LS/EECS

Building E-commerce Applications

Views with Ajax for REST APIs

Objectives

By the end of this lab, students should be able to:

- Understand the basics of Ajax and how it enables asynchronous communication between client and server using jQuery library.
- Create HTML views that interact with REST APIs using Ajax.
- Handle API responses and dynamically update the user interface without reloading the page.
- Apply CSS styling to enhance the appearance of HTML views.

Before you start

- Make sure you have Lab 4 finished and working. Try your curl/postman commands from the CLI
- Review the MVC lecture and Lab 2
- Review the Lecture on AJAX

Introduction

In this lab, we will focus on creating views for the Student Information System that you developed in the previous lab. Each view will interact with a specific REST API method using Ajax, allowing for asynchronous data exchange between the client and server. This approach provides a seamless user experience by updating the user interface dynamically without requiring a page reload.

Furthermore, you will enhance the user interface of your views by applying CSS styling. This will not only make your application more visually appealing but also improve the user experience by providing a more intuitive and engaging interface.

Understanding Ajax

Ajax (Asynchronous JavaScript and XML) is a set of web development techniques that allows web pages to update asynchronously by exchanging small amounts of data with the server behind the scenes. This results in a dynamic and fast user experience as only parts of a web page are updated without reloading the whole page.

Basic Ajax Request

Below is a basic example of an Ajax request using jQuery:

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $.ajax({
            url: "your-api-endpoint",
            type: 'GET',
            success: function(result){
                console.log(result);
            },
            error: function(error){
                console.log(error);
            }
        });
    });
});
</script>
```

Components of an Ajax Request

- **URL:** The URL to which the request is sent.
- **Type:** The type of request: GET, POST, PUT, DELETE, etc.
- **Data:** The data to be sent to the server. It is used when creating or updating data on the server.
- **Success:** A callback function to be executed if the request succeeds.
- **Error:** A callback function to be executed if the request fails.

Asynchronous Processing

Ajax allows for asynchronous processing, meaning that the user can continue to interact with the page while the request is being processed. Once the request is complete, the page can be updated with the new data without requiring a full page reload.

Handling Responses

Ajax can handle different types of data as responses, including XML, HTML, JSON, and text. The `dataType` property of the Ajax request can be used to specify the type of data expected from the server.

Example: Handling JSON Response

```
<script>
$.ajax({
    url: "your-api-endpoint",
    type: 'GET',
    dataType: 'json',
    success: function(result){
        console.log(result);
    },
    error: function(error){
        console.log(error);
    }
});
</script>
```

Sending Data to the Server

When sending data to the server, you need to serialize it into a format that can be transmitted over the network, typically as a string. JSON is a common format for this purpose.

Example: Sending JSON Data

```
<script>
var studentData = {
    name: "John Doe",
    age: 22,
    major: "Computer Science"
};
$.ajax({
    url: "your-api-endpoint",
    type: 'POST',
    contentType: 'application/json',
    data: JSON.stringify(studentData),
    success: function(result){
        console.log(result);
    },
    error: function(error){
        console.log(error);
    }
});
</script>
```

Task 1: Set Up HTML Files

You start with Lab 4. Have it implemented and working properly.

Create separate HTML files for each view. Each file should have the basic HTML structure, including linking to jQuery and any other necessary scripts and stylesheets. Call the pages: Get.html, Post.html, Put.html and Delete.html , Use the <servlet> and <servlet-mapping> elements in web.xml to map your html files with a URL.

Example for a basic HTML file structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Student View</title>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
<body>
    <!-- Content will go here, see below the content -->
</body>
</html>
```

Task 2: Create View for Listing All Students (Get.html)

1. **HTML Structure:** Create an empty table where the student data will be displayed.
2. **Ajax Request:** When the page loads, automatically make an Ajax GET request to list all students.
3. **Display Data:** Populate the table with the data received from the server.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>List Students</title>
    <link rel="stylesheet" href="styles.css">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
<body>

<header>
    <h1>Student Information System</h1>
</header>

<div class="container">
    <table id="studentsTable" border="1">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Age</th>
            <th>Major</th>
        </tr>
        <!-- Rows will be dynamically added here -->
    </table>
</div>

<footer>
    <p>© 2023 Student Information System</p>
</footer>

<script>
$(document).ready(function(){
    $.ajax({
        url: "http://localhost:8080/studentapp/api/students",
        type: 'GET',
        success: function(result){
            var table = $("#studentsTable");
            result.forEach(function(student){
                var row = $("<tr></tr>");
                row.append($("<td></td>").text(student.id));
                row.append($("<td></td>").text(student.name));
                row.append($("<td></td>").text(student.age));
                row.append($("<td></td>").text(student.major));
                table.append(row);
            });
        },
        error: function(error){
            console.log(error);
        }
    });
});
</script>

</body>
</html>
```

Task 3: Create View for Adding a New Student(Post.html)

1. **HTML Structure:** Create a form with input fields for student details.
2. **Ajax Request:** On form submission, make an Ajax POST request to add a new student.
3. **Handle Response:** Display a confirmation message upon successful addition.

Example:

```
<!-- Inside body tag -->
<form id="addStudentForm">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br>
    <label for="age">Age:</label>
    <input type="number" id="age" name="age" required><br>
    <label for="major">Major:</label>
    <input type="text" id="major" name="major" required><br>
    <input type="submit" value="Add Student">
</form>
<p id="confirmationMessage"></p>

<script>
$("#addStudentForm").submit(function(e){
    e.preventDefault();
    var studentData = {
        name: $("#name").val(),
        age: $("#age").val(),
        major: $("#major").val()
    };
    $.ajax({
        url: "http://localhost:8080/studentapp/api/students",
        type: 'POST',
        contentType: 'application/json',
        data: JSON.stringify(studentData),
        success: function(result){
            $("#confirmationMessage").text("Student added successfully!");
        },
        error: function(error){
            console.log(error);
        }
    });
});
```

```
</script>
```

Task 4: Create View for Updating a Student(Put.html)

1. **HTML Structure:** Create a form with input fields for student ID and details to be updated.
2. **Ajax Request:** On form submission, make an Ajax PUT request to update a student.
3. **Handle Response:** Display a confirmation message upon successful update.

Example:

```
<!-- Inside body tag -->
<form id="updateStudentForm">
    <label for="id">Student ID:</label>
    <input type="number" id="id" name="id" required><br>
    <label for="name">New Name:</label>
    <input type="text" id="name" name="name"><br>
    <label for="age">New Age:</label>
    <input type="number" id="age" name="age"><br>
    <label for="major">New Major:</label>
    <input type="text" id="major" name="major"><br>
    <input type="submit" value="Update Student">
</form>
<p id="updateConfirmationMessage"></p>

<script>
$("#updateStudentForm").submit(function(e){
    e.preventDefault();
    var studentData = {
        name: $("#name").val(),
        age: $("#age").val(),
        major: $("#major").val()
    };
    var studentId = $("#id").val();
    $.ajax({
        url: "http://localhost:8080/studentapp/api/students/" + studentId,
        type: 'PUT',
        contentType: 'application/json',
        data: JSON.stringify(studentData),
        success: function(result){
            $("#updateConfirmationMessage").text("Student updated successfully!");
        },
        error: function(error){
            console.log(error);
        }
    });
});
</script>
```

Task 5: Create View for Deleting a Student(Delete.html)

1. **HTML Structure:** Create a form with an input field for student ID.
2. **Ajax Request:** On form submission, make an Ajax DELETE request to delete a student.
3. **Handle Response:** Display a confirmation message upon successful deletion.

Example:

```
<!-- Inside body tag -->
<form id="deleteStudentForm">
    <label for="id">Student ID:</label>
    <input type="number" id="id" name="id" required><br>
    <input type="submit" value="Delete Student">
</form>
<p id="deleteConfirmationMessage"></p>

<script>
$("#deleteStudentForm").submit(function(e){
    e.preventDefault();
    var studentId = $("#id").val();
    $.ajax({
        url: "http://localhost:8080/studentapp/api/students/" + studentId,
        type: 'DELETE',
        success: function(result){
            $("#deleteConfirmationMessage").text("Student deleted successfully!");
        },
        error: function(error){
            console.log(error);
        }
    });
});
</script>
```

Task 6: Style the Views with CSS

1. **Create a CSS File:** Create a new CSS file to define the styles for your HTML elements. Name it `styles.css`.
2. **Link CSS to HTML:** Link the `styles.css` file to each of your HTML files using the `<link>` element in the `<head>` section:
   `<link rel="stylesheet" href="styles.css">`
3. **Style Elements:** Add styles to enhance the appearance of your forms, tables, buttons, headers, and footers.

Example of `styles.css`:

```
body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
}

header {
    background: #50b3a2;
    color: #ffffff;
    padding: 1em 0;
    text-align: center;
}
```

```css
footer {
    background: #50b3a2;
    color: #ffffff;
    text-align: center;
    padding: 1em 0;
    position: absolute;
    bottom: 0;
    width: 100%;
}

form {
    background: #ffffff;
    margin: 2em 0;
    padding: 2em;
    border-radius: 8px;
    box-shadow: 0px 0px 15px 1px rgba(0, 0, 0, 0.2);
}

label {
    display: block;
    margin: 0 0 1em 0;
}

input {
    width: 100%;
    padding: 8px;
    box-sizing: border-box;
    margin-bottom: 1em;
    border-radius: 4px;
    border: 1px solid #ccc;
}

input[type="submit"] {
    cursor: pointer;
    background-color: #50b3a2;
    color: #ffffff;
    border: none;
    padding: 10px 20px;
    border-radius: 4px;
}

input[type="submit"]:hover {
    background-color: #4a9188;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin: 2em 0;
}

th, td {
    padding: 8px 12px;
    border: 1px solid #ddd;
}
```

```
th {
    background-color: #50b3a2;
    color: white;
}
```

Testing

Test each view to ensure it interacts correctly with the REST APIs. Pay attention to the handling of API responses and the dynamic updating of the user interface.