Kevin Riggs Module 2 Assignment WriteUp

Requirements:

Define a class that can accumulate information about a sequence of numbers and

calculate its average and Standard Deviation (STD).

For testing purposes, use the class to calculate the average and Standard Deviation of the

sequence of number that you submitted to the first discussion.

The class should be usable by any code that needs to accumulate statistics on a sequence

of values.

Design:

I want this class to be able to accept doubles as an input.

I want this class to provide the average and the standard deviation of the inputted values.

One decision I made was to prevent the user from directly accessing the doubles once inserted. The values can be modified, but only through a helper.

The second decision I made was how to calculate average and standard deviation. One option I thought about to maximize efficiency was to use a running average and standard deviation. Every time a new value is added or changed, the average/standard deviation is updated, not recalculated. Then the value can be quickly accessed. I landed on the easy approach, which actually can be more efficient in certain scenarios. My approach is to calculate the statistics using the current values when asked.

I did also use a private variable itemCount, which makes things easier when calculating the statistics and using the class.

Test Design:

I took a semi-functional test approach on this one. I have a test for both the average and the standard deviation. Both test the ability to add a value and verify that the statistic returned is correct. Additionally, I have a test that validates the test framework for my own sake.

Testing Output:

[==========] Running 3 tests from 2 test suites.

[----------] Global test environment set-up.

[----------] 1 test from newgt

[ RUN ] newgt.newgtc

[ OK ] newgt.newgtc (0 ms)

[----------] 1 test from newgt (0 ms total)

[----------] 2 tests from statistics

[ RUN ] statistics.average

[ OK ] statistics.average (0 ms)

[ RUN ] statistics.stddev

[ OK ] statistics.stddev (0 ms)

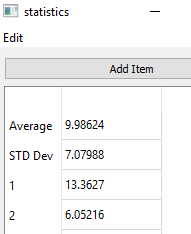
[----------] 2 tests from statistics (0 ms total)

[----------] Global test environment tear-down

[==========] 3 tests from 2 test suites ran. (0 ms total)

[ PASSED ] 3 tests.

Test on my data set:

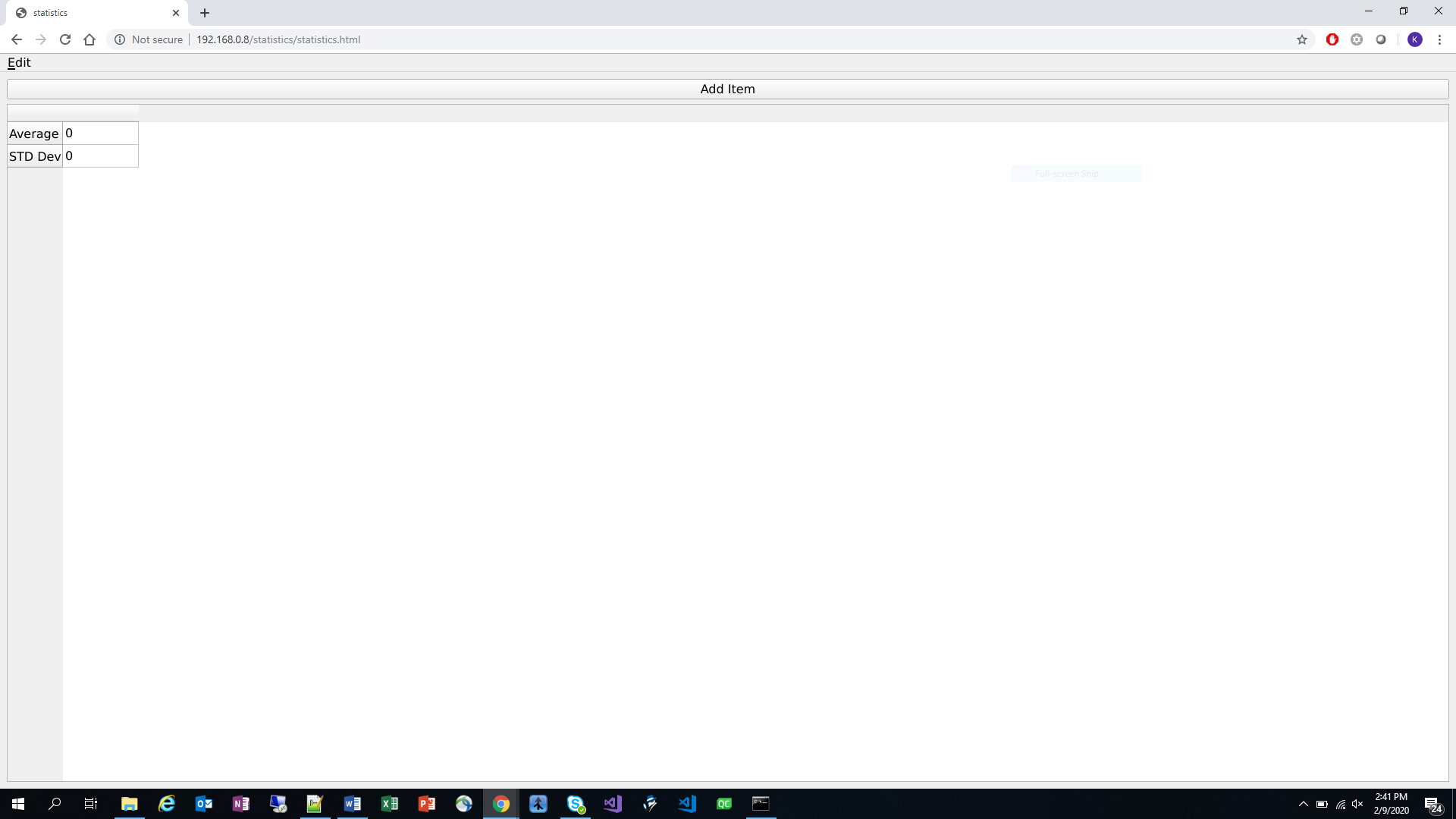


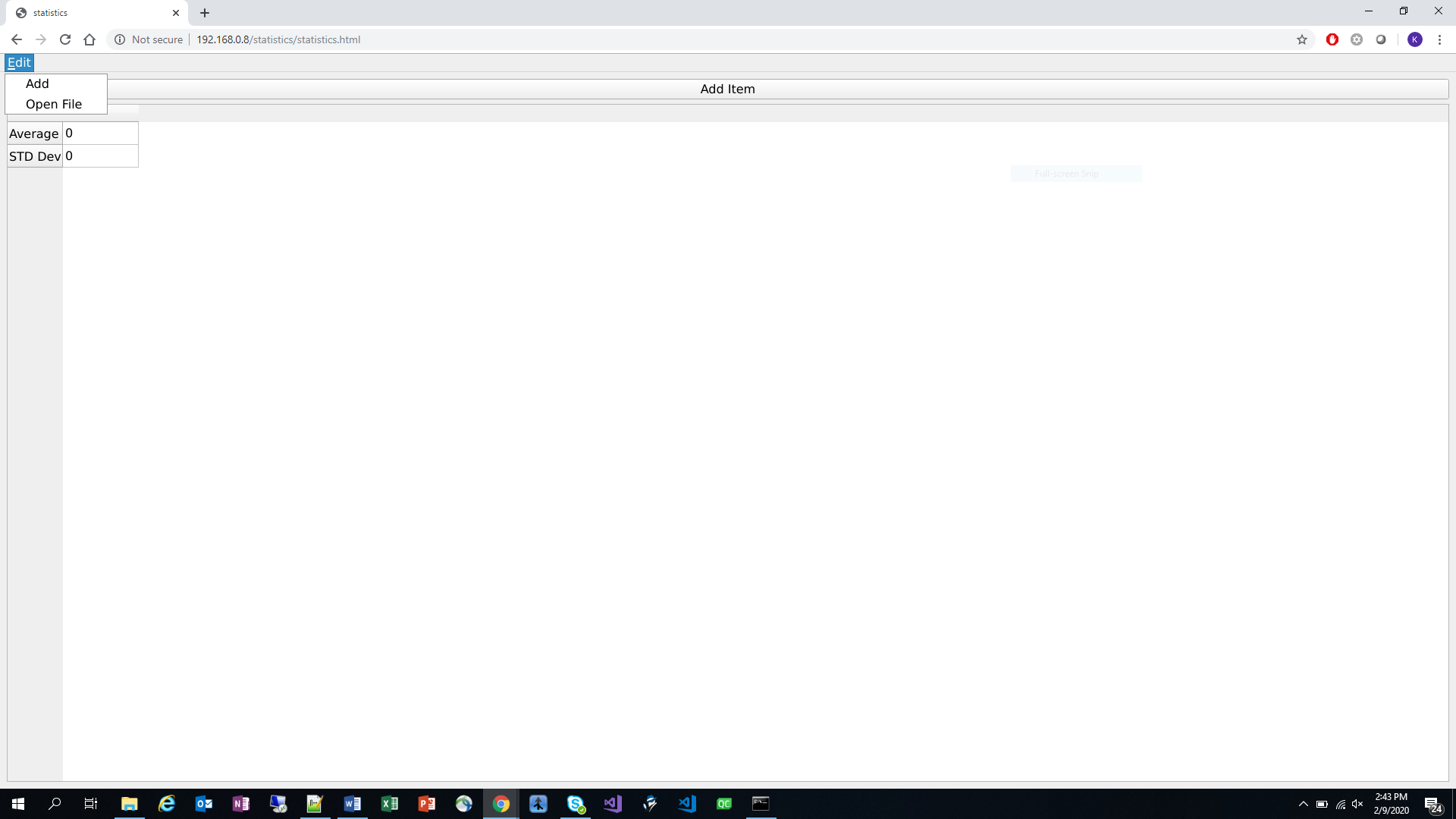
Implementation Preview:

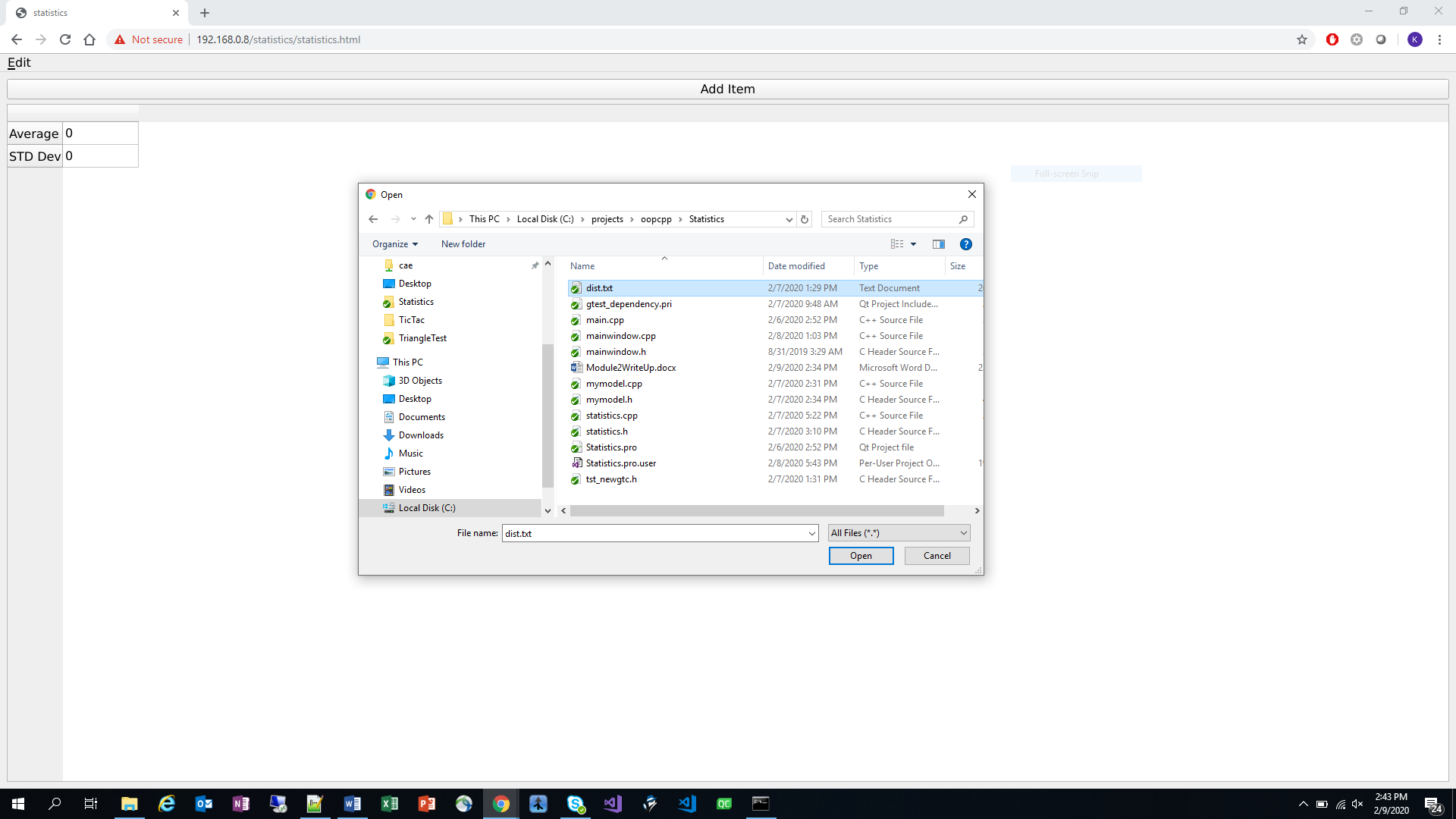
I have a webapp hosted that demonstrates the functionality of the statistics class. Right now I’m using a raspberry pi but I’m thinking of using gitlab. I can add a link to this and future implementations if there is interest.

