**WEB DEVELOPMENT FILE**

**Department of Information Technology**

**Web Development using Mern Stack**





**Submitted By: Tarun Daderwal**

**Enrollment Number: 10813203121**

**Class: IT-2(FSD)**

**Submitted To: Mrs. Kritika Balihar**

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Practical | Date | Signature |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Web dev lab   
1(a) To introduce the 3 tier architecture of mern stack.  
1(b) To elucidate the HTML elements.

Q2(a) To introduce CSS and its types.  
(b) Introduction to Java Script and its types.

**Practical – 3(a)**

**Aim:** Write a programto Create Time Table using HTML.

**Code:**

<!doctype html>

<html lang="en">

<head>

<title> Time-Table </title>

<meta charset="utf-8">

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

<style>

h2{

text-align: center;

}

</style>

</head>

<body>

<h2>Time Table</h2>

<table border="1" style="border-collapse: collapse;" cellspacing="10px" cellpadding="6px" align="center">

<tr>

<td colspan="9" align="center">3rd Year, IT(FSD)</td>

</tr>

<tr width="100%" height="30px" align="center">

<td></td>

<td>08:00-09:00</td>

<td>09:00-10:00</td>

<td>10:00-11:00</td>

<td>11:00-12:00</td>

<td>BREAK</td>

<td>12:30-01:30</td>

<td>01:30-02:30</td>

<td>02:30-03:30</td>

</tr>

<tr align="center">

<td>&nbsp Monday &nbsp </td>

<td colspan="2">Advanced Java Lab\_B <hr>Ms. Gurjapna Kaur</br><br>Lab 12</td>

<td colspan="2">Web Development Lab\_B <hr> Ms. Kritika <br><br> Lab 12</td>

<td>--X--</td>

<td>E-Commerce\_Th <hr> Mr. Kalyan Singh <br><br> Lect 32</td>

<td>PME\_Th <hr> Mr. Sachin<br><br> Lect 34</td>

<td>Web Development\_Th <hr> Ms. Kritika <br><br> Lect 34</td>

</tr>

<tr align="center">

<td>&nbsp Tuesday &nbsp </td>

<td colspan="2">Advanced Java\_Th <hr>Ms. Gurjapna Kaur</br><br>Lect 32</td>

<td>DBM\_Th<hr> Ms. Upasana <br><br> Lect 32</td>

<td>PME\_Th <hr> Mr. Sachin <br><br> Lect 32</td>

<td>--X--</td>

<td>E-Commerce\_Th <hr> Mr. Kalyan Singh <br><br> Lect 34</td>

<td>Web Development\_Th <hr> Ms. Kritika <br><br> Lect 34</td>

<td>Web Technology\_Th <hr> Ms. Gurpreet Kaur<br><br> Lect 33</td>

</tr>

<tr align="center">

<td>&nbsp Wednesday &nbsp </td>

<td colspan="2">Web Development Lab\_B <hr> Ms. Kritika <br><br> Lab 12</td>

<td colspan="2">Web Technology Lab\_B <hr>Ms. Gurpreet Kaur</br><br>Lab 12</td>

<td>--X--</td>

<td>Web Technology\_Th <hr> Ms. Gurpreet Kaur<br><br> Lect 33</td>

<td>E-Commerce\_Th <hr> Mr. Kalyan Singh <br><br> Lect 33</td>

<td>DBM\_Th<hr> Ms. Upasana <br><br> Lect 34</td>

</tr>

<tr align="center">

<td>&nbsp Thursday &nbsp </td>

<td>Web Development\_Th <hr> Ms. Kritika <br><br> Lect 33</td>

<td>DBM\_Th<hr> Ms. Upasana <br><br> Lect 33</td>

<td>Web Technology\_Th <hr> Ms. Gurpreet Kaur<br><br> Lect 32</td>

<td>UHV-1\_Th<hr> Ms. Chetna Nassa <br><br> Lect 32</td>

<td>--X--</td>

<td>PME\_Th <hr> Mr. Sachin<br><br> Lect 32</td>

<td>E-Commerce\_Th <hr> Mr. Kalyan Singh <br><br> Lect 32</td>

<td>-----</td>

</tr>

<tr align="center">

<td>&nbsp Friday &nbsp </td>

<td colspan="2">Advanced Java Lab\_A <hr>Ms. Gurjapna Kaur</br><br>Lab 12</td>

<td>DBM\_Th<hr> Ms. Upasana <br><br> Lect 32</td>

<td>Advanced Java\_Th <hr>Ms. Gurjapna Kaur</br><br>Lect 32</td>

<td>--X--</td>

<td colspan="2">Web Technology Lab\_B <hr> Ms. Gurpreet Kaur <br><br> Lab 12</td>

<td>-----</td>

</tr>

<tr align="center">

<td>&nbsp Saturday &nbsp </td>

<td>-----</td>

<td>-----</td>

<td>-----</td>

<td>-----</td>

<td>--X--</td>

<td>-----</td>

<td>-----</td>

<td>-----</td>

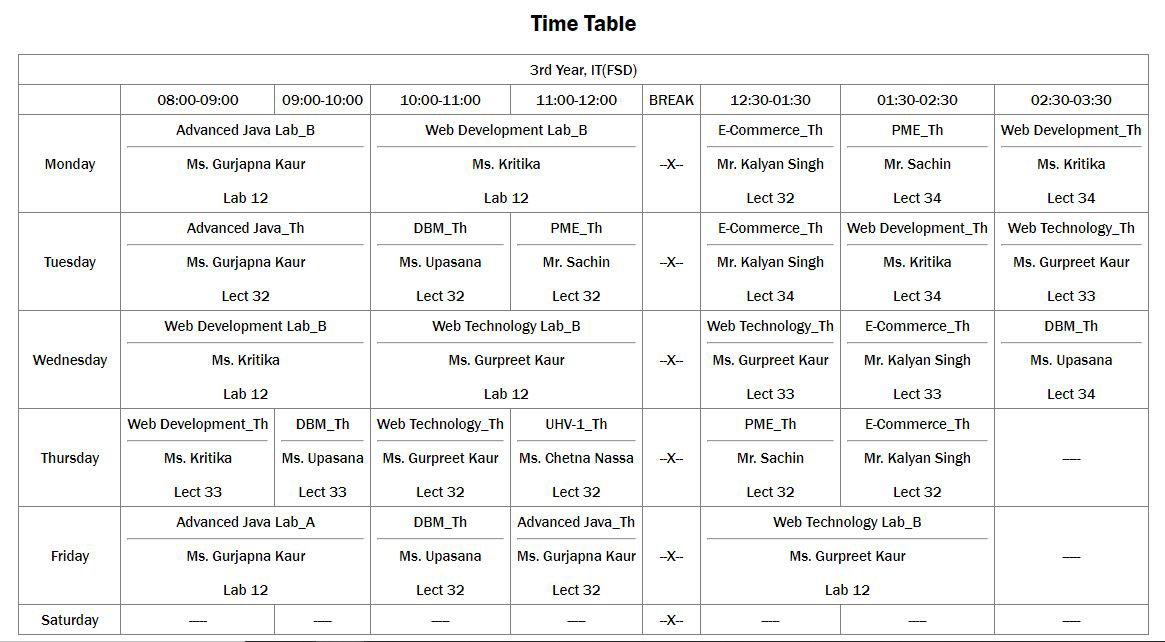
</tr>

</table>

</body>

</html>

**Output:**



**Practical – 3(b)**

**Aim:** Write a program to create Nested List (Ordered, Unordered & Definition List) using HTML.

**Code:**

<!doctype html>

<html lang="en">

<head>

<title>Ordered-Unordered List</title>

<meta charset="utf-8">

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

</head>

<style>

h2, dt{

font-family: courier;

}

li, dd{

font-family: verdana;

}

dt{

font-size: 120%;

font-weight: bold;

}

</style>

<body>

<h2><u>Example of Unordered List by an example of Names</u></h2>

<ul>

<li>Raghav</li>

<li>Mohit</li>

<li>Raman</li>

<li>Shikhar</li>

<li>Udhav</li>

</ul>

<h2><u>Example of ordered list start from 20</u></h2>

<ol start = "20">

<li>Pushpa</li>

<li>Purvi</li>

<li>Ravi</li>

<li>Sameer</li>

<li>Seema</li>

</ol>

<h2><u>Example of reverse ordered list</u></h2>

<ol reversed>

<li>Parth</li>

<li>Sujay</li>

<li>Hemant</li>

<li>Surya</li>

<li>Ammanat</li>

</ol>

<div>

<h2><u>Definition List</u></h2>

<dl>

<dt>Godfather</dt>

<dd>A man who serves as sponsor for a child at baptism or any male sponsor or guardian.</dd></br>

<dt>Defenestrate</dt>

<dd>To throw a thing out of a window.</dd></br>

<dt>Extension</dt>

<dd>The state of being extended, lengthened, or stretched out.</dd></br>

<dt>Trio</dt>

<dd>A company of three singers or players.</dd></br>

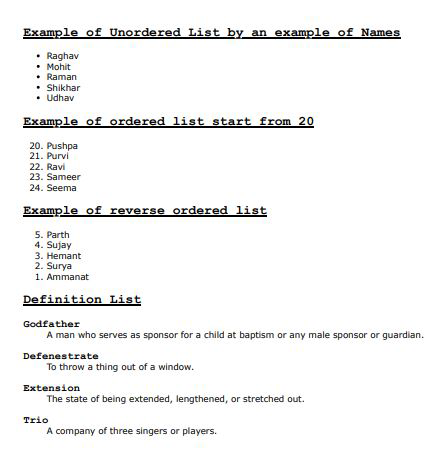
</dl>

</div>

</body>

</html>

**Output:**

****

**Practical – 4(a)**

**Aim:** Write a Program to Create Registration Form Using HTML.

**Code:**

<!doctype html>

<html>

<head>

<title> Registration form </title>

<meta charset="utf-8">

</head>

<body>

<style>

body {

font-family: verdana;

margin: 0;

padding: 0;

background-color: #f4f4f4;

}

#container {

width: 40%;

margin: 10vh auto;

padding: 20px;

background-color: #fff;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

form {

padding: 15px;

}

label {

display: block;

margin-bottom: 10px;

font-weight: bold;

}

input[type="text"], input[type="email"], input[type="password"] {

width: 100%;

padding: 10px;

border: 1px solid #ccc;

border-radius: 5px;

box-sizing: border-box;

}

input[type="submit"] {

background-color: #333;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

}

input[type="submit"]:hover {

background-color: #555;

