Composite Presentation/Project Profile

Project Scope:

My goal is to program a one-pager website on which chemical students can calculate required amounts of different solutions for them to mix their target solution together. This can serve as a calculator for anything from diluting a single solution to creating a complex mixture of several basic solutions (e.g., a DNA buffer). All a user needs to do is to enter the molarity of each solution they are using and the desired molarity of those solutions within the target mixture. The website will then calculate concentration and return a specific volume for each individual solution and the volume of the filler (e.g., distilled water). An example would be the creation of 1ml of G-quadruplexes with the components (and their molarity) potassium chloride (100mM), lithium cacodylate buffer (10mM), copper sulfate (4µM) and DNA (4µM). Available for use (with their respective molarities) are potassium chloride (1M), lithium cacodylate buffer (100mM), copper sulfate (1mM) and DNA (1mM). In this case the user would enter the values from above and click the “Calculate” button. For output they’d receive that they need 100µl of potassium chloride and lithium cacodylate buffer as well as 4µl of copper sulfate and DNA; the combination of those should be filled up with 792µl of distilled water.

Target Group:

As my brother is currently studying chemical biology and mentioned how that would be something nice to have, I thought this is a great project for my first web application. The target group of my project is of course not only my brother but everyone that needs to calculate solutions and concentrations frequently. This calculator should help speed up the process, especially when planning these things out in advance. While this is of course possible to calculate by hand, it takes time and a single human error in calculation can lead to hours of chemical processes and precious solutions being wasted. In my research online I’ve found several websites that offer similar calculators, but so far, I’ve not found one that would provide precisely this functionality. On the one hand I’ve found websites that are more specialized and come with preselected recipes, which makes them superior at calculating the creation of those specific recipes but unfortunately unable to provide functionality beyond their database of recipes (https://www.aatbio.com/resources/buffer-preparations-and-recipes). On the other hand, I’ve found simple concentration/molarity calculators that work the same way as my project, but only ever include a single solution (https://www.graphpad.com/quickcalcs/molarityform/); meaning calculating a solution of my test case would require 4 individual requests to the website and some additional manual calculation for the final solution. My goal with my project is to provide a simple to use, all in one solution to anyone in the filed of calculating molarity and mixing solutions.

Software development methodology:

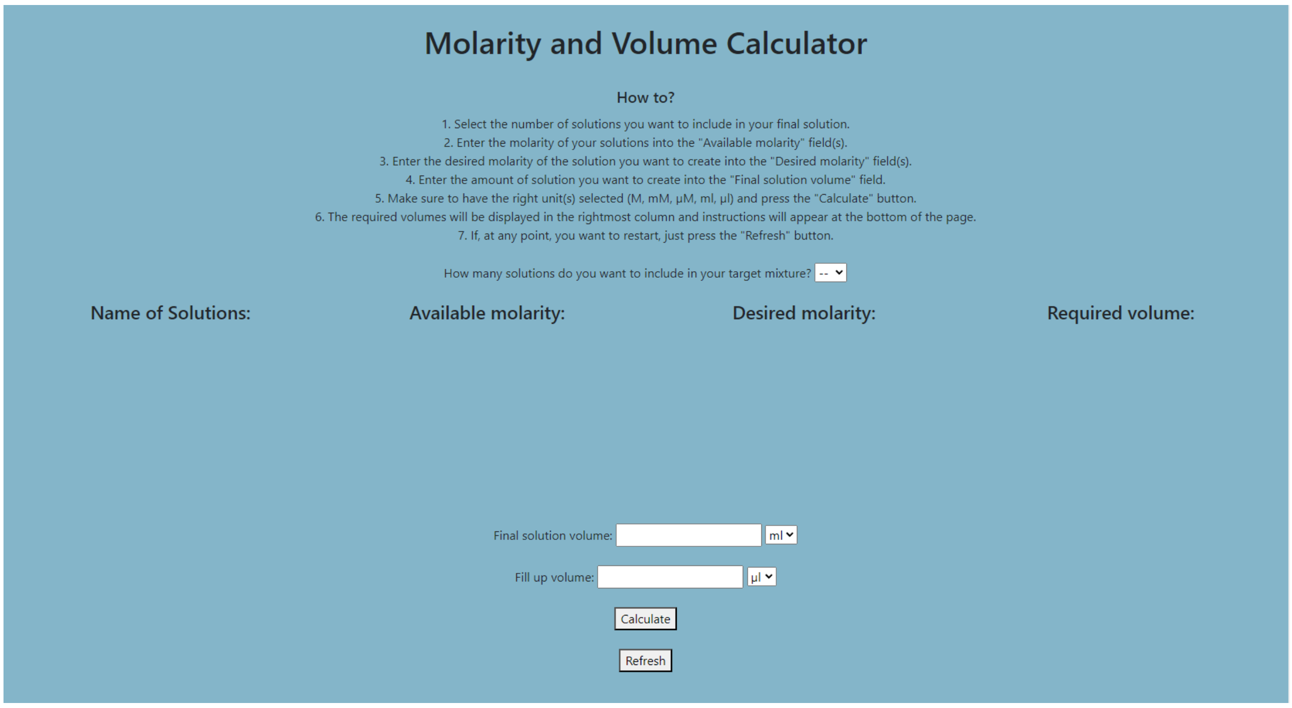
I plan on employing the big bang model I learned in my computer science class, as it is a simple and fast model and requires little planning or resources. The big bang model does not have a defined structure and order of processes. Following the big bang model means to focus all possible resources on coding and software development to get results as fast as possible. Requirements are implemented along the way whenever they arise. Its advantages include the extreme simplicity, the ease of management (as there are no hard structures to follow), the low amount of planning required, and the flexibility developers get. Additionally, it is considered a great tool for students or “newbies” to learn and practice their skills. The easy (almost nonexistent structure) also creates many disadvantages. For one, as there is no structure, there are no processes or milestones to discuss. This also means literature regarding this model is virtually nonexistent and it is only ever mentioned alongside other software development lifecycle models, but never discussed in detail as it lacks details to discuss. It carries very high risks and gets more and more unreliable the more complex the project gets. Especially for long projects that include many developers it is a bad fit as it can increase time and costs by a lot or even lead to a total abandonment of the project. But for small cases, such as this one-man project of an almost complete beginner, it seems like the perfect tool. As this is a one-person project with limited time and scope the risks regarding complexity and expensive failure cost are outweighed by the benefit of fast code delivery, learning along the way and massive flexibility. This means that I will write the code as fast as possible and solve issues I run into whenever I do. So, while I will lay out a clear idea in my composite presentation, this project has the potential to change along the way.

Tools:

I’m using visual studio code as my code editor. That’s the IDE/code editor I am most familiar and comfortable with, and I already used it for other university projects. As this is a web application and I will code the structure, style and functionality, I am using HTML, CSS and JavaScript. For my CSS framework I am using bootstrap (https://getbootstrap.com/) for their grid system and containers/rows/columns. That is the only CSS framework I’ve ever used so the choice was easy. I have written my API with python using FastAPI (https://fastapi.tiangolo.com/), Uvicorn (https://www.uvicorn.org/) and the Requests library (https://pypi.org/project/requests/) to also create a sample request with the example discussed in the project presentation. For testing the API and visualizing the functionality I’ve used Insomnia (https://insomnia.rest/). As I was completely new to the topic of APIs and had no idea how to develop one, I just went with the most intuitive looking and popular tools I could find, so I’d have more tutorial material available. Additionally, using Python that I feel more familiar with was a big help. For version control I’ve used Git (https://git-scm.com/) and the integration provided by visual studio code to connect with GitHub (https://github.com/) to keep track of code changes in my repository. For hosting the website I've used Amazon Web Services' Amplify (https://aws.amazon.com/amplify/). And finally, for the test suite I've used the JavaScript testing framework Jest (https://jestjs.io/).

User Interface:

For visualization purposes, here is a capture of my current website design:



Molarity and Volume Calculator

How to?

