Abstract of project presented to the School of Technology Management in fulfillment of the requirements for the degree of Bachelor in Computer Science.

**Web Based Payroll System**

By

**Siew Wing Fei**

**Jan 2013**

As for the abstract, it usually encompasses four (4) elements:  
  
1. Statement of problem of the research or project - issues addressed  
2. method used for implementation  
3. results and finding  
4. conclusion

Web based applications have evolved significantly over recent years and with improvements in security and technology there are plenty of good opportunities to develop a system as web based application.

This dissertation has covered a full report on the Web Based Payroll System developed as a prototype to solve the manual payroll system.

The Web Based Payroll System has the ability to update and maintain employee details, define deductions, tax, pay rates, overtime pay rates, generate payslip, generate charts, etc by using Ruby on Rails web framework, jQuery, jQuery UI, and Highcharts JS.

The Web Based Payroll System uses AJAX technology to exchange data with a server, and update parts of a web page without reloading the whole page, which makes the applications similar to a desktop application.

# Acknowledgment

First and foremost I thank to my family for helping me all the way from the very first day of my life.

Right after that I offer my sincerest gratitude to my supervisor, Mr. Amjad Hanesh, who has supported me throughout my studies and thesis with his patience and knowledge whilst allowing me the room to work in my own way. I attribute the level of my bachelor's degree to his encouragement and effort and without him this project and thesis, too, would not have been completed or written. Besides my academic studies, I have learned more than a lot from him, which includes RFID technology, Dot Net Framework, Mobile development, CCTV camera programming, and FTIR Multi Touch Screen development etc. One simply could not wish for a better and friendlier supervisor.

My sincere thanks go to all lecturers and members of the staff of the School of Technology Management, Binary University College, who helped me in many ways and made my education journey at Binary University College pleasant and unforgettable. During my part time studies I have been blessed and helped with a friendly and cheerful group of college staff, and fellow students, Mr. Bilal (School of Technology Management Coordinator),

I acknowledge my sincere indebtedness and gratitude to my parents for their love, dream and sacrifice throughout my life. I am really thankful for their sacrifice, patience, and understanding that were inevitable to make this work possible. Their sacrifice had inspired me from the day I learned how to read and write until what I have become now. I cannot find the appropriate words that could properly describe my appreciation for their devotion, support and faith in my ability to achieve my dreams.

Finally, I would like to thanks any person which contributes to my final year project directly or indirectly. I would like to acknowledge their comments and suggestions, which was crucial for the successful completion of this study.

# Declaration

I declare this project work in my original work except for the quotations and citation which has been accordingly acknowledged. I also declare this project work has not been previously, and is not currently, submitted for any other degree at Binary University College.

(Signature)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SIEW WING FEI**

# List of Figures

[Figure 1: The synchronous interaction pattern of a traditional web application (top) compared with the asynchronous pattern of an Ajax application (bottom) 4](#_Toc353665585)

[Figure 2: The traditional model for web applications (left) compared to the Ajax model (right). 17](#_Toc353665586)

[Figure 3: A typical collaboration of the MVC components. 22](#_Toc353665587)

[Figure 4: Use Case Diagram – Administrator/HR Administrator 26](#_Toc353665588)

[Figure 5: Use Case Diagram - Employee 27](#_Toc353665589)

[Figure 6: Generic Overview of Web Based Payroll System 41](#_Toc353665590)

[Figure 7: Level 0 Data Flow Diagram 42](#_Toc353665591)

[Figure 8: Level 1 Data Flow Diagram 43](#_Toc353665592)

[Figure 9: Class Diagram of Domain Model 44](#_Toc353665593)

[Figure 10: Class Diagram of Controller 45](#_Toc353665594)

[Figure 11: All-in-one Architecture Model 47](#_Toc353665595)

[Figure 12: Flat Architecture Model 47](#_Toc353665596)

[Figure 13: Hub-and-spoke (or Daisy) pattern 48](#_Toc353665597)

[Figure 14: Web Based Payroll System Accordion 49](#_Toc353665598)

[Figure 15: Paging interface of Web Based Payroll System 49](#_Toc353665599)

[Figure 16: rails server command 54](#_Toc353665600)

[Figure 17: The main project folder 59](#_Toc353665601)

[Figure 18: The application folder 60](#_Toc353665602)

[Figure 19: Running unit test in Rails 61](#_Toc353665603)

[Figure 20: Running functional test in Rails 61](#_Toc353665604)

[Figure 21: Running integration test in Rails 62](#_Toc353665605)

[Figure 22: Black Box testing 63](#_Toc353665606)

# List of Tables

[Table 1: Actors Grid 28](#_Toc353665607)

[Table 2: Use Cases Grid 29](#_Toc353665608)

[Table 3: Use Case Diagram Summary 31](#_Toc353665609)

[Table 4: Operating Environment 35](#_Toc353665610)

[Table 5: Chart used 36](#_Toc353665611)

[Table 6: Five levels of testing 62](#_Toc353665612)

[Table 7: Test Case Planning Format 63](#_Toc353665613)

[Table 8: Test cases for Web Based Payroll System 64](#_Toc353665614)

**TABLE OF CONTENTS**

[Abstract ii](#_Toc353835335)

[Acknowledgment iii](#_Toc353835336)

[Declaration iv](#_Toc353835337)

[List of Figures v](#_Toc353835338)

[List of Tables vi](#_Toc353835339)

[CHAPTER ONE 1](#_Toc353835340)

[INTRODUCTION 1](#_Toc353835341)

[1.1: Introduction 1](#_Toc353835342)

[1.2: Problem Statement 1](#_Toc353835343)

[1.3: Significance of the project 2](#_Toc353835344)

[1.3.1: Benefits of web based application 2](#_Toc353835345)

[1.4: Payroll System 3](#_Toc353835346)

[1.5: Ajax web application 3](#_Toc353835347)

[1.5.1: Why Ajax is widely used in Web Applications 4](#_Toc353835348)

[1.5.2: Technologies used in Ajax 5](#_Toc353835349)

[1.6: Methodology 6](#_Toc353835350)

[1.7: Objectives 6](#_Toc353835351)

[1.8: Project write-up Layout 6](#_Toc353835352)

[1.9: Limitations 7](#_Toc353835353)

[CHAPTER TWO 8](#_Toc353835354)

[LITERATURE REVIEW 8](#_Toc353835355)

[2.1: Web based application 8](#_Toc353835356)

[2.1.1: History 8](#_Toc353835357)

[2.1.2: Interface 10](#_Toc353835358)

[2.1.3: Structure 10](#_Toc353835359)

[2.1.4: Business use 10](#_Toc353835360)

[2.1.5: Web application development 11](#_Toc353835361)

[2.1.6: Benefits 11](#_Toc353835362)

[2.1.7: Drawbacks 12](#_Toc353835363)

[2.2: Single-page application 13](#_Toc353835364)

[2.2.1: Architectural characteristics 13](#_Toc353835365)

[2.2.2: Technical approaches 14](#_Toc353835366)

[2.2.3: Running locally 15](#_Toc353835367)

[2.2.4: Challenges with the SPA model 15](#_Toc353835368)

[2.2.5: Page lifecycle 16](#_Toc353835369)

[2.3: Ajax 16](#_Toc353835370)

[2.3.1: History 17](#_Toc353835371)

[2.3.2: Technologies 18](#_Toc353835372)

[2.3.3: Drawbacks 18](#_Toc353835373)

[2.4: Payroll System 20](#_Toc353835374)

[2.5: Model-view-controller (MVC) design pattern 21](#_Toc353835375)

[2.5.1: History 21](#_Toc353835376)

[2.5.2: Component interactions 21](#_Toc353835377)

[2.5.3: Dependency hierarchy 22](#_Toc353835378)

[2.5.4: Use in web applications 22](#_Toc353835379)

[CHAPTER THREE 24](#_Toc353835380)

[REQUIREMENT AND ANALYSIS 24](#_Toc353835381)

[3.1: Requirements Analysis in general 24](#_Toc353835382)

[3.1.1: Software prototyping 25](#_Toc353835383)

[3.1.2: Outline of the prototyping process 25](#_Toc353835384)

[3.2: Software Requirements Specification 26](#_Toc353835385)

[3.2.1: USE CASE Diagram 26](#_Toc353835386)

[3.2.2: Requirement Diagram 34](#_Toc353835387)

[3.2.3: Operating Environment 35](#_Toc353835388)

[3.2.4: Constraints and Dependencies 35](#_Toc353835389)

[CHAPTER FOUR 40](#_Toc353835390)

[DESIGN 40](#_Toc353835391)

[4.1: Design in Software Engineering 40](#_Toc353835392)

[4.2: Flow of the data 42](#_Toc353835393)

[4.3: Class Diagram 44](#_Toc353835394)

[4.4: Overall Sequence diagram 46](#_Toc353835395)

[4.5: Website Architecture 46](#_Toc353835396)

[4.5.1: Architecture Model 47](#_Toc353835397)

[4.6: Navigation Design 48](#_Toc353835398)

[4.6.1: Accordion 48](#_Toc353835399)

[4.6.2: Paging 49](#_Toc353835400)

[CHAPTER FIVE 50](#_Toc353835401)

[IMPLEMENTATION 50](#_Toc353835402)

[5.1: Implementation in Software Engineering 50](#_Toc353835403)

[5.2: Before writing the Web Based Payroll System 50](#_Toc353835404)

[5.3: Selecting the right tools, programming language, frameworks, technologies, and IDEs 51](#_Toc353835405)

[5.3.1: APIs and Libraries 51](#_Toc353835406)

[5.3.2: Frameworks 53](#_Toc353835407)

[5.3.3: Technologies 55](#_Toc353835408)

[5.3.4: Integrated Development Environment 56](#_Toc353835409)

[CHAPTER SIX 57](#_Toc353835410)

[TESTING 57](#_Toc353835411)

[6.1: Testing in Software Engineering 57](#_Toc353835412)

[6.2: Goals and Types of Testing 57](#_Toc353835413)

[6.2.1: System Testing 58](#_Toc353835414)

[6.2.2: Component Testing 58](#_Toc353835415)

[6.3: Black box testing 63](#_Toc353835416)

[6.4: Test Cases 63](#_Toc353835417)

[CHAPTER SEVEN 72](#_Toc353835418)

[FUTURE WORK/MAINTENANCE 72](#_Toc353835419)

[7.1: Recommendations and Features 72](#_Toc353835420)

[7.1.1: Leave Management System 72](#_Toc353835421)

[7.1.2: Time Management System 72](#_Toc353835422)

[7.1.3: Import/Export 73](#_Toc353835423)

[References 74](#_Toc353835424)

[APPENDIX A 76](#_Toc353835425)

[COMPLETE SOURCE CODE 76](#_Toc353835426)

[config/application.rb 77](#_Toc353835427)

[config/environment.rb 79](#_Toc353835428)

[config/database.yml 80](#_Toc353835429)

[config/routes.rb 81](#_Toc353835430)

[app/models/attendance.rb 86](#_Toc353835431)

[app/models/department.rb 87](#_Toc353835432)

[app/models/designation.rb 88](#_Toc353835433)

[app/models/employee.rb 89](#_Toc353835434)

[app/models/employee\_contact.rb 90](#_Toc353835435)

[app/models/employee\_job.rb 91](#_Toc353835436)

[app/models/employee\_qualification.rb 92](#_Toc353835437)

[app/models/employee\_salary.rb 93](#_Toc353835438)

[app/models/employment\_status.rb 95](#_Toc353835439)

[app/models/job\_category.rb 96](#_Toc353835440)

[app/models/overtime\_rate.rb 97](#_Toc353835441)

[app/models/pay\_rate.rb 98](#_Toc353835442)

[app/models/salary\_adjustment.rb 99](#_Toc353835443)

[app/models/user.rb 100](#_Toc353835444)

[app/controllers/application\_controller.rb 102](#_Toc353835445)

[app/controllers/admin/admin\_controller.rb 105](#_Toc353835446)

[app/controllers/admin/attendance\_controller.rb 108](#_Toc353835447)

[app/controllers/admin/department\_controller.rb 110](#_Toc353835448)

[app/controllers/admin/designation\_controller.rb 113](#_Toc353835449)

[app/controllers/admin/employee\_controller.rb 116](#_Toc353835450)

[app/controllers/admin/employment\_status\_controller.rb 122](#_Toc353835451)

[app/controllers/admin/hourly\_payroll\_chart\_controller.rb 125](#_Toc353835452)

[app/controllers/admin/job\_category\_controller.rb 127](#_Toc353835453)

[app/controllers/admin/overtime\_chart\_controller.rb 130](#_Toc353835454)

[app/controllers/admin/overtime\_rate\_controller.rb 132](#_Toc353835455)

[app/controllers/admin/pay\_rate\_controller.rb 135](#_Toc353835456)

[app/controllers/admin/payslip\_controller.rb 138](#_Toc353835457)

[app/controllers/admin/salary\_adjustment\_controller.rb 141](#_Toc353835458)

[app/controllers/admin/total\_work\_hours\_chart\_controller.rb 144](#_Toc353835459)

[app/controllers/admin/user\_controller.rb 146](#_Toc353835460)

# CHAPTER ONE

# INTRODUCTION

## 1.1: Introduction

Web based applications have evolved significantly over recent years and with improvements in security and technology there are plenty of good opportunities to develop a system as web based application. The Payroll System project is a fully Ajax enabled web application which is developed to solve the manual payroll system. The framework that was used in the project is Ruby on Rails, which is open source full-stack web application framework for the Ruby programming language. Ruby on Rails uses the Model-View-Controller architecture pattern to organize application programming. In this dissertation, we will go through all the steps that were involved in the development process of the system.

## 1.2: Problem Statement

Manual payroll system calculates the employee's salary entirely on paper. If the system is not systematic, searching for particular employee detail is very difficult. In addition, the pay slips for the employee will have to be manually generated. This problem will increase the amount of time to process the payroll transaction. Besides, it is very easy to make mistakes when processing payroll transaction manually, especially mistakes in taxing, which can be very costly. Furthermore, if the detail of the employee is kept on document, the document maybe seen by other people. All of this problem can be a burden for the payroll administrator due to the amount of manual works to be done by the payroll administrator before the pay slips can be given to employee. In order to solve the manual payroll system, a computerized payroll system is essential. Some suggestion has been made to solve the problem that faced by the payroll administrator. For example, a user has to enter their username and password in order to access the system. This will make the system more secure. The payroll administrator also can control the user authority to make sure the data is secure and only can be accessed by the authorized user. Moreover the chances of mistakes are lower because all the calculation is done by the system itself. The data is kept in a systematic way where it allows data to be searched. For example, payroll administrator can search by employee id, employee name, and others. The pay slips for employee will also automatically generated by the system with a mouse click. Payroll administrator can print out the pay slips anytime they want, and the pay slips will display all the details for the calculation of employee's salary. Data modifications are also easy because it can be done with just a mouse click.

## 1.3: Significance of the project

The significance of the project is listed below:

* The Payroll System will help to reduce work load and reduce the risk of information loss.
* The Payroll System will help the payroll administrator to manage the employee’s salary in an efficient manner and increase work performance.
* The Payroll System will able to calculate the salary based on overtime, hourly rate, deduction and allowances according to the data inserted by the administrator.
* User can retrieve and view the records of employee's detail by using the search function from the system or generate various charts.
* The system will be more user friendly with web based application. The system will eliminate the time-consuming and potentially inaccurate method of handwritten notes and manually counted salary.
* The work performance of the payroll administrator will be improved, since using computerized system can save a lot of time.
* The information can be retrieved in a shorter time compared to the manual system. It will also reduce the data input time.
* Payroll administrator can maintain the employee's detail in an easier way by just using the update and delete function from the system.
* The system uses database to store the employee’s detail, which encourages the integration of data and makes data more widely available.

### 1.3.1: Benefits of web based application

There are certain numbers of benefits of the project to be developed as a web based application. The benefits include:

* **Cross platform compatibility**

Most web based applications are far more compatible across platforms than traditional installed software. Typically the minimum requirement would be a web browser of which there are many. (Internet Explorer, Firefox, Google Chrome, to name but a few). These web browsers are available for a multitude of operating systems which includes Windows, Linux, or Mac OS.

* **More manageable**

Web based systems need only be installed on the server placing minimal requirements on the end user workstation. This makes maintaining and updating the system much simpler as usually it can all be done on the server. Any client updates can be deployed via the web server with relative ease.

* **Highly deployable**

Due to the manageability and cross platform support deploying web applications to the end user is far easier. They are also ideal where bandwidth is limited and the system and data is remote to the user. At their most deployable the developer simply need to provide the user a website address to log in to and provide them with internet access.

This has huge implications allowing the developer to widen access to the systems, streamline processes and improve relationships by providing more users with access to the systems.

* **Secure live data**

Typically in larger more complex systems data is stored and moved around separate systems and data sources. In web based systems these systems and processes can often be consolidated reducing the need to move data around.

Web based applications also provide an added layer of security by removing the need for the user to have access to the data and back end servers.

* **Reduced costs**

Web based applications can dramatically lower costs due to reduced support and maintenance, lower requirements on the end user system and simplified architecture. [1]

## 1.4: Payroll System

A Payroll System helps an employer to process its payroll. Consequently, payroll cannot be process without a payroll system. A payroll system allows the employer to pay employees on time and accurately, plus comply with other statutory regulations. [2]

## 1.5: Ajax web application

Ajax (an acronym for Asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. [3] In classic web application model, most user actions in the interface trigger an HTTP request back to a server. The server does some processing and then returns a HTML page to the client. This approach makes a lot of technical sense, but it doesn’t make for a great user experience. [4] With Ajax, web applications can send data to, and retrieve data from, a server asynchronously without interfering with the display and behavior of the existing page.

Through Ajax, a web page feels like a desktop application. [3]



Figure 1: The synchronous interaction pattern of a traditional web application (top) compared with the asynchronous pattern of an Ajax application (bottom)

*http://www.adaptivepath.com/uploads/archive/images/publications/essays/ajax-fig2\_small.png*

### 1.5.1: Why Ajax is widely used in Web Applications

* They can use a standard web browser, such as Firefox, Internet Explorer or Safari, as their only user interface.
* They don't force the user to wait for the web server every time the user clicks a button. This is what "asynchronous" means. For instance, Gmail fetches new email messages in the background ("asynchronously") without forcing the user to wait. This makes an AJAX application respond much more like a "real" application on the user's computer, such as Microsoft Outlook.
* The Ajax engine works within the Web browser (through JavaScript and the DOM) to render the Web application and handle any requests that the customer might have of the Web server. The beauty of it is that because the Ajax engine is handling the requests, it can hold most information in the engine itself, while allowing the interaction with the application and the customer to happen as asynchronously and independently of any interaction with the server.
* They use standard JavaScript features (including the unofficial XMLHTTPRequest standard, pioneered by Microsoft and adopted by Firefox and other browsers) to fetch data in the background and display different email messages or other data "on the fly" when the user clicks on appropriate parts of the interface.
* Usually they manipulate data in XML format. This allows AJAX applications to interact easily with server-side code written in a variety of languages, such as PHP, Perl/CGI, Python and ASP.NET. Using XML isn't absolutely necessary, and in fact many "AJAX" applications don't -- they use the XMLHTTPRequest object to send and receive data "on the fly," but they don't actually bother packaging that data as XML.
* with Ajax, the JavaScript that is loaded when the page loads handles most of the basic tasks such as data validation and manipulation, as well as display rendering the Ajax engine handles without a trip to the server. At the same time that it is making display changes for the customer, it is sending data back and forth to the server. But the data transfer is not dependent upon actions of the customer. [5]

### 1.5.2: Technologies used in Ajax

JavaScript

* Loosely typed scripting language
* JavaScript function is called when an event in a page occurs
* Glue for the whole Ajax operation

DOM

* API for accessing manipulating structured documents
* Represents the structure of XML and HTML documents

CSS

* Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript

XMLHttpRequest

* JavaScript object that performs asynchronous interaction with the server [6]

## 1.6: Methodology

The methodology that was used in the Payroll System is Software Prototyping technique. The Software Prototyping technique refers to the activity of creating prototypes of software applications. A prototype typically simulates only a few aspects of, and may be completely different from the final product. The process of prototyping involves the following steps, which are identifying basic requirements, develop initial prototype, review, and revise and enhance the prototype.

## 1.7: Objectives

The objective of the project is to develop a web based payroll system using Ruby on Rails and Ajax technologies to:

* allow payroll administrator to create new or maintain existing employee details and the salary structure
* organize the entire employee’s salary structure in one database that can only be accessed by authorized administrators
* allow payroll administrator to define the deductions, tax, generate payslip, and generate various charts
* allow employee to maintain his/her own details
* allow employee to view payslip
* allow employee to generate various charts

## 1.8: Project write-up Layout

The dissertation is divided into seven chapters, briefly described as follow:

* Chapter Two: will demonstrate the literature review on traditional Payroll System that was not developed as Ajax web application.
* Chapter Three: Analysis and Requirements Specification reveals the purpose, goal, scope of the Web Based Payroll System and preliminary investigation. It also clearly specifies the functional and non-functional requirements, which identifies the complete specification of requirements for the system development.
* Chapter Four: System Design presents that how the system process flow and the design of the system process for the development of the system.
* Chapter Five: Implementation presents that how the system is being written and developed according to the design that has been developed to lead the implementation.
* Chapter Six: System testing presents the testing strategy and test case to test the Web Based Payroll System and test results of the system.

## 1.9: Limitations

In this final year project illustrating its concept, there are subjected to some limitations which include:

* The time constraint
* The allocated project budget
* Knowledge in developing the software
* Knowledge on developing good payroll system software
* Selection of material or database information based on its simplicity, availability, affordability, and reliability
* The project is only a prototype, where it is not intended for production use due to time constraint

# CHAPTER TWO

# LITERATURE REVIEW

In the process of completing this project, the developers have taken upon to properly research the literature that already exists for the Web Based Application, Single-Page Application, Ajax technology, Payroll System, and Model-View-Controller (MVC) design pattern. The purpose is to get a better understanding on the subject before going further into the research and design phase of the development.

As such this paper will detail the sources in which the developer refer to and the knowledge gathered by the process. The main point of this research consists of example, and type of preexisting system that currently in used for the purpose of student information.

## 2.1: Web based application

A web based application is an application that is accessed by users over a network such as the Internet or an intranet. The term may also mean a computer software application that is coded in a browser-supported programming language (such as JavaScript, combined with a browser-rendered markup language like HTML) and reliant on a common web browser to render the application executable.

Web based applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client, sometimes called a thin client. The ability to update and maintain web based applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. [7]

### 2.1.1: History

In earlier computing models, e.g. in client-server, the load for the application was shared between code on the server and code installed on each client locally. In other words, an application had its own client program which served as its user interface and had to be separately installed on each user's personal computer. An upgrade to the server-side code of the application would typically also require an upgrade to the client-side code installed on each user workstation, adding to the support cost and decreasing productivity.

In contrast, web applications use web documents written in a standard format such as HTML and JavaScript, which are supported by a variety of web browsers. Web applications can be considered as a specific variant of client-server software where the client software is downloaded to the client machine when visiting the relevant web page, using standard procedures such as HTTP. Client web software update may happen each time the web page is visited. During the session, the web browser interprets and displays the pages, and acts as the universal client for any web application.

