Requirements Specification

for

Employee Management System

-Raghabendra Sharma

**Table of Contents**

1 Introduction

1.1 Purpose

1.2 Document Conventions

1.3 Project Scope

1.4 References

2 System Description

3 Functional Requirements

3.1 System Features

3.1.1 System Feature 1

3.1.2 System Feature 2

3.2 UML Diagrams

3.2.1 Class Diagram

3.2.2 Use Case Diagram

3.3 Entity Relationship Diagrams

3.4 Data Flow Diagram

3.4.1 Context/0 Level DFD

3.4.2 1st Level DFD   
 3.4.3 2nd Level DFD  
3.5 Data Dictionary

4 External Interface Requirements

5 Technical Requirements (Non functional)

5.1 Performance

5.2 Scalability

5.3 Security

5.4 Maintainability

5.5 Usability

5.6 Multi lingual Support

5.7 Auditing and Logging

5.8 Availability

6 Open Issues

# Introduction

## Purpose

To Build an Employee Management System. An application where users like employee and admin can log in. The admins can manage professional data of employees and can permit or decline leaves whereas Employees can access their profile and make personal changes as well as apply for leaves.

## Document Conventions

The naming Conventions for all the data types and their attributes are followed in such a way that it is helpful for reviewers to understand the business logic and relate with the data scene on the user’s side.

1. All data models are named as per project required conventions and are singular eg. Employee, Leave.
2. All Django forms are name in accordance with respective data models. Eg. EmployeeForm for Employee.
3. Function based views are used to serve the endpoints.
4. MVT architecture is followed strictly.

## Project Scope

* **Motivation** - In this world of growing technologies everything has been computerized. With large number of opportunities, the human workforce has increased. Thus, there is a need of a system which can handle the data of such a large number of employees in an organization. This project simplifies the task of maintain records because of its user-friendly nature.
* **Solution Implemented-** The "EMPLOYEE MANAGEMENT SYSTEM" has been developed to override the problems prevailing in the practical manual system. This software is supported to eliminate and, in some cases, reduce the hardships faced by the existing system. Moreover, this system is designed for the particular need of the company to carry out operations in a smooth and effective manner.
* **Usage-** The system is reduced as much as possible to avoid errors while entering the data. It also provides error message while entering the invalid data. No formal knowledge is needed for the user to use this system. Thus, by this all it is user friendly.

## References

Links to all external documents and web resources referred in the requirements:

1. Django Documentation - <https://docs.djangoproject.com/en/3.2/>
2. Bootstrap - <https://getbootstrap.com/>
3. Bootstrap icons - <https://icons.getbootstrap.com/>

# System Description

This system should consist of an application program, on one hand, and a database model on the other. The program should perform the basic operations upon the database as retrieving, inserting, updating data apart from user creation and authentication using login.

Any additional functionality is a goal of a further module development. It is a kind of strategy to start the development from designing and constructing the database, as this structure will further map different data tables together. The data model tables and the relationships should respond to the given task and cover the basic requirements.

The Interface of the program should be user-friendly, and the program should be easy for use. Both controls and forms should logically and functionally be related within the program and fully respond to the structure of the database.

# Functional Requirements

## System Features

This system is expected to be user friendly and will offer easy access to data as well as services

such as online leave management and employee profile management.

### Registration:

* The users can register on the system with valid details. The user registration is split into two

parts, one as admin registration and other one as employee registration.

* Registration using invalid details or non-distinct field data like username, email will lead to validation errors.

### Authentication:

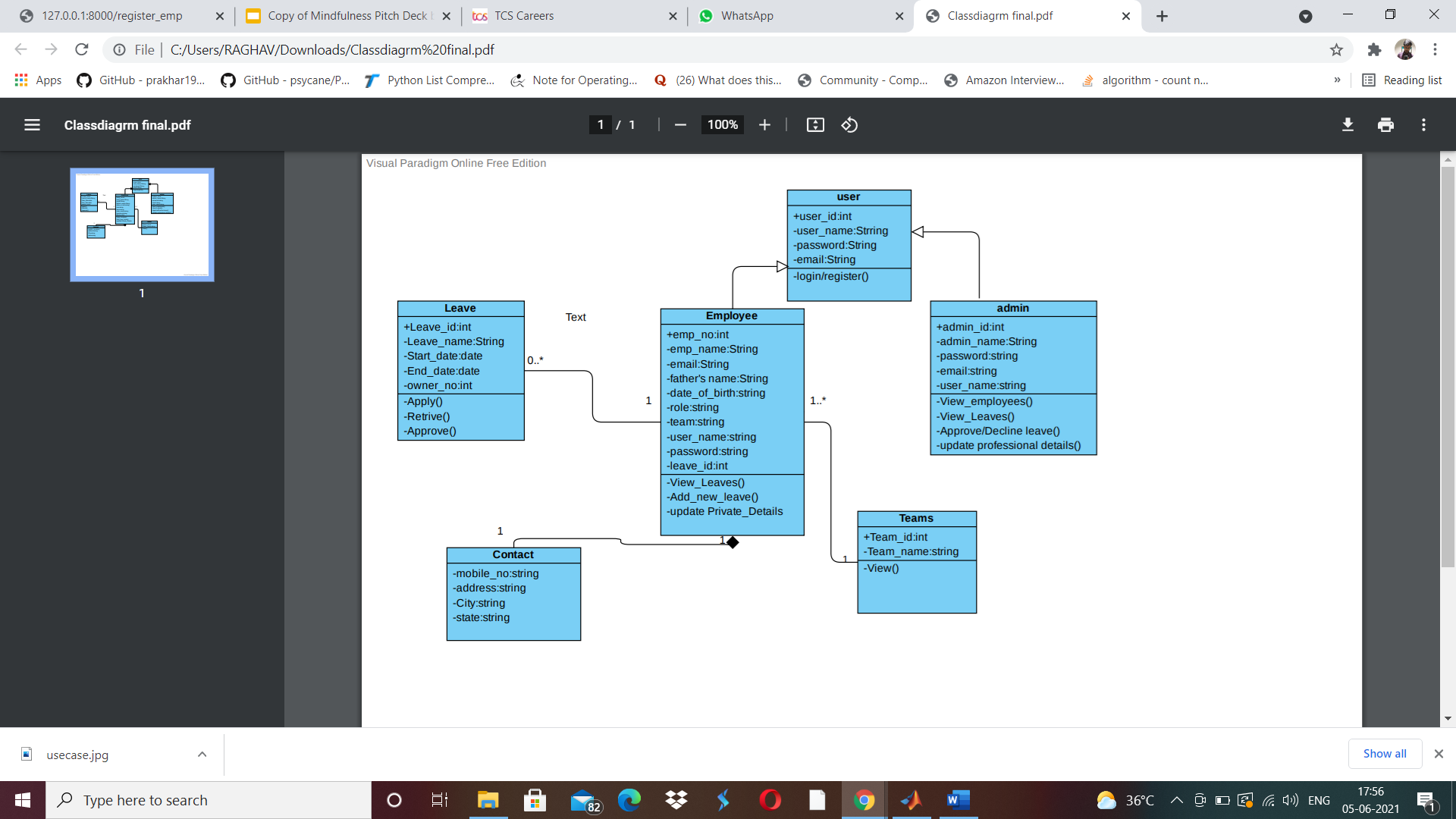
* Login- The user can login to the EMS system with his/her username and password.
* Logout- The user can log out from the EMS system.
* Login failure- If the user does not exist in the database or the user has provided invalid credentials, then validation message is displayed.
  + 1. **Authorization:**
* User role check- After logging in, the user role will be checked from the database and the dashboard user interface will be displayed according to their role (Admin, Employee).

### Process Data:

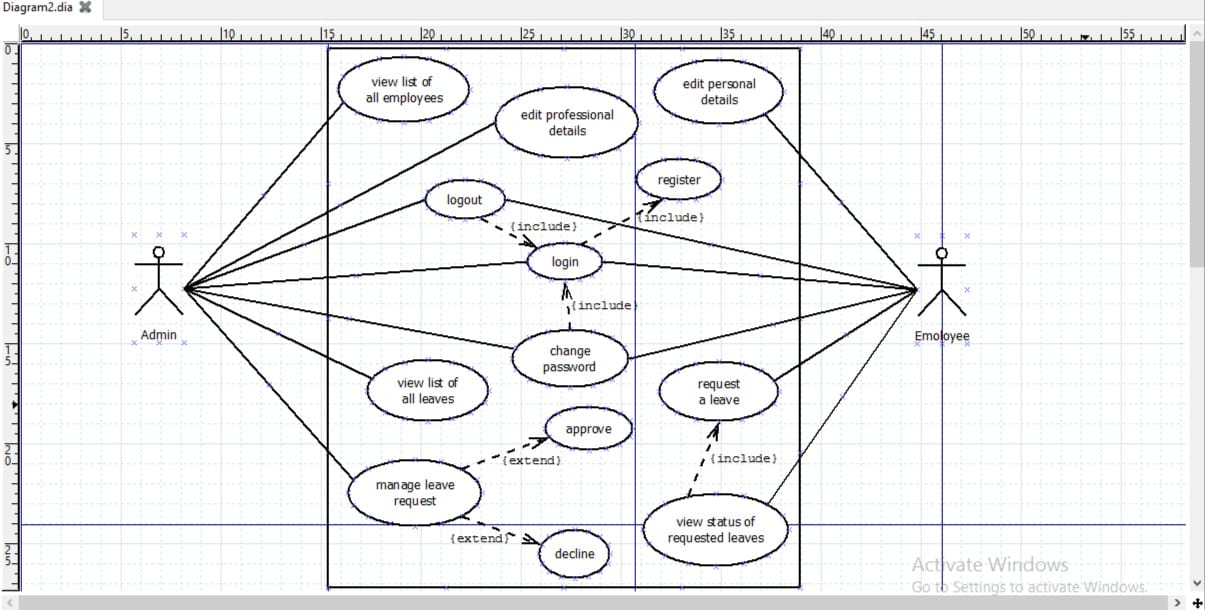
* Display- User with defined roles can have their content displayed on the interface. Being more specific, employee can only view his/her personal information and applied leaves. Admin can view all employee profiles and all leaves applied by employees.
* Edit/Update- A user with employee role can edit his/her specific personal information and apply for new leaves. A user with admin role can edit professional information of any employee and update status of applied leaves by employees to “approved” or “denied”.
  + 1. **Leave Management:**
* Leave application- The employee can be able to fill in leave application form with the appropriate fields.
* Leave approval- The admin can be able to approve leave applications.

## UML Diagrams

### Class Diagrams



### Use Case Diagrams



**Login:**

|  |  |
| --- | --- |
| **ID** | *Login* |
| **Description** | *This use case allows user to login into the system to access the relevant functions according to the user's role.* |
| **Actors** | *Users (admin and employees).* |
| **Preconditions** | Users has to have a valid account. |
| **Basic Steps** | *1.The user enters the staff ID and password.*  *2. The user submits the staff ID and password.*  *3. The system validates the staff ID and password.*  *4. The system verifies the staff ID and password.*  *5. The system displays the user's home page.*  *6. The use case ends.* |
| **Postconditions** | The system displays the relevant home page. |

|  |  |
| --- | --- |
| Id | Edit personal details. |
| Description | This use case allows employees to edit or update their personal details. |
| Actors | Employees |
| Preconditions | The employee must have a valid account in the system. |
| Basic steps | 1. Login. 2. View profile.  3. *Edit personal details.*  *4. Submit the new updated details.*  *5. The system displays the updated profile.*  *6. The use case ends.* |
| Postconditions | The system displays the updated profile of the employee. |

|  |  |
| --- | --- |
| Id | View list of all employees. |
| Description | This use case allows admin to view the list of all employees in an organization. |
| Actors | Admin |
| Preconditions | The admin must have to login with a valid account. |
| Basic steps | 1. Login 2. *The admin have to click on view list to check the list of all employees.*  *3. The system shows the list.*  *4. The use case ends.* |
| Postconditions | The system displays the list of employees. |

|  |  |
| --- | --- |
| Id | Edi.professional details |
| Description | This use case allows admin to edit the professional details of an employee like to assign a new project to the employee. |
| Actors | Admin |
| Preconditions | 1. *The admin must have to login.*  *2. List must contain the profit of the required employee.* |
| Basic steps | 1. Login.  2. View list of employees.  3. Choose an employee for which the updation is required.  4. Edit the details.  5. Submit the new details.  6. The system displays the new profile of that employee.  7. The use case ends. |
| Postconditions | The system displays the relevant updated details. |

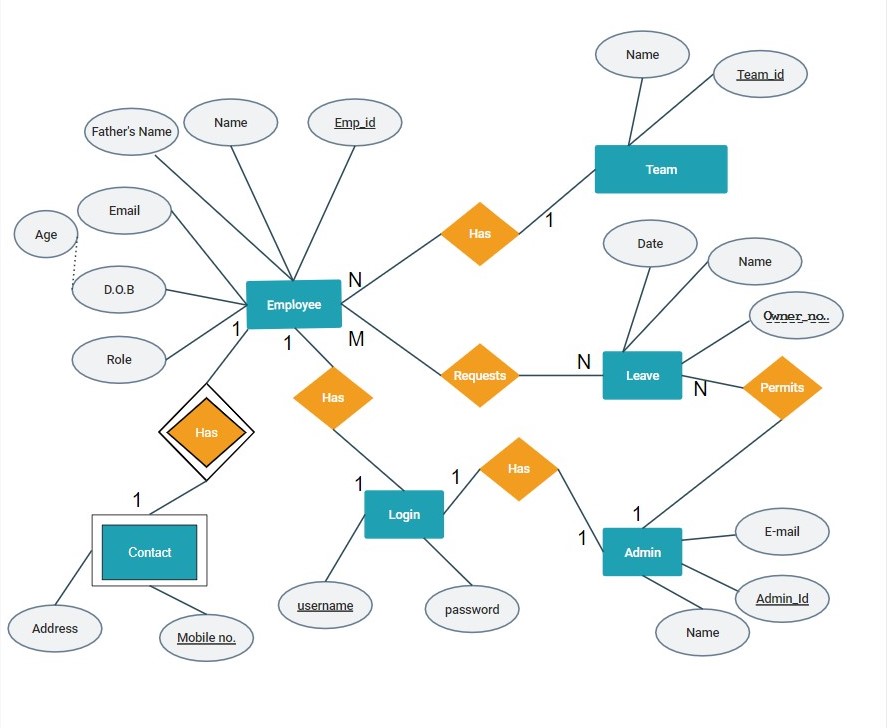
|  |  |
| --- | --- |
| ***Id*** | *Apply for a leave.* |
| ***Description*** | This use case allows employees to apply for a leave. |
| ***Actors*** | Employees |
| ***Preconditions*** | Employees must have balanced leaves. |
| ***Basic steps*** | 1. Login.  2. Open the profile.  3. Check the balanced leaves and then apply for a leave.  4. The system validate the request.  5. The use case ends. |
| ***Postconditions*** | The system displays the request. |

|  |  |
| --- | --- |
| Id | View list of all leaves. |
| Description | This use case allows the admin to view the list of leaves applied by the employees. |
| Actors | Admin |
| Preconditions | Employees must have applied for leaves. |
| Basic steps | 1. Login. 2. Click on list of leaves menu.  3. The system displays the relevant list of leaves applied by the employees.  4. The use case ends. |
| Postconditions | The system displays the required homepage. |

|  |  |
| --- | --- |
| Id | Manage leave request. |
| Description | This use case allows admin to manage the leave request for the employees. |
| Actors | Admin. |
| Preconditions | View list of leaves. |
| Basic steps | 1. Login. 2. View list of leaves.  3. Manage (i.e. grant or deny) the leaves.  4. The system validate the function.  5. The use case ends. |
| Postconditions | The system update the status of request leaves. |

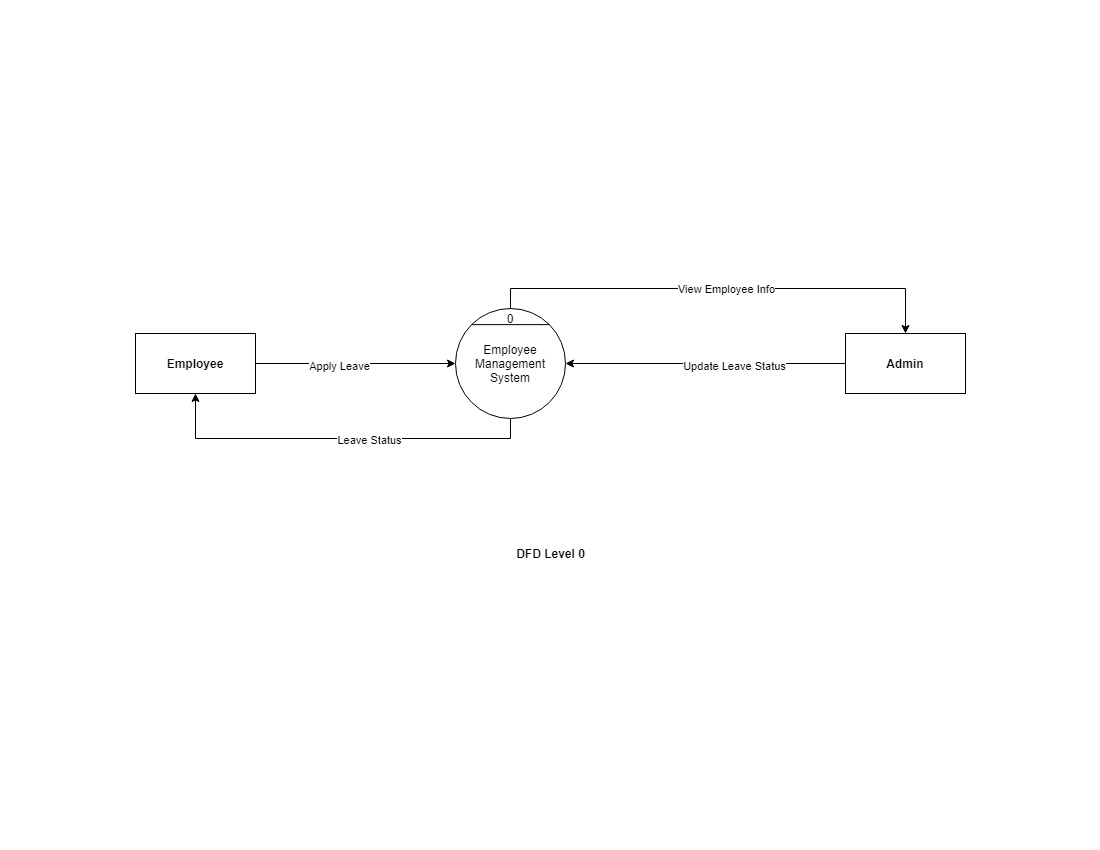
|  |  |
| --- | --- |
| Id | Logout. |
| Description | This use case allows user to logout the profile from the system. |
| Actors | Users(admin and employees). |
| Preconditions | Users must login to their account. |
| Basics steps |  |
| Postconditions | Logout from the profile. |

## Entity Relationship Diagrams

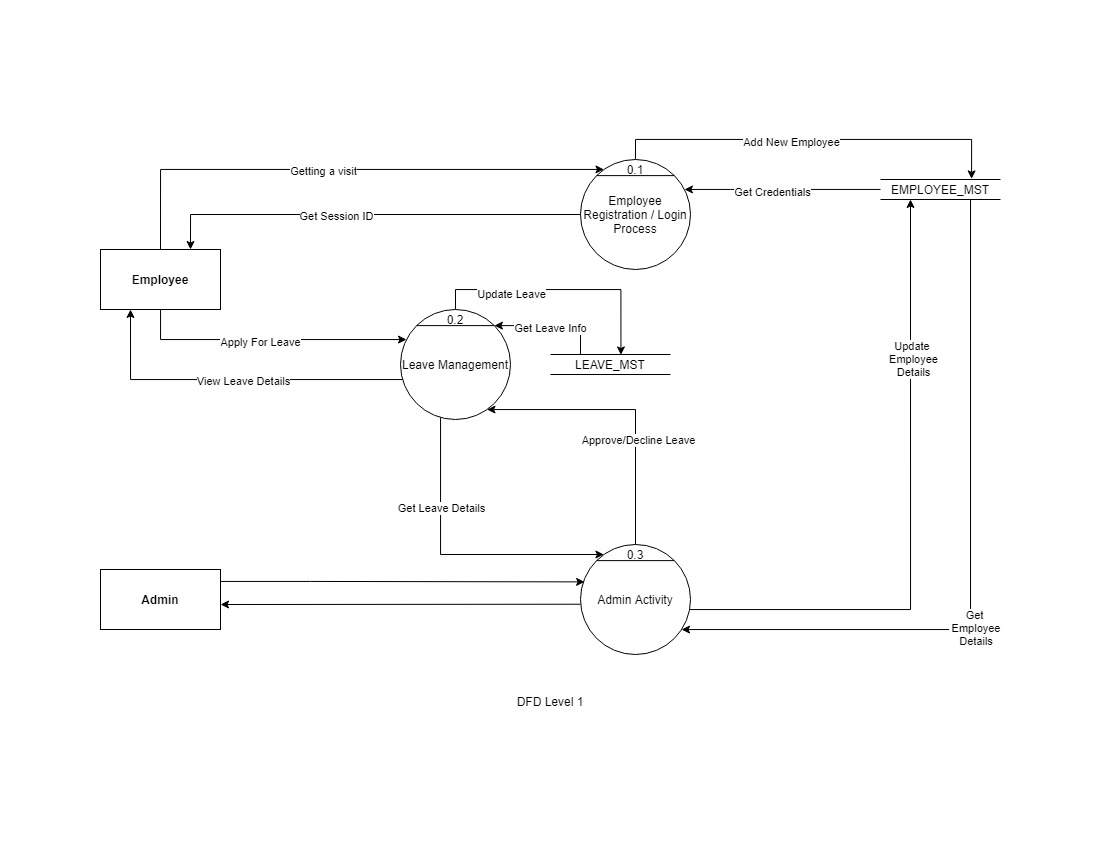
****

## 3.4 Data Flow Diagram

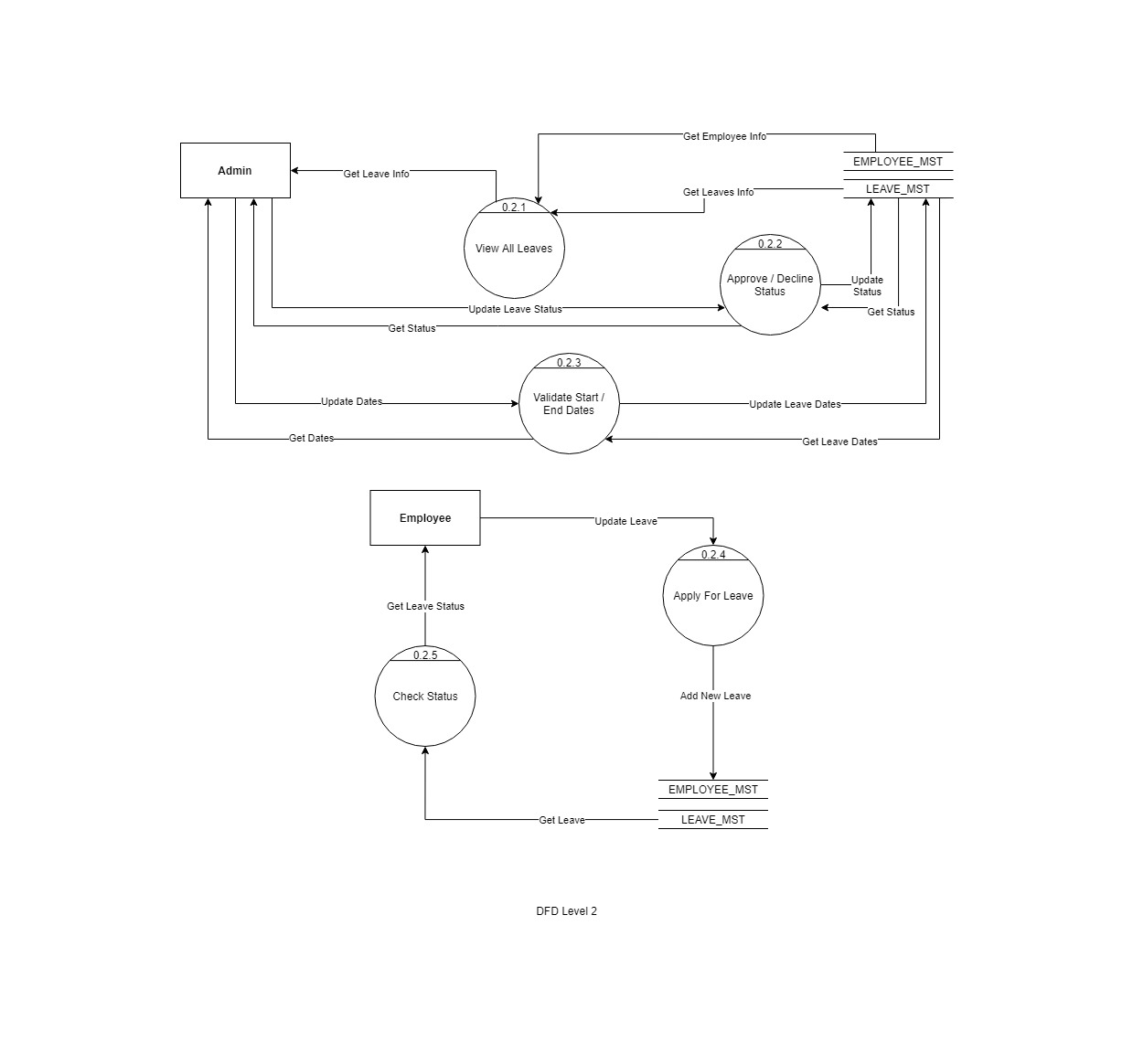
**3.4.1 Context Level/ 0 Level DFD**

****

**3.4.2 First Level DFD**

****

**3.4.3 Second Level DFD**

****

## 3.5Data Dictionary

### 3.5.1 Employee

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field\_name | Data\_Type | Constraints | max\_length | Description |
| Name | CharField | not null | 200 | Name of the employee |
| Fathers’s Name | CharField | not null | 200 | Fathers name of the employee |
| Gender | CharField | not null | 50 | gender of employee |
| User\_Name | CharField | Unique Key | 200 | user\_name to which employee logged in |
| Data of Birth | DateField | not null | ---- | DOB of employee |
| Role | CharField | not null | 100 | Role of his job |
| Department\_id | CharField | not null | 100 | Department in which he works |
| Email | EmailField | not null | --- | Employee’s Email |
| Emp\_Id | IntField | Primary Key | 50 | Employee’s unique Id |

### 3.5.2 Admin

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field\_Name | Field\_Type | Constraints | Max\_Length | Description |
| Admin\_id | CharField | Primary Key | 50 | Admin’s Unique Id |
| Email | EmailField | not null | --- | Email |
| User\_Name | IntField | Unique | 200 | User\_name to which admin logged in |

**3.5.3 Leave**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field\_Name | Field\_Type | Constraints | Max\_Length | Description |
| Leave\_id | IntField | Primary Key | 100 | Id of the associated Leave |
| Owner | IntField | Foreign Key References to Emp\_id (Employee) | 50 | Owner id the leave is associated with. |
| Status | CharField | not null | 20 | Status of the Applied Leave |
| Start\_Date | DateField | not null | --- | Start Date of the leave |
| End\_Date | DateField | not null | --- | End Date of the leave |
| Name | CharField | not null | 20 | Name of the leave |

# External Interface Requirements

No external interfaces or services were used or referred to during the development of this project.

# Technical Requirements (Non functional)

## Security

Django applications provide excellent security protections out of the box. The following protections are enabled in Django by default:

* Cross site scripting (XSS) protection

Django's template system protects us against the majority of XSS attacks by escaping specific characters ( ‘ “ < > & ) that are "dangerous" in HTML.

* Cross site request forgery (CSRF) protection

Django has built-in protection against most types of CSRF attacks. CSRF protection works by checking for a secret in each POST request.

* SQL injection protection

Django’s query sets are protected from SQL injection since their queries are constructed using query parameterization. A query’s SQL code is defined separately from the query’s parameters.

* User Data Protection

The system uses User Authentication using username and password which stored as part of Django Models corresponding to each user.

# 6 Open Issues

Some of the open issues that can be addressed in future scope can be:

* Improving the validations with respect to leave application and approval. Adding more fields could be another area of improvement.
* Additional functionalities like project assigning and management, work schedule management can be introduced into the scope of things.