

CST-350 Activity 7 REST API and Right Click Events

Alex M. Frear

College of Science, Engineering, and Technology, Grand Canyon University

Course Number: CST-350

Professor Brandon Bass

12/4/2024

GitHub Link:

https://github.com/amfrear/cst350/tree/main/Activity_7

Contents

GitHub Link	1
Part 1: RESTful Services and CRUD Operations.....	3
Screenshots	3
1. Testing ShowAllProducts Endpoint.....	3
2. Testing GetProductById Endpoint.....	5
3. Testing SearchForProducts Endpoint	7
4. Testing CreateProduct Endpoint.....	11
5. Testing UpdateProduct Endpoint	13
6. Testing DeleteProduct Endpoint	15
Summary of Key Concepts (Part 1)	16
Part 2: Right Click Event and Button Updates.....	17
Screenshots	17
1. Left-Click Functionality.....	17
2. Right-Click Functionality	22
Summary of Key Concepts (Part 2)	27

Part 1: RESTful Services and CRUD Operations

Screenshots

This section demonstrates the implementation and testing of RESTful endpoints for CRUD operations in the ProductsApp. Each step highlights the setup, execution, and verification of different API endpoints using both the browser and Postman.

1. Testing ShowAllProducts Endpoint

The ShowAllProducts endpoint is tested in the browser to confirm it returns a 200 OK status with all products.

The screenshot shows a web browser at the URL `localhost:7293/api/v1/products`. The page displays a JSON array of three product objects. The Network tab is open, showing a successful GET request to the `products` endpoint with a 200 status and a response size of 1.5 kB.

```
[
  {
    "id": "5",
    "name": "Laptop (edit)",
    "price": 1500,
    "formattedPrice": "$1,500.00",
    "description": "A Laptop for Work or Fun.",
    "createdAt": null,
    "formattedDateTime": null,
    "imageUrl": "LaptopProduct.png",
    "imageFile": null,
    "estimatedTax": 120,
    "formattedEstimatedTax": null
  },
  {
    "id": "6",
    "name": "Smart Phone",
    "price": 600,
    "formattedPrice": "$600.00",
    "description": "A smart phone for everyday needs.",
    "createdAt": null,
    "formattedDateTime": null,
    "imageUrl": "SmartphoneProduct.png",
    "imageFile": null,
    "estimatedTax": 48,
    "formattedEstimatedTax": null
  },
  {
    "id": "7",
    "name": "Headphones (edit)",
    "price": 200,
    "formattedPrice": "$200.00",
    "description": "Headphones for gaming or music.",
    "createdAt": null,
    "formattedDateTime": null,
    "imageUrl": "HeadphonesProduct.png",
    "imageFile": null,
    "estimatedTax": 15,
    "formattedEstimatedTax": null
  }
]
```

Name	Status	Type	Initiator	Size	Time
products	200	document	Other	1.5 kB	75 ms
favicon.ico	200	x-icon	Other	(disk cache)	1 ms
CircularXXWeb-Book.woff2	200	font	Other	69.0 kB	48 ms

3 requests | 70.5 kB transferred | 75.8 kB resources | Finish: 645 ms | DOMContentLoaded: 136 ms | Load: 496 ms

Figure 1 The browser displays all products successfully.

The screenshot displays the Postman application interface. The top navigation bar includes 'Home', 'Workspaces', and 'API Network'. The left sidebar shows 'My Workspace' with a tree view of collections and environments. The main workspace is titled 'ProductsApp / ShowAllProducts' and shows a 'GET' request to 'https://localhost:7293/api/v1/products'. The 'Send' button is visible. Below the request bar, the 'Params' tab is active, showing a table for 'Query Params' with columns 'Key', 'Value', and 'Description'. The 'Body' tab is also visible, showing a '200 OK' status with a response time of 33 ms and a size of 1.46 KB. The response body is displayed in 'Pretty' format, showing a JSON array of product objects. The bottom status bar indicates '1 Warning' and 'All Logs'.

ProductsApp / ShowAllProducts

GET https://localhost:7293/api/v1/products

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (4) Test Results

200 OK • 33 ms • 1.46 KB

Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": "5",
4     "name": "Laptop (edit)",
5     "price": 1500.00,
6     "formattedPrice": "$1,500.00",
7     "description": "A Laptop for Work or Fun.",
8     "createdAt": null,
9     "formattedDateTime": null,
10    "imageUrl": "LaptopProduct."
  }
```

GET https://localhost:7293/api/v1/products 200 33 ms

Figure 2 Postman confirms the endpoint displays all products correctly.

2. Testing GetProductById Endpoint

The GetProductById endpoint is tested in the browser and Postman to retrieve a specific product by its ID.

The screenshot shows a web browser at `localhost:7293/api/v1/products/5`. The response is a JSON object representing a product:

```
{
  "id": "5",
  "name": "Laptop (edit)",
  "price": 1500,
  "formattedPrice": "$1,500.00",
  "description": "A Laptop for Work or Fun.",
  "createdAt": null,
  "formattedDateTime": null,
  "imageUrl": "LaptopProduct.png",
  "imageFile": null,
  "estimatedTax": 120,
  "formattedEstimatedTax": null
}
```

The Network tab shows the following request details:

Name	Status	Type	Initiator	Size	Time
5	200	document	Other	507 B	36 ms
CircularXXWeb-Book.woff2	200	font	Other	69.0 kB	56 ms

Summary: 2 requests | 69.5 kB transferred | 69.5 kB resources | Finish: 618 ms | DOMContentLoaded: 114 ms | Load: 468 ms

Figure 3 Browser displays a product retrieved successfully.

The screenshot shows the Postman interface with a workspace named 'ProductsApp'. A GET request is configured for the endpoint `https://localhost:7293/api/v1/products/5`. The request is sent, and the response is displayed in the 'Body' tab, showing a 200 OK status with a response time of 9 ms and a body size of 594 B. The response body is a JSON object representing a product.

Key	Value	Description
Key	Value	Description

```

1 {
2   "id": "5",
3   "name": "Laptop (edit)",
4   "price": 1500.00,
5   "formattedPrice": "$1,500.00",
6   "description": "A Laptop for Work or Fun.",
7   "createdAt": null,
8   "formattedDateTime": null,
9   "imageUrl": "LaptopProduct.
  png

```

The console at the bottom shows the request log: GET https://localhost:7293/api/v1/products/5 with a status of 200 and a response time of 9 ms.

Figure 4 Postman confirms the endpoint returns the product details correctly.

3. Testing SearchForProducts Endpoint

The SearchForProducts endpoint is tested in the browser for both successful searches and error handling when parameters are missing.

The screenshot shows a web browser at `localhost:7293/api/v1/products/search?searchTerm=laptop&inTitle=true&inDescription=true`. The left pane displays the JSON response for a search result:

```
[
  {
    "id": "5",
    "name": "Laptop (edit)",
    "price": 1500,
    "formattedPrice": "$1,500.00",
    "description": "A Laptop for Work or Fun.",
    "createdAt": null,
    "formattedDateTime": null,
    "imageURL": "LaptopProduct.png",
    "imageFile": null,
    "estimatedTax": 120,
    "formattedEstimatedTax": null
  }
]
```

The right pane shows the Network tab with two requests:

Name	Status	Type	Initiator	Size	Time
search?searchTerm=laptop...	200	document	Other	509 B	34 ms
CircularXXWeb-Book.woff2	200	font	Other	69.0 kB	55 ms

At the bottom of the Network tab, the following summary is shown: 2 requests | 69.5 kB transferred | 69.5 kB resources | Finish: 680 ms | DOMContentLoaded: 111 ms | Load: 462 ms

Figure 5 Browser displays search results successfully.

The screenshot shows a web browser at the URL `localhost:7293/api/v1/products/search`. The console displays a JSON error response:

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "errors": {
    "searchTerm": [
      "The searchTerm field is required."
    ]
  },
  "traceId": "00-54e5fe2f7e1105f67ef0cd2136631085-104189b14c2238d7-00"
}
```

The network tab shows two requests:

Name	Status	Type	Initiator	Size	Time
search	400	document	Other	355 B	25 ms
CircularXXWeb-Book.woff2	200	font	Other	69.0 kB	46 ms

Summary: 2 requests | 69.4 kB transferred | 69.3 kB resources | Finish: 581 ms | DOMContentLoaded: 116 ms | Load: 447 ms

Figure 6 Browser displays a 400 Bad Request error when required parameters are missing.

The screenshot displays the Postman interface with a workspace named 'ProductsApp'. The active collection is 'SearchForProducts', and the selected request is a GET request to the endpoint `https://localhost:7293/api/v1/products/search?searchTerm=laptop&inTitle=true&inDescription=true`. The request parameters are defined as follows:

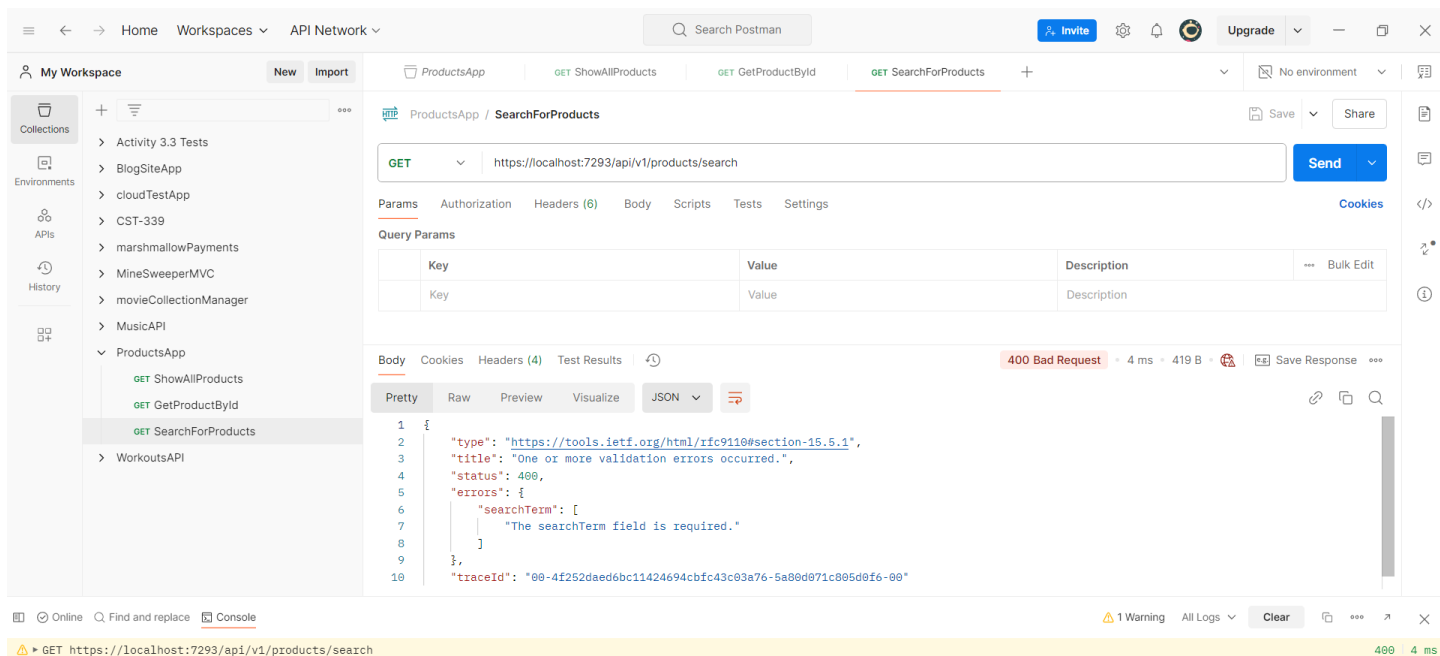
Key	Value	Description
<input checked="" type="checkbox"/> searchTerm	laptop	
<input checked="" type="checkbox"/> inTitle	true	
<input checked="" type="checkbox"/> inDescription	true	

The response status is **200 OK** with a response time of 17 ms and a body size of 596 B. The response body is displayed in JSON format:

```
1 [
2   {
3     "id": "5",
4     "name": "Laptop (edit)",
5     "price": 1500.00,
6     "formattedPrice": "$1,500.00",
7     "description": "A Laptop for Work or Fun.",
8     "createdAt": null,
9     "formattedDateTime": null,
10    "imageUrl": "LaptopProduct."
  }
```

The bottom status bar shows a warning icon and the text '1 Warning All Logs Clear'. The console at the bottom displays the GET request and its response time of 200 ms.

Figure 7 Postman confirms successful search results.



The screenshot shows the Postman interface with a workspace named "My Workspace". The selected collection is "ProductsApp" and the specific request is "SearchForProducts". The request method is "GET" and the URL is "https://localhost:7293/api/v1/products/search". The response status is "400 Bad Request" with a response time of 4 ms and a size of 419 B. The response body is a JSON object:

```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
3   "title": "One or more validation errors occurred.",
4   "status": 400,
5   "errors": {
6     "searchTerm": [
7       "The searchTerm field is required."
8     ]
9   },
10  "traceId": "00-4f252daed6bc11424694cbfc43c03a76-5a00d071c005d0f6-00"
```

The console at the bottom shows the request details: GET https://localhost:7293/api/v1/products/search, with a status of 400 and a response time of 4 ms.

Figure 8 Postman displays an error message when no parameters are provided.

4. Testing CreateProduct Endpoint

A new product is created using Postman, and its successful addition is verified by fetching all products.