In the early days of the Web each individual web page was delivered to the client as a static document, but the sequence of pages could provide an interactive experience, as user input is returned through web form elements embedded in the page markup.

In 1995 Netscape introduced a client-side scripting language called JavaScript allowing programmers to add some dynamic elements to the user interface that ran on the client side. So instead of sending data to the server in order to generate an entire web page, the embedded scripts of the downloaded page can perform various tasks such as input validation or showing/hiding parts of the page.

In 1996, Macromedia introduced Flash, a vector animation player that could be added to browsers as a plug-in to embed animations on the web pages. It allowed the use of a scripting language to program interactions on the client side with no need to communicate with the server.

In 1999, the "web application" concept was introduced in the Java language in the Servlet Specification version 2.2. At that time both JavaScript and XML had already been developed, but Ajax had still not yet been coined and the XMLHttpRequest object had only been recently introduced on Internet Explorer 5 as an ActiveX object.

In 2005, the term Ajax was coined, and applications like Gmail started to make their client sides more and more interactive. A web page script is able to contact the server for storing/retrieving data without downloading an entire web page.

In 2011, HTML5 was finalized, which provides graphic and multimedia capabilities without the need of client side plugins. HTML5 also enriched the semantic content of documents. The APIs and document object model (DOM) are no longer afterthoughts, but are fundamental parts of the HTML5 specification. WebGL API paved the way for advanced 3D graphics based on HTML5 canvas and JavaScript language. These have significant importance in creating truly platform and browser independent rich web applications. [7]

### 2.1.2: Interface

Web developers often use client-side scripting to add functionality, especially to create an interactive experience that does not require page reloading. Recently, technologies have been developed to coordinate client-side scripting with server-side technologies such as PHP. Ajax, a web development technique using a combination of various technologies, is an example of technology which creates a more interactive experience. [7]

### 2.1.3: Structure

Applications are usually broken into logical chunks called "tiers", where every tier is assigned a role. Traditional applications consist only of 1 tier, which resides on the client machine, but web applications lend themselves to a n-tiered approach by nature. Though many variations are possible, the most common structure is the three-tiered application. In its most common form, the three tiers are called presentation, application and storage, in this order. A web browser is the first tier (presentation), an engine using some dynamic Web content technology (such as ASP, ASP.NET, CGI, ColdFusion, JSP/Java, PHP, Perl, Python, Ruby on Rails or Struts2) is the middle tier (application logic), and a database is the third tier (storage). The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface. [7]

### 2.1.4: Business use

An emerging strategy for application software companies is to provide web access to software previously distributed as local applications. Depending on the type of application, it may require the development of an entirely different browser-based interface, or merely adapting an existing application to use different presentation technology. These programs allow the user to pay a monthly or yearly fee for use of a software application without having to install it on a local hard drive. A company which follows this strategy is known as an application service provider (ASP), and ASPs are currently receiving much attention in the software industry.

Security breaches on these kinds of applications are a major concern because it can involve both enterprise information and private customer data. Protecting these assets is an important part of any web application and there are some key operational areas that must be included in the development process. This includes processes for authentication, authorization, asset handling, input, and logging and auditing. Building security into the applications from the beginning can be more effective and less disruptive in the long run.

In cloud computing model web applications are software as a service (SaaS). There are business applications provided as SaaS for enterprises for fixed or usage dependent fee. Other web applications are offered free of charge, often generating income from advertisements shown in web application interface. [7]

### 2.1.5: Web application development

Writing of web applications is often simplified by open source software such as Django, Ruby on Rails or Symfony called web application frameworks. These frameworks facilitate rapid application development by allowing a development team to focus on the parts of their application which are unique to their goals without having to resolve common development issues such as user management. While many of these frameworks are open source, this is by no means a requirement.

The use of web application frameworks can often reduce the number of errors in a program, both by making the code simpler, and by allowing one team to concentrate on the framework while another focuses on a specified use case. In applications which are exposed to constant hacking attempts on the Internet, security-related problems can be caused by errors in the program. Frameworks can also promote the use of best practices such as GET after POST.

In addition, there is potential for the development of applications on Internet operating systems, although currently there are not many viable platforms that fit this model. [7]

### 2.1.6: Benefits

* Web applications do not require any complex "roll out" procedure to deploy in large organizations. A compatible web browser is all that is needed.
* Browser applications typically require little or no disk space on the client.
* They require no upgrade procedure since all new features are implemented on the server and automatically delivered to the users.
* Web applications integrate easily into other server-side web procedures, such as email and searching.
* They also provide cross-platform compatibility in most cases (i.e., Windows, Mac, Linux, etc.) because they operate within a web browser window.
* With the advent of HTML5, programmers can create richly interactive environments natively within browsers. Included in the list of new features are native audio, video and animations, as well as improved error handling. [7]

### 2.1.7: Drawbacks

* In practice, web interfaces, compared to thick clients, typically force significant sacrifice to user experience and basic usability.
* Web applications absolutely require compatible web browsers. If a browser vendor decides not to implement a certain feature, or abandons a particular platform or operating system version, this may affect a huge number of users.
* Standards compliance is an issue with any non-typical office document creator, which causes problems when file sharing and collaboration becomes critical.
* Browser applications rely on application files accessed on remote servers through the Internet. Therefore, when connection is interrupted, the application is no longer usable. However, if it uses HTML5 API's such as Offline Web application caching,[10] it can be downloaded and installed locally, for offline use. Google Gears, although no longer in active development, is a good example of a third party plugin for web browsers that provides additional functionality for creating web applications.
* Since many web applications are not open source, there is also a loss of flexibility, making users dependent on third-party servers, not allowing customizations on the software and preventing users from running applications offline (in most cases). However, if licensed, proprietary software can be customized and run on the preferred server of the rights owner.
* They depend entirely on the availability of the server delivering the application. If a company goes bankrupt and the server is shut down, the users have little recourse. Traditional installed software keeps functioning even after the demise of the company that produced it (though there will be no updates or customer service).
* Likewise, the company has much greater control over the software and functionality. They can roll out new features whenever they wish, even if the users would like to wait until the bugs have been worked out before upgrading. The option of simply skipping a weak software version is often not available. The company can foist unwanted features on the users or cut costs by reducing bandwidth. Of course, companies will try to keep the good will of their customers, but the users of web applications have fewer options in such cases unless a competitor steps in and offer a better product and easy migration.
* The company can theoretically track anything the users do. This can cause privacy problems. [7]

## 2.2: Single-page application

A single-page application (SPA), also known as single-page interface (SPI), is a web application or web site that fits on a single web page with the goal of providing a more fluid user experience akin to a desktop application.

In an SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or partial changes are performed loading new code on demand from the web server, usually driven by user actions. The page does not automatically reload during user interaction with the application, nor does control transfer to another page. Updates to the displayed page may or may not involve interaction with a server. [8]

### 2.2.1: Architectural characteristics

#### 2.2.1.1: The problem

The way traditional web applications work causes disruption in the user experience and workflow.

Traditional web applications work by reloading the entire web page. In order to advance through a workflow, the user interacts with page elements (such as hyperlinks and form submit buttons) that cause the browser to issue a request to the server for a completely new page.

**User experience**: Continual page redraws disrupt the user experience because the network latencies cannot be hidden from the user. There is typically a perceivable transitional jolt from one page to the next. The next page's data is retrieved from the server, the old page is unloaded, and the new page is rendered to screen.

Stable UI affordances, such as toolbars, navigation elements, database query results and so forth, continually disappear and reappear.

**Performance**: The complete page reload that occurs on each user interaction results in unnecessary re-transmission of data over the wire. This can make the overall performance of the web-site, when the entire session is taken into account, slower.

#### 2.2.1.2: How SPAs address the problem

SPAs address these issues by requiring no page reload by the browser during an application session. All user interaction and changes of the application state are handled in the context of a single Web document.

The user experience becomes more continuous and fluid, and network latencies can be hidden more easily. [8]

### 2.2.2: Technical approaches

There are various techniques available that enable the browser to retain a single page even when the application requires server communication.

**AJAX**

The most prominent technique currently being used is Ajax. Predominantly using the XMLHttpRequest object from JavaScript, other AJAX approaches include using IFRAME or script HTML elements. Popular libraries like jQuery, that normalize AJAX behavior across browsers from different manufacturers, have further popularized the AJAX technique.

**Node.js/SignalR**

Asynchronous calls to the server may also be achieved using Node.js or SignalR in conjunction with Socket.io.

**Browser plugins**

Asynchronous calls to the server may also be achieved using browser plug-in technologies such as Silverlight, Flash or Java applets.

**Data transport (XML, JSON and AJAX)**

Requests to the server typically result in either raw data (e.g. XML or JSON), or new HTML being returned. In the case where HTML is returned by the server, JavaScript on the client updates a partial area of the DOM (Document Object Model). When raw data is returned, oftentimes a client-side JavaScript XML / (XSL) process (and in case of JSON a template) is used to translate the raw data into HTML, which is then used to update a partial area of the DOM.

**Thin server architecture**

An SPA moves logic from the server to the client. This results in the role of the web server evolving into a pure data API or web service. This architectural shift has, in some circles, been coined "Thin Server Architecture" to highlight that complexity has been moved from the server to the client, with the argument that this ultimately reduces overall complexity of the system.

**Thick server architecture**

The server keeps the necessary state in memory of the client state of the page. In this way, when any request hits the server (usually user actions), the server sends the appropriate HTML and/or JavaScript with the concrete changes to bring the client to the new desired state (usually adding/deleting/updating a part of the client DOM). At the same time the state in server is updated. Most of the logic is executed in server and HTML is usually also rendered in server. In some ways the server simulates a web browser, receiving events and performing delta changes in server state which are automatically propagated to client. This approach needs more server memory and server processing, but the advantage is a simplified development model because a) the application is usually fully coded in server, b) data and UI state in server are shared in the same memory space with no need of custom client/server communication bridges. [8]

### 2.2.3: Running locally

Some SPAs may be executed from a local file using the file URI scheme. This gives users the ability to download the SPA from a server and run the file from a local storage device, without depending on server connectivity. If such an SPA wants to store and update data, it must be self-modifying. That is, the SPA must be capable of writing itself to disk, including a representation of the state that is to be persisted. These applications benefit from advances available with HTML5, particularly Web Storage. [8]

### 2.2.4: Challenges with the SPA model

Because the SPA is an evolution away from the stateless page-redraw model that browsers were originally designed for, some new challenges have emerged. Each of these problems has an effective solution with:

* Client-side JavaScript libraries addressing various issues.
* Server side web frameworks that specialize in the SPA model.
* The evolution of browsers and the HTML5 specification aimed at the SPA model.

#### 2.2.4.1: Browser history

With an SPA being, by definition, "a single page", the model breaks the browser's design for page history navigation using the Forward/Back buttons. This presents a usability impediment when a user presses the back button, expecting the previous screen state within the SPA, but instead the application's single page unloads and the previous page in the browser's history is presented.

The traditional solution for SPA's has been to change the browser URL's hash fragment identifier in accord with the current screen state. This can be achieved with JavaScript, and causes URL history events to be built up within the browser. As long as the SPA is capable of resurrecting the same screen state from information contained within the URL hash, the expected back button behavior is retained.

To further address this issue, the HTML5 specification has introduced pushState and replaceState providing programmatic access to the actual URL and browser history. [8]

### 2.2.5: Page lifecycle

An SPA is fully loaded in the initial page load and then page regions are replaced or updated with new page fragments loaded from the server on demand. To avoid excessive downloading of unused features, an SPA will often progressively download more features as they become required, either small fragments of the page, or complete screen modules.

In this way an analogy exists between "states" in an SPA and "pages" in a traditional web site. Because "state navigation" in the same page is analogous to page navigation, in theory any page based web site could be converted to single-page replacing in the same page only the changed parts result of comparing consecutive pages in a non-SPA.

The SPA approach on the web is similar to the Single Document Interface (SDI) presentation technique popular in native desktop applications. [8]

## 2.3: Ajax

Ajax (also AJAX; an acronym for Asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. With Ajax, web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data can be retrieved using the XMLHttpRequest object. Despite the name, the use of XML is not required (JSON is often used instead), and the requests do not need to be asynchronous.

Ajax is not a single technology, but a group of technologies. HTML and CSS can be used in combination to mark up and style information. The DOM is accessed with JavaScript to dynamically display, and to allow the user to interact with the information presented. JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads. [3]

[](http://www.adaptivepath.com/uploads/archive/images/publications/essays/ajax-fig1.png)

Figure 2: The traditional model for web applications (left) compared to the Ajax model (right).

*http://www.adaptivepath.com/uploads/archive/images/publications/essays/ajax-fig1\_small.png*

### 2.3.1: History

In the 1990s, most web sites were based on complete HTML pages. Each user action required that the page be reloaded from the server (or a new page loaded). This process was inefficient, as reflected by the user experience: all page content disappeared then reappeared. Each time a page was reloaded due to a partial change, all of the content had to be re-sent, even though only some of the information had changed. This placed additional load on the server and used excessive bandwidth.

In 1996, the iframe tag was introduced by Internet Explorer to load content asynchronously.

In 1998, Microsoft Outlook Web Access team implemented the first component XMLHTTP by client script.

In 1999, Microsoft utilized its iframe technology to dynamically update the news stories and stock quotes on the default page for Internet Explorer, and created the XMLHTTP ActiveX control in Internet Explorer 5, which was later adopted by Mozilla, Safari, Opera and other browsers as the XMLHttpRequest JavaScript object. Microsoft has adopted the native XMLHttpRequest model as of Internet Explorer 7, though the ActiveX version is still supported. The utility of background HTTP requests to the server and asynchronous web technologies remained fairly obscure until it started appearing in full scale online applications such as Outlook Web Access (2000) and Oddpost (2002).

Google made a wide deployment of standards-compliant, cross browser Ajax with Gmail (2004) and Google Maps (2005).

The term Ajax was coined on 18 February 2005 by Jesse James Garrett in an article entitled "Ajax: A New Approach to Web Applications", based on techniques used on Google pages.

On 5 April 2006 the World Wide Web Consortium (W3C) released the first draft specification for the XMLHttpRequest object in an attempt to create an official web standard. [3]

### 2.3.2: Technologies

The term Ajax has come to represent a broad group of web technologies that can be used to implement a web application that communicates with a server in the background, without interfering with the current state of the page. The following technologies are incorporated in Ajax:

* HTML (or XHTML) and CSS for presentation
* The Document Object Model (DOM) for dynamic display of and interaction with data
* XML for the interchange of data, and XSLT for its manipulation
* The XMLHttpRequest object for asynchronous communication
* JavaScript to bring these technologies together

There have been a number of developments in the technologies used in an Ajax application, and the definition of the term Ajax. XML is not required for data interchange and therefore XSLT is not required for the manipulation of data. JavaScript Object Notation (JSON) is often used as an alternative format for data interchange, although other formats such as preformatted HTML or plain text can also be used. [3]

### 2.3.3: Drawbacks

* In pre-HTML5 browsers, pages dynamically created using successive Ajax requests did not automatically register themselves with the browser's history engine, so clicking the browser's "back" button may not have returned the browser to an earlier state of the Ajax-enabled page, but may have instead returned to the last full page visited before it. Such behavior — navigating between pages instead of navigating between page states — may be desirable, but if fine-grained tracking of page state is required then a pre-Ajax workaround was to use invisible iframes to trigger changes in the browser's history. A workaround implemented by Ajax techniques is to change the URL fragment identifier (the part of a URL after the '#') when an Ajax-enabled page is accessed and monitor it for changes. HTML5 provides an extensive API standard for working with the browser's history engine.
* Dynamic web page updates also make it difficult to bookmark and return to a particular state of the application. Solutions to this problem exist, many of which again use the URL fragment identifier. The solution provided by HTML5 for the above problem also applies for this.
* Depending on the nature of the Ajax application, dynamic page updates may interfere disruptively with user interactions, especially if working on an unstable Internet connection. For instance, editing a search field may trigger a query to the server for search completions, but the user may not know that a search completion popup is forthcoming, and if the internet connection is slow, the popup list may show up at an inconvenient time, when the user has already proceeded to do something else.
* Because most web crawlers do not execute JavaScript code, publicly indexable web applications should provide an alternative means of accessing the content that would normally be retrieved with Ajax, thereby allowing search engines to index it.
* Any user whose browser does not support JavaScript or XMLHttpRequest, or simply has this functionality disabled, will not be able to properly use pages which depend on Ajax. Devices such as smartphones and PDAs may not have support for the required technologies, though this is becoming less of an issue. The only way to let the user carry out functionality is to fall back to non-JavaScript methods. This can be achieved by making sure links and forms can be resolved properly and not relying solely on Ajax.
* Similarly, some web applications which use Ajax are built in a way that cannot be read by screen-reading technologies, such as JAWS. The WAI-ARIA standards provide a way to provide hints in such a case.
* Screen readers that are able to use Ajax may still not be able to properly read the dynamically generated content.
* The same origin policy prevents some Ajax techniques from being used across domains, although the W3C has a draft of the XMLHttpRequest object that would enable this functionality. Methods exist to sidestep this security feature by using a special Cross Domain Communications channel embedded as an iframe within a page, or by the use of JSONP.
* The asynchronous callback-style of programming required can lead to complex code that is hard to maintain, to debug and to test. [3]

## 2.4: Payroll System

A payroll system involves everything that has to do with the payment of employees and the filing of employment taxes. This includes keeping track of hours, calculating wages, withholding taxes and other deductions, printing and delivering checks and paying employment taxes to the government.

Withholding and paying taxes is one of the most important responsibilities of the payroll system. In the United States, the following are the major withholdings required by the government:

* Federal income tax
* State and local income taxes (where applicable)
* Social Security tax
* Medicare tax

When an employer withholds taxes from a paycheck, he acts as the trustee for those funds until they are paid to the IRS, the Social Security Administration (SSA) or other government agency. To avoid confusing this money with profits or other business income, all withheld taxes must be held in a separate bank account or trust fund.

In the case of Social Security and Medicare withholdings, when it's time to hand that money over to the government, the employer is required to match the employee's contributions. For example, if an employee is paying 6.2 percent of every check for Social Security, then the employer has to pay an equal 6.2 percent.

In addition to matching Social Security and Medicare contributions, the employer has to pay federal and state unemployment taxes (FUTA and SUTA) for each employee. The employer pays these taxes himself, meaning nothing is withheld from the paycheck.

There are numerous other possible deductions, withholdings and contributions that can be subtracted from an employee's gross wages and that need to be tracked by the payroll system:

* Health insurance or life insurance premiums
* 401(k) or other retirement fund contributions
* Workman's compensation
* Union dues
* Vacation days
* Sick days
* Employee loans
* Court-ordered wag­e garnishments (for outstanding debts)
* Child support payments

At the end of the year, an employer uses the payroll system to take all of the wage and withholding information from the previous year and summarize it on a W-2 form for full-time employees or a 1099 form for contract workers. Copies of that form must be sent to the employee, the IRS and the SSA. [9]

## 2.5: Model-view-controller (MVC) design pattern

Model–view–controller (MVC) is a software architecture pattern that separates the representation of information from the user's interaction with it. The model consists of application data, business rules, logic, and functions. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible, such as a bar chart for management and a tabular view for accountants. The controller mediates input, converting it to commands for the model or view. The central ideas behind MVC are code reusability and separation of concerns. [10]

### 2.5.1: History

The model-view-controller pattern was originally formulated in the late 1970s by Trygve Reenskaug at Xerox PARC, as part of the Smalltalk system. [10]

### 2.5.2: Component interactions

In addition to dividing the application into three kinds of components, the MVC design defines the interactions between them.

* A controller can send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). It can also send commands to the model to update the model's state (e.g., editing a document).
* A model notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.
* A view requests from the model the information that it needs to generate an output representation. [10]

[](http://en.wikipedia.org/wiki/File:MVC-Process.png)

Figure 3: A typical collaboration of the MVC components.

*http://upload.wikimedia.org/wikipedia/commons/thumb/f/fd/MVC-Process.png/200px-MVC-Process.png*

### 2.5.3: Dependency hierarchy

There is usually a kind of hierarchy in the MVC pattern. The Model knows only about itself. That is, the source code of the Model has no references to either the View or Controller.

The View however, knows about the Model. It will poll the Model about the state, to know what to draw. That way, the View can draw something that is based on what the Model has done. But the View knows nothing about the Controller.

The Controller knows about both the Model and the View. To take an example from a game: If you click on the "fire" button on the mouse, the Controller knows what fire function in the Model to call. If you press the button for switching between first and third person, the Controller knows what function in the View to call to request the display change.

The reason to keep it this way is to minimize dependencies. No matter how the View class is modified, the Model will still work. Even if the system is moved from Windows to an app in a smart phone, the Model can be moved with no changes. But the View probably needs to be updated, as will the Controller. [10]

### 2.5.4: Use in web applications

Although originally developed for personal computing, Model View Controller has been widely adapted as architecture for World Wide Web applications in all major programming languages. Several commercial and noncommercial application frameworks have been created that enforce the pattern. These frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server.

Early web MVC frameworks took a thin client approach that placed almost the entire model, view and controller logic on the server. In this approach, the client sends either hyperlink requests or form input to the controller and then receives a complete and updated web page (or other document) from the view; the model exists entirely on the server. As client technologies have matured, frameworks such as JavaScriptMVC and Backbone have been created that allow the MVC components to execute partly on the client. [10]

# CHAPTER THREE

# REQUIREMENT AND ANALYSIS

## 3.1: Requirements Analysis in general

Requirements Analysis is the process of determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software or system requirements. [1]

Requirements are description of how a system should behave or a description of system properties or attributes. It can alternatively be a statement of ‘what’ an application is expected to do. [1]

Requirements analysis includes three types of activities:

* Eliciting requirements: the task of identifying the various types of requirements from customers and users. This is sometimes also called requirements gathering.
* Analyzing requirements: determining whether the stated requirements are clear, complete, consistent and unambiguous, and resolving any apparent conflicts.
* Recording requirements: Requirements may be documented in various forms, usually including a summary list and may include natural-language documents, use cases, user stories, or process specifications. [1]

Requirements analysis can be a long and arduous process during which many delicate psychological skills are involved. New systems change the environment and relationships between people, so it is important to identify all the stakeholders, take into account all their needs and ensure they understand the implications of the new systems. Analysts can employ several techniques to elicit the requirements from the customer. These may include the development of scenarios (represented as user stories in agile methods), the identification of use cases, the use of workplace observation or ethnography, holding interviews, or focus groups (more aptly named in this context as requirements workshops,

or requirements review sessions) and creating requirements lists. Prototyping may be used to develop an example system that can be demonstrated to stakeholders. Where

necessary, the analyst will employ a combination of these methods to establish the exact requirements of the stakeholders, so that a system that meets the business needs is produced. [1]

In general, requirements are partitioned into functional requirements and non-functional requirements. Functional requirements are associated with specific functions, tasks or behaviors the system must support (e.g. the system must output the data in json format, or the system must provide the login function). While non-functional requirements are constraints on various attributes of these functions or tasks. The functional requirements address the quality characteristic of functionality while the other quality characteristics are concerned with various kinds of non-functional requirements (e.g. the system must be easily accessible, the quality must follow a particular standard). Because non-functional requirements tend to be stated in terms of constraints on the results of tasks which are

given as functional requirements (e.g. constraints on the speed or efficiency of a given task)?

