Module 06

15/03/2025

Page 83

1. Explain what a Single Page Application (SPA) is and its advantage over traditional multi-page applications.

→

- A Single Page Applications (SPA) is a web applications that dynamically updates content on the same page without requiring full-page reloads. It relies on AJAX and JavaScript frameworks like React.js, Angular or Vue.js to create a seamless user experience.

→ Advantage over Traditional Multi-page applications :-

- Faster Performance :- Only required data is updated, reducing load times.
- Better User Experience :- No Page refresh, Smooth transitions and real-time Interaction.
- Reduced Sever Load :- Backend only provide data(API), reducing bandwidth usage.
- Easier Frontend Development :- SPAs work well with component-based Frameworks like React.js.

2. Describe the components of the MERN Stack and their roles in development SPAs.

→ The MERN stack consists of four technology that work together to build a full-stack web application, especially SPAs.

→ MERN Stack Components:-

Component	Role in SPAs.
MongoDB	NoSQL database that stores application data in JSON-like documents.
Express.js	Backend framework that handles HTTP requests and API routes.
React.js	Frontend framework for Building dynamic UI components in SPAs.
Node.js	JavaScript runtime that allows Express.js to run on the server.

→ MERN Stack Flow In SPAs:-

i) React.js (frontend)
ii) Express.js + Node.js (Backend)
iii) MongoDB (Database)
iv) Response sent back to React.js.

3. Discuss the challenges associated with designing and developing SPAs compared to traditional web Applications.

→

Challenge	Explanation
- SEO Optimization	SPAs rely on JavaScript, making it harder for search engines to index content. Solutions include SSR (Server-Side Rendering) with Next.js

- Initial Load Time	Large JavaScript bundles can slow down the first page load. Optimization techniques like code splitting and Lazy loading help.
- Security Risks	SPAs are vulnerable to XSS since they rely heavily on client-side JavaScript.
- Client Side state Management	Managing application state can be complex. Tools like Redux, context API or Recoil help manage state.
- Browser compatibility	Older browsers may not fully support modern JavaScript frameworks used in SPAs.

→ Solution to Overcome challenges:-

- Use SSR (Server-Side Rendering) with Next.js for SEO.
- Optimize JavaScript bundles to improve intial load time.
- Implement Proper authentication using JWT and OAuth.

4. Explain the importance of client-side routing in SPAs and how it differs from Server-Side routing.

→ Client-Side Routing in SPAs:-

- Uses React Router, Vue Router or Angular Route to handle routes.

- Only updates necessary parts of the page, improving Performance.

→ Differences :-

Feature	Client-Side ServerRouting	Server-Side Routing
- Page Reload	No reload, only content updates	Full-page reload on every request
- Performance	Faster transitions, smoother UI	Slower due to full Reload.
- SEO	Harder to optimize	Better for SEO as content is rendered on the server.
- Example	React Router <Route path="/home" />	Express.js router app.get('/home');

5. Discuss the advantages of using MongoDB as the database component of the MERN stack for storing and retrieving data.

→ Advantages of MongoDB in the MERN stack:

- Schema Flexibility
- JSON-Like Documents
- Scalability
- High Speed Performance
- Built-in sharing & Replication

→ Example - Storing user Data in MongoDB.

```
const mongoose = require("mongoose")
const userSchema = new mongoose.Schema({
    name: String,
    email: String,
    age: Number
});

const User = mongoose.model("User",
                            userSchema);
User.create({name: "Utsav", email: "abc@
    gmail.com", age: 20})
  .then(() => console.log("user added"));
  .catch(err => console.log(err));
```

6. Compare and contrast SPAs with multi-page applications (MPAs) in terms of performance, user experience, and development complexity.

→ SPAs vs MPAs :-

Feature	Single Page Application (SPA)	Multi page Application (MPA)
- Performance	Faster since only necessary content is updated dynamically.	slower due to full page reloads on navigations.
- User Experience	Smooth, app-like experience with no visible Page reload.	More traditional experience with noticeable Page transitions.
- Initial Load Time	Slower Initial load due to large Java-	Faster Initial load, but slower

	Script bundles	overall navigations
- SEO optimization.	Harder to optimize for SEO	Easier to optimize since content is rendered on the server
- Routing	Uses client-side Routing	Uses Server-side Routing
- Data Handling	Relies on APIs to fetch data dynamically	Data is usually rendered on the server and sent as HTML pages.
- Examples	- Gmail - Facebook - Twitter - Trello.	- Amazon - Wikipedia - Online Banking Sites..
- Development Complexity	Requires JavaScript frameworks like React, Vue or Angular	Simple structure using standard HTML, CSS and backend framework.

7. Build a task management application with a MERN stack, allowing user to create tasks, assign them to users, set due dates and mark them as complete, with features for filtering and storing tasks.

→

(1) Setup Project & Install Dependencies.

mkdir mern-task-manger && cd mern-task-manger

npx create-react-app client

mkdir server && cd server
npm init -y
npm install express mongoose cors dotenv body-parser

(2) Backend (Node.js & Express.js) - Define Task Model :-

→ server/models/Task.js

```
const mongoose = require("mongoose");
const taskSchema = new mongoose.Schema({
    title : String,
    assignedTo : String,
    dueDate : Date,
    completed : Boolean
});
module.export = mongoose.model("Task",
                                taskSchema);
```

(3) Backend - Execute REST API (Express.js + MongoDB) :-

→ Server/index.js

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const Task = require("./models/Task");

const app = express();
app.use(express.json());
app.use(cors());
mongoose.connect("mongodb://localhost:
    2115/tasksDB", {useNewURLParser: true,
```

```
    useUnifiedTopology: true });

app.post("/tasks", async (req, res) => {
    const newTask = new Task(req.body);
    await newTask.save();
    res.json(newTask);
});

app.get("/tasks", async (req, res) => {
    const tasks = await Task.find();
    res.json(tasks);
});

app.put("/tasks/:id", async (req, res) => {
    const task = await Task.findByIdAndUpdate(req.params.id,
        { completed: req.body.completed },
        { new: true }
    );
    res.json(task);
});

app.delete("/tasks/:id", async (req, res) => {
    await Task.findByIdAndDelete(req.params.id);
    res.json({ message: "Task deleted" });
});

app.listen(2005, () => console.log("Server running on port 2005"));
```

(4) Frontend (React.js) - Fetch & Display Tasks.

→ client/src/App.js :-

```
import React, { useEffect, useState } from
                                    "react";
const App = () => {
  const [tasks, setTasks] = useState([]);
  useEffect(() => {
      fetch("http://localhost:2005/tasks")
        .then(response => response.json())
        .then(data => setTasks(data));
  }, []);

  return (
    <div>
      <h1> Task Manager </h1>
      <ul>
        {tasks.map(task => (
          <li key={task.id}>
            {task.title} - {task.
              completed ? "completed"
                : "Pending"}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default App;
```