Advanced Software Engineering

CEN 5011 U01/RXA-Fall 2015

Payroll Management System

Team number: 1

Team Members

Joma Rodriguez
Sharan Tej Kondumuru
Amit H Shenoy
Prafulla P Ghadage
Srinivas Reddy Manda
Sai Chaithra Allala
Yao Xiao

Submission Date: 11/4/2015 Professor: Peter J. Clarke

Abstract

Payroll is an important operation for every company or an organization to pay the employees of the respective company or organization. Based upon some details like basic pay and taxes, the salary calculation will be done. Taking these client inputs, the system consequently creates pay charges, pay slips, all timetables for debit and credit installments. There are many other services provided to do these payroll operations. But, this services consists of many errors and the security level is very low in the existing applications. The Payroll Management System is our proposed solution.

Payroll Management System, as the name suggests, used for calculating the pay slips of the employee's and other services like generating the Portable Document Format (PDF). There are functions like reporting the hours, submitting the timesheets and downloading a PDF file of the paycheck. All the operations which are involved with Payroll Management System are very well efficient, accurate and bug-free. Payroll Management System can be run on any desktop or laptop and thereby access the timesheets and submit them.

The utilization of Java dialect for system advancement, permits simple alteration of the system design. So the system design can be straight forwardly actualized to some other platform with slight adjustments. The main advantage of Payroll Management System is that instead of going through endless paper documents the software does everything. Payroll Management is an easy and simple way for gathering data about employee hours worked that are to be transferred into the system by isolating yet another sheet of manual processing.

This document provides an overview of the final system document for the Payroll Management application. It provides a thorough plan for the overall development of our application which includes the Requirements analysis and elicitation, Software Design which contains all the subsystem diagrams required for designing and the final Testing process.

Contents

1 Introduction-

This section is the introduction section which contains the purpose of designing Payroll Management system, the scope of the system, the Development methodology we have followed throughout the development process, a table containing all the terms used in the document along with their description and lastly an overview of the entire document.

1.1 Purpose of System-

Payroll administration can be very simple, involving the payment of just a handful of employees,
Or very complicated, involving payroll for employees. In some, very small companies, payroll may be-handled by the owner of the company or an employee. However, other companies may have many employees to pay and keep track of necessitating a well-planned, efficient payroll administration system. Payroll Management System is a simple web-based application and it is user-friendly.

The main purpose of Payroll Management System is to become a quick and easy to use web applications for users (Employee's/Employer's) that need to fill in the timesheets and the generating the pay slips. The process consists of calculation of salaries and tax deductions of the employees. The rate of tax deductions are pulled up from the taxfoundation.org where we can find the tax rates in different states. This application (P.M.S) is available on desktop platforms and it can be accessed through many browsers such as google chrome, Mozilla Firefox.

1.2. Scope of system.

This Application works in Multiple PC's installed on multiple Computers but sharing same database by which users of different department can use it sitting at different locations simultaneously. Basically, the users are the registered employees with Payroll Management System and an employer who maintains the timesheets and pay checks of employees. Gross pay calculation, calculation of payroll related taxes and contributions. Asks related to new employees and departing employees. Preparation and submission of tax declarations

Payroll disbursement services which is movement of all payments, calculated during the pay processing cycle net salary and tax. We can make the Application where the database will be hosted in order to manage the all departments which will be located in different places and by keeping domain of Application as Online

1.3 Development Methodology:

The software development model that we are implementing for Payroll Management System is

Unified Software Development Process (USDP). This is an incremental and iterative software development process. This software process is use case driven and risk driven. It uses the Unified Modelling Language (UML) to present the different models of the Payroll Management System. To describe the artifacts of the system we are using UML 2.0 notation. The diagrams such as class diagrams, use case diagrams, state machines, sequence diagrams and deployment diagrams are some of the models that are being used for the detailed design of Payroll Management System.

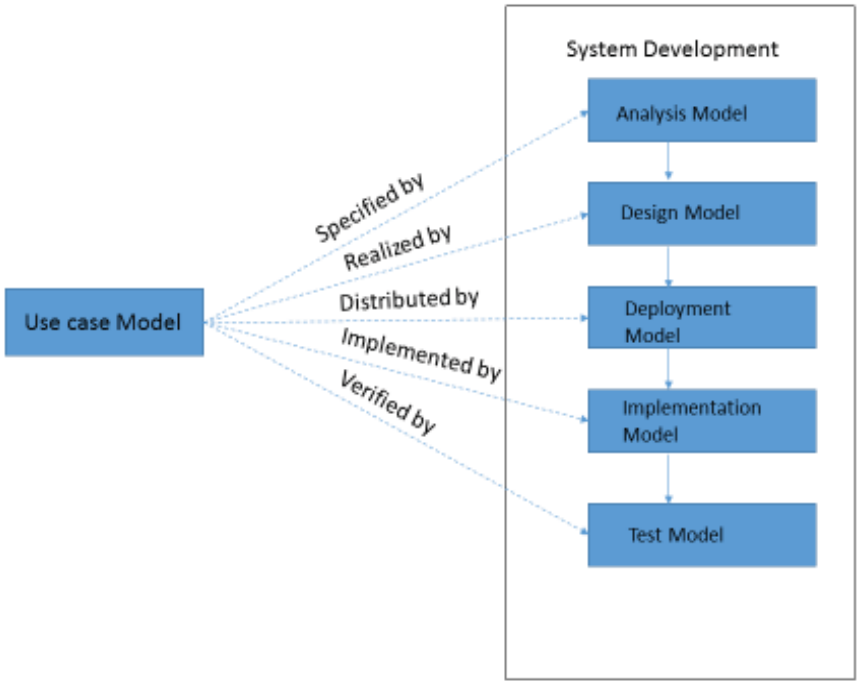


Figure 1.3.1: Unified Software Development Process

1.4 Definitions, Acronyms and Abbreviations

Actors: Represent roles played by individuals or any external objects.

Class Diagram: The diagram which shows the structure of the system with the help of classes and relationship between them. This diagram depicts the static structure.

DD: Design Document ER: Entity Relationship

Milestone: The end point of a software process activity. MVC: Model View Controller

OCL: Object Constraint Language. PMS: Payroll Management System

Sequence Diagram: The communication between different objects is represented through messages.

SRD: Software Requirements Document.

State Machine: It depicts the behavior of single object by going through all the states sequentially. It is basically a life cycle.

System: Framework which describes functionalities to achieve some objectives. UML: Unified Modeling Language

USDP: Unified Software Development Process

Use Case: A graphical representation of interaction between the objects of the system.

1.5 Overview of the document:

This Document outlines the procedure of developing the Payroll Management System. This document is composed of eleven chapters. And these chapters have sections. The Chapter 1 consist of the introduction to the Payroll Management System, It includes the purpose of the system and the requirements having its functional and nonfunctional requirements. This chapter also has a section where the design methodology is highlighted. This chapter is concluded with the definition acronyms and abbreviations. The Chapter 2 consist of the present existing system. It describes how the current system is working. The third chapter describes how the project plan was executed having the project organization with hardware and software requirements to complete the project. The fourth chapter consist of the requirements of the system having the functional requirements and the use case diagrams, scenarios. The chapter also consists of requirements analysis which has all the object diagrams and the class diagrams and also the sequence diagram.

The fifth chapter of this document consists of the proposed software architecture. This also has the subsystem decomposition, hardware and software mapping. The sixth chapter consist of the overview of the detailed design which has the state machines the object interaction diagram and the detailed class diagram. The seventh chapter in this document is the testing part of the Payroll Management System.

This chapter has detail explanation of the testing mechanism used to test the Payroll Management System. Then follows the Glossary which is the eighth chapter. The signature of all members of the project is the Ninth Chapter. The Chapter 11 is Appendix consisting of all the project schedule, the user interface diagrams, the detail class diagram showing attributes and methods for each class. The chapter also consist of the code for the subsystem and the code for the Test Driver and finally it has the diary of meeting and task as the last part of this document.

2. Current System:

There some well-documented shortcomings of payroll software, which include the expense of purchasing and upgrading the systems and the time and resources needed to train staff to use the programs.

Payroll software has some other interesting limitations that often are not considered when businesses map out the "ifs" and "hows" of implementing a program into operation, including:

Constant Archiving – Mountains of information are collected with each payroll cycle, and most Payroll software programs are designed to keep up. But, that information needs to be archived continuously. And, this daily process needs the help of a human hand. In a busy operation, setting aside this particular slice of time in a workday can become a slight hassle.

Limited Access – In most instances, payroll software is loaded onto one computer and that data Can only be accessed from that machine. This can be a hassle, especially if the payroll processing computer goes off the rails. The result could be that whole payroll process goes with it, which can cause headaches throughout the business.

Added Weight to Overhead – Implementing payroll software may need additional responsibilities for the employer. With an added loaded work employee comes the cost of an additional salary and benefits. Also to be considered is the cost of technical support when the inevitable software glitches occur.

Risk of under withholding – The Internal Revenue Service notes that some payroll software systems are unable to distinguish additional voluntary withholding amounts from regular withholding when calculating catch-up withholding for the current tax year. This kind of glitch does not apply to all payroll software programs, but if not identified or accounted for, a business can be fined for under withholding.

When considering these limitations of payroll software, it's key to keep in mind that not all payroll software programs are made the same, so they don't all have the same shortcomings. Finding the

best fit for a business means identifying the payroll software that offers the most benefits with the least amount of limitations.

3. Project Plan:

The project plan is the governing document that establishes the means to execute, monitor, and control projects. The following subsections provide the complete overview of the software specifications requirements documentation for the product Employee Payroll Management. The plan serves as the main communication vehicle to ensure that everyone is aware and knowledgeable of project objectives and how they will be accomplished. A project plan is used to control schedule and delivery.

Determine which tasks are dependent on other tasks, and develop critical path. Develop schedule, which puts all tasks and estimates in a calendar. It shows by chosen time period (week, month, quarter or year) which resource is doing which tasks, how much time each task is expected to take, and when each task is scheduled to begin and end.

3.1. Project organization:

Phase - I

Joma- Software Design & Modeler Sharan-Timekeeper & Modeler Amit-Analyst & Modeler Prafulla-Minute taker & Modeler Srinivas Reddy- Analyst & Modeler Sai Chaithra Allala-Tester & Modeler Yao Xiao- Design & Modeler

Phase-II

Joma- Team Leader & UI Developer Sharan-Timekeeper & Document Editor Amit-Analyst & Object designer Prafulla-Minute taker & Object designer

Srinivas Reddy- Software Design & Developer Sai Chaithra Allala-Testing & Document editor Yao Xiao-Team Leader & Developer

Phase III

Joma- Timekeeper & Software design Sharan-Testing & Configuration Manager Amit-Analyst & UI Developer

Prafulla-Developer & Configuration Manager Srinivas Reddy-Team Leader & UI Developer Sai Chaithra Allala-Minute taker & Implementer Yao Xiao-Software Design & Implementer

3.2. Hardware and software requirements to complete the project.

Client	Desktop or PC
Processor	Intel Celeron CPU 1000M 1.8 GHz
RAM	4.00 GB
System Type	64-bit, x64-based processor

Software for client sub-system:

Operating system	Windows XP, Windows 7, Windows 8, Windows 8.1, Windows 10
Browser	Google Chrome, Mozilla Firefox, Internet Explorer

3.3. Work breakdown:

The tasks in the development of Payroll Management System were divided into three milestones:

Milestone 1 consisted of the completion of the Use Case Phase and the Analysis Phase. This Resulted in a software requirements document handed in to the client. It also covered Static and Dynamic models.

Milestone 2 will consist of the completed of Design Phase and started on the model driven Software development approach. After generating models and code, the necessary Transformations will be made. This resulted in a design document handed in to the client.

Finally,

Milestone 3 will consist of the completion of the Testing Phase as well as the completion of the entire project. Please refer to Appendix A for project schedule and Appendix D for diary.

3.4. Cost estimate for the project.

Functional Points:

			ILF			EIF			EI			EO			
Features	Comments	Reference	Low	Avg	High	Low	Avg	High	Low	Avg	High	Low	Avg	High	Low
Login			0	2	0	0	0	0	0	0	0	1	0	0	1
Search Timesheet			1	0	0	0	0	0	0	1	0	0	0	0	0
Approve Timesheet			0	0	0	0	0	0	0	1	0	0	0	0	0
Submit Timesheet			0	1	0	0	0	0	0	0	0	0	0	0	0
Gross Salary			1	0	0	0	0	0	0	0	0	0	0	0	0
Net Salary			1	0	0	0	0	0	0	0	0	0	0	0	1
New User			0	1	0	0	0	0	1	0	0	0	0	0	0
Security Question			0	0	1	0	0	0	0	0	1	0	0	0	1
Work Profile			0	1	0	0	0	0	0	1	0	0	0	0	0
Logout			1	0	0	0	0	0	2	0	0	0	0	0	0
Total Artifacts			4	5	1	0	0	0	3	3	1	1	0	0	3

Total FP	149

SLOC:

Features	Comments	Reference	Language	Estimated SLOC
Paycheck Generaotr			Java	200
No Duplicte Submission			Java	300
			Total:	**500**

Effort:

Cocomo II

PM = A*Size^E*Product(All Effort Multipliers - EM)
Exponent E = B+(0.01*SUM(Scaling Factors))

Scaling Factors

SF	Description	Level	Value
Maturity	Process Maturity	Nominal	4.68
PREC	Experience of similar Projects	Low	4.96
FLEX	Flexibility required in the System	Very Low	0
TEAM	Team Conhesiveness	Extremely High	0
RESL	Project Risk and Architectural Complexity	Low	1.41

Effort Multiplier EM

EM	Description	Level	Value
RCPX	System reliability, complexity and size indicator	Very Low	0.6
RUSE	Reusablity cocern with respect to current and future projects	Nominal	1
PDIF	Platform Difficulty	Extra Low	0.87
PERS	Personal capability of team. Like technical capability of Programmers, Designers and testers.	High	0.83

Constants	Value
B	0.91
A	2.94
E	1.0205
EM	0.3769362

Consolidated Size and Effort

Technology	Java
Increase due to lifecycle	0%
SLOC per FP	53
SLOC	8397
PM	9.7204
Man-days	184.6875
FP from LOC	158.4340
Hours per FP	9.3257

SF Scale	PREC	FLEX	TEAM	RESL	Maturity
Very Low	6.20	0.00	5.48	0.00	7.80
Low	4.96	1.01	4.38	1.41	6.24
Nominal	3.72	2.03	3.29	2.83	4.68
High	2.48	3.04	2.19	4.24	3.12
Very High	1.24	4.05	1.10	5.65	1.56
Extremely High	0.00	5.07	0.00	7.07	0.00

EM Scale	RCPX	RUSE	PDIF	PERS	PREX	FCIL	SCED
Extra Low	0.49	0.95	0.87	2.12	1.59	1.43	1.00
Very Low	0.60	0.95	0.87	1.62	1.33	1.30	1.00
Low	0.83	0.95	0.87	1.26	1.22	1.10	1.00
Nominal	1.00	1.00	1.00	1.00	1.00	1.00	1.00
High	1.33	1.07	1.29	0.83	0.87	0.87	1.14
Very High	1.91	1.15	1.81	0.63	0.74	0.73	1.43
Extremely High	2.72	1.24	2.61	0.50	0.62	0.62	1.43

Effort Summary:

Category	Effort	Comments/Assumptions
Requirements/Design Effort	55.41	Outputs are Technical Design and Information Architecture doc. Information Architecture doc will have guidelines for interface design.
Development Effort	64.64	Source Code
Test Planning	18.47	Output are Test Plan and Test Cases
Testing Effort	18.47	Output is Test Results
Documentation Effort	11.08	Outputs are Deployment and Build Documents
Deployment Effort	3.69	Effort to deploy application
Off-shore Management	9.23	Output is Detailed Project Plans, Risk and Issues tracking, Weekly status reports etc.
On-site Management Effort	3.69	Client Communication. Output is usually any list of issues or improvement opportunities
Total Effort	**184.69**	

4. Requirements of System

In Payroll Management System (P.M.S) we propose to have a centralized application over the web that would handle the requests from the employee and the employer. In this system we propose to have a separate login for the employee as well as the employer. This grant's them different access to different resources. The purpose of our system is to simplify the tasks behind handling the payroll of a company since, payroll is crucial as payroll and payroll taxes considerably affect the net income of most companies and because they are subject to laws and regulation.

In PMS, the employee has the option to fill his timesheet and submit it for processing. The employer would then approve the timesheet and further process it. The timesheet approved is further used for calculating the salary and calculate the taxes which would then in turn produce a paycheck. This paycheck can also be viewed by the employee. The employer also has the option to modify the submitted timesheet of a use if he/she finds some error in it.

4.1 Functional Requirements. –

In this section, we are presenting the functional requirements of Payroll Management System (PMS). Generally, the functional requirements depicts the specific behavior or describes high level functionalities of Payroll Management System.

1. The system shall display the homepage of the Payroll management system. (see use case PMS_01_Login)
2. The system shall allow the employer to delete the employee. (see use case PMS_02_DeleteEmp)
3. The system shall employer to search the Employee's timesheet information. (see use case PMS_03_SearchEmpTS)
4. The system shall modify the employee's timesheet information. (see use case PMS_04_ModifyTS)
5. The system shall display the homepage of the Payroll management system. (see use case PMS_05_ApproveTS)
6. The system shall calculates the tax amount that needs to be withheld from the modeler's paycheck. (see use case PMS_06_Tax)
7. The system shall display the calculated salary of the respective employee according to the timesheet per week schedule. (see use case PMS_07_CalcSal)
8. The system shall allow to save the Timesheet. (see use case PMS_08_SaveTS)
9. The system shall generates a downloadable PDF file to show the pay check of an employee. (see use case PMS_ 09_PayCheck)
10. The system shall allow the employer to deposit the salary in to the employee bank account. (see use case PMS_ 10_DirectDeposit)
11. The system shall allow the employer to add a new department. (see use case PMS_11_AddDept)
12. The system shall allow the employer to remove the department. (see use case PMS_12_RemoveDept)
13. The system shall allow the employer to add a new employee. (see use case PMS_ 13_AddEmployee)
14. The system shall the employer to remove an employee from a particular department. (see use case PMS_ 15_RemoveEmployee)
15. The system shall allow the employer to remove an employee from a particular department. (see use case PMS_15_RemoveEmployee)

16. The system shall allow the employee to change/update its information. (see use case PMS_16_UpdateEmployee)
17. The system shall allow to see the work profile of an employee. (see use case PMS_17_WorkProfile)
18. The system shall allow the employee to submit the timesheet. (see use case PMS_18_SubmitTimesheet)
19. The system shall generate password automatically when an employee is newly registered. (see use case PMS_19_ForgetPasswor)
20. The system shall 20. PMS shall allow the employee to change the password (see use case PMS_20_changepwd)
21. The system shall allow the employee/employer to logout of the system and directs it to homepage (see use case PMS_21_logout)
22. The system shall allow the employee/employer to logout of the system and directs it to homepage. (see use case PMS_001_Injector)
23. The system shall notify the employee/employer through an email when the password is changed. (see use case PMS_002_PwdNotificator)
24. The system shall deny simultaneous logins. (see use case PMS_003_Multi.login)
25. The system shall deny the duplicate submission of timesheets. (see use case PMS_004_duplicate)
26. The system shall ask security question, when an employee tries to retrace his/her password. (see use case PMS_005_security)
27. The system shall logs out automatically if there is no certain action for a specific period of time. (see use case PMS_006_Timeout)
28. The system shall change the input to label when an employee submits the timesheet. (see use case PMS_007_textToLabel)

In this section we provide the non-functional requirements of Payroll Management System. These requirements are composed of reliability, usability, security, maintainability, scalability and performance. These attributes pose the constraints on the Payroll Management System. These have the same importance of functional requirements as they ensure the efficiency and performance.

Usability: No basic training is required for an individual to use this application.

Reliability: Mean Time to Failure – for most use cases one failure per week of operation is within acceptable parameters. Average Downtime – for most use cases one hour per month of operation

is within acceptable parameters. The PMS_05_ApproveTS, PMS_17_WorkProfile have possibility of failure when parameters are not accepted.

Performance: Most use cases have requests that return within average of 8 seconds. The PMS_05_ApproveTS, PMS_06_Tax, PMS_03_SearchEmpTS and PMS_10_DirectDeposit respond within 10 seconds and PMS_006_Timeout occurs with 2 minutes.

Supportability: The system shall support all modern browser implementations including mobile. The system is constructed purely in Java 8 and thus will be portable provided an appropriate environment. The system is guaranteed to run on Google Chrome, Mozilla Firefox (desktop).

Implementation: Client is implemented with HTML. Customer mandates the implementation to support mobile devices and that the server be constructed using Java and the Eclipse IDE

4.2 Use Case Diagrams.

4.2.1 for all use cases.

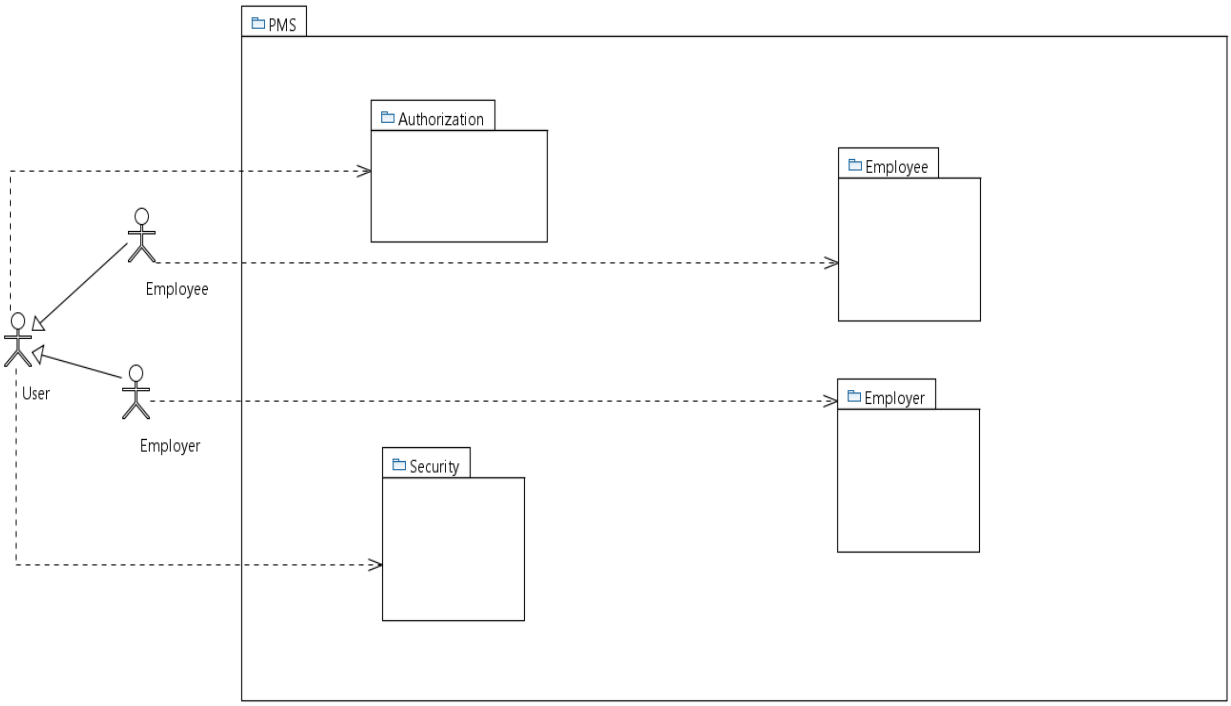


Fig 4.2.1: Use case for all the use cases in Payroll Management System

4.2.2 Implemented Use Case's. -

4.2.2.1 All Use Case's Implemented:

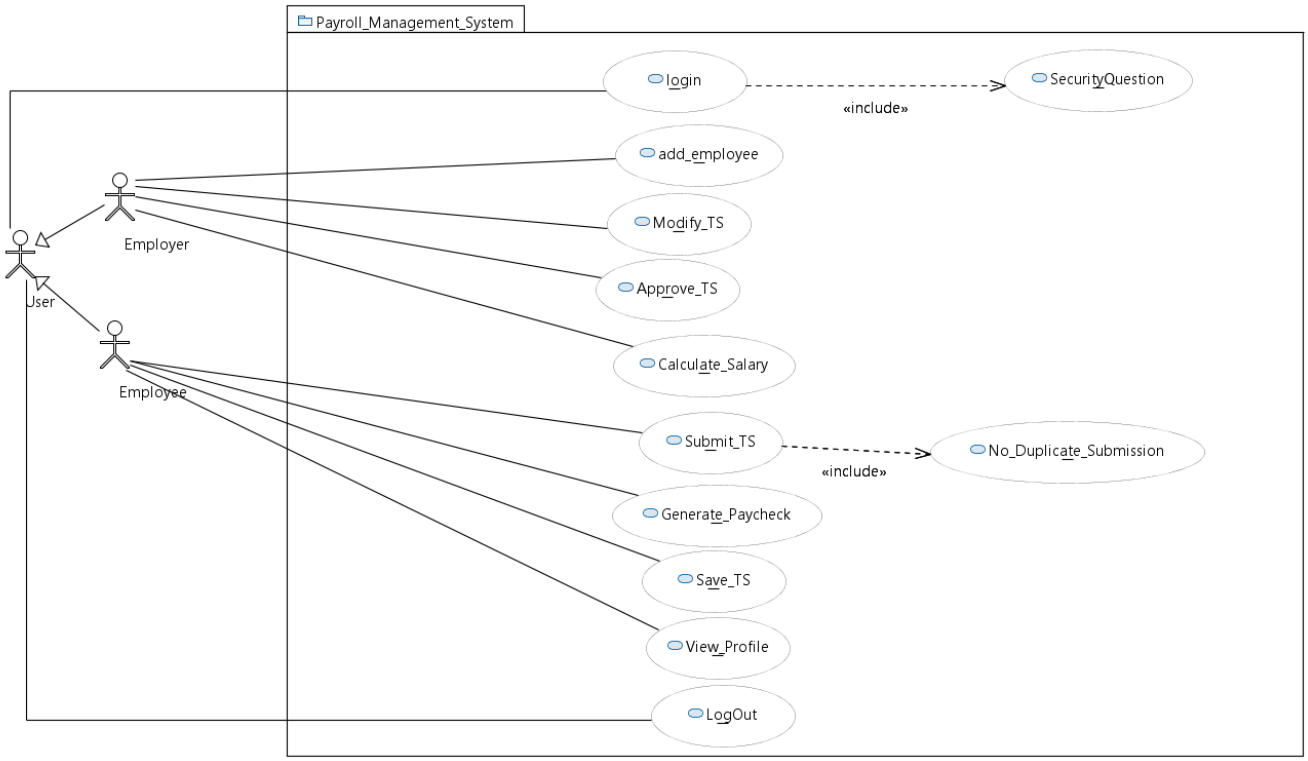


Fig 4.2.2.1 Implemented Use Case's

4.2.2.2 Individual Use Case Diagram:

Authorization Package Use Case Diagram:

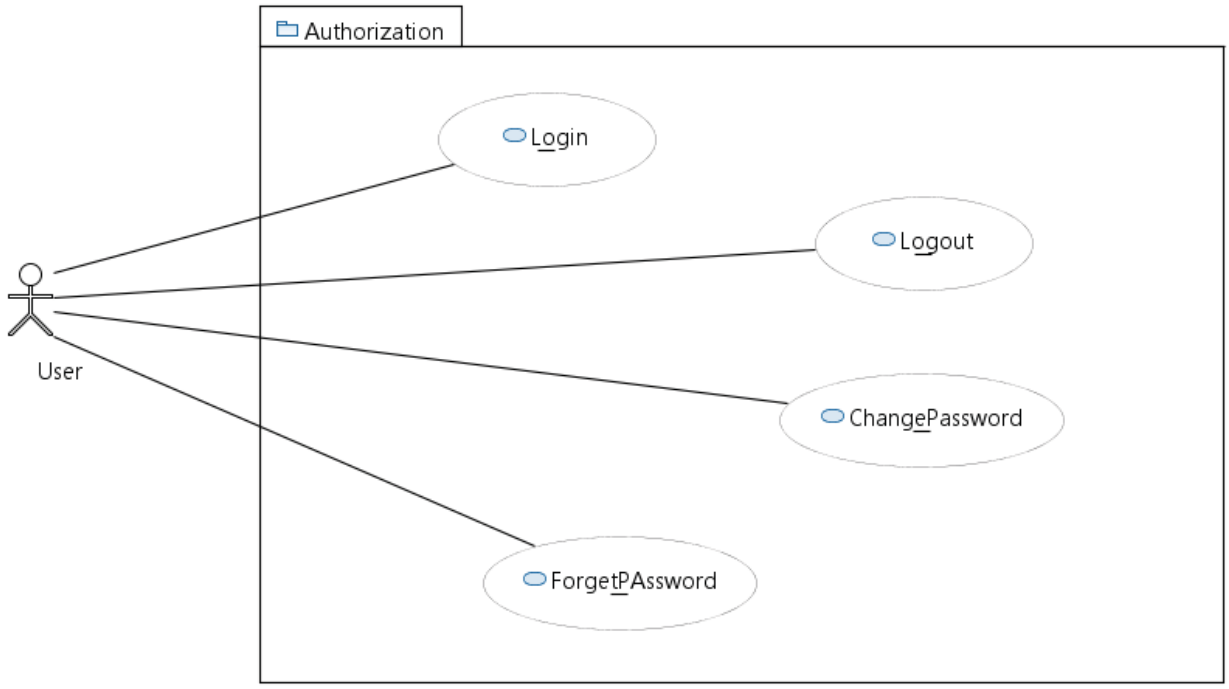


Fig 4.2.2.1 Authorization Package Use Case Diagram.

Employee Use Case Diagram:

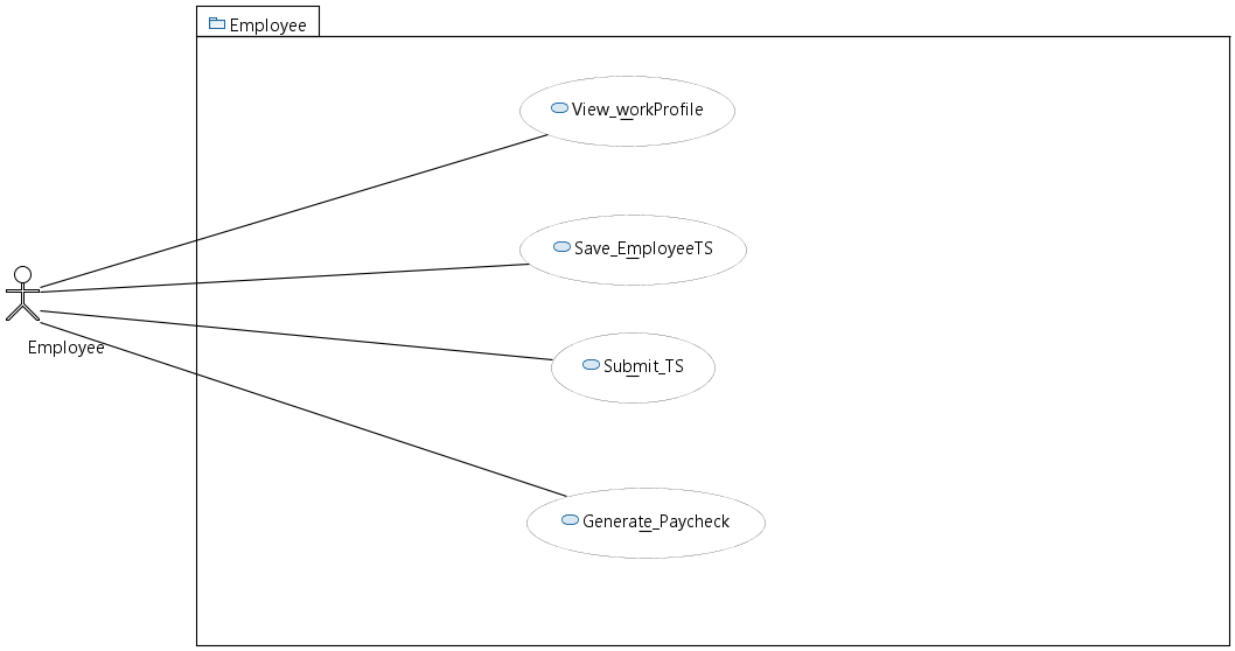


Fig 4.2.2.2: Employee Use Case Diagram.

Employer Use Case Diagram:

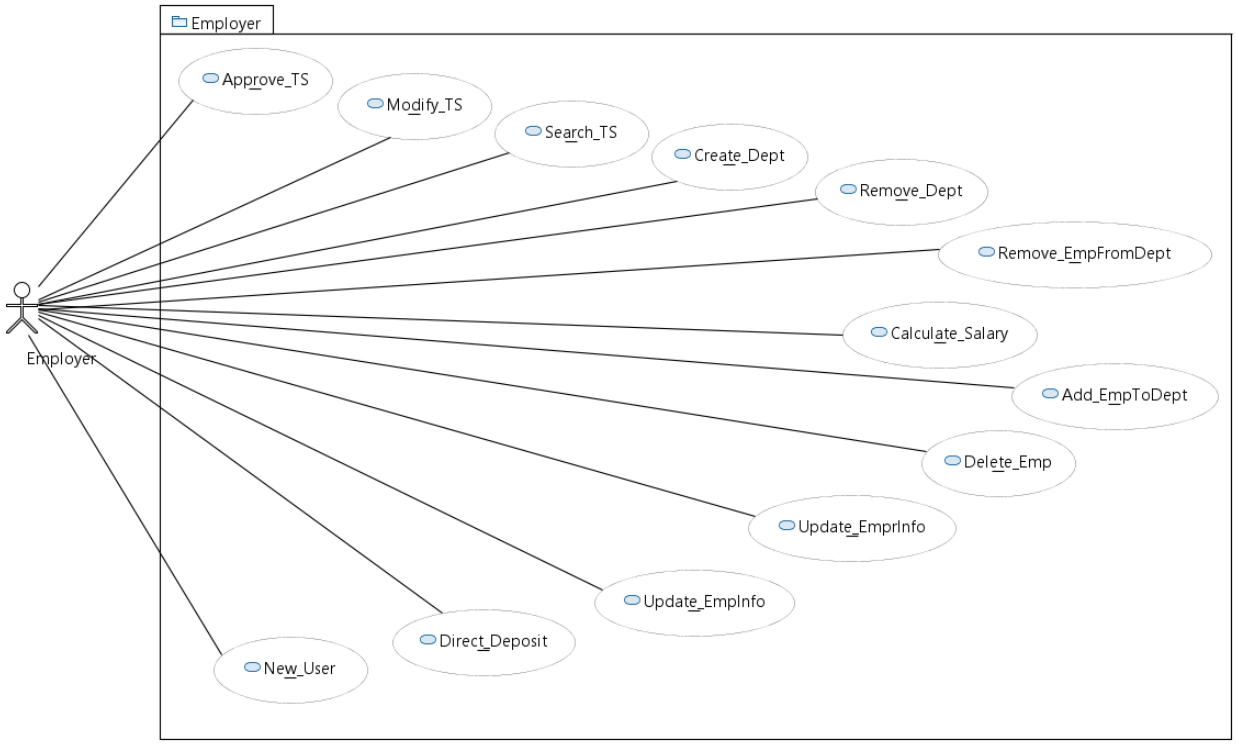


Fig 4.2.2.3: Employer Use Case Diagram.

Security Use Case Diagram:

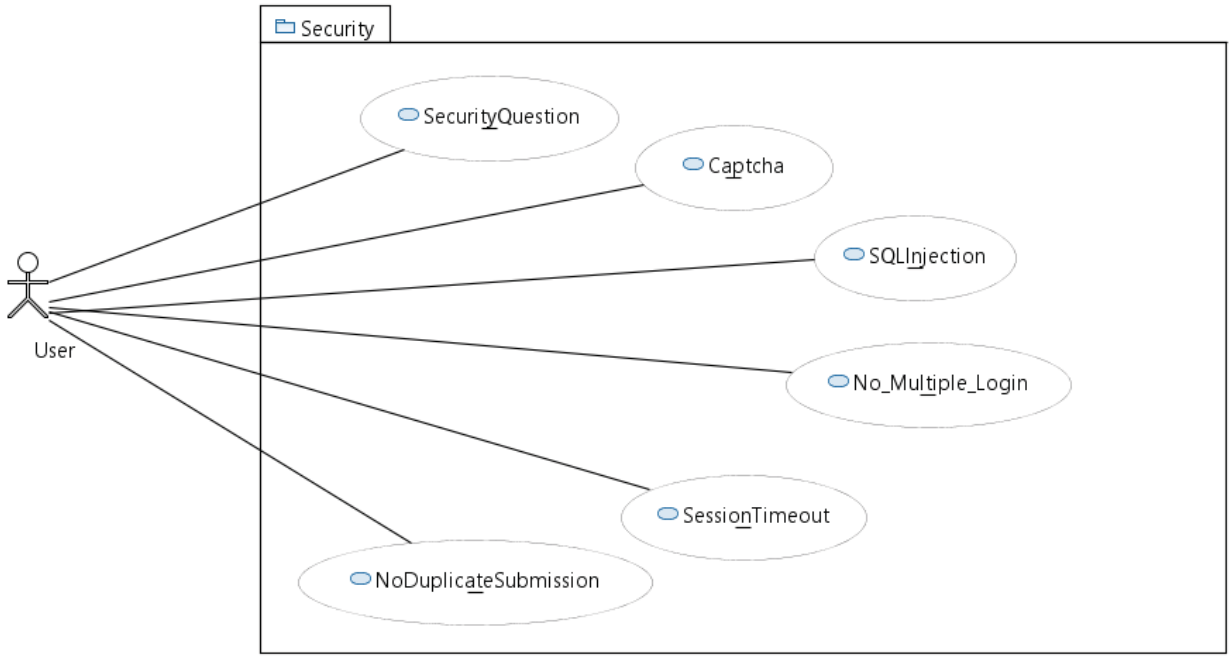


Fig 4.2.2.4: Security Use Case Diagram.

4.3 Requirement Analysis.

In this section, we present the scenario descriptions for the 10 use cases that we are implementing of which there are two security use cases. Requirements analysis in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software or system requirements.

4.3.1 Description of Use case scenarios.

Use Case ID: PMS_13_AddEmployee.

Actor: the actor in this case is Amit who is the Employee.

Pre-conditions:

1. Amit is logged in using valid employee_id "ashen007".
2. Amit is in the manage tab of the webpage.

Description:

Trigger:

Amit clicks on the manage tab. He has two options to manage Employee and Employee.

1) Amit clicks on the "Employee" option
2) The PM system will then give options to Amit like to add a Employee.
3) Amit clicks on add Employee.
4) Then Amit fills in the info like:

DeptName: Deployment.

DeptLocation: Room155.

DeptManager: Popya.

5) Then Amit clicks on save button.

Post-conditions: The PM system will create a new Employee with name "Popya" to the department "Deployment"

2)Use Case ID: PMS_08_SaveTS

Actors: Tom (Employee).

Pre-conditions:

1. The user Tom logs in with her credentials.
2. **Description:**

The employee Tom's timesheet will saved.

Trigger: Tom clicks "Timesheet" menu item in PMS.

The system responds by …

1. Tom fills the Timesheet for particular duration of work. 2. The system shows values inserted in Timesheet.

3. Tom saves the Timesheet values inserted using SaveTimesheet button.

Post-conditions:

1. The timesheet is forwarded towards calculating the Paycheck.

3)Use Case: PMS_09_PayCheck

Actors: Maria, Employee

Pre-conditions:

1. Maria has user id "1123" in payroll management system.
2. Maria logged into the system with username "maria007" and password "pmsmaria123".

Description:

Trigger: Maria clicks the "View Pay Check" button in the login page. The system responds by …

1) PMS presents the "View pay checks" page.
2) Maria views a list of payments on her browser.
3) Maria clicks on view button on the payment date "09/23/2015".
4) Maria sees the full salary details like date, amount and status.

Post-conditions:

1. Maria continues to use PMS system.

4) Use Case ID: PMS_07_CalcSal

Actors: Maria (Employer).

Pre-conditions:

1) The user Maria logs in with her credentials.

Description:

1) The employer Maria will click on the salary in the menu.

Trigger: Maria clicks "CalcSalary" menu item in PMS.

The system responds by …

1) Maria sees the Salary page opened.
2) Maria searches Tom's information by Tom's ID (001).
3) The system shows the salary in the menu.
4) Now Maria will select the Calcsalary button of the Tom..
5) Now Maria will view the Calcsalary of the Tom.

Post-conditions:

1) The timesheet is moved back to salary window.

5)Use Case ID: PMS_05_ModifyTS

Actors: Maria (Employer).

Pre-conditions:

The user Maria logs in with her credentials.

Description:

The employer Maria will modify Tom's timesheet which had already been submitted.

Maria clicks "SearchEmployee" menu item in PMS.

The system responds by …

1) Maria searches Tom's information by Tom's ID (001).

2) The system shows the unapproved timesheet for the Tom's ID i.e. 001 that Maria is searching for.

3) Now Maria will select the timesheet she is looking for and will click on the edit button.

4) Now Maria will edit the timesheet and click the approve button when she verifies that the information provided is valid.

Post-conditions:

1) The timesheet is forwarded towards calculating the Paycheck.

6) Use Case ID: PMS_03_SearchEmpTS **Actors:** Maria(Employer)

Pre-conditions:

Maria logs in with her id and password.

Description:

Maria will search for Tom's timesheet.

