SMP Project - Backend Test Plan

Developed by: Aishwary Sharma, Sarthak Daksh, Mohit Sharma, Vishesh Jain

Writing Tests in Django

In Django, tests are written using Python's unittest framework or Django's test case classes. These tests are typically placed in a tests.py file within each Django app. In our case, we have created a separate folder called tests within the backend code where all our test files are present.
Django's TestCase classes provide a set of useful methods for setting up test data, making assertions, and cleaning up after tests. These classes enable developers to define test cases that cover various scenarios and edge cases. The code contains over 150+ test cases for backend verifying Databases, API functionality, and client requests. Most of these are for:

- Testing Models: Models are the backbone of Django applications, and tests for models often involve creating instances of models, saving them to the database, and then asserting on their attributes or behavior.

- Testing Views: Views handle requests and responses in Django applications. Testing views typically involves making requests to views, either directly or through Django's test client, and then asserting on the response status code, content, or other attributes.

- Testing Forms: Django's forms handle user input validation and processing. Tests for forms involve creating instances of form classes, populating them with data, and then asserting on the validity of the form and its cleaned data.

Django provides a command-line utility, manage.py test, to run tests for all apps in a Django project or specific apps or test modules. This utility automatically discovers test modules and executes them, providing detailed output about test results and any failures or errors encountered.

Below is a short description of many of the test cases that have been created to verify various functionality. The table contains the name of the test case followed by what it is used to test for, a short description of the test, and which file is the test present in.

(This set of test cases was used for the initial development and delivery of the application. More test cases will be added as the application evolves.)

Test Cases for Back-End Testing

Sr. no.	Function name	Functionality Tested	Test Case description	File name test present in
1.	`test_attandance_views_get_attandance`	Valid url submission and valid function call	This test case is designed to test the URL routing and view function mapping in a Django application for retrieving attendance data.	tests_urls.py
2.	`test_attandance_views_update_attandance`	Valid url submission and valid function call	This test ensures that the URL routing for updating attendance data is correctly mapped to the update_attendance view function in the Django application.	tests_urls.py
3.	`test_auth_views_get_id_by_email`	Valid url submission and valid function call	This test verifies that the URL routing for retrieving user IDs by email is correctly mapped to the get_id_by_email view function, ensuring proper handling of user authentication.	tests_urls.py
4.	`test_forms_views_get_form_response`	Valid url submission and valid function call	This test validates that the URL routing for fetching form responses is correctly mapped to the get_form_response view function, essential for managing form submissions.	tests_urls.py
5.	`test_forms_views_get_form_status`	Valid url submission and valid function call	This test ensures that the URL routing for obtaining form statuses is accurately linked to the get_form_status view function, crucial for querying form status information.	tests_urls.py
6.	`test_forms_views_update_form_status`	Valid url submission and valid function call	This test confirms that the URL routing for updating form statuses is accurately linked to the update_form_status view function, necessary for modifying form status records.	tests_urls.py
7.	`test_forms_views_submit_consent_form`	Valid url submission and valid function call	This test checks that the URL routing for submitting consent forms is appropriately linked to the submit_consent_form view function, vital for managing consent form submissions.	tests_urls.py
8.	`test_forms_views_mentee_filled_feedback`	Valid url submission and valid function call	This test ensures that the URL routing for retrieving feedback from mentees is correctly associated with the mentee_filled_feedback view function, crucial for gathering	tests_urls.py