Software prototyping are used to gather the functional and non-functional requirements.

### 3.1.1: Software prototyping

Software prototyping refers to the activity of creating prototypes of software applications,

i.e., incomplete versions of the software program being developed. It is an activity that can occur in software development and is comparable to prototyping as known from other

fields. A prototype typically simulates only a few aspects of, and may be completely

different from the final product. The main purpose of a prototype is to allow users of the software to evaluate developers' proposals for the design of the eventual product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions. Prototyping can also be used by end users to describe and prove requirements that developers have not considered. [2]

### 3.1.2: Outline of the prototyping process

The process of prototyping involves the following steps

1. Identify basic requirements

Determine basic requirements including the input and output information desired. Details, such as security, can typically be ignored.

1. Develop Initial Prototype

The initial prototype is developed that includes only user interfaces.

1. Review

The customers, including end-users, examine the prototype and provide feedback on additions or changes.

1. Revise and Enhance the Prototype

Using the feedback both the specifications and the prototype can be improved. Negotiation about what is within the scope of the contract/product may be

necessary. If changes are introduced then a repeat of steps #3 and #4 may be needed. [2]

## 3.2: Software Requirements Specification

A Software requirements specification (SRS), a requirements specification for a software system, is a complete description of the behavior of a system to be developed and may include a set of use cases that describe interactions the users will have with the software. In addition it also contains non-functional requirements. [3]

We will discuss the Use case diagram on the next page.

### 3.2.1: USE CASE Diagram



Figure 4: Use Case Diagram – Administrator/HR Administrator



Figure 5: Use Case Diagram - Employee

Table 1: Actors Grid

|  |  |  |
| --- | --- | --- |
| ID | Name | Related use cases |
| 1 | Administrator/HR Administrator | Manage User,  Manage Employee,  View Attendance,  Manage Pay Rate,  Manage Job Title,  Manage Employment Status,  Manage Job Category,  Manage Department,  Manage Overtime Pay Rate,  Manage Salary Adjustment,  View Overtime Chart,  View Total Work Hours Chart,  View Hourly Payroll Chart,  View Payslip |
| 2 | Monthly Paid Employee | Update Personal Details,  View Job Details,  View Salary Details,  View Total Work Hours Chart,  View Overtime Chart,  View Payslip |
| 3 | Hourly Paid Employee | Update Personal Details,  View Job Details,  View Salary Details,  View Total Work Hours Chart,  View Hourly Payroll Chart,  View Payslip |

Table 2: Use Cases Grid

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | Primary actors | Supporting actors |
| 1 | Manage User | Administrator/HR Administrator |  |
| 2 | Manage Employee | Administrator/HR Administrator |  |
| 3 | View Attendance | Administrator/HR Administrator |  |
| 4 | Manage Pay Rate | Administrator/HR Administrator |  |
| 5 | Manage Job Title | Administrator/HR Administrator |  |
| 6 | Manage Employment Status | Administrator/HR Administrator |  |
| 7 | Manage Job Category | Administrator/HR Administrator |  |
| 8 | Manage Department | Administrator/HR Administrator |  |
| 9 | Manage Overtime Pay Rate | Administrator/HR Administrator |  |
| 10 | View Overtime Chart | Administrator/HR Administrator,  Monthly Paid Employee |  |
| 11 | View Total Work Hours Chart | Administrator/HR Administrator,  Monthly Paid Employee,  Hourly Paid Employee |  |
| 12 | Manage Salary Adjustment | Administrator/HR Administrator |  |
| 13 | View Hourly Payroll Chart | Administrator/HR Administrator,  Hourly Paid Employee |  |
| 14 | View Payslip | Administrator/HR Administrator,  Monthly Paid Employee,  Hourly Paid Employee |  |
| 15 | Update Personal Details | Monthly Paid Employee,  Hourly Paid Employee |  |
| 16 | View Job Details | Monthly Paid Employee,  Hourly Paid Employee |  |
| 17 | View Salary Details | Monthly Paid Employee,  Hourly Paid Employee |  |

Table 3: Use Case Diagram Summary

|  |  |
| --- | --- |
| Name | Documentation |
| Image1.png [Administrator/ HR](#MA0pDpSGAqACA1AL) Administrator | Manage employee details, manage salary structure for each employee, and generate employee payslip at end of month. |
| Image2.png [Manage](#Ji0JDpSGAqACAzOd) User | Create, Update, and Delete user. A user is required in order to access the system. |
| Image2.png [Manage](#V3TJDpSGAqACA0Gc) Employee | Create, Update, and Delete employee. Employee details include personal details, contact details, job details, salary details, and qualification details. |
| Image2.png [View](#nd3JDpSGAqACA0Yy) Attendance | View employee’s attendance record. |
| Image2.png Manage Pay Rate | Create, Update, and Delete pay rate. |
| Image2.png [Manage](#z6rZDpSGAqACA371) Job Title | Create, Update, and Delete job title. |
| Image2.png [Manage](#iGWZDpSGAqACA3eV) Employment Status | Create, Update, and Delete employment status. |
| Image2.png [Manage](#j5nZDpSGAqACA4Cr) Job Category | Create, Update, and Delete job category. |
| Image2.png [Manage](#_GvZDpSGAqACA4Jd) Department | Create, Update, and Delete department. |
| Image2.png [Manage](#0qu5DpSGAqACA5TJ) Overtime Pay Rate | Create, Update, and Delete overtime pay rate. |
| Image2.png [Manage](#eH05DpSGAqACA429) Salary Adjustment | Create, Update, and Delete salary adjustment. |
| Image2.png [View](#E615DpSGAqACA5r1) Overtime Chart | View monthly employee’s overtime chart. |
| Image2.png [View](#NeT5DpSGAqACA5zR) Total Hours Worked Chart | View hourly paid employee’s total hours worked chart. |
| Image2.png [View](#5zCFDpSGAqACA6p9) Hourly Payroll Chart | View employee’s payroll chart. |
| Image2.png [View](#AKA5DpSGAqACA4QH) Payslip | The home page of the user where all the friend's, group's, page's, activities appears. |
| Image1.png Monthly Paid [Employee](#rQWpDpSGAqACA1CD)/Hourly Paid Employee | The employee who can access system to update personal details, view salary details, view payroll charts, and generate payslip. |
| Image2.png Update Personal Details | Allows authenticated employee to update his/her personal details. |
| Image2.png View Job Details | Allows authenticated employee to view his/her job details. |
| Image2.png View Salary Details | Allows authenticated employee to view his/her salary details. |
| Image2.png View Total Hours Worked Chart | Allows authenticated hourly paid employee to view his/her total number of hours worked chart. |
| Image2.png View Hourly Payroll Chart | Allows authenticated hourly paid employee to view his/her hourly payroll chart. |
| Image2.png View Overtime Chart | Allows authenticated monthly paid employee to view his/her overtime chart. |
| Image2.png View Payslip | Allows authenticated employee to view/print his/her payslip. |
| Image3.png [Payroll](#ovOhDpSGAqACAyo2) System | A system which allows Administrator to manage employee details, manage salary structure for each employee, and generate employee payslip at end of month, and allows employee to update personal details, view salary details, view payroll charts, and generate payslip. |

### 3.2.2: Requirement Diagram



### 3.2.3: Operating Environment

In computing, an operating environment is the environment in which users run

application software, whether by a command-line interface (such as in MS-DOS or

the UNIX shell) or a graphical user interface (such as in the Macintosh operating

system or a web browser). [4] Table X has briefly shows the operating environment

of the Web based Payroll System.

Table 4: Operating Environment

|  |  |  |
| --- | --- | --- |
| **No** | **Requirement** | **Reason** |
| 1 | Google Chrome | Google Chrome web browser best work with the application |
| 2 | Any Operating System with decent web browser | To access the system via local intranet or internet |
| 3 | Ruby on Rails with jQuery UI as the client side user interface, jQuery as the javascript library, and Highcharts to display charts | To develop the system |

### 3.2.4: Constraints and Dependencies

The Payroll System is dependent on jQuery [5], JQuery UI [6], and Highcharts [7]

JavaScript library. jQuery is used to send Ajax request to Ruby code on the server

side, resulting in a fast and asynchronous data transmission in a web browser. Meanwhile jQuery UI is used to create highly interactive web interface.

Highcharts is used to display intuitive and interactive charts in the payroll web application.

**3.2.4.1: Dependency on jQuery**

The system is completely dependent on the jQuery as the system is a fully ajax enabled web application. The goal of using jQuery is because it is fast, small, and feature-rich

JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

**3.2.4.2: Dependency on jQuery UI**

The system uses this library to create the front end web interface. jQuery UI is built on top

of jQuery JavaScript library that provides abstractions for low-level interaction and

animation, advanced effects and high-level, themeable widgets.

This library is completely dependent on the jQuery library.

**3.2.4.3: Dependency on Highcharts**

Highcharts is a charting library written in pure JavaScript, which offers intuitive, interactive charts to a web application. The system uses this library to show various chart types, such as line, bar, and pie.

Table 5: Chart used

|  |  |  |
| --- | --- | --- |
| **Category** | **Chart Type** | **Description** |
| Overtime Chart | Bar Chart | Shows monthly paid employee’s overtime record for a particular year. |
| Total Work Hours Chart | Line Chart | Shows hourly paid employee’s total working hours worked record for a particular year. |
| Hourly Payroll Chart | Pie Chart & Bar Chart | Shows hourly paid employee’s payroll amount for a particular year. |

**3.2.5: System Features requirement**

Section below will illustrate the required features of the Payroll System. The feature requirements fall under the function requirements because they interact directly with the user. [8]

#### 3.2.5.1: Feature 1: User CRUD functions

Create, Read, Update, and Delete User. This feature allows administrator to create or maintain users which allow to access the system. Each user can be enabled or disabled. If

a user is disabled, the user will not able to access the system. The administrator can

search by username, employee name, user role, and status.

#### 3.2.5.2: Feature 2: Employee CRUD functions

Create, Read, Update, and Delete Employee. This feature allows administrator to create or maintain employees in the system. Each employee can be associated with a user to allow them to access the system. The administrator can search by employee, employment

status, department, staff id, and job title.

**3.2.5.3: Feature 3: View Attendance**

This feature allows the administrator to view employee’s attendance record.

The administrator can search by employee and working date.

**3.2.5.4: Feature 4: Job Title CRUD**

Create, Read, Update, and Delete Job Title. This feature allows administrator to create or maintain job titles in the system. Each employee should have a job title. The administrator can search by keyword.

**3.2.5.5: Feature 5: Employment Status CRUD**

Create, Read, Update, and Delete Employment Status. This feature allows administrator to create or maintain employment statuses in the system. Each employee should have an employment status. Example of employment status would be Confirmed, Probation. The administrator can search by keyword.

**3.2.5.6: Feature 6: Job Category CRUD**

Create, Read, Update, and Delete Job Category. This feature allows administrator to

create or maintain job categories in the system. Each employee should have a job

category. The administrator can search by keyword.

**3.2.5.7: Feature 7: Department CRUD**

Create, Read, Update, and Delete Department. This feature allows administrator to create

or maintain departments in the system. Each employee should belong to a department.

The administrator can search by keyword.

**3.2.5.8: Feature 8: Pay Rate CRUD**

Create, Read, Update, and Delete Pay Rate. This feature allows administrator to create or maintain pay rates in the system. The administrator can define pay rate for each hourly

paid employee, for each year. The administrator can search by staff id, year, and month.

**3.2.5.9: Feature 9: Overtime Pay Rate CRUD**

Create, Read, Update, and Delete Overtime Pay Rate. This feature allows administrator to create or maintain overtime pay rates in the system. The administrator can define rate for each year. The administrator can search by year.

**3.2.5.10: Feature 10: Salary Adjustment CRUD**

Create, Read, Update, and Delete Salary Adjustment. This feature allows administrator to create or maintain salary adjustments in the system. The administrator can define adjustment for a particular employee, for a particular year. The administrator can search by staff id, year, and month.

**3.2.5.11: Feature 11: Overtime Chart**

Shows monthly paid employee’s overtime record for a particular year. The administrator can see the chart for a particular employee and for a particular year. It also has option to select months for the chart to be generated. The employee also can see his/her own chart.

**3.2.5.12: Feature 12: Total Hours Worked Chart**

Shows hourly paid employee’s total working hours worked record for a particular year.

The administrator can see the chart for a particular employee and for a particular year. It also has option to select months for the chart to be generated. The employee also can see his/her own chart.

**3.2.5.13: Feature 13: Hourly Payroll Chart**

Shows hourly paid employee’s payroll amount for a particular year. The administrator can see the chart for a particular employee and for a particular year. It also has option to select months for the chart to be generated. The employee also can see his/her own chart.

**3.2.5.14: Feature 14: View Payslip**

Shows employee payslip. The administrator and employee can view and print the payslip.

**3.2.5.15: Feature 15: Update Personal Details**

Allows the employee to update the personal details, including contact details and qualification details.

**3.2.5.16: Feature 16: View Job Details**

Allows the employee to view the job details.

**3.2.5.17: Feature 17: View Salary Details**

Allows the employee to see his/her own salary details.

# CHAPTER FOUR

# DESIGN

## 4.1: Design in Software Engineering

A software design is a meaningful engineering representation of some software product that is to be built. It can be traced to the customer’s requirements and can be accessed for quality against predefined criteria. In the software engineering context, design focuses on four major areas of concern: data, architecture, interfaces, and components. [1]

The design process is very important, because the emphasis in design is on quality; therefore it provides the representation of software that can be accessed for quality.

Furthermore, this is the only phase in which the customer’s requirements can be accurately translated into a finished software product or system. Thus, software design serves as the foundation for all software engineering steps that follow regardless of which process model is being employed. [1]

During the design process the software specifications are transformed into design models that describe the details of the data structures, system architecture, interface and components. Each design product is reviewed for quality before moving to the next phase of software development. At the end of the design process a design specification document is produced, which is composed of the design models that describe the data, architecture, interfaces and components. [1]



Figure 6: Generic Overview of Web Based Payroll System

## 4.2: Flow of the data



Figure 7: Level 0 Data Flow Diagram



Figure 8: Level 1 Data Flow Diagram

## 4.3: Class Diagram

A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them. A class is the description of a set of objects having similar attributes, operations, relationships and behavior.



Figure 9: Class Diagram of Domain Model



Figure 10: Class Diagram of Controller

## 4.4: Overall Sequence diagram

A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. A sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur.



## 4.5: Website Architecture

Website architecture is an approach to the design and planning of websites which, like architecture itself, involves technical, aesthetic and functional criteria. As in traditional architecture, the focus is properly on the user and on user requirements. This requires particular attention to web content, a business plan, usability, interaction design, information architecture and web design.

"Website architecture" has the potential to be a term used for the intellectual discipline of organizing website content. "Web design", by way of contrast, describes the practical tasks, part-graphic and part-technical, of designing and publishing a website.

Before I got started with anything, I had to keep in mind few design guidelines to lead the project design. The design of the website should utilize the functionality of AJAX in order to make the web as user friendly as possible.

I ensure the following so AJAX can be utilized.

* Ensure no page reloads by dynamically update the content which needs to be updated.
* Ensure only JSON, html, and plain text format to be used in the data transmission between the client side and the server side.
* Ensure there is a progress status on every AJAX request is made so that the user is aware that the request is being sent to the server.

### 4.5.1: Architecture Model

There are quite a few website architecture models that can be adopted, depends on the type of website you are developing. Following are some of the architecture models; I will briefly explain three models and then the model that has been used to develop the Web Based Payroll System.

Website Architecture Models:

#### 4.5.1.1: All-in-one model



Figure 11: All-in-one Architecture Model

*http://www.webdesignfromscratch.com/snippets/ia\_diagram\_allinone.gif*

This is the simplest possible model. Everything goes on a single Home page.

#### 4.5.1.2: Flat model



Figure 12: Flat Architecture Model

*http://www.webdesignfromscratch.com/snippets/ia\_diagram\_monocline2.gif*

A flat pattern is where all pages are arranged as peers, and everyone is accessible from every other one. This is very common for simple sites, where there are a few standard topics, such as: Home, About Us, Contact Us, Products.

*I will be using flat model for the Web Based Payroll System so the accessing and navigation can be simpler and easier.*

#### 4.5.1.3: Hub-and-spoke / Daisy model



Figure 13: Hub-and-spoke (or Daisy) pattern

*http://www.webdesignfromscratch.com/snippets/ia\_diagram\_hub\_and\_spoke.gif*

This model is useful for multiple, distinct linear workflows. A good example may be an email application, where you will return to your inbox at several points, e.g. after reading a message, after sending a message, or after adding a new contact.

## 4.6: Navigation Design

Navigation design is the design of moving from one page, content, or area of the website to another. It organizes in such a way that the user will be navigating from one page to another more easily. As of Web Based Payroll System, the user will navigate using the mouse to interact with the web user interface.

### 4.6.1: Accordion

Accordion is a widget which displays collapsible content panels for presenting information in a limited amount of space.

*Accordion is used in the Web Based Payroll System to provide navigation in the web application.*



Figure 14: Web Based Payroll System Accordion

### 4.6.2: Paging

Paging is a term that will be familiar to all web users. This is where you get a piece of content that spans several pages. You are given standard tools that let you navigate previous, next, or jump directly to specific page. In Web Based Payroll System user can choose whether 20, 50, or 100 results is shown upon the request. The previous/next button will be disabled if there is no previous/next page.



Figure 15: Paging interface of Web Based Payroll System

# CHAPTER FIVE

# IMPLEMENTATION

## 5.1: Implementation in Software Engineering

In software implementation phase, we build the components of the system, or we can say that implementation is the phase of start writing the system or start doing programming, based on the given architecture documentation from the design phase and the requirement document from the analysis phase. Hence, the team should build exactly what has been requested, though there is still room for innovation and flexibility. For instance, a component may be narrowly designed for this particular system, or the component may be made more general to satisfy a reusability guideline. Therefore, the architecture document should give guidance. Sometimes, this guidance is found in the requirement document. This phase deals with issues of quality, performance, baselines, libraries, and debugging. The end deliverable is the product itself and the source code of the system. [1]

## 5.2: Before writing the Web Based Payroll System

An implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system through programming and deployment. Many implementations may exist for a given specification or standard. For example, web browsers contain implementations of World Wide Web Consortium-recommended specifications, and software development tools contain implementations of programming languages. [2]

The implementation phase is a phase where you have to be very clear about each and every requirement of the customer because software implementation is a collaborative effort between the software vendor and the customer. Secondly, clear and open communication is essential. Of course, customer needs to communicate their objectives to the software vendor. But even more important, they must listen to what the software vendor tells you. The biggest sources of failure are the misunderstandings that develop between what the customer expects and what the software vendor can deliver. Be on your guard and learn the software’s capabilities and limitations. The software vendor may not volunteer the product’s failings unless customer does enough inquiring and it is possible that vendor’s power of influence may convince customer to purchase a product that may not fully address the needs.

In the Web Based Payroll System’s case, we listed down quite a few functionalities of the system, and the prototype can be demonstrated online at <https://payroll-wfsiewapp.rhcloud.com>. Since it is a prototype, more functions can be added to the system at a later time.

## 5.3: Selecting the right tools, programming language, frameworks, technologies, and IDEs

The next and important step is to select the right tools, programming language, frameworks, technologies, and Integrated Development Environment. Below I will explain briefly that **what** and **why** I have used to implement the system and I will also be telling **what** I have **not** used and **why not** used. [3]

### 5.3.1: APIs and Libraries

There are there libraries used in the system, which are jQuery, jQuery UI, and Highcharts JS. jQuery was used as the core JavaScript library to simplify the client-side scripting of HTML and the development of Ajax web application. The reason I choose jQuery is because it is easy to use, fast, small, and feature-rich JavaScript library. Besides, it is the most popular JavaScript library in use today. The library is free and open source software.

jQuery includes the following features:

* DOM element selections using the multi-browser open source selector engine
* DOM traversal and modification (including support for CSS 1-3)
* DOM manipulation based on CSS selectors that uses node elements name and node elements attributes (id and class) as criteria to build selectors
* Events
* Effects and animations
* AJAX
* Extensibility through plug-ins
* Utilities - such as user agent information, feature detection
* Compatibility methods that are natively available in modern browsers but need fall backs for older ones - For example the inArray() and each() functions
* Multi-browser (not to be confused with cross-browser) support [4]

The second library that I used is jQuery UI. I used jQuery UI to create the web user interface.

jQuery UI is a JavaScript library that provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, built on top of the jQuery JavaScript library, that can be used to build interactive web applications. The library includes the following:

**Widgets**

* **Accordion** – Accordion containers
* **Autocomplete** – Auto-complete boxes based on what the user types
* **Button** – Enhanced button appearance, turn radio buttons and checkboxes into pushbuttons
* **Datepicker** – Advanced date-picker
* **Dialog** – Show dialog boxes on top of other content, easily and robustly
* **Menu** – Show a Menu
* **Progressbar** – Progress bars, both animated and not
* **Slider** – Fully customizable sliders with various features
* **Spinner** – Show a Number Spinner
* **Tabs** – Tabbed user interface handling, with both inline and demand-loaded content
* **Tooltip** – Show a Tooltip

**Effects**

* **Color Animation** – Animate the transition from one color to another
* **Toggle Class, Add Class, Remove Class, Switch Class** – Animate the transition from one set of styles to another
* **Effect** - A variety of effects (appear, slide-down, explode, fade-in, etc.)
* **Toggle** - Toggle an effect on and off
* **Hide, Show** – Using the effects above

**Utilities**

* **Position** - Set an element's position relative to another element's position (alignment) [5]

The third library that I used is Highcharts JS. I used Highcharts JS to show charts in the system. Highcharts is a charting library written in pure JavaScript, offering an easy way of adding interactive charts to your website or web application. Highcharts supports line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, arearange, areasplinerange, columnrange and polar chart types. It works in all modern browsers including the iPhone/iPad and Internet Explorer from version 6. Standard browsers use SVG for the graphics rendering. In legacy Internet Explorer graphics are drawn using VML. It is also free to use for personal website, a school site or a non-profit organization. Since I use it for the final year project, I can freely use it without the license. It is solely based on native browser technologies and doesn’t require client side plugins like Flash or Java. Furthermore, no installation is required on the server. Besides, it also supports jQuery library.

There is also exporting module, where users can export the chart to PNG, JPG, PDF or SVG format at the click of a button, or print the chart directly from the web page. [6]

There a many more libraries available in the open source today, besides jQuery, JQuery UI, and Highcharts JS. Some examples are MooTools, Prototype, Dojo, and YUI.

The main reason I choose jQuery over these libraries is because I found out that jQuery is easier to learn and use, and the code is shorter while achieving the same functionality.

