**IFT458 – Topic5**

**Project Component:** Git, JSON, and API

Introduction

This phase of the project introduces you to 3 important concepts in web development: API, JSON, and Git. Please review the PPT slides and go through the reading materials before starting this activity.

There are 2 parts to this deliverable:

In part 1, you’ll apply API & JSON to obtain the stock price of a company of your choice.

In part 2, you’ll use Git to create a version control for your project.

Part I: API & JSON

Problem Definition

Suppose you want to find out the current stock price of a company. There are a number of sites that you can go to in order to get the price of a stock. One very useful one is the site of the NASDAQ stock exchange. You can use any browser and enter the following URL to get the current price of Apple stock

To get a company stock price, You will need:

1. The stock symbol

If you don’t know a company’s stock symbol, you can find it by using a search engine and entering “stock symbol for xxx,” where xxx is the name of the company. For example, let’s say you wanted to find the price of Apple stock. By typing **stock symbol for Apple** in a search engine, you’ll have **AAPL**. To get the stock price for Apple whose symbol is AAPL using the NASDAQ stock exchange, type in a browser: <https://www.nasdaq.com/symbol/aapl>

When you press Enter or Return, the browser sends the request. The NASDAQ server receives the request and looks up current information about Apple stock. It then builds the appropriate HTML code to send back to the browser. The browser receives all of the returned HTML as a single string, formats the information, and displays the page on the screen. You see the current stock price for Apple near the top of the page. You also see the change and percent change of the stock price, some key stock data (such as the best bid/ask price, 1-year target, today’s high value, and many more), along with an interactive chart of previous prices, and even some recent news about Apple.

In Python, instead of the browser making the request to get information from the NASDAQ site (or other similar site), we can write a Python program that can make the same request. The response—all the HTML text—will come back to our program, and we can save all that text into a variable. But instead of painting the entire page like the browser does, we’ll just look for the specific information we want to find: the price of a stock.

Python has a module called urllib and a submodule called request that provide the code needed to allow programs to make requests over the Internet. You bring this package into your code with this line:

import urllib.request

Once you have done that, the following two lines can be used to make the request across the Internet:

connection = urllib.request.urlopen()

responseString = connection.read().decode()

Where < URL> is the URL you want to connect to.

For example, to get stock-quote information for Apple , we can use this code:

nasdaqAppleURL = 'https://www.nasdaq.com/symbol/aapl'

connection = urllib.request.urlopen(nasdaqAppleURL)

responseString = connection.read().decode()

When these lines run, the variable responseString is set to the exact same underlying HTML information that the browser gets for that page when it makes the request. We know that the stock price is included somewhere in the returned text because when the browser makes the request and gets the response, we see the stock price on the screen. If we were to spend a lot of time analyzing the HTML we got back as a response, we could find an identifying tag that precedes the actual price string.

1. Base URL

Companies that want to share their data make it available to computer programmers by publishing a set of guidelines for retrieving this type of data in a much more efficient way. These guidelines are called an API.

An API (Application Programming Interface) is a set of URLs and parameters that is designed to be called by programs across the Internet.

The idea is that instead of receiving a full page of HTML designed for display, using an API programmers can ask for and get just the information they want and use it in their programs.

For example, there is a company named **Alpha Vantage** that has a set of APIs for retrieving data about stocks, physical and crypto currencies, stock indicators, and sector performances. It is fully documented at [www.alphavantage.co/documentation/](http://www.alphavantage.co/documentation/)

To get a stock quote, there is a single URL with a number of different parameters you can specify. The base URL is www.alphavantage.com/query . But in order to make a useful request, you must also specify a number of additional values.

1. Request with values

If we think of the request/response model being like a function call, it would make the model more complete if we could pass data with a request, just as we pass values when we make a call to a Python function. It turns out that you can do exactly that. However, passing data with a URL has a different syntax than the way we do it in Python. The reason for this difference is that requests over the Internet are independent of the programming language. When passing data with a URL, there has to be a very general syntax. That syntax looks like this:

http://<URL>?<parameterName1>=<value1>&<parameterName2>=<value2> etc.

At the end of the base URL, you add a question mark to indicate that more information is coming. Following the question mark, you build any number of sequences of the form parameterName=value (with no spaces). Each grouping is commonly known as a name/value pair .

