# **Online Learning Platform Documentation**

## **Table of Contents**

* Introduction
  + 1.1 Objective
  + 1.2 Overview of the Project
* User Roles
  + 2.1 Student
  + 2.2 Teacher
  + 2.3 Admin
* Features
  + 3.1 User Authentication
  + 3.2 Application Approval
  + 3.3 Dashboard
  + 3.4 Course Purchase
  + 3.5 Live Video Conferencing
  + 3.6 Communication
  + 3.7 Payment Integration
* Technical Stack
  + 4.1 UI/UX Design
  + 4.2 Frontend Technologies
  + 4.3 Backend Technologies
  + 4.4 Database
  + 4.5 Authentication
  + 4.6 Video Conferencing
  + 4.7 Payment Integration
* System Architecture
  + 5.1 Architecture Diagram
  + 5.2 Data Flow
* Implementation
  + 6.1 Backend Implementation
  + 6.2 Frontend Implementation
  + 6.3 Integration of Features
* Testing
  + 7.1 Testing Strategies
  + 7.2 User Acceptance Testing
* Deployment
  + 8.1 Hosting Options
  + 8.2 Continuous Deployment
* Future Enhancements
* Conclusion
* References

## **Introduction**

This project aims to develop a comprehensive online learning platform using the MERN stack, catering to Students, Teachers, and Administrators. It will facilitate course creation and management, incorporating a rigorous approval process to ensure quality content. The platform will also feature live video conferencing for real-time interaction between students and instructors. By enhancing accessibility and engagement, this platform seeks to revolutionize the online learning experience. Join us in building an innovative educational environment that bridges geographical divides.

### **1.1 Objective**

The objective of this documentation is to provide a comprehensive guide for developers, administrators, and users of the online learning platform. It outlines the system's architecture, features, user roles, and implementation details to ensure a smooth experience for all stakeholders.

### **1.2 Overview of the Project**

The online learning platform is designed to facilitate remote education by connecting students with teachers through a virtual environment. It offers various features such as course management, live video conferencing, and payment processing to enhance the learning experience.

An online learning platform built with the MERN stack

provides a virtual space where students can enroll in

courses, attend live classes, communicate with

instructors, and make payments for premium courses.

Instructors can create and manage courses, conduct live

video sessions, and interact with students. Admins can

monitor activities, approve course applications, and

ensure platform integrity.

This project aims to enhance remote education

accessibility and engagement by combining essential

learning tools into one unified platform.

## **User Roles**

### **2.1 Student**

Students can create accounts, browse courses, enroll in classes, participate in live sessions, and communicate with instructors and peers.

### **2.2 Teacher**

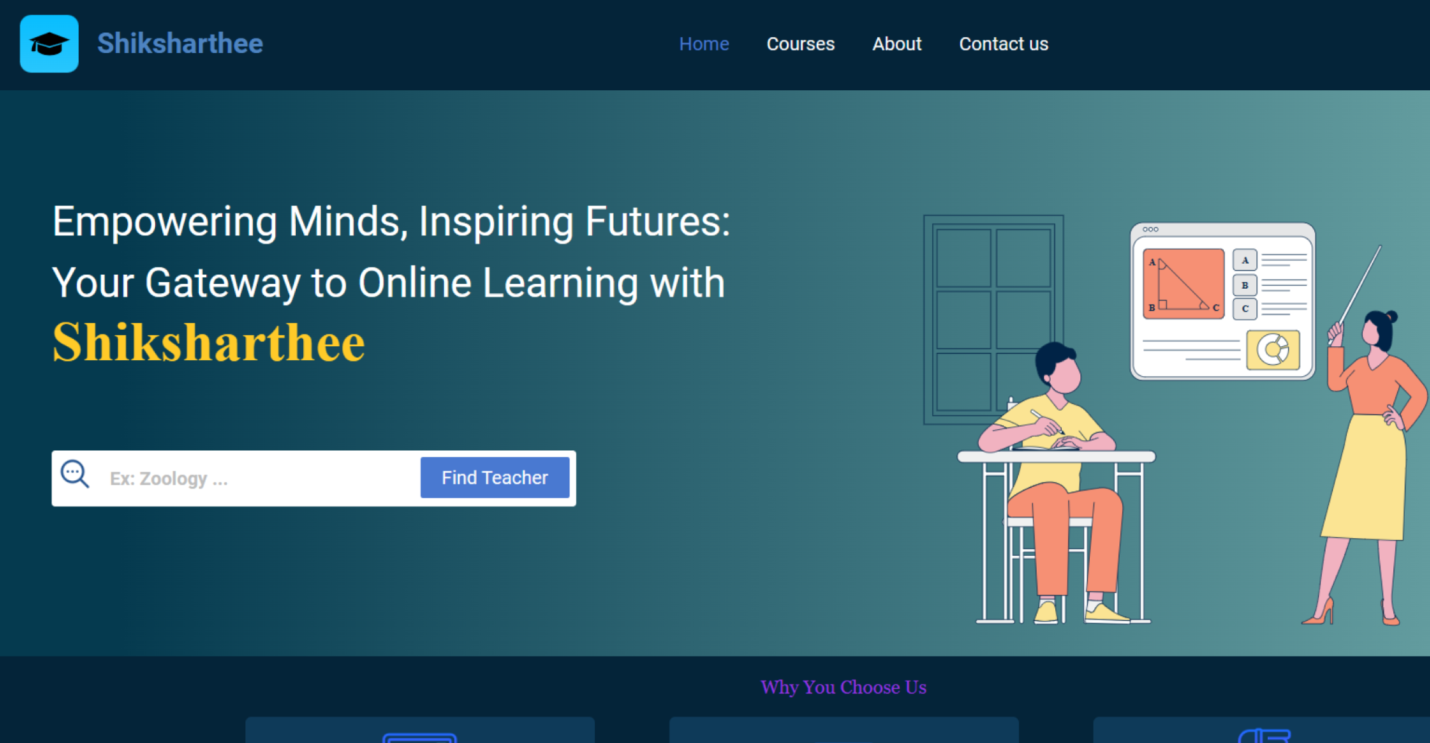
Teachers can create and manage courses, conduct live classes, grade assignments, and interact with students through the platform.

### 

### **2.3 Admin**

Admins oversee the platform's operations, manage user accounts, approve course applications, and ensure the platform's integrity and security.

