2021

Q1…To style the HTML document as described with the given DOM tree, we can create CSS rules for each styled element using appropriate selectors. Although the exact values in pixels or colors aren't important, I'll provide a general example of the CSS rules and explain their role in the page appearance:

1. `h1` Selector:
   - Role: Styles the heading "Welcome!" at the top of the page.
   - CSS Rule: `h1 {
     font-size: 36px; /* Reasonable font size */
     color: #333; /* Reasonable text color */
     text-align: center; /* Center-align the text */
   }`

2. `section` Selector:
   - Role: Styles the section containing the navigation links.
   - CSS Rule: `section {
     background-color: #f0f0f0; /* A light background color */
     padding: 20px; /* Add some padding for spacing */
   }`

3. `ul` Selector:
   - Role: Styles the unordered list inside the section.
   - CSS Rule: `ul {
     list-style: none; /* Remove bullet points */
     padding: 0; /* Remove default padding */
   }`

4. `li` Selector:
   - Role: Styles the list items within the unordered list.
   - CSS Rule: `li {

margin-bottom: 10px; /* Add spacing between list items */

}`

5. `a` Selector:

- Role: Styles the hyperlinks within list items.

- CSS Rule: `a {

color: #007acc; /* A reasonable link color */

text-decoration: none; /* Remove underline */

}`

These rules provide a basic styling for the elements in the document, including font size, colors, alignment, and spacing. Keep in mind that these are general examples, and you can adjust the values as needed for the desired appearance.

QUESTION 2

a. The HTTP request that the browser will send to the server if a user enters 'Bliss' in the title field and 'Peter Carey' in the author field using the GET method will look like this:

```
GET /favourite?title=Bliss&author=Peter+Carey HTTP/1.1

Host: www.example.com
```

In this request, the form data is appended to the URL as query parameters. The 'title' parameter is set to 'Bliss,' and the 'author' parameter is set to 'Peter Carey.' This is a typical way of sending form data via a GET request.

b. Using GET for a form submission to register favorite books in a 'top 100 books of all time' poll is not appropriate. The reason is that GET requests should be used for safe and idempotent operations, meaning they should not have any side effects on the server or data. In this case, registering a favorite book is not a safe operation as it modifies server-side data. Additionally, GET requests have limitations on the length of the URL, which may restrict the amount of data that can be sent. Using

POST would be more appropriate for this form submission because it can handle larger data payloads and is intended for operations that modify data on the server.

c. An SQL injection attack occurs when an attacker manipulates an application's input to execute unintended SQL commands on the application's database. Here's an example and what would be needed to carry out the attack:

Example:

Suppose there's a login page with a username and password field, and the application uses SQL queries to authenticate users:

1. Input:

   - Username: `john_doe`

   - Password: `' OR 1=1 --`

2. SQL Query (simplified):

   ```sql

   SELECT * FROM users WHERE username = 'john_doe' AND password = '' OR 1=1 --';

   ```

The attacker uses the `' OR 1=1 --` input to manipulate the query. Here's what's happening:

- `'` closes the initial single-quote for the password field.

- `OR 1=1` always evaluates to true, bypassing the password check.

- `--` comments out the rest of the query, preventing any errors.

To carry out this attack, the attacker would need to know the structure of the database, the table names, and the column names. They would also need to know that the application is vulnerable to SQL injection.

QUESTION 3

a. Three-Tier Architecture in Web Applications:

1. Presentation Tier (Frontend):

- Description: The presentation tier is the topmost layer that interacts directly with the user. It consists of the user interface components, such as web pages, forms, and graphical elements. This tier is responsible for presenting information to users and collecting user input.

- Role: It handles the user interface and user experience, often using technologies like HTML, CSS, JavaScript, and client-side frameworks.

2. Application Tier (Middleware):

- Description: The application tier, also known as the middle tier, acts as an intermediary between the frontend and the backend. It contains the application logic, business rules, and processes that govern the behavior of the web application. It processes user requests, performs computations, and communicates with the backend.

- Role: It manages the application's functionality, processes data, and often uses server-side scripting languages like Java, Python, Ruby, or PHP to implement business logic.

3. Data Tier (Backend):

- Description: The data tier, sometimes referred to as the backend or database tier, is responsible for storing and managing data used by the application. It includes databases and data storage systems that store, retrieve, and manipulate data. The data tier ensures data integrity and persistence.

- Role: It handles data storage, retrieval, and manipulation, typically using database management systems (DBMS) like MySQL, PostgreSQL, MongoDB, or Oracle.

b. Service-Oriented Architecture (SOA):

SOA is a software architectural approach that emphasizes the creation and use of services to build scalable and flexible applications. Here's an explanation of key concepts in SOA:

- **Services**: Services are self-contained, reusable software components that provide specific functionality. They can be accessed and utilized by other parts of the application or even by external applications through standardized interfaces.

- **Loose Coupling**: SOA promotes loose coupling between services, meaning that services are independent and can change without affecting other services. Loose coupling enhances flexibility and maintainability.

