Colorectal Cancer Survey TD

Technical Design Specification

Project name	Colorectal Cancer Survey
Project Supervisor	Dr. François Modave
Modification date	16 DEC 2014
Document Version	2.0
Status	Release
Author	Aria Moshari
Component Version	2.0

Document History

Version	Date	State	Author	Summary of changes
1.0	10 Jun 2014	Draft	Aria Moshari	Original document
1.1	03 July 2014	Draft	Aria Moshari	On section 3.2 Development Technologies: Telerik AppBuilder is replaced by VS Nomad extension tools for developing Mobile Application component. Both of them are using PhoneGap (Apache Cordova) as open-source mobile development framework. As AppBuilder is not a free CASE tools is replaced by VS Nomad which is free.
2.0	16 Dec 2014	Draft	Aria Moshari	Add description of the component in implementation environment

Reviews

Name	Title	Version	Date
Dr. François Modave	Project Supervisor	1.0	25 Jun 2014
Dr. François Modave	Project Supervisor	2.0	31 DEC 2014

Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose

This document describes the design of the Colorectal Cancer Survey project.
The Colorectal Cancer Survey software offers different types of online applications to facilitate patient-physician communication. Patients (as target users) can use the applications of this project to interact with decision-making process.
Also, all of the required patient data and information will be exposing from the central database.

1.2 Scope

The document is aimed at software engineers involved in the development and maintenance of the Colorectal Cancer Survey software.

Technically speaking the Colorectal Cancer Survey limits itself to performing certain type's patient information and base on the decision making engines, it will provide results and information to the user.

1.3 References

Reference ID	Document ID / Location	Description
001	Non-additive Measures: A Theoretical Approach to Medical Decision-Making	Research Article by Francois Modave and Navkiran K Shokar
002	MSDN: Designing Web Applications	The guidelines for a layered structure; guidelines for performance, security, and deployment
003	MSDN: Designing Mobile Applications	Key design considerations for mobile applications
004	MSDN: Communication and Messaging	It is a rich technology foundation designed for building distributed service-oriented applications.
005	Architectural Styles and the Design of Network-based Software Architectures Link: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm	Based on the Roy Thomas Fielding theory. Representational State Transfer (REST) Service.
006	Identification of fuzzy measures from sample data with genetic algorithms	Research Article by Pedro Miranda

1.4 Terms & Abbreviations

Term or Abbreviation	Description
API	Application Program Interface
CRC	Colorectal Cancer
DAL	Data Access Layer
BLL	Business Logic Layer
UI	User Interface
DAO	Data Access Object(s)
DLL	Dynamic Link Library. A file containing code that is installed in the computer file system.
WCF	Windows Communications Foundation (.NET 3.0)
XML	Extensible Markup Language

2 Applications and Components

All of the components and applications are always belongs to one of the front-end or back-end part of the system.
It is important to understand the definition of the front-end and back-end concept in the software design. In Figure 1, components of the system are distinguished and categorized into each mentioned parts.

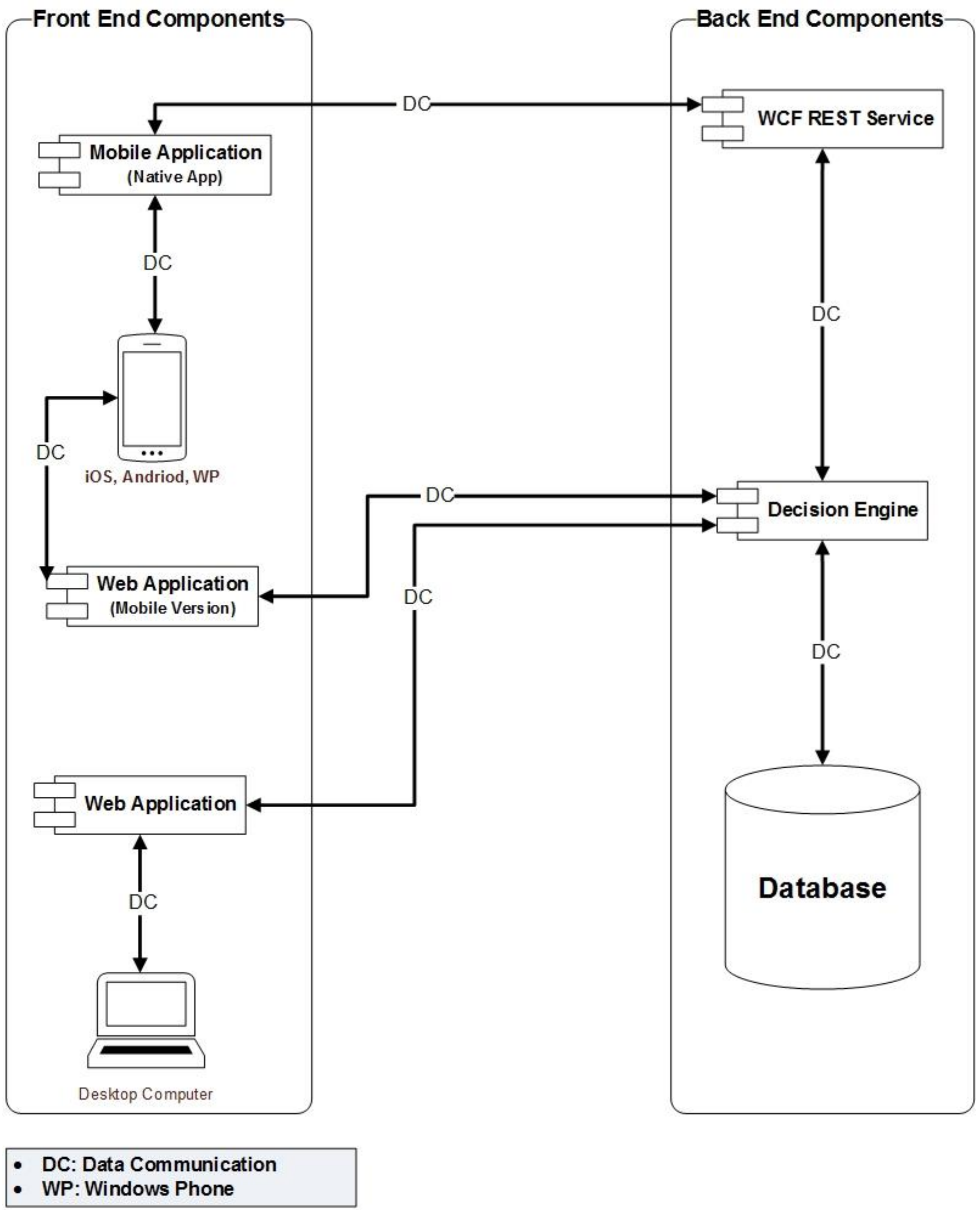


Figure *1* – Applications and Components

2.1 Front-End Components

Front-end applications and components are performed by end-users (target users). In other words, the main user of this components are end/final users of the system. Here, patients are the user of the applications that interact with components of front-end applications.

2.1.1 Web Application

Like all other web applications, this is the website that reachable from the internet network. Here, we assume that users of this application will use their desktop computer (Mac, PC) and installed browsers (IE, Safari, Firefox, Chrome, etc.) to open this website and interact with it. All functionalities, should be accessible from this web application to serve our target users.

2.1.2 Web Application (Mobile Version)

This application is like what was described above, with one different: the layout and other UI parts are designed to present on mobile browsers. That is why is called mobile version. In fact, this is application is a website that is accessible only from browser of the mobile devices. For example, accessing website from Safari browser of iOS on iPhone or iPad.

2.1.3 Mobile Application

This application is a native mobile application that will be installed locally on mobile devices. This mobile application should be able to installed and worked on all mainly different platforms such as iOS, Android and Windows Phone (cross-platform mobile application). This application will communicate WCF REST Service to send/receive data to/from the system.
Moreover, it is designed to communicate with servers via internet network and only will send and get data from system to server main services. It will never include the main logic of the project.