Trigger:

Maria clicks "SearchEmployee" menu item in PMS.

The system responds by…

1) Maria types in Tom's 007.
2) Maria clicks "Search" button.
3) EMS system shows all the timesheets of Tom.

Post Condition: None

7) Use Case ID: PMS_02_Login

Use Case Level:

Details:

Tom enters his/her username "Maria001" and password "Password1234" to log in to the system.

Actor: the actor in this case is Employer(Maria).

Pre-conditions:

Maria should be authenticated employer of the system.

Description:

Trigger:

The system responds by …

a. The use case begins when the Maria enters username "Maria001" and password "Password1234" in the login page.

b. Maria then clicks on "Sign in" button in the login page.

c. PMS system authenticates username "Maria001" and password "Password1234"

Relevant requirements:

Post-conditions:

1) Maria is taken to PMS Homepage.

8) Use Case ID: PMS_21_logout**.**

Actors: Maria, Employee

Pre-conditions:

1) Maria has an account in payroll management system.
2) Maria tries to login with username “maria007” and password “pmsmaria123”.

Description: This will help the user to logout.

Trigger:

Maria clicks the “logout” button in the home page. The system responds by …

1) Maria is logged out from the system.
2) Maria is shown a message saying “Successfully logged out”.

Post-conditions:

1) Maria is redirected to login page.

9) Use Case ID: PMS_17_WorkProfile

Actor: Modeler

Pre-conditions: The modeler wish to see the profile for the employee where he sees of that employee’s information.

Description: The modeler will check the details of that employee.

Trigger: The modeler clicks the “work profile” button on the page.

The system reponds by…

1) Redirecting the modeler to the work profile page.
2) The modeler selects an employee.
3) An SQL query is executed to show the information on the database.
4) The modeler is redirected to the employee’s profile.

Post-condition: The modeler can now see the information of the employee.

10) Use Case ID: PMS_05_ApproveTS

Actors: Maria (Employer).

Pre-conditions:

1) The user Maria logs in with her credentials.

Description:

The employer Maria will approve the employee Tom's timesheet which had already been submitted.

Trigger: Maria clicks "SearchEmployee" menu item in PMS.

The system responds by …

1) Maria searches Tom's information by Tom's ID (001).
2) The system shows the unapproved timesheet for the Tom's ID i.e. 001 that Maria is searching for.
3) Now Maria will select the timesheet she is looking for and will click on the view button. Now Maria will view the timesheet and click the approve button when she verifies that the information provided is valid.

Post-conditions:

1) The timesheet is forwarded towards calculating the Paycheck.

11) Use Case ID: PMS_004_duplicate.

Actors: Maria, Employee

Pre-conditions:

Maria logs in with her employee name "maria007" and password "pmsmaria123".

Maria fills timesheets.

Description: The system checks for no duplicate submission.

Trigger: Maria clicks "submit timesheets" button in the submit timesheets page.

The system responds by …

1) Maria clicks "submit timesheets" button.

2) PMS saves the form values “001” in the database.

3) A transaction id flag “1” is stored in the database.

4) Maria tries to submit the timesheets.

5) PMS checks for transaction id flag in the database.

6) If the transaction id flag is “1” in the database, an error message saying “timesheet already submitted” is displayed to Maria.

Post-conditions:

1) Maria clicks on “continue” button after submitting form.

2) Transaction id “ST004” is deleted from the database after two weeks.

12) Use Case ID: PMS_005_security.

Use Case Level: High level.

Details:

1) When Popya forgets the password for his account and wants a auto generated password from the PM system

2) When Popya wants to change the auto generated password “12345” to the password of his choice.

3) Popya interacts with the system using his id that is “Popya007”.

4) This is done if Popya forgets his password or wants to change his password for the Security purposes.

Actor: the actor in this case is Popya who is the Employee.

Pre-conditions:

1) Popya should have a valid employee_id that is “Popya007” in the case of forget password and change password.

2) Popya should be authenticated employee of the PM system.

3) Popya should have successfully logged in to the system using user_id:-Popya007 and Password: “12345”, when he wants to change the password.

4) The user should have already answered the security questions.

(a) Which is your favorite movie?

(b) Which is your favorite soccer team?

(c) Which is your favorite computer game?

Answers

Border

Arsenal

Age of Empire

Description: First case is Change Password when the security question arises.

When Popya who is the authenticated employee wants to change the password which is "12345". After Popya has successfully logged in to the system using user_id:-Popya007 and Password: "12345"and feels the need to change the password "12345" of his account.

Trigger:

Popya clicks the "Profile" menu item provided in the PM system user interface of homepage page.

The system responds by …

1) The Popya clicks on the "Profile" menu and selects the "change password" option from the dropdown menu of the homepage of PM system.

2) The system brings up the security question which have already been answered by Popya

(a) Which is your favorite movie?

(b) Which is your favorite soccer team?

(c) Which is your favorite computer game

Popya answers the security questions

(d) Border

(e) Arsenal

(f) Age of Empires

Relevant requirements: None

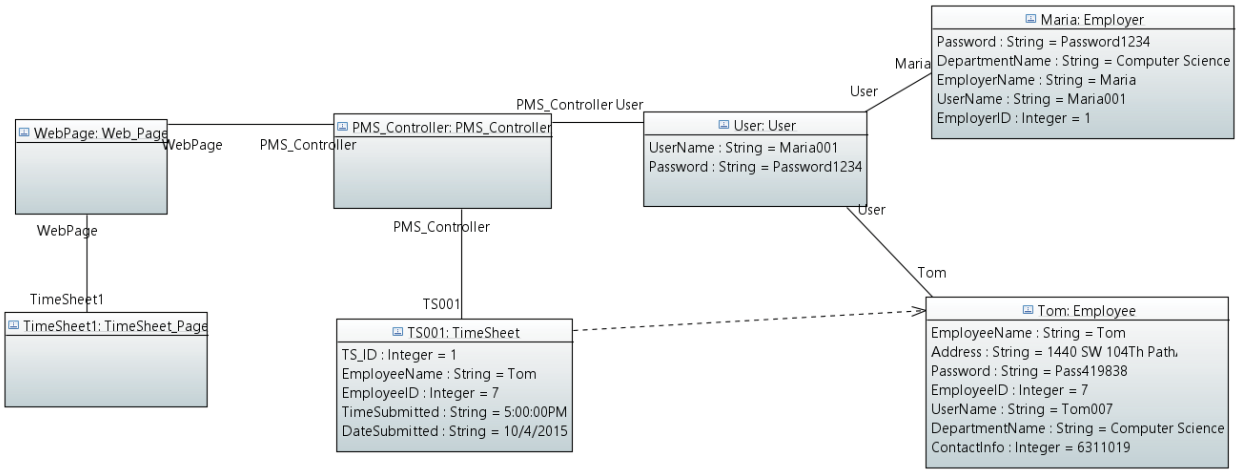
Post-conditions: The PM system will recognise Popya and in the change password case will take Popya to the window where Popya can change his password "12345" to new password "419838" and in the forget password case PM system will generated an automated email with new password.

Email:- hello Popya,

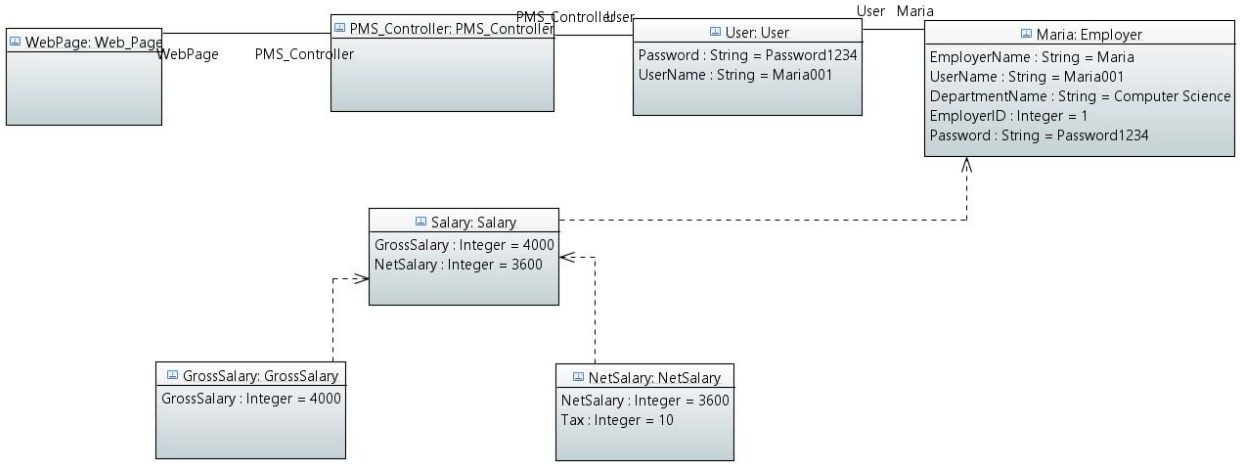
Your new password is "419838".

4.3.2 Object Diagrams:

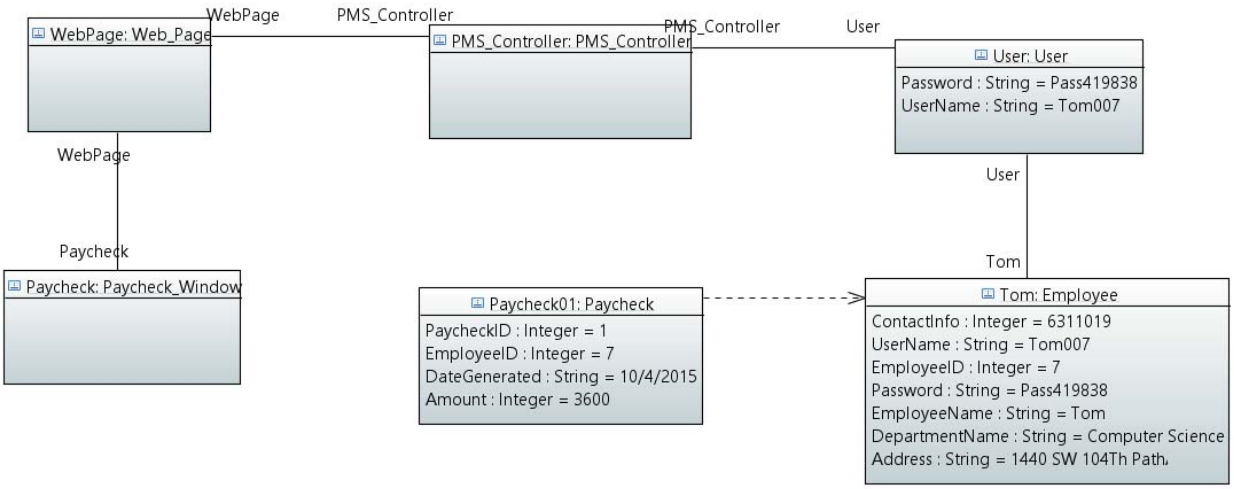
Object Diagram 1:



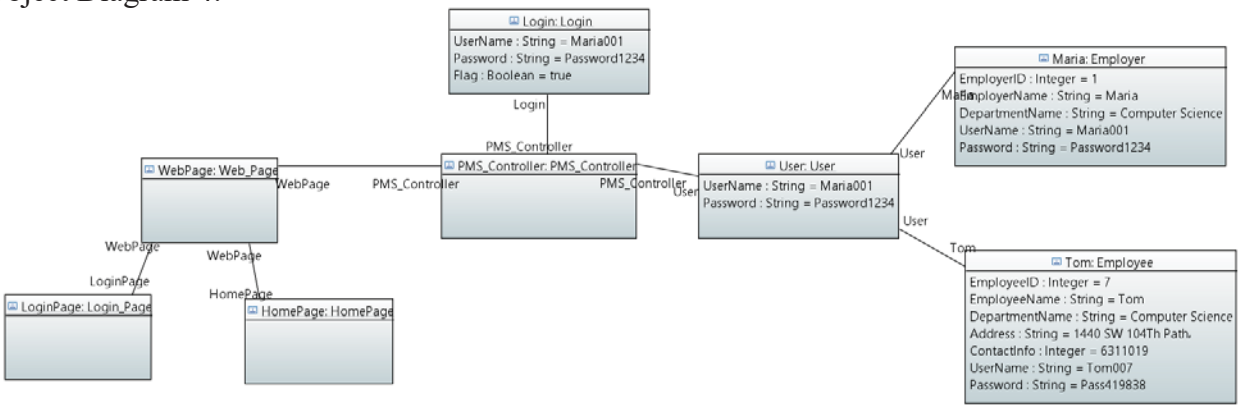
Object Diagram 2:



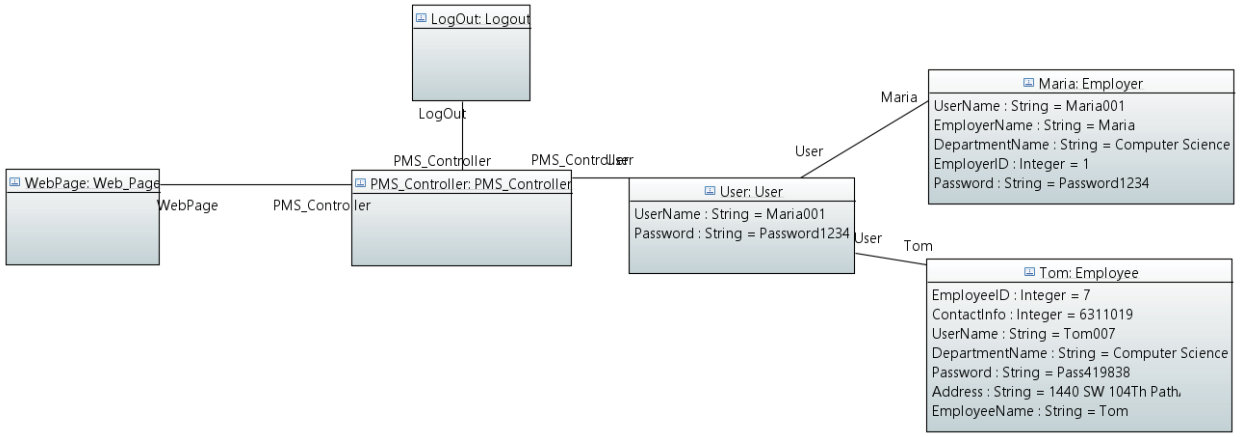
Object Diagram 3:



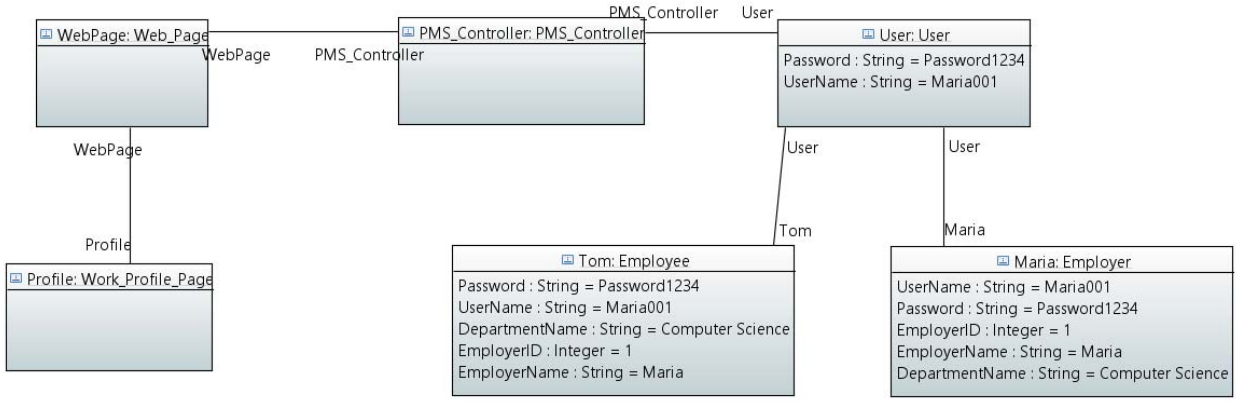
Object Diagram 4:



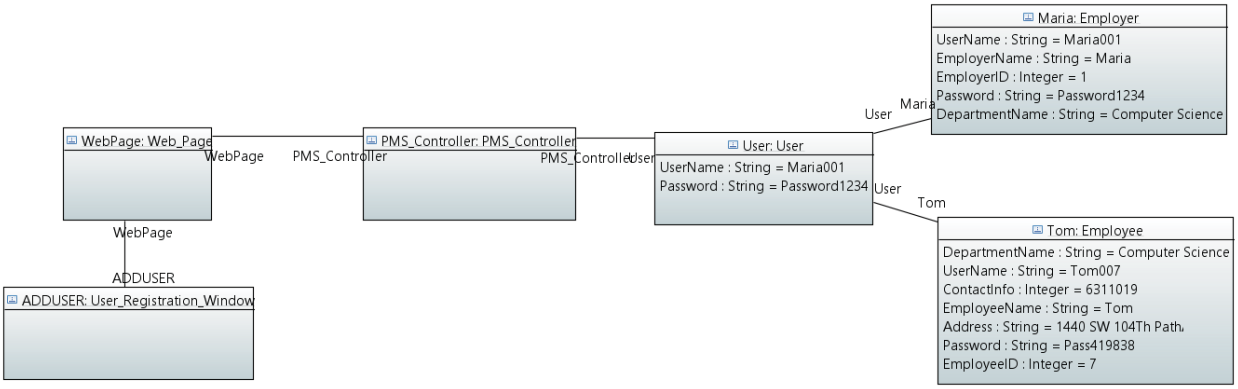
Object Diagram 5:



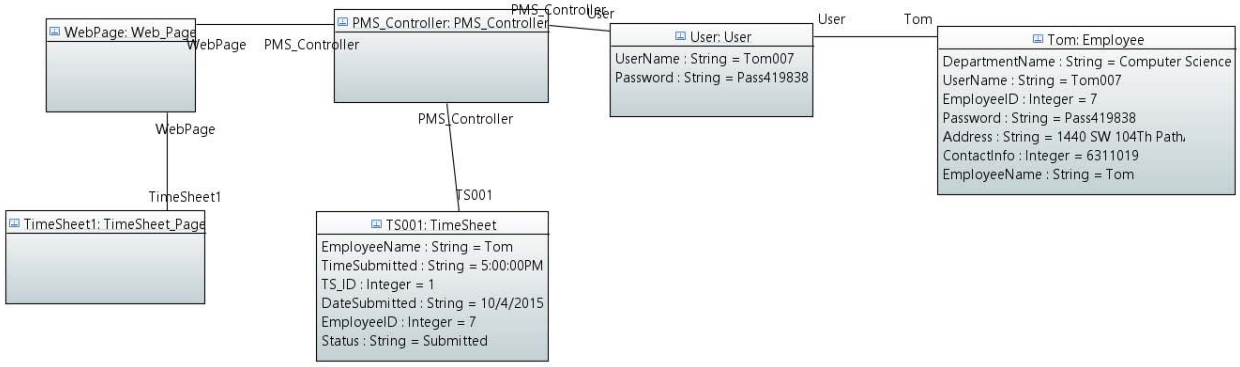
Object Diagram 6:



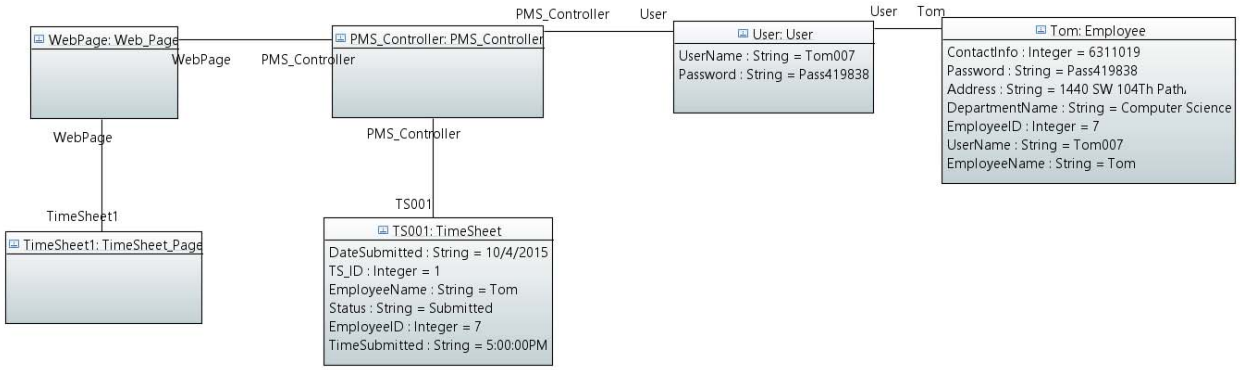
Object Diagram 7:



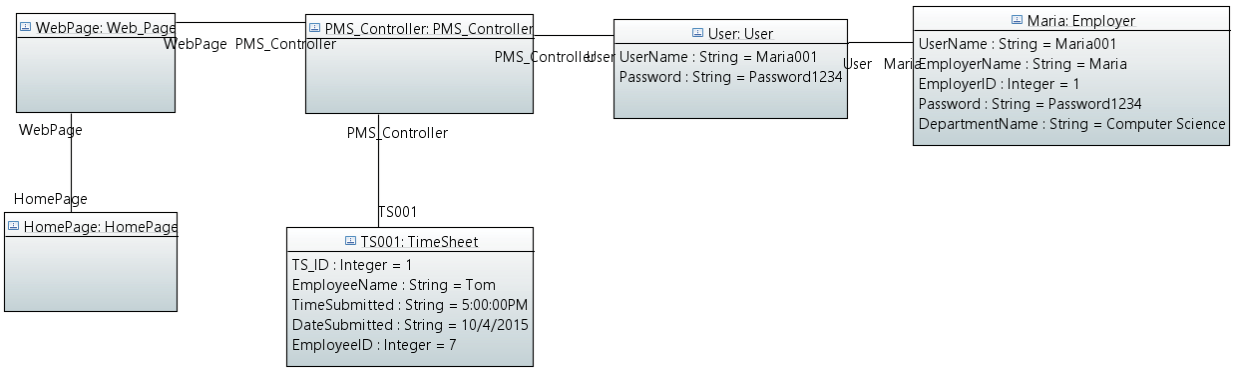
Object Diagram 8:



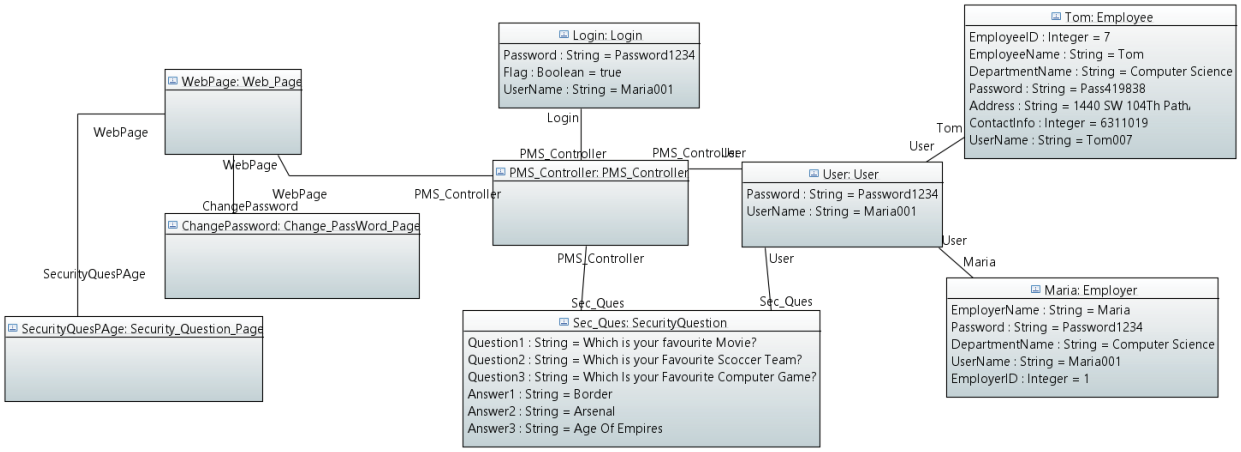
Object Diagram 9:



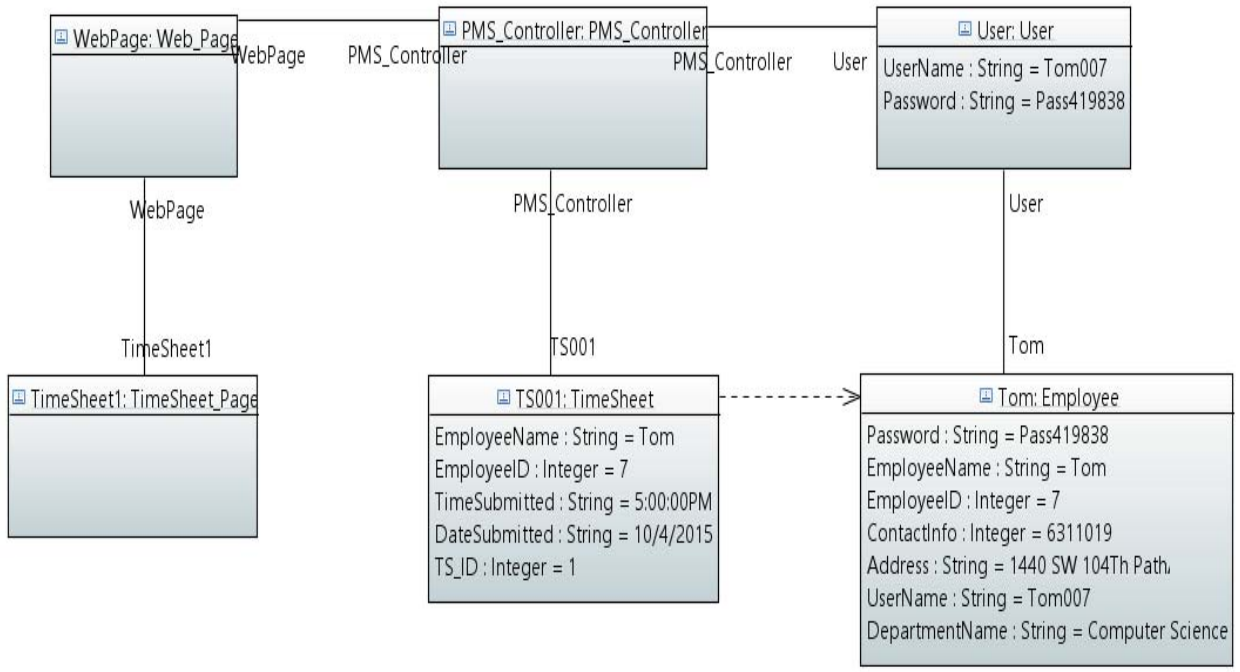
Object Diagram 10:



Object Diagram 11:



Object Diagram 12:



4.3.3 Class Diagram:

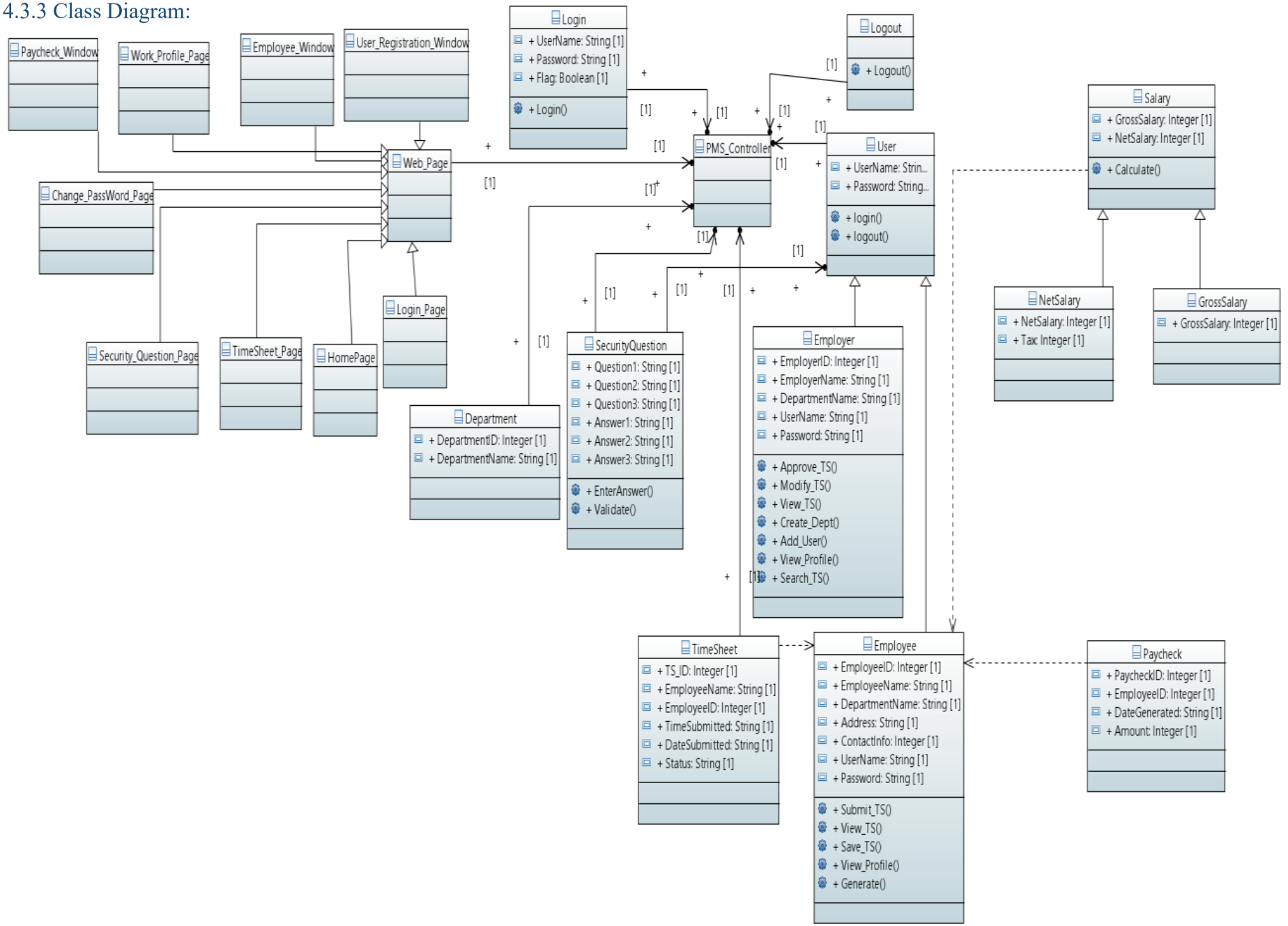


Fig 4.3.3.1: Class Diagram

4.3.4 Sequence Diagram

Approve Timesheet Sequence Diagram:

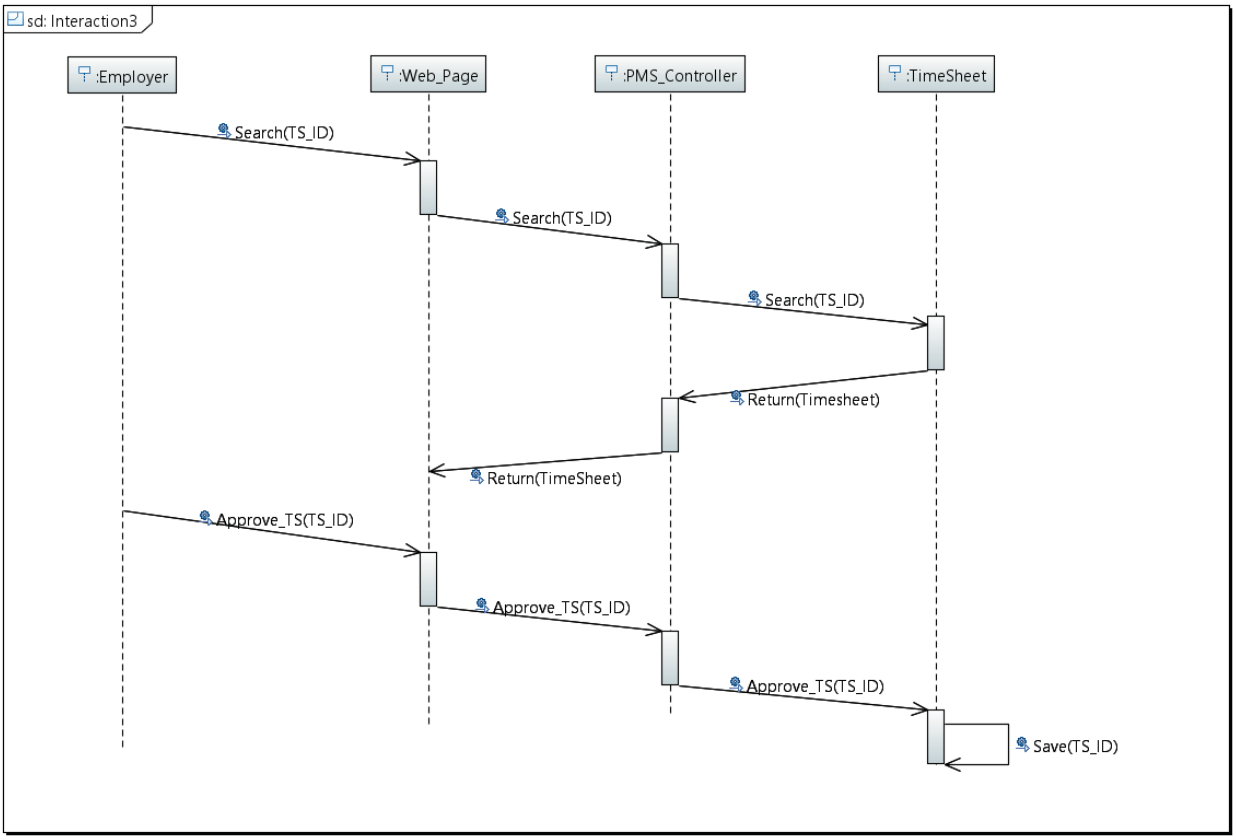


Fig 4.3.4.1: Approve Timesheet Sequence Diagram

2) Calculate Salary Sequence Diagram:

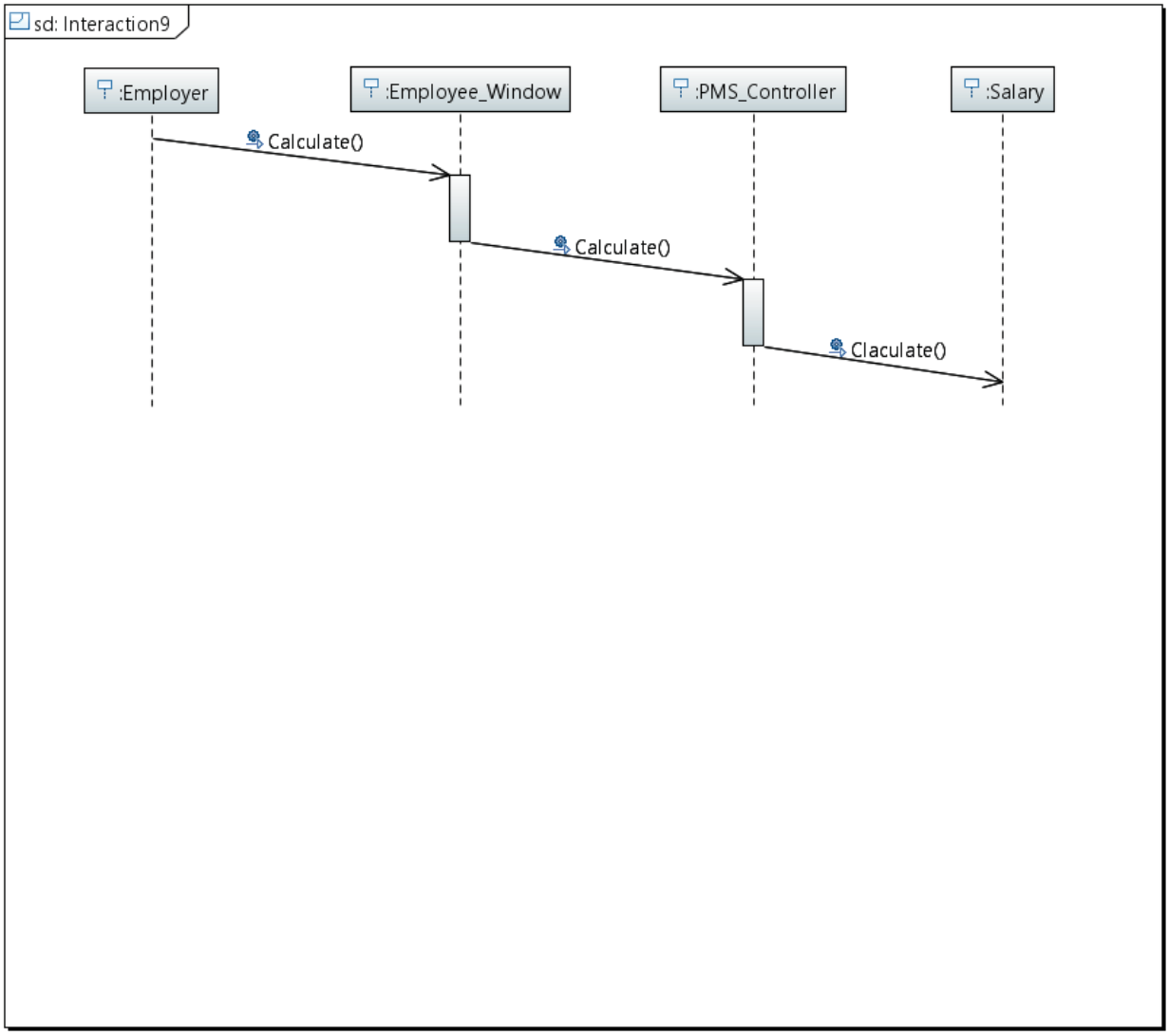


Fig 4.3.4.2: Calculate Salary Sequence Diagram.

3) Generate Paycheck Sequence Diagram.

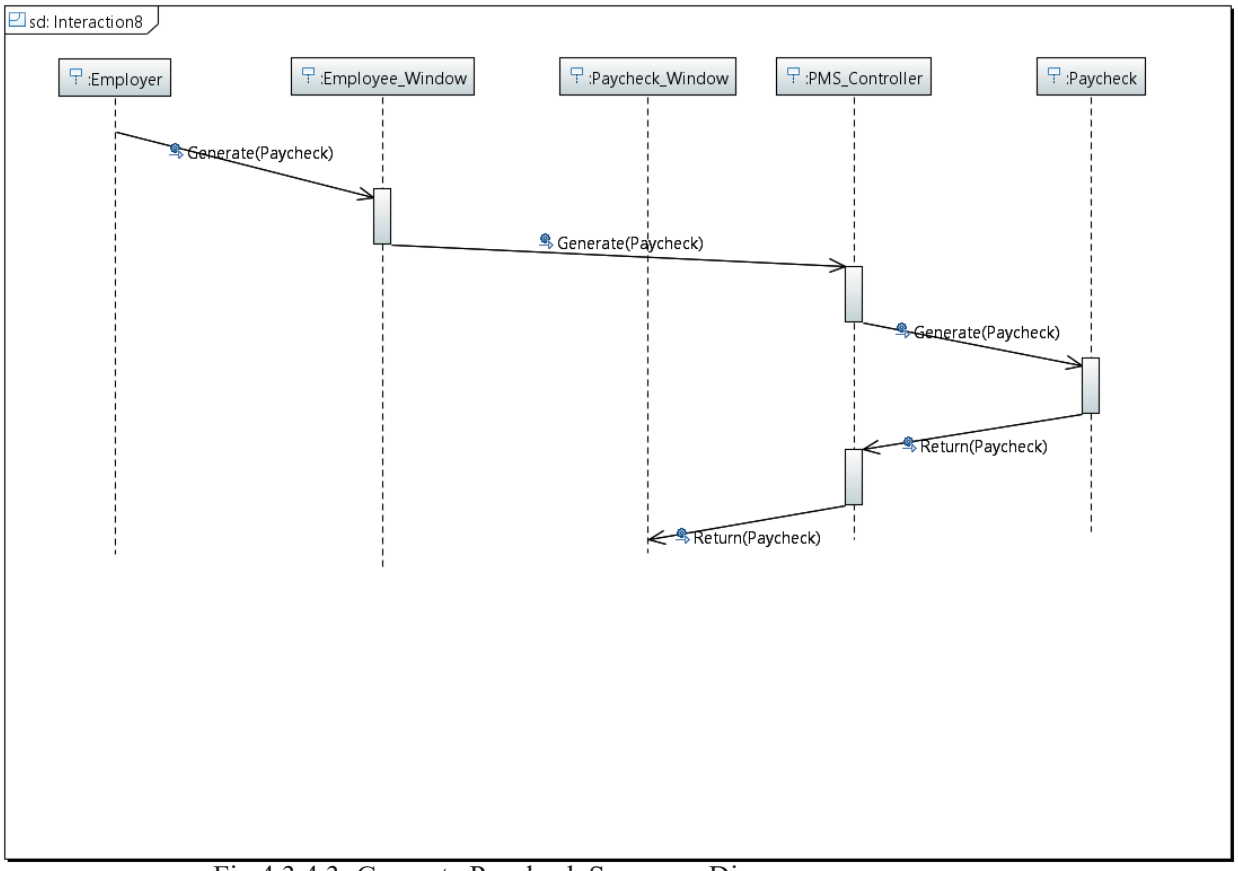


Fig 4.3.4.3: Generate Paycheck Sequence Diagram.

4) Login Sequence Diagram.