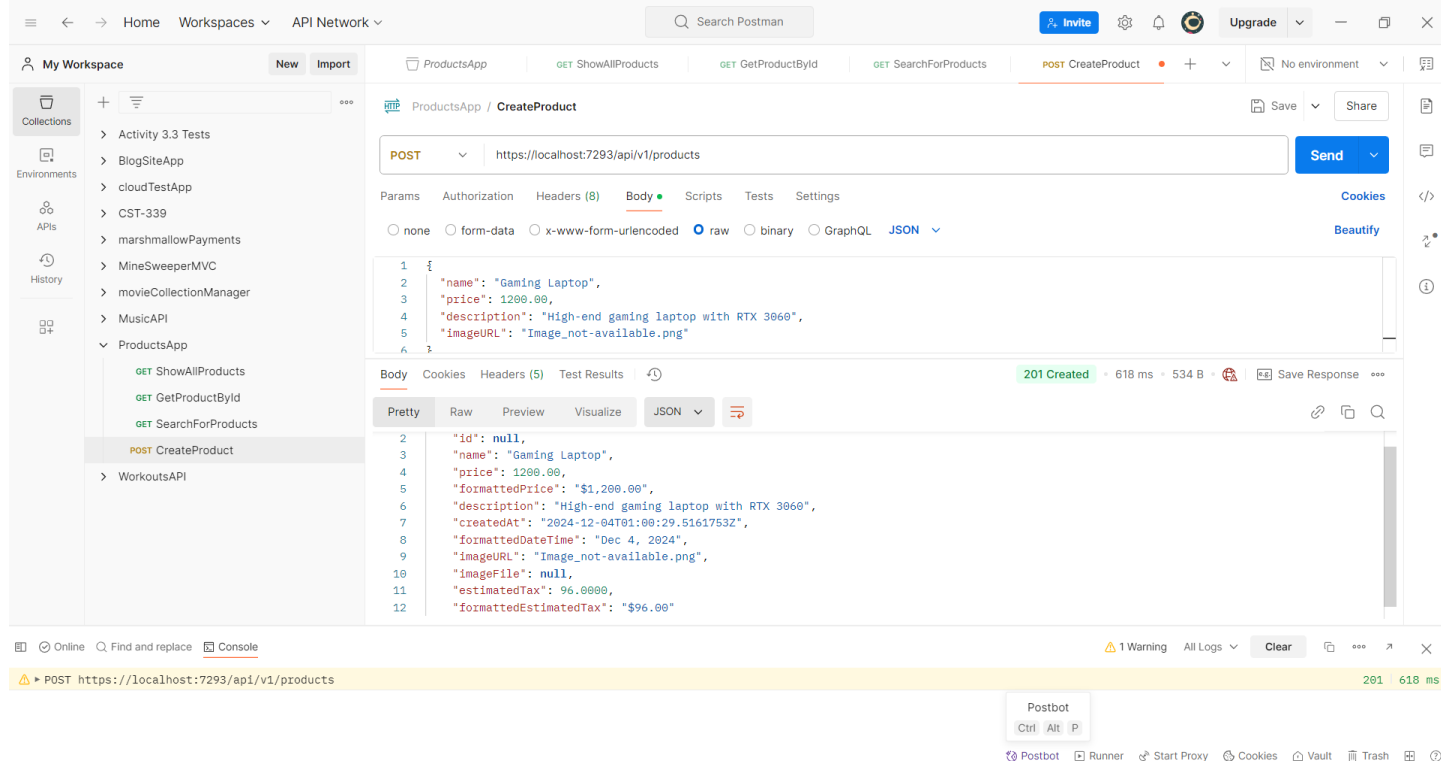


Figure 9 Postman confirms the product was created successfully.

The screenshot displays the Postman application interface. The top bar includes navigation icons, a search bar, and an 'Upgrade' button. The left sidebar shows the 'My Workspace' with a tree view of collections and environments. The main area is divided into tabs for 'ProductsApp' and 'ShowAllProducts'. The 'ShowAllProducts' tab is active, showing a GET request to 'https://localhost:7293/api/v1/products'. The response is a 200 OK status with a JSON body containing product details. The console at the bottom shows the request and response times.

GET `https://localhost:7293/api/v1/products` **Send**

Params Authorization Headers (6) Body Scripts Tests Settings **Cookies**

Query Params

Key	Value	Description
-----	-------	-------------

Body Cookies Headers (4) Test Results **200 OK** 63 ms 2.86 KB Save Response

Pretty Raw Preview Visualize JSON **JSON**

```
67 {
68   "id": "1004",
69   "name": "Gaming Laptop",
70   "price": 1200.00,
71   "formattedPrice": "$1,200.00",
72   "description": "High-end gaming laptop with RTX 3060",
73   "createdAt": "2024-12-04T01:00:29.517",
74   "formattedDateTime": "Wednesday, December 4, 2024",
75   "imageUrl": "Image_not-available.png",
76   "imageFile": null,
77   "estimatedTax": 96.0000,
78   "formattedEstimatedTax": null
79 }
80 ]
```

2 Warnings All Logs **Clear**

Request	Response
POST <code>https://localhost:7293/api/v1/products</code>	201 618 ms
GET <code>https://localhost:7293/api/v1/products</code>	200 63 ms

Postbot **Ctrl Alt P**

Postbot Runner Start Proxy Cookies Vault Trash

Figure 10 Postman verifies the new product is included in the list of all products.

5. Testing UpdateProduct Endpoint

An existing product is updated using Postman, and the changes are verified by fetching the updated product details.

