TIME TRACKING SOFTWARE

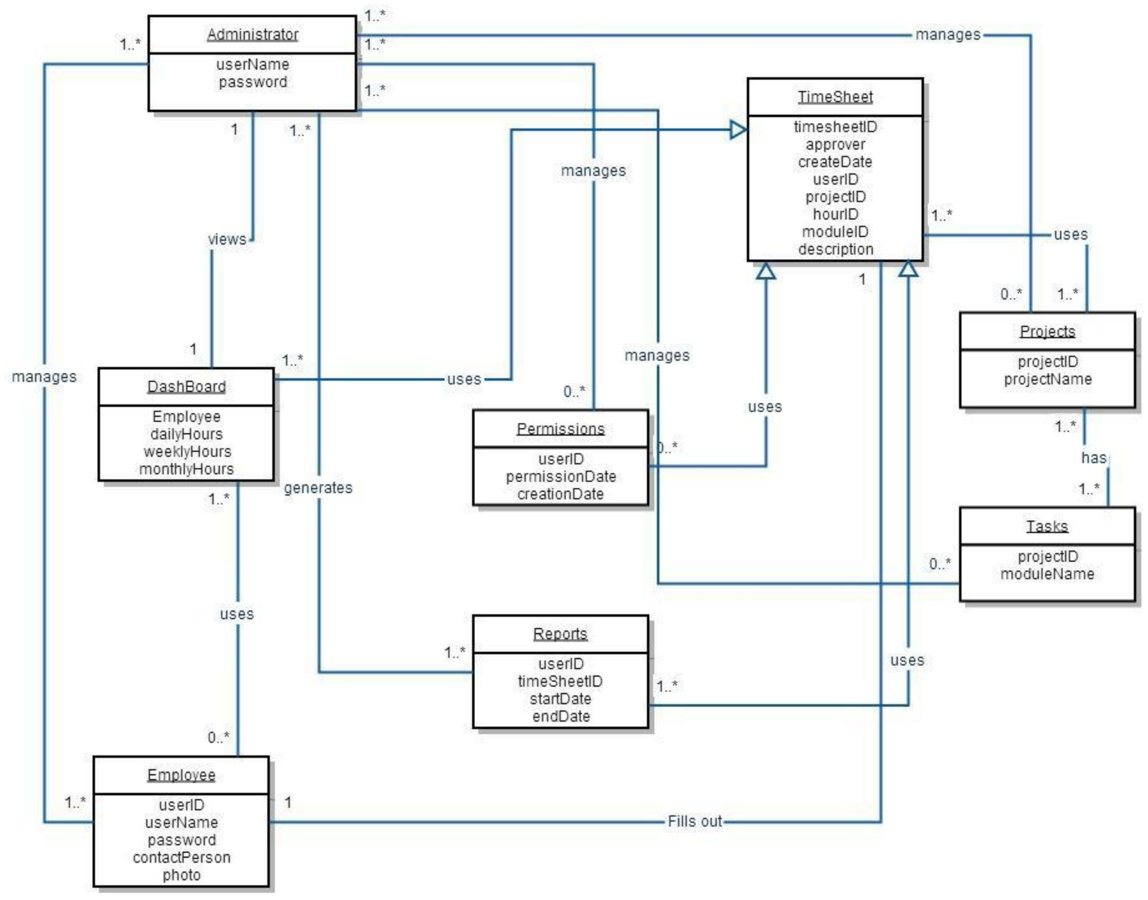
Student Name	Student ID
Chitrang Pandit	6554032
Chilat Shah	6780024
Sahil Chaudhary	6619630
Akash Kanaujia	6560180
Gautam Sidharath verma	6610900
Date : 03-march-2014	

Summary of Project

Time Tracking Software is an open source web application developed using C#, Ajax, MS-SQL Server. It enables the organization to keep records of the similar task's timelines assigned to different employees or actual time spent on particular projects. It provides reports to measure productivity and performance of individual and team which is quite helpful for better future planning.

