# Salary Portal

Akshay Jain

Advisor: Dr. Rakesh Matam

Department of Computer Science and Engineering
Indian Institute of Information Technology Guwahati

IIIT Guwahati April 2021

## Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Akshay Jain
April 2021

## Table of contents

## 1. Introduction

Generally, in the majority of educational institutes all calculations such as employee salary, employee tax, handling employee leaves, address allowances, are being done manually by the appropriate department which is a time-consuming task. Hence, a system is required that can perform all above said operations automatically. Moreover, the system should be user friendly, flexible, fast and secure.

Thus, we aim to automate the process in which the finance department handles the payment of the salaries of the employees.

We propose a portal which can be used by **all** the employees of the Institute, using this portal they can get their salary slip for each month. They can apply for leaves, claim their allowances and address any grievances with the finance department if any.

## 2. Existing Systems

There are many software available in the market that are being used by the organizations to calculate the salaries of its employees, however majority of them are either expensive or don't consider the variable needs of the organizations. Few of the popular software include Collabrill, Kredily, Timetrex[1].

The majorly common features include

1. Automatic Payroll processing
2. Flexible and easy to use
3. Encourages employee interaction and communication with the concerned department on the platform itself
4. Allows Employees and Management to upload and read the documents

## 3. Why a customized solution?

Upon analysis of different software mentioned[1] we recognized one important fact, that different organizations have different mechanisms, different parameters and different sets of guidelines to calculate their salaries.

Majority of open-source or low-cost software are not dynamic enough to accommodate the changing needs of the organization or taxes mentioned by

the government. Thus, a customized solution which shall give the end users more granular control is required.[2]

# 4. Design Process

## 4.1 Identifying the goals of the system

As discussed in section 3 of the report citing the need for dynamic requirements, we propose a web-portal which can be used by **all** the employees of the Institute, using this portal they can get their salary slip for each month and discussed in Section 1 ,employees can apply for leaves, claim their allowances and address any grievances with the finance department if any.

We further aim to reduce the work of the finance department, by pre-empting the employee salary info thus automating the salary calculation process for the Finance Department. Moreover, our system must be able to accommodate needs of different Institutes independently.

Our system automates the salary calculation process, provide a web interface for employees to view their salary and same interface for the admin (Finance Department) to add Employees, generate salary receipts, approve or reject leaves and handle all the requirements which are mentioned in the SRS.[3]

## 4.2 Identifying the stakeholders involved

The proposed system shall be majorly used **by the Institute Admin** (i.e., the finance department) and the Institute **Employee**

In Our System we have the following stakeholders,

1. **Super Admin** (Responsible for registering an Institute with our platform and assigning the admin)
2. **Admin** (Responsible for adding Employees to the platform, generate salary , approve/reject leave, allowances etc)
3. **Employee** (Applies for leaves, Claims Allowances, views Salary)
    a. We Broadly Classify All Employees into 3 Categories
        i. Staff
        ii. Faculty
        iii. Management

## 4.3 Constructing the Use Case Diagram

Based on the discussion in the section 4.1 and 4.2 and taking reference from existing works[2] we constructed the use case diagram which can be found here[4]

## 4.4 Choosing the tech stack

Given the proficiency of the team with the MERN (Mongo, Express, React.js, Node.js) stack and citing the time constraints and manpower constraints for the deliverables.

We chose MongoDB, Express.js[6], Node.js and EJS[7] as our tech stack.

### 4.4.1 Database Choice

We already mentioned that due to variable requirements amongst different organizations, data with such a platform will be loosely coupled and less structured. It's hard to determine all the properties beforehand and this will be a problem with fast paced agile development, since the agile model is the most suitable for development given the manpower and timeframe to present the deliverables.

NoSQL databases allow developers to be in control of the structure of the data, they are a good fit with modern Agile development practices based on sprints, quick iterations, and frequent code pushes.[5]

Using SQL, we would have spent a significant amount of time with database design, which would have been a dangerous decision given that all entities were not known beforehand.

# 5. Proposed System Overview

In the following section we discuss the functionalities provided to each stakeholder these are also mentioned briefly in the SRS Document[3]

1. **Super Admin**

    a. Register Institute with the system
    b. Register Admin
    c. Assign Admin to the Institute

2. **Institute Admin Role**
    a. Add Information about Institute
    b. Register all employees of Institute
        i. Individually
        ii. Via CSV
    c. Upload Salary Details (Tax and allowances included) of all Employees
        i. Individually
        ii. Via CSV
    d. Upload all Departments of the Institute
        i. Individually
        ii. Via CSV
    e. View all leaves applied by employees
    f. Approve/Reject leaves
    g. Approve/Reject Allowances
    h. Generate Salary of the Employee
        i. In the platform itself
        ii. Tax Receipt
        iii. PDF
3. **Employee Role**
    a. Apply for Leave
    b. View Salary Receipt
        i. In Web Platform Itself
        ii. PDF
        iii. Tax Receipt
    c. Claim Allowances

# 6. Development Process

## 6.1 Software Design Pattern followed

Given that current functional requirements[3] are not highly complex, given the stakeholders and time constraints we developed our application as a monolith to avoid complex design issues as compared to developing it as a set of microservices.

