Asynchronous Requests

Jerry Cain and Avi Gupta
CS 106AX
November 17, 2023

Asynchronous Data Requests

- When you load a website, the initial HTML document often references many other resources (images, JavaScript files, CSS stylesheets, and so forth) to be downloaded as well.
- Some resources—images and videos, in particular—can be very large. The larger the resources, the longer it takes for the application to load and properly function.
- Most web applications minimize the amount of data needed for the initial user interaction. By doing so, their website loads more quickly, and the application is operational with minimal lag time.
- As new resources are needed, the website can download those resources in the background—that is, ***asynchronously***. Once the new resources have been downloaded, JavaScript can programmatically update the DOM to incorporate them.

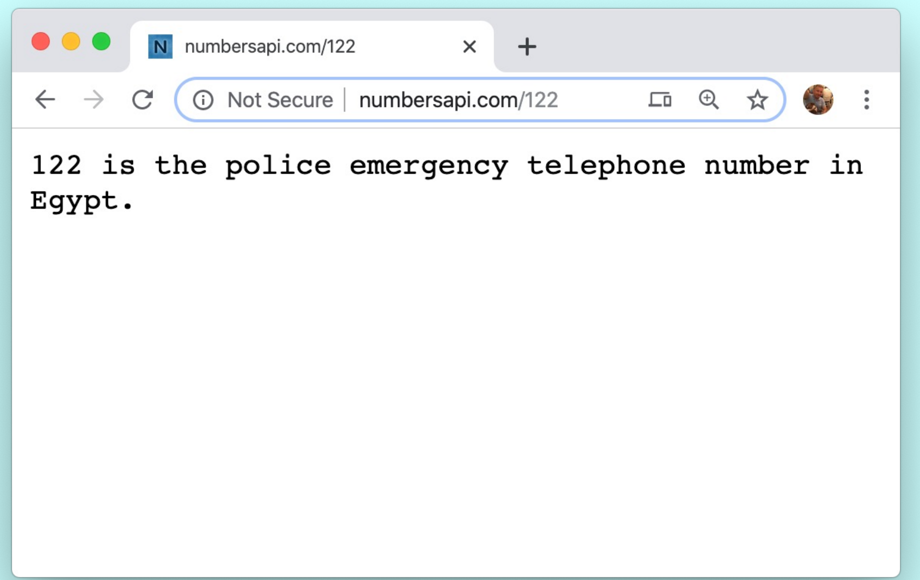
Asynchronous Data Requests

- The initial resource needed by most web applications is almost always formatted as HTML, which includes data and information about how they should be presented in the browser.
- Asynchronous requests, however, generally fetch data as plaintext or JSON format without the surrounding HTML. The new data can be programmatically inserted into the DOM.
- As with the initial request for the HTML document, asynchronous data requests are framed as URLs, as with:

  - **`graph.facebook.com/me`** *(Facebook API)*
  - **`api.x.com/1.1/statuses/user_timeline.json`** *(X/Twitter API)*
  - **`numbersapi.com/122`** *(Toy API providing fun facts about numbers)*
  - **`numbersapi.com/122?json=true`** *(Same API that formats response as JSON)*

- Most servers can infer from the URL itself whether the response should be HTML, plaintext, or JSON.

