Peng Dou(pdou)
GitHub: https://github.com/cmu-17-625/a4-api-evolution-DP-Victor.git

Running Tests

The test cases should run in two terminals. One terminal is to start the server. The other is to run the test cases. The api is implemented by javalin.

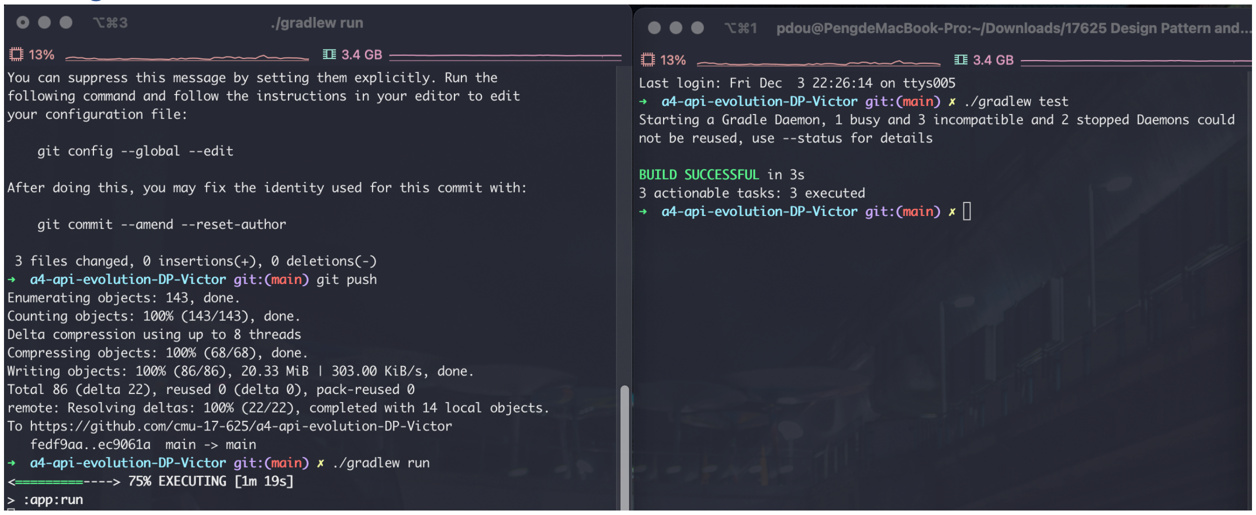
Terminal1: Open the server

Open a terminal at folder a4-api-evolution-DP-Victor.
$ ./gradlew run

Terminal2: Run the test

Don't close terminal 1.
Open another terminal at folder a4-api-evolution-DP-Victor.
$ ./gradlew test

Testing Result


⌥⌘3
./gradlew run
13%
3.4 GB
⌥⌘1
pdou@PengdeMacBook-Pro:~/Downloads/17625 Design Pattern and...
13%
3.4 GB

```
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 3 files changed, 0 insertions(+), 0 deletions(-)
➜  a4-api-evolution-DP-Victor git:(main) git push
Enumerating objects: 143, done.
Counting objects: 100% (143/143), done.
Delta compression using up to 8 threads
Compressing objects: 100% (68/68), done.
Writing objects: 100% (86/86), 20.33 MiB | 303.00 KiB/s, done.
Total 86 (delta 22), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (22/22), completed with 14 local objects.
To https://github.com/cmu-17-625/a4-api-evolution-DP-Victor
   fedf9aa..ec9061a  main -> main
➜  a4-api-evolution-DP-Victor git:(main) ✗ ./gradlew run
<=========----> 75% EXECUTING [1m 19s]
> :app:run
```

```
Last login: Fri Dec  3 22:26:14 on ttys005
➜  a4-api-evolution-DP-Victor git:(main) ✗ ./gradlew test
Starting a Gradle Daemon, 1 busy and 3 incompatible and 2 stopped Daemons could
not be reused, use --status for details

BUILD SUCCESSFUL in 3s
3 actionable tasks: 3 executed
➜  a4-api-evolution-DP-Victor git:(main) ✗
```

