

INFORMATION

STUDENT TO COMPLETE IN FULL (To be fastened securely to the front of all coursework. It is your responsibility to also attach any additional items. E.g. CD, etc.)

Please tick if you are:

Foundation Diploma Undergraduate Postgraduate Others

Please Use Block Capitals

1	Module Code & Title	Advanced Software Engineering
2	Course	MSC IN SOFTWARE ENGINEERING
3	Name of Lecturer	Mr. Riyaz Ahamed
4	Submission Date	25th April, 2016

Please tick if this assignment is re-submission

Please ensure that you have signed the declaration below before submitting your assignment. For group assignments, all students in the group must sign the declaration.

I/We understand that:

This assessment item is entirely **my/our** own original work, except where **I/we** have acknowledged use of source material [such as books, journal articles, other published material, the Internet, and the work of other student/s or any other person/s]. This assessment item has not been submitted for assessment for academic credits in this, or any other course at FTMS College or elsewhere.

I/We understand that:

The assessor of this assessment item may, for the purpose of assessing this item, reproduce this assessment item and provide a copy to another member of the college. The assessor may communicate a copy of this assessment item to a **plagiarism** checking service (which may retain a copy off the assessment item on its database for the purpose of future plagiarism checking).

STUDENT NAME	Student ID	Signature	Date
AMIR ANSARI	77172892		
EMMANCIPATE TAKUNDA MUSEMWYA	77172238		
UADIALE HENRY AKHERE	77172169		

See information about plagiarism & academic misconduct from the Student Handbook

First marker's comment

Second marker's comment

Assessment Board/External Examiner:	Total Marks	First Marker	Second Marker	Agreed Marks
	Signature&Date			

TABLE OF CONTENT

Contents

1. INTRODUCTION.....	6
1.1 AIM	6
1.2 OBJECTIVE.....	7
1.3 REQUIREMENTS.....	7
<i>Functional Requirements</i>	<i>7</i>
<i>Non- Functional Requirements</i>	<i>8</i>
1.4 FEASIBILITY STUDY.....	11
1.5 SCOPE	11
1.5 PLATFORM SPECIFICATION.....	12
<i>1.5.1 Tools Used</i>	<i>12</i>
<i>1.5.2 Languages Used</i>	<i>13</i>
1.6 SECURITY ISSUE.....	14
<i>1.6.1 Data Integrity & Validation Checks.....</i>	<i>14</i>
1.7 VALUE CONSTRAINTS AND RANGES	15
1.8 VALIDATION CHECKS	16
2.0 SYSTEM ANALYSIS AND DESIGN.....	17
2.1 SYSTEM DESIGN.....	17
<i>Figure 1.1: Belong to system design.....</i>	<i>18</i>
2.2 DATA FLOW DIAGRAMS.....	18
<i>LOGIN PROCESS.....</i>	<i>19</i>
<i>Fig 1.2 Login Process</i>	<i>19</i>
<i>LOGOUT PROCESS.....</i>	<i>19</i>
<i>Fig 1.3 Logout Process</i>	<i>19</i>
2.3 USE CASE DIAGRAM	20
<i>LIST OF USE CASES.....</i>	<i>20</i>
2.4 CLASS DIAGRAM.....	23
<i>Fig 1.5 Class Diagram.....</i>	<i>23</i>
3.0 IMPLEMENTATION.....	24
3.1 ADMIN ACCOUNT	24
<input type="checkbox"/> ADMIN MENU	24
<input type="checkbox"/> ADD EMPLOYEE	24

<i>3.1.1 Admin Login</i>	24
<i>Fig 3.1.1 admin login.....</i>	25
<i>3.1.2 Home menu Admin.....</i>	25
<i>Fig 3.1.2 Admin home page</i>	25
<i>3.1.3 Add Employee</i>	25
<i>Fig 3.1.3 information about the employee</i>	26
<i>3.1.4 Update Employee</i>	26
<i>Fig 3.1.4 Update Data</i>	27
<i>3.1.5 Search Record.....</i>	27
<i>Fig 3.1.5 Searching employee.....</i>	27
<i>3.1.6 Delete Record</i>	27
<i>Fig 3.1.6 Delete employee record.....</i>	28
<i>3.1.7 Calculate Allowance.....</i>	28
<i>Fig 3.1.7 Allowance Calculation</i>	28
<i>3.1.8 Calculate Deduction.....</i>	29
<i>Fig 3.1.8 Deduction Calculation</i>	29
<i>3.1.9 Make Payment</i>	29
<i>Fig 3.1.9 Make payment of particular employee</i>	30
<i>3.1.10 Expenditure Report.....</i>	30
<i>Fig 3.1.10 Expenditure report</i>	30
<i>3.1.11 Print Report.....</i>	30
<i>Fig 3.1.21 Print report.....</i>	31
3.2 MANAGER ACCOUNT	31
<i>Fig 3.2 manager Account.....</i>	32
4.0 TESTING.....	32
4.1 UNIT TESTING.....	32
4.2 MANUAL TESTING	32
<i>4.2.1 LEGEND.....</i>	33
LOGIN ADMIN.....	34
LOGIN MANAGER.....	34
ADD EMPLOYEE	34
<i>4.3.1 Module Name: Admin Login</i>	34
<i>4.3.2 Module Name: Manager Login</i>	35
<i>4.3.3 Module Name: Add Employee</i>	35
<i>4.3.4 Module Name: Update Employee.....</i>	36
<i>4.3.5 Module Name: Search Records</i>	37

<i>4.3.6 Module Name: Delete Records.....</i>	38
<i>4.3.7 Module Name: Allowance</i>	38
<i>4.3.8 Module Name: Deduction.....</i>	39
<i>4.3.10 Module Name: Expenditure Report.....</i>	41
<i>4.311 Module Name: Expenditure Report by Month.....</i>	41
<i>4.3.12Module Name: Logout</i>	42
4.4 JUNIT TESTING	42
<i> 4.4.1 Admin Account.....</i>	42
<i> Fig 4.4.1 admin class</i>	43
<i> Fig 4.4.2 Test admin's username.....</i>	43
<i> Fig 4.4.3 Test password</i>	43
<i> Fig 4.4.4 test suit admin login test.....</i>	44
<i> 4.4.5 Manager Login.....</i>	44
<i> Fig 4.4.5 manager class.....</i>	45
<i> Fig 4.4.6 Test username of the manager</i>	45
<i> Fig 4.4.7 test password of the manager.....</i>	45
<i> Fig 4.4.8 Test suit of manager class</i>	46
<i> 4.5.1 Add Employee Class.....</i>	46
<i> Fig 4.5.1 Actual format of the value which is needed to fill in the text box.</i>	47
<i> Fig 4.5.2 test the array values which are entered by the admin.....</i>	47
<i> Fig 4.5.3 test blank value</i>	48
<i> Fig 4.5.3 test again after changing some value.....</i>	49
<i> Fig 4.5.4 wrong email id.....</i>	49
<i> Fig 4.5.5 test email id.....</i>	50
<i> 4.6.1Allowance Amount</i>	50
<i> Fig 4.6.1 Array values of the allowance</i>	51
<i> Fig 4.6.2 Comparing two values.....</i>	51
<i> Fig 4.6.3 Test allowance of the employee.....</i>	52
<i> 4.7.1 Deduction Amount.....</i>	52
<i> Fig 4.7.1 Array values of the deduction which have to perform operation.....</i>	53
<i> Fig 4.7.2 check the actual value and system generated value.....</i>	53
<i> Fig 4.7.3 check the actual deduction value and system generated value.....</i>	54
<i> 4.8.1 Expenditure Report By Month.....</i>	55
<i> Fig 4.8.1 Array value of the expenditure class for a specific month.....</i>	55
<i> Fig 4.8.2 test the a specific date</i>	55
<i> 4.9.1 Expenditure Report</i>	57
<i> Fig 4.9.1 Array values of the expenditure</i>	57
<i> Fig 4.9.2 test net salary.....</i>	57

<i>Fig 4.9.3 testing two values</i>	58
<i>4.10.1 Search Records</i>	59
<i>Fig 4.10.1 Array value of the employee.....</i>	59
<i>Fig 4.10.2</i> check the employee id 59	
<i>Fig 4.10.3 Test Employee Id exist or not</i>	60
<i>4.11.1 Update Employee</i>	61
<i>Fig 4.11.1 exist array values of the employee</i>	61
<i>4.11.3 testing before the updating records of the employee</i>	61
<i>Fig 4.11.4 testing after updating some value of the employees.....</i>	62
5.0 JAVADOC	63
<i>5.1 Generating Java doc.....</i>	63
<i>Fig 5.2 Generating Javadoc</i>	63
<i>5.3 Generated Java Doc</i>	64
<i>Fig 5.4 Generated java doc</i>	64
6.0 VERSIONS CONTROL	65
COMMITS IN GITHUB IMAGE SHOWING COMMITTED FILES IN GITHUB.....	65
6.1 README FILE IN GITHUB	66
7.0 DEBUGGING.....	66
7.1 BREAKPOINT.....	67
<i>Fig 7.2 Agile SDLC (tutorials point, 2016)</i>	70
7.3 COMPARING AGILE APPROACH WITH THE TRADITIONAL APPROACH	70
<i>7.3.1 Following are the Agile principles are:</i>	71
<input type="checkbox"/> <i>Individuals and interactions</i>	71
<input type="checkbox"/> <i>Working software.....</i>	71
<input type="checkbox"/> <i>Customer collaboration.....</i>	71
<input type="checkbox"/> <i>Responding to change.....</i>	71
8.0 CONCLUSION	72
CODING STANDARDS	73
COMMENTING	73
CODE PARAGRAPHS AND INDENTATION	74
FUNCTIONS AND VARIABLES NAMING	75
REFERENCE	76

1. Introduction

Information is one of the most valuable things of the current information society. Fast access to needed information is critical in business at the present time. Companies have to spend huge amount of money to store the information about the employee and do changed when it's needed. The payroll system is an account system which stores the information of the employees at a one place and only transaction related action will be occurred here and if there are any changes required in the future it can be easily updated. There are two accounts in IBM Payroll System and they have different permission to perform action.

First one is admin account which is the most important account in this payroll system and he can involve all the section of the employees and can perform some operation here. For example, hire the employee, delete the employee details, update salary etc. Manager account which is the second account in this system can access only few section of the employees. For example, expenditure report, print etc. Payroll management system will produce summary or salary report of all the employees or a specific employee at a specific date. The result of payroll accounting system is exact records to the costs connected with a wide range of pay, and other additional payments and remunerations to the workers. Payroll administration plays a key role in a business development. Any issues with the payrolls of an organization can prompt the workers disappointment that can have an immediate bearing on the proficiency of their administrations. A perfect payroll system influences each part of a business, from the spirit of workers to the sustainability and stability of the organization.

1.1 Aim

The goal of the IBM Payroll system is to develop an account system which can store all the information about the employees and can perform a quick action related to transaction and it will also keep the history of all the transaction which can be accessed quickly it will save much money and time during the paying salary to the employees. For example, Allowance, Deduction or other benefits will be added automatically and system will generate net salary.

1.2 Objective

The main objective of the project is to develop an “**IBM Payroll System**” which provides more security during every transaction of the company to the employees and only those users can see the data of the employees who have appropriate permission. The basic idea behind the project is to reduce time during the payment of the employees or wasting a time during searching the allowance of a specific employee so in this system thousands of employees can be searched within a minute. This is a separate section so it will not effect of the companies’ work so no more employee needed to work in this account so it will reduce the number of employees.

1.3 Requirements

Functional Requirements

Functional requirements explain a function of a system and its components. It describes the inputs the system will accept and the output it will produce. The table below shows the functional requirement of the system.

No.	Requirement Name	Description	Priority
1	Add New Employee	The system must allow the administrator to add new employees to the payroll system	H
2	Update Employee Details	The system must allow the administrator to update employee details	H
3	Log in	The system must allow users to log in with a unique username and password	H
4	Calculate Allowances	The system must be able to calculate total allowances and add them to the gross salary	H
5	Calculate Deductions	The system must be able to calculate total deductions and subtract	H

		them from the Payroll System	
6	Make Payment	The system must be able to make Payments to the employees	H
7	Generate Payslip	The system must be able to generate a payslip for an employee	H
8	Generate Overall Expenditure Report	The system must be able to calculate the overall company expenditure report	M
9	Generate Expenditure Report By Month	The system must be able to generate expenditure report by month	H
10	Print Reports	The system must be able to print generated reports	H
11	Logout User	The system must be able to logout or kill logged in user session.	H

Non- Functional Requirements

Non-functional requirement do not explain behavior but rather explain the ways that can be used to qualify the operations of the system. The table below shows the non-functional requirements for the online ordering system.

No.	Requirement Name	Description	Why needed
1	Availability	Hours of Operation- the system should be available 24/7 except maintaining times. Locations of Operation- The system will be available to everyone online but only allows access to people in a	-To serve customers at any time -deliver distance restrictions

		100mile radius from IBM offices	
2	Security	Must allow users to login with a unique user name and password and restrict some areas to people with authority to access them only like the administrator	Prevent fraud and manipulation of information
3	Performance	<p>Response time- application must load and refresh fast.</p> <p>Processing time- application must perform calculations fast.</p> <p>Query and report times- Application must load initial and subsequent loads fast</p>	Provide results fast to user
4	Capacity	<p>Throughput- the system should be able to handle over 100 transactions per hour</p> <p>Storage-the system should be able to store 300GB of data</p> <p>The system should have room to grow</p>	Amount of items the system can handle can be over capacitated
5	Reliability	<p>Mean time between failures – less than 4,000 hours per year. The system must be reliable.</p> <p>Mean time to recover- if down the system must take less than an hour to recover</p>	Customers need to access the system at all times.

6	Compatibility	<p>The system should be compatible with Shared applications- it should communicate with flash players and web browsers etc</p> <p>3rd party applications- it should live amicably with antivirus software</p> <p>Operating systems- it should be able to run on window 7 and above, and above, linux and mac OS.</p> <p>Platforms- it should work on different hardware platforms and mobile devices</p>	To make the system work on different situations.
7	Maintainability	The system must be easy to maintain, upgrade and grow	For development and error correction
8	Usability	Look and feel - the interface must be user friendly with the colors, text, space, keyboard shortcuts all welcoming for the user.	Easy and simple to use
9	Audit	System should allow auditing of some data elements like payment details	For security reasons.

1.4 Feasibility Study

This Payroll System is feasible in many ways which can be described from the following points.

- **Response Time:** The system has quick data access so the response time is very less.
- **Data Integrity:** This portal shall have a centralized data which will be accurate.
- **Reliability:** A system is reliable if it fulfills requirements and recognizes and prevents errors. This project is reliable for this way.
- **Communicativeness:** This system will be accessible through Internet.
- **Environment Factor:** The portal will save paper because of data files stores on hardware. This one decreases air pollution.
- **Compatibility:** As the project will be accessible through Internet, so it will be made to compatible with the standard hardware and software.
- **Social Objective:** In today's busy and last world where there is a chance to transfer payment in wrong account so it will prevent all the problems which can happen during the transaction.

1.5 Scope

This project is developed keeping in mind all the future facilities of Payroll system and it is the new trend of internet banking where company can also perform same operation like bankingsystem from their place. This system will help to employee and their salary will be transferred after every month in their account with some allowance or deduction without any mistaken because everything will be generated by the system. In this system there are many post of the employees so salary will be generated according to the post so it will save the time and

money of the company and only one accountant can perform all the operation in the transaction. This site is developed keeping all facilities of the Payroll System in mind but if in the future some changes required in this system so it can changed quickly because a lot of components developed for this project can be reused for future projects with a little change.

1.5 Platform Specification

Every project needs some tools and programming language that has to be decided in advance during the analysis. This project also uses quite a number of tools as well as languages which are explained below in detail. This project can work on both Windows and Linux platforms but I am developing and testing only on windows platform:

1.5.1 Tools Used

- **Java Development kit (JDK)** -> is a tool which is used to development the JAVA SE, JAVA ME, or JAVA program in a system etc.
- **Java Runtime Environment (JRE)** -> this application is used to run the java program on a system not for development. JDK is important for development so for this project both Software needs. This project will be required JDK 1.7 and JRE 7 and also it depends on the tomcat server.
- **Eclipse:** This software is used to implement the JAVA, JSP or any other programming languages. Web application eclipse IDE for Java EE Developers will be needed because this is a web application. Eclipse Mars.1 (4.5.1) is used to implement this project.
- **Tomcat:** This is server which is used to deploy the code and output can be seen on the browser. Tomcat 6.0 or above version is required for this project.

-
- **MySQL:** This is a database which is used to store the data about the client or another information at one place and can be accessed easily.
- **Web Browser:** There are two web browser to see the output first one internal which is embedded with eclipse and other is external browser (Google Chrome, Firefox etc.).

1.5.2 Languages Used

The project uses the **J2EE platform** and some another technologies will be used for a little work and some will be used throughout. The following is a complete list of all technologies and their use in the project in detail:

- **JAVA** – It is one of the most powerful Object Oriented Platform Independent Language. Java is used in most of the project in one form or another.
- **JSP** – Java Server Pages or JSP for short is a server-side technology that takes Java language with its inherent simplicity, and uses it to create highly interactive and flexible web applications. JSP will be used in the project to create most of the web pages that interact with the user.
- **JavaScript** – JavaScript is used for client- side scripting. It enables the web pages to have some programmatic functionality in the browser. JavaScript works with and can manipulate the HTML page in which it is embedded. JavaScript is used in the project to validate the data entered in the web pages by the user before it is send to the server.
- **Servlet** – A Servlet is a Java program that generates dynamic web content. They are written using the Java Servlet API and are managed by a Servlet container such as

Tomcat. The Servlet processes the user request, builds a response, and passes it to the container which it back to the user. I have used a Servlet for making the controller in the web based system. It processes the user requests and gives appropriate response according to them.

- **HTML** – HTML or Hyper Text markup Language is used creating web pages. HTML pages are static and do not interact with the user but can be made interactive by adding JSP elements them. Most of the web pages in the project are designed in HTML and after that JSP elements are added to them.
- **XML** – Extensible Markup Language or XML for short has become the de facto standard for data interchange on the Internet. We have not used it directly but a lot of configuration files of Apache Tomcat Server as well as Apache James E-mail server are written in this format.

1.6 Security Issue

Portal is secure both client side and server side no one can see the employees' details except admin and manager. Admin and manager have different types of accessibility but before they perform any operation with the employees, they should have some unique username and password. Care of security is taken for individual servers as well as the whole system. Steps are taken to counter three types of security issues which are as follows:

- ❖ Unauthorized access to database server.
- ❖ Unauthorized access to payroll system.
- ❖ Steps taken against hacking of system.

1.6.1 Data Integrity & Validation Checks

Data integrity is very important in any project because invalid data is of no use so various measures are taken for maintaining data integrity. In web based payroll system the most important data for the smooth functioning of the system is the data contained of the employee. There are two steps for maintaining data integrity:

❖ **Value Constraints and Ranges**

❖ **Validation Checks**

1.7 Value Constraints and Ranges

The first step for maintaining data integrity is to identify various value ranges of various attributes. This is done in the design phase in the payroll system admin can make changes to the data constraints are defined on the data entered by the admin which are as follows:

<u>Attribute</u>	<u>Value constraint and range</u>
username	User id contain a-z characters.
password	Contains password a-z character and 0-9 number
Employee ID:	Employee ID can contain a-z and 0-9 characters.
First Name	Only can contains a-z character
Last Name:	Can contains only a-z character
Gender	Can contain only Male or Female
Date Of Birth:	Must be in date format MM/DD/YYYY
Email Address:	An email address.
Phone Number:	Can contains only numeric value 0-9
Home Address:	Can contain any character
Designation:	Can select any post
Department:	Can select any department

Date Joined	Must be in date format MM/DD/YYYY
-------------	-----------------------------------

1.8 Validation Checks

Validation checks are the second step in maintaining data integrity. These are implemented in the coding phase of the project development. When the admin enters data validation checks are performed over it before using it. If it is found to violate its value range then an appropriate message is shown to the admin.

