import FooterBanner from "./FooterBanner";
import { getAllCourses } from "@/api/api";

const Home = () => {
  const [courses, setCourses] = useState([]);
  useEffect(() => {
    const getCourses = async () => {
      try {
        const data = await getAllCourses(3);
        console.log(data);
        setCourses(data.courses);
      } catch (error) {
        console.log(error);
      }
    };
    getCourses();
  }, []);

  return (
    <div>
      <HeroBanner />


      <Features />
      <PopularCourses courses={courses} />
      <FooterBanner />
    </div>
  );
};
export default Home;
```

client/src/pages/courses/index.jsx

```
import React, { useEffect, useState } from "react";
import { Button } from "@/components/ui/button";
import {
   Card,
   CardContent,
   CardDescription,
   CardFooter,
   CardHeader,
   CardTitle,
} from "@/components/ui/card";

import { badgeVariants } from "@/components/ui/badge";

import Searchbar from "./Searchbar";
import { Link, useNavigate, useSearchParams } from "react-router-dom";
import { useUserContext } from "@/context/userContext";
import { Book } from "lucide-react";
import { getAllCategorys, getAllCourses } from "@/api/api";
import PaginationButton from "./PaginationButton";

const Courses = () => {
   const [categorys, setCategorys] = useState([]);
   const [courses, setCourses] = useState([]);
   const [totalPages, setTotalPages] = useState(1);

   const [searchParams, setSearchParams] = useSearchParams();
   const navigate = useNavigate();
   const { user } = useUserContext();

   const cat = searchParams.get("cat");
   const pg = searchParams.get("page");

   const addtoCart = (e) => {
      e.stopPropagation();
```

```
};
const getCourses = async (limit, page, category) => {
    try {
        const data = await getAllCourses(limit, page, category);
        setCourses(data.courses);
        setTotalPages(data.totalPages);
    } catch (error) {
        console.log(error);
    }
};
useEffect(() => {
    if (cat);
    getCourses(null, pg, cat);
}, [cat]);

useEffect(() => {
    if (pg);
    getCourses(1, pg);
}, [pg]);

useEffect(() => {
    const getCategorys = async () => {
        try {
            const data = await getAllCategorys();
            setCategorys(data);
        } catch (error) {
            console.log(error);
        }
    };
    getCategorys();
}, []);

useEffect(() => {
    getCourses(10);
  }, []);

  return (
    <div>
      <Searchbar />
      <div className="flex justify-between items-center mt-10">
        {categorys.map((data) => (
          <Link
            to={`?cat=${data._id}`}
            className={badgeVariants({ variant: "" })}
          >
            {data.category}
          </Link>
        ))}
      </div>
      <div className="grid grid-cols-3 gap-5 my-10">
        {courses.map((course) => (
          <Card onClick={() => navigate(course._id.toString())}>
            <CardHeader>
              <CardTitle className="line-clamp-1">
                {course.title}
              </CardTitle>
            </CardHeader>

            <CardContent>
              <img
                src={course.image}
                alt="banner"
                className="w-full  h-[250px] object-cover"
              />
              <br />
              <CardDescription className="line-clamp-2">
```

```
                    {course.description}
                  </CardDescription>
                </CardContent>

                <CardFooter className="flex flex-col items-start ">
                  <div className="flex justify-center items-center mb-2 text-sm text-
muted-foreground">
                    <Book className="h-5 w-5" /> &nbsp;
                    {course.chapters.length} chapters
                  </div>
                  <b>$ {course.price}</b>
                </CardFooter>
              </Card>
            ))}
          </div>
          <div className="mb-5">
            {!cat && (
              <PaginationButton
                totalPages={totalPages}
                setSearchParams={setSearchParams}
                currentPage={pg}
              />
            )}
          </div>
        </div>
  );
};
export default Courses;
```

client/src/pages/carts/index.jsx

```
import React, { useEffect, useState } from "react";
import {
  Card,
  CardContent,
    CardDescription,
    CardFooter,
    CardHeader,
    CardTitle,
} from "@/components/ui/card";
import { useNavigate } from "react-router-dom";
import { useUserContext } from "@/context/userContext";
import { Book } from "lucide-react";
import { Button } from "@/components/ui/button";
import { getCartItems } from "@/api/api";
const Cart = () => {
    const [courses, setCourses] = useState([]);
    const navigate = useNavigate();
    const { user } = useUserContext();
    useEffect(() => {
        const getCartCourses = async () => {
            try {
                const data = await getCartItems();

                setCourses(data);
            } catch (error) {
                console.log(error);
            }
        };
        getCartCourses();
    }, []);
    return (
        <div className="grid grid-cols-3 gap-5 my-10 min-h-[70vh]">
            {courses &&
                courses.map((course) => (
                    <Card
                        onClick={() =>
                            navigate(
                    "/courses/" + course.courseId._id.toString()
                  )
                }
                className="h-fit"
              >
                <CardHeader>
                  <CardTitle className="line-clamp-1">
                    {course.courseId.title}
                  </CardTitle>
                </CardHeader>

                <CardContent>
                  <img
                    src={course.courseId.image}
                    alt="banner"
                    className="w-full  h-[250px] object-cover"
                  />
                  <br />
                  <CardDescription className="line-clamp-2">
                    {course.courseId.description}
                  </CardDescription>
                </CardContent>
                <CardFooter className="flex flex-col items-start ">
                  <div className="flex justify-center items-center mb-2 text-sm text-
muted-foreground">
                    <Book className="h-5 w-5" /> &nbsp;5 chapters
                  </div>
                  <b>$ 40</b>
                  {/* {user && (
                    <Button onClick={() => {}}>Add To Cart</Button>
                  )} */}
                </CardFooter>
              </Card>
```

```
        ))}
      </div>
    );
  };

  export default Cart;
```

client/src/pages/start-learning/index.jsx

```
import React, { useEffect, useState } from "react";
import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle,
  CardFooter,
} from "@/components/ui/card";
import { useNavigate } from "react-router-dom";
import { Progress } from "@/components/ui/progress";
import { Book } from "lucide-react";
import { getAllEnrolledCourses } from "@/api/api";
const StartLearning = () => {
  const [courses, setCourses] = useState([]);

  const navigate = useNavigate();

  useEffect(() => {
    const getCourses = async () => {
      try {
        const data = await getAllEnrolledCourses();
        console.log(data);
        setCourses(data);
      } catch (error) {
        console.log(error);
```

```
          }
        };
        getCourses();
    }, []);
    return (
        <div>
            <div className="grid grid-cols-3 gap-5 my-10">
                {courses.map((course) => (
                    <Card
                        onClick={() =>
                            navigate(
                                "/videos/" + course.courseId._id.toString()
                            )
                        }
                    >
                        <CardHeader>
                            <CardTitle className="line-clamp-1">
                                {course.courseId.title}
                            </CardTitle>
                        </CardHeader>

                        <CardContent>
                            <img
                                src={course.courseId.image}
                                alt="banner"
                                className="w-full"
                            />
                            <br />
                            <CardDescription className="line-clamp-2">
                                {course.courseId.description}
                            </CardDescription>
                        </CardContent>
                        <CardFooter className="flex flex-col items-start text-sm">
                        <div className="flex justify-center items-center mb-3 text-sm text-
muted-foreground">
                          <Book className="h-5 w-5" /> &nbsp;
                          {course.courseId.chapters.length} chapters
                        </div>
                        <Progress
                          value={course.progress}
                          className="h-2 mb-3"
                        />
                        {course.progress} % Complete
                      </CardFooter>
                    </Card>
                  ))}
                </div>
              </div>
  );
};

export default StartLearning;
```

client/src/pages/messages/index.jsx

```
import React, { useEffect, useState } from "react";
import Sidebar from "./Sidebar";
import Messages from "./Messages";

import io from "socket.io-client";
import { useUserContext } from "@/context/userContext";
import { useSearchParams } from "react-router-dom";
import { MessageCircle } from "lucide-react";
import { getAllMessages, getChats, sendMessage } from "@/api/api";

const ENDPOINT = "http://localhost:5000";
var socket, selectedChatCompare;
const Message = () => {
```

```
const [chats, setChats] = useState([]);
const [messages, setMessages] = useState([]);
const [users, setUsers] = useState([]);
const [socketConnected, setSocketConnected] = useState(false);

const [searchParams, setSearchParams] = useSearchParams();
const search = searchParams.get("search");
const type = searchParams.get("type");

const { user } = useUserContext();

const handleSend = async (message) => {
   try {
      const res = await sendMessage(search, { message, type });
      socket.emit("new message", {
         message: res,
         users,
      });

      setMessages((prev) => [...prev, res]);
   } catch (error) {
      console.log(error);
   }
};

const connectSocket = () => {
   console.log(user);
   socket = io(ENDPOINT);
   socket.emit("setup", user._id);
   socket.on("connected", () => {
      console.log("connected successfully");
      setSocketConnected(true);
   });
};
const getAllChats = async () => {
   try {
      const data = await getChats();

      setChats(data);
   } catch (error) {
      console.log(error);
   }
};

useEffect(() => {
   const getMessages = async () => {
      try {
         const data = await getAllMessages(search, type);

         setUsers(data.users);
         setMessages(data.messages);
      } catch (error) {
         console.log("error occured");
         console.log(error);
      }

      socket.emit("join chat", { roomId: search + type });
   };
   if (search && type) getMessages();
}, [search, type]);

useEffect(() => {
   if (user && user?._id) connectSocket();
   getAllChats();
}, []);
```

```
// const setMsg = (msg) => {
//     console.log("**************");
// if (msg._id !== messages[messages.length - 1]._id) {
//         console.log(msg);
//         console.log(messages);

//         console.log("**************");
//     }
// };
useEffect(() => {
    if (socket)
        socket.on("message recieved", (message) => {
            if (message.type == type) {
                console.log(message);
                if (message._id !== messages[messages.length - 1]._id)
                    setMessages((prev) => [...prev, message]);
                console.log(messages);
            }
        });
}, [socket]);

return (
    <div className="flex h-[80vh] mb-5  ">
        <div className="w-1/3">
            <Sidebar chats={chats} />
        </div>
        <div className="w-2/3">
            {search == null ? (
                <div className="h-full flex justify-center flex-col items-center gap-5">
                    <MessageCircle className="w-[10rem] h-[10rem]" />
                    Select a chat
                </div>
            ) : (
          <Messages
            search={search}
            type={type}
            messages={messages}
            setMessages={setMessages}
            user={user}
            handleSend={handleSend}
          />
        )}
      </div>
    </div>
  );
};

export default Message;
```

client/src/pages/instructorHome/index.jsx

```
import { Button } from "@/components/ui/button";
import React, { useEffect, useState } from "react";
import { Link } from "react-router-dom";
import {
  Table,
  TableBody,
  TableCaption,
  TableCell,
  TableFooter,
  TableHead,
  TableHeader,
  TableRow,
} from "@/components/ui/table";
import { Link2 } from "lucide-react";
import { getAllCoursesForInstructor } from "@/api/api";

const AdminHome = () => {
```

```
const [courses, setCourses] = useState([]);

useEffect(() => {
  const getAllCourses = async () => {
    try {
      const res = await getAllCoursesForInstructor();
      setCourses(res);
    } catch (error) {
      console.log(error);
    }
  };
  getAllCourses();
}, []);

return (
  <div className="min-h-[70vh]">
    <div className="flex justify-between items-center">
      <h1 className="text-2xl font-semibold leading-none tracking-tight">
        Hello 'Subin'
      </h1>
      <Link to="create-course-title">
        <Button variant="outline">Create Course</Button>
      </Link>
    </div>
    <Table>
      <TableCaption>A list of your courses.</TableCaption>
      <TableHeader>
        <TableRow>
          <TableHead className="w-[100px]">Status</TableHead>
          <TableHead>Category</TableHead>
          <TableHead>name</TableHead>
          <TableHead className="text-right">Price</TableHead>
        </TableRow>
```