}

</style>

<div id="container">

<form>

<label for="username">Username:</label><br>

<input type="text" id="username" name="username" placeholder="Username" required><br><br>

<label for="email">Email:</label><br>

<input type="email" id="email" name="email" placeholder="Email" required><br><br>

<label for="password">Password:</label><br>

<input type="password" id="password" name="password" placeholder="Password" required><br><br>

<label for="confirm\_password">Confirm Password:</label><br>

<input type="password" id="confirm\_password" name="confirm\_password" placeholder="Confirm Password" required><br><br>

<input type="submit" value="Register">

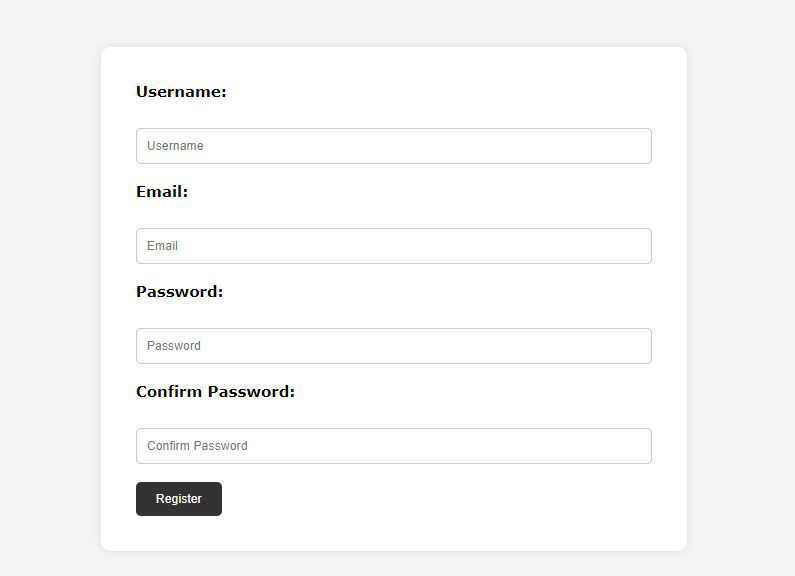
</form>

</div>

</body>

</html>

**Output:**



**Practical – 4(b)**

**Aim: write a program (WAP) to embed two websites using iframes and display hyperlinks using HTML.**

**Code:**

<!doctype html>

<html>

<head>

<title> Frame and Hyperlinks </title>

<meta charset="utf-8">

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

</head>

<body>

<style>

body {

font-family: Arial, sans-serif;

padding: 0;

background-color: #f4f4f4;

}

p {

display: flex;

justify-content: space-between;

align-items: center ;

font-size: 20px;

padding: 10px;

margin-top: 20px;

text-align: center;

background-color: black;

}

a {

text-decoration: none;

color: white;

}

</style>

<iframe src="https://webflow.com/made-in-webflow/iframe" width="49%" height="570px"></iframe>

<iframe src="https://www.pexels.com/" width="50%" height="570px"></iframe>

<p>

<u>Some more links - ></u>

<i><a href="https://www.accuweather.com/en/in/delhi/202396/weather-forecast/202396">Accu Weather</a></i><br>

<i><a href="https://www.google.com">Google</a></i><br>

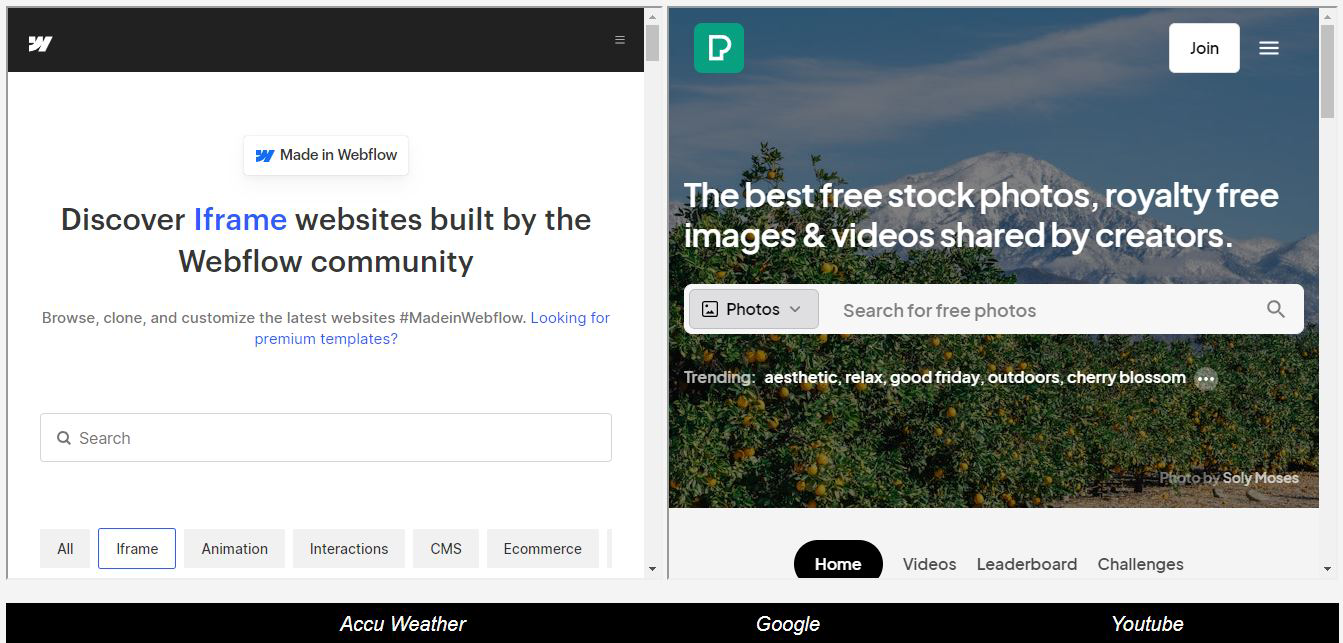
<i><a href="https://www.youtube.com">Youtube</a></i><br>

</p>

</body>

</html>

**Output:**



**Practical – 5(a)**

**Aim: write a program (WAP) to create a static webpage with a header, navigation menu, image, query form, and footer, using HTML and CSS, and to deploy the website on Netlify.**

**Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Static Webpage</title>

</head>

<body>

<style>

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

}

header {

background-color: #333;

color: #fff;

padding: 10px 0;

text-align: center;

}

nav {

background-color: black;

padding: 10px;

}

nav a {

text-decoration: none;

color: white;

padding: 5px 10px;

margin: 0 5px;

}

nav a:hover{

background-color: #323333;

}

img{

display: block;

margin-left: auto;

margin-right: auto;

width: 900px;

height: 190px;

}

#query{

text-align: center;

}

#query p{

font-family: verdana;

margin-left: auto;

margin-right: auto;

border-radius: 20px;

border: 1px solid #303333;

padding: 10px;

width: 300px;

height: 20px;

background-color: #b0b2b5;

}

#input1{

border-radius: 12px;

border: 1px solid black;

padding: 10px;

width: 200px;

height: 15px;

}

#button{

border-radius: 12px;

padding: 10px;

font-weight: bold;

}

#button:hover{

background-color: #b5b8b4;

cursor: pointer;

}

section {

padding: 20px;

}

footer {

background-color: #333;

color: #fff;

text-align: center;

padding: 19px 0;

position: flexible;

bottom: 0;

width: 100%;

}

</style>

<header>

<h1>Static Webpage</h1>

</header>

<div id="navigation">

<nav>

<a href="#">Home</a>

<a href="#">About</a>

<a href="#">Services</a>

<a href="#">Contact</a>

</nav>

</div>

<section>

<div>

<h2>Welcome to my Static Webpage</h2>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ullamcorper urna eget tortor accumsan, vel vulputate mauris placerat. Nam in ex id ipsum interdum aliquam in vel nulla. Suspendisse sit amet turpis vestibulum, sagittis purus sit amet, efficitur sem. Cras id dolor at nibh pellentesque volutpat non id ipsum. Fusce dui libero, tristique nec eleifend id, egestas non lacus. Suspendisse in sem sit amet velit bibendum venenatis. Donec dolor nulla, fermentum nec laoreet ac, pulvinar vitae odio. Sed sed neque at est suscipit posuere bibendum a tortor. Mauris faucibus purus molestie ligula consectetur faucibus. Sed dapibus viverra tortor a vestibulum. Ut eu ex a purus malesuada fringilla eget sed dolor. Pellentesque habitant morbi tristique senectus et netus et malesuada. </p>

</div>

<div id="image">

<img src="https://images.pexels.com/photos/51387/mount-everest-himalayas-nuptse-lhotse-51387.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1">

</div>

<div id="query">

<p><u><strong>Have queries!! Send us...</strong></u></p>

<form name="contact" method="post">

<input type="text" id="input1" name="query" />

<input type="submit" name="button" id="button" value="Submit" />

</form>

</div>

</section>

<footer>

<div id="currentTime"></div>

</footer>

<script>

function updateTime() {

var now = new Date();

var hours = now.getHours();

var minutes = now.getMinutes();

var seconds = now.getSeconds();

minutes = (minutes < 10 ? '0' : '') + minutes;

seconds = (seconds < 10 ? '0' : '') + seconds;

var timeString = hours + ':' + minutes + ':' + seconds;

document.getElementById('currentTime').textContent = '© 2024 My Static Website | Time: '+timeString;

}

setInterval(updateTime, 1000);

updateTime();

