MIZAN INSTITUTE OF TECHNOLOGY (MiT)

Full Stack (MERN) Web Development Internship Program

Complete Development Documentation

Dynamic Website & Learning Management System (LMS)

Project Scope:
Production-Level Client Management System
Advanced Learning Management Platform

Technology Stack:
MERN (MongoDB, Express.js, React.js, Node.js) and many more

June 16, 2025

https://mizantechinstitute.com/ contact@mizantechinstitute.com / +2519 87 14 3030 / P.O. Box: 24659 Addis Ababa, Ethiopia

Contents

CHAPTER 1

Executive Summary

1.1 Project Overview

This entire document is the whole guide to MiT's Full Stack Development internship program. Students will design two interdependent systems:

1. **Dynamic Content Management Website** - A website designed for clients with a basic admin panel for content management
2. **Advanced Learning Management System (LMS)** - An expansive edtech solution based on industry leaders like LearnWorlds

1.2 Learning Objectives

By completing this internship program, students will master:

Core Competencies

- Full-stack development using MERN stack
- RESTful API design and implementation
- Database architecture and optimization
- User authentication and authorization
- Content management systems
- Learning management platforms
- Production deployment strategies
- Modern UI/UX design principles

1.3 Project Timeline

Phase	Duration	Deliverables
1	Week 1-2	Project Setup, Figma Design, & Database Design
2	Week 3-4	Backend API Development
3	Week 5-6	Frontend Development
4	Week 7-8	LMS Core Features
5	Week 9-10	Advanced Features & Integration
6	Week 11-12	Testing, Optimization & Deployment

CHAPTER 2

System Architecture & Technology Stack

2.1 Technology Stack Overview

2.1.1 Frontend Technologies

Frontend Stack

- **React.js 18+** - Core frontend framework
- **TypeScript** - Type safety and better development experience
- **Material-UI (MUI)** or **Ant Design** - UI component library
- **React Router** - Client-side routing
- **Redux Toolkit** - State management
- **React Query** - Server state management
- **Formik + Yup** - Form handling and validation
- **Chart.js/Recharts** - Data visualization
- **Socket.io-client** - Real-time communication

2.1.2 Backend Technologies

Backend Stack

- **Node.js** - Runtime environment
- **Express.js** - Web application framework
- **TypeScript** - Type safety for backend
- **MongoDB with Mongoose** - Database and ODM
- **JWT** - Authentication tokens
- **Bcrypt** - Password hashing
- **Multer** - File upload handling
- **Socket.io** - Real-time communication
- **Nodemailer** - Email services
- **Joi** - Input validation

2.1.3 Development & Deployment Tools

DevOps & Tools

- **Git & GitHub** - Version control
- **Docker** - Containerization
- **AWS/Digital Ocean** - Cloud deployment
- **Nginx** - Reverse proxy and load balancing
- **PM2** - Process management
- **Jest & Supertest** - Testing frameworks
- **ESLint & Prettier** - Code quality

2.2 System Architecture

2.2.1 High-Level Architecture

The system follows a microservices-inspired modular monolith architecture:

1. **Presentation Layer** - React.js frontend applications
2. **API Gateway Layer** - Express.js routing and middleware

3. **Business Logic Layer** - Service classes and controllers
4. **Data Access Layer** - Mongoose models and repositories
5. **Database Layer** - MongoDB with proper indexing
6. **External Services** - Email, file storage, payment processing

CHAPTER 3

Database Design & Schema

3.1 Database Architecture

3.1.1 MongoDB Collections Overview

Collection	Purpose	Key Fields
users	User management	_id, email, password, role, profile, status
courses	Course information	_id, title, description, instructor, modules, pricing
modules	Course modules	_id, courseId, title, lessons, order, status
lessons	Individual lessons	_id, moduleId, title, content, type, duration
enrollments	User enrollments	_id, userId, courseId, progress, status, enrolledAt
assignments	Course assignments	_id, courseId, moduleId, title, description, dueDate
submissions	Assignment submissions	_id, assignmentId, userId, content, grade, submittedAt
discussions	Course discussions	_id, courseId, userId, title, content, replies
notifications	User notifications	_id, userId, type, title, message, isRead
payments	Payment records	_id, userId, courseId, amount, status, method
content	CMS content	_id, type, title, content, slug, status, author
media	File management	_id, filename, originalName, mimetype, size, url

3.1.2 Detailed Schema Definitions

User Schema

```
const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  role: {
    type: String,
    enum: ['student', 'instructor', 'admin', 'content_manager'],
    default: 'student'
  },
  avatar: { type: String },
  bio: { type: String },
  dateOfBirth: { type: Date },
  phone: { type: String },
  address: {
    street: String,
    city: String,
    state: String,
    zipCode: String,
    country: String
  },
  preferences: {
    language: { type: String, default: 'en' },
    timezone: { type: String, default: 'UTC' },
    notifications: {
      email: { type: Boolean, default: true },
      push: { type: Boolean, default: true }
    }
  },
  isActive: { type: Boolean, default: true },
  lastLogin: { type: Date },
  emailVerified: { type: Boolean, default: false },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});
```

