CS355 Final Project (Fall 2021)

In this project students will choose two API's and create a mashup that uses them **synchronously**. Once the code is completed students will then present their projects in a short recoded screencast walking through their code and demoing the final result.

You will need to complete the HTTP and API playlist and most of assessment 4 to understand how to write code to utilize the HTTP and HTTPS protocols on Node.js. Your project must use the techniques we covered this semester.

Some notable examples from previous semesters includes:

- Nasa API + Google Drive: Connect to a users Google drive and uploads a new photo from Space.
- OpenDota API + Wunderlist: Create Calendar events for all league matches.
- OpenWeatherMap + OpenBreweryDB: Let the user search for brewery's by zip code and display weather information for the surrounding area.

Requirements

A successful project will have at minimum four phases in code execution:

1. When a end user visits the home page, your server send them a form to fill out.
2. When a end user submits the form, use the captured data to send the first API request.
3. Upon receiving the response from the first API request, your server will parse the response and generate a request to the second API.

   You have to be careful here: the user cannot be the driver of this secondary request, if they have to interact with the page at all, (for example clicking a button) this is considered two separate requests and not two synchronous requests. In the GitHub Jobs x Todoist mashup upon receiving a response from GitHub our application immediate contacts Todoist for the next phase.

4. Upon receiving the response from the second API request, your server will parse the response and finally send results back to the end user.

The project requires two API's to be chosen, **one of which MUST have some form of authentication**. The authentication model chosen will determine the maximum score that can be earned. The other chosen API does not need authentication. (You only need to do authentication once)

These two API's **MUST** be queried synchronously without race conditions. **Having two API's operate asynchronously will result in a failing score.**

Deliverables

If any of these are missing no partial credit will be awarded:

1. Project code that satisfies restrictions. Please be careful with this, projects submitted that do not follow the minimum guidelines will be awarded a zero.
2. An HTTP sequence diagram which shows all **HTTP** requests/responses made between client and server. (Be sure to **not** include TCP handshakes and ACKs which should be abstracted away)

   Here is an example of what I want
3. A screencast where students will record them walking through their project and code.

In the past semesters students have submitted code without a recording, those students earned a zero.

Score Ceilings and Authentication Model

There will be score ceilings dependent on the authentication model chosen.

- API Keys are typically the easiest as you get them when registering your application.
- OAuth 2.0 Client Credential requires your application to **dynamically** trade Client ID and Client secret for an access token, before being able to interact with the API. The access token should not be hard coded into your program. (You are encouraged to cache it after it's been run once though)
- A three-legged OAuth request is the most complex as it involves explicitly getting permissions from a third entity, the end user.

Each step increases project complexity, but there is a larger reward for those willing to take a risk with a more difficult work flow. (To gauge the difficulty the Spotify API in assessment 4 used Client Credential authorization workflow)

Authentication	Maximum Score
API Key	90%
OAuth 2.0 Client Credential	100%
OAuth 2.0 Three Legged	110%

This score ceiling is based on the most difficult API you choose:

- If you choose to do a mashup between an OAuth Three Legged and a public API you will have a ceiling of 110%.
- If you choose two Client Credential APIs you will have a ceiling of 100%.
- If you choose an API key and a public API your ceiling will be 90%
- A project with two public API's will not be accepted.

List of API's

https://github.com/public-apis/public-apis

You should avoid OAuth 1.0 API's as we did not cover certificate signing this semester.

You are not restricted to just API's on this list.

Restrictions

The point of this project is to demonstrate your understanding of low-level Node.js. Specific tools and libraries that

Third party modules are banned (eg. NPM / github)

This project tasks you with writing an HTTP server to communicate with various API's. Some API's offer helper libraries to automate the authentication process. You **cannot** use these libraries. (You can still use those API's, but you must do so using **only HTTP / HTTPS** Node.js modules)

Some popular libraries that are **banned** from this assignment

- request https://www.npmjs.com/package/request
- express https://www.npmjs.com/package/express

How to avoid accidentally used a third party module

The only libraries you are allowed to use are built in libraries. **I need to emphasize this because of the number of students that received zeros in previous semesters.**

If the library is available on **nodejs.org** ✅ you can use it (fs, http, https, querystring, url, ...). If it's anywhere else including **npmjs.com** ❌ or **github.com** ❌ it's barred from this project. Specifically the only libraries you should be using are the one's found in the sidebar on the nodejs documentation page.