### 5.3.2: Frameworks

The framework that was used in the server-side is Ruby on Rails, which is an open source web framework created in Ruby programming language that’s optimized for programmer happiness and sustainable productivity. It lets the programmer write beautiful code by favoring convention over configuration. The framework is the best MVC framework I had ever used compared to ASP.NET MVC 3 (Microsoft), Django (python), and Spring MVC framework (java). Besides, it is also easy to deploy to the OpenShift, which is a free, auto-scaling Platform as a Service (PaaS) for applications. [7] In addition Ruby is also a dynamic and object-oriented programming language.

There are few useful commands in Rails which helps the programmer in the development, which are listed below:

* rails console
* rails new app\_name
* rails server
* rails generate
* rails dbconsole
* rake

1. **rails console**

The *console* command lets the programmer interact with the application from the command line. This is useful for testing out quick ideas with code and changing data server-side without touching the website.

1. **rails new**

It creates a new Rails application.

1. **rails server**

This command launches a small web server named WEBrick which comes bundled with Ruby. It is needed to access the application through a web browser.



Figure 16: rails server command

1. **rails generate**

The *rails generate* command uses templates to create a whole lot of things. Running *rails generate* by itself gives a list of available generators. Using generators will save a large amount of time by writing boilerplate code, code that is necessary for the application to work. The common usage would be *rails generate controller*, which is used to generate controller, and *rails generate model*, which is used to generate data model.

1. **rails dbconsole**

*rails dbconsole* figures out which database are currently using and drops the programmer into whichever command line interface where the programmer would use with it.

1. **rake**

Rake is used for common administration tasks, especially ones that build off of each other. Some common usage would be *rake db:create* (creates the database), *rake db:migrate* (runs database migrations), *rake db:seed* (load seed data into the database), and *rake assets:precompile* (precompile the assets, such as JavaScripts and CSS files).

### 5.3.3: Technologies

The project uses Ajax technology, with Model-View-Controller design pattern at the server-side, which is Ruby on Rails. The web application uses Single-page application technique to provide a more fluid user experience similar to a desktop application. The client-side uses Ajax to send HTTP request to the server-side, which returns JSON, plain text, and html response. The reason I use JSON instead of XML is because JSON is simpler to understand and it requires less configuration overhead. XML is good for situations in which

* you need message validation
* you’re using XSLT
* your messages include a lot of marked-up text
* you need to interoperate with environments that don’t support JSON
* you need attributes or namespacing [8]

Example of JSON response:

{"menu": {

"id": "file",

"value": "File",

"popup": {

"menuitem": [

{"value": "New", "onclick": "CreateNewDoc()"},

{"value": "Open", "onclick": "OpenDoc()"},

{"value": "Close", "onclick": "CloseDoc()"}

]

}

}}

The same response expressed as XML:

<menu id="file" value="File">

<popup>

<menuitem value="New" onclick="CreateNewDoc()" />

<menuitem value="Open" onclick="OpenDoc()" />

<menuitem value="Close" onclick="CloseDoc()" />

</popup>

</menu>

### 5.3.4: Integrated Development Environment

The Integrated Development Environment that was used to develop the project is Aptana Studio 3, which is a professional, open source development tool for the open web. The core features include:

**HTML, CSS, and JavaScript Code Assist**

* Aids in authoring of HTML, CSS, JavaScript, PHP, and Ruby.
* Supports the latest HTML5 specifications.
* Includes information about the level of support for each element in the major web browsers.

**Integrated Debugger**

* Set breakpoints, inspect variables, and control execution.
* The integrated Ruby & Rails and JavaScript debuggers help debugging easier.

**Git Integration**

* Easily put projects under git source code control, such as Github.
* Facilitates git-based deployments

**Built-in Terminal**

* Quickly access a command line terminal for execution of operating system commands and language utilities such as gem, rake, etc. [9]

# CHAPTER SIX

# TESTING

## 6.1: Testing in Software Engineering

Testing is an activity that is used to discover errors and correct them, so that we are able to create a defect-free product for our customer or user. Testing is an important phase in the software development life cycle. The objective of testing is to evaluate if we have created the system correctly. During the earlier stages, the focus was to check what is being built but in testing when we have the end product ready, our focus shifts to validate whether the product that has been built has been built correctly or not. Hence, the focus shifts from building the product right to building the right product. [1]

There are two basic types of software testing, which are black box testing and white box testing. General testing process for large system development starts with the testing of individual program units such as functions, classes or objects. These are then integrated into sub-system and systems, and the interactions of these units were tested. Finally after delivery of the system, the customer may carry out a series of acceptance tests to check that the system performs as specified. [2]

Whereas, for smaller system or for system that are developed through scripting or reuse, there are often fewer distinct stages in the process.

The two fundamental testing activities are component testing, testing the parts of the system – and system testing, testing the system as a whole. [2]

## 6.2: Goals and Types of Testing

Basically, there are two distinct goals of the software testing process:

* To demonstrate to the developer and the customer that the software meets its requirements.
* To discover faults or defects in the software where the behavior of the software is incorrect, undesirable or does not conform to its specification.

The first goal, where you expect the system to perform correctly using a given set of test cases that reflect the systems expected use, leads to validation testing. The second goal leads to defect testing, where the test cases are designed to expose defects. The main types of testing approaches are defined below: [2]

### 6.2.1: System Testing

System testing involves integrating two or more components that implement system functions or features and then testing this integrated system. For most complex systems, there are two distinct phases to system testing – Integration Testing and Release Testing. As for the Web Based Payroll System, the system had to go through both Integration and Release testing.

#### 6.2.1.1: Integration Testing

Integration testing is mostly concerned with finding defects in the system, where the test team has access to the source code of the system. If the problem is discovered, the team goes through the source code to find the components that have to be debugged. [2]

Integration testing was done after every unit or feature being added to the system. For example, if the current has three features (show payroll chart, overtime chart, total hours worked chart), when the fourth feature, unit, or component if being attached or added to the system, integration testing had to be done throughout although there are still many units or components to be added to the system.

#### 6.2.1.2: Release Testing

In release testing that version of the system is tested that could be released to users or customers. The test team here validates if the system meets its requirements and also ensures system dependability. It is usually black-box testing where the test team is simply concerned with demonstrating the system does or does not work properly. If problems are discovered then they are reported to the development team whose job is to debug the program. Acceptance Testing is key aspect of release testing, where the customers or users are involved in release testing. If the release is good enough, the customer or user may then accept it for use.

After all the units were integrated and combined together to form a complete system, a release test was ran to make sure that system’s components are not affecting the other components after integrating them. Few feedbacks from the users were collected, according to the feedback the system has to go through the debugging and testing again. [2]

### 6.2.2: Component Testing

Also known as Unit Testing is the process of testing individual components in the system, to expose faults in these components, and the software developers are responsible for this testing. There are different types of component that may be tested at this stage:

1. Individual functions or methods within an object
2. Object classes that have several attributes and methods
3. Composite components made up of several different objects or function.

These composite components have a defined interface that is used to access their functionality.

The Web Based Payroll System was created in components and units. A new Rails project was created in such a way that each model, view, and controller are organized in separate folder. Below is the screenshot of the project structure.



Figure 17: The main project folder



Figure 18: The application folder

Figure above shows the units of the Web Based Payroll System which later were integrated as a whole and was tested using the system testing method.

In Rails, there are three different kinds of tests that can be written, which are unit, functional, and integration test.

* Unit testing tests the models (the command to run the unit test is **rake test:units**).



Figure 19: Running unit test in Rails

* Functional testing tests the controllers (the command to run the functional test is **rake test:functionals**).



Figure 20: Running functional test in Rails

* Integration testing tests at a high level through multiple controllers (the command to run the integration test is **rake test:integration**).

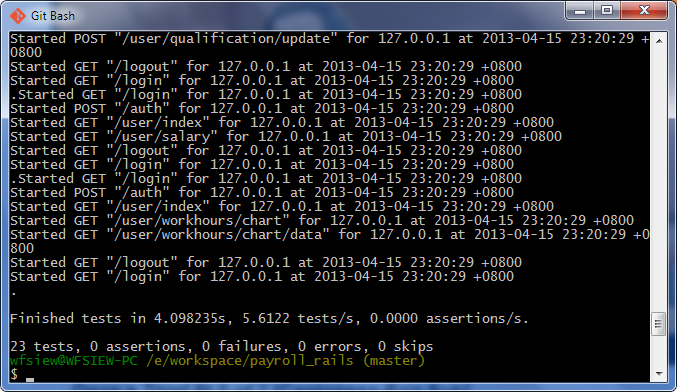


Figure 21: Running integration test in Rails

Table 6: Five levels of testing



## 6.3: Black box testing

Black box testing, also known as functional testing and behavioral testing, focuses on determining whether or not a program does what it is supposed to do based on its functional requirements. Black box testing attempts to find errors in the external behavior of the code in the following categories (1) incorrect or missing functionality; (2) interface errors; (3) errors in data structures used by interfaces; (4) behavior or performance errors; and (5) initialization and termination errors. Through this testing, we can determine if the functions appear to work according to specifications. However, it is important to note that no amount of testing can unequivocally demonstrate the absence of errors and defects the code.



Figure 22: Black Box testing

*http://2.bp.blogspot.com/\_fOOSCGT3XIw/Sa4cVsX7T7I/AAAAAAAAAAM/i\_YOWD6xHIQ/s320/BlackBoxTesting.gif*

## 6.4: Test Cases

A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not.

The format of the test case design is very important. I will use a particular format for the test cases, as shown in Table.

Table 7: Test Case Planning Format



Table 8: Test cases for Web Based Payroll System

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Description | Expected Results | Actual Results |
| 1 | Admin user entered the username and password. | Admin user should be redirected to the admin home page. | Admin user redirected to the admin home page.  **Test case passed** |
| 2 | Admin user creates a user. | The newly created user should appear in the list. | The newly created user appears in the list.  **Test case passed** |
| 3 | Admin user edits a user. | The user should be successfully updated. | The user successfully updated.  **Test case passed** |
| 4 | Admin user selects user and delete. | The selected users should be successfully deleted. | The selected users successfully deleted.  **Test case passed** |
| 5 | Admin user search user. | The list of users should be displayed based on the search criteria. | The list of users displayed based on the search criteria.  **Test case passed** |
| 6 | Admin user creates a job title. | The newly created job title should appear in the list. | The newly created job title appears in the list.  **Test case passed** |
| 7 | Admin user edits a job title. | The job title should be successfully updated. | The job title successfully updated.  **Test case passed** |
| 8 | Admin user selects job title and delete. | The selected job titles should be successfully deleted. | The selected job titles successfully deleted.  **Test case passed** |
| 9 | Admin user search job title. | The list of job titles should be displayed based on the search criteria. | The list of job titles displayed based on the search criteria.  **Test case passed** |
| 10 | Admin user creates an employment status. | The newly created employment status should appear in the list. | The newly created employment status appears in the list.  **Test case passed** |
| 11 | Admin user edits an employment status. | The employment status should be successfully updated. | The employment status successfully updated.  **Test case passed** |
| 12 | Admin user selects employment status and delete. | The selected employment statuses should be successfully deleted. | The selected employment statuses successfully deleted.  **Test case passed** |
| 13 | Admin user search employment status. | The list of employment statuses should be displayed based on the search criteria. | The list of employment statuses displayed based on the search criteria.  **Test case passed** |
| 14 | Admin user creates a job category. | The newly created job category should appear in the list. | The newly created job category appears in the list.  **Test case passed** |
| 15 | Admin user edits a job category. | The job category should be successfully updated. | The job category successfully updated.  **Test case passed** |
| 16 | Admin user selects job category and delete. | The selected job categories should be successfully deleted. | The selected job categories successfully deleted.  **Test case passed** |
| 17 | Admin user search job category. | The list of job categories should be displayed based on the search criteria. | The list of job categories displayed based on the search criteria.  **Test case passed** |
| 18 | Admin user creates a department. | The newly created department should appear in the list. | The newly created department appears in the list.  **Test case passed** |
| 19 | Admin user edits a department. | The department should be successfully updated. | The department successfully updated.  **Test case passed** |
| 20 | Admin user selects department and delete. | The selected departments should be successfully deleted. | The selected departments successfully deleted.  **Test case passed** |
| 21 | Admin user search department. | The list of departments should be displayed based on the search criteria. | The list of departments displayed based on the search criteria.  **Test case passed** |
| 22 | Admin user creates a employee. | The newly created employee should appear in the list. | The newly created employee appears in the list.  **Test case passed** |
| 23 | Admin user edits an employee. | The employee should be successfully updated. | The employee successfully updated.  **Test case passed** |
| 24 | Admin user selects employee and delete. | The selected employees should be successfully deleted. | The selected employees successfully deleted.  **Test case passed** |
| 25 | Admin user search employee. | The list of employees should be displayed based on the search criteria. | The list of employees displayed based on the search criteria.  **Test case passed** |
| 26 | Admin user search employee’s attendance. | The list of attendance should be displayed based on the search criteria. | The list of attendance displayed based on the search criteria.  **Test case passed** |
| 27 | Admin user creates a pay rate. | The newly created pay rate should appear in the list. | The newly created pay rate appears in the list.  **Test case passed** |
| 28 | Admin user edits a pay rate. | The pay rate should be successfully updated. | The pay rate successfully updated.  **Test case passed** |
| 29 | Admin user selects pay rate and delete. | The selected pay rates should be successfully deleted. | The selected pay rates successfully deleted.  **Test case passed** |
| 30 | Admin user search pay rate. | The list of pay rates should be displayed based on the search criteria. | The list of pay rates displayed based on the search criteria.  **Test case passed** |
| 31 | Admin user creates an overtime pay rate. | The newly created overtime pay rate should appear in the list. | The newly created overtime pay rate appears in the list.  **Test case passed** |
| 32 | Admin user edits an overtime pay rate. | The overtime pay rate should be successfully updated. | The overtime pay rate successfully updated.  **Test case passed** |
| 33 | Admin user selects overtime pay rate and delete. | The selected overtime pay rates should be successfully deleted. | The selected overtime pay rates successfully deleted.  **Test case passed** |
| 34 | Admin user search overtime pay rate. | The list of overtime pay rates should be displayed based on the search criteria. | The list of overtime pay rates displayed based on the search criteria.  **Test case passed** |
| 35 | Admin user creates a salary adjustment. | The newly created salary adjustment should appear in the list. | The newly created salary adjustment appears in the list.  **Test case passed** |
| 36 | Admin user edits a salary adjustment. | The salary adjustment should be successfully updated. | The salary adjustment successfully updated.  **Test case passed** |
| 37 | Admin user selects salary adjustment and delete. | The selected salary adjustments should be successfully deleted. | The selected salary adjustments successfully deleted.  **Test case passed** |
| 38 | Admin user search salary adjustment. | The list of salary adjustments should be displayed based on the search criteria. | The list of salary adjustments displayed based on the search criteria.  **Test case passed** |
| 39 | Admin user view all employees overtime chart. | The chart should be displayed. | The chart is displayed.  **Test case passed** |
| 40 | Admin user generates filtered overtime chart. | The chart should be displayed based on the filtered criteria. | The chart is displayed based on the filtered criteria.  **Test case passed** |
| 41 | Admin user view all hourly paid employees total work hours chart. | The chart should be displayed. | The chart is displayed.  **Test case passed** |
| 42 | Admin user generates filtered total work hours chart. | The chart should be displayed based on the filtered criteria. | The chart is displayed based on the filtered criteria.  **Test case passed** |
| 43 | Admin user view all employees hourly payroll chart. | The chart should be displayed. | The chart is displayed.  **Test case passed** |
| 44 | Admin user generates filtered hourly payroll chart. | The chart should be displayed based on the filtered criteria. | The chart is displayed based on the filtered criteria.  **Test case passed** |
| 45 | Admin user view all employees payslip. | The payslip should be opened in a window. | The payslip is opened in a window.  **Test case passed** |
| 46 | Admin user search employee and view selected employee’s payslip. | The list of employees should be displayed based on the search criteria and the payslip should be opened in a window. | The list of employees displayed based on the search criteria and the payslip is opened in a window.  **Test case passed** |
| 47 | Admin user clicks Logout button. | Admin user should be redirected to the login page. | Admin user redirected to the login page.  **Test case passed** |
| 48 | Hourly paid employee entered the username and password. | Hourly paid employee should be redirected to the hourly paid employee home page. | Hourly paid employee redirected to the hourly paid employee home page.  **Test case passed** |
| 49 | Hourly paid employee edits personal details. | The details should be successfully updated. | The details successfully updated.  **Test case passed** |
| 50 | Hourly paid employee edits contact details. | The details should be successfully updated. | The details successfully updated.  **Test case passed** |
| 51 | Hourly paid employee view job details. | The job details should be displayed. | The job details displayed.  **Test case passed** |
| 52 | Hourly paid employee view salary details. | The salary details should be displayed. | The salary details displayed.  **Test case passed** |
| 53 | Hourly paid employee edits qualification details. | The details should be successfully updated. | The details successfully updated.  **Test case passed** |
| 54 | Hourly paid employee view total work hours chart. | The chart should be displayed. | The chart is displayed.  **Test case passed** |
| 55 | Hourly paid employee generates filtered total work hours chart. | The chart should be displayed based on the filtered criteria. | The chart is displayed based on the filtered criteria.  **Test case passed** |
| 56 | Hourly paid employee view hourly payroll chart. | The chart should be displayed. | The chart is displayed.  **Test case passed** |
| 57 | Hourly paid employee generates filtered hourly payroll chart. | The chart should be displayed based on the filtered criteria. | The chart is displayed based on the filtered criteria.  **Test case passed** |
| 58 | Hourly paid employee view payslip for selected month and year. | The payslip should be opened in a window. | The payslip is opened in a window.  **Test case passed** |
| 59 | Hourly paid employee clicks Logout button. | Hourly paid employee should be redirected to the login page. | Hourly paid employee redirected to the login page.  **Test case passed** |
| 60 | Monthly paid employee entered the username and password. | Monthly paid employee should be redirected to the monthly paid employee home page. | Monthly paid employee redirected to the monthly paid employee home page.  **Test case passed** |
| 61 | Monthly paid employee edits personal details. | The details should be successfully updated. | The details successfully updated.  **Test case passed** |
| 62 | Monthly paid employee edits contact details. | The details should be successfully updated. | The details successfully updated.  **Test case passed** |
| 63 | Monthly paid employee view job details. | The job details should be displayed. | The job details displayed.  **Test case passed** |
| 64 | Monthly paid employee view salary details. | The salary details should be displayed. | The salary details displayed.  **Test case passed** |
| 65 | Monthly paid employee edits qualification details. | The details should be successfully updated. | The details successfully updated.  **Test case passed** |
| 66 | Monthly paid employee view total work hours chart. | The chart should be displayed. | The chart is displayed.  **Test case passed** |
| 67 | Monthly paid employee generates filtered total work hours chart. | The chart should be displayed based on the filtered criteria. | The chart is displayed based on the filtered criteria.  **Test case passed** |
| 68 | Monthly paid employee view overtime chart. | The chart should be displayed. | The chart is displayed.  **Test case passed** |
| 69 | Monthly paid employee generates filtered overtime chart. | The chart should be displayed based on the filtered criteria. | The chart is displayed based on the filtered criteria.  **Test case passed** |
| 70 | Monthly paid employee view payslip for selected month and year. | The payslip should be opened in a window. | The payslip is opened in a window.  **Test case passed** |
| 71 | Monthly paid employee clicks Logout button. | Monthly paid employee should be redirected to the login page. | Monthly paid employee redirected to the login page.  **Test case passed** |

71 test cases were developed to accomplish the Black Box testing for the Web Based Payroll System.

# CHAPTER SEVEN

# FUTURE WORK/MAINTENANCE

As for the future work on the Web Based Payroll System, few tasks, ideas, features, and functionalities are queued up to be added in the system. Not all the Payroll features have been added to the Web Based Payroll System but can be added in the future to the system.

The source code of the project can be accessed at <https://github.com/wfsiew/Payroll>, and the web application can be accessed at <https://payroll-wfsiewapp.rhcloud.com>

## 7.1: Recommendations and Features

Not all the Payroll features has been added to the Web Based Payroll System as the system is the proof of concept and does not encompass the full features. More time and developing team are required to develop the Web Based Payroll System with the full features of the Payroll.

The features that are going to be added to the Web Based Payroll System in the future are the following:

### 7.1.1: Leave Management System

When an employee applies for leave he/she can know how many leaves are in balance (Annual Leave, Emergency Leave, Sick Leave, etc) through the Leave Management System. During Leave Application, the system captures the following data:

* Leave Application Date
* Leave type, such as Annual Leave, Sick Leave (shows the Balance Leaves to the employee while applying)
* Leave Period i.e. the total number of days for which the leave is applied
* Reason for applying the Leave
* Address and Contact Number during Leave, in case of Emergency purposes

The Leave which the employee has applied for, can be Approved fully or partly.

The details given in the leave application are linked to the payroll system and the employee’s salary is generated in accordance with his attendance.

### 7.1.2: Time Management System

Employees can be assigned to different shifts. The management can define Shifts and the following data is captured:

* Shift name
* Shift number
* Shift period
* Shift Start timing and End timing
* Late coming limit and Late sitting limit
* Overtime in minutes
* Break time (Lunch, Breakfast, etc)
* Description related to Shift
* Weekly off (can be any day in the week)

The time management takes care of physical absence due to travel undertaken for official purposes (in case of marketing personnel, etc).

### 7.1.3: Import/Export

Allow administrator to import data to the system from csv file, or excel file, and export the data to csv file, or excel file.