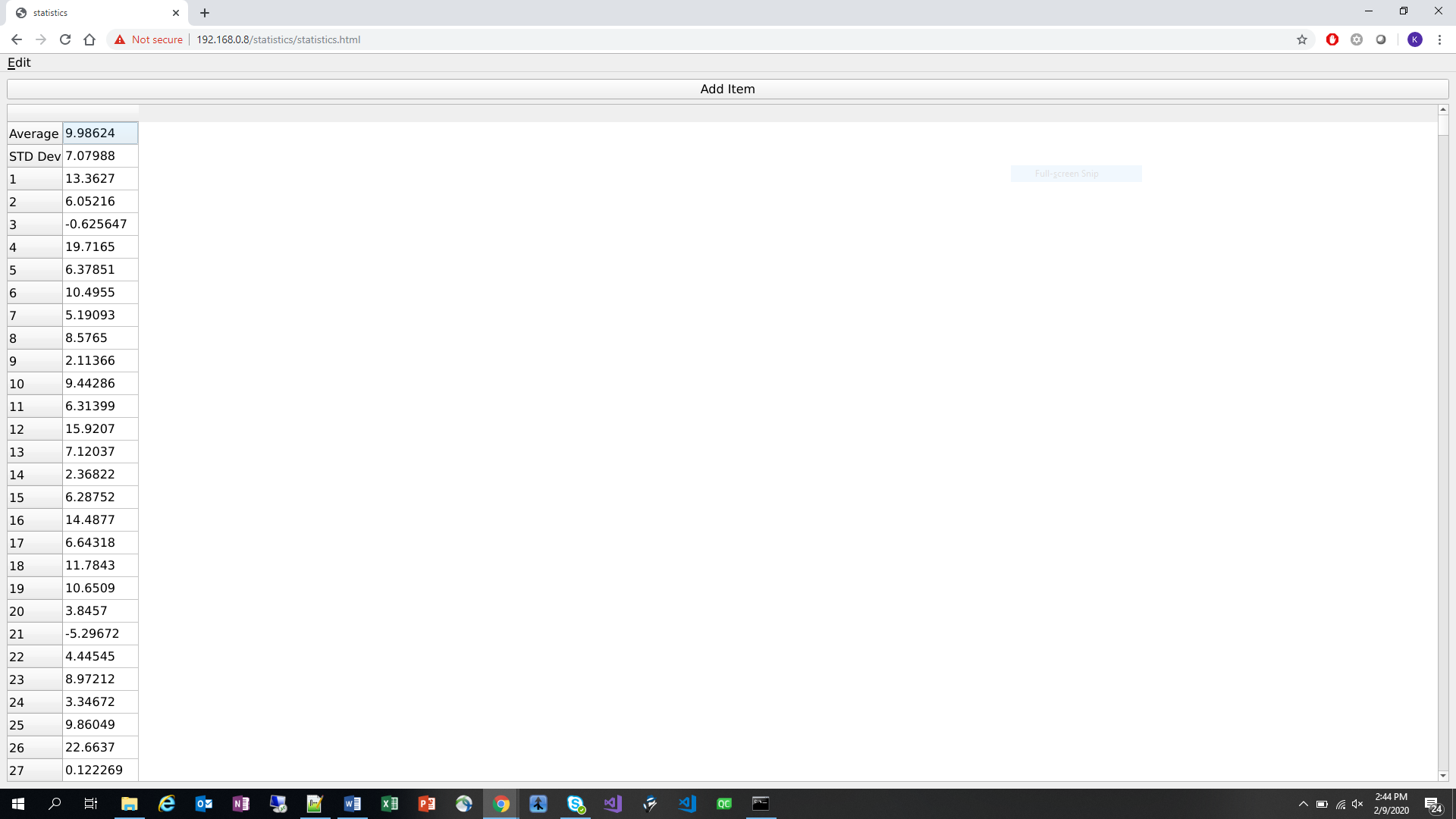
You can enter the values manually or select a file of doubles separated by newline characters. I am considering supporting separation by commas and spaces.