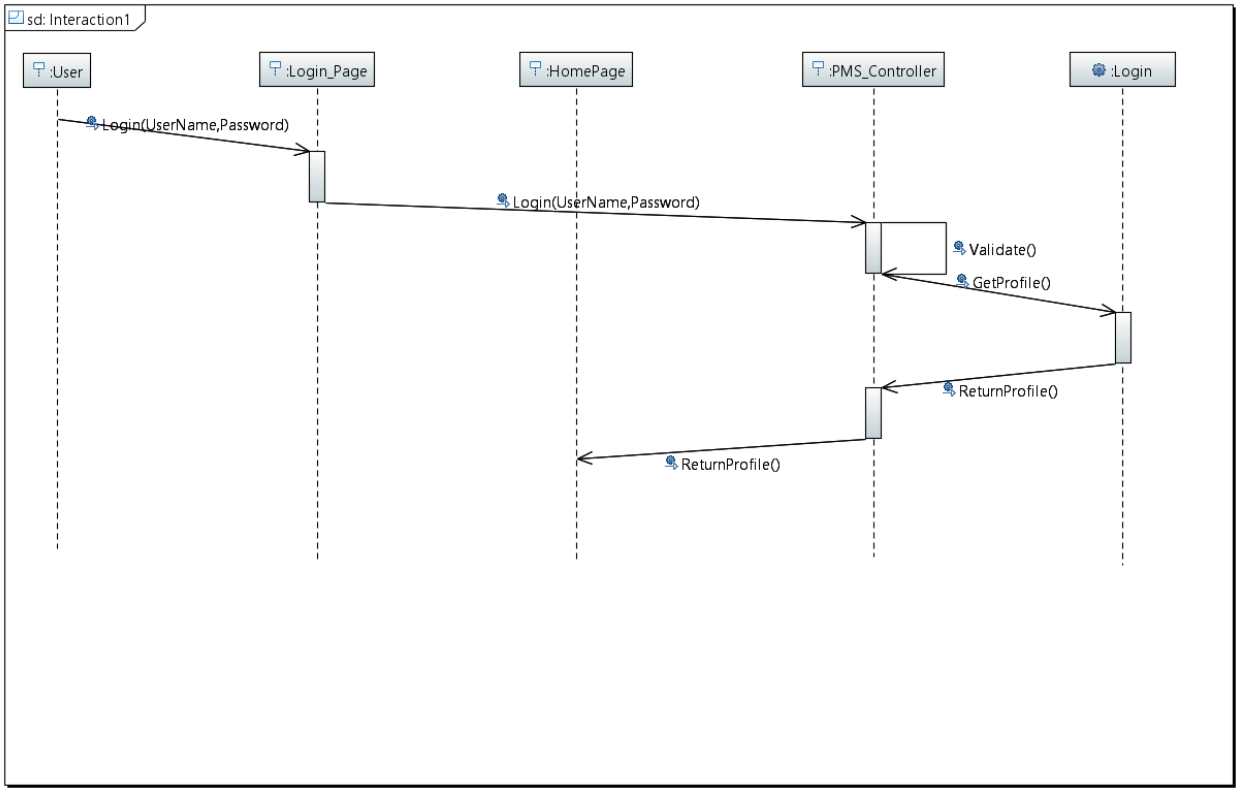


Fig 4.3.4.4: Login Sequence Diagram.

5) Logout Sequence Diagram:

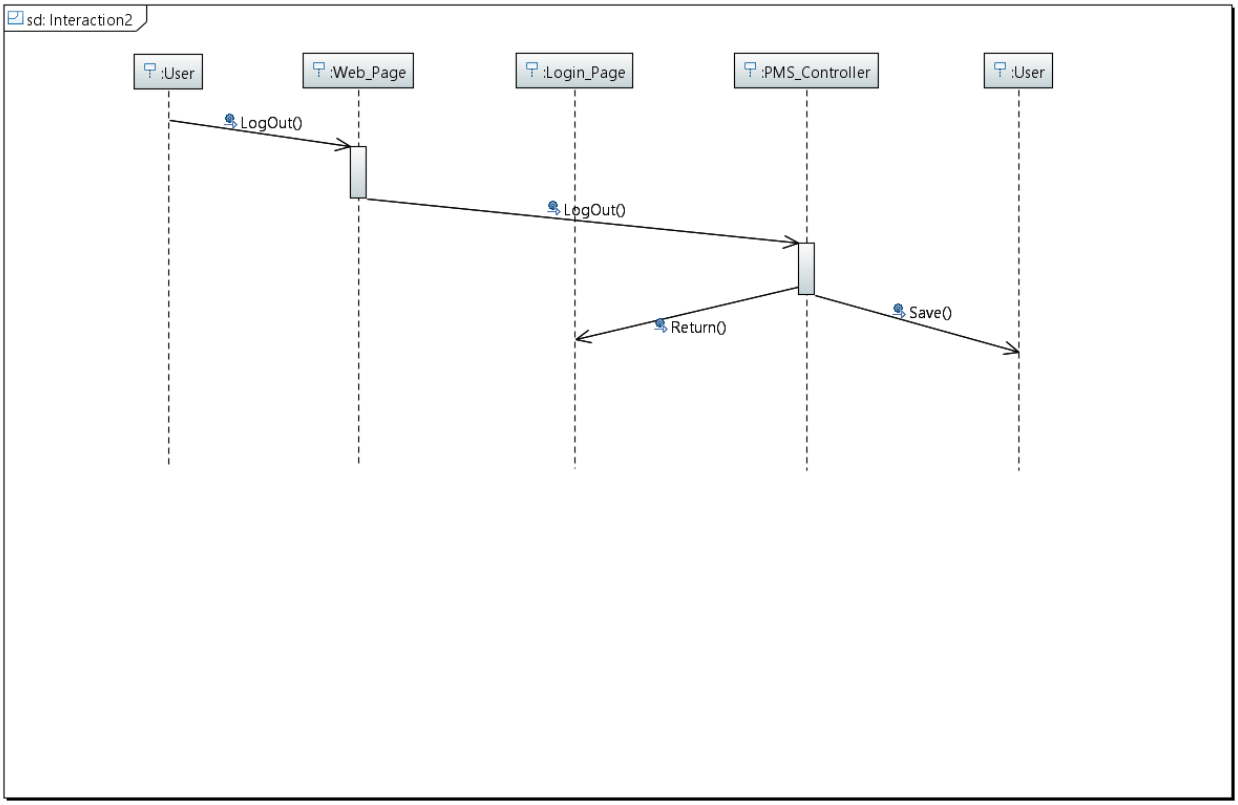


Fig 4.3.4.5: Logout Sequence Diagram

6) View User Profile Sequence Diagram:

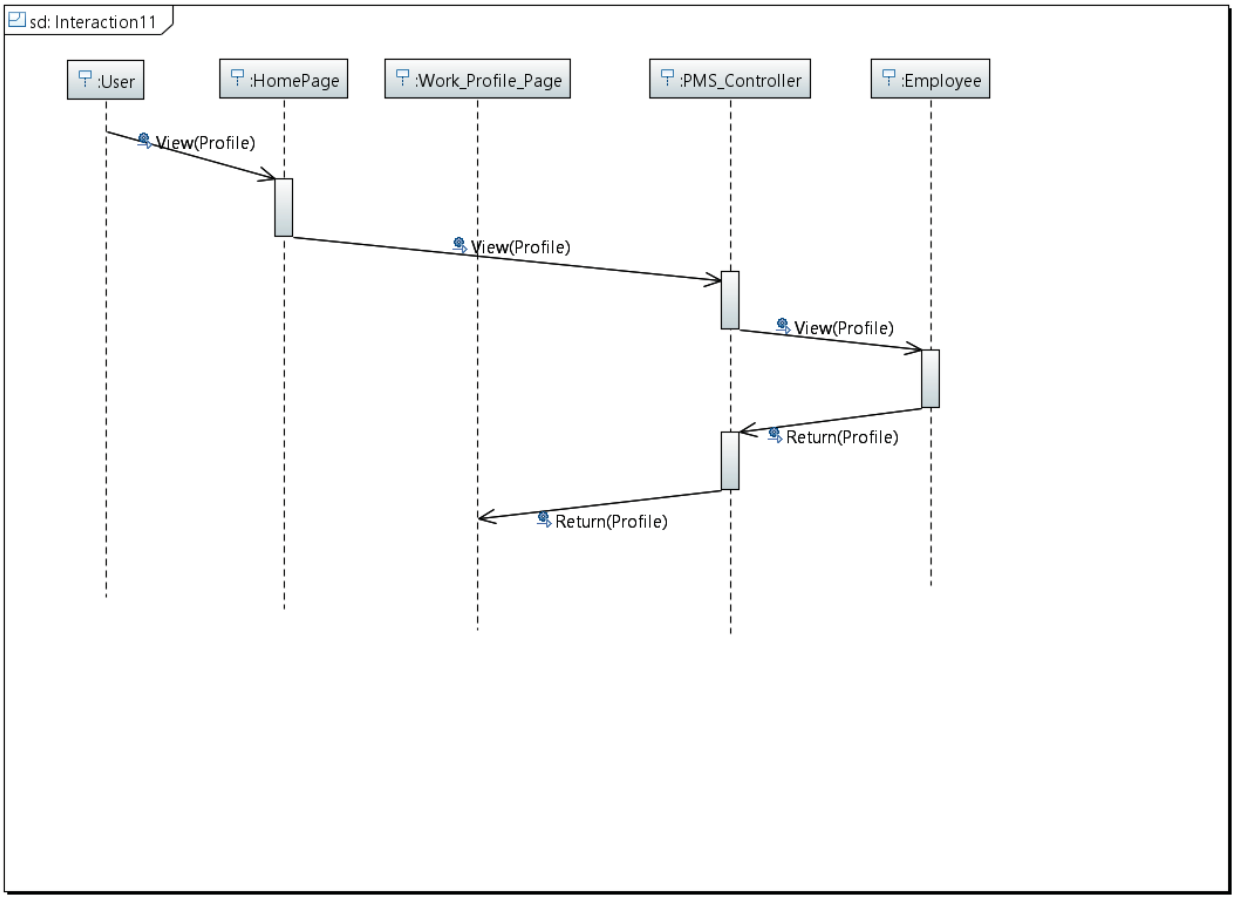


Fig 4.3.4.6: View User Profile Sequence Diagram.

7) Add User Sequence Diagram:

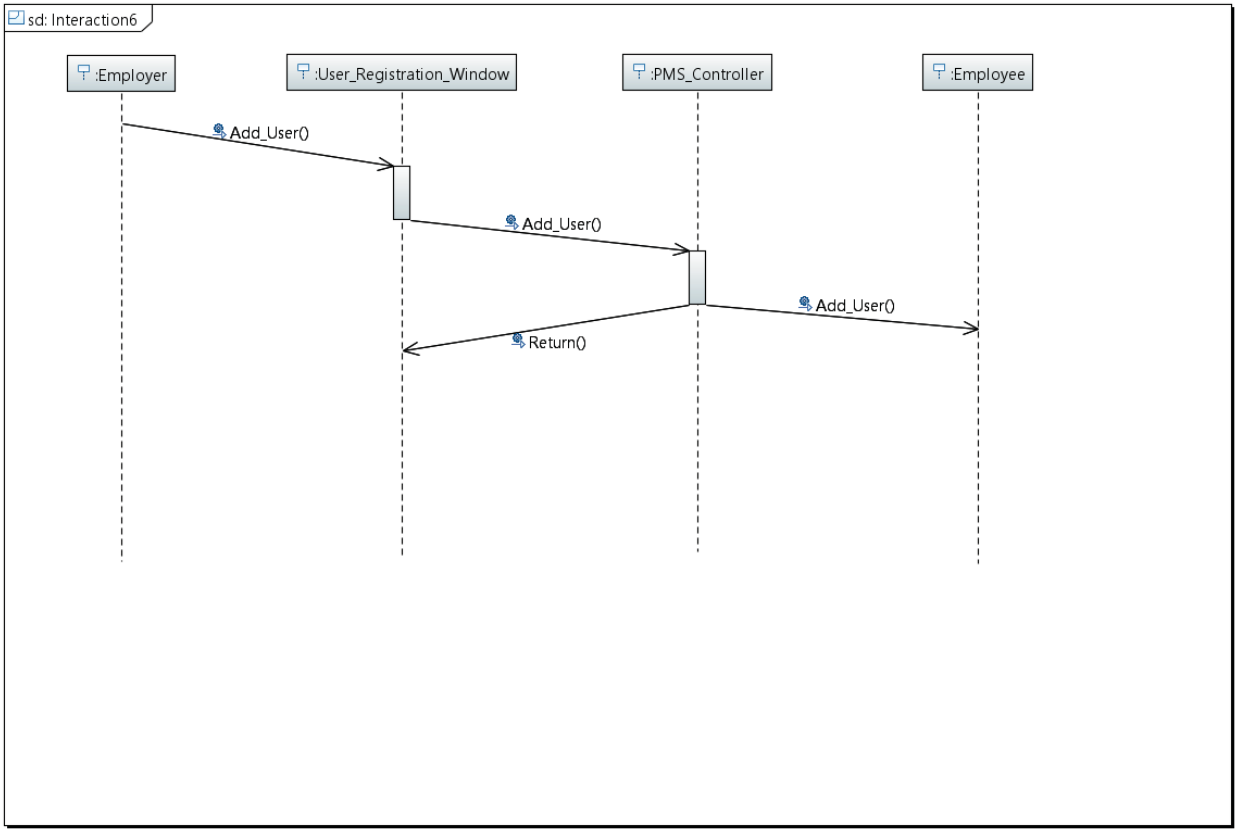


Fig 4.3.4.7: Add User Sequence Diagram.

8) Duplicate Submit Timesheet Sequence Diagram:

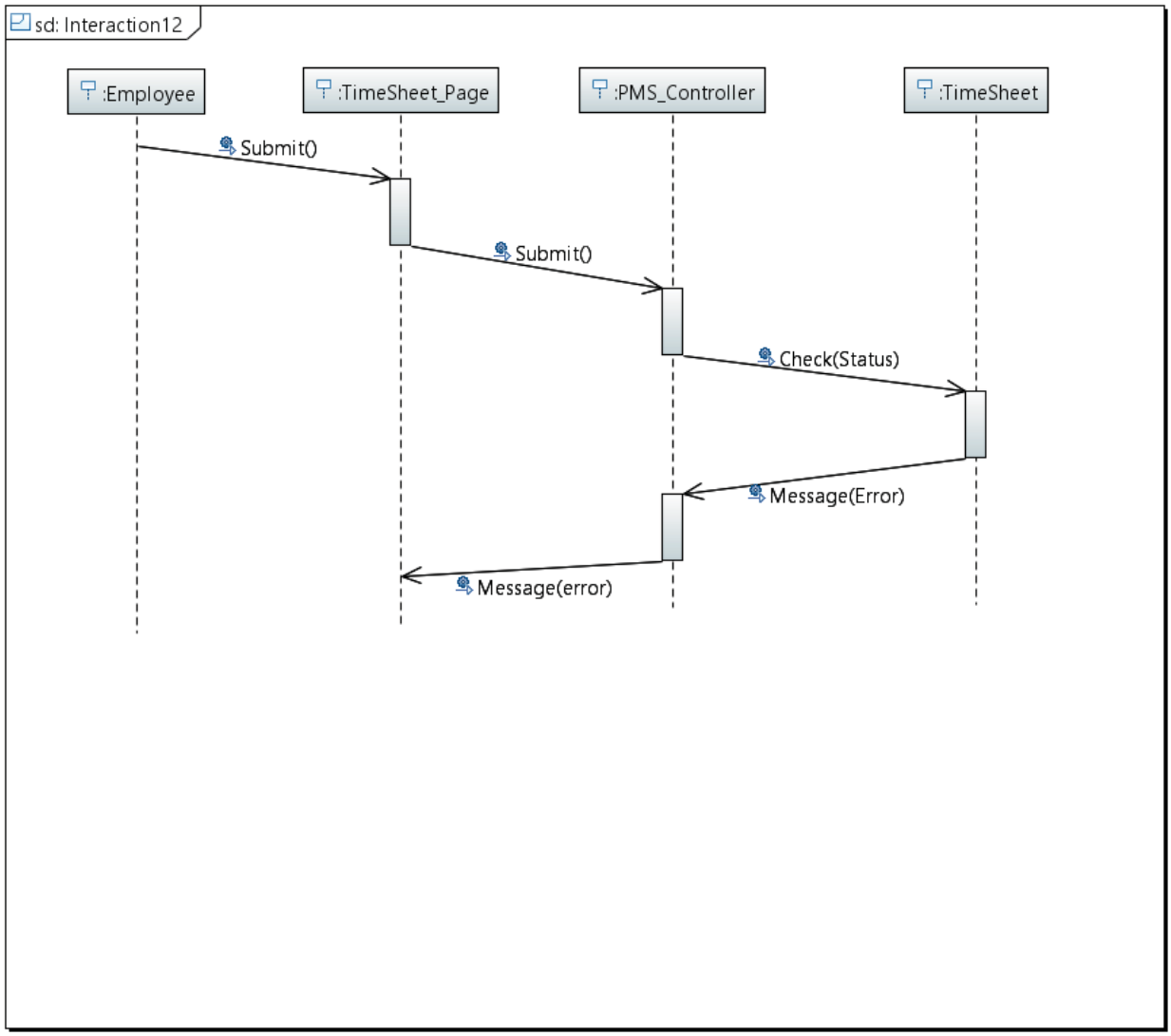


Fig 4.3.4.8: Duplicate Submit Timesheet Sequence Diagram.

9) Save Timesheet Sequence Diagram:

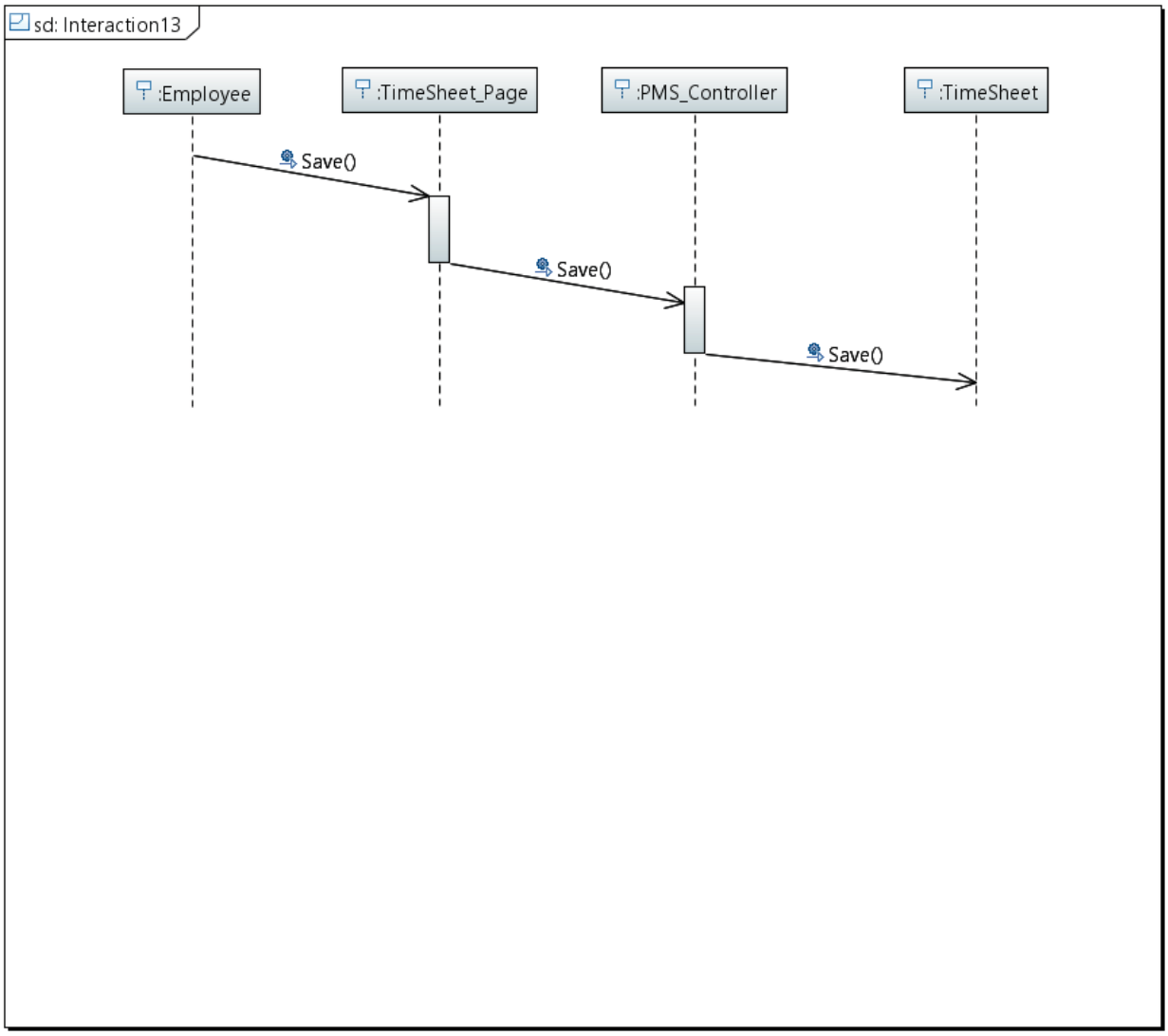


Fig 4.3.4.9: Save Timesheet Sequence Diagram.

10) Search Timesheet Sequence Diagram:

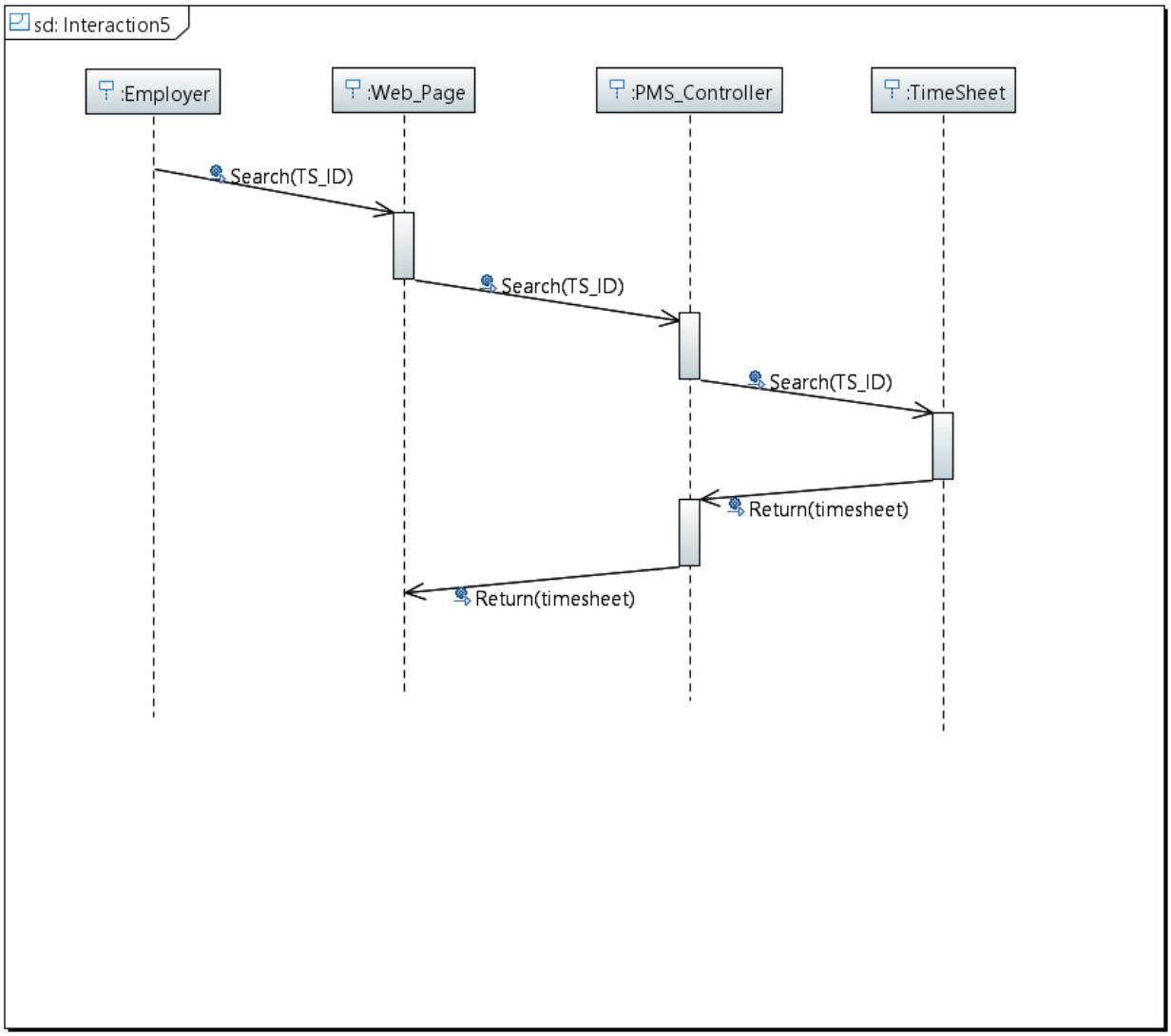


Fig 4.3.4.10: Search Timesheet Sequence Diagram.

11) Security Question Sequence Diagram:

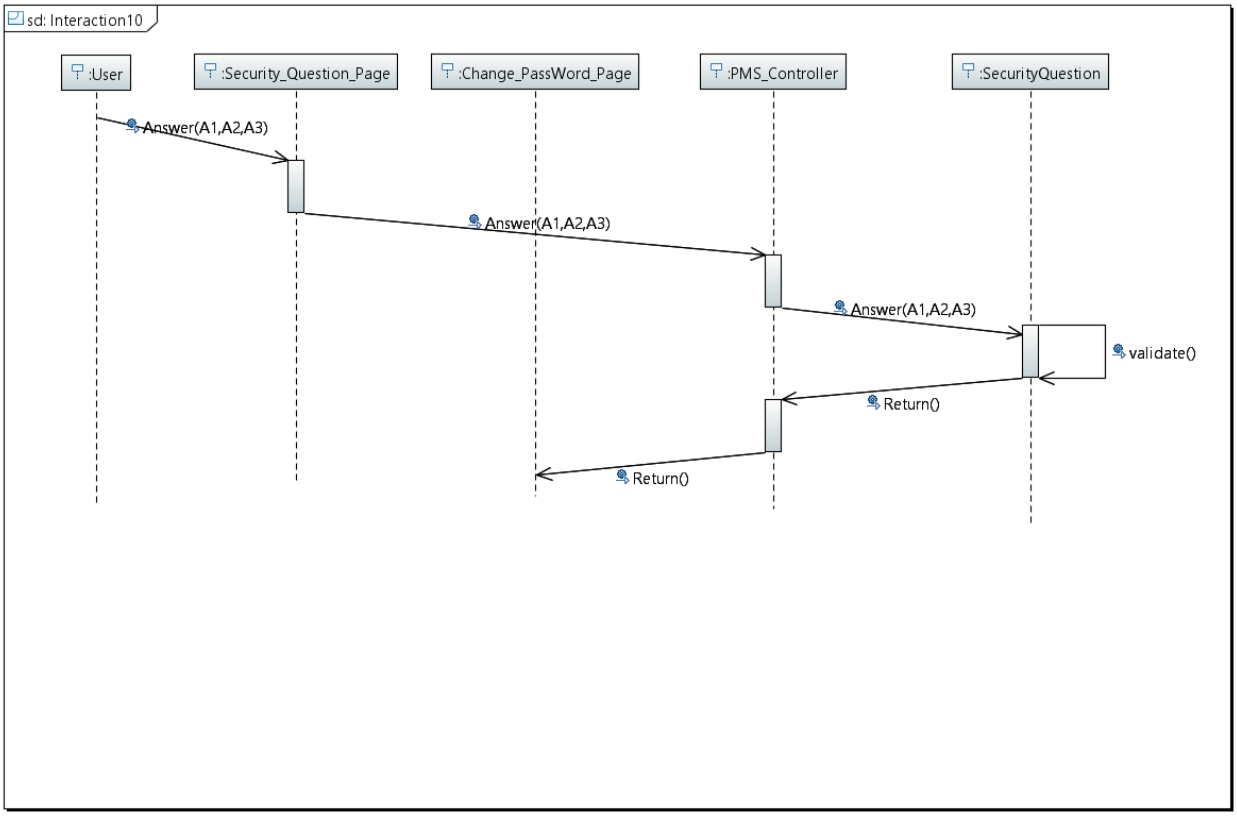


Fig 4.3.4.11: Security Question Sequence Diagram.

12) Submit Timesheet Sequence Diagram.

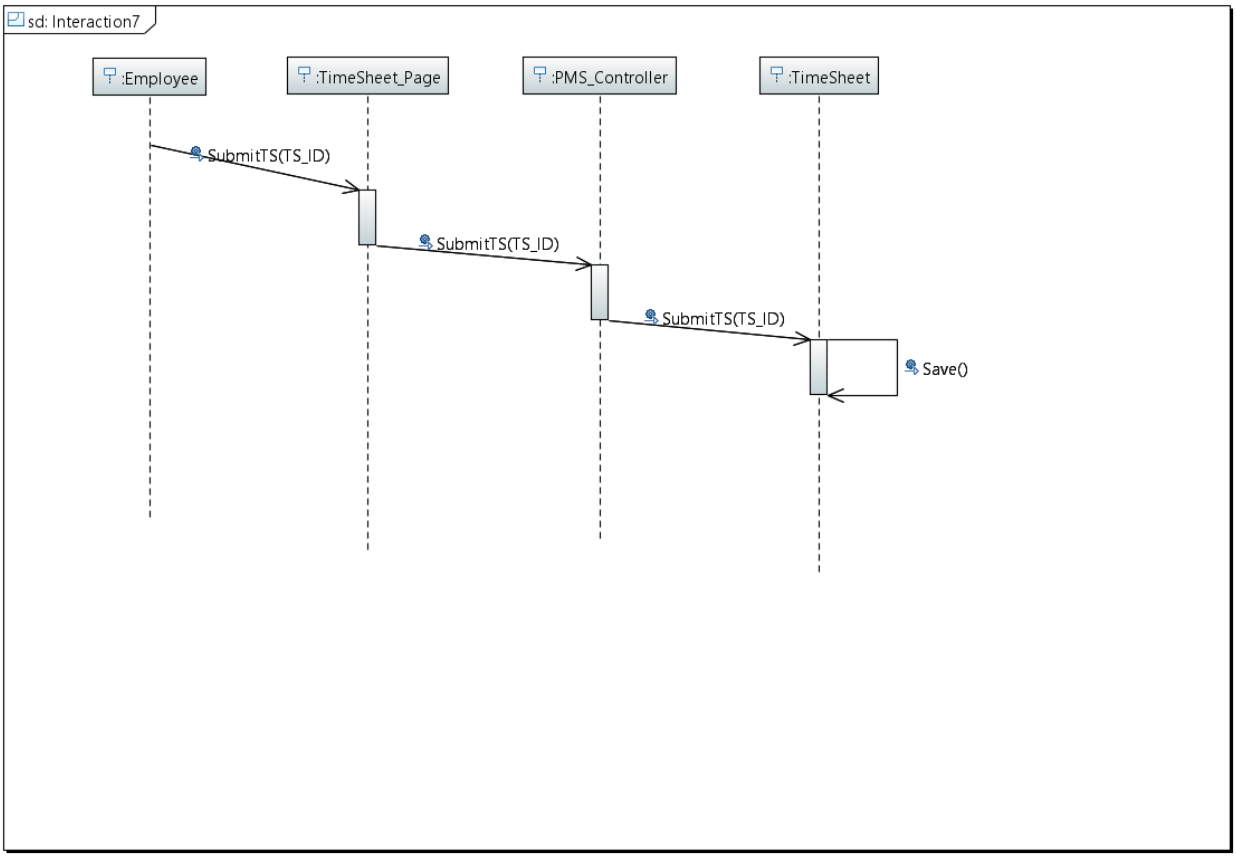


Fig 4.3.4.12: Submit Timesheet Sequence Diagram.

5. Proposed Software Architecture.

In this chapter we are presenting the architecture patterns that are being used in Payroll Management System. A pattern is a solution to recurring design problem. There are 3 types of patterns and they are architectural, design and idiom. Each pattern has its own importance. To express a structural schema for software systems we use architectural patterns, design pattern is used when we need to provide a scheme for refining the subsystems of a software system and idiom is a low-level pattern which is related to programming language. UML has many features in it, such as creating a profile. The profile creation helps in providing a generic extension mechanism for customizing UML diagrams for specific platforms and domains.

5.1 Overview of Software Architecture.

Payroll Management System is built using 3-tier architecture and Model View Controller (MVC) architecture, where 3-tire is primary architecture and MVC is secondary architecture. Payroll Management system is a web based application which includes 3 different parts i.e. client, server and database which are geographically separated. Since there are different components and are geographically separated Payroll Management System agrees with the 3-tier architecture.

In addition to this, we are implementing MVC architecture in business layer. MVC is our secondary architectural pattern which we are implementing in Payroll Management System. The reason we are using MVC architecture is that this architecture clearly lays out the different functionalities of our application i.e. model, view and controller. The view component is responsible for displaying all the interfaces. The controller component is composed of all the controllers such as Timesheet controller, authentication controller etc. in order to run the functionalities. The model component represents only data and nothing else. The main reason behind selecting MVC architecture is that it separates the view logic from business logic. It also enables to work simultaneously between developers and others who are responsible for different components such as UI layers and core logic. Both the properties make the MVC an excellent choice for the business logic that is embedded in 3-tier architecture of Payroll Management System. The primary use of 3-tier architecture us that the performance is calculated in terms of response time for any web application. So using 3-tier architecture helps in checking whether the non-functional (Appendix B) requirements are calculated accurately.

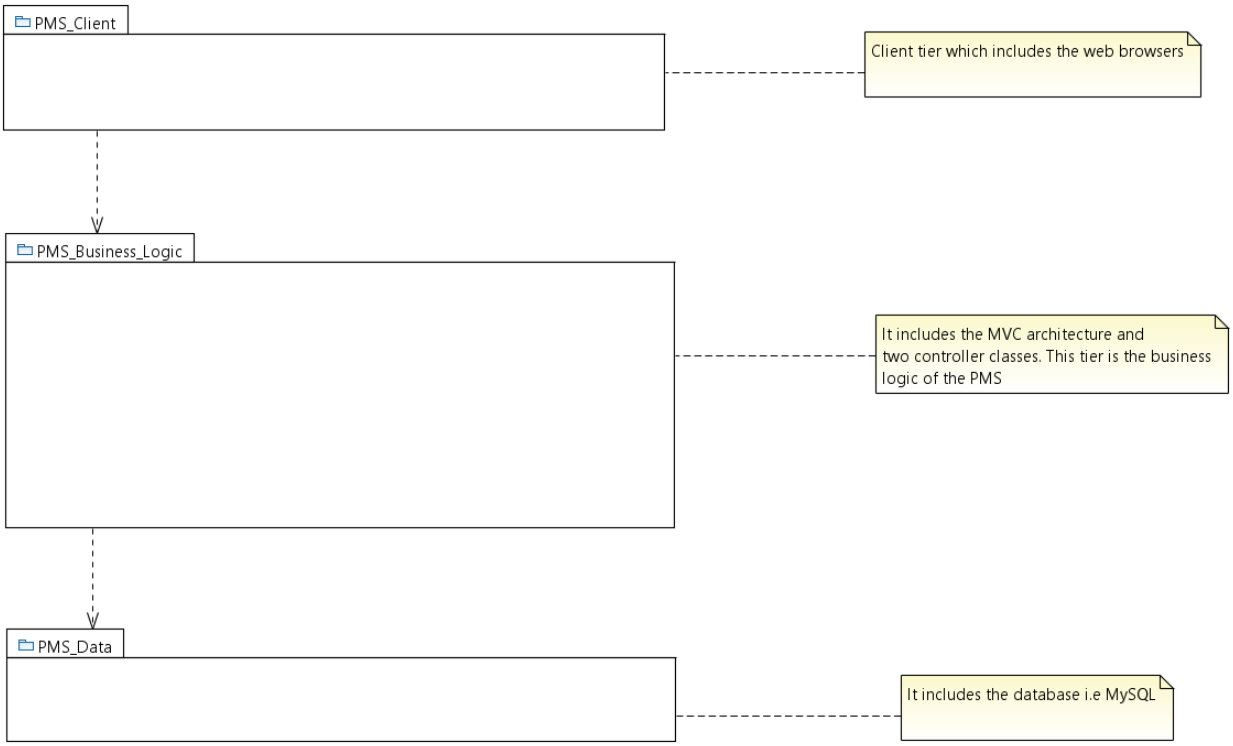


Figure 5.1.1: 3-Tier Architectural Pattern.

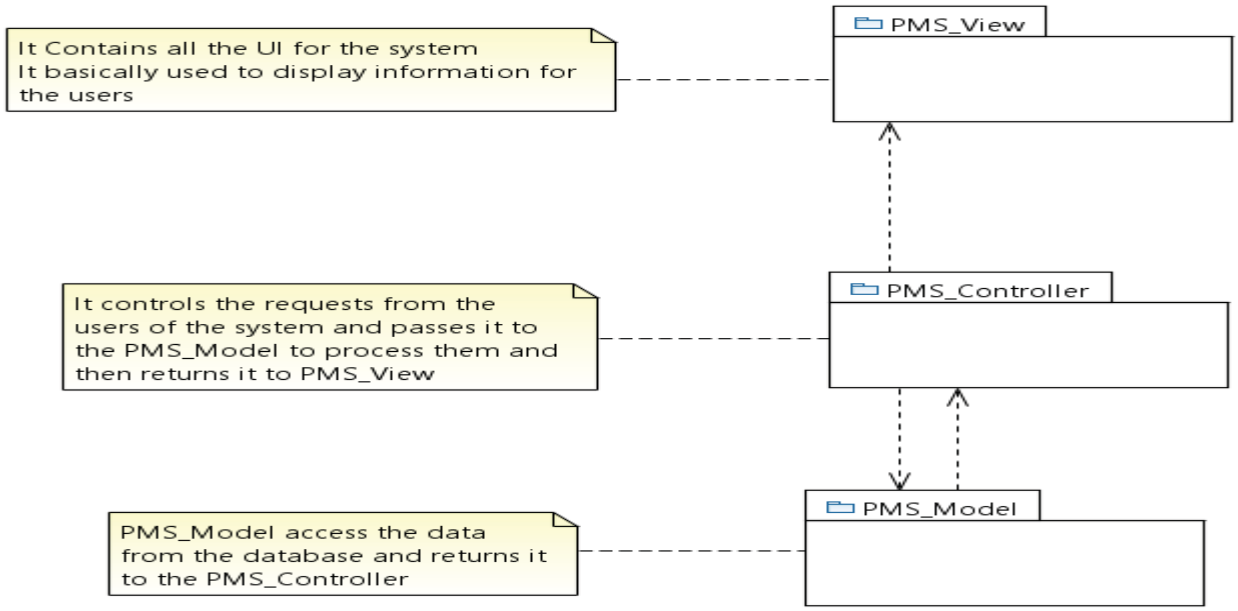


Figure 5.1.2: MVC Architectural Pattern.

5.2 Subsystem Decomposition:

In this section, we are going to discuss the sub-systems of Payroll Management System. Collection of classes, operations and relations is called as a Sub-System. The 3 different Sub-systems of Payroll Management System are Client, Business Logic and Database.

PMS_Client: The client sub-system is composed of packages which are related to View package. It consists of different classes that an employee can see views such as PMS_01_Login, PMS_08_SaveTS, PMS_10_DirectDeposit and PMS_16_UpdateEmployee. The view model package composes of the employee controller which are responsible for control flow of the activities on the client side.

PMS_Business_Logic: this is the main sub-system which is the component of program that is responsible for encoding the business rules which helps in creation, display, storage of data. In this sub-system, it determines how data is calculated. This sub-system basically determines the work flows of the application. It consists of MVC architecture and two controller classes. This sub-system shows the implemented logic of PMS. Coming to the MVC architecture, it consist of the user interface which displays the information for the employees. It also has the controller which is responsible for managing the requests from the employee's. Here in this sub-system, the Model controller processes the requests and then returns to the View controller. Then the Model controller access the data from the database and returns it to the controller of PMS. Some of the use cases related to the sub-system are PMS_07_CalcSal, PMS_04_ModifyTS and PMS_05_ApproveTS.

PMS_Data: all the data which is available on the application is stored on the data base sub-system. The attributes of employee/ employer is stored on the database. Whenever the employer wants to pull out the data regarding an employee, the employer can do it by using the employee ID and the information is pulled from the database. The use cases related with data base sub-system are PMS_03_SearchEmp_TS, PMS_01_Login, PMS_09_PayCheck, and PMS_17_WorkProfile. Even the for the security mechanism i.e. PMS_005_security, this sub-system is responsible. For checking the answers of the security questions, the data is stored on the database.

5.3 Hardware and Software Mapping.

The deployment diagram describes the physical deployment of information generated by the software program on hardware components. The information that is generated by the software is called an artifact. The hardware where the software is deployed is called a node. The deployment diagram for PMS system, there is a web server that runs the application, a database server that the data stores, and the client machine to view the website. For the client side, a web browser like Google Chrome is needed, for the app server side, a server like Apache Tomcat is needed, and for the data server side, a SQL server like MySQL is needed.

