Design Document – UPS Assignment

Contents

[1 Summary 2](#_Toc55999024)

[2 MVVM Overview 2](#_Toc55999025)

[2.1 Model Classes 4](#_Toc55999026)

[2.1.1 Interfaces and classes 4](#_Toc55999027)

[*2.1.2* Singleton Pattern 6](#_Toc55999028)

[2.2 View Model Classes 7](#_Toc55999029)

[2.3 View Classes 8](#_Toc55999030)

[2.4 Derived Classes 9](#_Toc55999031)

[2.5 Dependency Injection 10](#_Toc55999032)

[2.6 Deployment Structure 10](#_Toc55999033)

[2.6.1 Config Folder 10](#_Toc55999034)

[2.6.2 Bin Folder 10](#_Toc55999035)

[2.6.3 Logs Folder 10](#_Toc55999036)

Table of Figures

[Figure 1: MVVM Architecture 3](#_Toc55999199)

[Figure 2: Model class 4](#_Toc55999200)

[Figure 3: Interfaces 5](#_Toc55999201)

[Figure 4: Derived classes 9](#_Toc55999202)

[Figure 5: Dependency Injection 10](#_Toc55999203)

[Figure 6: Deployment Structure 10](#_Toc55999204)

[Figure 7: Application configuration 11](#_Toc55999205)

# Summary

“UPS Employee Information System” is used to manage the employee details. This document explains the technical aspects concerning the architecture and design pattern used in this tool.

This tool has been designed as a desktop application. Below table explains the technologies being used altogether.

|  |  |
| --- | --- |
| Architecture Pattern/Style | Model View “ViewModel” (MVVM) |
| Design Principles | Interface Segregation, Single Responsibility |
| Design Patterns | Singleton, Dependency Injection and Strategy |
| Programming Language | C#.Net (.NET Framework 4.6.1) |
| Nuget Packages | log4net.2.0.12 and Newtonsoft.Json.12.0.3 |
| UI Design | WPF |
|  |  |

Below are my findings.

* The WEB API GET works with any API Key
* Only the return HTTP status code does not guarantee the success. Need to verify the content of the response

Below features are not implemented due to time constraint

1. I could not apply mocking, since I have designed the connection handler as the singleton and sealed. Also this small project does not have scope for mocking. I have kept some mocking code in the Unit Testing project for the reference
2. I have left out the “Remove” and “Export” functionalities. I have implemented the “Remove” in the model layer, but did not link it to view due to the time constraint
3. Viewing the last page if the input is more than the number of pages is not done

# MVVM Overview

MVVM is the architectural style in which the code is organized to follow the one or more of SOLID design principles. The main goal of this architecture is to have loose coupling between various layers and reducing the hard binding between them for better maintainability.

Model – This layer holds the data and has nothing to do with any of the business logic.

ViewModel − It acts as the link/connection between the Model and View. It reads the data from Model and change them if required to provide what is to be presented in the View

View − It simply holds the formatted data and essentially delegates everything to the ViewModel.

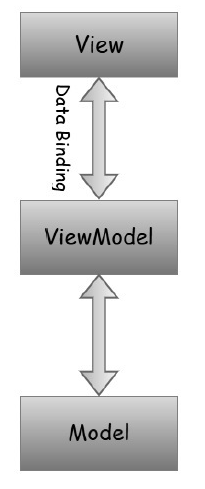


Figure 1: MVVM Architecture

Following sections explain the classes used in “UPS Employee Information System” tool in all layers of MVVM.

## Model Classes

### Interfaces and classes

+int Id

+string Name

+string Email

+string Gender

+string status

Employee

Figure 2: Model class

+ ConnectionInfo

+AuthenticationInfo

<interface>

IConnectionHandler

+Connect

+ NoofRecordsOrPages

<interface>

IEmployeeQueryHandler

+GetEmployeesAsync()

+GetEmployeesByNameAsync(string)

+GetEmployeeById(int)

+ NoofRecordsOrPages

<interface>

IEmployeeUpdateHandler

+AddEmployee (): Employee

+RemoveEmployee (int): bool

+EditEmployee (int): bool

Figure 3: Interfaces

### Singleton Pattern

“ConnectionAgent” is the singleton class that has connection properties to either web server or database server. Other classes in the system uses this singleton object to retrieve the connection details like “User ID”, “Project URL” and so on.

