

**Headcount**

**Allocation**

**Project**

Tests Document

Nofar Cohen Sedek

Gili Veltz

Noa Malul

Hadas Printz

Version: 1.0

Introduction

This document outlines the complete test plan for validating the Headcount Allocation system. It ensures that all functionalities described in the system requirements are tested systematically. The plan includes unit, integration, acceptance, and non-functional tests, and establishes traceability between tests and system use cases.

Test Strategy

We used the following test types:

* **Unit Testing**: Testing individual functions and methods.
* **Integration Testing**: Ensuring modules work together correctly.
* **Acceptance Testing**: Validating end-to-end business scenarios.
* **Non-Functional Testing**: Evaluating performance, security, and usability.

Unit Tests

Employee

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Function** | **Input** | **Expected result** | **Test Type** |
| 1 | EncryptPassword (password) | String password | Returned password != password.  Return password not null. | success |
| 2 |  | Null | Gets error message | Fail |
| 3 | VerifyPassword (password, hashedpassword) | Password and the password hashed | True | Success |
| 4 |  | Password and different password hashed | False | Fail |
| 5 | AssignEmployeeToRole(role) | role | Roles contains the role | Success |
| 6 |  | Null | Gets error message | Fail |
| 7 | EditEmail (email) | Valid email | Email = new email | Success |
| 8 |  | Invalid email | Gets error message  Email = email | Fail |
| 9 | EditPhoneNumber(phonenumber) | Valid phoneNumber | phoneNumber = new phoneNumber | Success |
| 10 |  | Valid phoneNumber | Gets error message | Fail |
| 11 | EditTimeZone(TimeZone) | Valid TimeZone | TimeZone = new TimeZone | Success |
| 12 |  | Invalid TimeZone | Gets error message | Fail |
| 13 | EditYearOfExperience(YearOfExperience) | Valid YearOfExperience | YearOfExperience = new YearOfExperience | Success |
| 14 |  | Invalid YearOfExperience | Gets error message | Fail |
| 15 | EditJobPercentage(jobPercentage) | Valid jobPercentage | jobPercentage = new jobPercentage | Success |
| 16 |  | Invalid jobPercentage | Gets error message | Fail |
| 17 | AddSkill(skill) | Valid skill | Skills contains new skill | Success |
| 18 |  | null | Gets error message | Fail |
| 19 | RemoveSkill(skillId) | Exists skill id | Skills no longer contain the skillId | Success |
| 20 |  | Non exists skill id | Gets error message | Fail |
| 21 | AddLanguage(language) | Valid language | Languages contains new language | Success |
| 22 |  | null | Gets error message | Fail |
| 23 | RemoveLanguage(languageId) | Exists language id | Languages no longer contain the LanguageId | Success |
| 24 |  | Non exists language id | Gets error message | Fail |

Project

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Function** | **Input** | **Expected result** | **Test Type** |
| 1 | AddRoleToProject() | roleName, timeZone, foreignLanguages, skills, yearsExperience, jobPercentage, description  all valid | Roles contains role with those fields.  DB contains the new role. | success |
| 2 |  | Invalid field | Gets error message, Roles and DB not contains the new role. | Fail |
| 3 | RemoveRole() | Exists roleId | Roles not contains the roleId.  DB not contains the roleId | Success |
| 4 |  | Non exists roleId | Gets error message | Fail |
| 5 | EditProjectName() | String name | Name = new name | Success |
| 6 |  | Null | Gets error message  Name = name | Fail |
| 7 | EditProjectDescription() | String description | description = new description | Success |
| 8 |  | Null | Gets error message  description = description | Fail |
| 9 | EditProjectDate() | Valid date | date = new date | Success |
| 10 |  | Invalid date | Gets error message  Date = date | Fail |
| 11 | EditProjectRequieredHours() | Valid hours | hours = new hours | Success |
| 12 |  | Invalid hours | Gets error message  Hours = hours | Fail |

Integration Tests

In those tests we test how our modules works together, so we will test the ManagerFacade class which contains calls for many base classes and cooperate with them:

ManagerFacade

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Function** | **Input** | **Expected result** | **Test Type** |
| 1 | CreateProject () | projectName, description, date, requiredHours, roles  all valid | Projects contains project with this projectName.  DB context has project | success |
| 2 |  | Invalid field | Gets error message, Projects and DB not contains the new project. | Fail |
| 3 | EditProjectName() | Exists projectId, newProjectName | The project with projectId is updated with the newProjectName  Project in DB updates name | Success |
| 4 |  | Non exists projectId | Gets error message  Project name remains the same | Fail |
| 5 | EditProjectDescription() | Exists projectId, newProjectDescription | The project with projectId is updated with the newProjectDescription  Project in DB updates Description | Success |
| 6 |  | Null projectDescription | Gets error message  Project description remains the same | Fail |
| 7 | EditProjectDate() | Exists projectId, newProjectDate | The project with projectId is updated with the newProjectDate  Project in DB updates date | Success |
| 8 |  | Invalid date | Gets error message  Project date remains the same | Fail |
| 9 | EditProjectRequiredHours() | Exists projectId, newProjectRequiredHours | The project with projectId is updated with the newProjectRequiredHours  Project in DB updates reguiredHours | Success |
| 10 |  | Invalid hours | Gets error message  Project required hours remains the same | Fail |
| 11 | DeleteProject() | Exists projectId | The project with projectId is deleted from Projects dictionary and database | Success |
| 12 |  | NonExists projectId | Gets error message  Project not deleted from dictionary and DB | Fail |
| 13 | AddRoleToProject() | roleName, projectId, timeZone, foreignLanguages, skills, yearsExperience, jobPercentage, description  All valid | A new Role is added to the specified project.  In the DB have a role with that roleName | Success |
| 14 |  | Invalid field | Gets error message, Roles and DB not contains the new role. | Fail |
| 15 | RemoveRole() | Exists projectId, roleId | The role with roleId is removed from the specified project.  DB removes the role | Success |
| 16 |  | Non exists projectId/ roleId | Gets error message, Roles and DB contains the role. | Fail |
| 17 | GetAllRolesByProject() | Exists projectId | Returns all roles of the specified project | Success |
| 18 |  | Non Exists projectId | Gets error message, return nothing | Fail |
| 19 | AssignEmployeeToRole() | Exists employeeId, role | The employee is assigned to the specified role and also in the EmployeeRole table in DB | Success |
| 20 |  | Non exists employeeId, role | Gets error message, employee is not assigned | Fail |
| 21 | GetAllProjects() |  | Returns a list of all projects | Success |
| 22 | EmployeesToAssign() | Exists role | Returns a list of employees sorted by their eligibility for the role | Success |
| 23 |  | Invalid role | Gets error message, return nothing | Fail |
| 24 | GetProjectById() | Exists projectId | Returns the project corresponding to the projectId | Success |
| 25 |  | Non exists projectId | Gets error message, return nothing | Fail |
| 26 | CreateEmployee() | name, phoneNumber, email, timezone, foreignLanguages, skills, yearsExperience, jobPercentage, isManager  All valid | A new employee is created and added to the employees dictionary and to the DB | Success |
| 27 |  | Invalid feild | Gets error message, Employees and DB not contains the new employee. | Fail |
| 28 | DeleteEmployee() | Exists employeeId | The employee with employeeId is removed, and DB removes the employee | Success |
| 29 |  | Non exists employeeId | Gets error message, Employees and DB contains the employee. | Fail |
| 30 | Login() | Exists userName, password of manager | Returns true | Success |
| 31 |  | Exists userName, password of employee | Returns false | Success |
| 32 |  | Incorrect username, password | Throws exception | Fail |
| 33 | EditEmail() | Exists userId, newEmail | The employee with employeeId is updated with the newEmail  employee in DB updates email | Success |
| 34 |  | Invalid email | Gets error message  Employee email remains the same | Fail |
| 35 | EditPhoneNumber() | Exists userId, newPhoneNumber | The employee with employeeId is updated with the newPhoneNumber  employee in DB updates phoneNumber | Success |
| 36 |  | Invalid phoneNumber | Gets error message  Employee phoneNumber remains the same | Fail |
| 37 | EditTimeZone() | Exists userId, newTimeZone | The employee with employeeId is updated with the newTimeZone  employee in DB updates TimeZone | Success |
| 38 |  | Invalid TimeZone | Gets error message  Employee TimeZone remains the same | Fail |
| 39 | EditYearOfExpr() | Exists userId, newyearOfExpr | The employee with employeeId is updated with the newyearOfExpr  employee in DB updates yearOfExpr | Success |
| 40 |  | Invalid yearOfExpr | Gets error message  Employee yearOfExpr remains the same | Fail |
| 41 | EditJobPercentage() | Exists userId, newJobPercentage | The employee with employeeId is updated with the newJobPercentage  employee in DB updates JobPercentage | Success |
| 42 |  | Invalid JobPercentage | Gets error message  Employee JobPercentage remains the same | Fail |
| 43 | AddSkill() | userId, Valid newSkill | The employee with userId has the new skill added.  employee in DB add skill | Success |
| 44 |  | Invalid newSkill | Gets error message  Employee skills remains the same | Fail |
| 45 | RemoveSkill() | userId, Exists skillId | The employee with userId remove the skill.  employee in DB remove skill | Success |
| 46 |  | Non exists skillId | Gets error message  Employee skills remains the same | Fail |
| 47 | AddLanguage() | userId, valid newLanguage | The employee with userId has the new language added. | Success |
| 48 |  | Invalid newLanguage | Gets error message  Employee languages remains the same | Fail |
| 49 | RemoveLanguage() | userId, Exists languageId | The employee with userId remove the language.  employee in DB remove language | Success |
| 50 |  | Non exists languageId | Gets error message  Employee language remains the same | Fail |
| 51 | AddTicket() | employeeId, employeeName, startDate, endDate, description  all valid | A new ticket is created, added to the Tickets dictionary and the database. | Success |
| 52 |  | Invalid fields | Gets error message  No ticket added to Tickets and DB | Fail |
| 53 | CloseTicket() | Exists ticketId | The ticket with ticketId is marked as closed, and the status is updated in the Tickets dictionary and database. | Success |
| 54 |  | Non Exists ticketId | Gets error message  No changes | Fail |
| 55 | GetOpensTickets() |  | Returns a list of open tickets from the Tickets dictionary. | Success |
| 56 | GetOpensTickets5days() |  | Returns a list of open tickets that have a start date within the next 5 days. | Success |