However, we recognise the fact that in given that we are doing agile development and complexity of the system may increase with increased testing or while moving into production, we followed the MVC architecture while designing our application.

Using MVC architecture we attain loose coupling between backend (Node.js & Express.js) and frontend (EJS), allowing us for easy migration in case the application is split into independent microservices.

## 6.2 Database Design

We followed agile model for development of software, we have embedded majority of the data in the employee and institute collection and avoided application-level joins, where ever we can[8] the existing ER diagram can be found here. [9]

# 7. Deliverables made by Akshay Jain

## 1. Implementation Details

a. Super Admin, following controllers were added
   i. Controller which allows super admin to register the Institute
   ii. Controller which allows super admin to add Institute admin and assign it to institute.
   iii. Populated the Super Admin frontend with relevant data

b. Admin, following controllers were added
   i. Controller to add departments of an Institute
   ii. Controller to add employee salary info
   iii. Controller to add allowances for an employee
   iv. Controller to register an employee, using the info of department, salary info, etc which admin enters into the system
   v. Controller to generate the salary
      a. Following controller generates the employee salary (for each month) from enrolment date till present month.
   vi. Controller to pay salary
      a. Following controller allows admin to mark salary as paid, and add receipts which are later visible to the employees.
      b. All the multimedia which is uploaded by the admin is saved under uploads/ directory in the same storage where the express server executes. (i.e., files are stored locally with the host OS, and served by express)
      c. Migration from local storage to AWS s3 is currently underway.
      d. Used npm library multer[10] to upload multimedia content
   vii. Controller to view complete salary details of the employee
   viii. Controller to add allowances
   ix. Populated the frontend templates for each page which uses the following controller

c. Employee, following controllers were added
   i. Controller to view salary details which were populated by the admin

## 2. Coding Practices

a. All the model, views and controllers reside in their separate directories.
b. Airbnb's coding style was followed while developing the application.[11]
c. While using version control git, master branch was locked all changes were made to sperate branches, only allowed way to merge changes was via pull requests on master, it allowed us to review code before merge.

## 3. Roadblocks faced

a. We used discriminators to inherit common employee schema , between faculty, staff and management, which caused all newly added employees to be of type management. After analysis we realized inheriting from same class is overriding properties, we separated the classes and didn't used inheritance.

   Following violated OOP principles, added a lot of redundant code and eliminated single source of truth. Although using this the error was resolved, however not using inheritance in the above case is a bad decision and hence remains an open issue.

## 4. Current Loopholes

a. Protection against Mongo Injection has not been implemented.
b. Since we are using session-based authentication it creates the following problems
    i. Using sessions makes the system prone to CSRF and XSS attacks we are not handling this case currently.
    ii. Session information on server side is lost in case of server failure, we can counter this problem by using an In-memory DB such as *Redis* and storing the sessions, however in that case it creates problem of scalability.
        a. Horizontal Scaling if we use Redis to store sessions
            i. In the following case we can't horizontally scale out redis thus we will have to rely on the load balancer, to route the client always to the same server where its session is stored.
        b. Vertical scaling is the solution, but it is short term expensive and adds a single point of failure.
c. All the multimedia content is currently saved in local storage and served by the express server, which is inefficient as compared to storing it on S3 and using nginx to serve the static files as with later we can define better caching strategies.[12]

### 5. AWS Deployment Strategy

We are using an AWS EC2 instance, to host our node.js application with express.js server the static content. The process is demonized by using npm library PM2

## 8. Future Work

We aim to complete missing functional requirements and leverage AWS services such as s3 to serve static content and SES to generate notifications.

Given the magnitude of the requirements, missing testing and required migration from local storage to AWS S3 and adding notifications via AWS SES, we require 3 months of further development before deploying v1 and moving into alpha testing.

## 9. References


1. https://www.softwaresuggest.com/blog/free-open-source-payroll-software/
2. http://www.diva-portal.org/smash/get/diva2:416795/fulltext01
3.https://docs.google.com/document/d/1v3PmEGj91w4GZOpxrm0OhanDCCrFSaxkBZtxmXSJHb8/edit?usp=drivesdk
4.https://drive.google.com/file/d/1J9SAV8BzzIdpP1dcFw5sKpuIvsQdFjgc/view?usp=drivesdk
5. https://www.mongodb.com/nosql-explained/when-to-use-nosql
6. http://expressjs.com/
7. https://ejs.co/
8. https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-3
9.https://drive.google.com/file/d/1TM6NKy9usrKba5ShoguPAsl7R2ZKhUXE/view?usp=drivesdk
10. multer - npm (npmjs.com)
11. https://github.com/airbnb/javascript
12. http://expressjs.com/en/advanced/best-practice-performance.html