The mapping of hardware and software for Payroll Management System is described in the later part. As discussed earlier, there are 3 sub-systems. Client sub-system is deployed in client device i.e. desktop or PC or laptop. The database of Payroll Management System is MySQL. It has different properties such as it is highly secured, scalable and it is very to use. Apart from these, it supports different development interfaces such as JDBC, ODBC. The Application server consists of the application sub-system.

Hardware for Client sub-system:

Client	Desktop or PC
Processor	Intel Celeron CPU 1000M 1.8 GHz
RAM	4.00 GB
System Type	64-bit, x64-based processor

Software for client sub-system:

Operating system	Windows XP, Windows 7, Windows 8, Windows 8.1, Windows 10
Browser	Google Chrome, Mozilla Firefox, Internet Explorer

Hardware for Application server:

Client	Desktop or PC
Processor	Intel i5 6200U CPU 2.3 GHz
RAM	4.00 GB
System Type	64-bit, x64-based processor

Software for Application server:

Operating system	Windows XP, Windows 7, Windows 8, Windows 8.1, Windows 10
Server	Apache Tomcat 7, PMS

Hardware for Database:

Client	Desktop or PC
Processor	Intel i5 6200U CPU 2.3 GHz
RAM	4.00 GB
System Type	64-bit, x64-based processor

Software for Database:

Server	MySQL

Transmission speed required for PMS is 1Gbps.

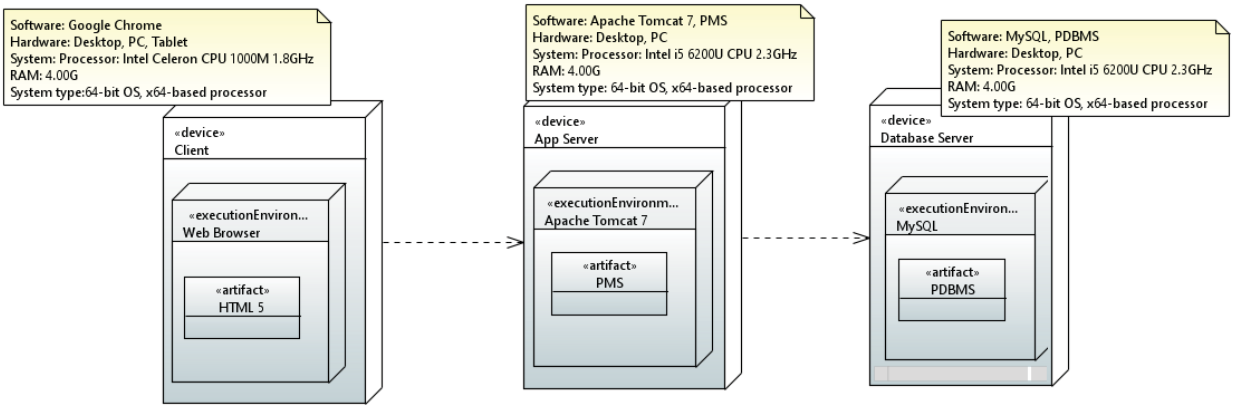


Figure 5.3.1: Deployment Diagram

5.4 Persistent Data Management.

The data is said to be persistent if it exists or survives from session to session. Generally, the persistent data is stored in a database. The database might be a disk or a tape. To achieve the persistence of data, the database should be transparent to the application developer. Payroll Management System is a simple application which involves some data transformation. There are certain pieces of data that has to be stored in to the database such as profile information of an employee, the working hours of the employee and the history of the timesheets that have been approved by the employer.

We are using the ER diagram to represent the structure of data used in the entity objects. Some of the data includes the login information of the employee and the employer, age of employee, the department in which the employee is working. Everything related to the employee is being stored on to the database. All the attributes of employee that are being stored is displayed in the below ER diagram. To show the database of Payroll Management System, ER diagram is given below. ER diagram is a model which describes the data. This is typically implemented as a database. Figure 2.4.1shows the data which is stored on the database.

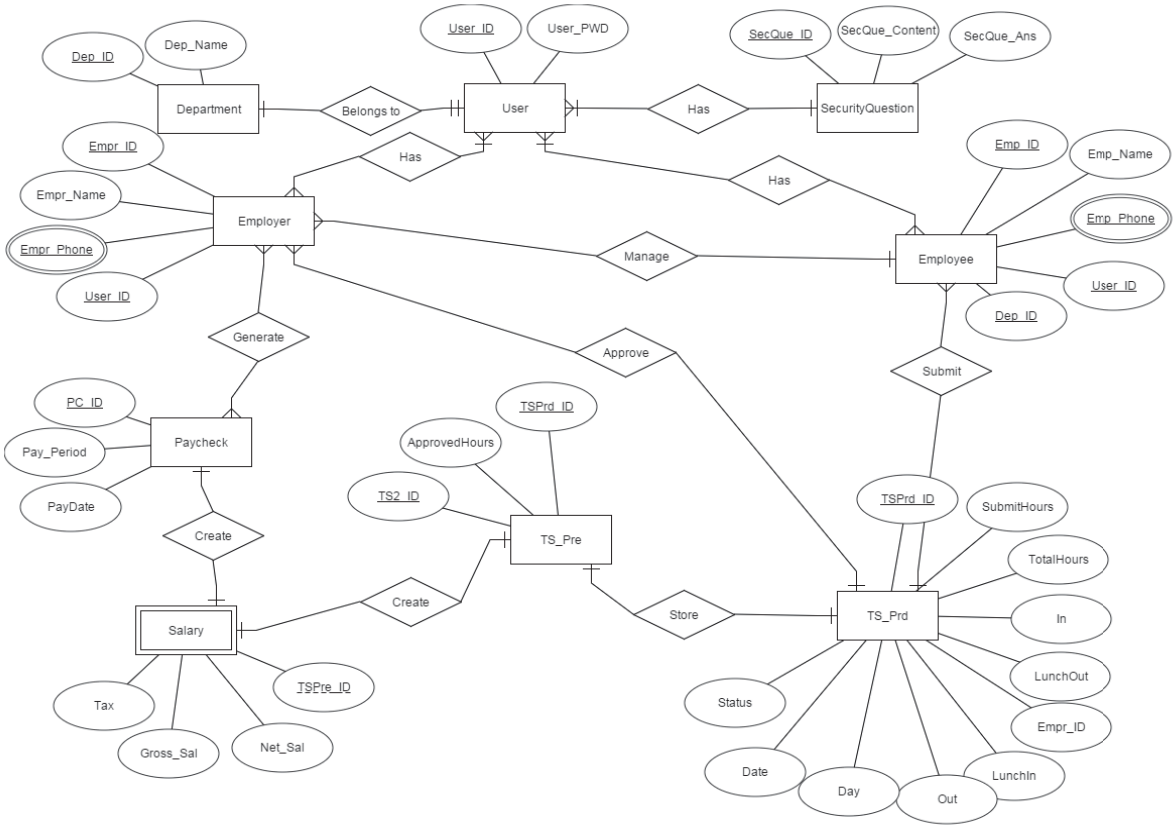


Figure 5.4.1: ER Diagram.

5.5 Security Management.

A software is developed from the initial phase with keeping security aspects in the mind. Having this, the software can tolerate, resist and recover from different attacks from the attackers. This would result in efficient performance of the software. Security management helps in managing the risks to the software against the security threats. When we apply a security mechanism to a piece of software, it protects from threats. There are security services being implemented in Payroll Management System. They are as follows:

1. **Security Question:** Security questions provide high level of authentication. This security mechanism is triggered when the employee or employer tries to change the password. When the employee/employer tries to change the password Payroll Management System arises the window which consists of 3 different questions. If the employee/employer correctly fills the answers for the given questions, then the application shall give the access to change the password. If any of the given answers are wrong, then PMS shall not allow the employee/employer to change the password. This security mechanism is very useful when an attacker tries to change the password and gain unauthorized access to the employee/employer account.

Figure 2.5.1: Security Question.

2. **No Duplicate Submission**: The main theme of Payroll Management System is the timesheet. It plays the central role on which everything depends. After the employee fills the timesheet, when he/she clicks the 'submit' button the tabular form of timings shall change to a label form. To be clear, the employee can modify the timesheet until he/she click on 'submit' button. But, if the submit button is selected, then the timesheet cannot be modified or the employee loses the access to modify the timesheet. This prevents the employee from making the duplicate timesheets. It also prevents from other third parties in such a way that a submitted timesheet cannot be modified. So, the third party has no access to change the timesheet.

The above two security mechanisms are implemented in the Payroll Management System which helps the application to withstand the threats from the attackers. These two mechanisms guarantee high level authentication to the Payroll Management system.

5.5

Use case	**Employee**	**Employer**
PMS_01_login	Yes	Yes
PMS_18_Submit_Timesheet	Yes	No
PMS_13_Add_Employee	No	Yes
PMS_08_SaveTS	Yes	No
PMS_17_Work_Profile	No	Yes
PMS_005_Security	Yes	No
PMS_04_Modify_TS	No	Yes
PMS_05_Approve_TS	No	Yes
PMS_07_CalcSal	No	Yes
PMS_09_PayCheck	Yes	Yes
PMS_004_Duplicate	No	No
PMS_21_Logout	Yes	Yes

6. Detailed Design

This chapter describes the Object design of Payroll Management System. This chapter is composed of minimal class diagram, state machines, sequence diagrams and the OCL. The Minimal class diagram is used to represent the Class diagram of all the sub-systems. There is State machine diagram which shows the main controller of each sub-system. The interaction between the objects is shown with the help of Sequence diagram. The detailed class diagram is explained and the purpose of each class has been mentioned. Coming to the last part of the detailed design i.e. OCL constraints are mentioned for the main controller in the main sub-system.

6.1 Overview of detailed design

ModelController: The ModelController connects the two packages of the PMS system that is the Controller and the Model. When a request is sent by the user from the view, the request is forwarded to the Controller and from the controller to the model controller to reach the Model and then the response is sent back from Model to the ModelController which is then sent to the controller and then the View that is to the user.

View_Controller: View_Controller connects the PMS_View package and the PMS_Controller package. When a request is sent by the user from the View Controller, the request is forwarded to the Controller package and the response from the PMS_Controller package is sent to the View_Controller.

Salary: The salary class is responsible for managing the data like the salary of the employees. This class mainly deals with the information like gross salary, net salary, tax, Approved hours etc.

Timesheet_Previous: The Timesheet_Previous class is responsible for storing the details of the timesheet of the previous week. When a request is called to view the previous timesheet, this class is called.

Timesheet_Period: The Timesheet_Period class is responsible for storing the details of the timesheet of the current week. This class details are forwarded then to calculate salary and generate paycheck.

Department: This class saves the information of the Organizational Departments. This class is called when any information regarding the department is needed.

User: The class User is the parent class which handles information of both the employee and the employer in the PMS. This is the parent class for the Employee and Employer class.

Employee: This employee class is the child class of the user class. This class stores the information of the employee that has been registered to the system. This class also contains personal information like contact number and address.

Employer: The Employer class is the child class of the user class. This class stores the information of the employer that has been registered to the system. This class also contains personal information like contact number and address.

Paycheck: The Paycheck class is responsible for managing the paycheck information. It contains information like pay frequency, net pay, gross salary, pay check ID.

Security_Question: The security question class is responsible for verifying the users. It stores questions and answers of the users so that the user can verify himself and then can log into the system.

Edit_Timesheet_Control: This class is mainly responsible when there are any changes that has to be done to the Timesheet. This class has the operation of updating the details of the timesheet.

Login: This class is responsible for account management. This class has the operation of validating the user's login credentials.

Cal_Hours: This class is responsible for calculating the hours that are entered in the Timesheet. This class has the operation of calculating the hours.

Timesheet_Control: This class is responsible for the operation of the hours entered on the Timesheet, This class has many operations like sending and receiving the timesheet info from other classes and also to save the timesheet as well as submit the timesheet.

Get_Profile: The Get_Profile class is responsible for fetching the user information. This class has the operation of fetching the user's profile information from the database.

Registration: When a new user has to be added to the PMS this class is called. This class has the operation to save new employees information to the database.

Paycheck_Generator: This class is responsible for generating the paychecks. This class has the operation of getting the information required for generating the paycheck and also has the operation of saving each and every paycheck that has been generated.

Authentication: This class is responsible for authenticating the response typed by the user in the security questions class. This class has the operation to validate the answers entered by the employee.

Cal_Sal: Cal_Sal is responsible for the calculations that has to be done for the hours the employee has worked for. This class has the operation of calculating the salary and then saving it.

Security_Questions_Page: This class is used where the user has to change the password and is in fron of the security question page. Here the user will enter the answers which will be sent by the view_controller to the PMS_Controller package.

Employer_Page: This class has the information of the employer like the Employer name his department ID & Name etc. This class has the operation of taking the employer to a new employee registration page.

Employee_Page: The Employee_Page class has the information of the employee like his name, his job title, department ID, department name etc. This class has the functions like to logout, goto security questions page or to view an employee.

Timesheet_Page: This is the timesheet_page class where the information of the timesheet has been saved. Here there are many operations like approve timesheet, edit the details, generate button, calculate button, submit button, save button.

Paycheck_Page: This class has the information about the paycheck. This class contains information like paycheck ID, tax, pay frequency, pay period etc.

Salary_Page: This class deals with the information about Salary. This class deals with the information like Tax, rate per hour, approved hours, timesheet ID etc.

New_User_Registration_Page: This class is used when a new user has to be added to the PMS. The new employee detail operations is used in this class to register a new user to the system.

WorkProfile_Page: This class has the information of the employees working in that organization. The class saves the personal information of the employee like his name, address, and contact info.

Change_Password_Page: This class has is called when the user wants to change his password. This class has the attributes like old password and asks the user to set new password.

Login_Page: This class is called when the employee or the employer wishes to login into the system. This class takes the credentials as the input from the user and forwards it to the PMS_Controller which then validates and logs the user in to the PMS.

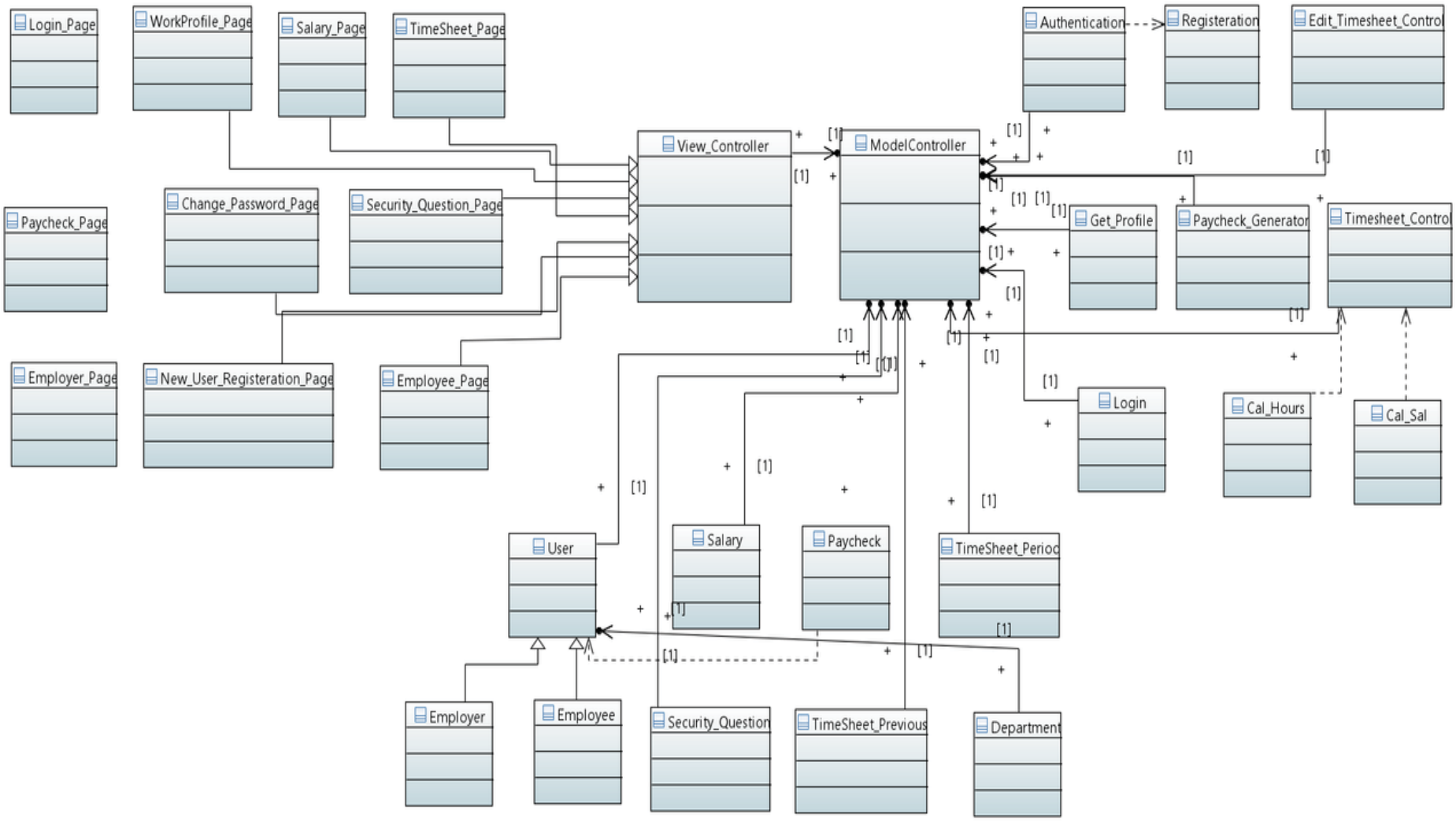


Fig 6.1.1 Minimal Class Diagram. For PMS.

6.2 State Machine

This section provides the state machines for the main control objects in each of the major subsystems and the overall System. The goal of UML state machines is to overcome the main limitations of traditional finite-state machines while retaining their main benefits.

Main State Diagram

This is the state machine representing the Overall Payroll Management System.

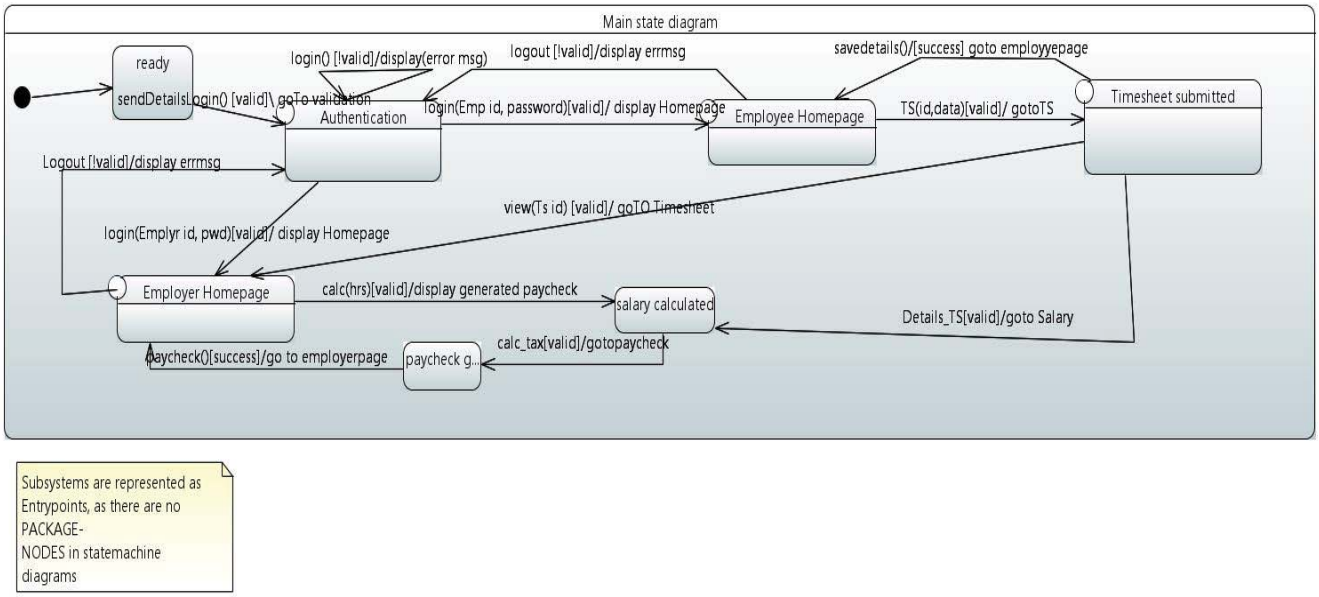
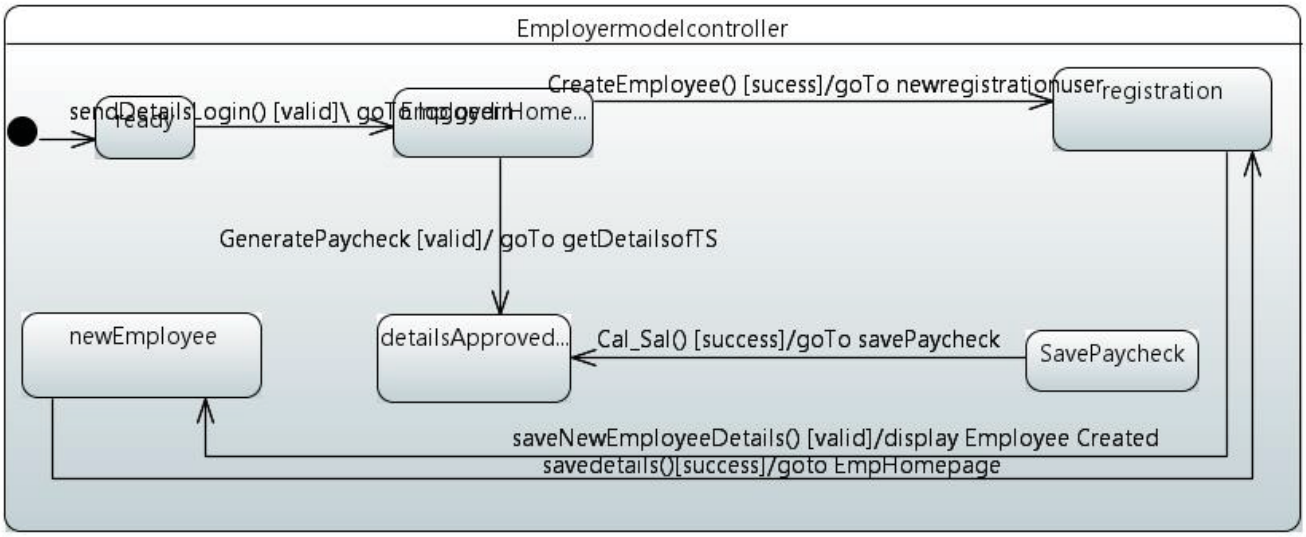


Figure.6.2.1: MainStateDiagram

Employer model controller

This is the state machine representing the transition of states from the Controller in the Employee



Subsystem.

Figure.6.2.2:Employermodelcontroller

Authentication Model Controller

This is the state machine representing the transition of states at the login of the users.

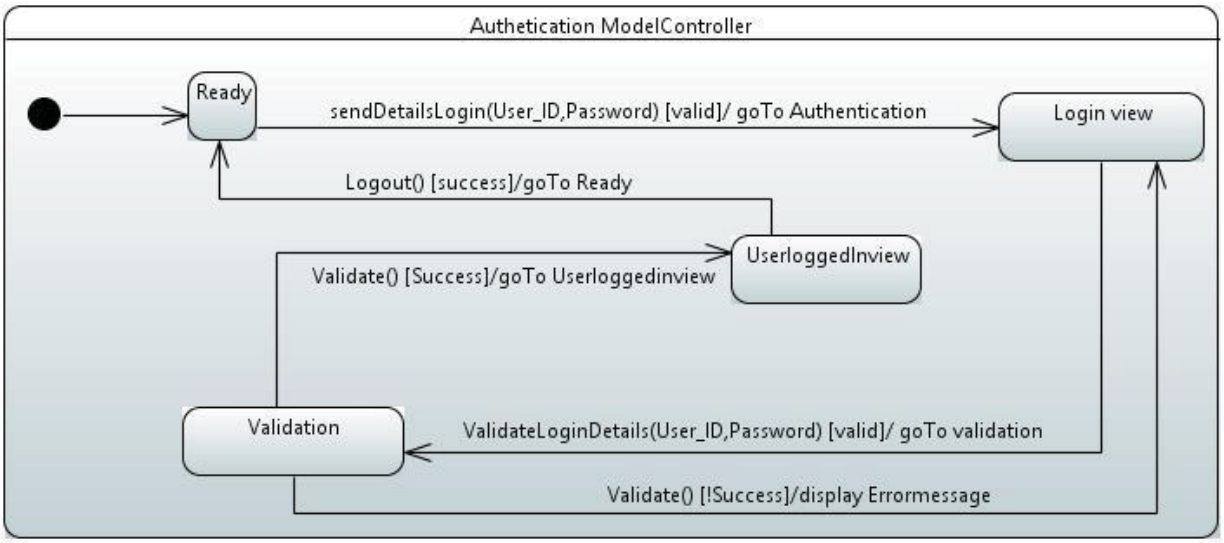


Figure.6.2.3: AuthenticationModelController

Employer model controller

This is the state machine representing the transition of states from the Controller in the Employer Subsystem.

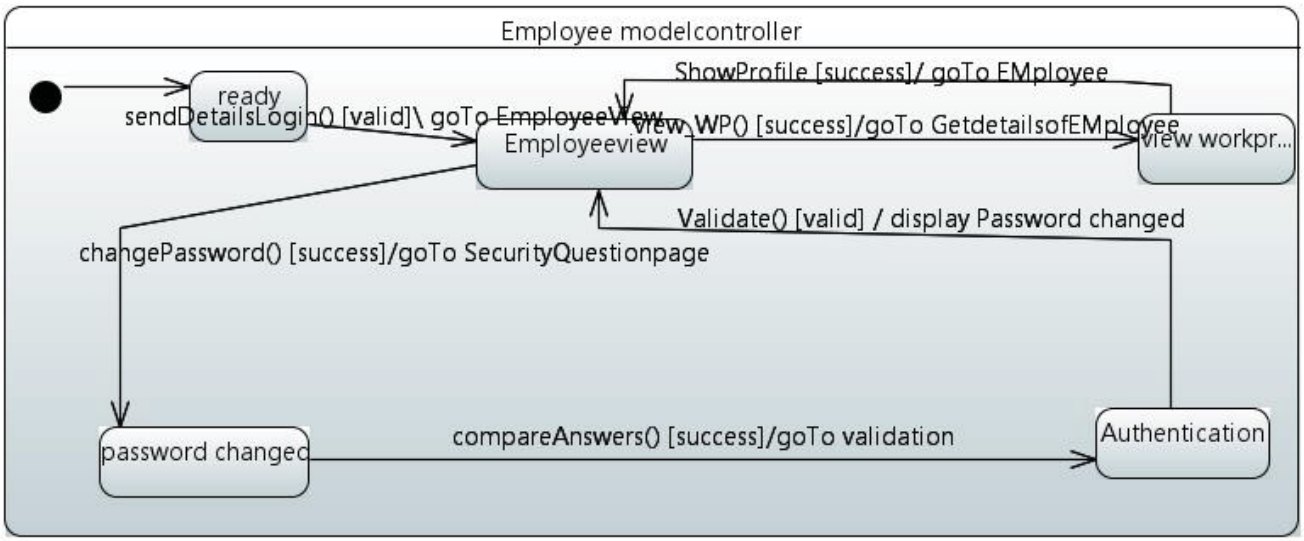


Figure.6.2.4:Employee model controller

Timesheet_controller

This is the state machine representing the transition of states from the Controller in the Time Sheet subsystem

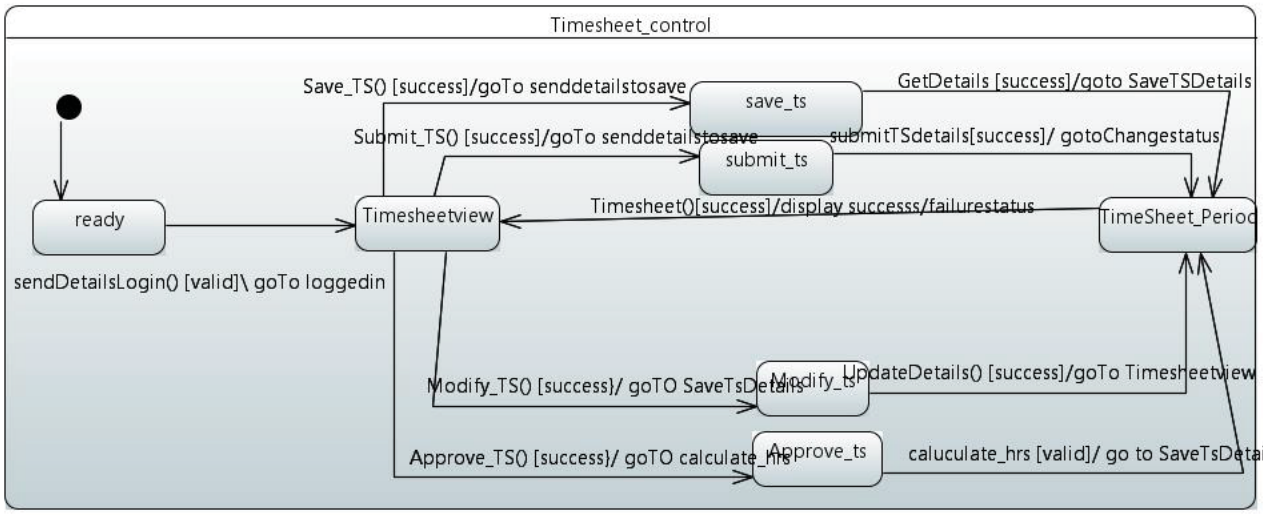
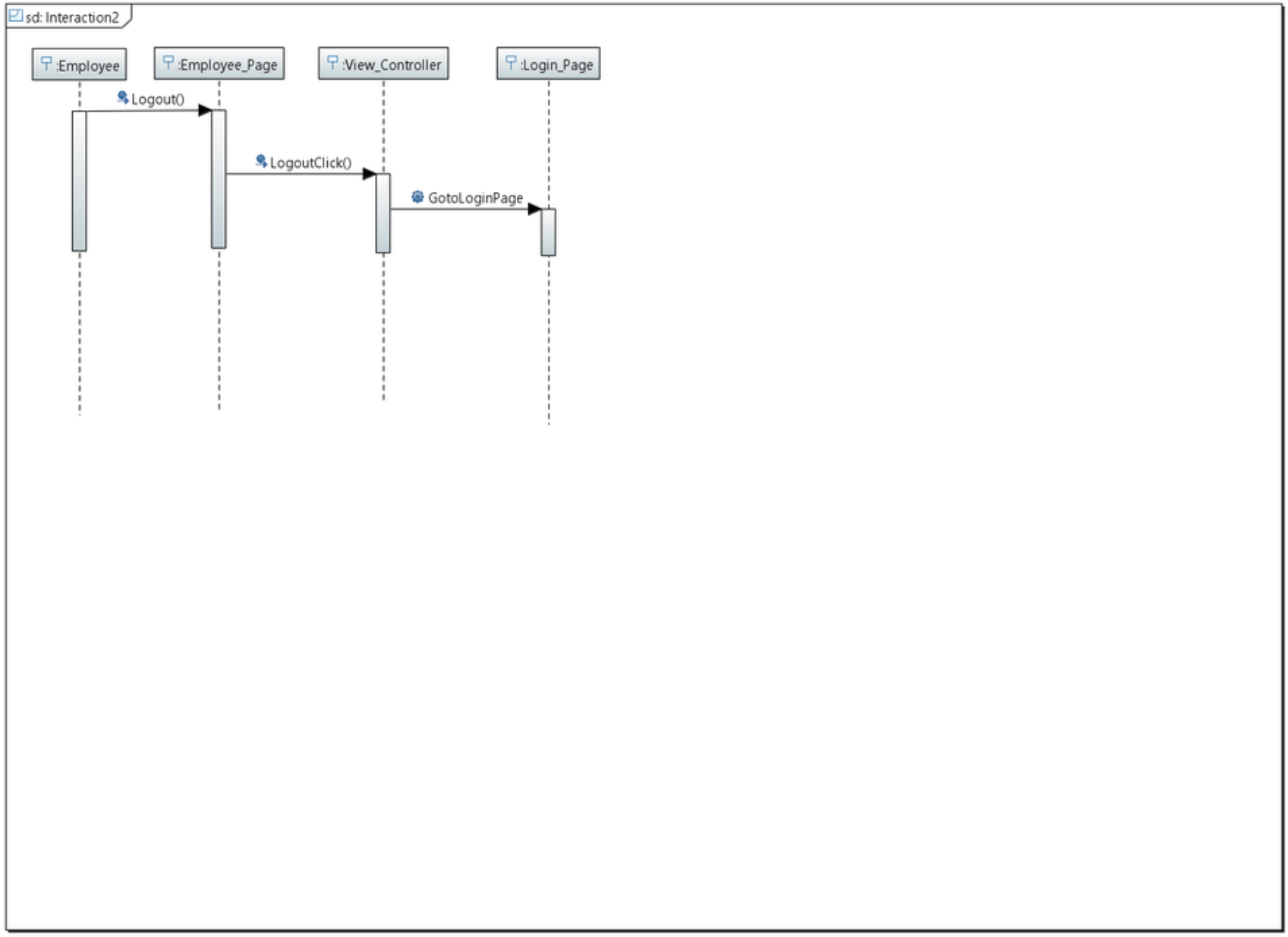


Figure.6.2.5:TimeSheet_controller

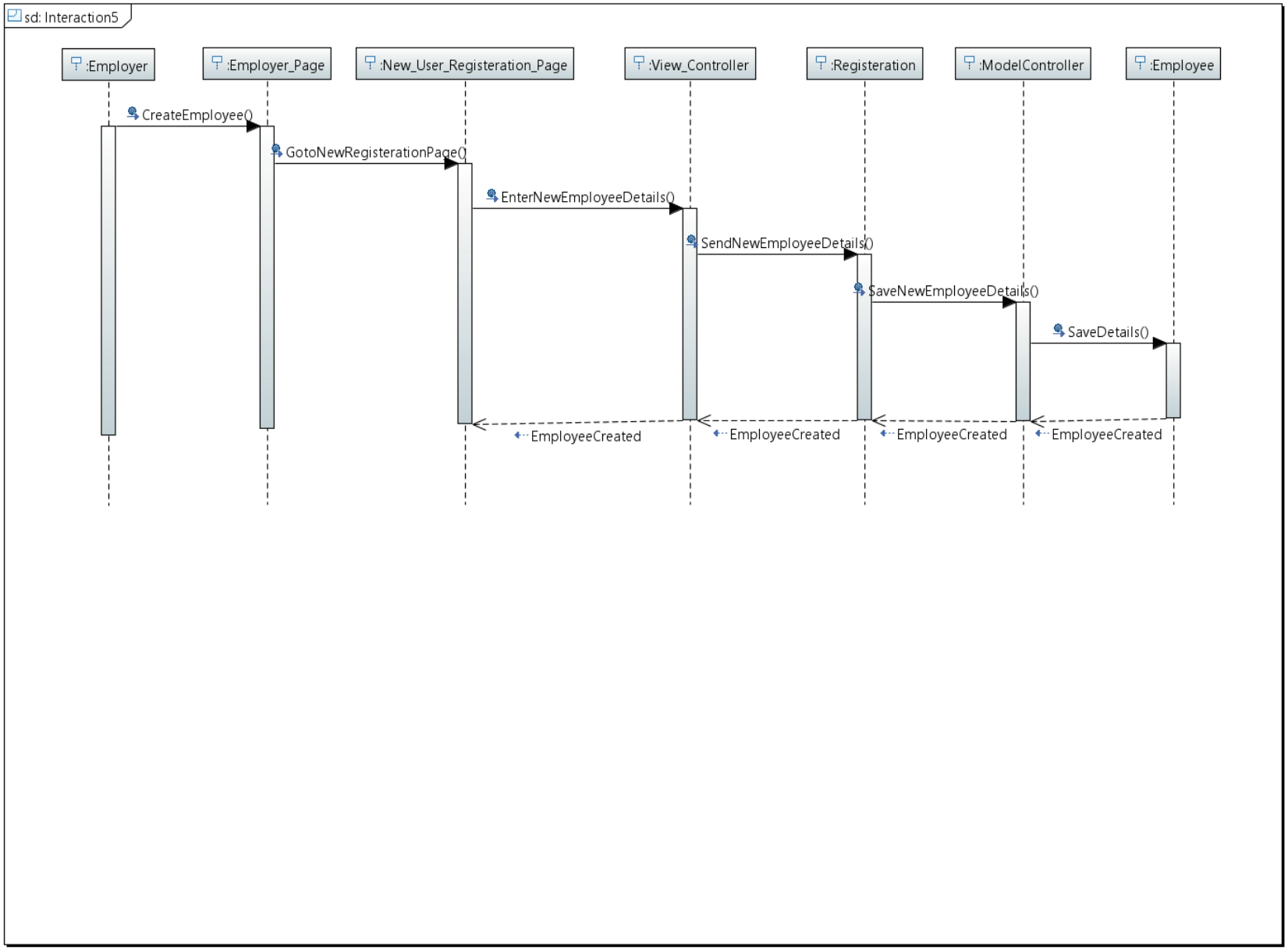
6.3 Object Interaction

The diagram which represents the interaction between the objects that describes the working of operations is called as a Sequence Diagram. Basically, the communication is in the form of messages. Each use case has its own sequence diagram where there are actors, events and lifelines. The primary use of sequence diagram is to design, document and validate the architecture. The other purpose of sequence diagram is to visualize the system behavior.

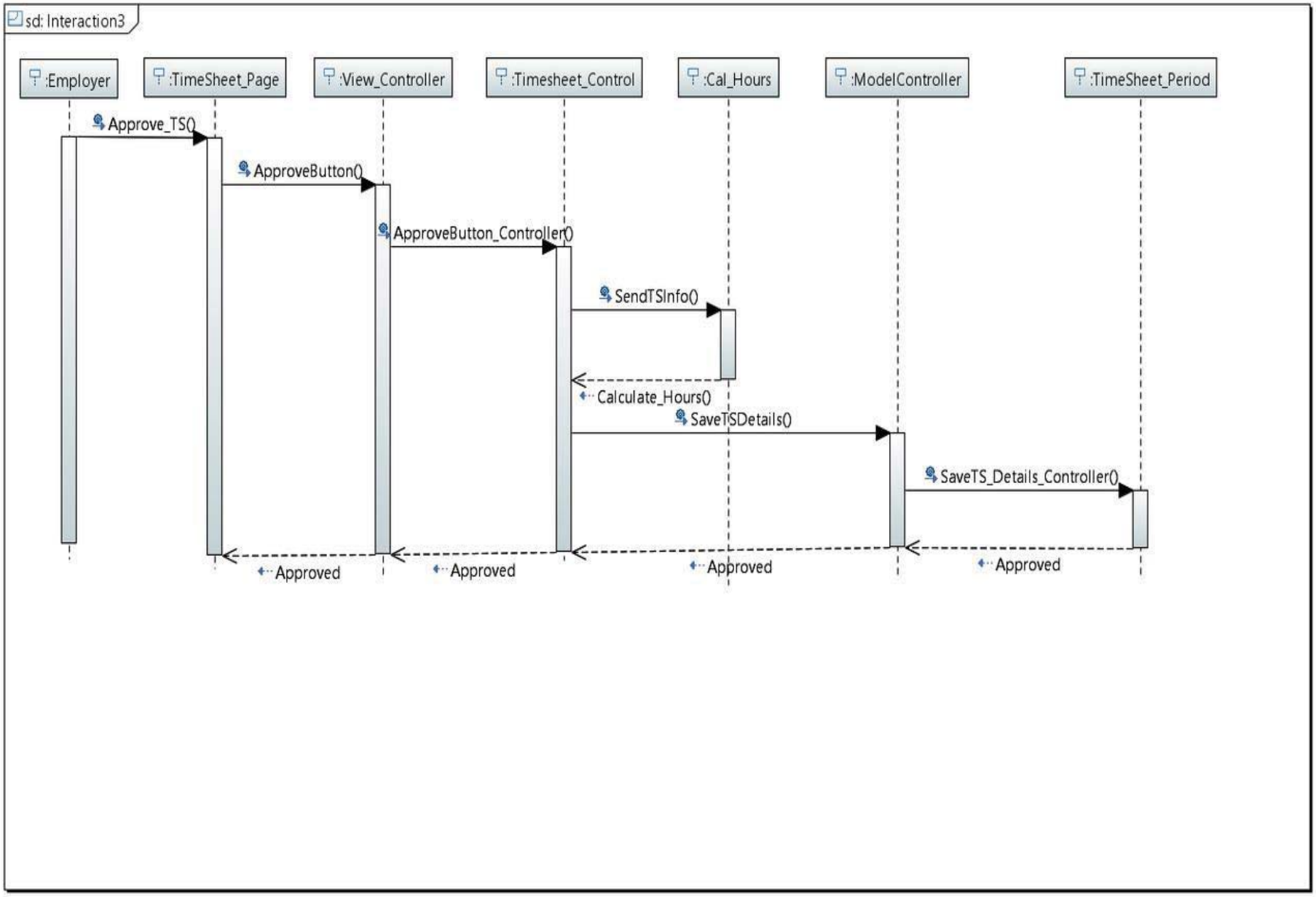
1) PMS_21_Logout: This diagram illustrates the employee log out.



