**PART B: API PROJECT**

Contents

[Assumptions 1](#_Toc32158392)

[UI Description 1](#_Toc32158393)

[API Documentation 2](#_Toc32158394)

[Test Cases 3](#_Toc32158395)

[GET 3](#_Toc32158396)

[PUT 3](#_Toc32158397)

[POST 4](#_Toc32158398)

[DELETE 5](#_Toc32158399)

## Assumptions

* API uses json as the communication format
* A SQL database is used
* Example API requests based on use of Python requests library, HTTPBasicAuth() for API authorisation and json module for converting data to json
* gameType represents non-competitive or competitive represented by values of 0 or 1 used in the API
* categories represents a list of tags that can be used to describe the game

## UI Description

The following is a list of assumption I've made about the UI which influenced the API documentation and test cases I've written:

* UI displays a table of schedule information
* There is an add button for adding schedules
* Each row in the table has an edit and a delete icon
* Add, Edit and Delete are displayed as modal forms
* Games can either have a date in the past, present or future
* Each row represents a game displaying date, opponent, isHome, gameType and categories on the UI
* gameId, sqlId, opponentId are hidden on the UI
* When adding or editing a game:
  + The opponent can be selected from a dropdown selector field which is pre-populated
  + The date is represented by a field with a calendar selector
  + isHome is represented by a checkbox
  + gameType is represented by a dropdown with the options non-competitive or competitive
  + categories is represented by a text box using comma separated values distinguish between categories. It accepts alphanumeric values

## API Documentation

This describes the API based on assumptions I have made

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Method | Endpoint | Path Parameters | Request Body Parameters | Authentication | HTTP Response Codes |
| GET | /schedule |  |  | **Username** – *string, required*  **Password** - s*tring, required* | 200 – OK  401 – unauthorised access  404 – no data found  500 – internal server error |
| PUT | /schedule/{gameId} | **gameId** – *string, required* | **gameId** – *string, required (unique id)*  **date** – *string/date-time, required (can be in past, present or future)*  **opponent** – *string, required (picked from list of opponents)*  **isHome** – *Boolean, optional (true if at home, false if away)*  **gameType** – *intege,r, optional (0 for non-competitive, 1 for competitive)*  **categories** – *array, optional (comma separated list of alphanumeric values)* | **Username** – *string, required*  **Password** - s*tring, required* | 200 – OK  400 – bad request body  401 – unauthorised access  500 – internal server error |
| POST | /schedule/{gameId} | **gameId** – *string, required* | **gameId** – *string, required (unique id)*  **date** – *string/date-time, required (can be in past, present or future)*  **opponent** – *string, required (picked from list of opponents)*  **isHome** – *Boolean, optional (true if at home, false if away)*  **gameType** – *integer, optional (0 for non-competitive, 1 for competitive)*  **categories** – *array, optional (comma separated list of alphanumeric values)* | **Username** – *string, required*  **Password** - *string, required* | 200 – OK  400 – bad request body  401 – unauthorised access  404 – no data found |
| DELETE | /schedule |  |  | **Username** – *string, required*  **Password** - *string, required* | 204 – OK  401 – unauthorised access  404 – no data found |

## Test Cases

These test cases test the API based on the API documentation.

NOTE: For authorisation the username is ‘user’ and the password is ‘pwd’. The values are shown unencrypted

### GET

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Test Case | Steps | Expected Outcome | API Request |
| TC1 | Confirm UI is populated correctly when schedule data is in the database | 1. Populate ‘schedules’ table in database with valid game records 2. Open the schedule page | * Response code 200 returned * The schedule table is populated with each row representing a game. * date, opponent, isHome, gameType and categories are displayed | requests.get(url='.../schedules', auth= HTTPBasicAuth('user', 'pwd')) |
| TC2 | Confirm message is displayed on UI if no data is in the database | 1. Remove all schedules from the ‘schedules’ table in the database 2. Open the schedule page | * Response code 404 returned * Message is displayed in place of the table indicating that there is no data to display | requests.get(url='.../schedules', auth= HTTPBasicAuth('user', 'pwd')) |
| TC3 | Confirm not-authorised message is displayed if user isn’t authorized to access API | 1. Change the users API password 2. Open the schedule page | * Response code 401 is returned * A message indicating that the user’s access to this API is not authorised | requests.get(url='.../schedules', auth= HTTPBasicAuth('user', 'pwdX')) |
| TC4 | Confirm an internal server error message is displayed when there is an unexpected key stored in database | 1. Insert a game into the ‘schedules’ database table that contains an invalid key, e.g. replace opponent with opp 2. Open the schedule page | * Response code 500 is returned * A message indicating an internal server error has occurred | requests.get(url='.../schedules', auth= HTTPBasicAuth('user', 'pwd')) |