```
            </TableHeader>
            <TableBody>
              {courses?.map((course) => (
                <TableRow key={course.name}>
                  <TableCell className="font-medium">
                    {course.published
                      ? "Published"
                      : "Not Published"}
                  </TableCell>
                  <TableCell>
                    {course?.category?.category
                      ? course?.category?.category
                      : "-"}
                  </TableCell>
                  <TableCell>{course.title}</TableCell>
                  <TableCell className="text-right">
                    {course?.price ? course?.price : "-"}
                  </TableCell>
                  <TableCell className="text-right">
                    <Link
                      to={`/instructor/create-course/${course._id}`}
                    >
                      <Link2 />
                    </Link>
                  </TableCell>
                </TableRow>
              ))}
            </TableBody>
          </Table>
        </div>
    );
  };
```

```
export default AdminHome;
```

client/src/pages/convert to instructor//index.jsx

```
import { Button } from "@/components/ui/button";
import React from "react";
import { Link } from "react-router-dom";

const ConvertInstructor = () => {
  return (
    <div className="h-svh items-center justify-center flex">
      <Button asChild>
        <Link to="/instructor/payment">Convert to instructor</Link>
      </Button>
    </div>
  );
};

export default ConvertInstructor;
```

client/src/pages/instructor-create-course/index.jsx

```
import { LayoutDashboard } from "lucide-react";
import React, { useEffect, useState } from "react";

import CourseImage from "./Course-image";
import CategoryForm from "./Category-form";
import LearningForm from "./Learning-form";
import PrerequistiesForm from "./Prerequisties-form";
import CourseChapter from "./Course-chapter";
import { Button } from "@/components/ui/button";
import CoursePrice from "./Course-price";
import DescriptionForm from "@/components/Description-form";
import TitleForm from "@/components/Title-form";
import { useNavigate, useParams } from "react-router-dom";
import { getSingleCourse, updateCourseDetails } from "@/api/api";
import { toast } from "react-toastify";
```

```
const CreateCourse = () => {
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [image, setImage] = useState("");
  const [category, setCategory] = useState("");
  const [learning, setLearning] = useState([]);
  const [prerequisties, setPrerequisties] = useState([]);
  const [chapters, setChapters] = useState([]);
  const [price, setPrice] = useState(0);
  const { id } = useParams();
  const navigate = useNavigate();
  const publishCourse = async () => {
    try {
      const data = await updateCourseDetails(id, { published: true });
      navigate("/instructor");

      toast("Course published successfully");
    } catch (error) {
      console.log(error);
    }
  };
  useEffect(() => {
    const getCourse = async (id) => {
      try {
        const course = await getSingleCourse(id);
        console.log(course);
        setTitle(course.title);
        setDescription(course?.description || "");
        setImage(course?.image || "");
        setCategory(course?.category?.category || "");
        setLearning(course.learnings);
        setPrerequisties(course.prerequisties);
```

```
            setChapters(course.chapters);
            setPrice(course?.price || 0);
        } catch (error) {
            console.log(error);
        }
    };

    getCourse(id);
}, []);
return (
    <div className=" my-5">
        <div className="flex justify-between">
            <div>
                <h1 className="text-2xl font-semibold leading-none tracking-tight">
                    Course setup
                </h1>
                <span className="text-sm text-muted-foreground">
                    Complete all fields (1 / 5)
                </span>
            </div>
            <Button onClick={publishCourse}>Publish</Button>
        </div>
        <div className="flex justify-between  mt-5 gap-5">
            <div className="  flex-1 space-y-5">
                <span className="flex gap-2">
                    <LayoutDashboard />
                    <h2 className="text-xl font-medium leading-none tracking-tight">
                        Customize you course
                    </h2>
                </span>
                <TitleForm
                    heading={"Course title"}
                    courseTitle={title}
                    setCourseTitle={setTitle}
                    id={id}
                />
                <DescriptionForm
                    heading={"Course description"}
                    courseDescription={description}
                    setCourseDescription={setDescription}
                    id={id}
                />
                <LearningForm
                    courseLearning={learning}
                    setCourseLearning={setLearning}
                    id={id}
                />
                <PrerequistiesForm
                    prerequisties={prerequisties}
                    setPrerequisties={setPrerequisties}
                    id={id}
                />

                <CourseImage
                    courseImage={image}
                    setCourseImage={setImage}
                    id={id}
                />
                <CategoryForm
                    courseCategory={category}
                    setCourseCategory={setCategory}
                    id={id}
                />
            </div>
            <div className=" flex-1 space-y-5">
                <CourseChapter
```

```
                    chapters={chapters}
                    setChapters={setChapters}
                    id={id}
                  />
                  <CoursePrice
                    coursePrice={price}
                    setCoursePrice={setPrice}
                    id={id}
                  />
                </div>
              </div>
            </div>
          );
        };

        export default CreateCourse;
```

server/server.js

```
const dotenv = require('dotenv');
dotenv.config();
const mongoose = require('mongoose');
const app = require('./app');
mongoose.connect(process.env.CONN_STR).then(() => {
  console.log('DB Connection Successful');
}).catch(err => {
  console.error('Error connecting to database:', err);
  process.exit(1);
});
const port = process.env.PORT || 5000;
const server = app.listen(port, () => {
  console.log('server has started...');
})
```

```
const io = require("socket.io")(server,{
    pingTimeout: 60000,
    cors:{
        origin: "http://localhost:5173",
    }
})

io.on("connection",(socket)=>{
    console.log("connected to socket.io");
    socket.on("setup",(userid)=>{
        console.log(userid);
        socket.join(userid);
        socket.emit("connected")
    })

    socket.on("join chat",(room)=>{
        socket.join(room.roomId);
        console.log("user joined room",room.roomId);
    })

    socket.on("new message", (newMessageRecived) => {
      console.log(newMessageRecived)
      if(!newMessageRecived.users) return console.log("chat users is not defined");

      newMessageRecived.users.forEach((user)=>{
       if(newMessageRecived.message.from == user) return;
       socket.in(user).emit("message recieved",newMessageRecived.message);
      })
    })

})
```

server/index.js

```
const express = require("express");
const cookieParser = require("cookie-parser");
var cors = require("cors");

const userRouter = require("./Routes/UserRouter");
 const courseRouter = require("./Routes/CourseRouter");
const cartRouter = require("./Routes/CartRouter");
const categoryRouter = require("./Routes/CategoryRouter");
const chapterRouter = require("./Routes/ChapterRouter");
const messageRouter = require("./Routes/MessageRouter");
const purcheaseRouter = require("./Routes/PurcheaseRouter");
const chatsRouter = require("./Routes/ChatRouter");
const instructorRouter = require("./Routes/InstructorRouter");

const CustomError = require("./Utils/CustomError");
const globalErrorHandler = require("./Controllers/errorController");

let app = express();

app.use(
  cors({
    origin: "http://localhost:5173",
    credentials: true,
  })
);
app.use(express.json());
app.use(cookieParser());

//USING ROUTES

app.use("/api/v1/users", userRouter);
 app.use("/api/v1/courses", courseRouter);
app.use('/api/v1/cart',cartRouter)
app.use('/api/v1/categorys',categoryRouter)
app.use('/api/v1/chapters',chapterRouter)
app.use('/api/v1/messages',messageRouter)
app.use('/api/v1/purcheases',purcheaseRouter)
app.use('/api/v1/chats',chatsRouter)

app.use("/api/v1/instructor", instructorRouter);

app.all("*", (req, res, next) => {
  res.status(404).json({
    status: "fail",
    message: `Can't find ${req.originalUrl} on the server!`,
  });
  // const err = new Error(`Can't find ${req.originalUrl} on the server!`);
  // err.status = 'fail';
  // err.statusCode = 404;
  const error = new CustomError(
    `Can't find ${req.originalUrl} on the server!`,
    404
  );
  next(err);
});

app.use(globalErrorHandler);

module.exports = app;
```

server/Routes/cartRouter.js

```
const express = require('express');
const messageController = require("../Controllers/MessageController")
const verify = require("../Utils/Verifytoken");
const router = express.Router();

router.route("/:chatId").get(verify.verifyToken,messageController.getAllMessages);
router.route("/send/:chatId").post(verify.verifyToken,messageController.sendMessage);

module.exports = router
```

server/Routes/CategoryRouter.js

```
const express = require('express');
const categoryController = require("../Controllers/CategoryController")
const verify = require("../Utils/Verifytoken");


const router = express.Router();


router.route("/").get(categoryController.getAllCategorys).post(categoryController.createCat
egory)

module.exports = router
```

server/Routes/ChapterRouter.js

```
const express = require('express');
const chapterController = require("../Controllers/ChapterController")
const verify = require("../Utils/Verifytoken");


const router = express.Router();



// router.route("/:courseId").get(verify.verifyToken,chapterController.getAllChapters)
router.route("/:chapterId").get(chapterController.getSingleChapter);

module.exports = router
```

server/Routes/ChatRouter.js

```
const express = require('express');
const chatsController = require("../Controllers/ChatController")
const verify = require("../Utils/Verifytoken");


const router = express.Router();

router.route("/").get(verify.verifyToken,chatsController.getAllChats);


module.exports = router
```

server/Routes/CourseRouter.js

```
const express = require('express');
const courseController = require("../Controllers/CourseController")
const verify = require("../Utils/Verifytoken");


const router = express.Router();

router.route("/").get(courseController.getAllCourses);
router.route("/:id").get(courseController.getSinleCourse);
router.route("/:id/enroll").post(verify.verifyToken,courseController.enroll)
```

server/Routes/InstructorRouter.js

```
const express = require('express');
const instructorController = require("../Controllers/InstructorController")
const verify = require("../Utils/Verifytoken");


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

```
router.route("/").post(verify.verifyToken,instructorController.userToInstructor);
router.route("/courses").get(verify.verifyToken,instructorController.getAllCourses).post(verify.verifyToken,instructorController.createNewCourse)
router.route("/courses/:id").put(verify.verifyToken,instructorController.updateCourseDetails)
.delete(verify.verifyToken,instructorController.deleteCourse);
router.route("/courses/:id/chapters").post(verify.verifyToken,instructorController.addNewChapter)
router.route("/courses/:courseId/chapters/:chapterId").put(verify.verifyToken,instructorController.updateChapterDetails).delete(verify.verifyToken,instructorController.deleteChapter);

//
router.route("/courses/:courseId/students").get(verify.verifyToken,instructorController.getEnrolledStudents)




module.exports = router
```

server/Routes/MessageRouter.js

```
const express = require('express');
const messageController = require("../Controllers/MessageController")
const verify = require("../Utils/Verifytoken");

const router = express.Router();
router.route("/:chatId").get(verify.verifyToken,messageController.getAllMessages);
router.route("/send/:chatId").post(verify.verifyToken,messageController.sendMessage);
module.exports = router
```

server/Routes/PurcheaseRouter.js

```
const express = require('express');
const purcheaseController = require("../Controllers/PurcheaseController")
const verify = require("../Utils/Verifytoken");
const router = express.Router();
router.route("/").get(verify.verifyToken,purcheaseController.getAllPurcheases);
router.route("/:courseId").get(verify.verifyToken,purcheaseController.getSingleCourse);
module.exports = router
```

server/Routes/UserRouter.js

```
const express = require("express");
const userController = require("../Controllers/UserController");
const AsyncErrorHandler = require("../Utils/AsyncErrorHandler");
const verify = require("../Utils/Verifytoken");
const router = express.Router();
router.route("/signup").post(userController.signup);
router.route("/signin").post(userController.signin);
router.route("/signout").post(userController.signout);
router.route("/verify").get(verify.verifyToken, userController.verify);
module.exports = router;
```