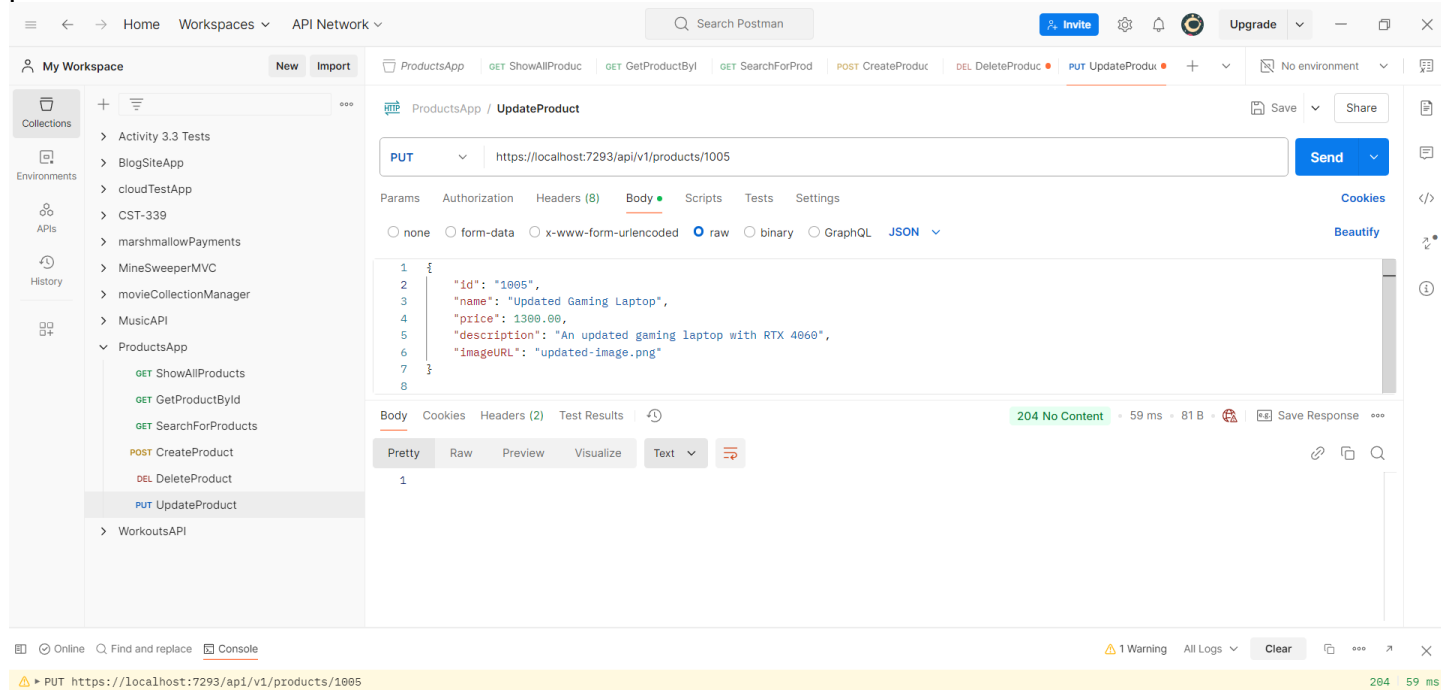


Figure 11 Postman confirms the product update was successful.

The screenshot displays the Postman application interface. The top bar shows navigation options like Home, Workspaces, and API Network, along with a search bar and user profile. The left sidebar lists collections and environments. The main workspace shows a GET request to `https://localhost:7293/api/v1/products/1005` with a status of `200 OK`. The response body is a JSON object representing a product.

```
1 {
2   "id": "1005",
3   "name": "Updated Gaming Laptop",
4   "price": 1300.00,
5   "formattedPrice": "$1,300.00",
6   "description": "An updated gaming laptop with RTX 4060",
7   "createdAt": null,
8   "formattedDateTime": null,
9   "imageURL": "updated-image.png",
10  "imageFile": null,
11  "estimatedTax": 104.00000,
12  "formattedEstimatedTax": null
13 }
```

The bottom console shows the execution of the PUT and GET requests, both returning 200 status codes.

Figure 12 Postman verifies the product's details reflect the update.

6. Testing DeleteProduct Endpoint

A product is deleted using Postman, and its removal is confirmed by fetching all products.

The screenshot shows the Postman interface with a workspace named 'My Workspace'. The 'ProductsApp' collection is selected, and the 'DeleteProduct' endpoint is highlighted. The request is a DELETE method to the URL 'https://localhost:7293/api/v1/products/1004'. The response is '204 No Content' with a status of 204 and a response time of 17 ms. The console at the bottom shows the request and response details.

ProductsApp / DeleteProduct

DELETE https://localhost:7293/api/v1/products/1004

Params Authorization Headers (6) Body Scripts Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (2) Test Results

204 No Content 17 ms 81 B

1

DELETED https://localhost:7293/api/v1/products/1004 204 17 ms

Figure 13 Postman confirms the product was deleted successfully.

Summary of Key Concepts (Part 1)

In this part of Activity 7, I developed and tested RESTful endpoints for CRUD operations in the ProductsApp. Each endpoint was verified using both browser-based tools and Postman. The activity reinforced my understanding of building and testing API endpoints and handling HTTP status codes for successful and error scenarios. This hands-on experience strengthened my skills in creating reliable and well-documented RESTful services for modern web applications.