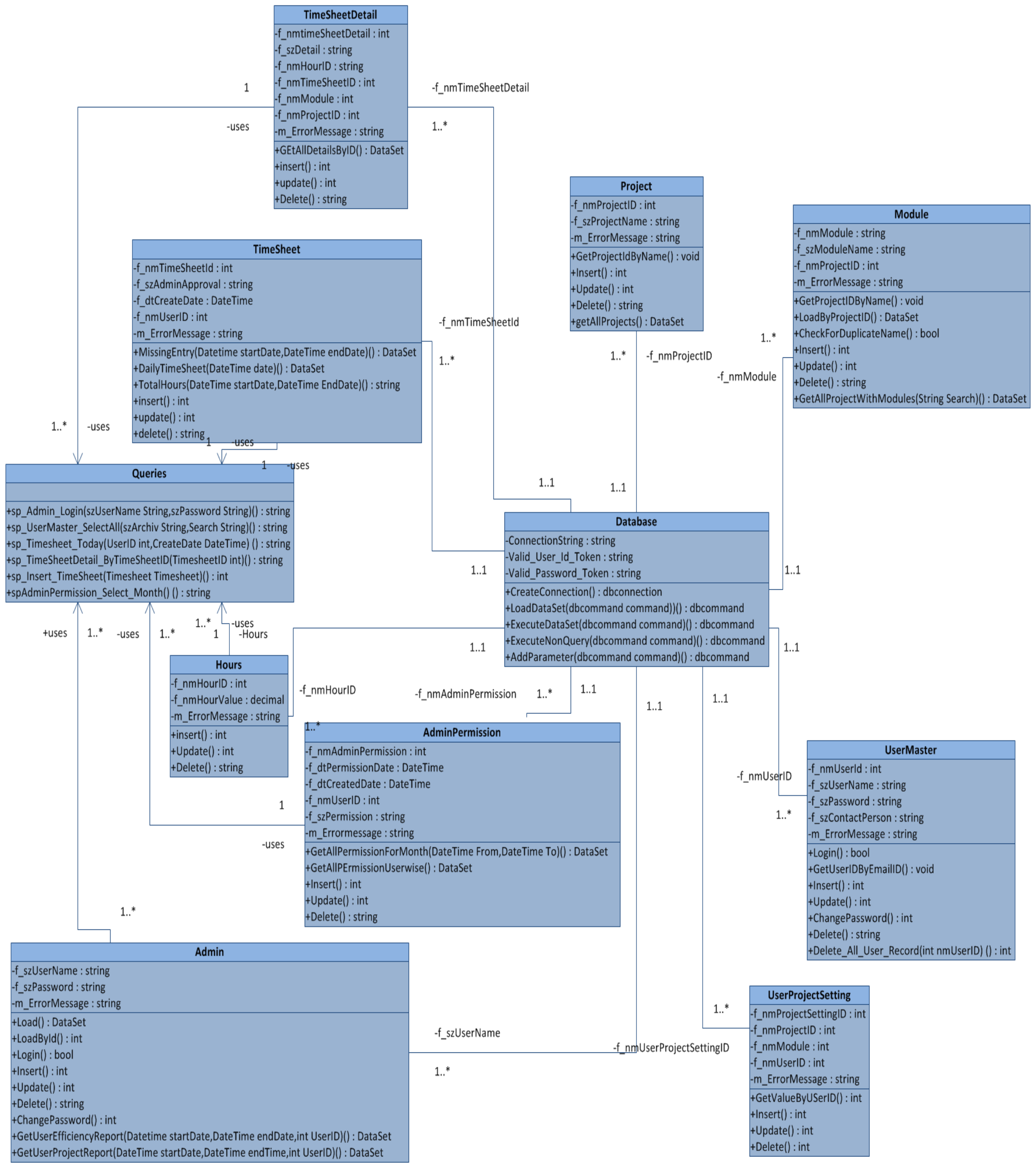
Time tracking software application consists of two modules. One for employees of the organizations – who can login to the system with their credential and enter their daily timesheet. Employee can choose project, task and enter the time to complete that task. The other one is for the Administrator of the organization – who can see all the timesheets fill by employee on its dashboard. Administrator can generate reports of timesheet by Project-wise/Employee-wise/Task-wise.

Domain Model of Actual System



Above domain model we have created as a part of M2. The next page contains actual class diagram of the system.