The project developed also performs a lot of validations on the data entered. Some of them are given below:

<u>Attribute</u>	<u>Validation Checks Performed</u>
username	Is thoroughly checked for value constraints before checking it with the server.
password	Checked only at login time to be more than six characters.
Employee ID:	Only administrator can give any employee id
First Name	Checked for blank entry.
Last Name:	Checked for blank entry.
Gender	Select any value
Date Of Birth:	Drop down list for entering the value.
Email Address:	Checked for blank entry.
Phone Number:	Checked for blank entry.
Home Address:	Checked for blank entry.
Designation:	Drop down list for entering the value.
Department:	Drop down list for entering the value.
Date Joined	Drop down list for entering the value.

2.0 System Analysis and Design

- **Output:** The project is created for report and printouts as output of the portal's all static or dynamic web pages.
- **Inputs:** Input of the project through form of Java Servlets under the following points.
- **Accuracy:** Data will be always accurate because data enter through textbox and combo box with fixed length and data type also. Wrong data not accepted by the project because each field uses data validation.
- **Timeliness:** Data is access very fast because the database is MySQL and not complex calculation present in the project.
- **Proper format:** For proper format input and output HTML, FrontPage and Java Servlet are used.

2.1 System Design

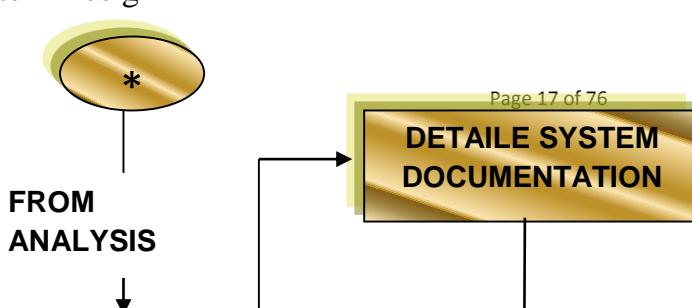


Figure 1.1: Belong to system design

2.2 Data Flow Diagrams

A data flow diagram is a graphical representation that depicts information flow and the transformations that are applied as data move from input to output. The DFD can be partitioned into levels that represent increasing information flow and functional detail.

LOGIN PROCESS

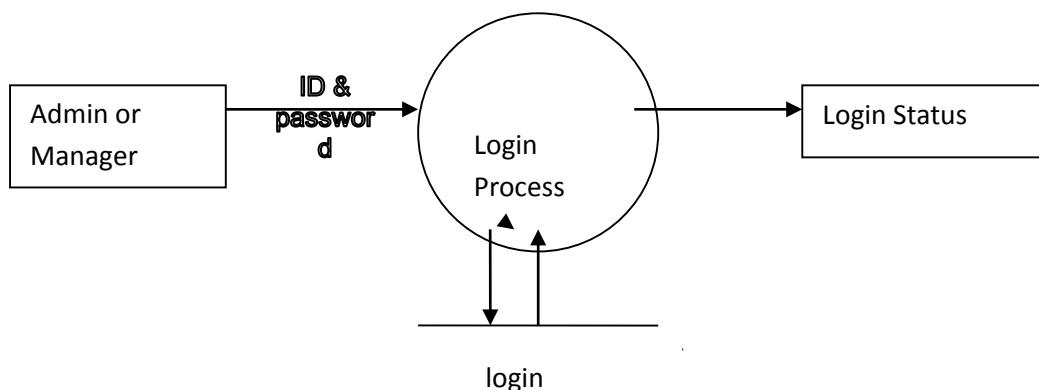


Fig 1.2 Login Process

LOGOUT PROCESS

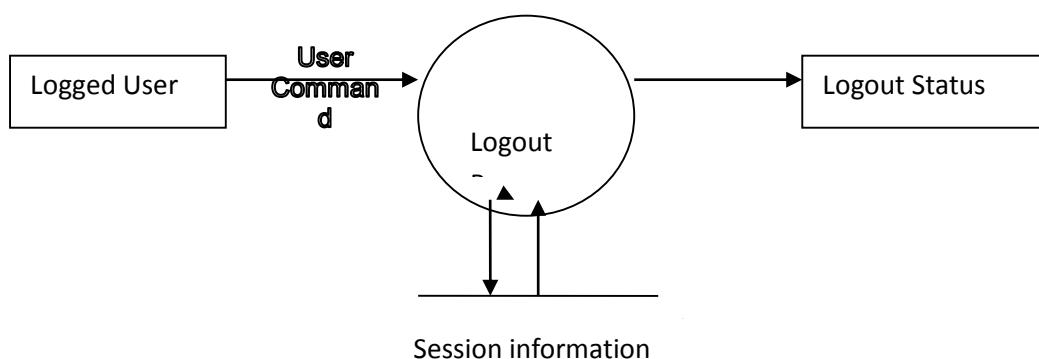
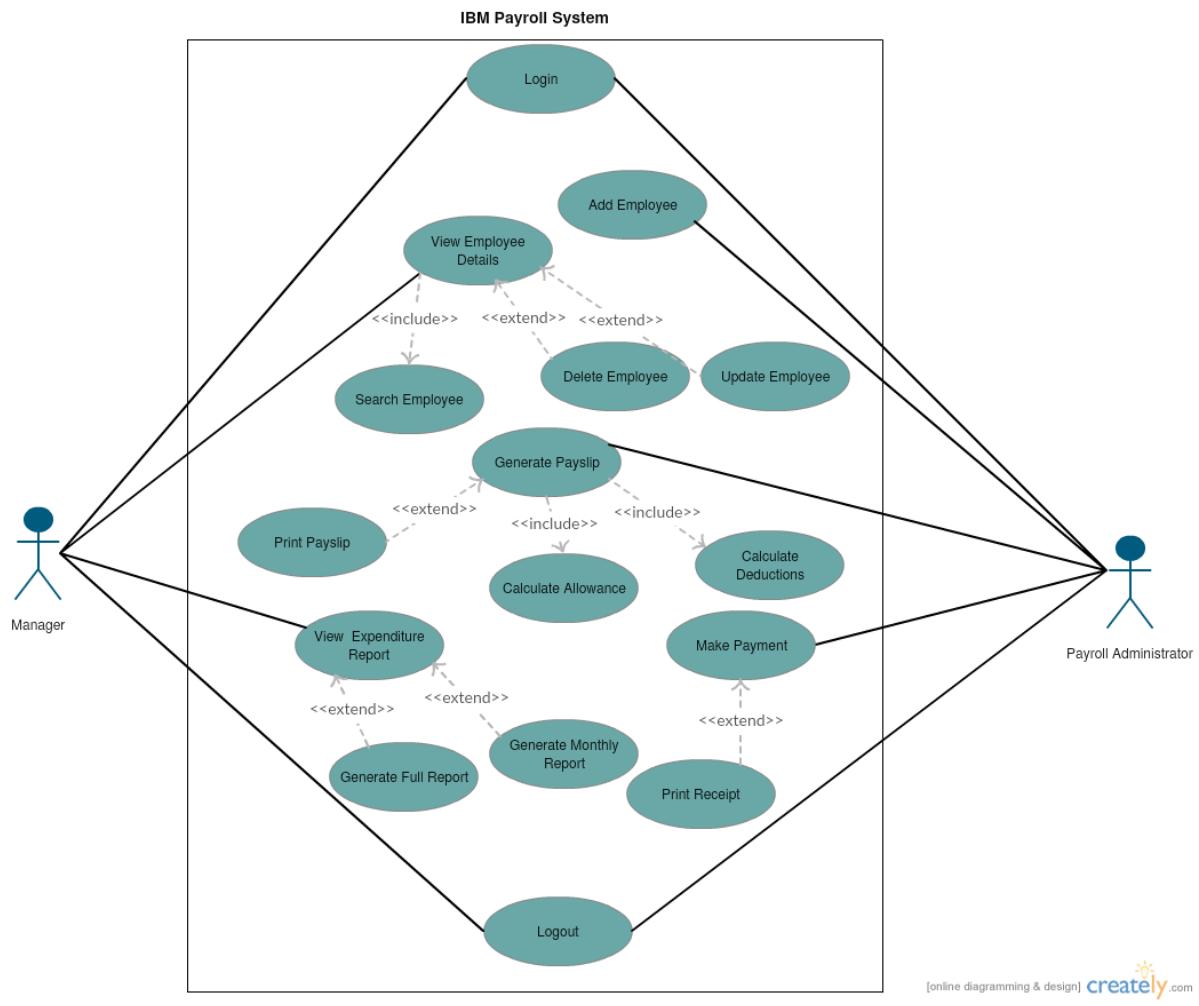


Fig 1.3 Logout Process

2.3 Use Case Diagram



Term	Synonym	Description
Administrator	Admin	An individual who maintains and monitors the database of the system and also performs main functions of the payroll system
Manager	Manager	A user who is only interested in viewing and printing reports. This user views and prints expenditure reports for accounting purposes

LIST OF USE CASES

Use case name	Use case Description	Actor
Log in	The system must allow users to log in with a unique username and password	Administrator (primary business) Manager (primary business)
Logout	This use case destroys logged in users session	Administrator (primary business) Manager (primary business)
Add Employee	The use case allows the administrator to add new employee to the payroll system.	Administrator (primary business)
View Employee Details	The use case allows the administrator to view employee details	Administrator (primary business)
Update Employee	The use case allows the administrator to update employee details	Administrator (primary business)
Search Employee	The use case allows the admin to search for an employee by employee ID	Administrator (primary business)
Delete Employee	The use case allows the administrator to delete an employee's details from the payroll system.	Administrator (primary business)
Generate Payslip	The use case allows the user to generate a payslip for a particular user.	Administrator (primary business)
Calculate Allowance	The use case allows calculation and addition of monthly allowances for a particular employee	Administrator (primary business)
Calculate Deductions	The use case allows calculation and subtraction of monthly deductions for a particular employee	Administrator (primary business)

Use case name	Use case Description	Actor
View Expenditure Report	The use case allows the user to view payroll expenditure reports	Administrator (primary business) Manager (primary business)
Generate Full Report	The use case allows the user to view total money spent on employees	Administrator (primary business) Manager (primary business)
Generate Monthly Report	The use case allows the user to view expenditure reports by month	Administrator (primary business) Manager (primary business)
Make Payment	The use case allows the money to be paid to the employee's account.	Administrator (primary business)
Print Payment Receipt	The use case allows the user to print the payment receipt.	Administrator (primary business)

2.4 Class Diagram

Class Diagram provides an overview of the target system by describing the objects and classes inside the system and the relationships between them (Visual-paradigm.com, n.d.). In the class diagram they are two main users of the system, the manage and the administrator. Both of this classes inherit attributes from the User super class. The administrator interacts with all the other classes while the manage can only view reports.

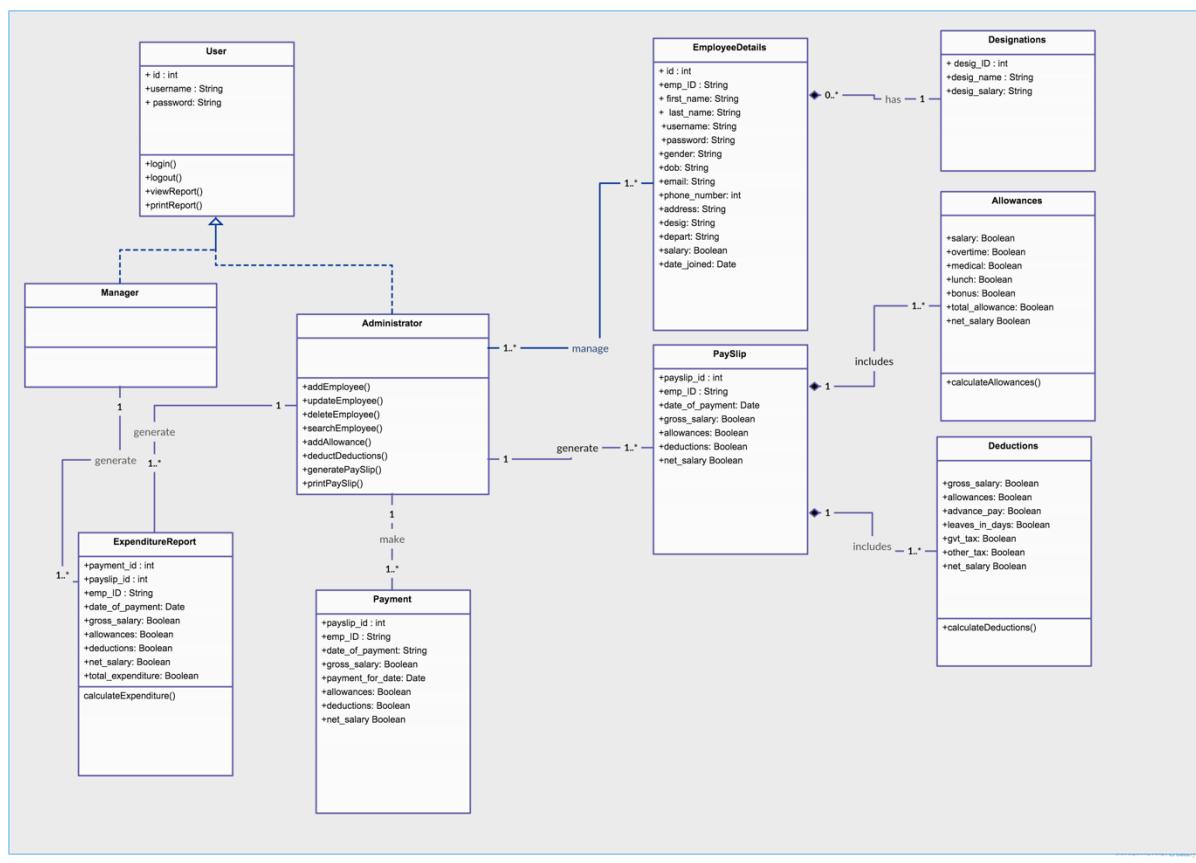


Fig 1.5 Class Diagram

3.0 Implementation

There are two accounts in the home page of the IBM payroll system which is playing different roles and only valid user can access these accounts and perform some actions.

A. Admin Account

B. Manager Account

3.1 Admin Account

After entering the a valid username and password, admin can perform these actions

- Admin Login
- Admin Menu
- Add Employee
- Update Employee
- Search Record
- Delete Record
- Allowance
- Deduction
- Make Payment
- Expenditure Report
- Print Report

3.1.1 Admin Login

This system is more secure and if the user enters a wrong user id and password or leave the blank space it will show the error only valid admin can access this account.

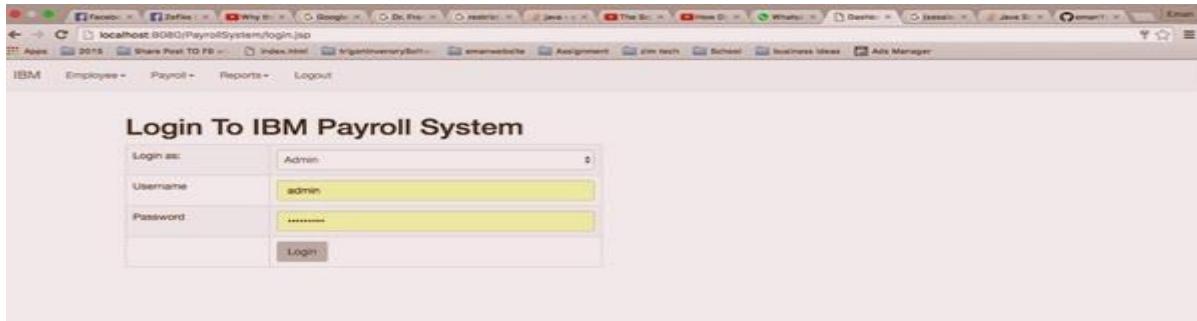


Fig 3.1.1 admin login

3.1.2 Home menu Admin

After successfully login admin can perform different actions which are showing in the payroll menu image.



Fig 3.1.2 Admin home page

3.1.3 Add Employee

Admin hire the new employees and fill their details in a proper form at the add employee page and can also provide the salary of the employee.

Add Employee

Employee ID:

First Name:

Last Name:

Gender:
 Male Female

Date Of Birth:
 dd/mm/yyyy

Email Address:

Phone Number:

Home Address:

Designation:
 Please Select

Department:
 Please Select

Salary In Ringgit:
 dd/mm/yyyy

Date Joined:
 dd/mm/yyyy

Reset Add

Fig 3.1.3 information about the employee

3.1.4 Update Employee

Employees' details are not fixed in this payroll system if in the future any correction needed so it can be easily updated by the admin. For example, name correction, salary increment, department change but Employee Id will be fixed it cannot be changed because it is a primary key in this system.

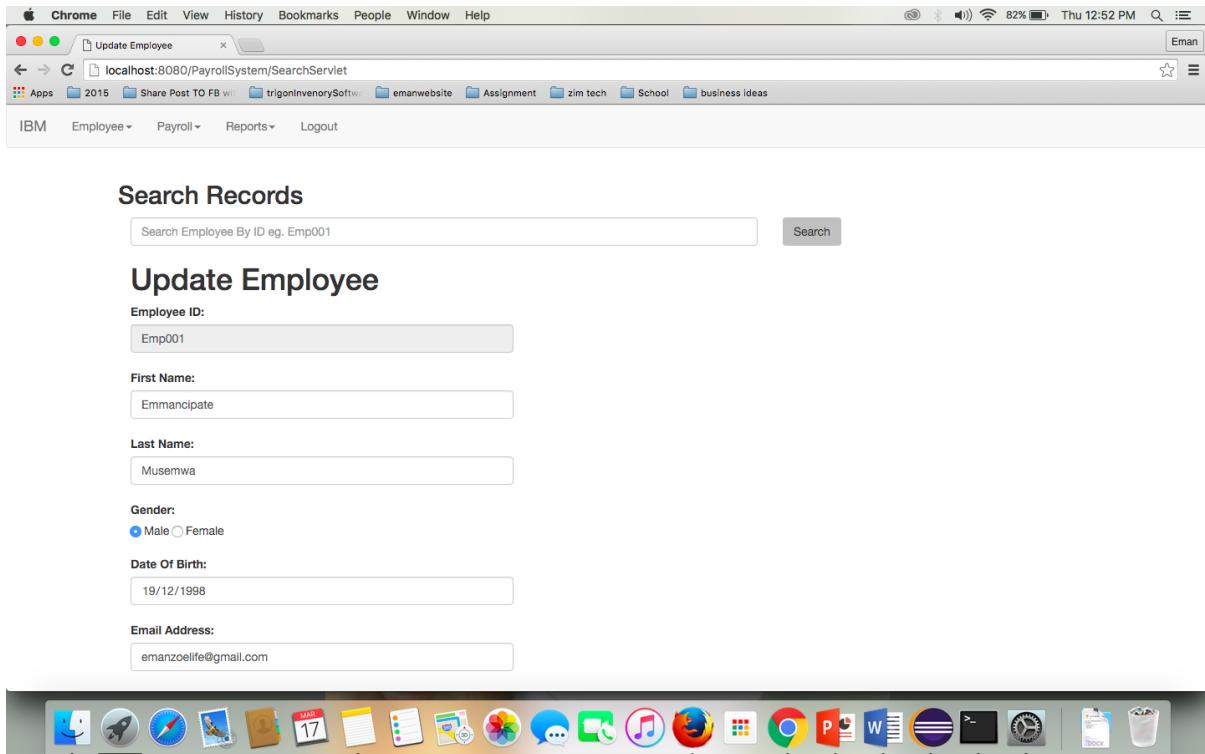


Fig 3.1.4 Update Data

3.1.5 Search Record

This system is providing search box where admin can easily get the information about the particular employee after entering a valid employee id and if the admin enter wrong employee id it will not show any results so will save the much time of the admin during updating any record of the employee.