Reflection

Q1. Comparing Previous Design

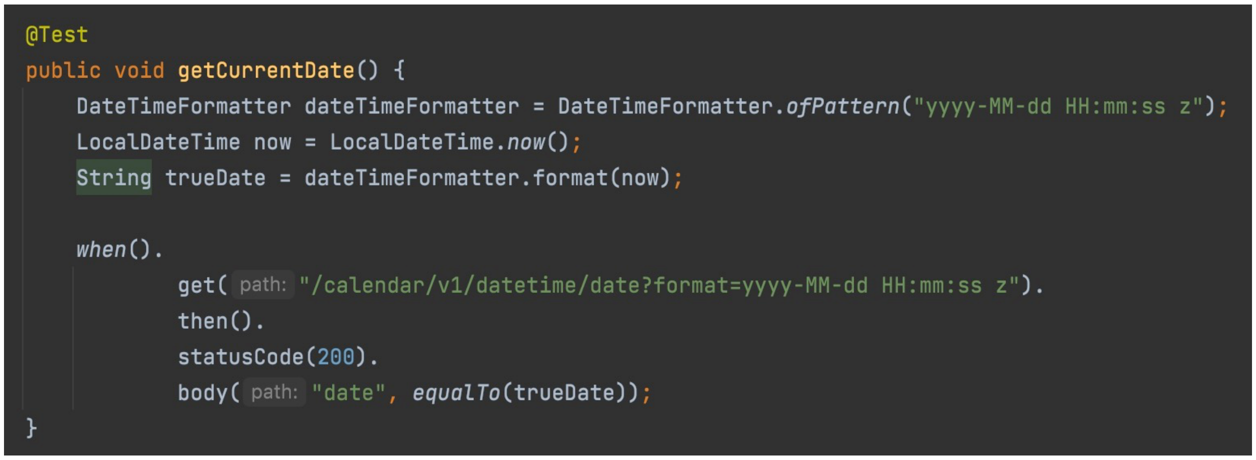
The documentation for this api is included in last homework. One difference is that I am actually using http request to get the recourses in each endpoint, so the return could be unpredictable. I should consider the consistency of the returns in this homework in order to make the api easy to use. I chose the lightweight object string as return without using the json object, and I designed similar return pattern for all valid and invalid cases. However, in my previous api, I did not consider such a thing since the return type is fixed. Additionally, my RESTful design has a lot of dependencies, thus the error might be more unpredictable, either for me or the user. For example, when I am trying to access empty resources in the test cases, the server returns status code 404, which is not specified by me in my api. As a developer, I should consider those situations to improve the usability in this case. However, in my previous api, almost all the returns could be handled by myself, so I did not have to consider like this so much.