2.2 Back-End Components

Back-end applications and components are performed and administrated by operation team of the project. In other words, the main user of this components are operational team or administrators of the system. That would be happened in the maintenance mode of the project.

2.2.1 WCF REST Service

A man named Roy Thomas Fielding first coined the term REST as a concept in his PhD dissertation ("Architectural Styles and the Design of Network-based Software Architectures"). REST emphasizes things like separation of concerns and layers, statelessness, and caching, which are common in many distributed architectures because of the benefits they provide. These benefits include interoperability, independent evolution, interception, improved scalability, efficiency, and overall performance."

Actually only the difference is how clients access our service. Normally, a WCF service will use SOAP, but a REST service, clients will be accessing service with a different architectural style (calls, serialization like JSON, etc.).

A WCF REST service can be categorized as *resource-based*, which means that a REST client sends an HTTP request to programmatically accomplish a business objective. These requests largely employ a **GET** that is sent to a URI. In return, the client receives the corresponding resource.

REST uses some common HTTP methods to insert/delete/update/retrieve information which is below:

1. **GET** - Requests a specific representation of a resource
2. **PUT** - Creates or updates a resource with the supplied representation
3. **DELETE** - Deletes the specified resource
4. **POST** - Submits data to be processed by the identified resource

Below are some points which will help developer to understand why to use the RESTful services.

1. Less overhead (no SOAP envelope to wrap every call in)
2. Less duplication (HTTP already represents operations like DELETE, PUT, GET, etc. that have to otherwise be represented in a SOAP envelope).

3. More standardized - HTTP operations are well understood and operate consistently. Some SOAP implementations can get finicky.
4. More human readable and testable (harder to test SOAP with just a browser).
5. Don't need to use XML

Here, the only component that send request to WCF REST service is mobile application component. In fact, the mobile application which is a native app that was installed locally on the mobile device, will communicate with WCF REST service to send and receive data.

One of the main advantages of this approach is that, the main logic of the system (decision engine) will not deployed and distributed on the clients (mobile devices). That will be result in much easier managing in maintenance mode with low cost by avoiding frequently updating applications on clients (mobile devices).

2.2.2 Decision Engine

This module contains main logic, business process of the project. Also it contains modules and procedures to access database with including all the permission and security. Consumers of this component are WCF REST Service, Web Application and Mobile Version Web Application. In fact, Decision Engine will get request from its consumers and then after process them base on inside algorithms and communication with its data access modules, will response to the consumers.

2.2.3 Database

This is the main data storage section of the system. All the user information and results of the decision making process will be stored in this module. Security and performance are the main characteristics of this module. In fact, centralized information by placing it on the back-end part, will be extremely helpful in maintenance mode of the project in changing and updating main data.

3 Methodology, Tools, and Techniques

Base in the latest updated technologies and fastest development tools, an also after requirements analysing process of the project, these following methodology and development CASE tools are chosen for design and development team.
Here, the list of the technologies and tools with brief description are mentioned.

3.1 Methodology

- **Object Oriented Design:** As it is a popular technical approach to analysing, designing an application, system, or business by applying the object-oriented paradigm and visual modelling throughout the development life cycles to foster better stakeholder communication and product quality, we choose it for the methodology of designing and developing software.
- **Unified Modelling Language (UML):** As it is a general-purpose modelling language in the field of software engineering, we use it to document the design process of software whenever we need.
- **Agile:** Agile development provides opportunities to assess the direction throughout the development lifecycle. Base on the resources and size of this project and development team, we choose agile method to speed up the process of developing.

3.2 Development CASE Tools

- **Microsoft Visual Studio .NET Express 2013:** Microsoft Visual Studio provides the most comprehensive solution to easily deliver applications across all platforms, including phone, web, desktop, tablet, server, and the cloud.

- **VS NOMAD:** VS NOMAD is a free extension for Microsoft Visual Studio .NET. VS Nomad provides the best **PhoneGap** Build experience inside the Visual Studio .NET IDE. PhoneGap is a **free** and **open source framework** that allows developers to create mobile apps using standardized web APIs for the different platforms. It allows developers to use standard web technologies such as HTML5, CSS3, and JavaScript for cross-platform development, avoiding each mobile platforms' native development language. Applications execute within wrappers targeted to each platform, and rely on standards-compliant API bindings to access each device's sensors, data, and network status. PhoneGap is associated with cross-platform mobile development that lets developers to build hybrid mobile apps for multiple mobile platforms such as any iOS device, Android and Windows Phone.

3.3 Technologies

- **ASP.NET 4.5:** ASP.NET is a unified Web development model that includes the services necessary for developer to build enterprise-class Web applications with a minimum of coding. ASP.NET is part of the .NET Framework, and when coding ASP.NET applications, developer have access to classes in the .NET Framework

- **ASP.NET AJAX 4.1:** AJAX features in ASP.NET enable developer to quickly create Web pages that include a rich user experience with responsive and familiar user interface (UI) elements. AJAX features include client-script libraries that incorporate cross-browser ECMAScript (JavaScript) and dynamic HTML (DHTML) technologies, and integration with the ASP.NET server-based development platform. By using AJAX features, developer can improve the user experience and the efficiency of our Web applications.

- **ADO.NET 4.5:** ADO.NET provides consistent access to data sources such as SQL Server and XML, and to data sources exposed through OLE DB and ODBC. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, handle, and update the data that they contain.
  In fact, this is our technology for data mapping process between database and other components. All of the communication between database and other components should be processed and handled by use of ADO.NET technology.

- **Enterprise Library 6.0:** Microsoft Enterprise Library is a collection of reusable software components (*application blocks*) addressing common cross-cutting concerns. Each application block is now hosted in its own repository.
  The two mainly features that provides are:

  1- **Logging Process:** All events, especially error messages can be logged and configured in many ways. We can log the events in a file, database or even send it to the testers or operational team.

2- **Exception Handling:** All exception that rise in any layer of the software can be handled by centralized and configurable policies in many ways. In other words, we can define policies for each application or layer to act what to do whenever they faced to the errors (exceptions).

- **Windows Communication Foundation REST 4.5:** A WCF REST service can be categorized as *resource-based*, which means that a REST client sends an HTTP request to programmatically accomplish a business objective. These requests largely employ a **GET** that is sent to a URI. In return, the client receives the corresponding resource.
  We use this technology for the communication between Native Mobile Application component and Decision Engine component.

3.4 Programming Languages

- **C#.NET:** C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. As an object-oriented language, C# supports the concepts of encapsulation, inheritance, and polymorphism. All variables and methods, including the Main method, the application's entry point, are encapsulated within class definitions.

- **HTML:** This is our client side programming language for creating web pages. We also use that for creating Native Mobile Application in VS Nomad extension.

- **XML:** We will use XML for configuration files and also for data communication between client-server via Web Application component and servers for performance issues.

- **CSS:** Cascade Style Sheet. We use it because we want to centralize our style, colours, size of our application user interfaces. We will use it for both Web Application and Mobile Application components.

- **JavaScript:** JavaScript is a cross-platform, object-oriented scripting language. JavaScript is a small, lightweight language; it is not useful as a standalone language, but is designed for easy embedding in other products and applications, such as web browsers. This is our client side language. We will use it inside web pages and also Native Mobile Application.

- **JQuery:** jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

- **T-SQL:** Transact-SQL is central to the use of Microsoft® SQL Server™. All applications that communicate with SQL Server do so by sending Transact-SQL statements to the server, regardless of an application's user interface.

3.5 Database Engine

- **Microsoft SQL Server Express 2014:** It is a relational database management system developed by Microsoft. As a database, it is a software product whose primary function is to store and retrieve data as requested by other software applications, be it those on the same computer or those running on another computer across a network (including the Internet).
  We choose it because of highly ability of integration with .NET Framework technology and also strong security and performance of SQL Server managing data.