- If an API provides a Node.js or JavaScript helper "library" you should **not** use that, instead look for API's that offer a REST or cURL interface. These are typically language agnostic and easy to test with Insomnia Core and then you can use `https.request()` or `https.get()` to interact with. These will be

the two main ways you will be pulling (or pushing) API data.
- If the instructions on an API tell you to run the command `npm install` at any point, it is directing you to install 3rd party module.
- Review the `require` statements on top of your code, there should only be built in libraries and one's you wrote yourself.
- If you find at any point in your project you have a `node_modules` directory that is required for your program to function, you are using an external library (and should rewrite your code to remove those).

Promise and Async/Away notation is banned

We have covered callback and observer pattern this semester, your solution must demonstrate understanding of these techniques.

setTimeout and equivalents are banned.

The code may look like it works initially, but it has race conditions, doesn't scale, and is always going to be slower in the scenario where it does work.

API's must be different than demos

As this is meant to be an independent research project, API's used in current and future lecture videos and in previous assignments are not allowed. This list includes four API's:

- https://jobs.github.com/api
- https://developer.usajobs.gov/
- https://developer.todoist.com/
- https://developer.spotify.com/
    - Exception: Three legged OAuth endpoints are allowed for Spotify API as we only used Client Credential.

Unrelated API's are okay

If you are having difficulty finding two related API's, it is fine to use two unrelated API's. In this scenario, students are *still* required to write their code in a synchronous fashion, and will instead *pretend* the first API request needs to be responded to, before the second can be sent. (Eg. I need to get the restaurant information, before I can request the weather data)

Phase 1: Choosing a pair of APIs

First check this list to see which pairing of API's your fellow classmates have chosen (this will be updated daily) **This document is set to read only - You will not be editing this document**

https://docs.google.com/spreadsheets/d/1rgAO0ssd4ZanSc_nN9aZ4mExy8JYazdeQ4h-369fmA8/edit

You cannot use the exact same pair as someone else, but you can change one of them to a different API and it will be considered a different pairing.

Don't rush to lock in your API without thoroughly testing it. The public API list currently has 773 different API's meaning there are ${}_{773}C_2$ = 298,378 possible pairings. It's very unlikely for two students choose the exact same pair of APIs at random.

There's little need to rush this process and it can be harmful if you lock yourself into an API that you later find incredibly difficult to use.

During this phase you should be looking through and setting up accounts with various API's to find a pair that you want to create your project with. Make sure to try using each API before deciding on your API, as some API's are much harder to use than others. Use an HTTP debugger like **Insomnia** or Postman to try to authenticate and make queries.

I would recommend first looking into API's that interest you, having contextual knowledge about a particular hobby can make working with the API much easier. (E.g. A Hearthstone player would understand

the intricate details of the Hearthstone API better than non-players)

During this phase you should:

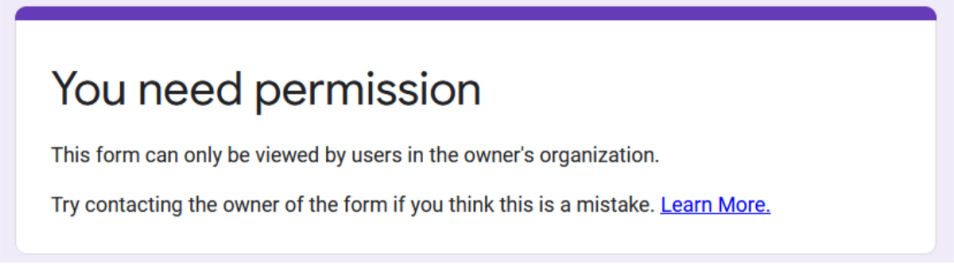
- Get the required API keys or Client ID / Client Secret pairs.
- Draw a draft of a sequence diagram that follows how a typical user would use your server and how your server interacts with 3rd party APIs. *(Not submitted yet)*
- For each network request to a third party (outbound arrow) in the sequence diagram, try to replicate them with an HTTP debugger (Insomnia or Postman).
- Ensure that you can correctly authenticate and access all endpoints.