server/Controllers/CartController.js

```
 const Cart = require("../Models/CartModel");
const asyncErrorHandler = require('../Utils/AsyncErrorHandler');
const CustomError = require('../Utils/CustomError');




exports.addToCart = asyncErrorHandler(async(req,res,next)=>{


    const isInCart = await Cart.findOne({userId:req.body.user.id, courseId:
req.body.courseid})


    if(isInCart){
        const error = new CustomError("User has already add cart this course", 400);
        return next(error);
    }



    const cart = await Cart.create({userId: req.body.user.id,courseId:req.body.courseid});

    res.status(201).json({
        status: 'success',
        data: {
            cart
        }
    });
})


exports.removeItem = asyncErrorHandler(async(req,res,next)=>{


    const isInCart = await Cart.findById(req.params.id)

    if(!isInCart){
        const error = new CustomError("Course is not in cart", 400);
        return next(error);
    }
    await Cart.findByIdAndDelete(req.params.id)

     res.status(200).json({
        status:"success",


    })
})

exports.getCartItems = asyncErrorHandler(async(req,res,next)=>{
    const cartItems = await Cart.find({userId:
req.body.user.id}).populate("courseId").select("image chapter title description price");
console.log(cartItems)
    res.status(200).json({
        status: "success",
        data: {
            cartItems
        }
    })
 })
```

server/Controllers/CategoryController.js

```
const Category = require("../Models/CategoryModel");
const asyncErrorHandler = require("../Utils/AsyncErrorHandler");

exports.createCategory = asyncErrorHandler(async(req,res,next)=>{



    const category = await Category.create({category: req.body.category});

    res.status(201).json({
        status: 'success',
        data: {
            category
        }
    });
})

exports.getAllCategorys = asyncErrorHandler(async(req,res,next)=>{



    const category = await Category.find();
    res.status(201).json({
        status: 'success',
        data: {
            category
        }
    });
})
```

server/Controllers/ChapterController.js

```
const Chapter = require("../Models/ChapterModel");
const asyncErrorHandler = require("../Utils/AsyncErrorHandler");

exports.getSingleChapter = asyncErrorHandler(async(req,res,next)=>{


    const {chapterId} = req.params;
    console.log(chapterId );


    const chapter = await Chapter.findById(chapterId);

    res.status(201).json({
        status: 'success',
        data: {
            chapter
        }
    });
})
```

server/Controllers/ChatController.js

```
const Chat = require("../Models/ChatModel")
const AsyncErrorHandler = require("../Utils/AsyncErrorHandler")
```

```
exports.getAllChats = AsyncErrorHandler(async(req,res,next)=>{


    const userId = req.body.user.id
    const chats = await Chat.find({
        $or: [
            { users: userId }, // Check if the user ID exists in the users array
            { instructor: userId } // Check if the user ID is the instructor
        ]
    }).populate({
        path:"courseId",
        select:"title image"
    }
    );

    res.status(200).json({
        status: "success",
        data: {
            chats
        }
    })
})
```

server/Controllers/CourseController.js

```
const  mongoose  = require("mongoose");
const Chat = require("../Models/ChatModel");
const Course = require("../Models/CourseModel")
const Purchease = require("../Models/PurcheasedCourse");
const asyncErrorHandler = require("../Utils/AsyncErrorHandler");
const CustomError = require("../Utils/CustomError");

const stripe = require("stripe")(process.env.STRIPE_KEY);

exports.getAllCourses = asyncErrorHandler(async (req, res, next) => {
```

```
let query = {published:true};
let page = req.query.page || 1; // Default page is 1
let limit = req.query.limit || 30; // Default limit per page

if (req.query.search) {
    const searchRegex = new RegExp(req.query.search, "i");
    query.$or = [
        { title: searchRegex },
        { description: searchRegex },


    ];
}

if (req.query.cat) {
    query.category = req.query.cat;
}
const count = await Course.countDocuments(query);


const totalPages = Math.ceil(count / limit);



page = Math.min(page, totalPages);


let skip = (page - 1) * limit ;
skip = skip < 0 ?0:skip


const courses = await Course.find(query).skip(skip).limit(limit);


res.status(200).json({
    status: "success",
    data: {
          courses,
          page,
          totalPages,
        },
    });
});

exports.getSinleCourse = asyncErrorHandler(async(req,res,next)=>{
 const courseId = req.params.id
const course = await Course.findById(courseId)
      .populate("category")
      .populate({
        path: "chapters",
        select: {
          title: true,
          video: { $cond: [{ $eq: ["$freePreview", true] }, "$video", "$$REMOVE"] }, //
Conditional selection based on freePreview
        },
      });



console.log(course);
   if(!course){
        const error = new CustomError("Course not found", 500);
        return next(error);
   }
   res.status(200).json({
    status: "success",
    data: {
        course


    },
});
})

exports.enroll = asyncErrorHandler(async(req,res,next)=>{
  const isPurcheased = await Purchease.findOne({userId:req.body.user.id,courseId:
req.params.id})

  if(isPurcheased){
    const error = new CustomError("User has already purcheased this course", 400);
    return next(error);
  }

  const course = await Course.findById(req.params.id).select("price");
  console.log(course);

  const paymentIntent = await stripe.paymentIntents.create({
    amount: course.price * 100,
    currency: "inr",
    // In the latest version of the API, specifying the `automatic_payment_methods`
parameter is optional because Stripe enables its functionality by default.
    automatic_payment_methods: {
      enabled: true,
    },
   });

  const purchease = await Purchease.create({userId:req.body.user.id,courseId:
req.params.id})

  await Chat.findOneAndUpdate(
    {courseId:req.params.id},
    {$addToSet: {users:req.body.user.id}},
    {new:true, upsert: true}
  )
```

```
    res.status(200).json({
        status: "success",
        data: {
            clientSecret: paymentIntent.client_secret,


        },
    })
})
```

server/Controllers/errorController.js

```
const CustomError = require('./../Utils/CustomError');

const devErrors = (res, error) => {
    res.status(error.statusCode).json({
        status: error.statusCode,
        message: error.message,
        stackTrace: error.stack,
        error: error
    });
}

const castErrorHandler = (err) => {
    const msg = `Invalid value for ${err.path}: ${err.value}!`
    return new CustomError(msg, 400);
}

const duplicateKeyErrorHandler = (err) => {
    console.log("dupli")
 const name = err.keyValue.name;
 const msg = `There is already a movie with name ${name}. Please use another name!`;
 return new CustomError(msg, 400);
}

const validationErrorHandler = (err) => {
   const errors = Object.values(err.errors).map(val => val.message);
   const errorMessages = errors.join('. ');
   const msg = `Invalid input data: ${errorMessages}`;

   return new CustomError(msg, 400);
}

const prodErrors = (res, error) => {
   if(error.isOperational){
         res.status(error.statusCode).json({
         status: error.statusCode,
         message: error.message
      });
   }else {
       res.status(500).json({
         status: 'error',
         message: 'Something went wrong! Please try again later.'
      })
   }
}

module.exports = (error, req, res, next) => {
   error.statusCode = error.statusCode || 500;
   error.status = error.status || 'error';


   if(process.env.NODE_ENV === 'development'){


      devErrors(res, error);
```

```
    } else if(process.env.NODE_ENV === 'production'){

        if(error.name === 'CastError') error = castErrorHandler(error);
        if(error.code === 11000) error = duplicateKeyErrorHandler(error);
        if(error.name === 'ValidationError') error = validationErrorHandler(error);

        prodErrors(res, error);
    }
}
```

server/Controllers/InstructorController.js

```
const Course = require("../Models/CourseModel");
const Chapter = require("../Models/ChapterModel");
const Chat = require("../Models/ChatModel");
const User = require("../Models/UserModel");
const asyncErrorHandler = require("../Utils/AsyncErrorHandler");
const Purchease = require("../Models/PurcheasedCourse");

const stripe = require("stripe")(process.env.STRIPE_KEY);


exports.getAllCourses = asyncErrorHandler(async (req, res, next) => {


    const instructor = req.body.user.id

    const data = await Course.find({instructor}).populate("category").select("title category
price published")

    res.status(201).json({
        status:"success",
        data:{
            data
        }
    })
})

exports.createNewCourse = asyncErrorHandler(async (req, res, next) => {
    const title = req.body.title;
    const instructor = req.body.user.id

    const data = await Course.create({title,instructor});

    const chat = await Chat.create({instructor,courseId:data._id})

    res.status(201).json({
        status:"success",
        data:{
            data
        }
    })
})

exports.updateCourseDetails = asyncErrorHandler(async(req,res,next)=>{
    const courseId = req.params.id


console.log(req.body)
    const data = await Course.findByIdAndUpdate(courseId,
req.body,{new:true}).populate("category");
    console.log(data);
    res.status(201).json({
        status:"success",
        data:{
            data
        }
    })
})
```

```
exports.deleteCourse = asyncErrorHandler(async(req,res,next)=>{
    const {id} = req.params;

    await Course.findByIdAndDelete(id);
    await Purchease.deleteMany({courseId:id})
    await Chat.deleteOne({courseId:id})



    res.status(200).json({
        status:"success",


    })
})



exports.addNewChapter = asyncErrorHandler(async(req,res,next)=>{
    const {title,description, freePreview,video} = req.body;
    const courseId = req.params.id
    const createdChapter = await Chapter.create({title,description,freePreview,video})

    await Course.findByIdAndUpdate(courseId,{$push:{chapters:createdChapter._id}})

    res.status(201).json({
        status:"success",
        data:{
            chapter:createdChapter
        }
    })
})
```

```
exports.updateChapterDetails = asyncErrorHandler(async(req,res,next)=>{
    const {chapterId, courseId} = req.params;
    console.log(chapterId, courseId);
    const chapter = await Chapter.findByIdAndUpdate(chapterId,req.body,{new:true})

    res.status(200).json({
        status:"success",
        data:{
            chapter
        }
    })
})


exports.deleteChapter = asyncErrorHandler(async(req,res,next)=>{
    const {chapterId, courseId} = req.params;

    await Chapter.findByIdAndDelete(chapterId);

    res.status(200).json({
        status:"success",


    })
})

exports.userToInstructor = asyncErrorHandler(async(req,res,next)=>{
    const user  = await
User.findOneAndUpdate({_id:req.body.user.id},{role:"instructor"},{new:true})
 const paymentIntent = await stripe.paymentIntents.create({
        amount: 1000 * 100,
        currency: "inr",
        automatic_payment_methods: {
          enabled: true,
        },
      });

     res.status(200).json({
        status: "success",
        data: {
            clientSecret: paymentIntent.client_secret,
            user
        },
    })
})
```

server/Controllers/MessageController.js

```
const Chat = require("../Models/ChatModel");
const Message = require("../Models/MessageModel");
const AsyncErrorHandler = require("../Utils/AsyncErrorHandler");

exports.sendMessage = AsyncErrorHandler(async(req,res,next)=>{

    const chatId = req.params.chatId;
    const from = req.body.user.id;
    const message = req.body.message;
    const type = req.body.type
    const to = req.body.to;

    let msgData = {};


    if(type == "private"){
       msgData =  {from,chatId, message, type,to}
    }else {
       msgData =  {from,chatId, message, type}
    }


    let  messagee = await Message.create(msgData)


    res.status(201).json({
        status:"success",
        data:{
            message: messagee
        }
    })
})

exports.getAllMessages = AsyncErrorHandler(async(req,res,next)=>{
    const chatId  = req.params.chatId;
    let chatUsers,users,messages=[];
    const type = req.query.type;
const to  = req.query.to



    if(type=="group"){
      messages = await Message.find({chatId,type})

     chatUsers = await Chat.findById(chatId).select("users instructor")

     users = [...chatUsers.users,chatUsers.instructor]

     console.log("if part worked");
    }else{
      messages = await Message.find({chatId,type,to})

        users = [to]
    }



    console.log(messages);
    console.log(users);

    res.status(200).json({
        status: "success",
        data: {
            messages,
            users
        }
    })
})
```

