

**Headcount**

**Allocation**

**Project**

Application Design Document

Nofar Cohen Sedek

Gili Veltz

Noa Malul

Hadas Printz

Version: 1.0

Contents

Chapter 1 – Use cases .…………………………………………….………………………..…….………. 3

Chapter 2 – System architecture…………………………………………….……………………...... 19

Chapter 3 – Data model …………………………………………………..………..…………….……… 22

* Description of data objects ………………..……………………………………….………… 22
* Data objects relationships …………………………………………………………..………… 25
* Data Bases ………………………………………………………………………………...………. 26

Chapter 4 – Behavioral Analysis …………………………………………………….………..………. 32

* Sequence diagrams………………………………………………….……………..….……….. 32
* Events ……………………………………………………………………….………………………. 42

Chapter 5 – Object Oriented Analysis…………………………………………………….…….. 46

* Class Diagrams ……………………………………………………………………………..……. 46
* Class Description ………………………………………………………………….…………….. 48
* Packages…………………………………………………………………………………………….. 62
* Unit testing ………………………………………………………………………….……………….63

Chapter 6 - User Interface …………………………………….……………………………………….. 64

Chapter 7 – Testing ………………………………………………………………………………….…….. 67

Chapter 8- Algorithm correctness prove ……………………………….

Chapter 1

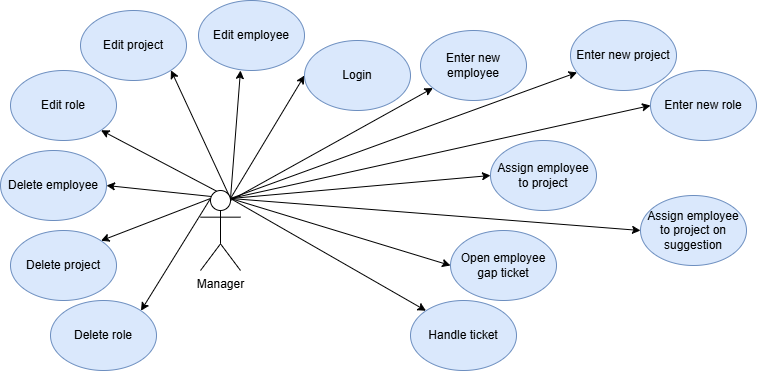
Use Cases

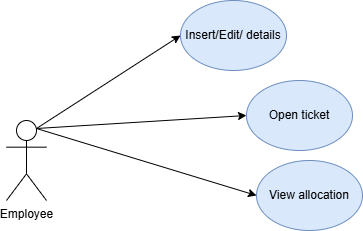
# User profile:

The system users are project managers / human resource managers. Technological skills: good knowledge of project management tools. Works in a mid-to-large company with multiple on-going projects and employees (even projects and employees around the world).

In some of those companies' employees work on more than one project simultaneously.

Some users will have good computer skills, but some users will be far from the computer world, so the system must be very user friendly.





תמונה שמכילה טקסט, קו, עיגול, תרשים

התיאור נוצר באופן אוטומטי

# Manager Use - Cases

## LogIn:

### Description:

Login the registered user into the system.

### Actor:

* User.

### Pre- conditions:

* The user's credentials are saved in the system and the user is a manager.

### Post- conditions:

* The home page of the system is visible to the user.

### Main scenario:

1. The user enters the Login page.
2. The user enters username and password.
3. The system checks if the user exists in the system. If so, then the continue the use case.  
   If the user not exists in the system, the system shows error message with the error description.
4. The system redirects the user to the manager page.

## Enter new employee:

### Description:

The manager enters a new employee to the system- with all its attributes.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as a manager.
* The employee is not in the system.
* All the attributes are correct and logical (hours, age…)

### Post- conditions:

* The new employee is in the system.
* The manager can see the new employee and assign it.

### Main scenario:

1. The manager logs in to the system.
2. The manager selects the option of adding new employee.
3. The manager enters all the requested attributes of the employee.
4. The manager enters username and password for the employee to be able to login to the system.
5. The system checks if the employee exists in the system. If so- the system shows error message to the user.  
   If the employee not exists in the system- then continue the use case.
6. The system checks for correctness of the fields.
7. The system saves the new employee and add it's credentials to the DB, and displays the new employee's details to the manager.

## Enter new project:

### Description:

The manager enters a new project to the system- with all its attributes.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as a manager.
* The project is not in the system.
* All the attributes are correct and logical (hours, age…)

### Post- conditions:

* The new project is in the system.
* The manager can see the new project and assign it.

### Main scenario:

1. The manager logs in to the system.
2. The manager selects the option of adding new project.
3. The manager enters all the requested attributes of the project.
4. The system checks if the project exists in the system. If so- the system shows error message to the user.  
   If the project not exists in the system- then continue the use case.
5. The system checks for correctness of the fields.
6. The system saves the new project and displays it's details to the manager.

## Enter a new role:

### Description:

The manager enters a new role to a project in the system- with all its attributes.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as a manager.
* The project is in the system.
* The role is not exist in the system.
* All the attributes are correct and logical.

### Post- conditions:

* The new project role is in the system.
* The manager can see the new role and assign it.

### Main scenario:

* 1. The manager logs in to the system.
  2. The system presents to the user all the projects to select project.
  3. The manager selects project to add role to it.
  4. The manager selects the option to add a new role in project.
  5. The manager enters all the requested attributes of the role including skill, and for each skill the required level and the priority of this skill.
  6. The system checks if the role exists in the system. If so- the system shows error message to the user.  
     If the project not exists in the system- then continue the use case.
  7. The system checks for correctness of the fields.
  8. The system saves the new role and displays it's details to the manager.

## Edit role:

### Description:

The manager edits the attributes of an existing role.

### Actor:

* The user.

### Pre- conditions:

* The user has access to the system, and is logged in as a manager.
* The role is in the system.
* All the attributes are correct and logical.

### Post- conditions:

* The details are updated.

### Main scenario:

1. The manager logs in to the system.
2. The system presents to the user all the projects to select project.
3. The manager enter to a project.
4. The system presents to the user all the projects roles to select role.
5. The manager selects role to edit.
6. The manager click on edit role button and edit the role as he wished.
7. The system checks for correctness of the fields.
8. The system saves change and displays it to the manager.

## Edit employee:

### Description:

The manager edits the attributes of an existing employee.

### Actor:

* The user.

### Pre- conditions:

* The user has access to the system, and is logged in as a manager.
* The employee is in the system.
* All the attributes are correct and logical (hours, age…)

### Post- conditions:

* The details are updated.

### Main scenario:

1. The manager logs in to the system.
2. The system presents the user all the employees to select employee for editing.
3. The manager selects the relevant employee and click edit.
4. The manager edits all the relevant attributes.
5. The system checks for correctness of the fields.
6. The system saves change and displays it to the manager.

## Edit project:

### Description:

The manager edits the attributes of an existing project.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as a manager.
* The project is in the system.
* All the attributes are correct and logical (hours, age…)

### Post- conditions:

* The details are updated.

### Main scenario:

1. The manager logs in to the system.
2. The system presents the user all the projects to select project for editing.
3. The manager selects the relevant project and click edit.
4. The manager edits all the relevant attributes.
5. The system checks for correctness of the fields.
6. The system saves change and displays it to the manager.

## Delete an employee:

### Description:

The manager deletes employee.

### Actor:

* User.

### Pre- conditions:

* The User has access to the system, and is logged in as a manager.
* The employee is in the system.

### Post- conditions:

* The employee is deleted.
* New gaps according to the delete are updated and visible to the manager.

### Main scenario:

1. The manager logs in to the system.
2. The system presents the user all the employee to select employee for delete.
3. The manager selects the relevant employee and click delete.
4. The system deletes employee.
5. The system brings all projects, and for each project, for each role, if the employee is assigned to this role then the system unassign it.

## Delete a project:

### Description:

The manager deletes project.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as a manager.
* The project is in the system.

### Post- conditions:

* The project is deleted.
* New gaps according to the delete are updated and visible to the manager.

### Main scenario:

* 1. The manager logs in to the system.
  2. The system presents the user all the projects to select project for delete.
  3. The manager selects the relevant project and click delete.
  4. The system deletes project.
  5. For each role in the project, the system deletes that role from the employee assigned to it if there is.

## Delete role:

### Description:

The manager delete role.

### Actor:

* The user.

### Pre- conditions:

* The user has access to the system, and is logged in as a manager.
* The role is in the system.

### Post- conditions:

* The role is deleted.

### Main scenario:

1. The manager logs in to the system.
2. The system presents to the user all the projects to select project.
3. The manager enter to a project.
4. The system presents to the user all the projects roles to select role.
5. The manager selects role to delete.
6. The manager click on delete role button.
7. The system saves change.
8. The system remove this role for the employee assigned to it if exists one.

## Assign employee to role:

### Description:

The manager assigns employee to existing role.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as manager.
* The employee and role are in the system.

### Post- conditions:

* The employee is assign to the role.

### Main scenario:

1. The manager logs in to the system.
2. The manager selects the option of view projects.
3. The system shows all existing projects.
4. The manager selects the relevant project.
5. The system shows all existing roles belongs to the project that was selected.
6. The manager selects the relevant role that he wants to assign the employee to.
7. The manager enters the employee he wants.
8. The system saves the new assignment and displays it to the manager.

## 11. Assign employee to role based on suggestion of the system:

### Description:

The manager assign employee to role according to the suggestion of the system.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as manager.
* The relevant role for assignment is in the system.

### Post- conditions:

* An employee is assign to the role.

### Main scenario:

1. The manager logs in to the system.
2. The manager selects the option of view projects.
3. The system shows all existing projects.
4. The manager selects the relevant project.
5. The system shows all existing roles belongs to the project that was selected.
6. The manager selects the relevant role that he wants to assign the employee to.
7. The manager selects the option of assigning employee based on system suggestion.
8. The system calculates the score of each employee in the system based on the employee's and role's attributes, and show the manager the relevant employees order by score.
9. The manager selects the employee he wants to assign to the role.
10. The system saves the new assignment and displays it to the manager.

## Open employee gap "ticket"

### Description:

The manager open a ticket of employee's absence.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as a manager.
* The employee is in the system.

### Post- conditions:

* The ticket is in the system and visible to the manager.
* The gaps in the projects where the employee is assigned are visible to the manager in the gaps page.

### Main scenario:

1. The manager logs in to the system.
2. The manager selects the option of view employees.
3. The system shows all existing employees.
4. The manager selects the relevant employee.
5. The manager selects the option of opening new employee's absence ticket.
6. The manager enters the relevant dates and reason of leaving.
7. The system creates gaps tickets for the employee with all the roles the employee is assigned to and the details of the dates.

## Handle ticket:

### Description:

The manager closes ticket by changing assignment of employees for each role in the ticket.

### Actor:

* User.

### Pre- conditions:

* The manager has access to the system, and is logged in as a manager.
* A gap ticket is open and visible to the manager.

### Post- conditions:

* Optional: the ticket is closed.
* Optional: new gap ticket is created.
* Changes of employees assignments are updated in the system and visible to the manager.

### Main scenario:

1. The manager logs in to the system.
2. The manager selects the option of view gaps tickets.
3. The system presents all the gaps tickets.
4. The manager selects option of solving a ticket for a specific ticket.
5. The system shows all the roles effected from the gap.
6. The manager goes into each role and:
   1. The manager assign employee to role according to use case "assign employee to role based on suggestion of the system"or **"**Assign employee to role"
   2. The system updates the change and shows the manager update image of the employee's assignment, projects gaps and utilization.
7. The system updates the new assignments and show the updated image to the manager.