- **Interoperability**: SOA emphasizes interoperability between different systems and technologies. Services are designed to work across heterogeneous environments and can communicate using standardized protocols like SOAP or REST.

- **Service Registry**: In SOA, there is often a service registry or directory that helps locate and manage available services. This registry provides a central point for service discovery and access.

- **Reusability**: Services are designed to be reusable, reducing redundancy in code and promoting a modular approach to development.

- **Scalability**: SOA allows for easy scalability as new services can be added or removed to meet changing business requirements.

- **Example**: An e-commerce application may have services for product catalog, order processing, and user authentication. These services can be independently developed, maintained, and scaled, making the application more agile and adaptable to changing demands.

SOA promotes a modular, flexible, and interoperable approach to software development, making it suitable for complex and distributed systems.

QUESTION 4

Creating a full JSP and Servlet code for managing student interview results, including listing results, deleting students, and updating student marks, is a comprehensive task that requires multiple files and logic. Below, I'll provide you with an outline of the steps and code structure to achieve this. You can then implement the details based on your specific requirements.

Create a JSP Page for Listing Results (list.jsp):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Student Interview Results</title>
</head>
<body>
    <h1>Interview Results</h1>
    <table border="1">
        <!-- Display a table of student results here -->
        <!-- Include columns for name, marks in each round, total marks, and actions (update/delete) -->
    </table>
    <br>
    <a href="add.jsp">Add New Student</a>
</body>
</html>
```

Create a JSP Page for Adding/Updating Student Information (add.jsp):

```
<!-- Create a form to add/update student information -->
<!-- Include fields for student name, marks in each round, and a submit button -->
```

Create a Servlet for Handling Student Actions (StudentServlet.java):

```
@WebServlet("/StudentServlet")
public class StudentServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String action = request.getParameter("action");

        if (action != null) {
            if (action.equals("list")) {
                // Handle listing of student results
                // Retrieve data from a database or data source and forward to list.jsp
            } else if (action.equals("delete")) {
                // Handle student deletion
                // Implement logic to delete a student based on their ID
            } else if (action.equals("update")) {
                // Handle updating student marks
                // Implement logic to update a student's marks based on their ID
            }
        }
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Handle adding a new student
        // Implement logic to add a new student with their name and marks
    }
}
```

4. **Database Integration:**
   - You should integrate a database (e.g., MySQL, PostgreSQL) to store student information and results.
   - Use JDBC or an ORM (e.g., Hibernate) to interact with the database.
5. **Implement Ranking System:**

- After each round, calculate the total marks for each student and assign ranks accordingly.

6. **Security:**
   - Implement proper security measures to prevent unauthorized access and ensure data privacy.

This outline provides a high-level structure for your JSP and Servlet code. You'll need to implement the specific details, database integration, and error handling based on your requirements and environment.

QUESTION 5

To create a Spring Boot web service that retrieves course data from a MySQL database based on the provided start and end times and outputs it as JSON, you can follow these steps:

1. Set up a Spring Boot project with Spring Data JPA and a MySQL database.
2. Create an entity class `Course` that represents the course data from the database. Ensure it has fields for code, startTime, endTime, seatsAvailable, seatsTotal, and name.

java Copy code

```java
@Entity
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String code;
    private int startTime;
    private int endTime;
    private int seatsAvailable;
    private int seatsTotal;
    private String name;

    // getters and setters
}
```

Summar

Create a Spring Data JPA repository interface for the `Course` entity.

```
@Repository
public interface CourseRepository extends JpaRepository<Course, Long> {
   List<Course> findByStartTimeAndEndTimeAndSeatsAvailableGreaterThan(int startTime, int endTime, int seatsAvailable);
}
```

Create a RESTful controller that handles the GET request with start and end parameters and returns JSON data.

```
@RestController
@RequestMapping("/courses")
public class CourseController {
   @Autowired
   private CourseRepository courseRepository;

   @GetMapping
   public ResponseEntity<Map<String, Object>> getCourses(
         @RequestParam("start") int start,
         @RequestParam("end") int end) {

      List<Course> matchingCourses = courseRepository.findByStartTimeAndEndTimeAndSeatsAvailableGreaterThan(start, end, 0);

      Map<String, Object> response = new HashMap<>();
      response.put("count", matchingCourses.size());
      response.put("courses", matchingCourses.stream()
            .map(course -> {
               Map<String, Object> courseData = new HashMap<>();
               courseData.put("code", course.getCode());
               courseData.put("seats", course.getSeatsTotal() - course.getSeatsAvailable());
```

```
            courseData.put("name", course.getName());
            return courseData;
        })
        .collect(Collectors.toList()));

    return ResponseEntity.ok(response);
  }
}
```

5. Configure your application.properties or application.yml file with the database connection details.
6. Populate your MySQL database with course data.
7. Run your Spring Boot application.
8. Access the endpoint with a URL like:
   `http://localhost:8080/courses?start=130&end=230`

This will return JSON data with the count of matching courses and the details of each course that matches the start and end times and has open seats.