server/Controllers/PurcheaseController.js

```
const AsyncErrorHandler = require("../Utils/AsyncErrorHandler");
const Purchease = require("../Models/PurcheasedCourse")

exports.getAllPurcheases = AsyncErrorHandler(async(req,res,next)=>{



    const courses = await
Purchease.find({userId:req.body.user.id}).populate("courseId").select("title image
description chapters")

    res.status(200).json({
        status: "success",
        data: {
            courses
        }
    })
})

exports.getSingleCourse = AsyncErrorHandler(async(req,res,next)=>{

    const {courseId} = req.params;

    const courses = await Purchease.findOne({userId:req.body.user.id, courseId})
    .populate({path:"courseId",select:"title chapters",populate:"chapters"})

    res.status(200).json({
        status: "success",
        data: {
            courses
        }
    })
})
```

server/Controllers/UserController.js

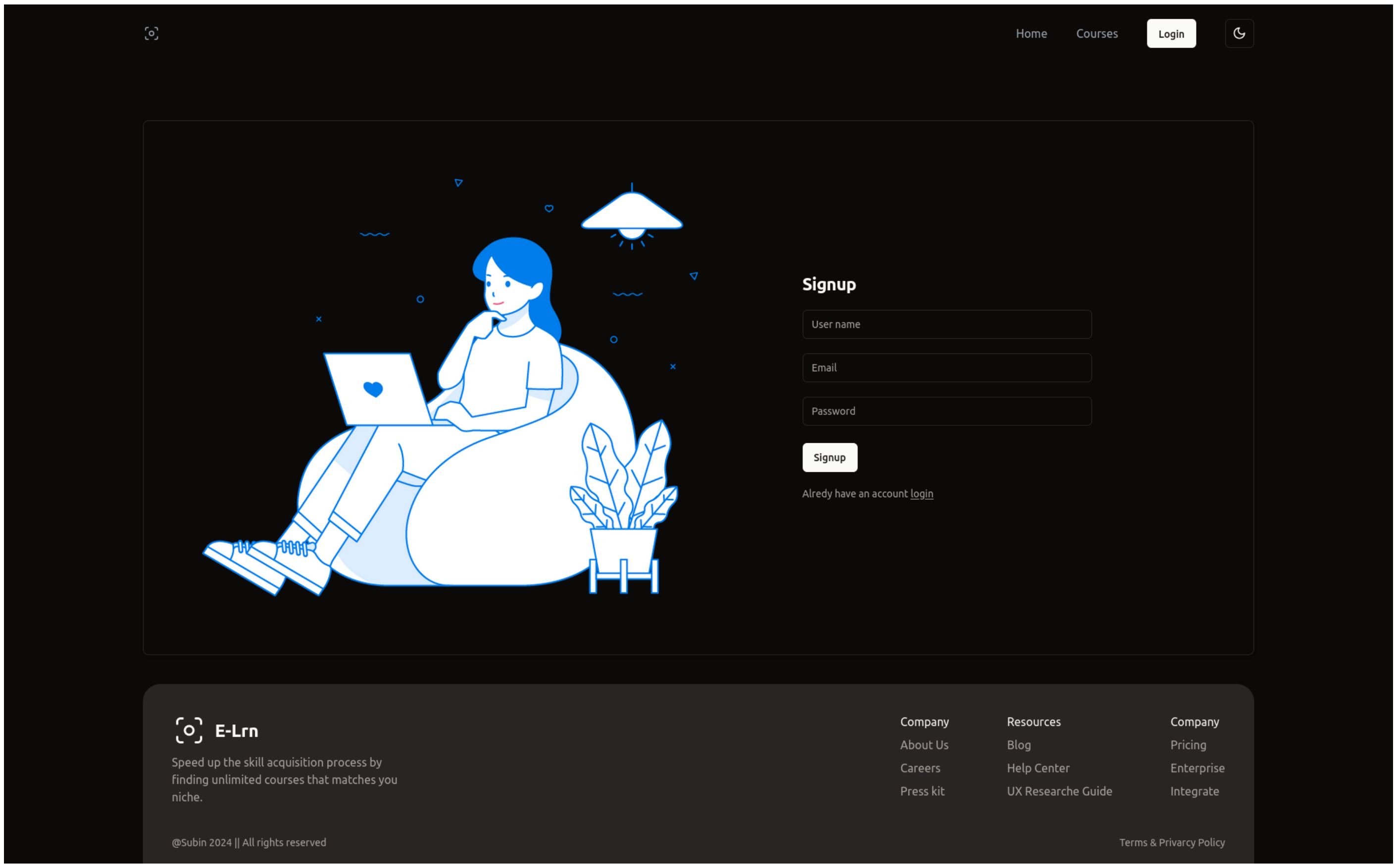
```
const jwt = require("jsonwebtoken");
const bcryptjs = require("bcryptjs");
const dotenv = require("dotenv");
dotenv.config();
const User = require("../Models/UserModel")

const asyncErrorHandler = require("../Utils/AsyncErrorHandler");
const CustomError = require("../Utils/CustomError");

exports.signup = asyncErrorHandler(async (req, res, next) => {
    const { email, name, password } = req.body;
    console.log(email, name, password);

    const hashed = await bcryptjs.hash(password, 8);
    const newUser = await User.create({ email, name, password: hashed });
    const token = jwt.sign(
        { id: newUser._id, role: newUser.role },
        process.env.JWT_SECRECT,
        {
            expiresIn: process.env.LOGIN_EXPIRES,
        },
    );

    newUser.password = undefined;
    var expirationDate = new Date();
    expirationDate.setDate(expirationDate.getDate() + 30);

    res.cookie("token", "bearer " + token, {
        expires: expirationDate,
    });

    res.status(201).json({
        status: "success",

        data: {
            user: newUser,
        },
    });
});

exports.signin = asyncErrorHandler(async (req, res, next) => {
```

```
const email = req.body.email;
const password = req.body.password;

if (!email || !password) {
   const error = new CustomError(
      "Please enter mail id and password for login",
      400,
   );
   return next(error);
}

const user = await User.findOne({ email });

if(!user){
   const error = new CustomError(
      "User not found",
      400,
   );
   return next(error);
}


const match = await bcryptjs.compare(password, user.password);

if (!match) {
   const error = new CustomError(
      "Please enter a correct email or password",
      400,
   );
   return next(error);
```

```
    }
    user.password = undefined;
    const token = jwt.sign(
        { id: user._id, role: user.role },
        process.env.JWT_SECRECT,
        {
            expiresIn: process.env.LOGIN_EXPIRES,
        },
    );

    var expirationDate = new Date();
    expirationDate.setDate(expirationDate.getDate() + 30);

    res.cookie("token", "bearer " + token, {
        expires: expirationDate,
    });

    res.status(201).json({
        status: "success",

        data: {
            user,
        },
    });
});



exports.signout = asyncErrorHandler(async(req,res,next)=>{
    res.clearCookie("token");

    res.status(200).json({
        status: "success",
        message: "User successfully signed out."
    });


})

exports.verify = asyncErrorHandler(async (req, res, next) => {
    const userId = req.body.user.id;

    console.log(userId);
    const user = await User.findById(userId).select("-password");
    res.status(200).json({
        status: "success",
        data: {
            user,
        },
    });
});
```

9.Output Layout

Signup Page:



Home Courses Login

Signup

User name

Email

Password

Signup

Alredy have an account login

E-Lrn

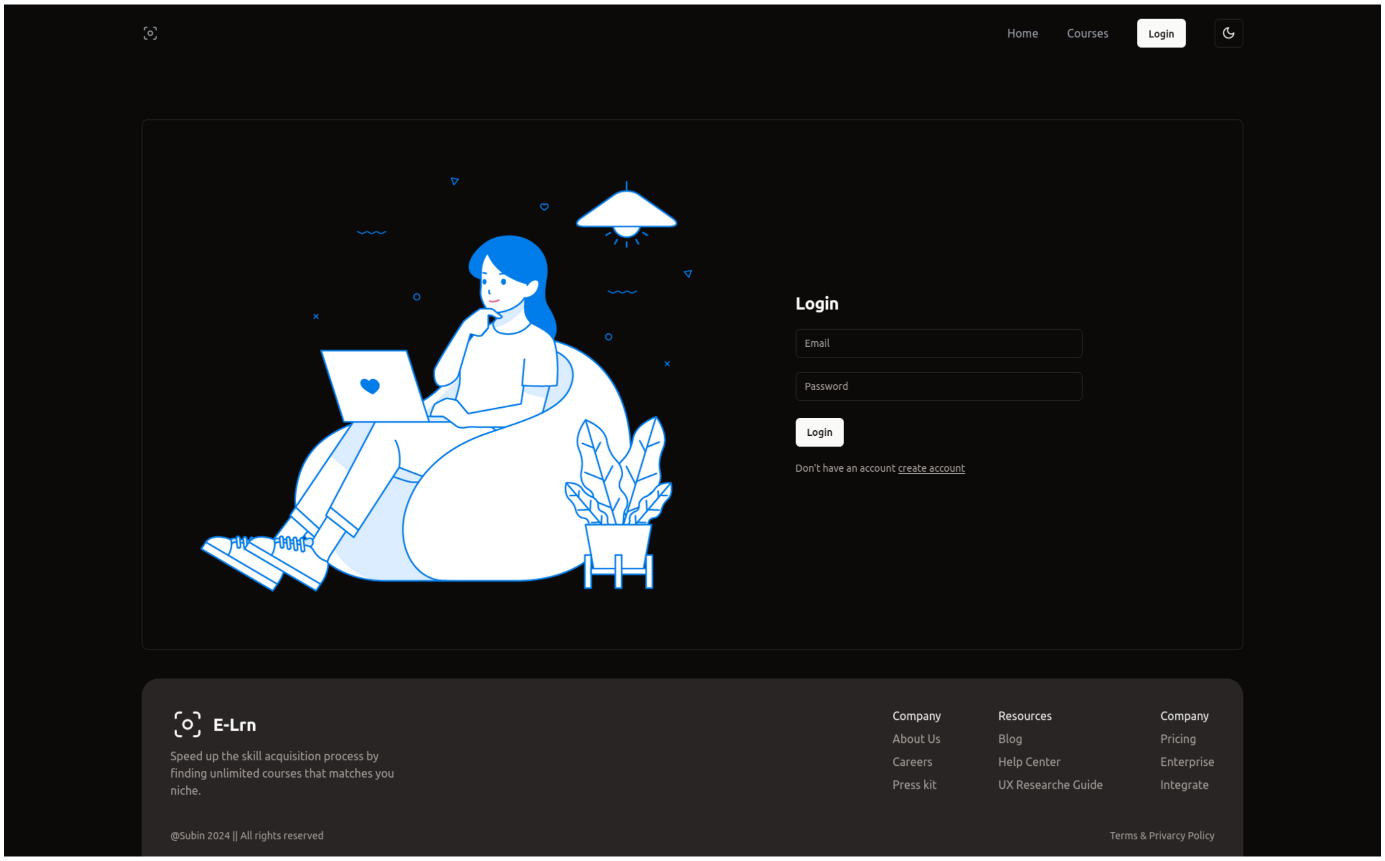
Speed up the skill acquisition process by finding unlimited courses that matches you niche.