# Employee Use - Cases

## Log in Employee:

### Description:

Login the registered user into the system.

### Actor:

* User.

### Pre- conditions:

* The user's credentials are saved in the system and he is an employee.

### Post- conditions:

* The home page of the system is visible to the user.

### Main scenario:

1. The user enters the Login page.
2. The user enters username and password.
3. The system checks if the user exists in the system. If so, then the continue the use case.  
   If the user not exists in the system, the system shows error message with the error description.
4. The system redirect the user to the employee page.

## Edit Employee details:

### Description:

The employee edit his details to the system- with all its attributes.

### Actor:

* User.

### Pre- conditions:

* The employee has access to the system, and is logged in as an employee.
* The employee is in the system.
* All the attributes are correct and logical (hours, age…)

### Post- conditions:

* The employee's details are in the system.
* The manager can see the employee and assign it.

### Main scenario:

1. The employee logs in to the system.
2. The employee selects the option of edit employee's details.
3. The employee enters all the requested attributes.
4. The system checks for correctness of the fields.
5. The system saves the new details and displays them to the manager.

## Open ticket:

### Description:

The employee opens a ticket for his absence.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as an employee.
* The employee's details are in the system.

### Post- conditions:

* The ticket is in the system and visible to the manager.
* The gaps in the projects where the employee is assigned are visible to the manager in the gaps page.

### Main scenario:

1. The employee logs in to the system.
2. The employee selects the option of opening a new absence ticket.
3. The employee enters the relevant details such as dates and reason for leaving.
4. The system creates gaps tickets for each project the employee is assigned to, with details of the gaps.

## View employee's allocation to projects:

### Description:

The employee views his allocation to projects- days of week, language, and percentages of work.

### Actor:

* User.

### Pre- conditions:

* The user has access to the system, and is logged in as an employee.
* The employee's details are in the system.

### Post- conditions:

* The employee gets a page with all the projects he is allocated to with all their details.

### Main scenario:

1. The employee logs in to the system.
2. The employee selects the option of view his current allocation status.
3. The system loads a page with all the employee's allocated projects and details.

# Big Management Use - Cases

## View Information per project and per employee:

### Description:

The manager views details about all the employees, the working hours for each project and more relevant statistics.

### Actor:

* The Big Manager.

### Pre- conditions:

* The Big Manager has access to the system, and is logged in.

### Post- conditions:

* The Big Manager gets a page with all the relevant information.

### Main scenario:

1. The Big Manager logs in to the system.
2. The Big Manager selects the option of view the employees and projects information.
3. The system loads a page with all the relevant information.

Chapter 2

System Architecture

**Component diagram:**

**DAL**

**Domain**

**Service**

**User Interface**

**API**



[תמונה זו](https://commons.wikimedia.org/wiki/File:User_icon_2.svg) מאת מחבר לא ידוע ניתן ברשיון במסגרת [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)



[תמונה זו](https://commons.wikimedia.org/wiki/File:User_icon_2.svg) מאת מחבר לא ידוע ניתן ברשיון במסגרת [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

**Alerts**

**Headcount DB**

**User Interface:**   
The UI provides the end-user with an interactive platform to interact with the system. It handles input, displays outputs, and ensures seamless user experience. Deployed on the client-side browser written in React.

**API:**

The API serves as a bridge between the UI and the backend layers. It handles incoming requests from the UI, routes them to the appropriate services, and returns responses.Deployed on the server-side written in C# and Using Swagger.

**Service:**

This layer contains the core business logic of the system. It processes the data received from the API and delegates specific tasks to the Domain Layer. Deployed on the server-side alongside the API, written in C#.

**Domain:**

The Domain Layer defines the core domain models and rules of the application. It acts as the intermediary between the service and data layers, ensuring consistency and enforcing domain-specific constraints. Deployed on the server-side, written in C#.

**Alert:**

The responsibilities of the alerting components are handling events which require alerting the user of his drowsiness. Once alerting is triggered, the alerting component is responsible for the visual presentation of the alert.

**DAL:**

The DAL handles all interactions with the database, including querying, updating, and managing data. It abstracts the database implementation details from other components. Written in C#.

**Data Base:**

The database is responsible for storing and managing the system’s persistent data. It holds all application data, including user information, settings, and transactional data. Using MSSQL.

Chapter 3

Data Model

3.1 Description of Data Objects:

**Employee:**

The Employee is a structured representation of an employee's Information.

1. EmployeeId: A unique identifier assigned by the system to distinguish each employee.
2. UserName: the employee User name.
3. PhoneNumber: The employee's phone number for contact purposes.
4. Email: The employee's email address used for communication.
5. TimeZone: The working time zone of the employee – Enum with the following values: Morning/ Noon/Evening/ Flexible.
6. ForeignLanguages: A list of foreign languages the employee knows, along with their proficiency levels.
7. JobPercentage: The percentage of the employee's working hours relative to a full-time job
8. Skills A list of skills the employee possesses, with their proficiency levels.
9. Roles: A list of roles assigned to the employee within the organization.
10. YearExp: The total number of years of experience the employee has.

**Language:**

The Language class represents an individual language known by a user or employee, along with their proficiency level.

1. LanguageID: A unique identifier for the language object, used to distinguish individual entries in the system.
2. LanguageType: Specifies the type of language being represented. This is defined as an enumeration.
3. Level: Represents the proficiency level of the user in the given language. Integer 1-3.

**Project:**

The Project class represents a project in the system, encapsulating essential details such as the project's name, description, timeline, required effort, and the roles associated with it. This class serves as the foundation for managing and organizing project-related data, such as assigning roles and tracking time requirements.

1. ProjectName: The name of the project.
2. ProjectId: A unique identifier for the project within the system.
3. Description: A detailed description of the project, outlining its objectives and scope.
4. Date: The Project deadline date.
5. RequiredHours: The total number of hours required to complete the project.
6. Roles: A dictionary mapping role IDs (int) to Role objects, representing the roles required for the project.

**Role:**

The **Role** class represents a specific role or position within a project, capturing details about its association with an employee, project, and relevant requirements such as languages, skills, and experience. It acts as a link between an employee and their responsibilities in a project, while also specifying the time zone and workload distribution.

1. RoleName: The name or title of the role.
2. RoleId: A unique identifier for the role.
3. EmployeeId: The ID of the employee assigned to this role. Nullable if the role is unassigned.
4. ProjectId: The ID of the project to which this role is assigned.
5. TimeZone: Specifies the working time zone of the role.
6. ForeignLanguages: representing the foreign language requirements for the role.
7. Skills: representing the skills required for the role.
8. YearsExperience: The minimum number of years of experience required for this role.
9. JobPercentage: The percentage of full-time equivalent hours expected for this role.

**Skill:**

The Skill class represents a specific skill required or possessed by an individual. It captures essential details such as the type of skill, proficiency level, and its priority. This class is particularly useful in defining the qualifications or requirements for roles, projects, or employee profiles.

1. SkillId: A unique identifier for the skill.
2. SkillType: Specifies the type or category of the skill. This is defined as an enumeration.
3. Level: Represents the proficiency level for the skill. Integer 1-3.
4. Priority: Specifies the importance or urgency of the skill for a given role.

**ManagerFacade**:

The **ManagerFacade** class serves as a centralized entry point for managing and interacting with the core elements of the system, such as projects and employees. It implements the Singleton design pattern to ensure a single instance of this class exists, providing global access to system-wide operations and data management.

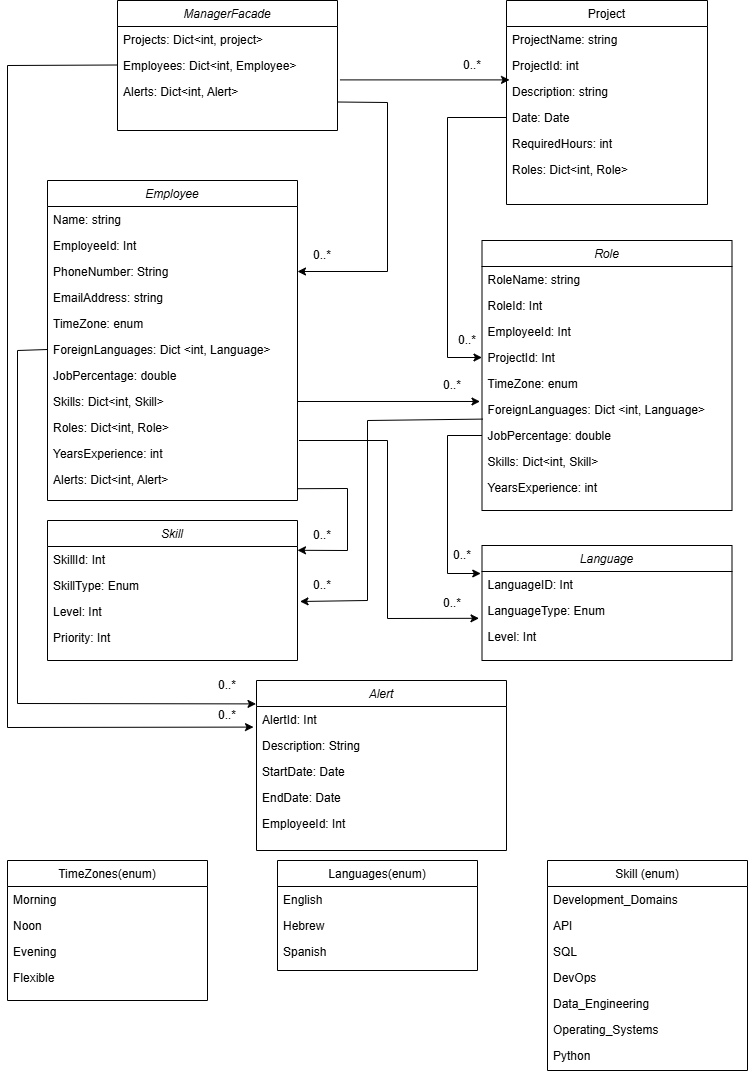
1. Projects: A dictionary mapping project IDs (int) to Project objects.
2. Employees: A dictionary mapping employee IDs (int) to Employee objects.

