



Online Art Gallery

B.Sc Information Technology (Lovely Professional University)

Online Art Gallery

CONTENTS

1. INTRODUCTION

2. DESIGN PRINCIPLES & EXPLANATION

2.1. MODULES

3.2. MODULE DESCRIPTION

4. PROJECT DICTIONARY

4.1. DATAFLOW DIAGRAMS

4.2. E-R DIAGRAMS

4.3. DATA DICTIONARY

5. FORMS & REPORTS

5.1. I/O SAMPLES

6. BIBLIOGRAPHY

ABSTRACT

Category:

Microsoft ASP.NET Based Web Application.

Objective:

Online **Art Gallery** is an online application, which is used to display and sell art works of artist irrespective of their nationality, gender and other narrow consideration, through auction.

Purpose of the system:

ONLINE ART GALLERY is a application software and it is very helpful for the art lovers and others who wants to know the addresses where this kind of arts will we sold.

This application helps the end-users to search their arts and paintings and they can place order for the selected pieces. The end-user can also get the information about the art exhibition and the respective address, so, that they can visit to those exhibitions.

Art Gallery brings you an opportunity to view online art exhibitions at our Online Art Gallery we bring you details of all art exhibitions held in the past and the forthcoming show. The Online Art Gallery is updated daily, so the user can view and buy the latest collection of contemporary art online from any where in the world. You can view and buy the latest Indian contemporary art collection available at their exhibitions and also at their online gallery.

MODULES:

- User Registration
- Art Exhibition
- Artist Directory
- Contact Us

HARDWARE AND SOFTWARE SPECIFICATIONS

SOFTWARE REQUIREMENTS:

Operating System	: Windows XP, 2000.
Language	: C#.NET
Technologies	: Microsoft.NET, ASP.NET, ADO.NET
Data Bases	: Microsoft Sql Server 2005
IDE	: Visual Studio 2008

HARDWARE REQUIREMENTS:

Processor	: Any Processor above 500 MHz.
Ram	: 128Mb.
Hard Disk	: 10 Gb.
Compact Disk	: 650 Mb.
Input device	: Standard Keyboard and Mouse.
Output device	: High Resolution Monitor.

1. INTRODUCTION

Online Art Gallery is an online application, which is used to display and sell art works of artist irrespective of their nationality, gender and other narrow consideration, through auction. Artist can register online for being a member in the art gallery and each artist can upload the digital copy of their art work under the respective categories. They can host their art work either for auction or for fixed price. The artist is liable to pay a fraction of the price of each art work to the web site to find the running fund for site. Art lovers have to go to the art exhibition to collect their favorite arts or painting. But now-a-days they are not getting enough time to go to the galleries and collect the arts and paintings.

Existing System:

Customer can also register online and they can browse art works that are arranged in different categories scientifically. Each Customer can create their own gallery to see his favorite art works with out much difficult. And each user has the right to purchase an art work using the integrated payment gateway and participate in auction by submitting their bids. Qualified bidder should remit the amount using payment gateway and after each valid payment the art work will be shipped within some days.

Proposed System:

ONLINE ART GALLERY is a application software and it is very helpful for the art lovers and others who wants to know the addresses where this kind of arts will we sold. This application helps the end-users to search their arts and paintings and they can place order for the selected pieces. The end-user can also get the

information about the art exhibition and the respective address, so, that they can visit to those exhibitions.

Art Gallery brings you an opportunity to view online art exhibitions at our Online Art Gallery we bring you details of all art exhibitions held in the past and the forthcoming show. The Online Art Gallery is updated daily, so the user can view and buy the latest collection of contemporary art online from any where in the world. You can view and buy the latest Indian contemporary art collection available at their exhibitions and also at their online gallery.

2. System Analysis

After analyzing the requirements of the task to be performed, the next step is to analyze the problem and understand its context. The first activity in the phase is studying the existing system and other is to understand the requirements and domain of the new system. Both the activities are equally important, but the first activity serves as a basis of giving the functional specifications and then successful design of the proposed system. Understanding the properties and requirements of a new system is more difficult and requires creative thinking and understanding of existing running system is also difficult, improper understanding of present system can lead diversion from solution.

2.1 Analysis Model

The model that is basically being followed is the WATER FALL MODEL, which states that the phases are organized in a linear order. First of all the feasibility study is done. Once that part is over the requirement analysis and project planning begins. If system exists one and modification and addition of new module is needed, analysis of present system can be used as basic model.

The design starts after the requirement analysis is complete and the coding begins after the design is complete. Once the programming is completed, the testing is done. In this model the sequence of activities performed in a software development project are :-

- Requirement
- Analysis
- System design
- Implementation
- Testing
- Maintenance

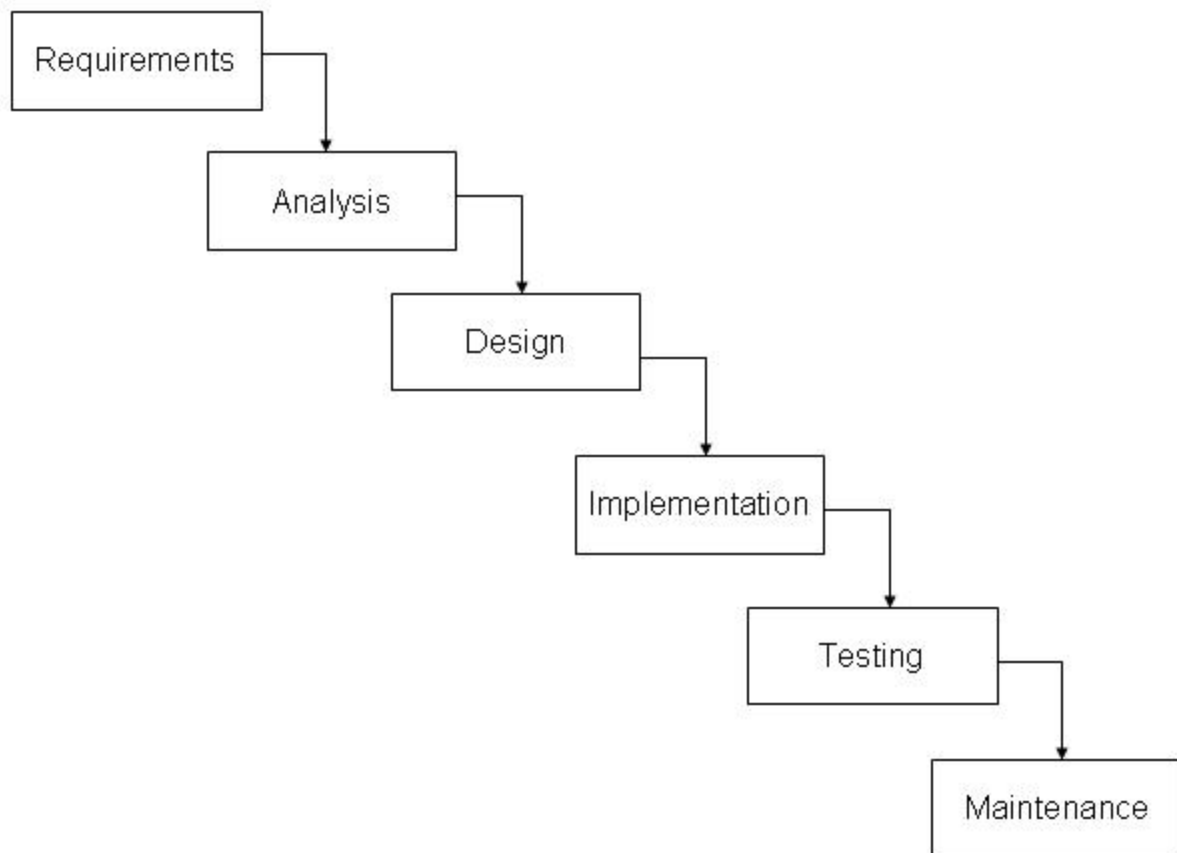


Fig: 2.1 Analysis Model

Problem Definition

Existing System:

Customer can also register online and they can browse art works that are arranged in different categories scientifically. Each Customer can create their own gallery to see his favorite art works with out much difficult. And each user has the right to purchase an art work using the integrated payment gateway and participate in auction by submitting their bids. Qualified bidder should remit the amount using payment gateway and after each valid payment the art work will be shipped within some days.

Proposed System:

ONLINE ART GALLERY is a application software and it is very helpful for the art lovers and others who wants to know the addresses where this kind of arts will we sold.This application helps the end-users to search their arts and paintings and they can place order for the selected pieces. The end-user can also get the information about the art exhibition and the respective address, so, that they can visit to those exhibitions.

Art Gallery brings you an opportunity to view online art exhibitions at our Online Art Gallery we bring you details of all art exhibitions held in the past and the forthcoming show. The Online Art Gallery is updated daily, so the user can view and buy the latest collection of contemporary art online from any where in the world. You can view and buy the latest Indian contemporary art collection available at their exhibitions and also at their online gallery.

Software Requirements Specification

3.1 What is SRS ?

Software Requirements Specification (SRS) is the starting point of the software developing activity. As system grows more complex it becomes evident that the goal of the entire system cannot be easily comprehended. Hence the need for the requirement phase arose. The software project is initiated by the client needs. The SRS is the means of translating the ideas of the minds of clients (the input) into a formal document (the output of the requirement phase).

The SRS phase consist of two basic activities:

3.1.1 Problem/Requirement Analysis:

The process is order and more nebulous of the two, deals with understand the problem, the goal and constraints.

3.1.2 Requirement Specification:

Here, the focus is on specifying what has been found giving analysis such as representation, specification languages and tools, and checking the specifications are addressed during this activity.

The requirement phase terminates with the production of the validate SRS document. Producing the SRS document is the basic of this phase.

3.1.3 Role of SRS:

The purpose of the SRS is to reduce the communication gap between the clients and the developers. SRS is the medium though which the client and user needs are accurately specified.

It forms the basis of software development. A good SRS should satisfy all the parties involved in the system.

HARDWARE AND SOFTWARE SPECIFICATIONS

SOFTWARE REQUIREMENTS:

Operating System	: Windows XP, 2000.
Language	: C#.NET
Technologies	: Microsoft.NET, ASP.NET, ADO.NET
Data Bases	: Microsoft Sql Server 2005
IDE	: Visual Studio 2008

HARDWARE REQUIREMENTS:

Processor	: Any Processor above 500 MHz.
Ram	: 128Mb.
Hard Disk	: 10 Gb.
Compact Disk	: 650 Mb.
Input device	: Standard Keyboard and Mouse.
Output device	: High Resolution Monitor.

Microsoft.NET Framework

The .NET Framework is a new computing platform that simplifies application development in the highly distributed environment of the Internet. The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that guarantees safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

The .NET Framework has two main components: the common language runtime and the .NET Framework class library. The common language runtime is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that ensure security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code. The class library, the other main component of the .NET Framework, is a comprehensive, object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on

the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

The .NET Framework can be hosted by unmanaged components that load the common language runtime into their processes and initiate the execution of managed code, thereby creating a software environment that can exploit both managed and unmanaged features. The .NET Framework not only provides several runtime hosts, but also supports the development of third-party runtime hosts.

For example, ASP.NET hosts the runtime to provide a scalable, server-side environment for managed code. ASP.NET works directly with the runtime to enable Web Forms applications and XML Web services, both of which are discussed later in this topic.

Internet Explorer is an example of an unmanaged application that hosts the runtime (in the form of a MIME type extension). Using Internet Explorer to host the runtime enables you to embed managed components or Windows Forms controls in HTML documents. Hosting the runtime in this way makes managed mobile code (similar to Microsoft® ActiveX® controls) possible, but with significant improvements that only managed code can offer, such as semi-trusted execution and secure isolated file storage.

The following illustration shows the relationship of the common language runtime and the class library to your applications and to the overall system. The illustration also shows how managed code operates within a larger architecture.

Features of the Common Language Runtime

The common language runtime manages memory, thread execution, code execution, code safety verification, compilation, and other system services. These features are intrinsic to the managed code that runs on the common language runtime.

With regards to security, managed components are awarded varying degrees of trust, depending on a number of factors that include their origin (such as the Internet, enterprise network, or local computer). This means that a managed component might or might not be able to perform file-access operations, registry-access operations, or other sensitive functions, even if it is being used in the same active application.

The runtime enforces code access security. For example, users can trust that an executable embedded in a Web page can play an animation on screen or sing a song, but cannot access their personal data, file system, or network. The security features of the runtime thus enable legitimate Internet-deployed software to be exceptionally feature rich.

The runtime also enforces code robustness by implementing a strict type- and code-verification infrastructure called the common type system (CTS). The CTS ensures that all managed code is self-describing. The various Microsoft and third-party language compilers

Generate managed code that conforms to the CTS. This means that managed code can consume other managed types and instances, while strictly enforcing type fidelity and type safety.

In addition, the managed environment of the runtime eliminates many common software issues. For example, the runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. This automatic memory management resolves the two most common application errors, memory leaks and invalid memory references.

The runtime also accelerates developer productivity. For example, programmers can write applications in their development language of choice, yet take full advantage of the runtime, the class library, and components written in other languages by other developers. Any compiler vendor who chooses to target the runtime can do so. Language compilers that target the .NET Framework make the features of the .NET Framework available to existing code written in that language, greatly easing the migration process for existing applications.

While the runtime is designed for the software of the future, it also supports software of today and yesterday. Interoperability between managed and unmanaged code enables developers to continue to use necessary COM components and DLLs.

The runtime is designed to enhance performance. Although the common language runtime provides many standard runtime services, managed code is never interpreted. A feature called just-in-time (JIT) compiling enables all managed code to run in the native machine language of the system on which it is executing. Meanwhile, the memory manager removes the possibilities of

fragmented memory and increases memory locality-of-reference to further increase performance.

Finally, the runtime can be hosted by high-performance, server-side applications, such as Microsoft® SQL Server™ and Internet Information Services (IIS). This infrastructure enables you to use managed code to write your business logic, while still enjoying the superior performance of the industry's best enterprise servers that support runtime hosting.

.NET Framework Class Library

The .NET Framework class library is a collection of reusable types that tightly integrate with the common language runtime. The class library is object oriented, providing types from which your own managed code can derive functionality. This not only makes the .NET Framework types easy to use, but also reduces the time associated with learning new

features of the .NET Framework. In addition, third-party components can integrate seamlessly with classes in the .NET Framework.

For example, the .NET Framework collection classes implement a set of interfaces that you can use to develop your own collection classes. Your collection classes will blend seamlessly with the classes in the .NET Framework.

As you would expect from an object-oriented class library, the .NET Framework types enable you to accomplish a range of common programming tasks, including tasks such as string management, data collection, database connectivity, and file access. In addition to these common tasks, the class library includes types that support a variety of specialized development scenarios. For example, you can use the .NET Framework to develop the following types of applications and services:

- Console applications.
- Scripted or hosted applications.
- Windows GUI applications (Windows Forms).
- ASP.NET applications.
- XML Web services.
- Windows services.

For example, the Windows Forms classes are a comprehensive set of reusable types that vastly simplify Windows GUI development. If you write an ASP.NET Web Form application, you can use the Web Forms classes.

Client Application Development

Client applications are the closest to a traditional style of application in Windows-based programming. These are the types of applications that display windows or forms on the desktop, enabling a user to perform a task. Client applications include applications such as word processors and spreadsheets, as well as custom business applications such as data-entry tools, reporting tools, and so on. Client applications usually employ windows, menus, buttons, and other GUI elements, and they likely access local resources such as the file system and peripherals such as printers.

Another kind of client application is the traditional ActiveX control (now replaced by the managed Windows Forms control) deployed over the Internet as a Web page. This application is much like other client applications: it is executed natively, has access to local resources, and includes graphical elements.

In the past, developers created such applications using C/C++ in conjunction with the Microsoft Foundation Classes (MFC) or with a rapid application development (RAD) environment such as Microsoft® Visual Basic®. The .NET Framework incorporates aspects of these existing products into a single, consistent development environment that drastically simplifies the development of client applications.

The Windows Forms classes contained in the .NET Framework are designed to be used for GUI development. You can easily create command windows, buttons, menus, toolbars, and other screen elements with the flexibility necessary to accommodate shifting business needs.

For example, the .NET Framework provides simple properties to adjust visual attributes associated with forms. In some cases the underlying operating system does not support changing these attributes directly, and in these cases the .NET Framework automatically recreates the forms. This is

one of many ways in which the .NET Framework integrates the developer interface, making coding simpler and more consistent.

Unlike ActiveX controls, Windows Forms controls have semi-trusted access to a user's computer. This means that binary or natively executing code can access some of the resources on the user's system (such as GUI elements and limited file access) without being able to access or compromise other resources. Because of code access security, many applications that once needed to be installed on a user's system can now be safely deployed through the Web. Your applications can implement the features of a local application while being deployed like a Web page.

C#.Net for Windows Application

Overview of the .NET Framework

The .NET Framework is a managed type-safe environment for application development and execution. The .NET Framework manages all aspects of your program's execution. It allocates memory for the storage of data and instructions, grants or denies the appropriate permissions to your application, initiates and manages application execution, and manages the reallocation of memory from resources that are no longer needed. The .NET Framework consists of two main components: the common language runtime and the .NET Framework class library.

The common language runtime can be thought of as the environment that manages code execution. It provides core services, such as code compilation, memory allocation, thread management, and garbage collection. Through the common type system (CTS), it enforces strict type-safety and ensures that code is executed in a safe environment by also enforcing code access security.

The .NET Framework class library provides a collection of useful and reusable types that are designed to integrate with the common language runtime. The types provided by the .NET Framework are object-oriented and fully extensible, and they allow you to seamlessly integrate your applications with the .NET Framework.

Languages and the .NET Framework

The .NET Framework is designed for cross-language compatibility, which means, simply, that .NET components can interact with each other no matter what supported language they were written in originally. So, an application written in Microsoft Visual Basic .NET might reference a dynamic-link library (DLL) file written in Microsoft Visual C#, which in turn might access a resource written in managed Microsoft Visual C++ or any other .NET language. This language interoperability extends to full object-oriented inheritance. A Visual Basic .NET class might be derived from a C# class, for example, or vice versa.

This level of cross-language compatibility is possible because of the common language runtime. When a .NET application is compiled, it is converted from the language in which it was written (Visual Basic .NET, C#, or any other .NET-compliant language) to Microsoft Intermediate Language (MSIL or IL). MSIL is a low-level language that the common language runtime can read and understand. Because all .NET executables and DLLs exist as MSIL, they can freely interoperate. The Common Language Specification (CLS) defines the minimum standards to which .NET language compilers must conform. Thus, the CLS ensures that any source code successfully compiled by a .NET compiler can interoperate with the .NET Framework.

The CTS ensures type compatibility between .NET components. Because .NET applications are converted to IL prior to deployment and execution, all primitive data types are represented as .NET types. Thus, a Visual Basic Integer and a C# int are both represented in IL code as a System.Int32. Because both languages use a common type system, it is possible to transfer data between components and avoid time-consuming conversions or hard-to-find errors.

