Web APIs

Overview

API: Application Programming Interface

- We all use software systems as users.
- What if we want to write a program or script that accesses a software system programmatically?
- APIs allow programmers to write programs to access software systems programmatically.

Web APIs Overview

- Web servers
- Web API servers
- Stateful web API servers
- Scaling up web API servers
- Screen scraping web pages
- Downloading files from web servers

Web Servers

- Simple case
  - User requests a static web page
  - Web server returns a static web page
- Advanced cases
  - Images, audio, video
  - Dynamic content
  - Client-side scripts
  - Client-side scripts which make web API calls

Web API Servers

- In the last bullet point, we saw that web servers support client-side scripts which make web API calls
- We can take that concept and expand it to be any program instead of limited to just web browsers
- Turns out web servers make excellent web API servers

Web API Servers (cont.)

- Program makes a web API call
- Web API server returns response
- Additional notes
  - Most web APIs send and receive data in JSON format.
  - Web servers and web API servers scale up easily by the nature of their design.

Stateful Web API Servers, Part I

- Web servers and web API servers are stateless
- No data is preserved on the web API server between web API calls
- Huge challenge

Stateful Web API Servers, Part II

- Solution
  - Client-side cookie for SID (session ID)
  - Server-side session data by SID
    (sometimes called server-side cookies)

Stateful Web API Servers, Part III

- Sequence
  - User makes login request with credentials such as username and password
  - Server verifies login, creates SID, returns SID to user
  - User now includes the SID with all requests
  - On each request, server retrieves and updates SID data for the user
  - User makes logout request when done
  - Server destroys SID data and notifies user logout was successful

Scaling Up Web API Servers

- Stateless nature of web servers and web API servers allows them to scale up
- Similar to big data scale up
  - Immutable model
  - Shared nothing
  - Etc.

Scaling Up Web API Servers (cont.)

- Simple web API server is broken into many layers
  - Each layer scaled up
  - Layers can be geographically distributed all over the world
  - CDN (content delivery network)
- Central transactional database is needed
  - Weakest link
  - Does not scale up
  - NoSQL can help
  - Immutable model from big data can help

Screen Scraping Web Pages

- What if a web server does not provide us with an API?
  - Programs cannot use the API to pull data
- Solution
  - Screen scraping web pages
- Issues
  - Not authorized, blocking techniques
  - Not exact, subject to high rates of error
  - Formats can and do change at any time without notice

Downloading Files From Web Servers

- Most datasets are stored in web servers
- We need to use web APIs to download them
- We will cover:
  - Text files
    - Encoding issues, UTF-8
  - Binary files: images, audio, video, Excel files, etc.
  - Zip files
    - Downloading
    - Unzipping
    - Encoding issues of interior files

The End

Web Servers

Concepts

Web Servers

- Simple case
  - User requests a static web page
  - Web server returns a static web page
- Advanced cases
  - Images, audio, video
  - Dynamic content
  - Client-side scripts
  - Client-side scripts which make web API calls

URL: Uniform Resource Locator

```
protocol://username:password@hostname:port/directory/file?param1=value1&param2=value2
```

protocol	http – unencrypted https – authenticated, encrypted, uses TLS (formerly known as SSL)
username, password	optional, rarely used, newer authentication methods we will cover later
hostname	required, can use an IP address
port	optional, assumes 80 for http, 443 for https
directory	optional, several layers possible
file	optional, web servers have a default such as index.html
param1, value1, param2, value2	optional parameters, add more with more ampersands

DNS: Domain Name System

- Translates a domain name (aka hostname) to an IP address
- Examples
  - berkeley.edu: 35.163.72.93
  - google.com: 216.58.194.142
  - amazon.com: 205.251.242.103
  - walmart.com: 161.170.230.170

URL Examples

- `https://google.com`
  - Uses https (TLS), assumes port 443, no username, no password, hostname is google.com, assumes directory /, assumes default file, no parameters
- `https://tools.usps.com/zip-code-lookup.htm?citybyzipcode`
  - Uses https (TLS), assumes port 443, no username, no password, hostname is tools.usps.com, assumes directory /, file is zip-code-lookup.htm, parameter is citybyzipcode no value
- `http://75.24.122.15`
  - Uses http, assumes port 80, no username, no password, hostname is the IP address 75.24.122.15, assumes directory /, file is default, no parameters
- `http://75.24.122.15:7000/w205/list.html`
  - Uses http, port 7000, no username, no password, hostname is the IP address 75.24.122.15, directory /w205, file is list.html, no parameters

