Employee Management System - Testing Strategy

1. Introduction

This document outlines the testing strategy for the **Employeest -** Employee Management System (EMS). The **Employeest** is an application designed to help businesses track active projects, manage employees, log work, and generate reports. This strategy focuses on manual testing procedures to ensure the application meets its functional and non-functional requirements, providing a high-quality and reliable system.

Purpose:

- To define the scope, objectives, and approach for testing the EMS.
- To provide a comprehensive set of manual test cases covering various aspects of the application.
- To establish a process for smoke testing and defect management.

Scope of Testing:
The testing will cover all backend functionalities exposed through the API, including:

- User Authentication and Authorization (Roles: owner, employee, topemployee, admin)
- User Profile Management
- Team Management
- Project Management (CRUD operations, charts)
- Task Management (CRUD operations, status transitions, assignments, filtering, sorting)
- WorkLog Management (CRUD operations)
- Dashboard views (Owner, Employee)
- Reporting and Statistics (Business-level and personal stats charts)
- API endpoint validation, error handling, and response codes.

While this document focuses on backend testing (simulating UI interactions via API calls or a testing client), UI test cases are included at a high level to guide frontend testing.

2. Testing Objectives

- Verify that all functional requirements are implemented correctly as per the specifications.
- Ensure that user roles and permissions are enforced correctly.
- Validate data integrity and consistency across the system.

- Identify and report defects in a timely and effective manner.
- Confirm that the system is usable and provides a satisfactory user experience (from an API interaction perspective).
- Verify basic non-functional aspects like error handling and response times for critical operations.

3. Testing Approach

A combination of manual testing techniques will be employed:

- **Functional Testing:** To verify that each function of the software application operates in conformance with the requirement specification.
- **Non-Functional Testing:** To check aspects like security (authorization), and basic usability of the API. Performance testing is out of scope for this manual plan but can be suggested.
- **UI Testing:** High-level checks for the user interface that would interact with this backend.
- **Smoke Testing:** To ensure the most critical functionalities are working before proceeding with more extensive testing.
- **Regression Testing (Implied):** After defect fixes or new feature implementations, relevant test cases will be re-executed.

4. Test Environment and Tools

- **Test Environment:** A dedicated test environment mirroring the production setup as closely as possible (e.g., using the Docker setup provided).
- **Tools:**
    - API Client (e.g., Postman, Insomnia) for sending requests and verifying responses.
    - Spreadsheet software (e.g., Google Sheets, MS Excel) for test case management and defect logging.
    - Database client (e.g., DB Browser for SQLite) for data verification if needed.

5. Roles and Responsibilities

- **Test Lead/QA Manager:** Oversees the testing process, reviews test plans and reports.
- **QA Tester(s):** Develops and executes test cases, logs defects, and provides test reports.
- **Developer(s):** Fixes defects, provides support to the QA team.

6. Smoke Testing

Smoke tests are a subset of critical test cases executed before detailed testing to ensure the main functionalities are operational and the build is stable.

Smoke Test Cases:

ID	Test Case Title	Priority
SM_001	User Login (Owner)	High
SM_002	User Login (Employee)	High
SM_003	Create a new Project (as Owner)	High
SM_004	Retrieve Project List	High
SM_005	Create a new Task within a Project (as Owner)	High
SM_006	Retrieve Task List	High
SM_007	View Owner Dashboard	High
SM_008	View Employee Dashboard	High
SM_009	Log work on a Task (as Employee)	High
SM_010	View User Profile	High

7. Manual Test Cases

The following sections detail the manual test cases.

Test Case Format:

- **ID:** Unique identifier for the test case.
- **Category:** Functional, Non-Functional, UI.
- **Sub-Category:** Specific module (e.g., Project Management).
- **Test Case Title:** Brief description of the test.
- **Preconditions:** Conditions that must be met before executing the test.
- **Steps:** Actions to perform.
- **Expected Results:** The anticipated outcome.
- **Priority:** High, Medium, Low.

7.1 Functional Test Cases

7.1.1 Authentication & Authorization (15 Test Cases)

ID	Test Case Title	Preconditions	Steps	Expected Results	Priority
TC_AUTH_001	Successful login with valid owner credentials	Owner user exists.	1. Send login request with valid owner username/password.	200 OK. Valid token/session received.	High
TC_AUTH_002	Successful login with valid employee credentials	Employee user exists.	1. Send login request with valid employee username/password.	200 OK. Valid token/session received.	High
TC_AUTH_003	Failed login with invalid username	-	1. Send login request with invalid username and valid password.	400/401 Error. Appropriate error message. No token.	High
TC_AUTH_004	Failed login with invalid password	User exists.	1. Send login request with valid username and invalid password.	400/401 Error. Appropriate error message. No token.	High
TC_AUTH_005	Access protected resource (e.g., Project list) without authentication	-	1. Send GET request to /api/v1/projects/ without auth token.	401/403 Error. Access denied.	High
TC_AUTH_006	Owner attempts to create a project	Owner logged in.	1. Send POST request to /api/v1/projects/ with valid	201 Created. Project created successfully. Owner field	High