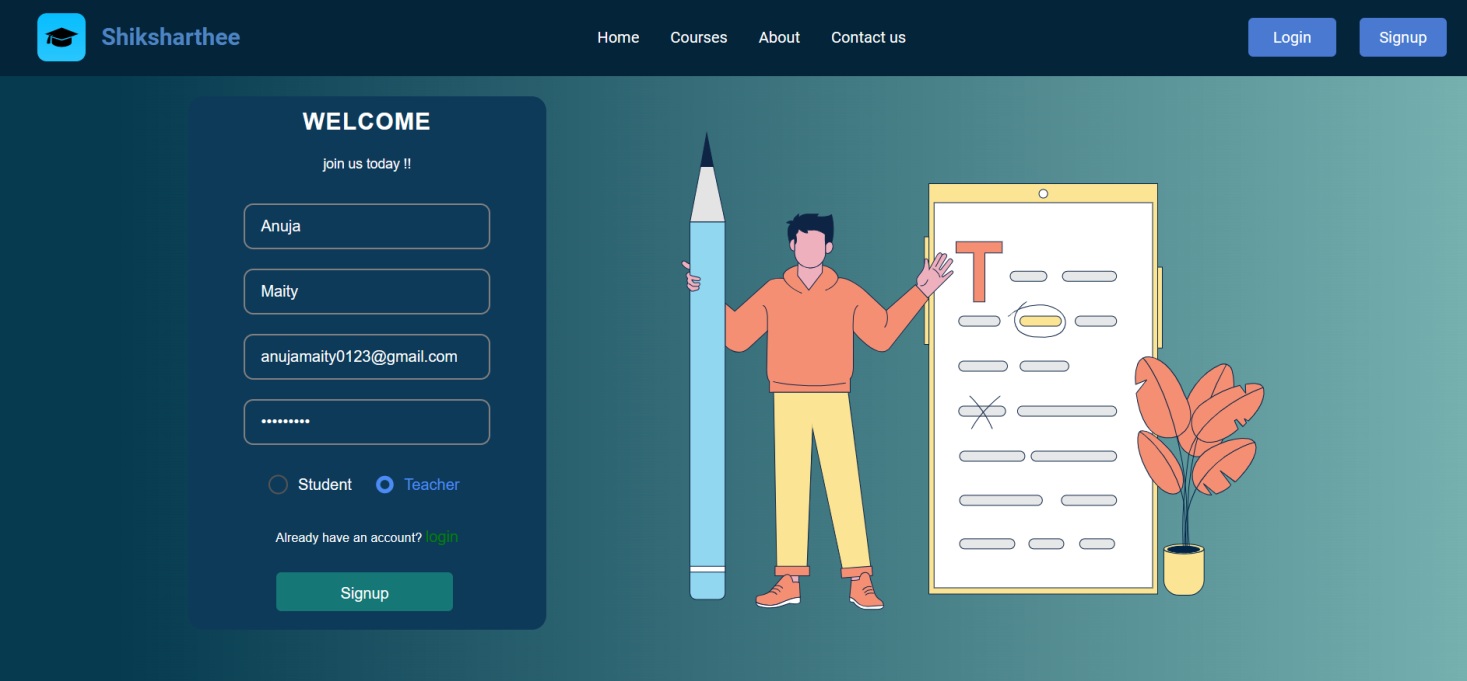
## **Features**

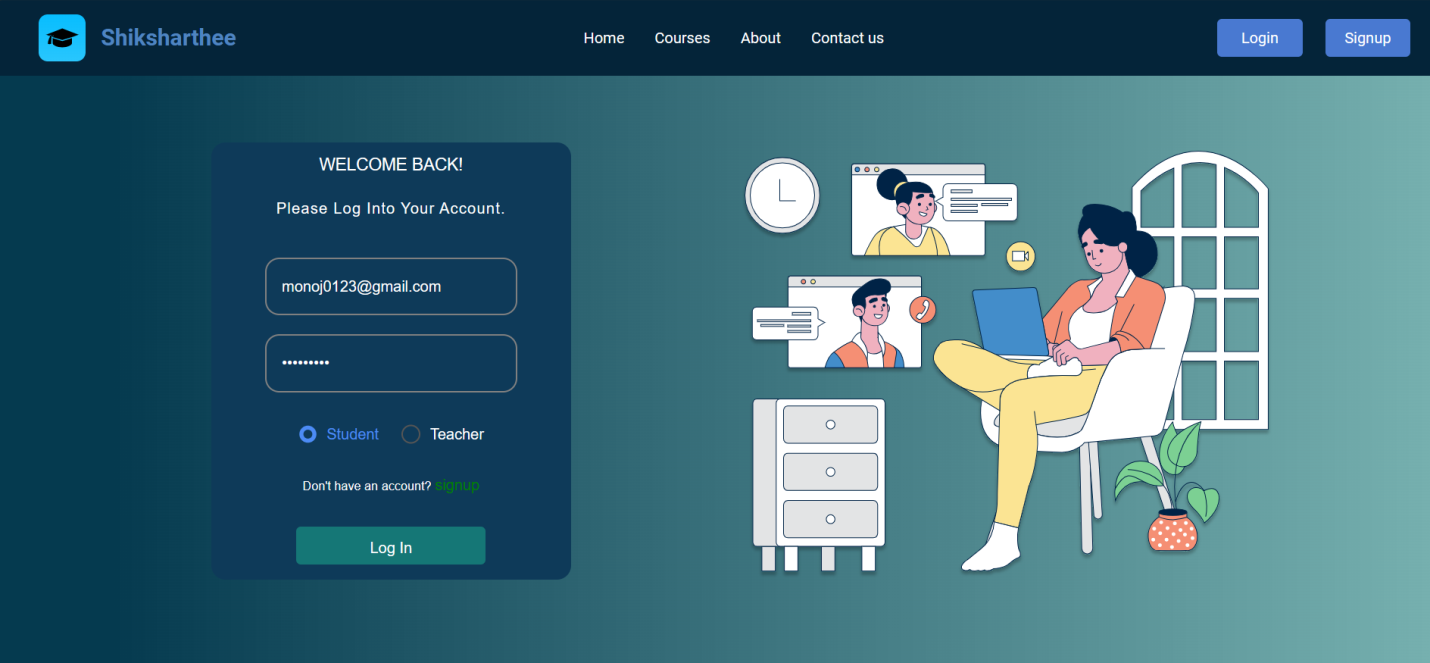


### 

### **3.1 User Authentication**

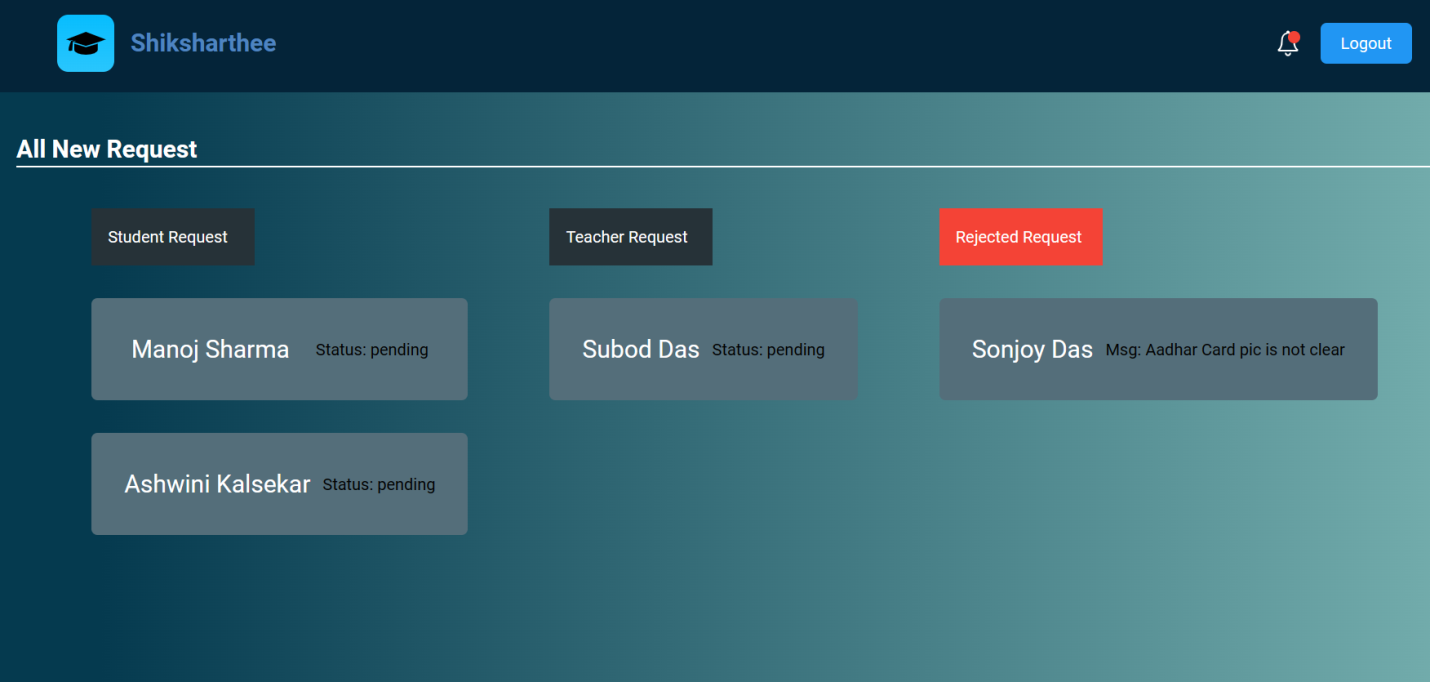
Secure login and registration processes for