CHAPTER 3

3.1 SYSTEM ARCHITECTURE

The system uses 3-tier architecture which is a powerful mechanism of handling all the resources of the system efficiently in order to obtain the desired goal successfully.

Uses of 3-Tier Architecture

1. To make application more understandable.

2. Easy to maintain, easy to modify application and we can maintain good look of architecture.

3. If we use this 3-Tier application we can maintain our application in consistency manner.

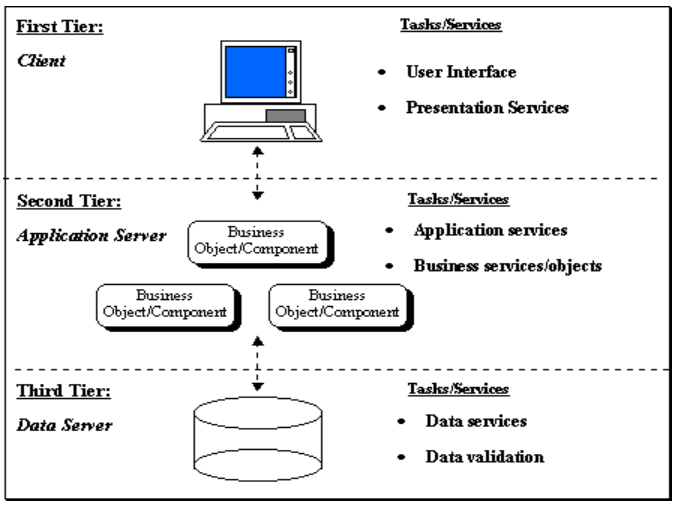

Fig 3.1 3-Tier Architecture

Basically 3-Tier architecture contains 3 layers

1. Application Layer or Presentation Layer

2. Business Access Layer(BAL) or Business Logic Layer(BLL)

3. Data Access Layer(DAL)

Application Layer or Presentation Layer

Presentation layer contains UI part of our application i.e., our aspx pages or input is taken from the user. This layer is mainly used for design purpose and get or set the data back and forth. Here we can include form used to take inputs from user. These forms act as an interface for user to interact with the system.

Business Logic Layer

This layer contains our business logic, calculations related with the data like insert data, retrieve data and validating the data. This acts as an interface between Application layer and Data Access Layer. This layer includes the business logic. In order to expand the modularity, this layer can be extended into multiple layers.

We use 2 types of classes here:

- Model Classes: Model classes define the basic variables and their data types.
- Logic Classes: Logic classes define the operations that will be performed by that entity. There will be one logic class of each entity in the system.

Data Access Layer

Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data. Data Access Layer also contains class file. It contains a common class with implementation methods for DQL and DML Queries.

The class we have used is DBUtility.cs. It is a C# class file. The sample code for the class is as shown:

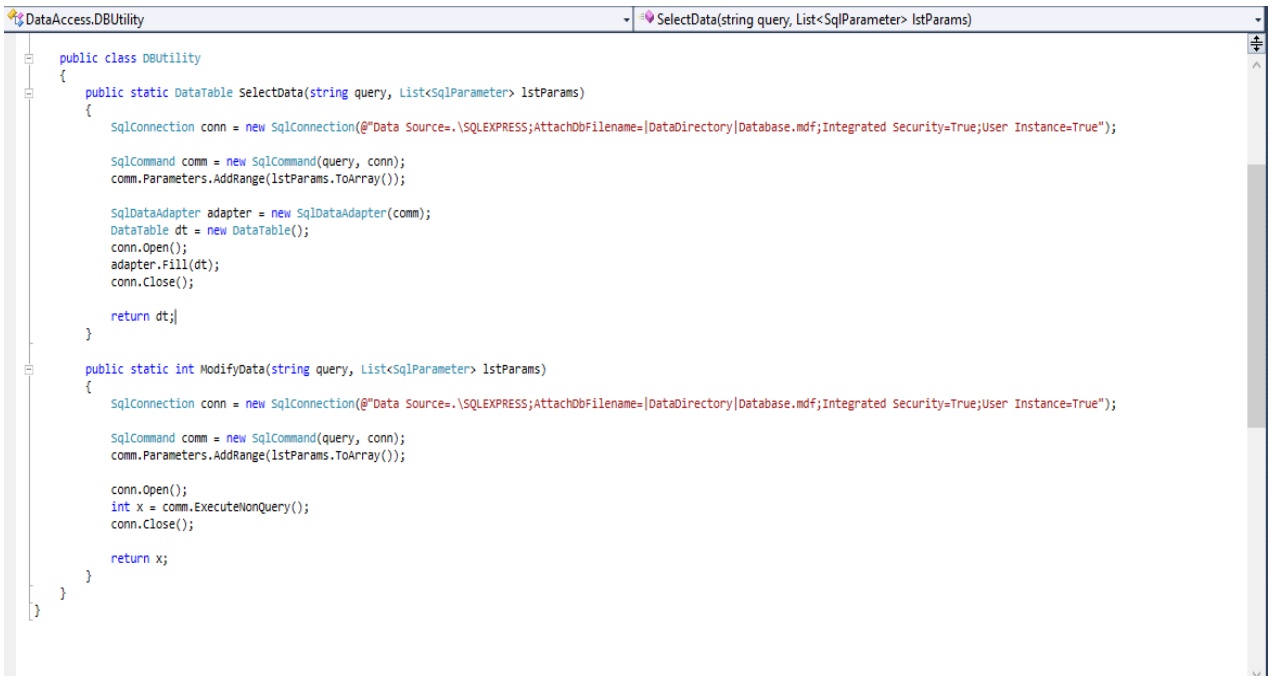

Fig 3.2 DBUtility.cs

All these 3 layers work in accordance to achieve the goal of efficient working of the application. They act hand in hand and take the decisions. The three tiers cannot work individually. They pass the data into each other and wait for the reply.

In a distributed environment like the system proposed by us, the 3-tier system provides prompt response to each user request and a flawless communication between layers makes it clean processing environment.

3.2 CODING STANDARDS

Use Pascal casing for Class names

public class **HelloWorld**

```
{
    ...
}
```

Use Camel casing for Method names

void **SayHello**(string name)

```
{
    ...
}
```

Use Camel casing for variables and method parameters

int **totalCount** = 0;

void SayHello(string name)

{

string **fullMessage** = "Hello " + name;

...

}

Use the prefix “I” with Camel Casing for interfaces (Example: **IEntity**)

Do not use Hungarian notation to name variables.

In earlier days most of the programmers liked it - having the data type as a prefix for the variable name and using m_ as prefix for member variables. Eg:

string m_sName;

int nAge;

However, in .NET coding standards, this is not recommended. Usage of data type and m_ to represent member variables should not be used. All variables should use camel casing.

Use Meaningful, descriptive words to name variables. Do not use abbreviations.

Good:

string address

int salary

Not Good:

string nam

string addr

int sal

Do not use single character variable names like i, n, s etc. Use names like index, temp

One exception in this case would be variables used for iterations in loops:

```
for ( int i = 0; i < count; i++ )
    {
    ...
    }
```

If the variable is used only as a counter for iteration and is not used anywhere else in the loop, many people still like to use a single char variable (i) instead of inventing a different suitable name.

Do not use underscores (_) for local variable names.

All member variables must be prefixed with underscore (_) so that they can be identified from other local variables.

Do not use variable names that resemble keywords.

Prefix boolean variables, properties and methods with "is" or similar prefixes.

Ex: private bool isFinished

File name should match with class name.

Use Pascal Case for file names.

Use appropriate prefix for the UI elements so that you can identify them from the rest of the variables.