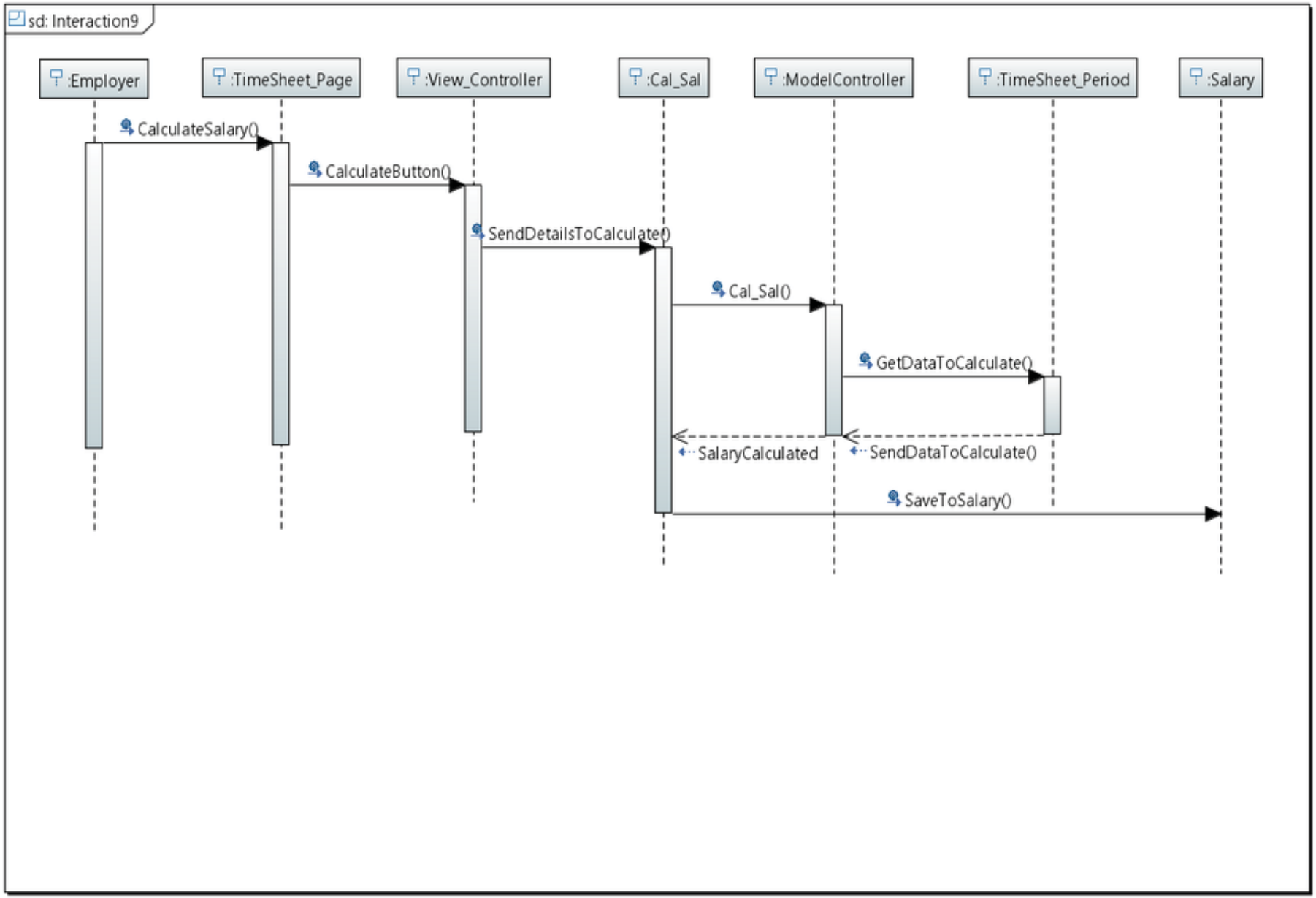
2) PMS_13_AddEmployee: This figure shows how to add an employee to the PMS application



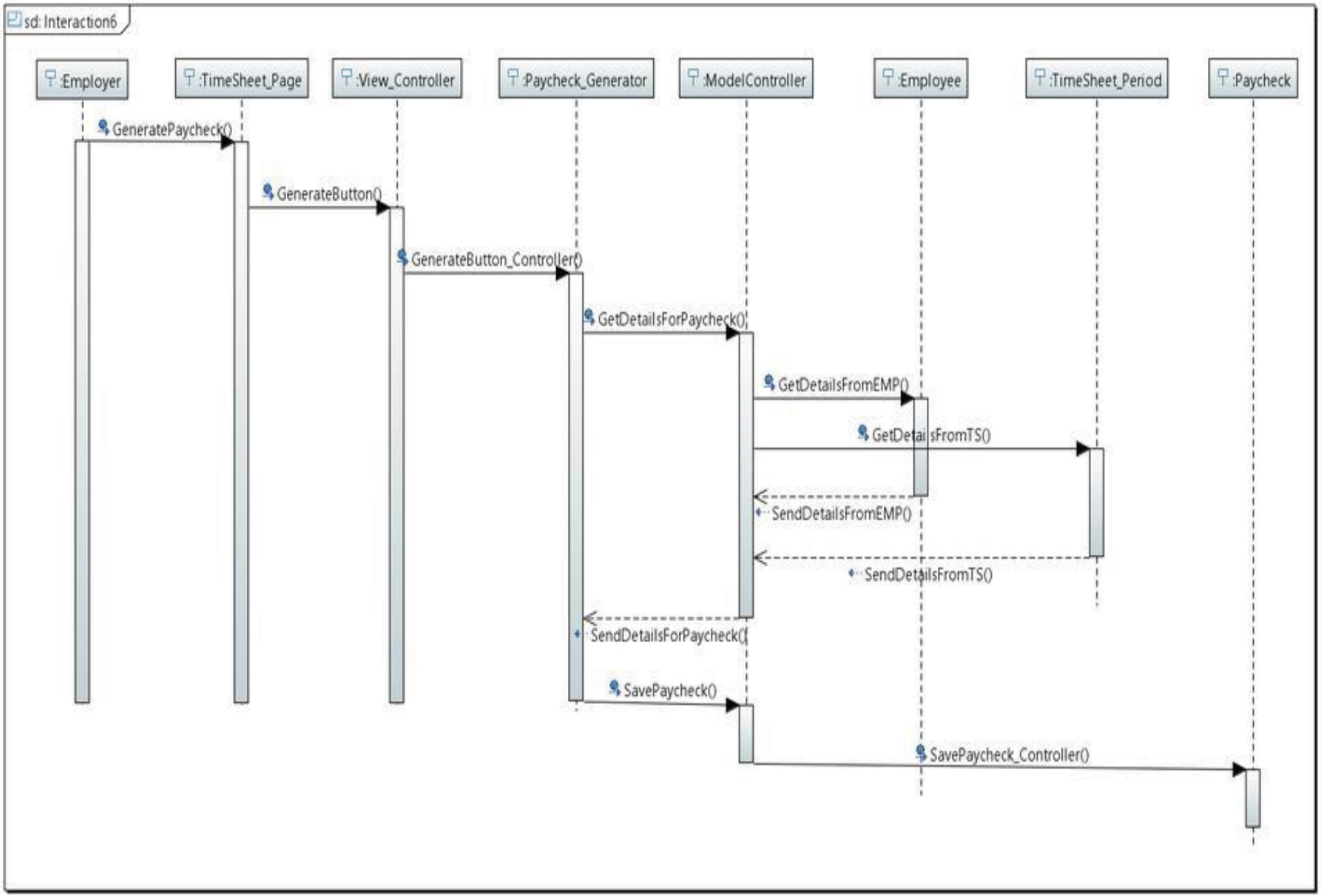
3) PMS_05_ApproveTS: This sequence diagram explains how the employer approves the timesheet.



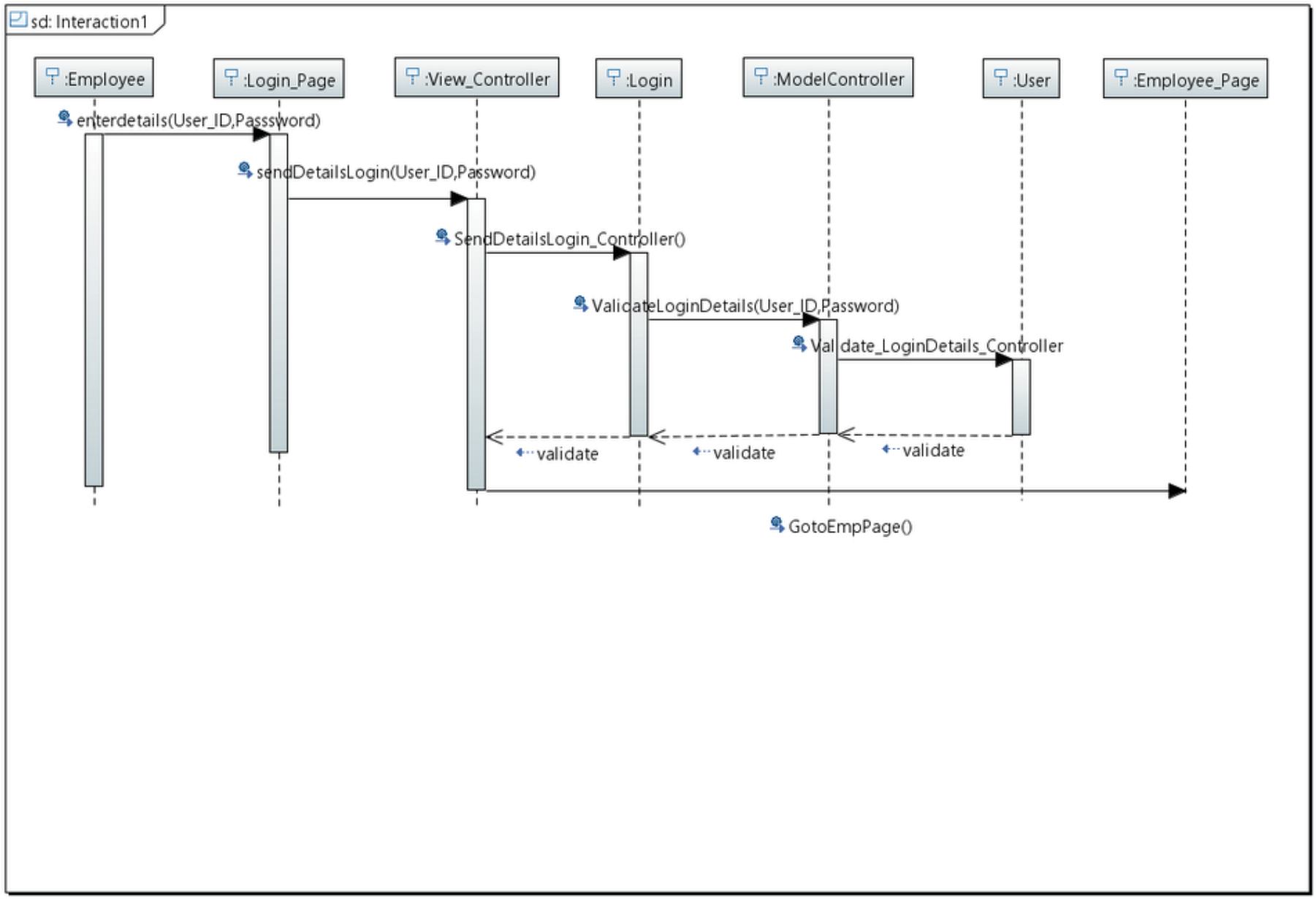
4) PMS_07_CalcSal: This figure explains how the salary is calculated for the employee's by PMS.



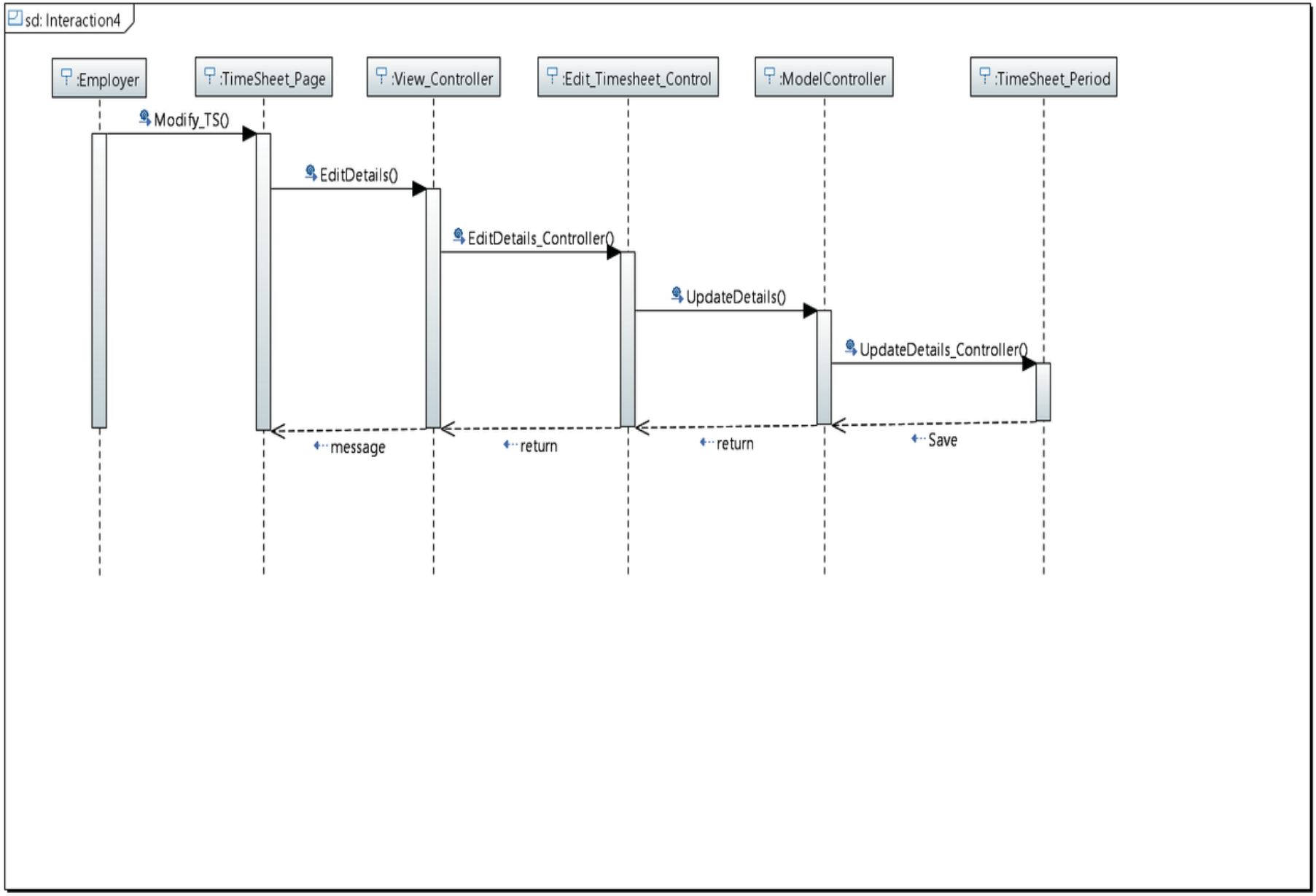
5) PMS_09_PayCheck: This figure illustrates how the PMS generates the paycheck.



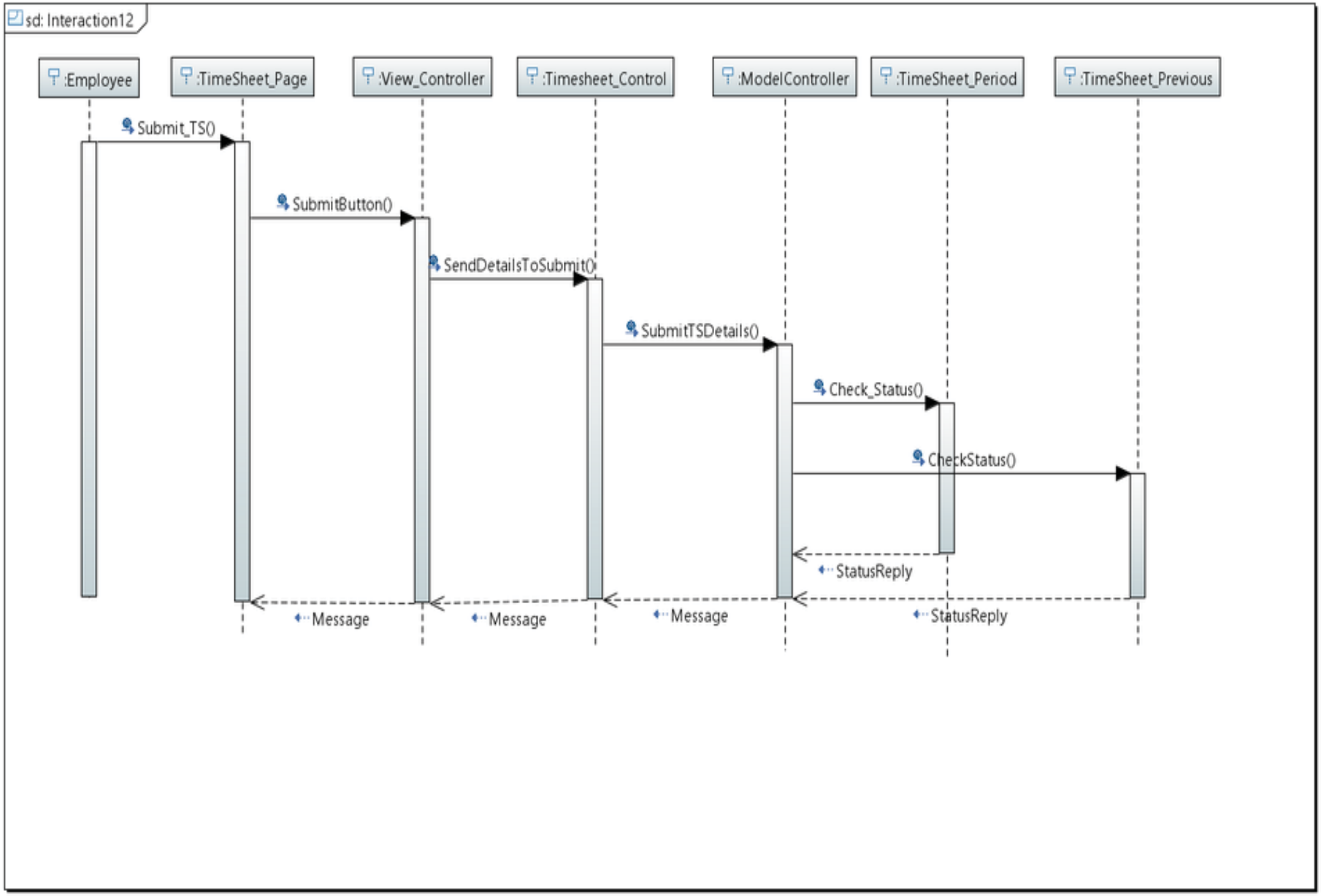
6) PMS_01_Login: This diagram explains the employee/ employer login.



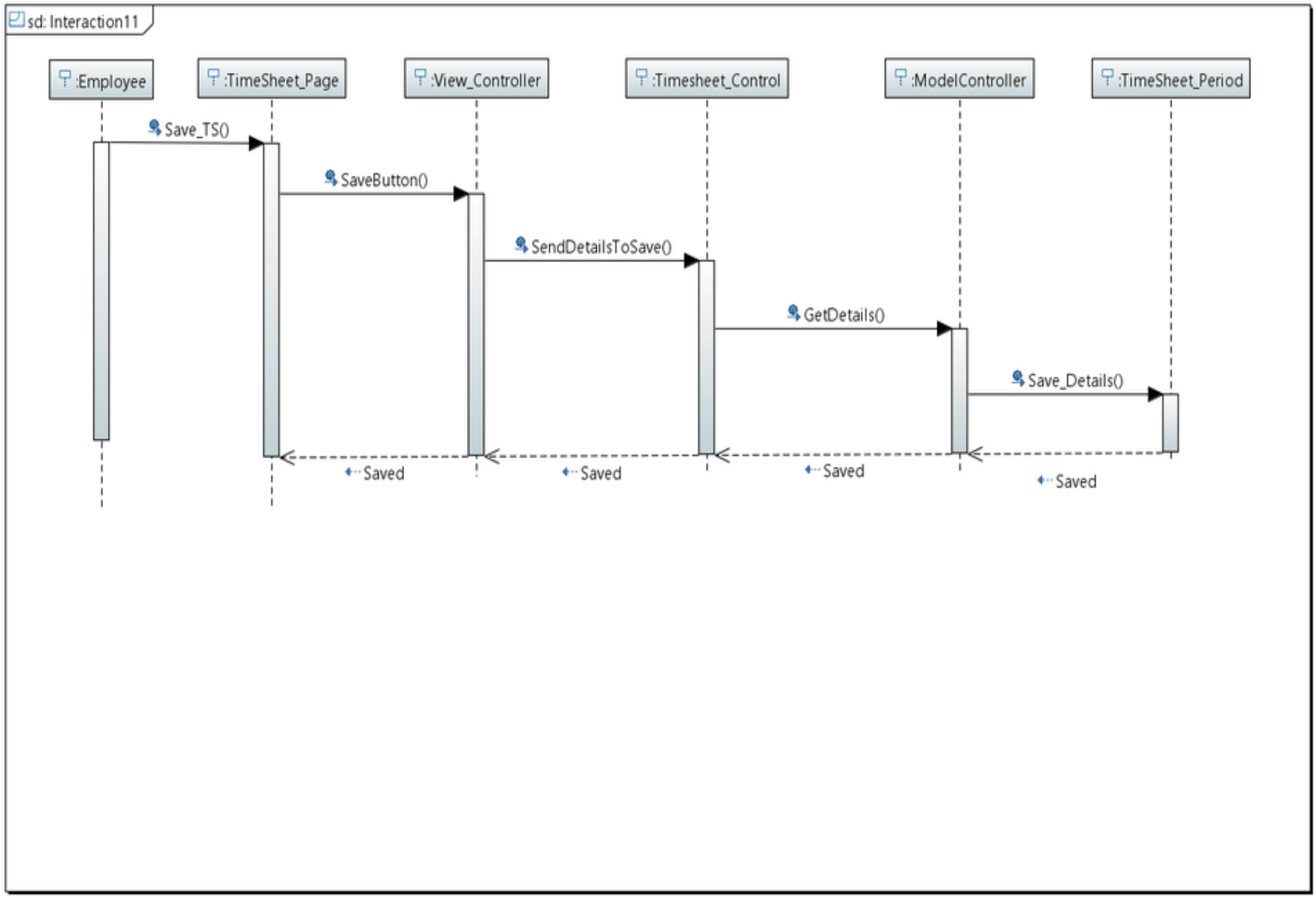
7) PMS_04_ModifyTS: This diagram shows how an employer can modify the timesheet.



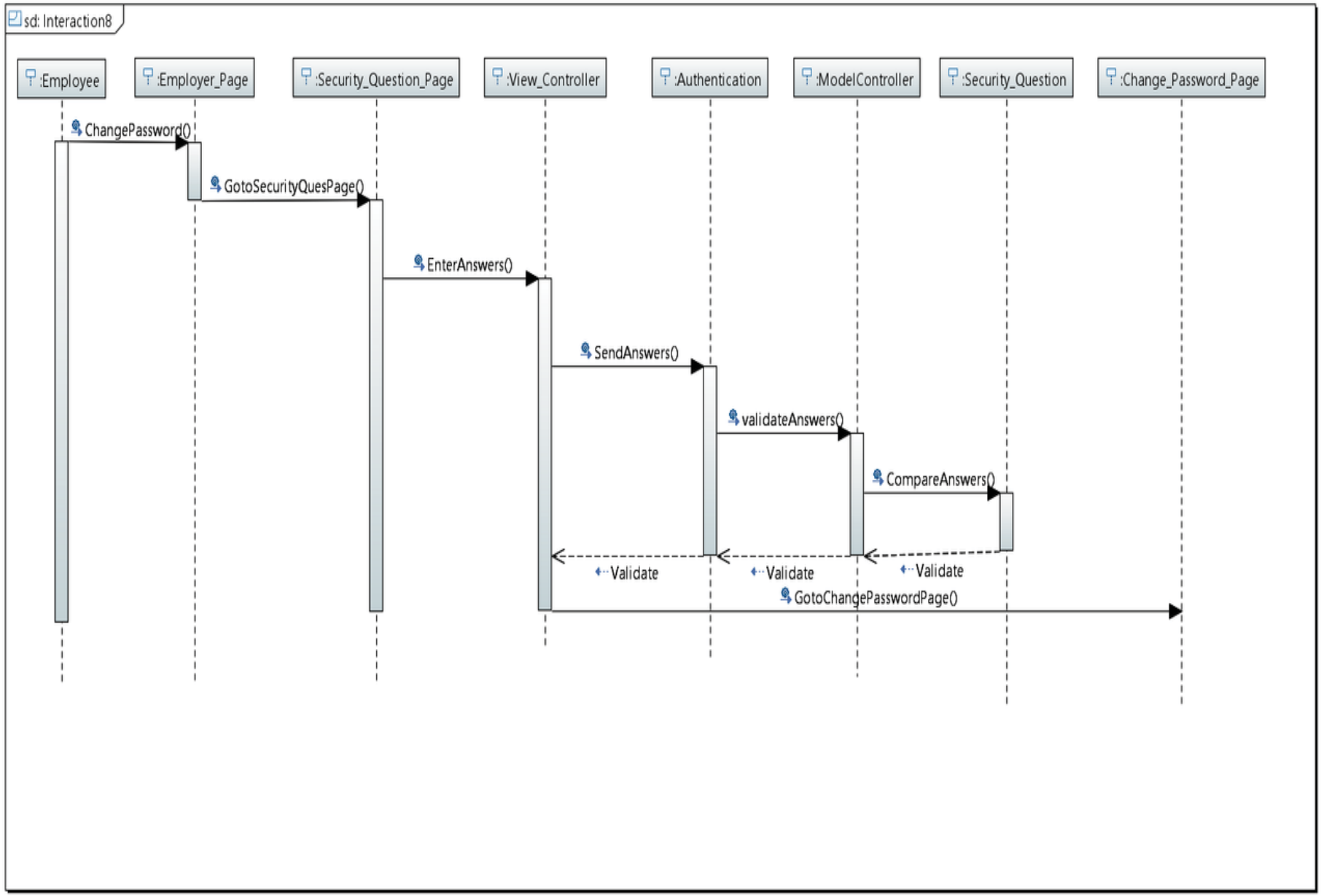
8) PMS_004_duplicate: This diagram shows the working of not allowing to submit the duplicate timesheets.



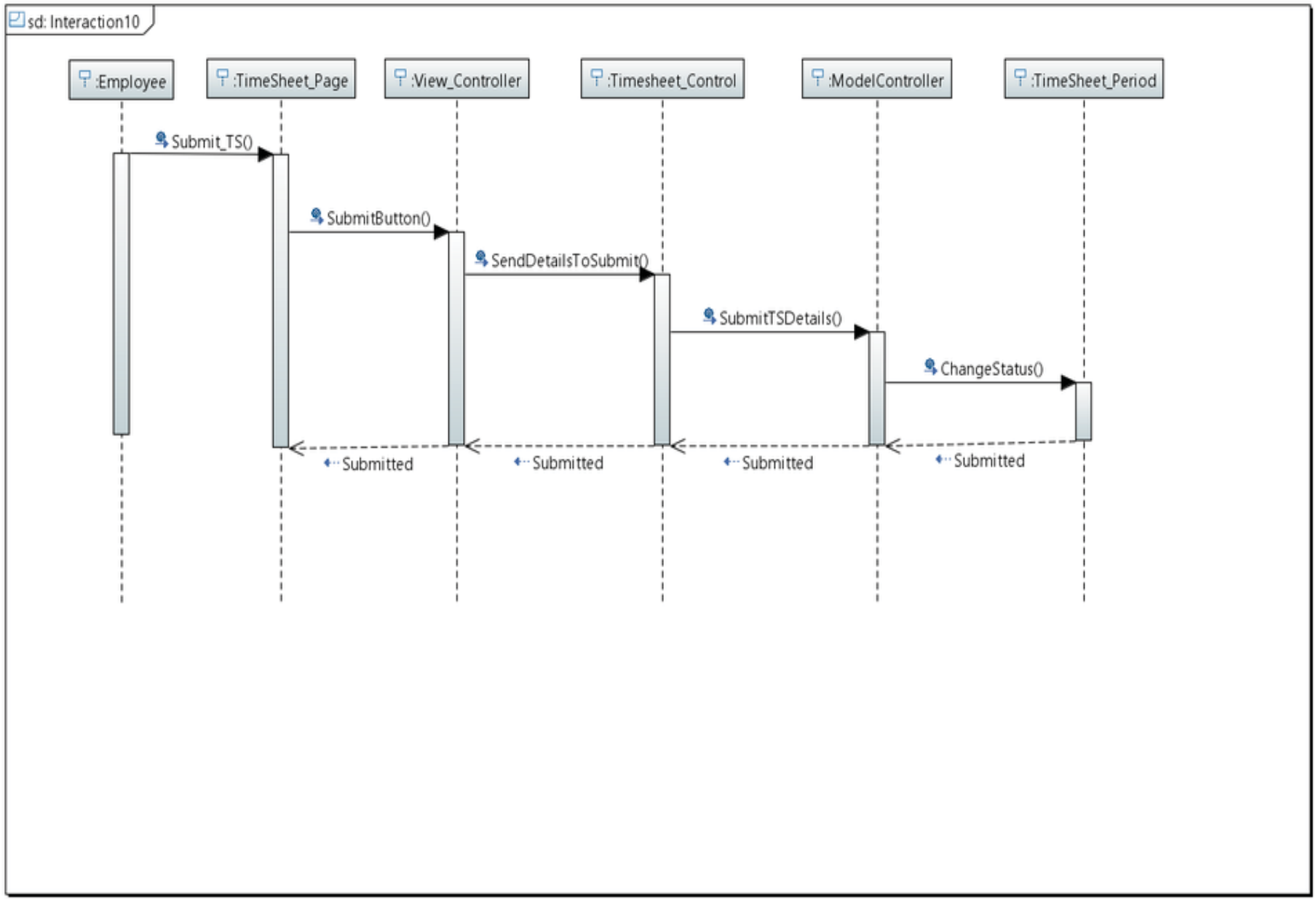
9) PMS_08_SaveTS: This shows the logic involved in saving the timesheet.



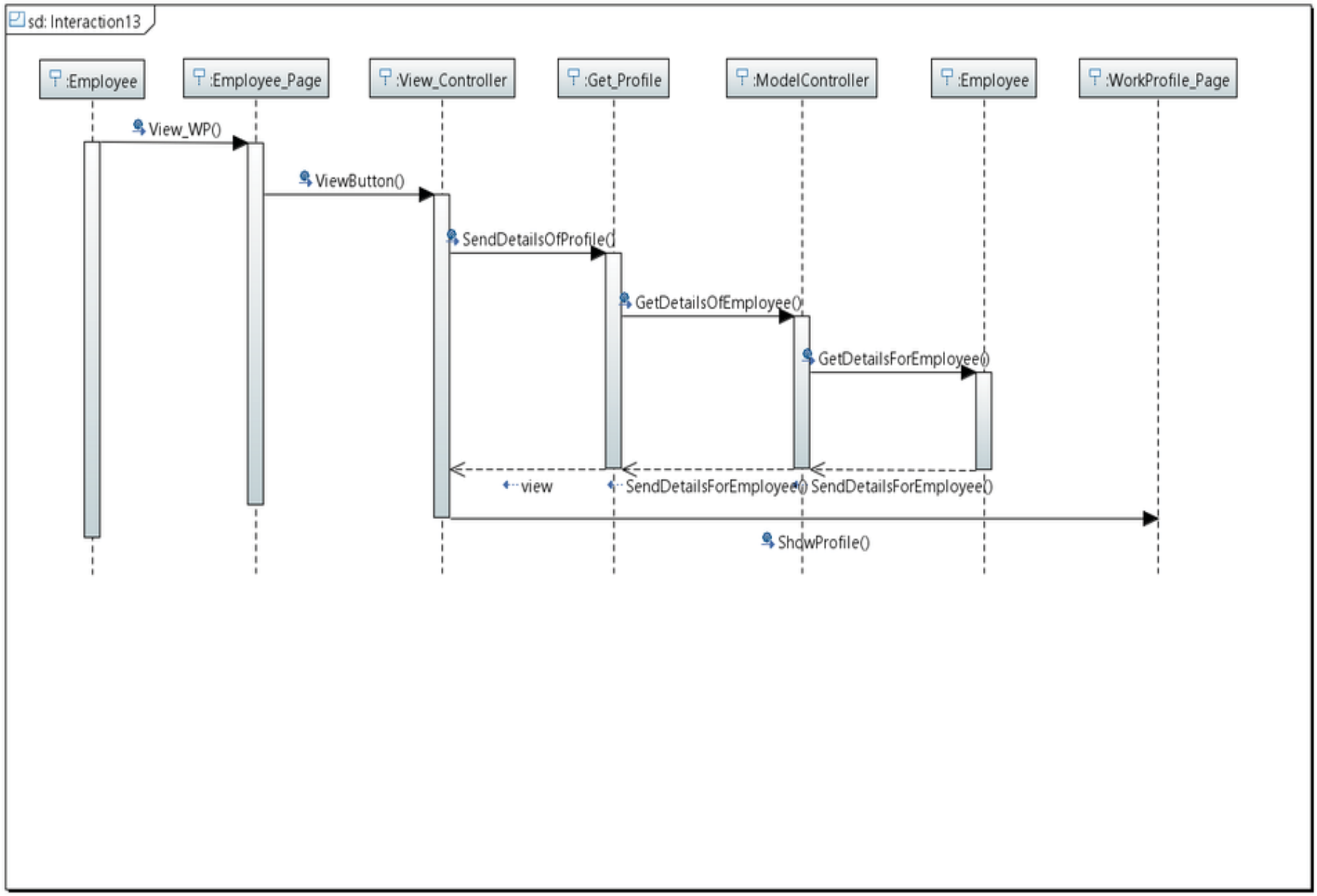
10) PMS_005_security: This shows the logic behind the change password functionality.



11) PMS_18_SubmitTS: This figure shows the working of submitting the timeheet.



12) PMS_17_WorkProfile: This figure illustrates how to view the work profile of an employee.



6.4 Detailed Class Design

In this section we describe the purpose of each subsystem and purpose of each class in the subsystem, the design patterns we implement in each class, the control objects for various subsystems and also the constraints using the Object Constraint Language.

6.4.1 Class OverView

In this section we mention the subsystems used in our system and the design patterns used in the subsystems and have explained the reason to use a particular design pattern.

PMS_Business_Logic:-

Model_Controller: The Model_Controller connects the two subsystems of the PMS system that is the PMS_Controller and the PMS_Model. This class is one of the most important classes of the system as this controller is used for every operational request made by the user.

View_Controller:- : View_Controller connects PMS_View subsystem and the PMS_Controller subsystem. When a request is sent by the user from the View_Controller , the request is forwarded to the Controller package and the response from the PMS_Controller package is sent to the View_Controller.

PMS_View Subsystem:-

Login_Page – This represents the login page of the system that will be displayed on the web browsers. This class uses the singleton design pattern to authenticate against the database data.

Employee_Page - This page represents the home page of the system for employee. Home screen has the options to go the time sheet page, to view the paycheck, to view the work profile. This class uses the builder pattern.

Employer_page – This page represents the homepage of the system for the employer. This home screen has the options to search for the timesheets, to view the work profile. This class uses the builder pattern.

Timesheet_page – this page represents the timesheet where the employee can add the timesheet, save the timesheet, submit the timesheet and the employer can modify the timesheet, approve the timesheet and calculate the salary and generate the paycheck by referring to the details from the timesheet. Both the user's i.e employee and the employer work on the timesheet as this is the important page of the system. This class uses the builder pattern.

Paycheck_page - This represents the paycheck when it is displayed. It has the information about the paycheck. This class will displays the information like like paycheck ID, tax, pay frequency, pay period. This class uses the builder pattern.

Salary_Page - This page represents all the information about the salary when employee views the salary. This class deals with the information like Tax, rate per hour, approved hours, timesheet ID etc. This class uses the builder pattern.

WorkProfile_Page - This page represents all the information about the employee when employee views his work profile. The class saves the personal information of the employee like his name, address, contact info. This class uses the builder pattern.

New_User_Registeration_Page – This page represents the page when the employer has to add a new employee to the system. The enter new employee detail operations is used in this class to register a new user to the system. This class uses the singleton design pattern to send information to the database.

Security_Question_Page – This page represents the page where the employee has to answer the security questions when he wants to change his or her password for security purpose. Here the user will enter the answers which will be sent by the view_controller to the PMS_Controller package. This class uses the singleton design pattern authenticates the answers against the database.

Change_Password_Page – This page represents the page when the employee needs to enter the old password and the new password to change his or her password. This class uses the singleton design pattern to send information to server to change the password.

PMS_Controller Subsystem:-

Timesheet_Control – This class is responsible for the operation of the hours entered on the Timesheet, This class has many operations like sending and receiving the timesheet info from other classes and also to save the timesheet as well as submit the timesheet. This is main control object for the operations which are carried out in the system. This class uses the singleton design pattern to redirect to the other timesheet function.

Edit_Timesheet_Control: This class is mainly responsible when there are any changes that has to be done to the Timesheet. This class has the operation1 of updating the details of the timesheet. The modify timesheet use case uses this class for its operation. This class uses the observer pattern

Login: This class is responsible for giving employee and the employer the access to the system. This class has the operation of validating the user's login credentials. This class uses the singleton design pattern to check for the credentials provided.

Cal_Hours: This class is responsible for calculating the hours that are entered in the Timesheet. This class has the operation of calculating the hours. This class uses the observer pattern

Get_Profile: The Get_Profile class is responsible for fetching the user information. This class has the operation of fetching the user's profile information from the database. This class uses the builder pattern.

Registration: When a new user has to be added to the PMS this class is called. This class has the operation to save new employees information to the database. This class uses the singleton design pattern.

Paycheck_Generator: This class is responsible for generating the paychecks. This class has the operation of getting the information required for generating the paycheck and also has the operation of saving each and every paycheck that has been generated. This class uses the observer pattern.

Authentication: This class is responsible for authenticating the response typed by the user in the security questions class. This class has the operation to validate the answers entered by the employee. This class uses the singleton design pattern.

Cal_Sal – This class is responsible for the calculations that has to be done for the hours the employee has worked for. This class has the operation of calculating the salary and then saving it. This class uses the observer pattern.

PMS_Model Subsystem :-

Timesheet_Period: The Timesheet_Period class is responsible for storing the details of the timesheet of the current week. This class details are forwarded then to calculate salary and generate paycheck. This Class provides the information to the model controller when the data is requested for calculating the salary. This class uses the singleton design pattern.

Timesheet_Previous: The Timesheet_Previous class is responsible for storing the details of the timesheet of the previous week. When a request is called to view the previous timesheet, this class is called. This Class provides the information to the model controller when the data is requested to view the submitted and approved timesheets. This class uses the singleton design pattern.

Salary: The salary class is responsible for managing the data like the salary of the employees. This class mainly deals with the information like gross salary, net salary, tax, Approved hours etc. This Class provides the information to the model controller when the data is requested during the view salary operation. This class uses the singleton design pattern.

Department: This class saves the information of the Organizational Departments. This class is called when any information regarding the department is needed. This class uses the Composite pattern.

User: The class User is the parent class which handles information of both the employee and the employer in the PMS. This is the parent class for the Employee and Employer class. This Class provides the information to the model controller when the data is requested during the login process. This class uses the Composite pattern.

Employee: The Employee class is the child class of the user class. This class stores the information of the employee that has been registered to the system. This class also contains personal information like contact number and address. This Class provides the information to the model controller when the data is requested to view the Work Profile of the employee. This class uses the Composite pattern.

Employer: The Employer class is the child class of the user class. This class stores the information of the employer that has been registered to the system. This class also contains personal information like contact number and address. This Class provides the information to the model controller when the data is requested for any operation. This class uses the Composite pattern.

Paycheck: The Paycheck class is responsible for managing the paycheck information. It contains information like pay frequency, net pay, gross salary, pay check ID. This Class provides the information to the model controller when the data is requested during the view paycheck option. This class uses the Composite pattern.

Security_Question: The security_question class is responsible for verifying the users. It stores questions and answers of the users so that the user can verify himself and then can log into the system. This Class provides the information to the model controller when the data is requested during the change password operation.