Course Schema

```
const courseSchema = new mongoose.Schema({
  title: { type: String, required: true },
  slug: { type: String, required: true, unique: true },
  description: { type: String, required: true },
  shortDescription: { type: String },
  thumbnail: { type: String },
  instructor: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  category: { type: String, required: true },
  subcategory: { type: String },
  level: { type: String, enum: ['beginner', 'intermediate', 'advanced'] },
  language: { type: String, default: 'en' },
  pricing: {
    type: { type: String, enum: ['free', 'paid', 'subscription'] },
    amount: { type: Number, default: 0 },
    currency: { type: String, default: 'USD' },
    discount: {
      percentage: Number,
      validUntil: Date
    }
  },
  duration: { type: Number }, // in minutes
  modules: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Module' }],
  tags: [{ type: String }],
  prerequisites: [{ type: String }],
  learningOutcomes: [{ type: String }],
  status: {
    type: String,
    enum: ['draft', 'published', 'archived'],
    default: 'draft'
  },
  featured: { type: Boolean, default: false },
  rating: {
    average: { type: Number, default: 0 },
    count: { type: Number, default: 0 }
  },
  enrollmentCount: { type: Number, default: 0 },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});
```

CHAPTER 4

Backend Development

4.1 Project Structure

Backend Directory Structure

```
backend/
 src/
    config/
       database.ts
       server.ts
       environment.ts
    controllers/
       auth.controller.ts
       user.controller.ts
       course.controller.ts
       content.controller.ts
       dashboard.controller.ts
    middleware/
       auth.middleware.ts
       validation.middleware.ts
       upload.middleware.ts
       error.middleware.ts
    models/
       User.model.ts
       Course.model.ts
       Module.model.ts
       Lesson.model.ts
       Content.model.ts
    routes/
       auth.routes.ts
       user.routes.ts
       course.routes.ts
       admin.routes.ts
       content.routes.ts
    services/
       auth.service.ts
       email.service.ts
       upload.service.ts
       payment.service.ts
    utils/
       validators.ts
       helpers.ts
       constants.ts
    types/
       index.ts
    app.ts
 tests/
 uploads/
 package.json
 tsconfig.json
 Dockerfile
```

4.2 API Endpoints

4.2.1 Authentication Endpoints

Method	Endpoint	Description
POST	/api/auth/register	User registration with email verification
POST	/api/auth/login	User login with JWT token generation
POST	/api/auth/logout	User logout and token invalidation
POST	/api/auth/forgot-password	Password reset request
POST	/api/auth/reset-password	Password reset confirmation
GET	/api/auth/verify-email/:token	Email verification
POST	/api/auth/refresh-token	JWT token refresh
GET	/api/auth/me	Get current user profile

4.2.2 Course Management Endpoints

Method	Endpoint	Description
GET	/api/courses	Get all published courses with filters
GET	/api/courses/:id	Get specific course details
POST	/api/courses	Create new course (instructor/admin)
PUT	/api/courses/:id	Update course details
DELETE	/api/courses/:id	Delete course
POST	/api/courses/:id/enroll	Enroll in a course
GET	/api/courses/:id/modules	Get course modules
POST	/api/courses/:id/modules	Add module to course
GET	/api/courses/:id/progress	Get user progress in course
POST	/api/courses/:id/review	Add course review/rating

4.2.3 Content Management Endpoints

Method	Endpoint	Description
GET	/api/content	Get all content items
GET	/api/content/:slug	Get content by slug
POST	/api/content	Create new content
PUT	/api/content/:id	Update content
DELETE	/api/content/:id	Delete content
POST	/api/content/upload	Upload media files
GET	/api/content/media	Get media library

4.3 Authentication & Authorization

4.3.1 JWT Implementation

JWT Service Implementation

```
import jwt from 'jsonwebtoken';
import { User } from '../models/User.model';

export class AuthService {
  static generateTokens(userId: string) {
    const accessToken = jwt.sign(
      { userId, type: 'access' },
      process.env.JWT_ACCESS_SECRET!,
      { expiresIn: '15m' }
    );

    const refreshToken = jwt.sign(
      { userId, type: 'refresh' },
      process.env.JWT_REFRESH_SECRET!,
      { expiresIn: '7d' }
    );

    return { accessToken, refreshToken };
  }

  static async verifyToken(token: string, type: 'access' | 'refresh') {
    const secret = type === 'access'
      ? process.env.JWT_ACCESS_SECRET!
      : process.env.JWT_REFRESH_SECRET!;

    const decoded = jwt.verify(token, secret) as any;
    const user = await User.findById(decoded.userId);

    if (!user || !user.isActive) {
      throw new Error('Invalid token');
    }

    return user;
  }
}
```

4.3.2 Role-Based Access Control

Authorization Middleware

```
export const authorize = (roles: string[]) => {
  return async (req: Request, res: Response, next: NextFunction) => {
    try {
      const token = req.header('Authorization')?.replace('Bearer ', '');

      if (!token) {
        return res.status(401).json({ message: 'Access denied' });
      }

      const user = await AuthService.verifyToken(token, 'access');

      if (!roles.includes(user.role)) {
        return res.status(403).json({ message: 'Insufficient permissions' });
      }

      req.user = user;
      next();
    } catch (error) {
      res.status(401).json({ message: 'Invalid token' });
    }
  };
};
```