Part 2: Right Click Event and Button Updates

Screenshots

This section demonstrates the implementation of left-click and right-click functionality for the buttons in the ButtonGrid application. AJAX-based methods are used to dynamically update the button state and image without refreshing the page.

1. Left-Click Functionality

The left-click functionality updates the button's state by incrementing it and cycling through different images.

Initial Button State

The screenshot displays the ButtonGrid application running on localhost:7276. The main content area shows a 5x5 grid of buttons, each with a unique color and a state value (e.g., 0, 3; 1, 3; 2, 2; 3, 2; 4, 2). A notification box above the grid states: "localhost:7276 says Left mouse button clicked on item 0." The browser's developer tools are open, showing the Network tab with a log of an XHR request: "negotiate?requestUrl=https%3A%2F%2Flocalhost:7276/...". The status is 200, and the initiator is "browserlink:21".

Name	Status	Type	Initiator	Size	Time
negotiate?requestUrl=https%3A%2F%2Flocalhost:7276/...	200	xhr	browserlink:21	955 B	3 ms

Figure 14 The button displays its initial state and image.

Left-Click Event Triggered

The screenshot shows a web browser window with the address bar at `localhost:7276/Button/Index`. The page displays a grid of 25 buttons arranged in 5 rows and 5 columns. Each button is a colored circle with a number and a letter below it. The buttons are: Row 1: Orange (0,2), Pink (1,3), Orange (2,2), Orange (3,2), Orange (4,2); Row 2: Orange (5,2), Green (6,1), Green (7,1), Blue (8,0), Blue (9,0); Row 3: Pink (10,3), Green (11,1), Green (12,1), Blue (13,0), Blue (14,0); Row 4: Blue (15,0), Orange (16,2), Pink (17,3), Pink (18,3), Blue (19,0); Row 5: Blue (20,0), Green (21,1), Pink (22,3), Orange (23,2), Orange (24,2). Above the grid, the text "The time is 12/4/2024 4:42:18 PM" is displayed. A modal dialog box is open in the center, titled "localhost:7276 says", with the message "Left mouse button clicked on item 0." and an "OK" button. The browser's developer tools are open, showing the Network tab. A single request is listed: `negotiate?requestUrl=https%3A%2F%2Flocalhost:7276/...` with status 200, type xhr, and initiator `browserLink21`. The bottom of the page shows a copyright notice: "© 2024 - ButtonGrid - Privacy".

ButtonGrid Home Privacy

The time is 12/4/2024 4:42:18 PM

0, 2 1, 3 2, 2 3, 2 4, 2

5, 2 6, 1 7, 1 8, 0 9, 0

10, 3 11, 1 12, 1 13, 0 14, 0

15, 0 16, 2 17, 3 18, 3 19, 0

20, 0 21, 1 22, 3 23, 2 24, 2

localhost:7276 says
Left mouse button clicked on item 0.

OK

Network

Name	Status	Type	Initiator	Size	Time
negotiate?requestUrl=https%3A%2F%2Flocalhost:7276/...	200	xhr	browserLink21	955 B	3 ms

1 / 20 requests | 955 B / 246 kB transferred | 655 B / 597 kB resources | Finish: 638 ms | DOMContentLoaded: 157 ms | L

© 2024 - ButtonGrid - Privacy

Figure 15 An alert confirms the left-click event was triggered.

Left-Click Function Executed

The screenshot shows a web browser window with the address bar at `localhost:7276/Button/Index`. The page displays a grid of 25 buttons arranged in 5 rows and 5 columns. Each button is a colored circle with a number pair below it. The buttons are: Row 1: 0,3 (pink), 1,3 (pink), 2,2 (orange), 3,2 (orange), 4,2 (orange); Row 2: 5,2 (orange), 6,1 (green), 7,1 (green), 8,0 (blue), 9,0 (blue); Row 3: 10,3 (pink), 11,1 (green), 12,1 (green), 13,0 (blue), 14,0 (blue); Row 4: 15,0 (blue), 16,2 (orange), 17,3 (pink), 18,3 (pink), 19,0 (blue); Row 5: 20,0 (blue), 21,1 (green), 22,3 (pink), 23,2 (orange), 24,2 (orange). A modal dialog box is open over the first button (0,3) with the text "localhost:7276 says do left click 0" and an "OK" button. The browser's developer tools are open to the Network tab, showing a single request: `negotiate?requestUrl=https%3A%2F%2Flocalhost%3A7276%2FButton%2FIndex%3FbuttonId%3D0`. The request details show a status of 200, type of xhr, and initiator of `browserLink21`. The bottom status bar indicates 1 / 20 requests, 955 B / 246 kB transferred, 655 B / 597 kB resources, Finish: 844 ms, and DOMContentLoaded: 275 ms.