			project data.	is set correctly.	
TC_AUTH_007	Employee attempts to create a project	Employee logged in.	1. Send POST request to /api/v1/projects/ with valid project data.	403 Forbidden. Employee should not be able to create projects directly (unless perform_create logic is changed or a different role grants it).	High
TC_AUTH_008	Owner attempts to update their own project	Owner logged in. Project created by this owner exists.	1. Send PUT/PATCH request to /api/v1/projects/{project_id}/ with valid data.	200 OK. Project updated.	High
TC_AUTH_009	Owner attempts to update another owner's project	Two owners exist. Project created by Owner B. Owner A logged in.	1. Owner A sends PUT/PATCH request to Owner B's project.	403 Forbidden. Access denied.	High
TC_AUTH_010	Employee (not assignee, not project owner) attempts to update a task	Employee logged in. Task exists, not assigned to this employee, project not owned by this employee.	1. Send PUT/PATCH request to /api/v1/tasks/{task_id}/.	403 Forbidden. Access denied.	High
TC_AUTH_011	Task Assignee attempts to	Employee (assignee) logged in.	1. Send PUT/PATCH request to	200 OK. Task updated.	High

	update their assigned task	Task assigned to this employee exists.	/api/v1/tasks/{task_id}/ with valid data.		
TC_AUTH_012	Project Owner attempts to update a task in their project (not assigned to them)	Owner logged in. Project and task exist. Task assigned to another user.	1. Send PUT/PATCH request to /api/v1/tasks/{task_id}/ with valid data.	200 OK. Task updated.	High
TC_AUTH_013	Admin user access to list all worklogs	Admin user logged in. Multiple worklogs by different users exist.	1. Admin sends GET request to /api/v1/worklogs/.	200 OK. All worklogs are listed.	Medium
TC_AUTH_014	Non-admin user attempts to list all worklogs (should see only their own)	Employee logged in. Worklogs by this employee and others exist.	1. Employee sends GET request to /api/v1/worklogs/.	200 OK. Only worklogs created by this employee are listed.	Medium
TC_AUTH_015	Access Owner Dashboard as Owner	Owner logged in.	1. Send GET request to /api/v1/dashboards/owner/.	200 OK. Dashboard data returned.	High
TC_AUTH_016	Access Owner Dashboard as Employee	Employee logged in.	1. Send GET request to /api/v1/dashboards/owner/.	403 Forbidden. Access denied.	High
TC_AUTH_017	Access Employee Dashboard as Employee	Employee logged in.	1. Send GET request to /api/v1/dashboards/emplo	200 OK. Dashboard data returned.	High

			yee/.		

7.1.2 User Profile Management (5 Test Cases)

ID	Test Case Title	Preconditions	Steps	Expected Results	Priority
TC_UP_001	View own user profile	User logged in.	1. Send GET request to /api/v1/profile/.	200 OK. User's profile data (id, username, email, first_name, etc.) is returned.	High
TC_UP_002	Verify user profile data accuracy	User logged in. User profile data is known.	1. Send GET request to /api/v1/profile/. 2. Compare response data.	Response data matches the known profile information.	Medium
TC_UP_003	Attempt to view another user's profile via /api/v1/profile/	User logged in.	1. (If possible) Try to manipulate request to get another user's profile.	Should only return the logged-in user's profile. No access to others.	Medium
TC_UP_004	User profile access after logout	User was logged in, then logged out.	1. Send GET request to /api/v1/profile/ without auth token.	401/403 Unauthorized/Forbidden.	High
TC_UP_005	User profile data structure	User logged in.	1. Send GET request to /api/v1/profile/.	Response contains expected fields: id, username, email,	Medium

				first_name, last_name, phone_number, role.	

7.1.3 Team Management (5 Test Cases)

ID	Test Case Title	Preconditions	Steps	Expected Results	Priority
TC_TM_001	Create a new Team	Admin/Owner user logged in.	1. Create a team with name, description, owner.	Team created successfully with correct attributes.	Medium
TC_TM_002	Add members to a Team	Team exists. Users exist.	1. Associate users with the team.	Users are correctly added as team members. User.team field is updated.	Medium
TC_TM_003	View Team details	Team exists.	1. Retrieve team details.	Correct team name, description, owner, and members are displayed.	Medium
TC_TM_004	Assign Project to a Team	Project exists. Team exists.	1. Link project to team in Project.team M2M field.	Project is associated with the team.	Medium
TC_TM_005	Remove member from a Team	Team exists with members.	1. Disassociate a user from the team.	User is removed from the team.	Medium