3.6 Technologies and Applications Mapping

By considering the mentioned technology and applications, we should know that which technologies should be used for developing each applications. In the following table, we can see the mapping between technologies/languages and applications/components of the project:

	Web App (Desktop Version)	Web App (Mobile Version)	Mobile App (Native App)	WCF REST	Decision Engine	Database
Visual Studio .NET	*	*		*	*	
VS Nomad			*			
ASP.NET	*	*				
AJAX	*	*				
ADO.NET					*	
Enterprise Library	*	*		*	*	
WCF REST			*	*		
C#.NET	*	*		*	*	
HTML	*	*	*			
XML	*	*				
CSS	*	*	*			
JavaScript	*	*	*			
JQuery	*	*	*			
T-SQL						*
SQL Server						*

Table 1 – Technologies and Applications Mapping

4 System Architecture

This chapter outlines the complete system architecture of the project and how it fits in an overall architecture where the component is deployed. We will need to understand this in order to properly partition the design into physical or logical artifacts. First we will introduce an overall picture of the architecture and then go into details before ultimately proceeding to the actual design of the applications and the component logic.

4.1 Software Architecture Decision

For many reason, we choose *Layered architecture* for the software design of the project. The key to *Layered Application* is dependency management. Components in one layer can interact only with peers in the same level or components from lower levels. This helps reduce the dependencies between components on different levels as shown in Figure 2.

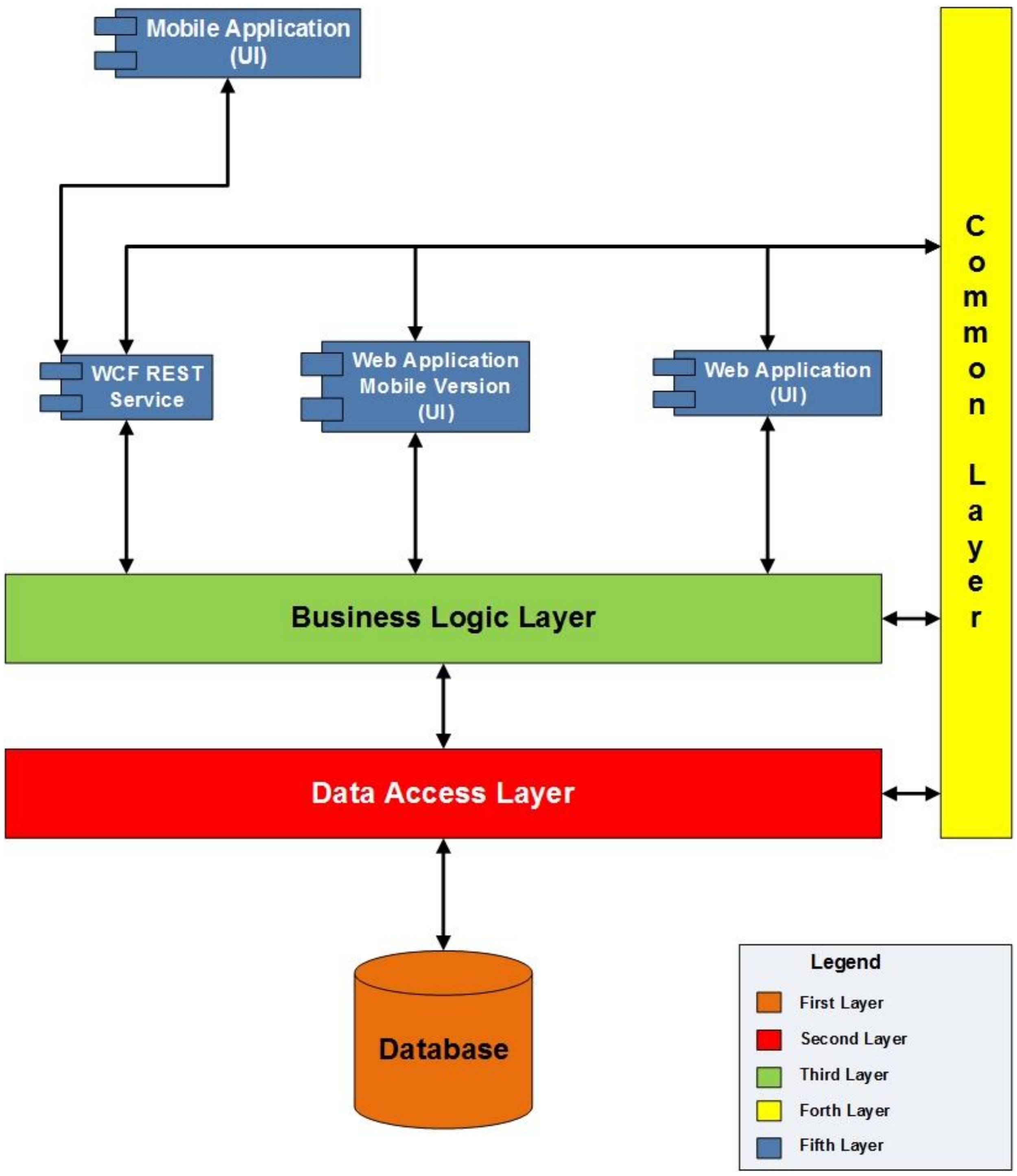


Figure 2- Layered Software Architecture

4.2 Software Architecture details

In this section we will concentrate on important architectural issues such as the contents of each layers and also the placement of each component. Here, each layer has its own specific functionality and responsibility. In the following, we list all the layers with details and possible special process that should be handled in it.

4.2.1 First Layer: Database

The lowest level of the software is the database layer. This layer is responsible for managing data in a way that security and performance are provided. The only layer that can access to this layer is data access layer. In fact database layer is only response to the request of the data access layer.

Also, the administrator of the database (DBA) has access to it at administration level. Applications is only access it via data access layer that are only allowed to change data (insert, delete and update) and not allowed to change structure of the data model.

The data model of this layer will be defined by the designer base on the requirements analysing.

Technically, this layer will be implemented by use of MS SQL Server technology as described before.

4.2.2 Second Layer: Data Access Layer

This layer is responsible to send/receive data from/to database. Also, changing data (insert, delete, and update) will be done by the commands of this layer. This layer only knows database, managing connections of database and etc. It doesn’t involve with the business model of the project. In other words, data access layer (DAL) will execute all of query and procedure of the database to manipulate data.

As all of the data type and other common and useful functions are placed on common layer, data access layer also communicate with common layer to use them.

The consumer of this layer is only business logic layer (BLL). BLL send request to DAL to get or set data. Actually, this layer is a part of decision engine module.

Technically, this layer will be implemented by use of C# language in Visual Studio .NET environment. The output of the implementation will be an executable DLL file.

4.2.3 Third Layer: Business Logic Layer

This layer is also known as BLL. It contains all of the logic and implementation of business model of the project. In fact, decision engine module and all possible algorithms will be embedded inside of it. Everything that received from user interface layers should be first handled by BLL layer.

The consumer of the BLL are user interfaces of the applications that pass the data to it as a request and receive the response from it. BLL also uses the useful functionality of common layer such as logging and exception handling processes.

For managing data, BLL will use functions of DAL to work with it. Actually, this layer is a part of decision engine module.

Technically, this layer will be implemented by use of C# language in Visual Studio .NET environment. The output of the implementation will be an executable DLL file.

4.2.4 Forth Layer: Common Layer

Actually, all of the shared codes, functions, variable, properties and Data Types Objects (DTO) will be placed in common layer. Moreover, some of the shared features such as exception handling, logging process, optimization process, building URL, security issues that will be used by all layers, will be implemented in this layer.
Also, one the most important concept of software design is configuration and settings that all of them will be placed in common layer.
As we can see in Figure 2, all layers will communicate with the common layer. This approach would be helpful when changing request will be appeared for any reason. In this case, changing are limited only to one point, common layer, and then will be applied to all other layers and applications.

Technically, this layer will be implemented by use of C# language in Visual Studio .NET environment. The output of the implementation will be an executable DLL file.

4.2.5 Fifth Layer: User Interfaces of Applications

This layer contains the user oriented functionality responsible for managing user interaction with the system, and generally consists of components that provide a common bridge into the core business logic encapsulated in the business layer. The user interface layer contains the components that implement and display the user interface and manage user interaction. This layer includes controls for user input and display, in addition to components that organize user interaction.

