# PART B: API PROJECT

## Assumptions

* API uses json format
* A SQL database is used
* Example API requests based on use of Python requests library
* 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

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
* Games can either have a date in the past or in the 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:
  + 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

API Route: /api/v1/schedules/

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| API Method | Path Parameters | Body Parameters | Header Parameters | HTTP Response Codes |
| GET |  |  | ‘Authorization’: ‘access\_token myToken’ | 200 – OK  401 – Unauthorised access  404 – no data found  500 – internal server error |
| PUT | gameId - *string required* | gameId – *string required (unique id)*  date - *string required (can be in past 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)* | ‘Authorization’: ‘access\_token myToken’ | 200 – OK  400 – bad request body  401 – Unauthorised access  500 – internal server error |
| POST |  | gameId – *string required (unique id)*  date - *string required (can be in past 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)* | ‘Authorization’: ‘access\_token myToken’ | 200 – Valid response  400 – bad request body  401 – Unauthorised access  404 – no data found |
| DELETE | gameId - *string required* |  | ‘Authorization’: ‘access\_token myToken’ | 204 – OK  401 – Unauthorised access  404 – no data found |

## Test Cases

These test cases test the API based on the API documentation

### GET Method

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test Case | Steps/Expected outcome | How data is retrieved/display | Example Test API Request | Expected HTTP Response |
| TC1 | Confirm UI populated correctly when data in database | Steps:   1. Populate ‘schedules’ table in database with valid games 2. View schedules page   Expected Outcome:  A table is populated with each row representing a game. date, opponent, isHome, gameType and categories are displayed | Call made to the schedules GET method when the schedules page is loaded. This gets all data from the 'schedules' table in the database and returns the data in json format. The appropriate json data is then inserted into the schedules page | requests.get(url='.../api/v1/schedules/', headers={‘Authorization’: ‘access\_token myToken’}) | 200 |
| TC2 | Confirm message displayed if no data in database | Steps:   1. Remove all schedules from the ‘schedules’ table in the database 2. View schedules page   Expected Outcome:  A message is displayed in place of the table indicating that there is no data to display | Call made to schedules GET method when page is loaded. Service logic detects that no records exist in the schedules database table and returns a 404 status code. 404 is detected by the schedules page and message displayed | requests.get(url='.../api/v1/schedules/', headers={‘Authorization’: ‘access\_token myToken’}) | 404 |
| TC3 | Confirm message displayed if user not authorized to access API | Steps:   1. Change the users stored API token 2. View schedules page   Expected Outcome:  A message indicating that access to this API is unauthorised | Call made to schedules GET method when page is loaded. Service logic detects that the api authorisation credentials are invalid returns a 401 status code. 401 is detected by the schedules page and message displayed | requests.get(url='.../api/v1/schedules/', headers={‘Authorization’: ‘access\_token myBadToken’}) | 401 |
| TC4 | Confirm an internal server error message displayed when unexpected key stored in database | Steps:   1. Insert a schedule into the ‘schedules’ table in the database that contains an invalid key 2. View schedules page   Expected Outcome:  A message indicating an internal server error has occurred | Call made to schedules GET method when page is loaded. Service logic detects the bad key and returns a 500 status code. 500 is detected by the schedules page and message displayed | requests.get(url='.../api/v1/schedules/', headers={‘Authorization’: ‘access\_token myToken’}) | 500 |