students, teachers, and admins, utilizing industry-standard practices.





### **3.2 Application Approval**

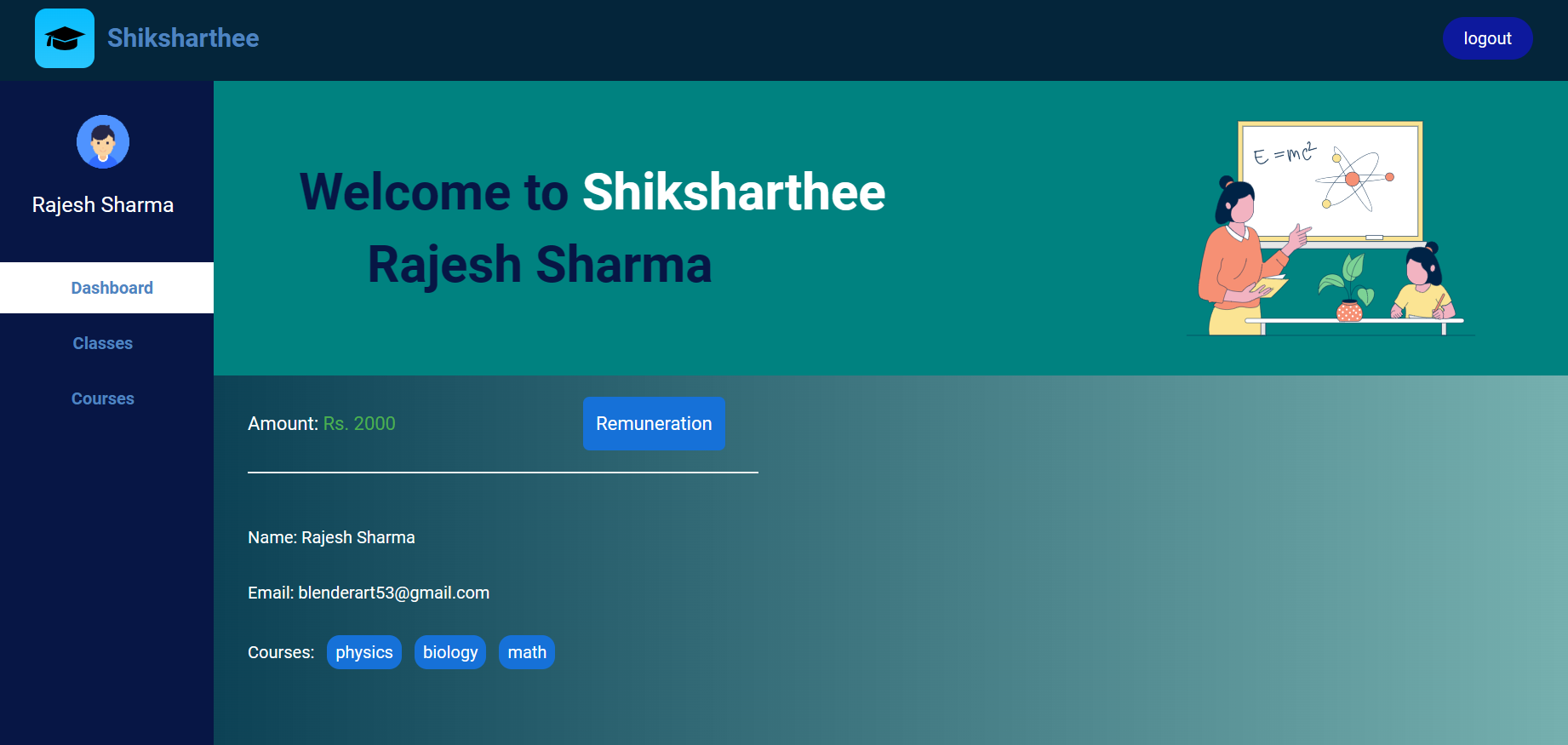
Admins can review and approve or reject course applications submitted by teachers.