# References

|  |  |
| --- | --- |
| [1] | "The Benefits of Web Based Applications and Systems," 2007. [Online]. Available: http://www.dbnetsolutions.co.uk/Articles/BenefitsOfWebBasedApplications.aspx. [Accessed March 2013]. |
| [2] | N. Grace, "An Introduction to Payroll Systems," [Online]. Available: http://www.ehow.com/info\_7771494\_introduction-payroll-systems.html. [Accessed March 2013]. |
| [3] | Wikipedia, "Ajax (programming) - Wikipedia, the free encyclopedia," March 2013. [Online]. Available: http://en.wikipedia.org/wiki/Ajax\_(programming). [Accessed March 2013]. |
| [4] | J. J. Garrett, "Ajax: A New Approach to Web Applications - Adaptive Path," [Online]. Available: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications. [Accessed March 2013]. |
| [5] | S. R. Pawar, "Importance of Ajax in web applications - Tutorial," [Online]. Available: http://www.javacertificate.net/ajax\_article.php. [Accessed March 2013]. |
| [6] | "Technologies Used in AJAX," [Online]. Available: http://www.tutorialspoint.com/ajax/ajax\_technology.htm. [Accessed March 2013]. |
| [7] | Wikipedia, "Web application - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Web\_application. [Accessed March 2013]. |
| [8] | Wikipedia, "Single-page application - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Single-page\_application. [Accessed March 2013]. |
| [9] | D. Roos, "HowStuffWorks "What is a Payroll System?"," [Online]. Available: http://money.howstuffworks.com/payroll-system1.htm. [Accessed March 2013]. |
| [10] | Wikipedia, "Model–view–controller - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. [Accessed March 2013]. |
| [11] | Wikipedia, "Requirements analysis - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Requirements\_analysis. [Accessed March 2013]. |
| [12] | Wikipedia, "Software prototyping - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Software\_prototyping. [Accessed March 2013]. |
| [13] | Wikipedia, "Software requirements specification - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Software\_requirements\_specification. [Accessed March 2013]. |
| [14] | Wikipedia, "Operating environment - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Operating\_environment. [Accessed March 2013]. |
| [15] | jQuery, "jQuery," [Online]. Available: http://jquery.com/. [Accessed March 2013]. |
| [16] | jQuery UI, "jQuery UI," [Online]. Available: http://jqueryui.com/. [Accessed March 2013]. |
| [17] | Highcharts JS, "Highcharts - Interactive JavaScript charts for your webpage," [Online]. Available: http://www.highcharts.com/. [Accessed March 2013]. |
| [18] | Wikipedia, "Functional requirement - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Functional\_requirements. [Accessed March 2013]. |
| [19] | A. A. &. C. Greenidge, March 2013. [Online]. Available: http://www.cavehill.uwi.edu/staff/eportfolios/paulwalcott/courses/comp2145/2009/design\_-\_concepts\_and\_principles.htm. |
| [20] | R. L. Burback, "The Implementation Phase," 14 December 1998. [Online]. Available: http://infolab.stanford.edu/~burback/watersluice/node17.html. [Accessed March 2013]. |
| [21] | Wikipedia, "Implementation - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Implementation. [Accessed March 2013]. |
| [22] | A. L. Tharp, "Selecting the “right” programming language," 1982. |
| [23] | OpenShift, "OpenShift by Red Hat," [Online]. Available: https://openshift.redhat.com/app/. [Accessed March 2013]. |
| [24] | P. Bergantino, "When would you use XML over JSON for Ajax? - Stack Overflow," January 2009. [Online]. Available: http://stackoverflow.com/questions/478855/when-would-you-use-xml-over-json-for-ajax. [Accessed March 2013]. |
| [25] | aptana, "Aptana | Studio," [Online]. Available: http://www.aptana.com/products/studio3. [Accessed March 2013]. |
| [26] | angad, "Testing In Software Engineering," 2 November 2012. [Online]. Available: http://techforum4u.com/content.php/417-Testing-In-Software-Engineering. [Accessed March 2013]. |
| [27] | A. Bertolino, "Software testing research and practice," 2003. |

# APPENDIX A

# COMPLETE SOURCE CODE

## config/application.rb

require File**.**expand\_path('../boot', \_\_FILE\_\_)

require 'rails/all'

**if** defined?(Bundler)

*# If you precompile assets before deploying to production, use this line*

  Bundler**.**require(**\***Rails**.**groups(:assets **=>** %w(development test)))

*# If you want your assets lazily compiled in production, use this line*

*# Bundler.require(:default, :assets, Rails.env)*

**end**

**module** Payroll

**class** **Application** **<** Rails::Application

*# Settings in config/environments/\* take precedence over those specified here.*

*# Application configuration should go into files in config/initializers*

*# -- all .rb files in that directory are automatically loaded.*

*# Custom directories with classes and modules you want to be autoloadable.*

*# config.autoload\_paths += %W(#{config.root}/extras)*

*# Only load the plugins named here, in the order given (default is alphabetical).*

*# :all can be used as a placeholder for all plugins not explicitly named.*

*# config.plugins = [ :exception\_notification, :ssl\_requirement, :all ]*

*# Activate observers that should always be running.*

*# config.active\_record.observers = :cacher, :garbage\_collector, :forum\_observer*

*# Set Time.zone default to the specified zone and make Active Record auto-convert to*

*# this zone.*

*# Run "rake -D time" for a list of tasks for finding time zone names. Default is UTC.*

*# config.time\_zone = 'Central Time (US & Canada)'*

    config**.**time\_zone **=** 'Kuala Lumpur'

*# The default locale is :en and all translations from config/locales/\*.rb,yml are auto*

*# loaded.*

*# config.i18n.load\_path += Dir[Rails.root.join('my', 'locales', '\*.{rb,yml}').to\_s]*

*# config.i18n.default\_locale = :de*

*# Configure the default encoding used in templates for Ruby 1.9.*

    config**.**encoding **=** "utf-8"

*# Configure sensitive parameters which will be filtered from the log file.*

    config**.**filter\_parameters **+=** **[**:password**]**

*# Enable escaping HTML in JSON.*

    config**.**active\_support**.**escape\_html\_entities\_in\_json **=** **true**

*# Use SQL instead of Active Record's schema dumper when creating the database.*

*# This is necessary if your schema can't be completely dumped by the schema dumper,*

*# like if you have constraints or database-specific column types*

*# config.active\_record.schema\_format = :sql*

*# Enforce whitelist mode for mass assignment.*

*# This will create an empty whitelist of attributes available for mass-assignment for*

*# all models in your app. As such, your models will need to explicitly whitelist or*

*# blacklist accessible parameters by using an attr\_accessible or attr\_protected*

*# declaration.*

    config**.**active\_record**.**whitelist\_attributes **=** **true**

*# Enable the asset pipeline*

    config**.**assets**.**enabled **=** **true**

*# Version of your assets, change this if you want to expire all your assets*

    config**.**assets**.**version **=** '1.0'

    config**.**assets**.**precompile **+=** %w(loginui.css \_payslip.css \_payslip.js

dark-hive/jquery-ui-1.10.2.custom.min.css admin.css admin.js user.css user.js)

    jquitheme **=** %w(blitzer dark-hive trontastic humanity)

    jquicss **=** 'jquery-ui-1.10.2.custom.min.css'

    jquitheme**.**each **do** **|**t**|**

      config**.**assets**.**precompile **<<** "#{t}/#{jquicss}"

**end**

**end**

**end**

## config/environment.rb

*# Load the rails application*

require File**.**expand\_path('../application', \_\_FILE\_\_)

*# Initialize the rails application*

Payroll::Application**.**initialize!

Rails**.**logger **=** Logger**.**new(STDOUT)

## config/database.yml

*# MySQL. Versions 4.1 and 5.0 are recommended.*

*#*

*# Install the MYSQL driver*

*# gem install mysql2*

*#*

*# Ensure the MySQL gem is defined in your Gemfile*

*# gem 'mysql2'*

*#*

*# And be sure to use new-style password hashing:*

*# http://dev.mysql.com/doc/refman/5.0/en/old-client.html*

development:

  adapter: mysql2

  encoding: utf8

  reconnect: false

  database: payroll\_development

  pool: 5

  username: root

  password: root

  host: 127.0.0.1

  port: 3307

*# Warning: The database defined as "test" will be erased and*

*# re-generated from your development database when you run "rake".*

*# Do not set this db to the same as development or production.*

test:

  adapter: mysql2

  encoding: utf8

  reconnect: false

  database: payroll\_test

  pool: 5

  username: root

  password: root

  host: 127.0.0.1

  port: 3307

production:

  adapter: mysql2

  encoding: utf8

  reconnect: false

  database: payroll\_development

  pool: 5

  username: root

  password: root

  host: 127.0.0.1

  port: 3307

## config/routes.rb

Payroll::Application**.**routes**.**draw **do**

  root :to **=>** 'admin/admin#index'

  match 'login' **=>** 'application#new', :as **=>** :login

  match 'auth' **=>** 'application#create', :as **=>** :auth

  match 'logout' **=>** 'application#destroy', :as **=>** :logout

  namespace :admin **do**

    match 'index' **=>** 'admin#index', :as **=>** :index, :via **=>** :get

    scope 'user', :as **=>** 'user' **do**

      match '' **=>** 'user#index', :via **=>** :get

      match 'list' **=>** 'user#list', :as **=>** :list, :via **=>** **[**:get, :post**]**

      match 'new' **=>** 'user#new', :as **=>** :new, :via **=>** :get

      match 'create' **=>** 'user#create', :as **=>** :create, :via **=>** :post

      match 'edit(/:id)' **=>** 'user#edit', :as **=>** :edit, :via **=>** :get

      match 'update(/:id)' **=>** 'user#update', :as **=>** :update, :via **=>** :post

      match 'delete' **=>** 'user#destroy', :as **=>** :delete, :via **=>** :post

**end**

    scope 'employee', :as **=>** 'employee' **do**

      match '' **=>** 'employee#index', :via **=>** :get

      match 'list' **=>** 'employee#list', :via **=>** **[**:get, :post**]**

      match 'new' **=>** 'employee#new', :as **=>** :new, :via **=>** :get

      match 'create' **=>** 'employee#create', :as **=>** :create, :via **=>** :post

      match 'edit(/:id)' **=>** 'employee#edit', :as **=>** :edit, :via **=>** :get

      match 'update(/:id)' **=>** 'employee#update', :as **=>** :update, :via **=>** :post

      match 'delete' **=>** 'employee#destroy', :as **=>** :delete, :via **=>** :post

**end**

    scope 'designation', :as **=>** 'designation' **do**

      match '' **=>** 'designation#index', :via **=>** :get

      match 'list' **=>** 'designation#list', :as **=>** :list, :via **=>** **[**:get, :post**]**

      match 'new' **=>** 'designation#new', :as **=>** :new, :via **=>** :get

      match 'create' **=>** 'designation#create', :as **=>** :create, :via **=>** :post

      match 'edit(/:id)' **=>** 'designation#edit', :as **=>** :edit, :via **=>** :get

      match 'update(/:id)' **=>** 'designation#update', :as **=>** :update, :via **=>** :post

      match 'delete' **=>** 'designation#destroy', :as **=>** :delete, :via **=>** :post

**end**

    scope 'empstatus', :as **=>** 'empstatus' **do**

      match '' **=>** 'employment\_status#index', :via **=>** :get

      match 'list' **=>** 'employment\_status#list', :as **=>** :list, :via **=>** **[**:get, :post**]**

      match 'new' **=>** 'employment\_status#new', :as **=>** :new, :via **=>** :get

      match 'create' **=>** 'employment\_status#create', :as **=>** :create, :via **=>** :post

      match 'edit(/:id)' **=>** 'employment\_status#edit', :as **=>** :edit, :via **=>** :get

      match 'update(/:id)' **=>** 'employment\_status#update', :as **=>** :update, :via **=>** :post

      match 'delete' **=>** 'employment\_status#destroy', :as **=>** :delete, :via **=>** :post

**end**

    scope 'jobcat', :as **=>** 'jobcat' **do**

      match '' **=>** 'job\_category#index', :via **=>** :get

      match 'list' **=>** 'job\_category#list', :as **=>** :list, :via **=>** **[**:get, :post**]**

      match 'new' **=>** 'job\_category#new', :as **=>** :new, :via **=>** :get

      match 'create' **=>** 'job\_category#create', :as **=>** :create, :via **=>** :post

      match 'edit(/:id)' **=>** 'job\_category#edit', :as **=>** :edit, :via **=>** :get

      match 'update(/:id)' **=>** 'job\_category#update', :as **=>** :update, :via **=>** :post

      match 'delete' **=>** 'job\_category#destroy', :as **=>** :delete, :via **=>** :post

**end**

    scope 'dept', :as **=>** 'dept' **do**

      match '' **=>** 'department#index', :via **=>** :get

      match 'list' **=>** 'department#list', :as **=>** :list, :via **=>** **[**:get, :post**]**

      match 'new' **=>** 'department#new', :as **=>** :new, :via **=>** :get

      match 'create' **=>** 'department#create', :as **=>** :create, :via **=>** :post

      match 'edit(/:id)' **=>** 'department#edit', :as **=>** :edit, :via **=>** :get

      match 'update(/:id)' **=>** 'department#update', :as **=>** :update, :via **=>** :post

      match 'delete' **=>** 'department#destroy', :as **=>** :delete, :via **=>** :post

**end**

    scope 'payrate', :as **=>** 'payrate' **do**

      match '' **=>** 'pay\_rate#index', :via **=>** :get

      match 'list' **=>** 'pay\_rate#list', :as **=>** :list, :via **=>** **[**:get, :post**]**

      match 'new' **=>** 'pay\_rate#new', :as **=>** :new, :via **=>** :get

      match 'create' **=>** 'pay\_rate#create', :as **=>** :create, :via **=>** :post

      match 'edit(/:id)' **=>** 'pay\_rate#edit', :as **=>** :edit, :via **=>** :get

      match 'update(/:id)' **=>** 'pay\_rate#update', :as **=>** :update, :via **=>** :post

      match 'delete' **=>** 'pay\_rate#destroy', :as **=>** :delete, :via **=>** :post

**end**

    scope 'hourly', :as **=>** 'hourly' **do**

      match 'chart' **=>** 'hourly\_payroll\_chart#index', :via **=>** :get

      match 'chart/data' **=>** 'hourly\_payroll\_chart#data', :via **=>** **[**:get, :post**]**

**end**

    scope 'payslip', :as **=>** 'payslip' **do**

      match '' **=>** 'payslip#index', :via **=>** :get

      match 'list' **=>** 'payslip#list', :via **=>** **[**:get, :post**]**

      match 'slip(/:id(/:month(/:year)))' **=>** 'payslip#payslip', :as **=>** :slip, :via **=>** :get

**end**

    scope 'att', :as **=>** 'att' **do**

      match '' **=>** 'attendance#index', :via **=>** :get

      match 'list' **=>** 'attendance#list', :via **=>** **[**:get, :post**]**

**end**

    scope 'overtime', :as **=>** 'overtime' **do**

      scope 'rate', :as **=>** 'rate' **do**

        match '' **=>** 'overtime\_rate#index', :via **=>** :get

        match 'list' **=>** 'overtime\_rate#list', :as **=>** :list, :via **=>** **[**:get, :post**]**

        match 'new' **=>** 'overtime\_rate#new', :as **=>** :new, :via **=>** :get

        match 'create' **=>** 'overtime\_rate#create', :as **=>** :create, :via **=>** :post

        match 'edit(/:id)' **=>** 'overtime\_rate#edit', :as **=>** :edit, :via **=>** :get

        match 'update(/:id)' **=>** 'overtime\_rate#update', :as **=>** :update, :via **=>** :post

        match 'delete' **=>** 'overtime\_rate#destroy', :as **=>** :delete, :via **=>** :post

**end**

      match 'chart' **=>** 'overtime\_chart#index', :via **=>** :get

      match 'chart/data' **=>** 'overtime\_chart#data', :via **=>** **[**:get, :post**]**

**end**

    scope 'workhours', :as **=>** 'workhours' **do**

      match 'chart' **=>** 'total\_work\_hours\_chart#index', :via **=>** :get

      match 'chart/data' **=>** 'total\_work\_hours\_chart#data', :via **=>** **[**:get, :post**]**

**end**

    scope 'salaryadj', :as **=>** 'salaryadj' **do**

      match '' **=>** 'salary\_adjustment#index', :via **=>** :get

      match 'list' **=>** 'salary\_adjustment#list', :as **=>** :list, :via **=>** **[**:get, :post**]**

      match 'new' **=>** 'salary\_adjustment#new', :as **=>** :new, :via **=>** :get

      match 'create' **=>** 'salary\_adjustment#create', :as **=>** :create, :via **=>** :post

      match 'edit(/:id)' **=>** 'salary\_adjustment#edit', :as **=>** :edit, :via **=>** :get

      match 'update(/:id)' **=>** 'salary\_adjustment#update', :as **=>** :update, :via **=>** :post

      match 'delete' **=>** 'salary\_adjustment#destroy', :as **=>** :delete, :via **=>** :post

**end**

**end**

  namespace :user **do**

    match 'index' **=>** 'user#index', :as **=>** :index, :via **=>** :get

    scope 'info', :as **=>** 'info' **do**

      match '' **=>** 'info#index', :via **=>** :get

      match 'update' **=>** 'info#update', :via **=>** :post

**end**

    scope 'contact', :as **=>** 'contact' **do**

      match '' **=>** 'contact#index', :via **=>** :get

      match 'update' **=>** 'contact#update', :via **=>** :post

**end**

    scope 'job', :as **=>** 'job' **do**

      match '' **=>** 'job#index', :via **=>** :get

**end**

    scope 'salary', :as **=>** 'salary' **do**

      match '' **=>** 'salary#index', :via **=>** :get

**end**

    scope 'qualification', :as **=>** 'qualification' **do**

      match '' **=>** 'qualification#index', :via **=>** :get

      match 'update' **=>** 'qualification#update', :via **=>** :post

**end**

    scope 'overtime', :as **=>** 'overtime' **do**

      match 'chart' **=>** 'overtime\_chart#index', :via **=>** :get

      match 'chart/data' **=>** 'overtime\_chart#data', :via **=>** **[**:get, :post**]**

**end**

    scope 'workhours', :as **=>** 'workhours' **do**

      match 'chart' **=>** 'total\_work\_hours\_chart#index', :via **=>** :get

      match 'chart/data' **=>** 'total\_work\_hours\_chart#data', :via **=>** **[**:get, :post**]**

**end**

    scope 'hourly', :as **=>** 'hourly' **do**

      match 'chart' **=>** 'hourly\_payroll\_chart#index', :via **=>** :get

      match 'chart/data' **=>** 'hourly\_payroll\_chart#data', :via **=>** **[**:get, :post**]**

**end**

    scope 'payslip', :as **=>** 'payslip' **do**

      match '' **=>** 'payslip#index', :via **=>** :get

      match 'slip(/:month(/:year))' **=>** 'payslip#payslip', :as **=>** :slip, :via **=>** :get

**end**

**end**

*# The priority is based upon order of creation:*

*# first created -> highest priority.*

*# Sample of regular route:*

*# match 'products/:id' => 'catalog#view'*

*# Keep in mind you can assign values other than :controller and :action*

*# Sample of named route:*

*# match 'products/:id/purchase' => 'catalog#purchase', :as => :purchase*

*# This route can be invoked with purchase\_url(:id => product.id)*

*# Sample resource route (maps HTTP verbs to controller actions automatically):*

*# resources :products*

*# Sample resource route with options:*

*# resources :products do*

*# member do*

*# get 'short'*

*# post 'toggle'*

*# end*

*#*

*# collection do*

*# get 'sold'*

*# end*

*# end*

*# Sample resource route with sub-resources:*

*# resources :products do*

*# resources :comments, :sales*

*# resource :seller*

*# end*

*# Sample resource route with more complex sub-resources*

*# resources :products do*

*# resources :comments*

*# resources :sales do*

*# get 'recent', :on => :collection*

*# end*

*# end*

*# Sample resource route within a namespace:*

*# namespace :admin do*

*# # Directs /admin/products/\* to Admin::ProductsController*

*# # (app/controllers/admin/products\_controller.rb)*

*# resources :products*

*# end*

*# You can have the root of your site routed with "root"*

*# just remember to delete public/index.html.*

*# root :to => 'welcome#index'*

*# See how all your routes lay out with "rake routes"*

*# This is a legacy wild controller route that's not recommended for RESTful applications.*

*# Note: This route will make all actions in every controller accessible via GET requests.*

*# match ':controller(/:action(/:id))(.:format)'*

**end**

## app/models/attendance.rb

*# Model for attendance table.*

**class** **Attendance** **<** ActiveRecord::Base

  attr\_accessible :id, :staff\_id, :time\_in, :time\_out, :work\_date

  self**.**table\_name **=** 'attendance'

  belongs\_to :employee, :foreign\_key **=>** 'staff\_id', :primary\_key **=>** 'staff\_id'

  validates\_presence\_of :work\_date, :message **=>** 'Work date is required'

  validates\_presence\_of :time\_in, :message **=>** 'Time in is required'

  validates\_presence\_of :time\_out, :message **=>** 'Time out is required'

**end**

## app/models/department.rb

*# Model for department table.*

**class** **Department** **<** ActiveRecord::Base

  attr\_accessible :id, :name

  self**.**table\_name **=** 'department'

  validates\_presence\_of :name, :message **=>** 'Name is required'

  validates\_uniqueness\_of :name, :message **=>** "Department %{value} already exist"

**end**

## app/models/designation.rb

*# Model for designation table.*

**class** **Designation** **<** ActiveRecord::Base

  attr\_accessible :id, :title, :desc, :note

  self**.**table\_name **=** 'designation'

  has\_many :employee\_job

  validates\_presence\_of :title, :message **=>** 'Job Title is required'

  validates\_uniqueness\_of :title, :message **=>** "Job Title %{value} already exist"

**end**

## app/models/employee.rb

*# Model for employee table.*

**class** **Employee** **<** ActiveRecord::Base

  attr\_accessible :dob, :staff\_id, :user\_id, :first\_name, :gender, :id, :is\_bumi,

                  :last\_name, :marital\_status, :middle\_name, :nationality, :new\_ic,

                  :old\_ic, :passport\_no, :place\_of\_birth, :race, :religion

  self**.**table\_name **=** 'employee'

  has\_one :employee\_contact, :foreign\_key **=>** 'id'

  has\_one :employee\_job, :foreign\_key **=>** 'id'

  has\_one :employee\_salary, :foreign\_key **=>** 'id'

  has\_one :employee\_qualification, :foreign\_key **=>** 'id'

  has\_one :employee, :foreign\_key **=>** 'id'

  has\_many :attendance, :foreign\_key **=>** 'staff\_id', :primary\_key **=>** 'staff\_id'

  belongs\_to :user

  validates\_presence\_of :staff\_id, :message **=>** 'Employee ID is required'

  validates\_presence\_of :first\_name, :message **=>** 'First Name is required'

  validates\_presence\_of :last\_name, :message **=>** 'Last Name is required'

  validates\_presence\_of :new\_ic, :message **=>** 'New IC No. is required'

  validates\_presence\_of :gender, :message **=>** 'Gender is required'

  validates\_presence\_of :marital\_status, :message **=>** 'Marital Status is required'

  validates\_presence\_of :nationality, :message **=>** 'Nationality is required'

  validates\_presence\_of :dob, :message **=>** 'Date of Birth is required'

  validates\_presence\_of :place\_of\_birth, :message **=>** 'Place of Birth is required'

  validates\_presence\_of :race, :message **=>** 'Race is required'

  validates\_uniqueness\_of :staff\_id, :message **=>** "Employee ID %{value} already exist"

**end**

## app/models/employee\_contact.rb

*# Model for employee\_contact table.*

**class** **EmployeeContact** **<** ActiveRecord::Base

  attr\_accessible :address\_1, :address\_2, :address\_3, :city, :country, :home\_phone, :id,

                  :mobile\_phone, :other\_email, :postcode, :state, :work\_email

  self**.**table\_name **=** 'employee\_contact'

  belongs\_to :employee, :foreign\_key **=>** 'id'

  validates\_presence\_of :address\_1, :message **=>** 'Address 1 is required'

  validates\_presence\_of :city, :message **=>** 'City is required'

  validates\_presence\_of :state, :message **=>** 'State is required'

  validates\_presence\_of :postcode, :message **=>** 'Postal Code is required'

  validates\_presence\_of :country, :message **=>** 'Country is required'

  validates\_presence\_of :work\_email, :message **=>** 'Work Email is required'