7.1.4 Project Management (15 Test Cases)

ID	Test Case Title	Preconditions	Steps	Expected Results	Priority
TC_PM_001	Create a new project with valid data	Owner user logged in.	1. Send POST to /api/v1/projects/ with name, description, owner_id.	201 Created. Project details returned. owner is correctly set to logged-in user. created_at, updated_at set.	High
TC_PM_002	Create a project with missing required fields (name)	Owner user logged in.	1. Send POST to /api/v1/projects/ without 'name'.	400 Bad Request. Error message indicating 'name' is required.	High
TC_PM_003	Retrieve list of all projects	User logged in. Multiple projects exist.	1. Send GET to /api/v1/projects/.	200 OK. List of projects returned with pagination if applicable. Includes tasks_count and tasks (simple).	High
TC_PM_004	Retrieve a specific project by ID	User logged in. Project with known ID exists.	1. Send GET to /api/v1/projects/{project_id}/.	200 OK. Correct project details returned.	High
TC_PM_005	Retrieve a non-existent project by ID	User logged in.	1. Send GET to /api/v1/projects/{non_existent_id}/.	404 Not Found.	Medium

TC_PM_006	Update an existing project with valid data (as Owner)	Owner logged in. Project created by this owner exists.	1. Send PUT to /api/v1/projects/{project_id}/ with updated name, description.	200 OK. Project details updated. updated_at is modified.	High
TC_PM_007	Partially update an existing project (as Owner)	Owner logged in. Project created by this owner exists.	1. Send PATCH to /api/v1/projects/{project_id}/ with updated description only.	200 OK. Project description updated. Name remains unchanged. updated_at is modified.	High
TC_PM_008	Attempt to update project owner field via API	Owner logged in. Project exists.	1. Send PUT/PATCH to /api/v1/projects/{project_id}/ attempting to change owner_id.	owner_id is write_only for creation. owner is read_only. Update of owner might be restricted by serializer. Verify behavior. Expected: Owner cannot be changed post-creation via this field.	Medium
TC_PM_009	Delete an existing project (as Owner)	Owner logged in. Project created by this owner exists.	1. Send DELETE to /api/v1/projects/{project_id}/.	204 No Content. Project is deleted.	High
TC_PM_010	Attempt to delete a	Owner A logged in.	1. Owner A sends	403 Forbidden.	High

	project owned by another user	Project owned by Owner B exists.	DELETE to /api/v1/projects/{project_id_of_owner_B}/.		
TC_PM_011	Attempt to delete a non-existent project	Owner logged in.	1. Send DELETE to /api/v1/projects/{non_existent_id}/.	404 Not Found.	Medium
TC_PM_012	Verify tasks_count in project list/detail	Project exists with multiple tasks.	1. Send GET to /api/v1/projects/ and /api/v1/projects/{project_id}/.	tasks_count field accurately reflects the number of tasks associated with the project.	Medium
TC_PM_013	Verify tasks (simple list) in project detail	Project exists with tasks.	1. Send GET to /api/v1/projects/{project_id}/.	tasks array contains simplified task details (id, name, status, assignee, deadline).	Medium
TC_PM_014	Create project with very long name/description	Owner logged in.	1. Send POST with name/description exceeding model max_length (if defined, else check DB limits).	400 Bad Request or DB error if not handled by validation. Expected: Validation error.	Medium
TC_PM_015	Project creation with non-existent owner_id (if owner_id	User logged in.	1. Send POST to /api/v1/projects/ with a non-existent	400 Bad Request (due to PrimaryKeyRelatedField	Medium

	was fully writable)		owner_id.	validation).	

7.1.5 Task Management (25 Test Cases)

ID	Test Case Title	Preconditions	Steps	Expected Results	Priority
TC_TSK_001	Create a new task with all valid fields	Project owner or authorized user logged in. Project exists. Assignee user exists.	1. Send POST to /api/v1/tasks/ with project_id, name, description, status, assignee_id, story_points, deadline, estimation_hours.	201 Created. Task details returned. created_at, updated_at set. project_name and assignee (simple) populated.	High
TC_TSK_002	Create a task with minimal required fields (project_id, name)	Project owner logged in. Project exists.	1. Send POST to /api/v1/tasks/ with project_id, name.	201 Created. Default status (TODO) applied. Other optional fields are null/blank.	High
TC_TSK_003	Create a task with non-existent project_id	User logged in.	1. Send POST to /api/v1/tasks/ with a non-existent project_id.	400 Bad Request. Error message "Project does not exist."	High
TC_TSK_004	Create a task with non-existent assignee_id	User logged in. Project exists.	1. Send POST to /api/v1/tasks/ with project_id,	400 Bad Request. Error message "Assignee	Medium