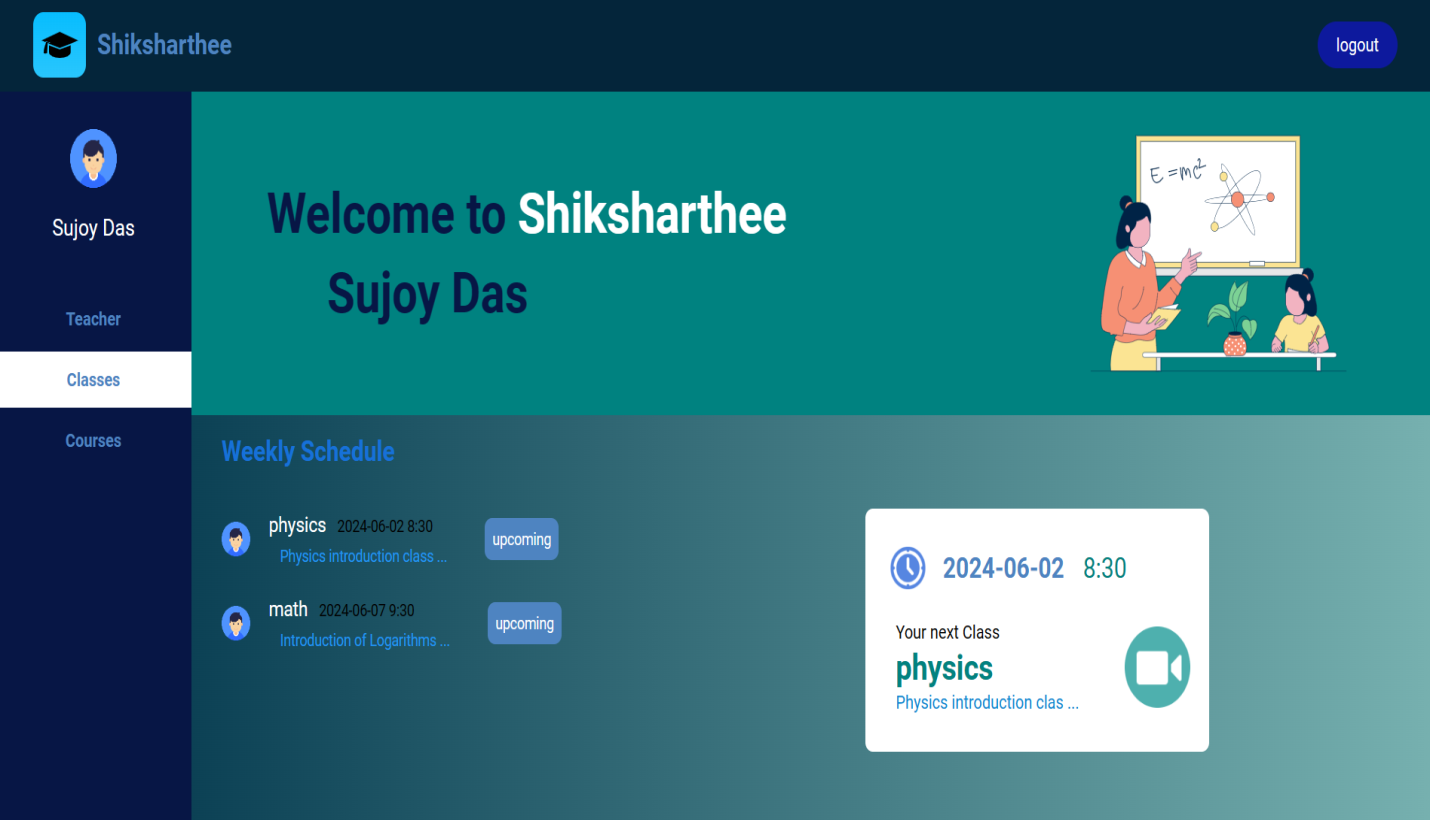


### 

### **3.3 Dashboard**

Personalized dashboards for students and teachers to track progress, upcoming classes, and notifications.