Q2. Modifications

Changed the return from JSON object to string

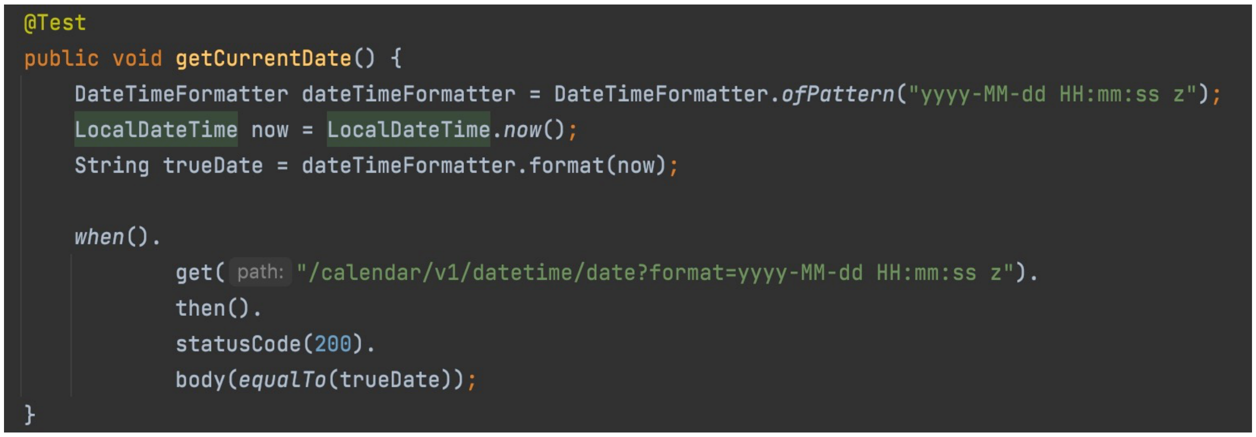
Modified test functions: All

Reason:
When I was designing the test cases, I did know that json object actually takes a lot of space. Now I modify the output body of the api, which is only a string, and it is sufficient to serve the user. This modification should reduce the load of the server and I will consider it in the future when I am developing my api.



```java
@Test
public void getCurrentDate() {
    DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss z");
    LocalDateTime now = LocalDateTime.now();
    String trueDate = dateTimeFormatter.format(now);

    when().
            get( path: "/calendar/v1/datetime/date?format=yyyy-MM-dd HH:mm:ss z").
            then().
            statusCode(200).
            body( path: "date", equalTo(trueDate));
}
```

Figure 1 Test case example before changing



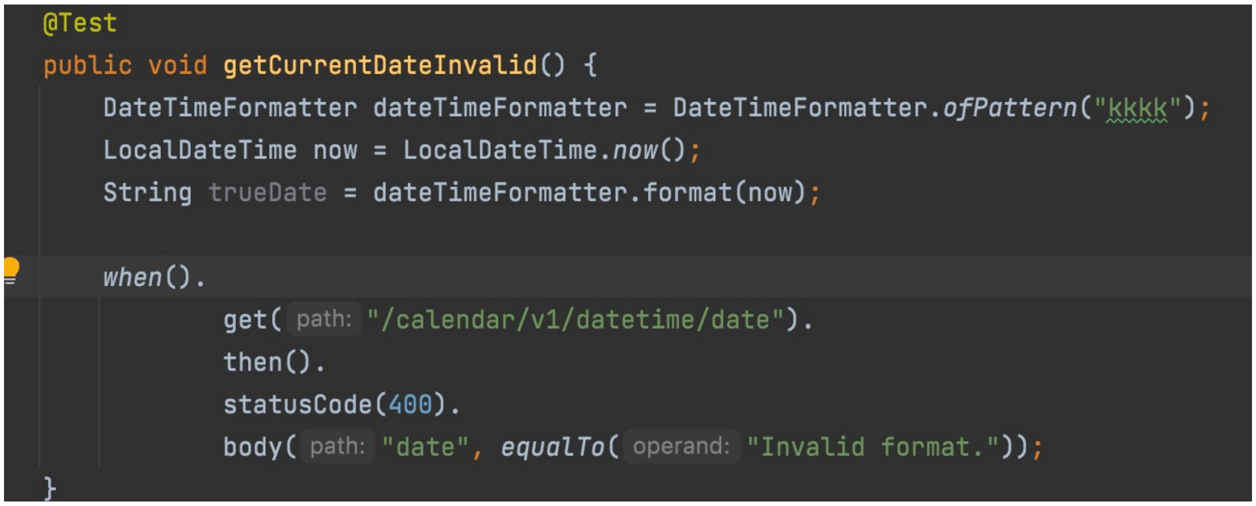
```java
@Test
public void getCurrentDate() {
    DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss z");
    LocalDateTime now = LocalDateTime.now();
    String trueDate = dateTimeFormatter.format(now);

    when().
            get( path: "/calendar/v1/datetime/date?format=yyyy-MM-dd HH:mm:ss z").
            then().
            statusCode(200).
            body(equalTo(trueDate));
}
```

Figure 2 Test case example after changing

Changed getCurrentDateInvalid()