Fig 3.1.5 Searching employee

3.1.6 Delete Record

This section will delete the record of the particular employee after searching employee id on the search box only admin can delete the records or employees can also approach to the admin for resign the job.

Search Records

Search Employee By ID eg. Emp001

Search

Employee Details

Employee ID: Emp001

First Name: Emmancipate

Last Name: Musemwa

Salary In Ringgits: 15000

Delete Employee



Fig 3.1.6 Delete employee record

3.1.7 Calculate Allowance

Admin provide the extra salary or all other benefits if the employees do the overtime work and it can be calculated easily because in this system most of the things are inbuilt which will calculate the exact amount of allowance and will add in the salary

Type	Amount
Salary	15000
Overtime	0
Medical	0
Lunch	0
Bonus	0

Fig 3.1.7 Allowance Calculation

3.1.8 Calculate Deduction

The same operation will be done as allowance but here admin will subtract all the amount from the salary which is taken in advance. For example, advance pay, leaves in days etc.

The screenshot shows a web application running in a Chrome browser on a Mac OS X desktop. The title bar says 'Allowance'. The address bar shows 'localhost:8080/PayrollSystem/SearchServlet'. The main content area has two sections: 'Employee Details' on the left and 'Deduction' on the right. In 'Employee Details', the Employee ID is Emp001, First Name is Emmancipate, Last Name is Musemwa, and Salary in Ringgits is 15000. In 'Deduction', it asks for 'Deductions for the month of' (month and year fields are empty). The deduction table shows:

Type	Amount
Gross Salary (Including Allowances)	
Advance Pay	0
Leaves in Days	0
Gvnt Tax 6%	0.06
Other Tax	0

At the bottom are 'Calculate' and 'Save' buttons. The Mac OS X dock at the bottom contains icons for various applications like Finder, Mail, Safari, and others.

Fig 3.1.8 Deduction Calculation

3.1.9 Make Payment

Admin will make a payment after all allowance and deduction amount from the salary for a particular employee and a specific month or can also print the receipt of it.

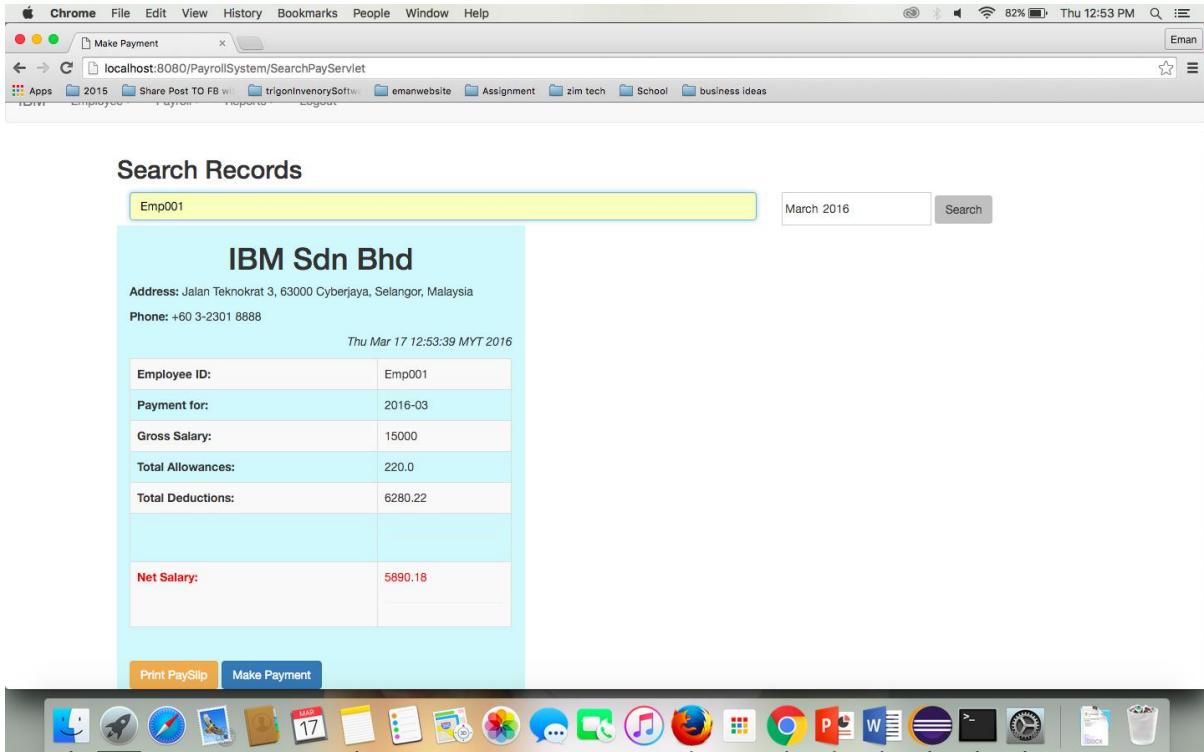


Fig 3.1.9 Make payment of particular employee

3.1.10 ExpenditureReport

Admin can see the expenditure report of all the employees which they have made a payment or can also see the expenditure report of a particular month.

Payment ID	Pay Slip ID	Employee ID	Payment For	Gross Salary	Allowances	Deductions	Net Salary
3	1	Emp001	2016-04	RM 15000	RM 220.0	RM 6280.22	RM 5890.18
4	1	Emp001	2016-03	RM 15000	RM 220.0	RM 6280.22	RM 5890.18
5	1	Emp001	2016-03	RM 15000	RM 220.0	RM 6280.22	RM 5890.18
Total Expenditure							
RM 17970							

Fig 3.1.10 Expenditure report

3.1.11 Print Report

Admin can also print the employee expenditure report if the employees needed.

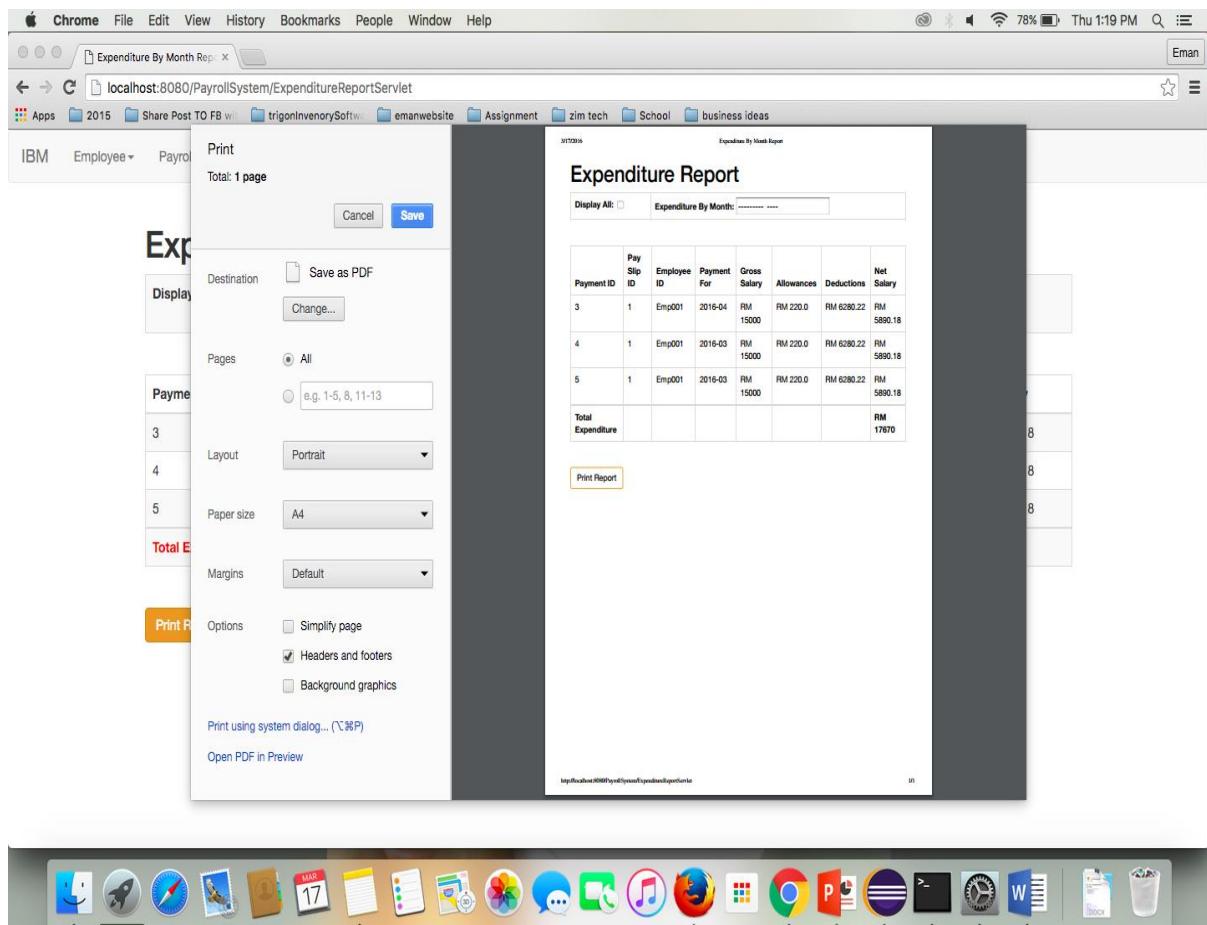
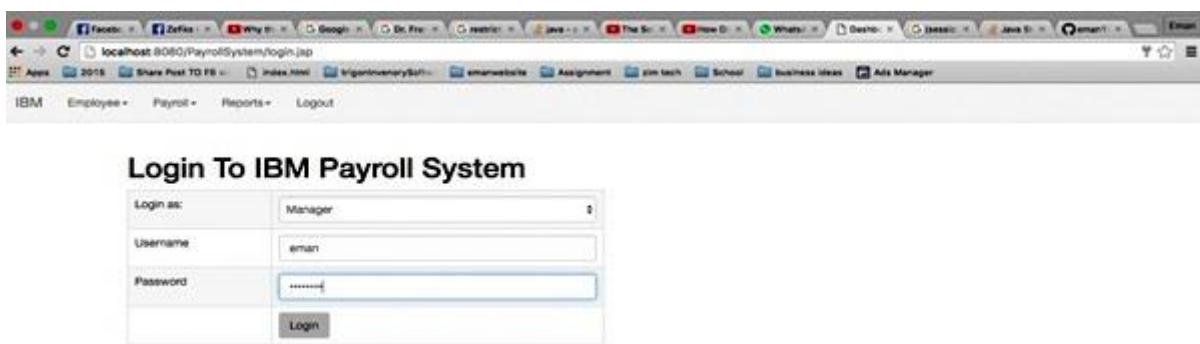


Fig 3.1.21 Print report

3.2 Manager Account

Manager can perform some task which are given below after successfully login

- Expenditure Report
- Expenditure Report by Month
- Print the Report



4.0 Testing

It is the job of programmer to test, as far as possible, that all parts of the programs work correctly. It should be realized that complete testing is not possible except in the case of the most trivial program. One can never be completely certain that all errors have been removed, but sufficient test can be performed to give a reasonable measure of confidences in the program.

4.1 Unit Testing

In unit testing the programmer tests all the individual modules that are making the system. Unit testing gives stress on the modules independently of each other. This helps in easily detecting the logical errors in the modules. In my project all the individual files that were coded were tested individually for any errors. They were tested for the inputs they take and were tested to see what happens when we enter an illegal value for a field. After the results of an individual module were found satisfactory

4.2 Manual testing

First this payroll system tested manually to find the error during entering any unexpected value into the textbox so every value is tested to check that it value is in valid form or invalid and data is submitting on the database.

4.2.1 LEGEND

A lot of symbols are used in the testing data and reports which are given as follows:

- 1.** N/A : Not Applied.
- 2.** A : Apply
- 3.** S : Select this field.
- 4.** BFP : Blank Field Prohibited.
- 5.** BFA : Blank Field can be applied.
- 6.** MBE : Must Be Existing.
- 7.** EMA : Error Message Appeared.
- 8.** D: Default Value

4.3 IBM Payroll System Classes

UNIT TESTING OVERALL RESULT			
S.No.	Module Description	Result	Successful/Failed

1	Login Admin	As Expected	Successful
2	Login Manager	As Expected	Successful
3	Add Employee	As Expected	Successful
4	Update Employee	As Expected	Successful
5	Search Record	As Expected	Successful
6	Delete Record	As Expected	Successful
7	Allowance	As Expected	Successful
8	Deduction	As Expected	Successful
9	Make Payment	As Expected	Successful
10	Expenditure Report	As Expected	Successful
11	Expenditure Report By Month	As Expected	Successful
12	Logout	As Expected	Successful

UNIT Test Case Status:	<i>Successful</i>
-------------------------------	-------------------

4.3.1 Module Name: Admin Login

ARRAY OF VALUES									
Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result	
Username	BFP	N/A	A	N/A	N/A	EMA	EMA	Pass	

Password	BFP	A	A	N/A	N/A	EMA	EMA	Pass
Module Test Status:						<i>Successful</i>		

4.3.2 Module Name: Manager Login

ARRAY OF VALUES									
Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result	
Username	BFP	N/A	A	N/A	N/A	EMA	EMA	Pass	
Password	BFP	A	A	N/A	N/A	EMA	EMA	Pass	
Module Test Status:						<i>Successful</i>			

4.3.3 Module Name: Add Employee

ARRAY OF VALUES									
Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result	
Employee ID	BFP	A	A	N/A	N/A	EMA	EMA	Pass	

First Name	BFP	N/A	A	N/A	N/A	EMA	EMA	Pass
Last Name:	BFP	N/A	A	N/A	N/A	EMA	EMA	Pass
Gender:	BFP	N/A	S	N/A	N/A	EMA	EMA	Pass
Date Of Birth:	BFP	S	N/A	N/A	N/A	EMA	EMA	Pass
Email Address:	BFP	A	A	A	A	EMA	EMA	Pass
Phone Number	BFP	A	N/A	N/A	N/A	EMA	EMA	Pass
Home Address	BFP	A	A	A	A	EMA	EMA	Pass
Designation	BFP	N/A	S	N/A	N/A	EMA	EMA	Pass
Department	BFP	N/A	S	N/A	N/A	EMA	EMA	Pass
Default Salary In Ringgits	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Date Joined:	BFP	S	N/A	N/A	N/A	EMA	EMA	Pass

Module Test Status:

Successful

4.3.4 Module Name: Update Employee

ARRAY OF VALUES								
Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result
Employee ID	BFP	A	A	N/A	N/A	EMA	EMA	Pass

First Name	BFP	N/A	A	N/A	N/A	EMA	EMA	Pass
Last Name:	BFP	N/A	A	N/A	N/A	EMA	EMA	Pass
Gender:	BFP	N/A	S	N/A	N/A	EMA	EMA	Pass
Date Of Birth:	BFP	S	N/A	N/A	N/A	EMA	EMA	Pass
Email Address:	BFP	A	A	A	A	EMA	EMA	Pass
Phone Number	BFP	A	N/A	N/A	N/A	EMA	EMA	Pass
Home Address	BFP	A	A	A	A	EMA	EMA	Pass
Designation	BFP	N/A	S	N/A	N/A	EMA	EMA	Pass
Department	BFP	N/A	S	N/A	N/A	EMA	EMA	Pass
Default Salary In Ringgits	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Date Joined:	BFP	S	N/A	N/A	N/A	EMA	EMA	Pass
				Module Test Status:			<i>Successful</i>	

4.3.5 Module Name: Search Records

ARRAY OF VALUES									
Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result	
Employee ID	BFP	A	A	N/A	N/A	EMA	EMA	Pass	

Module Test Status:	<i>Successful</i>
----------------------------	-------------------

4.3.6 Module Name: Delete Records

ARRAY OF VALUES									
Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result	
Employee ID	BFP	A	A	N/A	N/A	EMA	EMA	Pass	
Module Test Status: <i>Successful</i>									

4.3.7 Module Name: Allowance

ARRAY OF VALUES									
Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result	
Employee ID	BFP	A	A	N/A	N/A	EMA	EMA	Pass	
First Name	BFP	N/A	D	N/A	N/A	EMA	EMA	Pass	

Last Name:	BFP	N/A	D	N/A	N/A	EMA	EMA	Pass
Salary	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Default Salary In Ringgits	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Bonus and incentives for month of	BFP	S	S	N/A	N/A	EMA	EMA	Pass
Overtime	BFP	A	N/A	N/A	N/A	EMA	EMA	Pass
Medical	BFP	A	N/A	N/A	N/A	EMA	EMA	Pass
Lunch	BFP	A	N/A	N/A	N/A	EMA	EMA	Pass
Bonus	BFP	A	N/A	N/A	N/A	EMA	EMA	Pass
Total Allowance	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Net Salary	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Module Test Status:						<i>Successful</i>		

4.3.8 Module Name: Deduction

ARRAY OF VALUES									
Field Name	Blank Entry	Numeric Entry	Alphabetic Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result	
Employee ID	BFP	A	A	N/A	N/A	EMA	EMA	Pass	
First Name	BFP	N/A	D	N/A	N/A	EMA	EMA	Pass	

Last Name:	BFP	N/A	D	N/A	N/A	EMA	EMA	Pass
Salary In Ringgits	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Deduction for the month of	BFP	S	S	N/A	N/A	EMA	EMA	Pass
Gross Salary(Including Allowances)	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Advance Pay	BFP	A	N/A	N/A	N/A	EMA	EMA	Pass
Leaves in Days	BFP	A	N/A	N/A	N/A	EMA	EMA	Pass
Government Tax in 6%	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Other Tax	BFP	A	N/A	N/A	N/A	EMA	EMA	Pass
Total Deduction	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Net Salary	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass

Module Test Status:	<i>Successful</i>
----------------------------	-------------------

4.3.9 Module Name: Make Payment

ARRAY OF VALUES									
Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result	
Employee ID	BFP	A	A	N/A	N/A	EMA	EMA	Pass	
Date	BFP	S	S	N/A	N/A	EMA	EMA	Pass	

Module Test Status:	<i>Successful</i>
----------------------------	-------------------

4.3.10 Module Name: Expenditure Report

ARRAY OF VALUES								
Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result
Display All	BFP	N/A	N/A	N/A	A	EMA	EMA	Pass
Expenditure By Month	BFP	S	S	N/A	N/A	EMA	EMA	Pass
Payment ID	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Pay Slip ID	BFP	D	N/A	N/A	N/A	EMA	EMA	Pass
Employee ID	BFP	D	D	N/A	N/A	EMA	EMA	Pass
Payment For	BFP	D	N/A	N/A	D	EMA	EMA	Pass
Gross Salary	BFP	D	D	N/A	N/A	EMA	EMA	Pass
Allowances	BFP	D	D	N/A	N/A	EMA	EMA	Pass
Deductions	BFP	D	D	N/A	N/A	EMA	EMA	Pass
Net Salary	BFP	D	D	N/A	N/A	EMA	EMA	Pass

Module Test Status:	<i>Successful</i>
----------------------------	-------------------

4.311 Module Name: Expenditure Report by Month

ARRAY OF VALUES								
------------------------	--	--	--	--	--	--	--	--

Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result
Date	BFP	S	S	N/A	N/A	EMA	EMA	Pass

Module Test Status:	<i>Successful</i>
----------------------------	-------------------

4.3.12 Module Name: Logout

ARRAY OF VALUES

Field Name	Blank Entry	Numeric Entry	Alphabet Entry	@ Entry	Special Symbol	Expected Result	Observed Result	Test Result
Logout	BFP	N/A	N/A	N/A	N/A	EMA	EMA	pass

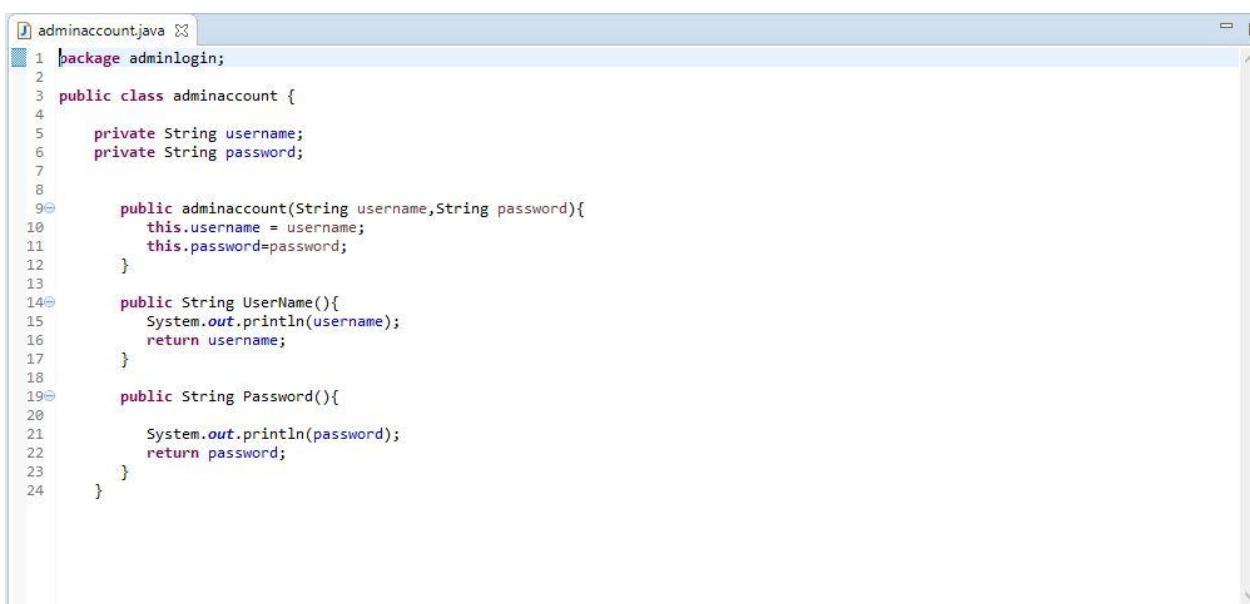
Module Test Status:	<i>Successful</i>
----------------------------	-------------------

4.4 Junit Testing

Joint testing is an automated testing which is fast in comparison to the manual testing and perform by the system. It will test all the class individually of the IBM Payroll System.