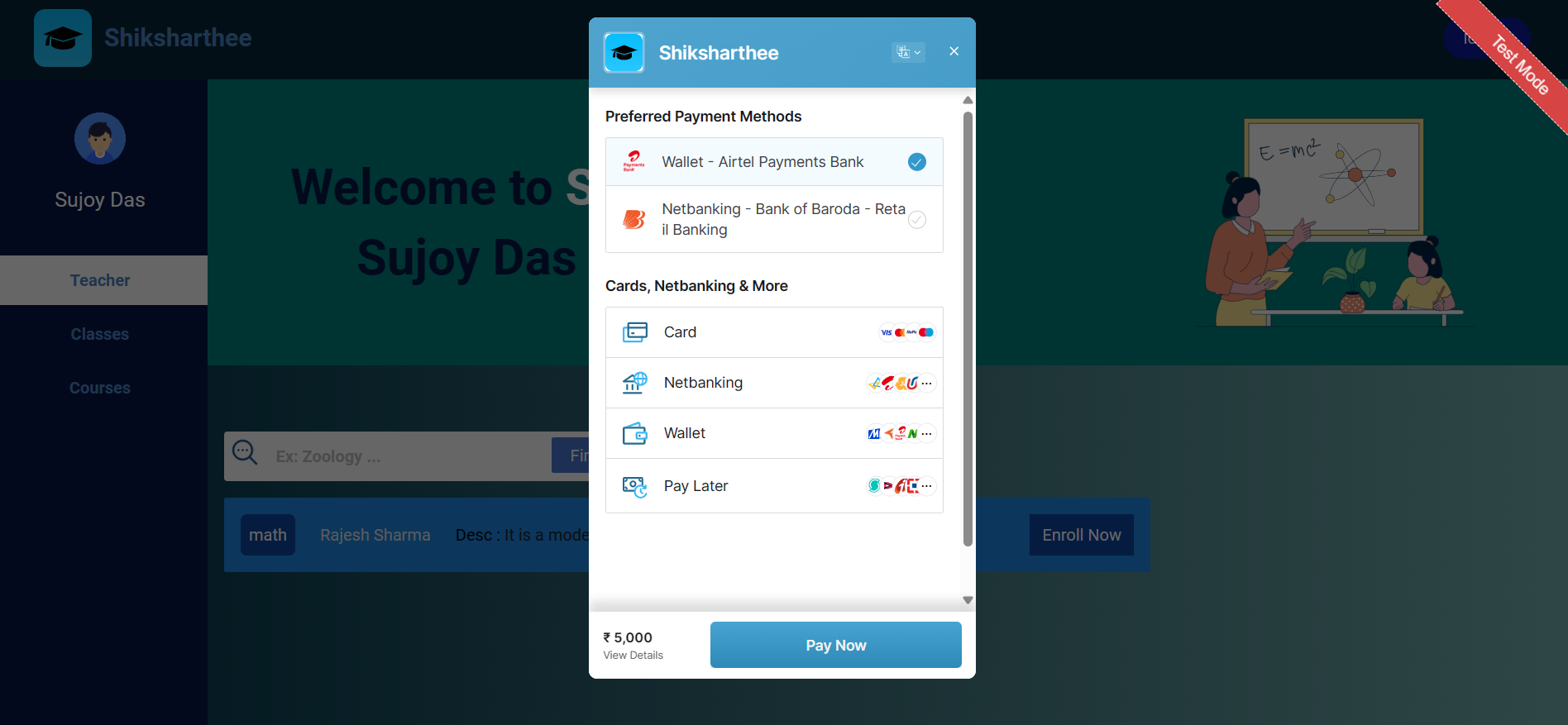




### 

### **3.4 Course Purchase**

Students can purchase courses through a secure payment gateway.



### 

### **3.5 Live Video Conferencing**

Integration of video conferencing tools for real-time interaction during classes. Integration with platforms like WebRTC, Zoom, or Jitsi enables real-time class sessions.

### **3.6 Communication Tools**

Inbuilt messaging system for communication between students and teachers. In-built messaging system allowing students and teachers to communicate effectively within the platform.

### **3.7 Payment Integration**

Secure payment processing for course purchases, including support for multiple payment methods. Students can buy courses securely using integrated payment gateways like Stripe, PayPal, or Razorpay.

## **Technical Stack**

### **4.1 UI/UX Design**

Responsive design principles to ensure a seamless user experience across devices.

### 

### **4.2 Frontend Technologies**

Utilization of HTML, CSS, JavaScript, and frameworks like React or Angular for the frontend development.

### **4.3 Backend Technologies**

Node.js or Python (Django/Flask) for server-side development, ensuring robust performance.

### **4.4 Database**

Relational (e.g., PostgreSQL, MySQL) or NoSQL (e.g., MongoDB) databases for data storage.

### **4.5 Authentication**

JWT (JSON Web Tokens) or OAuth for secure user authentication.

### 

### **4.6 Video Conferencing**

Integration with platforms like Zoom, WebRTC, or Jitsi for live class sessions.

### 

### **4.7 Payment Integration**

Use of payment gateways such as Stripe, PayPal, or Razorpay for processing transactions.

### **Project Implementation**

1. **Backend Implementation**  
   Includes API endpoint setup, controllers, and database configurations.
2. **Frontend Implementation**  
   User interface development and connection to the backend.
3. **Integration of Features**  
   Step-by-step feature integration, ensuring cohesive functionality.

### **Testing**

* **Testing Strategies**: Unit, integration, and end-to-end testing.
* **User Acceptance Testing**: Ensure platform meets user expectations.

## 

## **System Architecture**

### **5.1 Architecture Diagram**

A visual representation of the system architecture, including frontend, backend, database, and external services.

### **5.2 Data Flow**

Description of how data moves through the system, from user input to database storage and retrieval.

## **Implementation**

### **6.1 Backend Implementation**

Details on setting up the server, API endpoints, and database interactions.

### **6.2 Frontend Implementation**

Instructions for building the user interface and connecting it with the backend.

### **6.3 Integration of Features**

Step-by-step guide on how to integrate various features into the platform.

## **Testing**

### **7.1 Testing Strategies**

Overview of testing methodologies including unit testing, integration testing, and end-to-end testing.

### **7. 2 User Acceptance Testing**

Guidelines for conducting user acceptance testing to ensure the platform meets user requirements.

