A1.2 Reflection

Jasmine Chu (xiyinc)

Code Implementation

(I put the code in assignment0's repository. It is in branch "rest")
https://github.com/cmu-17-625/assignment-0-SheenC/tree/rest

Reflection

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

Function	Design	Alternative Options	Choice Reason
Get the current date	GET /DAI/current/date	GET /DAI/date	Creating a new taxonomy "/current" will be better for future extension. If in the future we would like to get the past or future date, we can easily add a new "folder". Besides, it will be more clear to put all current related attributes in the same "/current" path. It reduces confusion about what date it is.
Get the current month	GET /DAI/current/month	GET /DAI/month	
Get the current year	GET /DAI/current/year	GET /DAI/year	
Get event(s) for a given date	GET /DAI/allEvents/year/{yyyy}/month/{mm}/date/{dd}	GET /DAI/allEvents?month={mm}&date={dd}&year={yyyy}	I choose the descriptive URL rather than query string parameters like search criteria. It will be more simple and direct for users to input.
Add event	POST /DAI/event?month={mm}&date={dd}&year={yyyy}&patient={p}&doctor{d}}&comment={c}	POST /DAI/event (Put parameter in request body)	I choose to put the request parameters in the url rather than in the request body. Because the input is not much and it will be more direct and simple to put parameters in the url. Besides, it will be easier for servers to parse the parameters since we do not need to deal with json or xml.
Modify event	PUT http://localhost:4567/DAI/even	PUT http://localhost:4567/D	I choose the search criteria formation rather than the

	t/{id}?month={mm}&patient={p}	AI/event/{id}/month/{mm}/patient/{p}	descriptive URI. Because users can modify part of an event. It is more flexible about the input parameters order.
Delete event	DELETE http://localhost:4567/DAI/event/{id}	DELETE http://localhost:4567/DAI/event?id={id}	I choose the descriptive URI rather than the search criteria formation. Because we only have one parameter id, using descriptive URI formation is more simple and direct.

- **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?**

Change 1:

I reduced the nest in the JSON response. For example, I write the test case for “List all events for a given date” using the expected response format as follow:

```
{
   "events":{
      "allEvents":[
            {
             "eventId" : 202211050001,
             "time" : {date: 5, month: 11, year: 2022},
             "patient" : "Jason123",
             "doctor" : "M.D.Julie",
             "comment" : "The patient got a Flu."
            },
            {
             "eventId" : 202211050002,
             "time" : {date: 5, month: 11, year: 2022},
             "patient" : "LucyLee",
             "doctor" : "M.D.Patrick",
             "comment" : "The patient had a wisdom tooth extraction."
            },
      ]
   }
}
```

Test Case:

```
@Test
public void dai_returns_200_with_expected_all_events_given_date() {

    when().
            get("/DAI/allEvents?year={yyyy}&month={mm}&date={dd}",
2022,11,5).
    then().
```

```
            statusCode(200).
            body("events.allEvents", hasItems(202211050001,202211050002));

}
```

But I realize that the response format does not have to be nested and complicated. It can be a simple json list as follow:

```
      [
            {
             "eventId" : 202211050001,
             "time" : {date: 5, month: 11, year: 2022},
             "patient" : "Jason123",
             "doctor" : "M.D.Julie",
             "comment" : "The patient got a Flu."
            },
            {
             "eventId" : 202211050002,
             "time" : {date: 5, month: 11, year: 2022},
             "patient" : "LucyLee",
             "doctor" : "M.D.Patrick",
             "comment" : "The patient had a wisdom tooth extraction."
            },
      ]
```

I revised my test case to check for the content of the json list directly. This test will be more accurate than before.

```
@Test
public void dai_returns_200_with_expected_all_events_given_date()
{
   RestAssured.baseURI = "http://localhost:4567/DAI";
   RequestSpecification httpRequest = RestAssured.given();
   Response response =
httpRequest.get("/allEvents/year/{yyyy}/month/{mm}/date/{dd}", 2022, 11,
5);

   int statusCode = response.statusCode();
   String responseBody = response.getBody().asString();

   assertEquals(200, statusCode, "Response status code 200");

assertEquals("[{\"eventID\":\"1\",\"month\":11,\"date\":5,\"year\":2022,\"
patient\":\"Jason123\",\"doctor\":\"M.D. Julie\",\"comment\":\"The patient
got a
Flu.\"},{\"eventID\":\"2\",\"month\":11,\"date\":5,\"year\":2022,\"patient
\":\"LucyLee\",\"doctor\":\"M.D. Patrick\",\"comment\":\"The patient had a
wisdom tooth extraction.\"}]", responseBody, "Return all events on that
date");
```

```
}
```

Besides I also found the descriptive URI formation is more convenient to parse parameters especially when I use SparkJava to write REST APIs. So I revised the original search criteria formation to descriptive URI formation. In this way, the change of test cases shed light on my design.

Change 2:

I also revised the status code in the test cases. Before, when I deal with the invalid user input, I suppose it will have a 400 data invalid error. For example, when a user input the month as a string “July” rather than integer 7, it will return an error code. But when I’m implementing the code, I found that the error code should be a 500 internal server error. Because the error happened in my code when parsing a string into an integer.

```
    @Test
    public void dai_returns_400_post_new_event_invalid() {
        String requestBody = "{\n" +
                "  \"eventId\": 202211050003,\n" +
                 "  \"time\": \n" +
                     "{\"date\": 5, \"month\": \"July\", \"year\":
               2022}, \n" +
                "  \"patient\": \"Leslie987\", \n" +
                 "  \"doctor\": \"M.D.Daniel\", \n" +
                 "  \"Comment\": \"Physical examination\" \n}";
        Response response = given()
                .header("Content-type", "application/json")
                .and()
                .body(requestBody)
                .when()
                .post("/DAI/event")
                .then()
                .extract().response();

        Assertions.assertEquals(400, response.statusCode());
}
```

The revised test case is as follows:

```
@Test
public void dai_returns_500_post_new_event_internal_server_error()
{
   RestAssured.baseURI = "http://localhost:4567/DAI";
   RequestSpecification httpRequest = RestAssured.given();
   Response response =
httpRequest.post("/event?month={mm}&date={dd}&year={yyyy}&patient={p}
&doctor{d}}&comment={c}",11,"Jul",2022,"kelly","M.D. Andrew", "The
patient got a headache.");

   int statusCode = response.statusCode();

   assertEquals(500, statusCode, "Response status code 500");
}
```

The change of the test cases also shed light on my design. Since the input is not much, I changed the request parameters in URI search criteria formation rather than in the request body. In this way, it is more convenient to code because we do not have to deal with JSON objects. We do not need to write long and complicated JSON string tests.

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

I picked the ‘Single Responsibility’ principle to describe my design.
For example, when designing the add/modify/delete event API, I divided them into three separate APIs rather than an integrated single API. If we use an integrated API like “POST /DAI/event/id/{id}?operation={add/modify/delete}&attribute={value}&...”, the API will handle too many jobs. The class and method handling this API will be large and hard to maintain and extend in the future. So separating them into three smaller APIs can make it easier to be implemented, maintained and extended. Besides, adhering to Single Responsibility principle can also make our API service easier to deploy on several microservices and distributed systems.