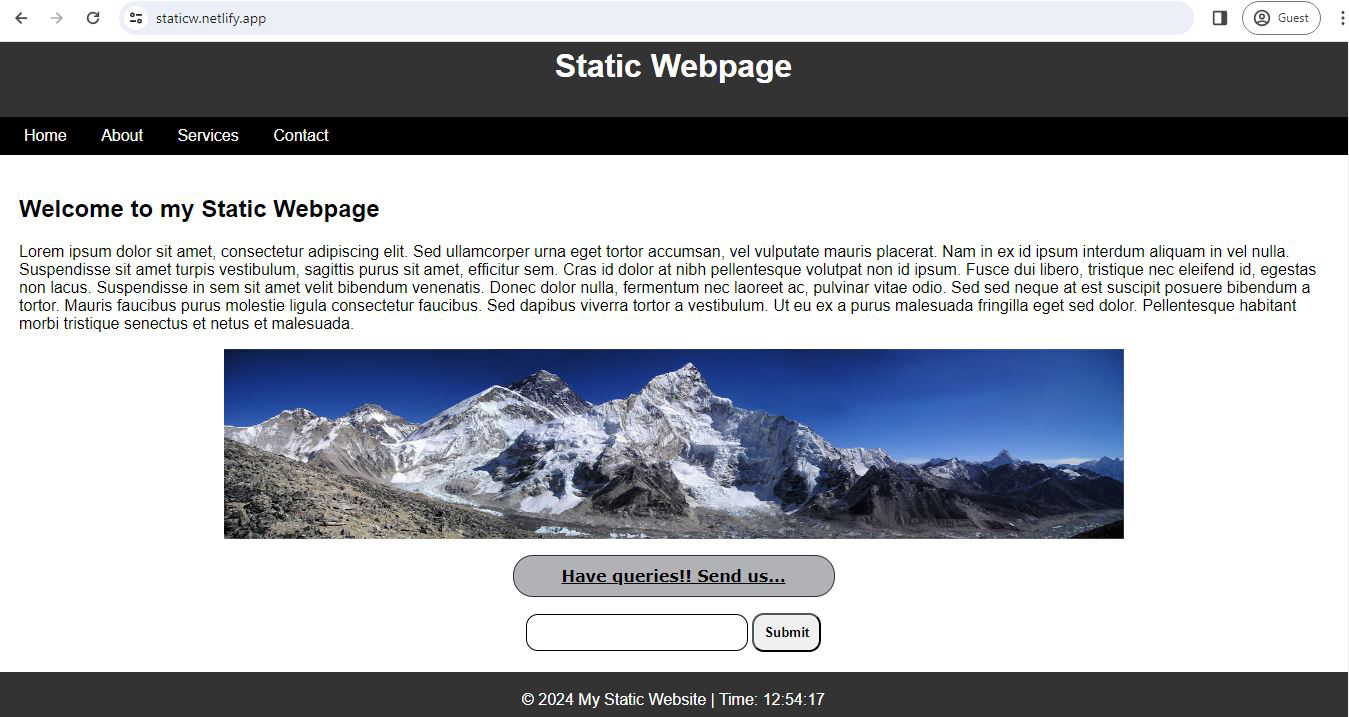
</script>

</body>

</html>

**Website Deployment:** I have deployed my website on Netlify because Netlify is fast, secure, cost-effective, and easily integrates with many tools. It is a great platform for hosting static websites. I have deployed my website on link <https://staticw.netlify.app/> .

**Output:**



**Practical – 5(b)**

**Aim:**

**Theory:**

Steps for installation:

1. **Download and Install Node.js:**

* Visit the official Node.js website at <https://nodejs.org/>.
* Download the appropriate installer for your operating system (Windows, macOS, or Linux).
* Run the installer and follow the on-screen instructions to complete the installation process.

1. **Verify Node.js Installation:**

* Open a command prompt or terminal window.
* Type the following command and press Enter:

node -v

* If Node.js is installed successfully, you should see the version number displayed in the terminal.



1. **Create a New Node.js Project:**

* Choose or create a directory where you want to set up your Node.js project.
* Open a command prompt or terminal window and navigate to the chosen directory.
* Run the following command to initialize a new Node.js project:

npm init -y

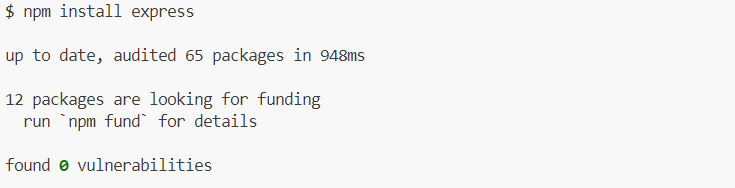
* This command creates a new **package.json** file in your project directory with default values.

1. **Install Express.js:**

* With Node.js initialized, you can now install Express.js as a dependency for your project.
* Run the following command in the terminal to install Express.js:

npm install express

* This command installs Express.js and adds it as a dependency in your **package.json** file.



1. **Create a Basic Express.js Application:**

* Create a new JavaScript file (e.g., **app.js**) in your project directory.
* Open **app.js** in a text editor and import Express.js:

const express = require('express'); const app = express();

* Define a route that responds with "Hello, World!" when accessed:

app.get('/', (req, res) => { res.send('Hello, World!'); });

* Start the Express.js server and listen for incoming requests on a specified port:

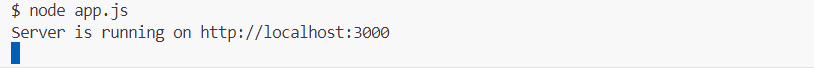
const port = 3000; app.listen(port, () => { console.log(`Server is running on http://localhost:${port}`); });

1. **Run the Express.js Application:**

* Save the changes to **app.js**.
* In the terminal, navigate to your project directory.
* Run the following command to start the Express.js server:

node app.js

* If everything is set up correctly, you should see the message "Server is running on http://localhost:3000" logged to the terminal.
* Open a web browser and navigate to **http://localhost:3000** to see the "Hello, World!" message displayed.





**Practical - 6**

1. Task management tool: Login/Register to the application, add daily tasks, Assign a due date of completion, Mark them as complete/incomplete and View weekly/monthly statistics of their to‐dos.

**Solution:**

1. **Setup MongoDB**: Install and set up MongoDB to store user data, tasks, and their statuses.
2. **Create Express.js Backend**: Set up your Express.js server to handle API requests. Create routes for user authentication, adding tasks, updating task status, and fetching task statistics.
3. **Implement User Authentication**: Create endpoints for user registration and login using JWT (JSON Web Tokens) for authentication.
4. **Build React Frontend**: Develop a user interface with React for users to login/register, add daily tasks, mark tasks as complete/incomplete, and view statistics.
5. **Connect Backend with Frontend**: Use Axios or Fetch API to send HTTP requests from the React frontend to the Express.js backend.
6. **Add Task Management Features**: Implement functionality to add daily tasks, assign due dates, mark tasks as complete/incomplete, and update task statuses.
7. **Calculate Weekly/Monthly Statistics**: Write logic on the backend to calculate and retrieve weekly/monthly statistics of user's to-dos based on their completion status and due dates.
8. **Display Statistics in Frontend**: Fetch and display the statistics in the React frontend using components and state management.
9. **Testing and Debugging**: Test the application thoroughly to ensure all features work as expected and debug any issues.
10. **Deployment**: Deploy your MERN stack application on a hosting platform like Heroku or AWS for users to access it online.
11. **Setup MongoDB**:

# Install MongoDB

# Start MongoDB service

sudo service mongod start

1. **Create Express.js Backend**:

// server.js

const express = require('express');

const app = express();

const PORT = process.env.PORT || 5000;

// Middleware

app.use(express.json());

// Routes

app.use('/api/auth', require('./routes/auth'));

app.use('/api/tasks', require('./routes/tasks'));

// Start server

app.listen(PORT, () => console.log(`Server is running on port ${PORT}`));

1. **Implement User Authentication**:

// routes/auth.js

const express = require('express');

const router = express.Router();

const { registerUser, loginUser } = require('../controllers/authController');

// Register user

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

// Login user

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

module.exports = router;

// controllers/authController.js

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

const jwt = require('jsonwebtoken');

// Register user

const registerUser = async (req, res) => {

// Implement user registration logic

};

// Login user

const loginUser = async (req, res) => {

// Implement user login logic

};

module.exports = { registerUser, loginUser };

1. **Build React Frontend**:

// App.js

import React from 'react';

import './App.css';

import TaskManager from './components/TaskManager';

function App() {

return (

<div className="App">

<TaskManager />

</div>

);

}

export default App;

// components/TaskManager.js

import React from 'react';

function TaskManager() {

// Implement task management logic

return (

<div>

{/\* Add task management components \*/}

</div>

);

}

export default TaskManager;

1. **Connect Backend with Frontend**:

// Example of using Axios to send HTTP requests from React frontend to Express.js backend

import axios from 'axios';

const BASE\_URL = 'http://localhost:5000/api';

export const loginUser = async (userData) => {

try {

const res = await axios.post(`${BASE\_URL}/auth/login`, userData);

return res.data;

} catch (error) {

console.error('Error logging in:', error.response.data);

throw error.response.data;

}

};

1. **Add Task Management Features**:

// routes/tasks.js

const express = require('express');

const router = express.Router();

const { addTask, updateTaskStatus } = require('../controllers/taskController');

// Add task

router.post('/', addTask);

// Update task status

router.put('/:id', updateTaskStatus);

module.exports = router;

// controllers/taskController.js

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

// Add task

const addTask = async (req, res) => {

// Implement logic to add a new task

};

// Update task status

const updateTaskStatus = async (req, res) => {

// Implement logic to update task status

};

module.exports = { addTask, updateTaskStatus };

1. **Calculate Weekly/Monthly Statistics**:

// Example of calculating weekly statistics for tasks completed

const calculateWeeklyStatistics = (tasks) => {

// Implement logic to calculate weekly statistics

};

1. **Display Statistics in Frontend**:

// components/Statistics.js

import React from 'react';

function Statistics({ statistics }) {

// Implement logic to display statistics

return (

<div>

{/\* Display statistics \*/}

</div>

);

}

export default Statistics;

1. **Testing and Debugging**: Test each feature thoroughly and debug any issues that arise.
2. **Deployment**: Deploy your MERN stack application on a hosting platform.

**Practical - 7**

2. Blogging platform:

**Solution:**

This is a basic setup for the frontend part of the MERN stack blogging platform. It includes setting up React components for different parts of the application, implementing routing with react-router-dom, integrating Axios for making HTTP requests to the backend API, and basic styling.