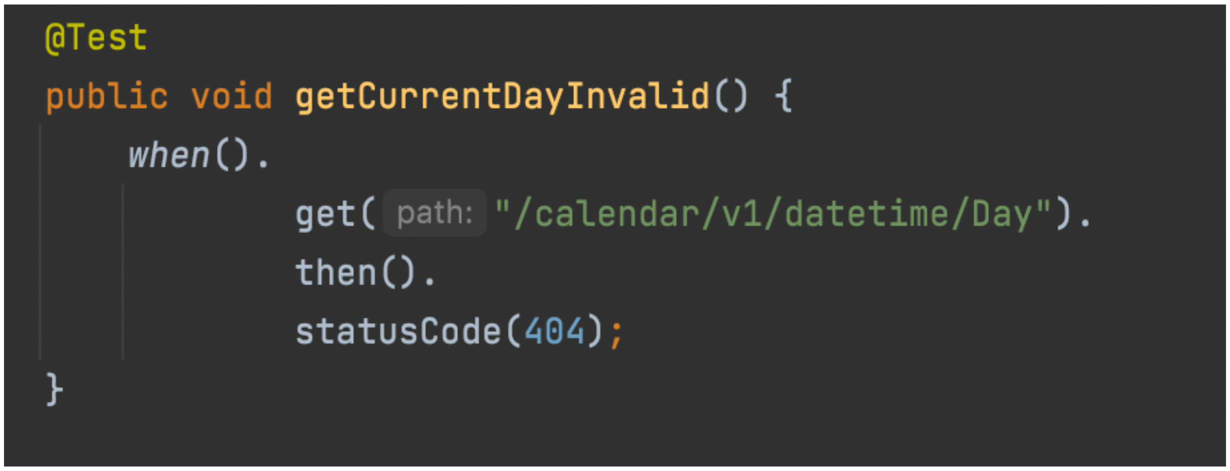
Modified test functions: getCurrentDateInvalid()

Reason:
The previous test case is inadequate to pass since it does not work as intended. This is actually a bug but I haven't notice it. Since we are writing the test cases before the implementation, this could happen because no test cases are supposed to work, and bugs are hard to find.



```
@Test
public void getCurrentDateInvalid() {
    DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("kkkk");
    LocalDateTime now = LocalDateTime.now();
    String trueDate = dateTimeFormatter.format(now);

    when().
            get( path: "/calendar/v1/datetime/date").
            then().
            statusCode(400).
            body( path: "date", equalTo( operand: "Invalid format."));
}
```

Figure 3 Function before changing



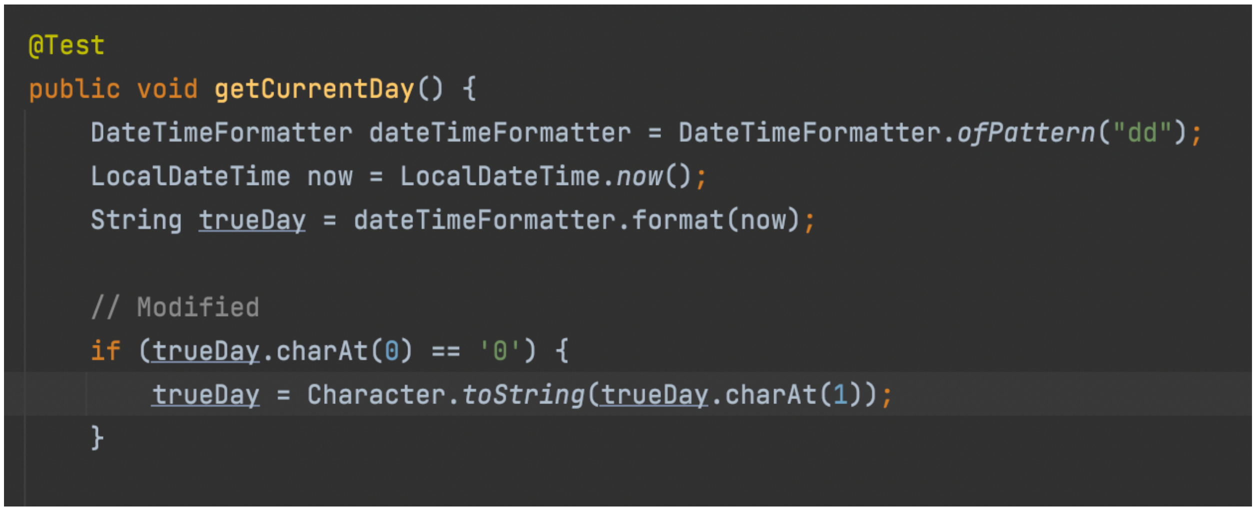
```
@Test
public void getCurrentDayInvalid() {
    when().
            get( path: "/calendar/v1/datetime/Day").
            then().
            statusCode(404);
}
```