			name, and a non-existent assignee_id.	(User) does not exist."	
TC_TSK_005	Create a task with invalid status value	User logged in. Project exists.	1. Send POST to /api/v1/tasks/ with an invalid status (e.g., "PENDING").	400 Bad Request. Error message indicating invalid choice for status.	Medium
TC_TSK_006	Retrieve list of all tasks	User logged in. Multiple tasks exist.	1. Send GET to /api/v1/tasks/.	200 OK. List of tasks returned with pagination if applicable.	High
TC_TSK_007	Retrieve a specific task by ID	User logged in. Task with known ID exists.	1. Send GET to /api/v1/tasks/{task_id}/.	200 OK. Correct task details returned.	High
TC_TSK_008	Retrieve a non-existent task by ID	User logged in.	1. Send GET to /api/v1/tasks/{non_existent_id}/.	404 Not Found.	Medium
TC_TSK_009	Update an existing task with valid data (as Owner/Assignee)	Owner/Assignee logged in. Task exists.	1. Send PUT to /api/v1/tasks/{task_id}/ with updated fields (name, description, status, assignee_id, story_points, deadline). project_id must be included for PUT.	200 OK. Task details updated. updated_at is modified.	High

TC_TSK_010	Partially update an existing task (as Owner/Assignee)	Owner/Assignee logged in. Task exists.	1. Send PATCH to /api/v1/tasks/{task_id}/ with updated status only.	200 OK. Task status updated. Other fields remain unchanged. updated_at is modified.	High
TC_TSK_011	Attempt to update task's project_id (as Owner/Assignee)	Owner/Assignee logged in. Task exists. Another project exists.	1. Send PUT/PATCH to /api/v1/tasks/{task_id}/ attempting to change project_id.	project_id is write_only. This should update the project association. Verify.	Medium
TC_TSK_012	Delete an existing task (as Owner/Assignee)	Owner/Assignee logged in. Task exists.	1. Send DELETE to /api/v1/tasks/{task_id}/.	204 No Content. Task is deleted.	High
TC_TSK_013	Filter tasks by status (e.g., ?status=TODO)	Tasks with various statuses exist.	1. Send GET to /api/v1/tasks/?status=TODO.	200 OK. Only tasks with status 'TODO' are returned.	Medium
TC_TSK_014	Filter tasks by assignee_id (e.g., ?assignee_id=X)	Tasks assigned to different users exist.	1. Send GET to /api/v1/tasks/?assignee_id={user_id}.	200 OK. Only tasks assigned to the specified user are returned.	Medium
TC_TSK_015	Filter tasks by project_id (e.g., ?project_id=Y)	Tasks in different projects exist.	1. Send GET to /api/v1/tasks/?project_id={project_id}.	200 OK. Only tasks belonging to the specified project are returned.	Medium
TC_TSK_016	Filter tasks by deadline_aft	Tasks with various deadlines	1. Send GET to /api/v1/tasks/	200 OK. Only tasks with	Medium

	er	exist.	?deadline_after=YYYY-MM-DD.	deadline on or after the specified date are returned.	
TC_TSK_017	Filter tasks by deadline_before	Tasks with various deadlines exist.	1. Send GET to /api/v1/tasks/?deadline_before=YYYY-MM-DD.	200 OK. Only tasks with deadline on or before the specified date are returned.	Medium
TC_TSK_018	Filter tasks by project_name (e.g., ?project_name=Alpha)	Tasks in projects with different names exist.	1. Send GET to /api/v1/tasks/?project_name=Alpha.	200 OK. Only tasks belonging to projects with names containing 'Alpha' are returned.	Medium
TC_TSK_019	Search tasks by name (e.g., ?search=keyword)	Tasks with 'keyword' in their name exist.	1. Send GET to /api/v1/tasks/?search=keyword.	200 OK. Tasks with 'keyword' in name or description are returned.	Medium
TC_TSK_020	Order tasks by deadline (e.g., ?ordering=deadline)	Tasks with different deadlines exist.	1. Send GET to /api/v1/tasks/?ordering=deadline.	200 OK. Tasks are ordered by deadline in ascending order.	Medium
TC_TSK_021	Order tasks by status descending (e.g., ?ordering=-status)	Tasks with different statuses exist.	1. Send GET to /api/v1/tasks/?ordering=-status.	200 OK. Tasks are ordered by status in descending order.	Medium
TC_TSK_022	Action: Start progress on	Assignee/Owner logged	1. Send POST to	200 OK. Task status	High

	a 'TODO' task	in. Task is 'TODO'.	/api/v1/tasks/{task_id}/start-progress/.	changed to 'IN_PROGRESS'. Response reflects this.	
TC_TSK_023	Action: Attempt to start progress on an 'IN_PROGRESS' task	Assignee/Owner logged in. Task is 'IN_PROGRESS'.	1. Send POST to /api/v1/tasks/{task_id}/start-progress/.	400 Bad Request. Error message indicating task cannot be moved.	Medium
TC_TSK_024	Action: Mark an 'IN_PROGRESS' task as 'DONE'	Assignee/Owner logged in. Task is 'IN_PROGRESS'.	1. Send POST to /api/v1/tasks/{task_id}/mark-as-done/.	200 OK. Task status changed to 'DONE'. updated_at is refreshed. Response reflects this.	High
TC_TSK_025	Action: Attempt to mark a 'TODO' task as 'DONE'	Assignee/Owner logged in. Task is 'TODO'.	1. Send POST to /api/v1/tasks/{task_id}/mark-as-done/.	400 Bad Request. Error message indicating task cannot be marked as done.	Medium