Static Content

- User requests a file on the web server
- Files are changed at the server level, not at the user request level
- Examples
  - Text: html
  - Formatting: css
  - Images: png, jpg, jpeg, gif
  - Audio: mp3
  - Video: mp4
  - Client-side scripts: javascript
  - Compressed files: zip, gz, 7z
  - Excel files: xlsx

Dynamic Content

- Content that is dynamically generated when a user requests it
- Examples
  - User enters a ZIP code into a form on a web page, web server looks up ZIP code information in a database, displays ZIP code information to user
  - User enters a city into a form on a web page, web server looks up weather information for that city, displays weather information to user
  - User enters a web page for game statistics, web server looks up game statistics, displays game statistics to user

Static vs. Dynamic

- Static content
  - Very low demands in terms of memory and CPU
  - Single thread can serve thousands of user connections
  - CDN (content delivery networks) are easy to scale out by replicating and pushing static content out to edge servers all around the world
- Dynamic content
  - Extremely high demands in terms of memory and CPU
  - Each dynamic request requires a separate thread of execution
  - Cannot use CDN to scale out
    - We will study techniques to scale out dynamic content later
  - Weak link

Client-Side Scripts

- JavaScript
  - Most widely known
  - Foundational layer
  - Additional, higher-level, easier-to-use layers built on top of JavaScript
- Can provide dynamic style content on the client side
- Great alternative to dynamic content on the server side
  - Does not require additional resources (memory and CPU) on the server side
- Can make web API calls
  - We will study later

The End

Web Servers

Business Cases

Familiar Business Cases

We are all familiar with these examples of using web servers:

- Search engines
- Email
- Social media
- Checking a store’s location and hours
- Driving directions, traffic, road closures, etc.
- Purchasing items online for pickup or home delivery
- Scheduling services online
- Online learning
- Etc.

Less Familiar Business Cases

- Web server instead of a desktop app
- GUIs (graphical user interfaces) for CLIs (command line interfaces)
- IoT (Internet of Things)

Web Server Instead of a Desktop App

- Company wants to write a desktop app
- Supporting desktop apps is very difficult and expensive
- Users have various makes and model of laptops, desktops, memory, CPU, OS versions, drivers, etc.
- Web servers only require a web browser, which everyone has
- Desktop apps require an install and update mechanism, which are always issue-prone
- Solution: use a web server instead of a desktop app

Issues With Web Servers Instead of App

Web servers:

- May not be as user-friendly as desktop apps
- Require an internet connection
- May be slower than desktop apps
- Require more infrastructure on the server side than desktop apps

GUIs for CLI

- CLIs such as Linux are often much harder to use than a GUI.
- Desktop GUIs have a huge overhead in terms of memory and CPU.
- Web servers, when scaled for a small number of users, have a much smaller overhead than a desktop GUI.
- Solution: Create a GUI using a small web server.

IoT: Internet of Things

- Many devices have small embedded processors in them.
  - Personal: cars, key fobs, TVs, washers, dryers, refrigerators, dish washers, ovens, garage door openers, HVAC, etc.
  - Robots: industrial, vacuum cleaner, hobby, experimental, etc.
  - Road: construction signs, warning sirens, etc.
- Not practical in some cases to have a display, keyboard, etc.
- Processors are too small for a GUI.
- Solution: Create a GUI using a small web server than can run in the small embedded processor.

The End

Web API Servers

Concepts

API: Application Programming Interface

- We all use software systems as users.
- What if we want to write a program or script that accesses a software system programmatically?
- APIs allow programmers to write programs to access software systems programmatically.

Client-Side Scripts Make Web API Calls

- Web servers support client-side scripts.
  - JavaScript
- Client-side scripts can make web API calls.

Program Instead of Browser

- What if we wrote a computer program to make web API calls to mimic client-side scripts running in a browser making web API calls?
- The program can do anything that any client-side script can do.
- If the API is robust enough, the program could do anything that the web app itself can do.
  - Some APIs are limited to only certain features.
  - Some APIs are comprehensive and can do anything the web app can do.