Company	Resources	Company
About Us	Blog	Pricing
Careers	Help Center	Enterprise
Press kit	UX Researche Guide	Integrate

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Signin Page:



Login

Email

Password

Login

Don't have an account create account

E-Lrn

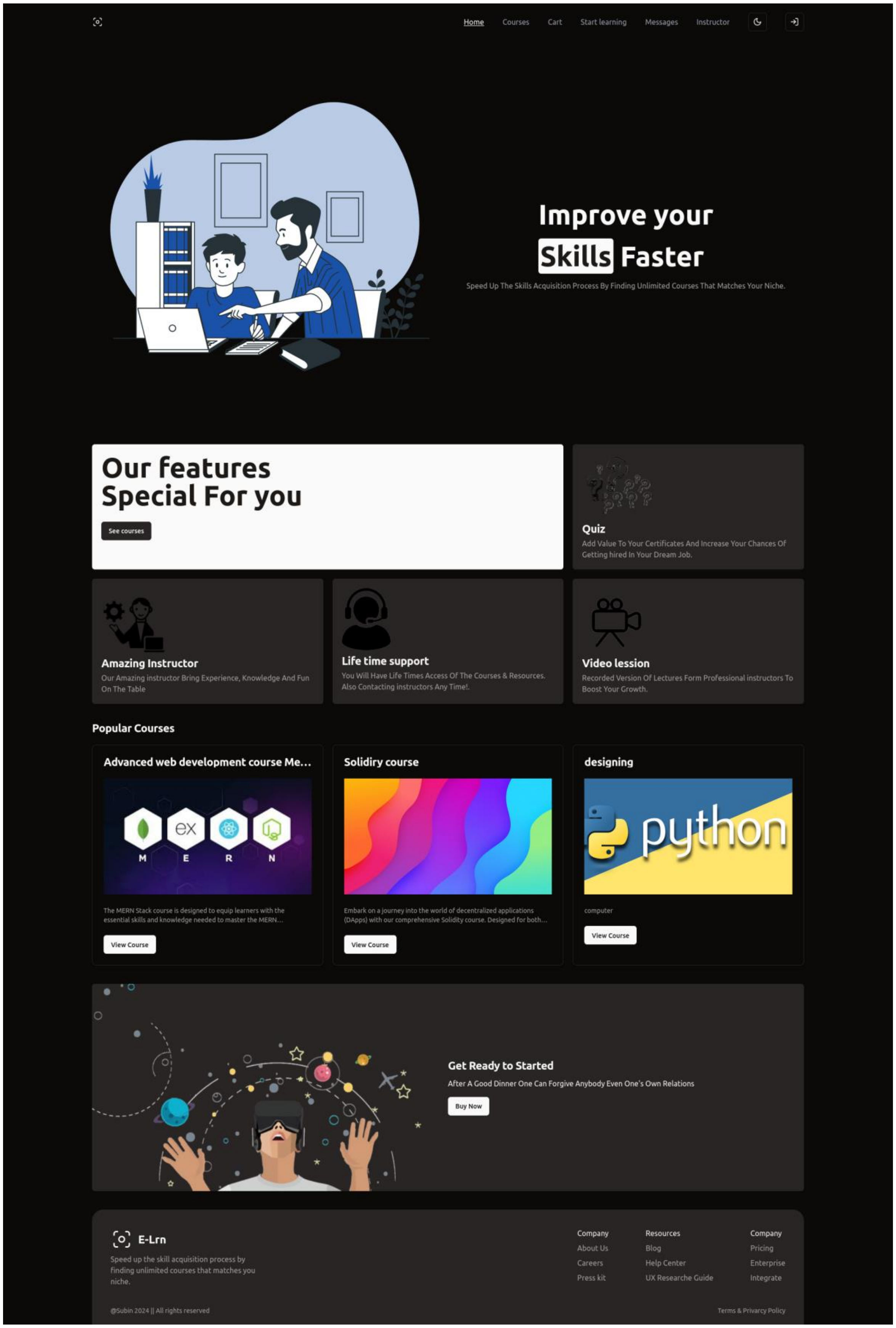
Speed up the skill acquisition process by finding unlimited courses that matches you niche.

Company	Resources	Company
About Us	Blog	Pricing
Careers	Help Center	Enterprise
Press kit	UX Researche Guide	Integrate

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Home Page



Improve your Skills Faster

Speed Up The Skills Acquisition Process By Finding Unlimited Courses That Matches Your Niche.

Our features Special For you

Quiz
Add Value To Your Certificates And Increase Your Chances Of Getting hired In Your Dream Job.

Amazing Instructor
Our Amazing instructor Bring Experience, Knowledge And Fun On The Table

Life time support
You Will Have Life Times Access Of The Courses & Resources. Also Contacting instructors Any Time!.

Video lession
Recorded Version Of Lectures Form Professional instructors To Boost Your Growth.

Popular Courses

Advanced web development course Me...
The MERN Stack course is designed to equip learners with the essential skills and knowledge needed to master the MERN...

Solidiry course
Embark on a journey into the world of decentralized applications (DApps) with our comprehensive Solidity course. Designed for both...

designing
computer

Get Ready to Started
After A Good Dinner One Can Forgive Anybody Even One's Own Relations

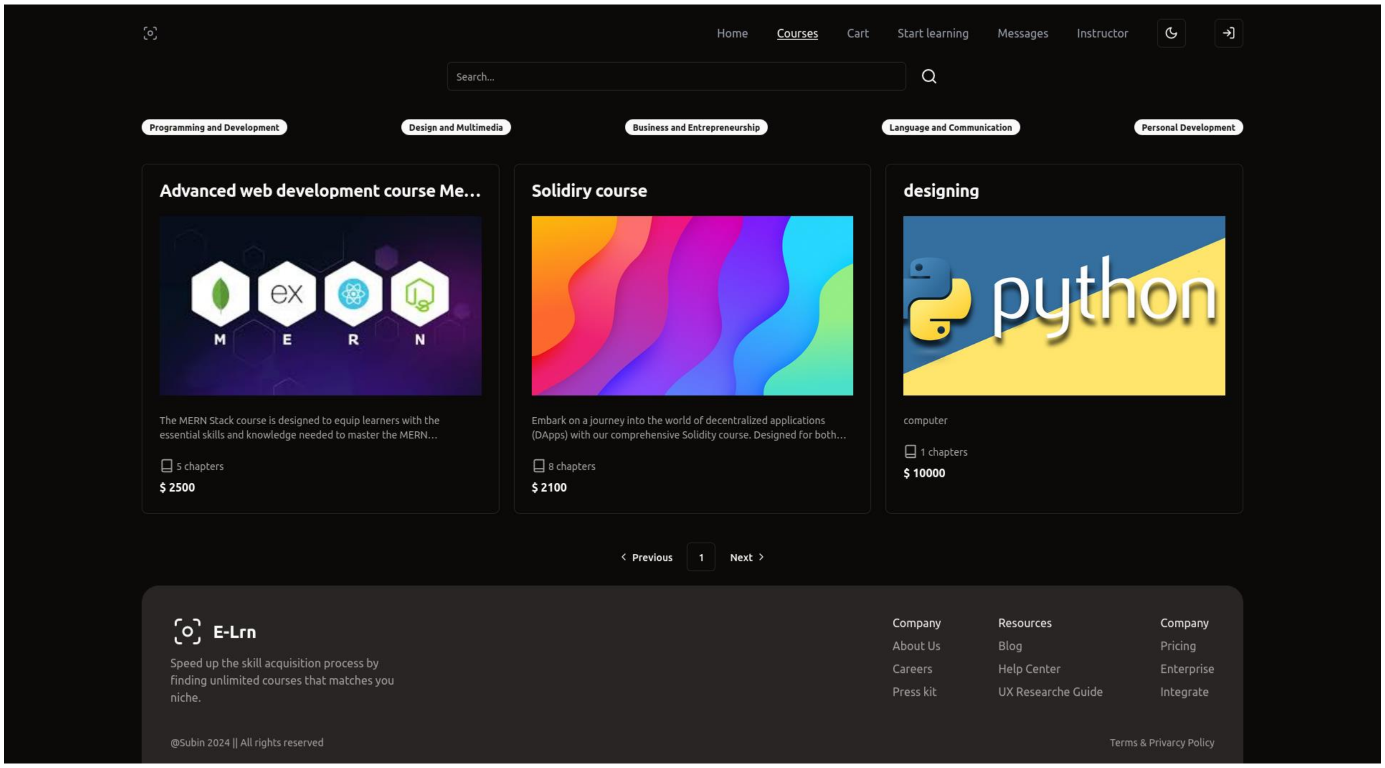
E-Lrn
Speed up the skill acquisition process by finding unlimited courses that matches you niche.

Company	Resources	Company
About Us	Blog	Pricing
Careers	Help Center	Enterprise
Press kit	UX Researche Guide	Integrate

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Courses Page



Advanced web development course Me...

The MERN Stack course is designed to equip learners with the essential skills and knowledge needed to master the MERN...

5 chapters

$ 2500

Solidiry course

Embark on a journey into the world of decentralized applications (DApps) with our comprehensive Solidity course. Designed for both...

8 chapters

$ 2100

designing

computer

1 chapters

$ 10000

E-Lrn

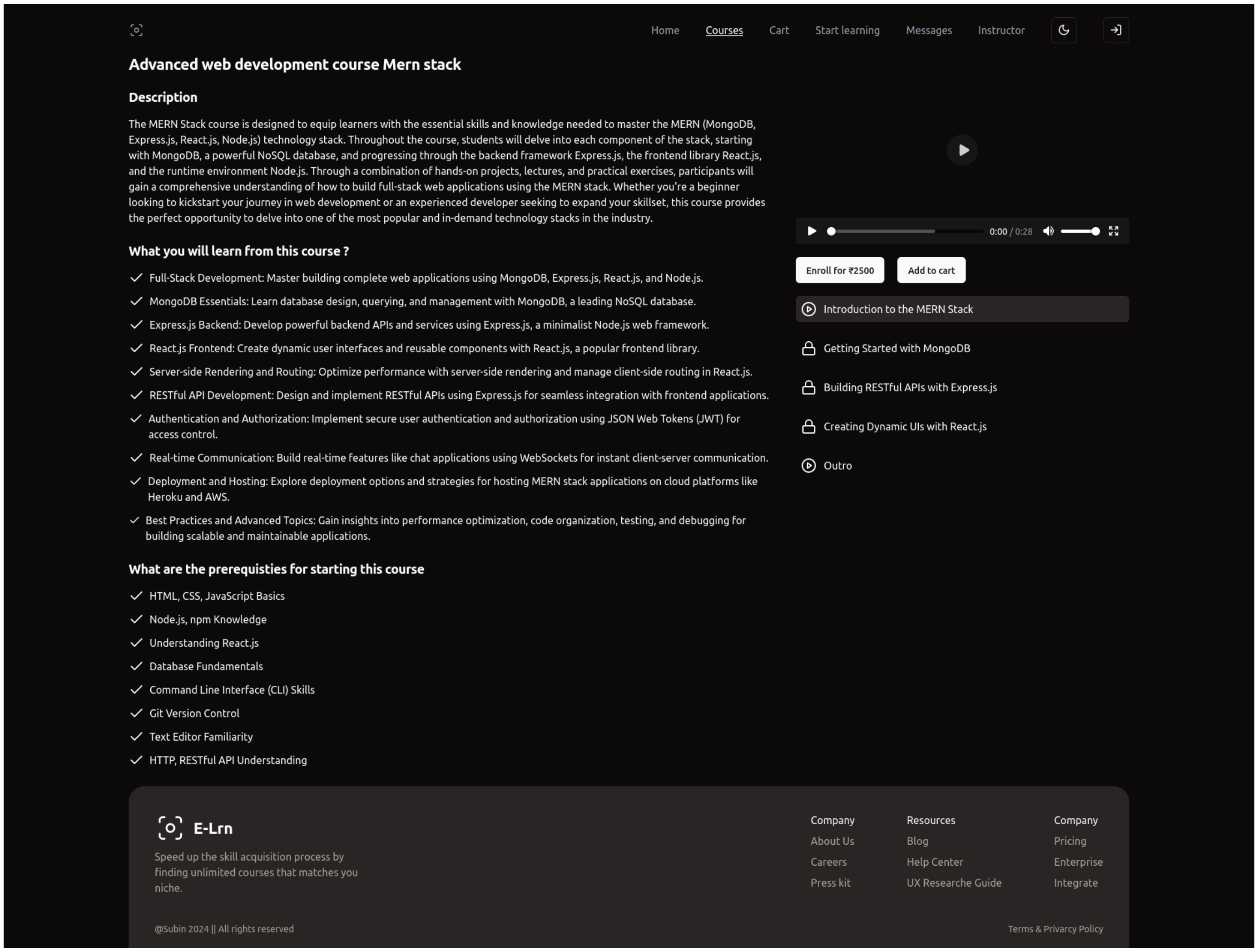
Speed up the skill acquisition process by finding unlimited courses that matches you niche.