Screenshots of the webapp version, host is RPi, client is Windows10:









Lessons learned:

WASM currently is not well suited for working with the client file system. Currently, a virtual file system can be created on the client end, but this does not give access to the client file system, just a sandbox for temporary files.

In order to accept files from the client, a client specific asynchronous process is opened(combination of JS and html) that allows a user to select a file. On the c++ WASM side, the file contents are converted into a bytearray. If you know the form of the file, you can deserialize the contents of the file. In my case, get a list of doubles.

Source Code:

My plan is to only copy the files directly relevant to the assignment and provide links to the rest of the files. My thought is that way you don’t have to dig through the extra stuff.

For the rest of the source files I can link to the git repo or to the webserver hosting the app? I have the sources and build in the same directory on the server.

Statistics.h // definition of statistics class

Statistics.cpp // implementation of statistics class

Tst\_newgtc.h // gtest tests

// statistics.h

#ifndef STATISTICS\_H

#define STATISTICS\_H

#include <vector>

#include <iostream>

/\*

\* Statistics Class

\* Class accepts a double as an input

\* Stores that double and will return

\* the average and standard deviation

\* of the doubles.

\*/

class Statistics

{

public:

//Constructor: initialize private data

Statistics();

// add an item to the statistis

void add(double x);

// get the value of an individual item

double getItem(int index) const;

// set the value

void setItem(int index, double x);

// get average

double getAverage() const;

// get Standard deviation: using the hint

double getSTD() const;

// get number of items

int getItemCount() const;

private:

// count containing the number of items

int itemCount;

// holds all entered items

std::vector<double> items;

};

#endif // STATISTICS\_H

// statistics.cpp

#include "statistics.h"

#include <math.h>

//Constructor:

Statistics::Statistics() :

itemCount{0}

{

}

// Adds a double to the existing doubles

void Statistics::add(double x)

{

this->items.push\_back(x);

this->itemCount++;

}

// Returns the value of an entered item based on the index

double Statistics::getItem(int index) const

{

try {

if(items.size() > index){

return this->items[index];

} else {

return 0;

}

} catch (...) {

return 0;

}

}

// Modifies an existing item

void Statistics::setItem(int index, double x)

{

items[index] = x;

}

// Returns the average of the doubles

double Statistics::getAverage() const

{

double total = 0;

for(double x : items){

total += x;

}

return items.empty() ? 0 : total / items.size();

}

// returns the standard deviation of the doubles

double Statistics::getSTD() const

{

if(items.empty()){

return 0.0;

} else {

double x2 = 0;

double average = getAverage();

for(double x : items){

x2 += x\*x;

}

double s2 = (x2 - items.size() \* pow(average, 2)) / (items.size() - 1);

return sqrt(s2);

}

}

// Returns the number of doubles entered

int Statistics::getItemCount() const

{

return this->itemCount;

}

// tst\_newgtc.h

#ifndef TST\_NEWGTC\_H

#define TST\_NEWGTC\_H

#include <gtest/gtest.h>

#include <gmock/gmock-matchers.h>

#include "statistics.h"

#include <iostream>

using namespace testing;

TEST(newgt, newgtc)

{

EXPECT\_EQ(1, 1);

ASSERT\_THAT(0, Eq(0));

}

TEST(statistics, average){

Statistics stats = Statistics();

stats.add(3.0);

stats.add(4.0);

stats.add(5.0);

ASSERT\_EQ(4, stats.getAverage());

ASSERT\_NE(3, stats.getAverage());

}

TEST(statistics, stddev){

Statistics stats = Statistics();

stats.add(3.0);

stats.add(4.0);

stats.add(5.0);

ASSERT\_EQ(1, stats.getSTD());

ASSERT\_NE(2, stats.getSTD());

}

#endif // TST\_NEWGTC\_H