+ConnectionInfo

+AuthenticationInfo

-ConnectionAgentWebServer

-\_mutexObject

-\_connectionHandlerLogger

ConnectionAgent: IConnectionHander

+GetInstance(string, string): instance

+ Connect(int):int

## View Model Classes

+TotalNoOfPages

+RetrievedEmployees

EmplyeeViewModelQuery

+EmployeeId

+EmployeeName

+EmployeeEmail

+EmployeeGender

+EmployeeStatus

EmployeeViewModelUpdate

: INotifyPropertyChanged, IDataErrorInfo

+ <async> GetEmployeesByPageAsync (int)

+ <async> GetEmployeesAsync (string)

<event> PropertyChangedEventHandler

+UpdateEmployee(): bool

## View Classes

UpdateEmployeePage

QueryEmployeePage

## Derived Classes

Following classes have been implemented as derived classes.

+ bool IsOperationSucceeded

+ string MessageToShow

MessageToUserEventArgs

EmployeeQueryHandlerWebServer

EmployeeUpdateHandlerWebServer

IEmployeeUpdateHandler

+ Message

EventArgs

IEmployeeQueryHandler

Figure 4: Derived classes

## Dependency Injection

After successful login, the object of “IConnectionHandler” is created as a singleton object in “LoginWindow” and injected into other objects in sequence via constructors until it reaches the class where the connection to the database or web server is used.

+ IConnectionHandler

LoginWindow

- IConnectionHandler

EmployeeQueryHandlerWebServer

- IConnectionHandler

WebApiClient

Figure 5: Dependency Injection

## Deployment Structure

“Employee Information System” is deployed in the following folder structure.

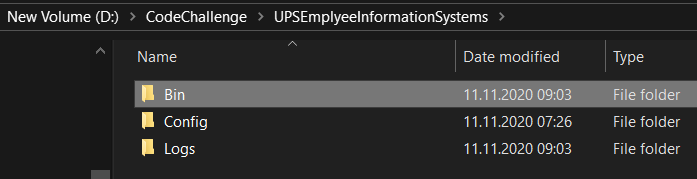


Figure 6: Deployment Structure

### Config Folder

This folder contains the application configuration and the logging configuration

### Bin Folder

This folder contains the binary files required for the application

### Logs Folder

This folder contains the log files where information and error messages are logged

#### Application configuration

Connection configuration is stored in the “App.Config” file as shown below. This information is mainly used while debugging. Otherwise the credentials are asked in the login screen.

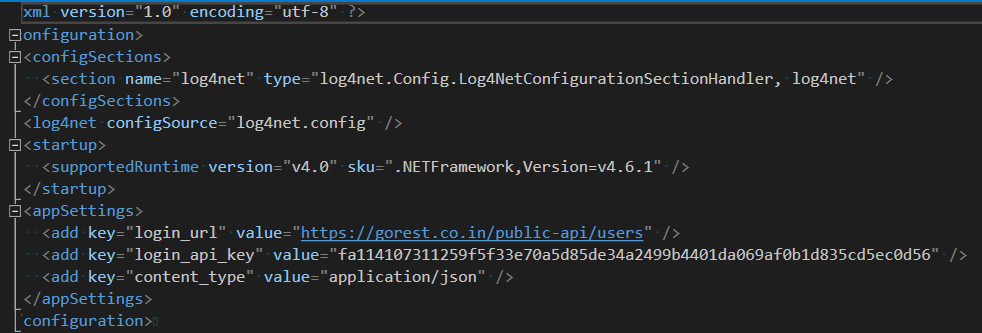


Figure 7: Application configuration