ClickTime Project Documentation

(The-Race-Around-The-World)

Angelo Reyes

Repo Link: ***https://github.com/angiereyes99/The-Race-Around-The-World***

Table of Contents

1 Introduction

1.1 Assumptions Made

Some assumptions I made for the problem were, when you click the start button, the time you started will be displayed in the history table and the history table will only be updated once you hit resume. And hitting resume will add entries to the time table. Another assumption I made, was recording entries in the time table will always display time, time zone, and the longitude and latitude. For the time zone aspect, I assumed it was okay to display the full date and the area we are in (i.e. Pacific).

1.2 Project Overview

The program in this repository is a stopwatch interface that uses a stop, resume, and reset button to interact with the watch and record data that includes time, time zone, and longitude and latitude measurements.

1.3 Technical Overview

Using Node.js, HTML/CSS, and Express, I implemented a frontend interface displaying a stop watch that records data in a history time table whenever the user clicks on the start and resume button. The program is run locally on localhost:8080 by running commands on the terminal. The server is ran on a js file named “app.js”.

1.4 Summary of Work Completed

The key highlights of the work I completed was mainly involved in the watchTime.ejs file. Almost all of the HTML elements were initialized in the JavaScript file along with their action events. I created the mandatory buttons start, stop, resume and reset in the file, that would be passed on to HTML. I was able to develop action events for each button and also used the geolocation API to obtain user’s longitude and latitude. I also used the Date() function to get current dates and time zone to be displayed in the time table when the start and resume button were clicked on.

2 Development Environment

I developed Click Time’s stop watch in the text editor Brackets, and ran the program locally using the terminal and its command: “node app.js”.

3 How to Build/Import your Project

To import this project (Assuming you have git installed in your device), clone the SSH or HTTPS key in the GitHub link above in the title. Open your terminal or GitBash and type “git clone -SSH or HTTPS keys you copied -. Once you have successfully cloned the repo, you should be able to see the folder in your computer’s desktop.

Afterwards, go on your terminal and direct yourself to the project folder. Follow the directions found in the README file in the repo and hit “node app.js” to run the program on localhost:8080.

4 How to Run your Project

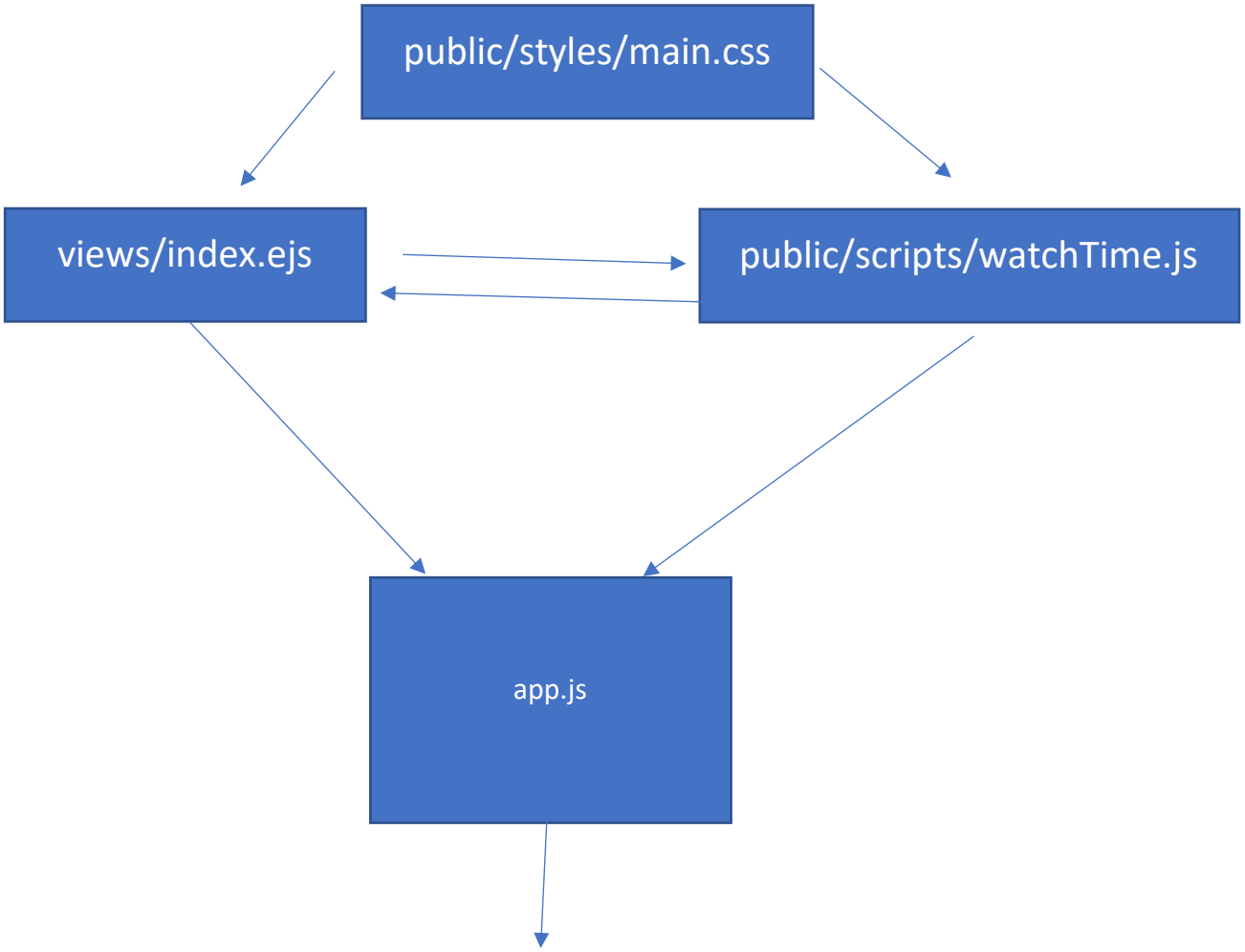
There are multiple ways to run this particular project. If you want to run the actual calculator GUI, then you simply find the file that is named “EvaluatorUI” inside the “evaluators” folder inside the calculator project folder. Right click the file and hit “Run EvaluatorUI”. In the next few seconds, it should redirect you to a small calculator where you can interact with the keys. Note that when you finish a calculation, you need to hit the “C” button first to clear previous calculations so that you can start a new one. Anything divided by 0 will also not be calculated.