			mentee feedback data.	
9.	`test_forms_views_send_consent_form`	Valid url submission and valid function call	This test verifies that the URL routing for sending consent forms is accurately linked to the send_consent_form view function, essential for handling consent form distribution.	tests_urls.py
10.	`test_mail_views_get_mail_subject_and_body`	Valid url submission and valid function call	This test validates that the URL routing for obtaining email subjects and bodies is correctly linked to the get_mail_subject_and_body view function, crucial for managing email content retrieval.	tests_urls.py
11.	`test_meeting_views_add_meeting`	Valid url submission and valid function call	This test ensures that the URL routing for adding meetings is accurately linked to the add_meeting view function, necessary for handling meeting creation.	tests_urls.py
12.	`test_meeting_views_edit_meeting_by_id`	Valid url submission and valid function call	This test verifies that the URL routing for editing meetings by ID is correctly mapped to the edit_meeting_by_id view function, essential for updating meeting details.	tests_urls.py
13.	`test_meeting_views_delete_meeting_by_id`	Valid url submission and valid function call	This test confirms that the URL routing for deleting meetings by ID is accurately linked to the delete_meeting_by_id view function, necessary for removing specific meeting records.	tests_urls.py
14.	`test_meeting_views_get_meetings`	Valid url submission and valid function call	This test checks that the URL routing for fetching meetings is correctly linked to the get_meetings view function, crucial for retrieving meeting data.	tests_urls.py
15.	`test_mentee_views_add_mentee`	Valid url submission and valid function call	This test ensures that the URL routing for adding mentees is accurately linked to the add_mentee view function, vital for managing mentee registration.	tests_urls.py
16.	`test_mentee_views_edit_mentee_by_id`	Valid url submission and valid function call	This test validates that the URL routing for editing mentees by ID is correctly mapped to the edit_mentee_by_id view function, necessary for updating mentee details.	tests_urls.py
17.	`test_mentee_views_get_all_mentees`	Valid url submission and valid function call	This test confirms that the URL routing for retrieving all mentees is accurately linked to the get_all_mentees view function, crucial for fetching mentee records.	tests_urls.py

18.	`test_mentee_views_upload_CSV`	Valid url submission and valid function call	This test checks that the URL routing for uploading CSV files is correctly linked to the upload_CSV view function, essential for handling CSV file uploads.	tests_urls.py
19.	`test_mentee_views_get_mentee_by_id`	Valid url submission and valid function call	This test ensures that the URL routing for fetching mentees by ID is accurately linked to the get_mentee_by_id view function, vital for retrieving specific mentee details.	tests_urls.py
20.	`test_mentee_views_delete_all_mentees`	Valid url submission and valid function call	This test verifies that the URL routing for deleting all mentees is correctly mapped to the delete_all_mentees view function, necessary for bulk mentee deletion.	tests_urls.py
21.	`test_mentee_views_delete_mentee_by_id`	Valid url submission and valid function call	This test confirms that the URL routing for deleting mentees by ID is accurately linked to the delete_mentee_by_id view function, crucial for removing specific mentee records.	tests_urls.py
22.	`test_mentor_views_add_mentor`	Valid url submission and valid function call	This test checks that the URL routing for adding mentors is accurately linked to the add_mentor view function, essential for managing mentor registration.	tests_urls.py
23.	`test_mentor_views_edit_mentor_by_id`	Valid url submission and valid function call	This test ensures that the URL routing for editing mentors by ID is accurately mapped to the edit_mentor_by_id view function, vital for updating mentor details.	tests_urls.py
24.	`test_mentor_views_add_candidate`	Valid url submission and valid function call	This test verifies that the URL routing for adding candidates is correctly linked to the add_candidate view function, crucial for managing candidate registration.	tests_urls.py
25.	`test_mentor_views_get_all_mentors`	Valid url submission and valid function call	This test confirms that the URL routing for retrieving all mentors is accurately linked to the get_all_mentors view function, essential for fetching mentor records.	tests_urls.py
26.	`test_mentor_views_get_mentor_by_id`	Valid url submission and valid function call	This test validates that the URL routing for fetching mentors by ID is accurately linked to the get_mentor_by_id view function, necessary for retrieving specific mentor details.	tests_urls.py