Table 3.1. UI element prefix

Control	**Prefix**
Label	Lbl
TextBox	txt
DataGrid	dtg
Button	btn
ImageButton	imb
Hyperlink	hlk
DropDownList	ddl
ListBox	lst
DataList	dtl
Repeater	rep
Checkbox	chk
CheckBoxList	cbl
RadioButton	rdo
RadioButtonList	rbl
Image	img
Panel	pnl

PlaceHolder	phd
Table	tbl
Validators	val

Indentation and Spacing

Use TAB for indentation. Do not use SPACES. Define the Tab size as 4.

Comments should be in the same level as the code (use the same level of indentation).

Good:

```
// Format a message and display
    string fullMessage = "Hello " + name;
    DateTime currentTime = DateTime.Now;
    string message = fullMessage + ", the time is : " +
    currentTime.ToShortTimeString();
    MessageBox.Show ( message );
```

Not Good:

```
// Format a message and display
        string fullMessage = "Hello " + name;
        DateTime currentTime = DateTime.Now;
        string message = fullMessage + ", the time is : " +
        currentTime.ToShortTimeString();
        MessageBox.Show ( message );
```

Curly braces ({}) should be in the same level as the code outside the braces.

There should be one and only one single blank line between each method inside the class.

The curly braces should be on a separate line and not in the same line as if, for etc.

Good:

```
if ( ... )
```

```
{

        // Do something
}
```

Not Good:

```
if ( ... ) {

        // Do something
}
```

Use a single space before and after each operator and brackets.

Good:

```
if ( showResult == true )
{
for ( int i = 0; i < 10; i++ )
        {

                //
        }
}
```

Not Good:

```
if(showResult==true)

{
        for(int     i= 0;i<10;i++)
        {
                //
        }
}
```

Use #region to group related pieces of code together. If you use proper grouping using #region, the page should like this when all definitions are collapsed.

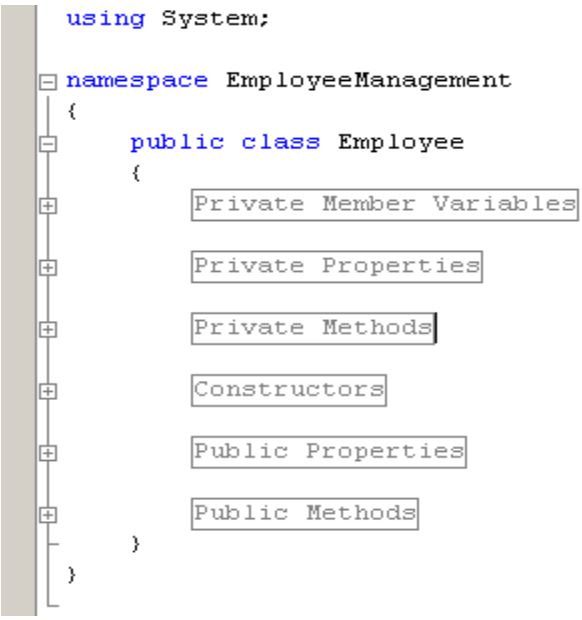

Fig 3.3 Coding Standards

ASP.NET standards:

Do not use session variables throughout the code. Use session variables only within the classes and expose methods to access the value stored in the session variables. A class can access the session using System.Web.HttpContext.Current.Session.

Do not store large objects in session. Storing large objects in session may consume lot of server memory depending on the number of users.

Always use style sheet to control the look and feel of the pages. Never specify font name and font size in any of the pages. Use appropriate style class. This will help you to change the UI of your application easily in future. Also, if you like to support customizing the UI for each customer, it is just a matter of developing another style sheet for them.