### PUT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Test Case | Steps | Expected outcome | API Request |
| TC1 | Confirm the UI is updated correctly when correct information is supplied to all editable fields | 1. Open the schedule page 2. Click the edit icon for game 123456 3. Change all fields using valid inputs 4. Click Update | * Response code 200 is returned * Schedule page displayed * Game 123456 displays the updated information | requests.put(url='.../schedule/123456', data=json.dumps({‘gameId’: ‘123456’, ‘date’: ‘2021-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |
| TC2 | Confirm the schedule is updated correctly when the selected date is in the past | 1. Open the schedule page 2. Click the edit icon for game 123456 3. Change the date to 2001-01-01 4. Click Update | * Response code 200 is returned * The schedule page is displayed * Game 123456 displays the updated date | requests.put(url='.../schedule/123456', data=json.dumps({‘gameId’: ‘123456’, ‘date’: ‘2001-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |
| TC3 | Confirm message is displayed on UI when non-alpha/numeric characters are used for the categories field | 1. Open the schedule page 2. Click the edit icon for game 123456 3. Enter !”£$%^& into the category field 4. Click Update | * Response code 400 is returned * A message is displayed indicating that the values supplied to the category field are invalid * Game 123456 is not updated | Requests.put(url='.../api/v1/schedules/123456', data=json.dumps({‘gameId’: ‘123456’, ‘date’: ‘2021-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘!”£$%^&’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |
| TC4 | Confirm message displayed on UI when a schedule is updated with an empty date value | 1. Open the schedule page 2. Click the edit icon for game 123456 3. Delete the date value 4. Click Update | * Response code 400 is returned * A message is displayed indicating that the value supplied to the date field is invalid * Game 123456 is not updated | requests.put(url='.../schedule/123456', data=json.dumps({‘gameId’: ‘123456’, ‘date’: ‘’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |
| TC5 | Confirm that updating a non-existent schedule displays a message on the UI | 1. Open the schedule page 2. Click the edit icon for game 654321 3. Delete game 654321 from the ‘schedules’ database table 4. On the UI change the opponent for the game 5. Click Update | * Response code 404 is returned * The schedule page is displayed * A message is displayed indicating that game 654321 cannot be found | requests.put(url='.../schedule/654321', data=json.dumps({‘gameId’: ‘654321’, ‘date’: ‘2021-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |
| TC6 | Confirm not-authorised message is displayed if user isn’t authorized to access API | 1. Open the schedule page 2. Click the edit icon for game 123456 3. Change the user’s API password 4. Change the opponent 5. Click Update | * Response code 401 is returned * A message indicating that the user’s access to this API is not authorised | requests.put(url='.../schedule/123456', data=json.dumps({‘gameId’: ‘123456’, ‘date’: ‘2021-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwdX')) |

### POST

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Test Case | Steps/Expected outcome |  | API Request |
| TC1 | Confirm that a newly added schedule is displayed on the UI when all fields are completed with correct information | 1. Open the schedule page 2. Click the add button 3. Complete all fields with valid content 4. Click Save | * Response code 200 is returned * Schedule page displayed * New game added to the table with correct information displayed | requests.post(url='.../schedule/', data=json.dumps({‘gameId’: ‘111111’, ‘date’: ‘2021-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |
| TC2 | Confirm a schedule can be added when the selected date is in the past | 1. Open the schedule page 2. Click the add button 3. Complete all fields with valid content and set a date of 2001-01-01 4. Click Save | * Response code 200 is returned * Schedule page displayed * New game added to the table with correct date displayed | requests.post(url='.../schedule/', data=json.dumps({‘gameId’: ‘111112’, ‘date’: ‘2001-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |
| TC3 | Confirm a message is displayed on the UI when a schedule is added with an empty date value | 1. Open the schedule page 2. Click the add button 3. Complete all fields with valid content except leave the date field empty 4. Click Save | * Response code 400 is returned * A message is displayed indicating that a date value must be supplied | requests.post(url='.../schedule/', data=json.dumps({‘gameId’: ‘111113’, ‘date’: ‘’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |
| TC4 | Confirm a message is displayed on the UI when a schedule is added with duplicate opponent and date information | 1. Open the schedule page 2. Click the add button 3. Complete all fields with valid content but set the opponent and the date to one that matches existing game 123456, e.g. opponent=’opponent’, date=’2021-01-01T19:00:00’ 4. Click Save | * Response code 400 is returned * A message is displayed indicating that a game with the same combination of selected opponent and date already exists | requests.post(url='.../schedule/', data=json.dumps({‘gameId’: ‘123456’, ‘date’: ‘2021-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |
| TC5 | Confirm not-authorised message is displayed if user isn’t authorized to access API | 1. Open the schedule page 2. Click the add button 3. Change the users API password 4. Complete all fields with valid content 5. Click Save | * Response code 401 is returned * A message indicating that the user’s access to this API is not authorised | requests.post(url='.../schedule/', data=json.dumps({‘gameId’: ‘111114’, ‘date’: ‘2021-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, auth= HTTPBasicAuth('user', 'pwd')) |

### DELETE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Test Case | Steps | Expected outcome | API Request |
| TC1 | Confirm that the deletion of a game removes the corresponding row from the UI | 1. Open the schedule page 2. Click the delete icon for game 123456 3. Click Confirm | * Response code 204 is returned * The schedule page is displayed * Game 123456 is not displayed in the table | requests.get(url='.../schedule/123456', auth= HTTPBasicAuth('user', 'pwd')) |
| TC2 | Confirm that deletion of a non-existent schedule displays a message on the UI | 1. Open the schedule page 2. Click the delete icon for game 111112 3. Delete the record with gameId equal to 111112 from the ‘schedules’ database table 4. Click Confirm | * Response code 404 is returned * The schedule page is displayed * A message is displayed indicating that the game 111112 cannot be found | requests.get(url='.../schedule/111112', auth= HTTPBasicAuth('user', 'pwd')) |
| TC3 | Confirm not-authorised message is displayed if user isn’t authorized to access API | 1. Open the schedule page 2. Click the delete icon for game 111113 3. Change the users API password 4. Click Confirm | * Response code 401 is returned * A message indicating that the user’s access to this API is not authorised | requests.get(url='.../schedule/111113', auth= HTTPBasicAuth('user', 'pwdX')) |