CHAPTER 5

Frontend Development

5.1 React Application Structure

Frontend Directory Structure

```
frontend/
 public/
    index.html
    manifest.json
 src/
    components/
       common/
          Header.tsx
          Footer.tsx
          Sidebar.tsx
          Layout.tsx
       forms/
          LoginForm.tsx
          CourseForm.tsx
          ContentForm.tsx
       ui/
           Button.tsx
           Modal.tsx
           DataTable.tsx
    pages/
       auth/
          Login.tsx
          Register.tsx
       dashboard/
          Dashboard.tsx
          CourseDashboard.tsx
          AdminDashboard.tsx
       courses/
          CourseList.tsx
          CourseDetail.tsx
          CoursePlayer.tsx
       cms/
           ContentManager.tsx
           MediaLibrary.tsx
    hooks/
       useAuth.ts
       useCourses.ts
       useContent.ts
    services/
       api.ts
       auth.service.ts
       course.service.ts
    store/
       index.ts
       authSlice.ts
       courseSlice.ts
    types/
       index.ts
```

5.2 Key Frontend Components

5.2.1 Main Application Layout

Layout Component

```
import React from 'react';
import { Layout as AntLayout, Menu } from 'antd';
import { useAuth } from '../hooks/useAuth';
import Header from './Header';
import Sidebar from './Sidebar';

const { Content, Sider } = AntLayout;

interface LayoutProps {
  children: React.ReactNode;
}

export const Layout: React.FC<LayoutProps> = ({ children }) => {
  const { user } = useAuth();
  const [collapsed, setCollapsed] = React.useState(false);

  return (
    <AntLayout style={{ minHeight: '100vh' }}>
      <Header />
      <AntLayout>
        <Sider
          collapsible
          collapsed={collapsed}
          onCollapse={setCollapsed}
          theme="light"
        >
          <Sidebar userRole={user?.role} />
        </Sider>
        <Content style={{ margin: '24px 16px', padding: 24 }}>
          {children}
        </Content>
      </AntLayout>
    </AntLayout>
  );
};
```

5.2.2 Course Management Interface

Course List Component

```
import React, { useState, useEffect } from 'react';
import { Table, Button, Space, Tag, Modal } from 'antd';
import { PlusOutlined, EditOutlined, DeleteOutlined } from '@ant-design/icons';
import { useCourses } from '../hooks/useCourses';
import { Course } from '../types';

export const CourseList: React.FC = () => {
  const { courses, loading, createCourse, updateCourse, deleteCourse } = useCourses
  const [modalVisible, setModalVisible] = useState(false);
  const [editingCourse, setEditingCourse] = useState<Course | null>(null);

  const columns = [
    {
      title: 'Title',
      dataIndex: 'title',
      key: 'title',
    },
    {
      title: 'Category',
      dataIndex: 'category',
      key: 'category',
    },
    {
      title: 'Status',
      dataIndex: 'status',
      key: 'status',
      render: (status: string) => (
        <Tag color={status === 'published' ? 'green' : 'orange'}>
          {status.toUpperCase()}
        </Tag>
      ),
    },
    {
      title: 'Enrolled',
      dataIndex: 'enrollmentCount',
      key: 'enrollmentCount',
    },
    {
      title: 'Actions',
      key: 'actions',
      render: (_, record: Course) => (
        <Space size="middle">
          <Button
            icon={<EditOutlined />}
            onClick={() => handleEdit(record)}
          >
            Edit
          </Button>
          <Button
```

CHAPTER 6

Learning Management System Features

6.1 Core LMS Features

6.1.1 Feature Comparison with Industry Leaders

Feature	LearnWorlds	Teachable	Our LMS	Implementation Notes
Course Builder				Drag-and-drop interface with React DnD
Video Streaming				Integration with Vimeo/YouTube API
Interactive Videos				Custom video player with annotations
Assessments & Quizzes				Multiple question types, auto-grading
Certificates				PDF generation with digital signatures
Discussion Forums				Real-time chat with Socket.io
Live Sessions				WebRTC integration for video calls
Progress Tracking				Detailed analytics dashboard
Mobile Learning				Progressive Web App (PWA)
Gamification				Points, badges, leaderboards
White Labeling				Custom branding and themes

Feature	LearnWorlds	Teachable	Our LMS	Implementation Notes
Drip Content				Scheduled content release
Multi-language				i18n with React Intl
Payment Integration				Stripe, PayPal integration
Affiliate System				Commission tracking system

6.1.2 Advanced LMS Features Implementation

Course Builder System

The course builder will feature:

- Drag-and-drop module organization
- Rich text editor for lesson content
- Video upload and streaming integration
- Interactive elements (quizzes, polls, assignments)
- Progress tracking and completion criteria
- Conditional content release