3.3 SCREENSHOTS

3.3.1 LOGIN SCREEN

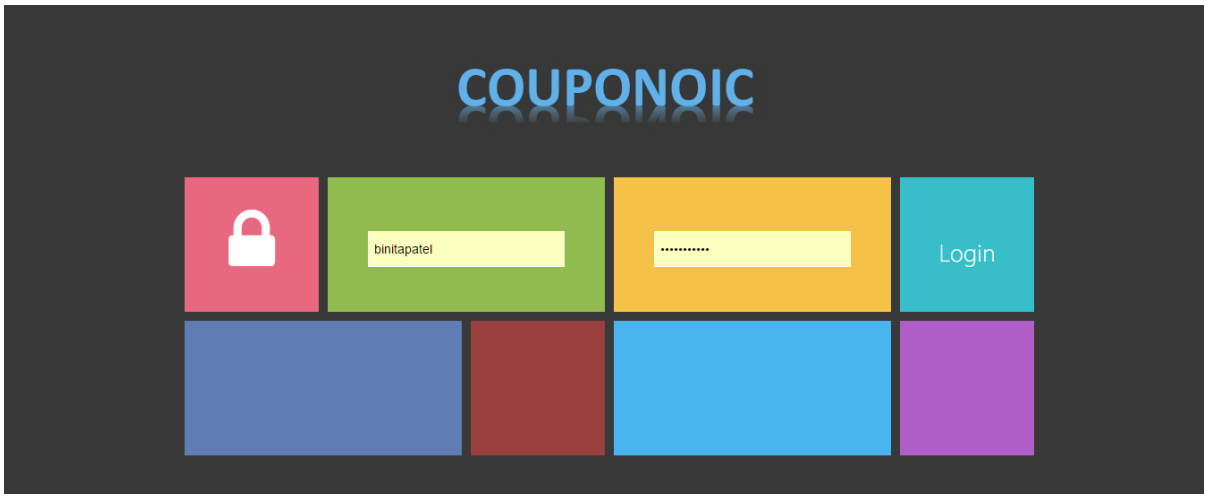

Fig. 3.4 Login Screen

3.3.2 CUSTOMER HOME

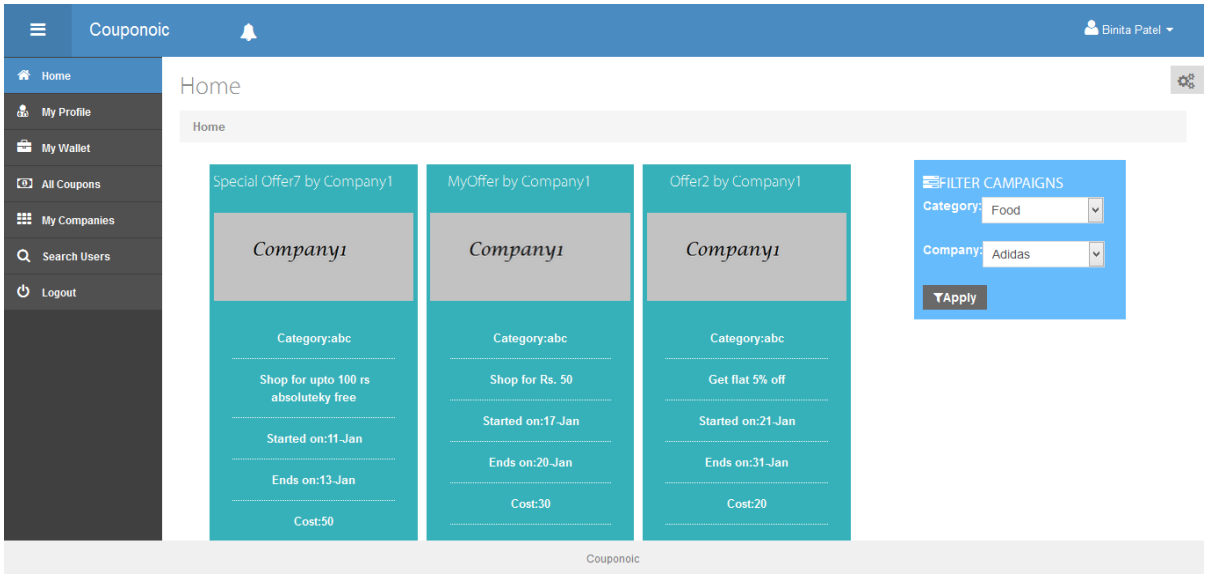

Fig. 3.5 Customer Home

3.3.3 VIEW AFFILIATES AND BUY COUPONS

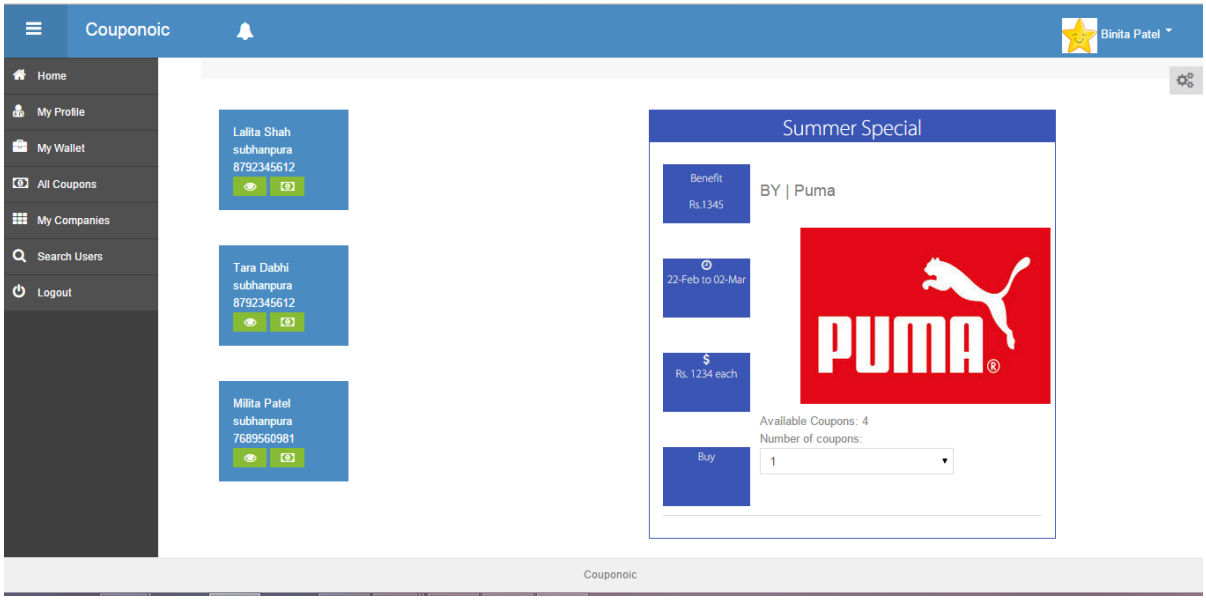

Fig. 3.6 Buy Coupons

3.3.5 CUSTOMER PROFILE

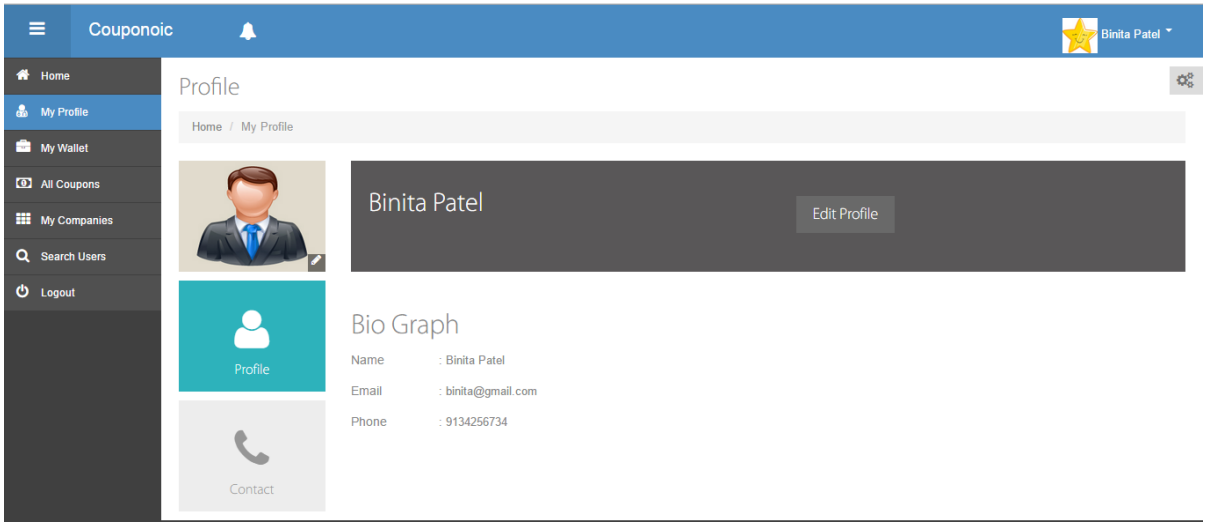

Fig. 3.7 Customer Profile

3.3.6 CUSTOMER WALLET

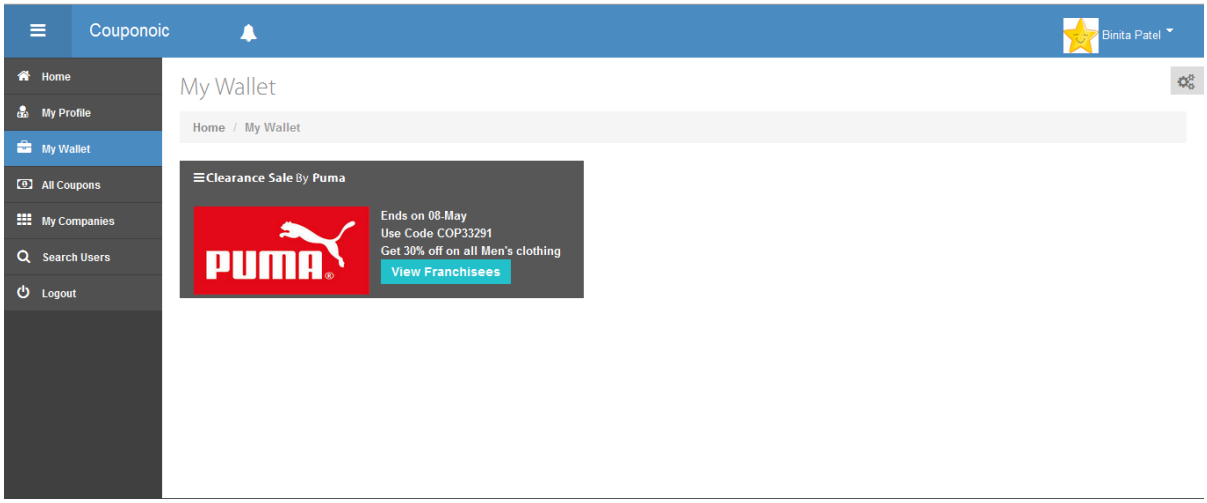

Fig. 3.8 Customer Wallet

3.3.7 ALL COUPONS

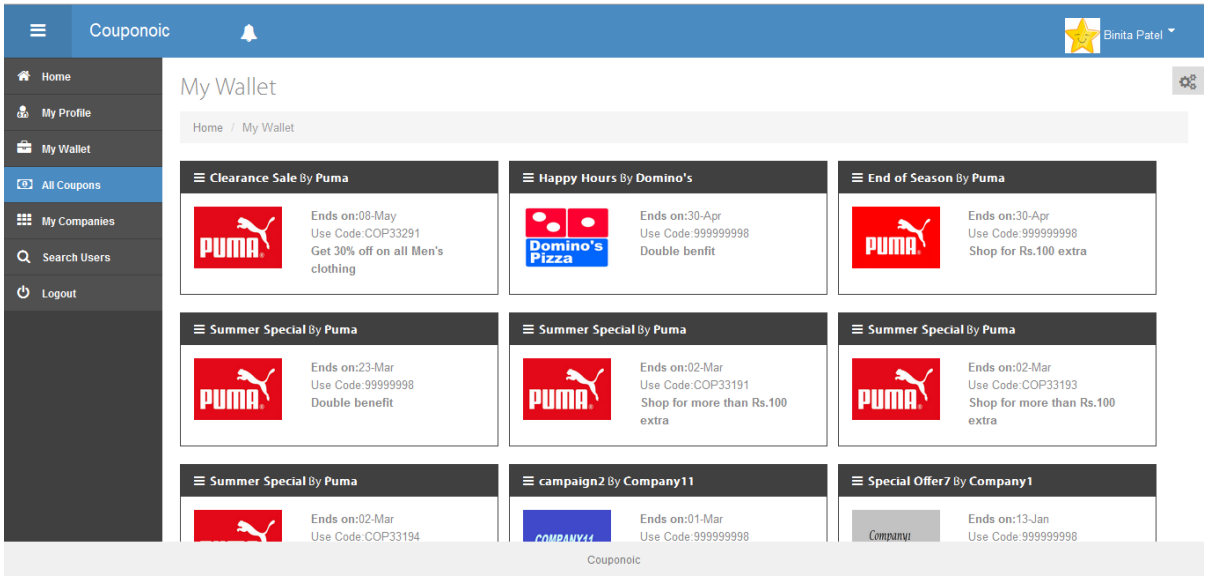

Fig. 3.9 All Coupons

3.3.6 MY COMPANIES

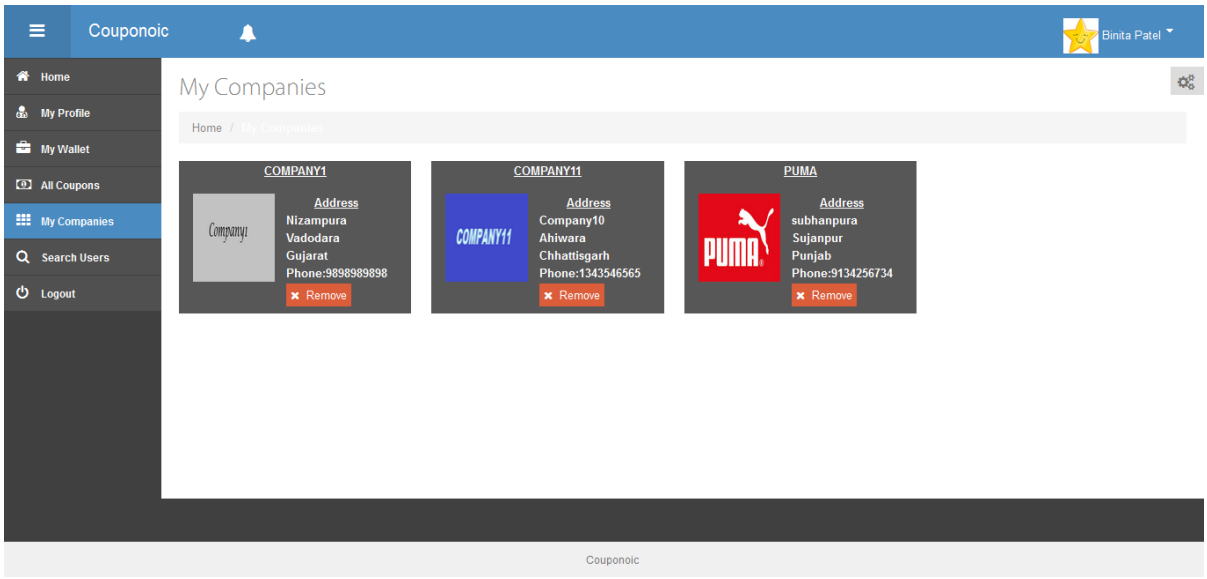

Fig. 3.10 My Companies

3.3.7 SEARCH USERS

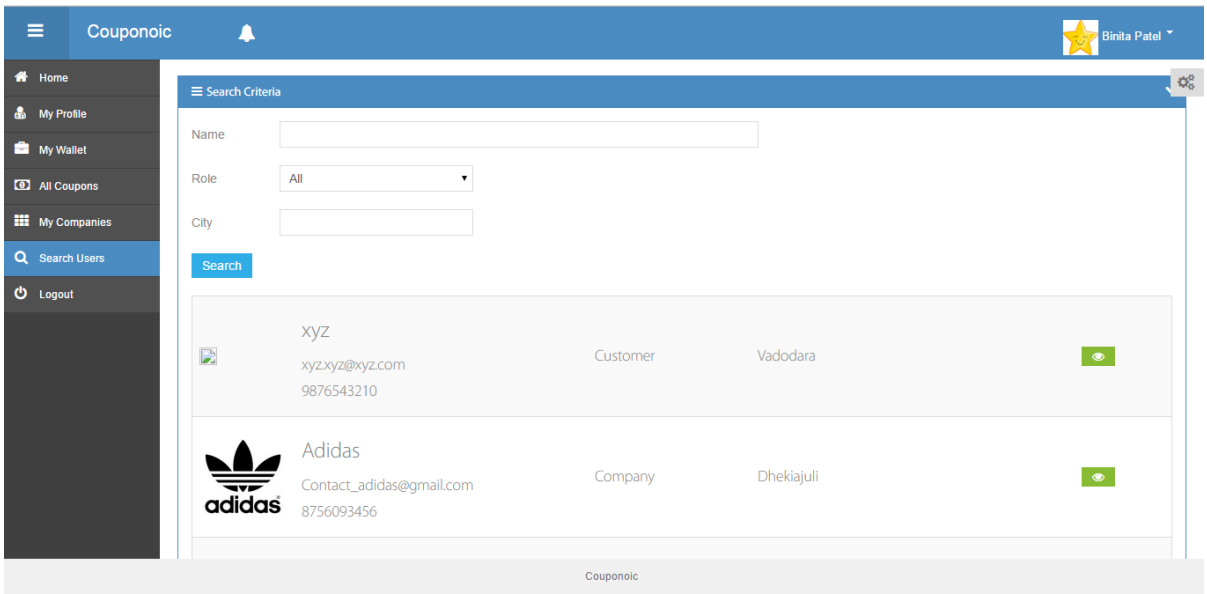

Fig. 3.11 Search Users

3.3.8 WELCOME PAGE

Fig. 3.12 Welcome Page

3.3.9 COMPANY HOME

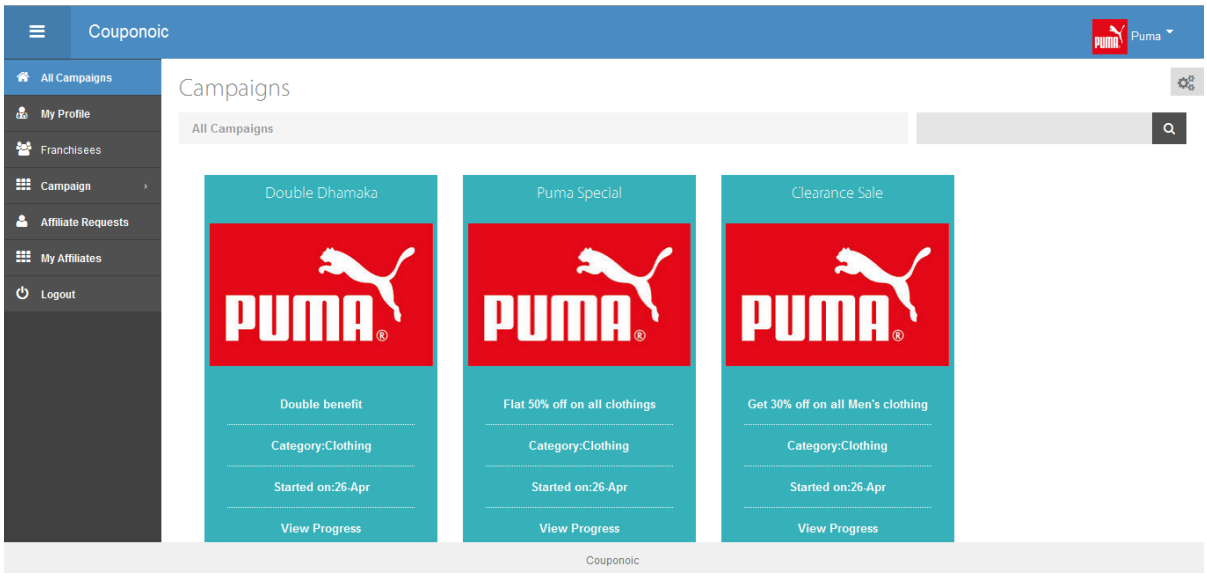

Fig. 3.13 All Campaigns

3.3.10 CAMPAIGN GENERATION

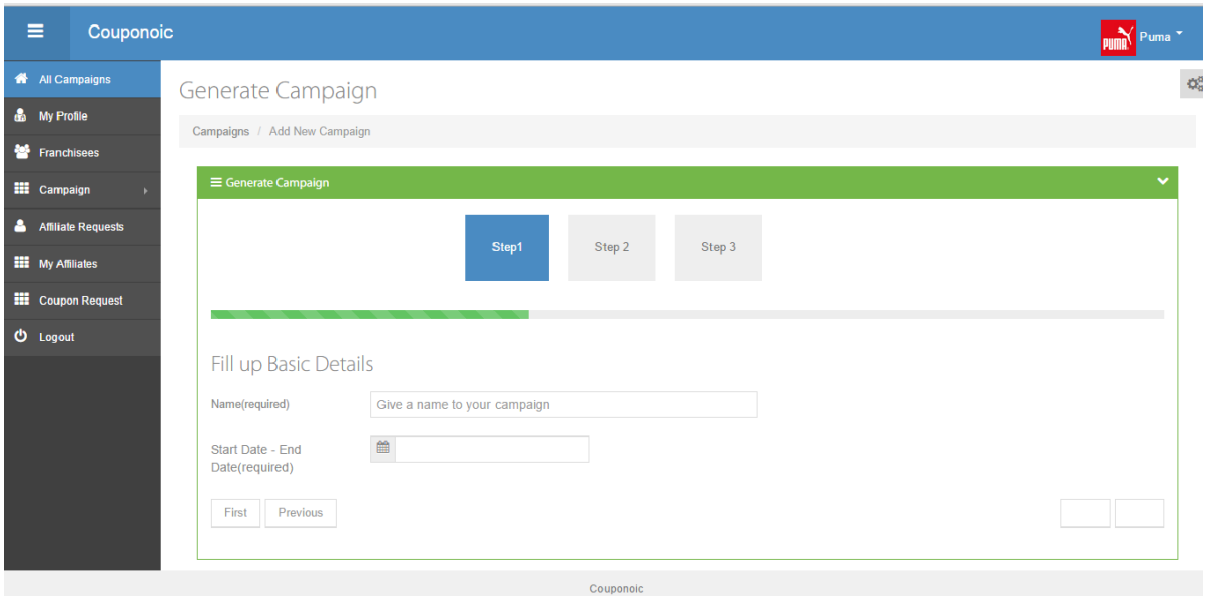

Fig. 3.14 Campaign Generation

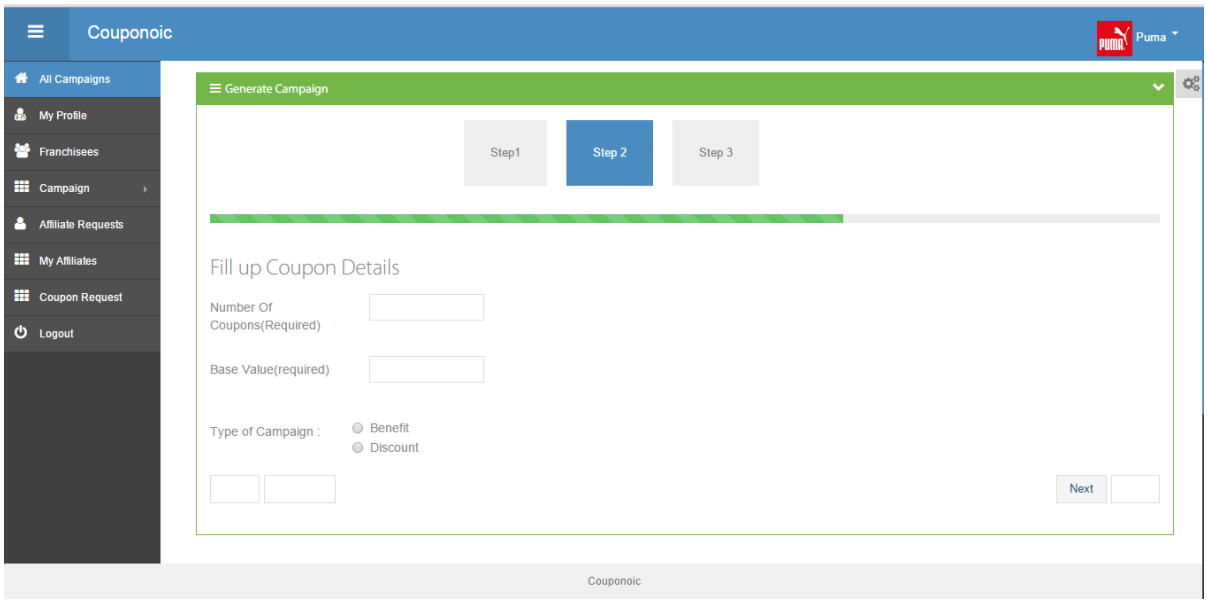

Fig. 3.15 Campaign Generation

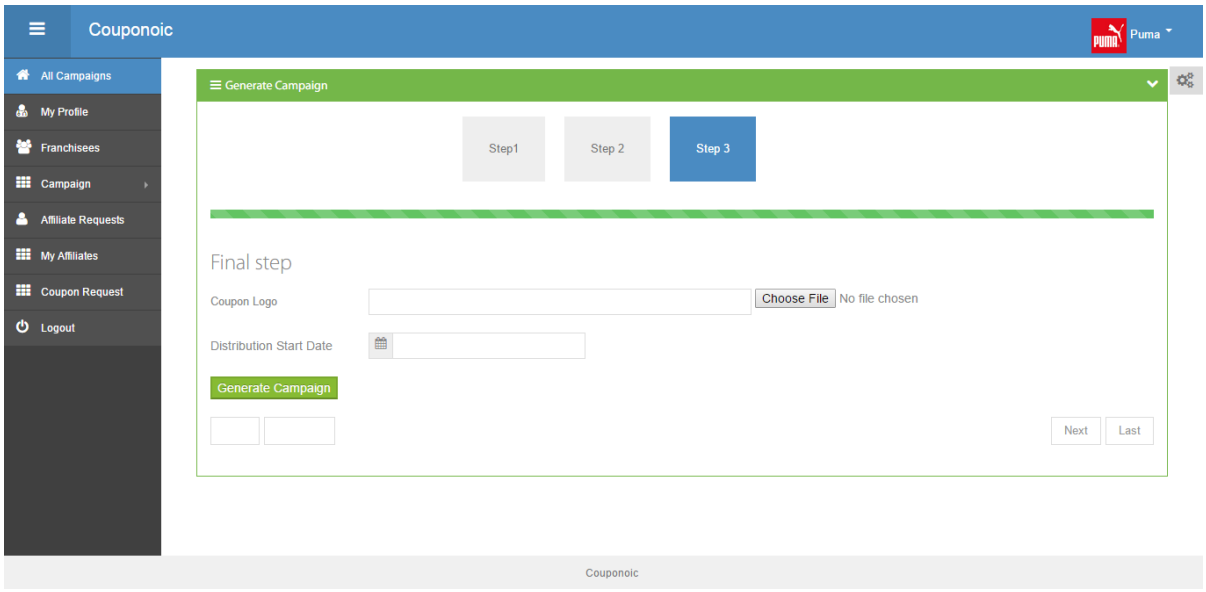

Fig. 3.16 Campaign Generation

3.3.11 AFFILIATE REQUESTS

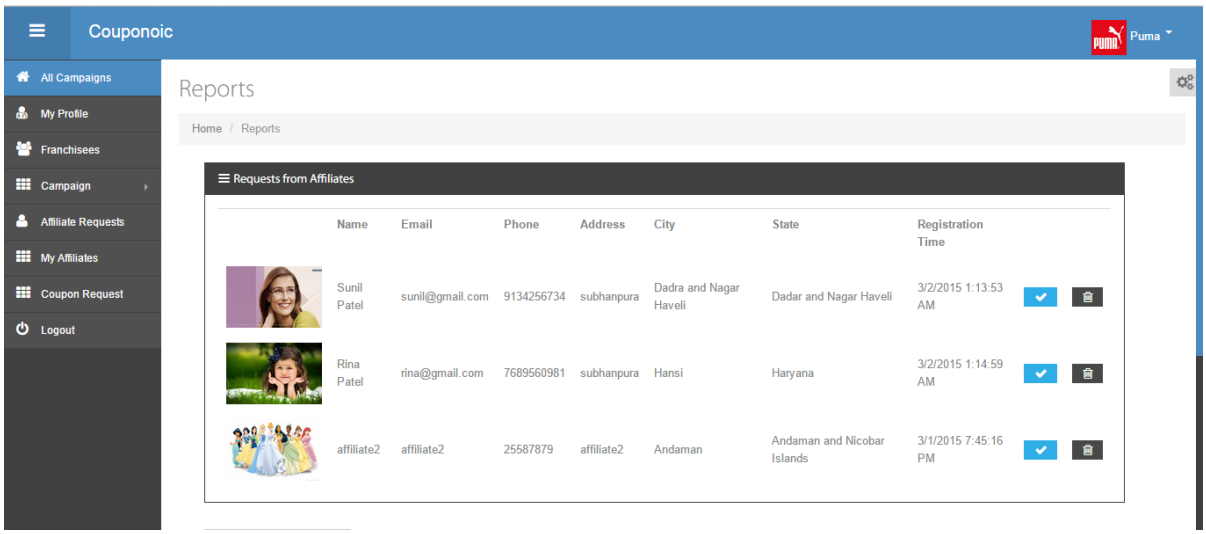

Fig. 3.17 Affiliate Requests

3.3.12 COUPON REQUESTS

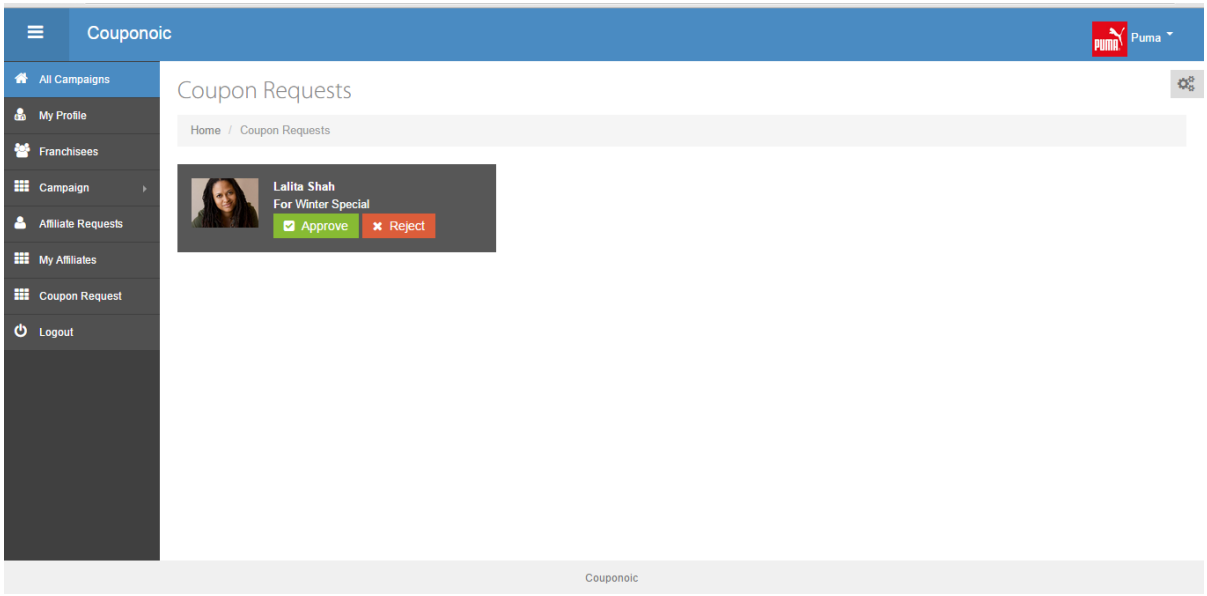

Fig. 3.18 Coupon Requests

3.3.13 AFFILIATE HOME

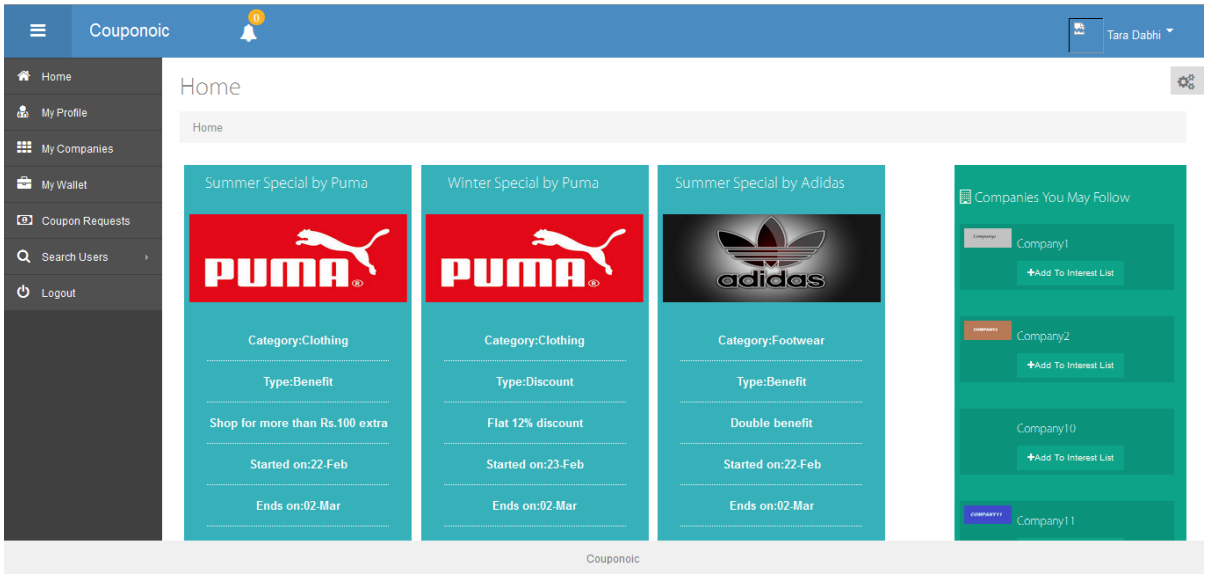

Fig. 3.19 Affiliate Home

3.3.14 AFFILIATE WALLET

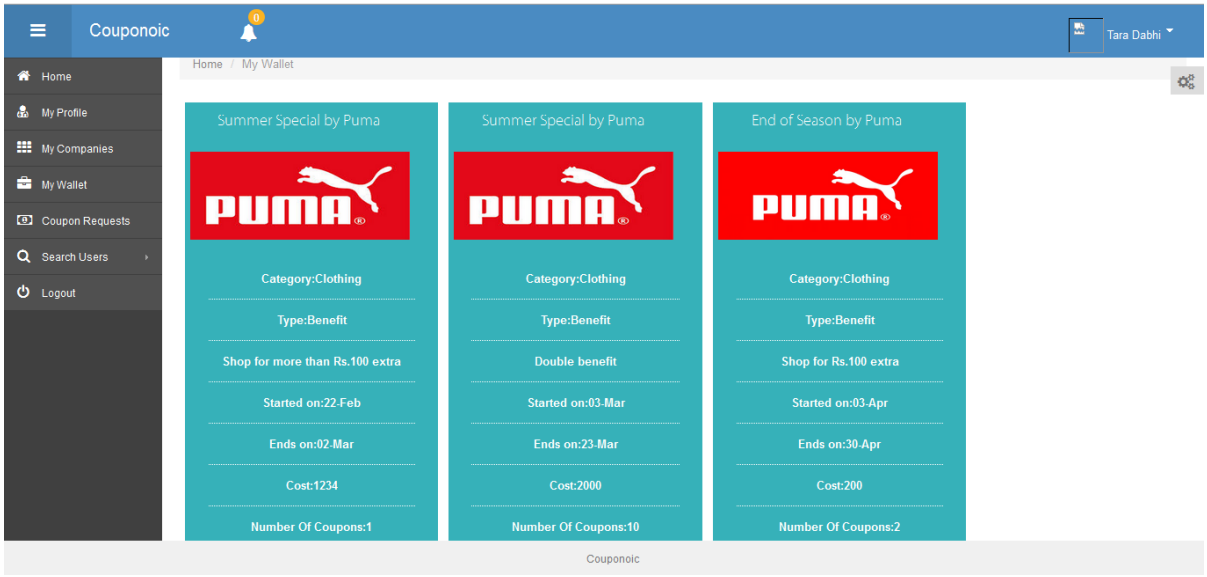

Fig. 3.20 Affiliate Wallet

3.3.15 ADMIN HOME

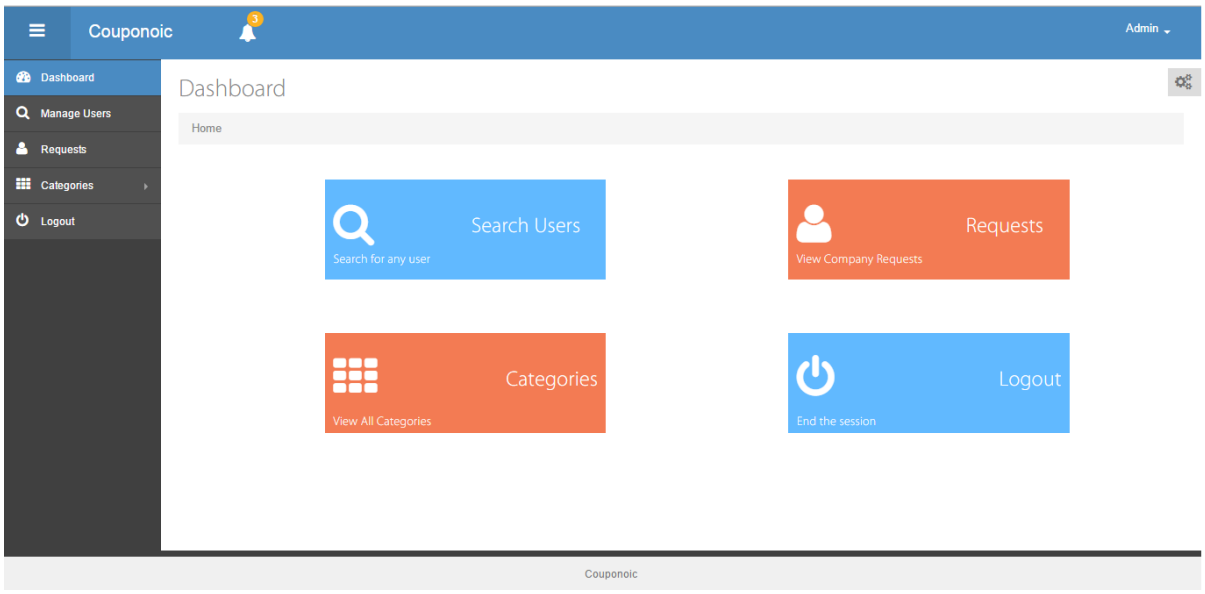

Fig. 3.21 Admin Home

3.3.16 CATEGORIES

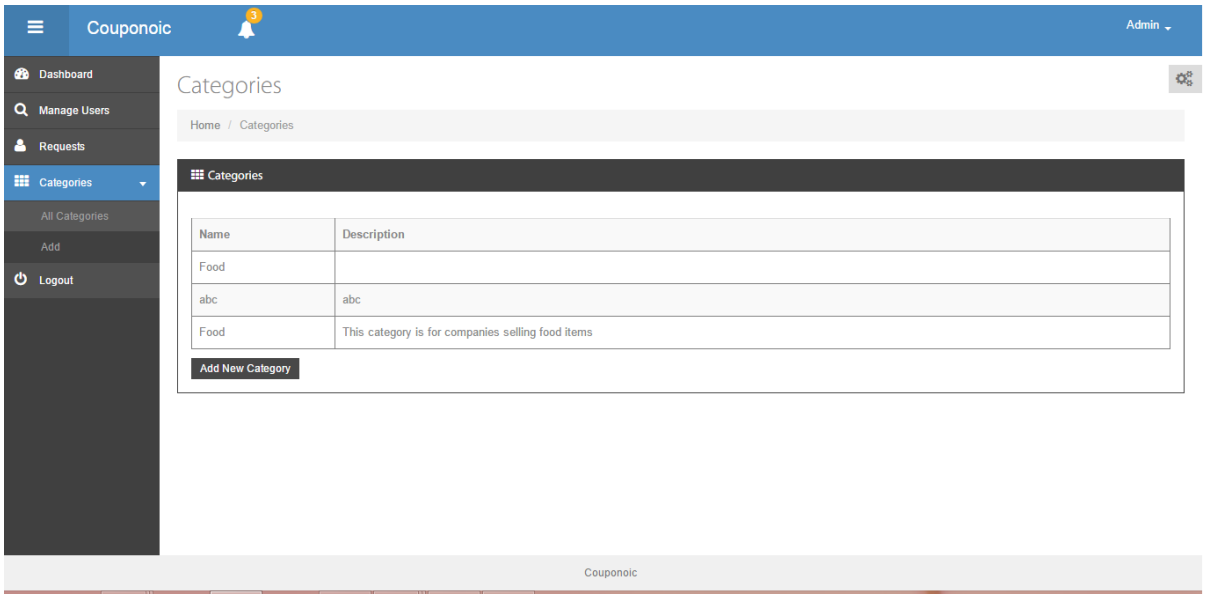

Fig. 3.22 Categories

