Multiplayer Game Programming

HTTP Requests

JSON

Unity

Chapter 11/13

Lecture 9 (chapter 11/13) **Objectives**

- **HTTP Requests**
  - How they work
  - What they are made of

- **JSON Data**
  - How it works
  - Where it’s used?

- **Unity**
  - How HTTP Requests work in Unity
  - How to read JSON data in Unity

RPC’s

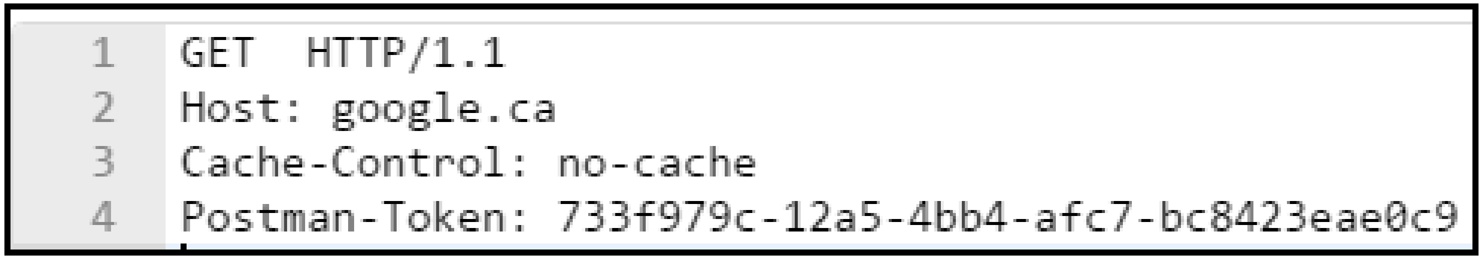
- RPC’s or Remote Procedural Calls are what the some game clients use to communicate with Game Servers.
  - Similar to an exposed API (Application Programming Interface).
  - Typically implemented through a HTTP (Hyper Text Transfer Protocol)
    - Allows an entry point that anyone on the internet can make a call in to.
  - Sockets are not strictly bound for the connection and instead are opened and closed quickly once the information is processed.
    - The advantage to this is we can have thousands of users communicating with the server at the exact same time.

HTTP REQUESTS

- HTTP requests are typically made using a TCP connection on port 80 by default.
  - When you type in http://www.google.ca you are in fact calling http://www.google.ca**:80**
    - Your browser automatically tacks on the missing port assuming you meant 80.
    - When we send in the name google.ca to our browser, this is known as a **Domain Name.**
      - In reality, a domain name is register with a specific IP address and makes life easier for us humans to remember than a sequence of 4, 3 digit segments like 172.217.1.3.
        - As an example, try typing in the above ip into a browser.
      - Web browsers also do other things for you behind the scenes, they formulate the HTTP request so the end point you specify knows what you want.

GET REQUESTS

- When a CLIENT builds a proper HTTP request and sends it to the SERVER with a header, or packet data.
- The initial packet data for a plain website access like google.ca simply adds a few lines of data to indicate that you wish to get information from the server to display to on your screen.
  - This is known as a GET HTTP request.
  - The formatted data is in multiple lines which separates each property of the GET request.
  - Using a tool called **POSTMAN**, we can observe the data from making an HTTP GET Request.



```
GET  HTTP/1.1
Host: google.ca
Cache-Control: no-cache
Postman-Token: 733f979c-12a5-4bb4-afc7-bc8423eae0c9
```

POST REQUESTS

- When a CLIENT browser accesses a more dynamic webpage which requires parameters like a form, the browser takes the clients entered information and forwards it with the request to the server.
  - This is known as a POST HTTP request.
  - This will be information the server uses to modify data or an object on the server.

- In terms of games, the POST request is issued when we call an RPC or exposed API which will modify data.

HTTP REQUESTS III

- GET and POST https requests are made to RESTFUL servers waiting to receive requests.
  - Requires a few parameters to properly communicate with the server.
    1. A URI to communicate which resource is being requested. (Uniform Resource Identifier)
       - https://www.someurl.com/**exposedAPIURI/**
    2. Headers – Modifiers which can be applied to the request to give additional details about the request.
       - Can contain Information required for the server to process the request like authorization data, and will often be rejected from the server if the request does not contain properly formed headers.
    3. Body: data required to complete the request.
       - This is typically the data you want the server to know about from the client.
       - Usually only used for POST requests

HTTP Request HEaders

- Headers are added to requests to define the context of the message/request as well the requirements for reading the request.
- Headers can have a variable number of parameters which are Key Value Pairs with each side requiring a string type.
- Header entries fall into two categories Standard and non-standard.
- Common Standard Headers Include:
  - Accept
  - Authorization
  - Content-Length
  - Content Type
- Non-Standard can be defined by the architect implementing the communication.
  - Typically preceded by a X- to indicate it is a custom non-standard header

- Full List here: https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

RESPONSE Errors

- An HTTP Response is used when the server processes the request and responds with a similar set of data to the request with the addition of a Response Code.
  - Responses also contain headers and a body
  - Response Headers contain most of the same headers as Requests, however, they also have a Status header (also known as a response code)

- Response Codes can be easily distinguished by their first value.
  - 100's = informational responses. (rarely used)
  - 200's = Successful message transfer
  - 300's = Redirection (rarely used)
  - 400's = bad requests (client side error with the request)
  - 500's = server side error processing the request.