**Alert:**

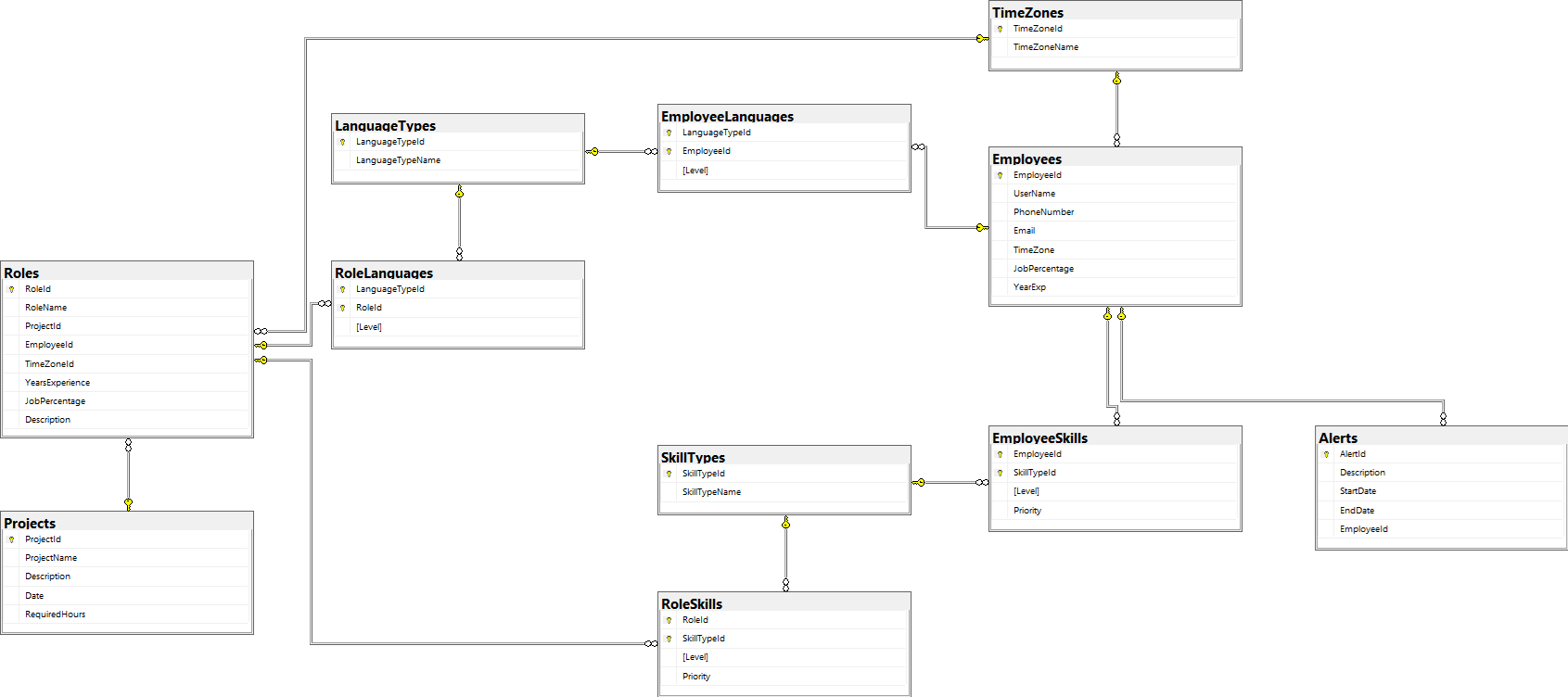
The **Alert** object is designed to manage notifications related to employee absences. It provides key details about the alert, such as its unique identifier, the description of the absence, and the associated employee information. This object helps managers stay informed and take necessary actions, like finding replacements for the absent employee's roles.

1. AlertId: A unique identifier for the alert.
2. Description: Details the reason for the alert.
3. StartDate: The date when the employee's absence begins.
4. EndDate: The date when the employee's absence ends.
5. EmployeeId: A unique identifier for the employee associated with the alert.

3.2 Data Objects Relationships:



3.3 Data Bases:

ERD:

**Tables and Their Structures**

1. **Roles**
   * **Fields:**
     + RoleId (Primary Key, Integer)
     + RoleName (String)
     + ProjectId (Foreign Key, Integer)
     + EmployeeId (Foreign Key, Integer)
     + TimeZoneId (Foreign Key, Integer)
     + YearsExperience (Integer)
     + JobPercentage (Double)
     + Description (String)
2. **Projects**
   * **Fields:**
     + ProjectId (Primary Key, Integer)
     + ProjectName (String)
     + Description (String)
     + Date (Date)
     + RequiredHours (Integer)
3. **Employees**
   * **Fields:**
     + EmployeeId (Primary Key, Integer)
     + UserName (String)
     + PhoneNumber (String)
     + Email (String)
     + TimeZoneId (Foreign Key, Integer)
     + JobPercentage (Double)
     + YearsExperience (Integer)
4. **SkillTypes**
   * **Fields:**
     + SkillTypeId (Primary Key, Integer)
     + SkillTypeName (String)
5. **RoleSkills**
   * **Fields:**
     + RoleId (Foreign Key, Integer)
     + SkillTypeId (Foreign Key, Integer)
     + Level (Integer)
     + Priority (Integer)
6. **EmployeeSkills**
   * **Fields:**
     + EmployeeId (Foreign Key, Integer)
     + SkillTypeId (Foreign Key, Integer)
     + Level (Integer)
     + Priority (Integer)
7. **LanguageTypes**
   * **Fields:**
     + LanguageTypeId (Primary Key, Integer)
     + LanguageTypeName (String)
8. **RoleLanguages**
   * **Fields:**
     + LanguageTypeId (Foreign Key, Integer)
     + RoleId (Foreign Key, Integer)
     + Level (Integer)
9. **EmployeeLanguages**
   * **Fields:**
     + LanguageTypeId (Foreign Key, Integer)
     + EmployeeId (Foreign Key, Integer)
     + Level (Integer)
10. **TimeZones**
    * **Fields:**
      + TimeZoneId (Primary Key, Integer)
      + TimeZoneName (String)
11. **Alerts**
    * **Fields:**
      + AlertId(primary Key, Integer)
      + Description(String)
      + StartDate(Datetime)
      + EndDate(Datetime)
      + EmployeeId(Foreign Key, Integer)

**Data Object Mapping**

**Mapping Objects to Database Entities**

1. **Employee**
   * **Fields:**
     + Name (UserName in Employees table)
     + EmployeeId (EmployeeId in Employees table)
     + PhoneNumber (PhoneNumber in Employees table)
     + EmailAddress (Email in Employees table)
     + TimeZone (TimeZoneId in Employees table)
     + JobPercentage (JobPercentage in Employees table)
     + YearsExperience (YearsExperience in Employees table)
     + ForeignLanguages (Maps to EmployeeLanguages)
     + Skills (Maps to EmployeeSkills)
     + Roles (Maps to Roles table)
2. **Role**
   * **Fields:**
     + RoleId (RoleId in Roles table)
     + RoleName (RoleName in Roles table)
     + EmployeeId (EmployeeId in Roles table)
     + ProjectId (ProjectId in Roles table)
     + TimeZone (TimeZoneId in Roles table)
     + ForeignLanguages (Maps to RoleLanguages)
     + Skills (Maps to RoleSkills)
     + YearsExperience (YearsExperience in Roles table)
3. **Skill**
   * **Fields:**
     + SkillId (SkillTypeId in SkillTypes table)
     + SkillType (SkillTypeName in SkillTypes table)
     + Level (Level in RoleSkills or EmployeeSkills)
     + Priority (Priority in RoleSkills or EmployeeSkills)
4. **Language**
   * **Fields:**
     + LanguageId (LanguageTypeId in LanguageTypes table)
     + LanguageType (LanguageTypeName in LanguageTypes table)
     + Level (Level in RoleLanguages or EmployeeLanguages)
5. **Project**
   * **Fields:**
     + ProjectId (ProjectId in Projects table)
     + ProjectName (ProjectName in Projects table)
     + Description (Description in Projects table)
     + Date (Date in Projects table)
     + RequiredHours (RequiredHours in Projects table)
6. **Alert**
   * **Fields:**
     + AlertId (AlertId in Alerts table)
     + Description (Description in Alerts table)
     + StartDate (StartDate in Alerts table)
     + EndDate (EndDate in Alerts table)
     + EmplyeeId (EmployeeId in Employees table)

**Main Transactions**

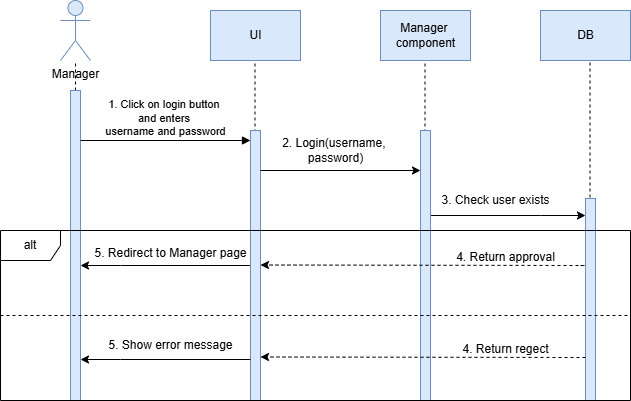
1. **Adding a New Employee**
   * Inserts a record into the **Employees** table.
   * Optional: Inserts associated records into **EmployeeSkills** and **EmployeeLanguages**.
2. **Assigning an Employee to a Role**
   * Update a record into the **Roles** table (add EmployeeId).
3. **Updating Employee Skills**
   * Updates records in the **EmployeeSkills** table with new skill levels or priorities.
4. **Creating a New Project**
   * Inserts a record into the **Projects** table.
5. **Creating a New Role**
   * Inserts a record into the **Roles** table.
   * Inserts related records into **RoleSkills** and **RoleLanguages** if applicable.