In fact, user interface layer is responsible to get data from user, process data by checking all possible error in formatting of data base on policies and requirements, pass only correct data to the BLL and get the result and then pass present to the user.

The user interface layout of all applications should be based on standard documents and defined policies to avoid any inconvenience in user point of view.

Here, we have four user interface application that are listed with details in the followings.

4.2.5.1 Web Application UI

This layer is belong to the user interface of Web Application module. As this application is designed for the desktop browser, the design of the layout should be based on the desktop user interface requirements. It should be worked and viewed in the same way for all kind of browsers (IE, Safari, Firefox, etc.) of different machine (PC, MAC).

4.2.5.2 Web Application Mobile Version UI

This layer is belong to the user interface of Web Application Mobile Version module. As this application is designed for the mobile devices browser, the design of the layout should be based on the browser of mobile devices user interface requirements. It should be worked and viewed in the same way for all kind of browsers (IE, Safari, Chrome, etc.) of different machine (iOS, Android, Windows Phone).

This is also known as rich Internet application (RIA). All of the page layout should be viewed without need of scrolling. All of the other standard layout for RIA should be applied.

4.2.5.3 WCF REST Service

This is the service that is act between BLL layer and Mobile Application on mobile devices. This service will deployed on the main server. WCF REST service provides specific interface that contains methods which Mobile Application can call.

4.2.5.4 Mobile Application (Native App)

This application was architected to use a thin mobile client deployed on iPad, iPhone & Android and Windows Phone devices that communicated with a server. This is actually Hybrid Mobile Application that is developed by use of HTML, CSS, JavaScript and libraries of **Apache Cordova** API.

As mobile devices require a simpler architecture, simpler UI, and other specific design decisions in order to work within the constraints imposed by the device hardware. Keep these constraints in mind and design specifically for the device instead of trying to reuse the architecture or UI from a desktop or Web application. The main constraints are memory, battery life, ability to adapt to difference screen sizes and orientations, security, and network bandwidth.

The key feature of design is that to keep logic on the server side and Mobile Application act as a thin client only to send and receive data.

5 Deployment Plan

5.1 Introduction

Applications must be deployed into a physical environment where infrastructure limitations may negate some of the architectural decisions. Therefore, we must consider the proposed deployment scenario and the infrastructure as part of our application design process. Here, we describe the options available for deployment of base on our chosen deployment strategy; base on scale the application; and guidance and patterns for performance, reliability, and security issues.

By considering the possible deployment scenarios for our application as part of the design process, we should prevent a situation where the application cannot be deployed successfully, or fails to perform to its design requirements because of technical infrastructure limitations.

For choosing the deployment strategy, we considered different aspect and specification of our project, and here we can see the list of these factors:

- The target physical environment for deployment.
- The architectural and design constraints based on the deployment environment.
- The security and performance impacts of our deployment environment.
- The number of transactions per second.
- The average size of the message.
- The messaging model and the number of subscriptions per message.

Important NOTE: In deployment plan, we assume that all the communication between clients and servers will be operated via Internet network.

5.2 Deployment Strategy

For creating deployment strategy, we should determine to use a distributed or a non-distributed deployment model. For building a simple intranet application for specific organization, which will be accessed by finite set of users, we should consider non-distributed deployment. On the other hand, for building a more complex application, which we must optimize for scalability and maintainability, we should think about a distributed deployment.

By considering all pf these factors, we chosen non-distributed strategy for deployment plan as shown in Figure 3.

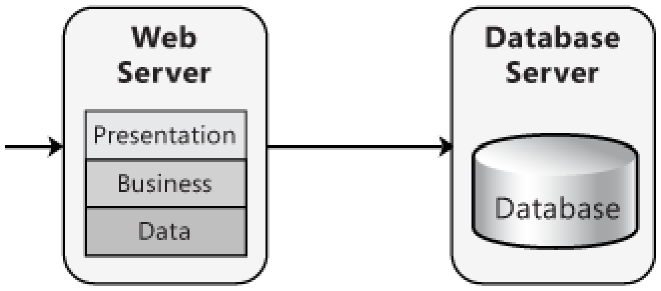


Figure *3* – Non-distributed Deployment Strategy

A non-distributed deployment, all of the functionality and layers reside on a single server except for data storage functionality.

This approach has the advantage of simplicity and minimizes the number of physical servers required. It also minimizes the performance impact inherent when communication between layers must cross physical boundaries between servers or server clusters.

The n-tier pattern represents a general pattern where components of the application are separated across one or more servers.

Here, we introduce **two different plan** that named 2-tier and 3-tier pattern. We can choose one of these approaches for deployment plan.

5.3 2-Tier Deployment Plan

In a 2-tier design, the client interacts with application software deployed on a server that contains all the web applications, services and also database, as shown in Figure 3. We might implement a firewall between the client tire and Web/App/Data tier. In this scenario, we have only one server that includes both web application, WCF REST service and database. This approach may be useful for finite user request and transactions.

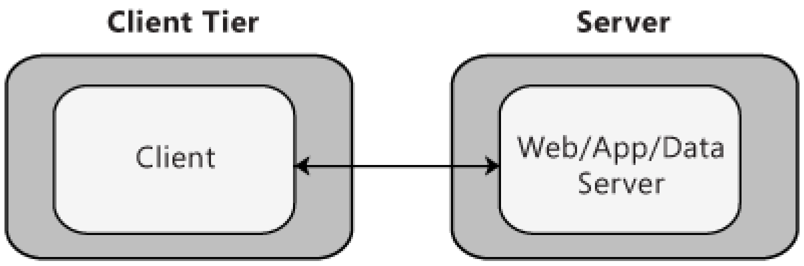


Figure *4* – 2-Tier Deployment

The important issue in this approach is that, we have only one server that contains all of web, application and database components. In fact, this server will be our web server and database server at the same time.

We should keep in mind that by using a single server, even though we minimize communication performance overhead, we can hamper performance in other ways. Because all of our layers share resources, one layer can negatively affect all of the other layers when it is under heavy utilization. In addition, the servers must be generically configured and designed around the strictest of operational requirements, and must support the peak usage of the largest consumers of system resources. The use of 2-tier reduces our overall scalability and maintainability because all the layers share the same physical hardware.

5.3.1 Components in 2-Tier Deployment Plan

In this section, we map all of the applications and components which described before, to the client-server 2-Tire deployment strategy as shown in Figure 5.