Figure 4 Function after changing

Modified the format of expected outcome in getCurrentDay()

Modified test functions: getCurrentDay()

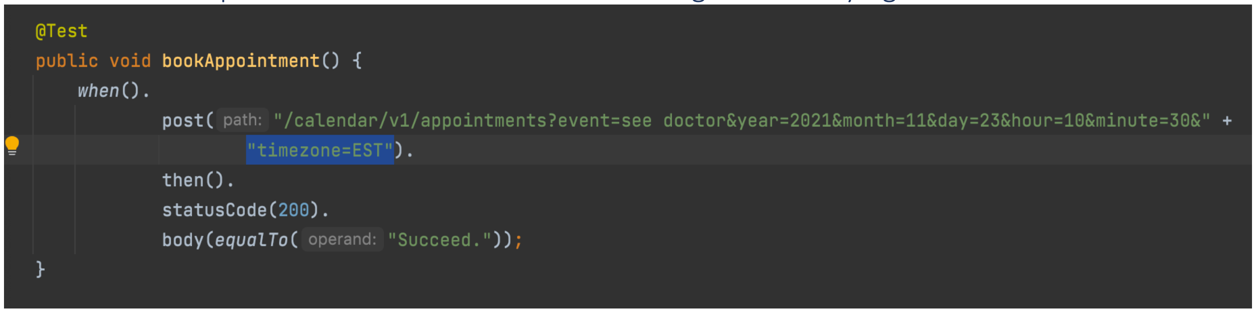
Reason:
When I was writing this part of the code, the date was 2 digits. But now, by my code, the reference day is 03. Although the api works well and returns 3, but the test case is expecting 03, so the test failed before. I should remember to create test cases not sensitive to time in the future.



```java
@Test
public void getCurrentDay() {
    DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("dd");
    LocalDateTime now = LocalDateTime.now();
    String trueDay = dateTimeFormatter.format(now);

    // Modified
    if (trueDay.charAt(0) == '0') {
        trueDay = Character.toString(trueDay.charAt(1));
    }
```

Figure 5 Modification in getCurrentDay()

Add a time zone parameter to the functions for adding and modifying events



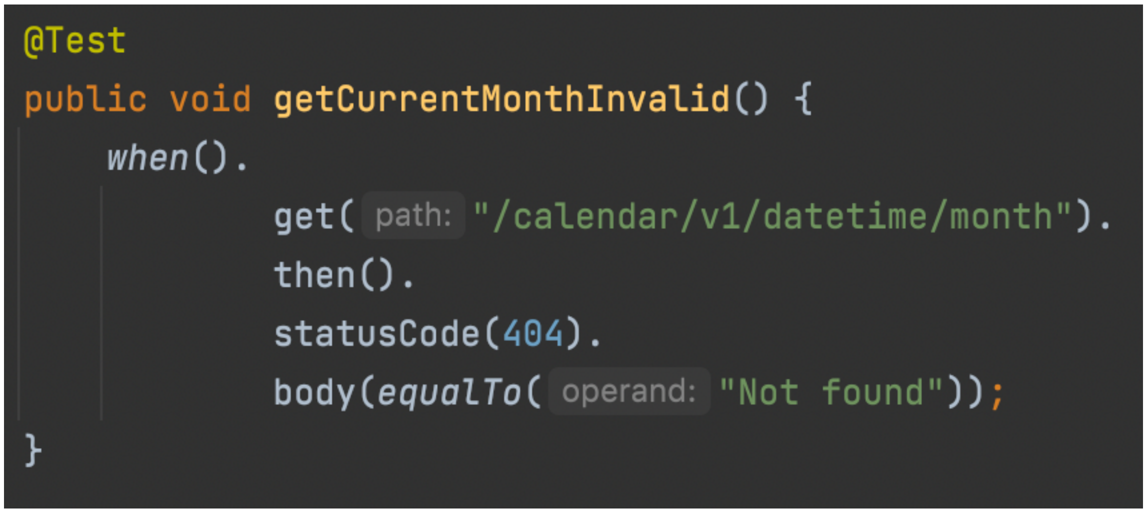
```java
@Test
public void bookAppointment() {
    when().
            post( path: "/calendar/v1/appointments?event=see doctor&year=2021&month=11&day=23&hour=10&minute=30&" +
                    "timezone=EST").
            then().
            statusCode(200).
            body(equalTo( operand: "Succeed."));
}
```