27.	`test_mentor_views_delete_all_mentors`	Valid url submission and valid function call	This test ensures that the URL routing for deleting all mentors is correctly mapped to the delete_all_mentors view function, vital for bulk mentor deletion.	tests_urls.py
28.	`test_mentor_views_delete_mentor_by_id`	Valid url submission and valid function call	This test checks that the URL routing for deleting mentors by ID is accurately linked to the delete_mentor_by_id view function, crucial for removing specific mentor records.	tests_urls.py
29.	`test_mmpairs_views_create_mentor_mentee_pairs`	Valid url submission and valid function call	This test confirms that the URL routing for creating mentor-mentee pairs is accurately linked to the create_mentor_mentee_pairs view function, necessary for managing mentor-mentee pairings.	tests_urls.py
30.	`test_views_index`	Valid url submission and valid function call	This test case validates that the root URL 'home' is correctly mapped to the 'index' view function, ensuring proper routing configuration.	tests_urls.py
31.	`test_attandance_views_get_attandance_POST`	Valid HTTP Post request submission	Verifies that the 'getAttendance' endpoint correctly handles POST requests, ensuring it returns either a successful (200) or a client error (400) status code.	tests_views.py
32.	`test_attandance_views_update_attandance`	Valid HTTP Post request submission	Tests the 'updateAttendance' endpoint to ensure it handles POST requests properly, expecting either a successful (200) or a client error (400) status code.	tests_views.py
33.	`test_auth_views_get_id_by_email`	Valid HTTP Post request submission	Validates that the 'getIdByEmail' endpoint correctly responds to POST requests with a successful (200) status code, facilitating user authentication.	tests_views.py
34.	`test_forms_views_get_form_response`	Valid HTTP Post request submission	Ensures that the 'getFormResponse' endpoint appropriately handles POST requests, returning a successful (200), client error (400), or resource not found (404) status code.	tests_views.py
35.	`test_forms_views_get_form_status`	Valid HTTP Post request submission	Validates that the 'getFormStatus' endpoint properly responds to POST requests with either a successful (200) or a client error (400) status code, essential for querying form status information.	tests_views.py

36.	`test_forms_views_update_form_status`	Valid HTTP Post request submission	Tests the 'updateFormStatus' endpoint to verify its handling of POST requests, expecting either a successful (200) or a client error (400) status code.	tests_views.py
37.	`test_forms_views_submit_consent_form`	Valid HTTP Post request submission	Checks that the 'submitConsentForm' endpoint appropriately responds to POST requests with a successful (200) status code, crucial for managing consent form submissions.	tests_views.py
38.	`test_forms_views_mentee_filled_feedback`	Valid HTTP Post request submission	Validates that the 'menteeFilledFeedback' endpoint correctly handles POST requests, ensuring it returns a successful (200) status code for gathering mentee feedback data.	tests_views.py
39.	`test_forms_views_send_consent_form`	Valid HTTP Post request submission	Verifies that the 'sendConsentForm' endpoint properly handles POST requests, expecting either a successful (200) or a client error (400) status code for consent form distribution.	tests_views.py
40.	`test_mail_views_get_mail_subject_and_body`	Valid HTTP Post request submission	Ensures that the 'getMailSubjectAndBody' endpoint appropriately handles POST requests, returning a successful (200), client error (400), or resource not found (404) status code.	tests_views.py
41.	`test_meeting_views_add_meeting`	Valid HTTP Post request submission	Validates that the 'addMeeting' endpoint correctly handles POST requests, ensuring it returns a successful (200) status code for meeting creation.	tests_views.py
42.	`test_meeting_views_edit_meeting_by_id`	Valid HTTP Post request submission	Tests the 'editMeetingById' endpoint to verify its handling of POST requests, expecting a successful (200) status code for updating meeting details.	tests_views.py
43.	`test_meeting_views_delete_meeting_by_id`	Valid HTTP Post request submission	Verifies that the 'deleteMeetingById' endpoint properly responds to POST requests with a successful (200) status code, essential for removing specific meeting records.	tests_views.py
44.	`test_meeting_views_get_meetings`	Valid HTTP Post request submission	Checks that the 'getMeetings' endpoint appropriately handles POST requests, returning a successful (200) status code for fetching meeting data.	tests_views.py
45.	`test_mentee_views_add_`	Valid HTTP Post	Ensures that the 'addMentee' endpoint correctly responds to	tests_views.py