Chapter 4

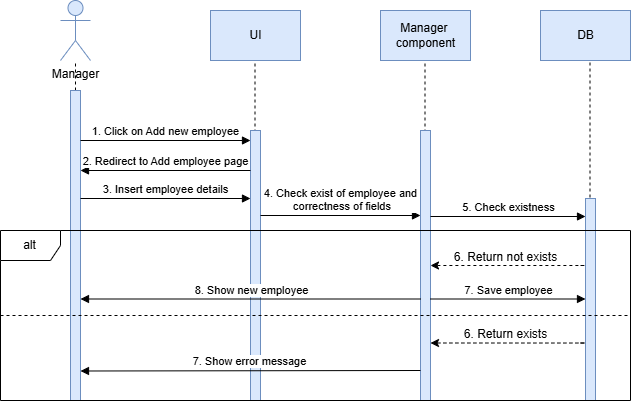
Behavioral Analysis

4.1 Sequence diagrams:

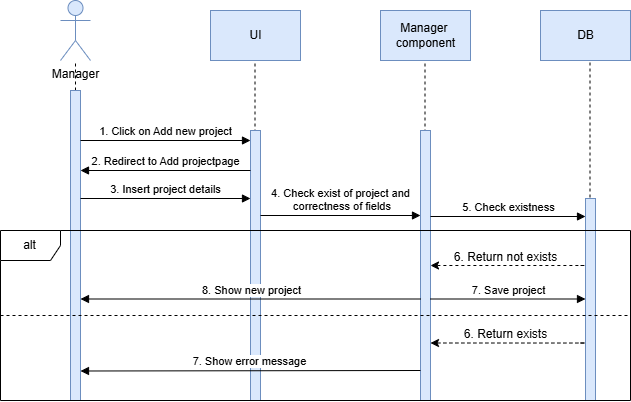
**4.1.1 UC1 Login**



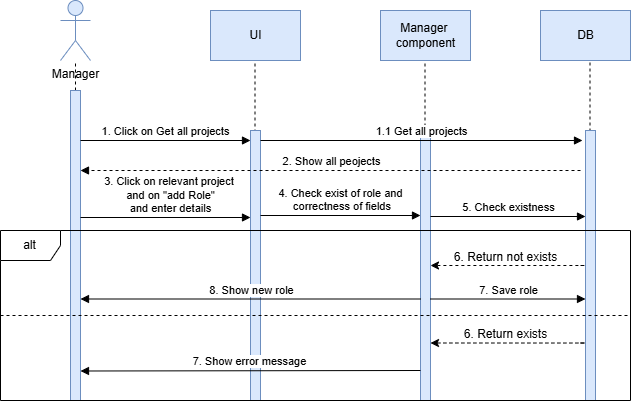
**4.1.2 UC2 Enter new employee**



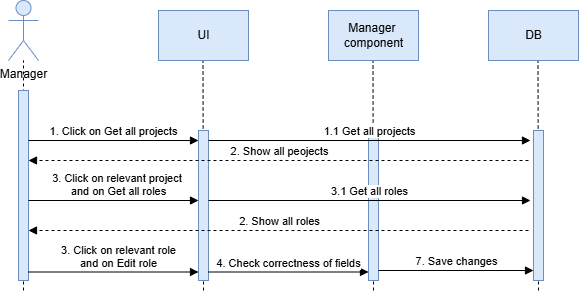
**4.1.3 UC3 Enter new project**



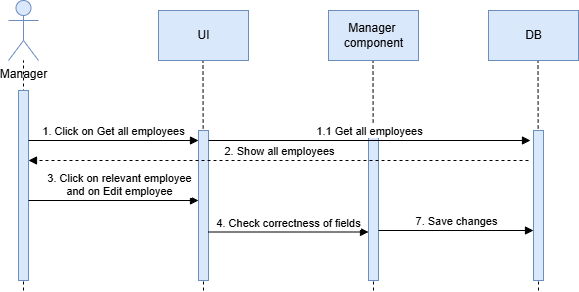
**4.1.4 UC4 Enter new role**



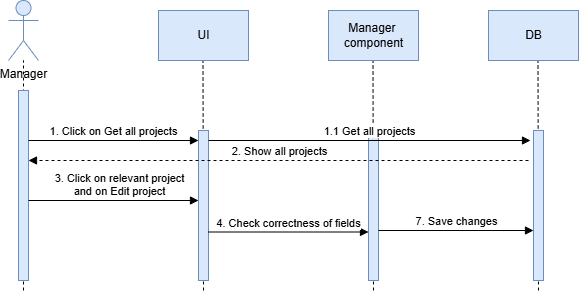
**4.1.5 UC5 Edit role**



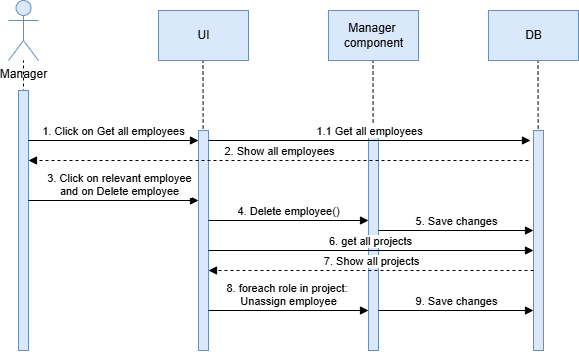
**4.1.6 UC6 Edit employee**



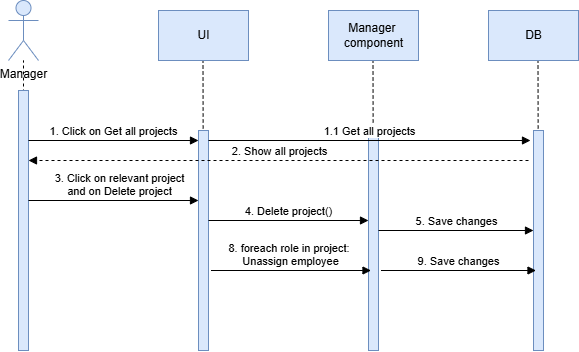
**4.1.7 UC7 Edit project**

****

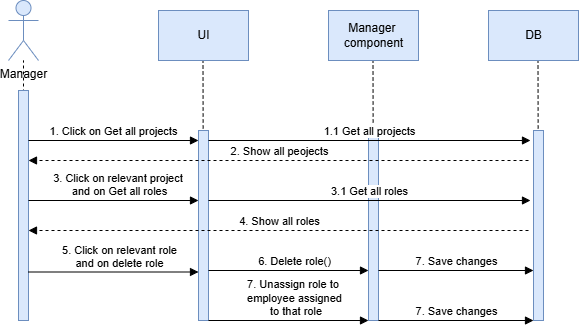
**4.1.8 UC8 Delete employee**



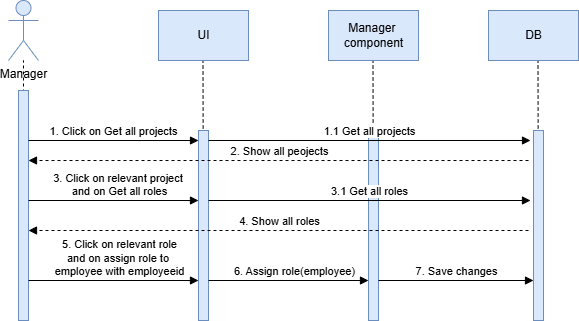
**4.1.9 UC9 Delete project**



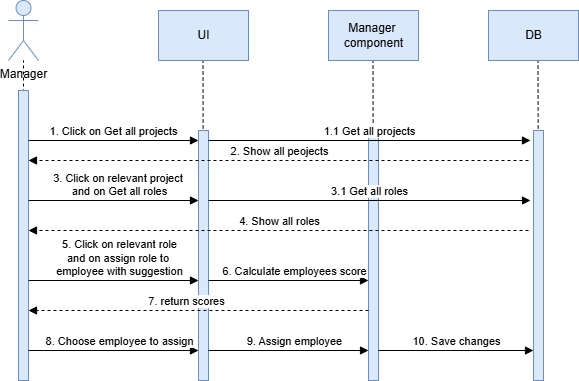
**4.1.10 UC10 Delete Role**



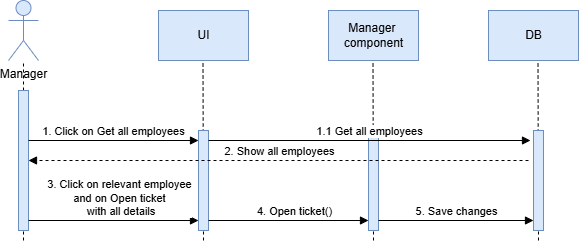
**4.1.11 UC11 Assign employee to Role**



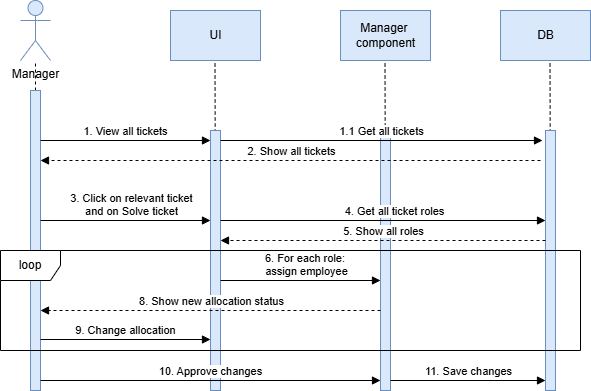
**4.1.12 UC12 Assign employee to Role based on suggestion of the system**

****

**4.1.13 UC13 Open employee gap ticket**



**4.1.14 UC14 Handle ticket**

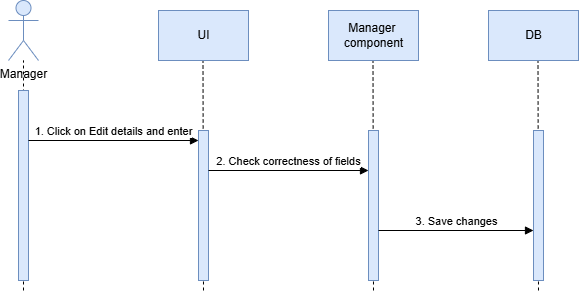
****

**4.1.15 UC15 Login employee**

תמונה שמכילה טקסט, צילום מסך, תרשים, מקביל

התיאור נוצר באופן אוטומטי

**4.1.16 UC16 Edit employee's details**

****

**4.1.17 UC17 Open gap ticket**

**תמונה שמכילה טקסט, תרשים, קו, צילום מסך

התיאור נוצר באופן אוטומטי**

**4.1.18 UC18 View employee's allocations**

תמונה שמכילה טקסט, תרשים, קו, צילום מסך

התיאור נוצר באופן אוטומטי

**4.1.19 UC19 View Information per project and per employee**

תמונה שמכילה טקסט, קו, תרשים, צילום מסך

התיאור נוצר באופן אוטומטי

4.2 Events:

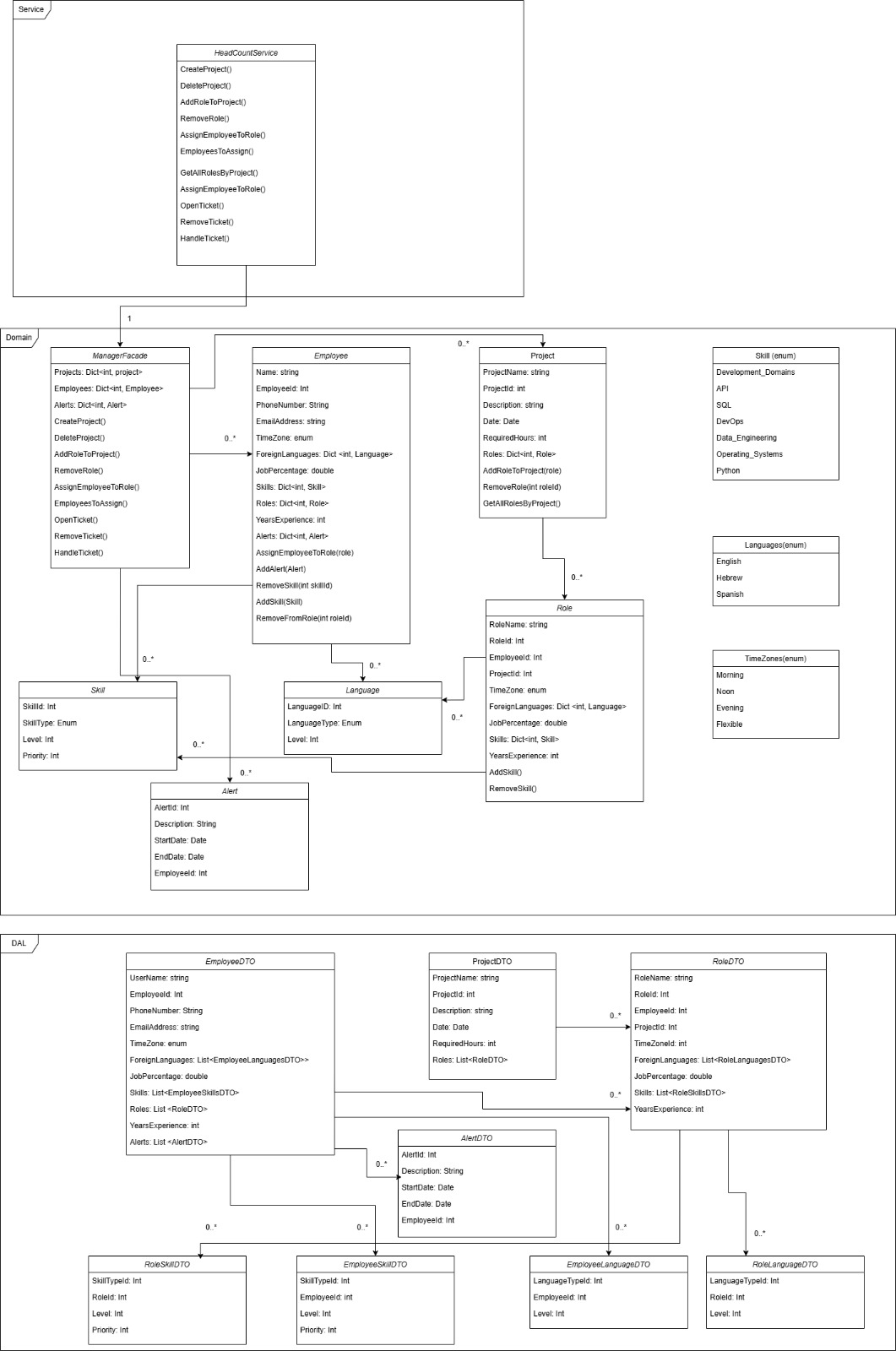
HeadCount Allocation system is mainly based on events caused by the user.