The frontend implementation using React would include components for displaying blog posts, creating/editing posts, user authentication forms, etc

Outline of the frontend development:

1. **Setup React Application**:

npx create-react-app blogging-platform

cd blogging-platform

1. **Install Required Packages**:

npm install axios react-router-dom

1. **Create Components**:
   * **Home Component (Displaying Blog Posts)**:

// components/Home.js

import React, { useEffect, useState } from 'react';

import axios from 'axios';

function Home() {

const [posts, setPosts] = useState([]);

useEffect(() => {

axios.get('http://localhost:5000/api/posts')

.then(res => setPosts(res.data))

.catch(err => console.error(err));

}, []);

return (

<div>

<h2>Blog Posts</h2>

{posts.map(post => (

<div key={post.\_id}>

<h3>{post.title}</h3>

<p>{post.content}</p>

</div>

))}

</div>

);

}

export default Home;

* + **Login/Register Component**:

// components/Auth.js

import React, { useState } from 'react';

import axios from 'axios';

function Auth() {

const [formData, setFormData] = useState({ username: '', password: '' });

const handleChange = e => {

setFormData({ ...formData, [e.target.name]: e.target.value });

};

const handleSubmit = e => {

e.preventDefault();

// Implement login/register logic with axios

};

return (

<div>

<h2>Login/Register</h2>

<form onSubmit={handleSubmit}>

<input type="text" name="username" placeholder="Username" value={formData.username} onChange={handleChange} />

<input type="password" name="password" placeholder="Password" value={formData.password} onChange={handleChange} />

<button type="submit">Login/Register</button>

</form>

</div>

);

}

export default Auth;

* + **Dashboard Component (Authenticated User)**:

// components/Dashboard.js

import React from 'react';

import { Link } from 'react-router-dom';

function Dashboard() {

return (

<div>

<h2>Dashboard</h2>

<Link to="/create">Create Post</Link>

<Link to="/posts">View Posts</Link>

</div>

);

}

export default Dashboard;

* + **Post Form Component (Create/Edit Post)**:

// components/PostForm.js

import React, { useState } from 'react';

import axios from 'axios';

function PostForm() {

const [formData, setFormData] = useState({ title: '', content: '' });

const handleChange = e => {

setFormData({ ...formData, [e.target.name]: e.target.value });

};

const handleSubmit = e => {

e.preventDefault();

// Implement create/edit post logic with axios

};

return (

<div>

<h2>Create Post</h2>

<form onSubmit={handleSubmit}>

<input type="text" name="title" placeholder="Title" value={formData.title} onChange={handleChange} />

<textarea name="content" placeholder="Content" value={formData.content} onChange={handleChange} />

<button type="submit">Submit</button>

</form>

</div>

);

}

export default PostForm;

1. **Implement Routing**:
   * **App Component**:

// App.js

import React from 'react';

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import Home from './components/Home';

import Auth from './components/Auth';

import Dashboard from './components/Dashboard';

import PostForm from './components/PostForm';

function App() {

return (

<Router>

<Switch>

<Route exact path="/" component={Home} />

<Route path="/auth" component={Auth} />

<Route path="/dashboard" component={Dashboard} />

<Route path="/create" component={PostForm} />

<Route path="/edit/:id" component={PostForm} />

</Switch>

</Router>

);

}

export default App;

1. **Integrate Axios for API Requests**:
   * Example usage of Axios for making HTTP requests to the backend API endpoints.
2. **Style Your Components**:
   * Use CSS or a UI library like Bootstrap to style your components and make them visually appealing.
3. **Testing**:
   * Test your components to ensure they work as expected across different scenarios.
4. **Deployment**:
   * Deploy your React application to a hosting platform like Netlify or Vercel.

**Backend Development (Node.js with Express)**

1. **Setup Express Server**:

// server.js

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const app = express();

// Middleware

app.use(express.json());

app.use(cors());

// Routes

app.use('/api/posts', require('./routes/posts'));

app.use('/api/auth', require('./routes/auth'));

// MongoDB Connection

mongoose.connect('mongodb://localhost:27017/blog', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

.catch(err => console.error(err));

// Start server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server is running on port ${PORT}`));

1. **Define MongoDB Schema**:

// models/Post.js

const mongoose = require('mongoose');

const postSchema = new mongoose.Schema({

title: {

type: String,

required: true

},

content: {

type: String,

required: true

},

author: {

type: mongoose.Schema.Types.ObjectId,

ref: 'User',

required: true

},

createdAt: {

type: Date,

default: Date.now

}

});

module.exports = mongoose.model('Post', postSchema);

1. **Create Routes for CRUD Operations**:

// routes/posts.js

const express = require('express');

const router = express.Router();

const { getAllPosts, createPost, getPostById, updatePost, deletePost } = require('../controllers/postController');

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

// Get all posts

router.get('/', getAllPosts);

// Create a new post

router.post('/', authMiddleware, createPost);

// Get a post by ID

router.get('/:id', getPostById);

// Update a post

router.put('/:id', authMiddleware, updatePost);

// Delete a post

router.delete('/:id', authMiddleware, deletePost);

module.exports = router;

1. **Implement Controller Functions**:

// controllers/postController.js

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

// Get all posts

const getAllPosts = async (req, res) => {

try {

const posts = await Post.find().populate('author', 'username');

res.json(posts);

} catch (error) {

console.error(error);

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

}

};

// Create a new post

const createPost = async (req, res) => {

// Extract data from request body

const { title, content } = req.body;

const author = req.user.id; // Assuming user ID is stored in req.user

try {

const newPost = new Post({ title, content, author });

await newPost.save();

res.json(newPost);

} catch (error) {

console.error(error);

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

}

};

// Other controller functions (getPostById, updatePost, deletePost) would follow a similar structure.

1. **Implement User Authentication (JWT)**:

// routes/auth.js

const express = require('express');

const router = express.Router();

const { registerUser, loginUser } = require('../controllers/authController');

// Register user

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

// Login user

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

module.exports = router;

// controllers/authController.js

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

const jwt = require('jsonwebtoken');

const bcrypt = require('bcryptjs');

// Register user

const registerUser = async (req, res) => {

// Implement user registration logic

};

// Login user

const loginUser = async (req, res) => {

// Implement user login logic

};

module.exports = { registerUser, loginUser };

Setup for the backend part of the MERN stack blogging platform.

It includes setting up an Express server, defining MongoDB schema, creating routes for CRUD operations, implementing controller functions for handling requests, and setting up user authentication with JWT.

**Practical - 8**

1. Social media platform

**Solution:**

**Backend Development**

1. **Setup Express Server**:

// server.js

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const app = express();

const PORT = process.env.PORT || 5000;

// Middleware

app.use(express.json());

app.use(cors());

// MongoDB Connection

mongoose.connect('mongodb://localhost:27017/social-media', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

.catch(err => console.error(err));

// Routes

app.use('/api/auth', require('./routes/auth'));

app.use('/api/posts', require('./routes/posts'));

// Start server

app.listen(PORT, () => console.log(`Server is running on port ${PORT}`));

1. **User Authentication (JWT)**:

// routes/auth.js

const express = require('express');

const router = express.Router();

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

const jwt = require('jsonwebtoken');

// Register user

router.post('/register', async (req, res) => {

// Implement user registration logic

});

// Login user

router.post('/login', async (req, res) => {

// Implement user login logic

});

module.exports = router;

1. **Define MongoDB Schema**:

// models/User.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

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

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

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

});

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

1. **Create Routes for CRUD Operations**:

// routes/posts.js

const express = require('express');

const router = express.Router();

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

// Get all posts

router.get('/', async (req, res) => {

try {

const posts = await Post.find();

res.json(posts);

} catch (error) {

console.error(error);

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

}

});

// Other CRUD operations (create, update, delete) would follow a similar structure.

module.exports = router;

**Frontend Development**

1. **Setup React Application**:

npx create-react-app social-media-app

cd social-media-app

1. **Create Components**:
   * **Authentication Component**:

// src/components/Auth.js

import React from 'react';

function Auth() {

return (

<div>

<h2>Login/Register</h2>

{/\* Add login/register form \*/}

</div>

);

}

export default Auth;

* + **Posts Component**:

// src/components/Posts.js

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function Posts() {

const [posts, setPosts] = useState([]);

useEffect(() => {

axios.get('http://localhost:5000/api/posts')

.then(res => setPosts(res.data))

.catch(err => console.error(err));

}, []);

return (

<div>

<h2>Posts</h2>

{posts.map(post => (

<div key={post.\_id}>

<h3>{post.title}</h3>

<p>{post.content}</p>

</div>

))}

</div>

);

}

export default Posts;

1. **Integrate Routing**:
   * **App Component**:

// src/App.js

import React from 'react';

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import Auth from './components/Auth';

import Posts from './components/Posts';

function App() {

return (

<Router>

<Switch>

<Route path="/auth" component={Auth} />

<Route path="/posts" component={Posts} />

</Switch>

</Router>

);

}

export default App;

1. **Style Components**:
   * Style your components using CSS or a CSS framework like Bootstrap to make the UI visually appealing.
2. **Testing and Deployment**:
   * Test your application thoroughly to ensure all features work as expected.
   * Deploy your MERN stack application to a hosting platform like Heroku, AWS, or Netlify for users to access it online.

This is rudimentary setup for both the frontend and the backend of a social media platform using the MERN stack. It can be expanded upon this by adding more features, such as user authentication, posting content, liking/commenting on posts, and more, based on your specific requirements.

**Practical - 9**

4. Weather Forecasting APP.

**Solution:**

Building a weather forecasting app involves integrating with a weather API to fetch weather data, designing a user interface to display the weather information, and possibly implementing additional features such as location detection and user settings.

Steps to create a basic weather forecasting app using the MERN stack:

**Backend Development**

1. **Setup Express Server**:

// server.js

const express = require('express');

const cors = require('cors');

const app = express();

const PORT = process.env.PORT || 5000;

// Middleware

app.use(express.json());

app.use(cors());

// Routes

app.use('/api/weather', require('./routes/weather'));

// Start server

app.listen(PORT, () => console.log(`Server is running on port ${PORT}`));

1. **Integrate with Weather API**:

You can use a weather API like OpenWeatherMap, Weatherstack, or AccuWeather to fetch weather data.

Here's an example using OpenWeatherMap:

// routes/weather.js

const express = require('express');

const router = express.Router();

const axios = require('axios');

router.get('/:city', async (req, res) => {

try {

const { city } = req.params;

const apiKey = 'YOUR\_API\_KEY';

const url = `http://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`;

const response = await axios.get(url);

res.json(response.data);

} catch (error) {

console.error(error);

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

}

});

module.exports = router;

**Frontend Development (React)**

1. **Setup React Application**:

npx create-react-app weather-forecast-app

cd weather-forecast-app

1. **Create Components**:
   * **Weather Component**:

// src/components/Weather.js

import React, { useState } from 'react';

import axios from 'axios';

function Weather() {

const [city, setCity] = useState('');

const [weatherData, setWeatherData] = useState(null);

const fetchWeather = async () => {

try {

const response = await axios.get(`http://localhost:5000/api/weather/${city}`);

setWeatherData(response.data);

} catch (error) {

console.error(error);

}

};

return (

<div>

<input type="text" value={city} onChange={e => setCity(e.target.value)} />

<button onClick={fetchWeather}>Get Weather</button>

{weatherData && (

<div>

<h2>{weatherData.name}</h2>

<p>Temperature: {weatherData.main.temp}°C</p>

<p>Weather: {weatherData.weather[0].main}</p>

</div>

)}

</div>

);

}

export default Weather;

1. **Style Components**:
   * Style your components using CSS or a CSS framework like Bootstrap to make the UI visually appealing.
2. **Integrate Routing**:
   * **App Component**:

// src/App.js

import React from 'react';

import Weather from './components/Weather';

function App() {

return (

<div>

<h1>Weather Forecasting App</h1>

<Weather />

</div>

);

}

export default App;

1. **Testing and Deployment**:
   * Test your application to ensure all features work as expected.
   * Deploy your MERN stack application to a hosting platform like Heroku, AWS, or Netlify for users to access it online.