### PUT Method

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test Case | Steps/Expected outcome | How data is retrieved/display | Example Test API Request | Expected HTTP Response |
| TC1 | Confirm the schedule is updated when correct information is supplied to all editable fields | Steps:   1. View schedules page 2. Click the edit icon for a game 3. Change all fields using valid inputs 4. Click Update   Expected Outcome:  The schedules page is displayed with the updated game reflecting the changes | Call is made to the schedules PUT method when update is clicked. The gameId is passed to the service logic. The put request contains new values in json format. The values are checked by the service logic. The gameId is used to find the game in the database and then updates it with the new values found in the json | 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’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, headers={‘Authorization’: ‘access\_token myToken’}) | 200 |
| TC2 | Confirm the schedule is updated correctly when selected date is in the past | Steps:   1. View the schedules page 2. Click the edit icon for a game 3. Change the date to one that is in the past 4. Click Update   Expected Outcome:  The schedules page is displayed with the updated game reflecting the changed date | Call is made to the schedules PUT method when update is clicked. The gameId is passed to the service logic. The put request contains new values in json format. The values are checked by the service logic. The gameId is used to find the game in the database and then updates it with the new values found in the json | requests.put(url='.../api/v1/schedules/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’, headers={‘Authorization’: ‘access\_token myToken’}) | 200 |
| TC3 | Confirm message displayed when non-alpha/numeric characters are used for categories | Steps:   1. View the schedules page 2. Click the edit icon for a game 3. Enter non-alpha/numeric characters into the category field 4. Click Update   Expected Outcome:  A message is displayed indicating that the values supplied to the category field are invalid | Call is made to the schedules PUT method when update is clicked. The put request contains new values in json format. The values are checked by the service logic, which determines that the value supplied is not valid and returns a 400 status code. 400 is detected by the edit game page and message displayed | 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’, headers={‘Authorization’: ‘access\_token myToken’}) | 400 |
| TC4 | Confirm message displayed when schedule is updated with an empty date value | Steps:   1. View the schedules page 2. Click the edit icon for a game 3. Delete the date value 4. Click Update   Expected Outcome:  A message is displayed indicating that a date value must be supplied | Call is made to the schedules PUT method when update is clicked. The put request contains new values in json format. The values are checked by the service logic, which determines that the value supplied is not valid and returns a 400 status code. 400 is detected by the edit game page and message displayed | requests.put(url='.../api/v1/schedules/123456', data=json.dumps({‘gameId’: ‘123456’, ‘date’: ‘’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, headers={‘Authorization’: ‘access\_token myToken’}) | 400 |
| TC5 | Confirm update of a non-existent schedule displays a message | Steps   1. View the schedules page 2. Click the edit icon for a game 3. Delete the game from the ‘schedules’ table in the database 4. Change the opponent 5. Click Update   Expected Outcome:  The schedules page is displayed and a message is displayed indicating that the game cannot be found | Call is made to the schedules PUT method when update is clicked. The gameId is passed to the service logic. The put request contains new values in json format. The service logic determines that the game does not exist by looking at the database and returns a 404 status code. 404 is detected by the edit game page and message displayed | 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’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, headers={‘Authorization’: ‘access\_token myToken’}) | 404 |
| TC6 | Confirm message displayed if user not authorized to access API | Steps:   1. View the schedules page 2. Click the edit icon for a game 3. Change the users stored API token 4. Change the opponent 5. Click Update   Expected Outcome  A message indicating that access to this API is unauthorised | Call made to schedules PUT method when page is loaded. Service logic detects that the api authorisation credentials are invalid returns a 401 status code. 401 is detected by the schedules page and message displayed | 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’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, headers={‘Authorization’: ‘access\_token myBadToken’}) | 401 |