My assistance will be limited for this project, in contrast to the Spotify API, I do not have intimate knowledge of your chosen APIs, if you find that you are having trouble using the API, **don't choose it!**

Claim QC GSuite for Education

All submissions must be made with a Queens College supplied G Suite for Education account. **Personal Google accounts will not be have access to the submission form.** The link below will walk students through the process of setting up an account.

http://ctl.qc.cuny.edu/claim-qc-google-apps-account/



Student's who receive a permission error are not signed in to the correct account. The link below should help with switching accounts.

https://accounts.google.com/AccountChooser

Submit Choices

Once testing is completed you will submit your API choice here:

https://docs.google.com/forms/d/e/1FAIpQLSdOm_XWv6TOR8FzvY648sSXQaQuIupKls2abr6YxQVqkoox1A/viewform?usp=sf_link

Within 24 hours of submitting the form (likely much sooner) , you will receive an email notifying you of your choices either be approved or rejected. API pair choice is assigned on a first come first serve.

In the case where someone submits the exact same pair of APIs as you, you will need to change one or more of the API's and resubmit.

Once approved you may move to the next phase.

If you want to change your API choices after they have been approved just submit the form again and wait 24 hours.

Phase 2: Creating the Project

First make a formalized HTTP Sequence Diagram describing a (non-cached) use of your application including all calls to each API

- https://sequencediagram.org/
- There should be 4 entities in your diagram: the user, your server, and the two APIs.

This will organize your thought so you can keep track of various endpoints and decide at any point if you are sending or receiving data.

After the diagram has been formalized, build the project.

Additional factors that will effect your grade:

- Is your application resilient to unexpected input? (It should deliver 404 in most of these cases)
- Do you utilize caching?

  If you have any cache files delete them before submitting. (Make sure you program is not reliant on the cache, test this by deleting the cache and running the program again)
- Sequence Diagram
  - Are all Requests included (No missing HTTP requests on diagram)
    (Ignore TCP segments as they are abstracted away)
  - Headers / Query / Post Data included where required?

If you have multiple similar requests, like when downloading multiple images, a single arrow with a remark is sufficient

Important: testing your project to ensure it is operating synchronously

Your project **MUST operate synchronously**.

If we have Requests A and B, **Request A must finish (received response)** before Request B begins, otherwise you risk a failing grade for not hitting requirements.

If B's request is dependent on A's response, (GithubJobs x Todoist example where I can't create calendar events without the Github search results) then this is obvious and you can skip this step.

But if the requests are independent, this becomes harder to test.

You can use the following snippet of code to test if your calls are being done synchronously.

```
const api_request_a = https.request(url, options, callback);
setTimeout( () => api_request_a.end(...) , 5000);
```

Specifically change your call to `.end()` with a version that has a delay. (Note we are only using this call to `setTimeout` to test our code, you should revert back to the original version before submitting. Don't use `setTimeout()` for anything other than testing)

Add a print statement each time one of your API is called. (API 1 has been called, API 2 has been called)

If **without any additional changes** your project takes an additional 5 seconds before it can display the exact same results, then it is likely fine, operating synchronously (as the second API is waiting for the results from the first)

If the code blows through the delay and makes the call to the second API without waiting for the first to finish (the message API 2 has been called should appear), then you are operating asynchronously and you need to make changes. Synchronous behavior means that if there is a delay for any reason, the second task will wait. Your code must work if I change that timeout to 10 seconds, 60 seconds, or 60 years.

My code is operating asynchronously, how do I make it synchronous?