Class Diagram of Actual System


TimeSheetDetail
-f_nmtimeSheetDetail : int
-f_szDetail : string
-f_nmHourID : string
-f_nmTimeSheetID : int
-f_nmModule : int
-f_nmProjectID : int
-m_ErrorMessage : string
+GEtAllDetailsByID() : DataSet
+insert() : int
+update() : int
+Delete() : string
TimeSheet
-f_nmTimeSheetId : int
-f_szAdminApproval : string
-f_dtCreateDate : DateTime
-f_nmUserID : int
-m_ErrorMessage : string
+MissingEntry(Datetime startDate,DateTime endDate)() : DataSet
+DailyTimeSheet(DateTime date)() : DataSet
+TotalHours(DateTime startDate,DateTime EndDate)() : string
+insert() : int
+update() : int
+delete() : string
Project
-f_nmProjectID : int
-f_szProjectName : string
-m_ErrorMessage : string
+GetProjectIdByName() : void
+Insert() : int
+Update() : int
+Delete() : string
+getAllProjects() : DataSet
Module
-f_nmModule : string
-f_szModuleName : string
-f_nmProjectID : int
-m_ErrorMessage : string
+GetProjectIDByName() : void
+LoadByProjectID() : DataSet
+CheckForDuplicateName() : bool
+Insert() : int
+Update() : int
+Delete() : string
+GetAllProjectWithModules(String Search)() : DataSet
Queries
+sp_Admin_Login(szUserName String,szPassword String)() : string
+sp_UserMaster_SelectAll(szArchiv String,Search String)() : string
+sp_Timesheet_Today(UserID int,CreateDate DateTime) () : string
+sp_TimeSheetDetail_ByTimeSheetID(TimesheetID int)() : string
+sp_Insert_TimeSheet(Timesheet Timesheet)() : int
+spAdminPermission_Select_Month() () : string
Database
-ConnectionString : string
-Valid_User_Id_Token : string
-Valid_Password_Token : string
+CreateConnection() : dbconnection
+LoadDataSet(dbcommand command)() : dbcommand
+ExecuteDataSet(dbcommand command)() : dbcommand
+ExecuteNonQuery(dbcommand command)() : dbcommand
+AddParameter(dbcommand command)() : dbcommand
Hours
-f_nmHourID : int
-f_nmHourValue : decimal
-m_ErrorMessage : string
+insert() : int
+Update() : int
+Delete() : string
AdminPermission
-f_nmAdminPermission : int
-f_dtPermissionDate : DateTime
-f_dtCreatedDate : DateTime
-f_nmUserID : int
-f_szPermission : string
-m_Errormessage : string
+GetAllPermissionForMonth(DateTime From,DateTime To)() : DataSet
+GetAllPErmissionUserwise() : DataSet
+Insert() : int
+Update() : int
+Delete() : string
UserMaster
-f_nmUserId : int
-f_szUserName : string
-f_szPassword : string
-f_szContactPerson : string
-m_ErrorMessage : string
+Login() : bool
+GetUserIDByEmailID() : void
+Insert() : int
+Update() : int
+ChangePassword() : int
+Delete() : string
+Delete_All_User_Record(int nmUserID) () : int
Admin
-f_szUserName : string
-f_szPassword : string
-m_ErrorMessage : string
+Load() : DataSet
+LoadById() : int
+Login() : bool
+Insert() : int
+Update() : int
+Delete() : string
+ChangePassword() : int
+GetUserEfficiencyReport(Datetime startDate,Datetime endDate,int UserID)() : DataSet
+GetUserProjectReport(DateTime startDate,DateTime endTime,int UserID)() : DataSet
UserProjectSetting
-f_nmProjectSettingID : int
-f_nmProjectID : int
-f_nmModule : int
-f_nmUserID : int
-m_ErrorMessage : string
+GetValueByUSerID() : int
+Insert() : int
+Update() : int
+Delete() : int
1
-uses
-f_nmTimeSheetDetail
1..*
-f_nmTimeSheetId
1..*
1..*
-f_nmProjectID
1..*
-f_nmModule
1..*
-uses
1
uses
1
-uses
1..1
1..1
1..1
1..1
+uses
1..*
-uses
1..*
1..*
1
-uses
-Hours
1..1
-f_nmHourID
-f_nmAdminPermission
1..*
1..1
1..1
1..1
1..1
1..*
-f_nmUserID
1..*
1
-uses
1..*
-f_szUserName
-f_nmUserProjectSettingID
1..*
1..*

Class diagram which we generated after doing Reverse engineering are shown above. The domain model is what we thought the system should look like and base on that we created conceptual classes. Where else Class Diagram is what we got after applying Reverse engineering to the given code.