This is a basic setup for both the frontend and the backend of a weather forecasting app. It can be expanded upon by adding more features, such as multiple city forecasts, hourly forecasts, or additional weather details.

Top of Form

**Practical - 10**

5. Bookstore Library and Stock keeping APP:

a) User Interface: Browse Books from library, filter them based on category, author, publications, pay & amp; rent them for a specific duration, like/review them

b) Admin interface: List/manage books, track rented books and their availability and send notifications via email to users once lease expires.

**Solution:**

The Backend:

// backend/server.js

const express = require('express');

const bodyParser = require('body-parser');

const mongoose = require('mongoose');

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

const app = express();

const PORT = process.env.PORT || 5000;

app.use(bodyParser.json());

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/bookstore', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

.catch(err => console.log(err));

// Routes

app.get('/api/books', async (req, res) => {

try {

const books = await Book.find();

res.json(books);

} catch (err) {

console.error(err);

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

}

});

app.post('/api/books', async (req, res) => {

try {

const newBook = new Book(req.body);

await newBook.save();

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

} catch (err) {

console.error(err);

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

}

});

// Define other routes (e.g., update, delete)

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

// backend/models/Book.js

const mongoose = require('mongoose');

const bookSchema = new mongoose.Schema({

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

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

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

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

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

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

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

});

module.exports = mongoose.model('Book', bookSchema);

The frontend:

// frontend/src/components/BookList.js

import React, { useEffect, useState } from 'react';

import axios from 'axios';

const BookList = () => {

const [books, setBooks] = useState([]);

useEffect(() => {

axios.get('/api/books')

.then(res => setBooks(res.data))

.catch(err => console.error(err));

}, []);

return (

<div>

<h2>Book List</h2>

<ul>

{books.map(book => (

<li key={book.\_id}>

<h3>{book.title}</h3>

<p>Author: {book.author}</p>

<p>Category: {book.category}</p>

<p>Publication: {book.publication}</p>

<p>Price: {book.price}</p>

<p>Rent Price: {book.rentPrice}</p>

<p>Available Copies: {book.availableCopies}</p>

</li>

))}

</ul>

</div>

);

};

export default BookList;

// frontend/src/App.js

import React from 'react';

import BookList from './components/BookList';

const App = () => {

return (

<div>

<h1>Bookstore Library</h1>

<BookList />

</div>

);

};

export default App;

In this code, we have a basic backend setup with Express.js and a MongoDB database. We define a Book model and implement GET and POST routes for books. On the frontend, we have a simple React component to display the list of books fetched from the backend using Axios.

Implementing the functionality to filter books based on category and to add a book through the admin interface.

**Backend:**

// backend/server.js

// Existing code...

app.get('/api/books/:category', async (req, res) => {

const category = req.params.category;

try {

const books = await Book.find({ category });

res.json(books);

} catch (err) {

console.error(err);

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

}

});

// Admin interface

app.post('/api/books/add', async (req, res) => {

try {

const newBook = new Book(req.body);

await newBook.save();

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

} catch (err) {

console.error(err);

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

}

});

// Existing code...

**Frontend:**

// frontend/src/components/BookFilter.js

import React, { useState } from 'react';

import axios from 'axios';

const BookFilter = () => {

const [category, setCategory] = useState('');

const [filteredBooks, setFilteredBooks] = useState([]);

const handleFilter = async () => {

try {

const response = await axios.get(`/api/books/${category}`);

setFilteredBooks(response.data);

} catch (error) {

console.error(error);

}

};

return (

<div>

<h2>Filter Books by Category</h2>

<input

type="text"

value={category}

onChange={(e) => setCategory(e.target.value)}

placeholder="Enter category"

/>

<button onClick={handleFilter}>Filter</button>

<ul>

{filteredBooks.map(book => (

<li key={book.\_id}>

<h3>{book.title}</h3>

<p>Author: {book.author}</p>

<p>Category: {book.category}</p>

<p>Publication: {book.publication}</p>

<p>Price: {book.price}</p>

<p>Rent Price: {book.rentPrice}</p>

<p>Available Copies: {book.availableCopies}</p>

</li>

))}

</ul>

</div>

);

};

export default BookFilter;

Update the frontend App component to include the BookFilter component:

// frontend/src/App.js

import React from 'react';

import BookList from './components/BookList';

import BookFilter from './components/BookFilter';

const App = () => {

return (

<div>

<h1>Bookstore Library</h1>

<BookList />

<BookFilter />

</div>

);

};

export default App;

Add functionality to handle renting books and implementing the admin interface to track rented books and send notifications once the lease expires.

**Backend:**

// backend/server.js

// Existing code...

// Rent a book

app.post('/api/books/rent/:id', async (req, res) => {

const bookId = req.params.id;

try {

const book = await Book.findById(bookId);

if (!book) {

return res.status(404).json({ message: 'Book not found' });

}

if (book.availableCopies === 0) {

return res.status(400).json({ message: 'No available copies' });

}

book.availableCopies -= 1;

await book.save();

res.status(200).json(book);

} catch (err) {

console.error(err);

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

}

});

// Admin interface to track rented books

app.get('/api/books/rented', async (req, res) => {

try {

const rentedBooks = await Book.find({ availableCopies: 0 });

res.json(rentedBooks);

} catch (err) {

console.error(err);

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

}

});

// Send notification via email once lease expires (not implemented here, just a placeholder)

app.post('/api/notify/:userId', async (req, res) => {

const userId = req.params.userId;

// Logic to send email notification goes here

res.status(200).json({ message: 'Notification sent' });

});

// Existing code...

**Frontend:**

// frontend/src/components/BookList.js

// Existing code...

const handleRent = async (id) => {

try {

await axios.post(`/api/books/rent/${id}`);

alert('Book rented successfully');

// Refresh book list after renting

fetchBooks();

} catch (error) {

console.error(error);

alert('Failed to rent the book');

}

};

// Existing code...

With these changes, you can now rent books from the frontend, track rented books from the admin interface, and send notifications when the lease expires. Note that the email notification logic is a placeholder and needs to be implemented separately using a service like nodemailer.

Add more frontend code to handle the rent operation and display rented books in the admin interface.

**Frontend:**

// frontend/src/components/AdminPanel.js

import React, { useState, useEffect } from 'react';

import axios from 'axios';

const AdminPanel = () => {

const [rentedBooks, setRentedBooks] = useState([]);

useEffect(() => {

fetchRentedBooks();

}, []);

const fetchRentedBooks = async () => {

try {

const response = await axios.get('/api/books/rented');

setRentedBooks(response.data);

} catch (error) {

console.error(error);

}

};

return (

<div>

<h2>Rented Books</h2>

<ul>

{rentedBooks.map(book => (

<li key={book.\_id}>

<h3>{book.title}</h3>

<p>Author: {book.author}</p>

<p>Category: {book.category}</p>

<p>Publication: {book.publication}</p>

<p>Price: {book.price}</p>

<p>Rent Price: {book.rentPrice}</p>

<p>Available Copies: {book.availableCopies}</p>

</li>

))}

</ul>

</div>

);

};

export default AdminPanel;

Update the App component to include the AdminPanel:

// frontend/src/App.js

import React from 'react';

import BookList from './components/BookList';

import BookFilter from './components/BookFilter';

import AdminPanel from './components/AdminPanel';

const App = () => {

return (

<div>

<h1>Bookstore Library</h1>

<BookList />

<BookFilter />

<AdminPanel />

</div>

);

};

export default App;

With these changes, you now have an admin panel component that displays the list of rented books. This component fetches data from the backend and renders the rented books accordingly.

Implement the functionality to handle book leases and send notifications when a lease expires.

**Backend:**

// backend/server.js

// Existing code...

// Rent a book

app.post('/api/books/rent/:id', async (req, res) => {

const bookId = req.params.id;

const { userId, duration } = req.body;

try {

const book = await Book.findById(bookId);

if (!book) {

return res.status(404).json({ message: 'Book not found' });

}

if (book.availableCopies === 0) {

return res.status(400).json({ message: 'No available copies' });

}

book.availableCopies -= 1;

await book.save();

// Calculate lease expiration date

const leaseExpirationDate = new Date();

leaseExpirationDate.setDate(leaseExpirationDate.getDate() + duration);

// Send notification about lease expiration

setTimeout(() => {

axios.post(`/api/notify/${userId}`);

}, duration \* 24 \* 60 \* 60 \* 1000);

res.status(200).json({ book, leaseExpirationDate });

} catch (err) {

console.error(err);

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

}

});

// Existing code...

**Frontend:**

// frontend/src/components/BookList.js

// Existing code...

const handleRent = async (id) => {

const userId = prompt('Enter your user ID:');

const duration = prompt('Enter rental duration (in days):');

try {

await axios.post(`/api/books/rent/${id}`, { userId, duration });

alert('Book rented successfully');

// Refresh book list after renting

fetchBooks();

} catch (error) {

console.error(error);

alert('Failed to rent the book');

}

};

// Existing code...

With these updates, when a user rents a book from the frontend, the backend will calculate the lease expiration date based on the provided duration and schedule a notification to be sent to the user when the lease expires.

**Practical - 11**

6. Build a simple CRUD application: create a web application that allows users to create, Read, Update and Delete data from a MongoDB database.

**Solution:**

build a simple CRUD (Create, Read, Update, Delete) application using the MERN stack (MongoDB, Express.js, React.js, Node.js).

First, make sure you have MongoDB installed and running on your system.

