# Low Level Design for Employee Management Director

# General Information

## Purpose and Scope

The document includes a solution on implementation of Employee Management Director Microservice to address role-based actions.

The solution is described in such detail that is possible to understand the business value, functionality and make high level effort estimation.

Any part of the proposed design may subject to change implementation.

## Input

This document is prepared based on the HLD referenced and attached below.



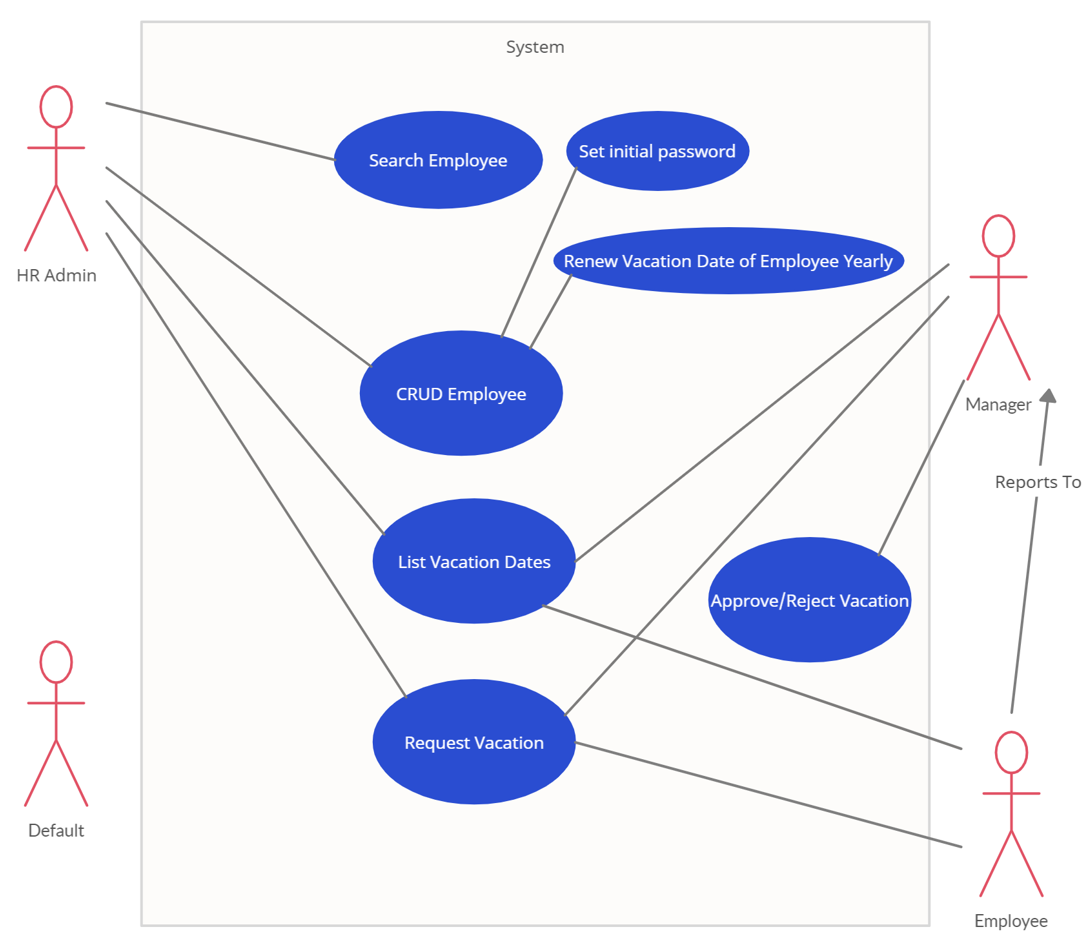
## Terminology

|  |  |  |
| --- | --- | --- |
| Term/Concept | Abbreviation | Definition |
|  |  |  |

## Revision Information

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Signature** | **Comments** |
| V1 | 11.12.2022 | IAKTAS | Initial Document has been created |
|  |  |  |  |
|  |  |  |  |

## Use Case Diagram



## Patent Aspects

The solution does not include any patentable part.