Now let’s build the full URL to get a stock quote using the Alpha Vantage API. According to its documentation, to get a stock quote we need to specify three pieces of information:

* function: for the stock, ?function=BATCH\_STOCK\_QUOTES
* symbol: For Apple, &symbols=AAPL
* The last piece of information is an identifier called an API key

1. API Key

Companies (and government agencies) that provide data via APIs often provide the data free to programmers who use their extensive APIs. But in order to prevent overuse and/or potential malicious intent, organizations often require that you obtain an identifying key from them.

An API key is a unique string that identifies you as the person whose code is making a request. API keys are typically given out for free by filling out an online form. Once you get an API key, you need use it in all your API queries to that company. If you expect to build a commercial program, or you expect to make extensive uses of their APIs, you may have to pay a fee to the company for the use of the API key.

The process of obtaining an API key from a company is typically very easy.

For the company **Alpha Vantage,** an API key can be obtained at this site: [www.alphavantage.co/support/#api-key](http://www.alphavantage.co/support/#api-key)

If you fill out the form and press the GET FREE API KEY button, you should soon get an e-mail back that includes a string of about 16 characters. This is your API key. You can use that key to make API requests to Alpha Vantage.

Now add the following to complete the full URL: &apikey=<yourAPIKey>

1. Full URL

Let’s put it all together. In order to get the stock proce of Apple, you would build up this URL :

https://www.alphavantage.co/query?function=BATCH\_STOCK\_QUOTES&symbols=AAPL&apikey=xxxxxxx The xxxxxx is the API key you received from AlphaVantage.

<https://github.com/RomelTorres/alpha_vantage/issues/184>

1. Response

If we make a call using the API just discussed, with a valid API key, the answer comes back as a single string that looks like this:

{

"Meta Data": {

"1. Information": "Batch Stock Market Quotes",

"2. Notes": "IEX Real-Time Price provided for free by IEX (<https://iextrading.com/developer/)>.",

"3. Time Zone": "US/Eastern"

},

"Stock Quotes": [

{

"1. symbol": "AAPL",

"2. price": "177.8500",

"3. volume": "20536464",

"4. timestamp": "2018-04-18 16:30:20"

}

]

}

API & JSON Deliverable

Now that we have all the pieces, you need to build a Python program, called ***japi.py*** to get the price of a stock using an API. Complete the tasks below:

1. Write a function called getStockData() that takes a stock’s symbol as parameter and return the JSON-formatted response-string
2. Write the main function that :

* infinitely asks the user for a stock symbol until the user enters ***quit***.
* Pass that symbol to the getStockData function.
* Print the JSON-formatted response on the screen
* Convert the response from JSON to Python dictionary
* Print the price ONLY in the form:

***The current price of <stock symbol> is: <Stock Price>***

1. Make a call to the main() function
2. Run the program with at least 5 symbols and save the output into the file ***japi.out***

Part II: Git

Problem Definition

Coding today is no longer a check-in/check-out business. Rather, it's more of a social effort. Several developers collaborate on a piece of code through tools such as Git and Mercurial, and the result is code that is fathered by many different hands.

Git and Mercurial are probably the distributed revision control systems that are most used today. They are essential tools designed to help teams of developers collaborate on the same software.

For this class, Git has been chosen as the version control and collaboration tool. From the next phase, your development will go through the following steps:

* Pull latest files from a central Git repository
* Run all tests locally
* Commit all local changes to Git
* Push to a remote central Git repository

In this phase, we focus on setting up our Git environment in a **Linux/Unix command-line** environment.

Git Activities

1. **Installing Git**

This section will guide you through the steps for installing Git on your system. If you already have Git installed, you can skip to the next section. Otherwise, using the PDF document progit.pdf (pages 22-25), select the appropriate section for your operating system, and follow the instructions

1. **Creating and Exploring a Git Repository**

In this section, you’ll use the script command to capture ALL the 48 activities into a file called **git\_activities** for submission. To do so, type BEFORE starting step 1):

$ *script git\_activities*

1. On your local disk, create a new directory called **japigit** and change (cd) into it. (This will be the directory you work in unless otherwise specified.)
2. Initialize a new repository by running the following command:

git init

This command creates a new git repository skeleton in a subdirectory named .git under the current directory—as indicated by the output message from the command. This means that you’re now able to start using other Git commands in the current directory.