7.1.6 WorkLog Management (15 Test Cases)

ID	**Test Case Title**	**Preconditions**	**Steps**	**Expected Results**	**Priority**
TC_WL_001	Create a new worklog for a task with valid data	User logged in. Task exists.	1. Send POST to /api/v1/worklogs/ with task_id, date, hours_spent, description.	201 Created. Worklog details returned. user is correctly set to logged-in user.	High

				created_at set.	
TC_WL_002	Create a new worklog for a project with valid data	User logged in. Project exists.	1. Send POST to /api/v1/worklogs/ with project_id, date, hours_spent, description.	201 Created. Worklog details returned. user is correctly set.	High
TC_WL_003	Create worklog with default date (date not provided)	User logged in. Task exists.	1. Send POST to /api/v1/worklogs/ with task_id, hours_spent.	201 Created. date field is set to today's date.	Medium
TC_WL_004	Create worklog with missing required fields (hours_spent)	User logged in. Task exists.	1. Send POST to /api/v1/worklogs/ with task_id, date.	400 Bad Request. Error message indicating 'hours_spent' is required.	High
TC_WL_005	Create worklog associated with both task_id and project_id	User logged in. Task and Project exist.	1. Send POST to /api/v1/worklogs/ with task_id, project_id, date, hours_spent.	400 Bad Request. Validation error: "Work log cannot be associated with both a task and a project simultaneously."	High
TC_WL_006	Create worklog associated with neither task_id nor project_id	User logged in.	1. Send POST to /api/v1/worklogs/ with date, hours_spent.	400 Bad Request. Validation error: "Work log must be associated with a task	High

				or a project."	
TC_WL_007	Retrieve list of own worklogs	User logged in. User has created multiple worklogs.	1. Send GET to /api/v1/worklogs/.	200 OK. List of worklogs created by the logged-in user is returned.	High
TC_WL_008	Admin retrieves list of all worklogs	Admin user logged in. Worklogs by different users exist.	1. Send GET to /api/v1/worklogs/.	200 OK. List of all worklogs in the system is returned.	Medium
TC_WL_009	Retrieve a specific own worklog by ID	User logged in. Worklog created by this user exists.	1. Send GET to /api/v1/worklogs/{worklog_id}/.	200 OK. Correct worklog details returned.	High
TC_WL_010	Attempt to retrieve another user's worklog by ID (as non-admin)	User A logged in. Worklog by User B exists.	1. User A sends GET to /api/v1/worklogs/{user_B_worklog_id}/.	404 Not Found (as per get_queryset filtering for non-admins) .	Medium
TC_WL_011	Update an existing own worklog with valid data	User logged in. Worklog created by this user exists. Task/Project for association exists.	1. Send PUT to /api/v1/worklogs/{worklog_id}/ with updated hours_spent, description, date. Include task_id or project_id.	200 OK. Worklog details updated.	High
TC_WL_012	Partially update an existing own worklog	User logged in. Worklog created by this user exists.	1. Send PATCH to /api/v1/worklogs/{worklog_id}/ with	200 OK. Worklog description updated. Other fields	High

			updated description only.	remain unchanged.	
TC_WL_013	Attempt to update another user's worklog (as non-admin)	User A logged in. Worklog by User B exists.	1. User A sends PUT/PATCH to /api/v1/worklogs/{user_B_worklog_id}/.	404 Not Found (due to queryset filtering) or 403 Forbidden (if IsWorkLogOwner permission triggers first on the attempt to fetch). Expected: 404.	High
TC_WL_014	Delete an existing own worklog	User logged in. Worklog created by this user exists.	1. Send DELETE to /api/v1/worklogs/{worklog_id}/.	204 No Content. Worklog is deleted.	High
TC_WL_015	Attempt to delete another user's worklog (as non-admin)	User A logged in. Worklog by User B exists.	1. User A sends DELETE to /api/v1/worklogs/{user_B_worklog_id}/.	404 Not Found / 403 Forbidden. Expected: 404.	High