Refer to the N Domains Synchronously topic of lecture 6.

https://venus.cs.qc.cuny.edu/~rlaw/cs355/lectures/06-asynchronous-programming/asynchronous-programming.html#n-domains-synchronously-recursion

Phase 3: Presenting your mashup with a screencast

Finally students will record a screencast presenting their project similar to my lecture videos. (Record your screen while explaining how your code works)

A screencast will consist of:

1. Description of the project:

- Which API's are used, which endpoints are used? What data do you give them (if any)? What data do you get back (if any)?
- Which one is called first, which one is called second? (synchronous requirement)

2. Going through the sequence diagram

- What requests are made between the client and your server, your server and the API, (and if using 3-Legged OAuth between the client and the API)

3. Going through the code:

- Go through the process of how the user gets the landing page form.
- What information does the user provide (if any)?
- Where do you catch that request and submitted information?
- Go through the process of how you process the user input
- Go through the process of generating and sending the first request
- Go through the process of capturing the first response.
- Repeat for 2nd request/response
- Go through the process of generating the response you will send to the user.

4. Run the program

- Make sure your server is resilient to various inputs
- Make sure your server can facilitate multiple requests (without the need to restart the server)

5. Answer the following questions:

- How do you guarantee that the first API is received before the second one is requested? (synchronous requirement)
- What did you cache? How long is each resource cached for?
  Can you engineer a scenario where your application might serve might serve cached data that is different than what the true results would be? (stale cache scenario, example: in assessment 4 if the album art changes on Spotify's end, the finished code continues to use the old image forever) What changes can be implemented to minimize this scenario?

Screencast Length

15 minutes is ideal. Try to keep your screencast under 20 minutes, this is not a requirement (I've received much longer screencasts 1hr+, this is also fine)

There is no minimum time requirement, if you think you can hit all the requirements quickly feel free to submit a shorter screencast - but note for shorter screencasts I will be very critical of things missing.

Don't worry too much about the quality of the recording as long as the bullet points are hit and you demonstrate understanding of your code, the video does not need to be professionally recorded and edited.

How to Record a Screencast

There are no specific requirements, you can use any screen recording software that you want including:

- Open Broadcast Software (OBS)
  - https://obsproject.com/
- YouTube Live streaming
  - Guide: https://outline.com/4Gybf6

Hosting:

Either upload the video file to the form below or upload an unlisted recording to YouTube and include the link in your submission.

I must be able to watch your video and read your code.

- If it's on YouTube keep the video accessible until you receive your Final Grade.

If you are concerned, after submission, you can send me an email and I will confirm receipt of files and quickly test to make sure everything is viewable on my end (I will not grade them, only test the integrity of the files to make sure nothing is corrupt)

Additional Notes

To earn a grade everything must be submitted (code, sequence diagram, and screencast)
No partial credit will be given to students stuck in earlier phases.

Be careful of scope creep, keep the project simple. **The usefulness of your project is not a factor in it's grading.** Your code will be graded on your programs functionality, reliability, caching, and whether all the requirements are hit.

All code including submissions from previous semesters are passed through Stanford Moss to check for plagiarism.

Submission

Review the code:

- Remember third party libraries are not allowed, your project should **not** have a `node_modules` directory.
- Delete any cache files before submitting. Make sure your program is not dependent on it's cache. (In assessment 4 the cache files would be the authentication-cache and the album art images)
- Optionally you can delete your Client ID / Secret / API Key in your authentication file, I will create an account with each of the APIs, should I need to run your code.

Save all files and sequence diagram into a directory `CS355-FP-########` replacing `#` with your Student ID

Zip the folder, then submit here:

https://docs.google.com/forms/d/e/1FAIpQLSeLYyAIeYAIoeFJYfxxgEuPQbwCL2hDJEj4gSSPU0yqxVIoVA/viewform?usp=sf_link

Due Date

- Monday, December 20th 23:59:59
- The form is scheduled to close automatically at this date and uploading may take a bit of time depending on your connection speed so have it submitted a bit before the deadline.