Course Builder Component Structure

```
// Course Builder Main Component
export const CourseBuilder: React.FC = () => {
  const [course, setCourse] = useState<Course | null>(null);
  const [modules, setModules] = useState<Module[]>([]);
  const [activeModule, setActiveModule] = useState<string | null>(null);

  return (
    <DndProvider backend={HTML5Backend}>
      <Layout className="course-builder">
        <Sider width={300}>
          <ModulesSidebar
            modules={modules}
            onModuleSelect={setActiveModule}
            onModulesReorder={setModules}
          />
        </Sider>
        <Content>
          <ModuleEditor
            moduleId={activeModule}
            onSave={handleModuleSave}
          />
        </Content>
      </Layout>
    </DndProvider>
  );
};

// Module Editor with Rich Content
export const ModuleEditor: React.FC<{moduleId: string}> = ({ moduleId }) => {
  const [lessons, setLessons] = useState<Lesson[]>([]);

  return (
    <div className="module-editor">
      <LessonList
        lessons={lessons}
        onLessonAdd={handleAddLesson}
        onLessonReorder={handleReorderLessons}
      />
      <LessonContentEditor
        onContentChange={handleContentChange}
      />
    </div>
  );
};
```

6.2 Video Learning Platform

6.2.1 Video Player Implementation

Custom Video Player

```
import React, { useRef, useState, useEffect } from 'react';
import { Progress, Button, Tooltip } from 'antd';
import { PlayCircleOutlined, PauseOutlined, FullscreenOutlined } from '@ant-design/

interface VideoPlayerProps {
  src: string;
  title: string;
  onProgress: (progress: number) => void;
  onComplete: () => void;
}

export const VideoPlayer: React.FC<VideoPlayerProps> = ({
  src, title, onProgress, onComplete
}) => {
  const videoRef = useRef<HTMLVideoElement>(null);
  const [isPlaying, setIsPlaying] = useState(false);
  const [progress, setProgress] = useState(0);
  const [duration, setDuration] = useState(0);
  const [currentTime, setCurrentTime] = useState(0);

  useEffect(() => {
    const video = videoRef.current;
    if (!video) return;

    const handleTimeUpdate = () => {
      const current = video.currentTime;
      const total = video.duration;
      const progressPercent = (current / total) * 100;

      setCurrentTime(current);
      setProgress(progressPercent);
      onProgress(progressPercent);

      // Mark as complete when 90% watched
      if (progressPercent >= 90 && !video.ended) {
        onComplete();
      }
    };

    const handleLoadedMetadata = () => {
      setDuration(video.duration);
    };

    video.addEventListener('timeupdate', handleTimeUpdate);
    video.addEventListener('loadedmetadata', handleLoadedMetadata);

    return () => {
      video.removeEventListener('timeupdate', handleTimeUpdate);
      video.removeEventListener('loadedmetadata', handleLoadedMetadata);
```

6.3 Assessment System

6.3.1 Quiz and Assignment Management

Quiz Component Implementation

```
interface Question {
  id: string;
  type: 'multiple-choice' | 'true-false' | 'short-answer' | 'essay';
  question: string;
  options?: string[];
  correctAnswer: string | string[];
  points: number;
  explanation?: string;
}

interface Quiz {
  id: string;
  title: string;
  description: string;
  questions: Question[];
  timeLimit?: number;
  attempts: number;
  passingScore: number;
}

export const QuizTaker: React.FC<{quiz: Quiz}> = ({ quiz }) => {
  const [currentQuestion, setCurrentQuestion] = useState(0);
  const [answers, setAnswers] = useState<Record<string, any>>({});
  const [timeRemaining, setTimeRemaining] = useState(quiz.timeLimit || 0);
  const [isSubmitted, setIsSubmitted] = useState(false);

  useEffect(() => {
    if (quiz.timeLimit && timeRemaining > 0 && !isSubmitted) {
      const timer = setTimeout(() => {
        setTimeRemaining(prev => prev - 1);
      }, 1000);
      return () => clearTimeout(timer);
    } else if (timeRemaining === 0 && !isSubmitted) {
      handleSubmit();
    }
  }, [timeRemaining, isSubmitted]);

  const handleAnswerChange = (questionId: string, answer: any) => {
    setAnswers(prev => ({
      ...prev,
      [questionId]: answer
    }));
  };

  const handleSubmit = async () => {
    try {
      const submission = {
        quizId: quiz.id,
        answers,
```

CHAPTER 7

Content Management System

7.1 Dynamic Content Management

7.1.1 CMS Architecture

The Content Management System allows non-technical users to manage website content through an intuitive interface:

CMS Core Features

- Rich text editor with media embedding
- Drag-and-drop page builder components
- SEO optimization tools
- Multi-language content support
- Version control and content scheduling
- Media library with image optimization
- User role-based content permissions