**Backend Setup:**

mkdir crud-app

cd crud-app

mkdir backend

cd backend

npm init -y

npm install express mongoose body-parser cors

Create a file named **server.js** in the **backend** directory:

// backend/server.js

const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

const mongoose = require('mongoose');

const app = express();

const PORT = process.env.PORT || 5000;

app.use(cors());

app.use(bodyParser.json());

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/crudDB', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

.catch(err => console.log(err));

// Define Schema

const itemSchema = new mongoose.Schema({

name: String,

description: String

});

const Item = mongoose.model('Item', itemSchema);

// CRUD Routes

app.get('/api/items', async (req, res) => {

try {

const items = await Item.find();

res.json(items);

} catch (err) {

console.error(err);

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

}

});

app.post('/api/items', async (req, res) => {

try {

const newItem = new Item(req.body);

await newItem.save();

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

} catch (err) {

console.error(err);

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

}

});

app.put('/api/items/:id', async (req, res) => {

const itemId = req.params.id;

try {

const updatedItem = await Item.findByIdAndUpdate(itemId, req.body, { new: true });

res.json(updatedItem);

} catch (err) {

console.error(err);

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

}

});

app.delete('/api/items/:id', async (req, res) => {

const itemId = req.params.id;

try {

await Item.findByIdAndDelete(itemId);

res.status(204).end();

} catch (err) {

console.error(err);

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

}

});

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

1. **Run the Application:**

Make sure your MongoDB server is running. Then, open two terminal windows, one for the backend and one for the frontend.

**Backend:**

cd backend

node server.js

**Frontend:**

cd frontend

npm start

**Practical - 12**

Design a web platform to help small businesses manage their inventory.

**Solution:**

Here's a high-level overview of the features and components that could be included in such a platform:

1. **User Authentication**: Allow users to sign up and log in securely to access the platform. Implement role-based access control to differentiate between administrators, managers, and regular employees.
2. **Dashboard**: Provide an intuitive dashboard that gives users an overview of their inventory status, including total inventory count, low-stock alerts, recent transactions, and any pending orders.
3. **Inventory Management**:
   * **Add/Edit/Delete Products**: Allow users to add new products to the inventory, edit existing product details (such as name, description, price, quantity, etc.), and delete products when necessary.
   * **Inventory Tracking**: Implement real-time tracking of inventory levels to ensure accurate stock counts. Provide alerts for low-stock items and automate reordering processes.
   * **Categorization and Tagging**: Allow users to categorize products into different categories and apply tags for easy organization and searching.
   * **Batch and Serial Number Tracking**: Provide the option to track products by batch or serial numbers, particularly for items with expiration dates or unique identifiers.
   * **Barcode/QR Code Integration**: Enable barcode or QR code scanning to streamline inventory management processes, such as receiving shipments, updating stock levels, and conducting audits.
4. **Order Management**:
   * **Purchase Orders**: Allow users to create and manage purchase orders for restocking inventory. Include features for tracking order status, supplier information, and expected delivery dates.
   * **Sales Orders**: Enable users to generate sales orders, track order fulfillment, and manage customer information for order processing and invoicing.
   * **Order Fulfillment**: Integrate with shipping carriers and logistics services to manage order fulfillment processes efficiently, including order picking, packing, and shipping.
5. **Reporting and Analytics**:
   * **Inventory Reports**: Generate detailed reports on inventory levels, turnover rates, stock valuation, and historical transactions. Provide insights into sales trends, product performance, and profitability.
   * **Forecasting**: Utilize historical data and predictive analytics to forecast demand, identify inventory trends, and optimize stocking levels to prevent stockouts and minimize excess inventory.
6. **Alerts and Notifications**:
   * **Low-Stock Alerts**: Automatically notify users when inventory levels fall below predefined thresholds to prevent stockouts and ensure timely reordering.
   * **Order Status Updates**: Send notifications to users regarding order status changes, such as order confirmations, shipment notifications, and delivery updates.
7. **Integration and Customization**:
   * **API Integration**: Allow integration with third-party applications, such as accounting software, e-commerce platforms, and point-of-sale (POS) systems, to streamline data exchange and workflow automation.
   * **Customization**: Provide customizable settings and configurations to adapt the platform to the unique needs and workflows of different businesses.
8. **Mobile Accessibility**: Develop a responsive web application that is accessible on mobile devices, enabling users to manage their inventory on-the-go.
9. **Security and Compliance**:
   * **Data Encryption**: Implement encryption protocols to secure sensitive data, such as user credentials, product information, and transaction records.
   * **Compliance**: Ensure compliance with data protection regulations, such as GDPR (General Data Protection Regulation) and CCPA (California Consumer Privacy Act), to protect user privacy and prevent data breaches.

Implementing the backend functionality for user authentication and product management using Node.js with Express.js and MongoDB.

1. **User Authentication:**

// backend/routes/auth.js

const express = require('express');

const router = express.Router();

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

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

// Register a new user

router.post('/register', async (req, res) => {

try {

// Check if the email is already registered

const existingUser = await User.findOne({ email: req.body.email });

if (existingUser) {

return res.status(400).json({ message: 'Email is already registered' });

}

// Hash the password

const hashedPassword = await bcrypt.hash(req.body.password, 10);

// Create a new user

const newUser = new User({

email: req.body.email,

password: hashedPassword,

role: 'regular' // Default role for new users

});

await newUser.save();

res.status(201).json({ message: 'User registered successfully' });

} catch (error) {

console.error(error);

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

}

});

// Login

router.post('/login', async (req, res) => {

try {

// Check if the user exists

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

if (!user) {

return res.status(401).json({ message: 'Invalid email or password' });

}

// Compare passwords

const isPasswordValid = await bcrypt.compare(req.body.password, user.password);

if (!isPasswordValid) {

return res.status(401).json({ message: 'Invalid email or password' });

}

// Generate JWT token

const token = jwt.sign({ userId: user.\_id, email: user.email, role: user.role }, 'secretKey', { expiresIn: '1h' });

res.status(200).json({ token });

} catch (error) {

console.error(error);

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

}

});

module.exports = router;

**Product Management:**

// backend/routes/products.js

const express = require('express');

const router = express.Router();

const authMiddleware = require('../middleware/auth');

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

// Get a

ll products

router.get('/', authMiddleware, async (req, res) => {

try {

const products = await Product.find();

res.json(products);

} catch (error) {

console.error(error);

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

}

});

// Add a new product

router.post('/', authMiddleware, async (req, res) => {

try {

const newProduct = new Product(req.body);

await newProduct.save();

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

} catch (error) {

console.error(error);

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

}

});

// Update a product

router.put('/:id', authMiddleware, async (req, res) => {

const productId = req.params.id;

try {

const updatedProduct = await Product.findByIdAndUpdate(productId, req.body, { new: true });

res.json(updatedProduct);

} catch (error) {

console.error(error);

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

}

});

// Delete a product

router.delete('/:id', authMiddleware, async (req, res) => {

const productId = req.params.id;

try {

await Product.findByIdAndDelete(productId);

res.status(204).end();

} catch (error) {

console.error(error);

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

}

});

module.exports = router;

**Middleware for Authentication:**

// backend/middleware/auth.js

const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {

try {

const token = req.headers.authorization.split(' ')[1];

const decodedToken = jwt.verify(token, 'secretKey');

req.userData = { userId: decodedToken.userId, email: decodedToken.email, role: decodedToken.role };

next();

} catch (error) {

console.error(error);

res.status(401).json({ message: 'Authentication failed' });

}

};

module.exports = authMiddleware;

**Product Model:**

// backend/models/Product.js

const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({

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

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

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

quantity: { type: Number, required: true }

});

module.exports = mongoose.model('Product', productSchema);

These are just the backend implementations for user authentication and product management. You would need to set up routes, controllers, models, and middleware accordingly in your Express.js application.

implementing some frontend functionality to interact with the backend APIs for user authentication and product management using React.

1. **User Authentication in React:**

// frontend/src/components/Login.js

import React, { useState } from 'react';

import axios from 'axios';

const Login = ({ onLogin }) => {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const handleLogin = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/auth/login', { email, password });

localStorage.setItem('token', response.data.token);

onLogin();

} catch (error) {

console.error(error);

alert('Invalid email or password');

}

};

return (

<div>

<h2>Login</h2>

<form onSubmit={handleLogin}>

<input type="email" value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" required />

<input type="password" value={password} onChange={(e) => setPassword(e.target.value)} placeholder="Password" required />

<button type="submit">Login</button>

</form>

</div>

);

};

export default Login;

**Product Management in React:**

// frontend/src/components/ProductList.js

import React, { useState, useEffect } from 'react';

import axios from 'axios';

const ProductList = () => {

const [products, setProducts] = useState([]);

useEffect(() => {

fetchProducts();

}, []);

const fetchProducts = async () => {

try {

const response = await axios.get('/api/products', {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

setProducts(response.data);

} catch (error) {

console.error(error);

}

};

return (

<div>

<h2>Product List</h2>

<ul>

{products.map(product => (

<li key={product.\_id}>

<div>{product.name}</div>

<div>{product.description}</div>

<div>{product.price}</div>

<div>{product.quantity}</div>

</li>

))}

</ul>

</div>

);

};

export default ProductList;

These are basic React components for user login and displaying a list of products. You would need to integrate them into your React application and add additional functionality for adding, updating, and deleting products as well as handling user authentication status.

Implementing functionality for adding, updating, and deleting products, as well as logout functionality in the React frontend.