## Front-End Assumption

When user login to the system, FE will trigger EmployeeManagementDirector service to list actions/journeys that group has.

if user is in **HR\_ADMIN** group, ‘Employee Search’ page will be opened. **HR\_Admin** can search employee then Employee page will be opened including employee data and vacation days list. HR Admin has right to manage employee info. There will be a menu in left bar and For HR Admin there will be these menu item will be shown ‘New Employee’, ‘Request Vacation’ (Request Vacation is for own user). In ‘Request Vacation ’ page user can see the list of his/her previous vacation days with request new vacation button.

If user is in **Manager** group. ‘Request Vacation’ will be shown in left bar menu. When a manager login to system, the waiting approval list will be listed. Manager will approve/reject each one.

# Technical Solution

The base API path should be **/employeeManagementDirector/v1**

This solution proposes the design of Employee Management Director Microservice which exposes API :

POST /vacation/new/{ employee\_username }

POST /vacation/approve/{ system\_username }

POST /vacation/reject/{ system\_username }

GET /vacation/{employee\_username}

GET / vacation/ approval/{ employee\_username }

GET /employee/{ system\_username }/{ employee\_username }

POST /employee/{ system\_username }

PATCH /employee/{ system\_username }/{ employee\_username }

DELETE /employee/{ system\_username }/{ employee\_username }

**New Vacation:**

Execution starts with a new vacation request.

POST /vacation/new/{username} API will be exposed.

Vacation will be created with the status of new for the employee who request with the username.

The username will be extracted out from incoming newVacation request parameter in the following attribute: username.

GetEmployee Api of Employee Manager Service will be called to get employee Details of username.

Validation will be applied for the employee before creating vacation record. VacationDayLeft will be compared with the requested days. If there is no VacationDayLeft remained for the new request, EmployeeManagementDirectorException will be thrown with reason “There is no vacation Day Left for Employee <ID> to create a new request”. In Validation there will be more item which checks vacation renewal day of Employee. If vacation renewal day is bigger than 1 year; first vacation day will be renewed.

If validation is passed, employeeIDs of reportsTo field will be taken from previously getEmployee response and getEmployee API of Employee Manager Service will be called again to get employee Details for each EmployeeID in reportsTo.

Set client\_id of Keycloak user as “requestVacation”. While getting token for each “operation action” and “system user group” will be compared. If action is not valid for the requester, request can not be completed.

Vacation status will be set as “NEW”

Vacation Request Date will be set as todays date.

Vacation will be added in Approval List of reportsTo employee and patchEmployee request will be called for the employee.

**Approve Vacation**

Execution starts with a approve vacation request.

POST /vacation/approve

Vacation will be approved with the status of approved for the employee who request with the EmployeeID.

Approver and Employee Records will be updated.

busedDays will be reduced from Remaining Vacation Days of Employee.

Set client\_id of Keycloak user as “manageVacation”.

ApprovedDate will be set as today’s date.

Approve/Reject reason is optional.

**Reject Vacation**

Execution starts with a reject vacation request.

POST /rejectVacation

Approver and Employee Records will be updated.

Vacation will be approved with the status of rejected for the employee who request with the EmployeeID.

client\_id of Keycloak user will be set as “manageVacation”.

ApprovedDate will be set as today’s date.

Approve/Reject reason is mandatory.

**List Vacation by Employee ID**

Execution starts with a below request:

GET /vacation/{employee\_username}

Vacations of employee will be listed for each employee. Additionally, ‘Status’ should be added as the search criteria as well.

Vacations will be listed as a result of server response.

client\_id of Keycloak user will be set as “getEmployee”.

**List Approvals by Employee ID**

Execution starts with below request:

GET / vacation/approval/{ employee\_username }

Below API will be called to get employee details with given employee\_username in the request.

GET /employeeManager/v1/employee/{ employee\_username }

ApprovalList will be extracted from the employee record and returned as a server response.

This Api will be executed to list approvals which are waiting on reportTo employee.

Front-End will trigger this API to list approvals of Employee. Dynamic search criteria will be applied.

**CRUD operations for Employee**

Execution starts with below requests:

GET /employee/{ system\_username }/{ employee\_username }

POST /employee/{ system\_username }

PATCH /employee/{ system\_username }/{ employee\_username }

DELETE /employee/{ system\_username }/{ employee\_username }

Each API triggers Employee Manager to manage Employee and keep in repo.

client\_id of Keycloak user will be set as “manageEmployee” for POST, PATCH and DELETE .

client\_id of Keycloak user will be set as “getEmployee” for GET .