Visual Studio .NET ships with languages such as Visual Basic .NET, Visual C#, and Visual C++ with managed extensions, as well as the JScript scripting language. You can also write managed code for the .NET Framework in other languages. Third-party tools and compilers exist for Fortran, Cobol, Perl, and a host of other languages. All of these languages share the same cross-language compatibility and inheritability. Thus, you can write code for the .NET Framework in the language of your choice, and it will be able to interact with code written for the .NET Framework in any other language.

.NET Framework Architecture



The Structure of a .NET Application

To understand how the common language runtime manages code execution, you must examine the structure of a .NET application. The primary unit of a .NET application is the assembly. An assembly is a self-describing collection of code, resources, and metadata. The assembly manifest contains information about what is contained within the assembly. The assembly manifest provides:

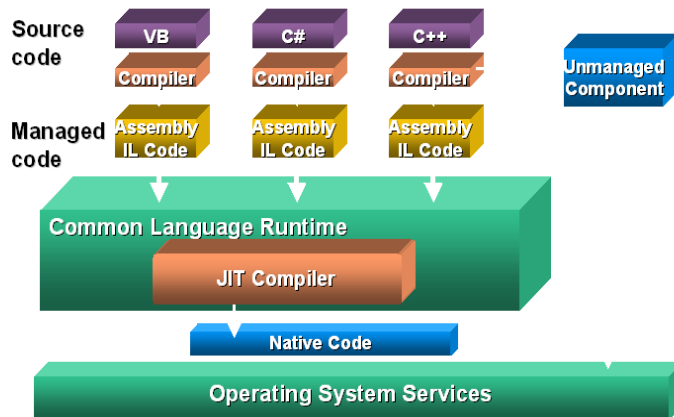
- Identity information, such as the assembly's name and version number
- A list of all types exposed by the assembly
- A list of other assemblies required by the assembly
- A list of code access security instructions, including permissions required by the assembly and permissions to be denied the assembly

Each assembly has one and only one assembly manifest, and it contains all the description information for the assembly. However, the assembly manifest can be contained in its own file or within one of the assembly's modules.

An assembly contains one or more modules. A module contains the code that makes up your application or library, and it contains metadata that describes that code. When you compile a project into an assembly, your code is converted from high-level code to IL. Because all managed code is first converted to IL code, applications written in different languages can easily interact. For example, one developer might write an application in Visual C# that accesses a DLL in Visual Basic .NET. Both resources will be converted to IL modules before being executed, thus avoiding any language-incompatibility issues.

Each module also contains a number of types. Types are templates that describe a set of data encapsulation and functionality. There are two kinds of types: reference types (classes) and value types (structures). These types are discussed in greater detail in Lesson 2 of this chapter. Each type is described to the common language runtime in the assembly manifest. A type can contain fields, properties, and methods, each of which should be related to a common functionality. For example, you might have a class that represents a bank account. It contains fields, properties, and methods related to the functions needed to implement a bank account. A field represents storage of a particular type of data. One field might store the name of an account holder, for example. Properties are similar to fields, but properties usually provide some kind of validation when data is set or retrieved. You might have a property that represents an account balance. When an attempt is made to change the value, the property can check to see if the attempted change is greater than a predetermined limit. If the value is greater than the limit, the property does not allow the change. Methods represent behavior, such as actions taken on data stored within the class or changes to the user interface. Continuing with the bank account example, you might have a Transfer method that transfers a balance from a checking account to a savings account, or an Alert method that warns users when their balances fall below a predetermined level.

CLR Execution Model



Compilation and Execution of a .NET Application

When you compile a .NET application, it is not compiled to binary machine code; rather, it is converted to IL. This is the form that your deployed application takes—one or more assemblies consisting of executable files and DLL files in IL form. At least one of these assemblies will contain an executable file that has been designated as the entry point for the application.

When execution of your program begins, the first assembly is loaded into memory. At this point, the common language runtime examines the assembly manifest and determines the requirements to run the program. It examines security permissions requested by the assembly and compares them with the system's security policy. If the system's security policy does not allow the requested permissions, the application will not run. If the application passes the system's security policy, the common language runtime executes the code. It creates a process for the application to run in and begins application execution. When execution starts, the first bit of code that needs to be executed is loaded into memory and compiled into native binary code from IL by the common language runtime's Just-In-Time (JIT) compiler. Once compiled, the code is executed and stored in memory as native code. Thus, each portion of code is compiled only once when an application executes. Whenever program execution branches to code that has not yet run, the JIT compiler compiles it ahead of execution and stores it in memory as binary code. This way, application performance is maximized because only the parts of a program that are executed are compiled.

2: The .NET Base Class Library

- The .NET base class library is a collection of object-oriented types and interfaces that provide object models and services for many of the complex programming tasks you will face. Most of the types presented by the .NET base class library are fully extensible, allowing you to build types that incorporate your own functionality into your managed code.

The .NET Framework base class library contains the base classes that provide many of the services and objects you need when writing your applications. The class library is organized into namespaces. A namespace is a logical grouping of types that perform related functions. For example, the `System.Windows.Forms` namespace contains all the types that make up Windows forms and the controls used in those forms.

Namespaces are logical groupings of related classes. The namespaces in the .NET base class library are organized hierarchically. The root of the .NET Framework is the `System` namespace. Other namespaces can be accessed with the period operator. A typical namespace construction appears as follows:

```
System
System.Data
System.Data.SqlClient
```

The first example refers to the `System` namespace. The second refers to the `System.Data` namespace. The third example refers to the `System.Data.SqlClient` namespace. Table 1.1 introduces some of the more commonly used .NET base class namespaces.

Table 1-1. Representative .NET Namespaces

Namespace	Description
System	This namespace is the root for many of the low-level types required by the .NET Framework. It is the root for primitive data types as well, and it is the root for all the other namespaces in the .NET base class library.

`System.Collections` This namespace contains classes that represent a variety

Table 1-1. Representative .NET Namespaces

Namespace	Description
	of different container types, such as ArrayList, SortedList, Queue, and Stack. You also can find abstract classes, such as CollectionBase, which are useful for implementing your own collection functionality.
System.ComponentModel	This namespace contains classes involved in component creation and containment, such as attributes, type converters, and license providers.
System.Data	This namespace contains classes required for database access and manipulations, as well as additional namespaces used for data access.
System.Data.Common	This namespace contains a set of classes that are shared by the .NET managed data providers.
System.Data.OleDb	This namespace contains classes that make up the managed data provider for OLE DB data access.
System.Data.SqlClient	This namespace contains classes that are optimized for interacting with Microsoft SQL Server.
System.Drawing	This namespace exposes GDI+ functionality and provides classes that facilitate graphics rendering.
System.IO	In this namespace, you will find types for handling file system I/O.
System.Math	This namespace is home to common mathematics functions such as extracting roots and trigonometry.
System.Reflection	This namespace provides support for obtaining information and dynamic creation of types at

runtime.

System.Security	This namespace is home to types dealing with permissions, cryptography, and code access security.
System.Threading	This namespace contains classes that facilitate the implementation of multithreaded applications.
System.Windows.Forms	This namespace contains types involved in creating standard Windows applications. Classes that represent forms and controls reside here as well.

The namespace names are self-descriptive by design. Straightforward names make the .NET Framework easy to use and allow you to rapidly familiarize yourself with its contents.

Reference Types and Value Types

Types in the .NET Framework come in two varieties: value types and reference types. The primary difference between value types and reference types has to do with the way variable data is accessed. To understand this difference, a little background on memory dynamics is required.

Application data memory is divided into two primary components, the stack and the heap. The stack is an area of memory reserved by the application to run the program. The stack is analogous to a stack of dinner plates. Plates are placed on the stack one on top of another. When a plate is removed from the stack, it is always the last one to have been placed on top that is removed first. So it is with program variables. When a function is called, all the variables used by the function are pushed onto the stack. If that function calls additional functions, it pushes additional variables onto the stack. When the most recently called function terminates, all of its variables go out of scope (meaning that they are no longer available to the application) and are popped off the stack. Memory consumed by those variables is then freed up, and program execution continues.

The heap, on the other hand, is a separate area of memory reserved for the creation of reusable objects. The common language runtime manages

allocation of heap memory for objects and controls the reclamation of memory from unused objects through garbage collection.

All the data associated with a value type is allocated on the stack. When a variable of a value type goes out of scope, it is destroyed and its memory is reclaimed. A variable of a reference type, on the other hand, exists in two memory locations. The actual object data is allocated on the heap. A variable containing a pointer to that object is allocated on the stack. When that variable is called by a function, it returns the memory address for the object to which it refers. When that variable goes out of scope, the object reference is destroyed but the object itself is not. If any other references to that object exist, the object remains intact. If the object is left without any references, it is subject to garbage collection. (See Lesson 6 of this chapter.)

Examples of value types include primitives, such as Integer (int), Boolean (bool), Char (char), and so on, as well as user-defined types such as Structure (struct) and Enumeration (enum). Classes represent the majority of reference types. Other reference types include the interface, delegate, and array types. Classes and structures are discussed in Lesson 3 of this chapter, and other reference and value types are discussed in Chapter 3.

Using .NET Framework Types in Your Application

When you begin writing an application, you automatically begin with a reference to the .NET Framework base class library. You reference it so that your application is aware of the base class library and is able to create instances of the types represented by it.

Value Types

```
int myInteger;
```

This line tells the runtime to allocate the appropriate amount of memory to hold an integer variable. Although this line creates the variable, it does not assign a value to it. You can assign a value using the assignment operator, as follows:

```
myInteger = 42;
```

You can also choose to assign a value to a variable upon creation, as shown in this example:

```
int myInteger = 42;
```

Reference Types

Creating an instance of a type is a two-step process. The first step is to declare the variable as that type, which allocates the appropriate amount of memory for that variable but does not actually create the object. The following syntax declares an object:

```
System.Windows.Forms.Form myForm;
```

This line tells the runtime to set aside enough memory to hold a Form variable and assigns it the name myForm, but it does not actually create the Form object in memory. The second step, called instantiation, actually creates the object. An example of instantiation follows:

```
myForm = new System.Windows.Forms.Form();
```

This line makes a call to the constructor method of the type System.Windows.Forms.Form by way of the New (new) keyword. The constructor is a special method that is invoked only at the beginning of an object's lifetime. It contains any code that must be executed for the object to work (assigning values to properties, for example). If any parameters were required by the constructor, they would be contained within the parentheses at the end of the line. The following example shows declaration and instantiation of a hypothetical Widget class that requires a string as a parameter in the constructor.

```
Widget myWidget;  
myWidget = new Widget("This string is required by the constructor");
```

If desired, you can also combine both declaration and instantiation into a single statement. By declaring and instantiating an object in the same line, you reserve the memory for the object and immediately create the object that resides in that memory. Although there was a significant performance penalty for this shortcut in previous versions of Visual Basic, Visual Basic .NET and Visual C# are optimized to allow this behavior without any performance loss. The following example shows the one-step declaration and instantiation of a new Form:

```
System.Windows.Forms.Form myForm = new
```

```
System.Windows.Forms.Form();
```

Both value types and reference types must be initialized before use. For class and structure fields in Visual Basic .NET, types are initialized with default values on declaration. Numeric value types (such as integer) and floating-point types are assigned zero; Boolean variables are assigned False; and reference types are assigned to a null reference.

In C#, variables of a reference type have a default value of null. It is recommended that you do not rely on the default value. These variables should not be used until they have been initialized.

Using Value Type and Reference Type Variables

A variable that represents a value type contains all the data represented by that type. A variable that represents a reference type contains a reference to a particular object. This distinction is important. Consider the following example:

```
int x, y;  
x = 15;  
y = x;  
x = 30;  
// What is the value of y?
```

In this example, two integer variables named x and y are created. X is assigned a value of 15, and then y is assigned the value of x. Next the value of x is changed to 30, and the question is posed: what is the value of y? The answer to this question might seem obvious, and it is y = 15 because x and y are two separate variables and have no effect on each other when changed. When the line y = x is encountered, the value of x is copied to the value of y, and there is no further connection between the two variables.

This situation changes, however, in the case of reference types. Let's reconsider the previous example using a reference type (Form) instead of a value type.

```
System.Windows.Forms.Form x,y;  
x = new System.Windows.Forms.Form();  
x.Text = "This is Form 1";  
y = x;
```

```
x.Text = "This is Form 2";  
// What value does y.Text return?
```

What value does `y.Text` return? This time, the answer is less obvious. Because `System.Windows.Forms.Form` is a reference type, the variable `x` does not actually contain a `Form`; rather, it points to an instance of a `Form`. When the line `y = x` is encountered, the runtime copies the reference from variable `x` to `y`. Thus, the variables `x` and `y` now point to the same instance of `Form`. Because these two variables refer to the same instance of the object, they will return the same values for properties of that object. Thus, `y.Text` returns “This is Form 2”.

The Imports and Using Statements

Up to this point of the chapter, if you wanted to access a type in the .NET Framework base class library, you had to use the full name of the type, including every namespace to which it belonged. For example:

```
System.Windows.Forms.Form
```

This is called the fully-qualified name, meaning it refers both to the class and to the namespace in which it can be found. You can make your development environment “aware” of various namespaces by using the `Imports` (Visual Basic .NET) or `using` (Visual C#) statement. This technique allows you to refer to a type using only its generic name and to omit the qualifying namespaces. Thus, you could refer to `System.Windows.Forms.Form` as simply `Form`. In Visual Basic .NET, the `Imports` statement must be placed at the top of the code window, preceding any other statement (except `Option`). In Visual C#, the `using` statement must occur before any other namespace element, such as a class or struct. This example demonstrates use of this statement:

```
using System.Windows.Forms;
```

When two types of the same name exist in more than one imported namespace, you must use the fully qualified name to avoid a naming conflict. Thus, if you are using `MyNameSpaceOne` and `MyNameSpaceTwo`, and each contains a `Widget` class, you would have to refer to `MyNameSpaceOne.Widget` or `MyNameSpaceTwo.Widget` to ensure the correct result.

In C#, you can resolve namespace conflicts such as these by creating an alias. An alias allows you to choose one name to refer to another class. You create an alias using the using keyword, as shown below:

```
using myAlias = MyNameSpaceTwo.Widget;
```

After implementing an alias, you can use it in code to represent the aliased class. For example:

```
// You can now refer to MyNameSpaceTwo as myAlias. The
// following two lines produce the same result:
MyNameSpaceTwo.Widget anotherWidget = new MyNameSpaceTwo.Widg
et();
myAlias anotherWidget = new myAlias();
```

You cannot create aliases for types in this manner in Visual Basic .NET.

Referencing External Libraries

You might want to use class libraries not contained by the .NET Framework, such as libraries developed by third-party vendors or libraries you developed. To access these external libraries, you must create a reference.

To create a reference to an external library

1. In the Solution Explorer, right-click the References node of your project.
2. From the pop-up menu, choose Add Reference. The Add Reference dialog box appears.
3. Choose the appropriate tab for the library you want to reference. .NET libraries are available on the .NET tab. Legacy COM libraries appear on the COM tab, and local Visual Studio projects appear on the Projects tab.
4. Locate the library you want to reference, and double-click it to add it to the Selected components box. Click OK to confirm the choice of that reference.

Introduction to Object-Oriented Programming

Programming in the .NET Framework environment is done with objects. Objects are programmatic constructs that represent packages of related data and functionality. Objects are self-contained and expose specific functionality to the rest of the application environment without detailing the inner workings of the object itself. Objects are created from a template called a class. The .NET base class library provides a set of classes from which you can create objects in your applications. You also can use the Microsoft Visual Studio programming environment to create your own classes. This lesson introduces you to the concepts associated with object-oriented programming.

Objects, Members, and Abstraction

An object is a programmatic construct that represents something. In the real world, objects are cars, bicycles, laptop computers, and so on. Each of these items exposes specific functionality and has specific properties. In your application, an object might be a form, a control such as a button, a database connection, or any of a number of other constructs. Each object is a complete functional unit, and contains all of the data and exposes all of the functionality required to fulfill its purpose. The ability of programmatic objects to represent real-world objects is called abstraction.

Classes Are Templates for Objects

Classes can be thought of as blueprints for objects: they define all of the members of an object, define the behavior of an object, and set initial values for data when appropriate. When a class is instantiated, an in-memory instance of that class is created. This instance is called an object. To review, a class is instantiated using the New (new) keyword as follows:

When an instance of a class is created, a copy of the instance data defined by that class is created in memory and assigned to the reference variable. Individual instances of a class are independent of one another and represent separate programmatic constructs. There is generally no limit to how many copies of a single class can be instantiated at any time. To use a real-world analogy, if a car is an object, the plans for the car are the class. The plans can be used to make any number of cars, and changes to a single car do not, for the most part, affect any other cars.

Objects and Members

Objects are composed of members. Members are properties, fields, methods, and events, and they represent the data and functionality that comprise the object. Fields and properties represent data members of an object. Methods are actions the object can perform, and events are notifications an object receives from or sends to other objects when activity happens in the application.

To continue with the real-world example of a car, consider that a Car object has fields and properties, such as Color, Make, Model, Age, GasLevel, and so on. These are the data that describe the state of the object. A Car object might also expose several methods, such as Accelerate, ShiftGears, or Turn. The methods represent behaviors the object can execute. And events represent notifications. For example, a Car object might receive an EngineOverheating event from its Engine object, or it might raise a Crash event when interacting with a Tree object.

Object Models

Simple objects might consist of only a few properties, methods, and perhaps an event or two. More complex objects might require numerous properties and methods and possibly even subordinate objects. Objects can contain and expose other objects as members. For example, the TextBox control exposes a Font property, which consists of a Font object. Similarly, every instance of the Form class contains and exposes a Controls collection that comprises all of the controls contained by the form. The object model defines the hierarchy of contained objects that form the structure of an object.

An object model is a hierarchical organization of subordinate objects contained and exposed within a main object. To illustrate, let's revisit the example of a car as an object. A car is a single object, but it also consists of subordinate objects. A Car object might contain an Engine object, four Wheel objects, a Transmission object, and so on. The composition of these subordinate objects directly affects how the Car object functions as a whole. For example, if the Cylinders property of the Engine subordinate object is equal to 4, the Car will behave differently than a Car whose Engine has a Cylinders property value of 8. Contained objects can have subordinate objects of their own. For example, the contained Engine object might contain several SparkPlug objects.

Encapsulation

Encapsulation is the concept that implementation of an object is independent of its interface. Put another way, an application interacts with an object through its interface, which consists of its public properties and methods. As long as this interface remains constant, the application can continue to interact with the component, even if implementation of the interface was completely rewritten between versions.

Objects should only interact with other objects through their public methods and properties. Thus, objects should contain all of the data they require, as well as all of the functionality that works with that data. The internal data of an object should never be exposed in the interface; thus, fields rarely should be Public (public).