The main events of the system are:

|  |  |  |
| --- | --- | --- |
| Event | Description | Action |
| Login | The system performs user login procedure. | The user performs login with his username and password which were saved earlier in the system DB (by Admin for managers and by manager for employees when they are added).  The system checks in the DB if there is a user which match this  information, if there is one who matches, it  perform a login process and and in the end  of it, it redirect the user into the system. |
| Add new employee | The system adds employee to the employees database | The user adds new employee by inserting all it's details. While entering the details the user enters username and password for the new employee. The system checks there is no such employee in the DB and then checks the correctness of all the details. The system saves the username and password of the new employee in the DB, and adds the employee to employees dictionary and to the DB. |
| Add new project | The system adds project to the projects database | The user adds new project by inserting all it's details. The system checks there is no such project in the DB and then checks the correctness of all the details. The system adds the project to projects dictionary and to the DB. |
| Add new role | The system adds new role to an existing project | The user clicks on "get all projects", the system returns list of all the projects from the DB. The user chooses the relevant project he wants to add role to from this list and clicks on it. The system opens the project page. The user clicks on "add new role" and insert all the relevant information of that role. The system checks there is no such role in the DB and then checks the correctness of all the details. The system adds the role to the project's list of roles and to the DB. |
| Edit employee/ project/ role | The system edits details of an existing employee/ project/ role. | The user clicks on get all employees/ projects. The system returns all the existing employees/ projects. The system clicks on the relevant employee/ project he wants to edit (or the project of the role he wants to edit). The system opens the employee's/ project's page. The user clicks on "edit" (if he wants to edit role then he clicks on roles, the system returns all the existing roles of the project and the user clicks on the role he wants to edit, then the system opens the role's page). The user edits the details he wants. The system checks the correctness of the details and save the updates in the employees/ projects/ roles list and in the DB. |
| Delete employee/ project/ role | The system deletes an existing employee/ project/ role. | The user clicks on get all employees/ projects. The system returns all the existing employees/ projects. The system clicks on the relevant employee/ project he wants to delete (or the project of the role he wants to delete). The system opens the employee's/ project's page. The user clicks on "delete" (if he wants to delete role then he clicks on roles, the system returns all the existing roles of the project and the user clicks on the role he wants to delete, then the system opens the role's page). The user delete. If the user deletes an employee, the system updates for all roles and remove the assign of this employee. If the user deletes project, then the system deletes all the roles of the project and updates for each employee- delete the roles that where deleted. If the user deletes role, then the system deletes for each employee this role.  The system saves all updates in the lists and in the DB. |
| Assign employee to role | The system assigns employee to role | The user clicks on get all projects. The system returns all the existing projects. The user selects the relevant project he wants to assign role in. The user clicks on get all roles of the project. The system returns all the existing roles of the project. The user selects the role he wants to assign, and the system opens the role's page.  The user clicks on assign role. The user enters the relevant employee he wants to assign this role to.  The system updates the employee Id in the role and adds the role to the employee's roles list.  The system saves the updates in the DB. |
| Assign employee to role based on suggestion | The system assigns employee to role after suggesting the best employees to the user and letting him choose. | The user clicks on get all projects. The system returns all the existing projects. The user selects the relevant project he wants to assign role in. The user clicks on get all roles of the project. The system returns all the existing roles of the project. The user selects the role he wants to assign, and the system opens the role's page.  The user clicks on assign role based on system suggestion. The system calculates score for each employee in the system based on his attributes and the role attributes and returns the user list of all the relevant employees order by their score, such that the best employee fit to the role gets the highest score.  The user select the employee he wants to assign to the role.  The system updates the employee Id in the role and adds the role to the employee's roles list.  The system saves the updates in the DB. |
| Open gap ticket | The system opens gap ticket of employee absence. | The user clicks on get all employees. The system returns all the existing employees.  The user clicks on the relevant employee and then clicks on open gap ticket.\  The user enters all the ticket details.  The system validates the correctness of the details and saves the ticket in the DB. |
| Handle gap ticket | The system handles gap ticket by reassigning employees to roles based on gaps. | The user clicks on view all gaps. The system returns all the existing gaps. The user clicks on the relevant gap and then the system opens the gap page. The user clicks on handle gap. The system returns all the roles affected by the employee gap.  The user clicks on each role, then the system shows the role page. The user clicks on assign employee to role by suggestion or by entering employee to this role.  The system shows the user the effected role after the assignment.  When the user finishes assigning all the roles of the ticket and approves the changes, the system saves the changes and add the roles to the relevant employee's role lists. |
| View allocation of employee | The system shows the employee his roles allocations. | The user clicks on view allocation. The system returns all the roles that this employee is assigned to. |
| View information on employees and projects | The system shows the user global information on employees and projects in the system. | The user clicks on view general information on employees/ projects. The system preforms calculations such as employees utilization, unassigned roles, amount of working hours per project or employee and more statistics and shows the results to the user. |
| Notifications | The system notifies the user when gaps are opening in the system. | When there is a gap in one of the roles in a project, the system notifies the user using system notification or email.  When the date of the start of the absence of an employee arrives of getting closer, the system notifies again if the role hasn't been assigned to another employee. |

Chapter 5

Object-Oriented Analysis

5.1 Class Diagrams:

תמונה שמכילה טקסט, תרשים, תוכנית, שרטוט טכני

התיאור נוצר באופן אוטומטי

5.2 Class Description:

**ManagerFaçade:**

Acts as a central interface for managing Projects, Employees, and their associations.

Supports CRUD operations for Projects and assignment of Employees to Roles within Projects.

**Key Methods**:

* **GetInstance()**: Implements the Singleton pattern to ensure only one instance of ManagerFacade.
  + **Invariant**: Only one instance of ManagerFacade exists.
  + **Precondition**: None.
  + **Postcondition**: Returns the single instance of ManagerFacade.
* **CreateProject()**: Creates a new project with specified attributes.
  + **Precondition**: projectName and description are non-empty; requiredHours > 0.
  + **Postcondition**: Adds the project to the Projects dictionary and persists it in ProjectRepo.
* **EditProject()**: Edits project attributes such as name, description, date, or required hours.
  + **Precondition**: projectId exists in the system.
  + **Postcondition**: Updates the project attribute both in memory and in ProjectRepo.
* **DeleteProject()**: Removes a project and its roles from the system.
  + **Precondition**: projectId exists in Projects.
  + **Postcondition**: Deletes the project and disassociates roles from employees.
* **AddEmployee():** Adds an employee to the system.
  + **Precondition**: employeeName and email are non-empty;  
    employee email is a valid email address, yearExperience > 0, employee's attributes are correct Enum values.
  + **Postcondition**: Adds the employee to the Employees dictionary and persists it in EmployeeRepo.
* **EditEmployee()**: Edits employee attributes such as name, description, date.
  + **Precondition**: employeeId exists in the system.
  + **Postcondition**: Updates the employee attribute both in memory and in EmployeeRepo.
* **DeleteEmployee()**: Removes an employee from the system.
  + **Precondition**: employeeId exists in Projects.
  + **Postcondition**: Deletes the employee, and unassign the employee from all roles in the system.
* **AddRoleToProject():** Adds a new role to a specified project, including details about required skills, foreign languages, job percentage, etc.
  + **Preconditions**: The projectId exists in the Projects dictionary. jobPercentage is between 0 and 100 (inclusive). yearsExperience is non-negative. foreignLanguages and skills dictionaries contain valid entries.
  + **Postconditions**: The role is added to the project's Roles dictionary. The role is persisted to the RoleRepo. Returns the newly created Role
* **RemoveRole():** Removes a role from a project, disassociating it from any employee assigned to it.
  + **Preconditions**: projectId exists in the Projects dictionary. roleId exists in the specified project's Roles.
  + **Postconditions**: The role is removed from the Roles dictionary of the project. If an employee was assigned to the role, their Roles dictionary is updated.
* **GetAllRolesByProject():** Retrieves all roles for a specified project.
  + **Preconditions:** The projectId exists in the Projects dictionary.
  + **Postconditions:** Returns a dictionary of all roles associated with the project.
* **EmployeesToAssign():** Computes and ranks employees based on their suitability for a specified role.
  + **Preconditions:** The role object is valid and contains attributes like YearsExperience, Skills, and ForeignLanguages.
  + **Postconditions:** Returns a sorted dictionary of employees with their suitability scores.Employees who do not meet the basic requirements (e.g., YearsExperience, language, or skill levels) are excluded.
* **AssignEmployeeToRole()**: Assigns an employee to a specific role.
  + **Precondition**: employeeId exists in Employees and roleId exists in any Project.
  + **Postcondition**: Updates the role and employee records.
* **OpenTicket()**: Opens an employee's absence ticket.
  + **Precondition**: employeeId exists in Employees and ticket's details are correct.
  + **Postcondition**: ticket created and added to the repositories.
* **HandleTicket():** handles ticket by reassigning the roles of the employee created the ticket.
  + **Precondition**: ticket exists in the system.
  + **Postcondition**: ticket solved and roles reassigned or not.

**Invariants**:

* projectCount and employeeCount are unique and incremented monotonically.
* Projects and Employees dictionaries maintain unique keys.

**HeadCountService:**

Provides services for managing projects, roles, and employee assignments.

Acts as an intermediary between the ManagerFacade and external clients.

**Key Methods**:

* **GetInstance()**: Singleton implementation to ensure a single service instance.
  + **Invariant**: Only one instance of HeadCountService exists.
  + **Precondition**: None.
  + **Postcondition**: Returns the single instance of HeadCountService.
* **CreateProject()**: Creates a project by delegating to ManagerFacade.
  + **Precondition**: projectName, description are non-empty; requiredHours > 0.
  + **Postcondition**: Adds the project to ManagerFacade.
* **EditProjec()**: Edits project attributes.
  + **Precondition**: Project ID exists.
  + **Postcondition**: Project is updated.
* **DeleteProject()**: Deletes project.
  + **Precondition**: Project ID exists.
  + **Postcondition**: Project is deleted.
* **CreateEmployee()**: Creates an employee by delegating to ManagerFacade.
  + **Precondition**: employeeName, email are non-empty; yearExperience > 0.
  + **Postcondition**: Adds the employee to ManagerFacade.
* **EditEmployee()**: Edits employee attributes by delegating to ManagerFacade.
  + **Precondition**: employee ID exists.
  + **Postcondition**: employee is updated.
* **DeleteEmployee()**: Deletes employee by delegating to ManagerFacade.
  + **Precondition**: employee ID exists.
  + **Postcondition**: employee is deleted.
* **GetAllRolesByProject():** Retrieves all roles for a specified project by delegating to ManagerFacade.
  + **Preconditions:** The projectId exists in the Projects dictionary.
  + **Postconditions:** Returns a dictionary of all roles associated with the project.
* **EmployeesToAssign():** Computes and ranks employees based on their suitability for a specified role by delegating to ManagerFacade.
  + **Preconditions:** The role object is valid and contains attributes like YearsExperience, Skills, and ForeignLanguages.
  + **Postconditions:** Returns a sorted dictionary of employees with their suitability scores.Employees who do not meet the basic requirements (e.g., YearsExperience, language, or skill levels) are excluded.
* **AddRoleToProject()**: Adds a role to a project by delegating to ManagerFacade.
  + **Precondition**: Project ID exists; jobPercentage is valid (0 ≤ jobPercentage ≤ 100).
  + **Postcondition**: Role is added to the project and returned in the response.
* **RemoveRole()**: Removes a role from a project by delegating to ManagerFacade.
  + **Precondition**: Role ID exists in the project.
  + **Postcondition**: Role is removed.
* **AssignEmployeeToRole()**: Assigns an employee to a role by delegating to ManagerFacade.
  + **Precondition**: Employee and role exist; role isn't already assigned.
  + **Postcondition**: Employee is assigned to the role.
* **OpenTicket()**: Opens an employee's absence ticket by delegating to ManagerFacade.
  + **Precondition**: employeeId exists in Employees and ticket's details are correct.
  + **Postcondition**: ticket created and added to the repositories.
* **HandleTicket():** handles ticket by reassigning the roles of the employee created the ticket by delegating to ManagerFacade.
  + **Precondition**: ticket exists in the system.
  + **Postcondition**: ticket solved and roles reassigned or not.