### POST Method

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test Case | Steps/Expected outcome | How data is retrieved/display | Example Test API Request | Expected HTTP Response |
| TC1 | Confirm a newly added schedule is displayed in the table when all fields are completed with correct information | Steps:   1. View the schedules page 2. Click the add button 3. Complete all fields with valid content 4. Click Save   Expected Outcome  Schedules page displayed with new game added to the table | Call made to the POST method containing json representation of data. The service logic validates the data and inserts it into the ‘schedules’ database table. When the schedules page is re-loaded with a call to GET, the new game is displayed | requests.post(url='.../api/v1/schedules/', 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’, headers={‘Authorization’: ‘access\_token myToken’}) | 200 |
| TC2 | Confirm a schedule can be added when selected date is in the past | Steps:   1. View the schedules page 2. Click the add button 3. Complete all fields with valid content and set a date in the past 4. Click Save   Expected Outcome  Schedules page displayed with new game added to the table | Call made to the POST method containing json representation of data. The service logic validates the data and inserts it into the ‘schedules’ database table. When the schedules page is re-loaded with a call to GET, the new game is displayed | requests.post(url='.../api/v1/schedules/', data=json.dumps({‘gameId’: ‘111111’, ‘date’: ‘2015-01-01T19:00:00’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, headers={‘Authorization’: ‘access\_token myToken’}) | 200 |
| TC3 | Confirm message displayed when schedule is added with an empty date value | Steps:   1. View the schedules page 2. Click the add button 3. Complete all fields with valid content except leave the date field empty 4. Click Save   Expected Outcome  A message is displayed indicating that a date value must be supplied | Call made to the POST method containing json representation of data. The service logic validates the data, determines that the date value shouldn’t be empty and returns a 400 status code. 400 is detected by the add game page and message displayed | requests.post(url='.../api/v1/schedules/', data=json.dumps({‘gameId’: ‘111111’, ‘date’: ‘’, ‘opponent’: ‘TestOpponent', ‘isHome’: true, ‘gameType: 0’, ‘categories’: [‘cat\_a’, ‘cat\_2’]}), content\_type=’application/json’, headers={‘Authorization’: ‘access\_token myToken’}) | 400 |
| TC4 | Confirm a message is displayed when schedule is added with duplicate opponent and date information | Steps:   1. View the schedules page 2. Click the add button 3. Complete all fields with valid content but set the opponent and the date to one that matches and existing game 4. Click Save   Expected Outcome  A message is displayed indicating that a game with the selected opponent and date already exists | Call made to the POST method containing json representation of data. The service logic validates the data, determines that the an existing game contains the same opponent and date and returns a 400 status code. 400 is detected by the add game page and message displayed | requests.post(url='.../api/v1/schedules/', 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’, headers={‘Authorization’: ‘access\_token myToken’}) | 400 |
| TC5 | Confirm message displayed if user not authorized to access API | Steps:   1. View the schedules page 2. Click the add button 3. Change the users stored API token 4. Change the opponent 5. Click Update   Expected Outcome  A message indicating that access to this API is unauthorised | Call made to schedules POST method when page is loaded. Service logic detects that the api authorisation credentials are invalid returns a 401 status code. 401 is detected by the schedules page and message displayed | requests.post(url='.../api/v1/schedules/', 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’, headers={‘Authorization’: ‘access\_token myBadToken’}) | 401 |

### DELETE Method

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test Case | Steps/Expected outcome | How data is retrieved/display | Example Test API Request | Expected HTTP Response |
| TC1 | Confirm deletion of a schedule removes the row from the table | Steps:   1. View the schedules page 2. Click the delete icon for a game 3. Confirm deletion   Expected Outcome  The schedules page is displayed and the game is not displayed in the table | Call made to the DELETE method when confirm clicked. The gameId is used to determine the game to delete and is passed to the service logic which deletes the team from the database. The schedules page is then displayed which calls the GET API and displays the table without the deleted game | requests.get(url='.../api/v1/schedules/123456', headers={‘Authorization’: ‘access\_token myToken’}) | 204 |
| TC2 | Confirm deletion of a non-existent schedule displays a message | Steps:   1. View the schedules page 2. Click the delete icon for a game 3. Delete the game from the ‘schedules’ table in the database 4. Confirm deletion   Expected Outcome  The schedules page is displayed and a message is displayed indicating that the game cannot be found | Call is made to the DELETE method when confirm is clicked. The gameId is used to determine the game to delete. The service logic checks the database for the game and determines that the game doesn’t and returns a 404 status code. 404 is detected by the edit game page and message displayed | requests.get(url='.../api/v1/schedules/111111', headers={‘Authorization’: ‘access\_token myToken’}) | 404 |
| TC3 | Confirm message displayed if user not authorized to access API | Steps:   1. View the schedules page 2. Click the delete icon for a game 3. Change the users stored API token 4. Confirm deletion   Expected Outcome  A message indicating that access to this API is unauthorised | Call made to schedules DELETE method when page is loaded. Service logic detects that the api authorisation credentials are invalid returns a 401 status code. 401 is detected by the schedules page and message displayed | requests.get(url='.../api/v1/schedules/123456', headers={‘Authorization’: ‘access\_token myBadToken’}) | 401 |