Returning to the Car example. If a Car object interacts with a Driver object, the Car interface might consist of a GoForward method, a GoBackward method, and a Stop method. This is all the information that the Driver needs to interact with the Car. The Car might contain an Engine object, for example, but the Driver doesn't need to know about the Engine object—all the Driver cares about is that the methods can be called and that they return the appropriate values. Thus, if one Engine object is exchanged for another, it makes no difference to the Driver as long as the interface continues to function correctly.

Polymorphism

Polymorphism is the ability of different classes to provide different implementations of the same public interfaces. In other words, polymorphism allows methods and properties of an object to be called without regard for the particular implementation of those members. For example, a Driver object can interact with a Car object through the Car public interface. If another object, such as a Truck object or a SportsCar object, exposes the same public interface, the Driver object can interact with them without regard to the specific implementation of that interface. There are two principal ways through which polymorphism can be provided: interface polymorphism and inheritance polymorphism.

Interface Polymorphism

An interface is a contract for behavior. Essentially, it defines the members a class should implement, but states nothing at all about the details of that implementation. An object can implement many different interfaces, and many diverse classes can implement the same interface. All objects implementing the same interface are capable of interacting with other objects through that interface. For example, the Car object in the previous examples might implement the IDrivable interface (by convention, interfaces usually begin with I), which specifies the GoForward, GoBackward, and Halt methods. Other classes, such as Truck, Forklift, or Boat might implement this interface and thus are able to interact with the Driver object. The Driver object is unaware of which interface implementation it is interacting with; it is only aware of the interface itself. Interface polymorphism is discussed in detail in Lesson 3.

Inheritance Polymorphism

Inheritance allows you to incorporate the functionality of a previously defined class into a new class and implement different members as needed. A class that inherits another class is said to derive from that class, or to inherit from that class. A class can directly inherit from only one class, which is called the base class. The new class has the same members as the base class, and additional members can be added as needed. Additionally, the implementation of base members can be changed in the new class by overriding the base class implementation. Inherited classes retain all the characteristics of the base class and can interact with other objects as though they were instances of the base class. For example, if the Car class is the base class, a derived class might be Sport car. The Sport car class might be the base class for another derived class, the ConvertibleSportsCar. Each newly derived class might implement additional members, but the functionality defined in the original Car class is retained.

4. PROJECT DICTIONARY

4.1. DATAFLOW DIAGRAMS

Data flow diagram is used to decrease analysis the movement of data through a system store of data in the system. Data flow diagrams are the central tool basing on which components are developed.

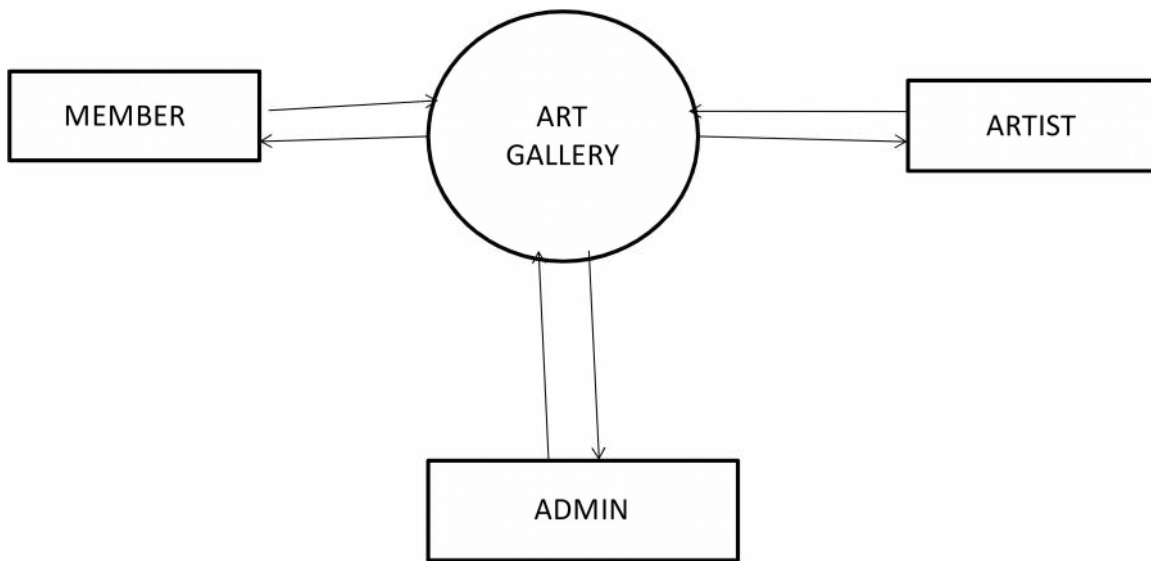
The transformation of data from input to output, through process may be describe logically and independently of physically components associated with the system. They are called logical data flow diagrams. In contrast physical data flow diagrams show the actual implementation and movement of data between people, Department, and work station.

The data flow diagram show functional composition of the system. The first level of conceptual level in context diagram is flowed by the description of input and output for each of entities the next level of dfd is level 0, which shows the main functions in the system.

Level 0 is followed by the description of the main functions. The main function further broken into functions and sub functions.

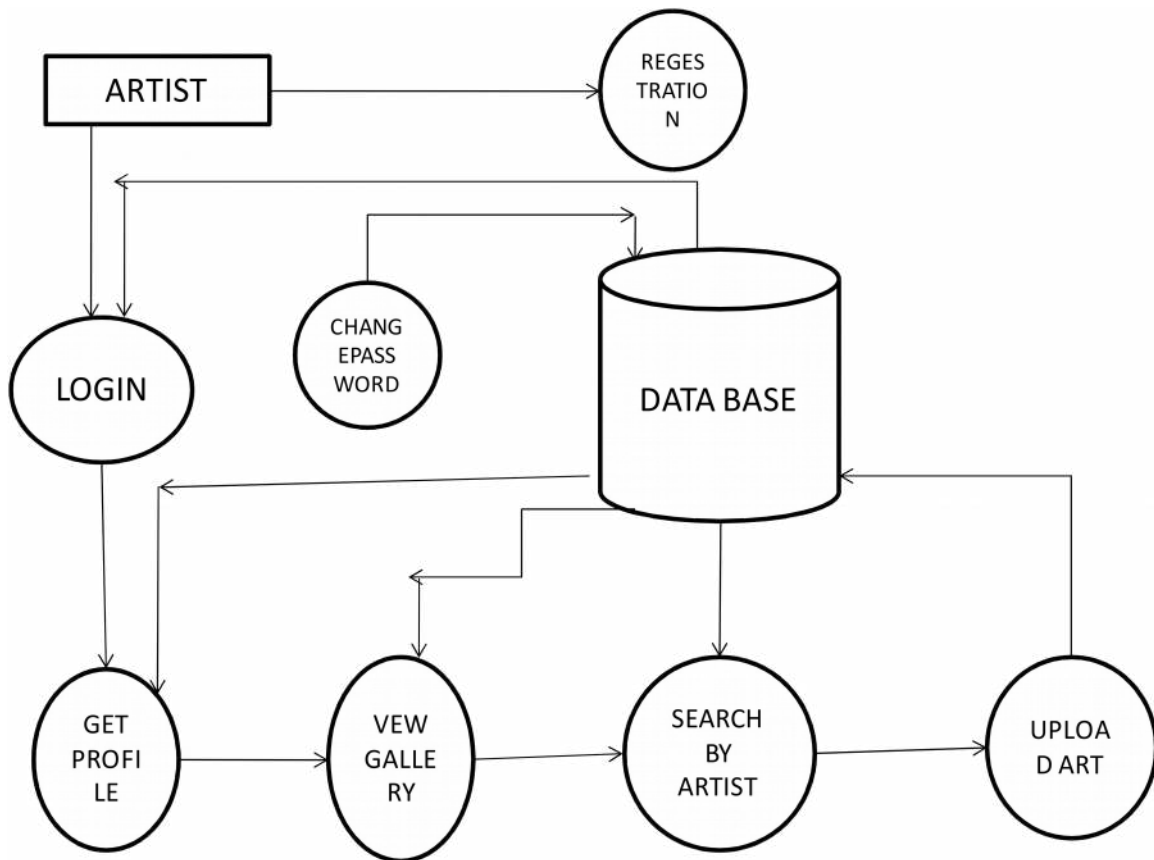
4.1. DFD Diagrams:

DFD level 0:

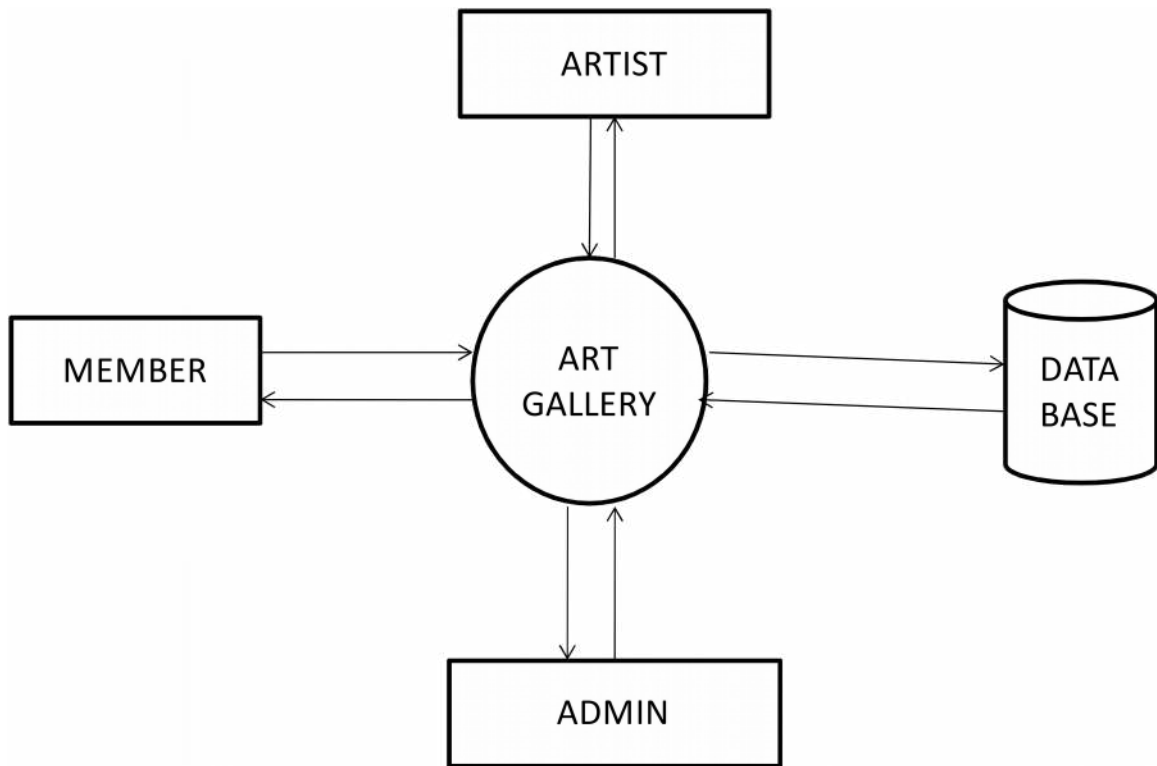


DFD level 1 member:

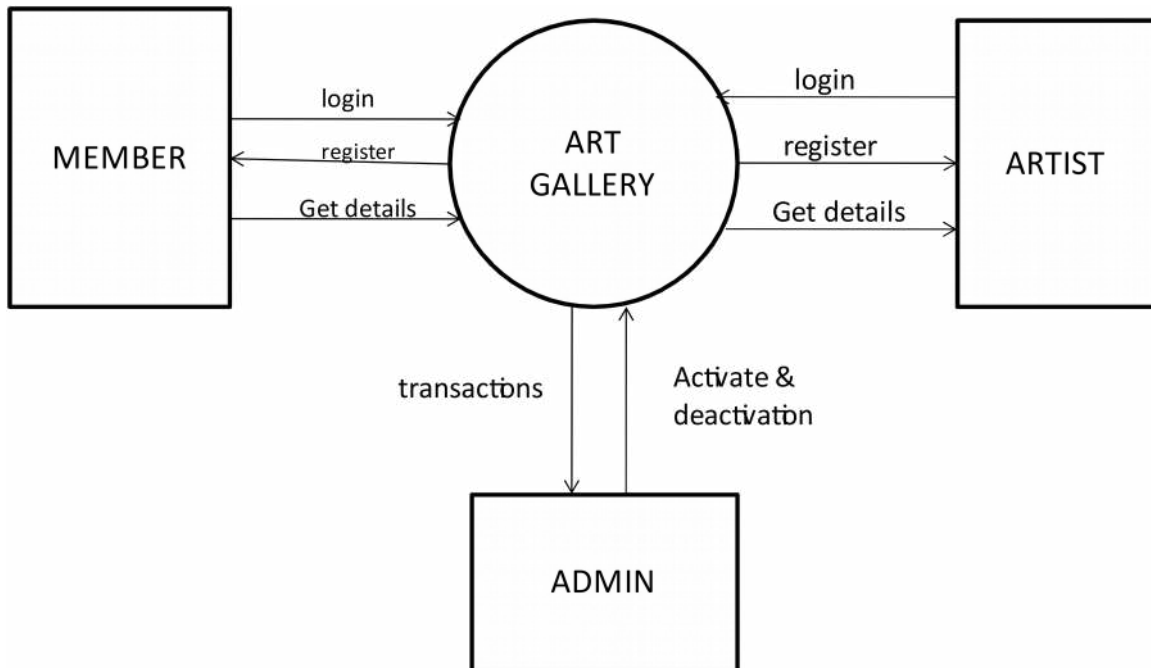
DFD Level 1 Artist:



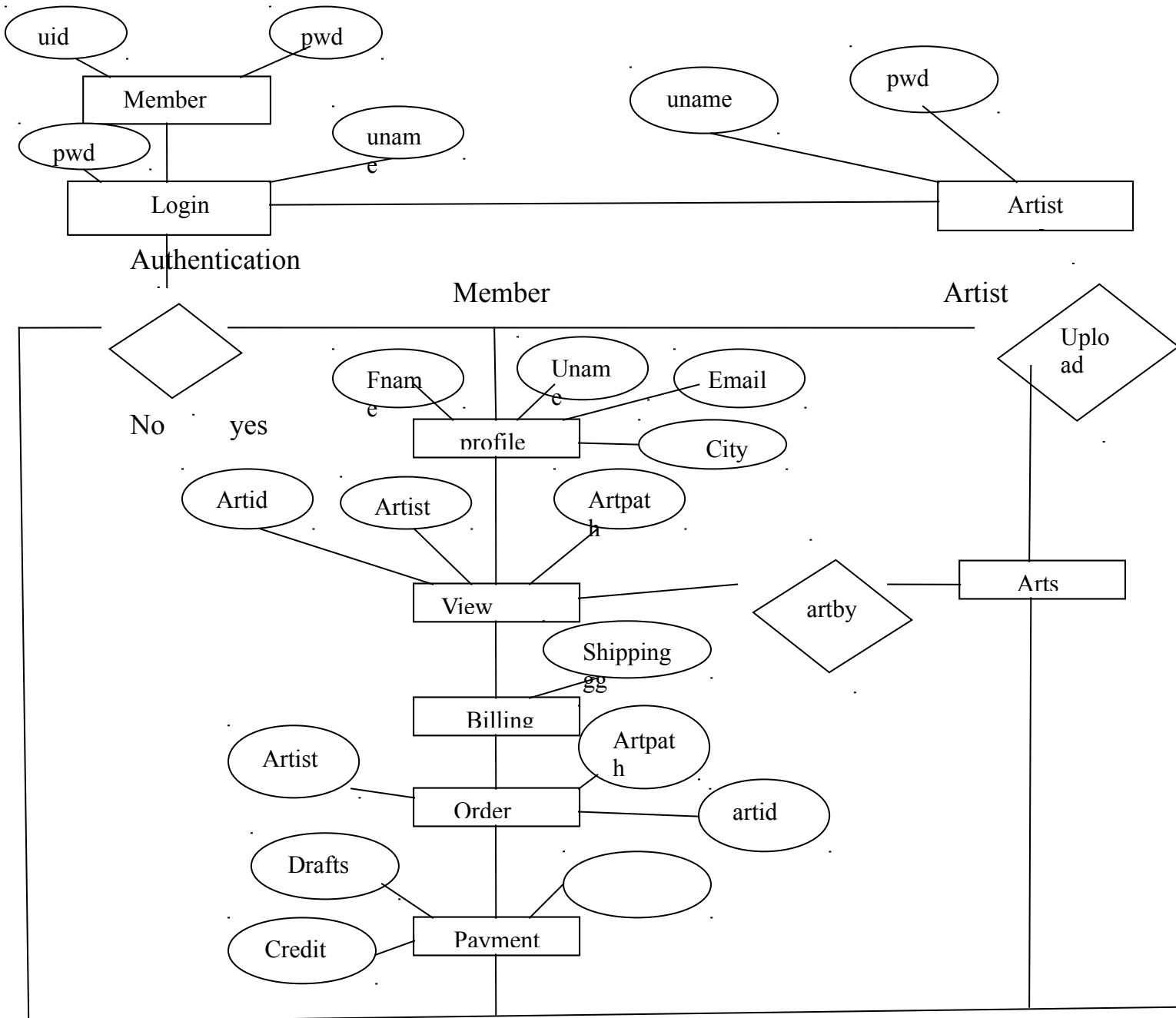
DFD Level 1 Admin:



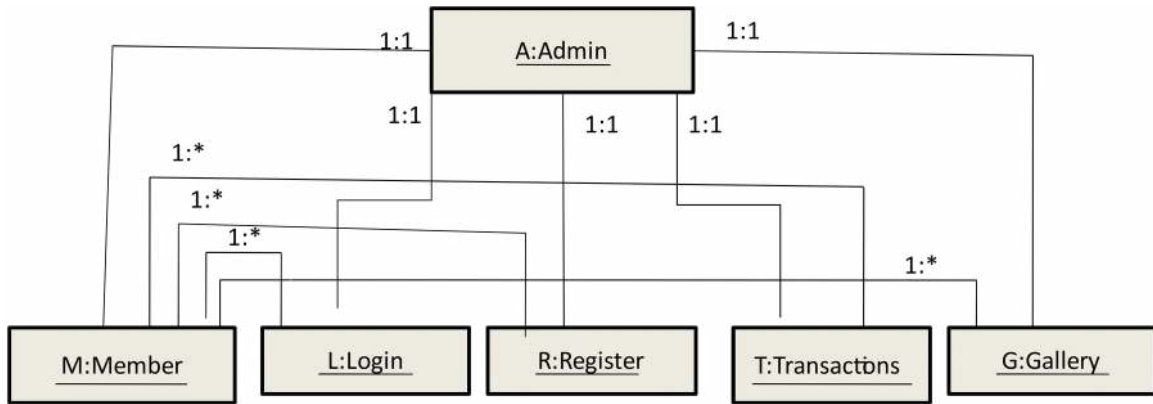
Context Diagram:



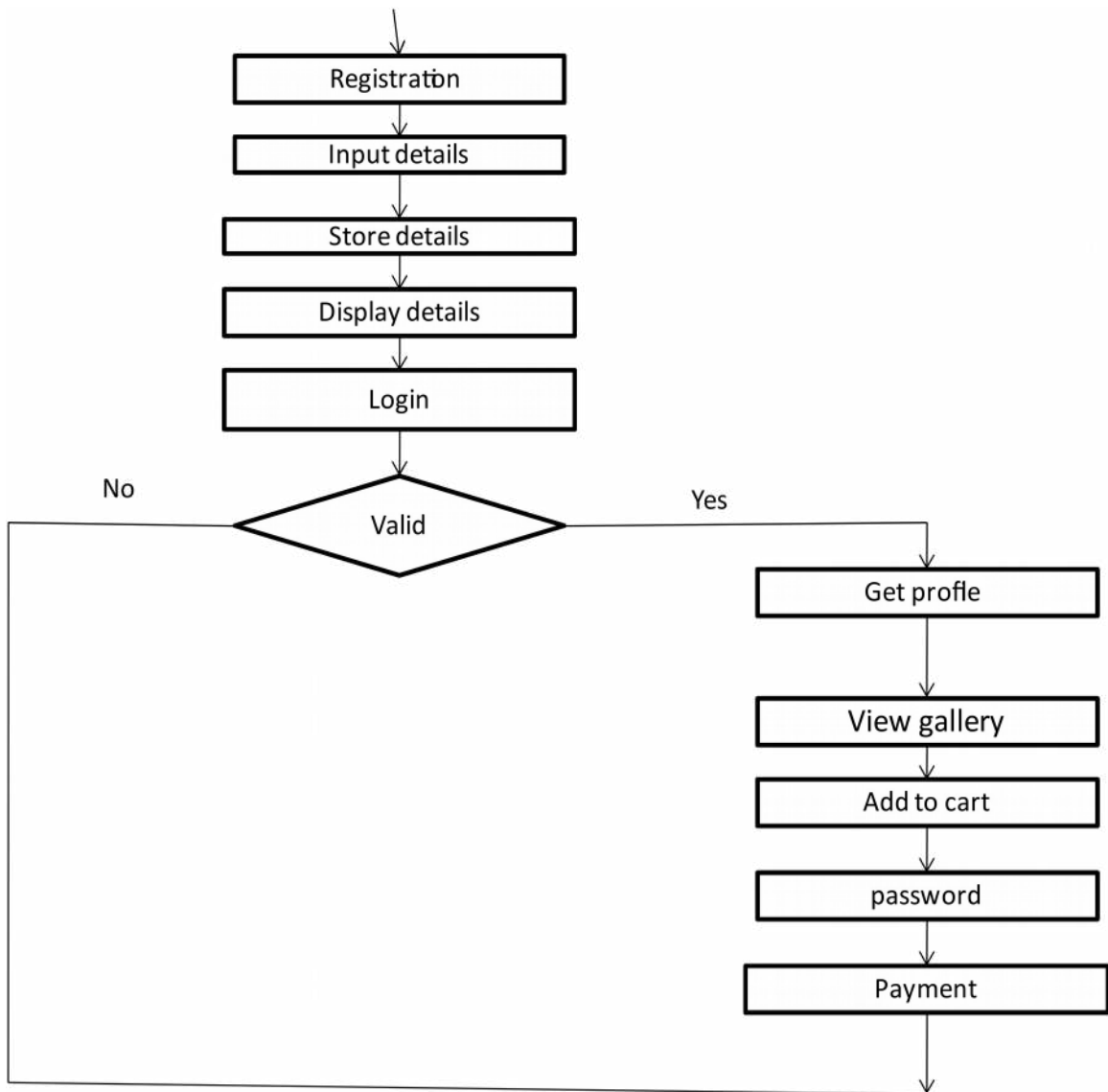
4.2. E-R Diagrams:



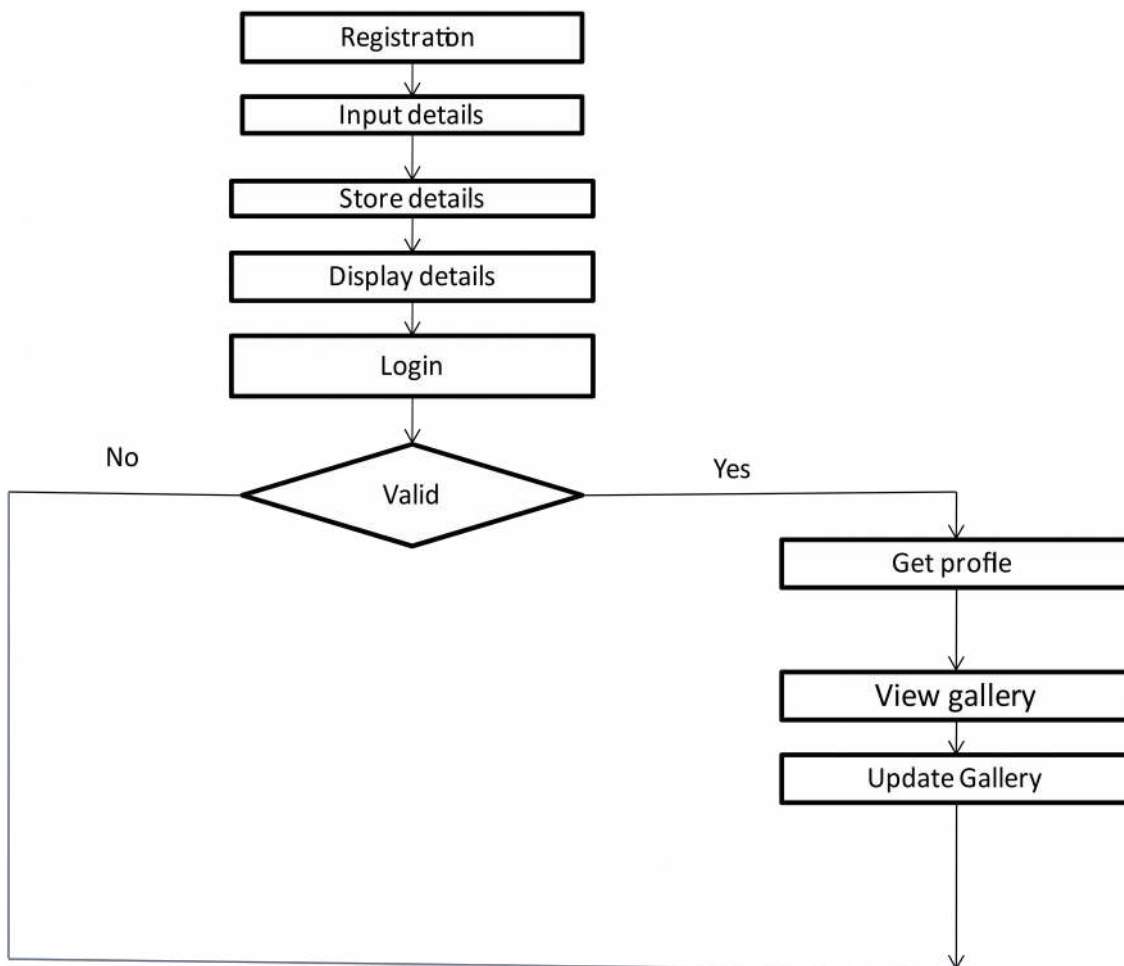
Object diagram:



Activity diagram for Member:

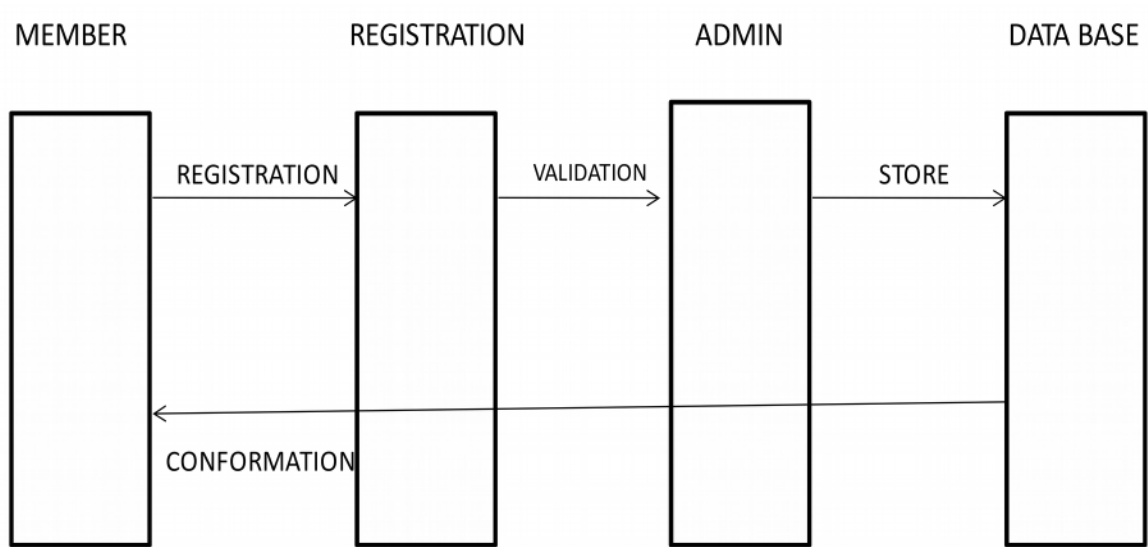


Activity diagram for Artist:

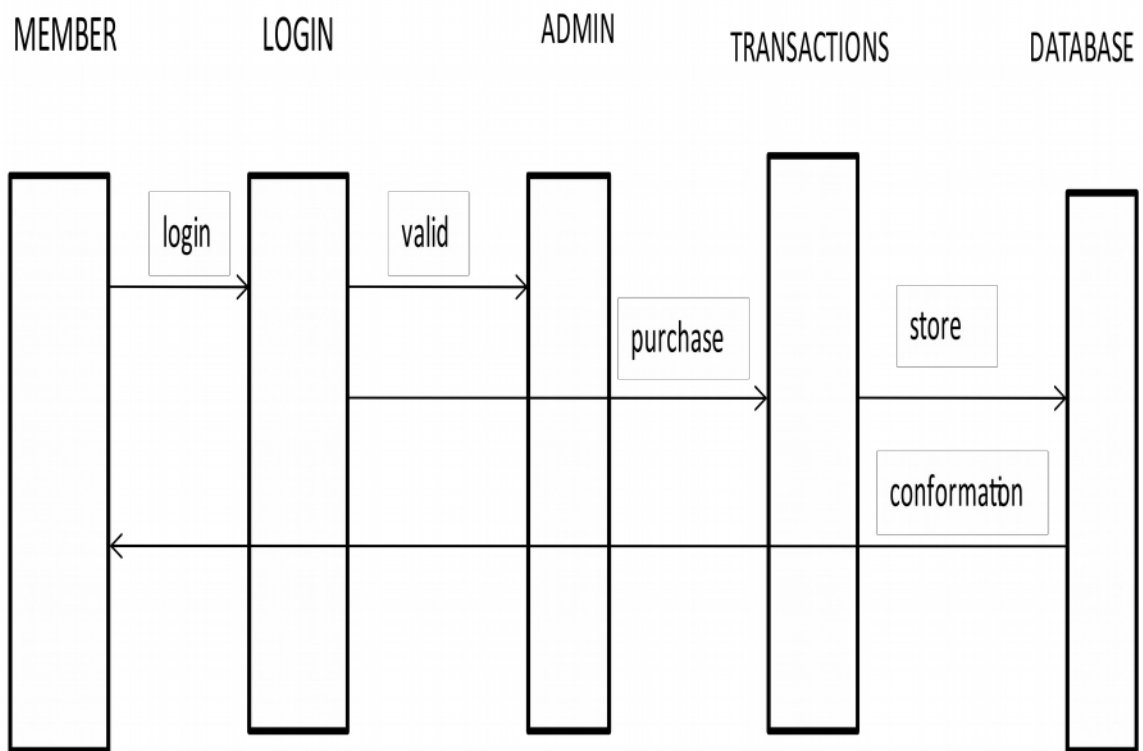


State Chat diagrams:

State chart Diagram for Online registration:



State chart Diagram for Online Transactions:



5.1 UML (Unified Modeling Language)

The Unified Modelling Language (UML) is probably the most widely known and used notation for object-oriented analysis and design. It is the result of the merger of several early contributions to object-oriented methods. The Unified Modelling Language (UML) is a standard language for writing software blueprints? The UML may be used to visualize, specify, construct, and document the artefacts. A Modelling language is a language whose vocabulary and rules focus on the conceptual and physical representation of a system. Modelling is the designing of software applications before coding. Modelling is an Essential Part of large software projects, and helpful to medium and even small projects as well. A model plays the analogous role in software development that blueprints and other plans (site maps, elevations, physical models) play in the building of a skyscraper. Using a model, those responsible for a software development project's success can assure themselves that business functionality is complete and correct, end-user needs are met, and program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, before implementation in code renders changes difficult and expensive to make.

The underlying premise of UML is that no one diagram can capture the different elements of a system in its entirety. Hence, UML is made up of nine diagrams that can be used to model a system at different points of time in the software life cycle of a system. The nine UML diagrams are:

- **Use case diagram:** The use case diagram is used to identify the primary elements and processes that form the system. The primary elements are termed as "actors" and the processes are called "use cases." The use case diagram shows which actors interact with each use case.
- **Class diagram:** The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship.

Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

- **Object diagram:** The object diagram is a special kind of class diagram. An object is an instance of a class. This essentially means that an object represents the state of a class at a given point of time while the system is running. The object diagram captures the state of different classes in the system and their relationships or associations at a given point of time.
- **State diagram:** A state diagram, as the name suggests, represents the different states that objects in the system undergo during their life cycle. Objects in the system change states in response to events. In addition to this, a state diagram also captures the transition of the object's state from an initial state to a final state in response to events affecting the system.
- **Activity diagram:** The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.
- **Sequence diagram:** A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".
- **Collaboration diagram:** A collaboration diagram groups together the interactions between different objects. The interactions are listed as numbered interactions that help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with other objects.
- **Component diagram:** The component diagram represents the high-level parts that make up the system. This diagram depicts, at a high level, what components form part of the system and how they are interrelated. A component diagram depicts the components culled after the system has undergone the development or construction phase.

- **Deployment diagram:** The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far most useful when a system is built and ready to be deployed.

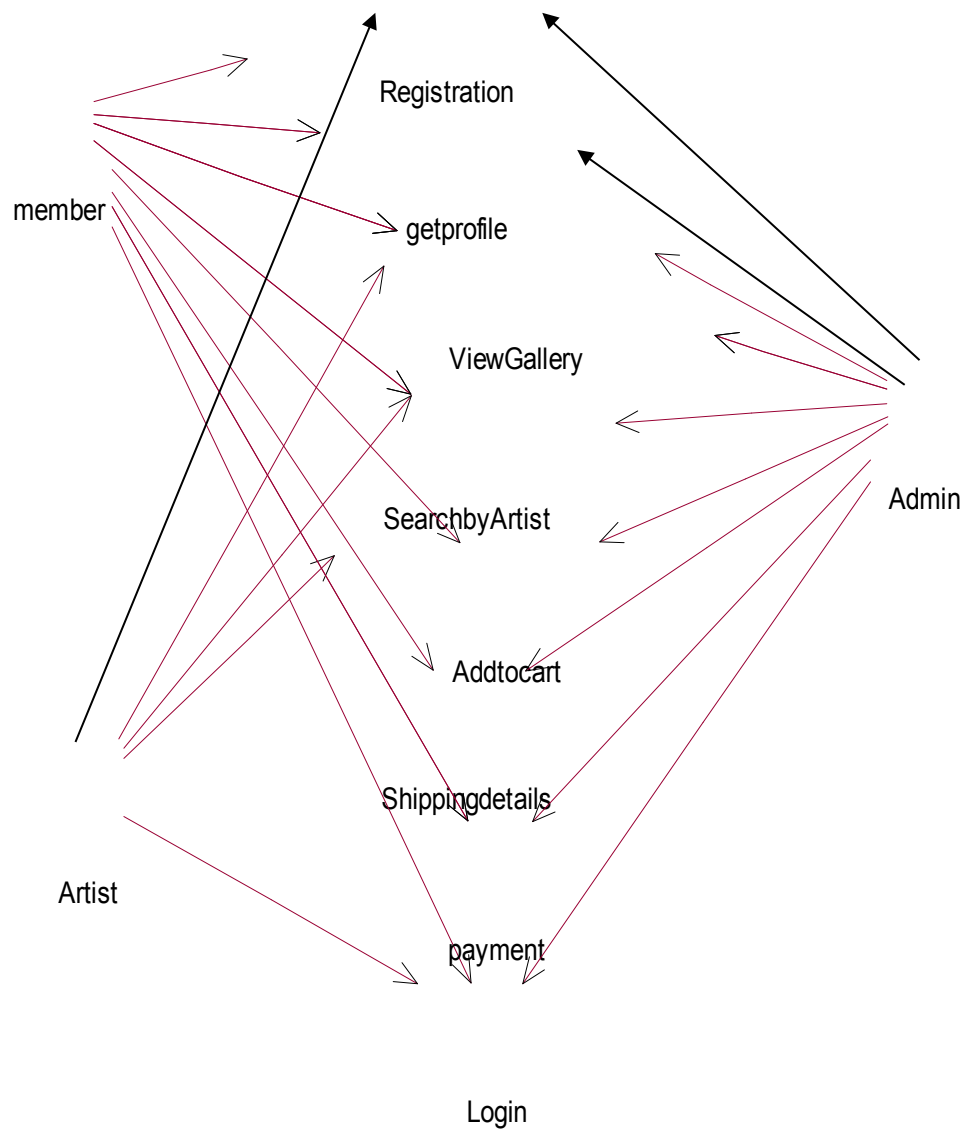
The UML diagrams that fall under each of these categories are:

- **Static**
 - Use case diagram
 - Class diagram
- **Dynamic**
 - Object diagram
 - State diagram
 - Activity diagram
 - Sequence diagram
 - Collaboration diagram
- **Implementation**
 - Component diagram
 - Deployment diagram

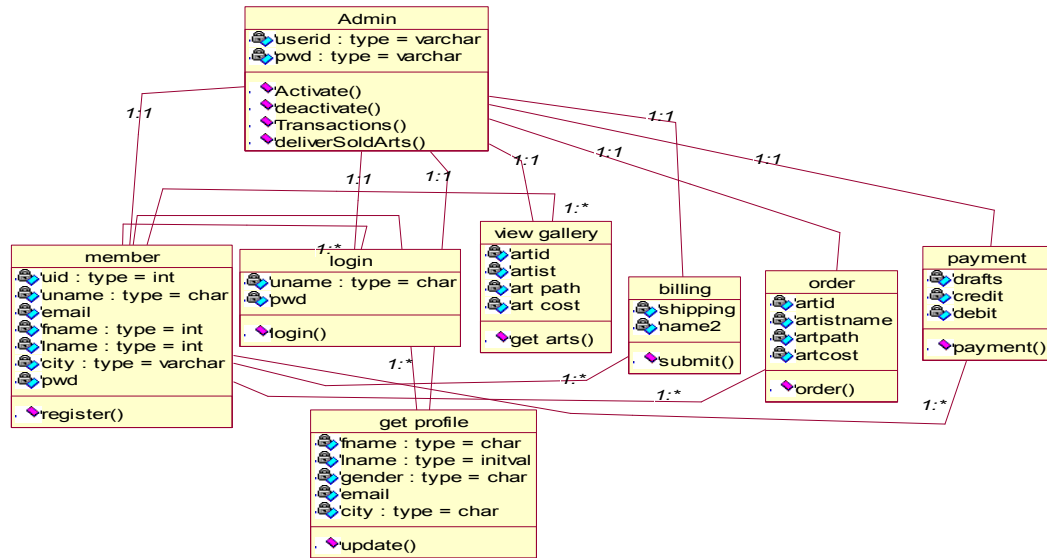
Uml diagrams:

For member:

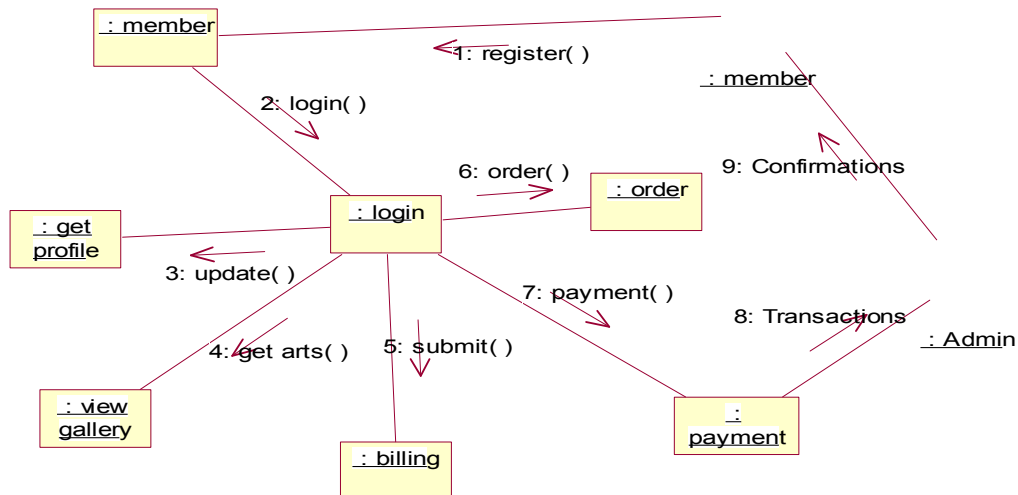
Use case diagram:



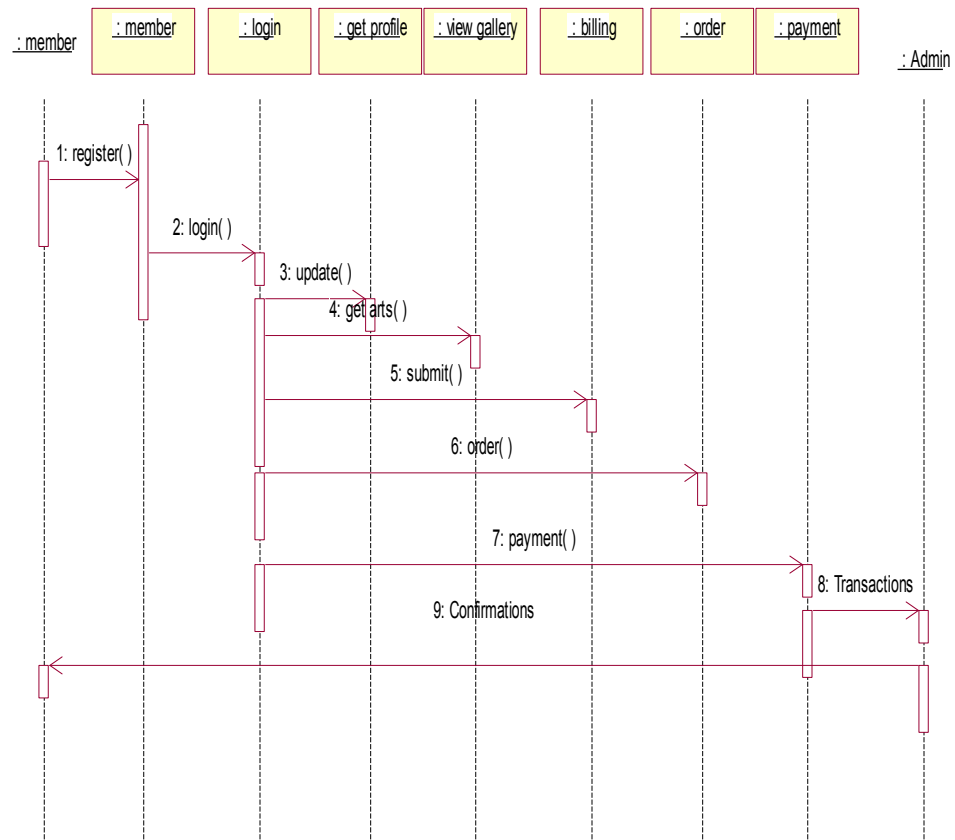
Class Diagram:



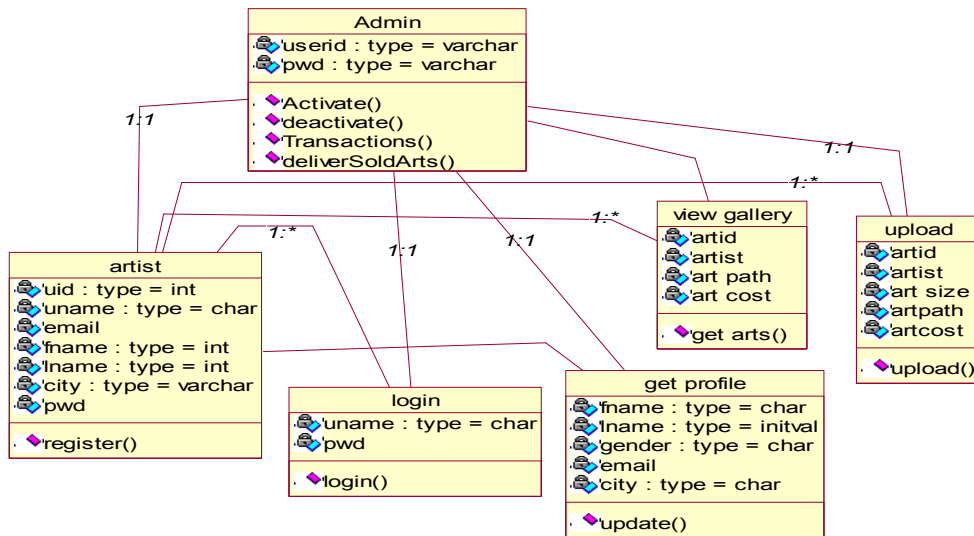
Collaboration:



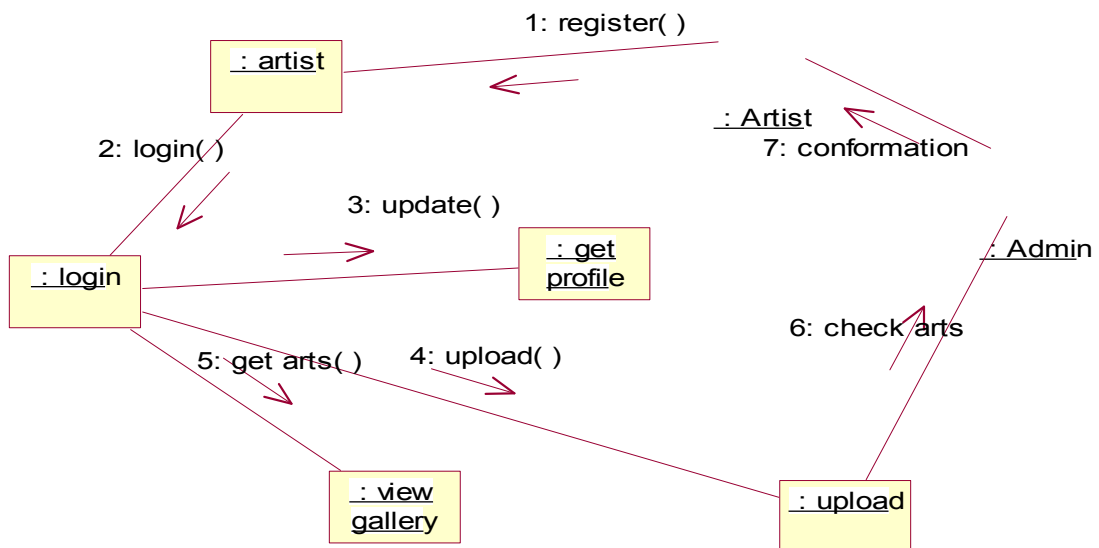
Sequence diagram:



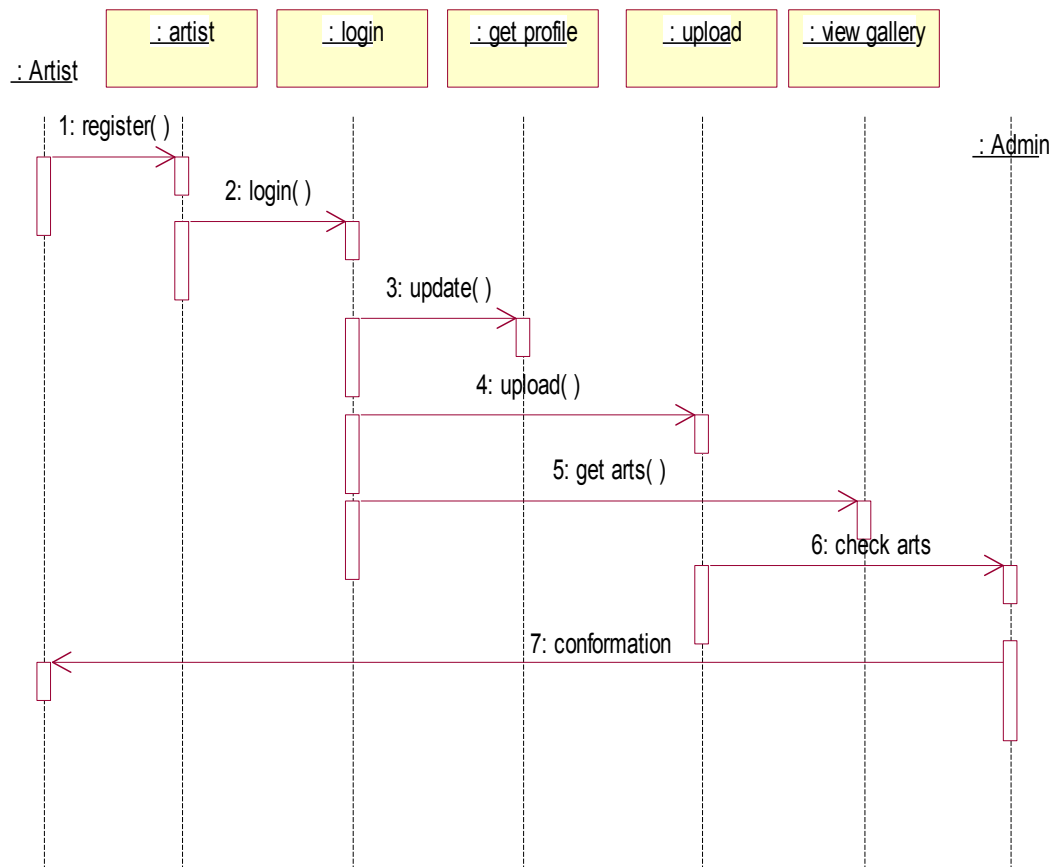
For artist:
Class diagram:



Collabora:

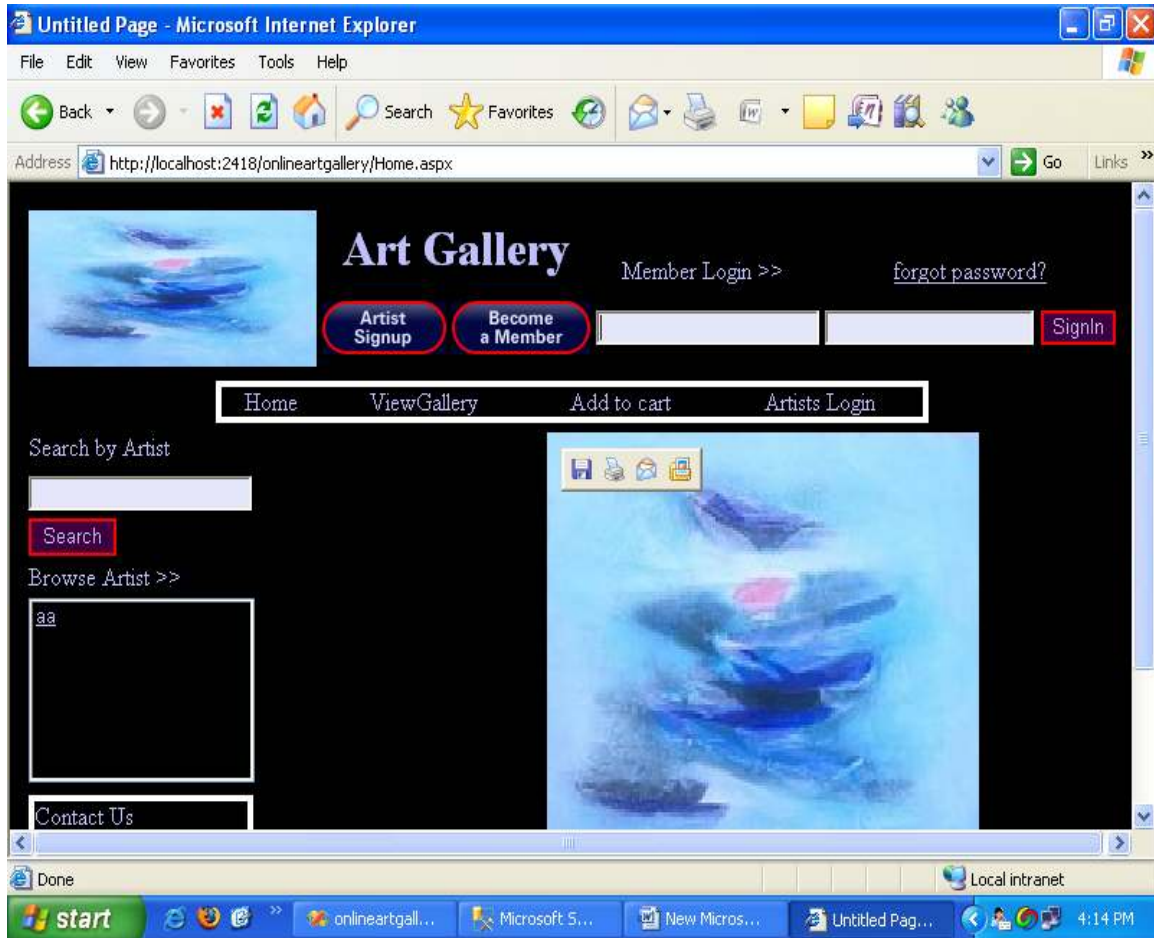


Sequence:

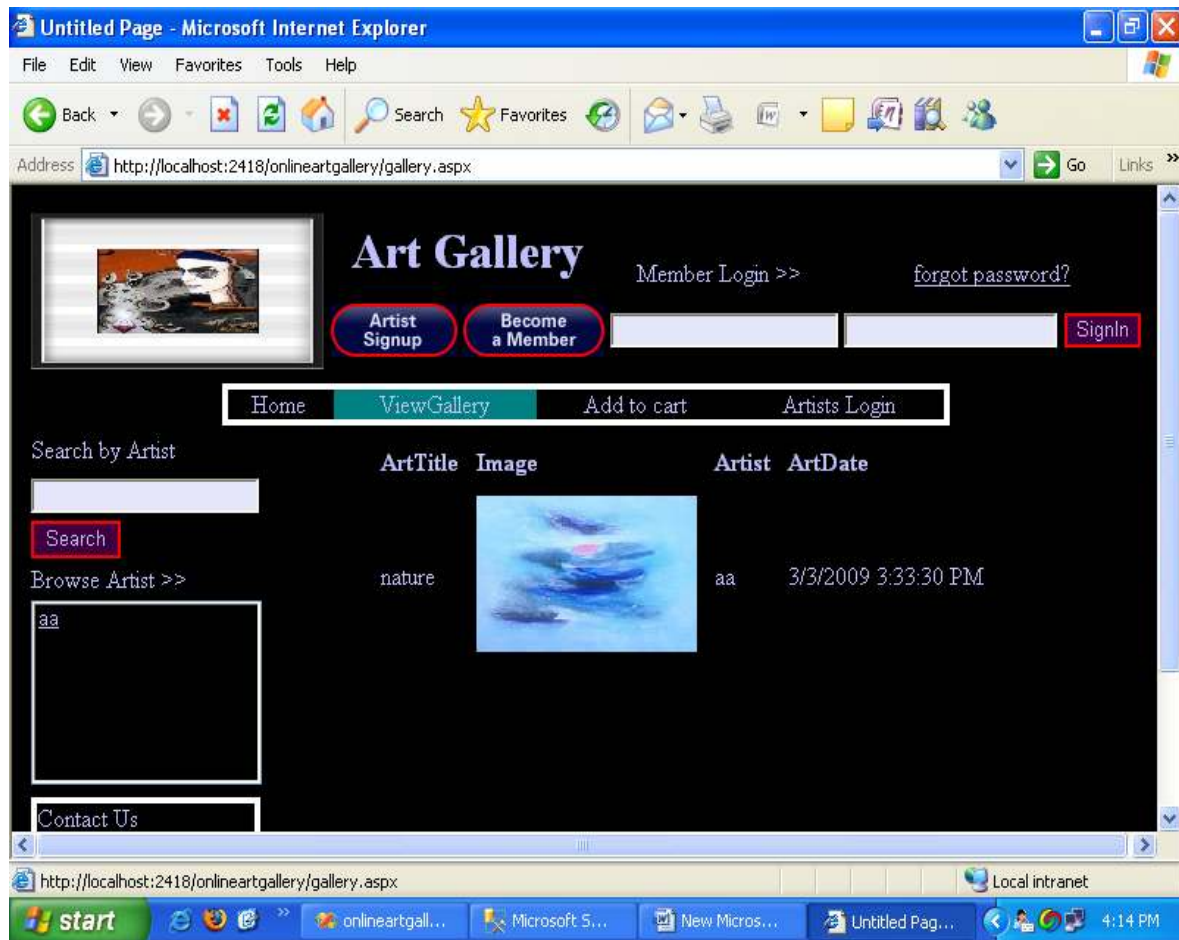


5. FORMS & REPORTS

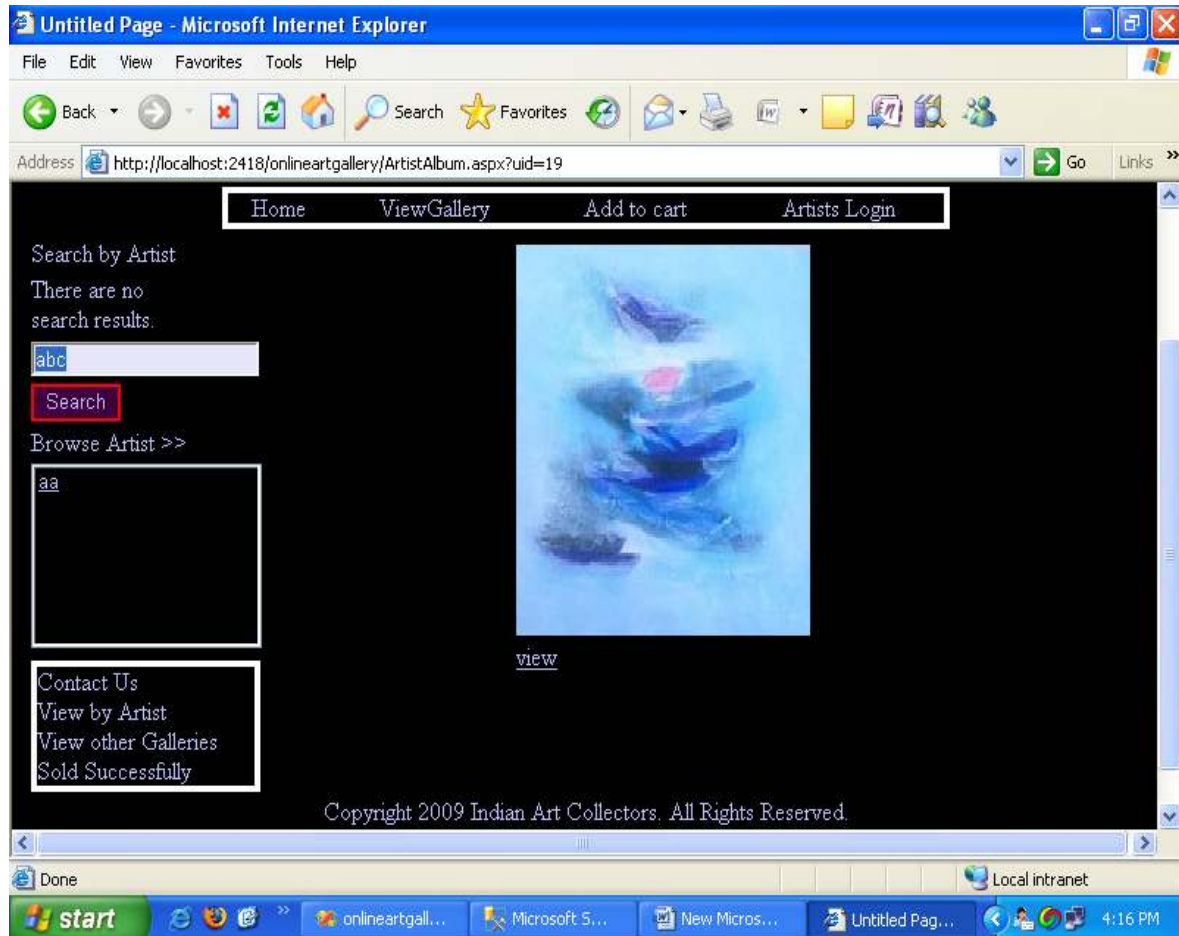
Screen shot for global users
This is home page



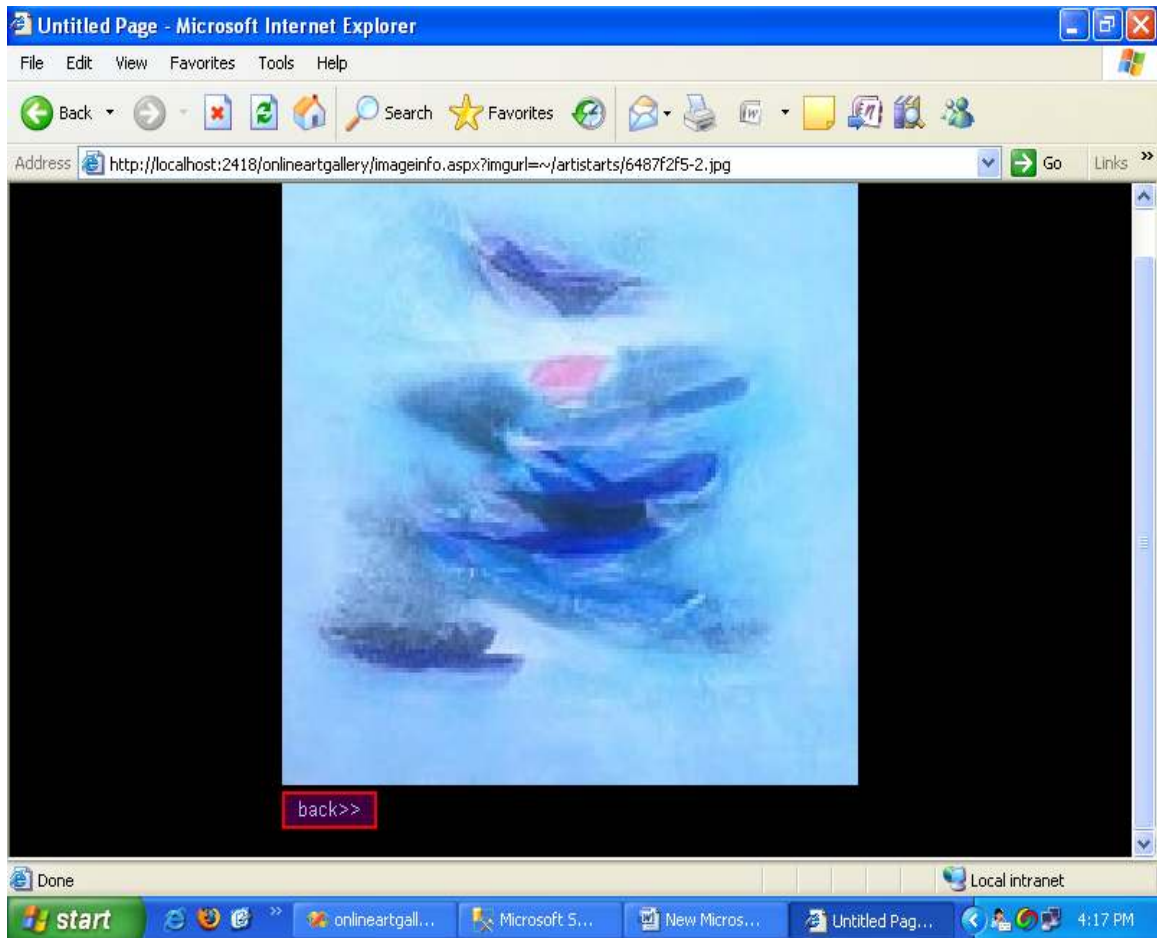
View browser page



Search for artists



View individual arts



Artist or member signup(registration)

Untitled Page - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Refresh Print Mail New Window New Tab

Address http://localhost:2418/onlineartgallery/registration.aspx Go Links >>

Signup a member

Home ViewGallery Add to cart Artists Login

Search by Artist

Search

Browse Artist >>

aa

Contact Us

View by Artist

View other Galleries

Sold Successfully

Register as a Artist>>

* All are mandatory>>

First Name : Last Name :

Gender : Email Address :

City : State :

Country : Mobile no :

Login Values >>>

User Name :

Password :

Enter Security text : 80C7B9

Submit>>

Local intranet

start onlineartgall... Microsoft S... New Micros... Untitled Pag... 4:18 PM


Member login

Untitled Page - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites

Address <http://localhost:2418/onlineartgallery/registration.aspx> Go Links



Art Gallery

Member Login >> [forgot password?](#)

[Artist Signup](#) [Become a Member](#) [SignIn](#)

[Home](#) [ViewGallery](#) [Add to cart](#) [Artists Login](#)

Search by Artist

[Search](#)

Browse Artist >>

aa

[Contact Us](#)

Register as a Artist>>

* All are mandatory>>

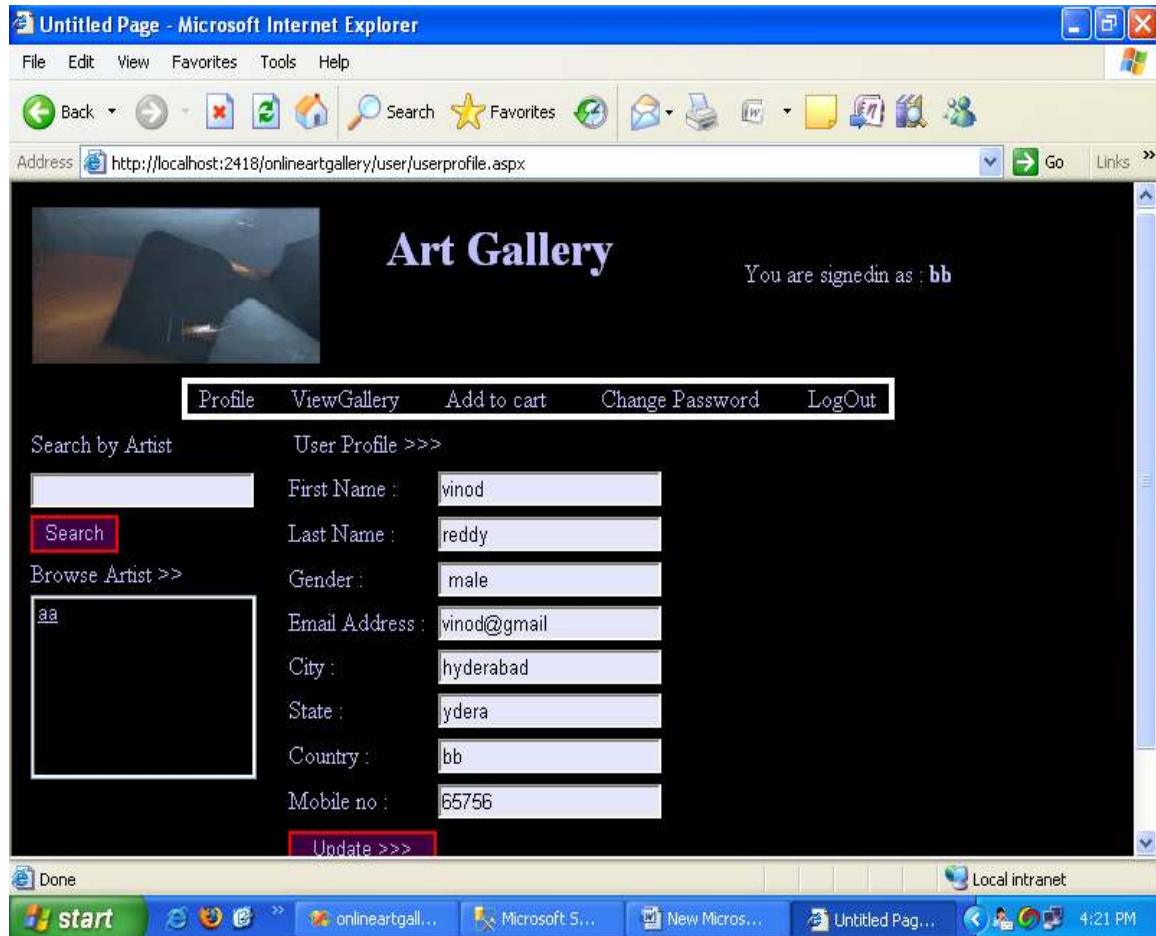
First Name :	<input type="text"/>	Last Name :	<input type="text"/>
Gender :	<input type="text"/>	Email Address :	<input type="text"/>
City :	<input type="text"/>	State :	<input type="text"/>
Country :	<input type="text"/>	Mobile no :	<input type="text"/>

Login Values >>>

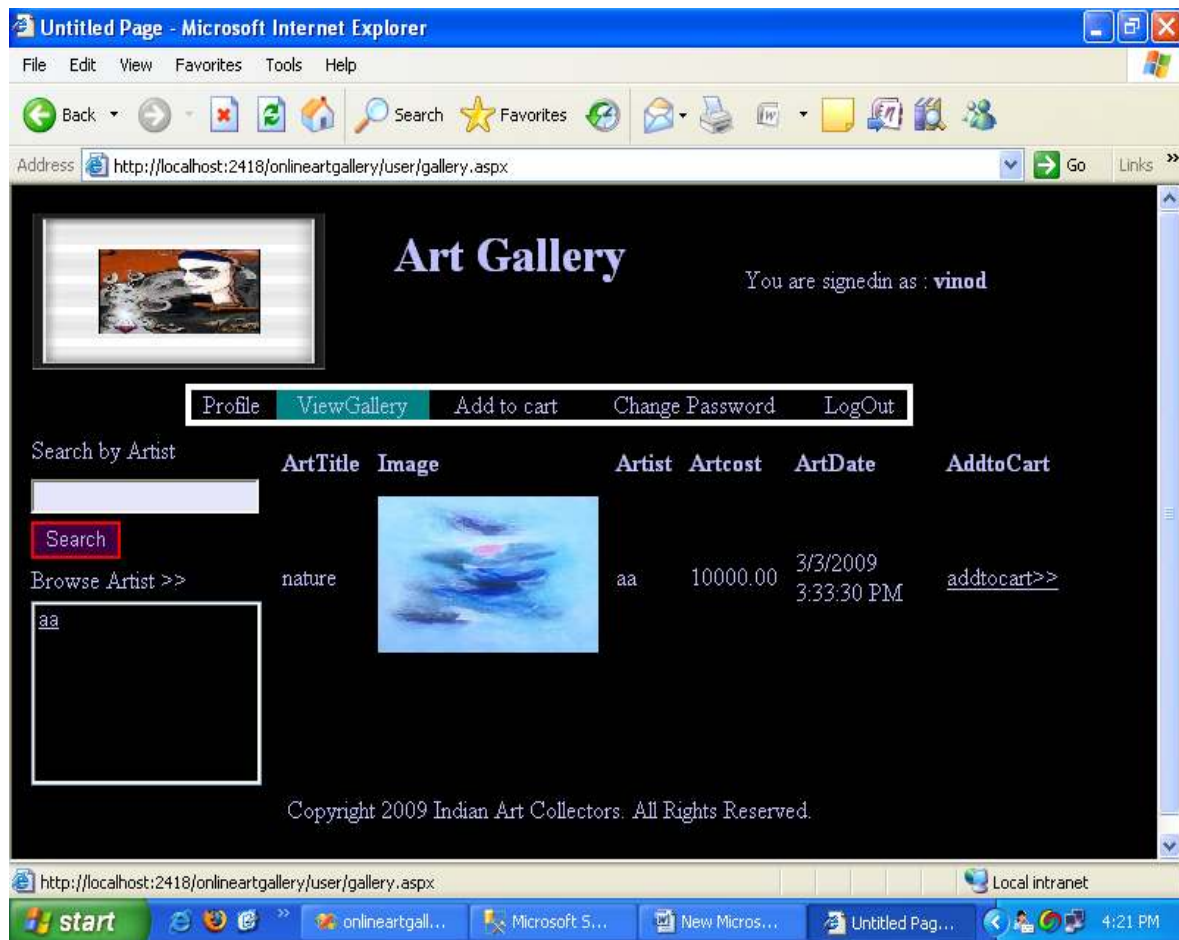
User Name :

Password :

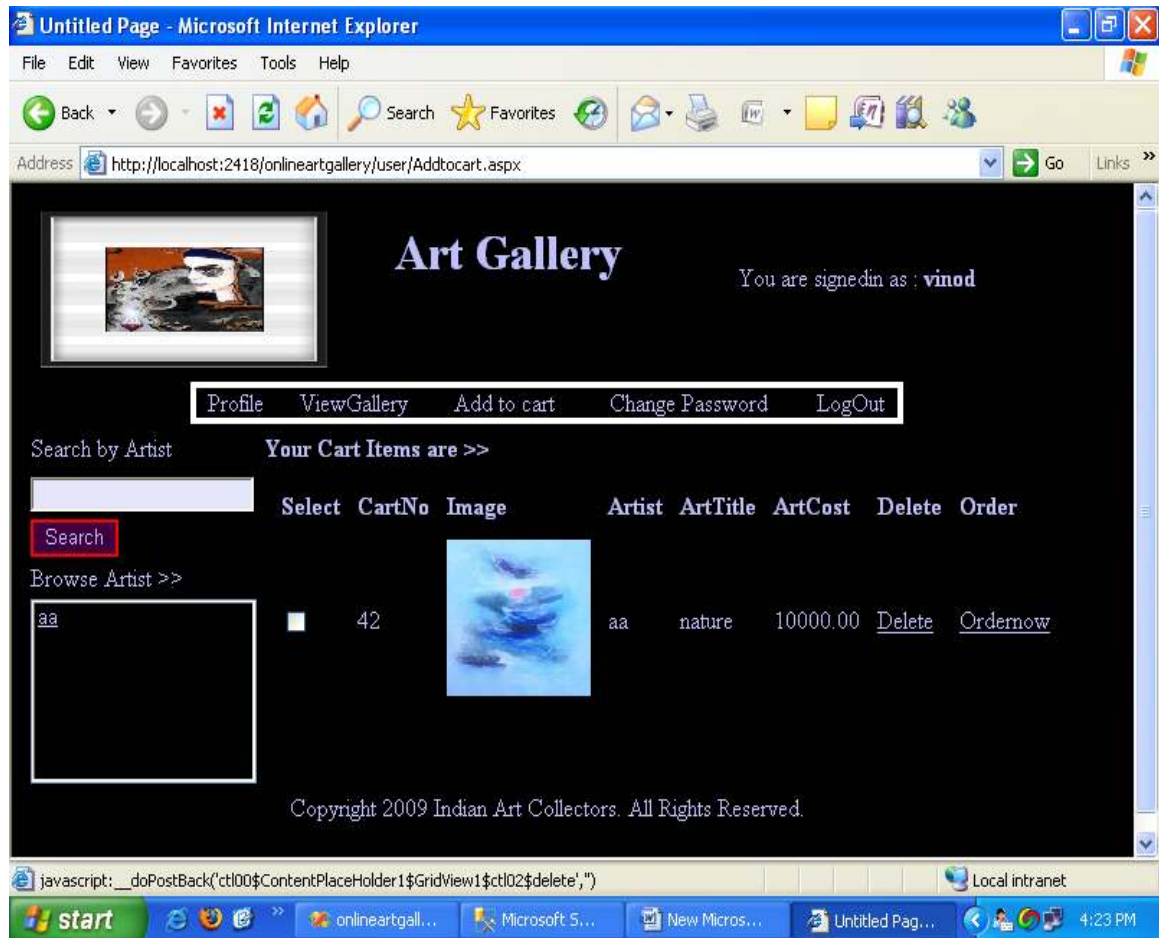
Updating member profile



Viewing gallery after login into the



Selecting individual art to cart details



Shipping and billing address for order

Untitled Page - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Reload Home Search Favorites RSS Print Mail W Word Excel PowerPoint Outlook

Address <http://localhost:2418/onlineartgallery/user/billingandshopping.aspx?artid=42> Go Links >>

Profile ViewGallery Add to cart Change Password LogOut

Search by Artist

Billing Address >>> Shipping Address >>>

☒ Same address

Search

Browse Artist >>

aa

First Name :	vinod	First Name :	vinod
Last Name :	reddy	Last Name :	reddy
Gender :	male	Gender :	male
E-mail :	vinod@gmail	E-mail :	vinod@gmail
City :	hyderabad	City :	hyderabad
State :	ydera	State :	ydera
Country :	bb	Country :	bb
Mobile no :	65756	Mobile no :	65756

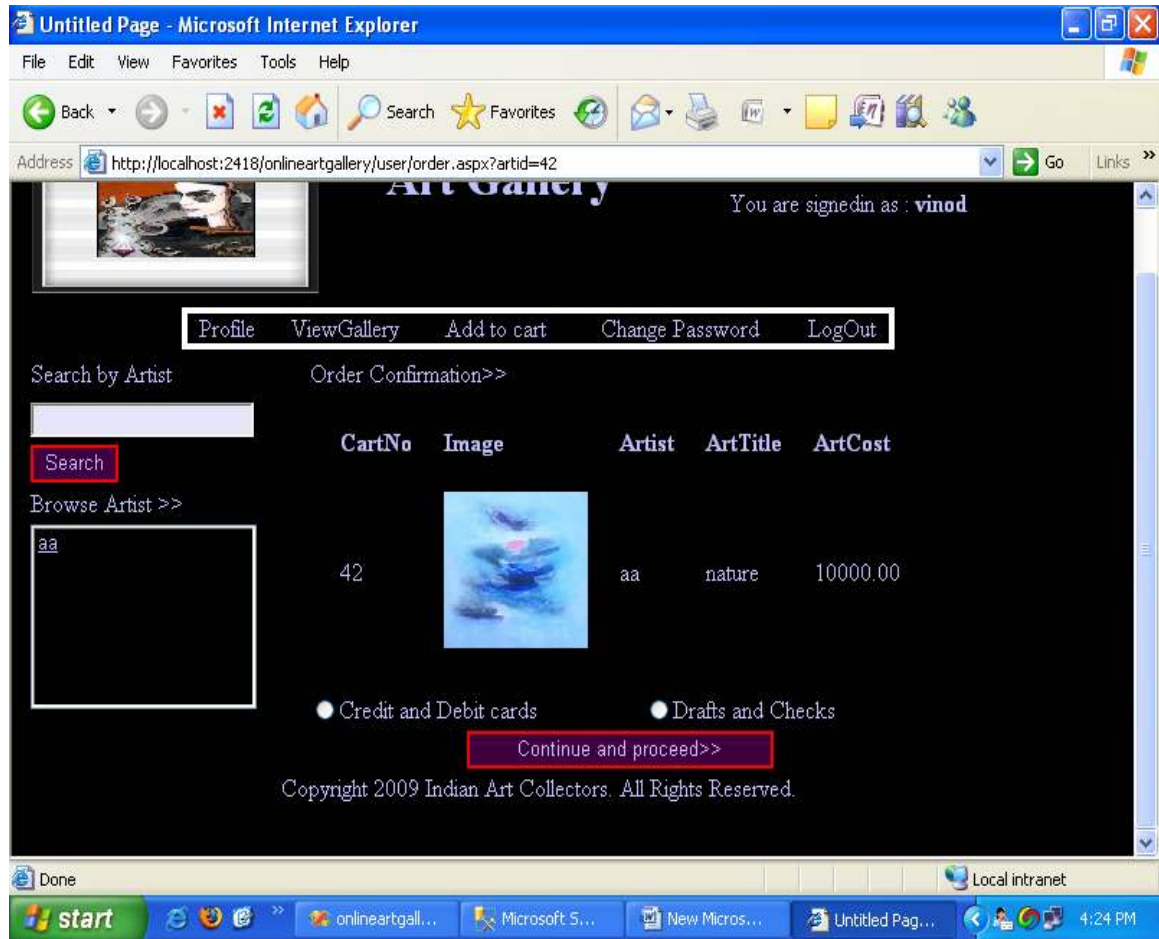
Proceed & Continue>>

Copyright 2009 Indian Art Collectors. All Rights Reserved.

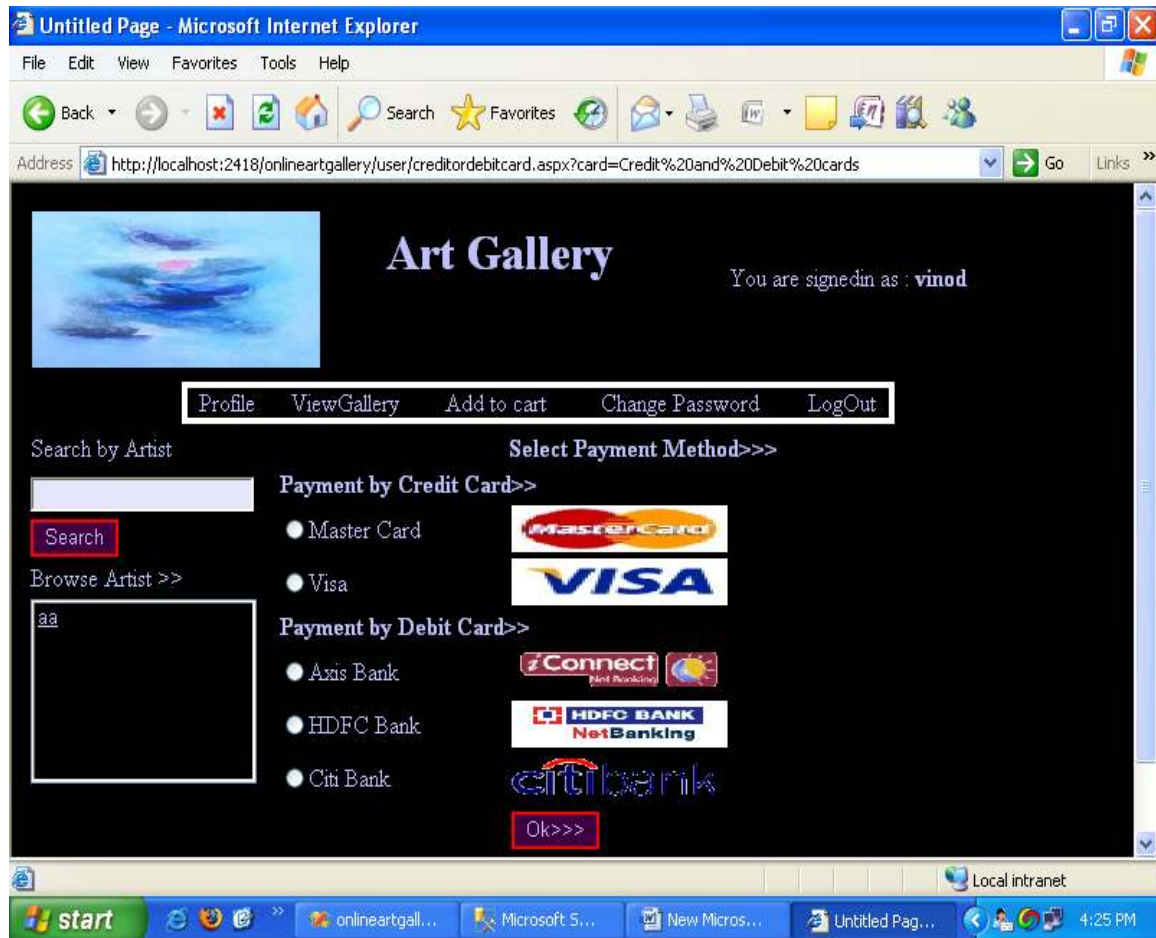
Done Local intranet

start onlineartgall... Microsoft S... New Micros... Untitled Pag... 4:24 PM

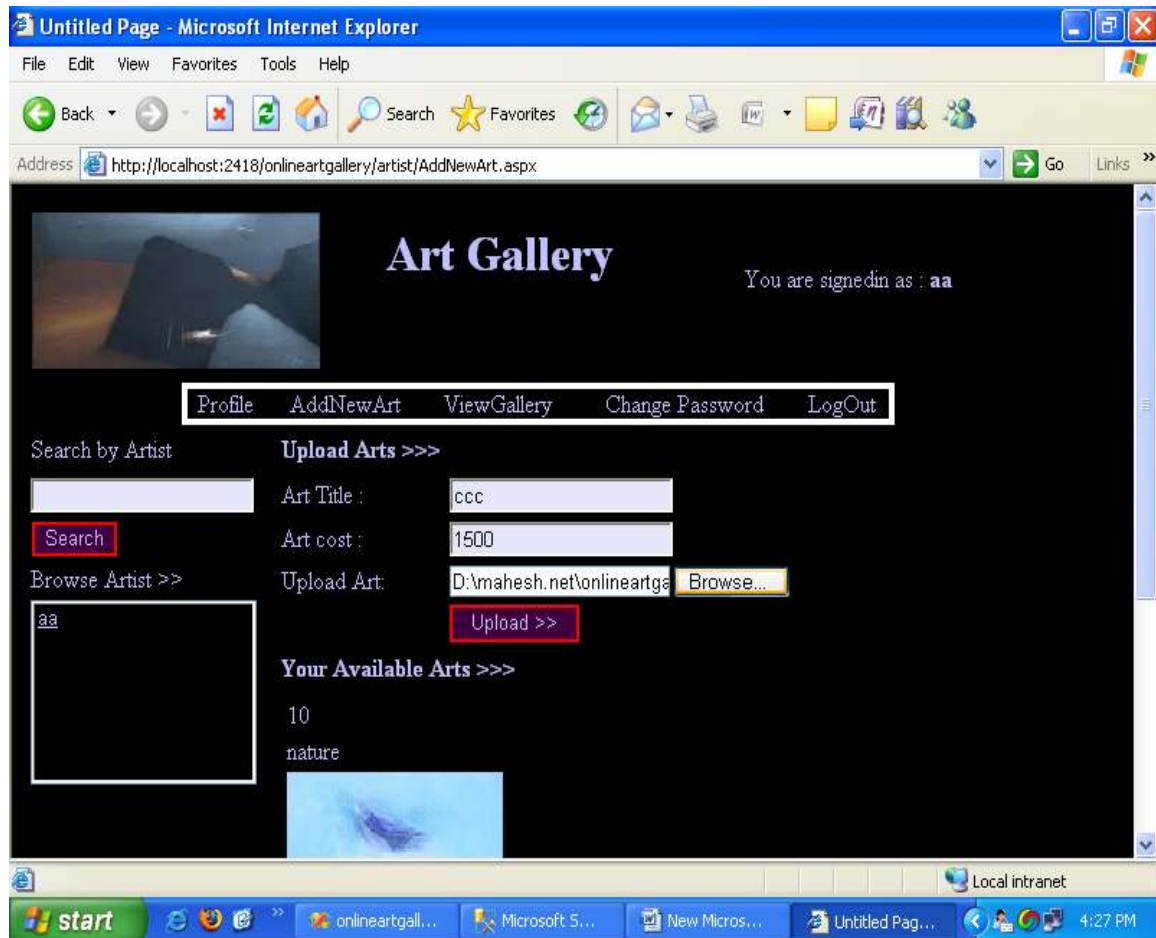
Payment details for selection



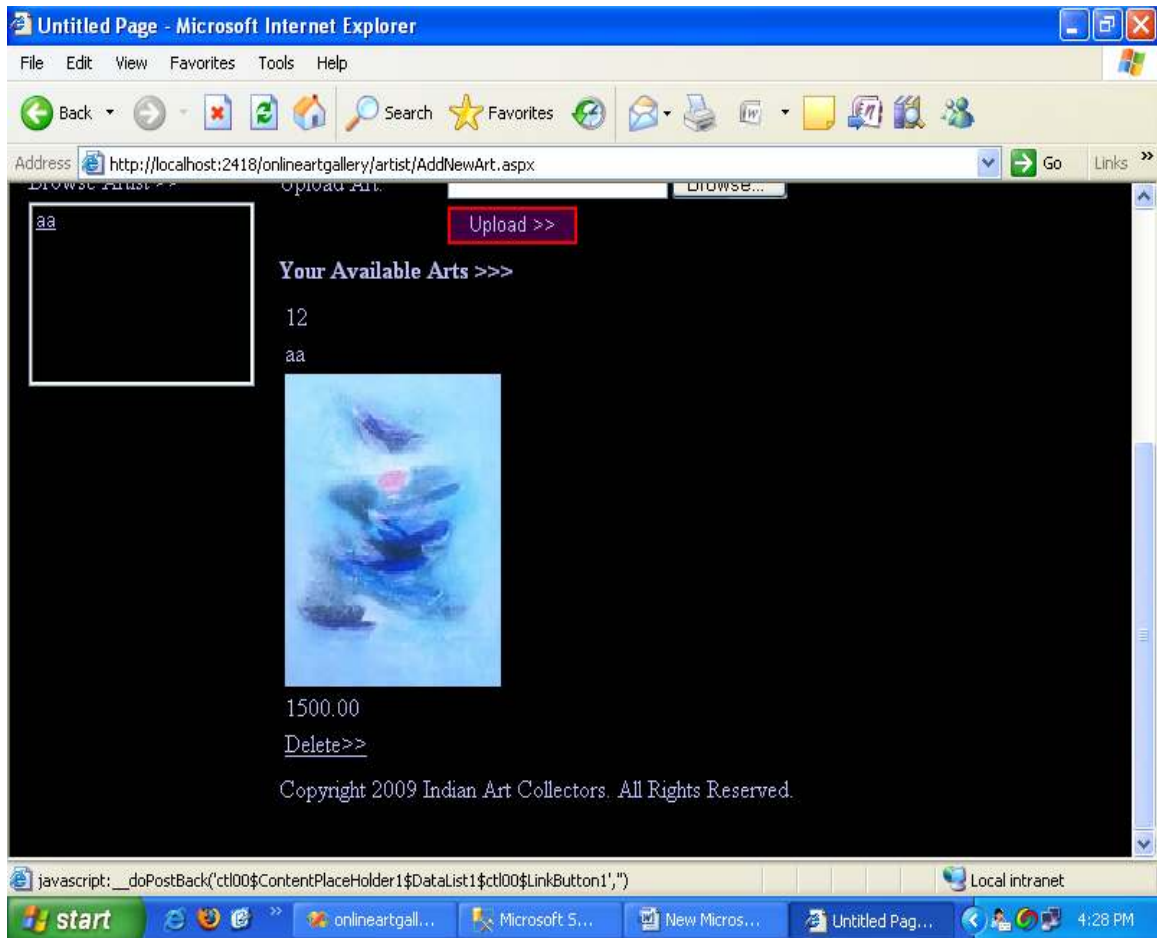
Payment mode for Billing details



Artist adding a art



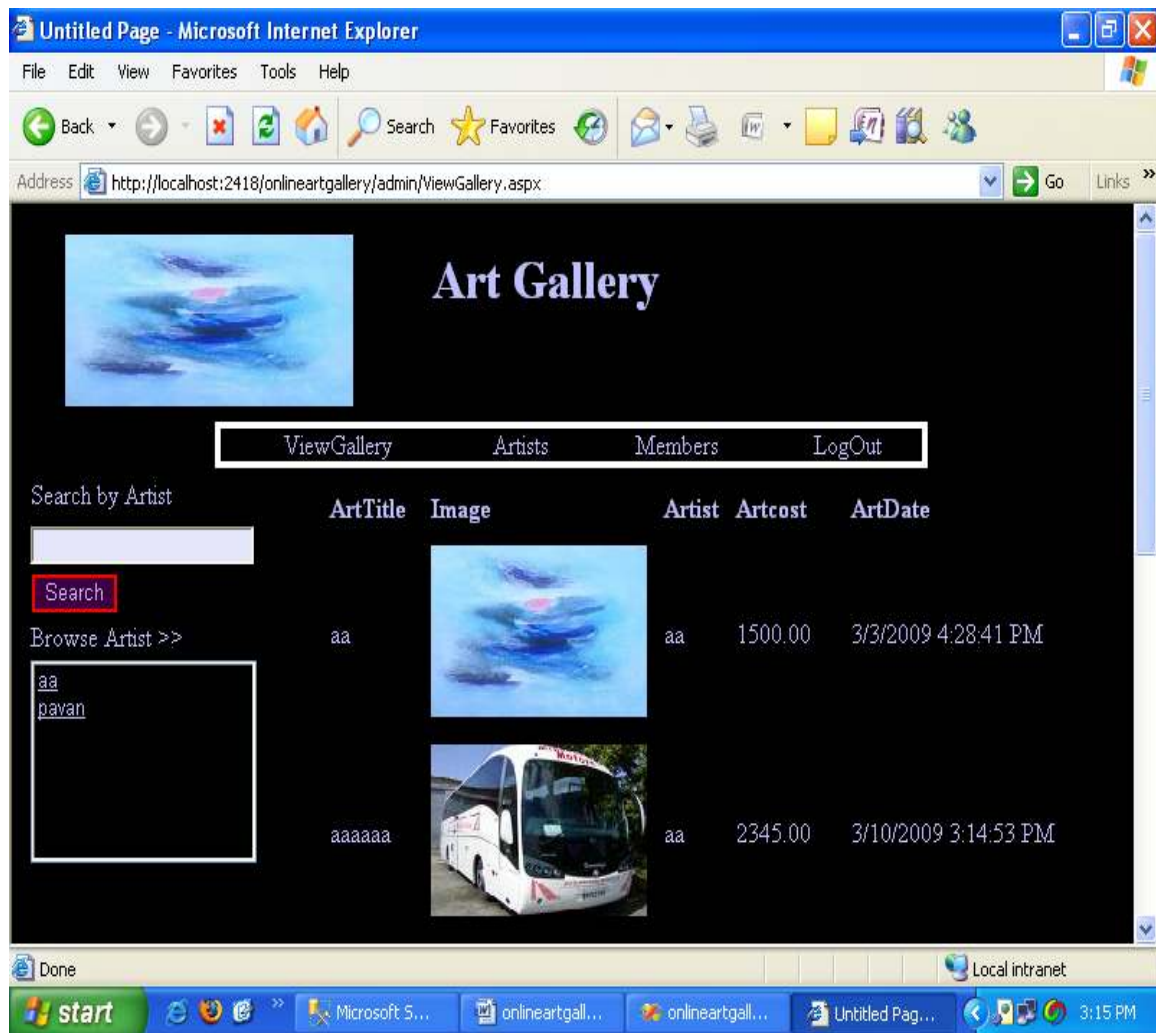
Artist deleting a art



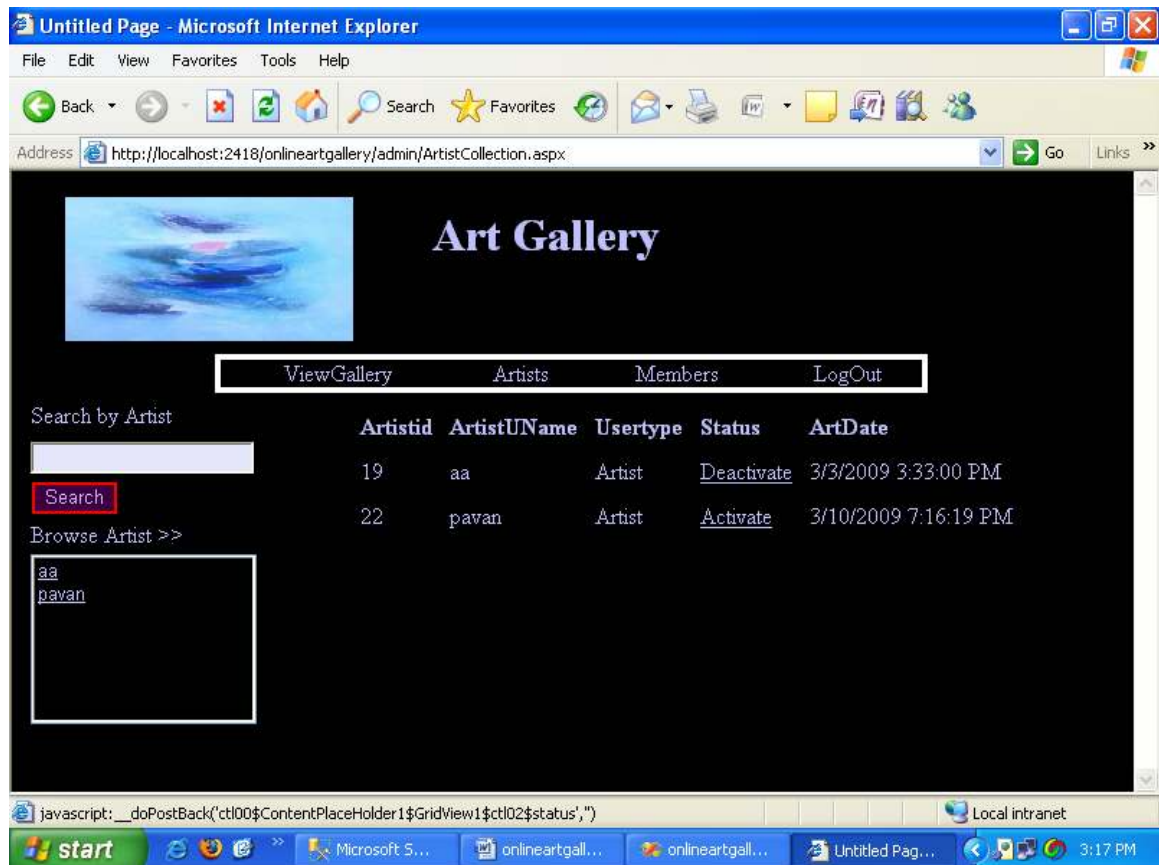
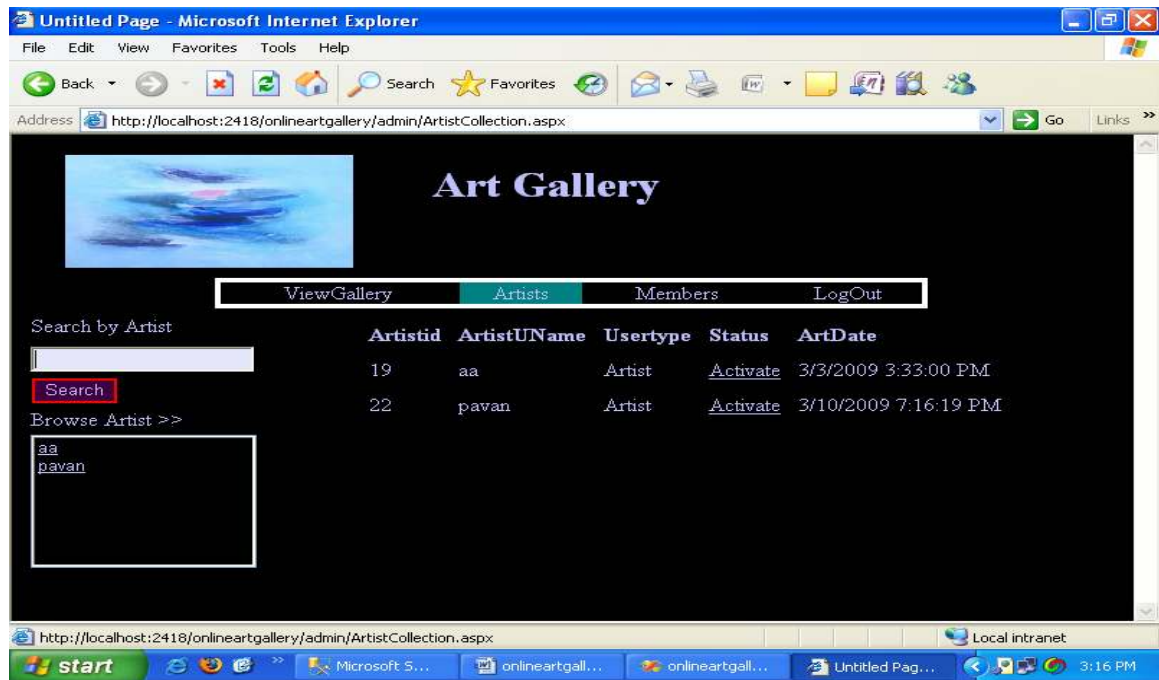
Admin Pages:

For member and artist activation and deactivation:

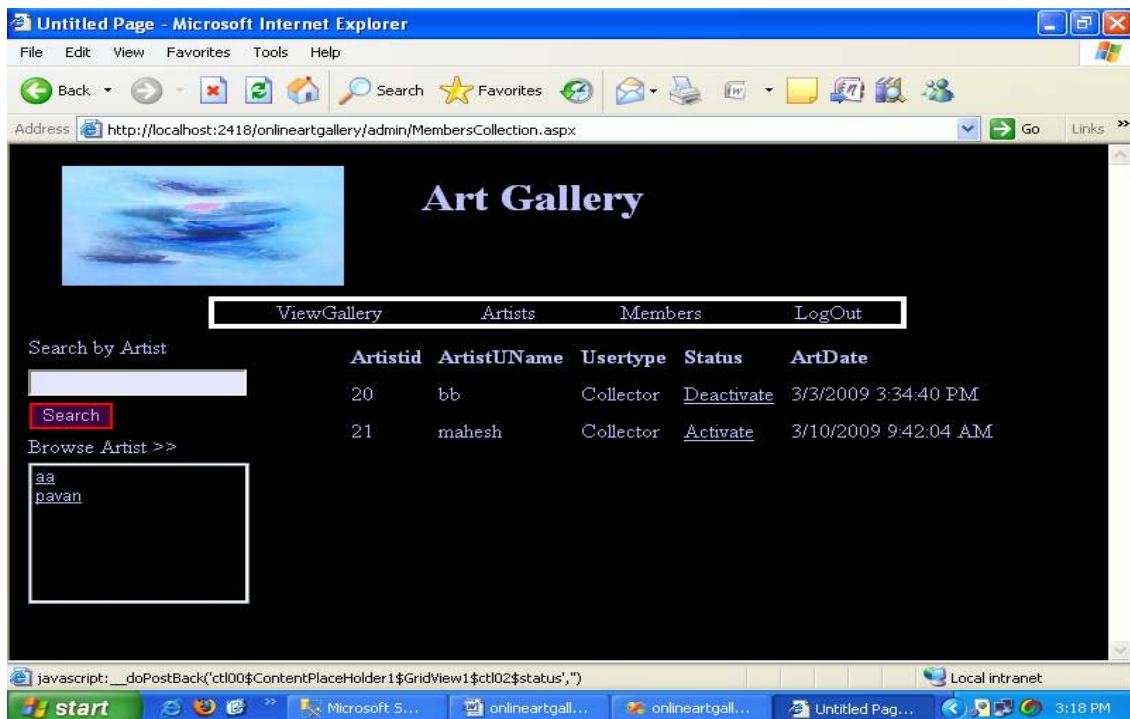
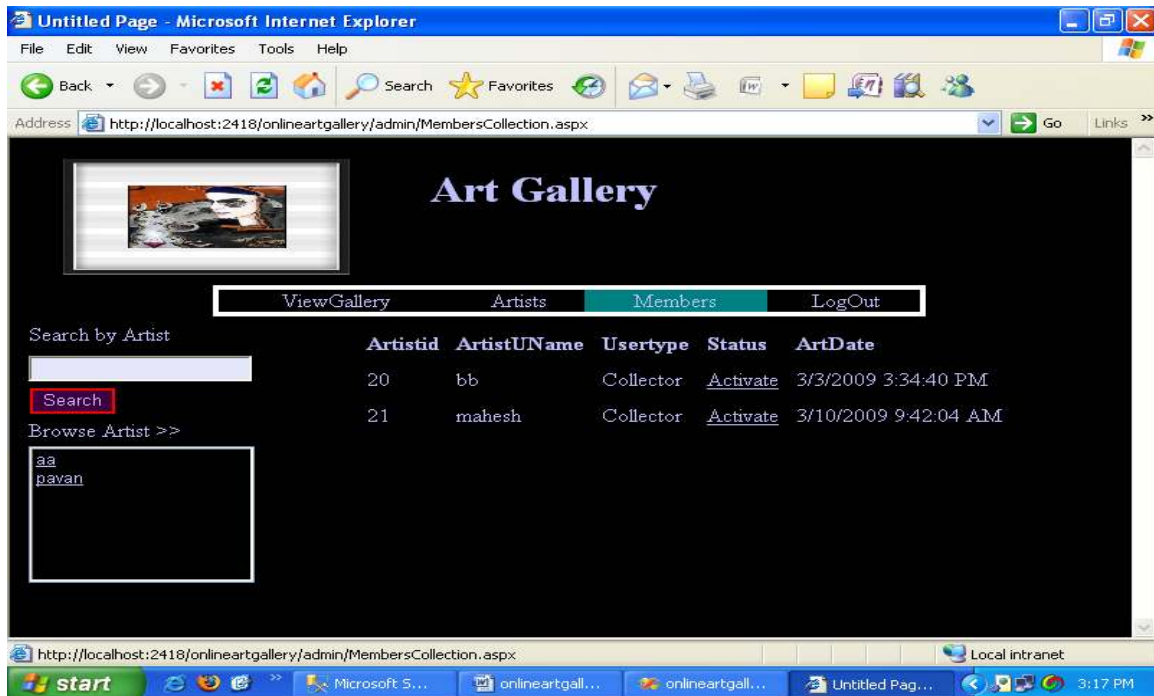
View gallery:



Artists activate and deactivation:



Members activate and deactivation:



6.1 Testing

Testing is the process of detecting errors. Testing performs a very critical role for quality assurance and for ensuring the reliability of software. The results of testing are used later on during maintenance also.

6.1.1 Objectives:

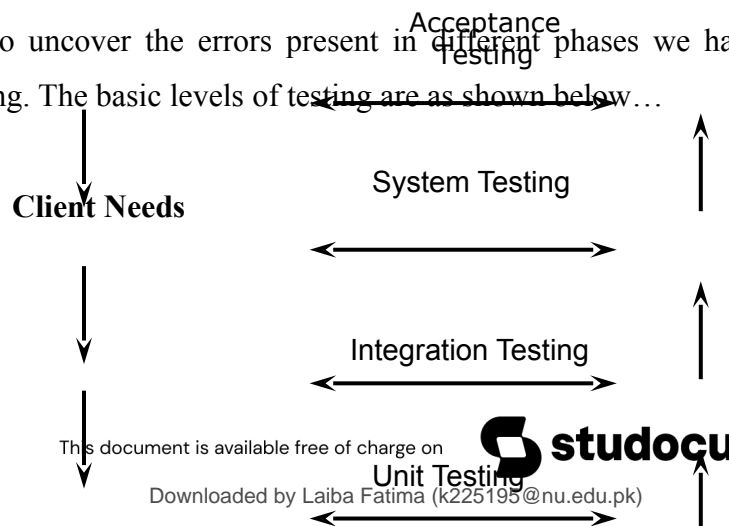
- To identify Testing Requirements (Scope):
 - to identify the software to be tested
 - to identify the testing objectives
 - to identify the test phases (testing coverage) within the testing life cycle that is required

- To identify Testing Approach:
 - to identify the methods and testing tools required
 - to identify any client assumptions/dependencies/limitations

- To identify Testing Tasks and Deliverables:
 - to identify the activities to perform within each testing phase.
 - to identify the external (client) deliverable document.
 - to identify the table of contents for each deliverable
 - to identify the internal deliverable documents
 - to identify document deliverable reviewer
- To compile Testing Estimates:
 - to identify the budgetary estimate for each identified phase of software testing.
- To determine Testing Schedules:
 - to identify the start and end date for each phase of software testing
 - to identify all testing phase overlaps in the schedule
 - to identify delivery dates for all document deliverables
- To determine Test Environments:
 - to identify the software/hardware requirements for each test phase
 - to identify the number of test environments
- To identify Test Team Roles and Responsibilities:
 - to identify the overall testing management responsibility and for each test phase
 - to identify client roles and responsibilities

6.2 Levels of Testing

In order to uncover the errors present in different phases we have the concept of levels of testing. The basic levels of testing are as shown below...



Requirements

Design

Code

Fig: 6.2.1 Levels of Testing

6.2.1 Code Testing:

This strategy examines the logic of the program. To follow this method we developed some test data that resulted in executing every instruction in the program and module i.e. every path is tested. Systems are not designed as entire nor are they tested as single systems. To ensure that the coding is perfect two types of testing is performed or for that matter is performed or that matter is performed or for that matter is performed on all systems.

Here the Aptitude Quest source code is developed in java and servlets which is tested by JDK.

6.2.2 Unit Testing:

The goal of unit testing is to assure that all functions and features of a single compilable unit of code perform as specified in the Design Specification.

A unit test covers the testing of a software unit, or a group of closely related units, as a single entity. Unit testing is performed in isolation, using test drivers to simulate higher level units, and/or stubs to simulate lower level units.

Unit Testing Procedures consist of:

- Creating a Unit Test Plan
- Creating test data
- Conducting tests according to the Unit Test Plan
- Reporting and reviewing the results of the test

These procedures are performed by the team member responsible for programming and testing of the unit.

A Unit Test Plan is a set of test cases arranged in the sequence of chronological execution. The Unit Test Plan is created before the programming of the unit is started, and the test cases should cover the functional, input, output, and function interaction of the unit.

The modules of **Aptitude Quest** i.e. User and Administration modules are tested whether it can interact with database and server.

6.2.3 Integration Testing:

The goal of integration testing is to ensure that all interacting subsystems in a system interface correctly with one another to produce the desired results. Furthermore, in trying to attain this goal, integration tests will ensure that the introduction of one or more subsystems into the system does not have an adverse affect on existing functionality.

An integration test covers the testing of interface points between subsystems. Integration testing is performed once unit testing has been completed for all units contained in the subsystems being tested.

Integration Testing Procedures consist of:

- Creating and integration test plan
- Creating test data
- Conducting tests according to the integration test plan
- Reporting and reviewing the results of the test

During this phase, the interaction between subsystems is tested. This includes interfaces through Inter Process Communications (IPC) and files. This phase is performed by an independent test team. This team prepares and executes integration tests, generates problem reports and is responsible for passing the integrated system on to the System Test Team for system testing. The Integration Test team then enters a support mode in which it will test problem reports generated by the System Test team before forwarding code fixes to the System Testing environment.

This phase is sometimes combined with the system test phase as per the client's request.

Each and every modules of **Aptitude Quest** are integrated and checked whether they interact with each other or not.

6.2.4 System Testing:

The goal of System Testing is to ensure that the system performs as per the functional requirements specified by client.

A system test covers the testing of functions within the system. System testing is performed once integration testing has been completed. System Testing procedures consist of:

- Creating Test Plans
- Creating test data
- Conducting tests according to the System Test Plan
- Reporting and reviewing the results of the test

Features to be tested during System Testing are:

- Functional Requirements
- Depending on the project, any regression tests deemed necessary

The developed **Aptitude Quest** is entirely tested by means of system testing.

6.2.5 Acceptance Testing:

Acceptance Test is performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here is focused on external behavior of the system; the internal logic of program is not emphasized. In this project 'VOIP' I have collected some data and tested whether project is working correctly or not.

Test cases should be selected so that the largest number of attributes of an equivalence class is exercised at once. The testing phase is an important part of software development. It is the process of finding errors and missing operations and also a complete verification to determine whether the objectives are met and the user requirements are satisfied.

CONCLUSION

ONLINE ART GALLERY is a application software and it is very helpful for the art lovers and others who wants to know the addresses where this kind of arts will we sold.

This application helps the end-users to search their arts and paintings and they can place order for the selected pieces. The end-user can also get the information about the art exhibition and the respective address, so, that they can visit to those exhibitions.

Art Gallery brings you an opportunity to view online art exhibitions at our Online Art Gallery we bring you details of all art exhibitions held in the past and the forthcoming show. The Online Art Gallery is updated daily, so the user can view and buy the latest collection of contemporary art online from any where in the world. You can view and buy the latest Indian contemporary art collection available at their exhibitions and also at their online gallery.

FUTURE SCOPE :

- This project “Online Art Gallery ” gives sopes for the further enhancement.
- The product can enter any user requirement.
- The application is currently stand alone and can be extended for the web.
- The product creation part is to done in a more elaborate manner.

BIBLIOGRAPHY

References and Resources:

- Learning C#, Liberty, 2002
- .NET framework essential – Thai, 2002
- Mysql cook book (cover Mysql 4.0) – Dubois, 2003
- Program ASP>NET AJAX: Build Rich, Web 2.0 – Style UI with ASP.NET AJAX – Wenz, 2007
- Program SQL server 2005 – Wildermuth, 2006.