	mentee	request submission	POST requests with a successful (200) status code, facilitating mentee registration.	
46.	test_mentee_views_upload_csv	Valid HTTP Post request submission	Validates that the 'uploadCSV' endpoint properly handles POST requests, expecting either a successful (200), client error (400), or method not allowed (405) status code for CSV file uploads.	tests_views.py
47.	test_mentee_views_edit_mentee_by_id	Valid HTTP Post request submission	Verifies that the 'editMenteeById' endpoint correctly handles POST requests, ensuring it returns a successful (200) status code for updating mentee details.	tests_views.py
48.	test_mentee_views_get_all_mentees	Valid GET request submission	Tests the 'getAllMentees' endpoint to verify its handling of GET requests, expecting a successful (200) status code for retrieving all mentee records.	tests_views.py
49.	test_mentee_views_get_mentee_by_id	Valid GET request submission	Checks that the 'getMenteeById' endpoint appropriately responds to GET requests with a successful (200) status code, facilitating retrieval of specific mentee details.	tests_views.py
50.	test_mentee_views_delete_all_mentees	Valid GET request submission	Validates that the 'deleteAllMentees' endpoint properly handles GET requests, expecting a successful (200) status code for bulk mentee deletion.	tests_views.py
51.	test_mentee_views_delete_mentee_by_id	Valid HTTP Post request submission	Verifies that the 'deleteMenteeById' endpoint correctly handles POST requests, ensuring it returns a successful (200) status code for removing specific mentee records.	tests_views.py
52.	test_mentor_views_add_mentor	Valid HTTP Post request submission	Ensures that the 'addMentor' endpoint correctly responds to POST requests with a successful (200) status code, facilitating mentor registration.	tests_views.py
53.	test_mentor_views_edit_mentor_by_id	Valid HTTP Post request submission	Validates that the 'editMentorById' endpoint properly handles POST requests, expecting a successful (200) status code for updating mentor details.	tests_views.py
54.	test_mentor_views_add_candidate	Valid HTTP Post request submission	Checks that the 'addCandidate' endpoint appropriately responds to POST requests with a successful (200) status code, essential for managing candidate registration.	tests_views.py

55.	`test_mentor_views_get_all_mentors`	Valid GET request submission	Verifies that the 'getAllMentors' endpoint properly handles GET requests, expecting a successful (200) status code for retrieving all mentor records.	tests_views.py
56.	`test_mentor_views_get_mentor_by_id`	Valid HTTP Post request submission	Validates that the 'getMentorById' endpoint correctly responds to POST requests with a successful (200) status code, facilitating retrieval of specific mentor details.	tests_views.py
57.	`test_mentor_views_delete_all_mentors`	Valid GET request submission	Ensures that the 'deleteAllMentors' endpoint correctly handles GET requests, expecting a successful (200) status code for bulk mentor deletion.	tests_views.py
58.	`test_mentor_views_delete_mentor_by_id`	Valid HTTP Post request submission	Verifies that the 'deleteMentorById' endpoint properly responds to POST requests, ensuring it returns a successful (200) status code for removing specific mentor records.	tests_views.py
59.	`test_mmpairs_views_create_mentor_mentee_pairs`	Valid HTTP Post request submission	Checks that the 'createMentorMenteePair' endpoint appropriately handles POST requests, returning either a successful (200) or a client error (400) status code for managing mentor-mentee pairings.	tests_views.py
60.	`test_add_candidate_success`	Adding candidate to the database	This test verifies the successful addition of a candidate by sending a POST request with valid candidate data. It checks that the endpoint returns a status code of 200 and a success message. Additionally, it ensures that the candidate data is correctly stored in the database. This test checks the behavior of the 'addCandidate' endpoint when receiving a GET request, which should be invalid for this endpoint. It verifies that the endpoint returns a status code of 200 and an error message indicating an invalid request method.	db_test_addCandidate.py
61.	`test_add_meeting`	Adding new meeting to the database	This test case verifies the successful addition of a meeting by sending a POST request with valid meeting data. It checks that the endpoint returns a status code of 200, adds the meeting to the database, and returns the expected success message. Additionally, it ensures that attempting to add the same meeting again results in an appropriate error response.	db_test_addMeeting.py