Asynchronous Data Requests



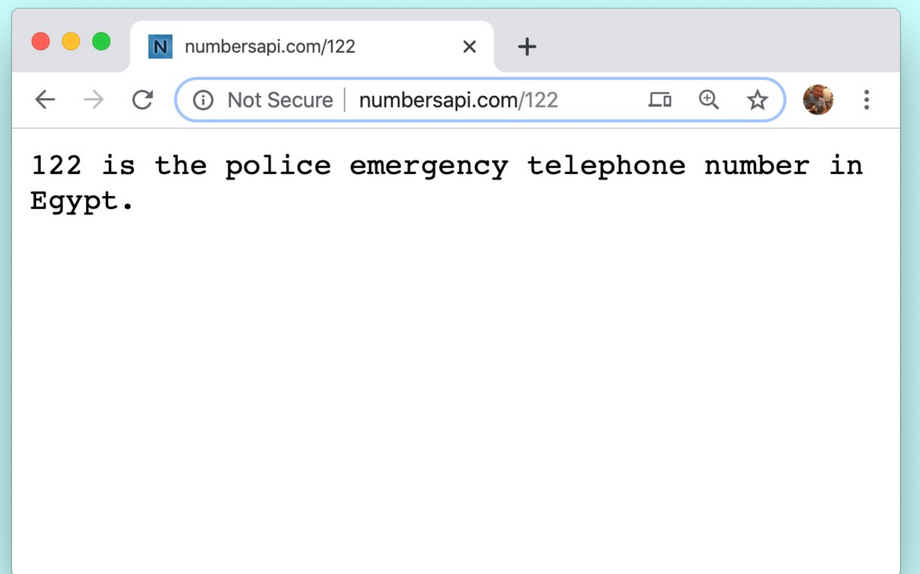
numbersapi.com/122?json=true

Not Secure | numbersapi.com/122?json=true


```
{
 "text": "122 is the police emergency telephone number in Egypt.",
 "number": 122,
 "found": true,
 "type": "trivia"
}
```


- The URL used on the left fetches a plaintext resource with trivia about the number 122 and renders it once it arrives.
- The URL on the right requests the same resource, formatted as JSON instead of plaintext.
- In both cases, the browser opens a connection to the relevant server. It then issues a request for the resource associated with the URL, waits for the server to respond, and renders the response’s payload once it arrives.

Asynchronous Data Requests



numbersapi.com/122?json=true

Not Secure | numbersapi.com/122?json=true

```
{
 "text": "122 is the police emergency telephone number in Egypt.",
 "number": 122,
 "found": true,
 "type": "trivia"
}
```

- Life would be terrific if we had JavaScript functionality to do the same thing a browser can—to programmatically request a resource through some URL, wait for a server response, and then process that response to suit the needs to the application.

Fortunately, life is terrific!

Asynchronous Data Requests

- JavaScript provides built-in libraries to help asynchronously fetch resources as they're needed. Unfortunately, they rely on JavaScript features slightly beyond the scope of our syllabus.
- The CS106AX libraries include two classes that layer over the complexities of those built-ins so we can still asynchronously fetch new resources.
- Those new classes are called **`AsyncRequest`** and **`AsyncResponse`**.

```
let req = AsyncRequest("http://numbersapi.com/538");
req.addParams({json: true});
req.setSuccessHandler(function(response) {
   console.log(response.getPayload())
});
req.send();
```

- The code above illustrates how these classes can be used to publish some trivia about the number 538 to the console.

AsyncRequest and **AsyncResponse**

AsyncRequest(*url***)**	Creates a request object primed to fetch a document from *url*.
*request***.addParam(***key, value***)**	Adds a key-value pair to the *url*'s of query string.
*request***.addParams(***params***)**	Adds all key-values pairs in *params* to the *url*'s query string.
*request***.setSuccessHandler(***callback***)**	Sets the *callback* to be executed when the server responds.
*request***.setErrorHandler(***callback***)**	Sets the *callback* to be executed if the server fails to fetch the *url*.
*request***.send()**	Initiates the request for the relevant resource.
*response***.getPayload()**	Returns the payload from the *response* passed to your *callback*.