1. Select the number of solutions you want to include in your final solution.
2. Enter the molarity of your solutions into the "Available molarity" field(s).
3. Enter the desired molarity of the solution you want to create into the "Desired molarity" field(s).
4. Enter the amount of solution you want to create into the "Final solution volume" field.
5. Make sure to have the right unit(s) selected (M, mM, µM, ml, µl) and press the "Calculate" button.
6. The required volumes will be displayed in the rightmost column and instructions will appear at the bottom of the page.
7. If, at any point, you want to restart, just press the "Refresh" button.

How many solutions do you want to include in your target mixture? --

Name of Solutions: | Available molarity: | Desired molarity: | Required volume:

Final solution volume: ml

Fill up volume: µl

Under the header of the page, I implemented a small introduction/guide on how to use the website. The first thing a user needs to decide is how many solutions they'd like to add together. They can select a number between 1-10 from the dropdown menu and that many rows of input fields will appear. The rows are divided into 4 columns. The outer left one is used to name the solution (this isn't necessary, but a quality of life feature my brother requested). On the central left column users can enter the molarity of their current solution into the input field and select the unit of concentration from a dropdown menu. The central right columns functions identically but is used to enter the desired molarity within the target solution. Finally, the outer right column is used for the output and includes read only fields of text that will be filled once the calculation is done. I also needed to include an option for users to input the amount of their target solution they want to create as that is essential for calculating the concentration of the individual solutions. This can be found under the input rows in the center. Under the target volume of the mixture is another read only text field that will show the fill up volume after the calculation. Lastly, I need a way to initiate the calculation which I plan on doing via the calculate button. Below the calculate button is a refresh button that will reset all input fields to their default values.

Functional requirements:

- As a user I should have the flexibility input any available solution with its respective molarity, so I can work with the solutions available to me.
- As a user I should be able to input the desired molarities of my solutions, so I am flexible to mix whatever target solution I want.
- As a user I want the website to calculate the volume of the specified available solutions and the fill up volume to save time and avoid potential human calculation errors.
- As a user I want to see the results in an ordered form, so I can quickly use them.
- As a user I want to decide when to initiate the calculation so I can start it whenever I am ready with my input.
- As the website provider I want to allow only the appropriate input into the input fields so that no errors fall back to me, and users are satisfied.
- As a user I want to be able to reset all input fields, so that I can start calculating the next target solution quickly.
- As the website provider I want the website to have a step-by-step guide on how to use its functionality, so that more people can benefit from it.
- As the user I want to choose the number of solutions I combine so I only see input fields that I must fill out.
- As the website provider I want to use responsive design, so that the website can be used on many different devices.

Nonfunctional requirements:

- I want the website to be usable for any high-school graduate within 5 minutes without any additional research.
- I want the website to give the user the opportunity to chose how many solutions they’d like to mix.
- The calculations and the result generation should be performed efficiently, providing a response within 2 seconds.
- The website should support keyboard navigation.
- The design should adapt to different screen sizes (phones, tablets, laptops and desktops) and their respective resolutions in a way to makes the website usable on any of those devices.