## 

## **Deployment**

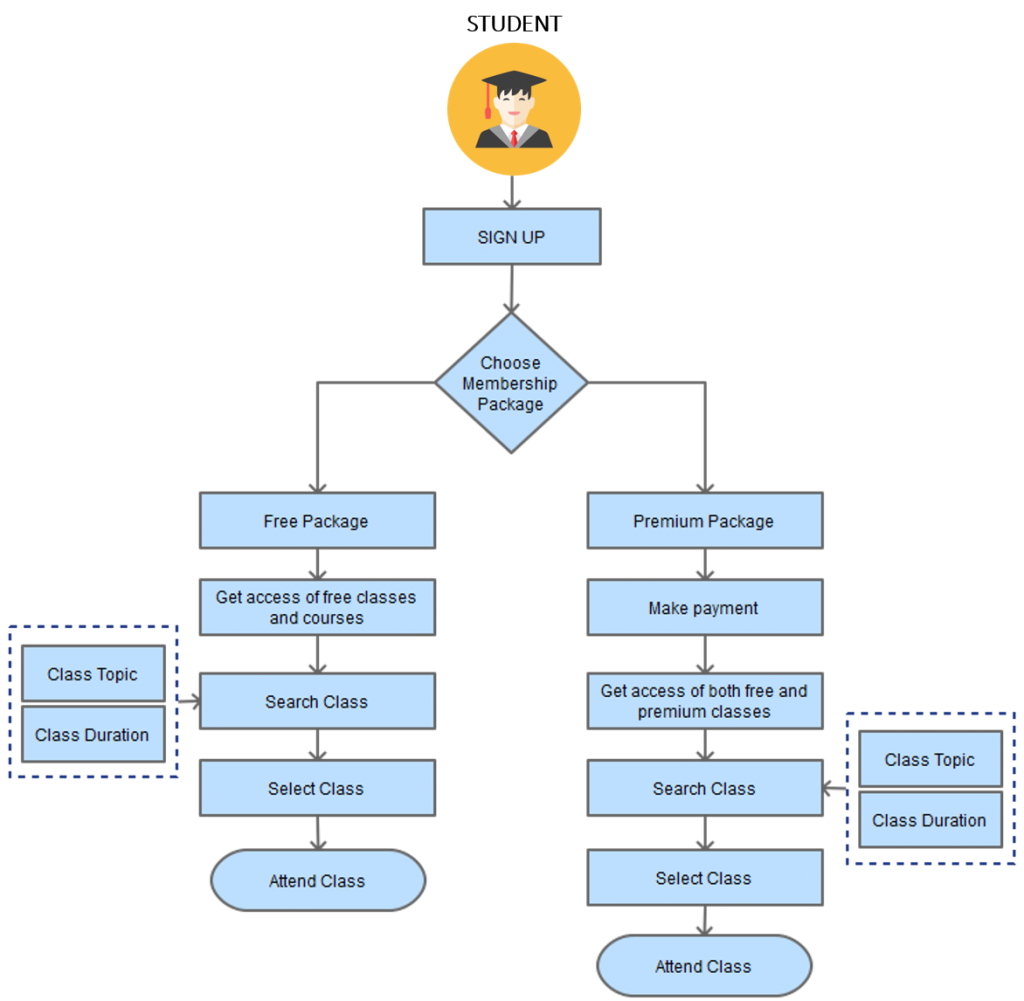
### **8.1 Hosting Options**

Comparison of hosting options, including cloud providers (AWS, Azure, Google Cloud), VPS, and dedicated servers.

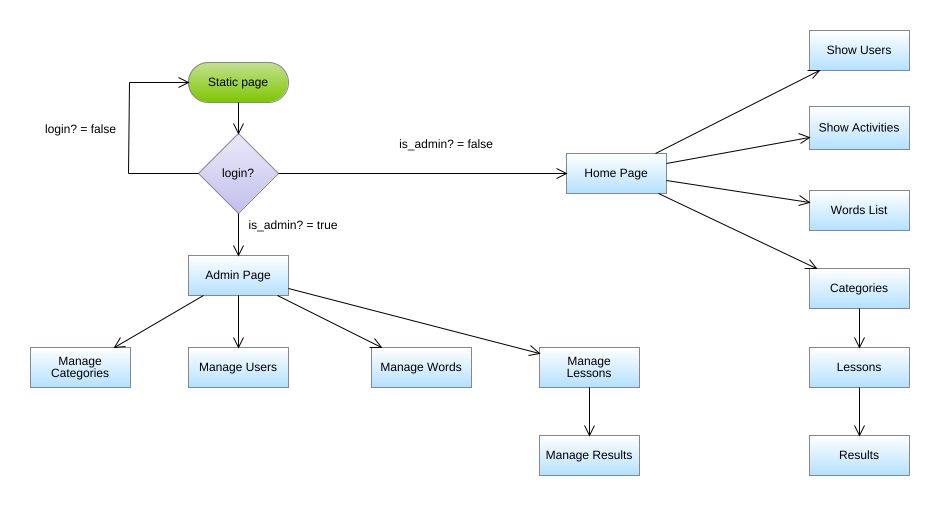
### **8.2 Continuous Deployment**

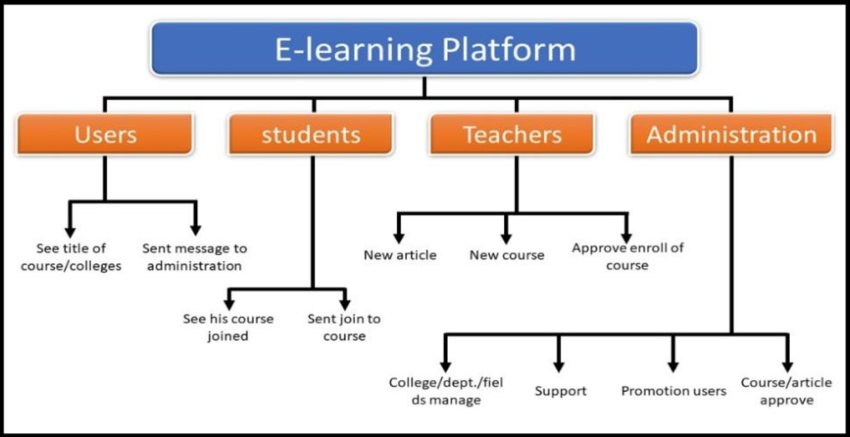
Automated deployment strategies using tools like Jenkins, Docker, or Kubernetes.

**FlowChart:**

****

**ER-Diagram**

****

****

**Coding**

* **Frontend**

**Src:**

**index.html**

**<!doctype html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8" />**

**<meta name="viewport" content="width=device-width, initial-scale=1.0" />**

**<meta name="description" content="Shiksharthee: Your online learning platform for courses, tutorials, and live classes." />**