Here, **Class diagram (CD)** of system is similar in many ways as what we have imagined in domain model (DM) in milestone 2. So like CD : **Admin class** is equals to DM : **Administrator** as both classes Authenticate ,insert ,update ,Delete of Administrator module. CD: **Admin Permission** is relates to class DM: **Permission** as both classes share the same functionality of providing access to admin to fill employee before dates timesheet. CD: **Hours** and DM: **Dashboard** are same. These both classes used to insert, Update and View the hours of employees of respective project modules.CD: **Project** is related to DM : **Projects** as both classes saves and retrieves Project information. CD: **Module** is equals to DM : **Task** as both classes contains the information about the tasks that are assign to employees as well as total number of task in project.

However there are certain differences in Domain Model (DM) and Class Diagram (CS).for instance DM :**Timesheet** which is divided in two separate classes in CD : **Timesheet ,Timesheet Detail** where CD: **Timesheet** contains the overview regarding projects where **TimesheetDetail** contains all the detail information about Timesheet of projects., same fundamentals applied to DM : **Employee** which divides in two classes CD : **User Master and User Project Setting**. The main surprising part is DM: **Reports class** to generate the report from system but CD has no class related to this as Report portion is **integrated in CD: Admin Class.** The Admin class make a call to database for generating the reports, which could be saved later by user locally.

Here in this project it completely follows MVC architecture. Because of that all classes are not directly interact with each other but its interacting through Model Layer of MVC. Majority of classes are directly connected to class name **"Queries"** which contains constants to get the values of query and based on that it interacts with database. So **"Database"** works as an interface that every class is passing through it, and from that it interacts with other classes indirectly.

Since we are concentrating on Admin module of the Time Tracking Software and project follows **Model-View-Controller** architecture. We have considered the business logic for class diagram which is described in Controller and its interaction with Model to access the database.

We tried to use some of available reverse engineering tools for extracting the class diagram from source code. The best we found is in-built plug-in for visual studio 2010 which generates the .cd files and renders the class diagram. While investigating further for association between classes we found "**AutoDiagrammer"** tool from code project website. This tool takes .dll files as an input and generates the class diagram including associations between classes. On the basis of class diagram generated by tool we used Microsoft Visio 2010 to create class diagram manually.

Relationship between Classes & Sample Code

We are considering Admin.cs class of controller and Database.cs of Model of MVC design pattern and put Admin Login method code which interacts with database for authentication.

```
public class Admin
{ #region Login
    public bool Login()
    {
      DataSet loDataSet = null;
      try
      {
        // default database service is determined through configuration.
        Database db = DatabaseFactory.CreateDatabase();         This line creates an object of database class of Model layer

        string sqlCommand = "sp_Admin_Login";
        DbCommand dbCommand = db.GetStoredProcCommand(sqlCommand);
          //Add Parameter.
        db.AddInParameter(dbCommand, "szUserName", DbType.String, this.szUserName);
        db.AddInParameter(dbCommand, "szPassword", DbType.String,clsSecure.EncryptText(this.szPassword));

        loDataSet = db.ExecuteDataSet(dbCommand);       Passes parameters of database ExecuteDataSet method.

        if (clsFunctions.GetRowCountOfDataSet(loDataSet, 0) == 0) return false;

        return true;
      }
      catch (Exception ex)
      {
        this.m_ErrorMessage = ex.Message;
        return false;
      }
    }
}
```

Above code snippets shows interaction between two classes for authenticating the Admin user of system. In return **database.cs** class executes stored procedure **"sp_Admin_Login"** in SQL server. In case authentication fails the Controller **Admin.cs** class returns false to view class **AdminLogin.aspx.cs.**

Code Smells and System Level Refactoring

Code Smells:

Time tracking software is following MVC design pattern for implementation. The implementation has View as user interface, controller which handles user interaction interacts with the model and eventually select a view to render the displays. The model interacts with database, retrieves and stores the data in the database.

We have found 3 major code smells in the Time Tracking Software:

- Feature Envy
- Duplicate Code
- Large Class

Following is the description of the code smells with the code snippets:

Feature Envy: The model layer of MVC design pattern handles the interaction with database and executes the query according to the parameters or values passed by the controller. The business logic is handled by the controller layer. In our project we found that controller is also working as model and handles some of the tasks that should be handled by model. Controller defines the queries need to be executed at database level. In future in case underlying database is changed, we need to change both controller and model layers.

