Software Design Document

Employee Management System (EMS)

v1.0

Created By: Syed Mohammad Ibrahim

Email: 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.

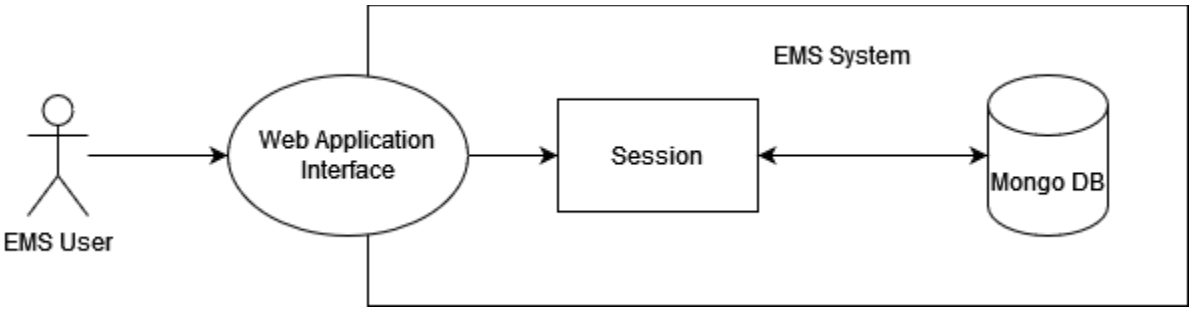

Figure 1 - System Design

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 {
        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: pwh, name: password_hash, type: String
        field: st, name: salt, type: String
        field: emng, name: manager_email, type: String, default: Null
}

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

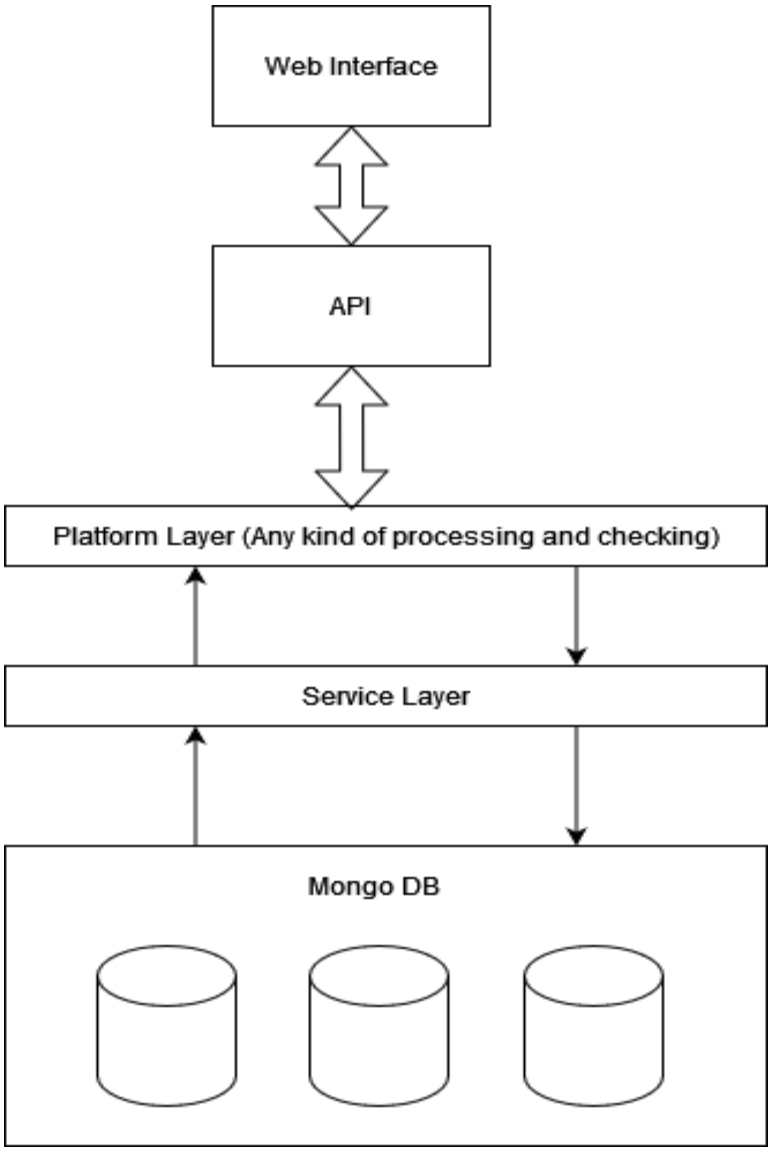

Figure 2 - System Architecture

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. A more detailed architecture is given below:

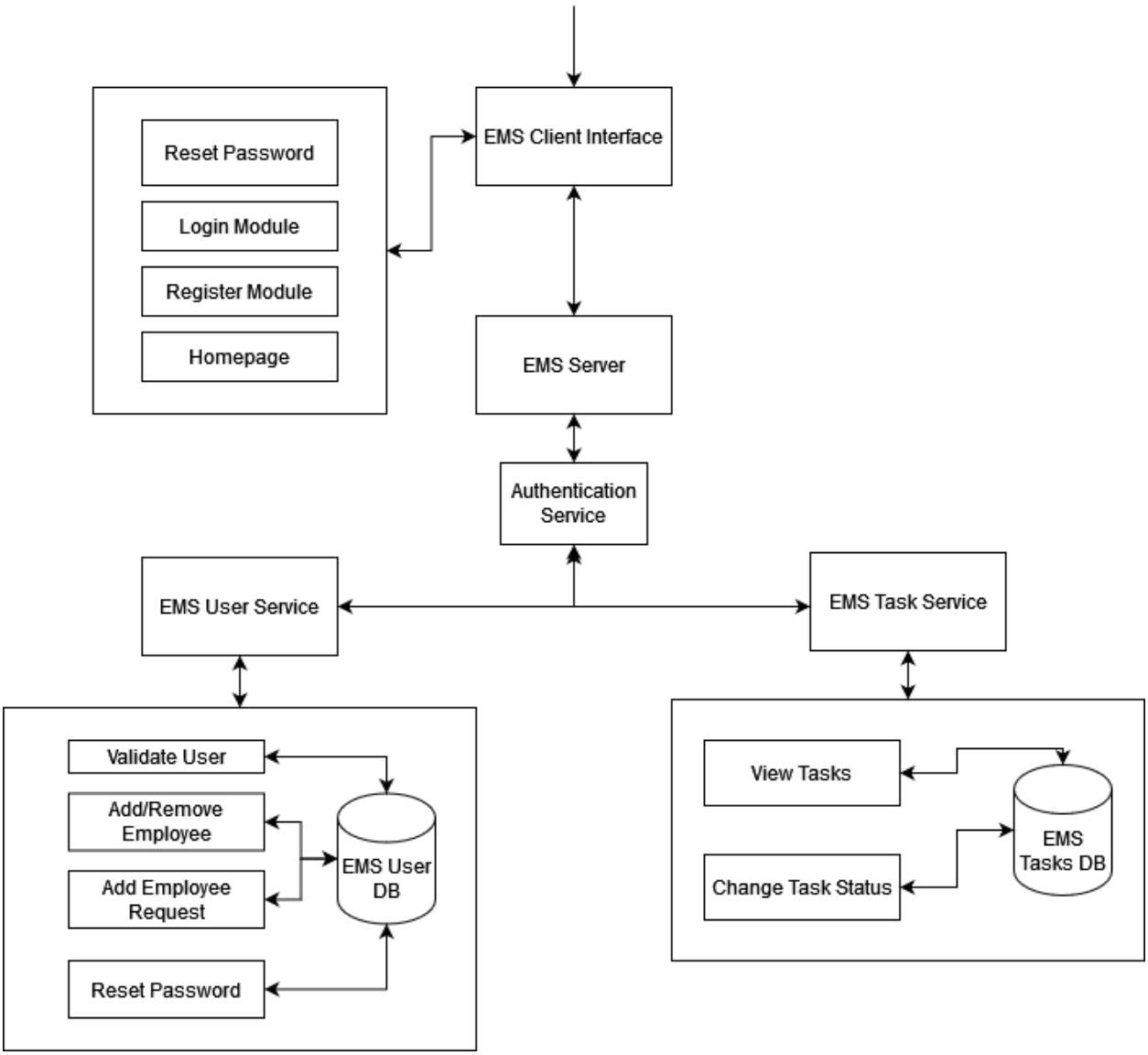

Figure 3 – System Architecture in Detail

2.3.1 EMS Client Interface

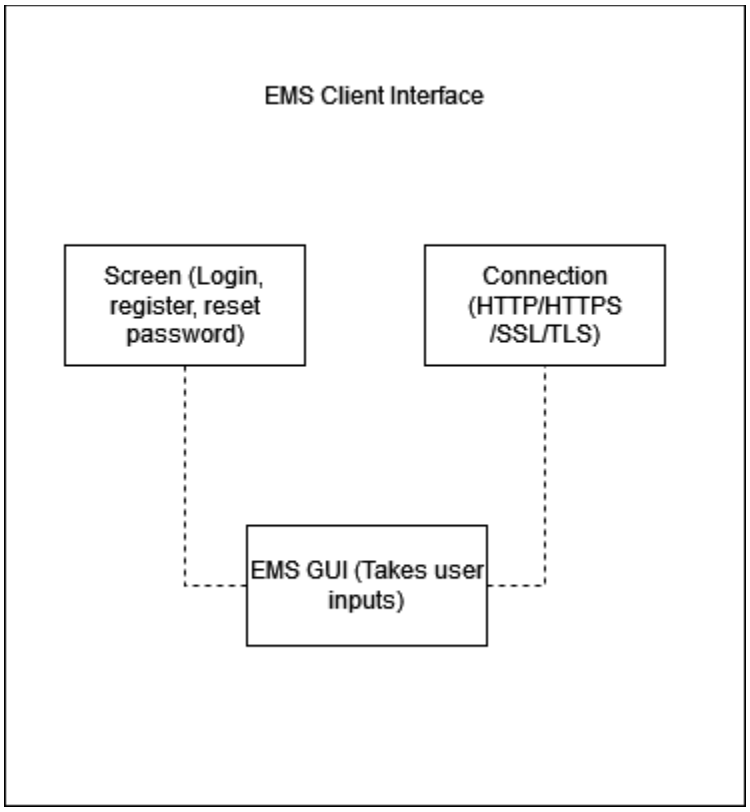

Figure 4 – EMS Client Preview

The EMS Web Client will have a connection security using HTTPS for all the visible screens. All the input handling will be done on the GUI and proper validations will be performed at this level initially. Once the inputs are verified, the data will be sent over to the EMS server to authenticate and verify, where additional sanity checking will be done, and a user token will be generated (if applicable).

2.3.2 EMS Server – User Service

The EMS user service will be responsible for any/all requests related to a user action. These include validation, authentication, add/remove of employee.

The user service will be handling the data inputs like username and password and will be storing it in the database or retrieving it. The plain text credentials will not be stored, rather a strong hash version will be stored in the database and the original one will be disposed off. The hashing algorithm to be used is PBKDF2 (with 100,000 iterations) available in .NET core. The following diagram depicts a clear view:

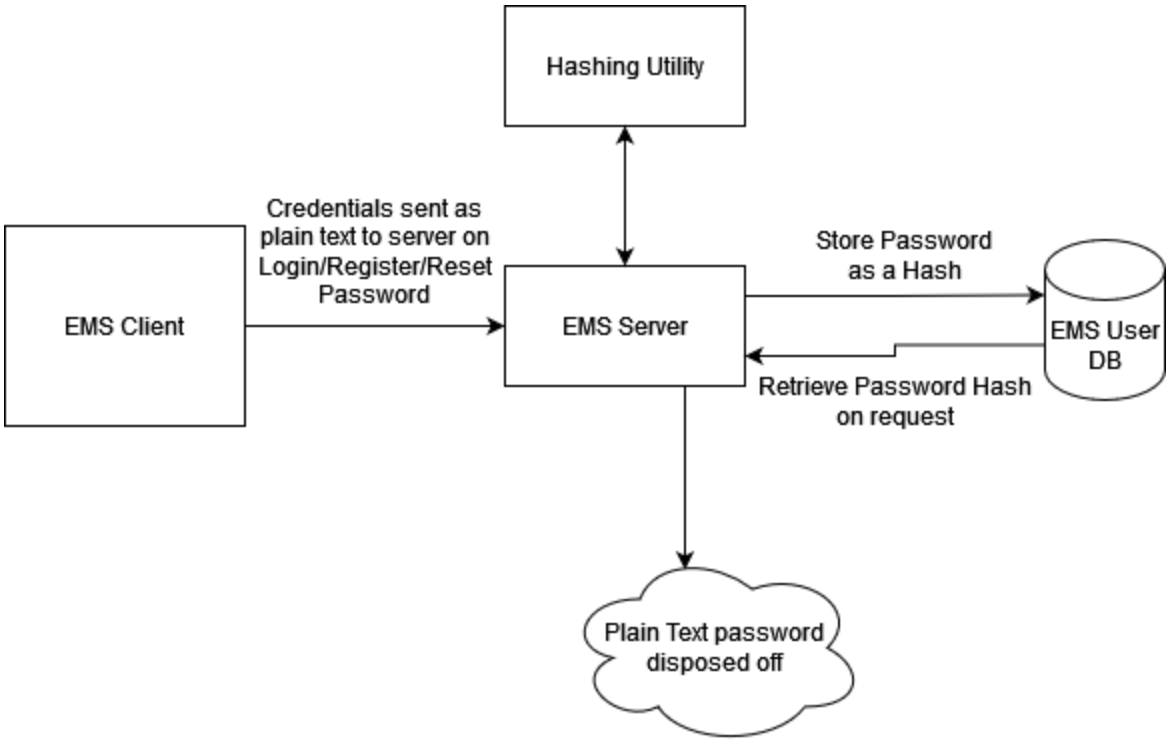

Figure 5 – Hashing Credentials

The following two things will be required by the hashing utility:

1. The plaintext password.
2. A randomly generated salt.

The plaintext password will be appended to the salt and then will be forwarded to the PBKDF2 algorithm to convert it to a hash. A new salt will be generated every time a password is to be stored. Once the hash is generated, it will be converted to a Base64 string and stored in the database.

The authentication service will be responsible for checking the requests made to the APIs and whether they contain a valid access token for the request.

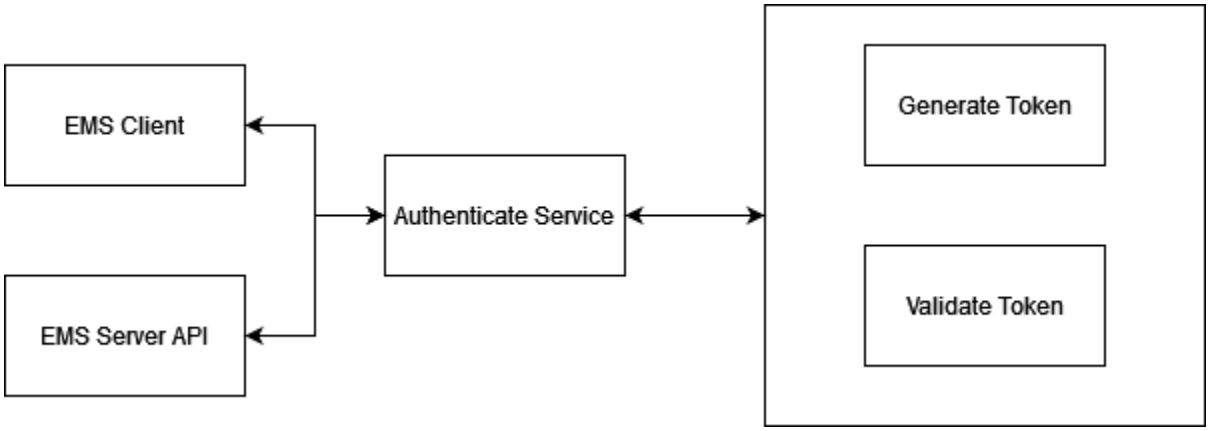

Figure 6 – Authenticate Service

2.4 Interface Design

The EMS Client can perform the following requests:

1. ValidateEmployeeCredentials()
2. RegisterUser()
3. GetEmployeeRole()
4. ResetPassword()
5. ChangeTaskStatus()
6. GetEmployeeTasks()
7. GetEmployeeTasksCompletedCount()
8. GetListOfEmployees()
9. GetPendingRequests()

The EMS Server can perform the following requests:

1. ValidateToken()
2. VerifyCredentials()
3. VerifyInputStream()
4. UpdateEmployee()
5. CreateEmployee()
6. AddEmployee()
7. RemoveEmployee()
8. GeneratePasswordHash()
9. VerifyPasswordHash()
10. StoreCredentials()
11. GetDataFromDatabase()

3. Testing

3.1 Unit Testing

- A valid user can login into the system with valid credentials.
- An invalid user is not able to login into the system with invalid credentials.
- A new user can sign up using the valid credentials.
- An employee can change the task status
- A manager can view the details of an employee.
- A manager can 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 can view employee details.
- An IT department person can remove an employee.

- A valid user can logout of the system successfully.