**<link rel="icon" type="image/svg+xml" href="/src/Pages/Images/logo.svg" />**

**<link rel="icon" type="image/png" href="/src/Pages/Images/logo.png" sizes="32x32" />**

**<link rel="stylesheet" href="/src/styles.css" /> <!-- Include your CSS file -->**

**<title>Shiksharthee</title>**

**</head>**

**<body>**

**<div id="root"></div>**

**<script type="module" src="/src/main.jsx"></script>**

**<script src="https://checkout.razorpay.com/v1/checkout.js"></script>**

**</body>**

**</html>**

**Index.css**

**@tailwind base;**

**@tailwind components;**

**@tailwind utilities;**

**\*{**

**margin: 0;**

**padding: 0;**

**}**

**body{**

**background:linear-gradient(90deg, #053a4f 8.26%, rgba(20, 120, 118, 0.58) 99.96%);**

**overflow: scroll;**

**}**

**body::-webkit-scrollbar{**

**display: none;**

**}**

**Postcss.config.js**

**export default {**

**plugins: {**

**tailwindcss: {},**

**autoprefixer: {},**

**},**

**}**

**tailwind.config.js**

**const withMT = require("@material-tailwind/react/utils/withMT");**

**module.exports = withMT({**

**content: ["./index.html", "./src/\*\*/\*.{vue,js,ts,jsx,tsx}"],**

**theme: {**

**extend: {},**

**},**

**plugins: [],**

**});**

**vite.config.js**

**import { defineConfig } from 'vite'**

**import react from '@vitejs/plugin-react'**

**// https://vitejs.dev/config/**

**export default defineConfig({**

**server:{**

**proxy:{**

**'/api': 'http://localhost:4400'**

**}**

**},**

**plugins: [react()],**

**})**

* **Backend:**

**app.js**

**import express from "express";**

**import cors from "cors";**

**import cookieParser from "cookie-parser";**

**import Razorpay from "razorpay"**

**const app = express();**

**app.use(cors())**

**app.use(express.json({limit: "16kb"}))**

**app.use(express.urlencoded({extended: true, limit: "16kb"}))**

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

**app.use(cookieParser())**

**export const instance = new Razorpay({**

**key\_id: process.env.KEY\_ID,**

**key\_secret: process.env.KEY\_SECRET**

**})**

**//student routes**

**import studentRouter from "./routes/student.routes.js";**

**app.use("/api/student", studentRouter)**

**//teacher routes**

**import teacherRouter from "./routes/teacher.routes.js"**

**app.use("/api/teacher", teacherRouter)**

**//course routes**

**import courseRouter from "./routes/course.routes.js"**

**app.use("/api/course", courseRouter)**

**import adminRouter from "./routes/admin.routes.js"**

**app.use("/api/admin", adminRouter)**

**import paymentRouter from "./routes/payment.routes.js"**

**app.use("/api/payment", paymentRouter)**

**export {app}**

**index.js**

**import dotenv from "dotenv"**

**import db from './database/db.js';**

**import {app} from './app.js'**

**dotenv.config({**

**path: './.env'**

**})**

**console.log(`${process.env.DB\_NAME}`);**

**db()**

**.then(() => {**

**app.listen(process.env.PORT || 8000, () => {**

**console.log(`⚙️ Server is running at port : ${process.env.PORT}`);**

**})**

**})**

**.catch((err) => {**

**console.log(" mongodb connection failed !!! ", err);**

**})**

### **1. Controllers**

Controllers manage business logic. Here’s an enhancement for the authController and courseController:

**authController.js**

* Add email verification functionality.
* Include error handling and response messages for better user experience.

javascript

// controllers/authController.js

const User = require('../models/User');

const jwt = require('jsonwebtoken');

const sendVerificationEmail = require('../utils/emailService');

exports.register = async (req, res) => {

const { username, email, password } = req.body;

try {

const newUser = new User({ username, email, password });

await newUser.save();

await sendVerificationEmail(email); // Send verification email

res.status(201).json({ message: 'User registered successfully. Check your email for verification.' });

} catch (error) {

res.status(400).json({ message: 'Error registering user: ' + error.message });

}

};

// Login user

exports.login = async (req, res) => {

const { email, password } = req.body;

try {

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

if (!user || !(await user.comparePassword(password))) {

return res.status(401).json({ message: 'Invalid credentials' });

}

const token = jwt.sign({ id: user.\_id, role: user.role }, process.env.JWT\_SECRET, { expiresIn: '1h' });

res.json({ token, user });

} catch (error) {

res.status(500).json({ message: 'Server error' });

}

};

**courseController.js**

* Implement error handling and response formatting.

javascript

// controllers/courseController.js

const Course = require('../models/Course');

exports.createCourse = async (req, res) => {

const { title, description, price } = req.body;

try {

const newCourse = new Course({ title, description, price, createdBy: req.user.id });

await newCourse.save();

res.status(201).json(newCourse);

} catch (error) {

res.status(400).json({ message: 'Error creating course: ' + error.message });

}

};

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

try {

const courses = await Course.find().populate('createdBy', 'username');

res.json(courses);

} catch (error) {

res.status(500).json({ message: 'Error fetching courses: ' + error.message });

}

};

### 2. Middlewares

Middlewares provide additional functionality for request handling. Here’s how you can refine them:

**authMiddleware.js**

* Enhance error handling and token validation.

javascript

// middlewares/authMiddleware.js

const jwt = require('jsonwebtoken');

exports.verifyToken = (req, res, next) => {

const token = req.headers['authorization'];

if (!token) return res.status(403).send('A token is required for authentication');

jwt.verify(token, process.env.JWT\_SECRET, (err, decoded) => {

if (err) return res.status(401).send('Invalid Token');

req.user = decoded; // Attach user info to request

next();

});

};

**adminMiddleware.js**

* Simplify the logic and add better error responses.

javascript

// middlewares/adminMiddleware.js

exports.isAdmin = (req, res, next) => {

if (req.user.role !== 'Admin') {

return res.status(403).send('Access denied: Admins only');

}

next();

};

### 3. Models

Ensure your models handle data integrity and hashing effectively.

**User.js**

* Include methods for password comparison.

javascript

// models/User.js

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({

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

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

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

role: { type: String, enum: ['Student', 'Teacher', 'Admin'], default: 'Student' },

});

// Hash password before saving

UserSchema.pre('save', async function(next) {

if (!this.isModified('password')) return next();

this.password = await bcrypt.hash(this.password, 10);

next();

});

// Compare password method

UserSchema.methods.comparePassword = async function(candidatePassword) {

return await bcrypt.compare(candidatePassword, this.password);

};

module.exports = mongoose.model('User', UserSchema);

**Course.js**

* Make sure to validate course fields as needed.

javascript

// models/Course.js

const mongoose = require('mongoose');

const CourseSchema = new mongoose.Schema({

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

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

price: { type: Number, required: true },

createdBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

studentsEnrolled: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],

});

module.exports = mongoose.model('Course', CourseSchema);

### 4. Routes

Set up the routes with necessary middleware.

**authRoutes.js**

javascript

// routes/authRoutes.js

const express = require('express');

const router = express.Router();

const authController = require('../controllers/authController');

router.post('/register', authController.register);

router.post('/login', authController.login);

module.exports = router;

**courseRoutes.js**

javascript

// routes/courseRoutes.js

const express = require('express');

const router = express.Router();

const courseController = require('../controllers/courseController');

const { verifyToken } = require('../middlewares/authMiddleware');

router.post('/', verifyToken, courseController.createCourse);

router.get('/', courseController.getAllCourses);

module.exports = router;

### 5. Utils

Utilities help with tasks like sending emails and processing payments.