**end**

## app/models/employee\_job.rb

*# Model for employee\_job table.*

**class** **EmployeeJob** **<** ActiveRecord::Base

  attr\_accessible :confirm\_date, :department\_id, :designation\_id, :employment\_status\_id,

                  :id, :job\_category\_id, :join\_date

  self**.**table\_name **=** 'employee\_job'

  belongs\_to :employee, :foreign\_key **=>** 'id'

  belongs\_to :designation

  belongs\_to :department

  belongs\_to :employment\_status

  belongs\_to :job\_category

  validates\_presence\_of :designation\_id, :message **=>** 'Designation is required'

  validates\_presence\_of :department\_id, :message **=>** 'Department is required'

  validates\_presence\_of :employment\_status\_id, :message **=>** 'Employment Status is required'

  validates\_presence\_of :job\_category\_id, :message **=>** 'Job Category is required'

  validates\_presence\_of :join\_date, :message **=>** 'Join Date is required'

**end**

## app/models/employee\_qualification.rb

*# Model for employee\_qualification table.*

**class** **EmployeeQualification** **<** ActiveRecord::Base

  attr\_accessible :end\_date, :gpa, :id, :institute, :level, :major, :start\_date, :year

  self**.**table\_name **=** 'employee\_qualification'

  belongs\_to :employee, :foreign\_key **=>** 'id'

  validates\_presence\_of :level, :message **=>** 'Qualification Level is required'

  validates\_presence\_of :institute, :message **=>** 'Institute name is required'

  validates\_presence\_of :year, :message **=>** 'Year obtained is required'

  validates\_presence\_of :start\_date, :message **=>** 'Start Date is required'

  validates\_presence\_of :end\_date, :message **=>** 'End Date is required'

**end**

## app/models/employee\_salary.rb

*# Model for employee\_salary table.*

**class** **EmployeeSalary** **<** ActiveRecord::Base

  attr\_accessible :allowance, :bank\_acc\_no, :bank\_acc\_type, :bank\_address, :bank\_name,

                  :epf, :epf\_no, :id, :income\_tax, :income\_tax\_no, :salary, :socso\_no,

                  :pay\_type, :socso

  self**.**table\_name **=** 'employee\_salary'

  belongs\_to :employee, :foreign\_key **=>** 'id'

  validates\_presence\_of :salary, :message **=>** 'Salary is required'

  validates\_presence\_of :bank\_name, :message **=>** 'Bank Name is required'

  validates\_presence\_of :bank\_acc\_no, :message **=>** 'Bank Account No. is required'

  validates\_presence\_of :bank\_acc\_type, :message **=>** 'Bank Account Type is required'

  validates\_presence\_of :bank\_address, :message **=>** 'Bank Address is required'

  validates\_presence\_of :epf\_no, :message **=>** 'EPF No. is required'

  validates\_presence\_of :pay\_type, :message **=>** 'Pay Type is required'

  validates\_presence\_of :epf, :message **=>** 'EPF Deduction is required'

  validates\_numericality\_of :salary, :greater\_than\_or\_equal\_to **=>** 0,

                                     :message **=>** 'Salary is invalid'

  validates\_numericality\_of :allowance, :greater\_than\_or\_equal\_to **=>** 0,

                                        :message **=>** 'Allowance is invalid'

  validates\_numericality\_of :epf, :greater\_than\_or\_equal\_to **=>** 0,

                                  :message **=>** 'EPF Deduction is invalid'

  validates\_numericality\_of :socso, :greater\_than\_or\_equal\_to **=>** 0,

                                    :message **=>** 'SOCSO Deduction is invalid'

  validates\_numericality\_of :income\_tax, :greater\_than\_or\_equal\_to **=>** 0,

                                         :message **=>** 'Income Tax Deduction is invalid'

**def** **allowance**

    a **=** read\_attribute(:allowance)

    a**.**blank? ? 0 : a

**end**

**def** **epf**

    a **=** read\_attribute(:epf)

    a**.**blank? ? 0 : a

**end**

**def** **income\_tax**

    a **=** read\_attribute(:income\_tax)

    a**.**blank? ? 0 : a

**end**

**def** **salary**

    a **=** read\_attribute(:salary)

    a**.**blank? ? 0 : a

**end**

**def** **socso**

    a **=** read\_attribute(:socso)

    a**.**blank? ? 0 : a

**end**

**def** **display\_pay\_type**

**case** self**.**pay\_type

**when** 1

      'Monthly'

**when** 2

      'Hourly'

**else**

      ''

**end**

**end**

**end**

## app/models/employment\_status.rb

*# Model for employment\_status table.*

**class** **EmploymentStatus** **<** ActiveRecord::Base

  attr\_accessible :id, :name

  self**.**table\_name **=** 'employment\_status'

  has\_many :employee\_job

  validates\_presence\_of :name, :message **=>** 'Name is required'

  validates\_uniqueness\_of :name, :message **=>** "Employment Status %{value} already exist"

**end**

## app/models/job\_category.rb

*# Model for job\_category table.*

**class** **JobCategory** **<** ActiveRecord::Base

  attr\_accessible :id, :name

  self**.**table\_name **=** 'job\_category'

  has\_many :employee\_job

  validates\_presence\_of :name, :message **=>** 'Name is required'

  validates\_uniqueness\_of :name, :message **=>** "Category %{value} already exist"

**end**

## app/models/overtime\_rate.rb

*# Model for overtime\_rate table.*

**class** **OvertimeRate** **<** ActiveRecord::Base

  attr\_accessible :duration, :pay\_rate, :year

  self**.**table\_name **=** 'overtime\_rate'

  validates\_presence\_of :duration, :message **=>** 'Duration is required'

  validates\_presence\_of :year, :message **=>** 'Year is required'

  validates\_presence\_of :pay\_rate, :message **=>** 'Pay Rate is required'

  validates\_numericality\_of :duration, :greater\_than\_or\_equal\_to **=>** 0,

                            :message **=>** 'Duration is invalid'

  validates\_numericality\_of :year, :greater\_than **=>** 0, :message **=>** 'Year is invalid'

  validates\_numericality\_of :pay\_rate, :greater\_than **=>** 0,

                            :message **=>** 'Pay Rate is invalid'

  validates\_uniqueness\_of :year,

                          :message **=>** "Overtime rate for year %{value} already exist"

**def** **duration**

    a **=** read\_attribute(:duration)

    a**.**blank? ? 0 : a

**end**

**def** **pay\_rate**

    a **=** read\_attribute(:pay\_rate)

    a**.**blank? ? 0 : a

**end**

**end**

## app/models/pay\_rate.rb

*# Model for pay\_rate table.*

**class** **PayRate** **<** ActiveRecord::Base

  attr\_accessible :hourly\_pay\_rate, :id, :month, :staff\_id, :year

  self**.**table\_name **=** 'pay\_rate'

  validates\_presence\_of :staff\_id, :message **=>** 'Staff ID is required'

  validates\_presence\_of :month, :message **=>** 'Month is required'

  validates\_presence\_of :year, :message **=>** 'Year is required'

  validates\_presence\_of :hourly\_pay\_rate, :message **=>** 'Hourly pay rate is required'

  validates\_numericality\_of :month, :greater\_than **=>** 0, :message **=>** 'Month is invalid'

  validates\_numericality\_of :month, :less\_than\_or\_equal\_to **=>** 12,

                                    :message **=>** 'Month is invalid'

  validates\_numericality\_of :year, :greater\_than **=>** 0, :message **=>** 'Year is invalid'

  validates\_numericality\_of :hourly\_pay\_rate, :greater\_than **=>** 0,

                                              :message **=>** 'Hourly pay rate is invalid'

**def** **hourly\_pay\_rate**

    a **=** read\_attribute(:hourly\_pay\_rate)

    a**.**blank? ? 0 : a

**end**

**end**

## app/models/salary\_adjustment.rb

*# Model for salary\_adjustment table.*

**class** **SalaryAdjustment** **<** ActiveRecord::Base

  attr\_accessible :id, :inc, :month, :staff\_id, :year

  self**.**table\_name **=** 'salary\_adjustment'

  validates\_presence\_of :staff\_id, :message **=>** 'Staff ID is required'

  validates\_presence\_of :inc, :message **=>** 'Increment is required'

  validates\_presence\_of :month, :message **=>** 'Month is required'

  validates\_presence\_of :year, :message **=>** 'Year is required'

  validates\_numericality\_of :inc, :greater\_than **=>** 0,

                                  :message **=>** 'Increment is invalid'

  validates\_numericality\_of :month, :greater\_than **=>** 0, :message **=>** 'Month is invalid'

  validates\_numericality\_of :month, :less\_than\_or\_equal\_to **=>** 12,

                                    :message **=>** 'Month is invalid'

  validates\_numericality\_of :year, :greater\_than **=>** 0, :message **=>** 'Year is invalid'

**def** **inc**

    a **=** read\_attribute(:inc)

    a**.**blank? ? 0 : a

**end**

**end**

## app/models/user.rb

require 'digest'

*# Model for user table.*

**class** **User** **<** ActiveRecord::Base

  attr\_accessible :id, :pwd, :role, :status, :username, :pwd\_confirmation

**attr\_accessor** :pwd

  self**.**table\_name **=** 'user'

  has\_one :employee

  validates :username, :uniqueness **=>** { :message **=>** "Username %{value} already exist" },

                       :length **=>** { :within **=>** 3**.**.50,

                                    :message **=>** "Minimum is %{count} characters" }

  validates :pwd, :confirmation **=>** { :message **=>** "Password doesn't match confirmation" },

                  :length **=>** { :within **=>** 4**.**.20,

                               :message **=>** "Minimum is %{count} characters" },

                  :presence **=>** { :message **=>** "Password is required" },

                  :if **=>** :password\_required?

  before\_save :encrypt\_new\_password

  UNCHANGED\_PASSWORD **=** '\*\*\*\*\*\*\*\*'

  ADMIN **=** 1

  NORMAL\_USER **=** 2

  @@roles **=** { 'Admin' **=>** 1, 'Normal User' **=>** 2 }

  @@statuses **=** { 'Enabled' **=>** 1, 'Disabled' **=>** 0 }

**def** **self.authenticate**(username, password)

    user **=** find\_by\_username(username)

**if** user **&&** user**.**authenticated?(password) **&&** user**.**enabled?

**return** user

**end**

**end**

**def** **authenticated?**(password)

    self**.**password **==** encrypt(password)

**end**

**def** **enabled?**

    self**.**status **==** **true**

**end**

**def** **self.roles**

    @@roles

**end**

**def** **self.statuses**

    @@statuses

**end**

**def** **role\_display**

    role **==** 1 **?** 'Admin' : 'Normal User'

**end**

**def** **status\_display**

    status **==** **true** ? 'Enabled' : 'Disabled'

**end**

**protected**

**def** **encrypt\_new\_password**

**return** **if** pwd**.**blank?

    self**.**password **=** encrypt(pwd)

**end**

**def** **password\_required?**

**if** pwd **==** UNCHANGED\_PASSWORD

**return** **false**

**end**

    self**.**password**.**blank? **||** pwd**.**present?

**end**

**def** **encrypt**(string)

    Digest::SHA1**.**hexdigest(string)

**end**

**end**

## app/controllers/application\_controller.rb

*# The main application's controller.*

**class** **ApplicationController** **<** ActionController::Base

  protect\_from\_forgery

  LAYOUT **=** {

    :admin **=>** 'admin',

    :chart **=>** 'chart',

    :list **=>** 'list',

    :user **=>** 'user'

  }

*# Authenticates a user.*

**def** **create**

    user **=** User**.**authenticate(params**[**:username**]**, params**[**:password**]**)

**if** user**.**present?

      session**[**:user\_id**]** **=** user**.**id

**if** user**.**role **==** User::ADMIN

        redirect\_to admin\_index\_path **and** **return**

**else**

        employee **=** user**.**employee

**if** employee**.**present?

          session**[**:employee\_id**]** **=** employee**.**id

          session**[**:staff\_id**]** **=** employee**.**staff\_id

          session**[**:supervisor\_id**]** **=** employee**.**id

          redirect\_to user\_index\_path **and** **return**

**else**

          flash**.**now**[**:alert**]** **=** %Q{No employee record found. Please contact the administrator

to create your employee record.}

**end**

**end**

**else**

      flash**.**now**[**:alert**]** **=** 'Incorrect username or password'

**end**

    render :action **=>** 'new'

**end**

*# Logs out a user.*

**def** **destroy**

    reset\_session

    redirect\_to login\_path

**end**

**protected**

*# Returns the current user's user\_id.*

**def** **current\_user**

**return** **unless** session**[**:user\_id**]**

    @current\_user **||=** session**[**:user\_id**]**

**end**

*# Checks whether a user is authenticated.*

**def** **authenticate**

    logged\_in? ? **true** : access\_denied

**end**

*# Checks whether a user is logged in.*

**def** **logged\_in?**

    current\_user**.**present?

**end**

*# Prevents a user from accessing the system without logging in.*

**def** **access\_denied**

    redirect\_to login\_path, :notice **=>** 'Please log in to continue' **and** **return** **false**

**end**

*# Formats a date into dd-mm-yyyy .*

*# Helper method.*

**def** **fmt\_date**(dt)

**if** dt**.**present?

      dt**.**strftime('%d-%m-%Y')

**end**

**end**

*# Formats a time into hh:mm:ss AM/PM .*

*# Helper method.*

**def** **fmt\_time**(t)

**if** t**.**present?

      t**.**in\_time\_zone('Kuala Lumpur')**.**strftime('%l:%M %p')

**end**

**end**

*# Returns the month name from a given number.*

*# Helper method.*

**def** **month\_name**(i)

    ApplicationHelper**.**month\_name(i)

**end**

*# Returns a hash of month names.*

**def** **month\_options**

    months **=** I18n**.**t('date.month\_names')

    o **=** {}

    (1**.**.12)**.**each **do** **|**m**|**

      o**[**months**[**m**]]** **=** m

**end**

    o

**end**

*# Returns the end year for year selection in view.*

*# Helper method.*

**def** **end\_year**

    2000

**end**

*# Returns the user\_id.*

**def** **get\_user\_id**

    session**[**:user\_id**]**

**end**

  helper\_method :current\_user

  helper\_method :logged\_in

  helper\_method :fmt\_date

  helper\_method :fmt\_time

  helper\_method :month\_name

  helper\_method :end\_year

**end**

## app/controllers/admin/admin\_controller.rb

*# The main application's controller.*

**class** **ApplicationController** **<** ActionController::Base

  protect\_from\_forgery

  LAYOUT **=** {

    :admin **=>** 'admin',

    :chart **=>** 'chart',

    :list **=>** 'list',

    :user **=>** 'user'

  }

*# Authenticates a user.*

**def** **create**

    user **=** User**.**authenticate(params**[**:username**]**, params**[**:password**]**)

**if** user**.**present?

      session**[**:user\_id**]** **=** user**.**id

**if** user**.**role **==** User::ADMIN

        redirect\_to admin\_index\_path **and** **return**

**else**

        employee **=** user**.**employee

**if** employee**.**present?

          session**[**:employee\_id**]** **=** employee**.**id

          session**[**:staff\_id**]** **=** employee**.**staff\_id

          session**[**:supervisor\_id**]** **=** employee**.**id

          redirect\_to user\_index\_path **and** **return**

**else**

          flash**.**now**[**:alert**]** **=** %Q{No employee record found. Please contact the administrator

to create your employee record.}

**end**

**end**

**else**

      flash**.**now**[**:alert**]** **=** 'Incorrect username or password'

**end**

    render :action **=>** 'new'

**end**

*# Logs out a user.*

**def** **destroy**

    reset\_session

    redirect\_to login\_path

**end**

**protected**

*# Returns the current user's user\_id.*

**def** **current\_user**

**return** **unless** session**[**:user\_id**]**

    @current\_user **||=** session**[**:user\_id**]**

**end**

*# Checks whether a user is authenticated.*

**def** **authenticate**

    logged\_in? ? **true** : access\_denied

**end**

*# Checks whether a user is logged in.*

**def** **logged\_in?**

    current\_user**.**present?

**end**

*# Prevents a user from accessing the system without logging in.*

**def** **access\_denied**

    redirect\_to login\_path, :notice **=>** 'Please log in to continue' **and** **return** **false**

**end**

*# Formats a date into dd-mm-yyyy .*

*# Helper method.*

**def** **fmt\_date**(dt)

**if** dt**.**present?

      dt**.**strftime('%d-%m-%Y')

**end**

**end**

*# Formats a time into hh:mm:ss AM/PM .*

*# Helper method.*

**def** **fmt\_time**(t)

**if** t**.**present?

      t**.**in\_time\_zone('Kuala Lumpur')**.**strftime('%l:%M %p')

**end**

**end**

*# Returns the month name from a given number.*

*# Helper method.*

**def** **month\_name**(i)

    ApplicationHelper**.**month\_name(i)

**end**

*# Returns a hash of month names.*

**def** **month\_options**

    months **=** I18n**.**t('date.month\_names')

    o **=** {}

    (1**.**.12)**.**each **do** **|**m**|**

      o**[**months**[**m**]]** **=** m

**end**

    o

**end**

*# Returns the end year for year selection in view.*

*# Helper method.*

**def** **end\_year**

    2000

**end**

*# Returns the user\_id.*

**def** **get\_user\_id**

    session**[**:user\_id**]**

**end**

  helper\_method :current\_user

  helper\_method :logged\_in

  helper\_method :fmt\_date

  helper\_method :fmt\_time

  helper\_method :month\_name

  helper\_method :end\_year

**end**

## app/controllers/admin/attendance\_controller.rb

*# This controller serves incoming requests to display out the Attendance records.*

**class** **Admin::**AttendanceController **<** Admin::AdminController

*# List all records.*

*# GET /att*

*# GET /att.json*

**def** **index**

    @data **=** AttendanceHelper**.**get\_all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /att/list*

*# GET /att/list.json*

**def** **list**

    \_work\_date **=** params**[**:work\_date**]**

    employee **=** params**[**:employee**].**blank? ? '' : params**[**:employee**]**

    work\_date **=** Date**.**strptime(\_work\_date, '%d-%m-%Y') **if** \_work\_date**.**present?

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? AttendanceHelper::DEFAULT\_SORT\_COLUMN

                                            : params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? AttendanceHelper::DEFAULT\_SORT\_DIR :

      params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

    filters **=** { :work\_date **=>** work\_date,

                :employee **=>** employee }

**if** work\_date**.**blank? **&&** employee**.**blank?

      @data **=** AttendanceHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** AttendanceHelper**.**get\_filter\_by(filters, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

**end**

## app/controllers/admin/department\_controller.rb

*# This controller serves incoming requests to display out the Department records.*

**class** **Admin::**DepartmentController **<** Admin::AdminController

*# List all records.*

*# GET /dept*

*# GET /dept.json*

**def** **index**

    @data **=** DepartmentHelper**.**get\_all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /dept/list*

*# GET /dept/list.json*

**def** **list**

    keyword **=** params**[**:keyword**].**blank? ? '' : params**[**:keyword**]**

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? DepartmentHelper::DEFAULT\_SORT\_COLUMN

                                            : params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? DepartmentHelper::DEFAULT\_SORT\_DIR :

      params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

**if** keyword**.**blank?

      @data **=** DepartmentHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** DepartmentHelper**.**get\_filter\_by(keyword, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the Department form.*

*# GET /dept/new*

*# GET /dept/new.json*

**def** **new**

    @dept **=** Department**.**new

    @form\_id **=** 'add-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @dept }

**end**

**end**

*# Create new Department record.*

*# POST /dept/create*

**def** **create**

    o **=** Department**.**new(:name **=>** params**[**:name**]**)

**if** o**.**save

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Department was successfully added.' }

**else**

      render :json **=>** DepartmentHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Display the Department form, with existing record to edit.*

*# GET /dept/edit/1*

*# GET /dept/edit/1.json*

**def** **edit**

    @dept **=** Department**.**find(params**[**:id**]**)

    @form\_id **=** 'edit-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @dept }

**end**

**end**

*# Update Department record.*

*# POST /dept/update/1*

**def** **update**

    o **=** Department**.**find(params**[**:id**]**)

**if** o**.**update\_attributes(:name **=>** params**[**:name**]**)

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Department was successfully updated.' }

**else**

      render :json **=>** DepartmentHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Delete a list of Department records.*

*# POST /dept/delete*

**def** **destroy**

    keyword **=** params**[**:keyword**].**blank? ? '' : params**[**:keyword**]**

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    ids **=** params**[**:id**]**

    Department**.**delete\_all(:id **=>** ids)

    itemscount **=** DepartmentHelper**.**item\_message(keyword, pgnum, pgsize)

    render :json **=>** { :success **=>** 1,

                      :itemscount **=>** itemscount,

                      :message **=>** "#{ids**.**size} Department(s) was successfully deleted." }

**end**

**end**

## app/controllers/admin/designation\_controller.rb

*# This controller serves incoming requests to display out the Designation records.*

**class** **Admin::**DesignationController **<** Admin::AdminController

*# List all records.*

*# GET /designation*

*# GET /designation.json*

**def** **index**

    @data **=** DesignationHelper**.**get\_all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /designation/list*

*# GET /designation/list.json*

**def** **list**

    find **=** params**[**:find**].**blank? ? 0 : params**[**:find**].**to\_i

    keyword **=** params**[**:keyword**].**blank? ? '' : params**[**:keyword**]**

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? DesignationHelper::DEFAULT\_SORT\_COLUMN

                                            : params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? DesignationHelper::DEFAULT\_SORT\_DIR :

      params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

**if** find **==** 0 **&&** keyword**.**blank?

      @data **=** DesignationHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** DesignationHelper**.**get\_filter\_by(find, keyword, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the Designation form.*

*# GET /designation/new*

*# GET /designation/new.json*

**def** **new**

    @designation **=** Designation**.**new

    @form\_id **=** 'add-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @designation }

**end**

**end**

*# Create new Designation record.*

*# POST /designation/create*

**def** **create**

    o **=** Designation**.**new(:title **=>** params**[**:title**]**, :desc **=>** params**[**:desc**]**,

                        :note **=>** params**[**:note**]**)

**if** o**.**save

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Job Title was successfully added.' }

**else**

      render :json **=>** DesignationHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Display the Designation form, with existing record to edit.*

*# GET /designation/edit/1*

*# GET /designation/edit/1.json*

**def** **edit**

    @designation **=** Designation**.**find(params**[**:id**]**)

    @form\_id **=** 'edit-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @designation }

**end**

**end**

*# Update Designation record.*

*# POST /designation/update/1*

**def** **update**

    o **=** Designation**.**find(params**[**:id**]**)

**if** o**.**update\_attributes(:title **=>** params**[**:title**]**, :desc **=>** params**[**:desc**]**,

                           :note **=>** params**[**:note**]**)

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Job Title was successfully updated.' }

**else**

      render :json **=>** DesignationHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Delete a list of Designation records.*

*# POST /designation/delete*

**def** **destroy**

    keyword **=** params**[**:keyword**].**blank? ? '' : params**[**:keyword**]**

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    ids **=** params**[**:id**]**

