<EmployeeManagement>

Analysis and Design Document

Student:Pali Anamaria

**Group:30235**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <13/04/2018> | <1.0> | <details> | Pali Anamaria |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

I will design and implement an application for employees of a company. The application will have three types of users (a regular user represented by the front desk employee, an administrator user and a manager) which have to provide a username and a password in order to use the application.

This project will be limited to the following:

• Employee profiles: Employees will have access to their personal profiles and will be able to edit their details.

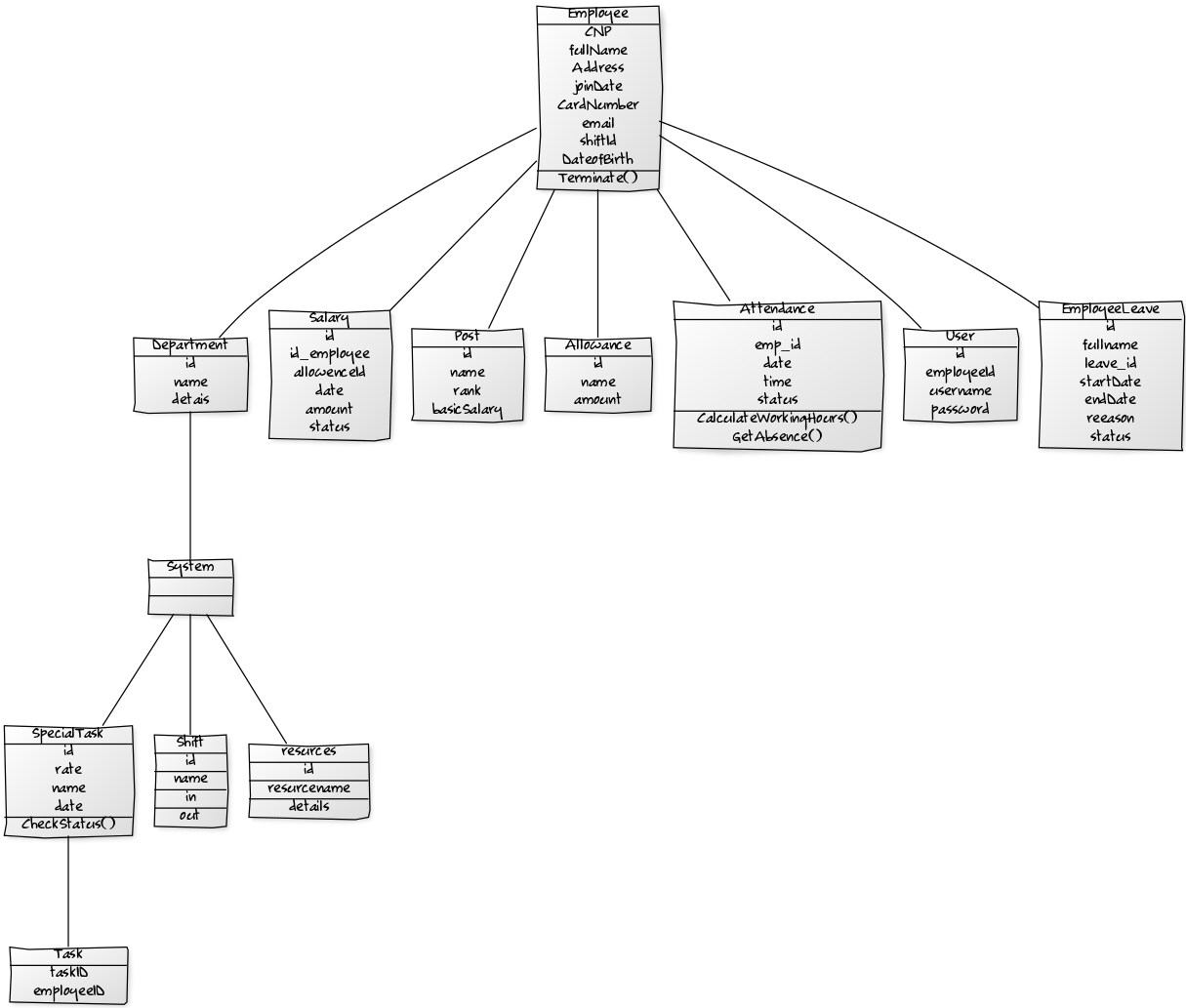
• Electronic leave application: Complete elimination of paperwork in leave management by enabling an employee apply for leave as well as check their leave status through the system. This will also enable the manager to accept/reject leave application through the system

• Project Management: Assign tasks and projects to employees and keep track of the progress.

• Report generation: The manager will be able to generate timely reports in order to monitor employees and this can be used for performance appraisals. The reports will be have all the information of an employee from educational background to projects done as well as technical skills.