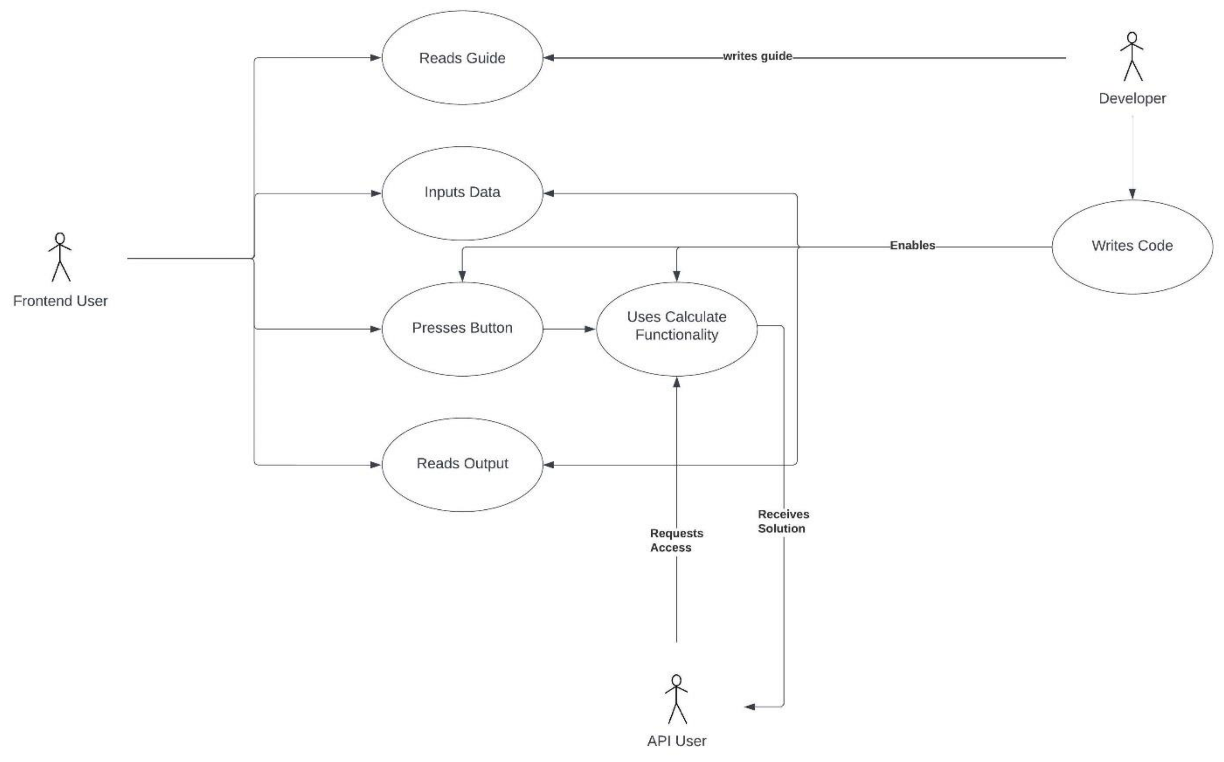
Glossary:

- Molarity (also molar concentration): a measure of concentration, in specific: the amount of substance per unit volume of a solution
- M, mM, µM: units for molarity. M (mole), mM (millimole), µM (micromole)
- Solution: a liquid mixture in which the solute (the minor component) is uniformly distributed within the solvent (the major component)
- potassium chloride, lithium cacodylate buffer, copper sulfate, DNA: chemical solutions that can be used together to produce G-quadruplexes.
- G-quadruplexes: secondary structures that can form in DNA/RNA by combination of four specific molecules combine and create a unique shape. They are the subject of much active research in the fields of molecular biology, biochemistry, and more.

System Design:

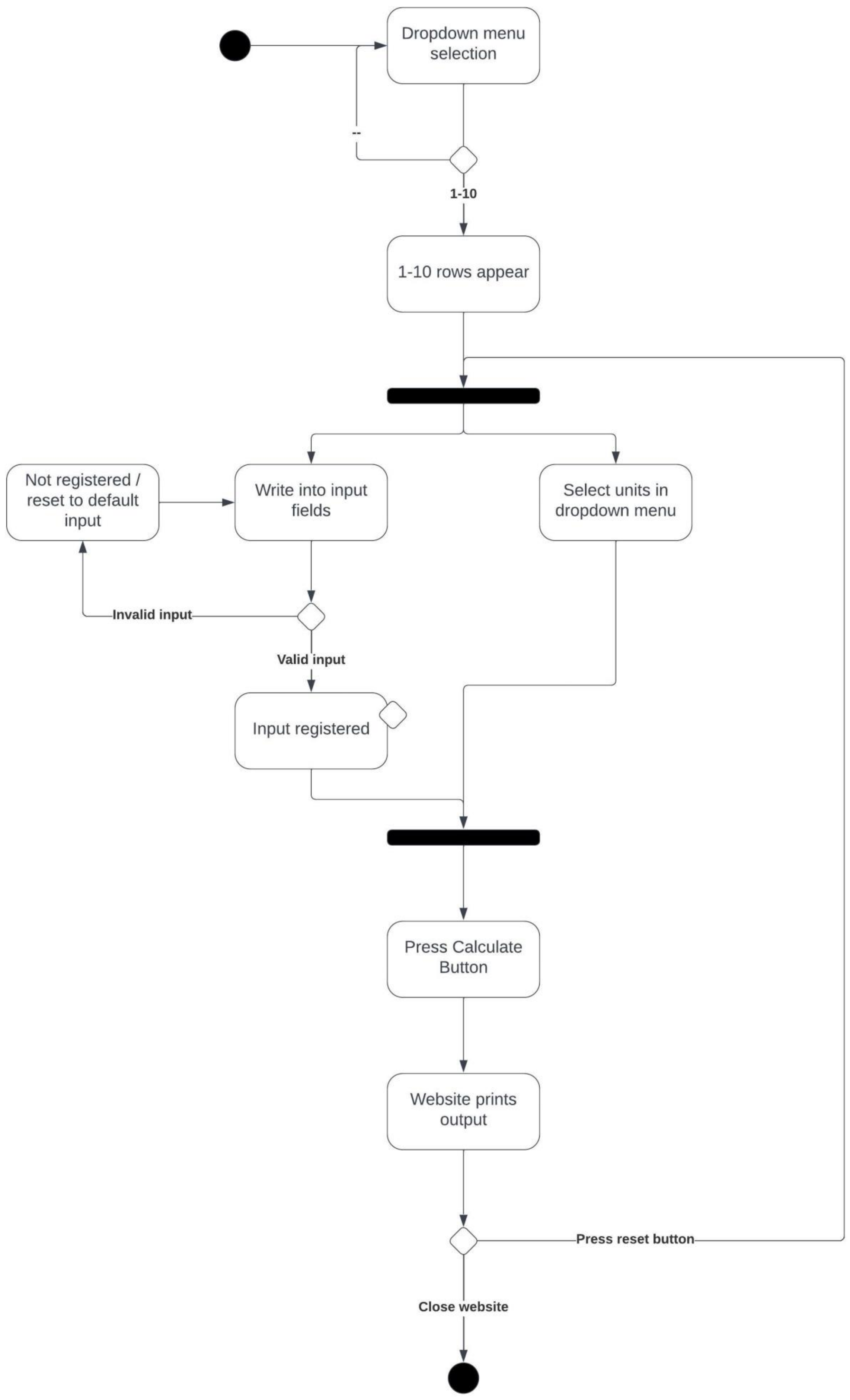
UML Use Case Diagram:

I've decided to go for a UML use case diagram in order to visualize the system. It looks like this:

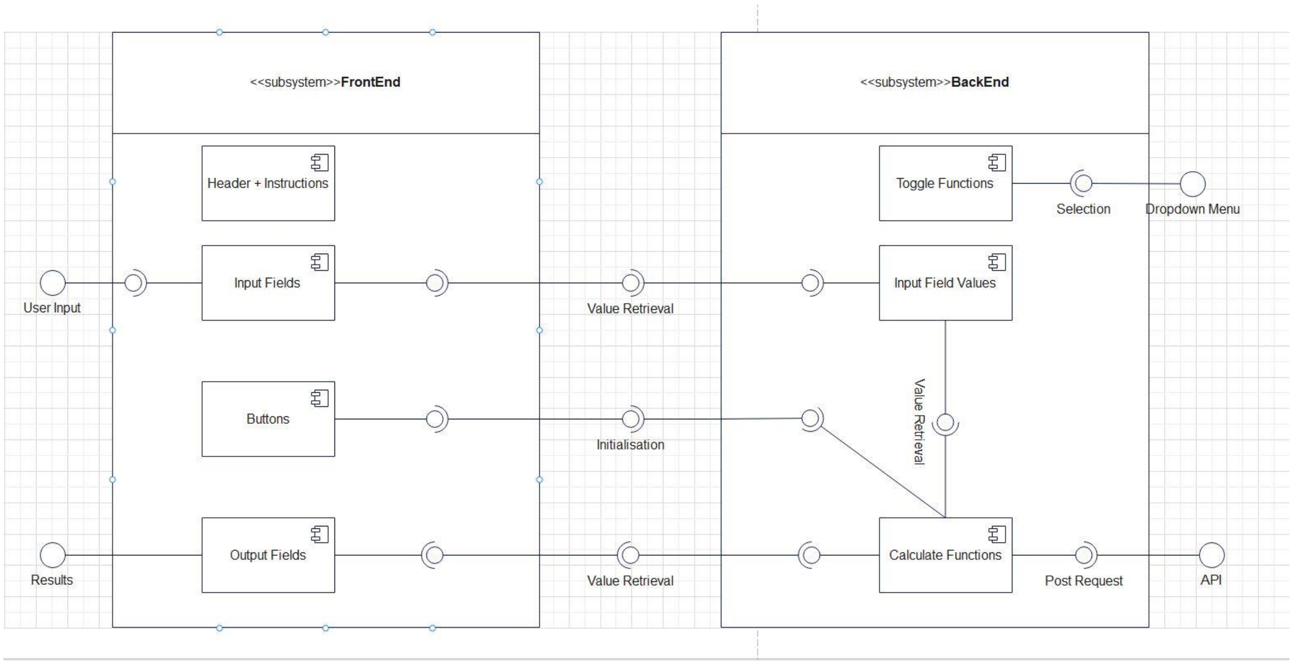


UML Activity Diagram:

In order to portrait the processes on the website I've created an activity diagram as follows:


Dropdown menu selection
--
1-10
1-10 rows appear
Not registered / reset to default input
Write into input fields
Select units in dropdown menu
Invalid input
Valid input
Input registered
Press Calculate Button
Website prints output
Press reset button
Close website

UML Component Diagram:



Implementation:

I have a html file that encompasses most of my website structure. Ever since I implemented bootstrap my CSS code shrunk a lot as I could save myself lots of work by using bootstraps row, column and container-fluid objects. I have a header area in which I wrote my short introduction and guide, below that my rows that can be toggled on or off, each with 4 columns. The 3 columns starting on the left all include input fields (1 text, 2 number) that all have their individual id (from 1 to 10). Each input field has a default value that allows me to calculate the volumes even if rows are left empty. The number fields are restricted to only enter numbers as I want to avoid undetected errors in calculation. The right most column is read only as it only will present the output calculated. Below the bootstrap grid I added 2 more centered input fields, one to take the target volume, the other (read only) to output the fill up volume. Lastly, I implemented a calculate and a refresh button below.

Regarding functionality, I have two functions that get called on load of the website via an event listener. One toggles the rows so that on website load the rows initially are not visible, the other stores the default values assigned to the individual input fields in an array so that they can be restored via the refresh button. The toggle rows function also allows for the user to select the number of rows made visible (the amount a user wants to use). To make the use of the input fields easier I implemented an event listener that clear the default values when an input field is focused. If the field already has some user input, it won’t get cleared though so users have the option to correct their input without having to type it again.

The base functionality of getting the input data, doing the volume calculation and providing output works as follows. When the calculate button get clicked the functions create an array, convert the value of the input fields to micromole (for calculations), and save the converted values in the array. The volume then gets calculated by dividing the total volume of the target solution by the quotient out of the available molarity and the desired molarity. The fill up volume simply is the difference between the previously calculated individual volumes and the total volume. Once calculated, these values then are output in the required volume column of the grid, and the fill up volume in its own text field.

Test suite:

For my test suite I was planning on simply importing the functions into a test file and run them all there. With most functions including some reference to the html file it seemed from my online research that the best way to include those in the test would be to use jsdom to simulate the DOM (document object model) of my website. After countless hours and several different configurations and tries this didn’t work for me. So, I instead created test files for each function and refactored the function code a little bit to include the input from the html files as parameters instead. This way the functionality still gets tested in isolation. Unfortunately, this also means that I only could write tests for the core functionality of the website. Other functions, such as the toggle function for the rows or the event listeners used to reset html input cells to their default value (or to remove that default value on click) are without tests.

API:

I’ve designed a simple API that allows a post request to the server accessing my calculate_volumes() function. This way it is possible for anyone to access the function without having to use the website or its interface. It is designed to receive a Json package including the data of available molarity, desired molarity and total volume. The return package contains the required volumes of each solution and the fill-up volume. I have added a simple requests file that includes the test case I’ve presented in the project scope and lets that combination of 4 solutions be calculated by my function via the API.