			This test checks the behavior of the 'addMeeting' endpoint when receiving a GET request, which should be invalid for this endpoint. It verifies that the endpoint returns a status code of 200 and an error message indicating an invalid request method.	
62.	`test_add_mentee_success`	Adding new mentee to the database	This test verifies the successful addition of a mentee by sending a POST request with valid mentee data. It checks that the endpoint returns a status code of 200 and a success message. Additionally, it ensures that the mentee data is correctly stored in the database with the associated mentor.	db_test_addMentee.py
63.	`test_add_mentee_existing_mentee`	Checking to add existing mentee	This test checks the behavior of the 'addMentee' endpoint when attempting to add a mentee with an existing ID. It sends a POST request with conflicting mentee data and verifies that the endpoint returns a status code of 200 and an error message indicating the existing mentee.	db_test_addMentee.py
64.	`test_add_mentee_invalid_method`	Checking HTTP GET response to add a new mentee	This test ensures that the 'addMentee' endpoint correctly handles GET requests, which should be invalid for this endpoint. It sends a GET request and verifies that the endpoint returns a status code of 200 and an error message indicating an invalid request method.	db_test_addMentee.py
65.	`test_create_mentor_mentee_pairs`	Adding mentor mentee pairs to the database	This method sets up test data by creating instances of 'Mentee' and 'Candidate' objects with predefined attributes. It prepares the database with initial data required for testing the functionality of creating mentor-mentee pairs.This test case verifies the functionality of creating mentor-mentee pairs by sending a POST request to the 'createMentorMenteePair' endpoint. It checks that the response status code is 200, confirms that mentor-mentee pairs are successfully created, and ensures that the response message matches the expected message.	db_test_createMentorMenteePairs.py
66.	`test_create_mentor_mentee_pair_invalid_method`	Checking HTTP GET response to add new pair to database	This test checks the behavior of the 'createMentorMenteePair' endpoint when receiving a GET request, which should be invalid for this endpoint. It verifies that the endpoint returns a status code of 200 and an error message indicating an invalid request method.	db_test_createMentorMenteePairs.py
67.	`test_delete_meeting_by`	Check POST request	This method sets up test data by creating an instance of the	db_test_deleteMeeti

	`_id`	and Deleting meeting from Database	'Meetings' model with predefined attributes. It prepares the database with initial data required for testing the functionality of deleting a meeting by its ID.This test case verifies the functionality of deleting a meeting by its ID. It sends a POST request to the 'deleteMeetingById' endpoint with the ID of the meeting to be deleted. It checks that the response status code is 200, confirms that the meeting is successfully deleted from the database, and ensures that the response message matches the expected message.	ngById.py
68.	`test_delete_meeting_invalid_method`	Checking HTTP GET request to delete id from database	This test checks the behavior of the 'deleteMeetingById' endpoint when receiving a GET request, which should be invalid for this endpoint. It verifies that the endpoint returns a status code of 200 and an error message indicating an invalid request method.	db_test_deleteMeetingById.py
69.	`test_get_request_success_message`	Checking valid json response on Valid GET request	This method sets up test data by creating instances of 'Candidate' and 'Mentee' objects with predefined attributes. It prepares the database with initial data required for testing the functionality of deleting a mentee by its ID. This test case verifies the behavior of the 'deleteMenteeById' endpoint when receiving a POST request. It sends a POST request with the ID of the mentee to be deleted and checks that the endpoint returns a status code of 200 and a success message indicating the deletion of database entries.	db_test_deleteMeteeById.py
70.	`test_non_get_request_error_message`	Checking json response on invalid GET request	This test checks the behavior of the 'deleteMenteeById' endpoint when receiving a non-GET request, which should be invalid for this endpoint. It sends a GET request and verifies that the endpoint returns a status code of 200 and an error message indicating an invalid request method.	db_test_deleteMeteeById.py
71.	`test_edit_meeting_by_id`	Changing meeting information from database	This method sets up test data by creating an instance of the 'Meetings' model with predefined attributes. It prepares the database with initial data required for testing the functionality of editing a meeting by its ID.This test case verifies the functionality of editing a meeting by its ID. It sends a POST request to the 'editMeetingById' endpoint with the ID of the meeting to be edited and the updated meeting data. It checks that the response status code is 200, confirms that the meeting is successfully edited in the database, and ensures that the response message indicates successful data addition.	db_test_editMeetingById.py