6.4.2 OCL Statements

In this section, we introduce the OCL constraints with the class invariants, pre and post conditions for the methods in the class of the control object in each major subsystem.

a.Subsystem Authentication modelcontroller for the login process:

Method:Login(User_ID, Password) for class:User

OCL:

context ModelController **inv:**

self. User_ID && **self**.Password == valid

context ModelController:: login() **pre:**

self.sendDetailsLogin ==success

context ModelController:: login()**post:**

if self. User_ID == valid && **self**.Password ==valid

then UserloggedInview

else errmsg

b.Subsystem Employee Modelcontroller for the Employee Operations

Method:ViewWP(User_ID, Password) for class:Employee

OCL:

Context ModelController **inv**:

Context ModelController::viewWP () **pre**:

If self. Emp_ID==valid && **self.**Password==valid && **self.**GetdetailsofEployee=success

then viewWP()

else errsmsg

Context ModelController::viewWP() **post**:

result=forall(E.Emp_Id|showProfile())

Method:ChangePassword (User_ID, Password) for class:Employee

OCL:

Context ModelController::ChangePassword () **pre**:

If self. Emp_ID==valid && **self.**Password==valid &&**self**.GetdetailsofEMployee=success

then SecurityQuestion page()

if self.validateanswers==true

then changepassowrd()

Context ModelController::ChangePassword () **post**:

result=ChangePassword->validatedanswers()

c.Subsystem Employer Modelcontroller for the Employer Operations

Method:CreateEmployee(Emp_Id) for class:Employer

OCL:

Context ModelController **inv**:

Context ModelController::CreateEmployee() **pre**:

If **self.**Emp_ID==valid && **self.**Password==valid

then create employee()

Context ModelController::CreateEmplpoyee() **post**:

result=savenewemployeedeatils()

Method:GeneratePaycheck(Paycheck_id) for class:Employer

OCL:

Context ModelController **inv**:

Context ModelController::GeneratePaycheck() **pre**:

If Self. Emp_ID==valid && **self.**Password==valid &&**self.**App_Ts==success

then GeneratePaycheck()

else errmsg

ContextModelController::GeneratePayCheck () **post**:

result=savePaycheck ()

d.Subsystem TimeSheet_controller for the Timesheet related Operations

Method:Modify_TS(Rep_hrs) for class:Employer

OCL:

Context Timesheet_period **inv:**

Context Timesheet_period :: Modify_TS(Rep_hrs: integer) **BooleanPre:**

if self. Emp_ID==valid && **self.**Password==valid

if self. Submit_TS==True && **self.**Modify_TS==success

then proceed;

else errmsg;

Post:result= Modify_TS->Timesheet_period)

Method:Approve_TS(Apr_hrs, Ts_Id) for class:Employer

OCL:

Context Timesheet_period **inv:**

Context Timesheet_period :: Approve_TS(App_hrs: integer, TS_Id:integer) **BooleanPre:**

if self. Emp_ID==valid && **self.**Password==valid

if self. Submit_TS==True && **self**.Approve_TS==success

then proceed;

else errmsg;

Context Timesheet_period :: Approve_TS(App_hrs: integer, TS_Id:integer) **Boolean Post:**
result=Calc_hrs->forall(App_hrs)

Method:Save_TS (Ts_Id) for class:Employee

OCL:

Context Timesheet_period **inv:**

Context Timesheet_period :: save_TS(TS_ID: string) Boolean

Pre: if self. Emp_ID==valid && **self**.Password==valid

if self.Save_TS==True

then proceed;

else errmsg;

Context Timesheet_period :: save_TS(TS_ID: string) **BooleanPost:**

result=save_TS->Timesheet_period) is modified

Method:Submit_TS(Ts_Id) for class:Employeee

OCL:

Context Timesheet_period **inv:**

Context Timesheet_period :: Submit_TS(TS_ID: string**) Boolean Pre:**

if **self.**Emp_ID==valid && **self.**Password==valid

if submit_TS==True

then proceed;

else errmsg;

Context Timesheet_period :: Submit_TS(TS_ID: string**) Boolean Post:**

result=submit_TS->Timesheet_period is modified

8. Glossary

Actors: Represent roles played by individuals or any external objects.

Application: Payroll management System

Approve Timesheet: The employer will approve the timesheet of an employee.

Class Diagram: The diagram which shows the structure of the system with the help of classes and relationship between them. This diagram depicts the static structure.

DD: Design Document

ER: Entity Relationship

IDE: Integrated Development Environment is a standard procedure which supports program development.

Milestone: The end point of a software process activity.

MVC: Model View Controller

Object Diagram: The possible object combinations of a specific class diagram are presented.

OCL: Object Constraint Language.

PMS: Payroll Management System

Sequence Diagram: The communication between different objects is represented through messages.

Session: the time interval for which the application is open.

SRD: Software Requirements Document.

State Machine: It depicts the behavior of single object by going through all the states sequentially. It is basically a life cycle.

System: Framework which describes functionalities to achieve some objectives.

Timeout: The specific period of time interval for which the application shall remain inactive.

UML: Unified Modeling Language

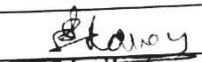
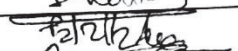
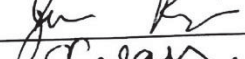
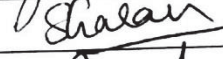
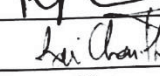
USDP: Unified Software Development Process

Use Case: A graphical representation of interaction between the objects of the system.

Warnings: The alerts and warnings with respect to validation of different field.

9. Signature

9. Signature

Date	Name	Signature
11/28/15	Amit H. Shenoy	
11/28/15	Prafulla P. Ghadage	
11/28/15	Joma Rodriguez	
11/28/15	Sharan Tej Kondumuru	
11/28/15	Srinivas Reddy Manda	
11/28/15	Sai Chaithra Allala	
11/28/15	Yao Xiao	

10. Reference

[1] Marston, T. (2012, October 14). What is the 3-Tier Architecture?

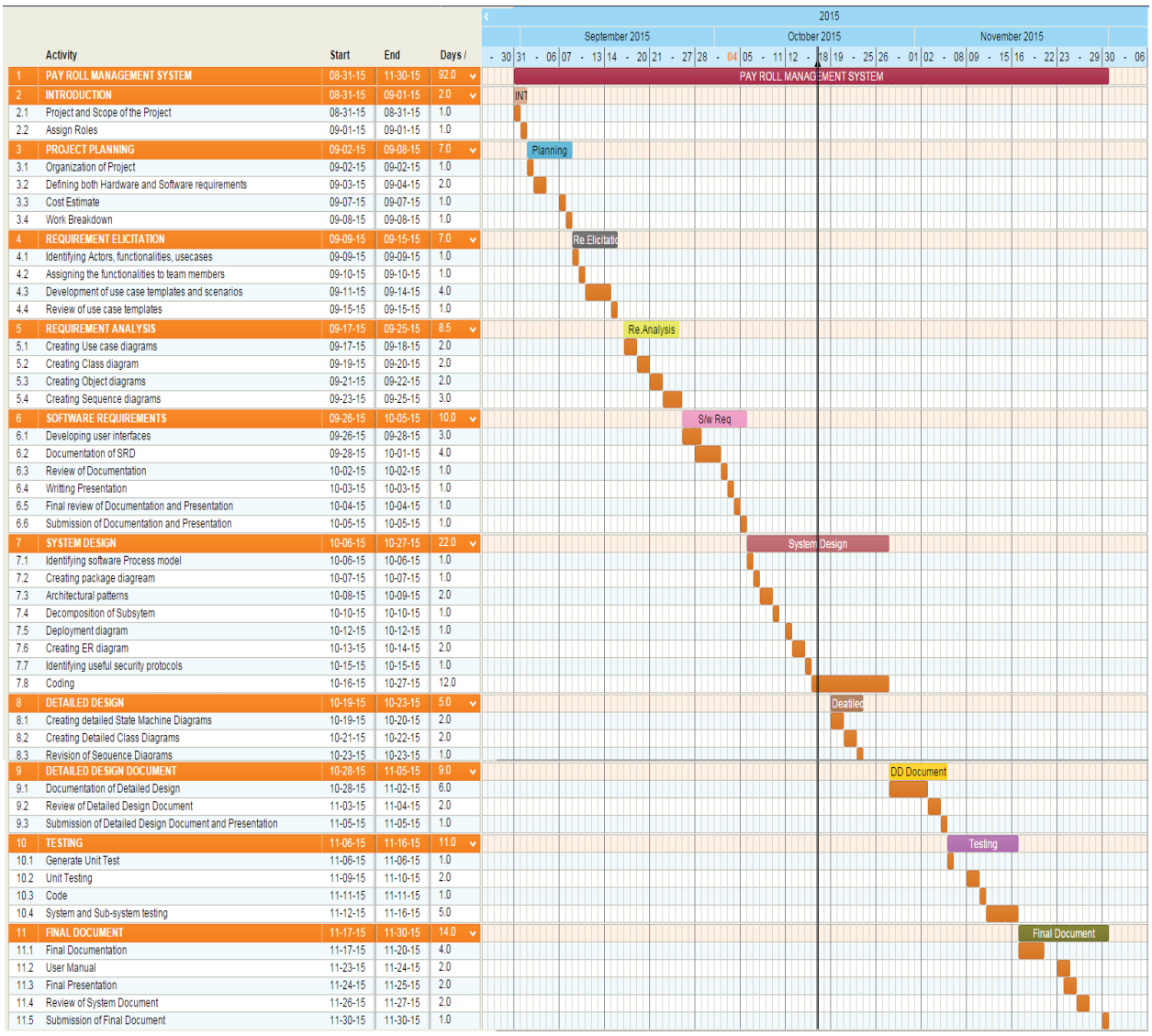
[2] Ambler, S. (n.d.). UML 2 State Machine Diagrams: An Agile Introduction

[3] Dalling, T. (2009, May 31). Model View Controller Explained.

[4] Bautista, N. (2010, July 7). A Beginner's Guide to Design Patterns - Tuts Code Article.

11. Appendix

11.1 Appendix A: Project Schedule.



	Activity	Start	End	Days
1	PAY ROLL MANAGEMENT SYSTEM	08-31-15	11-30-15	92.0
2	INTRODUCTION	08-31-15	09-01-15	2.0
2.1	Project and Scope of the Project	08-31-15	08-31-15	1.0
2.2	Assign Roles	09-01-15	09-01-15	1.0
3	PROJECT PLANNING	09-02-15	09-08-15	7.0
3.1	Organization of Project	09-02-15	09-02-15	1.0
3.2	Defining both Hardware and Software requirements	09-03-15	09-04-15	2.0
3.3	Cost Estimate	09-07-15	09-07-15	1.0
3.4	Work Breakdown	09-08-15	09-08-15	1.0
4	REQUIREMENT ELICITATION	09-09-15	09-15-15	7.0
4.1	Identifying Actors, functionalities, usecases	09-09-15	09-09-15	1.0
4.2	Assigning the functionalities to team members	09-10-15	09-10-15	1.0
4.3	Development of use case templates and scenarios	09-11-15	09-14-15	4.0
4.4	Review of use case templates	09-15-15	09-15-15	1.0
5	REQUIREMENT ANALYSIS	09-17-15	09-25-15	8.5
5.1	Creating Use case diagrams	09-17-15	09-18-15	2.0
5.2	Creating Class diagram	09-19-15	09-20-15	2.0
5.3	Creating Object diagrams	09-21-15	09-22-15	2.0
5.4	Creating Sequence diagrams	09-23-15	09-25-15	3.0
6	SOFTWARE REQUIREMENTS	09-26-15	10-05-15	10.0
6.1	Developing user interfaces	09-26-15	09-28-15	3.0
6.2	Documentation of SRD	09-28-15	10-01-15	4.0
6.3	Review of Documentation	10-02-15	10-02-15	1.0
6.4	Writting Presentation	10-03-15	10-03-15	1.0
6.5	Final review of Documentation and Presentation	10-04-15	10-04-15	1.0
6.6	Submission of Documentation and Presentation	10-05-15	10-05-15	1.0
7	SYSTEM DESIGN	10-06-15	10-27-15	22.0
7.1	Identifying software Process model	10-06-15	10-06-15	1.0
7.2	Creating package diagream	10-07-15	10-07-15	1.0
7.3	Architectural patterns	10-08-15	10-09-15	2.0
7.4	Decomposition of Subsytem	10-10-15	10-10-15	1.0
7.5	Deployment diagram	10-12-15	10-12-15	1.0
7.6	Creating ER diagram	10-13-15	10-14-15	2.0
7.7	Identifying useful security protocols	10-15-15	10-15-15	1.0
7.8	Coding	10-16-15	10-27-15	12.0
8	DETAILED DESIGN	10-19-15	10-23-15	5.0
8.1	Creating detailed State Machine Diagrams	10-19-15	10-20-15	2.0
8.2	Creating Detailed Class Diagrams	10-21-15	10-22-15	2.0
8.3	Revision of Sequence Diagrams	10-23-15	10-23-15	1.0
9	DETAILED DESIGN DOCUMENT	10-28-15	11-05-15	9.0
9.1	Documentation of Detailed Design	10-28-15	11-02-15	6.0
9.2	Review of Detailed Design Document	11-03-15	11-04-15	2.0
9.3	Submission of Detailed Design Document and Presentation	11-05-15	11-05-15	1.0
10	TESTING	11-06-15	11-16-15	11.0
10.1	Generate Unit Test	11-06-15	11-06-15	1.0
10.2	Unit Testing	11-09-15	11-10-15	2.0
10.3	Code	11-11-15	11-11-15	1.0
10.4	System and Sub-system testing	11-12-15	11-16-15	5.0
11	FINAL DOCUMENT	11-17-15	11-30-15	14.0
11.1	Final Documentation	11-17-15	11-20-15	4.0
11.2	User Manual	11-23-15	11-24-15	2.0
11.3	Final Presentation	11-24-15	11-25-15	2.0
11.4	Review of System Document	11-26-15	11-27-15	2.0
11.5	Submission of Final Document	11-30-15	11-30-15	1.0

Fig. 11.1.1 – Gantt chart

11.2 Appendix B: All use cases with non-functional requirements.

This section lists out all the functional and non-functional requirements of the implemented use cases of Payroll Management System.

DeleteEmployee

Use Case ID: PMS_01_DeleteEmp

Use Case Level: High-level

Details:

Actor: Employer

Pre-conditions: Employer has to login into the PMS.

Description:

The Employer clicks on to "delete user" in PMS

Trigger: The Employer clicks the "Delete employee" menu item in the main page of the employer.

The system responds by …

1. The model presents the Delete Window.
2. The employer checks for the specific employee id.
3. The employer clicks on the delete button of the respective employee.
4. Employees confirms the Employee to be deleted.
5. The system displays the message that the user has been deleted.

Relevant requirements: None.

Post-conditions: Delete user window is opened.

Alternative Courses of Action: The Employer can cancel deletion of user at any point of time from step 3 to step 5.

Extensions: None

Exceptions: A deleted user cannot be retraced.

Concurrent Uses: None

Related Use Cases: PMS_02_Login

Decision Support

Frequency: The deletion may or may not occur frequently. 1 time in month

Criticality: This function is very important for using PMS.

Risk: Medium. As unauthorized users shouldn't access.

Constraints:

- **Usability:** Beginner level i.e. any kind of training is not needed.
- **Reliability:** There might be possibility of 1failure every 24 hours.
- **Performance:** The delete user doesn't have any validations unlike new user.
- **Supportability:** The client-side interface is developed using HTML, which is supported by majority of web browsers
- **Implementation:** HTML is used on front-end, and SQL on the back-end.

Modification History:

Owner: Sharan Tej Kodumuru

Initiation date: N/A

Date last modified: N/A

Login

Use Case ID: PMS_02_ login.
Details: User enters his/her username and password to log in to the system.
Actors: Employee, Employer

Pre-Conditions:

1. The user has an account.
2. The user is trying to login with their credentials.

Description:

Trigger: User tries to login to the system.
The system responds by …

1. The use case begins when the user enters username and password in the login page.
2. User then clicks on "Sign in" button in the login page.
3. PMS system authenticates username and password.
4. If username and password are authorized user is redirected to home page.

Post conditions:

1. User can use PMS system functionalities after successful login.

Alternative Courses of Action: None
Exceptions:

1. User should have a valid username and password.

Related Use Cases: PMS_19_ForgotPwd
Decision Support:

Frequency: User can login minimum once every day.
Criticality: High. This functionality is important to restrict unauthorized user from using the system.
Risk: Medium. As unauthorized users shouldn't access.

Constraints:

- **Usability:** No previous training is needed.
- **Reliability:** There might be possibility of 1failure every 24 hours
- **Performance:** At least 7 logins per day
- **Supportability:** The application will run on internet explorer, fire fox and mobile devices that supports HTML.
- **Implementation:** HTML is used on front-end, and SQL on the back-end

Modification History
Owner: Prafulla Ghadage
Initiation date: 9/7/2015
Date last modified:

Search Employee

Use Case ID: PMS_03_SearchEmp

Use Case Level: System-level End-to-end

Details:

Actor: Employer

Pre-conditions: There is existing department and employee information in the system. The employer has already logged in and the employee has a timesheet.

Description: The employer will search the Employee's timesheet information.

Trigger: The employer clicks “Search Employee” menu item.

The system responds by...

1. The employer types in the employee id.
2. The employer click “Search”.
3. The system shows the employee’s timesheets.

Post-conditions:

Alternative Courses of Action: Employer can click “Cancel” button.

Extensions: None.

Exceptions: If the input employee id is incorrect, there is an exception.

Concurrent Uses: None

Related Use Cases: None

Decision Support

Frequency: At least once in two weeks for one employer.

Criticality: This function is very important for using PMS.
Risk: Medium. As unauthorized users shouldn’t access.

Constraints:

- **Usability:** No prior training is required to use this use case**.**
- **Reliability:** System can fail once in every 24 hours.
- **Performance:** Search results should be displayed in not more than 10 seconds.
- **Supportability:** The application will run on internet explorer, fire fox and mobile devices that supports HTML
- **Implementation:** The application will be developed in Java.

Modification History --

Owner: Yao Xiao

Initiation date: 9/8

Date last modified: 9/9

Modify Timesheet

Use Case ID: PMS_04_ModifyTS

Use Case Level: System-level End-to-end

Details:

Actor: Employer

Pre-conditions: There is existing department and employee information in the system. The employer has already logged in and the employee has a timesheet.

Description: The employer will modify the Employee's timesheet information**.**

Trigger: The employer clicks "Search Employee" menu item.

The system responds by...

1. The employer searches employee by using the employee id.
2. The system shows the employee's timesheets.
3. The employer modifies the employee's information such as: In and Out time, Total Working Time, Working Type, Memo.
4. The employer clicks "Save" button.
5. The system save the Information and change the status to "Modify by Employer" of the timesheet.

Post-conditions:

Alternative Courses of Action: Employer can click "Cancel" button.

Extensions: None.

Exceptions: If the input employee id is incorrect, there is an exception.

Concurrent Uses: None

Related Use Cases: PMS_03_SearchEmp

Decision Support

Frequency: At least once in two weeks for one employer.

Criticality: This function is very important for using PMS.

Risk: Low

Constraints:

- **Usability:** Beginner level i.e. any kind of training is not needed.
- **Reliability**: System can fail once every 24 hours.
- **Performance:** Search results should be displayed in not more than 10 seconds.

  Changes should be saved in 10 seconds.

  Supportability: The application will run on internet explorer, fire fox and mobile devices that supports HTML.
- **Implementation:** The application will be developed in Java.

Modification History --

Owner: Yao Xiao

Initiation date: 9/8

Date last modified: 9/9

Approve Timesheet

Use Case ID: PMS_05_ApproveTS

Use Case Level: System-level End-to-end

Details:

Actor: Employer

Pre-conditions: There is existing department and employee information in the system. The employer has already logged in and the employee has a timesheet that has already been submitted.

Description: The employer will Approve the Employee's timesheet submit information.

Trigger: The employer clicks "Search Employee" menu item.

The system responds by...

1. The employer searches employee by using the employee id.
2. The system shows all the unapproved timesheets for the employee id that the employer had searched.

3. The employer must select one of the timesheet and clicks “View Timesheet” button.
4. The system shows the employee’s timesheet information.
5. The employer clicks “Approve” button.

6. The system change the Approval Monitor to “Approval” of the timesheet.

Post-conditions: The timesheet is forwarded towards calculating the Paycheck.

Alternative Courses of Action: Employer can click “Cancel” button.

Extensions: None.

Exceptions: If employee id is incorrect, there is an exception.

Concurrent Uses: PMS_03_SearchEmp.

Related Use Cases: PMS_04_ModifyTS

Decision Support

Frequency: At least once in two weeks for each employee.

Criticality: This function is very important for using PMS.

Risk: Low.

Constraints:

- **Usability:** No prior training is required to use this use case.
- **Reliability:** System can fail once in every 24 hours.
- **Performance:** Model should approve in not more than 10 Seconds.
- **Supportability:** The application will run on internet explorer, fire fox and mobile devices that supports HTML.
- **Implementation:** The implementation will be done in Java on Eclipse framework.

Modification History –

Owner: Yao Xiao

Initiation date: 9/8

Date last modified: 9/13

Calculate_Pay

Use Case ID: PMS_06_Cal_Pay

Use Case Level: System level

Details:

Actor: Modeler

Pre-conditions: A timesheet must have already been filled.

Description: The system will calculate the tax amount that needs to be withheld from the modeler's paycheck.

Trigger: The modeler initiates the action by filling out his timesheet and clicking "submit"

The system responds by...

1. Accepting the timesheet if it is valid.
2. Calculating the total hours times the rate of pay of the modeler.
3. Once the final amount is calculated, the system will calculate the taxes to be withheld from the modeler.
4. The system will display the total amount of taxes being withheld and the net pay of the modeler.

Relevant requirements: The modeler must submit a timesheet.

Post-conditions: The system will display the amount of taxes being withheld from the modeler's paycheck.

Alternative Courses of Action None.

Extensions: None

Exceptions: None.

Concurrent Uses: Every week when the modeler is filling out his timesheet.

Related Use Cases: PMS_05_ApproveTS

Decision Support

Frequency: Every week when the modeler is filling out his timesheet.

Criticality: High – it is a federal requirement to calculate taxes.

Risk: High – if the calculations are done incorrectly, the employee and the company could face federal prosecution.

Constraints:

- **Usability**: No training needed
- **Reliability**: 1 failure every 24 hours of operation is acceptable.
- **Performance:** The calculation shall be done within 5 seconds.
- **Supportability**: The calculation will be done using SQL functions.
- **Implementation**: The implementation will be done in MySQL.

Modification History --

Owner: Joma

Initiation date: September 7, 2015

Date last modified: September 7, 2015

GrossSalary

Use Case ID: PMS_07_GrossSal

Use Case Level: System-level end-to-end

Details:

Actor: The actor is Employee

Pre-conditions:

1. The employee should have worked and filled his time sheet.
2. The employer must have approved the timesheet.

Description: The model shows the Gross salary of the respective employee according to his time sheet per week schedule.

Trigger: The Model clicks the "Salary" menu item in the main page of the employee.

The system responds by …

1. The model presents the Salary Window.
2. The employer searches employee by the Employee id.
3. The employer clicks the Gross salary button of the respective Employee.
4. The Model calculates the Gross salary
5. The Model shows the calculated Gross salary.

Relevant requirements: According to the regulations of u.s government calculation gross salary is done.

Post-conditions: The Model is pre-designed to calculate the salary mathematically.

Alternative Courses of Action: The employee can hit PMS_08_Netsal instead of gross salary in step 2 and make a navigation.

Extensions: None.

Concurrent Uses: Can understand the process of calculating gross salary.

Related Use Cases: PMS_08_Netsal

Decision Support

Frequency: Low. Used every time to check the salary. It could be 1 time in a week.

Criticality: The main functionality of the project is to assist employee on his salary. Without this project would have no meaning.

Risk: Medium

Constraints:

- **Usability:** Will need to have information about taxes.
- **Reliability:** There might be possibility of 1failure every 24 hours
- **Performance:** Use Case will be used 6 times per 2 weeks.

- **Supportability:** The application will rely on the Java platform so it can be ported to any environment where Java can be installed.
- **Implementation:** The implementation will be done in Java on eclipse Framework.

Modification History

Owner: Sai Chaithra Allala

Initiation date: 9/8/2015

Date last modified: 9/8/2015

NetSalary

Use Case ID: PMS_08_NetSal

Use Case Level: System-level end-to-end

Details:

Actor: Employer.

Pre-conditions:

1. The employee should have worked and filled his time sheet.
2. The employer must have approved the timesheet.

Description: The model shows the Netsalary of the respective employee according to his time sheet per week schedule.

Trigger: The Model clicks the "Salary " menu item in the main page of the employer.

The system responds by …

1. The model presents the Salary Window.
2. The employer checks for the specific employee id
3. The employer clicks the Netsalary button of respective Employee.
4. The Model calculates the Netsalary.
5. The Model shows the calculated Netsalary.

Relevant requirements: According to the regulations of u.s government of calculating Netsalary

is used.

Post-conditions: The Model is pre-designed to calculate the salary mathematically.

Alternative Courses of Action: The employee can hit PMS_07_GrossSal instead of Netsalary in step 2 and make a navigation.

Extensions: None.

Concurrent Uses: Can understand the process of calculating Netsalary.

Related Use Cases: PMS_07_GrossSal

Decision Support

Frequency: Low. Used every time to check the salary. It could be 1 time in a week.

Criticality: The main functionality of the project is to assist employee on his salary. Without this project would have no meaning.

Risk: Medium

Constraints:

- **Usability:** No training needed.
- **Reliability:** There might be possibility of 1failure every 24 hours
- **Performance:** Use Case will be used 6 times per 2 Weeks.
- **Supportability:** The application will rely on the Java platform so it can be ported to any environment where Java can be installed.
- **Implementation:** The implementation will be done in Java on eclipse Framework.

Modification History --

Owner: Sai Chaithra Allala

Initiation date: 9/8/2015

Date last modified: 9/8/2015

PaystubGenerator

Use Case ID: PMS_09_PayCheck

Use Case Level: System-level end-to-end

Details:

Actors: Employee

Pre-Conditions:

1. Employee must be enrolled in payroll management system.
2. Employee should submit the Time card information to management.
3. Time cards should be approved by the management.

Description:

Trigger:

The system shall respond by:

1. The use case begins when the Employee clicks on the "View Pay Check" button in the payroll menu.
2. The system generates a list of payments to the user.
3. User clicks on view button on list to see the full salary status.

Post conditions:

1. User can click continue button to use PMS functionalities

Alternative Courses of Action: None

Exceptions:

1. In step 1, pay check is not generated if the pay check is not authorized by the manager of Employee.

Related Use Cases: PMS_002_login

Decision Support:

Frequency: Typically, the Employer can view pay check any time after each timesheet is authorized. Approximately ten times a day.

Criticality: High. This functionality is very important for an Employee to check the acceptance of timesheets.

Risk: Medium. Implementing this use case requires to check if the user have an authorized timesheets.

Constraints:

- **Usability:** No prior training is needed.

- **Reliability:** Continue button should be available to Employee at any time.
- **Performance:** The system should generate the pay check page in not more than 10 seconds.
- **Supportability:** The application will run on internet explorer, fire fox and mobile devices that supports HTML.
- **Implementation:** The implementation will be done in Java on eclipse Framework.

Modification History

Owner: Srinivasa Reddy Manda

Initiation date: 9/7/2015

Date last modified:

DirectDeposit

Use Case ID: PMS_10_DirectDep

Use Case Level: High-level

Details:

Actor: Employer

Pre-conditions: Employer has to be logged in. The timesheet of employee must be approved by the employer.

Description: The employer deposits the salary of the employees.

Trigger: The employer clicks “Direct Deposit” on menu item.

The system responds by...

1. The application shall display the page where the employee’s information about the card details are to be filled.
2. The employer types in the employee id.
3. The employer click “Search”.
4. The system shows the employee’s account details and amount to be despoited.
5. The employee clicks on “Deposit” button.
6. The system shall prompt the employer to confirm the deposit action.
7. The employer clicks “Ok” to proceed with deposit of money to his account.
8. The system shall prompt that the money has been credited to the employee

bank account.

Relevant requirements: None.

Post-conditions: The post condition of this use case is the employee gets the confirmation message and it shall navigate it to the home page.

Alternative Courses of Action: There is an alternate option 'Cancel', if the employee is not interested in a direct deposit from step 5 to 8. By clicking this button, it redirects to the homepage.

Extensions: None

Exceptions: If the data connection is lost, and the employee action might not get reflected on the server side.

Concurrent Uses: None

Related Use Cases: PMS_001_Login, PMS_08_NetSal

Decision Support

Frequency: Low – the employee can do a direct deposit only 1 in 2 weeks per Employee.

Criticality: Low - Application doesn't get effected if this feature is down.

Risk: Medium – Proper connection must be there in order to get the confirmation from the employee about the direct deposit.

Constraints:

- **Usability:** Beginner level.
- **Reliability:** System may be shut down one in 24 hours for maintenance.
- **Performance:** Application should respond in 10 seconds.
- **Supportability:** The client side interface is developed using HTML as it is widely accepted by all the browsers.
- **Implementation:** HTML is used on front-end, and SQL on the back-end.

Modification History:

Owner: Sharan Tej Kodumuru

Initiation date: 9/8

Date last modified: 9/9

AddDepartment

Use Case ID: PMS_11_AddDept

Use Case Level: System-level end-to-end

Details:

- When the employer wants to manage the Departments he can visit the Manage tab and select the Department and then add new department.
- The Employer interacts with the system using his id.
- This is done if the employer wants to add a new department.

Actor: Employer.

Pre-conditions:

The user should have a valid employee_id.

Description:

Trigger:

The employee when enters his credentials and then logs in to system he has the option to click on the manage tab where he can manage employees and departments on the webpage.

The system shall respond by…

1. The user clicks on the "manage" option
2. The system brings up the option to manage Department.
3. The user selects Employee in the option.
4. Then the user has the option to add Department.
5. The user then selects add Department.
6. There he fills the required details and then clicks the save button.

Relevant requirements:

Post-conditions: the system recognizes the employee and then system will add a new Department.

Alternative Courses of Action:

1. From step 2 to step 6 the user can hit the cancel button in both the cases.

Extensions: None

Exceptions:

1. If the Department to be added already exists.
2. If the user inputs wrong data at wrong field.

Concurrent Uses: None

Related Use Cases: None

Decision Support

Frequency: when a new Employee has to be added to the system.

Criticality: This function is very important for using PMS.

Risk: Invalid input.

Constraints:

- **Usability:** No previous training is needed. Just the employee ID.
- **Reliability:** There might be possibility of 1failure every 24 hours
- **Performance:** this use case will be used only when new employee is added.
- **Supportability:** The application will run on internet explorer, fire fox and mobile devices that supports HTML.
- **Implementation:** The implementation will be done in Java on eclipse Framework..

Use Case ID: PMS_11_AddDept

Use Case Level: Functional Sub use case.

Details:

- When the employer wants to manage the Departments he can visit the Manage tab and select the Department and then add new department.
- The Employer interacts with the system using his id.
- This is done if the employer wants to add a new department.

Actor: The actor in this case is Employer.

Pre-conditions:

- The user should have a valid employee_id.

Description:

Trigger:

The employee when enters his credentials and then logs in to system he has the option to click on the manage tab where he can manage employees and departments on the webpage

7. The user clicks on the "manage" option
8. The system brings up the option to manage Department.
9. The user selects Employee in the option.
10. Then the user has the option to add Department.
11. The user then selects add Department.
12. There he fills the required details and then clicks the save button.

Relevant requirements:

Post-conditions: the system recognizes the employee and then system will add a new Department.

Alternative Courses of Action:

- From step 2 to step 6 the user can hit the cancel button in both the cases.

Extensions:

Exceptions:

- If the Department to be added already exists.
- If the user inputs wrong data at wrong field.

Concurrent Uses: has no concurrent uses.

Related Use Cases: None

Decision Support

Frequency: when a new Employee has to be added to the system.

Criticality: This function is very important for using PMS.

Risk: Invalid input.

Constraints:

- **Usability**: No previous training is needed. Just the employee ID.
- **Reliability:** There might be possibility of 1failure every 24 hours
- **Performance:** use case is used once in 6 months.
- **Supportability:** The application will run on internet explorer, firefox and mobile devices that supports HTML.
- **Implementation:** The implementation will be done in Java on eclipse FrameWork.
- **Modification History –**

Owner: Amit Shenoy

Initiation date: 9/7/2015

Date last modified: 9/7/2015

RemoveDepartment

Use Case ID: PMS_12_RemoveDept

Use Case Level:

Details:

1. When the employer wants to manage the departments he can visit the Manage tab and select the departments and then remove department.
2. The Employer interacts with the system using his id.
3. This is done if the employer wants to remove a department.

Actor: The actor in this case is Employer.

Pre-conditions: The employee should have a valid employee_id.

Description:

Trigger: The employee when enters his credentials and then logs in to system he has the option to click on the manage tab where he can manage employees and departments on the webpage.

The system shall respond by.

1. The user clicks on the "manage" option
2. The system brings up the option to manage employee or manage department.
3. The user selects department in the option.
4. Then the user has the option to remove department.
5. The user then selects remove department.
6. There he selects the department to be removed and then clicks the save button.

Relevant requirements: None

Post-conditions: The system recognizes the employee and then system will remove the selected department.

Alternative Courses of Action: From step 2 to step 6 the user can hit the cancel button in both the cases.

Extensions : None

Exceptions: If the department to be removed doesn't exists.

Concurrent Uses: None.

Related Use Cases: None.

Decision Support

Frequency: when a department has to be removed to the system.

Criticality: This function is very important for using PMS.

Risk: Invalid input.

Constraints:

- **Usability:** No previous training is needed. Just the employee ID.
- **Reliability:** There might be possibility of 1failure every 24 hours
- **Performance:** use case is used once a year.
- **Supportability:** The application will run on internet explorer, Firefox and mobile devices that supports HTML.
- **Implementation:** The implementation will be done in Java on eclipse Framework.

Modification History --

Owner: Amit Shenoy

Initiation date: 9/7/2015
Date last modified: 9/7/2015

Add Employee

Use Case ID: PMS_13_AddEmployee

Use Case Level:

Details:

1. When the employer wants to manage the Employees he can visit the Manage tab and select the Employee and then add new Employee to the department
2. The Employer interacts with the system using his id.
3. This is done if the employer wants to add a new Employee.

Actor: The actor in this case is Employer.

Pre-conditions: The employee should have a valid employee_id.

Description:

Trigger: The employee when enters his credentials and then logs in to system he has the Option to click on the manage tab where he can manage employees and Employees on the webpage.

The system shall respond by.

1. The user clicks on the "manage" option
2. The system brings up the option to manage Employee.
3. The user selects Employee in the option.
4. Then the user has the option to add Employee.
5. The user then selects add Employee.
6. There he fills the required details and then clicks the save button.

Relevant requirements: None

Post-conditions: The system recognizes the employee and then system will add a new Employee.

Alternative Courses of Action: From step 2 to step 6 the user can hit the cancel button in both the cases.

Extensions: None

Exceptions:

1. If the Employee to be added already exists.
2. If the user inputs wrong data at wrong field.

Concurrent Uses: None

Related Use Cases: None

Decision Support

Frequency: when a new Employee has to be added to the system.

Criticality: This function is very important for using PMS.

Risk: Invalid input.

Constraints:

- **Usability:** No previous training is needed. Just the employee ID.
- **Reliability:** There might be possibility of 1failure every 24 hours
- **Performance:** use case used once a month
- **Supportability:** The application will run on internet explorer, fire fox and mobile devices that supports HTML.
- **Implementation:** The implementation will be done in Java on eclipse Framework.

Modification History –

Owner: Amit Shenoy

Initiation date: 9/7/2015

Date last modified: 9/7/2015

UpdateEmployer

Use Case ID: PMS_14_UpdateEmployer

Use Case Level: System-Level End-to-End

Details:

Actor: Modeler

Pre-conditions: Modeler has the employee profile already open and filled with information. There is existing info that he wants to modify.

Description: The modeler will change the information in the profile of the employee.

Trigger: The modeler clicks the "Update Employee Information" on the page.

The system responds by...

1. The modeler is redirected to the employee profile in edit mode where information can be edited.
2. The modeler changes the information that needs to be updated by using the input dialog boxes where the information is written.
3. The modeler clicks Save when he is done changing all the information that needs to be updated.
4. The modeler is redirected to the employee screen where all the updated info will be displayed.

Relevant requirements: None of the fields can be saved with invalid information (ex. Text in the phone number area.

Post-conditions: The information is updated in the correct table in the database.

Alternative Courses of Action The modeler can hit the cancel button in the page and no changes will be made. It will return the modeler to the previous screen where the original employee information will be displayed with no changes.

Extensions: None.

Exceptions: The modeler cannot input invalid information on any of the fields.

Concurrent Uses: None

Related Use Cases: PMS_16_Employee

Decision Support

Frequency: Occasional – Performed only when the modeler needs to update information of the employee (only one time whenever a change needs to be made).

Criticality: High – The modeler needs to be able to change information of the employee as some of the information is critically important such as bank information in order for the employee to get paid.

Risk: Low

Constraints:

- **Usability**: No training needed
- **Reliability**: 1 failure every month is acceptable
- **Performance**: The information must be updated within 15 seconds
- **Supportability**: The tasks will be executed with SQL and updated using update queries
- **Implementation:** The implementation will be carried out in MySQL

Modification History –

Owner: Joma

Initiation date: Sept 7, 2015

Date last modified: Sept 7, 2015

RemoveEmployee

Use Case ID: PMS_15_RemoveEmployee

Use Case Level:

Details:

1. When the employer wants to manage the employees he can visit the Manage tab and select the employees and then remove employee from the department.
2. The Employer interacts with the system using his id.
3. This is done if the employer wants to remove an employee from the department.

Actor: The actor in this case is Employer.

Pre-conditions: The employee should have a valid employee_id.

Description:

Trigger: The employee when enters his credentials and then logs in to system he has the option to click on the manage tab where he can manage employees on the webpage.

The system shall respond by.

1. The employee clicks on the "manage" option
2. The system brings up the option to manage employee or manage department.
3. The employee selects manage employee in the option.
4. Then the employee has the option to remove employee.
5. The employee then selects remove employee.
6. There he selects the employee to be removed from the department and then clicks the save button.

Relevant requirements: None

Post-conditions: The system recognizes the employee and then system will remove the selected employee.

Alternative Courses of Action: From step 2 to step 6 the employee can hit the cancel button in both the cases.

Extensions: None

Exceptions: If the employee to be removed doesn't exists.

Concurrent Uses: None.

Related Use Cases: None

Decision Support

Frequency: when an employee has to be removed to the system.

Criticality: high

Risk: Invalid input.

Constraints:

- **Usability:** No previous training is needed. Just the employee ID.
- **Reliability:** May shutdown once in 14hrs
- **Performance:** Removes user with in 10seconds.
- **Supportability:** The application will run on internet explorer, fire fox and mobile devices that supports HTML.
- **Implementation:** The implementation will be done in Java on eclipse Framework.

Modification History --

Owner: Amit Shenoy

Initiation date: 9/7/2015

Date last modified: 9/7/2015

UpdateEmployee

Use Case ID: PMS_16_UpdateEmployee

Use Case Level: System-Level End-to-End

Details:

Actor: Modeler

Pre-conditions: Modeler has the profile section open. There is data already filled in his profile which he wishes to modify.

Description: The modeler will edit the existing data and will save the changes on the database.

Trigger: The modeler clicks the "Edit information" button on the screen

The system responds by...

1. It takes the modeler to an edit screen where the information is transferred to text boxes.
2. The modeler then edits the desired information he wishes to change on the certain field pertaining to that value.
3. The modeler clicks the "Save Changes" button.
4. The modeler is redirected back to the previous page where it displays the updated information.

Relevant requirements: The modeler must already have had a profile with information filled in.

Post-conditions: The employee profile will be updated with the new information and the tables in the database will be updated.

Alternative Courses of Action The modeler can hit the "Cancel" button in the edit window and the it will be redirected the previous screen with none of the information being changed.

Extensions: None.

Exceptions: The modeler could enter information that is invalid. (Ex. Letters for a phone number)

Concurrent Uses: None

Related Use Cases: PMS_18_NewEmployee

Decision Support

Frequency: Occasional – Performed only when the modeler needs to update information on his profile.

Criticality: High – Holds information such as bank account number in order for the modeler to get paid.

Risk: Low.

Constraints:

- **Usability**: No training needed.
- **Reliability**: 1 failure every month is acceptable.
- **Performance:** The data should be updated within 15 seconds.
- **Supportability:** The application will be hosted on an SQL database and will be done executing SQL queries.
- **Implementation:** The implementation will be carried out in MySQL.

Modification History --

Owner: Joma

Initiation date: September 7, 2015

Date last modified: September 7, 2015

Work Profile

Use Case ID: PMS_17_WorkProfile

Use Case Level: System-Level End-to-End

Details:

Actor: Employer

Pre-conditions: Employer has the employee profile already filled with information. All the information of every employee is shown here.

Description: The modeler will display the information of the employee.

Trigger: The modeler clicks the "Work Profile" on the page.

The system responds by...

1. The modeler is redirected to list of all employees in the modeler.
2. The modeler clicks the employee's profile from the list.
3. The modeler is redirected to the employee screen with all the info is displayed.

Relevant requirements: None of the fields can be saved with invalid information (ex. Text)

Post-conditions: The information is updated in retrieved from the correct table in the database.

Alternative Courses of Action: The modeler can hit the back button in the page, it will return the modeler to the previous screen where the list of all employees is shown.

Extensions: None.

Exceptions: The modeler cannot input new information of the employee.

Concurrent Uses: This use can occur concurrently with the uses listed in this section.}

Related Use Cases: PMS_16_updateEmployee, PMS_18_NewEmployee

Decision Support

Frequency: Performed only when the modeler needs to check information of the employee. 1 time in a week.

Criticality: Low.

Risk: Low.

Constraints:

- **Usability**: No training needed
- **Reliability**: 1 failure every month is acceptable
- **Performance**: The information must be retrived within 15 seconds
- **Supportability**: The tasks will be executed with SQL and updated using update queries
- **Implementation**: The implementation will be carried out in MySQL

Modification History –

Name: Sai Chaitra

Initiation date: Sept 14, 2015

Date last modified: Sept 14, 2015

New Employee

Use Case ID: PMS_18_NewEMployee

Use Case Level: High-level

Details:

Actor: Employer

Pre-conditions: Employer has to login into the PMS.

