Backend Test Suite

for

Kraken

Last modified: Mar 28, 2024

Test Case ID	Test Description	Test Steps	Expected Results
DB-001	Verify the getDb function returns a database object.	1. Call the getDb function.	1. A database object is returned.
DB-002	Verify toObjectId correctly handles a valid object ID.	1. Provide a valid object ID string ("5f6e7d5f7e3f5e7d5f7e3f5e") to the toObjectId function.	1. A valid ObjectId is returned.
DB-003	Verify toObjectId raises an HTTPException for an invalid ID.	1. Provide an invalid object ID string ("123") to the toObjectId function.	1. An HTTPException is raised.
MIL-001	Verify successful milestone creation	1. Create a user. 2. Create a project with the user. 3. Call the createMilestone endpoint with valid data.	1. Status code 200 is returned. 2. Response contains newly created milestone data.
MIL-002	Verify creation fails if the project does not exist	1. Create a user. 2. Call the createMilestone endpoint with valid data and a non-existent project ID.	1. Status code 404 is returned.
MIL-003	Verify creation fails if the user doesn't have access to the project	1. Create two users (user and otherUser). 2. Create a project with user. 3. Call the createMilestone endpoint with otherUser and the project ID.	1. Status code 403 is returned.
MIL-004	Verify creation fails with a database error	1. Mock insert_one to return acknowledged=False 2. Create a user, project. 3. Call createMilestone.	1. Status code 500 is returned.
MIL-005	Verify successful milestone retrieval	1. Create user, project, and milestone. 2. Call getMilestone endpoint with the milestone ID.	1. Status code 200 is returned. 2. Response contains correct milestone data.
MIL-006	Verify retrieval fails if the milestone does not exist	1. Create a user. 2. Call the getMilestone endpoint with a non-existent milestone ID	1. Status code 404 is returned.
MIL-007	Verify retrieval fails if the user doesn't have access to the milestone	1. Create users (user, otherUser), project, and milestone (with user). 2. Call getMilestone with otherUser and the milestone ID.	1. Status code 403 is returned.
MIL-008	Verify successful milestone update	1. Create user, project, and milestone. 2. Call updateMilestone endpoint with new data and milestone ID.	1. Status code 200 is returned. 2. Response contains updated milestone data.
MIL-009	Verify update fails if the milestone does	1. Mock find_one to return	1. Status code 404 is

	not exist	None. 2. Create user, project, and call updateMilestone with non-existent milestone ID.	returned.
MIL-010	Verify update fails with database error	1. Mock find_one_and_update to return None. 2. Create user, project, milestone. 3. Call updateMilestone.	1. Status code 500 is returned.
MIL-011	Verify update fails if the user doesn't have access to the milestone	1. Create users, project, milestone. 2. Call updateMilestone with otherUser and milestone ID.	1. Status code 403 is returned.
MIL-012	Verify successful milestone deletion	1. Create user, project, milestone. 2. Call deleteMilestone. 3. Attempt to retrieve the milestone.	1. Status code 200 on delete. 2. Status code 404 when attempting to retrieve.
MIL-013	Verify deletion fails if the milestone does not exist	1. Mock find_one to return None. 2. Create user, project. 3. Call deleteMilestone with non-existent ID.	1. Status code 404 is returned.
MIL-014	Verify deletion fails with database error (delete_one)	1. Mock delete_one (return .deleted_count = 0). 2. Create user, project, milestone. 3. Call deleteMilestone.	1. Status code 500 is returned.
MIL-015	Verify deletion fails with database error (update_many) on milestones	1. Mock update_many to fail on milestones. 2. Create user, project, milestone. 3. Call deleteMilestone.	1. Status code 500 is returned.
MIL-016	Verify deletion fails with database error (update_many) on tasks	1. Mock update_many to fail on tasks. 2. Create user, project, milestone. 3. Call deleteMilestone.	1. Status code 500 is returned.
PROJ-001	Verify successful project creation	1. Create a user. 2. Call createProject endpoint with valid data, including project name and description.	1. Status code 200 is returned. 2. Response contains newly created project data.
PROJ-002	Verify creation fails with database error (insert_one)	1. Mock insert_one to return acknowledged=False. 2. Create a user and project data. 3. Call createProject endpoint.	1. Status code 500 is returned.
PROJ-003	Verify creation fails if the user cannot be found (find_one_and_update)	1. Mock find_one_and_update to return None. 2. Create user data. 3. Call createProject.	1. Status code 500 is returned.
PROJ-004	Verify project retrieval (including	1. Create a user, project,	1. Status code 200 is

	milestones, tasks, and sprints)	milestone, task, and sprint (associated with the project). 2. Call getProject endpoint with the project's ID.	returned. 2. Response contains correct project, milestone, task, and sprint data.
PROJ-005	Verify retrieval fails if the project does not exist	1. Create user. 2. Call getProject endpoint with a non-existent project ID.	1. Status code 404 is returned.
PROJ-006	Verify retrieval fails if the user doesn't have access to the project	1. Create users (user and otherUser), project, with user. 2. Call getProject with otherUser and project ID.	1. Status code 403 is returned.
PROJ-007	Verify successful project deletion	1. Create user, project, related milestone, task, and sprint. 2. Call deleteProject. 3. Attempt to retrieve the project, milestone, task and sprint.	1. Status code 200 on delete. 2. Status code 404 when attempting to retrieve entities.
PROJ-008	Verify deletion fails if the project does not exist	1. Mock find_one to return None. 2. Create a user. 3. Call deleteProject with non-existent ID.	1. Status code 404 is returned.
PROJ-009	Verify deletion fails with database error on projects (delete_one)	1. Mock delete_one in projects to fail. 2. Create user, project. 3. Call deleteProject.	1. Status code 500 is returned.
PROJ-010	Verify deletion fails with database error on milestones (delete_many)	1. Mock delete_many in milestones to fail. 2. Create user, project. 3. Call deleteProject.	1. Status code 500 is returned.
PROJ-011	Verify deletion fails with database error on tasks (delete_many)	1. Mock delete_many in tasks to fail. 2. Create user, project. 3. Call deleteProject.	1. Status code 500 is returned.
PROJ-012	Verify deletion fails with database error on sprints (delete_many)	1. Mock delete_many in sprints to fail. 2. Create user, project. 3. Call deleteProject.	1. Status code 500 is returned.
PROJ-013	Verify deletion fails with database error on users (find_one_and_update)	1. Mock find_one_and_update on users to fail. 2. Create user and project. 3. Call deleteProject.	1. Status code 500 is returned.
PROJ-014	Verify deletion fails with database error on users (update_many)	1. Mock update_many on users to fail. 2. Create user and project. 3. Call deleteProject.	1. Status code 500 is returned.
PROJ-015	Verify project update	1. Create user, project. 2. Call updateProject with new name/description and	1. Status code 200 is returned. 2. Response contains updated project

		project ID.	data.
PROJ-016	Verify update fails if the project does not exist	1. Mock find_one_and_update to return None. 2. Create user, project data. 3. Call 'updateProject' with non-existent project ID.	1. Status code 404 is returned.
PROJ-017	Verify update fails with database error (find_one_and_update)	1. Mock find_one_and_update to return None. 2. Create project, user, and update data. 3. Call updateProject.	1. Status code 500 is returned.
PROJ-018	Verify user can join a project	1. Create users (user and user2), project (with user). 2. Call joinProject endpoint (user2).	1. Status code 200. 2. User data (user2) reflects joined project.
PROJ-019	Verify join fails with a non-existent project	1. Create user. 2. Call joinProject with non-existent project ID.	1. Status code 404 is returned.
PROJ-020	Verify joining fails with database error (find_one_and_update)	1. Mock find_one_and_update to fail. 2. Create user, project.	1. Status code 500 is returned.
SPRT-001	Verify successful sprint creation	1. Create user, project. 2. Call createSprint endpoint with valid data, including dates and project ID.	1. Status code 200 is returned. 2. Response contains newly created sprint data.
SPRT-002	Verify creation fails if the project does not exist	1. Create user. 2. Call createSprint endpoint with valid data and non-existent project ID.	1. Status code 404 is returned.
SPRT-003	Verify creation fails with unauthorized user	1. Create users (user and otherUser), project (with user). 2. Call createSprint with otherUser and project ID.	1. Status code 403 is returned.
SPRT-004	Verify creation fails with database error (insert_one)	1. Mock insert_one to return acknowledged=False. 2. Create a user, project, and sprint data. 3. Call createSprint.	1. Status code 500 is returned.
SPRT-005	Verify sprint retrieval	1. Create user, project, and sprint. 2. Call getSprint endpoint with the sprint ID.	1. Status code 200 is returned. 2. Response contains correct sprint data.
SPRT-006	Verify retrieval fails with unauthorized user	1. Create users (user, otherUser), project, sprint (with user). 2. Call getSprint with otherUser and sprint ID.	1. Status code 403 is returned.
SPRT-007	Verify retrieval fails if the sprint does not	1. Create user, project. 2. Call	1. Status code 404 is

	exist	getSprint with non-existent sprint ID.	returned.
SPRT-008	Verify successful sprint update	1. Create user, project, and sprint. 2. Call updateSprint endpoint with new data and sprint ID.	1. Status code 200 is returned. 2. Response contains updated sprint data.
SPRT-009	Verify update fails if the sprint does not exist	1. Mock find_one_and_update to return None. 2. Create a user, project and sprint data. 3. Call updateSprint.	1. Status code 404 is returned.
SPRT-010	Verify update fails with unauthorized user	1. Create users (user, otherUser), project, and sprint (with user). 2. Call updateSprint with otherUser and sprint ID.	1. Status code 403 is returned.
SPRT-011	Verify update fails with database error (find_one_and_update)	1. Mock find_one_and_update to return None. 2. Create a user, project, sprint, and updates. 3. Call updateSprint.	1. Status code 500 is returned.
SPRT-012	Verify successful sprint deletion	1. Create user, project, sprint. 2. Call deleteSprint. 3. Attempt to retrieve sprint.	1. Status code 200 on delete. 2. Status code 404 when attempting to retrieve.
SPRT-013	Verify deletion fails if the sprint does not exist	1. Mock find_one to return None. 2. Create a user and project. 3. Call deleteSprint with non-existent ID.	1. Status code 404 is returned.
SPRT-014	Verify deletion fails with unauthorized user	1. Create users (user, otherUser), project, and sprint (with user). 2. Call deleteSprint with otherUser and sprint ID.	1. Status code 403 is returned.
SPRT-015	Verify deletion fails with database error (delete_one)	1. Mock delete_one to fail. 2. Create user, project, and sprint. 3. Call deleteSprint.	1. Status code 500 is returned.
TASK-001	Verify successful task creation	1. Create user, project, and milestone. 2. Call createTask endpoint with valid data, including project and milestone IDs, and the task payload (i.e., self.testTask).	1. Status code 200 is returned. 2. Response contains newly created task data.
TASK-002	Verify creation fails with non-existent project	1. Create user and milestone. 2. Call createTask with valid data, non-existent project ID, and milestone ID.	1. Status code 404 is returned.
TASK-003	Verify creation fails with non-existent	1. Create user and project. 2.	1. Status code 404 is

	milestone	Call createTask with valid data, project ID, and non-existent milestone ID.	returned.
TASK-004	Verify creation fails with mismatched project and milestone	1. Create users, two projects (project1, project2), and a milestone (in project1). 2. Call createTask with project2 ID, milestone ID, and valid task data.	1. Status code 400 is returned.
TASK-005	Verify creation fails with unauthorized user	1. Create users (user and anotherUser), project, and milestone (with user). 2. Call createTask with anotherUser, project ID and milestone ID.	1. Status code 403 is returned.
TASK-006	Verify creation fails with database error (insert_one)	1. Mock insert_one to return acknowledged=False. 2. Create user, project, milestone, and task data. 3. Call createTask.	1. Status code 500 is returned.
TASK-007	Verify task retrieval	1. Create user, project, milestone, and task. 2. Call getTask endpoint with the task ID.	1. Status code 200 is returned. 2. Response contains correct task data.
TASK-008	Verify retrieval fails with unauthorized user	1. Create users (user and anotherUser), project, milestone, and task (with user). 2. Call getTask with anotherUser and task ID.	1. Status code 403 is returned.
TASK-009	Verify successful task update	1. Create user, project, milestone, and task. 2. Call updateTask endpoint with new data and task ID, including any updates to qaTask.	1. Status code 200 is returned. 2. Response contains updated task data.
TASK-010	Verify task update with project and milestone change	1. Create two projects (project1, project2), two milestones (milestone1 in project1, milestone2 in project2), and a task in project1 associated with milestone1. 2. Call updateTask with the task ID, new project ID (project2), and new milestone ID (milestone2).	1. Status code 200 is returned. 2. Updated task has the correct project and milestone IDs.
TASK-011	Verify update fails when project does not exist	1. Create user, project, milestone, and task. 2. Call updateTask with a	1. Status code 404 is returned.

		non-existent project ID.	
TASK-012	Verify update fails with non-existent milestone	1. Create user, project, milestone, and task. 2. Call updateTask with a non-existent milestone ID.	1. Status code 404 is returned.
TASK-013	Verify update fails with non-existent task	1. Mock find_one_and_update to return None. 2. Create user, project, milestone and update data. 3. Call updateTask with non-existent task ID.	1. Status code 404 is returned.
TASK-014	Verify update fails with unauthorized user	1. Create users (user and anotherUser), project, milestone, and task (with user). 2. Call updateTask with anotherUser and task ID.	1. Status code 403 is returned.
TASK-015	Verify update fails with database error (find_one_and_update)	1. Mock find_one_and_update to return None. 2. Create user, project, milestone, task, and update data. 3. Call updateTask.	1. Status code 500 is returned.
TASK-016	Verify task deletion	1. Create a user, project, milestone, and task. 2. Call deleteTask endpoint with the task ID.	1. Status code 200 on delete. 2. Status code 404 when attempting to retrieve the task.
TASK-017	Verify task deletion with dependent tasks	1. Create a user, project, milestone, and two tasks (task1, task2) where task2 has dependency on task1. 2. Call deleteTask with task1 ID.	1. Status code 200. 2. Verify task1 is deleted. 3. Verify task2 has empty dependencies.
TASK-018	Verify deletion fails with non-existent task	1. Create a user, project, and milestone. 2. Call deleteTask endpoint with a non-existent task ID.	1. Status code 404 is returned.
TASK-019	Verify deletion fails with unauthorized user	1. Create users (user, anotherUser), project, milestone, and task (created by user). 2. Call deleteTask with anotherUser and task ID.	1. Status code 403 is returned.
TASK-020	Verify deletion fails with database error (delete_one)	1. Mock delete_one to fail. 2. Create a user, project, milestone, and task. 3. Call deleteTask.	1. Status code 500 is returned.
TASK-021	Verify deletion failure updates milestones (update_many)	1. Mock update_many (in milestones collection) to fail.	1. Status code 500

		2. Create a user, project, milestone, and task. 3. Call deleteTask.	
USER-001	Verify successful user registration	1. Call registerUser endpoint with valid user data (username, password, email).	1. Status code 201 is returned. 2. Response contains user data (excluding raw password).
USER-002	Verify registration fails with invalid data	1. Call registerUser with invalid data (e.g., missing email, short password).	1. Status code 422 is returned. 2. Response contains error details.
USER-003	Verify registration fails if user already exists (username)	1. Register a user. 2. Call registerUser again with the same username.	1. Status code 400 is returned. 2. Response contains an appropriate error message.
USER-004	Verify registration fails if user already exists (email)	1. Register a user. 2. Call registerUser again with a different username but the same email.	1. Status code 400 is returned. 2. Response contains an appropriate error message.
USER-005	Verify registration fails with database error (insert_one)	1. Mock insert_one to return acknowledged=False. 2 . Call registerUser with valid data.	1. Status code 500 is returned.
USER-006	Verify registration fails with token generation error (find_one_and_update)	1. Mock find_one_and_update to return None. 2. Call registerUser with valid data.	1. Status code 500 is returned.
USER-007	Verify successful login	1. Register a user. 2. Call loginUser with the correct username and password.	1. Status code 200 is returned. 2. Response contains a valid access token.
USER-008	Verify login fails with invalid credentials (incorrect password)	1. Register a user. 2. Call loginUser with the correct username and an incorrect password.	1. Status code 401 is returned.
USER-009	Verify login fails with invalid credentials (non-existent user)	1. Call loginUser with a non-existent username and any password.	1. Status code 401 is returned.
USER-010	Verify getting current user (valid authorization)	1. Register a user. 2. Get access token (call loginUser). 3. Call /users/me with the token in the "Authorization: Bearer {token}" header.	1. Status code 200 is returned. 2. Response contains the correct user data.
USER-011	Verify getting current user fails with invalid token	1. Call /users/me with an invalid token in the "Authorization: Bearer {token}" header.	1. Status code 401 is returned.
USER-012	Verify current user not found (find_one	1. Register a user. 2. Get a	1. Status code 404 is

	failure)	valid access token. 3. Mock find_one to return None. 4. Call /users/me with a valid token.	returned.
USER-013	Verify successful password change	1. Register a user. 2. Get a valid access token. 3. Call /users/password/reset with a valid token and a new password.	1. Status code 200 is returned. 2. New password can be used to login successfully.
USER-014	Verify password change fails on database error (find_one_and_update)	1. Register a user. 2. Get a valid access token. 3. Mock find_one_and_update to return None. 4. Call /users/password/reset.	1. Status code 500 is returned.