**Adding Products:**

// frontend/src/components/AddProduct.js

import React, { useState } from 'react';

import axios from 'axios';

const AddProduct = ({ onAdd }) => {

const [name, setName] = useState('');

const [description, setDescription] = useState('');

const [price, setPrice] = useState('');

const [quantity, setQuantity] = useState('');

const handleAddProduct = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/products', { name, description, price, quantity }, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

onAdd(response.data);

setName('');

setDescription('');

setPrice('');

setQuantity('');

} catch (error) {

console.error(error);

alert('Failed to add product');

}

};

return (

<div>

<h2>Add Product</h2>

<form onSubmit={handleAddProduct}>

<input type="text" value={name} onChange={(e) => setName(e.target.value)} placeholder="Name" required />

<input type="text" value={description} onChange={(e) => setDescription(e.target.value)} placeholder="Description" required />

<input type="number" value={price} onChange={(e) => setPrice(e.target.value)} placeholder="Price" required />

<input type="number" value={quantity} onChange={(e) => setQuantity(e.target.value)} placeholder="Quantity" required />

<button type="submit">Add Product</button>

</form>

</div>

);

};

export default AddProduct;

**Updating Products:**

// frontend/src/components/UpdateProduct.js

import React, { useState } from 'react';

import axios from 'axios';

const UpdateProduct = ({ product, onUpdate }) => {

const [name, setName] = useState(product.name);

const [description, setDescription] = useState(product.description);

const [price, setPrice] = useState(product.price);

const [quantity, setQuantity] = useState(product.quantity);

const handleUpdateProduct = async (e) => {

e.preventDefault();

try {

const response = await axios.put(`/api/products/${product.\_id}`, { name, description, price, quantity }, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

onUpdate(response.data);

} catch (error) {

console.error(error);

alert('Failed to update product');

}

};

return (

<div>

<h2>Update Product</h2>

<form onSubmit={handleUpdateProduct}>

<input type="text" value={name} onChange={(e) => setName(e.target.value)} required />

<input type="text" value={description} onChange={(e) => setDescription(e.target.value)} required />

<input type="number" value={price} onChange={(e) => setPrice(e.target.value)} required />

<input type="number" value={quantity} onChange={(e) => setQuantity(e.target.value)} required />

<button type="submit">Update Product</button>

</form>

</div>

);

};

export default UpdateProduct;

**Deleting Products:**

// frontend/src/components/ProductItem.js

import React from 'react';

import axios from 'axios';

const ProductItem = ({ product, onDelete }) => {

const handleDeleteProduct = async () => {

try {

await axios.delete(`/api/products/${product.\_id}`, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

onDelete(product.\_id);

} catch (error) {

console.error(error);

alert('Failed to delete product');

}

};

return (

<li>

<div>{product.name}</div>

<div>{product.description}</div>

<div>{product.price}</div>

<div>{product.quantity}</div>

<button onClick={handleDeleteProduct}>Delete</button>

</li>

);

};

export default ProductItem;

**Logout Functionality:**

// frontend/src/components/Logout.js

import React from 'react';

const Logout = ({ onLogout }) => {

const handleLogout = () => {

localStorage.removeItem('token');

onLogout();

};

return (

<div>

<button onClick={handleLogout}>Logout</button>

</div>

);

};

export default Logout;

These are React components for adding, updating, and deleting products, as well as logout functionality.

**Practical- 13**

8. Create a web‐based system to streamline the process of booking appointments.

**Solution:**

Here's an overview of the functionality and components that could be included in such a system:

1. **User Authentication and Access Control**:
   * Allow users to sign up and log in securely to access the system.
   * Implement role-based access control to differentiate between administrators, staff members, and regular users.
2. **Appointment Booking Interface**:
   * Provide a user-friendly interface for users to browse available appointment slots and book appointments.
   * Allow users to select preferred dates, times, and service providers (if applicable).
   * Implement real-time availability checking to prevent double booking of appointment slots.
3. **Calendar Integration**:
   * Integrate with calendar systems (such as Google Calendar, Outlook Calendar) to sync appointment bookings and avoid conflicts with existing commitments.
   * Allow users to view their booked appointments within their preferred calendar application.
4. **Notification and Reminder System**:
   * Send email or SMS notifications to users upon successful booking of an appointment.
   * Send reminders to users before their scheduled appointment time to reduce no-shows.
5. **Appointment Management for Administrators**:
   * Provide an administrative dashboard for staff members to manage appointment schedules, view bookings, and make modifications as needed.
   * Allow administrators to add, edit, or cancel appointments on behalf of users.
6. **User Profile Management**:
   * Allow users to manage their profile information, including contact details, preferences, and appointment history.
   * Provide options for users to update their availability or cancel appointments if necessary.
7. **Integration with Payment Gateway**:
   * Enable online payment processing for appointment bookings, if applicable.
   * Support various payment methods and securely handle payment transactions.
8. **Reporting and Analytics**:
   * Generate reports on appointment bookings, cancellation rates, revenue generated, and other key metrics to track system performance and user engagement.
   * Provide insights into peak booking times, popular services, and user demographics to inform business decisions.

implementing some backend functionality for user authentication, appointment management, and notification handling using Node.js with Express.js and MongoDB.

// backend/routes/auth.js

const express = require('express');

const router = express.Router();

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

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

// Register a new user

router.post('/register', async (req, res) => {

try {

// Check if the email is already registered

const existingUser = await User.findOne({ email: req.body.email });

if (existingUser) {

return res.status(400).json({ message: 'Email is already registered' });

}

// Hash the password

const hashedPassword = await bcrypt.hash(req.body.password, 10);

// Create a new user

const newUser = new User({

email: req.body.email,

password: hashedPassword

});

await newUser.save();

res.status(201).json({ message: 'User registered successfully' });

} catch (error) {

console.error(error);

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

}

});

// Login

router.post('/login', async (req, res) => {

try {

// Check if the user exists

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

if (!user) {

return res.status(401).json({ message: 'Invalid email or password' });

}

// Compare passwords

const isPasswordValid = await bcrypt.compare(req.body.password, user.password);

if (!isPasswordValid) {

return res.status(401).json({ message: 'Invalid email or password' });

}

// Generate JWT token

const token = jwt.sign({ userId: user.\_id, email: user.email }, 'secretKey', { expiresIn: '1h' });

res.status(200).json({ token });

} catch (error) {

console.error(error);

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

}

});

module.exports = router;

**Appointment Management:**

// backend/routes/appointments.js

const express = require('express');

const router = express.Router();

const authMiddleware = require('../middleware/auth');

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

// Create a new appointment

router.post('/', authMiddleware, async (req, res) => {

try {

const newAppointment = new Appointment(req.body);

await newAppointment.save();

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

} catch (error) {

console.error(error);

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

}

});

// Get all appointments

router.get('/', authMiddleware, async (req, res) => {

try {

const appointments = await Appointment.find({ userId: req.userData.userId });

res.json(appointments);

} catch (error) {

console.error(error);

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

}

});

module.exports = router;

**Notification Handling:**

// backend/services/notificationService.js

const sendNotification = (email, message) => {

// Placeholder function for sending email notifications

console.log(`Sending notification to ${email}: ${message}`);

};

module.exports = { sendNotification };

Implementing some frontend functionality in React for user authentication, appointment booking, and notification handling.

1. **User Authentication in React:**

// frontend/src/components/Login.js

import React, { useState } from 'react';

import axios from 'axios';

const Login = ({ onLogin }) => {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const handleLogin = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/auth/login', { email, password });

localStorage.setItem('token', response.data.token);

onLogin();

} catch (error) {

console.error(error);

alert('Invalid email or password');

}

};

return (

<div>

<h2>Login</h2>

<form onSubmit={handleLogin}>

<input type="email" value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" required />

<input type="password" value={password} onChange={(e) => setPassword(e.target.value)} placeholder="Password" required />

<button type="submit">Login</button>

</form>

</div>

);

};

export default Login;

1. **Appointment Booking Form:**

// frontend/src/components/AppointmentForm.js

import React, { useState } from 'react';

import axios from 'axios';

const AppointmentForm = ({ onSuccess }) => {

const [date, setDate] = useState('');

const [time, setTime] = useState('');

const [service, setService] = useState('');

const handleBookAppointment = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/appointments', { date, time, service }, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

onSuccess(response.data);

setDate('');

setTime('');

setService('');

} catch (error) {

console.error(error);

alert('Failed to book appointment');

}

};

return (

<div>

<h2>Book Appointment</h2>

<form onSubmit={handleBookAppointment}>

<input type="date" value={date} onChange={(e) => setDate(e.target.value)} required />

<input type="time" value={time} onChange={(e) => setTime(e.target.value)} required />

<input type="text" value={service} onChange={(e) => setService(e.target.value)} placeholder="Service" required />

<button type="submit">Book Appointment</button>

</form>

</div>

);

};

export default AppointmentForm;

**Notification Component:**

// frontend/src/components/Notification.js

import React from 'react';

const Notification = ({ message }) => {

return (

<div>

<h3>Notification</h3>

<p>{message}</p>

</div>

);

};

export default Notification;

React components for user authentication, appointment booking, and a placeholder notification component.

Add more functionality to the appointment booking system by implementing features such as appointment listing, user profile management, and notifications.

**Appointment Listing Component:**

// frontend/src/components/AppointmentList.js

import React, { useState, useEffect } from 'react';

import axios from 'axios';

const AppointmentList = () => {

const [appointments, setAppointments] = useState([]);

useEffect(() => {

fetchAppointments();

}, []);

const fetchAppointments = async () => {

try {

const response = await axios.get('/api/appointments', {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

setAppointments(response.data);

} catch (error) {

console.error(error);

}

};

return (

<div>

<h2>My Appointments</h2>

<ul>

{appointments.map(appointment => (

<li key={appointment.\_id}>

<div>Date: {appointment.date}</div>

<div>Time: {appointment.time}</div>

<div>Service: {appointment.service}</div>

</li>

))}

</ul>

</div>

);

};

export default AppointmentList;

**User Profile Management Component:**

// frontend/src/components/Profile.js

import React, { useState, useEffect } from 'react';

import axios from 'axios';

const Profile = () => {

const [email, setEmail] = useState('');

useEffect(() => {

fetchUserProfile();

}, []);

const fetchUserProfile = async () => {

try {

const response = await axios.get('/api/user/profile', {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

setEmail(response.data.email);

} catch (error) {

console.error(error);

}

};

return (

<div>

<h2>My Profile</h2>

<p>Email: {email}</p>

</div>

);

};

export default Profile;

**Notification Service Integration:**

// frontend/src/services/notificationService.js

import axios from 'axios';

const sendNotification = async (email, message) => {

try {

await axios.post('/api/notifications', { email, message });

} catch (error) {

console.error(error);

}

};

export default sendNotification;

**Notification Component Update:**

// frontend/src/components/Notification.js

import React from 'react';

const Notification = ({ message }) => {

return (

<div>

<h3>Notification</h3>

<p>{message}</p>

</div>

);

};

export default Notification;

implementing backend functionality for appointment listing, user profile management, and notification handling.

**User Profile Management in Express.js:**

// backend/routes/user.js

const express = require('express');

const router = express.Router();

const authMiddleware = require('../middleware/auth');

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

// Get user profile

router.get('/profile', authMiddleware, async (req, res) => {

try {

const user = await User.findById(req.userData.userId);

res.json({ email: user.email });

} catch (error) {

console.error(error);

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

}

});

module.exports = router;

**Notification Handling in Express.js (Placeholder):**

// backend/routes/notifications.js

const express = require('express');

const router = express.Router();

const { sendNotification } = require('../services/notificationService');

// Send notification

router.post('/', async (req, res) => {

const { email, message } = req.body;

try {

await sendNotification(email, message);

res.status(200).json({ message: 'Notification sent successfully' });

} catch (error) {

console.error(error);

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

}

});

module.exports = router;

1. **Update Notification Service (Placeholder):**

// backend/services/notificationService.js

const sendNotification = async (email, message) => {

// Placeholder function for sending notifications

console.log(`Sending notification to ${email}: ${message}`);

};

module.exports = { sendNotification };

1. **Appointment Listing in Express.js:**

// backend/routes/appointments.js

const express = require('express');

const router = express.Router();

const authMiddleware = require('../middleware/auth');

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

// Get all appointments for a user

router.get('/', authMiddleware, async (req, res) => {

try {

const appointments = await Appointment.find({ userId: req.userData.userId });

res.json(appointments);

} catch (error) {

console.error(error);

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

}

});

module.exports = router;

**Practical-14**

9. Develop a web‐based tool for medical professionals to easily store and access patient records.

**Solution:**

Start by implementing the backend functionality for user authentication, patient record management, and appointment scheduling using Node.js with Express.js and MongoDB. Then we'll proceed to implement the frontend functionality using React.js.