3.3.17 ADD CATEGORY

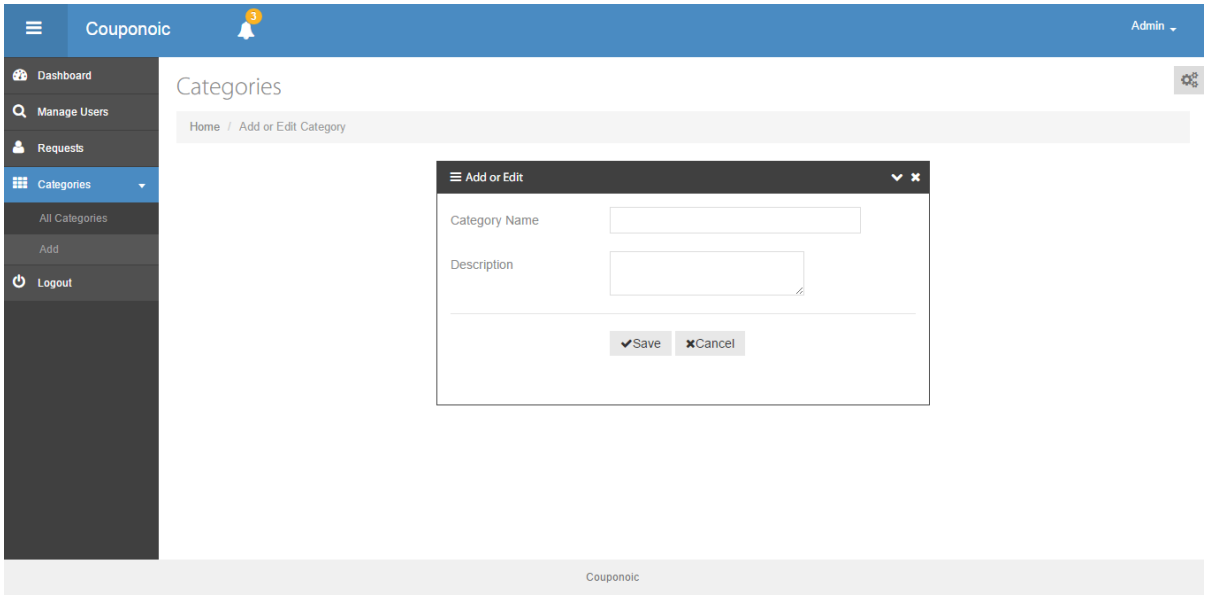

Fig. 3.23 Add Category

3.4. TESTING PLAN

Software testing has a dual function; it is used to establish the presence of defects in the program and it is also used to help judge whether or not the program is usable in practice. Software testing is used for validation and verification, which ensures that the software conforms to its specification and meets the need of the software customer. The testing is done by our Team members that act as novice users and test the application with all possible ways to find the bugs and error as well as check validation.

The Testing process

We tested the software process activities such as Design, Implementation and Requirement Engineering. As design errors are costlier to repair, once, the system has started to operate it is therefore quite obvious to repair them all at the initial stage of the system.

Figure 3.24 Testing phase in SDLC

Requirements Traceability

As most interested portion in the system is system meeting its requirements therefore testing should be planned so that all requirements are individually tested, we checked the output of certain combination of inputs, which gives desirable output or not. Strictly going along our requirements specifications gave us the path to get desirable result for system users.

Tested Items

Our test items were like, validation of each and every field when user enters the data. The user is not allowed to enter incorrect data and also he is not allowed to leave the text boxes blank. The mandatory fields which were the necessary fields were tested to contain data and not blank. Places where only numeric or alphabetic characters were required, testing was done to see that no data other than required was entered.