# Elaboration – Iteration 1.1

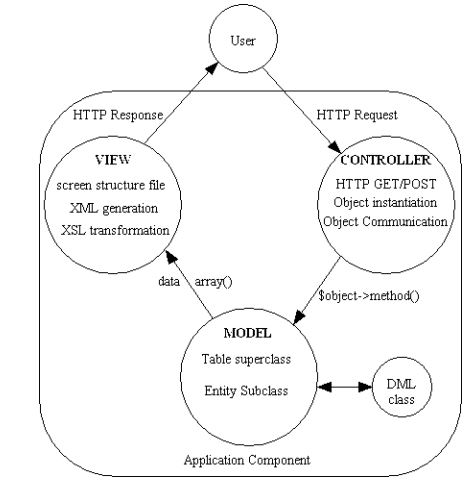
# Domain Model

**

# Architectural Design

## Conceptual Architecture

Model View Controller architecture (MVC) In the implementation, as shown in figure the whole application is broken down into a series of top-level components which may be referred to as tasks, actions, functions, operations or transactions (that's user transactions, not database transactions), each of which is may be related to a Use Case. Each transaction component references a single controller, one or more models, and usually a single view. Some components do not have a view as they are called from other components in order to perform a service, and once this service has been completed they return control to the calling component. Each component is self-executing in that it deals with both the HTTP GET and POST requests.



### Model

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

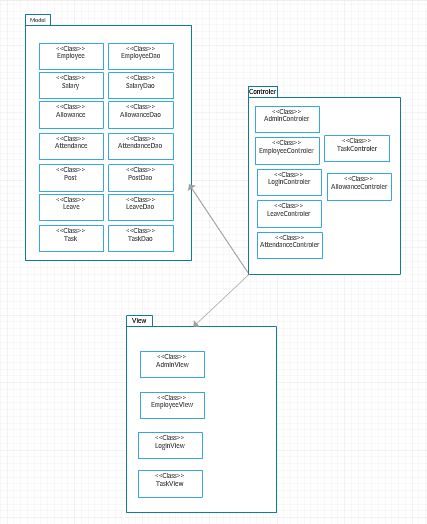
### View

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

### Controller

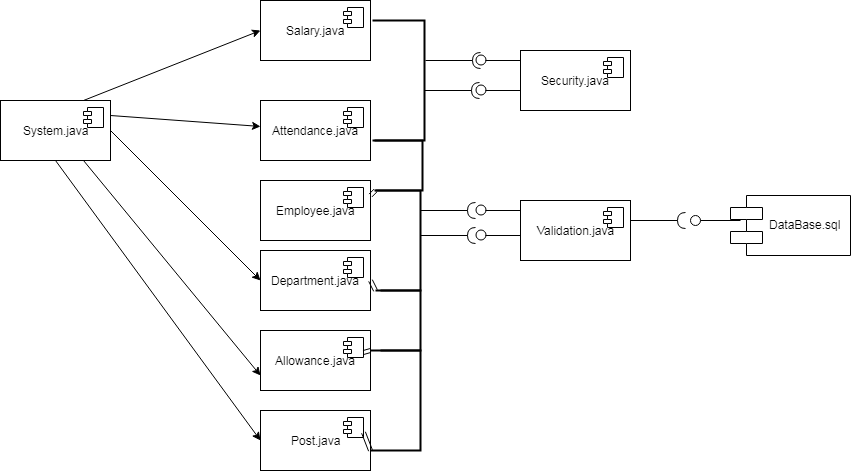
Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