Code snippets:

Code in RED should be part of another class in Model.

```
public DataSet GetUserProjectReport(DateTime dtStartDate,
                                        DateTime dtEndDate,
                                        int lnUserID)
    {
      // Create the Database object, using the default database service. The
      // default database service is determined through configuration.
      Database db = DatabaseFactory.CreateDatabase();

      DbCommand  dbCommand  =  db.GetSqlStringCommand(Queries.GetUserProjectReport(dtStartDate,  dtEndDate,
lnUserID));
      // DataSet that will hold the returned results
      DataSet loDataSet = null;
```

```
        loDataSet = db.ExecuteDataSet(dbCommand);

        // Note: connection was closed by ExecuteDataSet method call

        return loDataSet;
    }
```

Queries is defined in another class **Queries** as of part of controller. The controller should pass the values and decide the business logic. It basically defining the logic using query but it should delegate this query design and execution to the model and work on dataset return by the model. We will **extract** the method and refactor by **Extract class** and introduce new class in the model layer. The cohesion

Duplicate Code: In our project we found another code smell as duplicate code. We went throw code description of all classes and we found that database object is created for all methods of class and same piece of execution code is common.

Code snippets:

```
public int Insert()
    {
      try
      {
        // Create the Database object, using the default database service. The
        // default database service is determined through configuration.
        Database db = DatabaseFactory.CreateDatabase();

        //create command
        string sqlCommand = "sp_Admin_Insert";
        DbCommand dbCommand = db.GetStoredProcCommand(sqlCommand);

        // Set Parameters.
        db.AddInParameter(dbCommand, "szUserName", DbType.String, this.szUserName);
        db.AddInParameter(dbCommand, "szPassword", DbType.String, clsSecure.EncryptText(this.szPassword));

        //Execute NonQuery.
        db.ExecuteNonQuery(dbCommand);
        return (int)clsMessage.RETURN_STATUS.OK;

      }
      catch (Exception ex)
      {
```

Code in Red is repeated in all method of class.

```
        m_ErrorMessage = ex.Message;
        return (int)clsMessage.RETURN_STATUS.NOT_OK;
      }
    }
```

We will use **Extract Method** refactoring to refactor this code smell in the project. The new method will contain common code and all other methods will refer it. We will create database object in new method and that method will interact will Model layer.

Large Class: In our software we also found that some of the classes have too much instance variable, which are trying to do too much.

The controller classes of the project are implementing the model side code and working on database to make class more cohesive we will use Extract class refactoring and create new class in model layer which interacts between database class and controller classes. The controller class only pass data values and decide the logic which needs to be executed at model layer.

The code for this code smell is same as **Feature Envy** mentioned above.

Anticipated System Refactoring:

Rename method:

The method name in the code is confusing with the name of standard Application Programming Interface (API)'s variable names. For example: "**ExecuteDataSet**" method of Database class in Model layer is same as "**ExecuteDataSet"** of data library of dot net. The name should be more specific to the project operation like "**TimeSheetExecuteDataSet**" which would be more readable for code comphresion.

The above rename method is applicable for all methods which has similar name as standard API's naming convention.

Extract Class:

Currently, controller class has responsibilities to decide the SQL queries and has visibility of underlying database by passing the database specific queries. Ideally, the database queries needs to decide at Model level in MVC and can retrieve the data and store it. The controller needs to execute the logic and pass the parameters on the basis of logic to Model.

To resolve the feature envy and large class we will introduce a new class at Model level which gets parameters from Controller and executes them using data access object class.

Extract Method:

Since, we found the duplicate code in the classes which increases the class size and results in redundancy. We will use Extract Method refactoring to resolve duplicate code smell and make a single method which could refer from another methods for common code execution. Hence, the size of class will be reduced and duplicity will be removed without changing the behavior of the system. The project has most of the classes which needs to remove the redundant code.