Testing schedule

We have tested each module back to back so that errors and omissions can be found as early as possible. Once the system has been developed fully we tested it on other machines by deploying it on various machines.

General Plan for Testing

The Sales and Inventory application is tested using Bottom-Up testing strategy.

1. Testing of each stored procedure (select / insert / update) with standard inputs.
2. Testing of each individual class of the assembly.
3. Testing of each individual user interface class.
4. Module wise testing (Also sub module wise testing) while code development.
5. Integration testing of system after integration of individual modules.
6. Security testing.
7. User level testing.

3.5. TESTING STRATEGY

Testing strategies is a general approach to the testing process rather than a method of devising particular system or components tests. Different testing strategies may be adopted depending on the type of system to be tested and the development process used. So considering functional oriented nature of this software we adopted mixture of following strategies.

Testing is the process carried out on software to detect the differences between its behavior and the desired behavior as stipulated by the requirements specifications.

Testing is advantageous in several ways. Firstly, the defects found help in the process of making the software reliable. Secondly, even if the defects found are not corrected, testing gives an idea as to how reliable the software is. Thirdly, over time, the record of defects found reveals the most common kinds of defects, which can be used for developing appropriate preventive measures such as training, proper design and reviewing.

The testing sub-process includes the following activities in a phase dependent manner:

- Create Test Plans.
- Create Test Specifications.
- Review Test Plans and Test Specifications.
- Conduct tests according to the Test Specifications, and log the defects.
- Fix defects, if any.
- When defects are fixed continue from activity.

The development process repeats this testing sub-process a number of times for the following phases.

- Unit Testing.
- Integration Testing.
- System Testing.
- Acceptance Testing.

Unit Testing tests a unit of code (module or program) after coding of that unit is completed. Integration Testing tests whether the various programs that make up a system, interface with each other as desired, fit together and whether the interfaces between the programs are correct. System Testing ensures that the system meets its stated design specifications. Acceptance Testing is testing by the users to ascertain whether the system developed is a correct implementation of the Software Requirements Specification.

Testing is carried out in such a hierarchical manner to ensure that each component is correct and the assembly/combination of components is correct. Merely testing a whole system at the end would most likely throw up errors in components that would be very costly to trace and fix.

3.6. TESTING METHODS

TESTING STAGES:

Unit testing