## Package Design

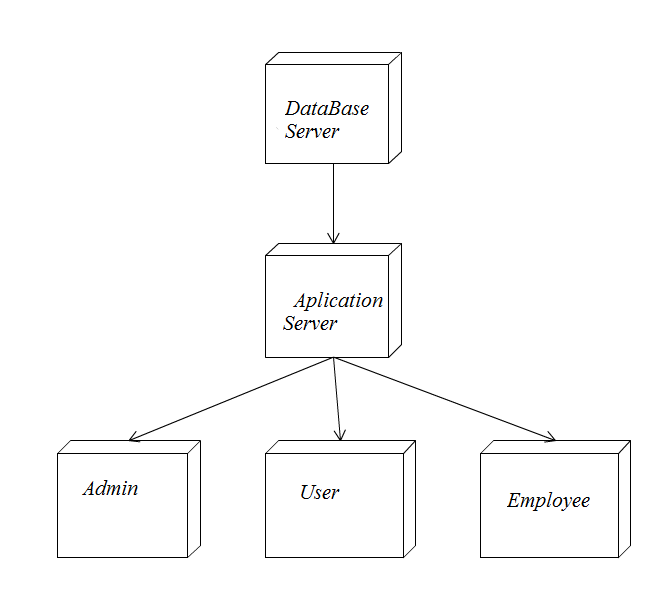
**

## Component and Deployment Diagrams

### Component diagram



### Deployment diagrams

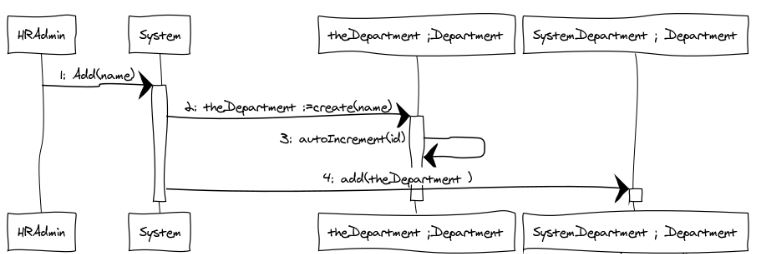


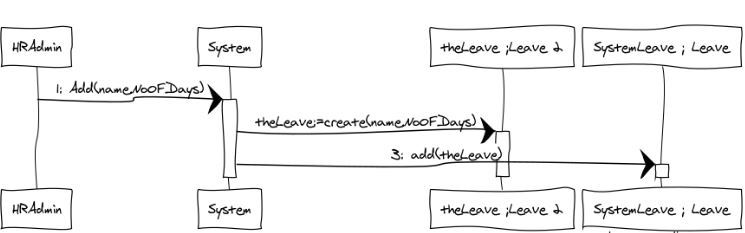
# Elaboration – Iteration 1.2

# Design Model

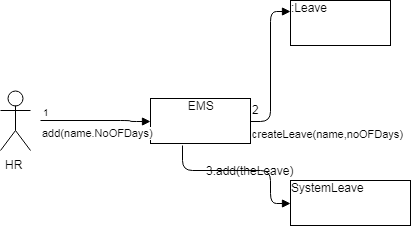
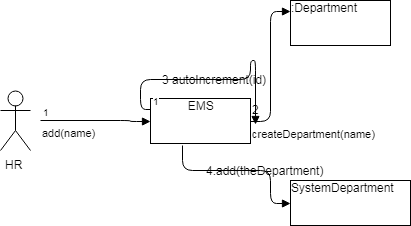
## Dynamic Behavior