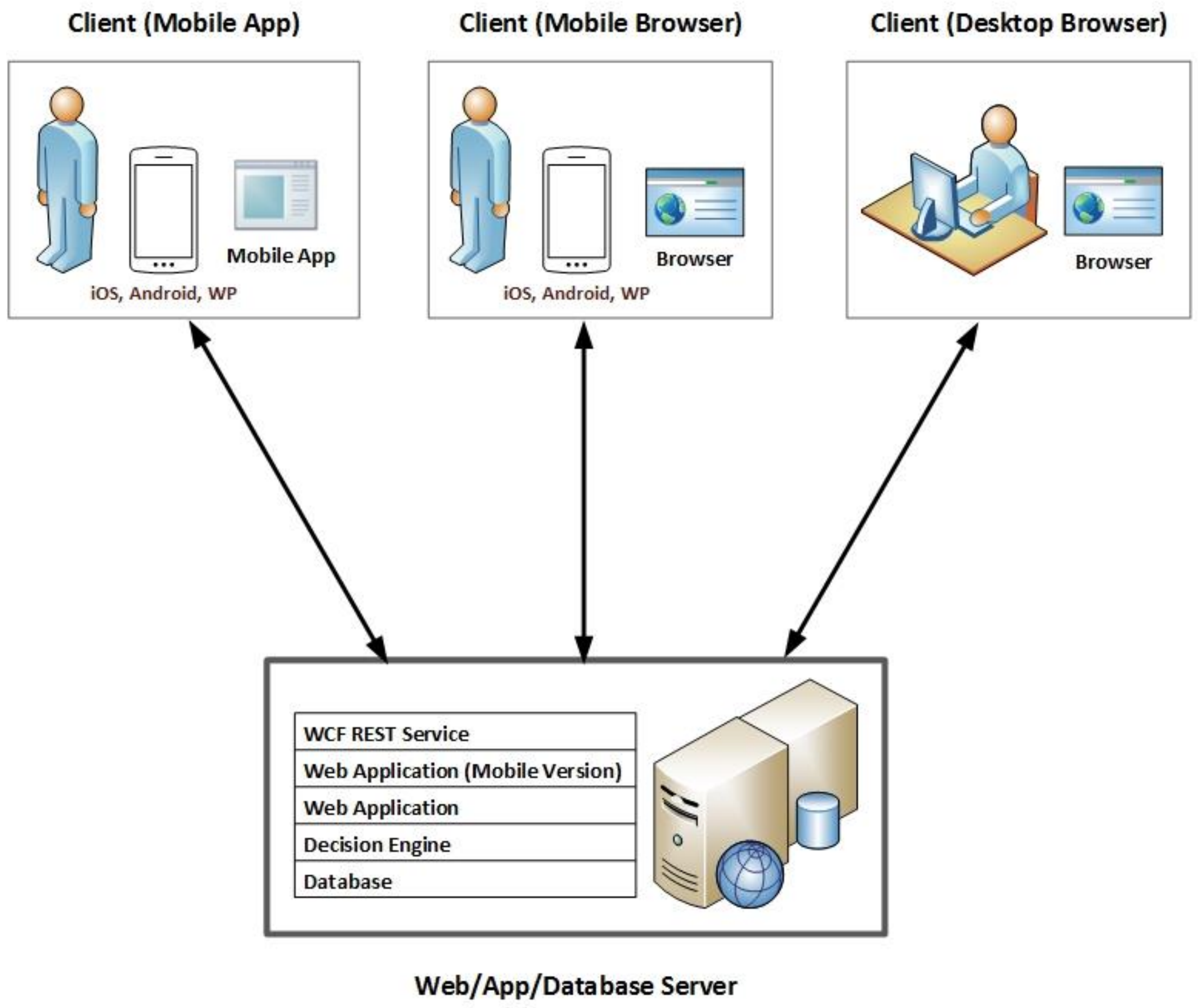


Figure *5* – 2-Tier Deployment with Applications and Components

As we can see in Figure 5, all of the components are located in a single server except Mobile Application component. As described before this native mobile application will be installed on the user mobile device and it will communicate with server via WCF REST Service component. The other two type of clients, client mobile browser and client desktop browser will communicate with Web Application Mobile Version and Web Application components respectively.

5.3.2 Software Requirements for 2-Tier Deployment Plan

Here it is the list of the software requirement for a single server:

- Microsoft Windows Server 2008/2012
- Microsoft SQL Server Express 2014

Here it is the list of the accounts that will be needed for submitting mobile app on various market place:

- Apple Developer Account: To deploy and submit on App Store.
- Google Developer Account: To deploy and submit on Google Play
- Microsoft Developer Account: To deploy and submit on Windows Phone Store

5.3.3 Hardware Requirements for 2-Tier Deployment Plan

The hardware specification for installing Microsoft Windows Server 2008 and Microsoft SQL Server Express 2014 and all others components are listed for a single server:

Component	Requirement
Processor	• Minimum: 1 GHz (x86 processor) or 1.4 GHz (x64 processor) • Recommended: 2 GHz or faster **Note:** An Intel Itanium 2 processor is required for Windows Server 2008 for Itanium-Based Systems.
Memory	• Minimum: 1 GB RAM • Recommended: 2 GB RAM or greater • Maximum (32-bit systems): 4 GB (Standard) or 64 GB (Enterprise and Datacenter) • Maximum (64-bit systems): 32 GB (Standard) or 1 TB (Enterprise and Datacenter) or 2 TB (Itanium-Based Systems)
Available Disk Space	• Minimum: 50 GB • Recommended: 100 GB or greater
Drive	DVD-ROM drive
Display and Peripherals	• Super VGA (800 x 600) or higher-resolution monitor • Keyboard • Microsoft Mouse or compatible pointing device

Table 2 – Hardware Requirements for 2-Tier Deployment Plan

5.3.4 Network Requirements for 2-Tier Deployment Plan

The network specification for accessing website and service are listed below (*Note: Network construction details are not listed here)*:

- Internet speed connection up to 100 Mbps
- Network Interface Card
- Valid Static IP Address
- Valid DNS name
- SSL Certification

5.4 3-Tier Deployment Plan

In a 3-tier design, the client interacts with application software deployed on a separate server, and the application server interacts with a database that is located on another server, as shown in Figure 4. This is a very common pattern for most Web applications and Web services, and sufficient for most general scenarios. We might implement a firewall between the client and the Web/App tier, and another firewall between the Web/App tier and the database tier.

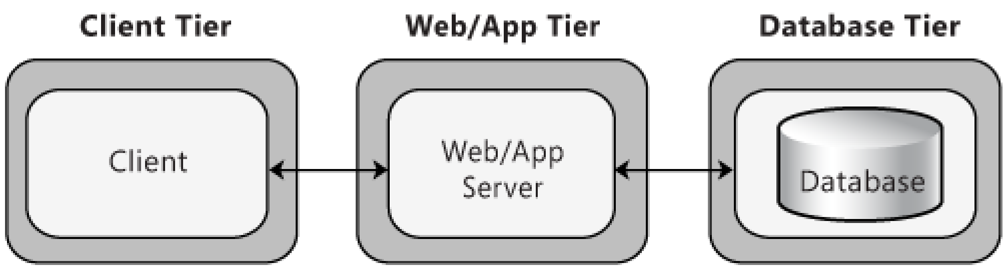


Figure *6* – 3-Tier Deployment Plan

3-tire tiers enable 3-tire environments. We can optimize each environment for a specific set of operational requirements and system resource usage. We can then deploy components onto the tier that most closely matches their resource needs to maximize operational performance and behaviour. The more tiers we use, the more deployment options we have for each component. Distributed deployment provides a more flexible environment where we can more easily scale out or scale up each physical tier as performance limitations arise, and when processing demands increase. However, we should keep in mind that adding more tiers adds complexity, deployment effort, and cost.

Another reason for adding tiers is to apply specific security policies. Distributed deployment allows us to apply more stringent security to the application servers; for example, by adding a firewall between the Web server and the application servers, and by using different authentication and authorization options.

5.4.1 Components in 3-Tier Deployment Plan

In this section, we map all of the applications and components which described before, to the client-server 2-Tire deployment strategy as shown in Figure 7.

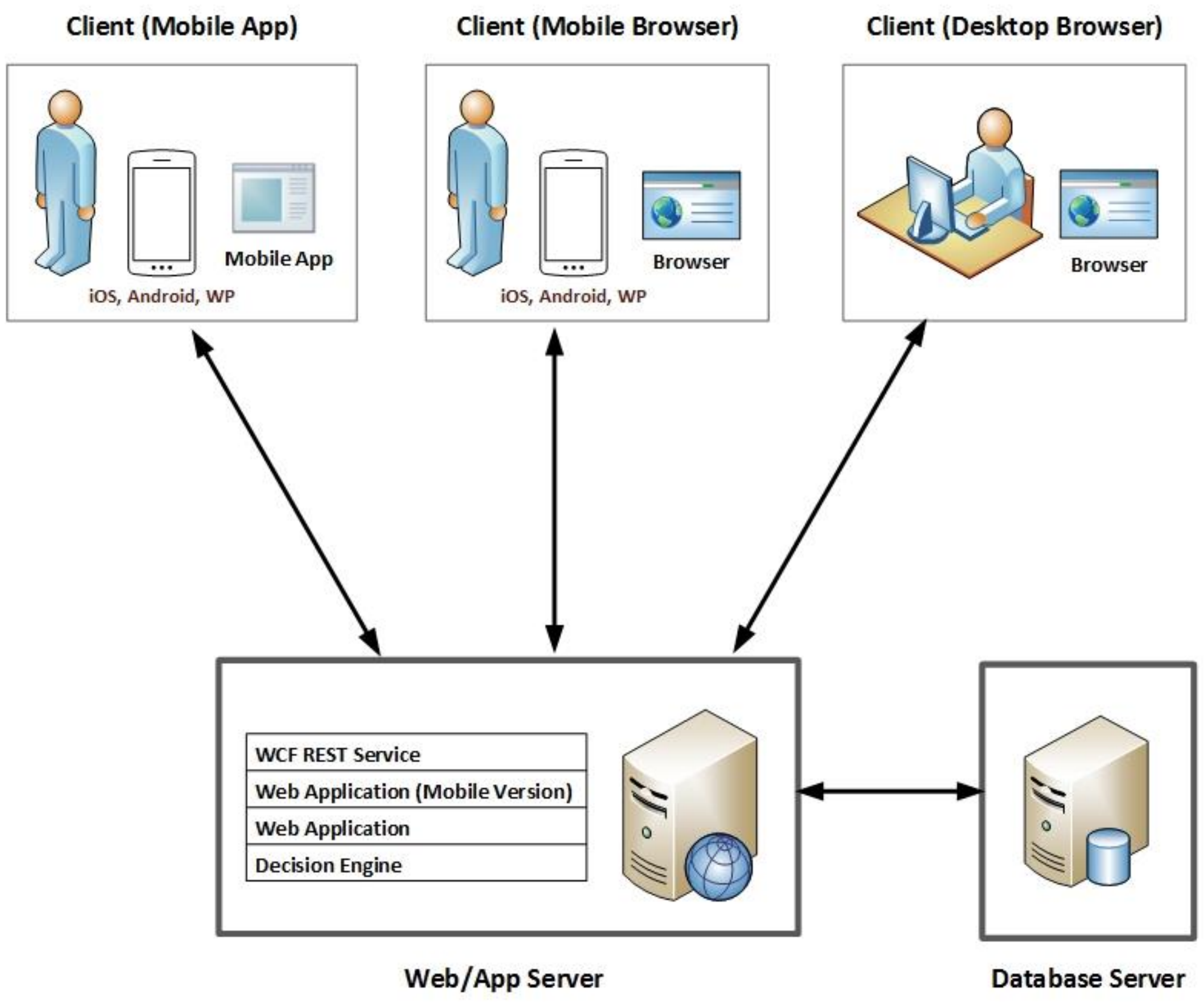


Figure 7 – 3-Tier Deployment with Applications and Components

Everything here is like exactly what we have in 2-Tire Deployment Plan with only one different. In this approach, we have two separate server. One of them is web and application server that contains all of components except database. Another one is database server that contains only database component. In fact, the web/app server will communicate database server whenever wants to get or set data.

5.4.2 Software Requirements for 3-Tier Deployment Plan

Here it is the list of the software requirement for a single server:

- Microsoft Windows Server 2008/2012
- Microsoft SQL Server Express 2014

Here it is the list of the accounts that will be needed for submitting mobile app on various market place:

- Apple Developer Account: To deploy and submit on App Store.
- Google Developer Account: To deploy and submit on Google Play
- Microsoft Developer Account: To deploy and submit on Windows Phone Store

5.4.3 Hardware Requirements for 3-Tier Deployment Plan

The hardware specification for installing Microsoft Windows Server 2008 and Microsoft SQL Server Express 2014 and all others components are listed here for each server:

Component	**Requirement**
Processor	• Minimum: 1 GHz (x86 processor) or 1.4 GHz (x64 processor) • Recommended: 2 GHz or faster **Note:** An Intel Itanium 2 processor is required for Windows Server 2008 for Itanium-Based Systems.
Memory	• Minimum: 1 GB RAM • Recommended: 2 GB RAM or greater • Maximum (32-bit systems): 4 GB (Standard) or 64 GB (Enterprise and Datacenter) • Maximum (64-bit systems): 32 GB (Standard) or 1 TB (Enterprise and Datacenter) or 2 TB (Itanium-Based Systems)
Available Disk Space	• Minimum: 50 GB • Recommended: 100 GB or greater
Drive	DVD-ROM drive
Display and Peripherals	• Super VGA (800 x 600) or higher-resolution monitor • Keyboard • Microsoft Mouse or compatible pointing device

Table 3 – Hardware Requirements for 3-Tier Deployment Plan

5.4.4 Network Requirements for 2-Tier Deployment Plan

The network specification for accessing website and service are listed below (*Note: Network construction details are not listed here)*:

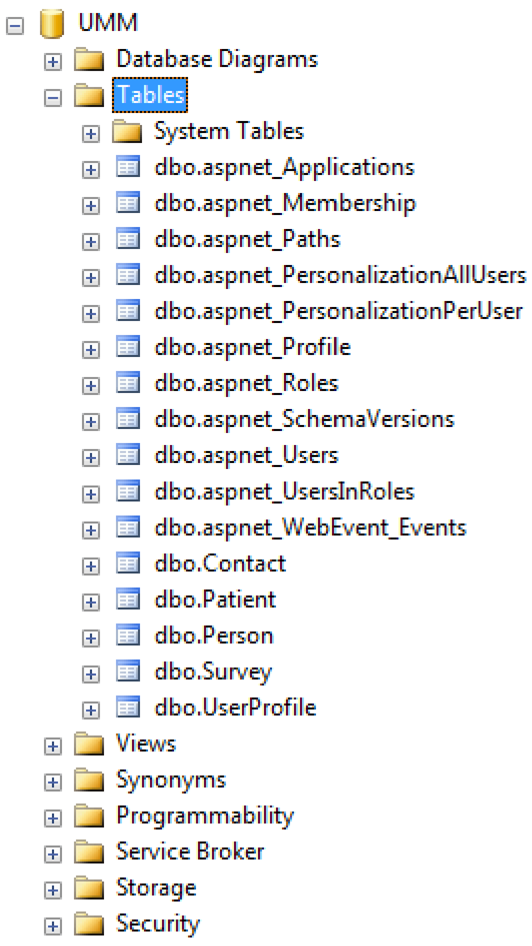
- Internet speed connection up to 100 Mbps
- Network Interface Card
- Valid Static IP Address
- Valid DNS name
- Ethernet, Fast Ethernet, or Gigabyte Ethernet cabling
- A hub or other device that performs the function of relaying network traffic between computers and devices.
- SSL Certification

6 Components in development

Description of components will be explained in five different levels base on main architecture of the software.

6.1 Database component

In the figure below, the list of tables of the database are listed.



```
UMM
  Database Diagrams
  Tables
    System Tables
    dbo.aspnet_Applications
    dbo.aspnet_Membership
    dbo.aspnet_Paths
    dbo.aspnet_PersonalizationAllUsers
    dbo.aspnet_PersonalizationPerUser
    dbo.aspnet_Profile
    dbo.aspnet_Roles
    dbo.aspnet_SchemaVersions
    dbo.aspnet_Users
    dbo.aspnet_UsersInRoles
    dbo.aspnet_WebEvent_Events
    dbo.Contact
    dbo.Patient
    dbo.Person
    dbo.Survey
    dbo.UserProfile
  Views
  Synonyms
  Programmability
  Service Broker
  Storage
  Security
```

Figure *8* – List of tables

6.1.1 Tables

- **Aspnet_*:** These tables are installed based on ASP.NET Membership. All of these tables are providing membership functions. To learn more about these tables you can check this URL: http://msdn.microsoft.com/en-us/library/yh26yfzy%28v=vs.140%29.aspx

- **Contact:** This table contains specification of the user contact information such as address, mobile and telephone number.
- **Patient:** This table contains specification of the patient in the system. Some of the attributes are ContactId, PersonId, etc.
- **Survey:** This table contains specification of the survey information in the system. This table is one of the core tables which contains near hundred attributes. This tables is designed base on exactly data item which requested to be stored.
- **Person:** This table contains specification of the person contact information such as First Name, Last Name, Birth Date and etc.
- **UserProfile:** This table contains specification of the user profile information such as PersonId, UserId, ContactId and etc. In fact this table is connected to a table of the ASP.NET Membership module (aspnet_Users).

6.1.2 Stored Procedures

The list of the stored procedures (database scripts) that are already using in the system have been listed below:

Important NOTES:

- **SELECT command:** All of the name of the scripts which are using for Select command are begin with 'sp'. E.g. 'spGetContact'.
- **INSERT Command:** All of the name of the scripts which are using for Insert command are begin with 'ip'. E.g. 'ipGetContact'.
- **DELETE Command:** All of the name of the scripts which are using for Delete command are begin with 'dp'. E.g. 'dpGetContact'.
- **UPDATE Command:** All of the name of the scripts which are using for Update command are begin with 'up'. E.g. 'upGetContact'.

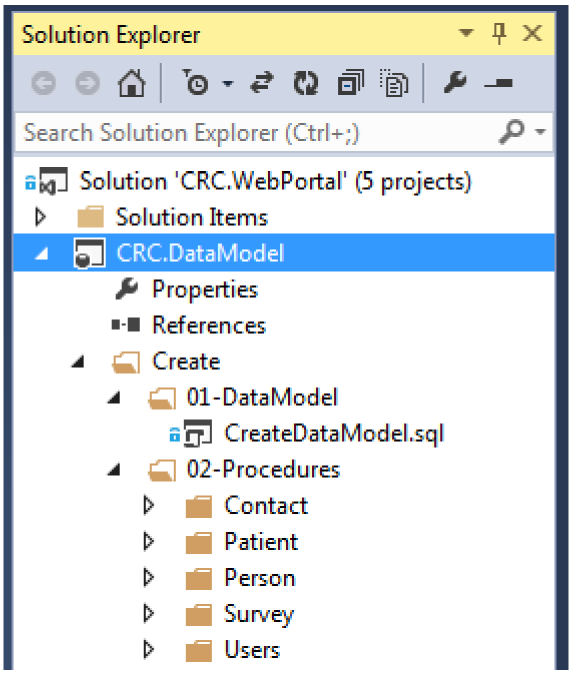
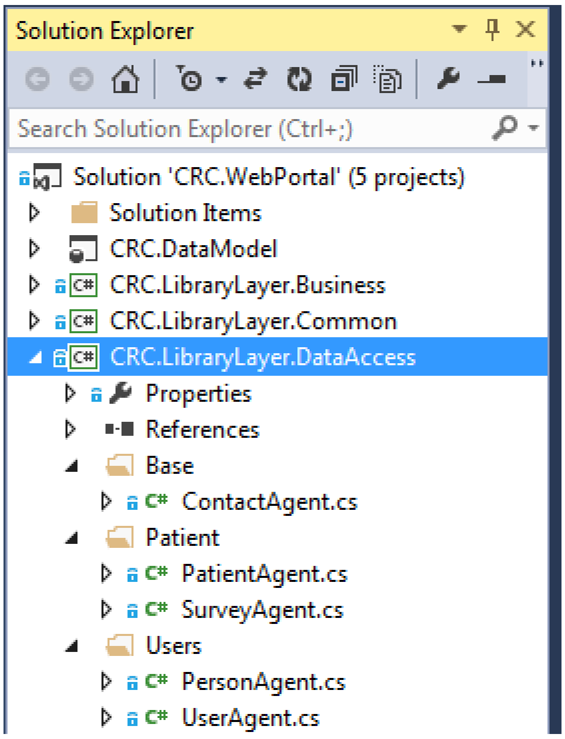


Figure 9 – List of Stored Procedures

- **CreateDataModel.sql:** This script is using for creating CRC Data Model from the scratch. In fact, by running this script target machine (client/server) the database schema with some required initialization data will be installed.
- **Contact:** All of the scripts which are under this folder are using for manipulating data (Select, Insert, Delete, Update) in Contact table.
- **Patient:** All of the scripts which are under this folder are using for manipulating data (Select, Insert, Delete, Update) in Patient table.
- **Person:** All of the scripts which are under this folder are using for manipulating data (Select, Insert, Delete, Update) in Person table.
- **Survey:** All of the scripts which are under this folder are using for manipulating data (Select, Insert, Delete, Update) in Survey table.
- **Users:** All of the scripts which are under this folder are using for manipulating data (Select, Insert, Delete, Update) in User table.

6.2 Data Access component

In the figure below, the list of classes of the project which are using for communicating with database to fetch, insert, delete and update data are listed. All of these classes are corresponding to the tables of the database which are described before.
In fact, these classes work as data mapper in the project. They get/set data from/to database via defined objects (DTO – Data type Objects).



```
Solution 'CRC.WebPortal' (5 projects)
  Solution Items
  CRC.DataModel
  CRC.LibraryLayer.Business
  CRC.LibraryLayer.Common
  CRC.LibraryLayer.DataAccess
    Properties
    References
    Base
      ContactAgent.cs
    Patient
      PatientAgent.cs
      SurveyAgent.cs
    Users
      PersonAgent.cs
      UserAgent.cs
```

Figure *10* – List of classes for Data Access components

- **ContactAgent:** This class is using for calling the stored procedure that related to the Contact table. This class is responsible for manipulating data of the Contact table in Data Access level of the project.
- **PatientAgent:** This class is using for calling the stored procedure that related to the Patient table. This class is responsible for manipulating data of the Patient table in Data Access level of the project.
- **PersonAgent:** This class is using for calling the stored procedure that related to the Person table. This class is responsible for manipulating data of the Person table in Data Access level of the project.
- **SurveyAgent:** This class is using for calling the stored procedure that related to the Survey table. This class is responsible for manipulating data of the Survey table in Data Access level of the project.
- **UsersAgent:** This class is using for calling the stored procedure that related to the User table. This class is responsible for manipulating data of the User table in Data Access level of the project.

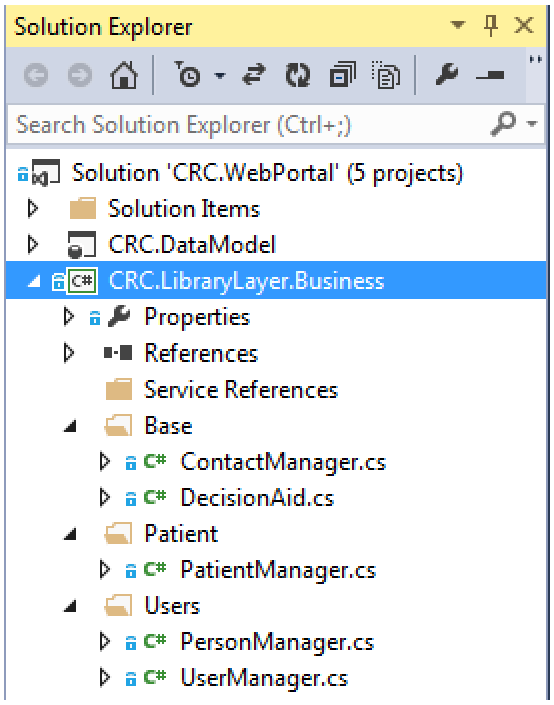
6.3 Business component

In the figure below, the list of classes of the project which are using for communicating with Data Access component and User Interface component are listed. All of these classes are responsible to get required data from the User Interface, do required logic and then pass it to Data Access component and vice versa.

In fact, the main logics and algorithms are implemented in this component.

Important NOTES:

- **Opening/Closing Database connection:** This layer is responsible for opening and closing database connection.
- **Handling Transactions:** All of the required transactions are handing in this component.



Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'CRC.WebPortal' (5 projects)
  Solution Items
  CRC.DataModel
  CRC.LibraryLayer.Business
    Properties
    References
    Service References
    Base
      ContactManager.cs
      DecisionAid.cs
    Patient
      PatientManager.cs
    Users
      PersonManager.cs
      UserManager.cs
```

Figure *11* – List of classes for Business components

- **ContactManager:** This class is using for getting data which related to contact information, apply required and defined related business and logic of contact working flow and then pass the output data to the Data Access component. On the other hand, this class also getting data from the Data Access layer, apply required and defined related business and logic of contact process and then pass the output data to the User Interface component
- **DecisionAid:** This class contains main algorithms of the decision making engine. This class covers all of the required algorithms which are defined in 'Non-Additive Measures: A Theoretical approach to Medical Decision Making – by Francois Modave and Navkiran K Shokar' and 'Identification of fuzzy measures from sample data with genetic algorithms – by Pedro Miranda' article.
    - `List<string> GetAllCombination(List<int> list)`
      Input: list of numbers. E.g. 1, 2, 3
      Output: All possible combinations. E.g. {1,2,3}{1,2}{2,3}….
    - `List<string> GetCominationForFuzzyMeasure()`
      Output: All possible combinations for numbers 1 to 15 except 17 items which including 1 to 15 itself (1 item), each numbers from 1 to 15 (15 items), and null (1 item).

- `Dictionary<string,double> GetCriterionValuesBaseOnMatrix(Dictionary<string, AlternativeInfo> alternativeValues)`
  Input: The values of the matrix which will be given from User Interface. In fact, the patient will fill out the matrix values of Criterion (15 items) and Alternatives (3 items).
  Output: Calculate the following formula. The list of the u(i) for 15 items.

$$\mu(i) = \frac{\sum_j x_i^j}{\sum_{i,j} x_i^j}$$

- `public Dictionary<string, double> GenerateFuzzyMeasure(Dictionary<string, double> uI)`
  Input: The output of the above function: ‘`GetCriterionValuesBaseOnMatrix`’.
  Output: Calculate base on the Algorithm which described in page 3060 of ‘Identification of fuzzy measures from sample data with genetic algorithms – by Pedro Miranda’ article. The list of the u(i) for 2^15-17 = 32,751 items.

- **PatientManager:** This class is using for getting data which related to patient information, apply required and defined related business and logic of patient working flow and then pass the output data to the Data Access component. On the other hand, this class also getting data from the Data Access layer, apply required and defined related business and logic of contact working flow and then pass the output data to the User Interface component

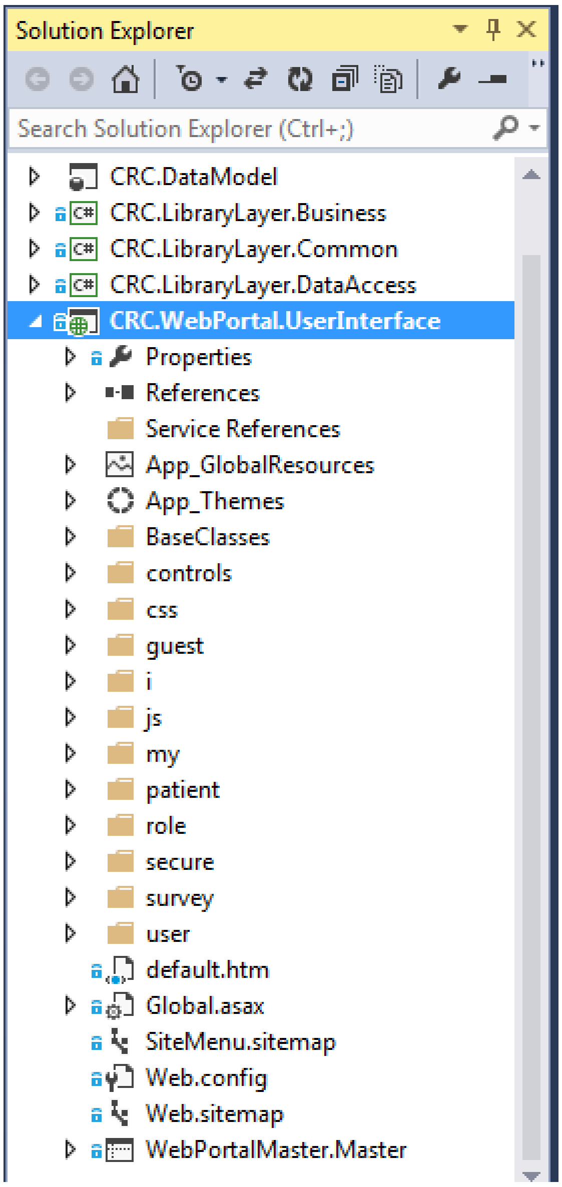
- **PersonManager:** This class is using for getting data which related to Person information, apply required and defined related business and logic of person working flow and then pass the output data to the Data Access component. On the other hand, this class also getting data from the Data Access layer, apply required and defined related business and logic of person working flow and then pass the output data to the User Interface component.

- **UserManager:** This class is using for getting data which related to user information, apply required and defined related business and logic of user working flow and then pass the output data to the Data Access component. On the other hand, this class also getting data from the Data Access layer, apply required and defined related business and logic of user working flow and then pass the output data to the User Interface component.

6.4 User Interface component

In the figure below, the list of classes and pages of the project which are using for communicating with Business component and End User are listed. All of these classes and pages are responsible to get required data from the End User, do required UI logic, check constrains and validity, and then pass the valid data to the Business component and vice versa.

In fact, all of the pages for interacting with end user are implemented in this component.



```
CRC.DataModel
CRC.LibraryLayer.Business
CRC.LibraryLayer.Common
CRC.LibraryLayer.DataAccess
CRC.WebPortal.UserInterface
    Properties
    References
    Service References
    App_GlobalResources
    App_Themes
    BaseClasses
    controls
    css
    guest
    i
    js
    my
    patient
    role
    secure
    survey
    user
    default.htm
    Global.asax
    SiteMenu.sitemap
    Web.config
    Web.sitemap
    WebPortalMaster.Master
```

Figure *12* – List of classes and pages for User Interface components

- **App_GlobalResources**: All of the resources for managing the text languages are placed here. E.g.:
    - **NewBaselineSurvey.en.resx:** All of the texts of the 'NewBaselineSurvey' page for English language are defined here.
    - **NewBaselineSurvey.es.resx:** All of the texts of the 'NewBaselineSurvey' page for Spanish language are defined here.

- **App_Themes**: All of Cascade Style Sheet which are related only to the COLOR of the web pages are defined here. E.g. Blue, Green. By default the Blue Theme is used. It is defined in web.config file.

- **BaseClasses:** All of the methods and functions which are shared between all of the pages are placed here.
- **controls:** All of the User Controls of the web pages are placed here. Some of the User Controls are Person, Contact, etc.
- **CSS:** All of the Cascade Style Sheet of the project are placed here.
    - **style-desktop.css:** The CSS file for desktop version web pages.
    - **style-iPad.css:** The CSS file for iPad version web pages.
- **guest**: All of the pages which can be visited by even anonymous users are placed here. E.g. Login page.
- **i:** All of the Images of the project are placed here. The 'I' character is used for increasing performance instead of whole word of 'image'.
- **js**: All of the java script which are used in all pages are placed here.
- **my**: All of the pages which can be visited by logged-in users (Authorized users) are placed here. E.g. home, profile.
- **Patient**: All of the pages which are used to show the patient information are placed here.
- **role**: All of the pages which are used to show the role information are placed here. E.g. RoleNew, RoleEdit, etc.
- **secure**: All of the pages which are should be opened in secure channel (with HTTPS protocol) are placed here. E.g. Login, UserNew.
- **survey**: All of the pages which are related to the survey work flow are placed here. E.g. NewBaselineSurvey.aspx
- **user**: All of the pages which are related to the user work flow are placed here. E.g.:
    - **UserEdit.aspx:** For editing user information.
    - **UserView.aspx:** For viewing user information.
- **SiteMenu.sitemap**: This is the configuration file for defining the horizontal menu items in desktop version.
- **Web.sitemap**: This is the configuration file for defining the left side menu items in desktop version and main menu for the iPad version.
- **Web.Config**: This is the configuration file for whole of the web site. This is editable in run time.