Keycloak user will be called with using system\_username.

employee\_username will be the user which will be edited/created by Employee Manager

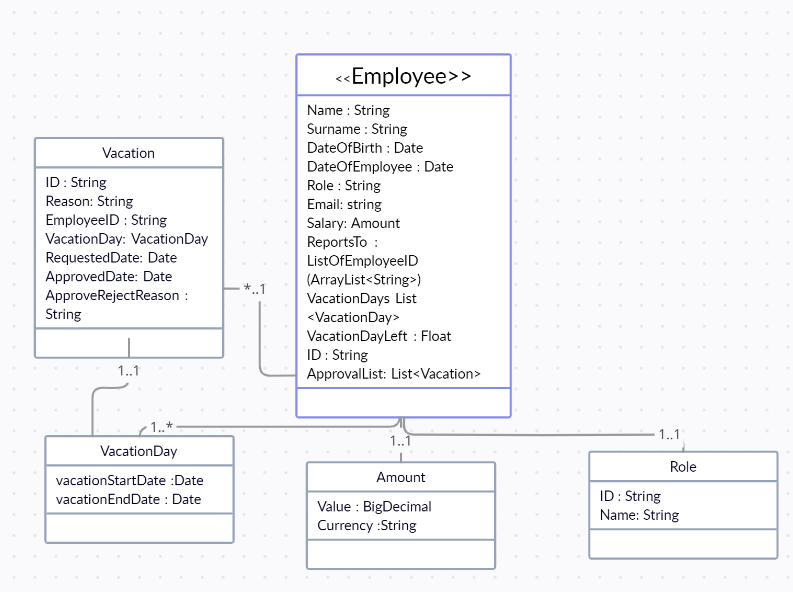
These APIs will be called to make CRUD operations :

POST /employee-manager-api/v1/employee

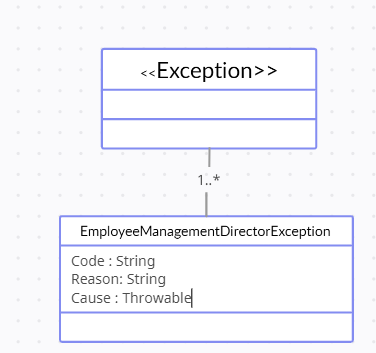
PATCH /employee-manager-api/v1/employee/{{user\_name}}

DELETE /employee-manager-api/v1/employee/{{user\_name}}

Domain Model contains Employee and Exception Model. A view of the domain model can bee seen in the following logical View – Domain Model Diagram.



**Figure 1 Employee Management Director Logical View, Domain Model**



**Figure 2 Employee Manager Logical View, Domain Model – Exception**

# Employee Json File should be attached

Sample Json representation of employee creation request is as below:

{

    "username": "iaktas",

    "name": "irem",

    "surname": "aktas",

    "role": {

        "id": "MANAGER",

        "value": "MANAGER"

    },

    "reportsTo": [

        "iaktas"

    ],

    "salary": {

        "value": 3,

        "unit": "TL"

    },

    "dateOfBirth": "1988-10-11T03:06:00.000Z",

    "dateOfEmployment": "2023-10-11T03:06:00.000Z",

    "email": "iaktas@xx.com",

    "groups" : [

        "MANAGER"

    ]

}

Sample Json representation of employee creation response is as below:

{

    "username": "iaktas",

    "name": "irem",

    "surname": "aktas",

    "role": {

        "id": "MANAGER",

        "value": "MANAGER"

    },

    "reportsTo": [

        "iaktas"

    ],

    "vacationDayLeft": 18.0,

    "salary": {

        "unit": "TL",

        "value": 3.0

    },

    "dateOfBirth": "1988-10-11T03:06:00.000Z",

    "dateOfEmployment": "2023-10-11T03:06:00.000Z",

    "vacationRenewalDate": "2022-12-13T19:36:22.401Z",

    "vacations": null,

    "email": "iaktas@xx.com",

    "groups": [

        "MANAGER"

    ],

    "approvalList": null

}

Config Manager Configuration which is stored in cache:

{  
 "configManagerMaps" : [  
 {  
 "groupName": "HR\_ADMIN",  
 "actionList": [  
 "manageEmployee",  
 "checkVacationList",  
 "requestVacation",  
 "getEmployee"  
 ]  
 },  
 {  
 "groupName": "MANAGER",  
 "actionList": [  
 "manageVacation",  
 "requestVacation",  
 "checkVacationList",  
 "getEmployee"  
 ]  
 },  
 {  
 "groupName": "EMPLOYEE",  
 "actionList": [  
 "checkVacationList",  
 "requestVacation",  
 "getEmployee"  
 ]  
 }  
 ]  
  
 }

# Other Functionalities to Be Added

If team capacity is enough for this sprint we need to add below functionalities in the microservice.

Micrometer and Prometheus should be implemented for performance logging.

Resilience to make microservice fault tolerant and Spring retry can be added to recall operation when request gets unauthorized or unreachable operations.

Sleuth should be used for microservice logging.

Test Coverage should not be under %80.

Docker File should be prepared.

# Employee Manager Reference

**Keycloak stub will be implemented.**

Not to wait for Front End Keycloak configuration, stub should be created.

Stub will get keycloak user which is created in create an employee scenario.

Stub will request for token to validate system user to perform specific action.

“client\_id” will store action which system user will perform.

“group” is user group which a system user is assigned to.

Stub will validate weather group – client-id relation is correct or not by comparing data in configMap. If not validate, stub will return 403-Forbidden Response.

GET employee-manager-api/v1/realm\_hr\_stub/users/{{user\_name}}

GET employee-manager-api/v1/realm\_hr\_stub/protocol/openid-connect/token

Sample Keycloak User Json:

{

    "username": "isabah",

    "firstName": "isa",

    "lastName": "sabah",

    "groups": [

        "EMPLOYEE"

    ],

    "credentials": [

        {

            "id": null,

            "createdDate": "2022-12-13T18:17:44.486Z",

            "type": "password",

            "value": "pwd\_hr\_123",

            "temporary": true

        }

    ],

    "email": "isabah@xx.com",

    "enabled": true,

    "client\_id": null,

    "lastAction": null,

    "lastActionDate": null

}

**Sample Token Response:**

{

    "access\_token": "eyJhbGciOiJSUzI1NiIsINVSHGhepnDu13SwRBL-v-y-04\_6e6IJbMzreZwPI-epwdVPQe-ENhpvms2WdGM\_DmgMLZ8YQFS4LDl9R7ZHT8AgXe-WCFV6OFkA7zvdeFwQ4kVVZE0HlNgHgoi4DrgMfwwz\_ku1yJNJP3ztTY1nEqmA",

    "refresh\_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRRnB5YlloMGVEektIdlhOb3JvaFUxdlRvWVdjdP3vbfvk7O0zvppK9N4-oaUqZSr0smHv5LkuLDQYdPuxA",

    "token\_type": "bearer",

    "session\_state": "bb1c586a-e880-4b96-ac16-30e42c0f46dc",

    "expires\_in": 300,

    "refresh\_expires\_in": 1800,

    "not\_before\_policy": null

}

**ConfigMap Cache will store user goup and its actions list.**

Below config map will be cached in microservice:

{  
 "configManagerMaps" : [  
 {  
 "groupName": "HR\_ADMIN",  
 "actionList": [  
 "manageEmployee",  
 "checkVacationList",  
 "requestVacation",  
 "getEmployee"  
 ]  
 },  
 {  
 "groupName": "MANAGER",  
 "actionList": [  
 "manageVacation",  
 "requestVacation",  
 "checkVacationList",  
 "getEmployee"  
 ]  
 },  
 {  
 "groupName": "EMPLOYEE",  
 "actionList": [  
 "checkVacationList",  
 "requestVacation",  
 "getEmployee"  
 ]  
 }  
 ]  
  
 }

# Assumptions

Keycloak service will be added to the environment setup. In this sprint It will be only used for registering user and password and assigning to group as stub. Role definitions, assigning functionalities to role and Assigning roles to groups will be implemented further sprint planning which will be added in project backlog.

Roles and its functionalities will be kept in Config Manager Service for this sprint.

Config Manager will be stored in Cache of Employee Manager.