Refactor example : Refactored extract method

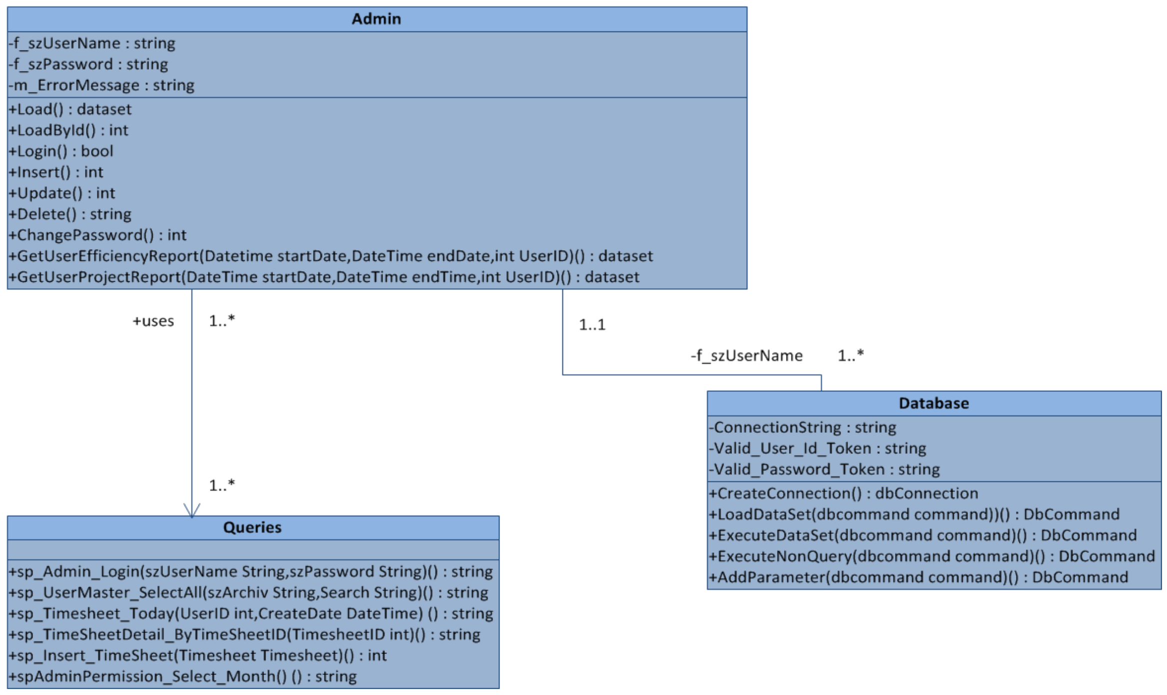
```
public DbCommand getDBCommand(string sqlCommand)
      {
          Database db = DatabaseFactory.CreateDatabase();
          DbCommand dbCommand = db.GetStoredProcCommand(sqlCommand);

          return dbCommand;
      }
```