Company	Resources	Company
About Us	Blog	Pricing
Careers	Help Center	Enterprise
Press kit	UX Researche Guide	Integrate

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Single Course Page



Advanced web development course Mern stack

Description

The MERN Stack course is designed to equip learners with the essential skills and knowledge needed to master the MERN (MongoDB, Express.js, React.js, Node.js) technology stack. Throughout the course, students will delve into each component of the stack, starting with MongoDB, a powerful NoSQL database, and progressing through the backend framework Express.js, the frontend library React.js, and the runtime environment Node.js. Through a combination of hands-on projects, lectures, and practical exercises, participants will gain a comprehensive understanding of how to build full-stack web applications using the MERN stack. Whether you're a beginner looking to kickstart your journey in web development or an experienced developer seeking to expand your skillset, this course provides the perfect opportunity to delve into one of the most popular and in-demand technology stacks in the industry.

What you will learn from this course ?

- ✓ Full-Stack Development: Master building complete web applications using MongoDB, Express.js, React.js, and Node.js.
- ✓ MongoDB Essentials: Learn database design, querying, and management with MongoDB, a leading NoSQL database.
- ✓ Express.js Backend: Develop powerful backend APIs and services using Express.js, a minimalist Node.js web framework.
- ✓ React.js Frontend: Create dynamic user interfaces and reusable components with React.js, a popular frontend library.
- ✓ Server-side Rendering and Routing: Optimize performance with server-side rendering and manage client-side routing in React.js.
- ✓ RESTful API Development: Design and implement RESTful APIs using Express.js for seamless integration with frontend applications.
- ✓ Authentication and Authorization: Implement secure user authentication and authorization using JSON Web Tokens (JWT) for access control.
- ✓ Real-time Communication: Build real-time features like chat applications using WebSockets for instant client-server communication.
- ✓ Deployment and Hosting: Explore deployment options and strategies for hosting MERN stack applications on cloud platforms like Heroku and AWS.
- ✓ Best Practices and Advanced Topics: Gain insights into performance optimization, code organization, testing, and debugging for building scalable and maintainable applications.

What are the prerequisties for starting this course

- ✓ HTML, CSS, JavaScript Basics
- ✓ Node.js, npm Knowledge
- ✓ Understanding React.js
- ✓ Database Fundamentals
- ✓ Command Line Interface (CLI) Skills
- ✓ Git Version Control
- ✓ Text Editor Familiarity
- ✓ HTTP, RESTful API Understanding

- Introduction to the MERN Stack
- Getting Started with MongoDB
- Building RESTful APIs with Express.js
- Creating Dynamic UIs with React.js
- Outro

E-Lrn

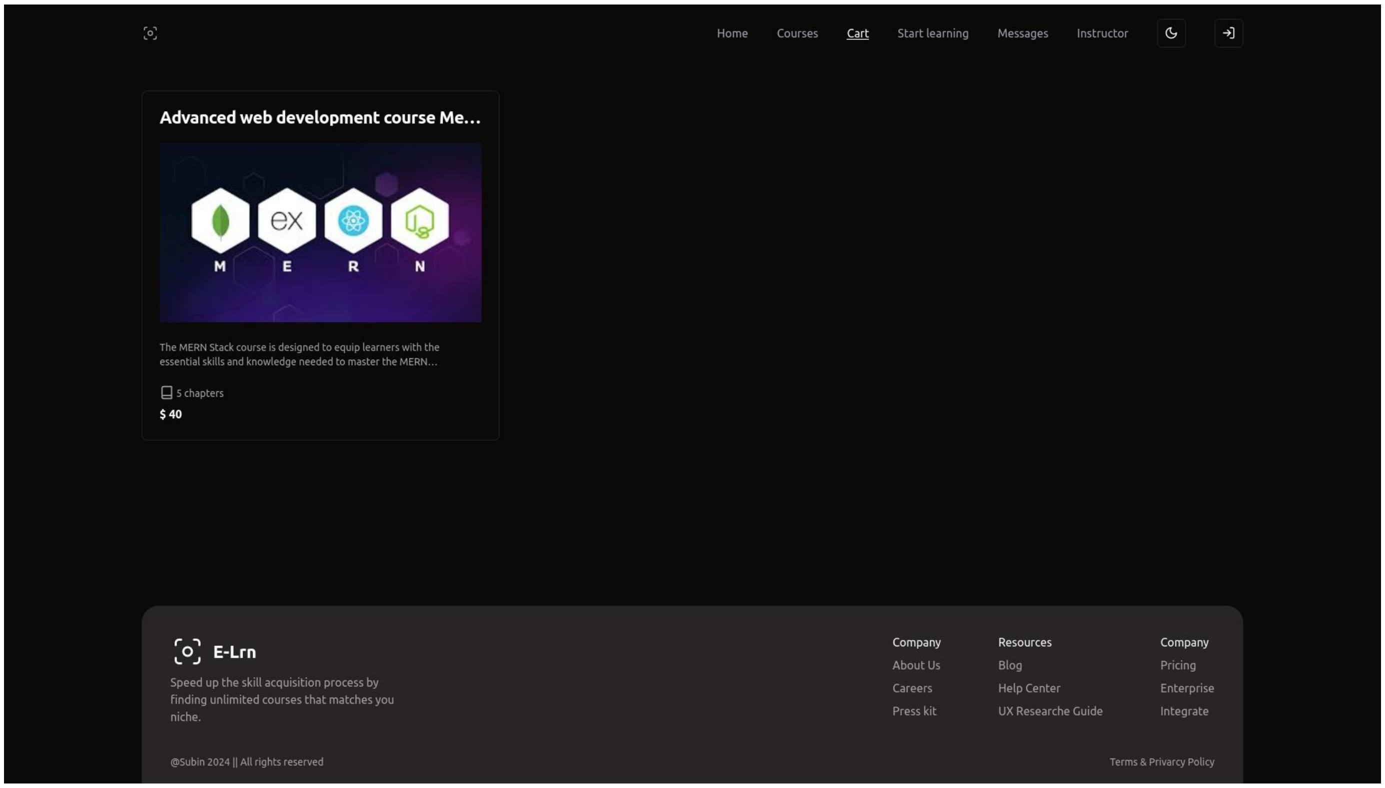
Speed up the skill acquisition process by finding unlimited courses that matches you niche.

Company	Resources	Company
About Us	Blog	Pricing
Careers	Help Center	Enterprise
Press kit	UX Researche Guide	Integrate

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Carts



Advanced web development course Me...

The MERN Stack course is designed to equip learners with the essential skills and knowledge needed to master the MERN...

5 chapters

$ 40

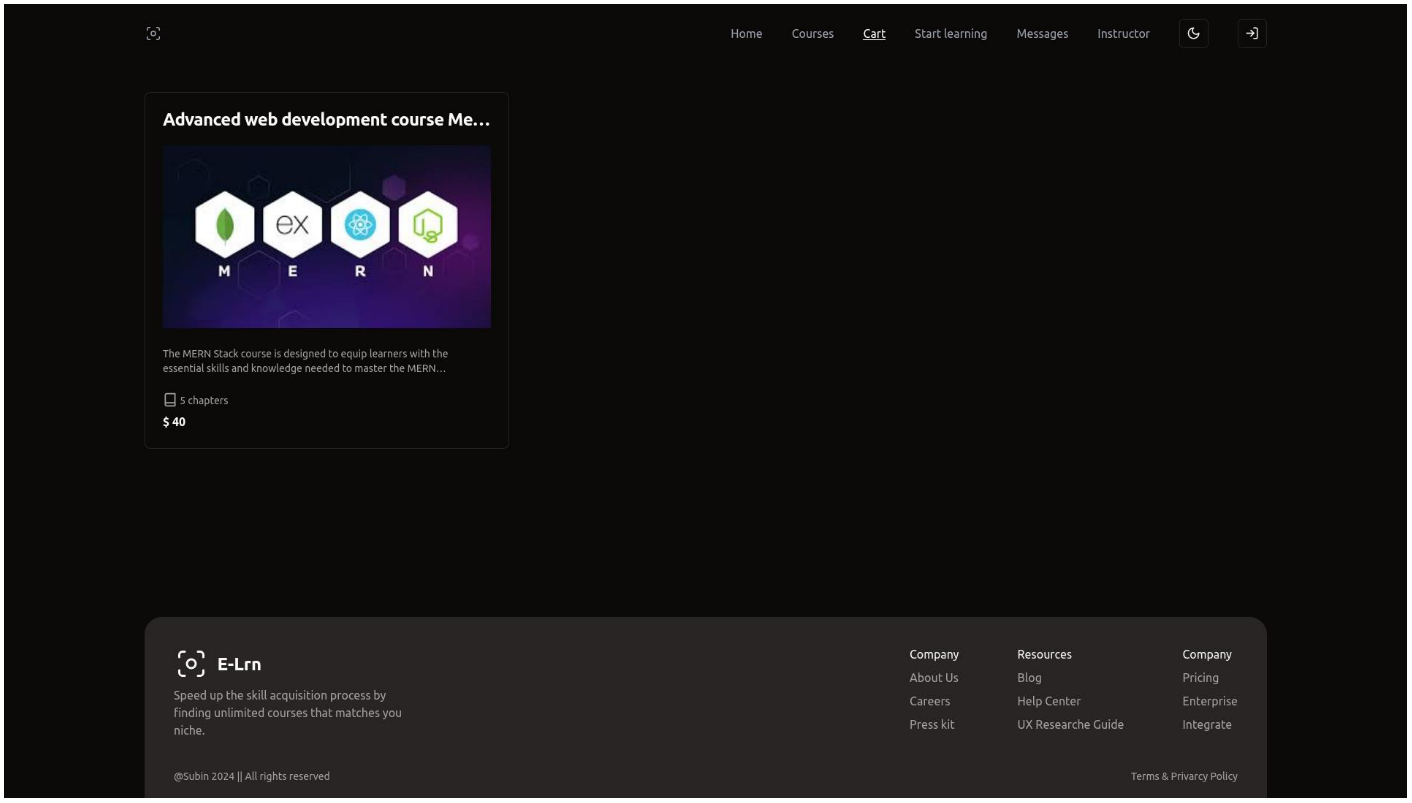
E-Lrn

Speed up the skill acquisition process by finding unlimited courses that matches you niche.

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Purcheased Courses



Advanced web development course Me...

The MERN Stack course is designed to equip learners with the essential skills and knowledge needed to master the MERN...