7.1.2 Page Builder Implementation

Page Builder Components

```
interface ContentBlock {
  id: string;
  type: string;
  content: any;
  styles: Record<string, any>;
  order: number;
}

interface PageBuilderProps {
  initialBlocks?: ContentBlock[];
  onSave: (blocks: ContentBlock[]) => void;
}

export const PageBuilder: React.FC<PageBuilderProps> = ({
  initialBlocks = [],
  onSave
}) => {
  const [blocks, setBlocks] = useState<ContentBlock[]>(initialBlocks);
  const [draggedBlock, setDraggedBlock] = useState<ContentBlock | null>(null);

  const blockTypes = [
    { type: 'text', label: 'Text Block', icon: <FileTextOutlined /> },
    { type: 'image', label: 'Image', icon: <PictureOutlined /> },
    { type: 'video', label: 'Video', icon: <PlayCircleOutlined /> },
    { type: 'gallery', label: 'Gallery', icon: <AppstoreOutlined /> },
    { type: 'form', label: 'Contact Form', icon: <FormOutlined /> },
    { type: 'testimonial', label: 'Testimonial', icon: <MessageOutlined /> },
  ];

  const addBlock = (type: string) => {
    const newBlock: ContentBlock = {
      id: `block_${Date.now()}`,
      type,
      content: getDefaultContent(type),
      styles: {},
      order: blocks.length
    };
    setBlocks([...blocks, newBlock]);
  };

  const updateBlock = (blockId: string, updates: Partial<ContentBlock>) => {
    setBlocks(blocks.map(block =>
      block.id === blockId ? { ...block, ...updates } : block
    ));
  };

  const deleteBlock = (blockId: string) => {
    setBlocks(blocks.filter(block => block.id !== blockId));
  };
```

7.2 Media Management

7.2.1 File Upload and Management

Media Library Implementation

```
export const MediaLibrary: React.FC = () => {
  const [files, setFiles] = useState<MediaFile[]>([]);
  const [loading, setLoading] = useState(false);
  const [selectedFiles, setSelectedFiles] = useState<string[]>([]);

  const uploadProps: UploadProps = {
    name: 'files',
    action: '/api/content/upload',
    headers: {
      authorization: `Bearer ${getAuthToken()}`,
    },
    multiple: true,
    accept: 'image/*,video/*,audio/*,.pdf,.doc,.docx',
    onChange: handleUploadChange,
    onDrop: handleDrop,
  };

  const handleUploadChange = (info: UploadChangeParam) => {
    const { status } = info.file;
    if (status === 'done') {
      message.success(`${info.file.name} uploaded successfully.`);
      fetchFiles(); // Refresh file list
    } else if (status === 'error') {
      message.error(`${info.file.name} upload failed.`);
    }
  };

  const handleDelete = async (fileId: string) => {
    try {
      await deleteFile(fileId);
      setFiles(files.filter(file => file.id !== fileId));
      message.success('File deleted successfully');
    } catch (error) {
      message.error('Failed to delete file');
    }
  };

  const renderFileCard = (file: MediaFile) => (
    <Card
      key={file.id}
      hoverable
      className="file-card"
      cover={
        file.type.startsWith('image/') ? (
          <img alt={file.name} src={file.url} />
        ) : (
          <div className="file-icon">
            <FileOutlined style={{ fontSize: 48 }} />
          </div>
```

CHAPTER 8

User Dashboards

8.1 Student Dashboard

8.1.1 Learning Progress Tracking

Student Dashboard Implementation

```
export const StudentDashboard: React.FC = () => {
  const { user } = useAuth();
  const [enrolledCourses, setEnrolledCourses] = useState<EnrolledCourse[]>([]);
  const [recentActivity, setRecentActivity] = useState<Activity[]>([]);
  const [achievements, setAchievements] = useState<Achievement[]>([]);

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

  const fetchDashboardData = async () => {
    try {
      const [coursesRes, activityRes, achievementsRes] = await Promise.all([
        getEnrolledCourses(),
        getRecentActivity(),
        getUserAchievements()
      ]);

      setEnrolledCourses(coursesRes.data);
      setRecentActivity(activityRes.data);
      setAchievements(achievementsRes.data);
    } catch (error) {
      message.error('Failed to load dashboard data');
    }
  };

  const calculateOverallProgress = () => {
    if (enrolledCourses.length === 0) return 0;
    const totalProgress = enrolledCourses.reduce((sum, course) =>
      sum + course.progress, 0);
    return Math.round(totalProgress / enrolledCourses.length);
  };

  return (
    <div className="student-dashboard">
      <Row gutter={[16, 16]}>
        {/* Welcome Section */}
        <Col span={24}>
          <Card className="welcome-card">
            <div className="welcome-content">
              <Avatar size={64} src={user?.avatar} />
              <div className="welcome-text">
                <h2>Welcome back, {user?.firstName}!</h2>
                <p>Continue your learning journey</p>
              </div>
            </div>
          </Card>
        </Col>
```