**Invariants**:

* Singleton instance consistency (\_headCountService).
* Error responses are handled gracefully.

**Project:**

Represents a project with attributes like Roles, RequiredHours, and Description.

Manages roles within the project.

**Key Methods**:

* **AddRoleToProject ()**: Adds a role to the project.
  + **Precondition**: Role attributes like jobPercentage are valid (e.g., 0 <= jobPercentage <= 100).
  + **Postcondition**: Role is added to the Roles dictionary and persisted in RoleRepo.
* **RemoveRole ()**: Deletes a role from the project.
  + **Precondition**: roleId exists in Roles.
  + **Postcondition**: Removes the role both from memory and RoleRepo.
* **EditProject ()**: Modify project attributes.
  + **Precondition**: Attributes are valid (e.g., RequiredHours > 0).
  + **Postcondition**: Updates the respective attribute.

**Invariants**:

* RoleCounter provides unique IDs for roles in the project.

**Employee:**

Represents an employee with attributes like skills, languages, and roles.

**Key Methods**:

* **AssignEmployeeToRole ()**: Associates an employee with a role.
  + **Precondition**: role is valid and not already assigned.
  + **Postcondition**: Adds the role to the Roles dictionary.

**Invariants**:

* YearsExperience >= 0
* Email format is valid (EmailAddress).

**Role:**

Represents a job role within a project.

Stores details like required skills, foreign languages, and job percentage.

**Invariants**:

* YearsExperience >= 0
* JobPercentage must be in range [0, 100].

**Skill:**

Represents a skill with an associated level and priority.

**Invariants**:

* Level >= 1 <=3
* Priority > 0

**Languages:**

Represents a language with a type and proficiency level.

**Invariants**:

* Level >= 1 <=3

**Response:**

Represents the result of a method call, especially for void functions.

Provides information about errors through ErrorMessage.

**Invariants**:

* If ErrorOccured is true, ErrorMessage must not be null.

**Response<T>:**

Extends Response for non-void methods.

Holds the returned value of a method.

**Key Methods**:

* **FromValue()**: Creates a successful response with the given value.
  + **Precondition**: value is valid.
  + **Postcondition**: ErrorOccured = false.
* **FromError()**: Creates an error response.
  + **Precondition**: msg is non-empty.
  + **Postcondition**: ErrorOccured = true.

**Invariants**:

* If ErrorOccured = true, Value is set to default.

**DAL:**

**EmployeeLanguagesRepo:**

Manages operations related to employees' language proficiencies, including CRUD operations.

Handles synchronization and ensures data integrity for employee languages in memory and in the database.

**Main Methods:**

**Add(Language language)**

* **Invariants**:
  + language.LanguageID is unique across the EmployeesLanguages dictionary.
* **Preconditions**:
  + language is not null.
  + language.LanguageID is not already present in EmployeesLanguages.
* **Postconditions**:
  + The language is added to both the EmployeesLanguages dictionary and the database.
* **Implementation Hint**:
  + Use lock to ensure thread safety during database operations.

**Delete(int id)**

* **Invariants**:
  + If id exists in EmployeesLanguages, it is removed from the dictionary and the database.
* **Preconditions**:
  + id exists in the EmployeesLanguages dictionary.
* **Postconditions**:
  + The id is removed from EmployeesLanguages and the database.

**EmployeeRepo:**

Manages employees' data, including their addition, deletion, and retrieval.

Ensures employees' data integrity in memory and in the database.

**Main Methods:**

**Add(Employee employee)**

* **Invariants**:
  + employee.EmployeeId is unique across the Employees dictionary.
* **Preconditions**:
  + employee is not null.
  + employee.EmployeeId is not already present in Employees.
* **Postconditions**:
  + The employee is added to both the Employees dictionary and the database.
* **Implementation Hint**:
  + Use lock to handle concurrent access during database operations.

**GetById(int id)**

* **Invariants**:
  + If id exists, the corresponding Employee object is retrieved from Employees or the database.
* **Preconditions**:
  + id is a valid identifier.
* **Postconditions**:
  + Returns the Employee object associated with the id if it exists.
  + Throws an exception if the id does not exist.

**EmployeeSkillsRepo:**

Manages employees' skills, providing CRUD operations.

Maintains synchronization between in-memory data and the database.

**Main Methods:**

**Add(Skill skill)**

* **Invariants**:
  + skill.SkillId is unique across EmployeesSkills.
* **Preconditions**:
  + skill is not null.
  + skill.SkillId is not already present in EmployeesSkills.
* **Postconditions**:
  + The skill is added to both EmployeesSkills and the database.

**ProjectRepo:**

Manages project data, including addition, deletion, and retrieval of projects.

Ensures synchronization between in-memory data and the database.

**Main Methods:**

**Add(Project project)**

* **Invariants**:
  + project.ProjectId is unique across \_projects.
* **Preconditions**:
  + project is not null.
  + project.ProjectId is not already present in \_projects.
* **Postconditions**:
  + The project is added to \_projects and the database.

**Delete(int projectId)**

* **Invariants**:
  + The project is removed from \_projects if it exists.
* **Preconditions**:
  + projectId exists in \_projects.
* **Postconditions**:
  + The project is removed from \_projects and the database.

**RoleRepo:**

Manages roles within projects, handling their CRUD operations.

Ensures synchronization between in-memory data and the database.

**Main Methods:**

**Add(Role role)**

* **Invariants**:
  + role.RoleId is unique across roles.
* **Preconditions**:
  + role is not null.
  + role.RoleId is not already present in roles.
* **Postconditions**:
  + The role is added to roles and the database.

**DBcontext:**

Acts as the data access layer to interact with the underlying database.

Defines entity relationships and constraints for the database schema.

Implements Singleton to ensure only one instance of the database context is active.

Manages CRUD operations for all major entities such as Employee, Project, Role, Skills, and Languages.

Provides mechanisms to configure and initialize the database.

**Key Attributes:**

**DbSet Properties**

Each DbSet represents a table in the database:

* **Employees**: Stores all employee data.
* **Projects**: Stores all project-related data.
* **Roles**: Stores information about roles in projects.
* **EmployeeSkills**: Links employees with their respective skills.
* **EmployeeLanguages**: Links employees with their language proficiencies.
* **RoleSkills**: Links roles with the skills required.
* **RoleLanguages**: Links roles with the required language proficiencies.
* **TimeZones, SkillTypes, LanguageTypes**: Reference tables for enum mappings.

**Main Methods:**

**1. GetInstance()**

* **Responsibilities**:
  + The implementation of DBcontext as a Singelton.
  + Ensures only one instance of DBcontext is active at a time.
  + Provides a thread-safe mechanism to access the database context.
* **Invariants**:
  + \_instance is either null or points to a valid instance of DBcontext.
* **Preconditions**:
  + None.
* **Postconditions**:
  + If \_instance is null, a new DBcontext is created and returned.
  + If \_instance already exists, the existing instance is returned.
* **Implementation Hint**:
  + Use lock for thread safety.

**OnConfiguring(DbContextOptionsBuilder optionsBuilder)**

* **Responsibilities**:
  + Configures the database connection string (DbPath).
  + Ensures the context uses SQL Server as the database provider.
* **Invariants**:
  + optionsBuilder is configured before any database operation is executed.
* **Implementation Hint**:
  + Ensure DbPath is always properly set before configuring.

**OnModelCreating(ModelBuilder modelBuilder)**

* **Responsibilities**:
  + Configures relationships, primary keys, and constraints for database entities.
  + Establishes foreign key relationships and cascade delete behaviors.
* **Key Configurations**:
  + **EmployeeDTO**:
    - Primary Key: EmployeeId.
    - Relationships:
      * One-to-Many: Employees can have multiple roles, skills, and languages.
  + **RoleDTO**:
    - Primary Key: RoleId.
    - Relationships:
      * One-to-Many: A role belongs to a project and can require multiple skills and languages.
      * Many-to-One: A role can optionally be assigned to an employee.
  + **ProjectDTO**:
    - Primary Key: ProjectId.
    - Relationships:
      * One-to-Many: Projects can have multiple roles.
  + Composite Keys:
    - **RoleLanguagesDTO**: {LanguageTypeId, RoleId}.
    - **RoleSkillsDTO**: {SkillTypeId, RoleId}.
    - **EmployeeLanguagesDTO**: {LanguageTypeId, EmployeeId}.
    - **EmployeeSkillsDTO**: {SkillTypeId, EmployeeId}.

**Dispose()**

* **Responsibilities**:
  + Releases resources held by DBcontext.
  + Clears all entity sets and resets the singleton instance.
* **Invariants**:
  + After Dispose(), \_instance is null, and all database-related resources are cleared.
* **Implementation Hint**:
  + Use lock to ensure thread safety when resetting resources.

**Invariants**

1. **Singleton Instance**:
   * \_instance must always be unique or null.
2. **Consistency**:
   * All DbSet properties must be properly configured before usage.
3. **Thread Safety**:
   * All public methods accessing or modifying DBcontext resources must use lock.