**emailService.js**

javascript

// utils/emailService.js

const nodemailer = require('nodemailer');

const sendEmail = async (email, subject, text) => {

const transporter = nodemailer.createTransport({

service: 'gmail',

auth: {

user: process.env.EMAIL\_USER,

pass: process.env.EMAIL\_PASS,

},

});

const mailOptions = {

from: process.env.EMAIL\_USER,

to: email,

subject,

text,

};

try {

await transporter.sendMail(mailOptions);

console.log('Email sent:', subject);

} catch (error) {

console.error('Error sending email:', error);

}

};

module.exports = sendEmail;

**paymentService.js**

javascript

// utils/paymentService.js

const stripe = require('stripe')(process.env.STRIPE\_SECRET\_KEY);

const processPayment = async (amount, currency, paymentMethod) => {

try {

const paymentIntent = await stripe.paymentIntents.create({

amount,

currency,

payment\_method: paymentMethod,

});

return paymentIntent;

} catch (error) {

console.error('Error processing payment:', error);

throw error; // Rethrow to handle in calling function

}

};

module.exports = processPayment;

## 

## **Future Enhancements**

List of potential future enhancements, including AI-powered course recommendations, gamification, and social learning features.

* **AI-Powered Recommendations**: Use machine learning to recommend courses based on student interests and past behavior.
* **Gamification**: Add badges, leaderboards, and rewards for course completion to encourage engagement.
* **Social Learning Features**: Allow peer discussions, group projects, and forums to enhance collaborative learning.

## 

## 

## **Conclusion**

The online learning platform is designed to provide a comprehensive and engaging educational experience. This documentation serves as a guide for developers, administrators, and users to ensure a seamless experience. Building an online learning platform with the MERN stack is a scalable, powerful solution for remote education. With real-time features like video conferencing, secure payment processing, and robust user management, the platform can deliver a seamless and engaging learning experience. Following this approach also allows for easy updates and integration of future enhancements like AI recommendations or gamification, making it adaptable to evolving educational needs

## **References**

List of sources used in the development of the platform, including research papers, articles, and online resources.

### . **MERN Stack Basics**

* **MERN Stack Tutorial for Beginners**: This tutorial from MongoDB covers setting up a full-stack project using MongoDB, Express, React, and Node.js. It includes installation, configuration, and project setup steps.
  + [MongoDB MERN Tutorial](https://www.mongodb.com/languages/mern-stack-tutorial)
* **MERN Stack Crash Course** by FreeCodeCamp: A thorough video tutorial that builds a simple MERN application, focusing on fundamental concepts and CRUD operations. Good for understanding the stack as a whole.
  + FreeCodeCamp MERN Tutorial

### **2. User Authentication**

* **JWT Authentication in MERN Stack**: Auth0 provides a practical guide to implementing JWT authentication in a MERN app, including how to protect routes based on user roles (admin, teacher, student).
  + Auth0 MERN Authentication Guide
* **Complete Authentication & Authorization Guide**: This Medium article explains setting up user roles and protecting routes using JWT with Express.js, a key part of implementing role-based access control.
  + Medium Authentication Guide

### 3. **Database Modeling with MongoDB and Mongoose**

* **Data Modeling in MongoDB**: This MongoDB documentation page explains schema design patterns, which is useful when setting up complex data models like users, courses, and enrollments.
  + [MongoDB Data Modeling Guide](https://www.mongodb.com/docs/manual/core/data-modeling-introduction/)
* **Using Mongoose for Data Modeling**: This Mongoose tutorial from DigitalOcean covers creating schemas and relationships, as well as defining and managing complex data models.
  + DigitalOcean Mongoose Tutorial

### 4. **Frontend (React) Development and State Management**

* **React and Vite Guide**: This guide from Vite's official documentation covers how to set up React projects with Vite, which optimizes development speed.
  + Vite and React Guide
* **React State Management with Context API**: A well-documented approach to handling global state in React, especially useful for managing user state (logged-in status, user role).
  + Context API Guide by Kent C. Dodds

### 5. **Video Conferencing with WebRTC or Zoom**

* **WebRTC Fundamentals**: This guide on WebRTC from MDN explains the core concepts needed for implementing real-time video and audio streaming between users.
  + [MDN WebRTC Guide](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Signaling_and_video_calling)
* **Zoom API Documentation**: The Zoom API documentation provides details on integrating Zoom into your application, including authentication, session management, and embedding meeting links.
  + Zoom API Documentation

### 6. **Payment Integration**

* **Stripe Payment Integration**: Stripe provides a comprehensive guide and sample code for integrating payment gateways in web apps, covering everything from frontend integration to backend verification.
  + [Stripe API Documentation](https://stripe.com/docs/api)
* **Razorpay Payment Gateway Integration**: This guide covers how to integrate Razorpay, a popular payment gateway in India, into a MERN application.
  + Razorpay Documentation

### 7. **Real-Time Communication and Notifications**

* **Socket.io Documentation**: Socket.io enables real-time bidirectional communication, which is essential for features like chat or live notifications on your learning platform.
  + Socket.io Documentation
* **Integrating Nodemailer for Email Notifications**: This tutorial covers how to set up and use Nodemailer to send email notifications from Node.js, such as for verification emails or course enrollment confirmations.
  + Nodemailer Guide

### 8. **Deployment and CI/CD**

* **Deploying MERN Stack on Heroku**: Heroku’s documentation guides you through deploying full-stack applications, handling environment variables, and managing deployment workflows.
  + Heroku Deployment Guide
* **CI/CD with GitHub Actions**: This GitHub Actions documentation explains how to set up automated workflows for building, testing, and deploying your application.
  + [GitHub Actions Guide](https://docs.github.com/en/actions)

### 9. **Additional Resources**

* **React + Redux for Complex State Management**: If your project has complex state management needs, Redux can help manage the state globally across the platform.
  + Redux Documentation
* **User Interface Design with Tailwind CSS**: Tailwind’s documentation provides guides on responsive and customizable UI components, making it ideal for consistent styling.
  + [Tailwind CSS Documentation](https://tailwindcss.com/docs)

These resources should give you a comprehensive understanding of each component and technology needed to build a fully functional, scalable, and user-friendly online learning platform using the MERN stack.

Top of Form

Bottom of Form