8.2 Instructor Dashboard

8.2.1 Course Management and Analytics

Instructor Dashboard

```
export const InstructorDashboard: React.FC = () => {
  const [courses, setCourses] = useState<Course[]>([]);
  const [analytics, setAnalytics] = useState<InstructorAnalytics | null>(null);
  const [earnings, setEarnings] = useState<EarningsData | null>(null);

  const analyticsData = [
    {
      title: 'Total Students',
      value: analytics?.totalStudents || 0,
      icon: <UserOutlined />,
      color: '#1890ff'
    },
    {
      title: 'Total Revenue',
      value: earnings?.totalRevenue || 0,
      prefix: ',
      icon: <DollarOutlined />,
      color: '#52c41a'
    },
    {
      title: 'Course Rating',
      value: analytics?.averageRating || 0,
      precision: 1,
      suffix: '/5',
      icon: <StarOutlined />,
      color: '#faad14'
    },
    {
      title: 'Completion Rate',
      value: analytics?.completionRate || 0,
      suffix: '%',
      icon: <CheckCircleOutlined />,
      color: '#722ed1'
    }
  ];

  return (
    <div className="instructor-dashboard">
      <PageHeader
        title="Instructor Dashboard"
        extra={[
          <Button key="create" type="primary" icon={<PlusOutlined />}>
            Create New Course
          </Button>
        ]}
      />

      {/* Analytics Overview */}
      <Row gutter={[16, 16]} style={{ marginBottom: 24 }}>
```

8.3 Admin Dashboard

8.3.1 System Oversight and Management

The Admin Dashboard provides a top-level view of the entire platform. Administrators can manage users, oversee all courses, monitor site health, and configure system-wide settings.

Admin Key Features

- **Global Analytics:** View platform-wide statistics like user growth, total enrollments, and revenue.
- **User Management:** Search, view, edit, and manage all users (students, instructors, admins).
- **Course Management:** Oversee all courses, approve new courses, and manage categories.
- **Content Approval Workflow:** Review and approve content submitted by Content Managers.
- **System Health Monitoring:** Check server status, database connections, and API health.
- **Site Configuration:** Manage site settings, payment gateways, and email templates.

Admin Dashboard Implementation

```
export const AdminDashboard: React.FC = () => {
  const [stats, setStats] = useState<AdminStats | null>(null);
  const [users, setUsers] = useState<User[]>([]);

  const statCards = [
    { title: "Total Users", value: stats?.totalUsers, icon: <UserOutlined /> },
    { title: "Total Courses", value: stats?.totalCourses, icon: <BookOutlined /> },
    { title: "Total Revenue", value: stats?.totalRevenue, prefix: '$', icon: <Dolla
    { title: "Pending Approvals", value: stats?.pendingApprovals, icon: <ClockCircl
  ];

  const userColumns = [
    { title: 'Name', dataIndex: 'name', key: 'name' },
    { title: 'Email', dataIndex: 'email', key: 'email' },
    { title: 'Role', dataIndex: 'role', key: 'role',
      render: (role) => <Tag>{role.toUpperCase()}</Tag> },
    { title: 'Joined', dataIndex: 'createdAt', key: 'createdAt' },
    { title: 'Status', dataIndex: 'isActive', key: 'status',
      render: (isActive) => <Tag color={isActive ? 'green' : 'red'}>
          {isActive ? 'ACTIVE' : 'INACTIVE'}
        </Tag>
    },
    { title: 'Actions', key: 'actions',
      render: (_, record) => <Space>
        <Button icon={<EditOutlined />}>Edit</Button>
        <Button danger icon={<UserDeleteOutlined />}>Disable</Button>
      </Space>
    }
  ];

  return (
    <div className="admin-dashboard">
      <PageHeader title="Administrator Dashboard" />

      {/* Stats Cards */}
      <Row gutter={[16, 16]}>
        {statCards.map(card => (
          <Col xs={24} sm={12} lg={6} key={card.title}>
            <Card>
              <Statistic
                title={card.title}
                value={card.value}
                prefix={card.prefix || card.icon}
              />
            </Card>
          </Col>
        ))}
      </Row>

      {/* Charts */}
      <Row gutter={[16, 16]} style={{ marginTop: 24 }}>
```

CHAPTER 9

Testing Strategy

9.1 Overview

A robust testing strategy is crucial for building a reliable and maintainable application. We will employ a multi-layered testing approach, including unit, integration, and end-to-end (E2E) tests.

Testing Pyramid

Our testing strategy follows the testing pyramid principle:

- **Unit Tests (Base):** Test individual functions and components in isolation. Fast and numerous.
- **Integration Tests (Middle):** Test the interaction between multiple components (e.g., API controller and service).
- **E2E Tests (Top):** Test complete user flows from the user's perspective. Slower and fewer.

9.2 Backend Testing

We will use **Jest** as our testing framework and **Supertest** for testing HTTP endpoints.