5.3 Packages:

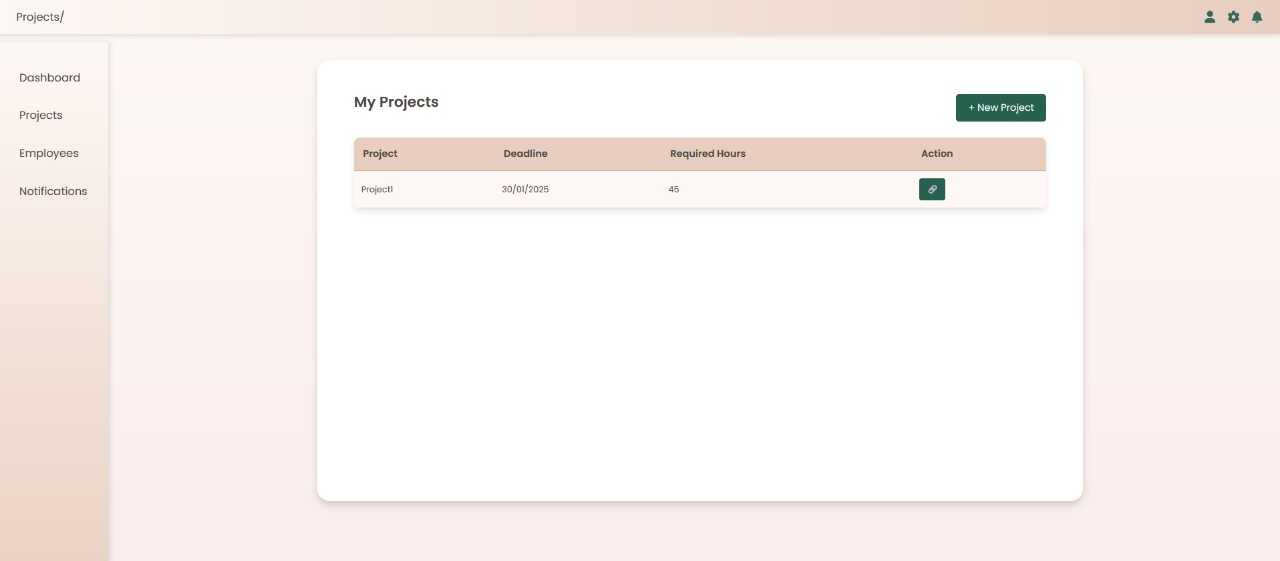
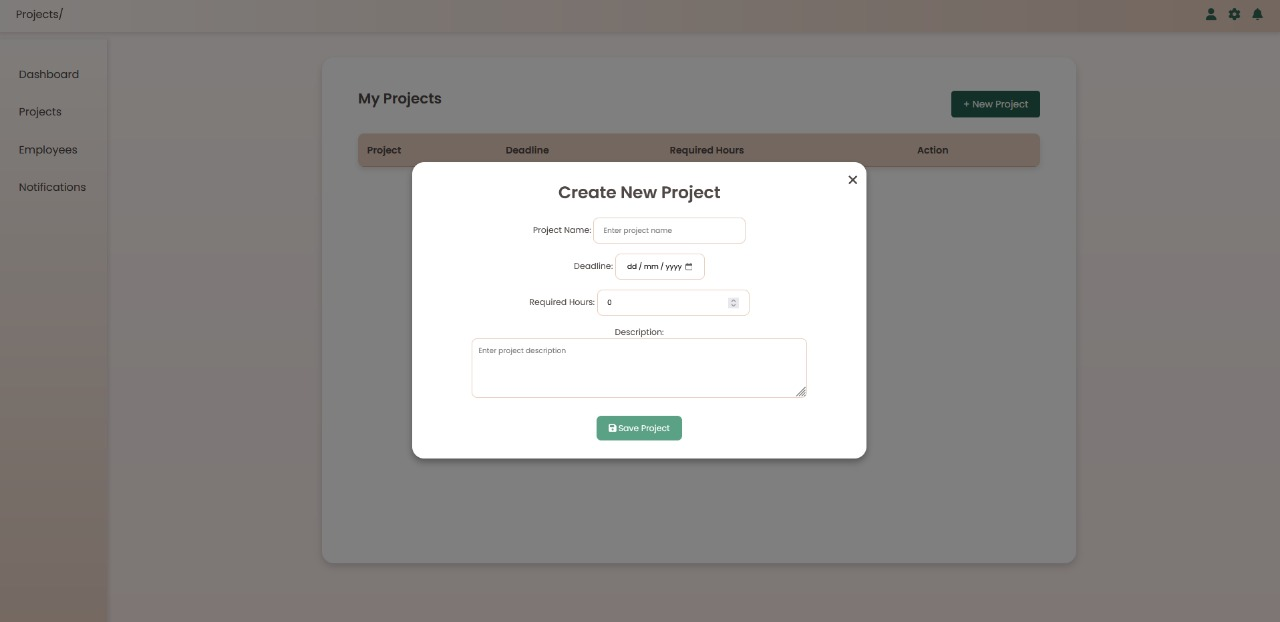


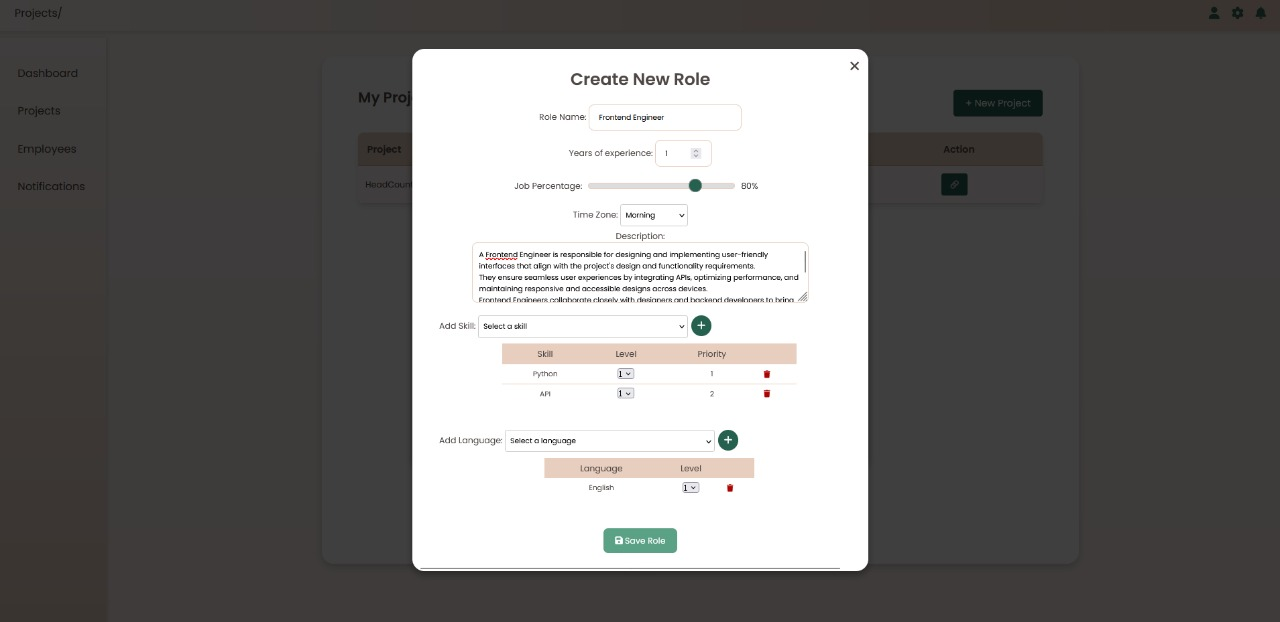
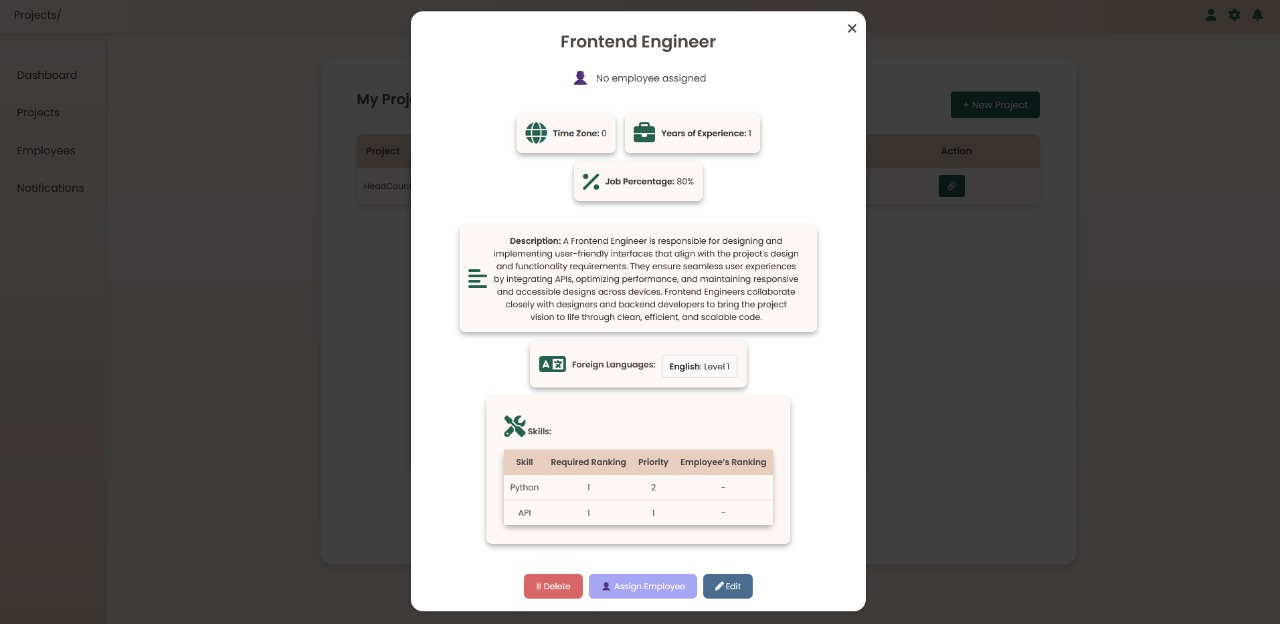
5.4 Unit Testing:

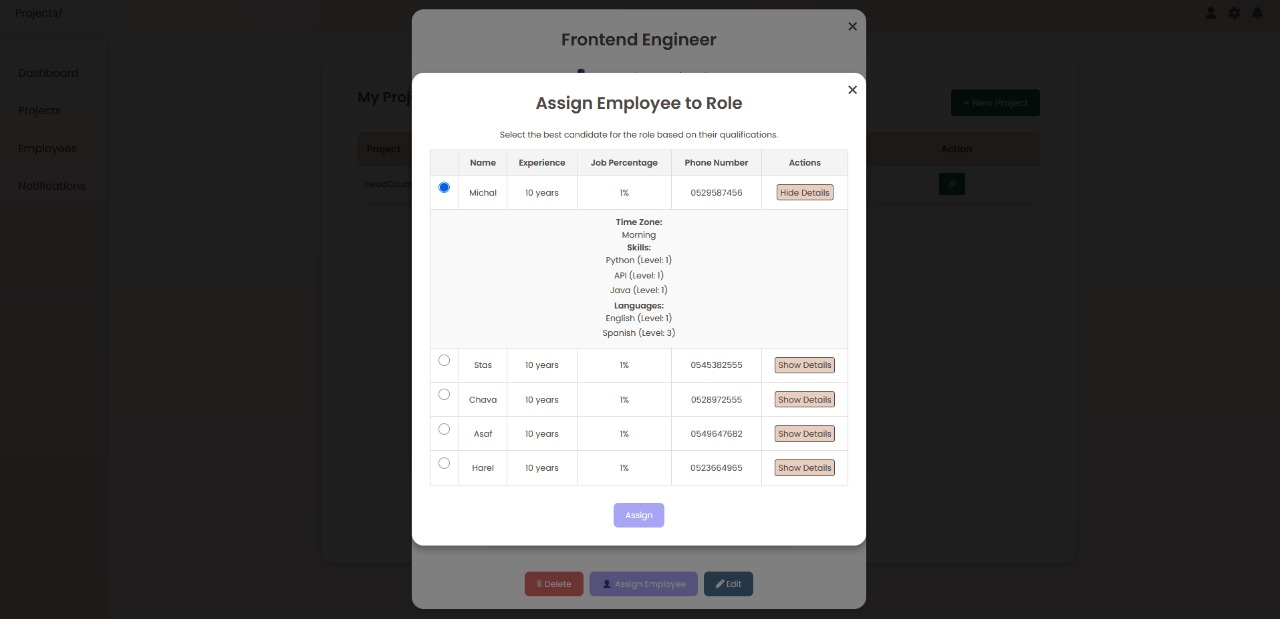
|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Function** | **Test description** | **Expected result** |
| 1 | AddEmployee(emp details) | Valid email | Add the employee |
| 2 |  | Invalid email | Gets error message |
| 3 | AddProject(project details) | Valid required hours | Add the project |
| 4 |  | Invalid required hours | Gets error message |
| 5 | AddRole(role details) | Valid yearExperience | Add the role |
| 6 |  | Invalid yearExperience | Gets error message |
| 7 | EditEmployee(emp details) | Valid email | Edit the employee |
| 8 |  | Invalid email | Gets error message |
| 9 | EditProject(project details) | Valid required hours | Edit the project |
| 10 |  | Invalid required hours | Gets error message |
| 11 | EditRole(role details) | Valid yearExperience | Edit the role |
| 12 |  | Invalid required hours | Gets error message |
| 13 | DeleteEmployee(emp details) | Valid employeeId | Delete the employee |
| 14 |  | Invalid employeeId | Gets error message |
| 15 | DeleteProject(project details) | Valid projectId | Delete the project |
| 16 |  | Invalid projectId | Gets error message |
| 17 | DeleteRole(role details) | Valid roleId | Delete the role |
| 18 |  | Invalid roleId | Gets error message |
| 19 | OpenEmployeeTicket | Valid dates | Open the ticket |
| 20 |  | Invalid dates | Gets error message |
| 21 | HandleEmployeeTicket | Valid employeeId assign | Close the ticket |
| 22 |  | Invalid employeeId assign | Gets error message |