The time is 12/4/2024 4:45:07 PM

localhost:7276 says
do left click 0

OK

Name	Status	Type	Initiator	Size	Time
negotiate?requestUrl=https%3A%2F%2Flocalhost%3A7276%2FButton%2FIndex%3FbuttonId%3D0	200	xhr	browserLink21	955 B	3 ms

1 / 20 requests | 955 B / 246 kB transferred | 655 B / 597 kB resources | Finish: 844 ms | DOMContentLoaded: 275 ms | L

Figure 16 The `doLeftClick` function is executed, sending an AJAX request to the server.

HTML Returned from the Server

The screenshot shows a web browser at `localhost:7276/Button/Index`. The page displays a grid of 25 buttons, each with a number and a letter (e.g., "0, 3", "1, 3", etc.). The time shown is "The time is 12/4/2024 4:45:07 PM".

An alert box is open, displaying the HTML returned from the server:

```
<div class="game-button" data-id="0">
  <button type="submit" name="id" value="0">
    
    <div>0, 0</div>
  </button>
</div>
```

The browser's developer tools are open, showing the Network tab. The table below lists the requests:

Status	Type	Initiator	Size	Time
200	xhr	browserLink21	955 B	3 ms
200	xhr	jquery.min.js-2	43 B	64 ms
200	xhr	browserLink21	258 B	3 ms
200	xhr	browserLink21	955 B	3 ms

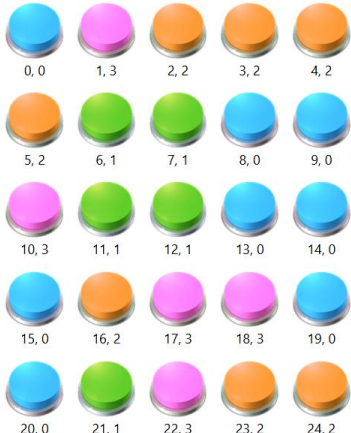
At the bottom of the browser window, the status bar shows: 4 / 24 requests | 2.2 kB / 248 kB transferred | 1.5 kB / 598 kB resources | Finish: 1.8 min | DOMContentLoaded: 275 ms

Figure 17 An alert displays the updated HTML returned by the server.

Updated Button State

ButtonGrid Home Privacy

The time is 12/4/2024 4:45:07 PM



© 2024 - ButtonGrid - [Privacy](#)

Network

Filter

Name	Status	Type	Initiator	Size	Time
negotiate?requestUrl=https%3A%2F%2Ffloc...	200	xhr	browserLink21	955 B	3 ms
PartialPageUpdate	200	xhr	jquery.min.js-2	246 B	63 ms
abort?transport=webSockets&connectionTo...	200	xhr	browserLink21	258 B	3 ms
negotiate?requestUrl=https%3A%2F%2Ffloc...	200	xhr	browserLink21	955 B	3 ms
abort?transport=webSockets&connectionTo...	200	xhr	browserLink21	258 B	2 ms
negotiate?requestUrl=https%3A%2F%2Ffloc...	200	xhr	browserLink21	955 B	3 ms

6 / 27 requests | 3.6 kB / 249 kB transferred | 2.2 kB / 599 kB resources | Finish: 2.4 min | DOMContentLoaded: 275 ms

Figure 18 The button reflects the new state and updated image after the left-click.

2. Right-Click Functionality

The right-click functionality updates the button's state by decrementing it and cycling through different images.

Initial Button State

The screenshot shows a web browser at `localhost:7276/Button/Index`. The page displays a 5x5 grid of buttons. Each button has a unique color and a coordinate pair (x, y) below it. The coordinates range from (0, 0) to (24, 2). The buttons are arranged in a grid that is 5 columns wide and 5 rows high. The colors of the buttons are: Row 0: Blue, Pink, Orange, Orange, Orange; Row 1: Orange, Green, Green, Blue, Blue; Row 2: Pink, Green, Green, Blue, Blue; Row 3: Blue, Orange, Pink, Pink, Blue; Row 4: Blue, Green, Pink, Orange, Orange.

The browser's developer tools are open, showing the Network tab. A request is visible with the name `negotiate?requestUrl=https%3A%2F%2Flocalhost%3A7276%2FButton%2FIndex`, status 200, type xhr, and initiator `browserLink21`. The size is 955 B and the time is 2 ms.

© 2024 - ButtonGrid - [Privacy](#)

Figure 19 The button displays its initial state and image.

Right-Click Event Triggered

The screenshot shows a web browser window with the address bar displaying `localhost:7276/Button/Index`. The page content includes a header with "ButtonGrid", "Home", and "Privacy" links. Below the header, a message states "The time is 12/4/2024 4:48:05 PM". A 5x5 grid of 25 buttons is displayed, each with a unique color and a coordinate pair (row, column) ranging from (0,0) to (24,2). An alert box is visible, stating "localhost:7276 says" and "Right mouse button clicked on item 1.", with an "OK" button. The browser's developer tools are open, showing the "Network" tab. A single request is listed: `negotiate?requestUrl=https%3A%2F%2Flocalhost:7276/...` with a status of 200, type of xhr, and initiator of `browserLink21`. The network log table is as follows:

Name	Status	Type	Initiator	Size	Time
<code>negotiate?requestUrl=https%3A%2F%2Flocalhost:7276/...</code>	200	xhr	<code>browserLink21</code>	955 B	2 ms

At the bottom of the browser window, a footer displays the copyright information: "© 2024 - ButtonGrid - [Privacy](#)".