Backend Unit Test Example (Jest)

```
// src/services/auth.service.test.ts
import { AuthService } from './auth.service';
import jwt from 'jsonwebtoken';

jest.mock('jsonwebtoken');

describe('AuthService', () => {
  describe('generateTokens', () => {
    it('should generate access and refresh tokens', () => {
      const userId = 'someUserId';
      (jwt.sign as jest.Mock)
        .mockReturnValueOnce('fakeAccessToken')
        .mockReturnValueOnce('fakeRefreshToken');

      const tokens = AuthService.generateTokens(userId);

      expect(tokens).toEqual({
        accessToken: 'fakeAccessToken',
        refreshToken: 'fakeRefreshToken',
      });
      expect(jwt.sign).toHaveBeenCalledTimes(2);
    });
  });
});
```

Backend Integration Test Example (Jest + Supertest)

```
// src/routes/auth.routes.test.ts
import request from 'supertest';
import app from '../app';
import { User } from '../models/User.model';

describe('POST /api/auth/register', () => {
  it('should register a new user successfully', async () => {
    const userData = {
      email: 'test@example.com',
      password: 'password123',
      firstName: 'Test',
      lastName: 'User',
    };

    const response = await request(app)
      .post('/api/auth/register')
      .send(userData)
      .expect(201);

    expect(response.body.message).toBe('Registration successful');
    const dbUser = await User.findOne({ email: userData.email });
    expect(dbUser).not.toBeNull();
  });
});
```

9.3 Frontend Testing

We will use **React Testing Library** with **Jest** to test our React components, focusing on user behavior rather than implementation details.

Frontend Component Test Example (React Testing Library)

```
// src/components/forms/LoginForm.test.tsx
import { render, screen, fireEvent } from '@testing-library/react';
import { LoginForm } from './LoginForm';

describe('LoginForm', () => {
  it('should display validation errors for empty fields', async () => {
    const mockOnSubmit = jest.fn();
    render(<LoginForm onSubmit={mockOnSubmit} />);

    fireEvent.click(screen.getByRole('button', { name: /log in/i }));

    expect(await screen.findByText('Email is required')).toBeInTheDocument();
    expect(await screen.findByText('Password is required')).toBeInTheDocument();
    expect(mockOnSubmit).not.toHaveBeenCalled();
  });

  it('should call onSubmit with form data for valid input', async () => {
    const mockOnSubmit = jest.fn();
    render(<LoginForm onSubmit={mockOnSubmit} />);

    fireEvent.change(screen.getByLabelText(/email/i), {
      target: { value: 'test@example.com' }
    });
    fireEvent.change(screen.getByLabelText(/password/i), {
      target: { value: 'password123' }
    });
    fireEvent.click(screen.getByRole('button', { name: /log in/i }));

    await waitFor(() => {
        expect(mockOnSubmit).toHaveBeenCalledWith({
            email: 'test@example.com',
            password: 'password123',
        });
    });
  });
});
```

9.4 End-to-End (E2E) Testing

We will use **Cypress** for E2E testing to simulate real user scenarios across the entire application stack.

E2E Test Example (Cypress)

```
// cypress/integration/enrollment.spec.js
describe('Course Enrollment Flow', () => {
  it('allows a logged-in user to enroll in a course', () => {
    cy.login('student@mit.com', 'password'); // Custom login command

    cy.visit('/courses');
    cy.contains('MERN Full Stack Development').click();

    cy.url().should('include', '/courses/mern-full-stack');
    cy.get('button').contains('Enroll Now').click();

    cy.contains('Successfully enrolled!').should('be.visible');
    cy.url().should('include', '/learning/mern-full-stack');
  });
});
```

CHAPTER 10

Deployment & DevOps

10.1 Environment Configuration

We will use ‘.env‘ files to manage environment-specific variables for development, staging, and production. Never commit ‘.env‘ files to version control.

Example .env file

```
# Server Configuration
NODE_ENV=development
PORT=5000

# Database
MONGO_URI=mongodb://localhost:27017/mit_lms_dev

# JWT Secrets
JWT_ACCESS_SECRET=your_super_secret_access_key
JWT_REFRESH_SECRET=your_super_secret_refresh_key

# Email Service
EMAIL_HOST=smtp.example.com
EMAIL_USER=user@example.com
EMAIL_PASS=password
```

10.2 Containerization with Docker

We will use Docker to create consistent, isolated environments for each part of our application.

Backend Dockerfile

```
# Stage 1: Build
FROM node:18-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Stage 2: Production
FROM node:18-alpine
WORKDIR /app
COPY --from=builder /app/package*.json ./
RUN npm install --only=production
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/public ./public

EXPOSE 5000
CMD ["node", "dist/app.js"]
```

Frontend Dockerfile

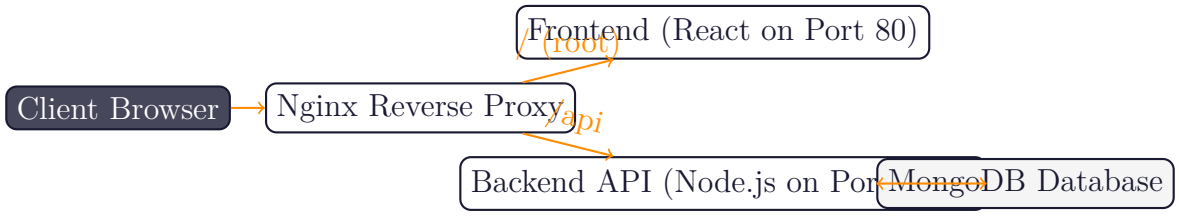
```
# Stage 1: Build
FROM node:18-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . ./
RUN npm run build

# Stage 2: Serve
FROM nginx:stable-alpine
COPY --from=build /app/build /usr/share/nginx/html
# Copy nginx config if you have one
# COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Docker Compose

