ONLINE EMPLOYEE EXPENSE MANAGEMENT APPLICATION

Navya Palawancha
B.E., Jawaharlal Nehru Technological University, India, 2007

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2011
ONLINE EMPLOYEE EXPENSE MANAGEMENT APPLICATION

A Project

by

Navya Palawancha

Approved by:

__________________________, Committee Chair
Dr. Jinsong Ouyang, Ph.D.

__________________________, Second Reader
Dr. Scott Gordon, Ph.D.

__________________________
Date
Student: Navya Palawancha

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

__________________________, Graduate Coordinator

Dr. Nikrouz Faroughi

Department of Computer Science
Abstract

Of

ONLINE EMPLOYEE EXPENSE MANAGEMENT APPLICATION

By

Navya Palawancha

Statement of the Problem: In today’s world, fast growing companies like ACG Inc., with employees working for several clients in different parts of the country need a better online solution for managing employee expenses. In addition, these companies also look solutions that are easy to use without compromising on efficiency.

Conclusions Reached: This project provides a one-stop solution for managing employee expenses. The web-based application allows employees located in different areas in the country to login and file for their expenses with ease. With integrated email notifications & alerts, this application is a more efficient and effective way to track employee expenses. In addition, the managers can also access employee expenses in one place and can accept or deny employee expense report thereby speeding the employee expense process.

_______________________, Committee Chair
Jinsong Ouyang, Ph.D

_______________________
Date
ACKNOWLEDGMENTS

I would like to thank Dr. Jinsong Ouyang, my project advisor for his guidance and support throughout the course of this project. I am also thankful to Dr. Scott Gordon, second reader of this project, for his time to review and revise this project.

My special thanks to Dr. Nikrouz Faroughi for his advise throughout my master degree and for his support and help in making this project possible.

Lastly, I would like to thank the entire faculty and staff of the Computer Science Department, California State University, Sacramento.
**TABLE OF CONTENTS**

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
</tr>
<tr>
<td>List of Tables</td>
</tr>
<tr>
<td>List of Figures</td>
</tr>
</tbody>
</table>

Chapter

1. **INTRODUCTION** ................................................................. 1
   1.1 Background ................................................................. 1
   1.2 Problem Statement ...................................................... 1
   1.3 Existing Solutions ....................................................... 2
   1.4 Goal of the Project ..................................................... 3

2. **REQUIREMENTS ANALYSIS** .................................................. 4
   2.1 Product Perspective .................................................... 4
   2.2 Product Features ....................................................... 4
   2.3 Product Usage ......................................................... 10
   2.4 Operating Environment ............................................... 11
   2.5 Assumptions and Dependencies .................................... 13
   2.6 Functional Requirements ........................................... 13
   2.7 External Interface Requirements .................................. 51
   2.8 Non-Functional Requirements ..................................... 52

3. **SYSTEM DESIGN** ............................................................ 54
   3.1 .NET Framework ...................................................... 54
   3.2 Building Web-Based Application .................................. 56
   3.3 .NET Application Architecture .................................... 57
   3.4 High Level Design ..................................................... 59

4. **DATABASE DESIGN** ........................................................ 62
4.1 About SQL Server 2005 ................................................................. 62
4.2 Database Normalization .............................................................. 62
4.3 Database Tables and Database Diagram ................................. 65
4.4 Sample of Database Scripts ....................................................... 75

5. IMPLEMENTATION DETAILS .................................................. 77
   5.1 Development of Web Interface using ASP.NET ......................... 77
   5.2 Implementation details of an Employee Expense ..................... 78
   5.3 Implementation details of Adding New Expense ..................... 80
   5.4 ADO.NET Architecture and Application to the project ............. 81
   5.5 Business Logic Implementation using C#.NET ......................... 90
   5.6 Implementation details for adding different types of expenses .. 92

6. THE RESULT .............................................................................. 97

7. CONCLUSION ............................................................................. 113

References ..................................................................................... 115
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Table 4.1 Employee Table</td>
<td>67</td>
</tr>
<tr>
<td>2.</td>
<td>Table 4.2 Login Details Table</td>
<td>68</td>
</tr>
<tr>
<td>3.</td>
<td>Table 4.3 Employee Expense Table</td>
<td>70</td>
</tr>
<tr>
<td>4.</td>
<td>Table 4.4 Roles Table</td>
<td>71</td>
</tr>
<tr>
<td>5.</td>
<td>Table 4.5 Employee Expense Categories Table</td>
<td>72</td>
</tr>
<tr>
<td>6.</td>
<td>Table 4.6 Expense Type Table</td>
<td>73</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1.</td>
<td>Figure 2.1 Use Case Notations</td>
<td>15</td>
</tr>
<tr>
<td>2.</td>
<td>Figure 2.2 Use Case Diagram</td>
<td>16</td>
</tr>
<tr>
<td>3.</td>
<td>Figure 2.3 New Employee Registration</td>
<td>18</td>
</tr>
<tr>
<td>4.</td>
<td>Figure 2.4 Manage Employees</td>
<td>19</td>
</tr>
<tr>
<td>5.</td>
<td>Figure 2.5 First-time employee login</td>
<td>21</td>
</tr>
<tr>
<td>6.</td>
<td>Figure 2.6 User Authentication</td>
<td>23</td>
</tr>
<tr>
<td>7.</td>
<td>Figure 2.7 Manage Roles</td>
<td>24</td>
</tr>
<tr>
<td>8.</td>
<td>Figure 2.8 Admin Expense Management</td>
<td>26</td>
</tr>
<tr>
<td>9.</td>
<td>Figure 2.9 Employee Expense Management</td>
<td>28</td>
</tr>
<tr>
<td>10.</td>
<td>Figure 2.10 Manage Profile</td>
<td>29</td>
</tr>
<tr>
<td>11.</td>
<td>Figure 2.11 Add Airline Expense</td>
<td>32</td>
</tr>
<tr>
<td>12.</td>
<td>Figure 2.12 Add Car Rental Expense</td>
<td>34</td>
</tr>
<tr>
<td>13.</td>
<td>Figure 2.13 Add Hotel Expense</td>
<td>36</td>
</tr>
<tr>
<td>14.</td>
<td>Figure 2.14 Add Misc. Expense</td>
<td>39</td>
</tr>
<tr>
<td>15.</td>
<td>Figure 2.15 View Expense History</td>
<td>40</td>
</tr>
<tr>
<td>16.</td>
<td>Figure 2.16 Edit/Delete Expense</td>
<td>42</td>
</tr>
<tr>
<td>17.</td>
<td>Figure 2.17 Submit Expense</td>
<td>44</td>
</tr>
<tr>
<td>18.</td>
<td>Figure 2.18 View Employee Expenses</td>
<td>45</td>
</tr>
<tr>
<td>19.</td>
<td>Figure 2.19 Update Expense Status</td>
<td>47</td>
</tr>
<tr>
<td>20.</td>
<td>Figure 2.20 Expense Reports</td>
<td>49</td>
</tr>
<tr>
<td>21.</td>
<td>Figure 2.21 Update Expense Status</td>
<td>51</td>
</tr>
<tr>
<td>22.</td>
<td>Figure 3.1 .NET Framework Components</td>
<td>54</td>
</tr>
<tr>
<td>23.</td>
<td>Figure 3.2 Client-Server Architecture</td>
<td>57</td>
</tr>
<tr>
<td>24.</td>
<td>Figure 3.3 Block Diagram</td>
<td>59</td>
</tr>
<tr>
<td>25.</td>
<td>Figure 3.4 Class Diagram</td>
<td>61</td>
</tr>
<tr>
<td>26.</td>
<td>Figure 4.1 Entity Relationship Diagram</td>
<td>74</td>
</tr>
</tbody>
</table>
27. Figure 5.1 ADO.NET Data Architecture ................................................. 82
28. Figure 6.1 Expense Management Home Page ......................................... 98
29. Figure 6.2 Login Error ............................................................................ 99
30. Figure 6.3 First time login – Change Password ....................................... 100
31. Figure 6.4 Main Page ............................................................................ 101
32. Figure 6.5 Employee Profile ..................................................................... 102
33. Figure 6.6 Employee Expense Page ....................................................... 103
34. Figure 6.7 Add New Expense ................................................................. 104
35. Figure 6.8 Auto Generated Email to Confirm Expense ............................. 105
36. Figure 6.9 Employee Expenses ............................................................... 105
37. Figure 6.10 View Reports ....................................................................... 106
38. Figure 6.11 Finance Manager ................................................................. 107
39. Figure 6.12 Admin Main Webpage ......................................................... 108
40. Figure 6.13 Add Employee .................................................................... 109
41. Figure 6.14 Manage Employee .............................................................. 110
42. Figure 6.15 Manage Roles .................................................................... 111
43. Figure 6.16 Manage Expenses .............................................................. 112
Chapter 1

INTRODUCTION

This chapter introduces the project, outlines the background to the problem, identifies the problem owner, and gives an outline of the project objectives.

1.1 Background

American Consulting Group, Inc is a software consulting firm that offers a wide range of solutions from consulting to implementing customized solutions to its clients. The firm helps clients re-engineer and re-invent their business processes that give the clients the competitive edge in their market space.

1.2 Problem Statement

Fast growing consulting companies like ACG Inc., have increasing number of employees working for several clients in different parts of the country. One of the important tasks for such a firm is to manage expenses for employees located in different parts of the country. The current system that is in place for expense management at this firm is a manual process. Employees fill out a monthly expense form and send it for approval along with expense receipts. Some of the potential problems identified with the existing method are as follows

*Slow & Inefficient* There is no one application that manages the entire expense process. This leads to a very slow and in-efficient way to manage the expenses.
Error prone A lot of steps in the current process involve manual work like data entry and manually verifying the expense details. Due to this there are high chances for errors which slow the entire expense process.

Communication issues There is no proper communication during the entire expense approval process. It is important that the employee is updated with the current expense status faster.

1.3 Existing Solutions

There are existing software solutions to manage employee expenses that could potentially address the issues raised in the earlier section. These existing solutions however have issues that are discussed below.

Complexity Most of the existing software solutions address a broad and general issues pertaining to financials of a company. Usually such software would be big, feature rich & complex. For a small size company most of the features from such software would not be relevant.

Cost The initial and the operating costs of such existing software are huge.

No single software There is a need for a single web-based solution that would address the entire requirements specific & focused for the company. To address this issue, lean & focused software needed to be developed to address only the requirements specific for this company instead of using multiple software solutions.
Along with the issues mentioned above, the client also needed a software solution that is flexible, easy to maintain & also integrate to the existing system with complete control to the software solution for future enhancements.

1.4 Goal of the Project

This project is being developed for “American Consulting Group” with the primary goals as follows:

**Improve efficiency** The firm wishes to improve the efficiency by automating the entire employee expense process.

**Improve communication** The firm requires the capability to better communicate with the employees by providing faster updates to the expenses using email alerts.

**Reduce manual errors** The firm requires an online application that covers the entire process of employee expense management eliminating majority of manual work to remove un-intended human errors.

**Ease of use** The online application should be easy to use for the employees with minimal training.
Chapter 2

REQUIREMENTS ANALYSIS

2.1 Product Perspective

This project is developed for American Consulting Group Inc., a Software Consulting firm that has considerable number of employees working for different clients around the country. The intent is for these employees to use this web-based product for filing their employee expenses, and also manage their employee profile. This product will become a one-stop solution for easy and faster approval of employee expenses. This product should also be user-friendly, quick to learn and reliable for the above purpose.

This project is intended to be a stand-alone product and should not depend on the availability of any other software. It should run on Windows based platform and should be independent of web browser.

2.2 Product Features

This project provides various functionalities including managing the employee profiles, adding/editing expenses, track expense status, review past expenses, email notifications, etc. The whole project is developed in 4 levels depending on the user access levels. This project supports the following access levels.

- Employee Access
- Manager Access
- Administrator Access
• Finance Manager Access

2.2.1 Employee Access This level of access is given to individual employees that will enable update of certain fields in employee profile and will also allow adding, editing, and deleting their own expense reports.

• The system should allow employees to edit/update employee profile. The Employee profile should include basic employee information along with bank details.

• The system should allow the employee to change login password.

• The system should allow the employees to review all the expenses filed in the past.

• Employees are expected to file expenses on monthly basis; however the system should be flexible to allow employees to file for expenses for any duration.

• The employees are expected to file different types of expenses per week. The system should allow the user to name these monthly expenses and should send one report to the manager when the expense is filed. The system should be easy to use and should allow the employee to see all the expenses based on the expense type. The system should show the expense total based on the expense type along with total per day.
• If the “Receipt Included” field is required, the employee should be provided with an option to upload the scanned image of the receipt with options to update/delete the image.

• The system should automatically calculate the total cost limit, if the expense is filed for more than one day.

• The system should send automatic email to the employee’s manager when a new expense is filed.

• The system should allow the employee to review the current status of all the expenses that were filed in the past.

2.2.2 Manager Access This level of access is given to Managers so that they can view the expense reports filed by the employees reporting to them. Managers will have additional permissions when compared to the employee access.

• Manager can submit expenses similar to the employee as described earlier.

• The system should allow the manager to review all the expenses filed by his/her employee.

• The system should allow the manager to lookup a particular employee in order for him to review the employee’s expenses.

• The system should allow the manager to “Accept” or “reject” an expense report along with reason if the manager rejects an expense.
• The system should automatically send email to the employee when the expense report is accepted or rejected.

• The system should allow the manager to generate web based reports.
  
  o The filters that should be used to generate these reports are
  
  ▪ All employees or lookup a particular employee
  
  ▪ Duration i.e. between two dates
  
  ▪ Status of the expenses reports

The system should also provide the manager an option to export the data in the above mentioned reports into excel sheet or PDF format.

2.2.3 Administrative Access An admin is a “super user” for the company and will have all administrative permissions. This access level will allow editing employee & manager profiles.

• The system should allow the admin to add/delete employees & Managers.

• The system should allow the admin to “activate” or “de-activate” an employee. An employee that is currently inactive should not be allowed to login.

• The system should allow the admin to setup different types of roles in the company. This feature should be flexible to add any number of roles.

• The system should allow the admin to assign/change a manager to certain employee.
• The system should allow the admin to setup different types of expenses.

• The system should allow the admin to setup cost limits to different types of expenses.

2.2.4 Finance Manager Access This user access is exclusively for reviewing the “approved” reports filed by either employees or managers.

• The system should allow the user with this access level to view all the “approved” or “reimbursed” expense reports filed by employees or managers

• The system should allow this user to update the status of the expense report to “Reimbursed” status along with “reimbursed date” and “check” or “direct deposit” information.

• The system should allow the finance manager to generate web based reports.
  
  o The filters that should be used to generate these reports are
    ▪ All employees or lookup a particular employee
    ▪ Duration i.e. between two dates
    ▪ Status of the expenses reports

The system should also provide the finance manager an option to export the data in the above mentioned reports into excel sheet or PDF format.
2.2.5 The functionalities in scope are as follows

1. The system should allow for first time user to login with temporary password and should instruct the user to change password upon login.

2. Employee profile will display employee, personal & bank details. There will be options available to edit/update certain fields in the profile.

3. This web-based application will allow both employees & managers to file for the following type of expenses.

   - Hotel expense
   - Airline expense
   - Rental car expense
   - Miscellaneous expense

   As the employees are expected to file expenses on monthly basis, all these types of expenses could be filed together under a certain “expense name”.

4. Each expense report should have the information for total expense per day along with total expense per expense type including the overall total expense.

5. Save employee expense and come back to it later before submitting the final expense report.

6. View and print the past employee expenses but cannot edit any “accepted” expense reports.
7. View expense reports filed by all employees or lookup a particular employee under a manager. The manager can retrieve detailed information on a particular expense filed by employee.

8. Manager should be able generate several useful reports that will enable him to easily manage his budget.

9. Manager can export the data from these reports into either excel or pdf format.

10. Email interface will allow easy access to employee email inbox in the web application.

11. Email notification system alerts the user & approving manager about a new expense report. This will enable quick response from the manager to approve/reject the filed expenses.

12. Admin with “super user” access can edit/delete or add an employee or manager.

13. Finance Manager’s role is to process all the approved expense and update the expense report when the amount has been reimbursed with appropriate status along with method of reimbursement i.e. either check or direct deposit.

### 2.3 Product Usage

Employees are expected to file for expenses on monthly basis. The “employee expense” system should be flexible enough to allow the users to file expenses on monthly basis along with expenses for short duration of time. When an employee files his/her expense, the employee should be able to track the current status by logging into the system. The user with “manager” access should be able to see expenses filed by employees that report
to him. The manager should be able to see complete details of each expense in order to thoroughly review it and make a decision to approve or reject the expense.

2.4 Operating Environment

2.4.1 Active Server Pages Web application provides a user friendly interaction with the system. Active Server Pages (ASP), also known as Classic ASP or ASP Classic, was Microsoft's first server side script engine for dynamically generated web engines [9]. An ASP page is basically a HTML page that can have server side scripts. These pages are then processed by a Web Server before it’s sent to the browser. Server-side scripts are executed when a browser requests for an .asp file from a web browser. Web Server processes the requested .asp file and it then scans the whole file and executes the script commands in the file. Web server then formats a web page for the script and sends it to web browser to display it [3].

The working of an active server page in a normal scenario is as follows [9]

- User requests an ASP page through the Internet browser by typing the web address of the asp page.
- The asp page is not directly sent to the user instead it is first processed by the Server.
- Server connects to the database and saves any data requested by the user or access data from the database for the user.
- Finally the html result generated by the IIS is sent back to the user through the web browser.
In this project ASP is used to create web application pages that interact with the system to provide an abstract layer to the user, therefore hiding all the details and just providing a user friendly environment. All the Active Server pages in this project are written in VBscript. Following are some of the important asp pages created to develop the web application.

**For Admin**

- Home.aspx
- Login.aspx
- RegisterUser.aspx
- Roles.aspx
- VerifyExpenses.aspx

**For Employee**

- EmpAddExpenses.aspx
- EmpExpCategories.aspx
- EmpExpenses.aspx
- Frm_changePwd.aspx

**For Manager**

- ManageReports.aspx
- TeamExpenses.aspx
- ManagersExpenses.aspx
Since the system will be implemented in Microsoft ASP.NET technology, the software will need to be hosted on an ASP.NET-compatible site. The system will also require one SQL database to be installed on the host space, as well as any additional software required sending email to users of the system. The system must be completely compatible with any browser that fully supports Microsoft ASP.NET technology.

2.4.2 IIS  
Internet Information Services (IIS) - formerly called Internet Information Server - is a set of Internet-based services for servers created by Microsoft for use with Microsoft Windows OS. IIS is a freeware which can be downloaded from a Microsoft website. ASP pages can be best run with the IIS server. IIS provides access to a large number of audiences and the main reason is that this application works with almost every browser and operating systems. Any end-user who wants to run this application can run by using a simple browser and don’t have to worry about installing software on their computers. The Web application developed in this project run its ASP pages on the IIS [10].

2.5 Assumptions and Dependencies

1. It is assumed that the system will be developed using the ASP.NET technology.

2. It is assumed that the system will be able to interface with an email server in order to send the alerts & notifications by email to the contacts in the system.

3. It is assumed that the system will interface with a SQL Server 2005 database.

2.6 Functional Requirements

The functional requirements of this project are described using use cases.
2.6.1 Use Cases

Use case modeling is a technique used to describe what the new software being developed should do or what the already available software does. It is used in the Analysis phase of software development to express clearly the high-level requirements of the software. Use case modeling helps to ensure that it captures the actual requirements of the customer for whom the system is being developed. Often, software projects fail when the developers do not understand customer’s needs [7].

2.6.1.1 Actor
An actor is user of the system. An actor is someone or something, which interacts with the system. An actor need not be a human being; it can be any application. An actor interacts with the system by sending or receiving messages.

2.6.1.2 Use Case
The functionality provided by the system is represented as a use case. It is typically described in verb form.

2.6.1.3 Use Case Diagram
The use case diagram for the Employee Expense system is shown in Figure 2.1. The major actors of the this system includes the new users, who are trying to register with the expense system; the registered users/employees, who already have an account; and the system administrator, who has full privileges on the system [7].
Figure 2.1 Use Case Notations
Figure 2.2 Use Case Diagram
UC1 - New User Registration

This use case describes how a new employee is registered to use the expense tracking system. This registration is done by user with “administrative” access.

Actors

New Employee, Administrator, System

Preconditions

The following preconditions must be true before the initiation of the use case.

1. User with “Admin” access logged into the system
2. New employee details are available.

Basic Flow

The following steps represent the basic flow of events occurring from the home page also shown in Figure 2.3 below.

1. Admin User selects the appropriate link from the home page.
2. User selects “Add Employee” and enters relevant new employee information.
3. User provides a Username & temporary password that could be used by the new user.
4. User selects “save” after entering the new user information.

Post Conditions

The following post conditions must be true after the completion of the use case.

1. System updates the database with new user information along with temporary password.
**UC2 - Manage Employees**

This use case describes how an admin can activate or deactivate an employee from logging into the system.

**Actors**

Employee, Administrator, System

**Preconditions**

The following preconditions must be true before the initiation of the use case.

1. User with “Admin” access logged into the system

**Basic Flow**

The following steps represent the basic flow of events occurring from the home page also shown in Figure 2.4 below.

---

**Figure 2.3 New Employee Registration**
1. Admin User selects the appropriate link from the home page.

2. User selects “Manage Employees”

3. System should display the list of employees along with their current status.

4. User should select “Activate” or “Deactivate” to change the status of the employee

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. System should allow login access to active employees while deny login access to inactive employees.

![Diagram showing the process of managing employees](image-url)

**Figure 2.4** Manage Employees
**UC3 - First-time User login**

This use case describes how a new employee can login to Employee Expense website from the Internet. Every new employee is assigned a temporary password that enables the first time login to setup the account.

**Actors**

New Employee, Client, System

**Preconditions**

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. New employee should have an assigned temporary password.

**Basic Flow**

The following steps represent the basic flow of events occurring from the home page also shown in Figure 2.5 below.

1. User selects the appropriate link from home page.
2. System determines that the user is a new employee.
3. System routes to the “Change password” page and will prompt the user to update the password. System should not allow the user to navigate the pages without changing the password.
4. System should logout to allow user to log back in with the updated password.

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. User has logged into the application with updated password
2. User is able to view/select various applications depending on access level of the user.

![Flowchart](image)

**Figure 2.5** First-time employee login

**UC4 - User Authentication**

This use case describes how to log into the Employee Expense website from the Internet.

Each employee at the consulting firm is assigned with login credentials.

**Actors**

Employee, Manager, Admin, Finance Manager, Client, System

**Preconditions**

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.
Basic Flow

The following steps represent the basic flow of events occurring from the home page also shown in Figure 2.6 below.

1. User selects the appropriate link from the home page.
2. System determines User's role from login.
4. System displays applicable pages associated with User's access level.

Post Conditions

The following post conditions must be true after the completion of the use case.

1. User has logged into the application.
2. User is able to view/select various applications depending on access level of the user.
UC5 - Manage Roles

This use case describes how user with “Admin” access level can manage different roles.

Actors

Admin, System

Preconditions

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.
**Basic Flow**

The following steps represent the basic flow of events occurring from the home page also shown in Figure 2.7 below.

1. User selects the appropriate link from the home page.
2. System determines User’s Admin role from login.
3. Admin selects “Manage Roles”
4. System should allow the “admin” to add/edit or delete “Role”.

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. User has logged into the application.
2. Updates made to the “Roles” are reflected in the database.

![Diagram](image)

**Figure 2.7 Manage Roles**
UC6 - Admin Expense Management

This use case describes how user with “Admin” access level can manage expenses.

**Actors**

Admin, System

**Preconditions**

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.

**Basic Flow**

The following steps represent the basic flow of events occurring from the home page also shown in Figure 2.8 below.

1. User selects the appropriate link from the home page.
2. System determines User's Admin role from login.
3. Admin selects “Manage Expenses”
4. Admin can add new “expense type” by providing Name, Approval limit & Expense status.
5. Admin can enable or disable expense type by editing the expense and selecting “enable/disable”
6. Admin can update the expense type with “approval limit” information
7. Admin can also delete a particular expense type by selecting “delete”

**Post Conditions**

The following post conditions must be true after the completion of the use case.
1. User has logged into the application.

2. New expense types added by admin should be saved in the database

3. Any updates to existing expense type should be reflected in the database.

**Figure 2.8 Admin Expense Management**

**UC7 - Employee Expense Management**

This use case describes employee expense management by either employee or a manager that can file expense reports.

**Actors**

Employee, Manager, System
Preconditions

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.

Basic Flow

The following steps represent the basic expense management activities. Expense screen also shown in Figure 2.9 below.

1. User navigates to the home page and selects My Expense
2. User can add a new expense by choosing a name for the expense.
3. When the user select “Add” the system should show the new expense name in the list of expenses filed by the user
4. User can add all his/her expenses by selecting the newly added expense name.
5. User can follow the use cases described below to add different types of expenses under one expense name.
6. User can view, edit or delete a particular expense name by selecting appropriate button.

Post Conditions

The following post conditions must be true after the completion of the use case.

1. User is able to add new expense name.
2. User is able to view, edit or delete the details in a particular expense.
UC8 - Manage Employee profile

This use case describes employee profile management by either employee or a manager that can file expense reports.

**Actors**

Employee, Manager, System

**Preconditions**

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.

2. System has validated user credentials entered in the Login Page.
Basic Flow

The following steps represent the basic expense management activities. Expense screen also shown in Figure 2.10 below.

1. User navigates to the home page and selects My Profile

2. User can update relevant information in the employee profile that includes personal, contact & bank details.

3. User can save the changes by selecting “Save”

Post Conditions

The following post conditions must be true after the completion of the use case.

1. User is able to update information in employee profile

Figure 2.10 Manage Profile
UC9 - Add a new Airline expense

This use case describes how to add new airline expense

Actors

Employee, Manager, System

Preconditions

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.
3. User has created an “Expense Name” under which the current airline expense could be added.

Basic Flow

The following steps represent the basic flow of events in Add Airline Expense screen also shown in Figure 2.11 below.

1. User navigates to the home page and selects “My Expenses”
2. User selects the expense name under which the current airline expense should be filed.
3. User selects “Airline Expense” from “Expense type” drop down menu
4. User provides relevant airline expense information – Date, Airline, Expense amount, ticket number, receipt availability & Fare Basis.
5. User can upload scanned image of the receipt by selecting “Upload Receipt”
6. User can save this expense by selecting “Add”
7. The system should display this expense in grid format with “expense type” in rows displayed according to the dates in the column.

8. The user has two ways to enter airline expenses
   a. By selecting next available “empty” grid block in the airline row. When selected this should display all the options that the user has to enter for airline expense. Once updated, the grid column name for this newly added airline expense should have the date that this expense was filed for.
   b. By selecting “Airline” in the drop down menu and entering the airline expenses. Once updated, the grid should add a new entry that should have the date as the column name.

9. The system should display expense total per category, expense total per day along with total expenses for the current expense name.

Post Conditions

The following post conditions must be true after the completion of the use case.

1. User is able to create single or multiple airline expenses.

2. User can save the newly created expense to come back later for completion.
UC10 - Add a Car Rental expense

This use case describes how to configure add new Car Rental expense

Actors

Employee, Manager, System

Preconditions

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.
**Basic Flow**

The following steps represent the basic flow of events in Add Car Rental Expense screen also shown in Figure 2.12 below.

1. User navigates to the home page and selects Add new expense
2. User selects the expense name under which the current airline expense should be filed.
3. User selects “Rental Car” from “Expense Type” drop down menu
4. User provides relevant car rental expense information – date, expense amount, receipt availability & rental car type.
5. User can upload scanned image of the receipt by selecting “Upload Receipt”
6. User can save this expense by selecting “Add”
7. The system should display this rental car expense in grid format in the “rental car” row and date column of the display grid.
8. The user has a two ways to enter rental car expenses
   a. By selecting next available “empty” grid block in the “rental car” row. When selected, this should display all the options that the user has to enter for rental car expense. Once updated, the grid column name for this newly added rental car expense should have the date that this expense was filed for.
   b. By selecting “Rental Car” in the drop down menu and entering the rental car expenses. Once updated, the grid should add a new entry that should have the date as the column name.
9. The system should display expense total per category, expense total per day along with total expenses for the current expense name.

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. User is able to create single or multiple Car Rental expenses.

![Diagram](Figure 2.12 Add Car Rental Expense)

**UC11 - Add Hotel expense**

This use case describes how to add Hotel expense

**Actors**
Employee, Manager, System

**Preconditions**

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.

**Basic Flow**

The following steps represent the basic flow of events in Add Hotel Expense screen also shown in Figure 2.13 below.

1. User navigates to the home page and selects Add new expense
2. User selects the expense name under which the current Hotel expense should be filed.
3. User selects “Hotel” from “Expense Type” drop down menu
4. User provides relevant Hotel expense information – dates, location, expense amount, and receipt availability.
5. User can upload scanned image of the receipt by selecting “Upload Receipt”
6. User can save this expense by selecting “Add”
7. The system should display this Hotel expense in grid format in the “Hotel” row & date column of the display grid.
8. The user has a two ways to enter Hotel expenses
   a. By selecting next available “empty” grid block in the “Hotel” row. When selected, this should display all the options that the user has to enter for
Hotel expense. Once updated, the grid column name for this newly added Hotel expense should have the date that this expense was filed for.

b. By selecting “Hotel” in the drop down menu and entering the hotel expenses. Once updated, the grid should add a new entry that should have the date as the column name.

9. The system should display expense total per category, expense total per day along with total expenses for the current expense name.

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. User is able to create single or multiple hotel expenses.

![Diagram](image)

**Figure 2.13 Add Hotel Expense**
UC12 Add a miscellaneous expense

This use case describes how to add miscellaneous expense

Actors

Employee, Manager, System

Preconditions

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.
3. Employees should have valid expense and explanation to use this category for filing expenses.

Basic Flow

The following steps represent the basic flow of events in Add Misc. Expense screen also shown in Figure 2.14 below.

1. User navigates to the home page and selects Add new expense
2. User selects the expense name under which the current Misc expense should be filed.
3. User selects “Misc” from “Expense Type” drop down menu
4. User provides relevant expense information – dates, location, expense amount, and receipt availability. User should provide detail description of this expense
5. User can upload scanned image of the receipt by selecting “Upload Receipt”

6. User can save this expense by selecting “Add”

7. The system should display this Misc. expense in grid format in the “Misc” row & date column of the display grid.

8. The user has two ways to enter Misc expenses
   a. By selecting next available “empty” grid block in the “Misc” row. When selected, this should display all the options that the user has to enter for Misc expense. Once updated, the grid column name for this newly added Misc expense should have the date that this expense was filed for.
   b. By selecting “Misc” in the drop down menu and entering the Misc. expenses. Once updated, the grid should add a new entry that should have the date as the column name.

9. The system should display expense total per category, expense total per day along with total expenses for the current expense name.

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. User is able to create single or multiple misc. expenses.
UC13 - View Expense history

This use case describes how to view a list of existing expenses by employee

Actors

Employee, Manager, System

Preconditions

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.
3. User must have filed expense reports in the past
Basic Flow

The following steps represent the basic flow of events in viewing employee expense history also shown in Figure 2.15 below.

1. User navigates to the home page and selects “My Expenses”
2. System displays list of expenses filed by the employee
3. System should also include the status of every report filed by the employee.
4. User can navigate back to the home page by selecting “Home” button.

Post Conditions

The following post conditions must be true after the completion of the use case.

1. User is able to view history of expense reports filed in the past.

![Diagram of the basic flow of events]

**Figure 2.15** View Expense History

UC14 - Edit/delete expense

This use case describes how to edit or delete existing expense
Actors

Employee, Manager, System

Preconditions

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.
3. Status of the expense should not be “approved”

Basic Flow

The following steps represent the basic flow of events in Edit/Delete Expense screen also shown in Figure 2.16 below.

1. User navigates to the home page and selects “My Expenses”
2. System displays list of expenses filed by the employee
3. User selects appropriate expense by selecting the “edit” link next to report name from the list
4. System navigates to page to display expense information
5. User edits relevant information – dates, location, expense amount, and receipt availability.
6. User can also delete the Expense report by selecting “delete” button

Post Conditions

The following post conditions must be true after the completion of the use case.

1. User is able to update or delete expense report.
2. User can save the newly created expense to come back later for completion.

![Diagram of Edit/Delete Expense process]

**Figure 2.16 Edit/Delete Expense**

**UC15 - Submit Expense**

This use case describes how to submit expense report

**Actors**

Employee, Manager, System

**Preconditions**

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet.
2. System has validated user credentials entered in the Login Page.
**Basic Flow**

The following steps represent the basic flow of events for submit Expense screen also shown in Figure 2.17 below.

1. User can file different types of expenses under one expense report with use cases UC7, UC9, UC10, UC11 & UC12.
2. User can check the summary of all the expenses for a particular report by selecting appropriate Expense name.
3. User then reviews all expenses and will have a chance to edit the expense using UC15.
4. User can submit the expense by selecting “submit” button.
5. System should generate confirmation message to confirm submission of expense report.
6. System should send email to the “approver” of the report indicating a new expense report needs attention.

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. System should automatically update the status of the report to either “pending approval”.
UC16 - View Employee Expenses

This use case describes how to view list of employee expenses

Preconditions

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet with “manager access”
2. System has validated user credentials entered in the Login Page.

Basic Flow

The following steps represent the basic flow of events for View Employee screen also shown in Figure 2.18 below.

1. User with “manager” access navigates to the home page and selects “Employee Expenses”.

Figure 2.17 Submit Expense
2. User can look up a particular employee expense by selecting either particular employee name from drop down menu or can see expenses filed by all the employees by selecting “All” in the drop down menu.

3. System displays list of expenses filed by selected employee

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. User is able to view all expenses filed by an employee.

![Diagram](https://via.placeholder.com/150)

**Figure 2.18 View Employee Expenses**

**UC17 - Update Expense status**

This use case describes how user with “manager” access can update employee expense status

**Preconditions**

The following preconditions must be true before the initiation of the use case.
1. User has logged into the Internet with “manager access”

2. System has validated user credentials entered in the Login Page.

**Basic Flow**

The following steps represent the basic flow of events for Expense approval process also shown in Figure 2.19 below.

1. User with “manager” access navigates to the home page and selects “Employee Expenses”.

2. User selects an employee name and the system should display list of expenses filed by the employee.

3. User can view all complete details of the expenses by selecting “View Expense Details” button.

4. User can review the expense and either select “accept” or “reject” if status is “approval pending”.

5. System should send email to employee when the expense report status is approved or rejected.

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. System should automatically update the status of the report to either “approved” or “rejected” based on update from the approver.
UC18 - Expense Reports

This use case describes how user with “Manager” or “finance manager” access can generate web based expense reports.

Preconditions

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet with “Manager” or “finance manager access”

2. System has validated user credentials entered in the Login Page.

Basic Flow

The following steps represent the basic flow of events for report generation also shown in Figure 2.20 below.
1. User with navigates to the home page and selects “Expense Reports”.

2. User has an option to view reports with following options.

3. The system should allow the manager to generate web based reports.
   
   a. The filters that should be used to generate these reports are
      
      i. All employees or lookup a particular employee
      
      ii. Duration i.e. between two dates
      
      iii. Status of the expenses reports
      
4. User should also have the option to export the data generated by the above report to either excel or pdf format.

Post Conditions

The following post conditions must be true after the completion of the use case.

1. Appropriate expense report should be generated
**UC19 - Reimbursement**

This use case describes how user with “finance manager” access can reimburse approved expense reports.

**Preconditions**

The following preconditions must be true before the initiation of the use case.

1. User has logged into the Internet with “finance manager access”
2. System has validated user credentials entered in the Login Page.
**Basic Flow**

The following steps represent the basic flow of events for expense reimbursement also shown in Figure 2.21 below.

1. User with “finance manager” access navigates to the home page and selects “Employee Expenses”.
2. User has an option to view employee expenses with status in either “approved” or “reimbursed” state.
3. When the user selects “approved” expense reports, the system should display all the expenses that are currently approved by managers and need expense reimbursement.
4. User can update the expense report status to “Reimbursed” when the request is processed.
5. User should also provide the check number or enter “Direct Deposit” in “payment method” field.

**Post Conditions**

The following post conditions must be true after the completion of the use case.

1. The status of the report should change to “Reimbursed”
2.7 External Interface Requirements

2.7.1 User Interfaces

The user interface for the system will be a web page on the Internet. The user interface will be limited to the types of controls that can be generated using HTML, JavaScript and Cascading Style Sheets.

2.7.2 Software Interfaces

1. Operating System: The software is being designed to run on Windows Vista/XP/7. These include the latest version of Internet Information Services.

2. Web Server: The software is being designed to run on ISS version 6.0.

4. Libraries: The software will be created using the Microsoft .NET version 3.5 framework.

2.8 Non-Functional Requirements

2.8.1 Security Requirements

1. Passwords shall be displayed as “*” in the web pages wherever required.

2. Proper authentication is required for users to access any of the web pages including the home page.

3. Every user of the system is assigned a unique login and password to access the application over the internet.

4. Microsoft .NET framework ensures security of data, for example passwords that are being transmitted over the internet.

2.8.2 System Constraints

This application is developed to be flexible enough to work on any system and independent of the type of web browser used. The only constraint is the client will need a web-enabled system with web browser.

2.8.3 Software Quality Attributes

Reliability This reliability of this project is one of the most critical attributes as it involves the financial aspect of the employee. The employee expense i.e., the data
entered must always be correctly stored into the database. In addition, the database should commit transactions so that partial entries are not stored in the database.

**Usability** The system should be very easy to use with minimal required training. Individuals of varying skill level and technical competence will use the system.

**Maintainability** The code and design need to be documented well enough and designed such that a new project member with the same amount of academic and co-op experience can easily ramp up the project.
Chapter 3

SYSTEM DESIGN

This chapter describes the System design of the project by starting with introduction to the developer environment followed by the System Architecture definition.

3.1 .NET Framework

This project is developed using Microsoft Visual Studio .NET Framework along with Microsoft SQL Server 2005 relational database system. The Microsoft .NET is the framework for which we develop applications. It sits in between our application programs and operating system. It supports both Windows and web applications. Figure 3.1 below shows important components for .NET infrastructure [1].

![.NET Framework Components](image)

Figure 3.1 .NET Framework Components
3.1.1 Common Language Runtime

The Common Language Runtime (CLR) is the environment where all programs are run in .NET. CLR provides various services like memory management and thread management. Programs that run in the CLR need not manage memory as it is managed by the CLR. For example, when a program needs a block of memory, CLR provides the block and releases the block when program is done with the block. The managed code is written in a high-level CLR-compliant language (such as Visual Basic .NET or C#) and is then compiled into an Intermediate Language (IL). The IL code itself cannot run directly on any computer, i.e., a Just-In-Time compiler or JITter interprets it. This intermediate step slows down the performance but provides the .NET Framework with a certain amount of platform independence as each platform can have its own JITter [1].

3.1.2 Class Library

The class library contains the foundation classes that are used to build applications. The library has a tree structure where each class inherits the functionalities of its parent. The developer can extend the .NET Framework by creating custom classes that inherit from those of the prebuilt tree.

3.1.3 Windows Forms

Windows forms are a package that provides ready-to-use user interface elements to build powerful front ends for desktop applications. Such elements, which are also called
Windows controls, include windows, buttons, dialog boxes, tree views, data grids, and so on.

3.1.4 ASP.NET

ASP.NET provides support to build and run Web applications. Its main functions include the following:

- Prefabricated controls that do for Hypertext Markup Language (HTML) pages what Windows controls do for desktop applications.

- A Web Server runtime environment that dynamically generates HTML pages in response to input received from the client.

- Advanced services such as data caching to speed up documents that are often downloaded, session state to personalize clients, and security to block malicious clients [3].

3.2 Building Web-based Application

This project is implemented using client-server architecture. It is hosted on a web server and it responds to clients as shown in figure below. Web Applications provide content from the server to the client. Clients obtain those contents from internet and view them using a web browser [3].
This project runs on MS IIS (Microsoft Internet Information Services) Web Server. IIS is a web server application and set of feature extension modules created by Microsoft for use with Microsoft Windows.

### 3.3 .NET Application Architecture

In the classic three-tier design of a .NET application, the architecture breaks down into three major areas of functionality [1][5]

1. The data layer manages the physical storage and retrieval of data
2. The business layer maintains business rules and logic
3. The presentation layer houses the user interface and related presentation code.

#### 3.1.1 Presentation Tier

In the presentation layer, the code-behind mechanism for ASP.NET pages and user controls is a prominent example of a layered design. Designers
do not have to worry about messing up code to make user interface changes, and developers do not have to worry about sifting through the user-interface to update code.

3.1.2 Data Tier The data tier has the tables that define the physical storage of data in a database, stored procedures and views that allow manipulating data as it goes into and out of those tables.

3.1.3 Business Tier A business object is a component that encapsulates the data and business processing logic for a particular business entity. Business tier contains logic for retrieving persistent data from the data-tier and placing it into business objects. The following is a block diagram of all the modules in the application and the relationships between them. The following view includes arrows to indicate flow of data and/or control.
3.4 High Level Design

The High-Level Design provides an overview of the design components of the web application. The design is represented through elaborations on class diagrams for each of the core sub-systems. The purpose of the high-level design is to define key classes, as well as the interactions among instances of these classes (objects). Inputs to the HLD consist of the functional requirements defined in the above sections [5].

The HLD represents a decomposition of all the sub-systems down to the class level. It is envisioned that the HLD will form the basis for the detailed design efforts.
Following are the key goals and constraints governing the design decisions in the HLD.

- High-level design should decompose complex business logic; identify dependencies among components, and arrive at a holistic view of the application.
- High-level design should address core functionality by identifying and defining key classes, their responsibilities, behavior and attributes.
- High-level design should clearly define application boundaries and their interfaces.
- High-level design should form the basis for planning and scoping subsequent phases of the project.
- High-level design should adopt object-oriented design principles such as encapsulation, information hiding, loose coupling, and high cohesion.

The figure below depicts the class diagram, which gives an overview of the entire system by showing its classes and relationships among them.
Figure 3.4 Class Diagram
Chapter 4

DATABASE DESIGN

Database design is very crucial for the performance of the system. A good database design will contribute to software development and improve performance. It facilitates data retrieval, storage and modification.

4.1 About SQL Server 2005

Microsoft SQL Server is a relational model database server produced by Microsoft. Its primary query languages are T-SQL and ANSI SQL. SQL Server 2005 includes native support for managing XML data, in addition to relational data. For relational data, T-SQL has been augmented with error handling features (try/catch) and support for recursive queries (Common Table Expressions). SQL CLR was introduced with the SQL Server 2005 to let it integrate with the .NET Framework. [4]

4.2 Database Normalization

In the design of a database management system, the process of organizing data to minimize redundancy is called normalization. The goal of database normalization is to decompose relations with anomalies in order to produce smaller, well-structured relations. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships [11].
Some of the normal forms relevant in this project are

- First Normal Form
- Second Normal Form
- Third Normal Form
- Boyce Codd Normal Form
- Fourth Normal Form

**First Normal Form** An entity is in First Normal Form (1NF) when all tables are two-dimensional with no repeating groups. A row is in first normal form (1NF) if all underlying domains contain atomic values only. 1NF eliminates repeating groups by putting each into a separate table and connecting them with a one-to-many relationship. Make a separate table for each set of related attributes and uniquely identify each record with a primary key [12].

- Eliminate duplicative columns from the same table.
- Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

**Second Normal Form** An entity is in Second Normal Form (2NF) when it meets the requirement of being in First Normal Form (1NF) and additionally [12]

- Does not have a composite primary key. Meaning that the primary key cannot be subdivided into separate logical entities.
- All the non-key columns are functionally dependent on the entire primary key.
• A row is in second normal form if, and only if, it is in first normal form and every non-key attribute is fully dependent on the key.

• 2NF eliminates functional dependencies on a partial key by putting the fields in a separate table from those that are dependent on the whole key. An example is resolving many:many relationships using an intersecting entity.

**Third Normal Form** An entity is in Third Normal Form (3NF) when it meets the requirement of being in Second Normal Form (2NF) and additionally [12]

• Functional dependencies on non-key fields are eliminated by putting them in a separate table. At this level, all non-key fields are dependent on the primary key.

• A row is in third normal form if and only if it is in second normal form and if attributes that do not contribute to a description of the primary key are move into a separate table. An example is creating look-up tables.

**Boyce-Codd Normal Form** BCNF is a further refinement of 3NF. A row is in Boyce-Codd normal form if, and only if, every determinant is a candidate key. Most entities in 3NF are already in BCNF. BCNF covers very specific situations where 3NF misses interdependencies between non-key attributes. Typically, any relation that is in 3NF is also in BCNF. However, a 3NF relation won't be in BCNF if (a) there are multiple candidate keys, (b) the keys are composed of multiple attributes, and (c) there are common attributes between the keys [12].

**Fourth Normal Form** An entity is in Fourth Normal Form (4NF) when it meets the requirement of being in Third Normal Form (3NF) and additionally [12]
• Has no multiple sets of multi-valued dependencies. In other words, 4NF states that no entity can have more than a single one-to-many relationship within an entity if the one-to-many attributes are independent of each other.
• Many:many relationships are resolved independently.

4.3 Database Tables and Database Diagram

Considering the nature of the project, the tables for the database are designed to minimize redundancy. This facilitates CASCADE DELETE and avoids any duplicate entries. When designing the database for the project, care was taken to break down the tables to address specific areas for the project to avoid redundancy and also to follow the database normalization procedures. Another advantage of this project is that the system can easily be extended to incorporate additional functionalities. These functionalities are discussed in the Future Scope section of the document.

For this project there are a total of 6 database tables. The following section describes each of these tables along with various field names, attributes, etc.

4.3.1 Employee Table

This table is used to store employee information in the table 4.1 below. The field empid is the system generated Primary key that identifies employee. This table collects the employee profile information including the bank details.

Analysis

1NF The employee table meets the requirements of first Normal form as the table does not require ordering of row or columns and the table does not have any duplicate rows.
2NF The primary key “empid” for this table cannot be subdivided and remaining attributes of the table are functionally dependent on the entire primary key.

3NF In this table there are non-key columns related to bank information that are mutually dependent and hence this table does not meet the requirements for 3NF.
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Null/ Not Null</th>
<th>Unique Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>empid</td>
<td>Int</td>
<td>System generated Employee ID</td>
<td>Not Null</td>
<td>Primary Key</td>
</tr>
<tr>
<td>empfname</td>
<td>nvarchar(50)</td>
<td>Employee First Name</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>emplname</td>
<td>nvarchar(50)</td>
<td>Employee Last Name</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>empsal</td>
<td>numeric(18, 0)</td>
<td>Employee Annual Salary</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>qualification</td>
<td>nchar(20)</td>
<td>Qualification</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>address1</td>
<td>nvarchar(50)</td>
<td>Employee Address</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>address2</td>
<td>nvarchar(50)</td>
<td>Employee Address</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Phone</td>
<td>nvarchar(20)</td>
<td>Employee Phone Number</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>mobile1</td>
<td>nvarchar(20)</td>
<td>Employee Phone Number</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>mobile2</td>
<td>nvarchar(20)</td>
<td>Employee Phone Number</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>emailed</td>
<td>nvarchar(30)</td>
<td>Employee Email Address</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>nvarchar(50)</td>
<td>City</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>nvarchar(50)</td>
<td>State</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>nvarchar(50)</td>
<td>Country</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>postcode</td>
<td>nvarchar(50)</td>
<td>Zip Code</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>designationId</td>
<td>Int</td>
<td>Designation</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>joiningdate</td>
<td>varchar(50)</td>
<td>Start Date</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>dateofbirth</td>
<td>varchar(50)</td>
<td>Date of Birth</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>sex</td>
<td>nvarchar(50)</td>
<td>Sex</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>roleid</td>
<td>Int</td>
<td>RoleID (defined in Roles Table)</td>
<td>Null</td>
<td>FK</td>
</tr>
<tr>
<td>managerid</td>
<td>Int</td>
<td>Manager ID</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>bankname</td>
<td>nvarchar(50)</td>
<td>Name of the Bank</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>acctype</td>
<td>nchar(10)</td>
<td>Account Type (savings or checking)</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>accno</td>
<td>nvarchar(20)</td>
<td>Account Number</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>modeoftransfer</td>
<td>nvarchar(20)</td>
<td>Mode of Transfer (Direct Deposit or Check)</td>
<td>Null</td>
<td></td>
</tr>
</tbody>
</table>
4.3.2 Login Details Table  This table is used to store the login information of each employee. Each employee with unique “empId” has a user name & password to login to the system.

Analysis

1NF The “Login Details” table meets the requirements of first Normal form as the table does not require ordering of row or columns and the table does not have any duplicate rows.

2NF The primary key “loginid” for this table cannot be subdivided and remaining attributes of the table are functionally dependent on the entire primary key.

3NF In this table all the non-key columns are mutually independent and are only relevant to the primary key “loginid”

4NF The “Login Details” table also meets the 4NF as it is 3NF and also does not contain attributes with two or more independent many-to-many relationships.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Null/ Not Null</th>
<th>Unique Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>loginid</td>
<td>Int</td>
<td>Login ID</td>
<td>Not Null</td>
<td>PK</td>
</tr>
<tr>
<td>loginName</td>
<td>nchar(50)</td>
<td>User Name</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>nchar(50)</td>
<td>User Password</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>empId</td>
<td>Int</td>
<td>Employee ID</td>
<td>Not Null</td>
<td>FK</td>
</tr>
<tr>
<td>loginstatus</td>
<td>Bit</td>
<td>Login (Active/Deactive)</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>lastlogindt</td>
<td>Datetime</td>
<td>Last login date information</td>
<td>Null</td>
<td></td>
</tr>
</tbody>
</table>

4.3.3 Employee Expense Table This table is used to store all the information regarding the expenses that employees submit as shown below.
Analysis

1NF This table meets the requirements of first Normal form as the table does not require ordering of row or columns and the table does not have any duplicate rows.

2NF The primary key “empExpID” for this table cannot be subdivided and remaining attributes of the table are functionally dependent on the entire primary key.

3NF This table does not meet the third normal form as non-key fields like “Rental_Agency”, “Car_class”, “Airline”, etc are dependent on non-key field “ExpenseID”
### Table 4.3 Employee Expense Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Null/ Not Null</th>
<th>Unique Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>empExpID</td>
<td>bigint</td>
<td>System generated Expense ID</td>
<td>Not Null</td>
<td>PK</td>
</tr>
<tr>
<td>Empid</td>
<td>int</td>
<td>Employee ID</td>
<td>Null</td>
<td>FK1</td>
</tr>
<tr>
<td>ExpenseID</td>
<td>bigint</td>
<td>This is connected to PK of ExpenseType table</td>
<td>Null</td>
<td>FK2</td>
</tr>
<tr>
<td>EmpExpCatogiryID</td>
<td>bigint</td>
<td>Expense Category ID</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>ExpenseFor</td>
<td>varchar(50)</td>
<td>Expense Type</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>ExpenseCost</td>
<td>float</td>
<td>Cost</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>ExpenseOn</td>
<td>datetime</td>
<td>Date of expense</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>HasReceipt</td>
<td>bit</td>
<td>Receipt available or not</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>ReceiptPath</td>
<td>varchar(500)</td>
<td>Receipt saved location</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Hotel_Name</td>
<td>varchar(100)</td>
<td>Name of the Hotel</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>DaysInHotel</td>
<td>int</td>
<td>Number of days in Hotel</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Airline</td>
<td>varchar(100)</td>
<td>Airline Name</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Ticket_Number</td>
<td>varchar(50)</td>
<td>Airline Ticket Number</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Fare_Basis</td>
<td>varchar(50)</td>
<td>Economy/First class</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Car_Class</td>
<td>varchar(50)</td>
<td>Car type</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Rental_Agency</td>
<td>varchar(50)</td>
<td>Name of the Rental Car Agency</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>varchar(100)</td>
<td>Location of the Rental Car Agency</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>CreatedDate</td>
<td>datetime</td>
<td>Expense Creation date</td>
<td>Null</td>
<td></td>
</tr>
</tbody>
</table>

#### 4.3.4 Roles Table

This table is primarily accessible to employee with admin credentials. Admin’s can add new role id i.e. “Manager, Finance Manager, etc” depending on the firm requirement. This information is stored in the Roles table as shown in the table below.

**Analysis**

1NF The “Roles” table meets the requirements of first Normal form as the table does not require ordering of row or columns and the table does not have any duplicate rows.
2NF The primary key “roleid” for this table cannot be subdivided and remaining attributes of the table are functionally dependent on the entire primary key.

3NF In this table all the non-key columns are mutually independent and are only relevant to the primary key “roleid”

4NF This table also meets the 4NF as it is 3NF and also does not contain attributes with two or more independent many-to-many relationships.

Table 4.4 Roles Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Null/ Not Null</th>
<th>Unique Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>roleid</td>
<td>int</td>
<td>Role ID</td>
<td>Not Null</td>
<td>PK</td>
</tr>
<tr>
<td>roleName</td>
<td>nchar(50)</td>
<td>Name of the Role</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>CreatedDate</td>
<td>datetime</td>
<td>Role Creation date</td>
<td>Null</td>
<td></td>
</tr>
</tbody>
</table>

4.3.5 Employee Expense Categories Table This table is holds the current status of the expenses filed by employees.

Analysis

1NF This table meets the requirements of first Normal form as the table does not require ordering of row or columns and the table does not have any duplicate rows.

2NF The primary key for this table cannot be subdivided and remaining attributes of the table are functionally dependent on the entire primary key.

3NF In this table all the non-key columns are mutually independent and are only relevant to the primary key.

4NF This table also meets the 4NF as it is 3NF and also does not contain attributes with two or more independent many-to-many relationships.
**Table 4.5 Employee Expense Categories Table**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Null/ Not Null</th>
<th>Unique Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmpExpCatogiryID</td>
<td>bigint</td>
<td>System generated expense category ID</td>
<td>Not Null</td>
<td>PK</td>
</tr>
<tr>
<td>empid</td>
<td>int</td>
<td>Employee ID</td>
<td>Null</td>
<td>FK</td>
</tr>
<tr>
<td>CatogiryName</td>
<td>varchar(50)</td>
<td>Expense Name</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>CreatedDate</td>
<td>datetime</td>
<td>Expense created date</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>VerifiedStatus</td>
<td>varchar(50)</td>
<td>Current Status</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>RejectReason</td>
<td>varchar(500)</td>
<td>Reason for rejection</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>VerifiedDate</td>
<td>datetime</td>
<td>Expense Verification date</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Reimbursedate</td>
<td>datetime</td>
<td>Expense Reimbursement date</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>Reimbursetype</td>
<td>varchar(500)</td>
<td>Type of Expense Reimbursement</td>
<td>Null</td>
<td></td>
</tr>
</tbody>
</table>

**4.3.6 Expense Type Table** This table is mainly updated by Admin as it defines the different kinds of expenses that employees could submit. The system is flexible to easily add/edit or delete any expense type depending on the requirements.

**Analysis**

**1NF** The “Expense Type” table meets the requirements of first Normal form as the table does not require ordering of row or columns and the table does not have any duplicate rows.

**2NF** The primary key “ExpenseID” for this table cannot be subdivided and remaining attributes of the table are functionally dependent on the entire primary key.

**3NF** In this table all the non-key columns are mutually independent and are only relevant to the primary key “ExpenseID”
4NF This table also meets the 4NF as it is 3NF and also does not contain attributes with two or more independent many-to-many relationships.

**Table 4.6 Expense Type Table**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Null/ Not Null</th>
<th>Unique Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExpenseID</td>
<td>Bigint</td>
<td>Expense ID</td>
<td>Not Null</td>
<td>PK</td>
</tr>
<tr>
<td>ExpenseType</td>
<td>varchar(20)</td>
<td>Expense Type</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>ExpenseStatus</td>
<td>Bit</td>
<td>Current Status of the Expense</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>ExpCostLimit</td>
<td>Float</td>
<td>Cost limit</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>CreatedDate</td>
<td>datetime</td>
<td>Expense Creation Date</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>UpdatedDate</td>
<td>datetime</td>
<td>Expense Updated Date</td>
<td>Null</td>
<td></td>
</tr>
</tbody>
</table>

The following figure 4.1 was developed using Microsoft Visio showing the Entity-Relationship diagram for this project.
Figure 4.1 Entity Relationship Diagram
4.4 Sample of Database Scripts

Following are some examples of scripts that are used in the project to create tables.

```sql
CREATE TABLE [dbo].[Tbl_EmpExpenses](
    [empExpID] [bigint] IDENTITY(1,1) NOT NULL,
    [empid] [int] NULL,
    [ExpenseID] [bigint] NULL,
    [EmpExpCatogiryID] [bigint] NULL,
    [ExpenseFor] [varchar](50) NULL,
    [ExpenseCost] [float] NULL,
    [ExpenseOn] [datetime] NULL,
    [HasReceipt] [bit] NULL,
    [ReceiptPath] [varchar](500) NULL,
    [Hotel_Name] [varchar](100) NULL,
    [DaysInHotel] [int] NULL,
    [Airline] [varchar](100) NULL,
    [Ticket_Number] [varchar](50) NULL,
    [Fare_Basis] [varchar](50) NULL,
    [Car_Class] [varchar](50) NULL,
    [Rental_Agency] [varchar](50) NULL,
    [Location] [varchar](100) NULL,
    [CreatedDate] [datetime] NULL,
    PRIMARY KEY [empExpID] );

CREATE TABLE [dbo].[Tbl_ExpenseType](
    [ExpenseID] [bigint] IDENTITY(1,1) NOT NULL,
    [ExpenseType] [varchar](20) NULL,
    [ExpenseStatus] [bit] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
    [ExpCostLimit] [float] NULL,
[CreatedDate] [datetime] NULL,
[UpdatedDate] [datetime] NULL,
[ModifiedBy] [bigint] NULL,
PRIMARY KEY [ExpenseID] );
Chapter 5
IMPLEMENTATION DETAILS

This chapter describes the implementation details of the project. The implementation phase of a project is a critical step towards achieving the desired goals. It actually involves bringing the system design into action by following the requirements and also making it work. This chapter describes the .NET technologies used for the project and also contains sample code to explain some of the main features in the application.

5.1 Development of Web Interface using ASP.NET

ASP.NET is a web application framework developed to allow programmers to build dynamic web sites, web applications and web services [3].

Advantages using ASP.NET [2]

1. ASP.NET drastically reduces the amount of code required to build large applications.
2. ASP.NET makes development simpler and easier to maintain with an event-driven, server-side programming model.
3. ASP.NET pages are easy to write and maintain because the source code and HTML are together. The source code is executed on the server.
4. The source code is compiled the first time the page is requested. Execution is fast as the Web Server compiles the page the first time it is requested. The server saves the compiled version of the page for use next time the page is requested.
5. The HTML produced by the ASP.NET page is sent back to the browser.
6. The Web server continuously monitors the pages, components and applications running on it. If it notices memory leaks, infinite loops, other illegal software or activities, it seamlessly kills those activities and restarts itself.

7. ASP.NET validates information (validation controls) entered by the user without writing a single line of code.

8. ASP.NET easily works with ADO.NET using data binding and page formatting features.

9. We can create the design of the web page by dragging and dropping appropriate controls in the Design view of Microsoft Visual Studio .net

5.2 Implementation details of an Employee Expense

One of the main aspects of the project is the ability of an employee to file expenses. The following section describes this process with sample code. Following are the major features to file an expense

- As the employee is expected to file expenses on monthly basis, an expense name has to be first created (e.g. April 2010) so that employee can submit expense for several days within that month.

- Update the existing the expense, provided the expense is not already submitted for approval

- Delete an existing expense if the expense is not submitted.

Following is the sample code generated for creating a text box and also an add button for adding “Expense name”.
When employee creates a new expense name, a data grid is displayed showing options for employee to either edit, delete or send the expense for approval. Following is the corresponding aspx code.

```html
<asp:GridView ID="GVRoles" AutoGenerateColumns="false" DataKeyNames="EmpExpCatogiryID,CatogiryName,VerifiedStatus" runat="server" onrowdeleting="GVRoles_RowDeleting" CellPadding="5" onrowediting="GVRoles_RowEditing" AllowPaging="True" onpageindexchanging="GVRoles_PageIndexChanging" ForeColor="#333333" onrowcommand="GVRoles_RowCommand" onrowdatabound="GVRoles_RowDataBound">

<Columns>
  <asp:BoundField Visible="false" DataField="EmpExpCatogiryID" HeaderText="ID"/>
  <asp:BoundField DataField="CatogiryName" HeaderText="Expense Name"/>
  <asp:BoundField DataField="VerifiedStatus" HeaderText="Status"/>
  <asp:BoundField DataField="CreatedDate" HeaderText="DATE" DataFormatString="{0:MM/dd/yyyy}"/>
  <asp:TemplateField HeaderText="View">
    <ItemTemplate>
      <asp:ImageButton ID="imgView" runat="server" ImageUrl="/images/view.png" onclick="btnaddrole_Click" ValidationGroup="val" Text="Add"/>
    </ItemTemplate>
  </asp:TemplateField>
</Columns>
```
5.3 Implementation details of Adding New Expense

Following are actions that the employee can perform for adding new expenses:

- The employee first selects the view button to view the data grid for entering expenses.
- The data grid contains “expense type” as rows and “date” for columns. The grid spans for the entire month and the employee has an option to view expenses for a different month by selecting appropriate month from drop down menu.
Following is the code when the expense data grid is generated.

```html
<asp:GridView ID="GridView1" runat="server"
  onrowcommand="GridView1_RowCommand" CellPadding="2"
  onrowdatabound="GridView1_RowDataBound" >
  </asp:GridView>
```

### 5.4 ADO.NET Architecture and Application to the project

Most applications need data access at one point of time making it a crucial component when working with applications. Data access is making the application interact with a database, where all the data is stored. Different applications have different requirements for database access. VB .NET uses ADO .NET (Active X Data Object) as its data access and manipulation protocol which also enables us to work with data on the Internet [5].

ADO.NET is a set of classes that expose data access services to the .NET programmer. It provides functionality to developers writing managed code. ADO.NET provides consistent access to data sources such as Microsoft SQL Server, as well as data sources exposed through OLE DB and XML. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data. The figure below shows the basic architecture of ADO.NET [6].
When the employee creates a new expense name, the following method is used to insert the expense information into the database.

```csharp
public int EmpExpCatogory_Insert(string EmpID, string CatogiryName,string VerifiedStatus)
{
    try
    { 
        string[] param = { "@empid","@EmpExpCatogiryID", "@CatogiryName", "@VerifiedStatus", "@TransType" };
        object[] values = {EmpID , 0, CatogiryName, VerifiedStatus, "I" };
        SqlDbType[] type = {SqlDbType .Int , SqlDbType.Int, SqlDbType.VarChar, SqlDbType.VarChar, SqlDbType.VarChar};
        return DBCon.executeNonQuery(param, values, type, "EmpExpCatogiries_SP");
    }
    catch (Exception ex)        {
        throw ex;
    }
```
The above mentioned function takes five input parameters and returns an integer which is then used by the page to insert the details entered by the user. Following stored procedure is used by above function and performs the insert operation.

```sql
alter procedure [dbo].[EmpExpCatogories_SP]
@EmpExpCatogiryID bigint =null,
@empid as int=null,
@CatogiryName varchar(50) =null,
@VerifiedStatus varchar(50)=null,
@TransType varchar(2)=notnull,
@RejectReason varchar(500)=null
as
begin
if @TransType='S'
begin
select EEC.* from EmpExpCatogories EEC where empid=@empid
end
else if @TransType='I'
begin
INSERT INTO [dbo].[EmpExpCatogories]
(empid,[CatogiryName]
,[VerifiedStatus])
VALUES
(@empid,@CatogiryName,
,@VerifiedStatus)
End
else if @TransType='U'
begin
UPDATE [dbo].[EmpExpCatogories]
SET [CatogiryName] = @CatogiryName
where EmpExpCatogiryID=@EmpExpCatogiryID and empid=@empid
end
else if @TransType='D'
begin
delete from [EmpExpCatogories] WHERE
EmpExpCatogiryID=@EmpExpCatogiryID and empid=@empid
```
delete from Tbl_EmpExpenses where
EmpExpCatogiryID=@EmpExpCatogiryID and empid=@empid;
end
else if @TransType='CS'
begin
UPDATE [dbo].[EmpExpCatogiries]
SET
[VerifiedStatus] = @VerifiedStatus, RejectReason=@RejectReason
where EmpExpCatogiryID=@EmpExpCatogiryID and empid=@empid
end
end

Now, the data table can be used to fill the data grid i.e., New Expense grid with the associated date. Following is the code to fill the data grid with table data.

GVRoles.DataSource = dt;
GVRoles.DataBind();

After adding the new expense name, the next step is to enter the different types of expenses like airline, car rental, hotel, etc for that particular expense.

The following code snippet shows getting a connection and inserting new expenses into the “Tbl_EmpExpenses” table

```csharp
public int EmpExpenses_Insert(string ExpenseID, string empid,
string ExpenseFor, string Cost, string ExpenseOn, string
EmpExpCatogiryID, string HasReceipt, string Hotel_Name, string
DaysInHotel, string Airline, string Ticket_Number, string
Fare_Basis, string Car_Class, string Rental_Agency, string Location,
string ReceiptPath)
{
    try
    {
        string[] param = {
            "@empExpID", "@empid", "@ExpenseID",
            "@ExpenseFor", "@ExpenseCost", "@ExpenseOn", "@TransType",
            "@HasReceipt", "@Hotel_Name", "@DaysInHotel", "@Airline",
            "@Ticket_Number", "@Fare_Basis", "@Car_Class", "@Rental_Agency",
            "@Location", "@Filepath"};
```
object[] values = { 0, empid, ExpenseID, ExpenseFor, Cost, ExpenseOn, "1", "", "", 0, EmpExpCatogiryID, HasReceipt, Hotel_Name, DaysInHotel, Airline, Ticket_Number, Fare_Basis, Car_Class, Rental_Agency, Location, ReceiptPath };  
    SqlDbType[] type = { SqlDbType.BigInt, SqlDbType.BigInt, SqlDbType.BigInt, SqlDbType.VarChar, SqlDbType.Int, SqlDbType.DateTime, SqlDbType.VarChar, SqlDbType.VarChar, SqlDbType.VarChar, SqlDbType.Int, SqlDbType.BigInt, SqlDbType.Int, SqlDbType.VarChar, SqlDbType.VarChar, SqlDbType.VarChar, SqlDbType.VarChar, SqlDbType.VarChar, SqlDbType.VarChar };  
    return DBCon.executeNonQuery(param, values, type, "EmpExpenses_SP2");  
}  
catch (Exception ex)  
{
    throw ex;
}  
finally  
{
}  
}

Following stored procedure is used by above function to insert the data into database

set ANSI_NULLS ON
set QUOTED_IDENTIFIER ON

go

ALTER procedure [dbo].[EmpExpenses_SP2]
    @empExpID bigint=null,
    @empid bigint=notnull,
    @ExpenseID bigint=null,
    @ExpenseFor varchar(500)=null,
    @ExpenseCost float=null,
    @ExpenseOn datetime =null,
    @TransType varchar(2)=notnull,
    @VerifiedStatus varchar(50)=null,
    @CusName varchar(500) =null,
    @IsDraft int=null,
    @EmpExpCatogiryID bigint=null,
    @HasReceipt int=null,
    @Hotel_Name varchar(500)=null,
    @DaysInHotel varchar(500) ,
    @Airline varchar(500)=null,
    @Ticket_Number varchar(500)
as

begin

if @TransType='I'
begin

declare @tempempExpID as bigint
select @tempempExpID=empExpID from Tbl_EmpExpenses where
ExpenseID=@ExpenseID and empid=@empid and
EmpExpCatogiryID=@EmpExpCatogiryID and
dbo.fnFormatDate(ExpenseOn,'D/M/YYYY')=dbo.fnFormatDate(@ExpenseOn,'D/M/YYYY')
select @tempempExpID
if @tempempExpID is null
begin
--insert
INSERT INTO [dbo].[Tbl_EmpExpenses]
([empid],[ExpenseID],[EmpExpCatogiryID],[ExpenseFor],[ExpenseCost]
, [ExpenseOn],[HasReceipt],[Hotel_Name],[DaysInHotel]
,[Airline],[Ticket_Number],[Fare_Basis],[Car_Class],[Rental_Agency],[Location],ReceiptPath)
VALUES
(@empid,@ExpenseID,@EmpExpCatogiryID
,@ExpenseFor,@ExpenseCost,@ExpenseOn,@HasReceipt
,@Hotel_Name,@DaysInHotel,@Airline,@Ticket_Number
,@Fare_Basis,@Car_Class,@Rental_Agency
,@Location,@Filepath
)
--end insert
end
else
begin
--Update

UPDATE [dbo].[Tbl_EmpExpenses]
SET
[ExpenseFor] = @ExpenseFor
,[ExpenseCost] = @ExpenseCost
WHERE empExpID=@tempempExpID

--end Update
end
end

else if @TransType='U'
begin

UPDATE [dbo].[Tbl_EmpExpenses]
SET
  [ExpenseFor] = @ExpenseFor
  ,[ExpenseCost] = @ExpenseCost
  ,[ExpenseOn] = @ExpenseOn
  ,[HasReceipt] = @HasReceipt
  ,[Hotel_Name] = @Hotel_Name
  ,[DaysInHotel] = @DaysInHotel
  ,[Airline] = @Airline
  ,[Ticket_Number] = @Ticket_Number
  ,[Fare_Basis] = @Fare_Basis
  ,[Car_Class] = @Car_Class
  ,[Rental_Agency] = @Rental_Agency
  ,[Location] = @Location
  ,[ReceiptPath]=@Filepath
WHERE empExpID=@empExpID

end

else if @TransType='S'
begin
select EE.*,ET.ExpenseType from dbo.Tbl_EmpExpenses
EE,dbo.Tbl_ExpenseType ET WHERE EE.ExpenseID=ET.ExpenseID and
ET.ExpenseStatus=1 and empid=@empid and
EmpExpCatogiryID=@EmpExpCatogiryID

end
else if @TransType='D'
begin
    delete from [Tbl_EmpExpenses] WHERE empExpID=@empExpID and empid=@empid;
end
end

Similarly, any other functionality can be implemented using the same logic for retrieving and storing data appropriately in the database.

The following code snippet shows deletion functionality if the employee decides to delete a particular expense.

```csharp
public int EmpExpCatogory_Delete(string empid,string EmpExpCatogiryID)
{
    try
    {
        string[] param = {"@empid", "@EmpExpCatogiryID", "@CatogiryName", "@VerifiedStatus", "@TransType" };
        object[] values = { empid,EmpExpCatogiryID, "", "", "D" ];
        SqlDbType[] type = {SqlDbType.Int , SqlDbType.Int, SqlDbType.VarChar, SqlDbType.VarChar, SqlDbType.VarChar };        return DBCon.executeNonQuery(param, values, type, "EmpExpCatogiries_SP");
        // return 0;
    }
    catch (Exception ex)
    {
        throw ex;
    }
    finally
    {
    }
}
```

Following stored procedure is used by the above code to delete operation from database.
alter procedure [dbo].[EmpExpCatogiries_SP]
 @EmpExpCatogiryID bigint =null,
 @empid as int=null,
 @CatogiryName varchar(50) =null,
 @VerifiedStatus varchar(50)=null,
 @TransType varchar(2)=notnull,
 @RejectReason varchar(500)=null
 as
 begin

 if @TransType='S'
 begin
 select EEC.* from EmpExpCatogiries EEC where empid=@empid
 end
 else if @TransType='I'
 begin
 INSERT INTO [dbo].[EmpExpCatogiries]
 (empid,[CatogiryName]
 ,[VerifiedStatus]
 )
 VALUES
 (@empid,@CatogiryName
 ,@VerifiedStatus)
 End
 else if @TransType='U'
 begin
 UPDATE [dbo].[EmpExpCatogiries]
 SET [CatogiryName] = @CatogiryName
 where EmpExpCatogiryID=@EmpExpCatogiryID and empid=@empid
 end
 else if @TransType='D'
 begin
 delete from [EmpExpCatogiries] WHERE
 EmpExpCatogiryID=@EmpExpCatogiryID and empid=@empid
 delete from Tbl_EmpExpenses where EmpExpCatogiryID=@EmpExpCatogiryID and empid=@empid;
 end
 else if @TransType='CS'
 begin
 UPDATE [dbo].[EmpExpCatogiries]
 SET
 [VerifiedStatus] =
 @VerifiedStatus,RejectReason=@RejectReason
 where EmpExpCatogiryID=@EmpExpCatogiryID and empid=@empid
 end
 end
5.5 Business Logic Implementation using C#.NET

C# is a multi-paradigm programming language encompassing imperative, functional, generic, object oriented and component oriented programming disciplines. C# is intended to be a simple, modern, general-purpose, object-oriented programming language [4].

Microsoft Visual C# is Microsoft's implementation of the C# language, which targets the .NET Framework, along with the language services that lets the Visual Studio IDE support C# projects. While the language services are a part of Visual Studio, the compiler is available separately as a part of the .NET Framework. The Visual C# 2008 compiler supports version 3.0 of the C# language specifications. Visual C# supports the Visual Studio Class designer, Forms designer, and Data designer among others. [4]

The business logic in the project is implemented using C#.NET. This includes several server-side validations for checking validity of data entered by user before saving data into the database.

After retrieving and displaying events expense data grid, the next step is to add appropriate business logic to the expense management functionality.

The following code can be used to populate the grid view with expenses filed under a particular expense name.

```csharp
protected void btnaddrole_Click(object sender, EventArgs e)
{
    try
    {
        if (btnaddrole.Text == "Add")
        {
            // Code to populate grid view
        }
    }
```
int status =
mybal.EmpExpCatogory_Insert(Session["EmpID"].ToString(),
txtCatogiryName.Text, "Not Submitted");
if (status == 1)
{
    GVDataBind();
    lblerr.Text = "Record Inserted Successfully";
    lblerr.MessageColor(true);
}
else
{
    lblerr.Text = "Try Again";
    lblerr.MessageColor(false);
}
else if (btnaddrole.Text == "Update")
{
    int status =
mybal.EmpExpCatogory_UpDate(Session["EmpID"].ToString(),
ViewState["EmpExpCatogiryID"].ToString(), txtCatogiryName.Text,
"");
if (status == 1)
{
    GVDataBind();
    lblerr.Text = "Record Updated Successfully";
    lblerr.MessageColor(true);
    btnaddrole.Text = "Add";
    txtCatogiryName.Text = "";
}
else
{
    lblerr.Text = "Try Again";
    lblerr.MessageColor(false);
}
}
}
catch (Exception ex)
{
    lblerr.Text = ex.Message;
}
In the above code, the string on the button is used to either insert or update the expense. If the string is “add” a call to the insert method is executed to insert the new expense name and if the string is “update” then update method is called to update the existing expense name.

The following code can be used to load the entire grid view by calling GVDataBind() function.

```csharp
DataSet dt = mybal.EmpExpCategory_Select(Session["EmpID"].ToString());
GVRoles.DataSource = dt;
ViewState["DT"] = dt;
GVRoles.DataBind();
txtCategoryName.Text = "";
}
```

### 5.6 Implementation details for adding different types of expenses

The code snapshot below shows the functionality of adding different types of expenses, a major module for the entire application. Adding Expenses provide the following functionalities

- Load expense based on the selected expense name
- Create new expenses like Airlines, Hotel, Rental Car etc
- Update details of an existing expense

Following code is used to insert the data into the grid view

```csharp
protected void btnaddExp_Click(object sender, EventArgs e)
{
    try
    {
        if (Convert.ToDouble(txtAmount.Text.Trim()) <=
            Convert.ToDouble(ViewState["CostLimit"].ToString()))
```
if (btnaddExp.Text == "Save")
{
    string FilePath = string.Empty;
    if ((FULReceipt.HasFile ||
        lnkattachement.CommandArgument.ToString().Length > 0) &&
        ChkReceiptInclude.Checked)
    {
        if (FULReceipt.HasFile)
        {
            FilePath = CheckFileUpload(FULReceipt,
                Request.QueryString["id"].ToString());
        }
        else if
        (lnkattachement.CommandArgument.ToString().Length > 0)
        {
            FilePath = lnkattachement.CommandArgument.ToString();
        }
    }

    int status =
        mybal.EmpExpenses_Insert(ddlExpType.SelectedValue.ToString(),
            Session["EmpID"].ToString(),
            txtExpenseFor.Text.Trim(),
            txtAmount.Text.Trim(),
            txtDate.Text.Trim(),
            Request.QueryString["id"].ToString(),
            ChkReceiptInclude.Checked
                ? "1": "0",
            txtHotelName.Text.Trim(),
            txtHotelNoOfDays.Text.Trim(),
            txtAirline.Text.Trim(),
            txtAirlineTicketNO.Text.Trim(),
            ddlAirlineFareBasis.SelectedValue.ToString(),
            ddlCarClass.SelectedValue.ToString(),
            txtCarRentalAgency.Text.Trim(),
            txtCarAgencyLocation.Text.Trim(), FilePath);
    if (status == 1)
    {
        if (txtDate.Text.Split('/')[0] != "10")
            string kk = txtDate.Text.Split('/')[0].Replace("0", "");
        ddlMonth.SelectedValue =
            txtDate.Text.Split('/')[0].Replace("0", "");
        else
            ddlMonth.SelectedValue = txtDate.Text.Split('/')[0];
        Clear();
        string uname =
            ((DataSet)Session["UserDT"])[0].Rows[0]["empfname"].ToString().Trim();
        GVDataBind();
        EmailContent em = new EmailContent();
        em.UserName = uname;
        em.ExpenseRejectReason = "";
        em.ExpenseReportName = "";
        em.mailType = EmailContent.MailType.ExpenseSave;
string Body = em.EmailData();

mybal.RK_SendMail(((DataSet)Session["UserDT"]).Tables[0].Rows[0]["emailId"].ToString().Trim(), "Confirmation Mail", Body);
lblerr.Text = "Record Inserted Successfully";
lblerr.MessageColor(true);
}
else
{
    lblerr.Text = "Try Again";
    lblerr.MessageColor(false);
}
else if (btnaddExp.Text == "Update")
{
    int status =
    mybal.EmpExpenses_UpDate(ViewState["empExpID"].ToString(),
txtExpenseFor.Text.Trim(), txtAmount.Text.Trim(),
txtDate.Text.Trim(), "0", txtHotelName.Text.Trim(),
txtHotelNoOfDays.Text.Trim(), txtAirline.Text.Trim(),
txtAirlineTicketNO.Text.Trim(),
ddlAirlineFareBasis.SelectedValue.ToString(),
ddlCarClass.SelectedValue.ToString(),
txtCarRentalAgency.Text.Trim(),
txtCarAgencyLocation.Text.Trim());
    if (status == 1)
    {
        GVDataBind();
        lblerr.Text = "Record Updated Successfully";
        lblerr.MessageColor(true);
        btnaddExp.Text = "Add";
    }
    else
    {
        lblerr.Text = "Try Again";
        lblerr.MessageColor(false);
    }
}
else
{
    lblerr.Text = "Expense cost exceeded the limit";
    lblerr.MessageColor(false);
}
}

catch (Exception ex)
{
}
In the above code, the data is inserted based on the string attached to the add button. In addition, the header labels to the grid view are dynamically added with the selected dates of the expense. Once a new expense is added for a particular month, E.g. October 2011, then the application automatically refreshes the page and the goes to the month of October. Therefore, the column headers are populated automatically based on the user selection.

Similarly, the data grid has to be refreshed automatically whenever the user changes the date in the calendar. In addition to adding new expenses, an expense can be deleted in a similar way. Following method is used for deleting the expense.

```csharp
protected void GVRoles_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    try
    {
        ViewState["EmpExpCatogiryID"] =
        GVRoles.DataKeys[e.RowIndex]["EmpExpCatogiryID"].ToString();
        int status =
        mybal.EmpExpCatogory_Delete(Session["EmpID"].ToString(),
        ViewState["EmpExpCatogiryID"].ToString());
        if (status > 0)
        {
            GVDataBind();
            lblerr.Text = "Record Deleted Successfully";
            lblerr.MessageColor(true);
        }
        else
        {
            lblerr.Text = "Action not completed.please try after some time";
            lblerr.MessageColor(false);
        }
    }
    catch (Exception ex)
    {
        lblerr.Text = ex.Message;
        lblerr.MessageColor(false);
    }
}
```
catch (Exception ex)
{
    lblerr.Text = ex.Message;
    lblerr.MessageColor(false);
}

}
Chapter 6
THE RESULT

This chapter describes the look & feel of the application with snapshots of several important pages. The interface for the expense management application is very intuitive. The screens are easy to understand, the controls and links on the screens are self-explanatory. It is very user friendly and quite simple to use. It is consistent throughout and requires no training to use the application. It encourages user to explore the interface and learn it without the fear of irreversible damage to the system. Meaningful error messages are provided throughout the interface to caution the user for impermissible actions.

The following screens show the snapshots of various screens available in the application. Figure 6.1 shows the home page of the application. Users with appropriate permissions can select either “Employee login” or “Admin login” to enter the credentials.
The following snapshot shows the screen for employee login. The user with employee credentials can enter the username and password to login to the system. If the user enters an invalid username or password, an error message is displayed to inform the user.
A new employee will be re-directed to change the temporary password when logging into the system for the first time as shown in the screen shot below.
If the user provides a valid authentication, they are taken to the main page of the expense application as shown in the Figure 6.4. In this page, the user can update his profile information, change password, check expenses that were filed earlier & file for new expense.
Figure 6.5 shows the employee profile update page. This webpage displays all the employee personal and financial information that is currently available with the company. If the employee wishes to update any of this information, he could do so and click on “Save”.
Figure 6.5 Employee Profile

The following snapshot in Figure 6.6 shows the screen for employee to add a new expense. The user also has options to edit the expense name, delete the expense or view the expense.
Figure 6.6 Employee Expense Page

After adding the expense name, the user can enter the details as shown in the snapshot below. The user selects the expense type from drop down menu and completes the rest of the expense information. The employee then saves this information by clicking on “Save”. The employee now has an option to add more expenses by selecting other expense types. The employee will be able to the expenses that were filed in the grid shown below along with the expense totals.
When an employee submits the expense for approval, an automated email is generated to confirm that the expense has been submitted to the manager. The snapshot of such email is shown below in the figure 6.8
Manager can also perform all the actions that were discussed in the above snapshots. In addition, the manager can view/edit & approve employee’s expenses. From the manager’s login, the manager can view the expenses filed by the employees and can either “accept” or “reject” the expense as shown in the figure 6.9 below.

**Figure 6.8 Auto Generated Email to Confirm Expense**

**Figure 6.9 Employee Expenses**
Manager can also view a report of all the expenses filed by the employees. There are several filters that could be used to filter the information. The following snapshot shows this information.

**Figure 6.10** View Reports
Finance manager’s role is to review the expenses that were accepted and to update the application when expense has been reimbursed. The following snapshot shows webpage where the finance manager can reimburse the expenses.

![Finance Manager](image)

**Figure 6.11 Finance Manager**

Admin’s role in the company is very critical. The admin performs several tasks including adding a new employee, enabling/disabling employee login, creating roles/expense types, etc. The following screen shows a snapshot of admin’s main webpage after login.
A new employee can be created by the admin by selecting “Add Employee”. Admin will then enter all the information of the new employee and provides a username along with temporary password for the employee to login for the first time as shown below in the screen shot.
If an employee leaves the company, the admin can de-activate the employee in order to prevent employee from logging into the system as shown in figure 6.14 below.
For ACG, the current requirement is to enable roles like “Admin, Employee, Manager, Senior Manager, and Finance Manager”. The following screen shot shows these roles are being enabled from Admin’s login. In future, if the firm decides on adding more roles, Admin could do so by using the screen shown below.
Admin will also have capability to add new expense type, edit/delete expense type. In addition, a limit could be set for any expense type. This limit will prevent the employee from filing expenses beyond certain amount per category.
Figure 6.16 Manage Expenses
Chapter 7

CONCLUSION

This project was developed for ACG Inc., as a one-stop solution for management of employee expenses. This application provides a faster and more efficient ways to managing the expenses. It also aims at improving the communication between the firm with its employees by providing current expense status information via emails. This system eliminates majority of manual work and thereby reducing the risk of false expense approvals.

The system has a simple user interface and easy to use. It also provides the flexibility of customization according to the changing requirements of the firm. The scope of the project has been defined in such a way that it can be submitted as a Master’s project. We can also extend some functionality in the project, which will be described in detail in the following section.

7.1 Functionality Extensions

The following functionalities can be added onto the existing system

1. The implementation of this project is done in a modular method. This will enable easy addition & implementation of any new roles that get defined in the firm

2. Import data from the existing system into the new database with any dependencies identified and prevent loss of data.
3. Integrate the current application into the firm’s financial application to streamline the re-imbursement process.

4. The firm plans to provide its employees with corporate travel credit cards that could be used on business trips. The current application could be extended to simplify the employee expense process by integrating the credit card accounts.
REFERENCES

http://www.developerfusion.com/samplechapter/1678/introducing-net/2/

http://ezinearticles.com/?What-is-ASP.NET?--Top-12-Advantages-of-
ASP.NET&id=1358877


ASP.NET, XML and ADO.NET. Pearson Education.

http://www.startvbdotnet.com/ado/default.aspx


http://en.wikipedia.org/Use_case_diagram

http://www.w3schools.com/aspnet/default.asp

http://en.wikipedia.org/wiki/Active_Server_Pages


http://psoug.org/reference/normalization.html