### Sequence Diagram





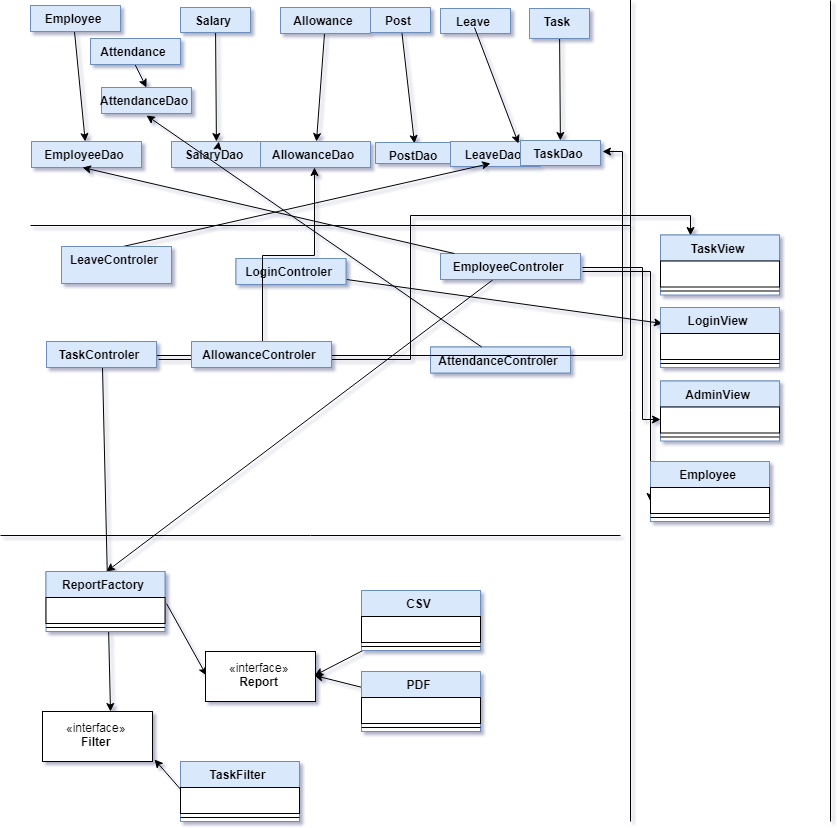
### Communication Diagram



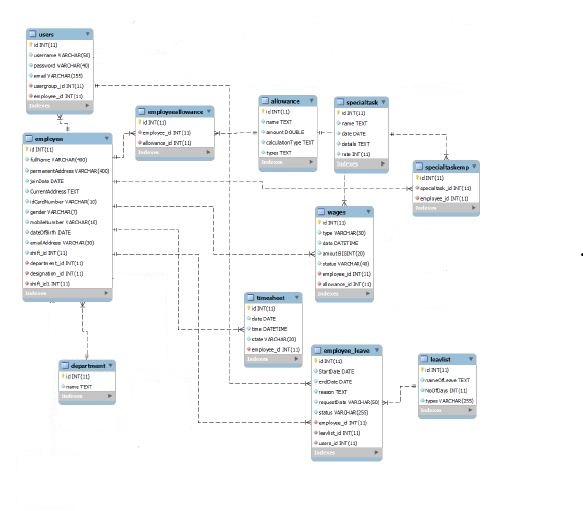
## Class Design

I will use Factory Pattern for exporting bills in PDF and CSV format.

Filter pattern or Criteria pattern is a design pattern that enables developers to filter a set of objects using different criteria and chaining them in a decoupled way through logical operations. This type of design pattern comes under structural pattern as this pattern combines multiple criteria to obtain single criteria. I used this pattern to take from my database all the tasks for an employee.



# Data Model

**

# Unit Testing

The Test Plan is derived from the Requirements, Functional Specifications, and detailed Design Specifications. The Test Plan identifies the details of the tests, identifying the associated test case areas within the produc.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *TestCase :1.0* | | | *Test Name: Create Leave* | |
| *System:Human Resource Management System* | | | *Subsystem:Leave page* | |
| *Pre-Condition* | | | *Employee object must exist*  *Department object must exist* | |
| *Step* | *Action* | ***Data*** | *Expected System Response* | *Pass/Fail* |
| *1* | *Enter number of days* | ***10*** | *Message display:”not sufficient days”* | *Pass* |
| *2* | *Enter number of days* | ***8*** | *Massage display:”Leave added”* | *Pass* |
| *Post-Condition* | | | | |
| *Leave is created* | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *TestCase :2.0* | | | *Test Name: Create Department* | |
| *System:Human Resource Management System* | | | *Subsystem:Department page* | |
| *Pre-Condition* | | | *Employee object must exist* | |
| *Step* | *Action* | ***Data*** | *Expected System Response* | *Pass/Fail* |
| *1* | *Enter name* | ***Human***  ***Resource*** | *Post object created* | *Pass* |
| *Post-Condition* | | | | |
| *Department object is created* | | | | |

# Elaboration – Iteration 2

# Architectural Design Refinement

In the package diagram I changed some parts of it like adding a new package for the exporting files called Pattern. The other ones were good implemented so I didn’t need to change anything. The sequence diagram is the same that was at the first time that I write it, and the other ones except the class diagram suffered no changes.

# Design Model Refinement

I didn’t make some important changes from the initial diagrams just by adding some classes. At the class diagram I make some new classes like Factory for using the factory design pattern ,or CSV and PDF classes that are used for exporting the database in the format that is wanted.

# Construction and Transition

# System Testing

The overall purpose of testing is to ensure the Employee Management System meets all of its functional and business requirements. The purpose of this chapter is to describe the overall test plan and strategy for testing the system.

The goals in testing this system include validating the quality, usability, reliability and performance of the application. Testing will be performed from a black-box approach. Tests will be designed around requirements and functionality.

Confirmation testing or re-testing: When a test fails because of the defect then that defect is reported and a new version of the software is expected that has had the defect fixed. In this case we need to execute the test again to confirm that whether the defect got actually fixed or not. This is known as confirmation testing and also known as re-testing. It is important to ensure that the test is executed in exactly the same way it was the first time using the same inputs, data and environments.

During confirmation testing the defect got fixed and that part of the application started working as intended. But there might be a possibility that the fix may have introduced or uncovered a different defect elsewhere in the software. The way to detect these ‘unexpected side-effects’ of fixes is to do regression testing. The purpose of a regression testing is to verify that modifications in the software or the environment have not caused any unintended adverse side effects and that the system still meets its requirements. Regression testing are mostly automated because in order to fix the defect the same test is carried out again and again and it will be very tedious to do it manually. Regression tests are executed whenever the software changes, either as a result of fixes or new or changed functionality

# Future improvements

An important improvement for this project will be to transform it to web service so that all the users that have access to the internet can use the application. Another update for the project will be to have the possibility to print the bills in another form like JSON.

Using more design patterns is every time a good improvement for a project,and this is not the exception for this rule.

# Bibliography

1. <https://www.tutorialspoint.com>
2. <http://www.oodesign.com/factory-pattern.html>