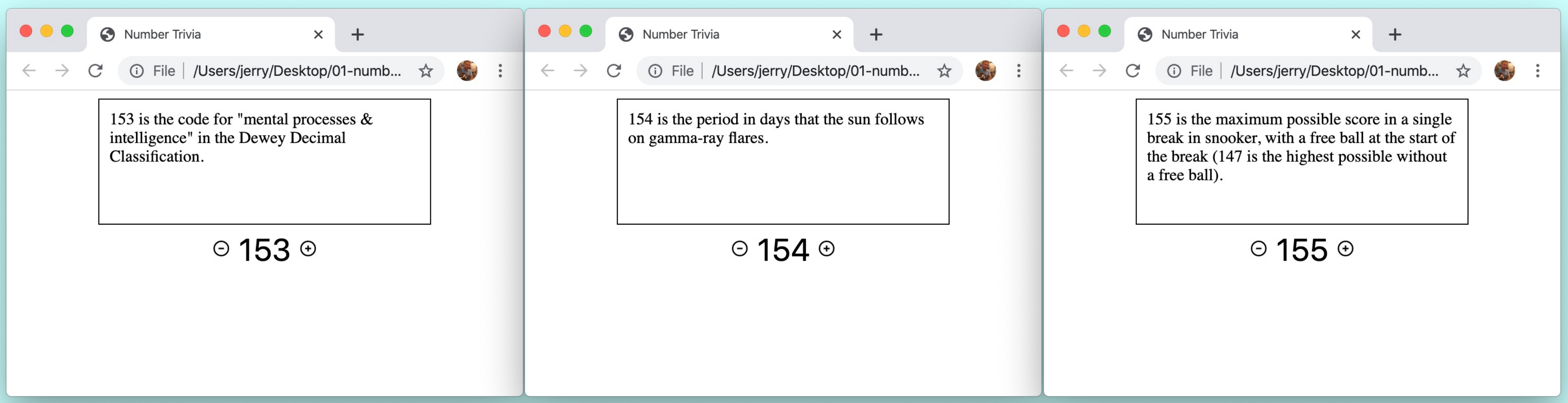
`AsyncResponse` and `JSON.parse`

- The callbacks passed to *request*.**`setSuccessHandler`** and *request*.**`setErrorHandler`** should be functions that take a single argument of type **`AsyncResponse`**.
- **`AsyncResponse`** exports several methods, but in a success scenario, the only one that matters is **`getPayload()`**.
- If the payload is formatted as a JSON string, then you should pass that string to the built-in **`JSON.parse`** function to rehydrate it into a true JavaScript object.

```
let req = AsyncRequest("http://numbersapi.com/222");
req.addParams({json: true});
req.setSuccessHandler(function(response) {
   let payload = response.getPayload();
   let info = JSON.parse(payload);
   console.log("Trivia: " + info.text);
});
req.send();
```

Exercise: Number Trivia Slideshow

Implement a program that generates a random number between 1 and 200 and presents a random piece of trivia about it. Allow the user to increment and decrement that number by clicking + and - buttons, and with each increment and decrement present some new morsel of information about the new number. Asynchronously fetch the trivia only as the number is being showcased.



Exercise: Number Trivia Slideshow

File: **number-trivia.html**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Number Trivia</title>
    <link rel="stylesheet" type="text/css"
          href="number-trivia.css">
    <script type="text/javascript" src="Async.js"></script>
    <script type="text/javascript" src="number-trivia.js"></script>
  </head>
  <body>
    <div id="showcase" class="showcase"></div>
    <input id="decrement-button" type="image"
           src="minus.png" class="control-row"/>
    <input id="number" class="control-row number"
           value="42" readonly/>
    <input id="increment-button" type="image"
           src="plus.png" class="control-row"/>
   </body>
</html>
```

Exercise: Number Trivia Slideshow

File: **number-trivia.js (page 1)**

```
function BootstrapNumberTrivia() {
   /* all four variables are referenced by inner callback functions */
   let contentArea = document.getElementById("showcase");
   let number = document.getElementById("number");
   let decrementButton = document.getElementById("decrement-button");
   let incrementButton = document.getElementById("increment-button");
   decrementButton.addEventListener("click", decrementNumber);
   incrementButton.addEventListener("click", incrementNumber);

   function showcaseTrivia(response) {
      while (contentArea.childNodes.length > 0) {
         contentArea.removeChild(contentArea.lastChild);
      }
      let info = JSON.parse(response.getPayload());
      contentArea.appendChild(document.createTextNode(info.text));
   }

   function initiateNumberTriviaFetch() {
      let req = AsyncRequest("http://numbersapi.com/" + number.value);
      req.addParams({json: true});
      req.setSuccessHandler(showcaseTrivia);
      req.send();
   }
```

Exercise: Number Trivia Slideshow

File: **number-trivia.js (page 2)**

```
    function decrementNumber(e) {
        incrementButton.disabled = false;
        let current = parseInt(number.value) - 1;
        number.value = current.toString();
        if (current === 1) decrementButton.disabled = true;
        initiateNumberTriviaFetch();
    }

    function incrementNumber(e) {
        decrementButton.disabled = false;
        let current = parseInt(number.value) + 1;
        number.value = current.toString();
        if (current === 200) incrementButton.disabled = true;
        initiateNumberTriviaFetch();
    }

    /* seed initial presentation with a random number from [1, 200] */
    number.value = (Math.floor(Math.random() * 200) + 1).toString();
    initiateNumberTriviaFetch();
}

document.addEventListener("DOMContentLoaded", BootstrapNumberTrivia);
```

The End