Chapter 6

User Interface

* Projects main Page - allows users to view, manage, and create projects with details like deadlines and required hours
* Create New Project Modal - allows users to input project details such as name, deadline, required hours, and description before saving.
* A screenshot of a computer

  Description automatically generatedProject Details Modal - provides an overview of a selected project, including its deadline, required hours, description, and assigned roles, allowing for role management, employee assignment, and project edits.
* Create New Role Modal - allows users to define and customize a new role within a project, including specifying experience requirements, job percentage, time zone, skills, and languages needed, ensuring alignment with project needs.
* Role Details Modal - provides an overview of a specific role, displaying key information such as required experience, job percentage, time zone, skills, and foreign language requirements, with options to assign an employee, edit, or delete the role.
* Assign Employee to Role Modal - team leads to select the most suitable employee for a role based on their qualifications, including experience, job percentage, time zone, skills, and languages, with an option to view more details before assigning



Chapter 7

Testing

**Non functional requirements:**

|  |  |  |
| --- | --- | --- |
| **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. |

**System functionalities tests:**

|  |  |  |
| --- | --- | --- |
| **Function** | **Test Description** | **Expected Result** |
| Login- success | Login with existing username and password. | Login success and main page of manager displays. |
| Login- fail | Login with too short password | Login fails and error message of short password shows. |
| Login- fail | Login with non-existing username. | Login fails and error message of no such user shows. |
| Add new employee- success | Add a new employee with a username which does not exist in the system. | Add new employee success and user see the employee added. |
| Add new employee- fail | Add a new employee with a username which exists in the system. | Add new employee fails and error message of existing employee shows. |
| Add new employee- fail | Add a new employee with a incorrect email address. | Add new employee fails and error message of incorrect email address shows. |
| Add new project- success | Add a new project with a project name which does not exist in the system. | Add new project success and user see the project added. |
| Add new project- fail | Add a new employee with a project name which exists in the system. | Add new project fails and error message of existing project shows. |
| Add new project- fail | Add new project with incorrect date | Add new project fails and error message of incorrect date shows. |
| Add new role- success | Add a new role with a role name which does not exist in the system. | Add new role success and user see the new role added. |
| Add new role- fail | Add a new role with a role name which exists in the system. | Add new role fails and error message of existing role shows. |
| Add new role- fail | Add a new role with incorrect dates. | Add new role fails and error message of incorrect dates shows. |
| Edit employee- success | Edit employee with correct details. | Edit employee success and user see the edited details. |
| Edit employee- fail | Edit employee with incorrect email address. | Edit employee fails and error message of incorrect email address shows. |
| Edit project- success | Edit project with correct details. | Edit project success and user see the edited details. |
| Edit project- fail | Edit project with incorrect dates. | Edit project fails and error message of incorrect dates shows. |
| Edit role- success | Edit role with correct details. | Edit role success and user see the updated details. |
| Edit role- fail | Edit role with incorrect dates. | Edit role fails and error message of incorrect dates shows. |
| Delete employee- success | Delete employee which exists in the system. | Delete employee success and user see the employee deleted. |
| Delete employee- fail | Delete employee which does not exist in the system. | Delete employee fails and error message of not existing employee shows. |
| Delete project- success | Delete project which exists in the system. | Delete project success and user see the project deleted. |
| Delete project- fail | Delete project which does not exist in the system. | Delete project fails and error message of not existing project shows. |
| Delete role- success | Delete role which exists in the system. | Delete role success and user see the role deleted. |
| Delete role- fails | Delete role which does not exist in the system. | Delete role fails and error message of not existing role shows. |
| Assign employee to role- success | Assign an existing employee to role. | Assign employee to role success, user can see the role in the employee's roles list, and in the role the employee's Id. |
| Assign employee to role- fail | Assign an non-existing employee to role. | Assign employee to role fails, an error message of non-existing employee shows to the user, and the role is not in the employee's roles list, and the employee Id in role remains null. |
| Assign employee to role based on suggestion- success | assign employee to role based on suggestion- get list of relevant employees order by their score.  Test with one perfect fit employee. | Assign employee to role based on suggestion success- make sure the first employee in the list is the one with the most fitting attributes to the role. |
| Assign employee to role based on suggestion- success | assign employee to role based on suggestion- get list of relevant employees order by their score.  Test with no fitting employees with languages missing. | Assign employee to role based on suggestion success- make sure return empty list of employees. |
| Assign employee to role based on suggestion- fail | Assign employee to role based on suggestion- of no existing role. | Assign employee to role based on suggestion fails- error message of non existing role shows to the user. |
| Open gap ticket- success | Open gap ticket with correct details. | Open gap ticket success and user see the ticket opened. |
| Open gap ticket- fail | Open gap ticket with incorrect dates (end date before start date). | Open gap ticket fails and error message of incorrect dates shows to the user. |
| Handle gap ticket- success | Handle gap ticket by assigning role to exists employee. | Handle gap ticket success and user see the new assignment of the roles in the ticket. |
| Handle gap ticket- fail | Handle gap ticket by assigning role to non- existing employee. | Handle gap ticket fails and error message of non- existing employee to assign role to shows to the user. |
| View employee allocation- success | View employee allocation of an existing employee. | View employee allocation success, and user can see all the roles that this employee is assigned to. |
| View employee allocation- fail | View employee allocation of non- existing employee. | View employee allocation fails and error message of non existing employee to show allocation shows to the user. |
| View information of employees and projects- success | View information if employees and projects of existing employee. | View information of employees success and user see all statistics information of the chosen employee. |
| View information of employees and projects- fail | View information if employees and projects of non- existing employee. | View information of employee fails and error message of non- existing employee to show information of shows to the user. |

Chapter 8

Algorithm Correctness Prove

Our algorithm for assign employee to roles based on the employee's and role's attributes is based on finding the most appropriate employee for the role, which have the most accurate attributes as the role needs, while saving the "perfect" employee with best score of the attributes for another role.

To test the results of our algorithm compare to other algorithms, we created scenario:

The scenario:

**Employees:**

* + - Alice
      * 3 years of experience
      * Python- 2, SQL- 3
      * English- 3
    - Bob
      * 6 years of experience
      * Java- 3, API- 1
      * English- 3, Spanish- 2
    - Charlie
      * 3 years of experience
      * UI- 2, Java- 2
      * Hebrew- 2, English- 2
    - Dana
      * 6 years of experience
      * Python- 3, SQL- 3, API- 2, Java- 3, UI- 2
      * English- 3, Spanish- 1, Hebrew- 2
    - Eli
      * 5 years of experience
      * Java- 2, UI- 3
      * English- 3, Hebrew- 2
    - Tamar
      * 4 years of experience
      * Python- 2, SQL- 2
      * English- 3

**Roles:**

* + - Data Engineer
      * 4 years of experience
      * Python- 3 priority- 0, SQL- 3 priority- 1
      * English- 3
    - Backend Developer
      * 6 years of experience
      * Java- 3 priority- 0, API- 2 priority -1
      * Spanish- 2
    - UI/UX Expert
      * 3 years of experience
      * UI- 2 priority- 0, Java- 2 priority -1
      * Hebrew- 2
    - API Integrator
      * 5 years of experience
      * API- 2 priority- 0, SQL- 3 priority-2
      * English- 3
    - Junior Developer
      * 2 years of experience
      * Java- 2 priority- 0
      * English- 2

We asked them to allocate employee each time to another role:

**Greedy algorithm**

which always pick the most scored employee gave the allocation:

**Role Data Engineer** -> **Dana**

**Role Backend Developer** -> **Bob**

**Role UI/UX Expert** -> **Dana**

**Role API Integrator** -> **Dana**

**Role Junior Developer** -> **Bob**

**Our algorithm**

gave the allocation:

**Role Data Engineer** -> **Dana**

**Role Backend Developer** -> **Bob**

**Role UI/UX Expert** -> **Charlie**

**Role API Integrator** -> **Dana**

**Role Junior Developer** -> **Charlie**

When we focus on the role UI/UX Expert which requires:

* 3 years of experience
* UI- 2 priority- 0, Java- 2 priority -1
* Hebrew- 2

We asked Chat GPT to allocate an employee for it and this is his answer:

We’re looking for:

* Experience ≥ 3 years
* Has UI and Java skills at required levels
* Has Hebrew at Level 2+

**🔍 Let’s go through the candidates:**

**1. Alice**

* Experience: 3 ✅
* Skills:
  + No UI ❌
  + No Java ❌
* Languages: English only ❌ **→ Rejected**

**2. Bob**

* Experience: 6 ✅
* Skills:
  + Java Level 3 ✅
  + No UI ❌
* Languages: English + Spanish ✅ **→ Missing UI & Hebrew ❌**

**3. Charlie**

* Experience: 3 ✅
* Skills:
  + UI Level 2 ✅
  + Java Level 2 ✅
* Languages: Hebrew Level 2 ✅ **→ ✅ All match!**

**4. Dana**

* Experience: 6 ✅
* Skills:
  + UI Level 2 ✅
  + Java Level 3 ✅
* Languages: Hebrew Level 2 ✅ **→ ✅ All match**

**5. Eli**

* Experience: 5 ✅
* Skills:
  + UI Level 3 ✅
  + Java Level 2 ✅
* Languages: Hebrew Level 2 ✅ **→ ✅ All match**

**6. Tamar**

* Experience: 4 ✅
* Skills:
  + No Java ❌
  + No UI ❌
* Languages: English ✅ **→ ❌ Not qualified**

**✅ Conclusion**

All of the following match the role:

* Charlie
* Dana
* Eli

Let’s choose **Eli** as the best candidate:

**✅ Final Pick: Eli**

**Why?**

* Has **UI Level 3** and **Java Level 2** → both at or above requirement
* Has **Hebrew Level 2**
* Experience: 5 ≥ 3
* Compared to Dana and Charlie:
  + Eli has **the highest UI skill level (3)**, which is key to a UI/UX role.
  + Equal or better in other areas.

Lets compare the three allocations:

| **Greedy** | **Our algorithm** | **Chat GPT** |
| --- | --- | --- |
| Dana | Charlie | Eli |

We can see that Dana and Eli has attributes with higher level than the level required by the role- Dana has Java level 3 while 2 is required and Eli has UI level 3 while 2 is required.

Charlie has the exactly level of all attributes as the role required and that is why he is the best employee to allocate.

**Our algorithm choose the most suitable employee for each role.**