Installing C++ REST SDK

For this assignment, we will be using Microsoft's C++ REST SDK, which is on GitHub at https://github.com/Microsoft/cpprestsdk. You will need to install it first, as outlined below.

Windows

1. You'll need at least 5GB of free space on your C: drive
2. Download the vcpkg zip from: https://github.com/Microsoft/vcpkg/archive/master.zip
3. Extract the zip
4. You should have a vcpkg-master folder and inside this you should see a few files, and some additional folders like docs, ports, and so on.
5. Move this vcpkg-master folder to the root of your C: drive
6. Run Windows PowerShell and execute the following commands (one at a time):
   ```
   cd C:\vcpkg-master
   .\bootstrap-vcpkg.bat
   .\vcpkg install cpprestsdk cpprestsdk:x64-windows
   ```
7. The last command will take anywhere from 20 minutes to an hour because it builds several dependencies
8. After finishing, you may notice that the C:\vcpkg-master directory now takes up 5GB. Most of this is the C:\vcpkg-master\buildtrees directory, which you can safely delete if you want to save ~4GB of space.
9. You should now be able to open your pa7 folder in Visual Studio as normal

Mac

From Terminal, run:

```
brew install cpprestsdk openssl
```

Note that for CMake to detect OpenSSL, you need to set the `OPENSSL_ROOT_DIR` environment variable to the OpenSSL directory. The directory OpenSSL will depend where brew installed OpenSSL. For example if brew says the binaries are in `/usr/local/opt/openssl@3/bin`, you would want to run this cmake command (omit the `/bin` part):

```
OPENSSL_ROOT_DIR=/usr/local/opt/openssl@3 cmake -G Xcode -B build
```

Linux

From a terminal, run:

```
sudo apt-get install libcpprest-dev
```

After this, you should be able to cmake as normal.

Installing Postman

You also should install Postman from (https://www.getpostman.com/). This makes testing the API easy. In your repo, under the `postman` directory, there is a collection of tests that you can import into Postman to test your server, once it's up and running.

What to Do

For this final assignment, there are no detailed instructions, to make this more like the "real world." This is like an actual programming test certain companies use to screen applicants.

You should read the rest of this document first before working on the assignment.

You need to create a REST server that serves information about movies on IMDB. The source data is described here: https://www.imdb.com/interfaces/. We are specifically interested in the `title.basics.tsv.gz` file. You'll need to download and extract this file locally. *Do not commit this file to git as it's 600+ MB in size*.

For the target, select the "**main**" target, not "tests". The tests one won't do anything meaningful on his assignment.

Your server takes a command line argument for the name of the title.basics.tsv file (this is so the file location works locally and on Travis).

When loading in the source data, you can ignore any titles that are not titleType "movie".

Your server should run on localhost port 12345.

Your server needs to support two API endpoints using the GET protocol.

GET `/movie/id/{titleId}`
Returns a JSON object with information about the title with specified `{titleId}`. For example:
`http://localhost:12345/movie/id/tt0076759`

Should reply with the following JSON object and a 200 (OK) status code:

```
{
    "ID": "tt0076759",
    "Runtime": 121,
    "Title": "Star Wars",
    "URL": "https://www.imdb.com/title/tt0076759/",
    "Year": 1977
}
```

Note that the URL points to the title on the actual IMDB website.

If a movie with that ID does not exist, you should reply with the following JSON object and a 404 (Not Found) status code:

```
{
    "Error": "No movie with that ID found"
}
```

GET `/movie/name/{movieName}`
Returns a JSON array with information every movie that exactly matches `{movieName}`. For example:
`http://localhost:12345/movie/name/Top%20Gun`

Should reply with the following JSON array and a 200 (OK) status code, as there are two movies with the name "Top Gun":

```
[

    {
        "ID": "tt0048734",
        "Runtime": 73,
        "Title": "Top Gun",
        "URL": "https://www.imdb.com/title/tt0048734/",
        "Year": 1955
    },
    {
        "ID": "tt0092099",
        "Runtime": 110,
        "Title": "Top Gun",
        "URL": "https://www.imdb.com/title/tt0092099/",
        "Year": 1986
    }
]
```

Note that you need only support exact matches.

If a movie with that name does not exist, you should reply with the following JSON object and a 404 (Not found) status code:

```
{
    "Error": "No movie by that name found"
}
```

Timing Requirements
Your server must fully load and be ready to accept requests on localhost:12345 within 10 seconds in release mode on Travis. If it takes longer than 10 seconds, no tests will run.

To pass each test, in addition to returning the correct JSON, the query must also complete in less than 200ms.

Note that it is possible that Travis will run more quickly than your local machine. If you want to test on your local machine, you will need to make sure you run in "Release" mode. Review the instructions for the password cracking and such assignments if you don't remember how to.

Hint
C++ REST SDK has functions/classes to handle both the HTTP server and the JSON output. Use them.

Grading Rubric

Part	**Points**
ID Tests (10x tests, 4 points each)	40
Name Tests (5x tests, 10 points each)	50
Code Quality	10
Total	**100**

Notes

- For this PA, there is no direct “grade” shown on Travis. You need to look at the number of passed tests at the end of the Travis log.
- If your server cannot load in less than 10s in release mode on Travis, you will receive a grade of 0.
- If you try to beat the 10s timing requirement by, for example, loading only a very small subset of the movies, you will receive a grade of 0. You need to load every title with titleType “movie” and support queries to any of them.