To run the test scripts, go to the test folder inside the calculator project folder and follow the same exact steps from the GUI and it should run the given tests inside the folder.

5 Implementation Discussion

The main data structures utilized in this project were Stacks and a HashMap. The HashMap was implemented to hold the operator symbols and their function, while the stacks were for organizing the priority of the operators and operands based on calculations done in the calculator. Much of the process of said priority was mainly implemented in the specific operator classes and the Evaluator class. The Evaluator class was implemented using numerous for and while loops to push and pop operands and operators in and out of the stacks based on the correct priority so that calculations will be correct. There are numerous docstrings and comments throughout the code that try to explain my thought process in specific code as well.

5.1 Class Diagram



6 Project Reflection

The project overall was a very good experience in refreshing java concepts that I haven't touched in quite some time. My process of coding the program mainly revolved around the "divide and conquer" technique. I was able to be self-sufficient in the work I did and utilize problem solving techniques to get the work done. I viewed numerous resources as well as the tutoring sessions to help me progress through the project. I made a couple mistakes along the way; specifically, the Evaluator class and Evaluator GUI. I also would try to clean up come of my code in the "check" method as the only approach I took was iterating through the strings in the HashMap and check to see if the token is equal to the keys.

An important piece of the project to note was when I created the "execute" method for the divide, power, and subtract operator classes. As you may see in the code, the arguments the method takes were op1 and op2. So, when doing the correct operation for the methods in its specific class (i.e. for divide it should be op1/op2 and subtract should be op1 - op2), the results would be incorrect. An example would be, if I were to do an operation such as 10 - 5, it would give me -5. Meaning that it was actually doing it backwards. This also applied when diving such as, 5/10 would give me 2. Seeing this, I ran the test scripts for the operators and they all passed. But when running the Evaluator test, only 3/8 tests passed. I decided to just switch the op placements (i.e. op2/op1, op2 - op1) and it surprisingly passed all tests for the Evaluator test class as well as correctly compute in the GUI. But for the particular operator tests, they would fail. I was not able to debug this issue, so I just finished off the project with the working GUI and the passed Evaluator test.

7 Project Conclusion/Results

Even though the bug mentioned in the reflection caused the Subtract, Divide, and Power Operator tests to fail, the calculator GUI is fully functional and calculates correct operations in the correct order that follow the PEMDAS rule. In addition to a working GUI, the Evaluator, Operator, and Operand Test classes all passed. The code provided tried to be readable and understandable for someone who views it. In certain methods, I still provided comments to clarify any thoughts about the code. I also provided citations of code that was influenced by outside sources.