# Lilly Technical Challenge Documentation

## Approach

*I went through the objectives in a linear order, sequentially top to bottom. The first part of the approach I came up with revolved around retrieving data from the backend and displaying it. To do this, I used the fetch function which, by default, makes a GET request. After retrieving the data using the API, I ran some validation to ensure that data was consistent and had integrity. I ensured that values were to 2 decimal places or if there was no price or name entered, it would say “Unknown Medicine” or “Price Unavailable”. I then entered the name and price into span elements and appended them to the same list item. This allowed for me to manipulate their design, and placement, with ease.*

*Then I worked down the remaining objectives sequentially. The first was a new medicine field, of which I first created a form with three input fields: one for the name, one for the price, and the final submit button with validation (min=0, increments of .01). I specified that the action was to connect to* [*http://localhost:8000/create*](http://localhost:8000/create) *with the POST method. After executing this, I realised that the webpage would redirect to the link above, which was unwanted, so I added event.preventDefault() inside of an event listener in JavaScript.*

I then worked on an implementation of the update medicine card. This used the same form as the previous card, but I changed the first box to be a dropdown of all items that currently exist in the database. This prevents the entering of erroneous data and allows for quicker interactions for the user. I then added a price input field, which follows the same rules as the add medicine field.

The delete medicine form, uses the same dropdown box as the update medicine box, however it has no price input field. It has a red delete button to show that it’s a potentially dangerous function, however it does not have a confirm popup.

After finishing the main objectives, I created an average function in main.py. I first approached this at a basic level, having a for loop that would get price from each entry in the dictionary. However, I needed to account for when the dictionary is empty with an if statement and return 0, if it was. This was then displayed beneath the medicine list header.

After finishing the basic look, I worked on improving the interface, and design. I made the medicine list alternating in colour for readability.

## Objectives - Innovative Solutions

* *Deletion of medicine: I wanted to create a dropdown option filled with all medicine names that allows you to select one and then delete one.*
* *Prevention of duplicates: I added a check when you press the “Add” button, that will check whether there currently exists a medicine of the same name. To achieve this, I used the medicines/{name} endpoint to see if there was a medicine that existed with that name, and if it did, I would return out of the function and alert the user. Previously, I had a function that would do a for loop to check the ul in the medicine tracker but realised that it could be exploited by inspecting and deleting elements on the client.*
* *Interactive medicine list. I made it so when you hovered over an item in the medicine-list, it would set the values of the select elements to that value.*

## Problems Faced

* *Validation and formatting of data: Such as with missing data, to keep integrity, I had to handle output formatting for readability.*
* *I had an issue with posting to the backend, as I needed to specify the http protocol in the localhost URL. The same applied for when I was retrieving all medicines.*
* *Submission of data for updating medicine price wouldn’t process as choosing an item in the dropdown wouldn’t submit that value. I realised later, that this was result of the name of the select element not being “name”.*
* *What happens if there are duplicate medication names? Do I prevent it entirely? Allow it and attach a unique identifier to allow for processing? As an example, when I implemented the update medicine card. I needed to either omit duplicates or find a way to access common names. I opted for ensuring each medicine was uniquely named.*
* *After adding the delete medicine fields, I had an issue “TypeError: window.fetch: HEAD or GET Request cannot have a body.” After doing some research with form method support, I found that the DELETE method wasn’t supported and would set the method to GET. To resolve this, I directly entered the method into the fetch statement in JavaScript.*
* *When making the medicine-list interactive, I had an issue where clicking the text in each list item would not cause the select boxes to set to that value. To solve this, I added a check to navigate to the medicine-name class and grab its text content.*
* *When deleting an item without a name from clicking in the medicine list, I needed a way to delete “”, as opposed to “Unknown Medicine”, to do this, I used an if statement to check the value of textContent before setting the dropdown value.*

## Evaluation

*Overall, the challenge, I felt, went well. There are parts that went not as smooth, such as the issue with the “DELETE” method not being accepted in the HTML form, so I had to research why and declare it manually in JavaScript. I managed to do the tasks within the time frame, it’s more so the design and appearance that took most of the time. If I was to approach this again, and were given more time, I would hide the update and delete function and make them appear when you click on a medicine in the list in a sort-of-dashboard-like manner. And, where I could, probably display a notification at the top right for the responses from the backend formatted for user feedback. Further increasing user feedback, I would change the outlines of boxes to display green or red when valid/invalid, and grey out the submit button, until all boxes are valid.*