    Designation**.**delete\_all(:id **=>** ids)

    itemscount **=** DesignationHelper**.**item\_message(keyword, pgnum, pgsize)

    render :json **=>** { :success **=>** 1,

                      :itemscount **=>** itemscount,

                      :message **=>** "#{ids**.**size} Job Title(s) was successfully deleted." }

**end**

**end**

## app/controllers/admin/employee\_controller.rb

require 'securerandom'

*# This controller serves incoming requests to display out the Employee records.*

**class** **Admin::**EmployeeController **<** Admin::AdminController

*# List all records.*

*# GET /employee*

*# GET /employee.json*

**def** **index**

    @data **=** EmployeeHelper**.**get\_all

    @employmentstatus **=** EmploymentStatus**.**order(:name)**.**all

    @designation **=** Designation**.**order(:title)**.**all

    @dept **=** Department**.**order(:name)**.**all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /employee/list*

*# GET /employee/list.json*

**def** **list**

    employee **=** params**[**:employee**].**blank? ? '' : params**[**:employee**]**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    employment\_status **=** params**[**:employment\_status**].**blank? ? 0 :

      params**[**:employment\_status**].**to\_i

    designation **=** params**[**:designation**].**blank? ? 0 : params**[**:designation**].**to\_i

    dept **=** params**[**:dept**].**blank? ? 0 : params**[**:dept**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? EmployeeHelper::DEFAULT\_SORT\_COLUMN

                                            : params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? EmployeeHelper::DEFAULT\_SORT\_DIR :

      params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

    filters **=** { :employee **=>** employee,

                :staff\_id **=>** staff\_id,

                :employment\_status **=>** employment\_status,

                :designation **=>** designation,

                :dept **=>** dept }

**if** employee**.**blank? **&&** staff\_id**.**blank? **&&** employment\_status **==** 0 **&&** designation **==** 0 **&&**

      dept **==** 0

      @data **=** EmployeeHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** EmployeeHelper**.**get\_filter\_by(filters, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the Employee form.*

*# GET /employee/new*

*# GET /employee/new.json*

**def** **new**

    @employee **=** Employee**.**new

    @employee\_contact **=** EmployeeContact**.**new

    @employee\_job **=** EmployeeJob**.**new

    @employee\_salary **=** EmployeeSalary**.**new

    @employee\_qualification **=** EmployeeQualification**.**new

    @form\_id **=** 'add-form'

    @users **=** User**.**order(:username)**.**all

    @designations **=** Designation**.**order(:title)**.**all

    @employment\_statuses **=** EmploymentStatus**.**order(:name)**.**all

    @job\_categories **=** JobCategory**.**order(:name)**.**all

    @departments **=** Department**.**order(:name)**.**all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @employee }

**end**

**end**

*# Create new Employee record.*

*# POST /employee/create*

**def** **create**

    o **=** EmployeeHelper**.**employee\_obj(params)

    b1 **=** EmployeeContactHelper**.**is\_empty\_params?(params)

    oc **=** EmployeeContactHelper**.**employee\_contact\_obj(o, params)

    b2 **=** EmployeeJobHelper**.**is\_empty\_params?(params)

    oej **=** EmployeeJobHelper**.**employee\_job\_obj(o, params)

    b3 **=** EmployeeSalaryHelper**.**is\_empty\_params?(params)

    osa **=** EmployeeSalaryHelper**.**employee\_salary\_obj(o, params)

    b4 **=** EmployeeQualificationHelper**.**is\_empty\_params?(params)

    oq **=** EmployeeQualificationHelper**.**employee\_qualification\_obj(o, params)

    v1 **=** o**.**valid?

    v2 **=** b1 ? **true** : oc**.**valid?

    v3 **=** b2 ? **true** : oej**.**valid?

    v4 **=** b3 ? **true** : osa**.**valid?

    v5 **=** b4 ? **true** : oq**.**valid?

**if** **!**v1 **||** **!**v2 **||** **!**v3 **||** **!**v4 **||** **!**v5

      employee\_errors **=** EmployeeHelper**.**get\_errors(o**.**errors)

      employee\_contact\_errors **=** EmployeeContactHelper**.**get\_errors(oc**.**errors)

      employee\_job\_errors **=** EmployeeJobHelper**.**get\_errors(oej**.**errors)

      employee\_salary\_errors **=** EmployeeSalaryHelper**.**get\_errors(osa**.**errors)

      employee\_qualification\_errors **=** EmployeeQualificationHelper**.**get\_errors(oq**.**errors)

      errors **=** { :error **=>** 1, :employee **=>** employee\_errors,

                              :employee\_contact **=>** employee\_contact\_errors,

                              :employee\_job **=>** employee\_job\_errors,

                              :employee\_salary **=>** employee\_salary\_errors,

                              :employee\_qualification **=>** employee\_qualification\_errors }

      render :json **=>** errors **and** **return**

**end**

    ActiveRecord::Base**.**transaction **do**

      o**.**save

      oc**.**save **if** b1

      oej**.**save **if** b2

      osa**.**save **if** b3

      oq**.**save **if** b4

**end**

    render :json **=>** { :success **=>** 1,

                      :message **=>** 'Employee was successfully added.' }

**end**

*# Display the Employee form, with existing record to edit.*

*# GET /employee/edit/1*

*# GET /employee/edit/1.json*

**def** **edit**

    @employee **=** Employee**.**find(params**[**:id**]**)

    @employee\_contact **=** @employee**.**employee\_contact**.**blank? ? EmployeeContact**.**new

                                                          : @employee**.**employee\_contact

    @employee\_job **=** @employee**.**employee\_job**.**blank? ? EmployeeJob**.**new

                                                  : @employee**.**employee\_job

    @employee\_salary **=** @employee**.**employee\_salary**.**blank? ? EmployeeSalary**.**new

                                                        : @employee**.**employee\_salary

    @employee\_qualification **=** @employee**.**employee\_qualification**.**blank? ?

      EmployeeQualification**.**new : @employee**.**employee\_qualification

    @form\_id **=** 'edit-form'

    @users **=** User**.**order(:username)**.**all

    @designations **=** Designation**.**order(:title)**.**all

    @employment\_statuses **=** EmploymentStatus**.**order(:name)**.**all

    @job\_categories **=** JobCategory**.**order(:name)**.**all

    @departments **=** Department**.**order(:name)**.**all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @employee }

**end**

**end**

*# Update Employee record.*

*# POST /employee/update*

**def** **update**

    o **=** Employee**.**find(params**[**:id**]**)

    oc **=** o**.**employee\_contact

    oej **=** o**.**employee\_job

    osa **=** o**.**employee\_salary

    oq **=** o**.**employee\_qualification

    oc\_new **=** **false**

**if** oc**.**blank?

      oc **=** EmployeeContactHelper**.**employee\_contact\_obj(o, params)

      oc\_new **=** **true**

**end**

    oej\_new **=** **false**

**if** oej**.**blank?

      oej **=** EmployeeJobHelper**.**employee\_job\_obj(o, params)

      oej\_new **=** **true**

**end**

    osa\_new **=** **false**

**if** osa**.**blank?

      osa **=** EmployeeSalaryHelper**.**employee\_salary\_obj(o, params)

      osa\_new **=** **true**

**end**

    oq\_new **=** **false**

**if** oq**.**blank?

      oq **=** EmployeeQualificationHelper**.**employee\_qualification\_obj(o, params)

      oq\_new **=** **true**

**end**

    b1 **=** EmployeeContactHelper**.**is\_empty\_params?(params)

    b2 **=** EmployeeJobHelper**.**is\_empty\_params?(params)

    b3 **=** EmployeeSalaryHelper**.**is\_empty\_params?(params)

    b4 **=** EmployeeQualificationHelper**.**is\_empty\_params?(params)

    v1 **=** o**.**valid?

    v2 **=** b1 ? **true** : oc**.**valid?

    v3 **=** b2 ? **true** : oej**.**valid?

    v4 **=** b3 ? **true** : osa**.**valid?

    v5 **=** b4 ? **true** : oq**.**valid?

**if** **!**v1 **||** **!**v2 **||** **!**v3 **||** **!**v4 **||** **!**v5

      employee\_errors **=** EmployeeHelper**.**get\_errors(o**.**errors)

      employee\_contact\_errors **=** EmployeeContactHelper**.**get\_errors(oc**.**errors)

      employee\_job\_errors **=** EmployeeJobHelper**.**get\_errors(oej**.**errors)

      employee\_salary\_errors **=** EmployeeSalaryHelper**.**get\_errors(osa**.**errors)

      employee\_qualification\_errors **=** EmployeeQualificationHelper**.**get\_errors(oq**.**errors)

      errors **=** { :error **=>** 1, :employee **=>** employee\_errors,

                              :employee\_contact **=>** employee\_contact\_errors,

                              :employee\_job **=>** employee\_job\_errors,

                              :employee\_salary **=>** employee\_salary\_errors,

                              :employee\_qualification **=>** employee\_qualification\_errors }

      render :json **=>** errors **and** **return**

**end**

    ActiveRecord::Base**.**transaction **do**

      EmployeeHelper**.**update\_obj(o, params)

**if** oc\_new

        oc**.**save

**else**

        EmployeeContactHelper**.**update\_obj(oc, params)

**end**

**if** oej\_new

        oej**.**save

**else**

        EmployeeJobHelper**.**update\_obj(oej, params)

**end**

**if** osa\_new

        osa**.**save

**else**

        EmployeeSalaryHelper**.**update\_obj(osa, params)

**end**

**if** oq\_new

        oq**.**save

**else**

        EmployeeQualificationHelper**.**update\_obj(oq, params)

**end**

**end**

    render :json **=>** { :success **=>** 1,

                      :message **=>** 'Employee was successfully updated.' }

**end**

*# Delete a list of Employee records.*

*# POST /employee/delete*

**def** **destroy**

    employee **=** params**[**:employee**].**blank? ? '' : params**[**:employee**]**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    employment\_status **=** params**[**:employment\_status**].**blank? ? 0 : params**[**:employment\_status**].**to\_i

    designation **=** params**[**:designation**].**blank? ? 0 : params**[**:designation**].**to\_i

    dept **=** params**[**:dept**].**blank? ? 0 : params**[**:dept**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    ids **=** params**[**:id**]**

    Employee**.**delete\_all(:id **=>** ids)

    filters **=** { :employee **=>** employee,

                :staff\_id **=>** staff\_id,

                :employment\_status **=>** employment\_status,

                :designation **=>** designation,

                :dept **=>** dept }

**if** employee**.**blank? **&&** staff\_id**.**blank? **&&** employment\_status **==** 0 **&&** designation **==** 0 **&&**

      dept **==** 0

      itemscount **=** EmployeeHelper**.**item\_message(**nil**, pgnum, pgsize)

**else**

      itemscount **=** EmployeeHelper**.**item\_message(filters, pgnum, pgsize)

**end**

    render :json **=>** { :success **=>** 1,

                      :itemscount **=>** itemscount,

                      :message **=>** "#{ids**.**size} employee(s) was successfully deleted." }

**end**

**end**

## app/controllers/admin/employment\_status\_controller.rb

*# This controller serves incoming requests to display out the EmploymentStatus records.*

**class** **Admin::**EmploymentStatusController **<** Admin::AdminController

*# List all records.*

*# GET /empstatus*

*# GET /empstatus.json*

**def** **index**

    @data **=** EmploymentStatusHelper**.**get\_all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /empstatus/list*

*# GET /empstatus/list.json*

**def** **list**

    keyword **=** params**[**:keyword**].**blank? ? '' : params**[**:keyword**]**

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? EmploymentStatusHelper::DEFAULT\_SORT\_COLUMN

                                            : params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? EmploymentStatusHelper::DEFAULT\_SORT\_DIR

                                      : params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

**if** keyword**.**blank?

      @data **=** EmploymentStatusHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** EmploymentStatusHelper**.**get\_filter\_by(keyword, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the EmploymentStatus form.*

*# GET /empstatus/new*

*# GET /empstatus/new.json*

**def** **new**

    @empstatus **=** EmploymentStatus**.**new

    @form\_id **=** 'add-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @empstatus }

**end**

**end**

*# Create new EmploymentStatus record.*

*# POST /empstatus/create*

**def** **create**

    o **=** EmploymentStatus**.**new(:name **=>** params**[**:name**]**)

**if** o**.**save

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Employment Status was successfully added.' }

**else**

      render :json **=>** EmploymentStatusHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Display the EmploymentStatus form, with existing record to edit.*

*# GET /empstatus/edit/1*

*# GET /empstatus/edit/1.json*

**def** **edit**

    @empstatus **=** EmploymentStatus**.**find(params**[**:id**]**)

    @form\_id **=** 'edit-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @empstatus }

**end**

**end**

*# Update EmploymentStatus record.*

*# POST /empstatus/update/1*

**def** **update**

    o **=** EmploymentStatus**.**find(params**[**:id**]**)

**if** o**.**update\_attributes(:name **=>** params**[**:name**]**)

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Employment Status was successfully updated.' }

**else**

      render :json **=>** EmploymentStatusHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Delete a list of EmploymentStatus records.*

*# POST /empstatus/delete*

**def** **destroy**

    keyword **=** params**[**:keyword**].**blank? ? '' : params**[**:keyword**]**

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    ids **=** params**[**:id**]**

    EmploymentStatus**.**delete\_all(:id **=>** ids)

    itemscount **=** EmploymentStatusHelper**.**item\_message(keyword, pgnum, pgsize)

    render :json **=>** { :success **=>** 1,

                      :itemscount **=>** itemscount,

                      :message **=>** %Q{#{ids**.**size} Employment Status(es) was successfully

deleted.} }

**end**

**end**

## app/controllers/admin/hourly\_payroll\_chart\_controller.rb

*# This controller serves incoming requests to display Hourly Payroll Chart.*

**class** **Admin::**HourlyPayrollChartController **<** Admin::AdminController

*# Display the main page.*

*# GET /hourly/chart*

**def** **index**

    @month\_hash **=** month\_options

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:chart**]** }

**end**

**end**

*# Get the hourly payroll json data to be populated to the chart.*

*# GET /hourly/chart/data*

**def** **data**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    month **=** params**[**:month**].**blank? ? '0' : params**[**:month**]**

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    title **=** 'Hourly Payroll'

    yaxis **=** 'Total Amount (RM)'

    months **=** I18n**.**t('date.month\_names')

    o **=** Array**.**new(12) { **|**x**|** **[**months**[**x **+** 1**]**, 0**]** }

    b **=** Array**.**new(12) { **|**x**|** 0 }

    categories **=** Array**.**new(12) { **|**x**|** months**[**x **+** 1**][**0**.**.2**]** }

**if** staff\_id**.**present?

      liststaff **=** **[**staff\_id**]**

**else**

      list **=** PayRate**.**select('distinct(staff\_id)')**.**all

      liststaff **=** list**.**collect { **|**x**|** x**.**staff\_id }

**end**

**if** year **!=** 0

      listyear **=** **[**year**]**

      title **=** "Hourly Payroll for #{year}"

**else**

      list **=** PayRate**.**select('distinct(year)')**.**all

      listyear **=** list**.**collect { **|**x**|** x**.**year }

**end**

**if** month **!=** '0'

      listmonth **=** month**.**collect { **|**x**|** x**.**to\_i }

**else**

      list **=** PayRate**.**select('distinct(month)')**.**all

      listmonth **=** list**.**collect { **|**x**|** x**.**month }

**end**

    listyear**.**each **do** **|**y**|**

      listmonth**.**each **do** **|**m**|**

        liststaff**.**each **do** **|**s**|**

          filters **=** { :year **=>** y, :month **=>** m, :staff\_id **=>** s }

          total\_hours **=** AttendanceHelper**.**get\_total\_hours(filters)

          rate **=** PayRateHelper**.**get\_pay\_rate(filters)

          v **=** total\_hours **\*** rate

          o**[**m **-** 1**][**1**]** **=** v

          b**[**m **-** 1**]** **+=** v

**end**

**end**

**end**

    c **=** b**.**collect { **|**x**|** x**.**round(2) }

    @data **=** { :pie **=>** o, :column **=>** { :data **=>** c,

                                      :categories **=>** categories,

                                      :yaxis **=>** yaxis },

                                      :title **=>** title }

    render :json **=>** @data

**end**

**end**

## app/controllers/admin/job\_category\_controller.rb

*# This controller serves incoming requests to display out the JobCategory records.*

**class** **Admin::**JobCategoryController **<** Admin::AdminController

*# List all records.*

*# GET /jobcat*

*# GET /jobcat.json*

**def** **index**

    @data **=** JobCategoryHelper**.**get\_all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /jobcat/list*

*# GET /jobcat/list.json*

**def** **list**

    keyword **=** params**[**:keyword**].**blank? ? '' : params**[**:keyword**]**

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? JobCategoryHelper::DEFAULT\_SORT\_COLUMN :

      params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? JobCategoryHelper::DEFAULT\_SORT\_DIR :

      params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

**if** keyword**.**blank?

      @data **=** JobCategoryHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** JobCategoryHelper**.**get\_filter\_by(keyword, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the JobCategory form.*

*# GET /jobcat/new*

*# GET /jobcat/new.json*

**def** **new**

    @jobcat **=** JobCategory**.**new

    @form\_id **=** 'add-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @jobcat }

**end**

**end**

*# Create new JobCategory record.*

*# POST /jobcat/create*

**def** **create**

    o **=** JobCategory**.**new(:name **=>** params**[**:name**]**)

**if** o**.**save

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Job Category was successfully added.' }

**else**

      render :json **=>** JobCategoryHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Display the JobCategory form, with existing record to edit.*

*# GET /jobcat/edit/1*

*# GET /jobcat/edit/1.json*

**def** **edit**

    @jobcat **=** JobCategory**.**find(params**[**:id**]**)

    @form\_id **=** 'edit-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @jobcat }

**end**

**end**

*# Update JobCategory record.*

*# POST /jobcat/update/1*

**def** **update**

    o **=** JobCategory**.**find(params**[**:id**]**)

**if** o**.**update\_attributes(:name **=>** params**[**:name**]**)

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Job Category was successfully updated.' }

**else**

      render :json **=>** JobCategoryHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Delete a list of JobCategory records.*

*# POST /jobcat/delete*

**def** **destroy**

    keyword **=** params**[**:keyword**].**blank? ? '' : params**[**:keyword**]**

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    ids **=** params**[**:id**]**

    JobCategory**.**delete\_all(:id **=>** ids)

    itemscount **=** JobCategoryHelper**.**item\_message(keyword, pgnum, pgsize)

    render :json **=>** { :success **=>** 1,

                      :itemscount **=>** itemscount,

                      :message **=>** %Q{#{ids**.**size} Job Categori(es) was successfully

deleted.} }

**end**

**end**

## app/controllers/admin/overtime\_chart\_controller.rb

*# This controller serves incoming requests to display Employee Overtime Chart.*

**class** **Admin::**OvertimeChartController **<** Admin::AdminController

*# Display the main page.*

*# GET /overtime/chart*

**def** **index**

    @month\_hash **=** month\_options

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:chart**]** }

**end**

**end**

*# Get the overtime json data to be populated to the chart.*

*# GET /overtime/chart/data*

**def** **data**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    month **=** params**[**:month**].**blank? ? '0' : params**[**:month**]**

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    criteria **=** Attendance

    title **=** 'Overtime'

    yaxis **=** 'Duration (hours)'

**if** staff\_id**.**present?

      criteria **=** criteria**.**where(:staff\_id **=>** staff\_id)

**end**

**if** year **!=** 0

      criteria **=** criteria**.**where('year(work\_date) = ?', year)

      title **=** "Overtime for Year #{year}"

**end**

**if** month **!=** '0'

      criteria **=** criteria**.**where('month(work\_date) in (?)', month)

**end**

    list **=** criteria**.**order(:work\_date)**.**all

    months **=** I18n**.**t('date.month\_names')

    b **=** Array**.**new(12) { **|**x**|** 0 }

    categories **=** Array**.**new(12) { **|**x**|** months**[**x **+** 1**][**0**.**.2**]** }

    list**.**each **do** **|**o**|**

      to **=** ApplicationHelper**.**localtime(o**.**time\_out)

      v **=** Time**.**new(to**.**year, to**.**month, to**.**day, 18, 0, 0, '+08:00')

      x **=** (to **-** v) **/** 3600**.**0

      m **=** o**.**work\_date**.**month

**if** x **>** 0

        b**[**m **-** 1**]** **+=** x

**end**

**end**

    c **=** b**.**collect { **|**x**|** x**.**round(2) }

    @data **=** { :data **=>** c, :categories **=>** categories, :title **=>** title, :yaxis **=>** yaxis }

    render :json **=>** @data

**end**

**end**

## app/controllers/admin/overtime\_rate\_controller.rb

*# This controller serves incoming requests to display out the OvertimeRate records.*

**class** **Admin::**OvertimeRateController **<** Admin::AdminController

*# GET /overtimerate*

*# GET /overtimerate.json*

**def** **index**

    @data **=** OvertimeRateHelper**.**get\_all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /overtimerate/list*

*# GET /overtimerate/list.json*

**def** **list**

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? OvertimeRateHelper::DEFAULT\_SORT\_COLUMN :

      params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? OvertimeRateHelper::DEFAULT\_SORT\_DIR :

      params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

    filters **=** { :year **=>** year }

**if** year **==** 0

      @data **=** OvertimeRateHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** OvertimeRateHelper**.**get\_filter\_by(filters, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the OvertimeRate form.*

*# GET /overtimerate/new*

*# GET /overtimerate/new.json*

**def** **new**

    @rate **=** OvertimeRate**.**new

    @form\_id **=** 'add-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @rate }

**end**

**end**

*# Create new OvertimeRate record.*

*# POST /overtimerate/create*

**def** **create**

    o **=** OvertimeRate**.**new(:duration **=>** params**[**:duration**]**, :year **=>** params**[**:year**]**,

                         :pay\_rate **=>** params**[**:pay\_rate**]**)

**if** o**.**save

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Overtime Rate successfully added.' }

**else**

      render :json **=>** OvertimeRateHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Display the OvertimeRate form, with existing record to edit.*

*# GET /overtimerate/edit/1*

*# GET /overtimerate/edit/1.json*

**def** **edit**

    @rate **=** OvertimeRate**.**find(params**[**:id**]**)

    @form\_id **=** 'edit-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @rate }

**end**

**end**

*# Update OvertimeRate record.*

*# POST /overtimerate/update*

**def** **update**

    o **=** OvertimeRate**.**find(params**[**:id**]**)

**if** o**.**update\_attributes(:duration **=>** params**[**:duration**]**, :year **=>** params**[**:year**]**,

                           :pay\_rate **=>** params**[**:pay\_rate**]**)

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Pay Rate was successfully updated.' }

**else**

      render :json **=>** OvertimeRateHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Delete a list of OvertimeRate records.*

*# POST /overtimerate/delete*

**def** **destroy**

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    ids **=** params**[**:id**]**

    OvertimeRate**.**delete\_all(:id **=>** ids)

    filters **=** { :year **=>** year }

**if** year **==** 0

      itemscount **=** OvertimeRateHelper**.**item\_message(**nil**, pgnum, pgsize)

**else**

      itemscount **=** OvertimeRateHelper**.**item\_message(filters, pgnum, pgsize)

