**The Project**

1) Employee data must include these properties: first name, last name and manager

2) A page to find an employee by name. If employee is not found, provide a link to enter a new

employee. If an employee is found, provide a link to edit.

3) A page to enter/edit employee record

In the business rule tier, please regulate the following rules:

1) An employee cannot be his/her own manager

2) Ensure new employee entry does not already exists

**Introduction**

This documents the architecture design, components and various considerations related to the above project. The design goal is to meet the requirements of documented in the project. The project provides the stage of the design and dictate many consideration. Some of basic rules that are focus of this design include:

Separation of concerns. The application is composed of three modules/layers, Data Access, Business and application. Data access layer exposes an interface IDac though a factory class, so that the development of the data access will separate from other development. Other development does not need any knowledge of data access layer and the IDac interface is only thing they need to know and program against it.

Data and function encapsulation. Limits the public interface to minimum, so that to reduce the confusion and maintain high object integrity. For example, to create a customer constructor, you have to also create a default constructor. But the default constructor, at least in the simple application, is basically useless and could be abused. This design makes it private. in the same consideration, the SQLDac class is defined internal.

Interface and factory patterns are used to separate data access layer from other parts. The switch database from SQL server to Oracle, from relational database to network, for example, will transparent to other development.

Reduce coupling and enhance cohesion is also reflected in the design and development. Refactoring code, limiting size of the function and promoting code cohesion.

.

**Overall architect**

Database

CBRDac

CBREmpLib

CBREmp Web

**Web Application Architect**

The web application is developed using MVC ( Model, View, Contorller), version 3.0 with Razor rendering engine. Separating of concern, promoting testing and TDD(test driven development), Facilitation web search are among the benefits of the MVC architecture. The benefit of MVC will not be discussed here.

Model (business logics)

will typically be inside the web application itself. But in the development, They are independently developed. Separating them from web application makes them reusable to other development.

The business layer actually includes The CBRDac and CBREmpLib. The internal of CBRDac is developed using Microsoft Entity Framework (Previous implementation ADO.net was used). It expose IDac interface for CBREmpLib to consume, so that that they can be developed independently and replaced independently. With a little modification, the previous Dac will fit into this architect easily.

IEmpRepository

MockEmpRepositopry

EFEmpRepositroy

EFEmpRepository -- represents EF data

MockEmpRepository -- represents mock data for test

Controller

is used as bridge between user and the business layer. The current Controller EmpManCotroller can handle actions: Search and Edit.

The router has been configured to default to EmpMan and search to meet this application's requirement.

View

Views displays/formats data and provide interfaces to the user. The view engine in this implementation is Razor.

**Unit tests**

The unit tests are implemented using unit test project template. It is part of Microsoft technologies.

**Database**

CBREmp table

CREATE TABLE [dbo].[CBREmp](

[EmpID] [varchar] (100) NOT NULL,

[FirstName] [varchar](100) NOT NULL,

[LastName] [varchar](100) NOT NULL,

[Manager] [varchar](100) NULL)

ALTER TABLE [dbo].[CBREmp] ADD CONSTRAINT [PK\_CBREmp] PRIMARY KEY CLUSTERED

([EmpID] ASC)

GO

ALTER TABLE CBREmp

ADD CONSTRAINT fk\_EmpManager FOREIGN KEY(Manager)REFERENCES CBREmp(EmpID)

GO

Comments:

The table defines only columns specified in the project doc. The manager column data is composed from firstname, lastname in the form of <lastname>, <fisrtName> format.

An EmpID would be helpful.

Manager should be a foreign key to the table. Since the current EF do not be able to handle self reference correctlty. The foreign key is not set in this implementation.

**CBRDac**

IDac

EFDac

This the data access layer and is holding in a class library (dll). The class "SQLDac" is defined as internal and it implements IDac.

public interface IDac

{

// retrieve empoyee information by name

DataTable GetEmp(string fn, string ln);

// retrieve all employees for make manager selection dropdown

DataTable GetEmps();

// insert new emp to emp table

bool InsertEmp(string ln, string fn, string mn);

// update emp info

bool UpdateEmp(string ln, string fn, string mn);

}

internal class EFDac : IDac // Entity framwork dac

{

...

}

The class "DacFactory" is also defined in this dll and it is only class exposed to the library user besides IDac Interface.

private DacFactory() { }

public DacFactory(string dbType, string conn)

{

MyDBType = dbType;

ConnStr = conn;

}

The "DacFactory" has private default constructor and one public custom constructor. It's main function is

public IDac GetDac(string myDBType, string connStr)

It returns an IDac depending on the parameter passed in so that isolates the actually implementation of data access inside the library

**CBREmpLib**

It is a class library and represents business layer of the application. It does not know the existence of Dac except there is a interface called IDac. It use DacFactory to require a IDac and use IDac to get required database functionality.

private EmpLib() { }

public EmpLib(string dbType, string conn)

{

if (\_dac == null)

{

\_dac = GetDac(dbType, conn);

}

}

It is also the place the business object tree to be build and to serve the web application

commnents:

Employee and manager has is-a relationship and good candidate for a bass class and derived class relationship. But Since the relationship is very simple and not much that the objects could do, and the project does not require any specific functionalty to manager. Actually many cases asp.net have a very efficient way to had database table. So in this implementation I am not create object for these entities.

**CBREmp Web**

The web application is consists of Model, View and Controller.

Master page is used for look and feel of the whole site.

CSS is used for many visual factors

Client/server site validation for data integrity

Jquery and Javascript are used for some of the function

Config file has been used for customize and deployment

**CBREMP Web Test**

EmpLibTest is test class for EmpLib.

The test is configurable by provide parameters when constructing lib = new CBREmpLib("ef"); Currently "ef" -- testing entity framework directly.

"mock" -- testing emplib without the Entity framework

The mock will use in memory storage objects to mimics the real data storage and focus to testing EmpLib.

The "ef" tests real entity framework , but could point to database to test both emplib and dac behind it.

To facilitate above feature, a Mock repository class is created. It creates a list of test data parsed to mock Dac..

Mock Dac is created to hold and mimics the search, save function of real Dac.

**Test cases:**

HasSixRecords()

The Mock the repository is initialized with 6 records. The test will typically fail after an insertion.

EmpFound()

Provide an existing emp, perform a search and compare result with provided emp.

EmpNotFound()

Provide a emp known not exist, perform a search and make sure return 0.

AddOneRecords()

Add an emp and check no error returned.

InsertExist()

Insert an emp and repeat the insertion with added emp, and verify "Record Exists already" will be the return

UpdateNotExist()

Update an emp that does not exist and verify that "records not Exist!" will return.