5 chapters

$ 40

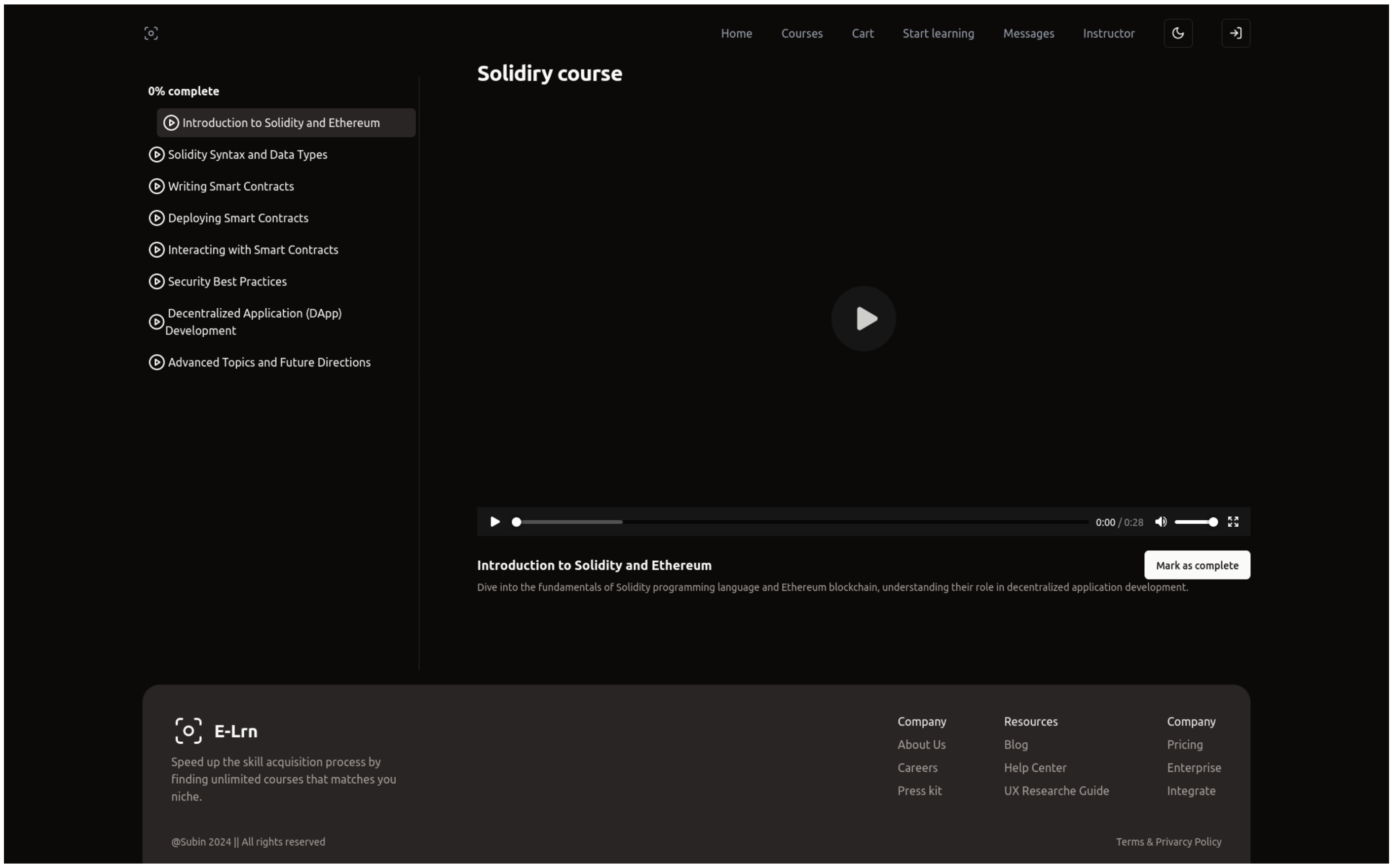
E-Lrn

Speed up the skill acquisition process by finding unlimited courses that matches you niche.

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Start Learning Page


Home Courses Cart Start learning Messages Instructor

Solidiry course

0% complete

- Introduction to Solidity and Ethereum
- Solidity Syntax and Data Types
- Writing Smart Contracts
- Deploying Smart Contracts
- Interacting with Smart Contracts
- Security Best Practices
- Decentralized Application (DApp) Development
- Advanced Topics and Future Directions

0:00 / 0:28

Introduction to Solidity and Ethereum

Mark as complete

Dive into the fundamentals of Solidity programming language and Ethereum blockchain, understanding their role in decentralized application development.

E-Lrn

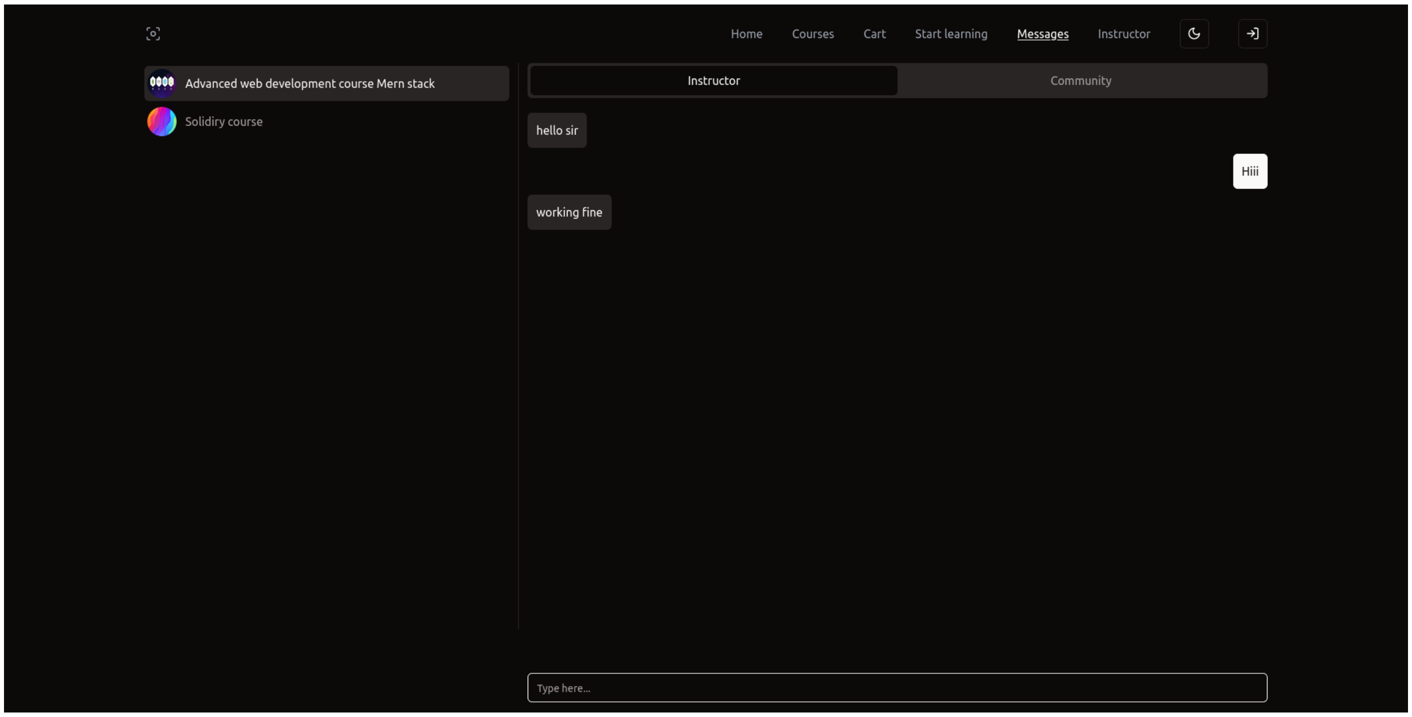
Speed up the skill acquisition process by finding unlimited courses that matches you niche.

Company	Resources	Company
About Us	Blog	Pricing
Careers	Help Center	Enterprise
Press kit	UX Researche Guide	Integrate

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Messages Page


Home Courses Cart Start learning Messages Instructor

Advanced web development course Mern stack

Solidiry course

Instructor Community

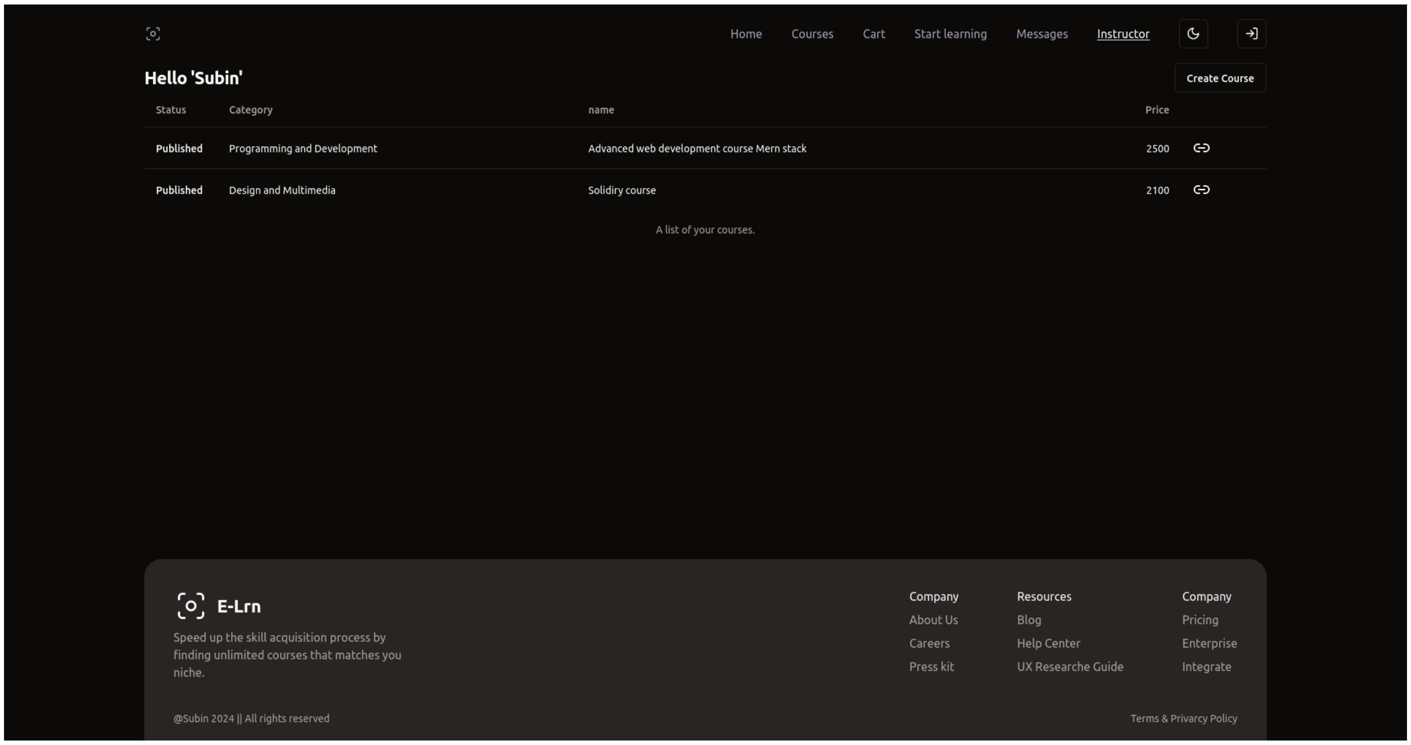
hello sir

Hiii

working fine

Type here...

Instructor’s Dashboard


Home Courses Cart Start learning Messages Instructor

Hello 'Subin'

Create Course

Status	Category	name	Price
Published	Programming and Development	Advanced web development course Mern stack	2500
Published	Design and Multimedia	Solidiry course	2100

A list of your courses.