Review:

My final product includes a website hosted via AWS, a test suite covering the functionality of the calculations using Jest, an installation guide, and an API (using FastAPI) with an example post request showcasing the calculations for the sample solution from the project scope. While I wish I’d been more familiar with CSS frameworks to improve the website design, I am happy looking back on a working project that includes me doing a lot of things for the first time ever.

Making of:

I started the project simply with the idea my brother gave me to create a molarity calculator. I’d done some html basics in high school but not touched it since then, so I had to relearn everything from scratch. Originally, I started out with a header, a small instruction guide and a fixed amount of input fields which were also used for the output. I struggled with the structure for a while until I implemented bootstrap’s grid system. This helped me format the columns in a responsive way. After that I implemented the dropdown menus for the units of the solutions to save some time for users typing in bigger numbers. Next, I started with the functionality. First, importing values from the input field and converting them in a base unit and then the calculations themselves. At that point I needed a way to initialize the calculation and added the button. With this, the basic functionality stood but I still wanted to add and change things, so I looked for a way to toggle rows and let the number of solutions (rows) included in the mixture be chosen by the user. With this I had to create more rows (and toggle them off at times) which led to the need for default values in each input field in order to keep the input fields with default values out of the calculation. Otherwise, I’d be getting errors every time I pressed the calculate button. This allowed also for the implementation of an event listener that would remove the default value on clicking an input field, but not other values (as to not make the user retype his values when trying to correct a single digit/character). These default values also get stored on loading of the website for the refresh button to have the data to restore everything back to default.

At that point the website was mostly finished, and I started working on the API. I first wanted to build a web API with node.js and Express, but when I was struggling with that and learned I could just build one with Python instead I jumped on that opportunity. After hours of trying and testing I managed to successfully create a request that represented the example test case and would correctly apply the calculate_volumes() function from my website and return a package which included the required volumes.

One of my last steps was writing the tests. Again, I struggled for a long time trying to figure out how to mock a DOM in order to just import a function and test it as it is in the code. When I wasn’t able to do that I resorted to refactoring the functions and passing the input values as arguments. I did that for the core functionality that doesn’t interact with the HTML.

Finally, I followed a basic tutorial on how to host a website on AWS and used AWS Amplify to launch my website.

Lessons learned:

I’ve learned a lot during this project. First off, basics of HTML, CSS and JavaScript that I’ve never been familiar with. This is my first time employing the bootstrap framework (and not just hearing or reading about it in a course book). I’ve learned to distinguish between “id” and “class” in HTML and CSS for styling individual elements or elements of a class. I’ve familiarized myself (at least to a certain degree) with writing functions in JavaScript, importing the scripts into the HTML file and using input from HTML in turn for functionality in JavaScript. I learned about event listeners and how to trigger functions without pressing a button but on different signals. I learned about default values, different input fields and buttons. I have learned what an API is (other than something that usually includes documentation on how to use stuff), and at least at a basic level how to write one and allow a certain type of requests to interact with website functionality. Lastly, I've familiarized myself with FastAPI, Insomnia, Jest, AWS Amplify and even some other tools that I ended up not using in the final product.

Technical debts:

- Despite hours of trying and researching, the toggleRows() function will hide and reveal the rows as intended, but even hidden the rows still take up space on the grid leading to an awkward looking gap between the column headers and the lower input fields and buttons. This could have been avoided by someone more experienced with bootstrap (or other frameworks?) but I couldn't find any guidance on how to fix it.
- The test coverage isn't as high as I'd like it to be. I struggled with setting up a DOM with JSDOM to simulate the html input fields for my test suite so I could just import the functions. Because of that I had to refactor the calculation functions and implement the input fields as function parameters and pass the otherwise user input as arguments. Ultimately, this was more work that could've been avoided with more familiarity with JavaScript testing frameworks which I unfortunately didn't have.

Links:

Github project is hosted on: https://github.com/dariuszarse/Molarity-and-Volume-Calculator/tree/main/Code

Website is hosted (via AWS) on: