# Software Design Document

## 

## 

## Employee Management System (EMS)

### v1.0

### 

### Created By: Syed Mohammad Ibrahim

### Email: [iamibi@umd.edu](mailto:iamibi@umd.edu)

### UMD ID: iamibi

## 1. Introduction

This software design document (SDD) is going to provide an architectural blueprint for Employee Management System (EMS). All the functionalities are discussed in detail here and how they are going to be met.

### 1.1 Purpose

The SDD document is going to illustrate all the designs and standards of EMS. This document is intended to help the developers, documentation personnels and the testers working on this project.

### 1.2 Scope

EMS is a web based application that will help the employees and IT department of an organization to manage their tasks or people. It will help employees to keep track of their tasks, managers to keep track of their employees and tasks completed, and the IT department to manage the employees within the EMS system. It is expected to reduce the workload of a manual setting environment where the organization either didn’t have a proper management system or were doing it manually. The goal is to implement the product as per the functional requirements, coding standards, security standards, software architecture and software interface. This is version 1.0 of EMS product and any new or missing feature(s) will be added in version 2.0 as per schedule.

## 2. Architectural Design

### 2.1 System Design

This section is going to outline the logical organization of the software and interaction between each of the software components.

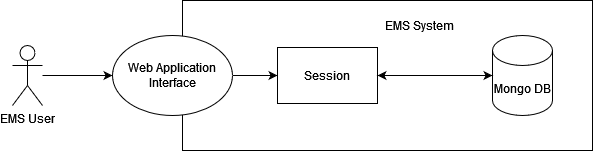


Figure 1 - System Design

An EMS Actor/User is going to be interacting with the Web Application Interface of the EMS system. The system is going to provide a session where all the functionalities are accessible as per the respective roles defined in the Software Requirements Specification (SRS). Any required data is going to be fetched from the mongo primary/secondary depending on the priority.

### 2.2 Database Design

The database used in this project is going to be MongoDB. Sharding will be performed on the email key stored in the database to make the indexing faster. MongoDB has two states, primary and secondary. Secondary state will be used when the required data is available and not urgent. This is done to avoid the load on mongo primary which is being written to and can impact the performance.

Following are the fields of database:

class EMSUsers {

// Create Indexes

index({ em: 1 }, { unique: true })

index({ \_id: 1 }, { unique: true })

field: \_id, name: employee\_id, type: BSON ID

field: fname, name: first\_name, type: String

field: lname, name: last\_name, type: String

field: rl, name: role, type: String

field: pnum, name: phone\_number, type: String

field: em, name: email, type: String, shard\_key: true

field: cat, name: created\_at, type: Time

field: uat, name: updated\_at, type: Time

field: pw, name: password, type: String

}

// For v1.0, the tasks will be automatically generated by the system and assigned to an employee.

class EMSTasks {

// Create Indexes

index({ emp\_id: 1 }, { unique: true })

index({ \_id: 1 }, { unique: true })

field: emp\_id, name: employee\_id, type: BSON ID

field: \_id, name: task\_id, type: BSON ID

field: st, name: status, type: String

field: uat, name: updated\_at, type: Time

}

The fields will be read and written to by a service layer. While inserting, the keys will be serialized and while fetching the keys will be deserialized.

### 2.3 System Architecture

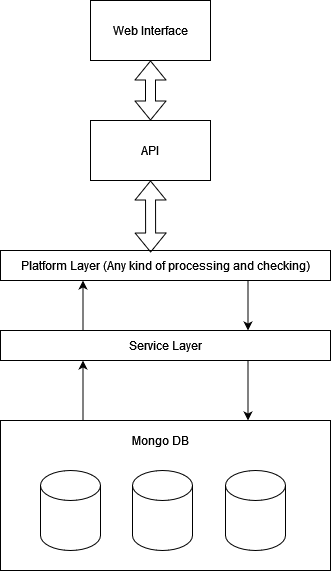


Figure 2 - System Architecture

The EMS system will have the database as Mongo DB at the lowest level which is only interacted by the service layer. The service layer will be responsible only for inserting, updating or removing any entry from the database. The service layer can be accessed by the platform layer where the processing and validations will take place. Only after the data is processed, can it go to the service layer. The web interface will be the one a user will be interacting with, providing inputs and getting responses. The requests are handled using the APIs dedicated for the purpose.

The API will be authenticated, that means they will require an access token to be passed for making a request or else the APIs will fail.

Following APIs are planned:

| API | POST /api/users/signup |
| --- | --- |
| Query Parameters | { first\_name, last\_name, email\_id, password, phone\_number, role } |
| Response Type | JSON |
| Description | This API will create a request for a new user that will be available to the IT department. |
| Response | { employee\_id, email, first\_name, last\_name, role, phone\_number } |

| API | GET /api/users/info |
| --- | --- |
| Query Parameters | { employee id } |
| Response Type | JSON |
| Description | This API will return the employee info upon request. It requires a requester id and employee id. |
| Response | { employee\_id, email, first\_name, last\_name, role, phone\_number } |

| API | POST /api/users/add |
| --- | --- |
| Query Parameters | { employee id, request id } |
| Response Type | JSON |
| Description | This will create a request to add an employee under a manager. |
| Response | {} |

| API | POST /api/tasks/update |
| --- | --- |
| Query Parameters | { employee id, task id } |
| Response Type | JSON |
| Description | This will update the task status. |
| Response | {} |

| API | GET /api/tasks/status |
| --- | --- |
| Query Parameters | { employee id } |
| Response Type | JSON |
| Description | Return the count of the tasks that are completed by the employee and total number of tasks that they had. |
| Response | { “tasks\_completed”: “xx”, “total\_tasks”: “xx”, “employee\_id”: “xxxx” } |

| API | DELETE /api/users/remove |
| --- | --- |
| Query Parameters | { employee id } |
| Response Type | JSON |
| Description | Removes the user and their associated tasks from the database. |
| Response | {} |

The supported status codes are:

200 - Success

400 - Bad Request

401 - Unauthorized

500 - Server Unavailable

## 3. Testing

### 3.1 Unit Testing

* A valid user is able to login into the system with valid credentials.
* An invalid user is not able to login into the system with invalid credentials.
* A new user is able to sign up using the valid credentials.
* An employee is able to change the task status
* A manager is able to view the details of an employee.
* A manager is able to request for an addition of an employee.
* An IT department person can approve a new employee request.
* An IT department person can approve a manager request to add an employee under them.
* An IT department person is able to view employee details.
* An IT department person is able to remove an employee.
* A valid user is able to logout of the system successfully.