Figure 6 Modification example for adding a time zone parameter

Changing invalid test cases (404) for get functions

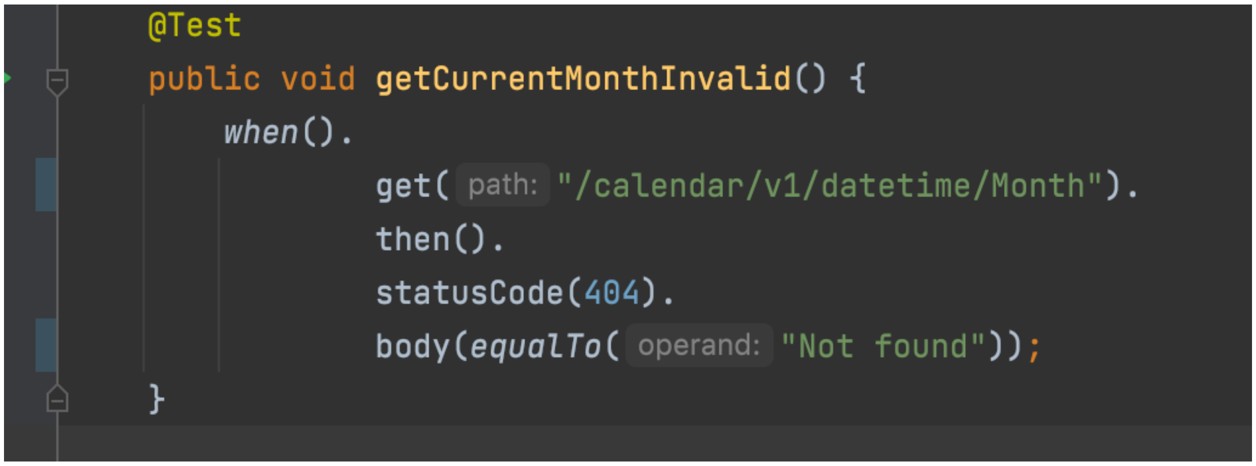
Modified test functions: getCurrentDay(), getCurrentMonth(), getCurrentYear()

Reason:
This is modified because those invalid cases are not testable for this assignment. Thus, I switch to the accessing of some empty resources as invalid cases. I should consider whether the test cases are testable when I am developing api in the furture.



```
@Test
public void getCurrentMonthInvalid() {
    when().
            get( path: "/calendar/v1/datetime/month").
            then().
            statusCode(404).
            body(equalTo( operand: "Not found"));
}
```

Figure 7 Example for invalid test case before modification



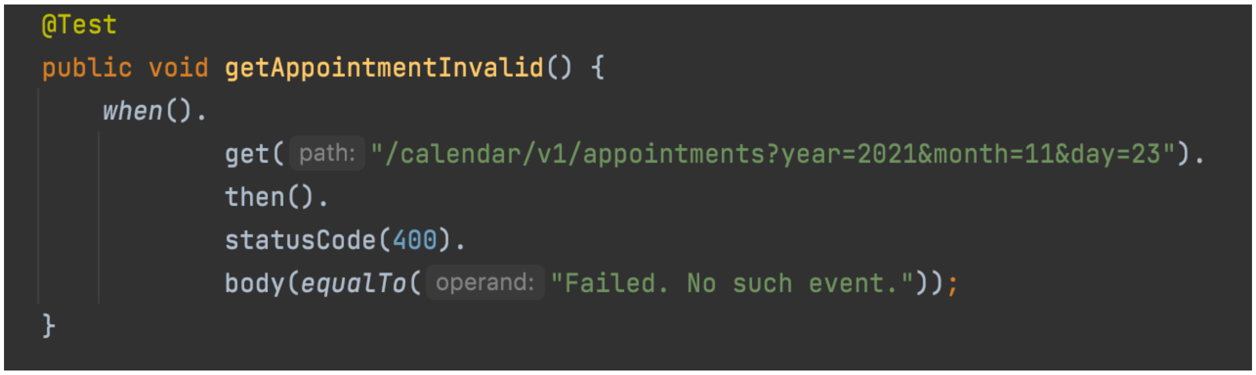
```
@Test
public void getCurrentMonthInvalid() {
    when().
            get( path: "/calendar/v1/datetime/Month").
            then().
            statusCode(404).
            body(equalTo( operand: "Not found"));
}
```