7.1.7 Dashboard Functionality (10 Test Cases)

ID	**Test Case Title**	**Preconditions**	**Steps**	**Expected Results**	**Priority**
TC_DSH_001	View Owner Dashboard as Owner	Owner logged in. Owner has projects and tasks.	1. Send GET to /api/v1/dashboards/owner/.	200 OK. Correct summary stats (total_projects, active_proje	High

				cts, task counts) and projects_list are returned.	
TC_DSH_002	Owner Dashboard: Verify active projects count	Owner logged in. Mix of active (TODO/IN_PROGRESS tasks) and inactive projects.	1. Send GET to /api/v1/dashboards/owner/.	active_projects count is accurate.	Medium
TC_DSH_003	Owner Dashboard: Verify task status counts	Owner logged in. Tasks with various statuses in owned projects.	1. Send GET to /api/v1/dashboards/owner/.	tasks_todo, tasks_inprogress, tasks_done counts are accurate for all tasks in owned projects.	Medium
TC_DSH_004	Owner Dashboard: Empty state (no projects)	Owner logged in. Owner has no projects.	1. Send GET to /api/v1/dashboards/owner/.	200 OK. Counts are zero. projects_list is empty.	Medium
TC_DSH_005	View Employee Dashboard as Employee	Employee logged in. Employee assigned to tasks/teams.	1. Send GET to /api/v1/dashboards/employee/.	200 OK. Correct my_projects (involved via tasks/teams), my_teams (if populated), and my_current_tasks (TODO/IN_PROGRESS) are returned.	High
TC_DSH_006	Employee Dashboard:	Employee logged in.	1. Send GET to	my_projects list includes	Medium

	Verify my_projects accuracy	Assigned to tasks in Project A, member of Team for Project B.	/api/v1/dashboards/employee/.	Project A and Project B.	
TC_DSH_007	Employee Dashboard: Verify my_current_tasks accuracy	Employee logged in. Has TODO, IN_PROGRESS, and DONE tasks assigned.	1. Send GET to /api/v1/dashboards/employee/.	my_current_tasks list includes only TODO and IN_PROGRESS tasks assigned to the employee.	Medium
TC_DSH_008	Employee Dashboard: Empty state (no assignments/teams)	Employee logged in. Not assigned to any tasks or teams.	1. Send GET to /api/v1/dashboards/employee/.	200 OK. my_projects, my_teams, my_current_tasks are empty lists.	Medium
TC_DSH_009	Attempt to view Owner Dashboard as Employee	Employee logged in.	1. Send GET to /api/v1/dashboards/owner/.	403 Forbidden.	High
TC_DSH_010	Attempt to view Employee Dashboard as Owner (if distinct logic)	Owner logged in.	1. Send GET to /api/v1/dashboards/employee/.	200 OK (Owners are also users, so they might see their own tasks/projects if structured that way, or specific employee view). Verify behavior.	Medium

7.1.8 Reporting & Charts (10 Test Cases)

ID	Test Case Title	Preconditions	Steps	Expected Results	Priority
TC_RPT_001	Project Task Status Chart: View for existing project (as Owner)	Owner logged in. Project exists with tasks in various statuses.	1. Send GET to /api/v1/projects/{project_id}/task-status-chart/.	200 OK. chart_url is returned. (Mock get_chart_url to verify config passed). Chart config has correct title, labels (statuses), data (counts).	High
TC_RPT_002	Project Task Status Chart: Project with no tasks (as Owner)	Owner logged in. Project exists but has no tasks.	1. Send GET to /api/v1/projects/{project_id}/task-status-chart/.	404 Not Found. Message "No tasks found...".	Medium
TC_RPT_003	Project Velocity Chart: View for existing project (as Owner)	Owner logged in. Project exists with DONE tasks with story points over time.	1. Send GET to /api/v1/projects/{project_id}/velocity-chart/.	200 OK. chart_url returned. Chart config has correct title, labels (weeks), data (sum of story points).	High
TC_RPT_004	Project Velocity Chart: Project with insufficient data (as Owner)	Owner logged in. Project has no DONE tasks with story points.	1. Send GET to /api/v1/projects/{project_id}/velocity-chart/.	404 Not Found. Message "Not enough data...".	Medium
TC_RPT_005	Business Statistics (Story Points	User logged in. DONE tasks with	1. Send GET to /api/v1/statist	200 OK. chart_url returned.	High

	Monthly): View as authorized user	story points exist across various months.	ics/business/story-points-monthly/.	Chart config has correct title, labels (months), data (sum of story points).	
TC_RPT_006	Business Statistics: No completed tasks with story points	User logged in. No DONE tasks with story points exist.	1. Send GET to /api/v1/statistics/business/story-points-monthly/.	404 Not Found. Message "No completed tasks with story points...".	Medium
TC_RPT_007	User Personal Task Stats Chart: View for own tasks	User logged in. User has DONE tasks assigned over various months.	1. Send GET to /api/v1/me/statistics/task-completion-chart/.	200 OK. chart_url returned. Chart config has correct title, labels (months), data (count of completed tasks).	High
TC_RPT_008	User Personal Task Stats Chart: User with no completed tasks	User logged in. User has no DONE tasks assigned.	1. Send GET to /api/v1/me/statistics/task-completion-chart/.	404 Not Found. Message "You have no completed tasks...".	Medium
TC_RPT_009	Chart endpoint access by unauthorized user (e.g., employee for project charts not owned)	Employee logged in. Project exists, not owned by employee's owner.	1. Employee sends GET to /api/v1/projects/{project_id}/task-status-chart/.	403 Forbidden (due to IsProjectOwner permission).	High
TC_RPT_010	Chart generation	User logged in. Data	1. Send GET request to a	500 Internal Server Error.	Medium

	API failure (get_chart_url returns None)	exists for chart. Mock get_chart_url to return None.	chart endpoint.	Message "Could not generate chart URL."	