- Full list is here: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

431

Request Header Fields Too Large

JSON

- Data for the body of HTTP requests are often made using JSON.
  - JavaScript Object Notation (JSON) is a data formatting which is easily readable and unordered.
    - Can be saved to a storage file with a .json extension.
  - A standard defined for data structure which allows for communication of data between multiple programming and scripting languages.
    - Allows objects to be serialized and deserialized from message data for easy transmission.
    - Most programming Languages have additional library supports available for free
  - Json stores data in a key value pair system where the key is always a string

401

Unauthorized

JSON Data

- Keys are always Strings in JSON however the value for the paired data can take a few different types:
  - String: a text based value
    - Example Json String: “StringKey” : “ValueExample”
  - Number: an Integer or floating point numeric value
    - Example Json Integer Number: “NumberIntKey” : 100
    - Example Json Float Number: “NumberFloatKey” : 100.0
  - Boolean: a true or false value
    - Example Json bool: “boolKey” : true

JSON Data Continued

The remaining data types for values:

- Array: Set of Data using any acceptable JSON value types including another array.
  - Arrays can be added to JSON objects as well defined using square braces [&] to contain it's elements.
  - Example Json Array: "MyArrayKey" : ["item1", "item2", "item3"]
- Object: structure containing one or more defines keyvalue pair
  - Uses curly braces { & } to compartmentalize objects.
  - As the entirety of the json is an object it must always start and end with a curly brace.
  - Example Json Object: "MyObjKey": { "Name" : "JOHN DOE" , "Age": 21 }
- Null: an empty value.
  - Example Json using null : "nullKey": null

▪ Data can be verified using tools or an online JSON viewer like:

» http://jsonviewer.stack.hu/

JSON Example Data

```
{
        “x”: 3.14,
        “name”: “McK”,
        “isReady”: true,
        “favoriteNumbers”: [ 1, 1, 2, 3, 5 ],
        “visitationCounts”:
        {
              “Los Angeles”: 33,
              “New York”: 12
        }
}
```

417

Expectation Failed

TESTING Requests

- There are many tools available to Developers to test Server API calls while implementing to avoid having to implement a client solution first to test.
  - Postman (Chrome Apps)
  - Rest API Client (Mobile)
  - https://www.hurl.it/ (web)

- Example Workflow:
  - Create RPC on Server
  - Test API request:
    - https://www.hurl.it/
      - http://pokeapi.co/api/v2/pokemon/#/
  - Evaluate RPC/API response Data:
    - http://jsonviewer.stack.hu/
  - Implement Client Side communication with RPC:

507

Insufficient Storage

UnityEngine.Networking

- The unity engine has a built in Networking section of code.
- You can gain access to the architecture by adding the following to your file:

```
using UnityEngine.Networking;
```

- This framework handles many types of networking options.
  - HTTP web requests
  - TCP protocol connections
  - RPC’s
  - Client Server Authority
  - MatchMaking
  - Etc…

599

Network connect timeout error

Web Requests

- Inside the *UnityEngine*.*Networking* is an object called a UnityWebRequest.
  - A UnityWebRequest can be created to communicate to an external web API.
  - Can be used to make GET or POST Requests.

- To Create a UnityWebRequest you can use the following:

```
UnityWebRequest req = new UnityWebRequest(uri);
```

Where uri is the full path for the API to contact.

404

Not Found

Sending Requests

- To then send a request, you can call:

```
yield return req.SendWebRequest();
```

  - This essentially sends the request off until it receives a response.

    - This being a yield return, has a dependency to be called asynchronously through am IEnumator function.

  - You can determine if a request was successful by checking the state of the request after it returns from the yield:

```
req.isNetworkError
```

  - If an error is detected, you can use the req.error string variable to determine more details about the error of the request.

Web Request Headers

- Some API calls will require specific headers be added to your HTTP request.
- You can set individual headers of an HTTP request in unity by calling the following:

```
req.SetRequestHeader(“key", “value");
```

  - Key: the name of the key for the header
  - Value: The value of that key.

- By default Unity will add some headers to your request without you specifying them:
  - content-length: [based on size of body]
  - x-unity-version: [unity version]
  - user-agent: Unity.

Getting Response Data

- In order to process the response from the server returned back to your request, you need to setup a Download Handler.

```
DownloadHandlerBuffer dH = new DownloadHandlerBuffer();
```

- This is a new object which is used to house the data processed from the response of a request.
  - By default each request has an associated variable for a DownloadHandler accessed by request.downloadHandler.
  - By default this is a null object, we need to assign a new DownloadHandler like so:

```
req.downloadHandler = dH;
```

- With a downloadHandler attached to our request, we can access the data of the response available after the call has been processed using:

```
req.downloadHandler.text
```

Unity JSON data

- Unity has a built in JSON package to format data to / from JSON into objects.
- To Serialize an object into JSON data use the following:

```
string JSONString = JsonUtility.ToJson(object myObject);
```

- To deserialize an object from JSON data use the following:

```
object myObject = JsonUtility.FromJson<ObjType>(JSONString);
```

Summary **Learned**

- **HTTP Requests**
  - How they work
  - What they are made of

- **JSON Data**
  - How it works
  - Where it's used?

- **Unity**
  - How HTTP Requests work in Unity
  - How to read JSON data in Unity