4.4.1 Admin Account

In the admin account username and password are required so it will test suit will test both class together.



```

1 package adminlogin;
2
3 public class adminaccount {
4
5     private String username;
6     private String password;
7
8
9     public adminaccount(String username, String password){
10        this.username = username;
11        this.password=password;
12    }
13
14     public String UserName(){
15        System.out.println(username);
16        return username;
17    }
18
19     public String Password(){
20        System.out.println(password);
21        return password;
22    }
23
24 }

```

Fig 4.4.1 admin class

The screenshot shows the Eclipse IDE interface. In the top left, the code editor displays `UserNameTest.java` with the following content:

```
1 package adminlogin;
2 import org.junit.Test;
3
4 public class UserNameTest {
5
6     String username = "admin";
7     adminaccount user = new adminaccount(username, null);
8
9     @Test
10    public void testPrintMessage() {
11        System.out.println("Username of the admin is");
12        assertEquals(username, user.UserName());
13    }
14 }
```

In the bottom right, the JUnit view shows the test results for `UserNameTest`:

- Runs: 1/1
- Errors: 0
- Failures: 0

The console output shows the printed message: `Username of the admin is admin`.

Fig 4.4.2 Test admin's username

The screenshot shows the Eclipse IDE interface. In the top left, the code editor displays `UserPasswordTest.java` with the following content:

```
1 package adminlogin;
2 import org.junit.Test;
3 import org.junit.Ignore;
4 import static org.junit.Assert.assertEquals;
5
6 public class UserPasswordTest {
7
8     String password = "admin1000";
9     adminaccount upassword = new adminaccount(null, password);
10
11    @Test
12    public void testPrintMessage() {
13        System.out.println("Password of the admin is");
14        assertEquals(password, upassword.Password());
15    }
16 }
```

In the bottom right, the JUnit view shows the test results for `UserPasswordTest`:

- Runs: 1/1
- Errors: 0
- Failures: 0

The console output shows the printed message: `Password of the admin is admin1000`.

Fig 4.4.3 Test password

Here there are two different test so test suit will be needed to run both test classes together to check the username and password of the admin.

The screenshot shows an IDE interface with two tabs at the top: 'UserPasswordTest.java' and 'admintestrunner.java'. The 'admintestrunner.java' tab is active, displaying the following code:

```

1 package adminlogin;
2
3 import org.junit.runner.RunWith;
4 import org.junit.runners.Suite;
5 @RunWith(Suite.class)
6 @Suite.SuiteClasses({
7     UserNameTest.class,
8     UserPasswordTest.class
9 })
10 public class admintestrunner {
11 }

```

Below the tabs is a 'Console' window showing the output of the test run:

```

<terminated> admintestrunner [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\ Username of the admin is admin Password of the admin is admin1000

```

To the right is a 'JUnit' window showing the test results:

- Finished after 0.014 seconds
- Runs: 2/2 Errors: 0 Failures: 0
- adminlogin.admintestrunner [Runner: JUnit 4] (0.001 s) Failure Trace
 - adminlogin.UserNameTest (0.001 s)
 - adminlogin.UserPasswordTest (0.000 s)

Fig 4.4.4 test suit admin login test

Junit Test	Description	Expected result	Actual result	Time elapsed
Admin Login	To check the username and password of the admin	Username and password are valid	Test passed	0.01 and 0.000 second

4.4.5 Manager Login

It will check the username and password of the manager account using test suit class.

The screenshot shows an IDE interface with a single tab: 'ManagerAccount.java'. The code is as follows:

```

1 package managerlogin;
2
3
4 public class ManagerAccount {
5
6     private String musername;
7     private String mpassword;
8
9
10    public ManagerAccount(String musername, String mpassword){
11        this.musername = musername;
12        this.mpassword=mpassword;
13    }
14
15    public String MUserName(){
16        System.out.println(musername);
17        return musername;
18    }
19
20    public String MPASSWORD(){
21        System.out.println(mpassword);
22        return mpassword;
23    }
24
25 }

```

Fig 4.4.5 manager class

The screenshot shows the Eclipse IDE interface. On the left, the code editor displays `MUserNameTest.java` with the following content:

```
1 package managerlogin;
2 import org.junit.Test;
3 import org.junit.Ignore;
4 import static org.junit.Assert.assertEquals;
5
6 public class MUserNameTest {
7
8     String musername = "eman";
9     ManagerAccount muser = new ManagerAccount(musername, null);
10
11    @Test
12    public void testPrintMessage() {
13        System.out.println("Username of the Manager is");
14        assertEquals(musername, muser.MUserName());
15    }
16 }
```

On the right, the JUnit view shows the test results:

- Console output: <terminated> MUserNameTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin
Username of the Manager is
eman
- JUnit view: Finished after 0.016 seconds
Runs: 1/1 Errors: 0 Failures: 0
managerlogin.MUserNameTest [Runner: JUnit 4] (0.000) Failure Trace
testPrintMessage (0.000 s)

Fig 4.4.6 Test username of the manager

The screenshot shows the Eclipse IDE interface. On the left, the code editor displays `MUserPasswordTest.java` with the following content:

```
1 package managerlogin;
2 import org.junit.Test;
3
4 public class MUserPasswordTest {
5
6     String mpassword = "eman1000";
7     ManagerAccount mpass = new ManagerAccount(null, mpassword);
8
9     @Test
10    public void testPrintMessage() {
11        System.out.println("Password of the Manager is");
12        assertEquals(mpassword, mpass.MPassword());
13    }
14 }
```

On the right, the JUnit view shows the test results:

- Console output: <terminated> MUserPasswordTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin
Password of the Manager is
eman1000
- JUnit view: Finished after 0.017 seconds
Runs: 1/1 Errors: 0 Failures: 0
managerlogin.MUserPasswordTest [Runner: JUnit 4] (0.000) Failure Trace
testPrintMessage (0.000 s)

Fig 4.4.7 test password of the manager

Here is also two different test classes so test suit will be needed so check the data.

The screenshot shows the Eclipse IDE interface. On the left, the code editor displays `ManagerRunnerTest.java` with the following content:

```

1 package managerlogin;
2
3 import org.junit.runner.RunWith;
4 @RunWith(Suite.class)
5 @Suite.SuiteClasses({
6     MUserNameTest.class,
7     MUUserPasswordTest.class
8 })
9 public class ManagerRunnerTest {
10 }
11 
```

The `Console` view shows the output of the test run:

```

<terminated> ManagerRunnerTest [JUnit] C:\Program Files\Java\jre1.8.0_71\jre\bin\java
Username of the Manager is
eman
Password of the Manager is
eman1000

```

The `JUnit` view shows the test results:

- Runs: 2/2
- Errors: 0
- Failures: 0
- managerlogin.ManagerRunnerTest [Runner: JUnit 4] (0.000 s)
- managerlogin.MUserNameTest (0.000 s)
- managerlogin.MUUserPasswordTest (0.000 s)

Fig 4.4.8 Test suit of manager class

Junit Test	Description	Expected result	Actual result	Time elapsed
Manager Login	To check the username and password of the admin	Username and password are valid	Test passed	0.00 and 0.000 second

4.5.1 Add Employee Class

Here system will test all the array values of the text box which are entered by the admin.

The screenshot shows the Eclipse IDE interface. On the left, the code editor displays `addEmployeeServlet.java` and `addEmployeeServletTest.java`. The `addEmployeeServlet.java` code is as follows:

```

1 package addEmployeePackage;
2 import java.util.Arrays;
3
4 public class addEmployeeServlet {
5
6     public String[] ReqParameter() {
7
8         String[] requestparameter = { "EMP201", "Jackson", "Bond", "01/21/1989", "Male", "jackson@yahoo.com" ,
9             "0172345658", "H.No 34,Jack Street,Malaysia" , "Developer" , "ICT", "6000", "02/21/2004" };
10
11     System.out.println("Request Parameters are : " + Arrays.toString(requestparameter));
12
13     return requestparameter;
14
15
16 }
17
18 }
19 
```

Fig 4.5.1 Actual format of the value which is needed to fill in the text box.



The screenshot shows an IDE interface with the following details:

- Code Editor:** The main window displays `addEmployeeServletTest.java`. The code defines a class `addEmployeeServletTest` containing a single test method `testParameter()`. The test method prints the passed parameters and asserts they match the expected array `reqpara`.
- Test Results:** Below the editor, the JUnit tab in the toolbar is selected. It shows 1 run, 0 errors, and 0 failures.
- Failure Trace:** A detailed failure trace is visible, showing the test method `testParameter()` and its duration of 0.006 seconds.

Fig 4.5.2 test the array values which are entered by the admin.

Junit Test	Description	Expected result	Actual result	Time elapsed
Add Employee Details	To check all the input parameter which are passed by the admin .It will check that these data type are valid format or not.	All the inputs are valid	Test passed	0.006 seconds

Test all the conditions for the input parameter. For example if user will not enter any input in a form else he enters wrong data type.

The screenshot shows an IDE interface with several tabs at the top: EmailTesting.java, addEmployeeServlet.java, AssertionsTest.java, and readme.java. The main area displays Java code for testing employee data. The code includes various string assignments and multiple assertEquals() assertions. One assertion involves a variable obj15, which is highlighted in blue. The right side of the interface shows the JUnit results window. It indicates "Finished after 0.024 seconds" with "Runs: 1/1", "Errors: 0", and "Failures: 1". A single failure is listed: "addEmployeePackage.EmailTesting [Runner: JUnit 4] (0.004 s)" with a sub-item "test (0.004 s)". Below this, the "Failure Trace" section shows the error message: "org.junit.ComparisonFailure: expected:<[]> but was:<[0172345658]>" and the stack trace: "at addEmployeePackage.EmailTesting.test(EmailTesting.java:52)".

```

50     String obj9 = "bond";
51     String obj10 = "01/21/1989";
52     String obj10 = "Male";
53     String obj12= "jackson@yahoo.com";
54     /* String obj12= "jackso.com"; */
55     String obj14 = "Male";
56     /*String obj15= "0172345658";*/
57     String obj15= "";
58     String obj18 = "H.NO 34,Jack Street,Malaysia";
59     String obj20= "Developer";
60     String obj22 = "ICT";
61     String obj24= "6000";
62     String obj26="02/21/2004";
63
64     /*Testing parameter when entering values are in correct form*/
65     /*assertEquals(obj1, obj2);*/
66     assertEquals(obj3, obj4);
67     assertEquals(obj5, obj6);
68     assertEquals(obj7, obj8);
69     assertEquals(obj9, obj10);
70     assertEquals(obj11, obj12);
71     assertEquals(obj13, obj14);
72     assertEquals(obj15, obj16);
73     assertEquals(obj17, obj18);
74     assertEquals(obj19, obj20);
75     assertEquals(obj21, obj22);
76     assertEquals(obj23, obj24);
77     assertEquals(obj25, obj26);
78
79     assertNotNull(obj15);
80
81

```

Fig 4.5.3 test blank value

Junit Test	Description	Expected result	Actual result	Time elapsed
Mobile number	Check mobile number column has blank value or not which is obj15	Mobile has not blank value	Test Failed (Admin did not enter any mobile number)	0.004 seconds

```

27     String obj2 = "EMP201";
28     String obj4 = "Jackson";
29     String obj6 = "Bond";
30     String obj8 = "01/21/1989";
31     String obj10 = "Male";
32     String obj12= "jackson@yahoo.com";
33     /* String obj12= "jackso.com";*/
34     String obj14 = "Male";
35     String obj15= "0172345658";
36     /* String obj15= "";*/
37     String obj18 = "H.NO 34,Jack Street,Malaysia";
38     String obj20= "Developer";
39     String obj22 = "ICT";
40     /*int obj22 = 22;*/
41     String obj24= "0000";
42     String obj26="02/21/2004";
43
44     /*Testing parameter when entering values are in corre
45     /*assertEquals(obj1, obj2);*/
46     assertEquals(obj3, obj4);
47     assertEquals(obj5, obj6);
48     assertEquals(obj7, obj8);
49     assertEquals(obj9, obj10);
50     assertEquals(obj11, obj12);
51     assertEquals(obj13, obj14);
52     /*assertEquals(obj15, obj16);*/
53     assertEquals(obj17, obj18);
54     assertEquals(obj19, obj20);
55     assertEquals(obj21, obj22);
56     assertEquals(obj23, obj24);
57     assertEquals(obj25, obj26);

```

Fig 4.5.3 test again after changing some value

Junit Test	Description	Expected result	Actual result	Time elapsed
Mobile Number	Check mobile number column	Admin has entered correct mobile number	Test passed	0.000 second

Validation on Email id which is obj12

```

1 package addEmployeePackage;
2 import org.junit.Assert;
3 import org.junit.Test;
4 public class addEmployeeServletTest {
5     addEmployeeServlet requestparameter = new addEmployeeServlet();
6     String[] repara = {"EMP201","Jackson","Bond","01/21/1989","Male","jackson@yahoo.com",
7     "0172345658","H.NO 34,Jack Street,Malaysia","Developer","ICT","0000","02/21/2004"};
8
9     public void testParameter() {
10         System.out.println("Parameter passed by the users()");
11         assertEquals(repara,requestparameter.ReqParameter());
12     }
13 }

```

Runs: 1/1 Errors: 0 Failures: 1

Failure Trace

arrays first different at element [5]: expected:<jackson@yahoo.com> but was:<jackson@yahoocom>

at addEmployeePackage.addEmployeeServletTest.testParameter(addEmployeeServletTest.java:17)

Fig 4.5.4 wrong email id

Junit Test	Description	Expected result	Actual result	Time elapsed
Email validation	Check that admin has entered correct email id format	Correct email id	Test Failed	0.0051 seconds

The screenshot shows the Eclipse IDE interface with the JUnit perspective active. On the left, there is a code editor window titled "EmailTesting.java" containing Java code for testing employee details. The code includes several assertEqual statements comparing various object variables (obj1 through obj26) against their expected values. On the right, the JUnit runner window displays the test results: "Finished after 0.016 seconds", "Runs: 1/1", "Errors: 0", and "Failures: 0". A green progress bar indicates a successful run.

```

27     String obj2 = "EMP201";
28     String obj4 = "Jackson";
29     String obj6 = "Bond";
30     String obj8 = "01/21/1989";
31     String obj10 = "Male";
32     String obj12= "jackson@yahoo.com";
33     /*String obj12= "jackso.com";*/
34     String obj14 = "Male";
35     String obj15= "0172345658";
36     /* String obj15= "";*/
37     String obj18 = "H.NO 34,Jack Street,Malaysia";
38     String obj20= "Developer";
39     String obj22 = "ICT";
40     /*int obj22 = 22;*/
41     String obj24= "6000";
42     String obj26="02/21/2004";
43
44     /*Testing parameter when entering values are in
45     /*assertEquals(obj1, obj2);*/
46     assertEquals(obj3, obj4);
47     assertEquals(obj5, obj6);
48     assertEquals(obj7, obj8);
49     assertEquals(obj9, obj10);
50     assertEquals(obj11, obj12);
51     assertEquals(obj13, obj14);
52     assertEquals(obj15, obj16);
53     assertEquals(obj17, obj18);
54     assertEquals(obj19, obj20);
55     assertEquals(obj21, obj22);
56     assertEquals(obj23, obj24);
57     assertEquals(obj25, obj26);

```

Fig 4.5.5 test email id

Junit Test	Description	Expected result	Actual result	Time elapsed
Validation Email Id	Check that email id is a valid form	Email id is valid	Test passed	0.000 second

4.6.1 Allowance Amount

It will test the actual allowance amount and system generated allowance.

```

calculateServlet.java
1 package calculatePackage;
2
3 public class calculateServlet {
4
5     public double allowance(double gross_salary, double overtime, double medical, double lunch, double bonus) {
6         double allowance = (medical + overtime + bonus + lunch);
7
8         double net_salary = (gross_salary + allowance);
9         System.out.println("Allowance values is : " + allowance);
10        System.out.println("Net Salary is values are: " + net_salary);
11
12        return allowance;
13    }
14 }
15
16
17
18

```

Fig 4.6.1 Array values of the allowance

```

calculateServletTest.java
1 package calculatePackage;
2 import static org.junit.Assert.assertEquals;
3 import org.junit.Test;
4
5
6 public class calculateServletTest {
7
8     calculateServlet al
9     = new calculateServlet();
10    double allowance =
11    al.allowance(15000, 1000, 500, 1200, 900);
12
13    /*success allowance double totalallowance = 3600; */
14    double totalallowance = 3599;
15    @Test
16    public void netsum() {
17        System.out
18        .println("Total Allowance of the specific "
19        + "employee is:" + allowance + "=" + totalallowance);
20
21    assertEquals(Double.doubleToLongBits(allowance),
22        Double.doubleToLongBits(totalallowance));
23
24    }
25

```

JUnit Results:

- Runs: 1/1
- Errors: 0
- Failures: 1

Failure Trace:

```

netsum (0.002 s) expected:<4660134898793709568> but was:<4660132699770454016>
calculateServletTest.netsum(calculateServletTest.java:21)

```

Console Output:

```

<terminated> calculateServletTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (Mar 26, 2016, 12:08:07 PM)
Allowance values is : 3600.0
Net Salary is values are: 18600.0
Total Allowance of the specific employee is:3600.0=3599.0

```

Fig 4.6.2 Comparing two values

Junit Test	Description	Expected result	Actual result	Time elapsed
Employee' Allowance	Calculation of net salary and allowance with comparing actual amount.	Allowance value has successfully calculated.	Test Failed (system generated value is different actual value)	0.002 second

The screenshot shows the Eclipse IDE interface. On the left, there are two Java files: `calculateServlet.java` and `calculateServletTest.java`. The `calculateServletTest.java` file contains the following code:

```

1 package calculatePackage;
2 import static org.junit.Assert.assertEquals;
3 import org.junit.Test;
4
5 public class calculateServletTest {
6     calculateServlet al
7         = new calculateServlet();
8     double allowance =
9     al.allowance(15000,1000, 500,1200, 900);
10
11     double totalallowance = 3600;
12     @test
13     public void netsum() {
14         System.out
15         .println("Total Allowance of the specific "
16         + "employee is:"+allowance+"=totalallowance");
17     assertEqual(Double.doubleToLongBits(allowance),
18             Double.doubleToLongBits(totalallowance));
19 }
20
21 }
22
23
24
25