72.	`test_edit_meeting_invalid_method`	Cehcking GET response on editing meeting	This test checks the behavior of the 'editMeetingById' endpoint when receiving a GET request, which should be invalid for this endpoint. It verifies that the endpoint returns a status code of 200 and an error message indicating an invalid request method.	db_test_editMeetingById.py
73.	`test_get_form_response`	Checking Valid POST request and retrieved database	This method sets up test data by creating instances of 'Candidate', 'Mentee', 'FormResponses', and 'FormStatus' objects with predefined attributes. It prepares the database with initial data required for testing the functionality of getting form responses, getting form status, and updating form status.	db_test_formStatus.py
74.	`test_get_form_status`	Check form is retrieved fromm the database correctly	This test verifies the functionality of getting form status. It sends a POST request to the 'getFormStatus' endpoint and checks that the response status code is 200. It ensures that form status is retrieved successfully and that the data matches the expected format and content.	db_test_formStatus.py
75.	`test_update_form_status`	Check if the form is updated in the database correctly	This test case verifies the functionality of updating form status. It sends a POST request to the 'updateFormStatus' endpoint with the form ID and updated form status. It checks that the response status code is 200 and ensures that the form status is updated successfully in the database.	db_test_formStatus.py
76.	`test_get_attendance`	Check valid attendance details added to database	This method sets up test data by creating instances of 'Admin', 'Candidate', 'Mentee', 'Meetings', and 'Attendance' objects with predefined attributes. It prepares the database with initial data required for testing the functionality of getting attendance for a meeting.This test case verifies the functionality of getting attendance for a meeting. It sends a POST request to the 'getAttendance' endpoint with the meeting ID and checks that the response status code is 200. It ensures that the response contains the correct attendance details for the specified meeting, including the IDs, names, emails, and attendance status of the attendees.	db_test_getAttendance.py
77.	`test_get_attendance_invalid_method`	Check GET response to retrieve attendance data	This test verifies the handling of invalid HTTP methods for the 'getAttendance' endpoint. It sends a GET request, which is not allowed for this endpoint, and checks that the response status code is 400 (Bad Request). It ensures that an appropriate error message is returned in the response JSON indicating the invalid request method.	db_test_getAttendance.py

78.	`test_get_meetings`	Checking valid added meetings,along with updates and correct categorisation of meetings	This method sets up test data by creating instances of 'Candidate', 'Mentee', and 'Meetings' objects with predefined attributes. It prepares the database with initial data required for testing the functionality of retrieving meetings for a mentor.: This test case verifies the functionality of retrieving meetings for a mentor. It sends a POST request to the 'getMeetings' endpoint with the mentor's ID and role, then checks that the response status code is 200. It ensures that the response contains two categories of meetings: 'previousMeeting' and 'upcomingMeeting'. It validates the correctness of the retrieved meetings' details, including meeting ID, scheduler ID, title, date, time, attendees, mentor branches, and description.	db_test_getMeetings.py
79.	`test_get_meetings_invalid_method`	Checking GET response to add meeting to database	This test verifies the handling of invalid HTTP methods for the 'getMeetings' endpoint. It sends a GET request, which is not allowed for this endpoint, and checks that the response status code is 200. It ensures that an appropriate error message is returned in the response JSON indicating the invalid request method.	db_test_getMeetings.py
80.	`test_submit_consent_form`	Adding consent form to database and get valid response message	This method sets up test data by creating an instance of the 'Candidate' object with predefined attributes. It prepares the database with initial data required for testing the functionality of submitting consent forms and mentee feedback.This test case verifies the functionality of submitting a consent form. It sends a POST request to the 'submitConsentForm' endpoint with the candidate's ID and consent questionnaire responses, then checks that the response status code is 200. It ensures that the consent form is updated successfully in the database with the new image source and size. Additionally, it validates that the response message indicates successful submission of the consent form.	db_test_submitForms.py
81.	`test_mentee_filled_feedback`	Add submitted mentee feedback	This test verifies the functionality of submitting mentee feedback. It sends a POST request to the 'menteeFilledFeedback' endpoint with the candidate's ID, mentor's ID, mentor's name, and feedback questionnaire responses, then checks that the response status code is 200. It ensures that the feedback form is stored successfully in the database with the corresponding data. Moreover, it validates that the response message confirms the successful submission of mentee feedback.	db_test_submitForms.py