1. Tell Git who you are by setting your basic identification configuration settings with the following commands, substituting in your name and email address as the values for the configuration. (Note the double dashes preceding global as you are spelling out the option. Also, values only require quotes if they contain a space.)

$ git config --global user.name "First-name Last-name"

$ git config --global user.email [emailAddress@asu.edu](mailto:emailAddress@asu.edu)

1. Now let’s get some content to put through the Git workflow. Move the files ***japi.py*** and ***japi.out*** created in part 1 here.

1. Stage the files with the add command. (If you prefer, you can add each separate file explicitly rather than using the “.”)

$ git add .

1. Commit the files using whatever comment you want.

$ git commit -m "comment string"

1. Notice the output you get. There is the branch name—the default branch—master, followed by an indicator that this is the first (root) commit and then the first few characters of the SHA1 for the commit.
2. Edit the python file (japi.py) by making it print the message “Stock Quotes retrieved successfully!” as the last line of the output.
3. Stage and commit the file with the shortcut, using whatever text you want for the commit message.

$ git commit –am "comment string"

1. Run the status command or the short form to see how it looks when you have no changes to be staged or committed.

$ git status

$ git status –s

1. Create a new file and view the status.

$ echo “hello there” > hello.txt

$ git status

$ git status –s

Question:

Is the file tracked or untracked?

Answer:

It’s untracked—you haven’t added the initial version to Git yet.

1. Stage the file and check its status.

$ git add . (or git add hello.txt)

$ git status (git status –s if you want)

Questions:

Is the file tracked or untracked?

What does Changes to be committed mean?

Answers:

The file is now tracked—you’ve added the initial version to Git.

Changes to be committed implies that files exist in the staging area and the next step for them is to be committed into the local repository.

1. Edit the file *hello.txt* in your working directory and check the status.

$ echo “We are testing git status” >> hello.txt

$ git status

Questions:

1. Why do you see the file listed twice?
2. Where is the version that’s listed as Changes to be committed (in the working directory, staging area, or local repository)?
3. Where is the version that’s listed as Changes not staged for commit (in the working directory, staging area, or local repository)?

Answers:

1. You see the file listed twice because there is one version of the same file in the working directory and another version in the staging area.
2. The version that’s listed as Changes to be committed is in the staging area. The phrase implies that this version’s next step or next level for promotion is to the local repository using a commit.
3. The version that’s listed as Changes not staged for commit is in the working directory. The phrase implies that this version’s next step or next level for promotion is to the staging area, because it’s currently not staged.
4. Do a diff between the version in the working directory and the version in the staging area.

$ git diff

1. Go ahead and commit and do another status check.

$ git commit –m "comment"

$ git status

Question:

Which version did you commit: the one in the staging area or the one in the working directory? (Hint: Which one is left [shows up in the status]? Note the Changes not staged for commit part of the status message.)

Answer:

The version in the staging area was the one committed. The content goes through the staging area and then into the local repository.

1. Stage the modified file you have in your working directory and do a status check.

$ git add .

$ git status

1. Look at the history you have so far in your small repository. To do this, run the log command. (In some terminals, your history may be longer than the screen and so you will need to press a key to continue. If you are paging through the log output on a Unix terminal and want to end the listing, press the q key.)

$ git log

$ git status –s

1. Often when looking at Git history information, users only want to see the first line of each entry, the subject line. This is why it is important to make that first line meaningful when using Git.

To see only the first line of each log message, you can use the --oneline option:

$ git log –oneline

1. Use the git branch command to look at what branches you currently have.

$ git branch

You see a line that says “\* master”. This indicates that there is currently only one branch in your repository: master. The asterisk (\*) next to it indicates that it is the current branch (the one you've switched to and are currently working in). If your terminal prompt is configured to show the current branch, it also says “master”.

1. Before you work with a new branch, you need to update the files in the master branch to indicate that these are the versions on master so it will be easier to see which version you have later. To do this, you can use a similar version of the same way you have been creating and updating other files. Run the following command for each file.

$ echo "master version" >> hello.txt

1. Stage and commit the updated files. Because these are files that Git already knows about, you can use the following shortcut command:

$ git commit -am "master version"

1. You have a new feature to work on, so you now create a feature branch with the name feature. Run the following command:

$ git branch feature

Notice that this command creates the branch, but does not switch to it.