Going Without a Web App

- We have talked so far about web servers which expose a web API
- Possible to write a web API that was never intended to be used with a web app
- Examples:
  - Phone apps
  - Tables apps
  - IoT apps
  - Desktop apps

HTTP: Hypertext Transfer Protocol

- Protocol used to make web API calls
- On top of TCP/IP
  (transmission control protocol/internet protocol)
- Not to be confused with HTML (hypertext markup language), which is a text format, not a protocol

HTTP Methods

- GET
- HEAD
- POST
- Lesser-used methods
  - PUT
  - DELETE
  - TRACE
  - OPTIONS
  - CONNECT
  - PATCH

HTTP GET Method

- Request a resource
- Resource can be:
  - Text
  - HTML
  - JSON
  - Binary data: images, audio, video, zip file, Excel file, etc.
- Parameters may be passed as key-value pairs
- Most modern web APIs are designed to return JSON unless binary data is requested

HTTP Response Message Format

- Status line
- Headers
- Empty line
- Message body

HTTP Status Line

HTTP version

Status codes				
1XX: Informational	**2XX: Successful**	**3XX: Redirection**	**4XX: Client error**	**5XX: Server error**
	200: OK	301: Moved	400: Bad request	500: Internal error
	201: Created	302: Found	401: Unauthorized	501: Not implemented
	202: Accepted	303: Method	403: Forbidden	502: Overloaded
		304: Not modified	404: Not found	503: Gateway timeout

HTTP Headers

- Key-value pairs
- One header per line
- Key, colon, space, value
- Value can be multivalued
- Empty line lets us know when the headers end
- If no headers (rare), we just get the empty line

HTTP Message Body

- Also called payload
- Content-type header tells us the format of the message body
- Common types
  - JSON: most common in modern era for web API calls
  - HTML: most common for regular web pages
  - CSS (cascading style sheets)
  - JavaScript—most common for client-side scripts
  - gif, jpeg, png, mp3, mp4, etc. for images, audio, video, etc.

Encoding Binary Message Bodies

- Binary message bodies
  - gif, jpeg, png, mp3, mp4, etc. for images, audio, video, etc.
- Binary data needs to be encoded to pass through network software that expects text
- MIME (multipurpose internet mail extensions)
- Encoding such as base64
  - Previously uuencode

HTTP HEAD Method

- Same as GET, but only returns the headers without the message body
- Allows us to see if new content has been updated without having to download the message body

HTTP POST Method

- Same as GET, but allows us to send a message body (payload) to the server
- Originally for submitting web forms, uploading files, etc.
- In web API usage:
  - We typically send JSON
  - We typically receive JSON
  - POST much more commonly used than GET

HTTPS: Hypertext Transfer Protocol Secure

- Encrypted version of HTTP
- Uses TLS (transport layer security)
  - Formerly known as SSL (secure socket layer)
  - Authenticates the web server or web API server we are connecting to using a trusted third-party certificate authority
  - Secure key exchange
  - End-to-end encryption
- Once we have a secure, trusted, encrypted connection, HTTPS uses the HTTP methods in the same manner
- HTTP defaults to port 80, HTTPS defaults to port 443

The End

Web API Servers

Business Cases

Obvious Web APIs

- Phone apps
- Tablet apps
- IoT apps
- Desktop apps
- With or without a web app

Granting Programming Access to Existing Website

- We have an existing website for years.
- Customers keep asking us to programming level API access.
- Find out the most common requests and start building APIs for them.
- Retrofitting an API is a lot harder than designing it from ground up.

New Website

- We are building a new website from the ground up.
- We know some of our users will want programmer API access.
- We need to design with API in mind from the ground up.

Separate Front End From Back End

- Best option is to separate front end from back end
- Back end
  - Design a comprehensive API that does everything
- Front end
  - Our website only uses our API
  - Non-API logic not allowed
- Bonus
  - Website, phone apps, tablet apps, desktop apps can all use the same API

User API Permissions

Should we allow users access to:

- All APIs?
- Subset of APIs?

Back End as a Service

- Suppose we have users who want to write their own custom front end
- Use our back end as a service
- We can build our own websites, phone apps, tablet apps, desktop apps, etc. using APIs to our back end
- Users can build their own custom websites, phone apps, tablet apps, desktop apps, etc. using APIs to our back end

APIs vs. Downloads

- Web API calls can be very resource intensive in terms of CPU, memory, database, etc.
- Consider creating downloads instead
- Allows users to download CSV and/or JSON files of data instead of making numerous individual API calls
- Hybrid
  - Users make an API call that creates a file for download
  - Users then download the file

The End