**end**

    render :json **=>** { :success **=>** 1,

                      :itemscount **=>** itemscount,

                      :message **=>** %Q{#{ids**.**size} Overtime rate(s) was successfully

deleted.} }

**end**

**end**

## app/controllers/admin/pay\_rate\_controller.rb

require 'securerandom'

*# This controller serves incoming requests to display out the PayRate records.*

**class** **Admin::**PayRateController **<** Admin::AdminController

*# GET /payrate*

*# GET /payrate.json*

**def** **index**

    @data **=** PayRateHelper**.**get\_all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /payrate/list*

*# GET /payrate/list.json*

**def** **list**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    month **=** params**[**:month**].**blank? ? 0 : params**[**:month**].**to\_i

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? PayRateHelper::DEFAULT\_SORT\_COLUMN :

      params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? PayRateHelper::DEFAULT\_SORT\_DIR : params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

    filters **=** { :staff\_id **=>** staff\_id,

                :month **=>** month,

                :year **=>** year }

**if** staff\_id**.**blank? **&&** month **==** 0 **&&** year **==** 0

      @data **=** PayRateHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** PayRateHelper**.**get\_filter\_by(filters, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the PayRate form.*

*# GET /payrate/new*

*# GET /payrate/new.json*

**def** **new**

    @payrate **=** PayRate**.**new

    @form\_id **=** 'add-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @payrate }

**end**

**end**

*# Create new PayRate record.*

*# POST /payrate/create*

**def** **create**

    o **=** PayRate**.**new(:id **=>** SecureRandom**.**uuid, :staff\_id **=>** params**[**:staff\_id**]**,

                    :month **=>** params**[**:month**]**, :year **=>** params**[**:year**]**,

                    :hourly\_pay\_rate **=>** params**[**:pay\_rate**]**)

**if** o**.**save

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Pay Rate successfully added.' }

**else**

      render :json **=>** PayRateHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Display the PayRate form, with existing record to edit.*

*# GET /payrate/edit/1*

*# GET /payrate/edit/1.json*

**def** **edit**

    @payrate **=** PayRate**.**find(params**[**:id**]**)

    @form\_id **=** 'edit-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @payrate }

**end**

**end**

*# Update PayRate record.*

*# POST /payrate/update*

**def** **update**

    o **=** PayRate**.**find(params**[**:id**]**)

**if** o**.**update\_attributes(:staff\_id **=>** params**[**:staff\_id**]**, :month **=>** params**[**:month**]**,

                           :year **=>** params**[**:year**]**, :hourly\_pay\_rate **=>** params**[**:pay\_rate**]**)

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Pay Rate was successfully updated.' }

**else**

      render :json **=>** PayRateHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Delete a list of PayRate records.*

*# POST /payrate/delete*

**def** **destroy**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    month **=** params**[**:month**].**blank? ? 0 : params**[**:month**].**to\_i

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    status **=** params**[**:status**].**blank? ? 0 : params**[**:status**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    ids **=** params**[**:id**]**

    PayRate**.**delete\_all(:id **=>** ids)

    filters **=** { :staff\_id **=>** staff\_id,

                :month **=>** month,

                :year **=>** year }

**if** staff\_id**.**blank? **&&** month **==** 0 **&&** year **==** 0

      itemscount **=** PayRateHelper**.**item\_message(**nil**, pgnum, pgsize)

**else**

      itemscount **=** PayRateHelper**.**item\_message(filters, pgnum, pgsize)

**end**

    render :json **=>** { :success **=>** 1,

                      :itemscount **=>** itemscount,

                      :message **=>** "#{ids**.**size} pay rate(s) was successfully deleted." }

**end**

**end**

## app/controllers/admin/payslip\_controller.rb

*# This controller serves incoming requests to display out the Employee records for pay*

*# slip generation.*

**class** **Admin::**PayslipController **<** Admin::AdminController

*# List all records.*

*# GET /payslip*

*# GET /payslip.json*

**def** **index**

    @data **=** EmployeeHelper**.**get\_all

    @employmentstatus **=** EmploymentStatus**.**order(:name)**.**all

    @designation **=** Designation**.**order(:title)**.**all

    @dept **=** Department**.**order(:name)**.**all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /payslip/list*

*# GET /payslip/list.json*

**def** **list**

    employee **=** params**[**:employee**].**blank? ? '' : params**[**:employee**]**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    employment\_status **=** params**[**:employment\_status**].**blank? ? 0 :

      params**[**:employment\_status**].**to\_i

    designation **=** params**[**:designation**].**blank? ? 0 : params**[**:designation**].**to\_i

    dept **=** params**[**:dept**].**blank? ? 0 : params**[**:dept**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? EmployeeHelper::DEFAULT\_SORT\_COLUMN :

      params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? EmployeeHelper::DEFAULT\_SORT\_DIR : params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

    filters **=** { :employee **=>** employee,

                :staff\_id **=>** staff\_id,

                :employment\_status **=>** employment\_status,

                :designation **=>** designation,

                :dept **=>** dept }

**if** employee**.**blank? **&&** staff\_id**.**blank? **&&** employment\_status **==** 0 **&&** designation **==** 0 **&&**

      dept **==** 0

      @data **=** EmployeeHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** EmployeeHelper**.**get\_filter\_by(filters, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the pay slip information.*

*# GET /payslip/slip/1/1/2012*

*# GET /payslip/slip/1/1/2012.json*

**def** **payslip**

    @employee **=** Employee**.**find(params**[**:id**]**)

    @employee\_salary **=** @employee**.**employee\_salary

    \_month **=** params**[**:month**].**to\_i

    month **=** month\_name(params**[**:month**].**to\_i)

    year **=** params**[**:year**].**blank? ? Time**.**now**.**year : params**[**:year**].**to\_i

    @period **=** "#{month}-#{year}"

**if** @employee\_salary**.**blank?

      @total\_earnings **=** 0

      @total\_deduct **=** 0

      @nett\_salary **=** 0

      @employee\_salary **=** EmployeeSalary**.**new

      respond\_to **do** **|**fmt**|**

        fmt**.**html { render 'payslip\_monthly' }

        fmt**.**json { render :json **=>** **[**@employee, @total\_earnings, @total\_deduct,

                                    @nett\_salary**]** }

**end**

**else**

*# checks whether the salary type is monthly*

**if** @employee\_salary**.**pay\_type **==** 1

        filters **=** { :year **=>** year, :month **=>** \_month, :staff\_id **=>** @employee**.**staff\_id }

        @total\_overtime **=** PayslipHelper**.**total\_overtime(filters)

        @total\_overtime\_earnings **=** PayslipHelper**.**total\_overtime\_earnings(filters,

          @total\_overtime)

        @adjustment **=** SalaryAdjustmentHelper**.**get\_salary\_adjustment(filters)

        @total\_earnings **=** PayslipHelper**.**total\_earnings(@employee\_salary, @adjustment,

          @total\_overtime\_earnings)

        @total\_deduct **=** PayslipHelper**.**total\_deductions(@employee\_salary)

        @nett\_salary **=** PayslipHelper**.**nett\_salary(@total\_earnings, @total\_deduct)

        @basic\_pay **=** @employee\_salary**.**salary **+** @adjustment

        respond\_to **do** **|**fmt**|**

          fmt**.**html { render 'payslip\_monthly' }

          fmt**.**json { render :json **=>** **[**@employee, @total\_earnings, @total\_deduct,

                                      @nett\_salary, @total\_overtime,

                                      @total\_overtime\_earnings, @adjustment, @basic\_pay**]** }

**end**

*# hourly type*

**else**

        filters **=** { :year **=>** year, :month **=>** \_month, :staff\_id **=>** @employee**.**staff\_id }

        @total\_earnings, @total\_hours, @hourly\_pay\_rate **=**

          PayslipHelper**.**total\_earnings\_hourly(@employee\_salary, filters)

        @total\_deduct **=** PayslipHelper**.**total\_deductions(@employee\_salary)

        @nett\_salary **=** PayslipHelper**.**nett\_salary\_hourly(@total\_earnings, @total\_deduct)

        respond\_to **do** **|**fmt**|**

          fmt**.**html { render 'payslip\_hourly' }

          fmt**.**json { render :json **=>** **[**@employee, @total\_earnings, @total\_deduct,

                                      @nett\_salary, @total\_hours, @hourly\_pay\_rate**]** }

**end**

**end**

**end**

**end**

**end**

## app/controllers/admin/salary\_adjustment\_controller.rb

require 'securerandom'

*# This controller serves incoming requests to display out the SalaryAdjustment records.*

**class** **Admin::**SalaryAdjustmentController **<** Admin::AdminController

*# GET /salaryadj*

*# GET /salaryadj.json*

**def** **index**

    @data **=** SalaryAdjustmentHelper**.**get\_all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /salaryadj/list*

*# GET /salaryadj/list.json*

**def** **list**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    month **=** params**[**:month**].**blank? ? 0 : params**[**:month**].**to\_i

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? SalaryAdjustmentHelper::DEFAULT\_SORT\_COLUMN :

      params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? SalaryAdjustmentHelper::DEFAULT\_SORT\_DIR :

      params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

    filters **=** { :staff\_id **=>** staff\_id,

                :month **=>** month,

                :year **=>** year }

**if** staff\_id**.**blank? **&&** month **==** 0 **&&** year **==** 0

      @data **=** SalaryAdjustmentHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** SalaryAdjustmentHelper**.**get\_filter\_by(filters, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the SalaryAdjustment form.*

*# GET /salaryadj/new*

*# GET /salaryadj/new.json*

**def** **new**

    @adj **=** SalaryAdjustment**.**new

    @form\_id **=** 'add-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @adj }

**end**

**end**

*# Create new SalaryAdjustment record.*

*# POST /salaryadj/create*

**def** **create**

    staff\_id **=** params**[**:staff\_id**]**

    o **=** SalaryAdjustment**.**new(:id **=>** SecureRandom**.**uuid, :staff\_id **=>** params**[**:staff\_id**]**,

                             :inc **=>** params**[**:inc**]**, :month **=>** params**[**:month**]**,

                             :year **=>** params**[**:year**]**)

**if** o**.**save

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Salary Adjustment successfully added.' }

**else**

      render :json **=>** SalaryAdjustmentHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Display the SalaryAdjustment form, with existing record to edit.*

*# GET /salaryadj/edit/1*

*# GET /salaryadj/edit/1.json*

**def** **edit**

    @adj **=** SalaryAdjustment**.**find(params**[**:id**]**)

    @form\_id **=** 'edit-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @adj }

**end**

**end**

*# Update SalaryAdjustment record.*

*# POST /salaryadj/update*

**def** **update**

    o **=** SalaryAdjustment**.**find(params**[**:id**]**)

**if** o**.**update\_attributes(:staff\_id **=>** params**[**:staff\_id**]**, :inc **=>** params**[**:inc**]**,

                           :month **=>** params**[**:month**]**, :year **=>** params**[**:year**]**)

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'Pay Rate was successfully updated.' }

**else**

      render :json **=>** PayRateHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Delete a list of SalaryAdjustment records.*

*# POST /salaryadj/delete*

**def** **destroy**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    month **=** params**[**:month**].**blank? ? 0 : params**[**:month**].**to\_i

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    status **=** params**[**:status**].**blank? ? 0 : params**[**:status**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    ids **=** params**[**:id**]**

    SalaryAdjustment**.**delete\_all(:id **=>** ids)

    filters **=** { :staff\_id **=>** staff\_id,

                :month **=>** month,

                :year **=>** year }

**if** staff\_id**.**blank? **&&** month **==** 0 **&&** year **==** 0

      itemscount **=** SalaryAdjustmentHelper**.**item\_message(**nil**, pgnum, pgsize)

**else**

      itemscount **=** SalaryAdjustmentHelper**.**item\_message(filters, pgnum, pgsize)

**end**

    render :json **=>** { :success **=>** 1,

                      :itemscount **=>** itemscount,

                      :message **=>** %Q{#{ids**.**size} salary adjustment(s) was successfully

deleted.} }

**end**

**end**

## app/controllers/admin/total\_work\_hours\_chart\_controller.rb

*# This controller serves incoming requests to display Employee Total Hours Worked Chart.*

**class** **Admin::**TotalWorkHoursChartController **<** Admin::AdminController

*# Display the main page.*

*# GET /workhours/chart*

**def** **index**

    @month\_hash **=** month\_options

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:chart**]** }

**end**

**end**

*# Get the total hours worked json data to be populated to the chart.*

*# GET /workhours/chart/data*

**def** **data**

    staff\_id **=** params**[**:staff\_id**].**blank? ? '' : params**[**:staff\_id**]**

    month **=** params**[**:month**].**blank? ? '0' : params**[**:month**]**

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    title **=** 'Total Hours Worked'

    yaxis **=** 'Duration (hours)'

    months **=** I18n**.**t('date.month\_names')

    b **=** Array**.**new(12) { **|**x**|** 0 }

    categories **=** Array**.**new(12) { **|**x**|** months**[**x **+** 1**][**0**.**.2**]** }

**if** staff\_id**.**present?

      liststaff **=** **[**staff\_id**]**

**else**

      list **=** PayRate**.**select('distinct(staff\_id)')**.**all

      liststaff **=** list**.**collect { **|**x**|** x**.**staff\_id }

**end**

**if** year **!=** 0

      listyear **=** **[**year**]**

      title **=** "Total Hours Worked for #{year}"

**else**

      list **=** PayRate**.**select('distinct(year)')**.**all

      listyear **=** list**.**collect { **|**x**|** x**.**year }

**end**

**if** month **!=** '0'

      listmonth **=** month**.**collect { **|**x**|** x**.**to\_i }

**else**

      list **=** PayRate**.**select('distinct(month)')**.**all

      listmonth **=** list**.**collect { **|**x**|** x**.**month }

**end**

    listyear**.**each **do** **|**y**|**

      listmonth**.**each **do** **|**m**|**

        liststaff**.**each **do** **|**s**|**

          filters **=** { :year **=>** y, :month **=>** m, :staff\_id **=>** s }

          total\_hours **=** AttendanceHelper**.**get\_total\_hours(filters)

          v **=** total\_hours

          b**[**m **-** 1**]** **+=** v

**end**

**end**

**end**

    c **=** b**.**collect { **|**x**|** x**.**round(2) }

    @data **=** { :data **=>** c, :categories **=>** categories, :title **=>** title, :yaxis **=>** yaxis }

    render :json **=>** @data

**end**

**end**

## app/controllers/admin/user\_controller.rb

require 'securerandom'

*# This controller serves incoming requests to display out the User records.*

**class** **Admin::**UserController **<** Admin::AdminController

*# List all records.*

*# GET /user*

*# GET /user.json*

**def** **index**

    @data **=** UserHelper**.**get\_all

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:list**]** }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# List records by filtering.*

*# GET /user/list*

*# GET /user/list.json*

**def** **list**

    username **=** params**[**:username**].**blank? ? '' : params**[**:username**]**

    role **=** params**[**:role**].**blank? ? 0 : params**[**:role**].**to\_i

    employee **=** params**[**:employee**].**blank? ? '' : params**[**:employee**]**

    status **=** params**[**:status**].**blank? ? 0 : params**[**:status**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    sortcolumn **=** params**[**:sortcolumn**].**blank? ? UserHelper::DEFAULT\_SORT\_COLUMN :

      params**[**:sortcolumn**]**

    sortdir **=** params**[**:sortdir**].**blank? ? UserHelper::DEFAULT\_SORT\_DIR : params**[**:sortdir**]**

    sort **=** ApplicationHelper::Sort**.**new(sortcolumn, sortdir)

    filters **=** { :username **=>** username,

                :role **=>** role,

                :employee **=>** employee,

                :status **=>** status }

**if** username**.**blank? **&&** role **==** 0 **&&** employee**.**blank? **&&** status **==** 0

      @data **=** UserHelper**.**get\_all(pgnum, pgsize, sort)

**else**

      @data **=** UserHelper**.**get\_filter\_by(filters, pgnum, pgsize, sort)

**end**

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'list' }

      fmt**.**json { render :json **=>** @data }

**end**

**end**

*# Display the User form.*

*# GET /user/new*

*# GET /user/new.json*

**def** **new**

    @user **=** User**.**new

    @form\_id **=** 'add-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @user }

**end**

**end**

*# Create new User record.*

*# POST /user/create*

**def** **create**

    status **=** params**[**:status**]**

    status **=** status **==** '1' ? **true** : **false**

    o **=** User**.**new(:id **=>** SecureRandom**.**uuid, :role **=>** params**[**:role**]**,

                 :username **=>** params**[**:username**]**, :status **=>** status,

                 :pwd **=>** params**[**:pwd**]**, :pwd\_confirmation **=>** params**[**:pwdconfirm**]**)

**if** o**.**save

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'User was successfully added.' }

**else**

      render :json **=>** UserHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Display the User form, with existing record to edit.*

*# GET /user/edit/1*

*# GET /user/edit/1.json*

**def** **edit**

    @user **=** User**.**find(params**[**:id**]**)

    @form\_id **=** 'edit-form'

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :partial **=>** 'form' }

      fmt**.**json { render :json **=>** @user }

**end**

**end**

*# Update User record.*

*# POST /user/update*

**def** **update**

    o **=** User**.**find(params**[**:id**]**)

    status **=** params**[**:status**]**

    status **=** status **==** '1' ? **true** : **false**

**if** o**.**update\_attributes(:role **=>** params**[**:role**]**, :username **=>** params**[**:username**]**,

                           :status **=>** status, :pwd **=>** params**[**:pwd**]**,

                           :pwd\_confirmation **=>** params**[**:pwdconfirm**]**)

      render :json **=>** { :success **=>** 1,

                        :message **=>** 'User was successfully updated.' }

**else**

      render :json **=>** UserHelper**.**get\_errors(o**.**errors)

**end**

**end**

*# Delete a list of User records.*

*# POST /user/delete*

**def** **destroy**

    username **=** params**[**:username**].**blank? ? '' : params**[**:username**]**

    role **=** params**[**:role**].**blank? ? 0 : params**[**:role**].**to\_i

    employee **=** params**[**:employee**].**blank? ? '' : params**[**:employee**]**

    status **=** params**[**:status**].**blank? ? 0 : params**[**:status**].**to\_i

    pgnum **=** params**[**:pgnum**].**blank? ? 1 : params**[**:pgnum**].**to\_i

    pgsize **=** params**[**:pgsize**].**blank? ? 0 : params**[**:pgsize**].**to\_i

    ids **=** params**[**:id**]**

    User**.**delete\_all(:id **=>** ids)

    filters **=** { :username **=>** username,

                :role **=>** role,

                :employee **=>** employee,

                :status **=>** status }

**if** username**.**blank? **&&** role **==** 0 **&&** employee**.**blank? **&&** status **==** 0

      itemscount **=** UserHelper**.**item\_message(**nil**, pgnum, pgsize)

**else**

      itemscount **=** UserHelper**.**item\_message(filters, pgnum, pgsize)

**end**

    render :json **=>** { :success **=>** 1,

                      :itemscount **=>** itemscount,

                      :message **=>** "#{ids**.**size} user(s) was successfully deleted." }

**end**

**end**

## app/controllers/user/contact\_controller.rb

*# This controller serves incoming requests to display out the EmployeeContact record.*

**class** **User::**ContactController **<** User::UserController

*# Display the contact.*

*# GET /contact*

*# GET /contact.json*

**def** **index**

    id **=** get\_employee\_id

    @employee\_contact **=** EmployeeContactHelper**.**find(id)

    respond\_to **do** **|**fmt**|**

      fmt**.**html

      fmt**.**json { render :json **=>** @employee\_contact }

**end**

**end**

*# Update the contact.*

*# POST /contact/update*

**def** **update**

    id **=** get\_employee\_id

    oc **=** EmployeeContactHelper**.**find(id)

    oc\_new **=** **false**

**if** oc**.**id**.**blank?

      o **=** Employee**.**find(id)

      oc **=** EmployeeContactHelper**.**employee\_contact\_obj(o, params)

      oc\_new **=** **true**

**end**

**if** oc\_new

**if** oc**.**save

        render :json **=>** { :success **=>** 1,

                          :message **=>** 'Contact Details was successfully updated.' }

**else**

        render :json **=>** EmployeeContactHelper**.**get\_errors(oc**.**errors)

**end**

**else**

**if** EmployeeContactHelper**.**update\_obj(oc, params)

        render :json **=>** { :success **=>** 1,

                          :message **=>** 'Contact Details was successfully updated.' }

**else**

        render :json **=>** EmployeeContactHelper**.**get\_errors(oc**.**errors)

**end**

**end**

**end**

**end**

## app/controllers/user/hourly\_payroll\_chart\_controller.rb

*# This controller serves incoming requests to display Hourly Payroll Chart.*

**class** **User::**HourlyPayrollChartController **<** User::UserController

*# Display the main page.*

*# GET /hourly/chart*

**def** **index**

    @month\_hash **=** month\_options

    respond\_to **do** **|**fmt**|**

      fmt**.**html { render :layout **=>** LAYOUT**[**:chart**]** }

**end**

**end**

*# Get the hourly payroll json data to be populated to the chart.*

*# POST /hourly/chart/data*

**def** **data**

    month **=** params**[**:month**].**blank? ? '0' : params**[**:month**]**

    year **=** params**[**:year**].**blank? ? 0 : params**[**:year**].**to\_i

    staff\_id **=** get\_staff\_id

    title **=** 'Hourly Payroll'

    yaxis **=** 'Total Amount (RM)'

    months **=** I18n**.**t('date.month\_names')

    o **=** Array**.**new(12) { **|**x**|** **[**months**[**x **+** 1**]**, 0**]** }

    b **=** Array**.**new(12) { **|**x**|** 0 }

    categories **=** Array**.**new(12) { **|**x**|** months**[**x **+** 1**][**0**.**.2**]** }

**if** year **!=** 0

      listyear **=** **[**year**]**

      title **=** "Hourly Payroll for #{year}"

**else**

      list **=** PayRate**.**select('distinct(year)')**.**all

      listyear **=** list**.**collect { **|**x**|** x**.**year }

**end**

**if** month **!=** '0'

      listmonth **=** month**.**collect { **|**x**|** x**.**to\_i }

**else**

      list **=** PayRate**.**select('distinct(month)')**.**all

      listmonth **=** list**.**collect { **|**x**|** x**.**month }

**end**

    listyear**.**each **do** **|**y**|**

      listmonth**.**each **do** **|**m**|**

        filters **=** { :year **=>** y, :month **=>** m, :staff\_id **=>** staff\_id }

        total\_hours **=** AttendanceHelper**.**get\_total\_hours(filters)

        rate **=** PayRateHelper**.**get\_pay\_rate(filters)

        v **=** total\_hours **\*** rate

        o**[**m **-** 1**][**1**]** **=** v

        b**[**m **-** 1**]** **+=** v

**end**

**end**

    c **=** b**.**collect { **|**x**|** x**.**round(2) }

    @data **=** { :pie **=>** o, :column **=>** { :data **=>** c,

                                      :categories **=>** categories,

                                      :yaxis **=>** yaxis },

                                      :title **=>** title }

    render :json **=>** @data

**end**

**end**