Description:

The Employer clicks on to "New user" in PMS

Trigger: The system shall respond by..

1. New user window is opened.
2. The page shall display a sign up box where the user can enter the following information into textbox fields: first name, last name, Email, email confirmation, password and account details all the necessary data.
3. The Employer clicks on the "Create New user" button.
4. If the validation succeeds.
5. The system displays a message confirmation the creation of a user account profile.

Relevant requirements: None.

Post-conditions: The Employer can cancel creation of new user at any point of time from step 2 to step 5.

Alternative Courses of Action: If the actor provides a username which is already existed in the database, the server shall prompt to enter a new username.

Extensions: None

Exceptions:

- The new user cannot leave without filling the required text fields.
- User enters password which does not meet minimum requirements.

Concurrent Uses: None

Related Use Cases: PMS_02_Login

Decision Support

Frequency: The users create their profile only once. So the frequency of this is low.

Criticality: Low

Risk: The user data must be securely stored and then the communication with server is

very important. There is medium level risk in this.

Constraints:

- **Usability:** Beginner level i.e. any kind of training is not needed.
- **Reliability:** There might be possibility of 1failure every 24 hours.
- **Performance:** The new user can complete his registration process very quickly.
- **Supportability:** The client-side interface is developed using HTML, which is supported by majority of web browsers
- **Implementation:** HTML is used on front-end, and SQL on the back-end.

Modification History:

Owner: Sharan Tej Kodumuru

Initiation date: N/A

Date last modified: N/A

Forgot Password

Use Case ID: PMS_19_ForgotPwd

Use Case Level:

Details:

1. During the Employee/ Employer login when he/she forgets the password for his/her account and wants a auto generated password from the system
2. The Employee/ Employer interacts with the system using his id.
3. This is done if the user forgets his/her password.

Actor: the actor in this case is Employee/Employer.

Pre-conditions:

1. The user should have a valid employee_id.
2. The user should have answered the security questions.

Description:

When employee/employer forgets his/her password.

During login when the user enters a valid id and forgets his password and he/she needs to login in the system by getting the password from the system which is automatically generated.

Trigger:

The user when enters a valid user id and clicks on the "forget password?" option on the web page.

1. The user clicks on the "forget password?" option
2. The system brings up the security question which has been already stored in the database.
3. The user answers the security question, if the answer is correct the system takes the user to next step else it logs out of the system.
4. if the answer is correct the system automatically generates a password and mails it to the user on the valid email id provided by the user during account creation.

Relevant requirements:

Post-conditions: the system will recognise the user from the system data base and then system will send an automated email with new password to the user.

Alternative Courses of Action: From step 2 to step 4 the user can hit the cancel button in both the cases.

Extensions : This use case is extension to the login use case.

Exceptions:

- If the user does not show up on the database.
- If the user inputs invalid answer of the security question.

Concurrent Uses: has no concurrent uses.

Related Use Cases: PMS_02_Login

Decision Support

Frequency: when user forgets his/her password.

Criticality: This function is very important for using PMS.

Risk: low

Constraints:

- **Usability:** No previous training is needed. Just the employee ID.
- **Reliability:** There might be possibility of 1failure every 24 hours
- **Performance:** The Use case will be used once in a month
- **Supportability**: The application will run on internet explorer, firefox and mobile devices that supports HTML.
- **Implementation:** The implementation will be done in Java on eclipse FrameWork.

Modification History --

Owner: Prafulla Ghadage.

Initiation date: 9/7/2015

Date last modified: 9/7/2015

PasswordNotifier

Use Case ID: PMS_20_PwdNotifier

Use Case Level: functional sub-use case

Details:

Actor: The actor is Employee

Pre-conditions:The employee should have an already existing password.

Description: Notification for the employee for a change in password via email.

Trigger: The Employee initiates an action by selecting Update password.

1. The system responds by asking security question.
2. Employee answers the Question.
3. Employee Now has to enter the Old password.
4. Employee enters the new password.
5. Employee clicks save and continue
6. System send an email to the employee notifying the change in the password.

Relevant requirements: None

Alternative Courses of Action: The employee can stop changing the password at any point of time from step1 to step6 and hit cancel button.

Extensions: Nothing more to extend.

Exceptions: Same old password given to new password.

Concurrent Uses: Helps in knowing unknown password change.

Related Use Cases: None

Decision Support

Frequency: Used very rarely maybe 2 times in 30 days.

Criticality: Implemeting this increases the secuirty for the user and helps to alarm on someone else changing password.

Risk: Medium

Constraints:

- **Usability:** Employee must know that this option exists.
- **Reliability:** There might be possibility of 1failure every 24 hours
- **Performance:** Once every 2 months.
- **Supportability**: The application will run on internet explorer, firefox and mobile devices that supports HTML..
- **Implementation:** The implementation will be done in Java on eclipse FrameWork.

Modification History --

Owner: Sai Chaithra Allala

Initiation date: 9/8/2015

Date last modified: 9/8/2015

Logout

Use Case ID: PMS_21_Logout.

Details:

- User clicks on logout button to logout of the system.

Actors: Employee, Employer

Pre-Conditions:

1. The User is already logged in.

Description:

Trigger: User clicks on logout button in the "home" page.

The system responds by …

1. The use case begins when the user clicks logout in the home page.
2. User is logged out of the system.
3. A message is shown to the user that he is logged out.

Post conditions: User is redirected to login page.

Alternative Courses of Action: None

Exceptions: None

Related Use Cases:

Decision Support:

Frequency: Used very every time to logout.
Criticality: This function is very important for using PMS.
Risk: Low

Constraints:

Usability: No previous training is needed.

Reliability: There might be possibility of 1failure every 24 hours.

Performance: Use case will be used at least 7 times per day.

Supportability: The application will run on internet explorer, fire fox and mobile devices that supports HTML.

Implementation: The implementation will be done in Java on eclipse Frame Work.

Modification History

Owner: Srinivasa Reddy Manda
Initiation date: 9/7/2015

Date last modified:

9.2.2Misuse case descriptions.

Misuse Case Template

Use Case ID: PMS_101_Injector

Use Case Level:

Actor: Misuser

Pre-conditions: Misuser should be on the Timesheet web page of the PM system.

Security Threat: The misuser tries to enter a sql statement at the field where he is supposed to put regular data.

Trigger: Misuser enters the sql statement in the timesheet time slot.

The system responds by:

1) The System gives a error saying “please enter valid Information”

Post Condition: The system throw a error message to the misuser.

Alternative course of action: Misuser can hit the cancel button anytime.

Extensions: None

Exceptions: None

Related Use Cases: None

Use Case Template

Use Case ID: PMS_001_Injector

Use Case Level: High-level, system-end

Details: SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution. This use case instead to protect the system from SQL injections.

Actor: Modeler

Pre-conditions: The modeler has the website open on a page where the application can take typed input. The modeler has bad intentions.

Description: The modeler will try to inject SQL statements like dropping a table.

Trigger: The user initiates an action by writing a SQL query on a text field and submitting the form in order to execute the SQL command.

The system responds by...

1. Not allowing invalid inputs from being sent
2. Asking the user to fix his input

Relevant requirements: Field validation

Post-conditions: The user is returned to the input data screen where a label will indicate that the input sent is invalid and needs to be changed.

Alternative Courses of Action None.

Extensions: None.

Exceptions: None.

Concurrent Uses: None

Related Use Cases: None

Decision Support

Frequency: This will occur every time an user input invalid data into a text field.

Criticality: Very high – directly responsible for security agaisnt SQL injection

Risk: Very high – the whole system could be compromised if this is to fail.

Constraints:

- **Usability:** No training needed.
- **Reliability:** This feature cannot ever fail.
- **Performance:** The system shall detect invalid input within 2 seconds.

- **Supportability:** This will be implemented using HTML validation
- **Implementation:** This will be implemented using HTML websites

Modification History --

Owner: Joma

Initiation date: September 7, 2015

Date last modified: September 7, 2015

MisUse Case Template

Use Case ID: PMS_103_Multi.Login

Use Case Level: System Level end to end

Actor: Misuser(Employee, Employer)

Pre-conditions:

1. The Misuser has an account.
2. The Misuser is trying to login with their credentials.

Security Threat: The misuser tries login in to their accounts using different Web Browsing Windows.

Trigger: MisUser tries to login to the system.

The system responds by …

1. When the Misuser logs in to the system by his/her credentials.
2. The System Checks if the Flag has already been set for the respective Account.
3. If the Flag is not set then the Misuser gains access to the Account but if the flag is set the system would pop up the error message.

Post Condition:

Error massage is popped up.

Alternative course of action: None

Extensions: None

Exceptions: None

Related Use Cases: PMS_02_Login

NoMultipleLogins

Use Case ID: PMS_003_Multi.login.

Details:

- A flag is stored in database whenever a user logs in to the system.
- User is restricted to login when the flag is set.

Actors: Employee, Employer

Pre-Conditions:

1. The user has an account.
2. The user is trying to login with their credentials.

Description:

Once the user is logged in to the system, access to the system from different browser or other computer is strictly prohibited by storing the flag of first logged in browser and checked against it for every request to the system.

Trigger: User tries to login to the system.

The system responds by …

1. The use case begins when the user logs in to the system by his/her credentials.
2. After successful login, a flag is stored in database until the user logs out.
3. If user tries to login, system checks if there is an existing flag in the database.
4. If there is an existing flag system doesn't allow user to login.

Post conditions:

1. User can use system functionalities after successful login.

Alternative Courses of Action:

1. From step 3 to step 4 user can logout from his current browser and login with different browser.

Exceptions:

1. User should logout after using the system.
2. User shouldn't exit the browser without successful logout.

Related Use Cases:

1. PMS_02_Login
2. PMS_21_Logout

Decision Support

Frequency: when user forgets his/her password.

Criticality: high

Risk: Invalid input.

Constraints:

- **Usability:** No previous training is needed. Just the employee ID.
- **Reliability:** This feature can fail once every 2 weeks
- **Performance:** The system shall detect invalid input within 2 seconds.
- **Supportability:** This will be implemented using HTML validation
- **Implementation:** the implementation to be done on eclipse.

Modification History –

Owner: Prafulla Ghadage.

Initiation date: 9/7/2015

Date last modified: 9/7/2015

MisUse Case Template

Use Case ID: PMS_104_duplicate

Use Case Level:

Actor: Misuser (Employee)

Pre-conditions: The Misuser has an account.

The Misuser is logged into the system.

The Misuser has filled the timesheet.

Security Threat: The misuser tries to re-submit the already submitted timesheet.

Trigger: MisUser tries to submit timesheet form to the system.

The system responds by …

1. The misuser clicks on "submit timesheet" button.
2. A transaction id flag is stored in user database for two weeks.
3. Form values are submitted.
4. The Misuser tries to resubmit the form in two weeks.
5. The PMS system checks if there is a transaction id flag set in the System.
6. If there is a transaction id flag present in database, the submission is declined.

Post Condition:

The Misuser is shown an alert message saying "Timesheet is already submitted".

Alternative course of action:None

Extensions: None

Exceptions: None

Related Use Cases: PMS_05_ApproveTS

Avoid duplicate submissions.

Use Case ID: PMS_004_duplicate

Details: Employee is restricted to submit timesheets form twice.

Actors: Employee

Pre-Conditions:

1. The employee has an account.
2. The employee is logged into the system.
3. The employee filled the timesheets.

Description:

After submission of timesheets a transaction id is stored in employee database for two weeks. Submission of timesheets made by the employee anytime in two weeks is declined.

Trigger: Employee tries to submit timesheets form to the system.

The system responds by …

1. The use case begins when the employee clicks on “submit timesheets” button.
2. A transaction id flag is stored in employee database for two weeks.
3. Form values are submitted to database.
4. The employee tries to resubmit the form in two weeks.
5. The PMS system checks if there is a transaction id flag in the database.
6. If there is a transaction id flag present in database, the submission is declined.
7. The employee is shown an alert message saying “Timesheets already submitted”.

Post conditions:

1. Transaction id is removed from the database after two weeks.
2. Employee can submit new timesheets after two weeks.

Alternative Courses of Action:

From step 1 to step 7 the employee can click “cancel” button to cancel the submit timesheets.

Exceptions: None

Related Use Cases: PMS_05_ApproveTS

Decision Support:

Frequency: Medium. The employee can submit timesheets not more than twice a week.

Criticality: High. This functionality is important to restrict employee from resubmitting forms to the system when a transaction is in progress.

Risk: Medium

Constraints:

Usability: No prior training is needed.

- **Reliability:** This feature can fail once every 2 weeks
- **Performance:** The system shall detect invalid input within 2 seconds.
- **Supportability:** This will be implemented using HTML validation
- **Implementation:** the implementation to be done on eclipse.

Modification History

Owner: Srinivasa Reddy Manda

Initiation date: 9/7/2015

Date last modified: 9/7/2015

Mis Use Case Template

Use Case ID: PMS_105_security

Use Case Level: System level end to end

- **Actor: Misuser**
- **Pre-conditions:** Misuser should be on the login web page of the PM system.
- **Security Threat:** The misuser tries to get unauthorized entry in to the system by clicking on the forget password link and then by answering the security questions.
- **Trigger:** Misuer enters the employee_id and does not have a password to get entry in to account, so he clicks on the forget password link and tries to get unauthorized entry in to the PM system.

  The system responds by:

  1. System takes the misuser to the security questions page.
  2. Misuser answers all the security question.
  3. The system checks the answers provided by the misuser and then system finds out that the answers are incorrect.
  4. The system then takes the misuser back to the login page.

- **Post Condition:**

  The system takes the misuser to the login page.

- **Alternative course of action:**
- **Extensions:** None
- **Exceptions:** None
- **Related Use Cases:** None

Security

Use Case ID: PMS_005_security

Use Case Level:

Details:

- During the Employee/ Employer login when he/she forgets the password for his/her account and wants an auto generated password from the system
- When the user wants to change the auto generated password to the password of his liking.
- The Employee/ Employer interacts with the system using his id.
- This is done if the user forgets his/her password or wants to change his/her password for their personal reasons.

Actor: the actor in this case is Employee/Employer.

Pre-conditions:

- The user should have a valid employee_id in both the cases.
- The users should be authenticated employee/employer of the system.
- The user should have successfully logged in to the system when he/she wants to change the password.
- The user should have already answered the security questions.

Description:

There are two cases when the security question arises.

a) When the authenticated employee/employer wants to change the password.

After user has successfully logged in to the system and feels the need to change the password of his account.

Trigger:

The user clicks the "Profile" menu item provided in the system user interface.

The system responds by …

1. The user clicks on the "Profile" menu and selects the "change password" option from the dropdown menu.

2. The system brings up the security question which has been already stored in the database.

3. The user answers the security question, if the answer is correct the system takes the user to next step else it logs out of the system.

4. If the answer is correct the next window to change the password comes up.

b) When employee/employer forgets his/her password.

During login when the user enters a valid id and forgets his password and he/she needs to Login in the system by getting the password from the system which is automatically generated.

Trigger:

The user when enters a valid user id and clicks on the "forgot password?" option on the web page.

1. The user clicks on the "forgot password?" option
2. The system brings up the security question which has been already stored in the Database.
3. The user answers the security question, if the answer is correct the system takes the user to next step else it logs out of the system.
4. If the answer is correct the system automatically generates a password and mails it to the user on the valid email id provided by the user during account creation.

Relevant requirements:

Post-conditions: the system will recognize the user from the system data base and in the first case will take the user to the window where user can change his password and in the second case system will generated an automated email with new password.

Alternative Courses of Action:

- From step 2 to step 4 the user can hit the cancel button in both the cases.

Extensions: This use case is extension to the login use case and change_password use case.

Exceptions:

- If the user does not show up on the database.
- If the user inputs invalid answer of the security question.

Concurrent Uses: This use case can be used concurrently during the PMS_002_login and the PMS_19_Forgotpwd use cases.

Related Use Cases: this use case is used during the PMS_002_login use case when user forgets his/her password or it is invoked when the user wants to change his password.

Decision Support

Frequency: as long as the user wants to change his password or when user forgets his/her Password.

Criticality: High. This functionality is important to restrict employee from resubmitting forms to the system when a transaction is in progress.

Risk: Invalid input.

Constraints:

- **Usability:** No previous training is needed. Just the employee ID.
- **Reliability:** This feature can fail once every 2 weeks
- **Performance:** The system shall detect invalid input within 2 seconds.
- **Supportability:** This will be implemented using HTML validation
- **Implementation:** the implementation to be done on eclipse.
-

Modification History –

Owner: Prafulla Ghadage.

Initiation date: 9/7/2015

Date last modified: 9/7/2015

MisUse Case Template

Use Case ID: PMS_106_timeout

Use Case Level: High-level

Actor: Misuser

Pre-conditions:

The employee has logged in into his account and there is active connection between the application and the server.

The employee should be inactive keeping the application open.

Security Threat: If the Employee has left their Account logged in the Misuser can get easy access to their account and can change the information.

Trigger:

Use case begins when employee is logged into his account and is on his profile page.

1. The system timer will be initialized as soon as the employee logs in.
2. When the application remains inactive for 1 minute, the employee is notified about the time out.
3. The employee is prompted if he wants to continue or not.
4. If there is no response to the notification in 10 seconds, the system time is out.
5. The employee is logged out and is redirected to the default home page of the application.
6. The use case ends when the user again reaches authentication step.

Post Condition: Misuser Cannot Access the Account.

□□**Alternative course of action:** None

□□**Extensions:** None

□□**Exceptions:** None

□□**Related Use Cases:** None

SessionTimeout

Use Case ID: PMS_006_Timeout

Use Case Level: High-level

Details:

Actor: Employee, payroll server.

Pre-Conditions:

I. The employee has logged in into his account and there is active connection between the application and the server.

II. The employee should be inactive keeping the application open. **Description:**

Trigger:

I. Use case begins when employee is logged into his account and is on his profile page.

II. The system timer will be initialized as soon as the employee logs in.

III. When the application remains inactive for 1 minute, the employee is notified about the time out.

IV. The employee is prompted if he wants to continue or not.

V. If there is no response to the notification in 10 seconds, the system time is out.

VI. The employee is logged out and is redirected to the default home page of the application

VII. The use case ends when the user again reaches authentication step

Relevant requirements: The employee should have a proper communication with the server.

Post- conditions:

1. The employee is successfully logged out, so if he wants to access his account again he has to re-login.
2. Any changes made are cancelled and are not applied to the profile.

Alternative courses of Action: The user can choose to cancel the communication by clicking cancel if he is unsure of what he wants to do next and save the changes he has made to the system from step 4 to step 8.

Extensions: None

Exceptions: No communication with the server.

Concurrent Uses: None

Related Use Cases: PMS_21_Logout.

Decision support

Frequency: Many times a day. Probably, around 100 times.

Criticality: Medium – If this function stops working, the application still runs. But, the security level might be less.

Risk: High - The employee may lose the data which he is working on.

Constraints:

- **Usability**: No previous training is required since it this feature depends on the System itself.
- **Reliability:** When active, this feature can cause loss of data.
- **Performance:** Depends on system timer.
- **Supportability:** This can be implemented using Java platform, which is platform dependent.
- **Implementation:** Using java based eclipse tool.

Modification History:

Owner: Sharan Tej

Initiation date: 09/09/2015

Date Last Modified: N/A

Security Use Case

Use Case ID: PMS_007_texTolabel

Use Case Level: System-level End-to-end

Details:

Actor: Employer

Pre-conditions: There is existing department and employee information in the system. The employer has already logged in and the employee has a timesheet that already been approved.

Description: The system will change the input part in Employee's timesheet become unchangeable.

Trigger: The employer clicks "Approve TS" button.

The system responds by...

1. The system change input part in Employee's timesheet become unchangeable.
2. The system changes the status of timesheet to "Approved".

Post-conditions:

Alternative Courses of Action: Employer can click "Cancel" button.

Extensions: None.

Exceptions: If the input employee id is incorrect, there is an exception.

Concurrent Uses: None

Related Use Cases: PMS_05_ApproveTS

Decision Support

Frequency: Many times a day. Probably, around 100 times.

Criticality: Medium – If this function stops working, the application still runs. But, the security level might be less.

Risk: High - The employee may lose the data which he is working on.

Constraints:

- **Usability:**
- **Reliability:** This feature can fail once every 2 weeks
- **Performance:** The system shall detect invalid input within 2 seconds.
- **Supportability:** This will be implemented using HTML validation
- **Implementation:** the implementation to be done on eclipse.

Modification History --

Owner: Yao Xiao

Initiation date: 9/8

Date last modified: 9/9

Scenario:

PMS_007_texTolabel

Actors: Maria (Employer)

Pre-conditions:

1. Maria logs in with her id and password.
2. Tom has already submitted his timesheet.
3. Maria has clicked ApproveTS button.

Description:

The system will change the input part in Tom's timesheet become

unchangeable.

Trigger: Maria clicks "ApproveTS" button in EMS.

The system responds by …

1. The system change input part in Employee's timesheet become unchangeable.
2. The system changes the status of Tom's timesheet to "Approved".

Post-conditions: None

MisUse Case Template

Use Case ID: PMS_102_Captcha

Use Case Level:

□□**Actor: Misuser**

□□**Pre-conditions:** Misuser should be on the Timesheet web page of the PM system.

□□**Security Threat:** The misuser tries an automated tool to fill in false values and hit submit button for the system to get overflowed with false data.

□□**Trigger:** Misuer runs the automated program over the web browser for it to fill in false timesheets and hit submit continuously.

The system responds by:

1. System prompts the user to enter the text displayed.
2. The misuser has to manually look into the label and enter the text carefully.
3. The system checks the text entered by the user.
4. The system then accepts the timesheet.

Post Condition:

The system accepts the timesheet.

Alternative course of action:

Misuser can hit the cancel button anytime from step 1 to step 4.

Extensions: None

Exceptions:

Related Use Cases:

Use Case Template

Use Case ID: PMS_002_Captcha

Use Case Level: System end to end level

Details:

- When the user fills in the timesheet before submitting he is asked to fill in a text box called captcha.
- The Employee/ Employer interacts with the system using his id.
- This is done to verify if the system is being used by a human and not a computer.

Actor: the actor in this case is Employee/Employer.

Pre-conditions:

- The user should have logged in successfully in the system.
- The users should be on the timesheet page and should have filled the timesheet.

Description:

Trigger:

1. The user fills his timesheet info.
2. The user wants to submit the timesheet info which he has filled.
3. The system asks user to enter text displayed on the label.
4. The user fills in the info on the textbox and hits submit.
5. The system checks the text and submits the Timesheet.

Relevant requirements:

Post-conditions: the system will recognize the user as a human and will submit his timesheet.

Alternative Courses of Action:

- From step 1 to step 4 the user can hit the cancel button in both the cases.

Extensions: None.

Exceptions: None.

Concurrent Uses: This use case can be used concurrently during the login and the change_password use cases.

Related Use Cases: None.

Decision Support

Frequency: Many times a day. Probably, around 100 times.

Criticality: Medium – If this function stops working, the application still runs. But, the security level might be less.

Risk: High - The employee may lose the data which he is working on.

Constraints:

- **Usability:** No previous training is needed. Just the employee ID.
- **Reliability:** This feature can fail once every 2 weeks
- **Performance:** The system shall detect invalid input within 2 seconds.
- **Supportability:** This will be implemented using HTML validation
- **Implementation:** the implementation to be done on eclipse.

Modification History -- *{Follow the standard corporate document versioning template}*

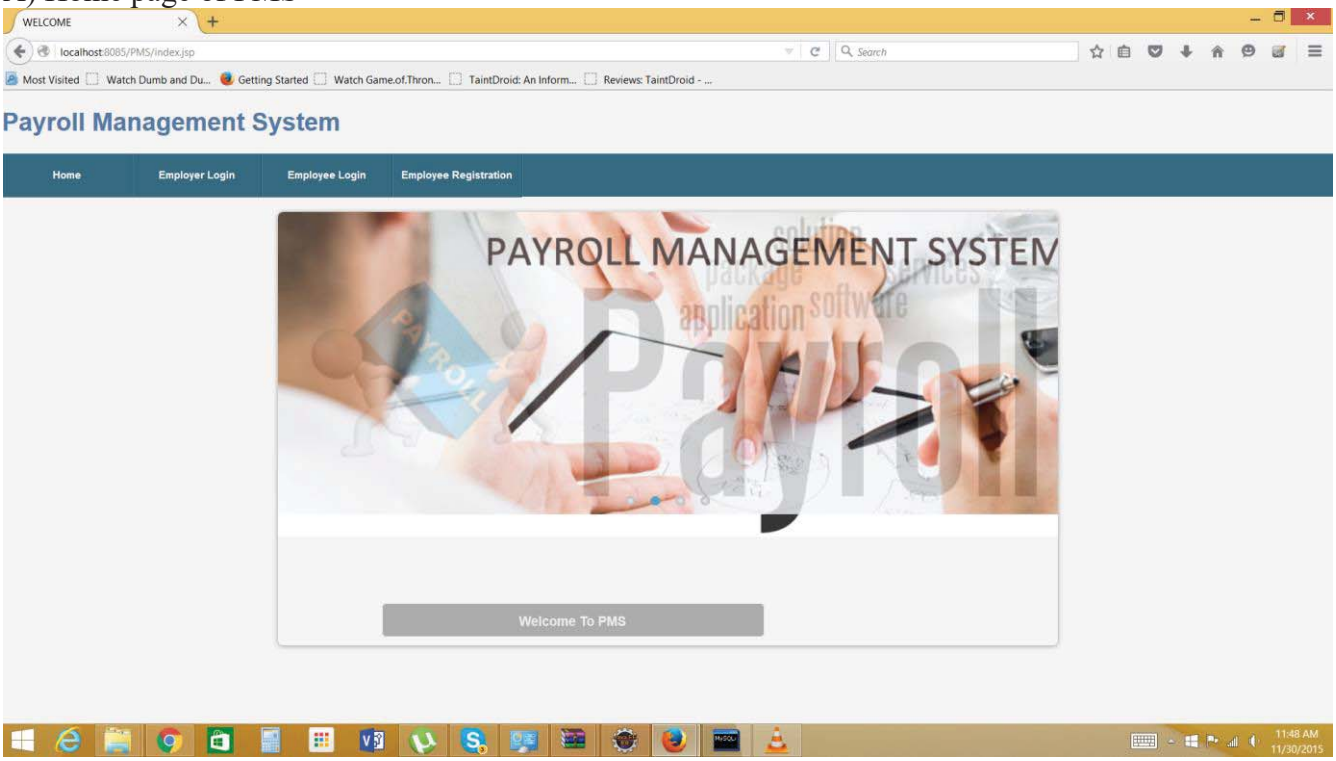
Owner: Amit Shenoy.

Initiation date: 9/7/2015

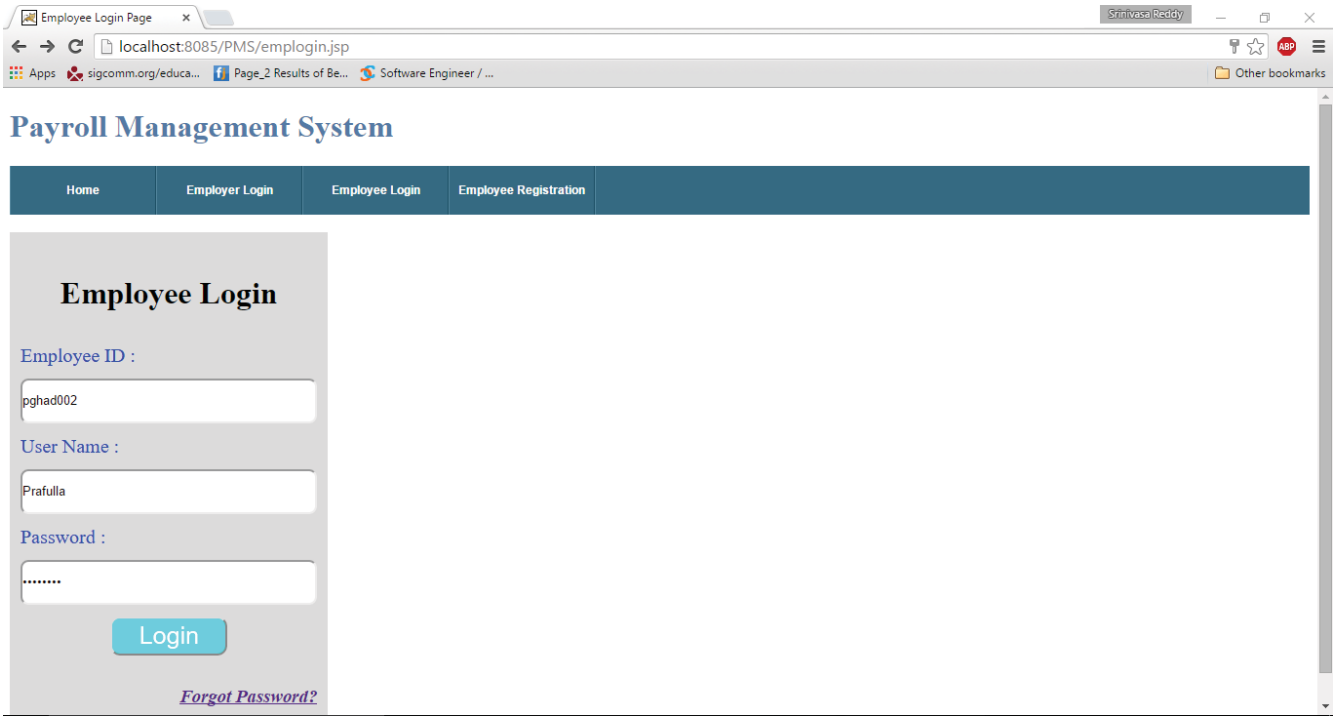
Date last modified: 9/7/2015

11.3 Appendix C – All User Interface Diagrams.

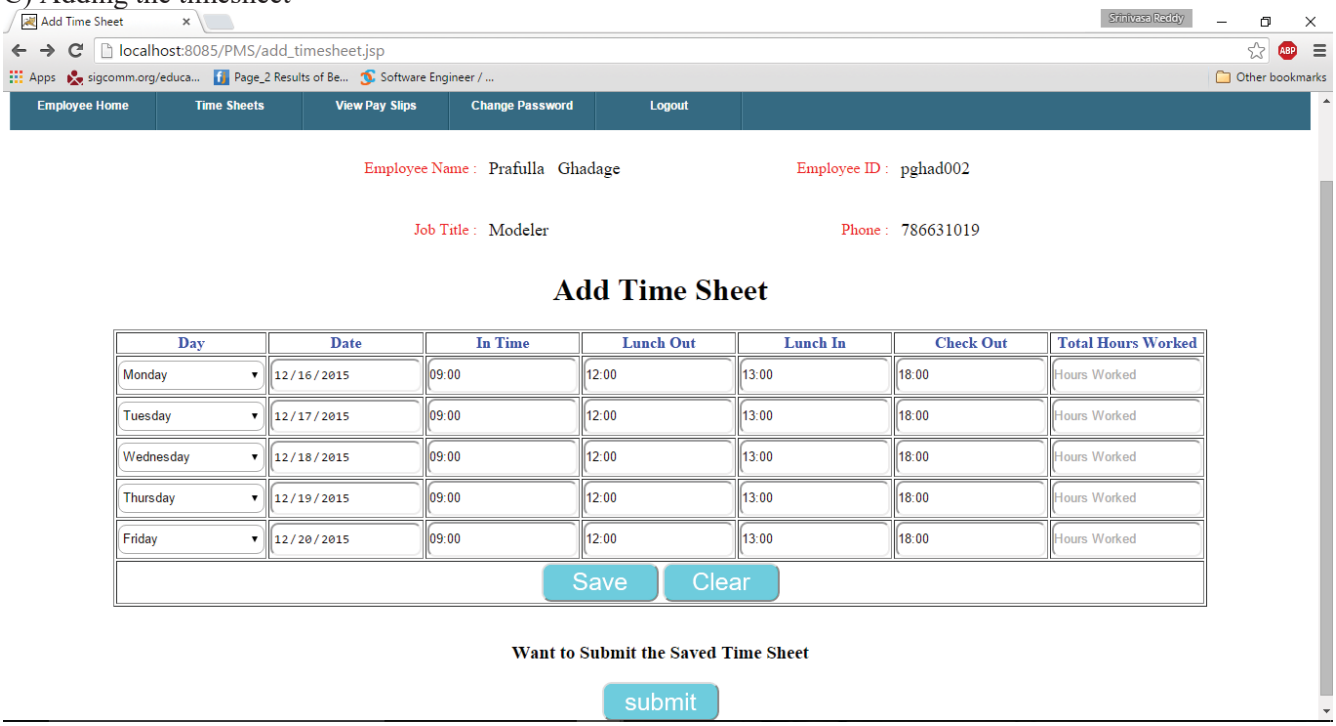
A) Home page of PMS



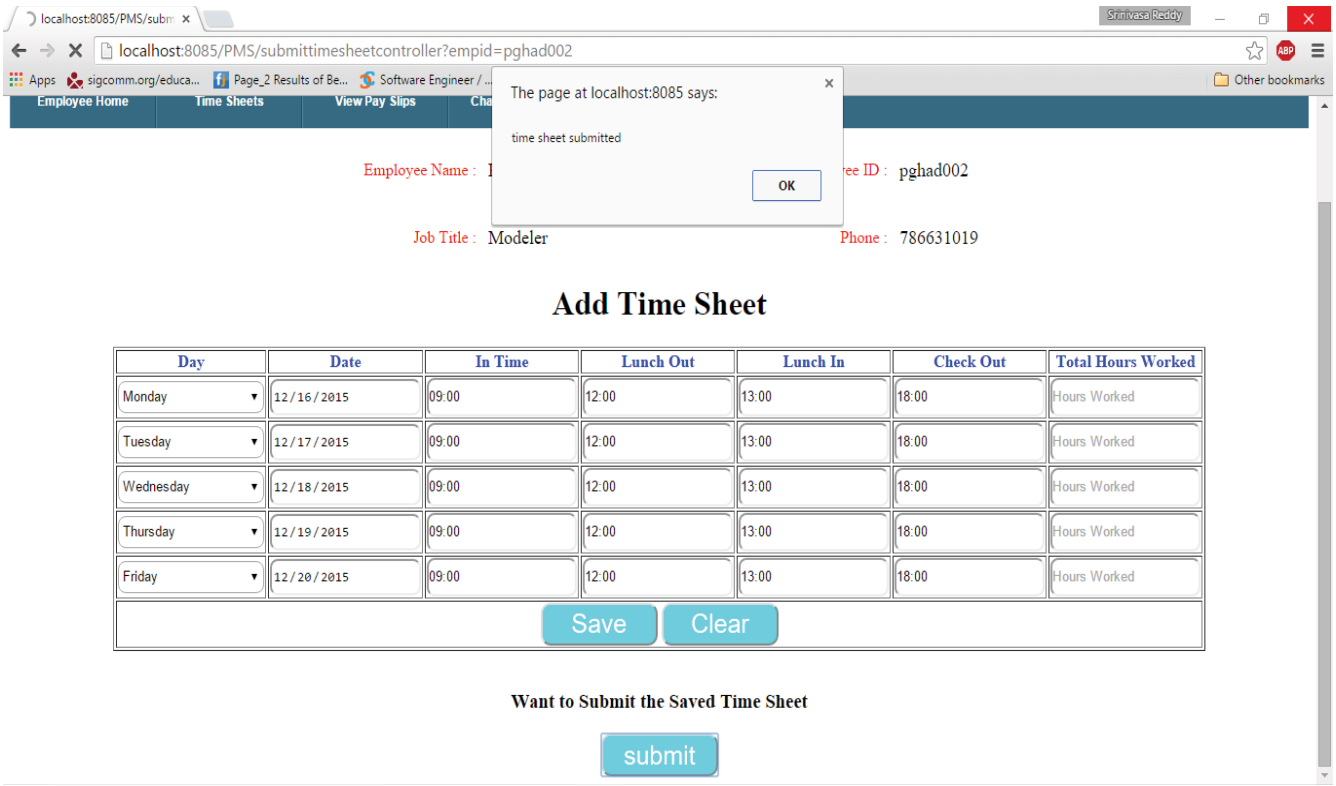
B) Login page for employee/ employer:



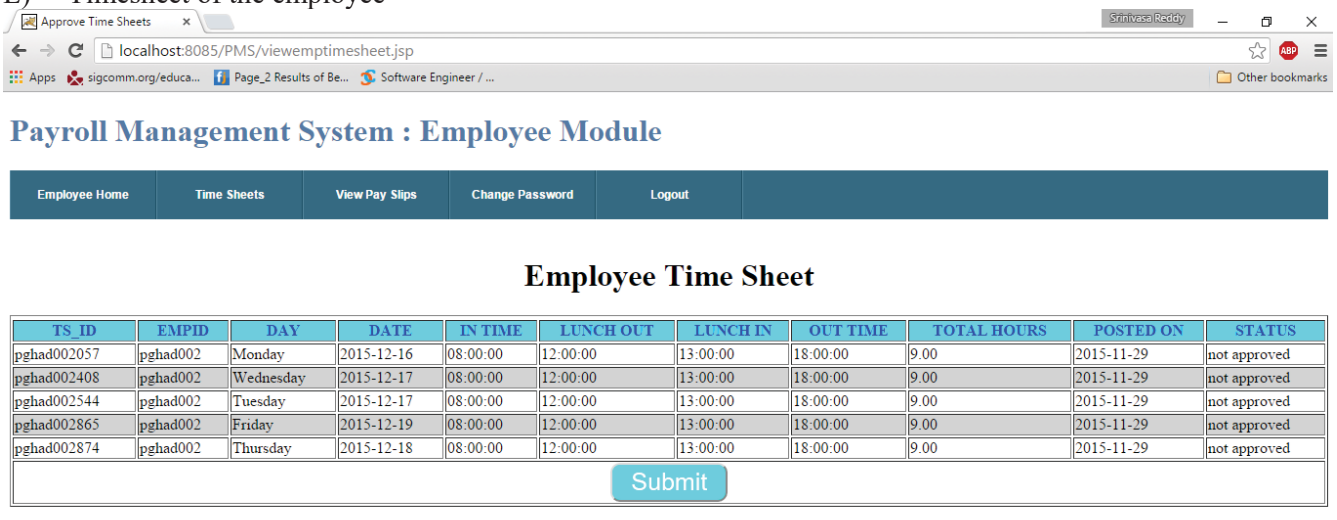
C) Adding the timesheet



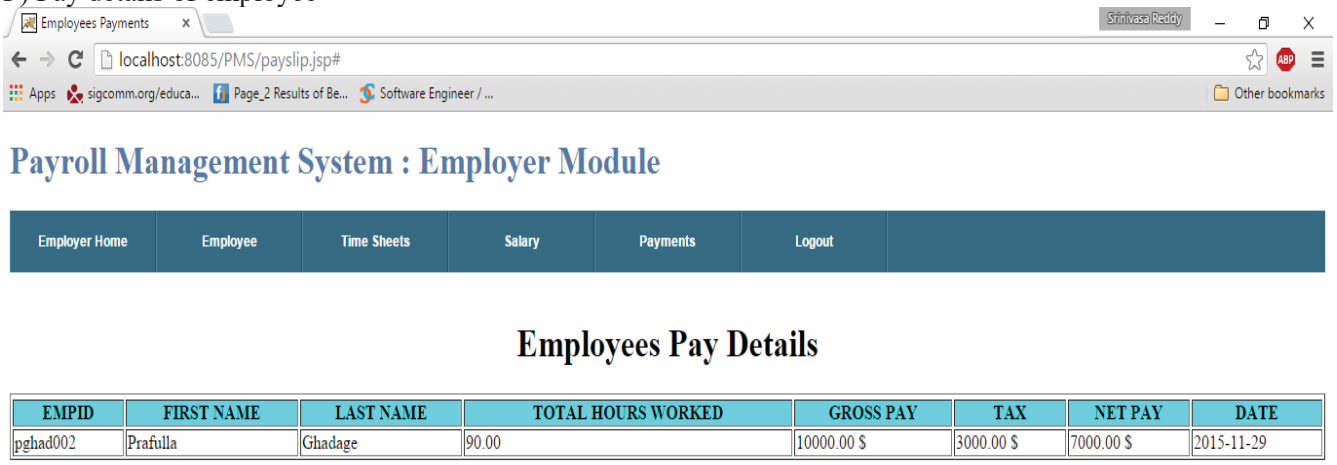
D) Submit the timesheet



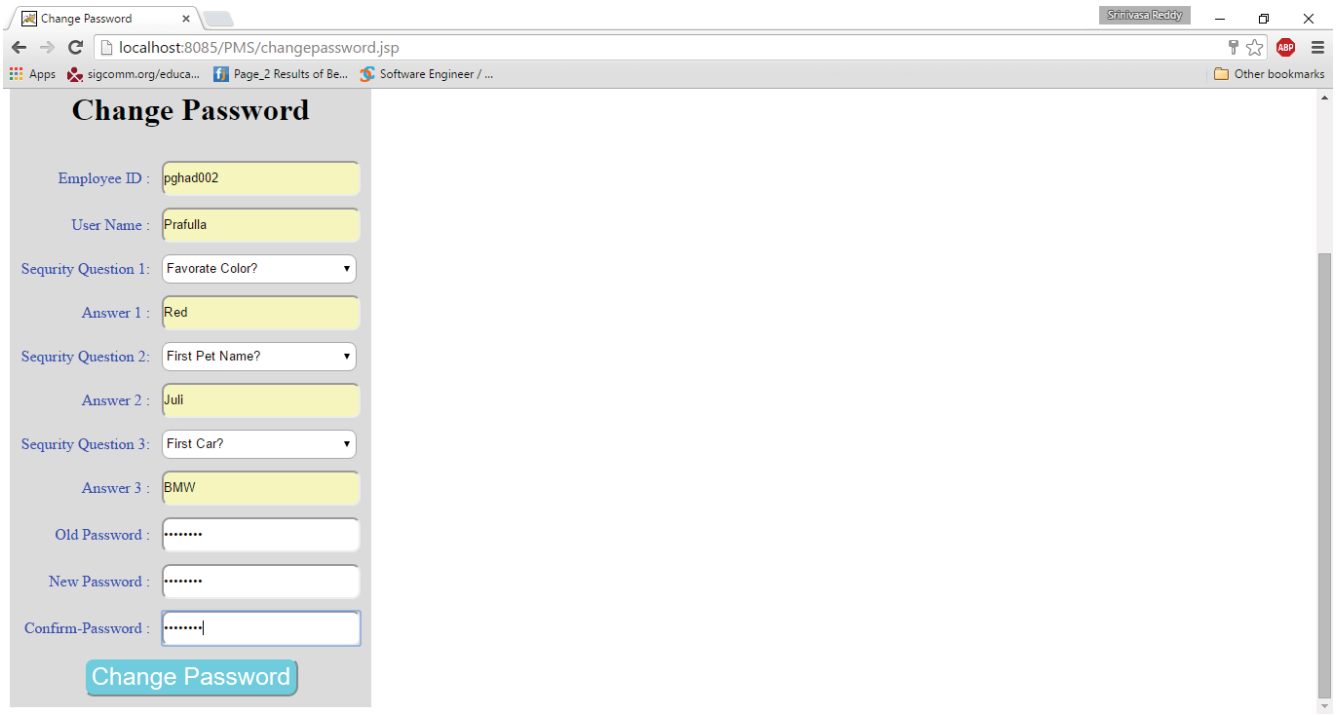
E) Timesheet of the employee



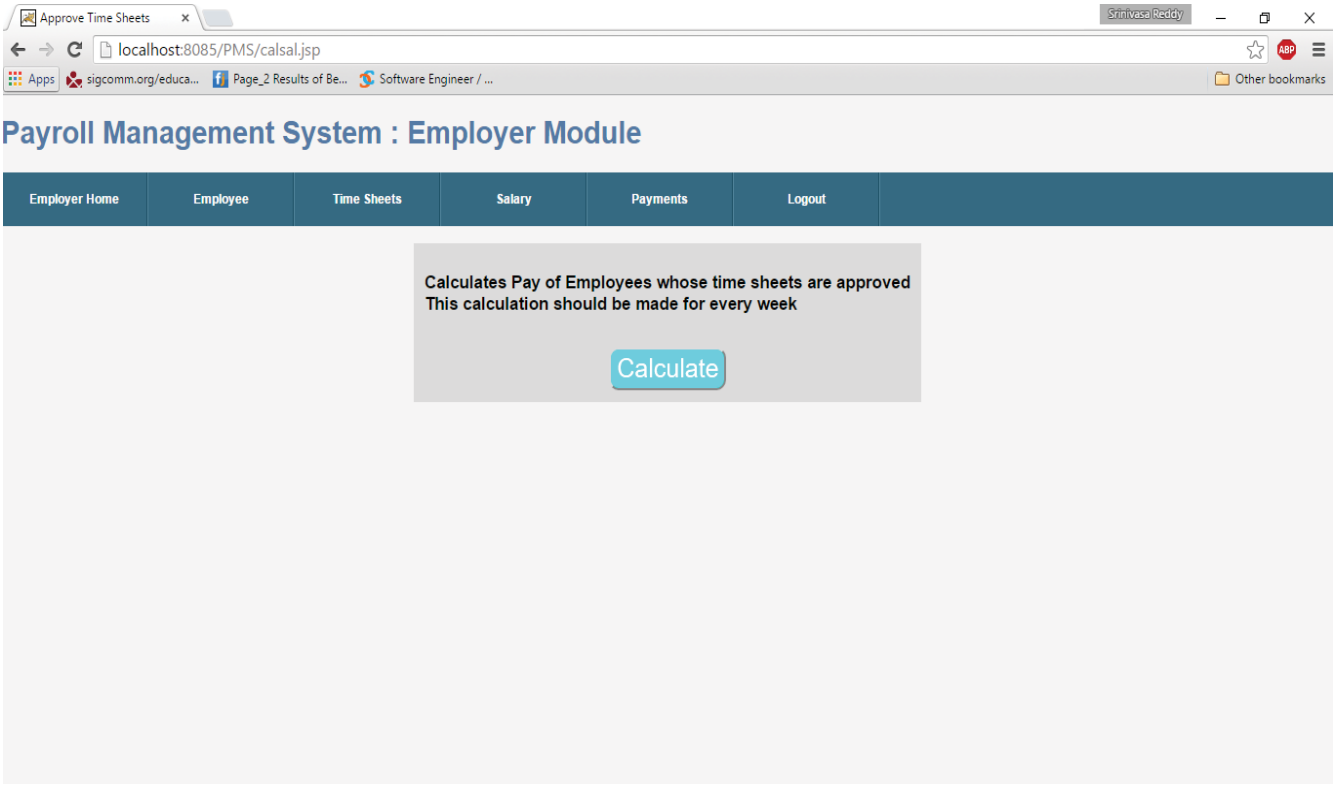
F) Pay details of employee



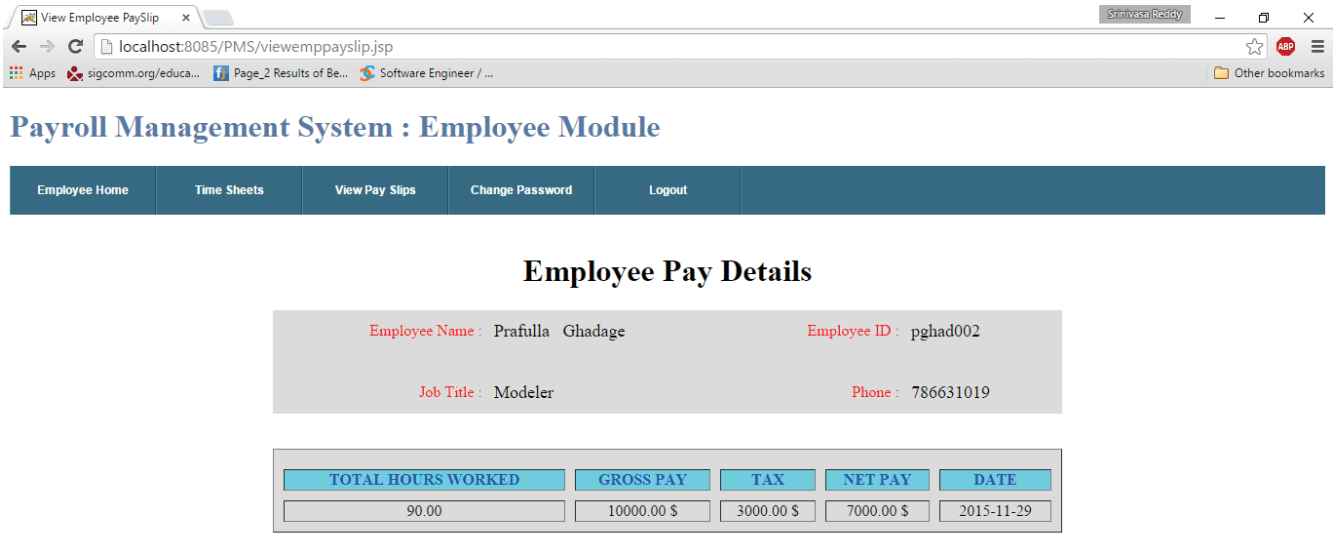
G) Change password



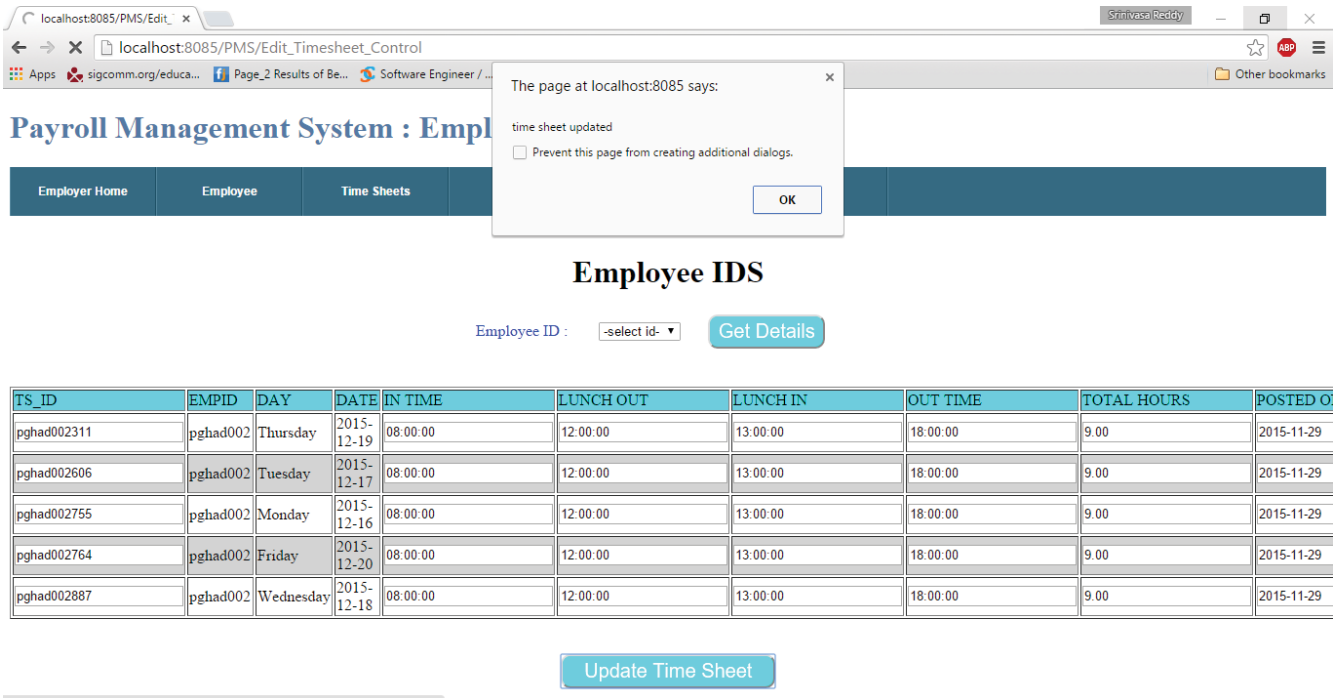
H) Calculate pay check



I) Pay details of employee



J) Update timesheet



K) View timesheets

Payroll Management System : Employee Module

Employee Home | Time Sheets | View Pay Slips | Change Password | Logout

Employee Time Sheet

TS_ID	EMPID	DAY	DATE	IN TIME	LUNCH OUT	LUNCH IN	OUT TIME	TOTAL HOURS	POSTED ON	STATUS
pghad002057	pghad002	Monday	2015-12-16	08:00:00	12:00:00	13:00:00	18:00:00	9.00	2015-11-29	not approved
pghad002408	pghad002	Wednesday	2015-12-17	08:00:00	12:00:00	13:00:00	18:00:00	9.00	2015-11-29	not approved
pghad002544	pghad002	Tuesday	2015-12-17	08:00:00	12:00:00	13:00:00	18:00:00	9.00	2015-11-29	not approved
pghad002865	pghad002	Friday	2015-12-19	08:00:00	12:00:00	13:00:00	18:00:00	9.00	2015-11-29	not approved
pghad002874	pghad002	Thursday	2015-12-18	08:00:00	12:00:00	13:00:00	18:00:00	9.00	2015-11-29	not approved

Submit

L) Pay check

Payroll Management System : Employer Module

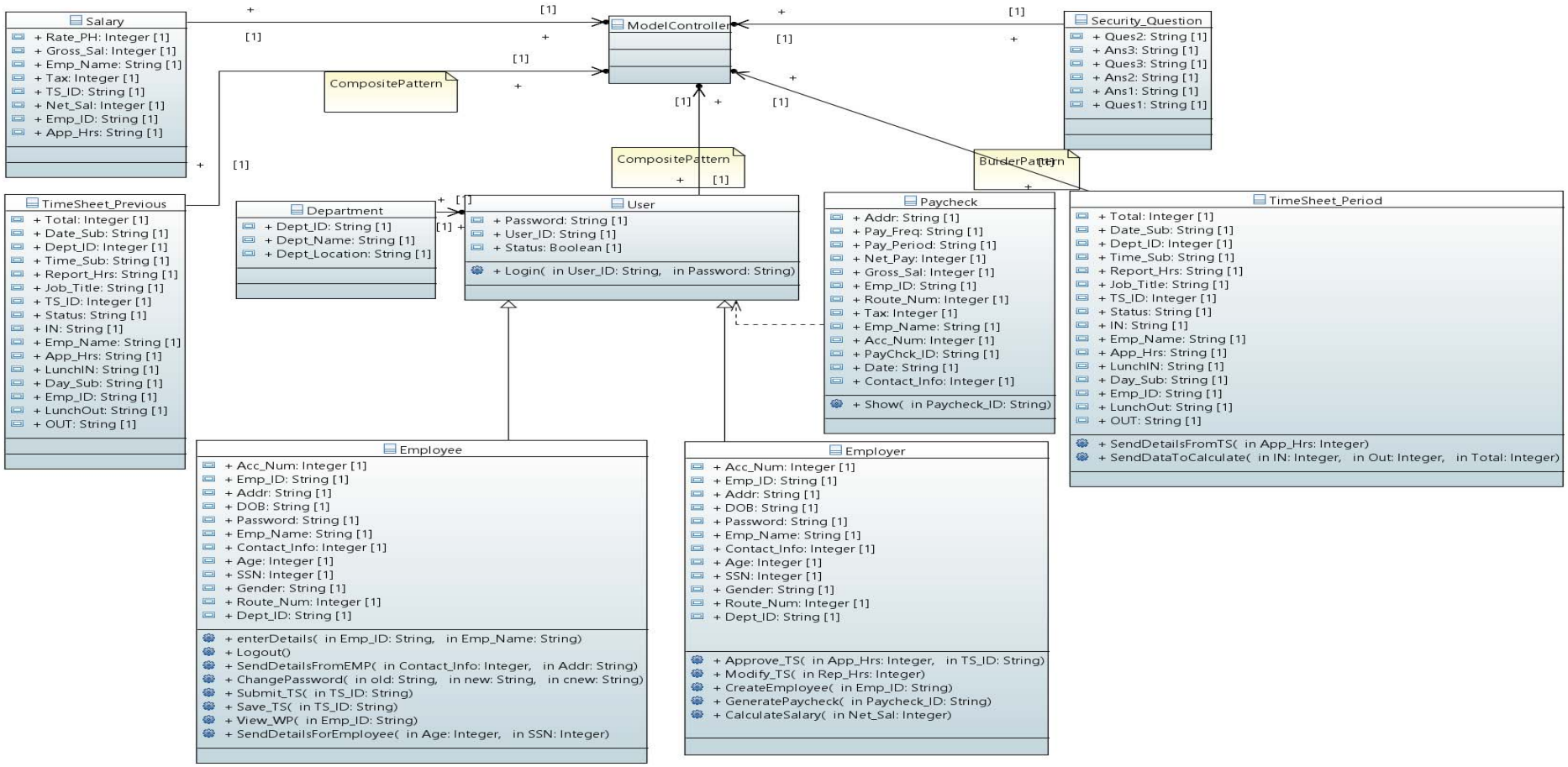
Employer Home | Employee | Time Sheets | Salary | Payments | Logout

Employees Pay Details

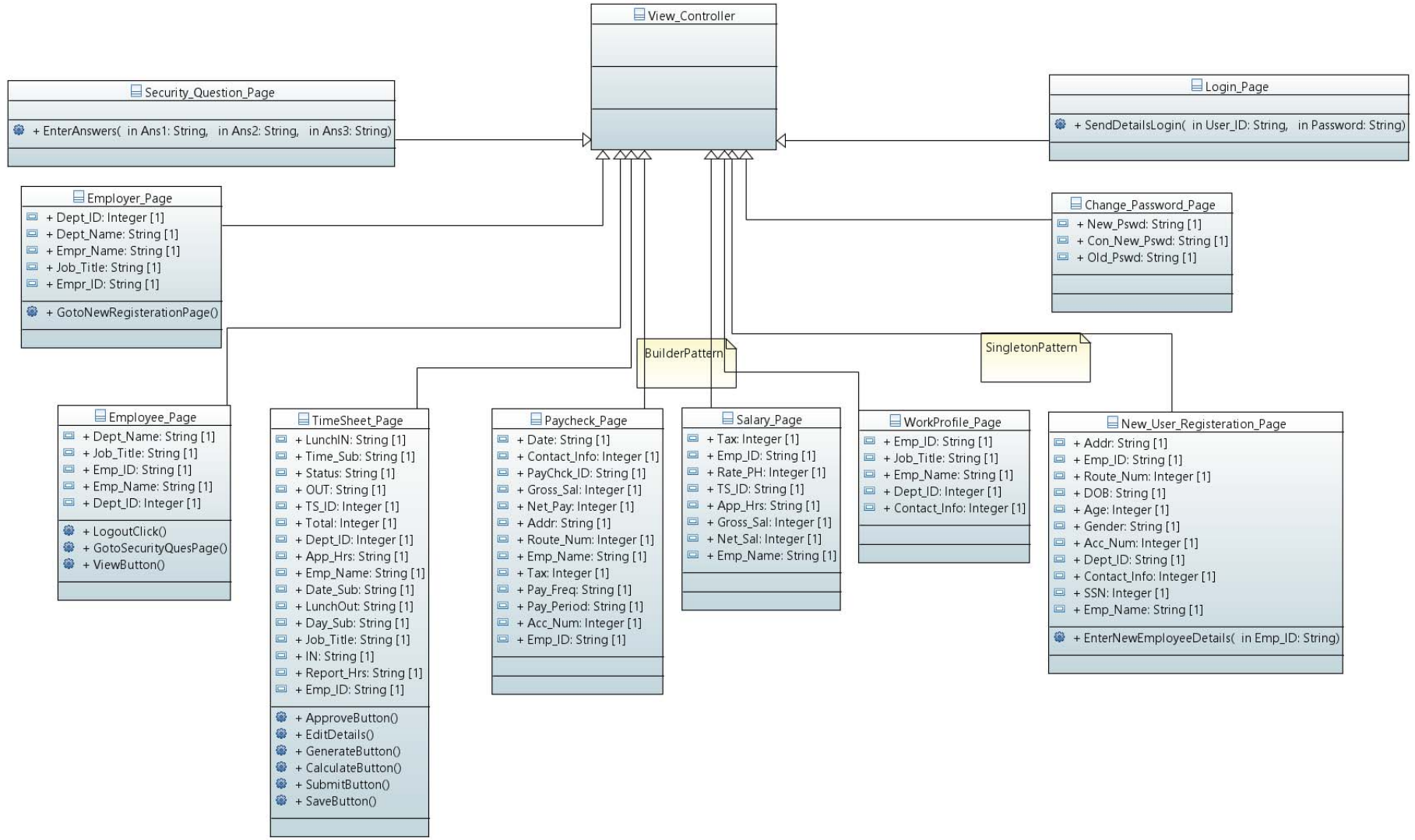
EMPID	FIRST NAME	LAST NAME	TOTAL HOURS WORKED	GROSS PAY	TAX	NET PAY	DATE
pghad002	Prafulla	Ghadage	90.00	10000.00 $	3000.00 $	7000.00 $	2015-11-29

11.4 Appendix D – Detail class diagrams showing attributes and methods for each class.

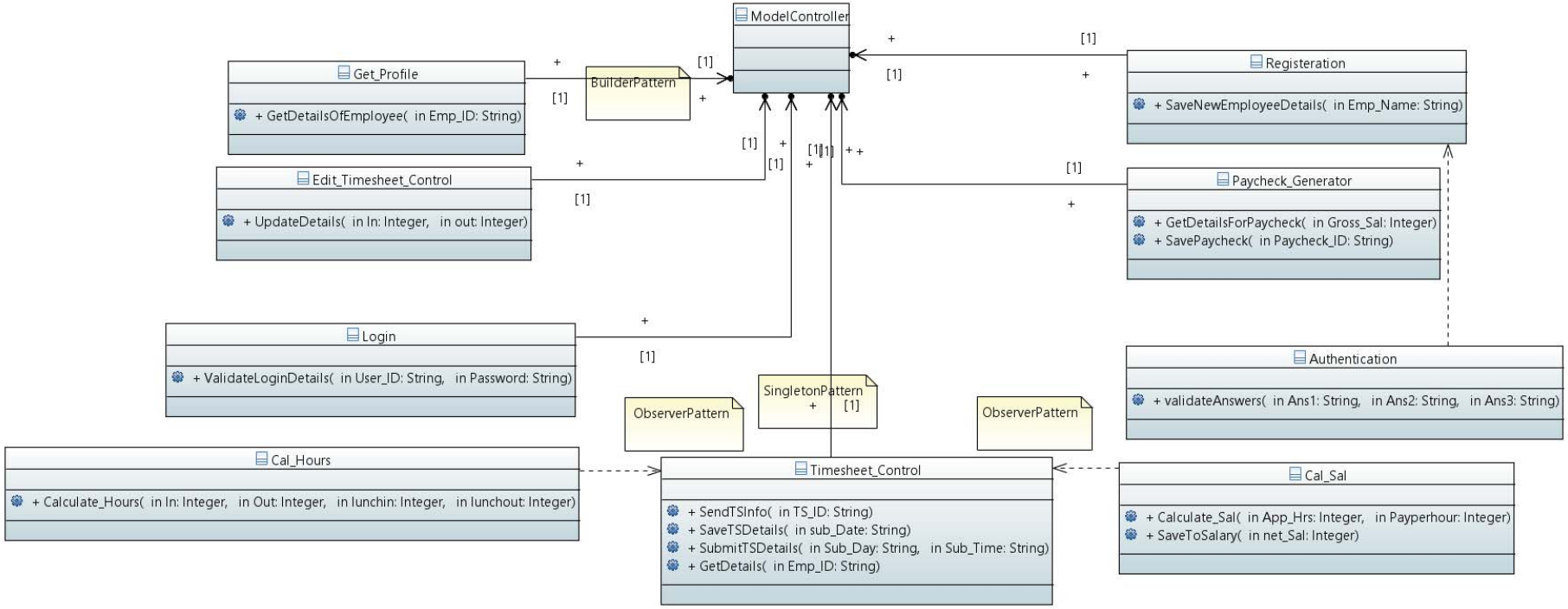
(a) Fig 11.4.1 Model Class Diagram



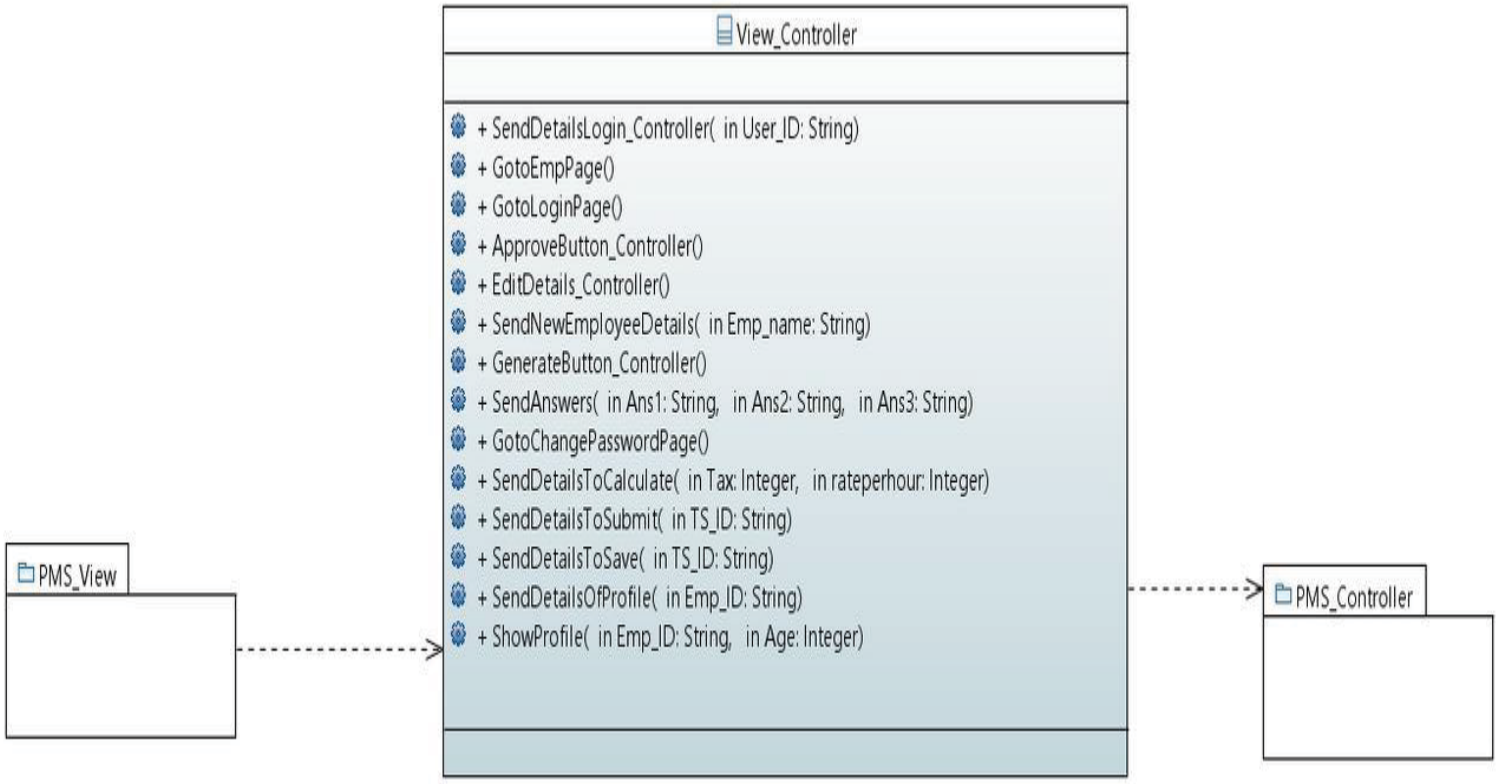
(b) Fig 11.4.2 View Class Diagram.



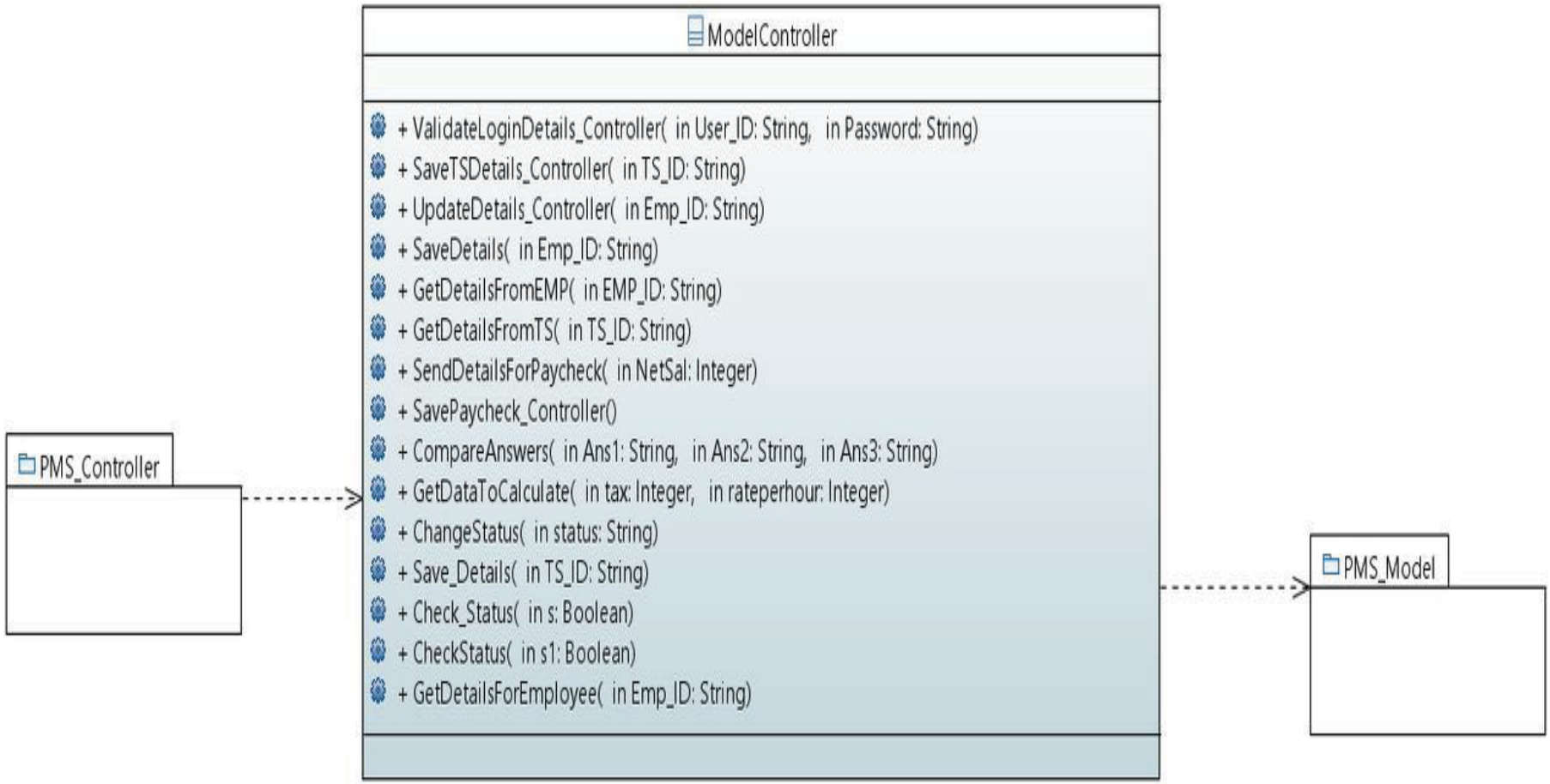
(c) Fig 11.4.3 Controller Class Diagram.



(d) Fig 11.4.4 View Controller



(g) Figure 11.5.5



11.5 Appendix E – Class Interface Code for the Subsystems.

1) Authentication:

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class changepasswordcontroller
 */
@WebServlet("/changepasswordcontroller")
public class Authentication extends HttpServlet {
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		// TODO Auto-generated method stub
PrintWriter pw=response.getWriter();


		String eid = request.getParameter("eid");
		String uid = request.getParameter("uid");
		String s1 = request.getParameter("s1");
		String a1 = request.getParameter("a1");
		String s2 = request.getParameter("s2");
		String a2 = request.getParameter("a2");
		String s3 = request.getParameter("s3");
		String a3 = request.getParameter("a3");
		String oldpass = request.getParameter("oldpass");
		String newpass = request.getParameter("newpass");
		String cpass = request.getParameter("cpass");

		if(cpass.equals(newpass))
		{



		//pw.print("ok");
		model.Employee obj=new model.Employee();
```

```
                String result=obj.changePassword(eid, uid, s1,a1,s2,a2,s3,a3, oldpass,
newpass);
                if(result.equals("success"))
                {
                        pw.print("<script language='javascript'>window.alert('Password
changed');window.location.replace('employeehome.jsp');</script>");

                }
                else
                {
                        response.sendRedirect("error.jsp?msg=Error :"+result);
                //      pw.print("<script language='javascript'>window.alert('Password
not changed');window.location='.jsp';</script>");
                }
                }
                else
                        pw.print("<script language='javascript'>window.alert('Passwords
not matched');window.location.replace('changepassword.jsp');</script>");


        }
}
```

2) Delete Employee Controller.

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class deleteempcontroller
 */
@WebServlet("/deleteempcontroller")
public class deleteempcontroller extends HttpServlet {
                protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

                        PrintWriter pw=response.getWriter();

                        String eid = request.getParameter("eid");
```

```
                        //pw.print("ok");
                        model.Employee obj=new model.Employee();


                String result=obj.deleteEmp(eid);

                        if(result.equals("success"))
                        {
                                pw.print("<script
language='javascript'>window.alert('Employee Details
Deleted');window.location.replace('emplrhome.jsp');</script>");
                        }
                        else
                        {
                                response.sendRedirect("error.jsp?msg="+result);
                        }

        }



}
```

3) Edit Timesheet Controller.

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/**
 * Servlet implementation class updatetimesheetcontroller
 */
@WebServlet("/updatetimesheetcontroller")
public class Edit_Timesheet_Control extends HttpServlet {

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		PrintWriter pw=response.getWriter();
		String result="success";
		HttpSession sn=request.getSession();

		for(int i=1;i<=5;i++)
		{
			String tid=(String)request.getParameter("etsid"+i);

		String intime = request.getParameter("intime"+i);
		String lout = request.getParameter("lunchout"+i);
		String lin = request.getParameter("lunchin"+i);
		String cout = request.getParameter("outtime"+i);
		String thours = request.getParameter("thours"+i);



		model.TimeSheet obj=new model.TimeSheet();

	result=obj.updateTimeSheet(tid, intime, lout, lin, cout,thours);
		if(!result.equals("success"))
			pw.print("<h1>Error :"+result);
		}
		pw.print("<script language='javascript'>window.alert('time sheet
updated');history.back();</script>");

}
```

```
}
```

3) Employer Login Controller.

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import model.Employer;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class emplrlogincontroller extends HttpServlet {



protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

	PrintWriter pw=response.getWriter();
	//pw.print("ok");


	String username = request.getParameter("uname");
	String password = request.getParameter("pwd");

	model.Employer obj=new model.Employer();

String result=obj.authenticate(username, password);

	if(result.equals("success"))
	{
		response.sendRedirect("emplrhome.jsp");
	}
	else
	{
		response.sendRedirect("error.jsp?msg=Admin account not found");
	}


}

}
```

5) Employee Registration Controller

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class empregistration
 */
@WebServlet("/empregistration")
public class empregistrationcontroller extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public empregistrationcontroller() {
        super();
        // TODO Auto-generated constructor stub
    }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		// TODO Auto-generated method stub
		response.getWriter().append("Served at:
").append(request.getContextPath());
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		PrintWriter pw=response.getWriter();

		String eid = request.getParameter("eid");
```

```
            String userid = request.getParameter("uid");
            String password = request.getParameter("pwd");
            String s1 = request.getParameter("s1");
            String a1 = request.getParameter("a1");

            String s2 = request.getParameter("s2");
            String a2 = request.getParameter("a2");

            String s3 = request.getParameter("s3");
            String a3 = request.getParameter("a3");

            //pw.print("ok");
            model.Security_Question obj=new model.Security_Question();


        String result=obj.registeremployee(eid, userid, password, s1, a1, s2, a2, s3, a3);

            if(result.equals("success"))
            {
                response.sendRedirect("emplogin.jsp");
            }
            else
            {
                response.sendRedirect("error.jsp?msg="+result);
            }




        }

}
```

6) Forgot Password Controller:

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class forgotpasswordcontroller
 */
@WebServlet("/forgotpasswordcontroller")
public class forgotpasswordcontroller extends HttpServlet {
		protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
			PrintWriter pw=response.getWriter();
			String eid = request.getParameter("eid");

			String userid = request.getParameter("uid");
			String s1 = request.getParameter("s1");
			String a1 = request.getParameter("a1");

			String s2 = request.getParameter("s2");
			String a2 = request.getParameter("a2");

			String s3 = request.getParameter("s3");
			String a3 = request.getParameter("a3");

		model.Employee obj=new model.Employee();
		String result=obj.getPassword(eid, userid, s1, a1, s2, a2, s3, a3);
		if(result.contains("fail"))
			pw.print("<script language='javascript'>window.alert('user details
not found');window.location.replace('emplogin.jsp');</script>");
		else if(result.contains("Error"))

			response.sendRedirect("error.jsp?msg="+result);
		else
		{
			String s[]=result.split(",");
			response.sendRedirect("error.jsp?msg=Your password="+s[1]);
		}

	}
```

```
}
```

7) Get User Profile:

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class updateempcontroller
 */
@WebServlet("/Get_Profile")
public class Get_Profile extends HttpServlet {

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
	{
		PrintWriter pw=response.getWriter();

		String eid = request.getParameter("eid");

		String fname = request.getParameter("fname");
		String lname = request.getParameter("lname");
		String gen = request.getParameter("gen");
		String dob = request.getParameter("dob");

		String job = request.getParameter("job");
		String contact = request.getParameter("contact");

		String email = request.getParameter("email");
		String addr = request.getParameter("addr");

		String accno = request.getParameter("accno");

		String bname = request.getParameter("bname");



		//pw.print("ok");
		model.Employee obj=new model.Employee();
```

```
        String result=obj.updateEmployee(eid, fname, lname, gen, dob, job, contact,
email, addr, accno, bname);

                if(result.equals("success"))
                {
                        pw.print("<script language='javascript'>window.alert('Employee
Details Updated');window.location.replace('update_emp.jsp');</script>");
                }
                else
                {
                        response.sendRedirect("error.jsp?msg="+result);
                }



        }

}
```

8) Login

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import model.Employer;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class Login extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

	PrintWriter pw=response.getWriter();
	pw.print("ok");

	String eid = request.getParameter("eid");

	String username = request.getParameter("uname");
	String password = request.getParameter("pwd");
	//pw.print("ok");
	model.User obj=new model.User();

String result=obj.authenticate(eid,username, password);

	if(result.equals("success"))
	{
		HttpSession sn=request.getSession();
		sn.setAttribute("empid",eid);
		sn.setAttribute("userid",username);
		response.sendRedirect("employeehome.jsp");
	}
	else
	{
		response.sendRedirect("error.jsp?msg="+result);
	}



}
```

```
}
```
9) Paycheck Generator:
```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class paymodecontroller
 */
@WebServlet("/paymodecontroller")
public class Paycheck_Generator extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

PrintWriter pw=response.getWriter();

		double  normal =Double.parseDouble(request.getParameter("normal"));

		double extra =Double.parseDouble(request.getParameter("extra"));

		model.Salary obj=new model.Salary();


	String result=obj.addPayMode(normal, extra);

		if(result.equals("success"))
		{
			pw.print("<script language='javascript'>window.alert('Pay mode
added');window.location.replace('paymode.jsp');</script>");
		}
		else
		{
			response.sendRedirect("error.jsp?msg="+result);
		}

	}



}
```

Registration:

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class EmpAdd
 */
@WebServlet("/EmpAdd")
public class Registration extends HttpServlet {
	private static final long serialVersionUID = 1L;


	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		// TODO Auto-generated method stub
PrintWriter pw=response.getWriter();

		String eid = request.getParameter("eid");

		String fname = request.getParameter("fname");
		String lname = request.getParameter("lname");
		String gen = request.getParameter("gen");
		String dob = request.getParameter("dob");

		String job = request.getParameter("job");
		String contact = request.getParameter("contact");

		String email = request.getParameter("email");
		String addr = request.getParameter("addr");

		String accno = request.getParameter("accno");

		String bname = request.getParameter("bname");



		//pw.print("ok");
		model.Employee obj=new model.Employee();
```

```
        String result=obj.addEmployee(eid, fname, lname, gen, dob, job, contact, email,
addr, accno, bname);

                if(result.equals("success"))
                {
                        //response.sendRedirect("emplogin.jsp");
                        pw.print("<script language='javascript'>window.alert('Employee
Detials Addded.');window.location='emplrhome.jsp';</script>");
                }
                else
                {
                        response.sendRedirect("error.jsp?msg=Employee Registration
Failed");
                }


        }


}
```

10) Search Employee:

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class submittimesheetcontroller
 */
@WebServlet("/submittimesheetcontroller")
public class Search_Employee extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		PrintWriter pw=response.getWriter();
		String empid=request.getParameter("empid");
		model.TimeSheet obj=new model.TimeSheet();
		obj.submitTimeSheet(empid);
		pw.print("<script language='javascript'>window.alert('time sheet
submitted');history.back();</script>");
	}


}
```

11) Timesheet Controller

```
package controller;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.*;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/**
 * Servlet implementation class addtimesheetcontroller
 */
@WebServlet("/addtimesheetcontroller")
public class Timesheet_Control extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

		PrintWriter pw=response.getWriter();
		String result="success";
		HttpSession sn=request.getSession();
		String empid=(String)sn.getAttribute("empid");
		for(int i=1;i<=5;i++)
		{

		String day = request.getParameter("day"+i);
		String dt = request.getParameter("dt"+i);
		String intime = request.getParameter("in"+i);
		String lout = request.getParameter("lout"+i);
		String lin = request.getParameter("lin"+i);
		String cout = request.getParameter("cout"+i);

		Random r=new Random();

		String tid=empid+""+r.nextInt(9)+r.nextInt(9)+r.nextInt(9);

		//pw.print("ok");
		model.TimeSheet obj=new model.TimeSheet();


		result=obj.addTimeSheet(tid, empid, day, dt, intime, lout, lin, cout);
		if(!result.equals("success"))
```

```
                {
                        response.sendRedirect("error.jsp?msg="+result);
                }
                }

                pw.print("<script language='javascript'>history.back();</script>");


        }


}
```