Figure 8 Example for invalid test case after modification

Modified getAppointmentInvalid()

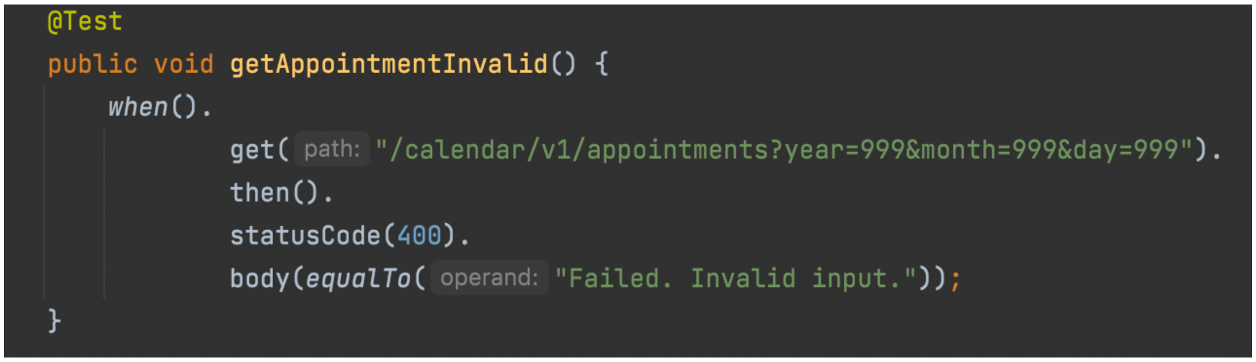
Modified test functions: getAppointmentInvalid()

Reason:
This is modified because when there is no event, the api should not return the error code, but should return empty event or some messages. This test case is modified to make it work as my previous (library-style) functions. I should be aware of the consistency of my design in the future.



```
@Test
public void getAppointmentInvalid() {
    when().
            get( path: "/calendar/v1/appointments?year=2021&month=11&day=23").
            then().
            statusCode(400).
            body(equalTo( operand: "Failed. No such event."));
}
```