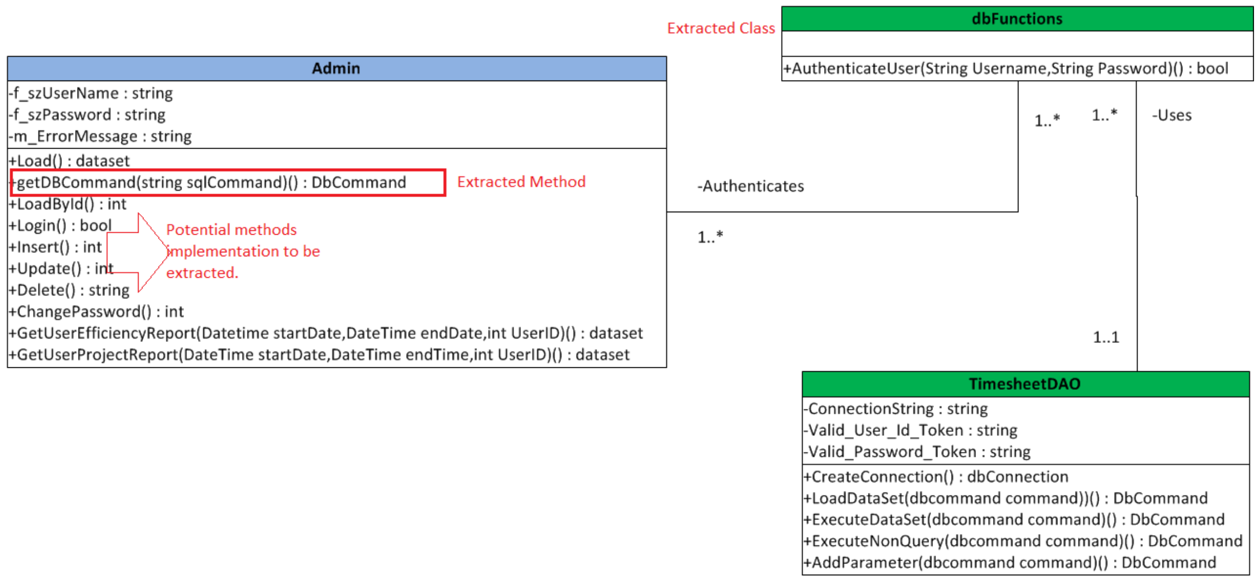
Refactoring in Architecture of system

The following architectures show the architecture before and after the refactoring:

Before Refactoring:


Admin
-f_szUserName : string
-f_szPassword : string
-m_ErrorMessage : string
+Load() : dataset
+LoadById() : int
+Login() : bool
+Insert() : int
+Update() : int
+Delete() : string
+ChangePassword() : int
+GetUserEfficiencyReport(Datetime startDate,DateTime endDate,int UserID)() : dataset
+GetUserProjectReport(DateTime startDate,DateTime endTime,int UserID)() : dataset
+uses
1..*
1..*
1..1
-f_szUserName
1..*
Database
-ConnectionString : string
-Valid_User_Id_Token : string
-Valid_Password_Token : string
+CreateConnection() : dbConnection
+LoadDataSet(dbcommand command))() : DbCommand
+ExecuteDataSet(dbcommand command)() : DbCommand
+ExecuteNonQuery(dbcommand command)() : DbCommand
+AddParameter(dbcommand command)() : DbCommand
Queries
+sp_Admin_Login(szUserName String,szPassword String)() : string
+sp_UserMaster_SelectAll(szArchiv String,Search String)() : string
+sp_Timesheet_Today(UserID int,CreateDate DateTime) () : string
+sp_TimeSheetDetail_ByTimeSheetID(TimesheetID int)() : string
+sp_Insert_TimeSheet(Timesheet Timesheet)() : int
+spAdminPermission_Select_Month() () : string

After Refactoring:


Extracted Class
dbFunctions
+AuthenticateUser(String Username,String Password)() : bool
Admin
-f_szUserName : string
-f_szPassword : string
-m_ErrorMessage : string
+Load() : dataset
+getDBCommand(string sqlCommand)() : DbCommand
Extracted Method
+LoadById() : int
+Login() : bool
+Insert() : int
+Update() : int
+Delete() : string
+ChangePassword() : int
+GetUserEfficiencyReport(Datetime startDate,DateTime endDate,int UserID)() : dataset
+GetUserProjectReport(DateTime startDate,DateTime endTime,int UserID)() : dataset
Potential methods implementation to be extracted.
-Authenticates
1..*
1..*
1..*
-Uses
1..1
TimesheetDAO
-ConnectionString : string
-Valid_User_Id_Token : string
-Valid_Password_Token : string
+CreateConnection() : dbConnection
+LoadDataSet(dbcommand command))() : DbCommand
+ExecuteDataSet(dbcommand command)() : DbCommand
+ExecuteNonQuery(dbcommand command)() : DbCommand
+AddParameter(dbcommand command)() : DbCommand

The above diagrams show the effect of refactoring on the architecture.

dbFunctions Class: This class is the result of **Extract Class** refactoring which we need to create in Model layer and implememts the functionality which is currently resides in Controller layer classes, which increases the cohesiveness of Controllers class and reduce the coupling the between database and controller layer. For example: Query for Login will be part of this class and admin class of controller only pass the parameters. We rename **database** class to **TimeSheetDAO** to reduce naming complexity.

getDBCommand(string SqlCommand) : This method is the result of **Extract Method** refactoring which we have applied on Controller classes. For example: In current system, each method is creating an object of database class and passing the parameter using dbCommand. Now, we have created a common method to create a database object and which returns dbCommand. This we need to do for all classes on Controller layer.

```
public DbCommand getDBCommand(string sqlCommand)
      {
          Database db = DatabaseFactory.CreateDatabase();
          DbCommand dbCommand = db.GetStoredProcCommand(sqlCommand);
          return dbCommand;
      }
```