Figure 20 An alert confirms the right-click event was triggered.

Right-Click Function Executed

The screenshot shows a web browser window at `localhost:7276/Button/Index`. The page displays a 5x5 grid of buttons, each with a unique color and a coordinate pair. The buttons are arranged in a grid with the following coordinates:

Row	Col 1	Col 2	Col 3	Col 4	Col 5
1	0, 0	1, 3	2, 2	3, 2	4, 2
2	5, 2	6, 1	7, 1	8, 0	9, 0
3	10, 3	11, 1	12, 1	13, 0	14, 0
4	15, 0	16, 2	17, 3	18, 3	19, 0
5	20, 0	21, 1	22, 3	23, 2	24, 2

A right-click event has been triggered on the button at coordinate (1, 3). A dialog box from `localhost:7276` says "do right click 1" and has an "OK" button. The browser's developer tools are open, showing the Network tab with a single request:

Name	Status	Type	Initiator	Size	Time
<code>negotiate?requestUrl=https%3A%2F%2Flocalhost:7276/...</code>	200	xhr	<code>browserLink21</code>	955 B	2 ms

The footer of the page reads: © 2024 - ButtonGrid - [Privacy](#).

Figure 21 The `doRightClick` function is executed, sending an AJAX request to the server.

HTML Returned from the Server

The screenshot shows a web browser at `localhost:7276/Button/Index`. The page displays a grid of 25 buttons arranged in 5 rows and 5 columns. Each button is a colored circle with a number pair below it. The numbers range from 0,0 to 24,2. The time displayed at the top of the page is 12/4/2024 4:48:05 PM.

An alert box is open, titled `localhost:7276 says`, showing the HTML returned from the server:

```
<div class="game-button" data-id="1">
  <button type="submit" name="id" value="1">
    
    <div>1, 0</div>
  </button>
</div>
```

The browser's developer tools are open, showing the Network tab. The table below lists the requests:

Status	Type	Initiator	Size	Time
200	xhr	browserLink21	955 B	2 ms
200	xhr	jquery.min.js-2	43 B	41 ms
200	xhr	browserLink21	258 B	18 ms

At the bottom of the browser window, the status bar shows: 3 / 22 requests | 1.3 kB / 246 kB transferred | 849 B / 598 kB resources | Finish: 1.7 min | DOMContentLoaded: 231 ms |

Figure 22 An alert displays the updated HTML returned by the server.

Updated Button State

The screenshot shows a web browser at `localhost:7276/Button/Index`. The page displays a 5x5 grid of buttons, each with a unique color and a coordinate pair. The time shown is 12/4/2024 4:48:05 PM.

The buttons are arranged in a 5x5 grid with the following coordinates (row, column):

- Row 1: (0, 0) Blue, (1, 0) Blue, (2, 2) Orange, (3, 2) Orange, (4, 2) Orange
- Row 2: (5, 2) Orange, (6, 1) Green, (7, 1) Green, (8, 0) Blue, (9, 0) Blue
- Row 3: (10, 3) Pink, (11, 1) Green, (12, 1) Green, (13, 0) Blue, (14, 0) Blue
- Row 4: (15, 0) Blue, (16, 2) Orange, (17, 3) Pink, (18, 3) Pink, (19, 0) Blue
- Row 5: (20, 0) Blue, (21, 1) Green, (22, 3) Pink, (23, 2) Orange, (24, 2) Orange

The network console shows the following requests:

Name	Status	Type	Initiator	Size	Time
negotiate?requestUrl=https%3A%2F%2Ffloc...	200	xhr	browserLink21	955 B	2 ms
RightClickShowOneButton	200	xhr	jquery.min.js-2	255 B	37 ms
abort?transport=webSockets&connectionTo...	200	xhr	browserLink21	258 B	18 ms
negotiate?requestUrl=https%3A%2F%2Ffloc...	200	xhr	browserLink21	955 B	17 ms

At the bottom of the page, there is a copyright notice: © 2024 - ButtonGrid - [Privacy](#).

Figure 23 The button reflects the new state and updated image after the right-click.

Summary of Key Concepts (Part 2)

In this part of Activity 7, I implemented left-click and right-click functionality for dynamically updating button states in the ButtonGrid application. Using JavaScript and AJAX, I was able to handle mouse events and update the button state and image efficiently without a full-page reload. This exercise reinforced my understanding of client-server interactions and AJAX-based updates for responsive web applications.