```
version: '3.8'

services:
  mongodb:
    image: mongo:latest
    container_name: mongodb
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db

  backend:
    build: ./backend
    container_name: backend-api
    ports:
      - "5000:5000"
    depends_on:
      - mongodb
    environment:
      - MONGO_URI=mongodb://mongodb:27017/mit_lms
      - PORT=5000
      # Add other env vars here

  frontend:
    build: ./frontend
    container_name: frontend-client
    ports:
      - "3000:80" # Map host port 3000 to container port 80
    depends_on:
      - backend

volumes:
  mongo-data:
```

10.3 Production Deployment

The application will be deployed to a cloud provider (e.g., AWS, DigitalOcean) using Nginx as a reverse proxy and PM2 for process management if not using containers.



10.4 CI/CD with GitHub Actions

A basic CI/CD pipeline will be set up to automate testing and building on every push to the main branch.

GitHub Actions CI Workflow

```
name: Node.js CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Use Node.js 18.x
      uses: actions/setup-node@v3
      with:
        node-version: 18.x
        cache: 'npm'
        cache-dependency-path: backend/package-lock.json

    - name: Install Dependencies
      run: cd backend && npm install

    - name: Run Linter
      run: cd backend && npm run lint

    - name: Run Tests
      run: cd backend && npm test
      env:
        CI: true
```

CHAPTER 11

Security Best Practices

Security is Paramount!

Security is not an afterthought; it must be integrated throughout the development lifecycle. A single vulnerability can compromise the entire system and its user data.

11.1 Backend Security

- **Input Validation:** Use libraries like **Joi** or **express-validator** to validate and sanitize all incoming data from requests.
- **Authentication & Authorization:** Implement robust JWT-based authentication with short-lived access tokens and secure refresh tokens. Enforce role-based access control (RBAC) on all protected routes.
- **Password Hashing:** Use **bcrypt** to hash and salt user passwords. Never store passwords in plain text.
- **Prevent NoSQL Injection:** Use an ODM like **Mongoose**, which provides schema validation and sanitizes inputs to mitigate NoSQL injection risks.
- **Security Headers:** Use **Helmet.js** to set various HTTP headers that protect against common attacks like clickjacking and cross-site scripting (XSS).
- **Rate Limiting:** Implement rate limiting on sensitive endpoints (e.g., login, password reset) to prevent brute-force attacks.
- **Error Handling:** Avoid leaking sensitive information (like stack traces) in production error messages.

11.2 Frontend Security

- **Cross-Site Scripting (XSS):** React automatically escapes content rendered in JSX, providing strong protection. However, be cautious when using 'dangerously-

SetInnerHTML‘. Sanitize any user-generated content that needs to be rendered as HTML.

- **JWT Storage:** Store JWTs securely. For web, using HttpOnly cookies is generally safer than Local Storage to prevent XSS attacks from stealing tokens.
- **Cross-Site Request Forgery (CSRF):** If using cookie-based sessions, implement CSRF protection using tokens (e.g., ‘csurf‘ package on the backend).
- **Secure API Calls:** Always use HTTPS for all API communications.

11.3 General Security

- **Use HTTPS:** Encrypt all traffic between the client and server using TLS/SSL certificates.
- **Environment Variables:** Store all secrets (API keys, database credentials, JWT secrets) in environment variables, never hard-coded in the source.
- **Dependency Management:** Regularly audit and update project dependencies to patch known vulnerabilities (use ‘npm audit‘).

CHAPTER 12

Conclusion & Future Work

12.1 Project Summary

Congratulations, interns! Through this guide, you will develop a complex, production-grade Full Stack application. You will have created a dynamic, client-focused website with a bespoke CMS driving it and a feature-rich Learning Management System. This project provides you with hands-on exposure to the entire MERN stack, advanced development workflows, and professional best practices.

12.2 Next Steps for Interns

Upon completion, your journey as a developer is just beginning. We encourage you to:

- **Refactor and Optimize:** Review your code. Can you make it more efficient, readable, or scalable?
- **Enhance Testing:** Increase test coverage. Write more integration and E2E tests.
- **Contribute to Documentation:** Update this document with any challenges you faced and the solutions you found.
- **Explore Further:** Investigate the technologies used more deeply. Why was a particular design pattern chosen? What are the alternatives?

12.3 Potential Future Features

This platform is a strong foundation. Future iterations could include:

- **AI-Powered Recommendations:** Suggest courses to students based on their learning history and interests.
- **Native Mobile App:** Develop a React Native application for iOS and Android.
- **Advanced Analytics:** Provide instructors with deeper insights into student engagement and performance.

- **Community Features:** Build social learning features like study groups and direct messaging.
- **Third-Party Integrations:** Integrate with tools like Zapier, Calendly, or other educational platforms.

Final Words

This internship should be enjoyable and challenging. Seize the opportunity to learn, collaborate, and make something amazing. The experience you gain here will be invaluable in your future careers as software developers. Welcome to the team, and happy coding!