Parallel Testing

In those tests we will test that our system is suitable for working with big amount of users who will use it in parallel.

In those tests we will test the big functions of the system with 100 Threads which will do them together.

Examples of functions we will test:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Function** | **Input** | **Expected result** | **Test Type** |
| 1 | Login() | username, password | 100 users can login together and they all in the DB in the end of the test | success |
| 2 | AddProject() | projectName, description, date, requiredHours, roles  all valid | 100 projects can be created together and they are all in the DB in the end of the test | success |
| 3 | RemoveProject() | Exists projectId | 100 projects can be deleted together and they are all not in the DB in the end of the test | Success |
| 4 | OpenTicket() | employeeId, employeeName, startDate, endDate, description | 100 tickets can be created together and they are all in the DB in the end of the test | Success |

Acceptance Tests

In our acceptance testing we will test the service layer using proxy.

We will test all the functions of the HeadCountService using proxy-

The proxy will hold our service and return true / false according to the Response object that returns from the service function- if it contains error or value.

Our tests will check if the value returned from the proxy match the expected result of the function according to the input.

Non-functional Tests

|  |  |  |
| --- | --- | --- |
| **ID** | **Test Description** | **Expected Result** |
| 1 | Critical screens (Assign employee, Open Ticket, and Handle Ticket) must load within 2 seconds under a load of up to 10 concurrent users. | Run the system with 10 users and click on the relevant pages with time counting below 2 seconds. |
| 2 | The system must handle up to 50 simultaneous requests without performance degradatio | Run the system with loop of 50 users, and perform action with thread on each user- add employee. Expect 50 employees in the DB. |
| 3 | The system must ensure no data loss during an unplanned shut down and recover to full functionality within 5 minutes. | Run the system, close the internet so the system will shut down.  Open the internet and make sure we can add project within 5 minutes. |
| 4 | The interface must be responsive and compatible with modern browsers (Chrome, Firefox, Edge). | Run the system on each browser and verify the system functionally fully works. |
| 5 | Access to administrative features must be restricted based on user roles and permissions. | Try to login with username which is not in the DB and verify the system won't let us in.  Log in with employee's username and verify we don’t have access to manager's pages. |
| 6 | The codebase must be well-documented to allow new developers to onboard quickly. | Let our costumer which is a software developer also, go over the code and change the main algorithm to make sure our code is easy to onboard. |