**Backend (Node.js with Express.js and MongoDB)**

1. **Middleware for Authentication:**

// backend/middleware/auth.js

const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {

const token = req.headers.authorization;

if (!token) {

return res.status(401).json({ message: 'Access denied. No token provided.' });

}

try {

const decoded = jwt.verify(token.split(' ')[1], 'secretKey');

req.userData = decoded;

next();

} catch (error) {

console.error(error);

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

}

};

module.exports = authMiddleware;

1. **MongoDB Configuration:**

// backend/config/db.js

const mongoose = require('mongoose');

const connectDB = async () => {

try {

await mongoose.connect('mongodb://localhost:27017/medical\_records\_db', {

useNewUrlParser: true,

useUnifiedTopology: true,

useCreateIndex: true,

useFindAndModify: false

});

console.log('MongoDB Connected');

} catch (error) {

console.error(error.message);

process.exit(1);

}

};

module.exports = connectDB;

1. **User Model:**

// backend/models/User.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

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

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

role: { type: String, enum: ['admin', 'doctor', 'nurse'], default: 'doctor' }

});

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

1. **Patient Model:**

// backend/models/Patient.js

const mongoose = require('mongoose');

const patientSchema = new mongoose.Schema({

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

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

dateOfBirth: { type: Date },

gender: { type: String, enum: ['Male', 'Female', 'Other'] },

contact: {

phone: { type: String },

email: { type: String }

},

address: {

street: { type: String },

city: { type: String },

state: { type: String },

zipCode: { type: String }

},

medicalHistory: { type: String },

allergies: [{ type: String }],

medications: [{ type: String }],

createdAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('Patient', patientSchema);

1. **Appointment Model:**

// backend/models/Appointment.js

const mongoose = require('mongoose');

const appointmentSchema = new mongoose.Schema({

patientId: { type: mongoose.Schema.Types.ObjectId, ref: 'Patient', required: true },

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

date: { type: Date, required: true },

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

reason: { type: String },

notes: { type: String },

createdAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('Appointment', appointmentSchema);

1. **Express App Configuration:**

// backend/app.js

const express = require('express');

const connectDB = require('./config/db');

// Initialize Express app

const app = express();

// Connect to MongoDB

connectDB();

// Middleware

app.use(express.json());

// Routes

app.use('/api/auth', require('./routes/auth'));

app.use('/api/patients', require('./routes/patients'));

app.use('/api/appointments', require('./routes/appointments'));

// Start the server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**Frontend (React.js)**

1. **User Authentication Component:**

// frontend/src/components/Login.js

import React, { useState } from 'react';

import axios from 'axios';

const Login = () => {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const handleLogin = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/auth/login', { email, password });

localStorage.setItem('token', response.data.token);

// Redirect to dashboard or homepage after successful login

} catch (error) {

console.error(error);

alert('Invalid email or password');

}

};

return (

<div>

<h2>Login</h2>

<form onSubmit={handleLogin}>

<input type="email" value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" required />

<input type="password" value={password} onChange={(e) => setPassword(e.target.value)} placeholder="Password" required />

<button type="submit">Login</button>

</form>

</div>

);

};

export default Login;

1. **Patient Record Management Component:**

// frontend/src/components/PatientRecord.js

import React, { useState } from 'react';

import axios from 'axios';

const PatientRecord = () => {

const [firstName, setFirstName] = useState('');

const [lastName, setLastName] = useState('');

// Add more state variables for other fields

const handleCreateRecord = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/patients', {

firstName,

lastName,

// Include other fields here

}, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

// Optionally handle success (e.g., display success message, redirect)

} catch (error) {

console.error(error);

alert('Failed to create patient record');

}

};

return (

<div>

<h2>Create Patient Record</h2>

<form onSubmit={handleCreateRecord}>

<input type="text" value={firstName} onChange={(e) => setFirstName(e.target.value)} placeholder="First Name" required />

<input type="text" value={lastName} onChange={(e) => setLastName(e.target.value)} placeholder="Last Name" required />

{/\* Add input fields for other patient information \*/}

<button type="submit">Create Record</button>

</form>

</div>

);

};

export default PatientRecord;

1. **Appointment Scheduling Component:**

// frontend/src/components/Appointment.js

import React, { useState } from 'react';

import axios from 'axios';

const Appointment = () => {

const [patientId, setPatientId] = useState('');

const [date, setDate] = useState('');

const [time, setTime] = useState('');

const [reason, setReason] = useState('');

const handleScheduleAppointment = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/appointments', {

patientId,

date,

time,

reason

}, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

// Optionally handle success (e.g., display success message, redirect)

} catch (error) {

console.error(error);

alert('Failed to schedule appointment');

}

};

return (

<div>

<h2>Schedule Appointment</h2>

<form onSubmit={handleScheduleAppointment}>

<input type="text" value={patientId} onChange={(e) => setPatientId(e.target.value)} placeholder="Patient ID" required />

<input type="date" value={date} onChange={(e) => setDate(e.target.value)} required />

<input type="time" value={time} onChange={(e) => setTime(e.target.value)} required />

<input type="text" value={reason} onChange={(e) => setReason(e.target.value)} placeholder="Reason for Appointment" />

<button type="submit">Schedule Appointment</button>

</form>

</div>

);

};

export default Appointment;

This covers the implementation of user authentication, patient record management, and appointment scheduling functionalities in both the backend (Node.js with Express.js and MongoDB) and the frontend (React.js). Now we have to set up routes, controllers, models, and middleware accordingly and integrate the frontend components into the React application.

**Practical-15**

10. Develop a web application to facilitate collaboration between teachers and students.

**Solution:**

Start by implementing the backend functionality for user authentication, creating and managing classes, and handling assignments using Node.js with Express.js and MongoDB. Then proceed to implement the frontend functionality using React.js.

**Backend (Node.js with Express.js and MongoDB)**

1. **Middleware for Authentication:**

// backend/middleware/auth.js

const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {

const token = req.headers.authorization;

if (!token) {

return res.status(401).json({ message: 'Access denied. No token provided.' });

}

try {

const decoded = jwt.verify(token.split(' ')[1], 'secretKey');

req.userData = decoded;

next();

} catch (error) {

console.error(error);

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

}

};

module.exports = authMiddleware;

1. **MongoDB Configuration:**

// backend/config/db.js

const mongoose = require('mongoose');

const connectDB = async () => {

try {

await mongoose.connect('mongodb://localhost:27017/collaboration\_app\_db', {

useNewUrlParser: true,

useUnifiedTopology: true,

useCreateIndex: true,

useFindAndModify: false

});

console.log('MongoDB Connected');

} catch (error) {

console.error(error.message);

process.exit(1);

}

};

module.exports = connectDB;

1. **User Model:**

// backend/models/User.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

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

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

role: { type: String, enum: ['teacher', 'student'], required: true }

});

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

1. **Class Model:**

// backend/models/Class.js

const mongoose = require('mongoose');

const classSchema = new mongoose.Schema({

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

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

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

assignments: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Assignment' }]

});

module.exports = mongoose.model('Class', classSchema);

1. **Assignment Model:**

// backend/models/Assignment.js

const mongoose = require('mongoose');

const assignmentSchema = new mongoose.Schema({

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

description: { type: String },

dueDate: { type: Date },

class: { type: mongoose.Schema.Types.ObjectId, ref: 'Class', required: true }

});

module.exports = mongoose.model('Assignment', assignmentSchema);

1. **Express App Configuration:**

// backend/app.js

const express = require('express');

const connectDB = require('./config/db');

// Initialize Express app

const app = express();

// Connect to MongoDB

connectDB();

// Middleware

app.use(express.json());

// Routes

app.use('/api/auth', require('./routes/auth'));

app.use('/api/classes', require('./routes/classes'));

app.use('/api/assignments', require('./routes/assignments'));

// Start the server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**Frontend (React.js)**

1. **User Authentication Component:**

// frontend/src/components/Login.js

import React, { useState } from 'react';

import axios from 'axios';

const Login = () => {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const handleLogin = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/auth/login', { email, password });

localStorage.setItem('token', response.data.token);

// Redirect to dashboard or homepage after successful login

} catch (error) {

console.error(error);

alert('Invalid email or password');

}

};

return (

<div>

<h2>Login</h2>

<form onSubmit={handleLogin}>

<input type="email" value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" required />

<input type="password" value={password} onChange={(e) => setPassword(e.target.value)} placeholder="Password" required />

<button type="submit">Login</button>

</form>

</div>

);

};

export default Login;

1. **Class Creation Component:**

// frontend/src/components/CreateClass.js

import React, { useState } from 'react';

import axios from 'axios';

const CreateClass = () => {

const [name, setName] = useState('');

const handleCreateClass = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/classes', { name }, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

// Optionally handle success (e.g., display success message, redirect)

} catch (error) {

console.error(error);

alert('Failed to create class');

}

};

return (

<div>

<h2>Create Class</h2>

<form onSubmit={handleCreateClass}>

<input type="text" value={name} onChange={(e) => setName(e.target.value)} placeholder="Class Name" required />

<button type="submit">Create Class</button>

</form>

</div>

);

};

export default CreateClass;

1. **Assignment Creation Component:**

// frontend/src/components/CreateAssignment.js

import React, { useState } from 'react';

import axios from 'axios';

const CreateAssignment = () => {

const [title, setTitle] = useState('');

const [description, setDescription] = useState('');

const [dueDate, setDueDate] = useState('');

const handleCreateAssignment = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/api/assignments', {

title,

description,

dueDate

}, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

Top of Form

This covers the implementation of user authentication, user model, class model, assignment model in both the backend (Node.js with Express.js and MongoDB) and the frontend (React.js).

Now we have to set up routes, controllers, models, and middleware accordingly and integrate the frontend components into the React application.