What were some of the alternative design options considered? Why did you choose the selected option?

In the design, I use the unique id as an identifier for each event. The unique id was generated by the server and returned to the client when adding the event to the calendar.

We can use the id to update/delete the event.

There is another option to update/delete an event, we can simply provide the name and date to do it.

My thought was that we may have multiple events with the same name in one day. So the alternative design does not apply to this situation.

What changes did you need to make to your tests (if any) to get them to pass. Why were those changes needed, and do they shed any light on your design?

The first version of the test is not well-rounded. And because I cannot run the test, some of the test codes are not runnable. In the final version, I adjust the test codes to make them able to run.

Except for that, instead of using the assertation functions provided by io.Restassured, I found that I need to realize more complicated logic, such as checking whether an event exists in a list. This logic can be easier to code using Junit Assertation.

Pick one design principle discussed in class and describe how your design adheres to this principle.

Test Driven Development

I write the test cases before designing the API (finish A1.1 before A1.2). I use the test cases to debug my design codes. For example, after I update the date of one of the events I created, according to my test case, I should not see it again in the events list of the original date. I failed to pass the test at first. Then I know that there is a flaw in my logic, so I go back to the design codes to fix it.