7.1.9 API General (Pagination, Error Handling) (5 Test Cases)

ID	Test Case Title	Preconditions	Steps	Expected Results	Priority
TC_API_001	Verify pagination for list endpoints (e.g., Projects)	More projects exist than the default page size.	1. Send GET to /api/v1/projects/. 2. Check for count, next, previous, results fields.	200 OK. Response is paginated. results contains one page of items. next link is present if more pages exist.	Medium
TC_API_002	Access next page using pagination link	Paginated list endpoint with a next link.	1. Send GET to the next URL from a previous paginated response.	200 OK. Next set of items is returned.	Medium
TC_API_003	Invalid input type for field (e.g., string for integer)	Endpoint expects an integer for a field (e.g., story_points).	1. Send POST/PUT request with a string value for story_points.	400 Bad Request. Clear error message indicating the type mismatch for the field.	Medium
TC_API_004	Request with malformed JSON body	POST/PUT endpoint.	1. Send request with a JSON body that is syntactically	400 Bad Request. Error message indicating	Medium

			incorrect.	JSON parsing error.	
TC_API_005	Server error simulation (500)	- (Requires ability to simulate a server-side unhandled exception)	1. Trigger a scenario that leads to an unhandled server error.	500 Internal Server Error. Generic error message (should not expose sensitive details). Error is logged on the server.	Low

7.2 Non-Functional Test Cases (High-Level) (5 Test Cases)

ID	Category	Sub-Category	Test Case Title	Preconditions	Steps	Expected Results	Priority
TC_NF_001	Non-Functional	Security	Attempt to access resources with expired token/session	User was logged in, token/session has expired.	1. Send request to a protected endpoint using the expired token/session.	401 Unauthorized. Appropriate error message.	High
TC_NF_002	Non-Functional	Usability	API Error messages are clear and informative	Trigger various error scenarios (400, 401, 403, 404).	1. Review error responses.	Error messages clearly indicate the nature of the problem	Medium

						without exposing sensitive information. Consistent error response structure.	
TC_NF_003	Non-Functional	Performance	Response time for critical GET requests (e.g., dashboard, project list)	System has a moderate amount of data.	1. Measure response time for /api/v1/dashboards/owner/, /api/v1/projects/.	Response times are within acceptable limits (e.g., < 2 seconds). (Specific NFRs needed for formal testing).	Medium
TC_NF_004	Non-Functional	Security	Check for sensitive information in API responses	Execute various GET requests.	1. Inspect API responses for any unintentionally exposed sensitive data (e.g., password hashes, excessive internal	No sensitive data that shouldn't be there is exposed. E.g., User serializer correctly excludes password.	High

					details).		
TC_NF_005	Non-Functional	Scalability (Conceptual)	System handles concurrent requests for read operations (Conceptual)	Multiple users accessing list endpoints or dashboards simultaneously.	1. (Simulate if possible, or discuss design) Assess how the system would handle N concurrent users making GET requests.	System remains responsive. No significant degradation in performance. Database queries are optimized. (This would typically require performance testing tools).	Medium

7.3 UI Test Cases (10 Test Cases)

These are general UI test cases applicable if a frontend consumes this backend.

ID	Category	Sub-Category	Test Case Title	Preconditions	Steps	Expected Results	Priority
TC_UI_001	UI	Login	Verify login page elements display correctly	User is on the login page.	1. Observe login form.	Username field, password field, login button are visible and	High

						correctly labeled.	
TC_UI_002	UI	Navigation	Verify main navigation menu items are present and functional	User is logged in.	1. Check main navigation menu (e.g., Projects, Tasks, Dashboard). 2. Click on each menu item.	Menu items are displayed as expected. Clicking navigates to the correct page/section.	High
TC_UI_003	UI	Forms	Verify form validation messages are displayed for incorrect input	User is on a form (e.g., Create Project).	1. Submit form with missing required fields or invalid data.	Clear, user-friendly validation messages are displayed next to the respective fields.	Medium
TC_UI_004	UI	Data Display	Verify lists (e.g., Project list) are displayed in a readable table/grid	User is viewing a list page.	1. Observe the data table/grid.	Data is well-organized. Columns are aligned. Important information is visible. Pagination controls (if any) are function	Medium