E-Lrn

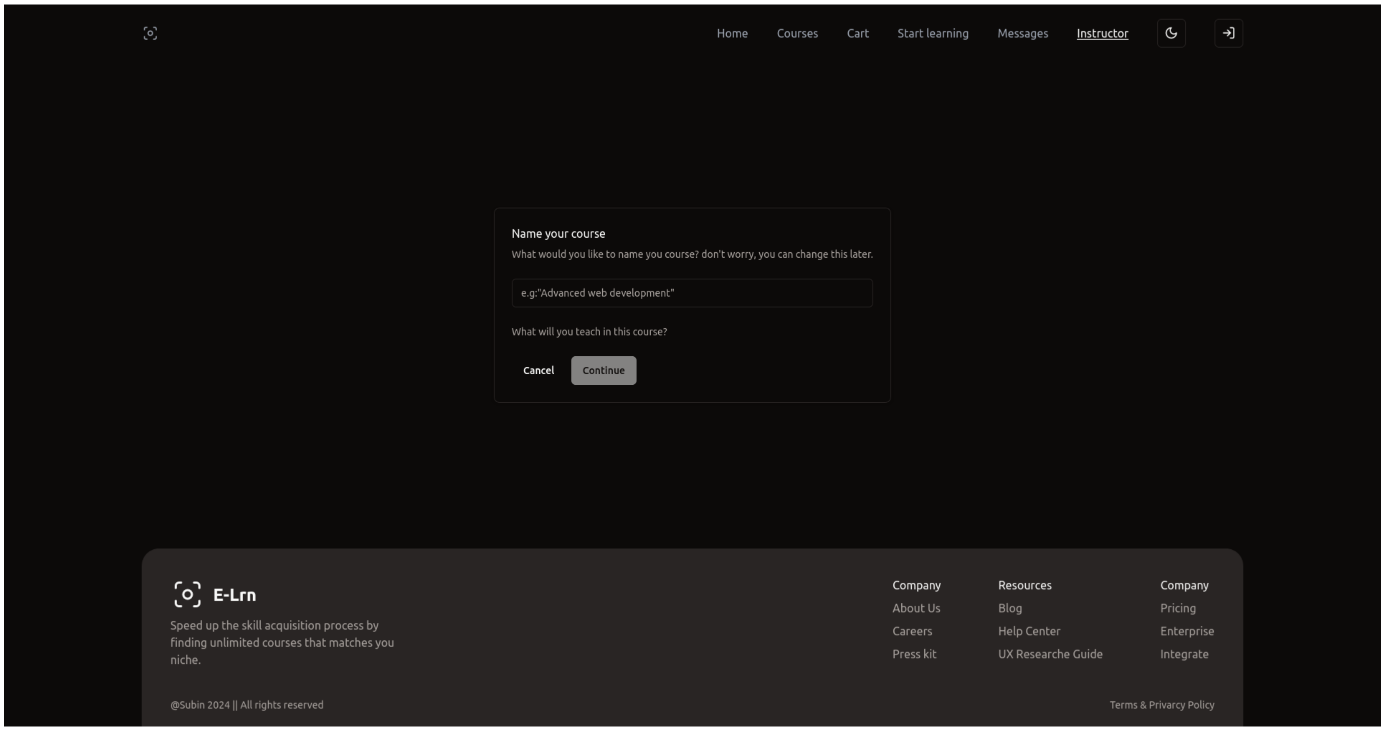
Speed up the skill acquisition process by finding unlimited courses that matches you niche.

Company	Resources	Company
About Us	Blog	Pricing
Careers	Help Center	Enterprise
Press kit	UX Researche Guide	Integrate

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Create Course Title Page


Home Courses Cart Start learning Messages Instructor

Name your course

What would you like to name you course? don't worry, you can change this later.

e.g:"Advanced web development"

What will you teach in this course?

Cancel Continue

E-Lrn

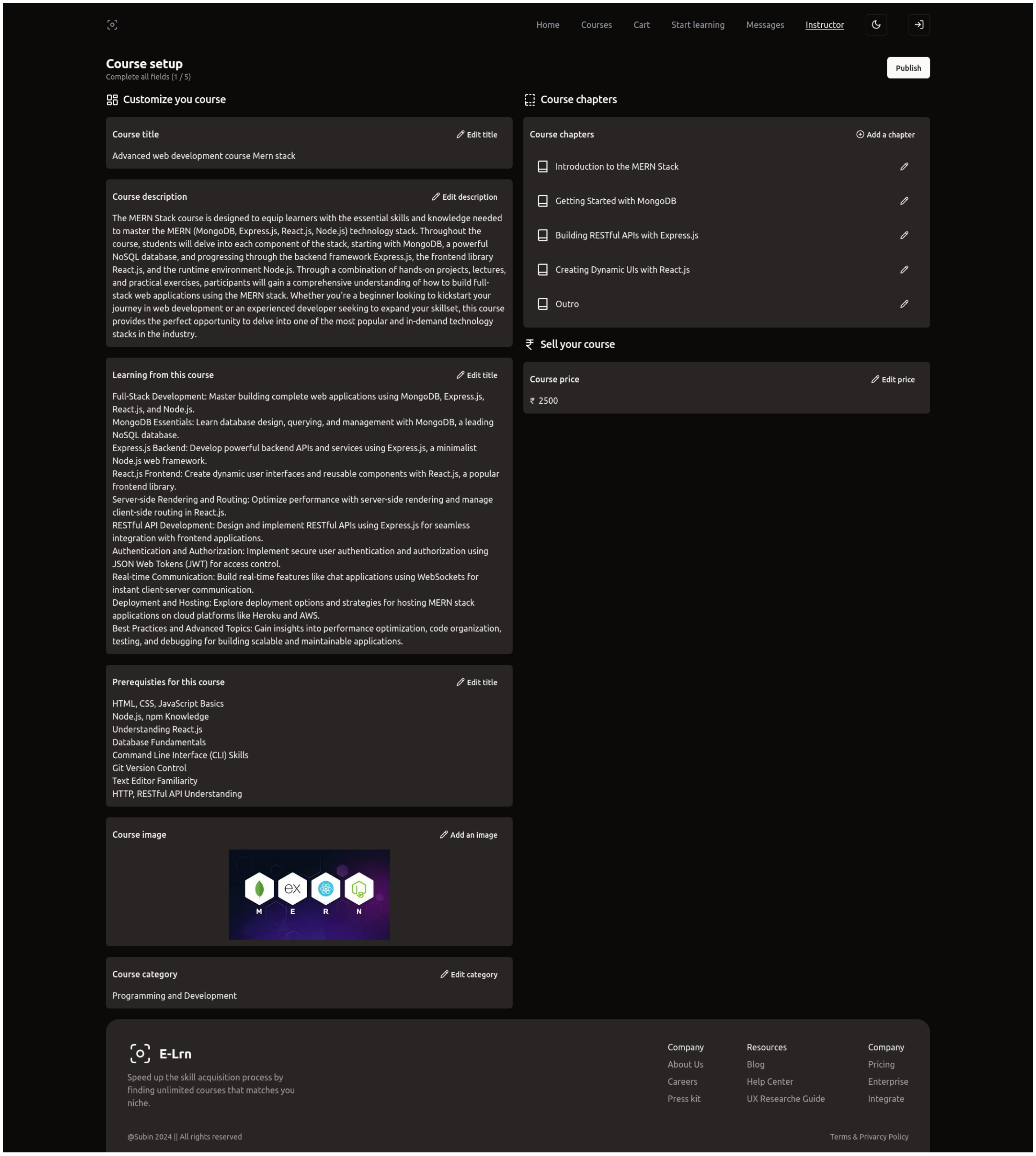
Speed up the skill acquisition process by finding unlimited courses that matches you niche.

Company	Resources	Company
About Us	Blog	Pricing
Careers	Help Center	Enterprise
Press kit	UX Researche Guide	Integrate

@Subin 2024 || All rights reserved

Terms & Privacy Policy

Create and Edit Course Page



Course setup

Complete all fields (1 / 5)

Customize you course

Course title

Advanced web development course Mern stack

Course description

The MERN Stack course is designed to equip learners with the essential skills and knowledge needed to master the MERN (MongoDB, Express.js, React.js, Node.js) technology stack. Throughout the course, students will delve into each component of the stack, starting with MongoDB, a powerful NoSQL database, and progressing through the backend framework Express.js, the frontend library React.js, and the runtime environment Node.js. Through a combination of hands-on projects, lectures, and practical exercises, participants will gain a comprehensive understanding of how to build full-stack web applications using the MERN stack. Whether you're a beginner looking to kickstart your journey in web development or an experienced developer seeking to expand your skillset, this course provides the perfect opportunity to delve into one of the most popular and in-demand technology stacks in the industry.

Learning from this course

Full-Stack Development: Master building complete web applications using MongoDB, Express.js, React.js, and Node.js.
MongoDB Essentials: Learn database design, querying, and management with MongoDB, a leading NoSQL database.
Express.js Backend: Develop powerful backend APIs and services using Express.js, a minimalist Node.js web framework.
React.js Frontend: Create dynamic user interfaces and reusable components with React.js, a popular frontend library.
Server-side Rendering and Routing: Optimize performance with server-side rendering and manage client-side routing in React.js.
RESTful API Development: Design and implement RESTful APIs using Express.js for seamless integration with frontend applications.
Authentication and Authorization: Implement secure user authentication and authorization using JSON Web Tokens (JWT) for access control.
Real-time Communication: Build real-time features like chat applications using WebSockets for instant client-server communication.
Deployment and Hosting: Explore deployment options and strategies for hosting MERN stack applications on cloud platforms like Heroku and AWS.
Best Practices and Advanced Topics: Gain insights into performance optimization, code organization, testing, and debugging for building scalable and maintainable applications.

Prerequisties for this course

HTML, CSS, JavaScript Basics
Node.js, npm Knowledge
Understanding React.js
Database Fundamentals
Command Line Interface (CLI) Skills
Git Version Control
Text Editor Familiarity
HTTP, RESTful API Understanding

Course image

Course category

Programming and Development

Course chapters

Course chapters

Introduction to the MERN Stack
Getting Started with MongoDB
Building RESTful APIs with Express.js
Creating Dynamic UIs with React.js
Outro

Sell your course

Course price

₹ 2500

E-Lrn

Speed up the skill acquisition process by finding unlimited courses that matches you niche.

@Subin 2024 || All rights reserved

10. Conclusion

In the dynamic realm of the software industry, the escalating demand for web applications is fueled by client companies' increasing expectations for innovative solutions. Our journey in automating an office application not only provided practical experience but also deepened our understanding of software development complexities. Embracing analytical problem-solving approaches and emphasizing collaboration, communication, and iterative refinement, we navigated through challenges, fostering efficiency, discipline, and adaptability. This profound learning experience has equipped us with invaluable insights and skills, laying a robust foundation for our future endeavors in technology and innovation.

11. Feature Enhancement

Looking ahead, our e-learning website project is poised for exciting advancements aimed at elevating the educational experience for students. Our future enhancements revolve around three key pillars:

- **Quizzes for Skill Assessment**: We plan to introduce interactive quizzes to assess students' understanding and proficiency in course materials.
- **Live Sessions with Video Call Functionality**: We aim to implement live sessions through video calls, fostering real-time interaction between instructors and students.
- **Downloadable PDF Resources**: We will incorporate downloadable PDF files to provide supplementary resources for offline learning and reference.

The introduction of quizzes will offer students the opportunity to engage with course content actively and receive immediate feedback on their progress. These assessments will not only gauge comprehension but also serve as valuable tools for reinforcing learning outcomes.

Live sessions via video call functionality will enable instructors to conduct dynamic lectures, interactive discussions, and Q&A sessions in real time. This immersive learning experience will foster collaboration, facilitate personalized instruction, and create a sense of community among learners.

Furthermore, the inclusion of downloadable PDF resources will enhance accessibility and flexibility, allowing students to review course materials at their own pace and offline. These resources will complement the interactive nature of the platform, providing additional support and enrichment for students' educational journey.

Through these future enhancements, we are dedicated to creating a more engaging, interactive, and accessible e-learning platform that empowers students to succeed in their academic pursuits.

12. Bibliography

For Reactjs

- https://www.w3schools.com/REACT/DEFAULT.ASP
- https://legacy.reactjs.org/docs/getting-started.html
- https://react.dev/learn

For Nodejs

- https://www.w3schools.com/nodejs/
- https://nodejs.org/docs/latest/api/

For Expressjs

- https://devdocs.io/express/

For Mongodb

- https://www.mongodb.com/docs/