Figure 9 getAppointmentInvalid() before modification



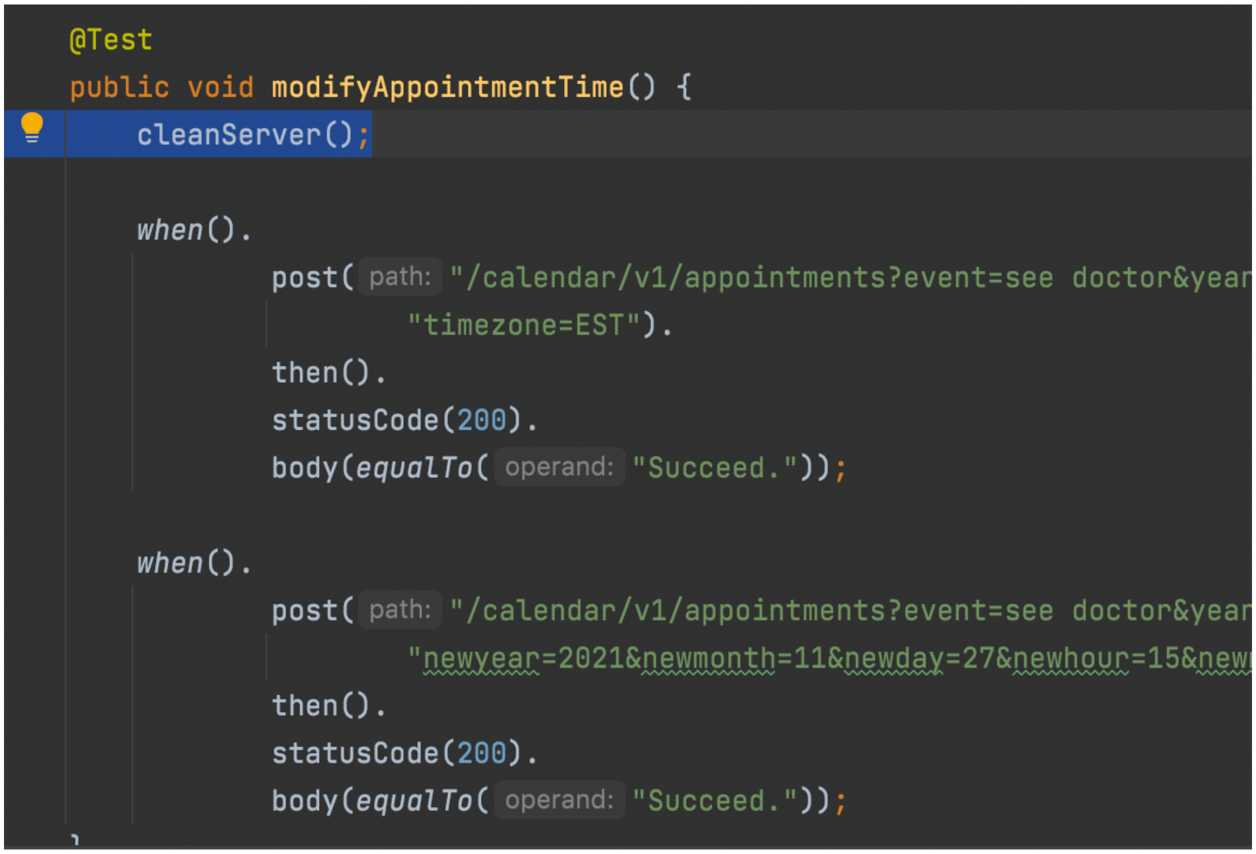
```
@Test
public void getAppointmentInvalid() {
    when().
            get( path: "/calendar/v1/appointments?year=999&month=999&day=999").
            then().
            statusCode(400).
            body(equalTo( operand: "Failed. Invalid input."));
}
```

Figure 10 getAppointmentInvalid() after modification

Clean the server before some tests

Modified test functions: modifyAppointmentTime(), modifyAppointmentTimeInvalid(), modifyAppointmentContent(), modifyAppointmentContentInvalid(), deleteAppointment(), deleteAppointmentInvalid(), getAppointment(), getAppointmentInvalid()

Reason:
This is modified because when there is no event, the api should not return the error code, but should return empty event or some messages. This test case is modified to make it work as my previous (library-style) functions. I should be aware of the consistency of my design in the future.



```
@Test
public void modifyAppointmentTime() {
    cleanServer();

    when().
            post( path: "/calendar/v1/appointments?event=see doctor&year
                    "timezone=EST").
            then().
            statusCode(200).
            body(equalTo( operand: "Succeed."));

    when().
            post( path: "/calendar/v1/appointments?event=see doctor&year
                    "newyear=2021&newmonth=11&newday=27&newhour=15&new
            then().
            statusCode(200).
            body(equalTo( operand: "Succeed."));
}
```

Figure 11 Example of cleaning server before testing

Q3. Design Principle

The most important design principle that I considered in this homework is the consistency. The steps I design are consistent. When the user is trying to add or modify the events, he/she will follow very similar procedures, which is post the events to the endpoint /calendar/v1/appointments but specifying different query parameters. Holding the consistency here makes the usage of the api more intuitive, since the user is mostly following some common pattern. Additionally, I also make the return of each endpoint consistent. The return values are all strings, and the format of the content is very similar. I did not use two words to describe same scenario, which does not make the user confused. The naming of the endpoints is also designed to be consistent, which follows lower case and is parallel when possible. For example, for …/datetime/day and …/datetime/month, their naming are intuitive and consistent, which makes the api easy to follow.