						al.	
TC_UI_005	UI	Buttons	Verify buttons (Save, Cancel, Delete) are consistently styled and labeled	User interacts with forms or items with actions.	1. Observe buttons across different pages.	Buttons have consistent appearance and clear labels indicating their action. Confirmation dialogs appear for destructive actions (e.g., Delete).	Medium
TC_UI_006	UI	Charts	Verify charts are rendered correctly and are understandable	User is viewing a page with charts.	1. Observe the chart display.	Charts are displayed without visual glitches. Labels, legends, and data points are clear and legible. Tooltips appear on hover (if applicable).	Medium
TC_UI_007	UI	Responsiveness	Verify basic	-	1. (If frontend	Layout adapts	Low

			responsiveness on different screen sizes (conceptual)		available) Resize browser window or use developer tools to simulate different devices.	reasonably to different screen sizes. No major element overlaps or content truncation. (Requires actual frontend for full testing).	
TC_UI_008	UI	Error Display	Verify user-friendly display of server errors (e.g., 500)	A server error occurs and is propagated to UI.	1. Trigger a server error scenario.	A generic, user-friendly error page or message is displayed (e.g., "Something went wrong, please try again later.") instead of a raw error stack trace.	Medium
TC_UI_009	UI	Accessibility	Verify basic keyboard navigation (concep	User is on a page with interactive element	1. Attempt to navigate and interact with	All interactive elements are focusable and	Low

			tual)	s.	form fields, buttons, links using only the keyboard (Tab, Enter, Space).	operable via keyboard. Focus indicator is visible. (Requires actual frontend).	
TC_UI_010	UI	Readability	Verify text is legible and has good contrast	-	1. Observe text content across various pages.	Font sizes are adequate. Sufficient contrast between text and background colors.	Medium

8. Defect Management

Defect Lifecycle:

1. **New:** Defect is initially reported.
2. **Open:** Defect is triaged, validated, and assigned to a developer.
3. **In Progress:** Developer is actively working on a fix.
4. **Fixed:** Developer has implemented a fix and deployed it to the test environment.
5. **Retest:** QA tests the fix.
6. **Closed:** If the fix is verified, the defect is closed.
7. **Reopened:** If the fix is not working, the defect is reopened and assigned back to the developer.
8. **Deferred:** Defect is valid but will be fixed in a later release.
9. **Rejected:** Defect is deemed invalid, duplicate, or as-designed.

Defect Report Template:

Field	**Description**	**Example**

Defect ID	Unique identifier	EMS_D_001
Summary	Concise title of the defect	Owner cannot create project if description field is empty (API returns 500)
Description	Detailed explanation of the defect	When an owner attempts to create a new project via POST /api/v1/projects/ and omits the optional 'description' field, the API returns a 500 Internal Server Error instead of a 201 or a specific validation error if description was implicitly required somewhere.
Module/Feature	Application area where the defect occurred	Project Management
Steps to Reproduce	Numbered steps to replicate the defect	1. Login as Owner. 2. Send POST request to /api/v1/projects/ with payload: {"name": "Test Project No Desc"}. 3. Observe response.
Expected Result	What the application should have done	API should return 201 Created, as 'description' is optional. Or, if it became mandatory, a 400 Bad Request with a clear validation message.
Actual Result	What the application actually did	API returns 500 Internal Server Error. Response body: (include if relevant)
Severity	Impact of the defect (Critical, High, Medium, Low)	High
Priority	Urgency to fix the defect (High, Medium, Low)	High
Status	Current state in the defect lifecycle (New, Open, Fixed, etc.)	New
Reported By	Name of the tester who found	John Doe

	the defect	
Date Reported	Date when the defect was found	2024-05-28
Assigned To	Developer responsible for fixing	(To be assigned)
Build Version	Software version where the defect was found	v1.0.0-alpha
Environment	Test environment details (e.g., Test Server, Browser/API Client version)	Test Server, Postman v10.x
Attachments	Screenshots, log files, request/response payloads	create_project_500_error.png, request_payload.json

List of Defects:

Defect ID	**Summary**	**Severity**	**Priority**	**Status**
EMS_D_001	Owner Dashboard: Active projects count incorrect when tasks are deleted	Medium	Medium	Fixed
EMS_D_002	Task API: Filtering by project_name is case-sensitive	Low	Low	Fixed
EMS_D_003	User Profile: phone_number not returned for Admin role	Medium	Medium	Fixed
EMS_D_004	WorkLog: Cannot log work for a project if no tasks exist in it	High	High	Fixed

9. Test Deliverables

- Testing Strategy Document (this document).
- Manual Test Cases.
- Smoke Test Suite.
- Defect Reports.
- Test Summary Report (upon completion of testing cycles).

10. Conclusion

This testing strategy provides a framework for manually validating the Employee Management System. Adherence to this strategy, including thorough execution of test cases and diligent defect management, will contribute significantly to the quality and reliability of the application. While this plan focuses on manual testing, future iterations should consider incorporating automated API testing (expanding on the existing tests.py) for regression suites and continuous integration.