```

In the center, the JUnit view shows the test results: "Finished after 0.017 seconds", "Runs: 1/1", "Errors: 0", and "Failures: 0". Below the JUnit view, the Console tab displays the output of the test run:

```

<terminated> calculateServletTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (Mar 26, 2016, 12:03:24 PM)
Allowance values is : 3600.0
Net Salary is values are: 18600.0
Total Allowance of the specific employee is:3600.0=3600.0

```

Fig 4.6.3 Test allowance of the employee

Junit Test	Description	Expected result	Actual result	Time elapsed
Employee' Allowance	Calculation of net salary and allowance with comparing actual amount.	Allowance value has successfully calculated.	Test passed (Actual value is same as system generated value)	0.000 second

4.7.1 Deduction Amount

It will test the deduction amount system generated with the actual amount.

```
CalculateDeductionServlet.java
1 package calculateDeductionPackage;
2
3 public class CalculateDeductionServlet {
4
5     public double netsalary(double advance_pay, double leaves,double gst,double other_tax,double gross_salary) {
6
7         double gvt = (gross_salary * gst);
8
9         double otherT = (gross_salary * other_tax);
10
11         double deduction = (advance_pay + leaves + otherT + gvt);
12
13         double net_salary = (gross_salary - deduction);
14
15
16         System.out.println("Deduction values are: " + deduction);
17         System.out.println("Net Salary is values are: " + net_salary);
18
19         return deduction;
20
21     }
22
23 }
24
```

Fig 4.7.1 Array values of the deduction which have to perform operation.

```
CalculateDeductionServletTest.java
1 package calculateDeductionPackage;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class CalculateDeductionServletTest {
7
8     CalculateDeductionServlet dect = new CalculateDeductionServlet();
9     double deduction= dect.netsalary(2000, 700, 0.06,0.05,15000);
10
11    /* double netdeduction=4350; success */
12
13    double netdeduction= 4000; // wrong calculation
14
15    @Test
16    public void netsum() {
17        System.out.println(" Comparison of two values of deduction is : "
18            + deduction+ " "
19            + ""+ "=" +netdeduction);
20
21        assertEquals(Double.doubleToLongBits(netdeduction),
22            Double.doubleToLongBits(deduction));
23    }
24 }
```

JUnit Results:

- Runs: 1 / Errors: 1 / Failures: 1
- Failure Trace:
 - netsum (0.003) 508095930368 but was:<466150489028>

Console Output:

```
<terminated> CalculateDeductionServletTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (Mar 26, 2016, 11:58:53 AM)
Deduction values are: 4350.0
Net Salary is values are: 10650.0
Comparison of two values of deduction is : 4350.0 = 4000.0
```

Fig 4.7.2 check the actual value and system generated value.

Junit Test	Description	Expected result	Actual result	Time elapsed
Deduction	Comparing the actual amount of deduction with the system generated.	System generated deduction value is same as actual value.	Test Failed	0.002 second

The screenshot shows the Eclipse IDE interface. On the left, the code editor displays `CalculateDeductionServletTest.java` with the following content:

```

1 package calculateDeductionPackage;
2 import static org.junit.Assert.*;
3 import org.junit.Test;
4
5 public class CalculateDeductionServletTest {
6
7     CalculateDeductionServlet dect = new CalculateDeductionServlet();
8     double deduction= dect.netsalary(2000, 700, 0.06,0.05,15000);
9
10
11    double netdeduction=4350; //success
12
13    /* double netdeduction= 4000; */ // wrong calculation
14
15@
16    public void netsum() {
17        System.out.println(" Comparison of two values of deduction is : "
18            + deduction+ " "
19            + "+" + " = " +netdeduction);
20
21        assertEquals(Double.doubleToLongBits(netdeduction),
22            Double.doubleToLongBits(deduction));
23    }
24}

```

On the right, the JUnit view shows the test results:

- Finished after 0.018 seconds
- Runs: 1/1 Errors: 0 Failures: 0
- case.CalculateDeductionServletTest [Runner: JUnit 4] (0.000 s)

The console output at the bottom shows:

```

<terminated> CalculateDeductionServletTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (Mar 26, 2016, 12:00:41 PM)
Deduction values are: 4350.0
Net Salary is values are: 10650.0
Comparison of two values of deduction is : 4350.0 = 4350.0

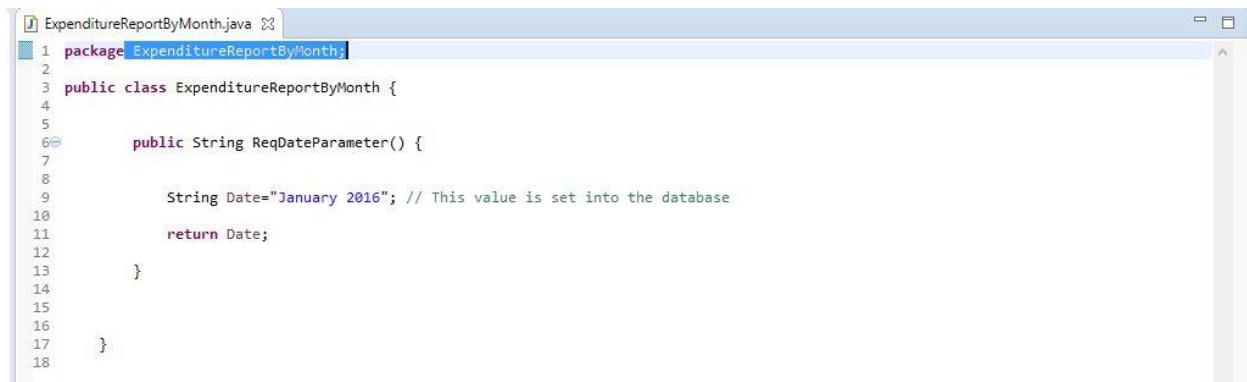
```

Fig 4.7.3 check the actual deduction value and system generated value.

Junit Test	Description	Expected result	Actual result	Time elapsed
Deduction	Calculate deduction amount and comparing with actual amount	System generated deduction value is same as actual value.	Test passed	0.000 second

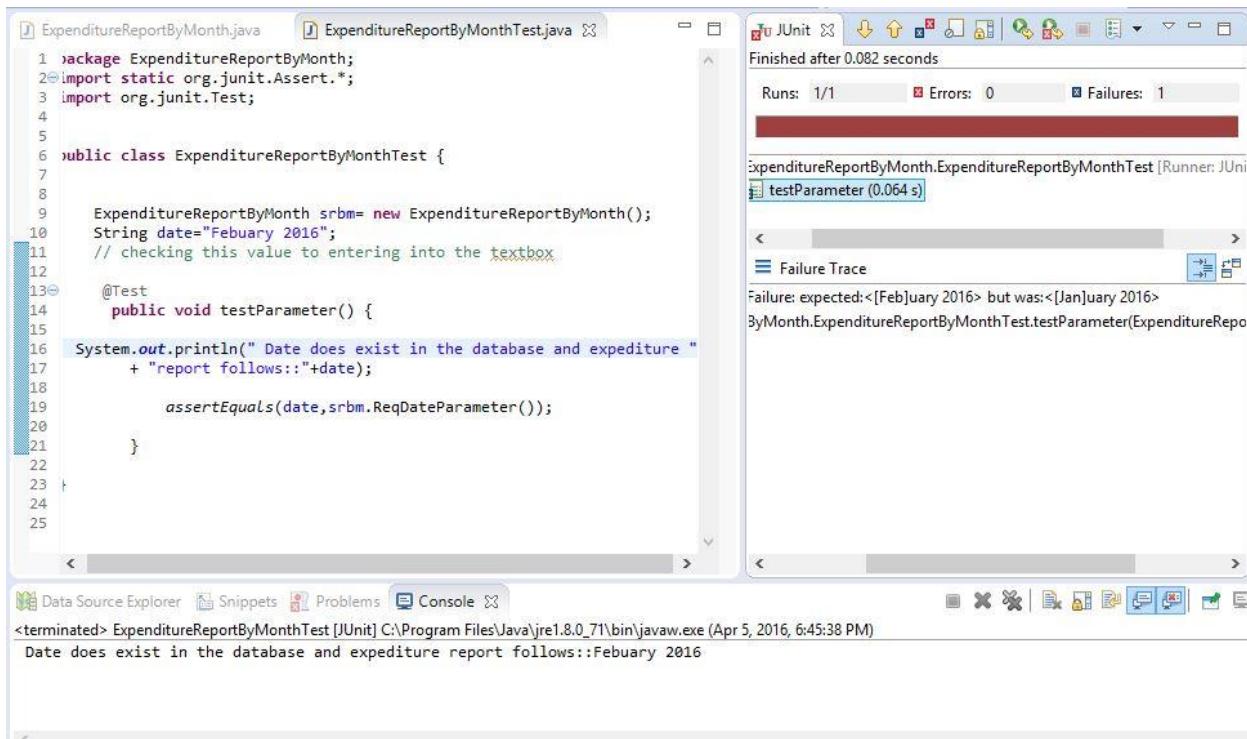
4.8.1Expenditure Report By Month

It will test net salary after all the allowance and deduction from the salary of an employee for a particular month.



```
1 package ExpenditureReportByMonth;
2
3 public class ExpenditureReportByMonth {
4
5     public String ReqDateParameter() {
6
7
8         String Date="January 2016"; // This value is set into the database
9
10        return Date;
11
12    }
13
14
15
16
17
18 }
```

Fig 4.8.1 Array value of the expenditure class for a specific month



```
1 package ExpenditureReportByMonth;
2 import static org.junit.Assert.*;
3 import org.junit.Test;
4
5
6 public class ExpenditureReportByMonthTest {
7
8
9     ExpenditureReportByMonth srbm= new ExpenditureReportByMonth();
10    String date="Febuary 2016";
11    // checking this value to entering into the textbox
12
13 @Test
14 public void testParameter() {
15
16     System.out.println(" Date does exist in the database and expenditure "
17     + "report follows::"+date);
18
19     assertEquals(date,srbm.ReqDateParameter());
20
21 }
22
23
24
25 }
```

JUnit Results:

- Runs: 1/1
- Errors: 0
- Failures: 1

Failure Trace:

Failure: expected:<[Febuary 2016> but was:<[January 2016>

ByMonth.ExpenditureReportByMonthTest.testParameter(ExpenditureRepo

Data Source Explorer Snippets Problems Console

<terminated> ExpenditureReportByMonthTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (Apr 5, 2016, 6:45:38 PM)

Date does exist in the database and expenditure report follows::Febuary 2016

Fig 4.8.2 test the a specific date

Junit Test	Description	Expected result	Actual result	Time elapsed
Expenditure ReportBy Month	Calculate the expenditure for a specific date.	Date is exist in the database for a specific employee and expenditure value is something	Test Failed (Date does not exist which is entered by the admin)	0.0064 second

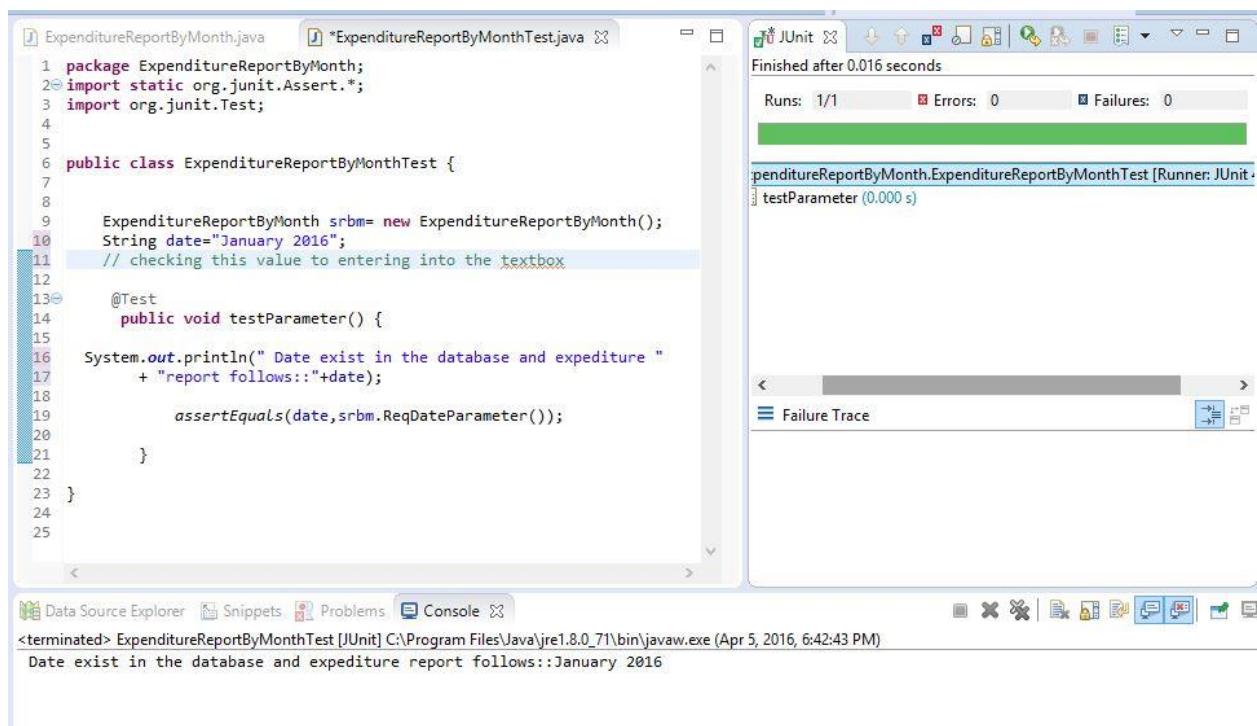


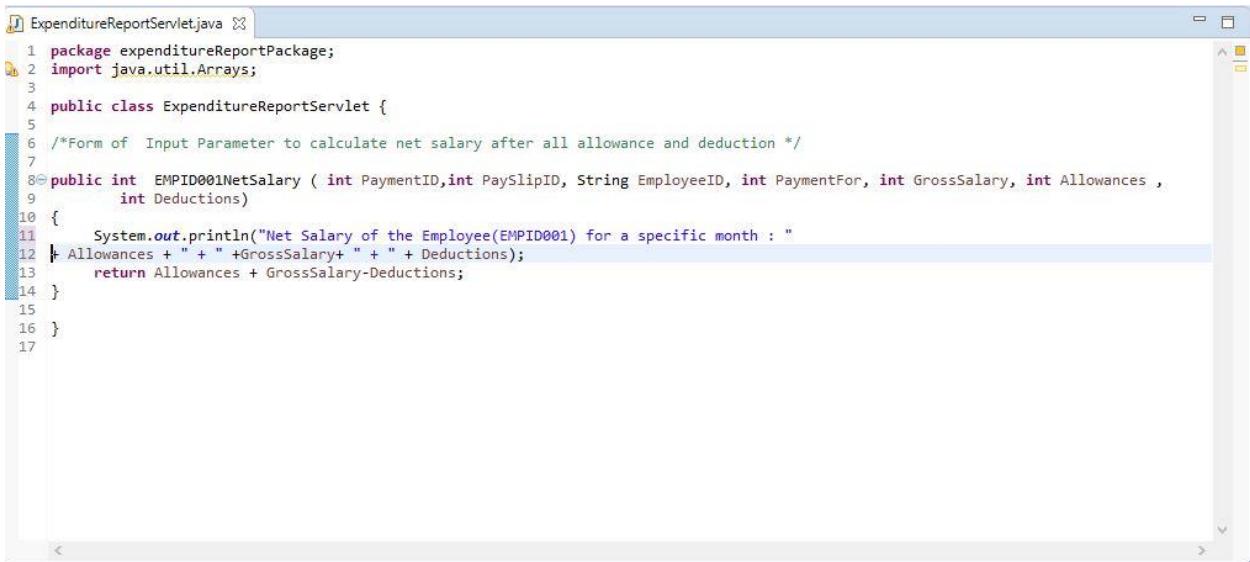
Fig 4.8.3

Test validate date

Junit Test	Description	Expected result	Actual result	Time elapsed
Expenditure ReportBy Month	Check the expenditure report of a specific date.	Date is exist in the database for a specific employee	Test passed (Date exist which is entered by the admin)	0.0064 second

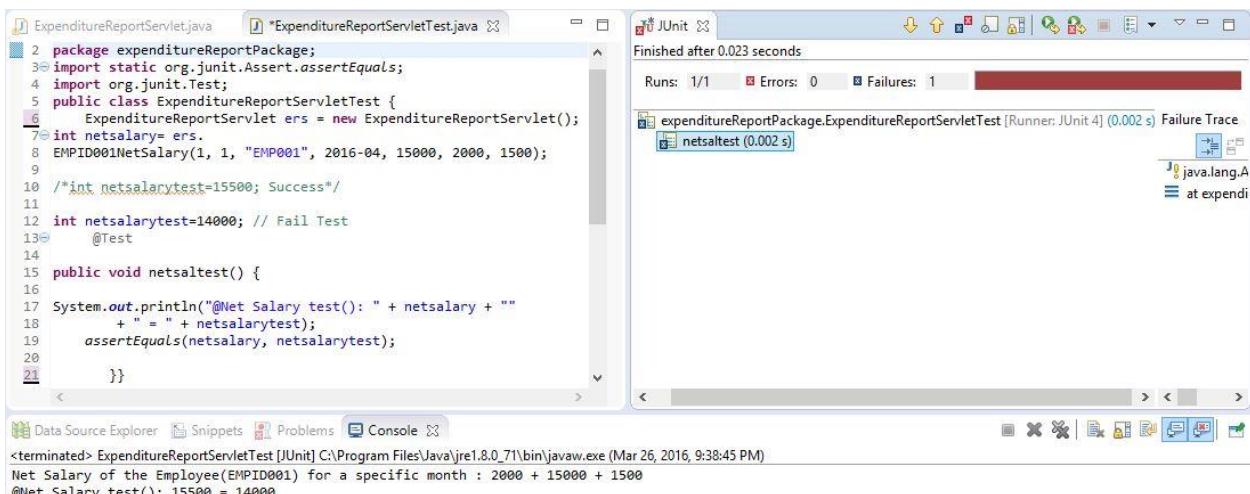
4.9.1 Expenditure Report

It will calculate the expenditure of a particular employee or all the employees.



```
ExpenditureReportServlet.java
1 package expenditureReportPackage;
2 import java.util.Arrays;
3
4 public class ExpenditureReportServlet {
5
6 /*Form of Input Parameter to calculate net salary after all allowance and deduction */
7
8@public int EMPID001NetSalary ( int PaymentID,int PaySlipID, String EmployeeID, int PaymentFor, int GrossSalary, int Allowances ,
9 int Deductions)
10 {
11     System.out.println("Net Salary of the Employee(EMPID001) for a specific month : "
12 + Allowances + " + " +GrossSalary+ " + " + Deductions);
13     return Allowances + GrossSalary-Deductions;
14 }
15
16 }
17
```

Fig 4.9.1 Array values of the expenditure



The screenshot shows an IDE interface with two tabs open: 'ExpenditureReportServlet.java' and 'ExpenditureReportServletTest.java'. The 'ExpenditureReportServletTest.java' tab contains the following code:

```
ExpenditureReportServletTest.java
2 package expenditureReportPackage;
3 import static org.junit.Assert.assertEquals;
4 import org.junit.Test;
5 public class ExpenditureReportServletTest {
6     ExpenditureReportServlet ers = new ExpenditureReportServlet();
7     int netsalary;
8     EMPID001NetSalary(1, 1, "EMP001", 2016-04, 15000, 2000, 1500);
9
10 /*int netsalarytest=15500; Success*/
11
12 int netsalarytest=14000; // Fail Test
13 @Test
14
15 public void netsaltest() {
16
17     System.out.println("@Net Salary test(): " + netsalary + ""
18         + " = " + netsalarytest);
19     assertEquals(netsalary, netsalarytest);
20
21 }
```

The 'JUnit' tab shows the test results: 'Finished after 0.023 seconds' with 'Runs: 1/1', 'Errors: 0', and 'Failures: 1'. The failure trace is for 'expenditureReportPackage.ExpenditureReportServletTest [Runner: JUnit 4] (0.002 s) Failure Trace' with a single entry 'netsaltest (0.002 s)'.

The 'Console' tab shows the output of the test execution:

```
<terminated> ExpenditureReportServletTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (Mar 26, 2016, 9:38:45 PM)
Net Salary of the Employee(EMPID001) for a specific month : 2000 + 15000 + 1500
@Net Salary test(): 15500 = 14000
```

Fig 4.9.2 test net salary

Junit Test	Description	Expected result	Actual result	Time elapsed

Expenditure Report	Comparing two salaries where one is system generated and other is actual value	Both values are same	Test Failed (system generated and actual value are different)	0.002 second
--------------------	--	----------------------	--	--------------

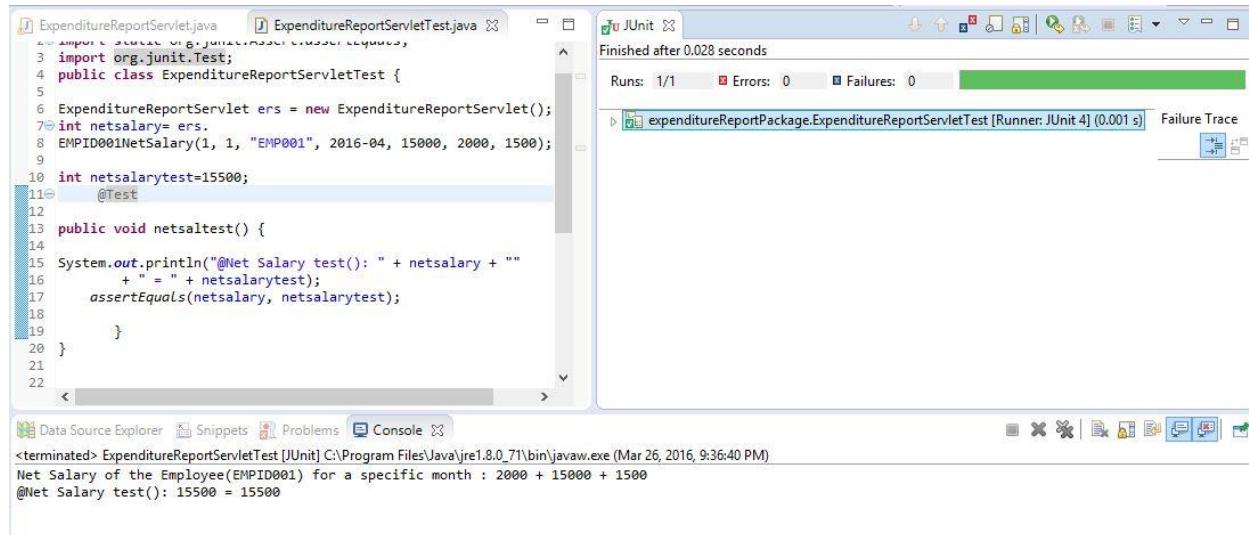
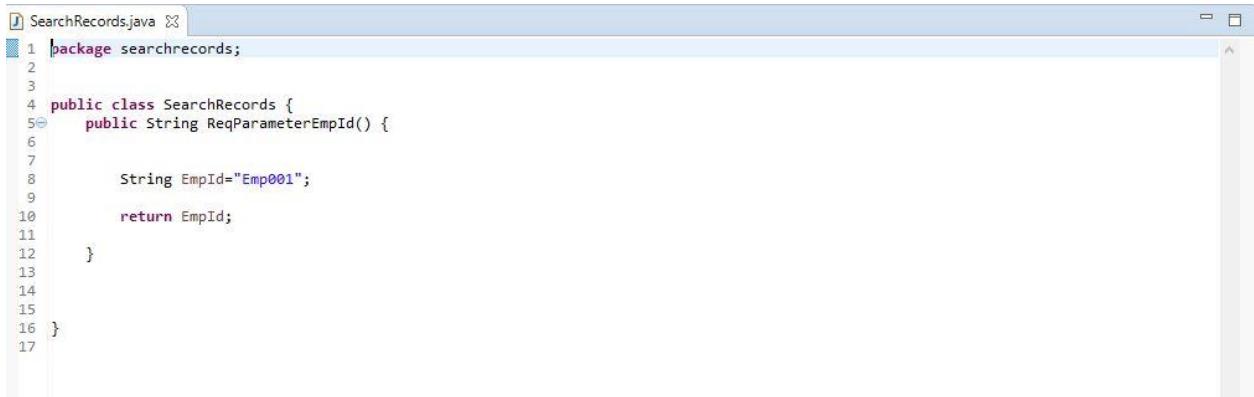


Fig 4.9.3testing two values

Junit Test	Description	Expected result	Actual result	Time elapsed
Expenditure Report	Check the expenditure report auto generated with actual net salary	Both values are same	Test Passed (system generated and actual value are same)	0.001 second

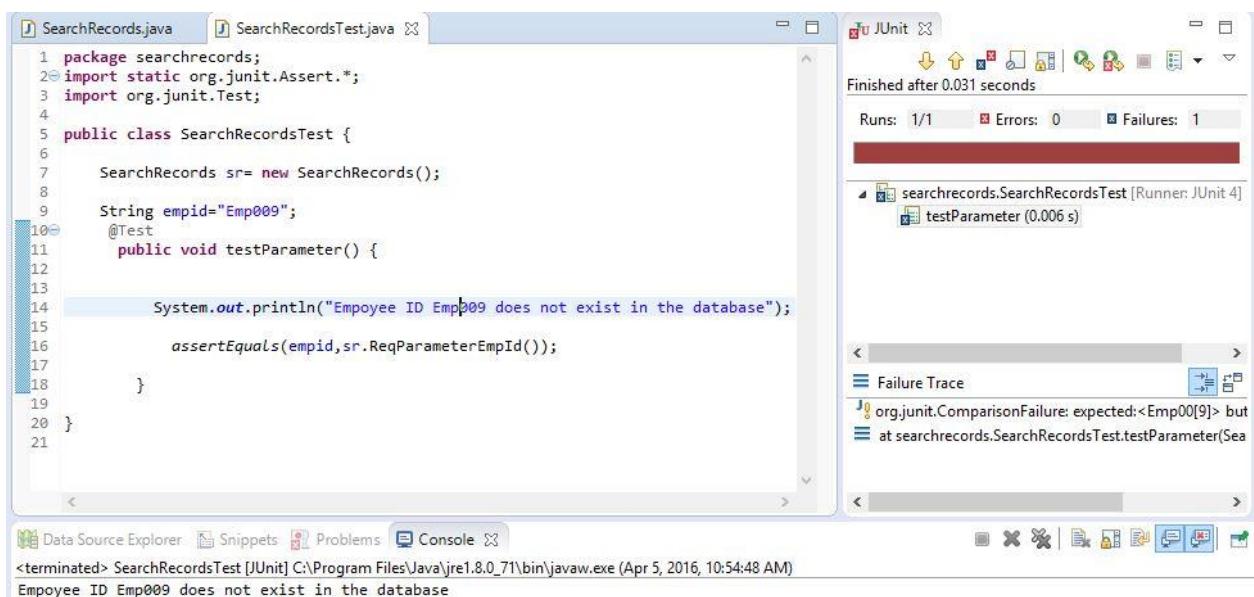
4.10.1 Search Records

It will try to find the specific employee with the help of the employee Id and will return some results.



```
1 package searchrecords;
2
3
4 public class SearchRecords {
5     public String ReqParameterEmpId() {
6
7         String EmpId="Emp001";
8
9         return EmpId;
10    }
11
12 }
13
14
15
16 }
17
```

Fig 4.10.1 Array value of the employee



```
1 package searchrecords;
2 import static org.junit.Assert.*;
3 import org.junit.Test;
4
5 public class SearchRecordsTest {
6
7     SearchRecords sr= new SearchRecords();
8
9     String empid="Emp009";
10    @Test
11    public void testParameter() {
12
13        System.out.println("Employee ID Emp009 does not exist in the database");
14
15        assertEquals(empid,sr.ReqParameterEmpId());
16
17    }
18
19
20 }
21
```

JUnit Results:

- Runs: 1/1
- Errors: 0
- Failures: 1

Failure Trace:

```
org.junit.ComparisonFailure: expected:<Emp009> but
at searchrecords.SearchRecordsTest.testParameter(Se
```

Console Output:

```
<terminated> SearchRecordsTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (Apr 5, 2016, 10:54:48 AM)
Employee ID Emp009 does not exist in the database
```

Fig 4.10.2 check the employee id

Junit Test	Description	Expected result	Actual result	Time elapsed
Search Employee Id	Check the that specific employee id exist or not	Email Id exist	Test Failed (Email Id does not exist in the database)	0.006 second

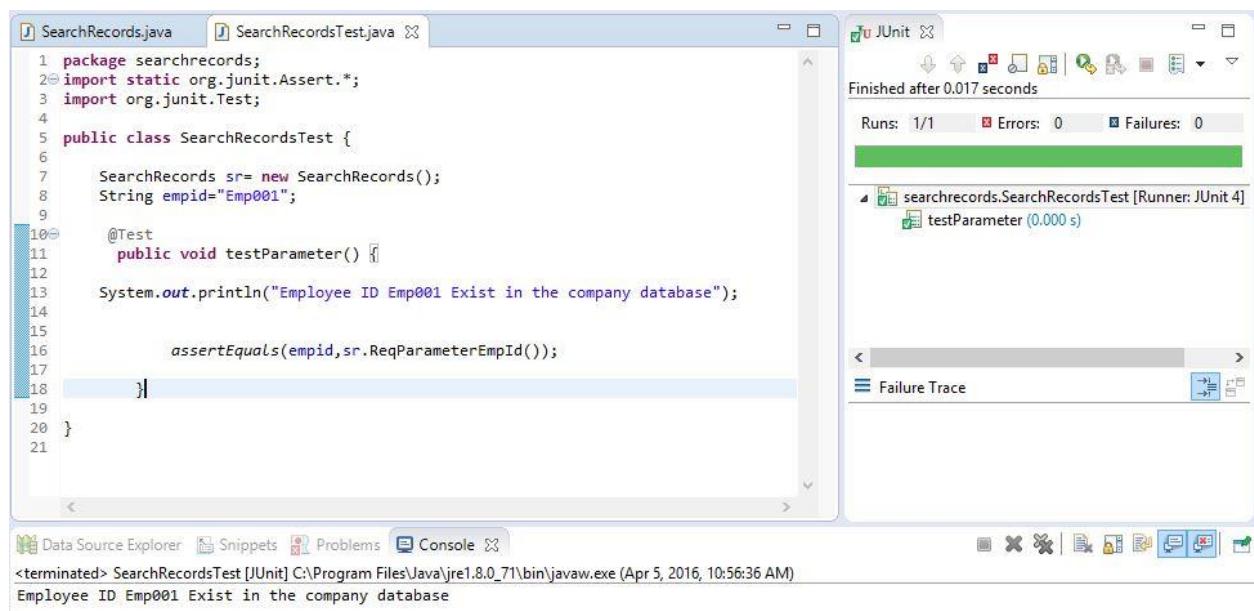


Fig 4.10.3 Test Employee Id exist or not

Junit Test	Description	Expected result	Actual result	Time elapsed
Search Employee Id	Check the that specific employee id exist or not	Email Id exist	Test Passed (Email Id exist in the database)	0.000 second

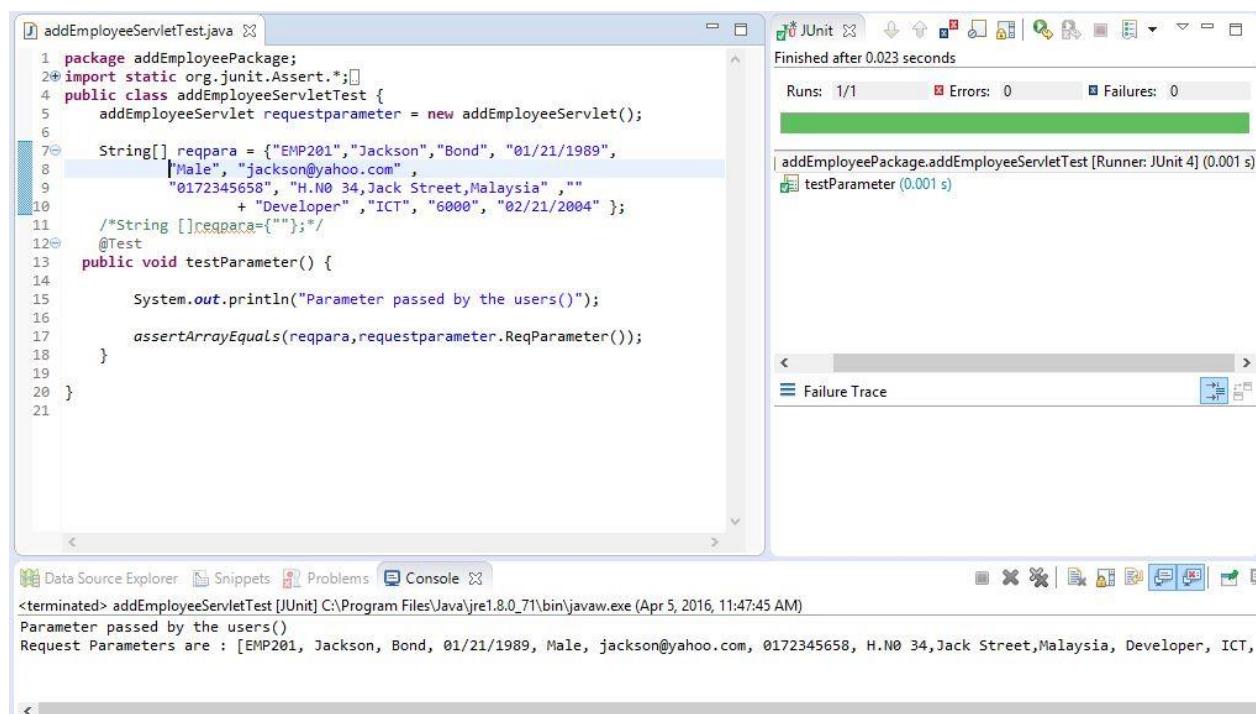
4.11.1 Update Employee

Test after the updating employee details.



```
UpdateEmployee.java
1 package UpdateEmployees;
2
3 import java.util.Arrays;
4
5 public class UpdateEmployee {
6
7     public String[] ReqParameterUpdate() {
8
9         String[] requestparameterforupdate = { "EMP201", "Jackson", "Bond", "01/21/1989", "Male", "jackson@yahoo.com" ,
10             "0172345658", "H.NO 34,Jack Street,Malaysia" , "Developer" , "ICT", "6000", "02/21/2004" };
11         // Employee Id fixed
12
13     System.out.println("Request Parameters are for update : " + Arrays.toString(requestparameterforupdate));
14
15     return requestparameterforupdate;
16
17 }
18
19 }
20
21
```

Fig 4.11.1 exist array values of the employee



```
addEmployeeServletTest.java
1 package addEmployeePackage;
2 import static org.junit.Assert.*;
3 public class addEmployeeServletTest {
4     addEmployeeServlet requestparameter = new addEmployeeServlet();
5
6     String[] repara = {"EMP201", "Jackson", "Bond", "01/21/1989",
7         "Male", "jackson@yahoo.com",
8         "0172345658", "H.NO 34,Jack Street,Malaysia" ,
9             "Developer" , "ICT", "6000", "02/21/2004" };
10
11     /*String []repara={"",""};*/
12     @Test
13     public void testParameter() {
14
15         System.out.println("Parameter passed by the users()");
16
17         assertEquals(repara,requestparameter.ReqParameter());
18     }
19
20 }
```

JUnit Results:

- Finished after 0.023 seconds
- Runs: 1/1 Errors: 0 Failures: 0
- addEmployeePackage.addEmployeeServletTest [Runner: JUnit 4] (0.001 s)
- testParameter (0.001 s)

Console Output:

```
<terminated> addEmployeeServletTest [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (Apr 5, 2016, 11:47:45 AM)
Parameter passed by the users()
Request Parameters are : [EMP201, Jackson, Bond, 01/21/1989, Male, jackson@yahoo.com, 0172345658, H.NO 34,Jack Street,Malaysia, Developer, ICT,
```

4.11.3 testing before the updating records of the employee

The screenshot shows the Eclipse IDE interface with several open windows:

- Left Window:** Displays the code for `ChangeEmployeeParamterTest.java`. The code defines a class `ChangeEmployeeParamterTest` with a single test method `testParameter` that asserts the equality of two arrays.
- Middle Window:** Shows the **JUnit** view with a summary: "Finished after 0.016 seconds", "Runs: 1/1", "Errors: 0", and "Failures: 0". Below this, it lists the test case `testParameter [Runner: JUnit 4] (0.000 s)`.
- Bottom Window:** The **Console** view displays the following output:

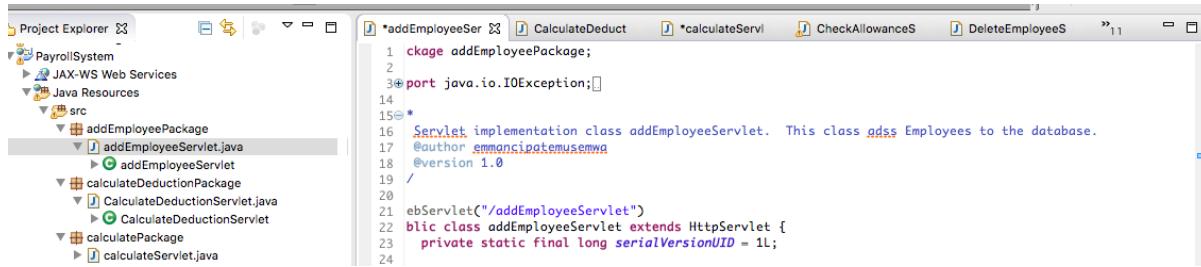

```
<terminated> ChangeEmployeeParamterTest.testParameter [JUnit] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (Apr 5, 2016, 11:38:35 AM)
Request to change input paramter by the users
Employee records succesfully updated the paramter : [EMP201, Jack, Bod, 02/12/1889, Male, jack@yahoo.com, 9987645738, H.NO 32,Jack Street,Singapore]
```

Fig 4.11.4 testing after updating some value of the employees

Junit Test	Description	Expected result	Actual result	Time elapsed
Update Records of the Employee	Update the some value of the employee	Employees record updated	Test Passed	0.000 second

5.0 Javadoc

Javadoc is a tools which is used to generate the API documentation which help during the developing an application and it is HTML format and it generates doc comments from source code. The following example of generating javadoc from the eclipse software.



```
1 package addEmployeePackage;
2
3 import java.io.IOException;
4
5 /**
6  * Servlet implementation class addEmployeeServlet. This class adds Employees to the database.
7  * @author emanmcpatmusemwa
8  * @version 1.0
9  */
10
11 @WebServlet("/addEmployeeServlet")
12 public class addEmployeeServlet extends HttpServlet {
13     private static final long serialVersionUID = 1L;
14
15     /**
16      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
17     */
18
19     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
20         // TODO Auto-generated method stub
21         // note outlines of test fields
22 }
```

5.1 Generating Java doc

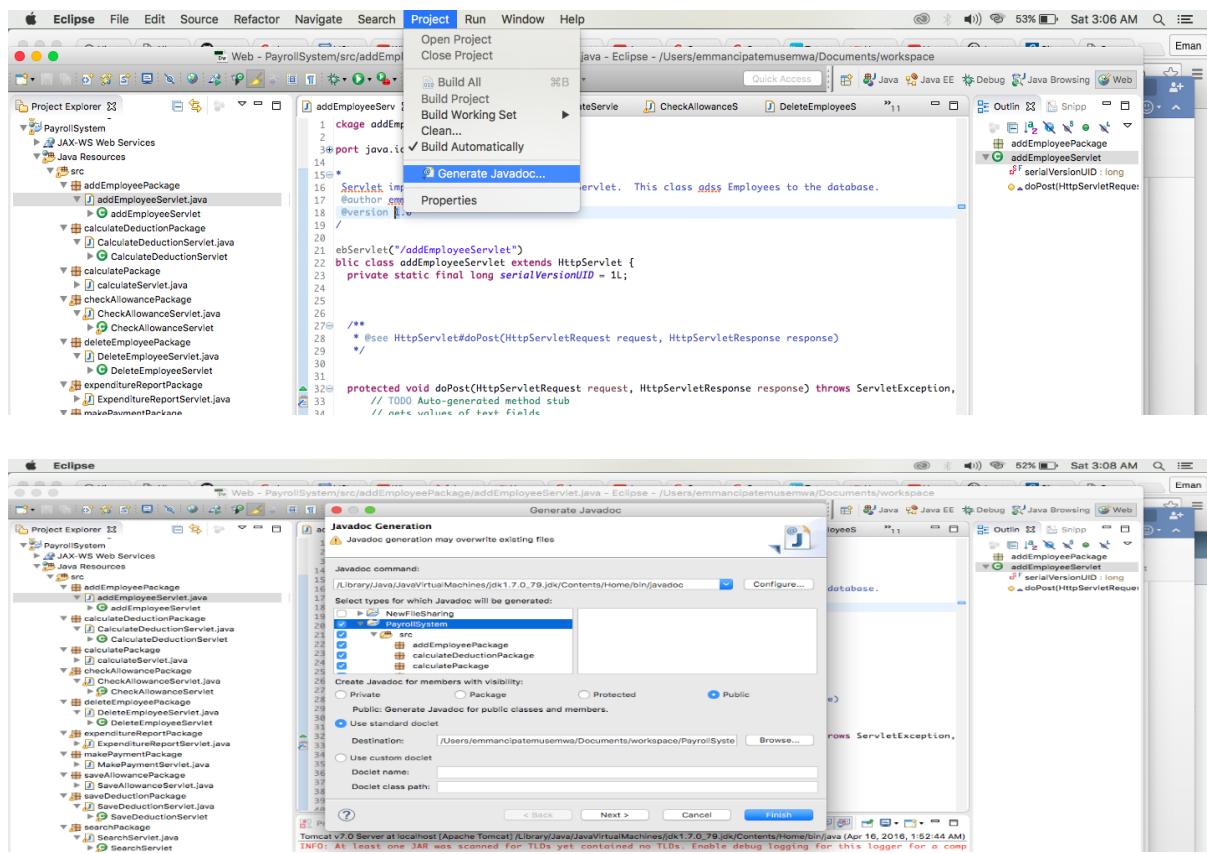


Fig 5.2 Generating Javadoc

The screenshot shows a Java development environment and a web browser. In the IDE, the left pane displays a file tree for a 'doc' directory containing various HTML files and a Java class definition for 'addEmployeePackage'. The right pane shows the Java code for the 'doPost' method of this class.

```

31
32 protected void doPost(HttpServletRequest re
33 // TODO Auto-generated method stub
34 // gets values of text fields
35
36     String emp_ID = request.getParameter("e
37     String first_name = request.getParame
38     String last_name= request.getParameter(
39     String dob = request.getParameter("dob"
40     String gender = request.getParameter("g
41     String email = request.getParameter("em
42     String phone_number = request.getParame
43     String address = request.getParameter("a
44     String desig = request.getParameter("de
45     String depart = request.getParameter("d
46     String salary = request.getParameter("s
47     String date_joined = request.getParamet
48
49
50
51     Connection conn = null;
52
53

```

In the browser, the URL is `file:///Users/emmancipatemusemwa/Documents/workspace/PayrollSystem/doc/index.html`. The page displays the generated JavaDoc for the `addEmployeeServlet` class, including its package, class hierarchy, implemented interfaces, version, author, and constructor summary.

5.3 Generated Java Doc

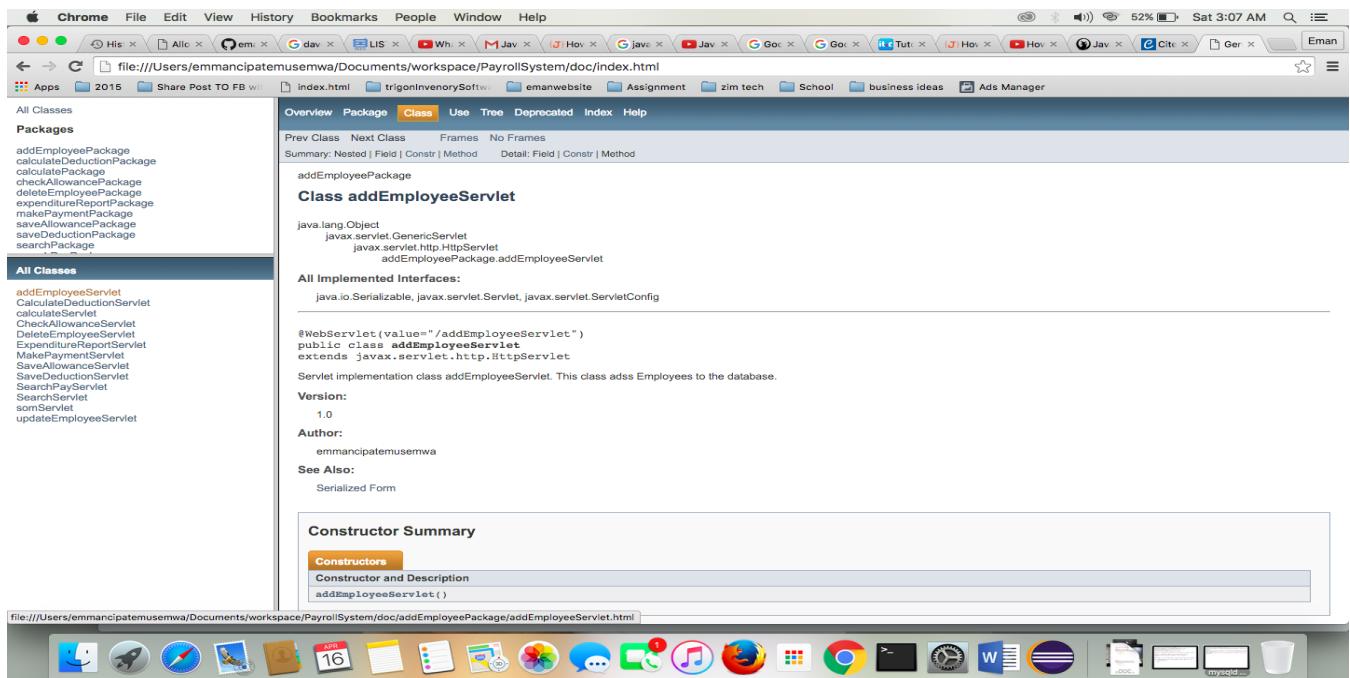
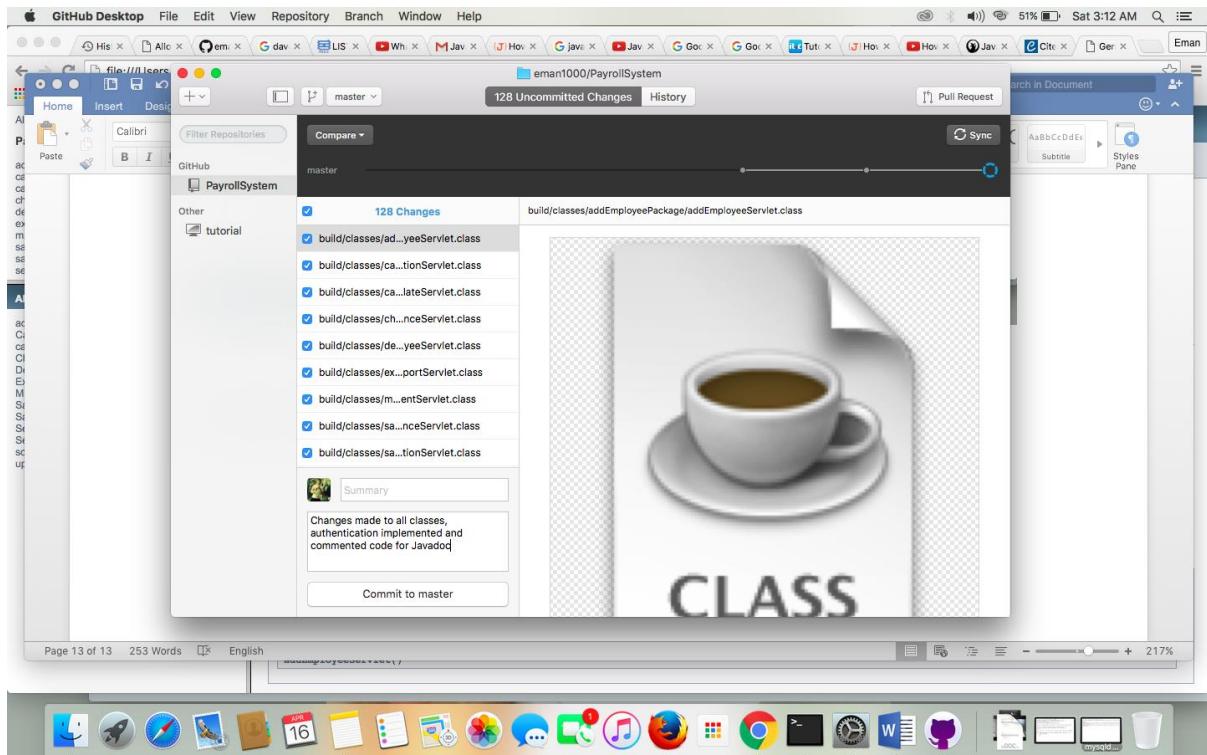


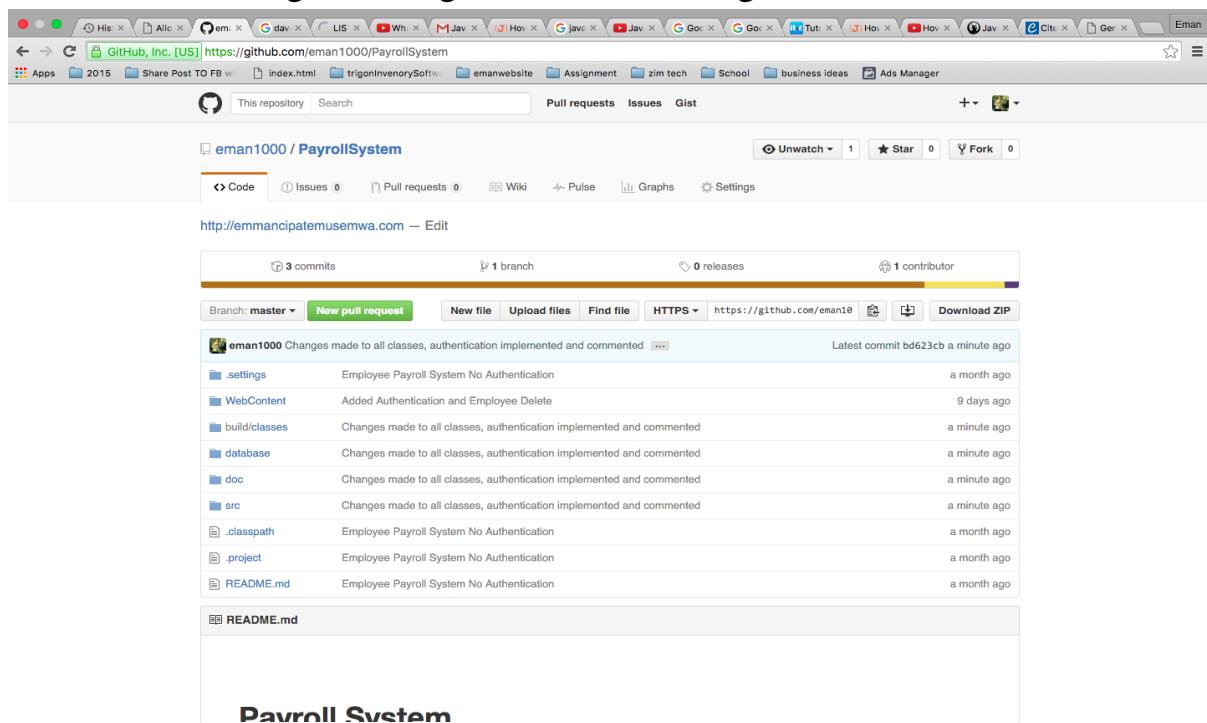
Fig 5.4 Generated java doc

6.0 Versions Control

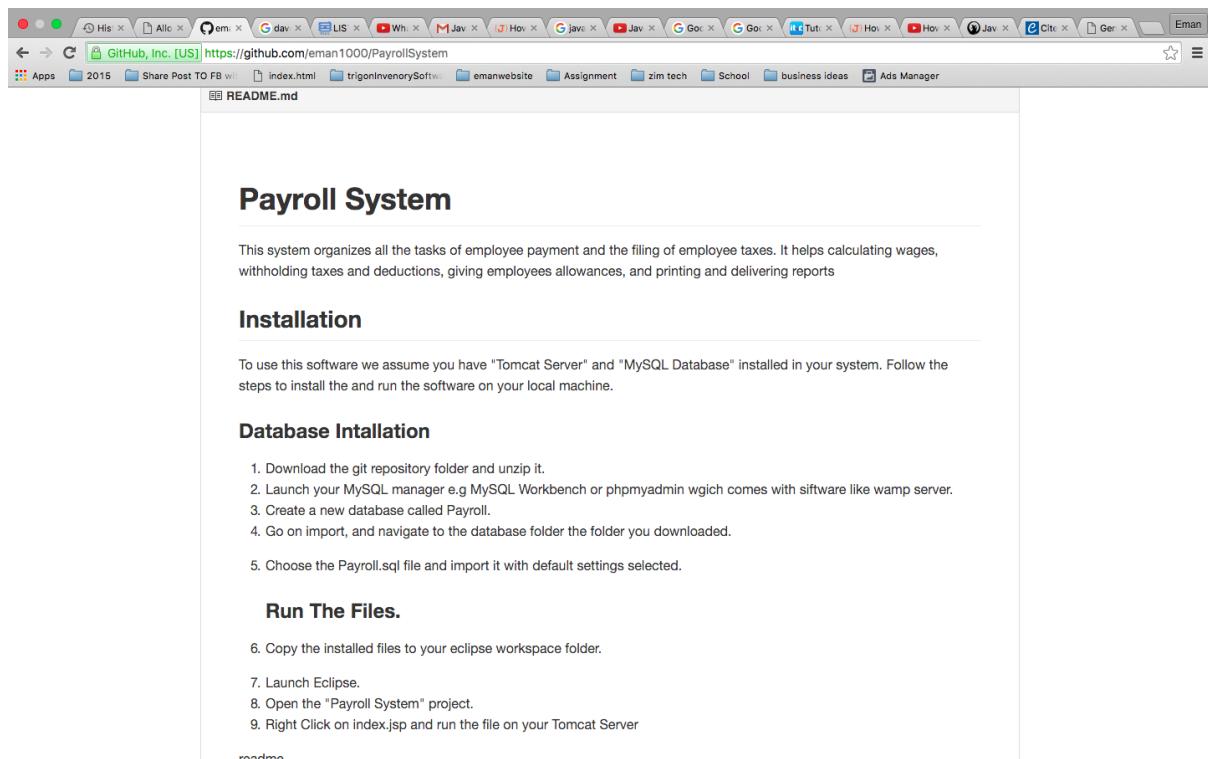
Here is the link to the version control on github <https://github.com/eman1000/PayrollSystem>
Image showing committing to github



Commits in GitHub Image showing committed files in github



6.1 Readme file in GitHub

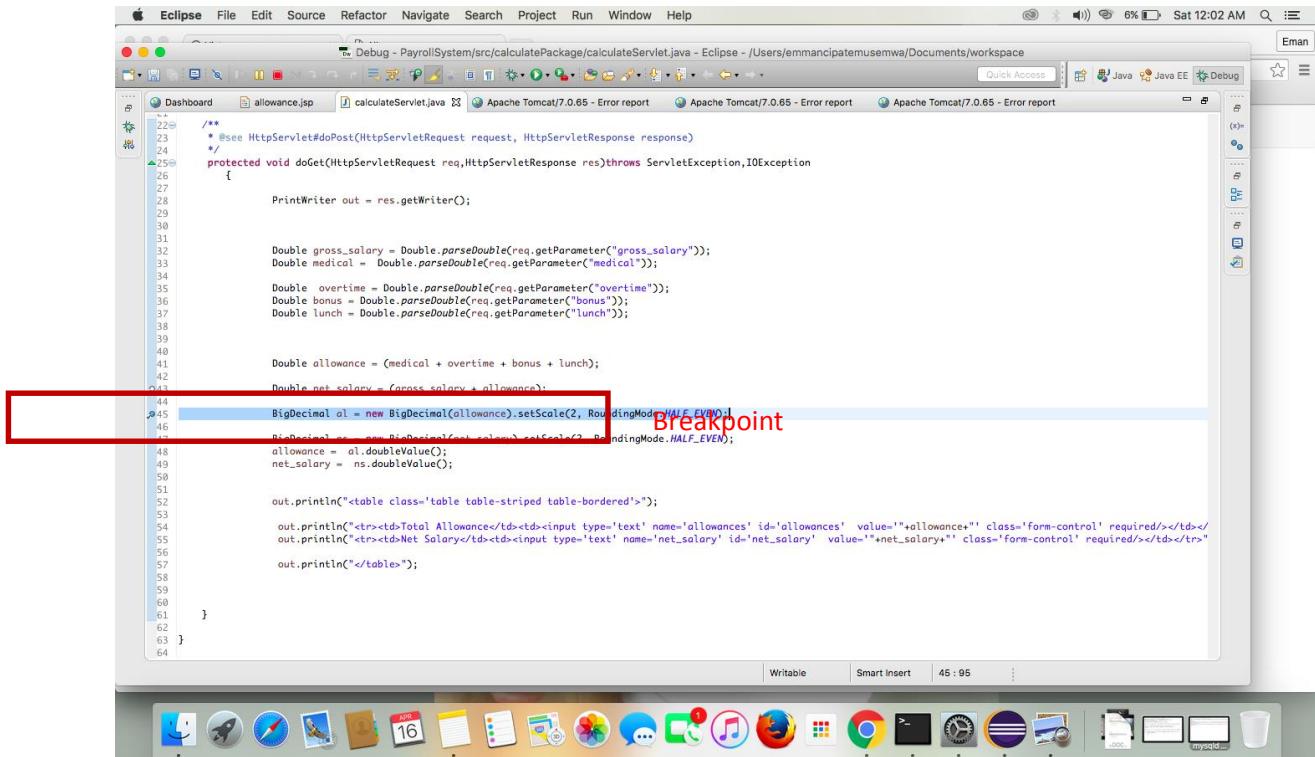


7.0 Debugging

According to (Vogel, 2013), Debugging allows you to run a program interactively while watching the source code and the variables during the execution. In this section calculateServlet.java is going to be debugged. The servlet is responsible for calculating allowance. The breakpoint or watchpoint will help analyse the values of valriables.

7.1 Breakpoint

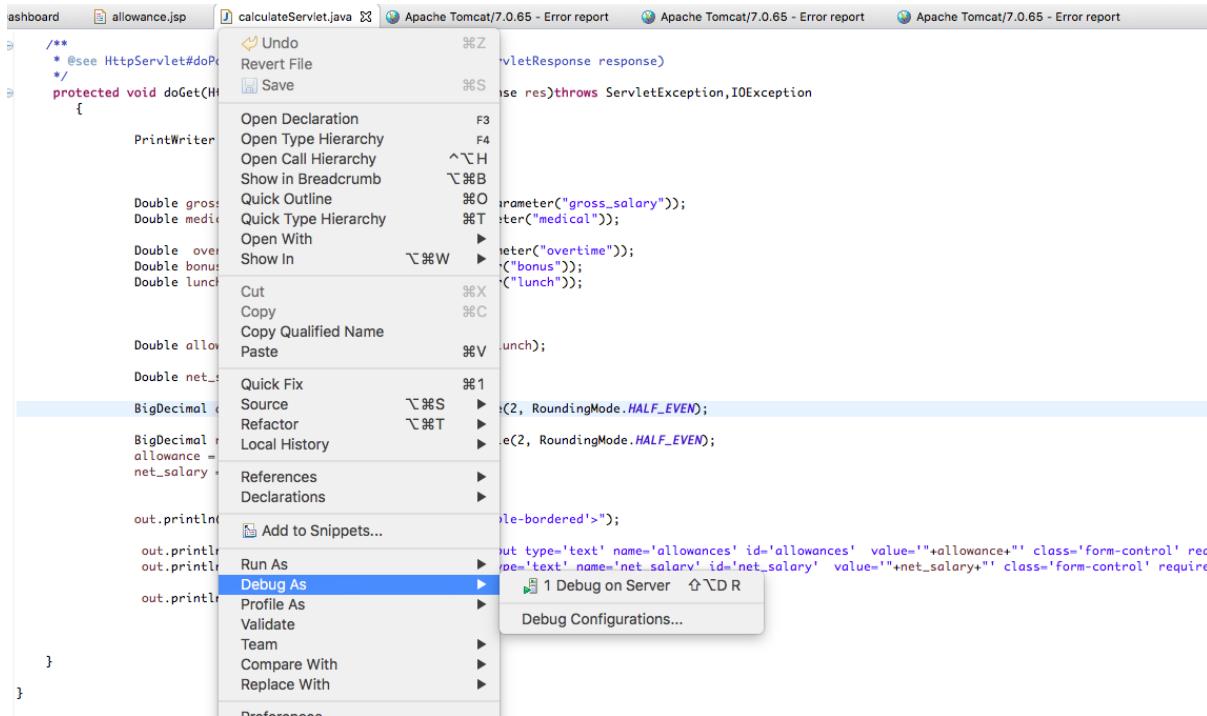
The following image shows a breakpoint specified on line 45. By breakpoints in the source code you specify where the execution of the program should stop (Vogel, 2013).



A screenshot of the Eclipse IDE interface. The title bar says "Eclipse" and the active window is "Debug - PayrollSystem/src/calculatePackage/calculateServlet.java". The code editor shows Java code for a servlet. A red rectangular box highlights line 45, which contains the line "BigDecimal ol = new BigDecimal(allowance).setScale(2, RoundingMode.HALF_UP);". The word "Breakpoint" is overlaid in red text on the right side of this line. The code includes imports for Double and BigDecimal, and various calculations involving medical, overtime, bonus, and lunch allowances.

```
22
23  /**
24  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
25  */
26 protected void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
27 {
28     PrintWriter out = res.getWriter();
29
30
31     Double gross_salary = Double.parseDouble(req.getParameter("gross_salary"));
32     Double medical = Double.parseDouble(req.getParameter("medical"));
33
34     Double overtime = Double.parseDouble(req.getParameter("overtime"));
35     Double bonus = Double.parseDouble(req.getParameter("bonus"));
36     Double lunch = Double.parseDouble(req.getParameter("lunch"));
37
38
39
40     Double allowance = (medical + overtime + bonus + lunch);
41
42     Double net_salary = (gross_salary + allowance);
43
44
45     BigDecimal ol = new BigDecimal(allowance).setScale(2, RoundingMode.HALF_UP);
46     BigDecimal ns = new BigDecimal(net_salary).setScale(2, RoundingMode.HALF_EVEN);
47
48     allowance = ol.doubleValue();
49     net_salary = ns.doubleValue();
50
51
52     out.println("<table class='table table-striped table-bordered'>");
53
54     out.println("<tr><td>Total Allowance</td><td><input type='text' name='allowances' id='allowances' value='"+allowance+"' class='form-control' required></td></tr>");
55     out.println("<tr><td>Net Salary</td><td><input type='text' name='net_salary' id='net_salary' value='"+net_salary+"' class='form-control' required></td></tr>");
56
57     out.println("</table>");
58
59
60
61 }
62
63 }
```

Now that a break point has been specified we can now debug the java file as follows

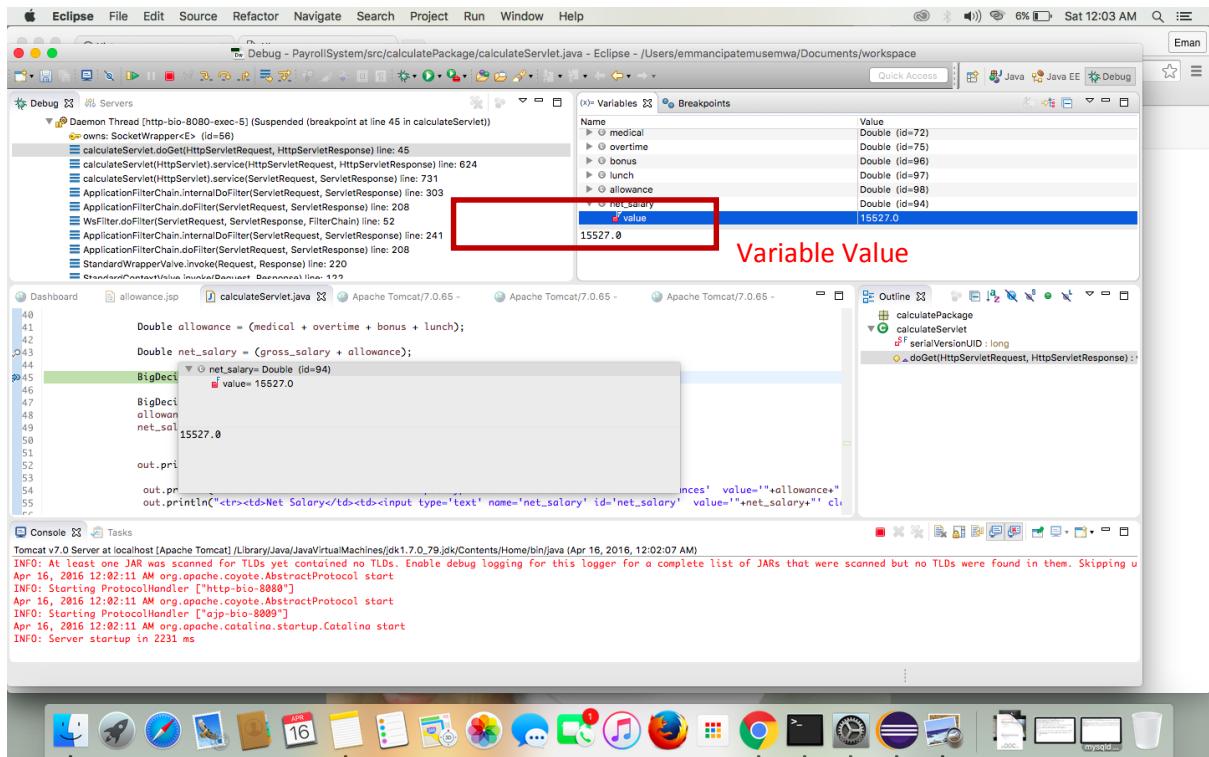


Once the debug is running we can use the system as normal, when the breakpoint specified is reached the program will stop giving us the current values of variables.

The screenshot shows a web browser window with multiple tabs open. The active tab is 'localhost:8080/PayrollSystem/SearchServlet'. The page displays 'Search Records' and 'Employee Details' sections. The 'Employee Details' section shows Employee ID: Emp001, First Name: Emmancipate, Last Name: Musemwa, and Salary In Ringgits: 15000. The 'Allowance' section shows a table with rows for Salary, Overtime, Medical, Lunch, and Bonus, with amounts 15000, 23, 100, 4, and 400 respectively. A 'Calculate' button is at the bottom of the allowance table. The browser's toolbar and status bar are visible at the top and bottom of the window.

Type	Amount
Salary	15000
Overtime	23
Medical	100
Lunch	4
Bonus	400

Once the calculate button is clicked, the calculate servlet will be run and the results are as follows.



7.2 Agile Approach

Agile methodology is the new trend in the software industry to develop a software and also popular due to its flexibility and adaptability. Agile model is a methodology to develop a software so it is also known as agile software development life cycle and it is the combination of the iterative and incremental process and it is used to deliver the software to the customers quickly. There are some strategy to work with Agile, here a product is divided into small fragments and each fragments have their life cycle. For example, planning, analysis, design, coding, unit testing etc. and the each iteration customers and stakeholder can see the project and also can give some suggestion.

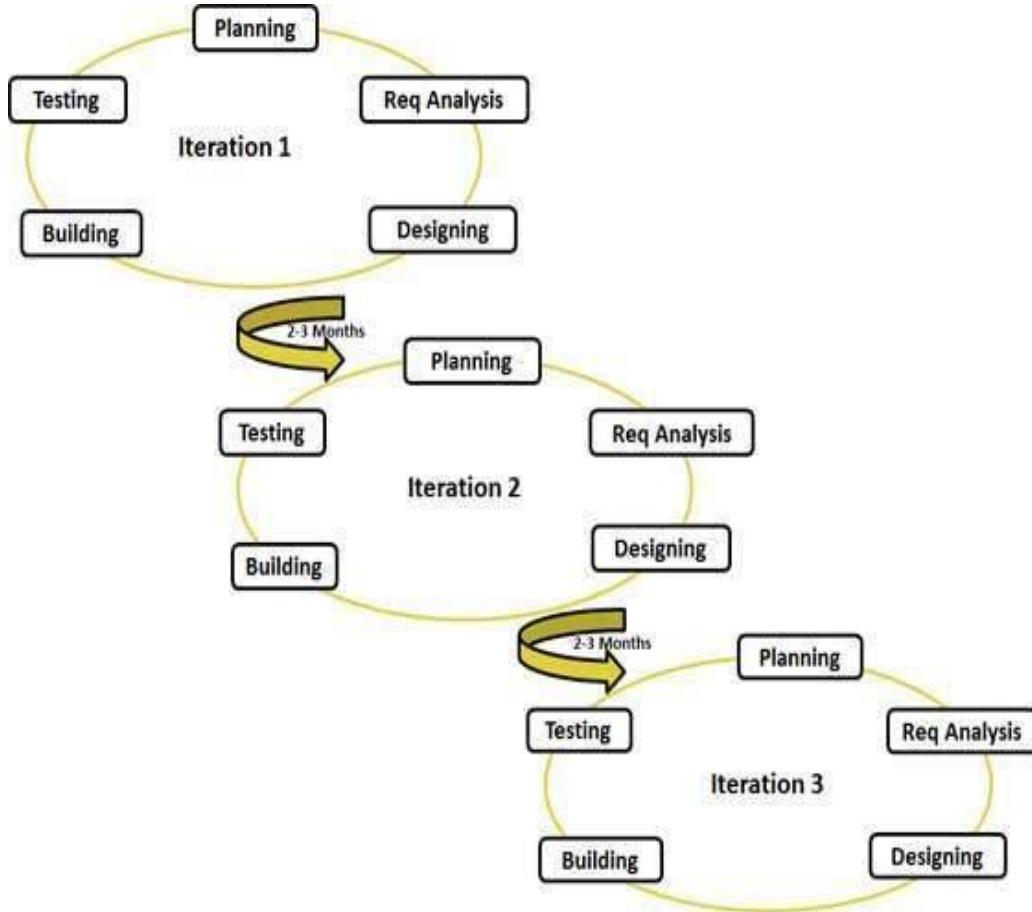


Fig 7.2 Agile SDLC (tutorialspoint, 2016)

Every project is based on some circumstances so during the development of the project will be needed some existing methodology with different form or new method. In the Agile approach, every part of the completion of a project is fixed date.

7.3 Comparing Agile Approach with the Traditional Approach

Agile Software Development Life Cycle (SDLC) follows the different strategy to develop the software comparing the traditional SDLC. For example, waterfall model. Waterfall model follows the predictive approach to the development and required full documentation and planning. In this approach, full information needed from the customers before developing the software which is called the requirements analysis of a project as well planning. The most

beneficial part of this model is that everything is documented based so if in the future any changes needed it can be changed easily with the help of documentation but the agile approach is different but fast. here there is no need all the requirements from the customers at a time and also no such a planning is needed for the development of a software .Customers can come at any time and can go through different stages of a project if they think any changes needed it will be changed and it is based on adaptive approach. Customer interaction is important in this model but the disadvantage of this model is that there is no documentation so if there is any changes needed in the future so same team member will be needed who have developed this instead of the new developer.

7.3.1 Following are the Agile principles are:

- **Individuals and interactions** - In this approach self-organization and motivation are important because it is team work so everyone has to consider as best member of a team.
- **Working software** – Working Software is the best demo to presenting as a part of the project in front of the customer so customers can understand easily instead to write everything in the document.
- **Customer collaboration** –The continuous interaction is important with customers during the developing of a software to full fill all the requirements instead of the beginning of the software.
- **Responding to change** – Every part of the project is completing with different team so if any changes will be needed the customer can go to that a specific team and can approach some changes in a particular section and it will be changed quickly.

The most popular agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001 (tutorialspoint, 2016).

8.0 Conclusion

Technology has given us a lot of advantages which can be used for different work and its make a life easier also save much money and time. The flexibility of internet has changed the users' mind and now they can interact with the bank easily and can do any type of the transaction. The IBM Payroll System also based same principle of the internet banking which also deals all transaction related to the employees. This system is more flexible with employees and can show all the transaction within a minute and this separate section of the company. It has more demand in the future to save the money in every transaction.

Coding Standards

(Mytton, 2004) stated in his article that best applications are coded properly. He then went on to explain that “properly” does not only mean the code does its job, but it is also easy to maintain, add to and debug. Coding standards document what developers should do while writing there code. Instead of following individual ways of coding all developers will follow a standard way. Other developers who look t the code know what to expect through the entire application (Mytton, 2004).

Commenting

It is very fundamental to comment on code. (Mytton, 2004) stated that Looking through unfamiliar code is much easier if it is laid out well and everything is neatly commented with details that explain any complicated constructs and the reasoning behind them. Below is an example of commented code for the IBM Payroll System. The “//” characters are used for single line comments in java and they have been used throughout the project to explain blocks of code

```
53
54     String message = null; // message will be sent back to client
55
56     try {
57         // 1. Get a connection to database
58         DriverManager.registerDriver(new com.mysql.jdbc.Driver());
59         conn = DriverManager.getConnection(
60             "jdbc:mysql://localhost:3306/Payroll", "root", "123456");
61
62
63         // constructs SQL statement
64
65         String sql = "INSERT INTO employees_tb (emp_ID, first_name, last_name, gender, dob, email, phone_number
66             + "values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
67         PreparedStatement statement = conn.prepareStatement(sql);
68         statement.setString(1, emp_ID);
69         statement.setString(2, first_name);
70         statement.setString(3, last_name);
71         statement.setString(4, gender);
72         statement.setString(5, dob);
73         statement.setString(6, email);
74         statement.setString(7, phone_number);
75         statement.setString(8, address);
76         statement.setString(9, design);
77         statement.setString(10, depart);
78         statement.setString(11, salary);
79         statement.setString(12, date_joined);
80
81
82         // sends the statement to the database server
83         int row = statement.executeUpdate();
84         if (row > 0) {
85             message = "Employee Data Saved in the Database";
86         }
87     } catch (SQLException ex) {
88         message = "ERROR: " + ex.getMessage();
89         ex.printStackTrace();
90     } finally {
91         if (conn != null) {
92             // closes the database connection
93             try {
94                 conn.close();
95             } catch (SQLException ex) {
96                 ...
97             }
98         }
99     }
100 }
```

Code Paragraphs and indentation

Blocks of code should be separated into paragraphs for easy readability. Once the end of a paragraph is reached, just like in a word document, one can easily know that a new function or activity is starting. In the image below code paragraphs and indentation can be clearly seen in use. If an external programmer looks into this code it would be easy for him/her to understand it.

```
33 // gets values of text fields
34 String payslip_id = request.getParameter("payslip_id");
35 String emp_ID = request.getParameter("emp_ID");
36 String payment_for = request.getParameter("payment_for");
37 String gross_salary = request.getParameter("gross_salary");
38 String allowances = request.getParameter("allowances");
39 String deductions = request.getParameter("deductions");
40 String net_salary = request.getParameter("net_salary");
41
42
43 //Connect to the database
44 Connection conn = null;
45 String message = null; // message will be sent back to client
46 try {
47     // 1. Get a connection to database
48     DriverManager.registerDriver(new com.mysql.jdbc.Driver());
49     conn = DriverManager.getConnection(
50         "jdbc:mysql://localhost:3306/Payroll", "root", "123456");
51
52     // constructs SQL statement
53     String sql = "INSERT INTO payments (payslip_id, emp_ID, payment_for, gross_salary, allowances, deduction
54     + "values (?, ?, ?, ?, ?, ?)";
55     PreparedStatement statement = conn.prepareStatement(sql);
56     statement.setString(1, payslip_id);
57     statement.setString(2, emp_ID);
58     statement.setString(3, payment_for );
59
60     statement.setString(4, gross_salary);
61     statement.setString(5, allowances);
62     statement.setString(6, deductions);
63     statement.setString(7, net_salary);
64
65     // sends the statement to the database server
66     int row = statement.executeUpdate();
67     if (row > 0) {
68         message = "Payment Made to the Employee";
69     }
70 } catch (SQLException ex) {
71     message = "ERROR: " + ex.getMessage();
72     ex.printStackTrace();
73 } finally {
74     if (conn != null) {
75         // closes the database connection
76     }
77 }
```

Functions and Variables Naming

Variables and functions ought to be named consistently throughout the code. They should also be named according to the function they are performing. They should briefly describe the data they contain (Mytton, 2004). The image below shows variables declared according to the data they contain.

```
//variable declaration|
Double advance_pay = Double.parseDouble(req.getParameter("advance_pay"));
Double leaves = Double.parseDouble(req.getParameter("leaves"));
Double gst = Double.parseDouble(req.getParameter("gst"));
Double other_tax = Double.parseDouble(req.getParameter("other_tax"));
Double gross_salary = Double.parseDouble(req.getParameter("gross_salary"));Double gvt = (gross_salary *
Double otherT = (gross_salary * other_tax);
Double deduction = (advance_pay + leaves + otherT + gvt);
Double net_salary = (gross_salary - deduction);
```

The screenshot below shows a class declaration which is named exactly according to its function.

```
18 */
19 //Deduction Calculation Class
20 @WebServlet("/CalculateDeductionServlet")
21 public class CalculateDeductionServlet extends HttpServlet {
22     private static final long serialVersionUID = 1L;
23
24
```

Reference

- [1] Vogel, L. (2013). Java Debugging with Eclipse - Tutorial. [online] Vogella.com. Available at: <http://www.vogella.com/tutorials/EclipseDebugging/article.html> [Accessed 15 Apr. 2016].
- [2] Visual-paradigm.com. (n.d.). Class Diagram - UML Diagrams - Unified Modeling Language Tool. [online] Available at: <https://www.visual-paradigm.com/VPGallery/diagrams/Class.html> [Accessed 15 Apr. 2016].
- [3] tutorialspoint. (2016, April 17). Retrieved from <http://www.tutorialspoint.com/>: http://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm
- [4] Mytton, D. (2004). Why You Need Coding Standards. [online] SitePoint. Available at: <http://www.sitepoint.com/coding-standards/> [Accessed 22 Apr. 2016].