82.	`test_update_attendance`	Update attendance in database and return valid response	This method sets up test data by creating instances of the 'Mentee' and 'Meetings' objects with predefined attributes. It prepares the database with initial data required for testing the functionality of updating attendance for a meeting. This test case verifies the functionality of updating attendance for a meeting. It first sends a POST request to the 'updateAttendance' endpoint with the meeting ID and attendee details indicating their presence. It checks that the response status code is 200 and confirms that the attendance is updated successfully in the database for the specified meeting and attendee. Additionally, it validates that the response message indicates successful attendance update. Next, it sends another POST request to the 'updateAttendance' endpoint with the same meeting ID but marks the attendee as absent (attendance=0). It ensures that the response status code is 200 and verifies that the attendance is updated to 0 (absent) successfully in the database. Moreover, it validates that the response message confirms the successful attendance update for the absence scenario.	db_test_updateAttendance.py
83.	`test_update_attendance_invalid_method`	Check Valid GET response to update database	This test ensures that sending a GET request to the 'updateAttendance' endpoint results in an error. It checks that the response status code is 200 and confirms that the response message indicates an invalid request method.	db_test_updateAttendance.py
84.	`test_add_candidate_success`	Check if the candidate is actually added to the database	This test case, test_add_candidate_success, verifies the successful addition of a candidate to the system. It sends a POST request to the endpoint responsible for adding candidates (reverse('addCandidate')) with a JSON payload containing candidate details. After receiving a response with a status code of 200 (indicating success), the test checks if the response message matches the expected value, confirming that the data was added successfully. Furthermore, the test queries the database to ensure that the candidate's information matches the provided data. It verifies attributes such as name, email, department, year, score, status, and imgSrc to ensure that the candidate was correctly stored in the database with the provided details.	test_views.py

85.	`test_add_candidate_missing_required_field`	Test if incomplete database is added	This test case, test_add_candidate_missing_required_field, ensures that the system correctly handles the scenario where a required field (name) is missing when adding a candidate. It sends a POST request to the endpoint responsible for adding candidates (reverse('addCandidate')) with a JSON payload containing candidate details but without the 'name' field. After sending the request, the test expects a response with a status code of 400, indicating a bad request due to the missing required field. Additionally, it verifies that the response contains an error message indicating the missing 'name' field. This ensures that the system correctly detects and reports missing required fields when adding candidates.	test_views.py
86.	`test_add_candidate_invalid_method`	Checks if the code performs correctly on failure of get request	This test case, test_add_candidate_invalid_method, verifies the behavior of the system when an invalid HTTP method (GET in this case) is used to access the endpoint responsible for adding candidates (reverse('addCandidate')). It sends a GET request to the addCandidate endpoint and expects a response with a status code of 200, indicating that the request was received by the server. However, since the method is not allowed for this endpoint, the response should include a message indicating that the request method is invalid. This test ensures that the system correctly handles and responds to requests made with an invalid HTTP method, helping to maintain the integrity and security of the application's endpoints.	test_views.py
87.				