Part B of Hudl QA Interview Project — API Integration Testing

For this testing exercise, I will take an outside-in approach (as I have no access to a front-end or source code) and focus on building test cases around the API endpoints and their responses. In this stage of testing, we are mainly concerned with the end-user (coach) experience — how the UI makes service calls to the REST API and validating that the correct information has been registered on the back-end to present back to the end-user for each given workflow.

Couple of set-up assumptions are:
- API base and endpoint URLS are as defined in my setup functions
- Two data points have already been pre-loaded into system (for testing on specific endpoints). They are defined in the setUp function of the GetRequestTest class.

GET Request Tests
As an end-user of the schedule page, after the user has logged into their **hudl.com** account, they should be able to pull up the schedule page of the web app and be presented with a list of all of their scheduled games on the page. If they have not logged in properly, they will not be able to access this page.

Baseline / "happy path" cases
Get all games: Client sends a get request with correct credentials, receives status code 200 and the response body is a json representation of all of the games for that user.
End-user result - Coach is able to successfully view all of their scheduled games.

Get one game: Client sends a get request with a specific id, receives status code 200 and the response body is a json representation of the one game they requested.
End-user result - Coach is able to view information for one specific game.

Negative testing scenarios:
test_no_auth — client sends get request without authorization parameters, receives an Unauthorized 401 status code.
test_wrong_id — client sends get request with an ID query parameter, receives a 404 Not Found status code.

PUT Request Tests
As an end-user of the schedule page, the user should be able to edit the fields schedule entry and on submit, successfully view the new edited entry in their schedule.
However, if the user submits invalid inputs, they will be told to change their edits and submit them again.

Baseline coverage - Client sends a put request with desired changes as the request parameters to put endpoint, receives status code 200 and the updated data in json form as the response contents.
End-user result: Coach edits a game entry, and then it is presented back to them with the performed updates.

Negative test scenarios:
test_invalid_date - client tries to input a malformed date, receives 400 error with no changes to data.
test_invalid_struct - client tries to input a malformed input data, receives 400 errors with no changes to data.

POST Request Tests
As an end-user of the schedule page, the user should be able to add a new schedule entry, and be presented with a schedule with the new entry on the updated page.
However, if they try to add an entry with invalid inputs, the web app should decline these entries with a helpful tip about what to change in order to successfully add an entry.

Baseline coverage — Client sends a post request with new data in json form as the request parameters, receives back status code 200 and response body with the new game entry.
End-user result: Coach sees that their game entry has been successfully entered and added to the schedule.

Negative testing scenarios:
test_empty_body — client sends post request with empty JSON, returns 403 error for forbidden
test_invalid_date — client sends post request with JSON that has invalid date formatting, returns 403 forbidden

DELETE Request Tests
As an end-user of the schedule page, the user should be able to delete a schedule entry and be presented with a schedule that does not include the deleted game. Another use of this method could be reflected in past games.

Baseline coverage - Client sends a delete request to a specific gameId endpoint, receives status code 204 and no/empty data (dict) object as response
Coach sees that on deleting an entry, their game has been removed from the schedule.

Negative testing scenarios:
test_get_deleted_entry - client deletes a resource then tries to get it again. Should receive 404 Not Found error code.