1. You can now check what branches you have and which is your current branch.

$ git branch

1. You can now see your new branch listed. Change into the feature branch to do some work:

$ git checkout feature

1. To verify that you're on the feature branch, run the following command, and observe that the asterisk (\*) is next to that branch:

$ git branch

1. create a new file and then update the files in the feature branch to indicate that they are the feature branch version.

$ echo "new file" > hello\_feature

$ echo "feature version" >> hello.txt

1. When you're done, stage and commit your changes.

$ git add .

$ git commit -m "feature version"

1. switch back to the master branch.

$ git checkout master

1. Verify that you're on the correct branch.

$ git branch

1. Create a new one-line file.

$ echo "Initial content" > merge.txt

1. Stage and commit the file on the master branch.

$ git add .

$ git commit -m "adding new file on master"

1. Create a new branch, but don't switch to it yet. (You can use whatever branch name you want.)

$ git branch newbranch

1. Change the same line in the new file (still on the master branch).

$ echo "Update on master" > merge.txt

1. Stage and commit that change (still on the master branch).

$ git add .

$ git commit -m "update on master"

1. Switch to your new branch.

$ git checkout newbranch

1. On the new branch, make a change to the same line of the same file.

$ echo "Update on newbranch" > merge.txt

1. Stage and commit the file with the change on the new branch.

$ git commit -am "update on newbranch"

1. Switch back to the master branch

$ git checkout master

1. Merge your new branch back into the master branch. (Git attempts to merge the new branch into the master branch.) You will end up with a merge conflict after this.

$ git merge newbranch

1. Check the status of your files in Git. Note the information that Git provides to you about the conflict.

$ git status

1. Look at the local file and note the conflict markers

$ cat merge.txt

1. Resolve the conflict in the file in the working directory. (For simplicity, you can just write over it to simulate that the conflict has been resolved.)

$ echo "merged version" > merge.txt

1. Stage and commit the fixed file. Note that this has to be done as two separate steps since this was the resolution to a merge conflict.

$ git add .

$ git commit -m "Fixed conflicts"

1. Check the status to make sure the merge issue is resolved.

$ git status

1. You're done with your new branch, so delete the branch.

$ git branch -d newbranch

1. To remove a commited file, use the rm command

$ echo “file to delete” > to\_delete.txt

$ git add .

$ git commit -m "yet another lab file"

$ git rm to\_delete.txt

$ git status

Run the ls command to find out whether the local file is still there.

$ ls

1. If You now change your mind, and you want the file back. Use the reset command to do that.

$ git reset --hard HEAD

$ git status

$ ls

1. Delete and commit to the deletion this time.

$ git rm to\_delete.txt

$ git status –sb # Check the status; the file should be staged for deletion.

$ git commit -m "<file deletion>"

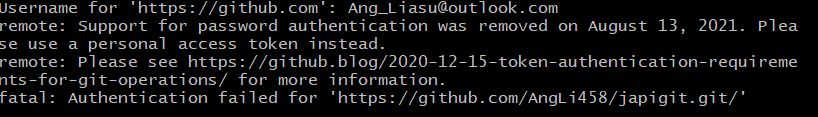
Exit the script capture

1. **Setting Up a GitHub Account and pushing to a Repository**

In this section, you'll get some practice with remotes by setting up a GitHub account, forking a repository, and cloning it down to y our system to work with.

Steps

1. Go to <https://github.com>.
2. Fill in the Pick a username, Your email address, and Create a password fields.
3. Click the Sign up for GitHub button.
4. Accept the defaults on the next screen and click the Continue button.
5. Follow the instructions to verify your email address. Then click the Start a project button.
6. Start a new project by clicking on “New” repository then provide a project name.. Let’s call it ***japigit\_<TeamID>.*** <TeamID> is basically your Group number. Note that the URL to your project will be [https://github.com/<user>/japigit\_<TeamID](https://github.com/%3cuser%3e/japigit_%3cTeamID)>.
7. Add your teammate and your instructor (username: kuitche)
8. Push the files japi.py and japi.out to the remote repository



Deliverables

You are to submit the following items:

1. The files japi.py and japi.out from Part I
2. The file git\_activities from Part II-B
3. A max of 1-page project report that includes

* An Introduction
* A description of what you achieved, what was learned, the challenges faced, and how they were overcome, and how can your work be improved
* Your GitHub URL. Note that the instructor should be a collaborator to the project japigit\_<TeamID>.