Unit testing focuses verification effort on the smallest unit of software design the software component or module. In this type of testing the individual modules were tested and verified whether accurate output was made available or not. The modules were tested individually as they could result into faster responses than when they were integrated.

Integration Testing

When the unit testing was over, all the modules were integrated one by one and tested as a whole. It might be possible that all modules may work individually, but they may not work when put together. Data can be lost across the interface, one module can have an adverse affect on the other or sub functions of another, when combined may not produce desired major function, individually acceptable imprecision may be magnified to unacceptable level; global data structure can present problem. So any system has to be tested this way so that the final output is the desired one. Also the common functions throughout the system were taken and formed into a class so that they could be accessed from the same place without creating any ambiguities.

Validation Testing

After the integration testing software is completely assembled as a package, interfacing errors have been uncovered and corrected, validation testing begins. Validation testing can be defined in many ways but a simple definition is that a validation succeeds when the software functions in a manner that can be reasonably accepted by the Client.

System testing

Any software is only one element of a larger computer based system. Ultimately software is incorporated with other system elements like hardware, people, information and a series of system integration and validation tests are conducted. System testing is actually a series of different test whose primary purpose is to fully exercise the computer based system. A

necessary check to be performed was that of the new system being accepted by the older window's application. The data is not being misplaced when both the systems exist.

Storage Testing

The database of the system has to be stored on the hard disk. So the storage capacity of the hard disk should be enough to store all the data required for the efficient running of the system.

3.6.1 ACCEPTANCE TESTING

Before delivering the product to the client, it will be tested for stability and correctness of results. By conducting the fore mention test cases, we should be able to conclude on the reliability of the product before the client tests it for acceptance. First tests will be conducted on a sample portion of the database. Later tests will also include testing on the entire database. The client will test the program for criteria mentioned in the next section before accepting the product.

- Specific **acceptance criteria** (the client expects the following):
- Conformity with the predefined requirements.
- Correct data returned from the SQL Server searches
- Functionality in multiple environments. Since this is a web product, it must be compatible to various browsers on different systems.

3.6.2 FEATURE LEVEL TESTING

Task-Oriented Functional Tests

The mandatory fields must not be left empty.

User name and Password must match with registered details.

Session must be generated for each User.

Searching facilities is performed by specified criteria.

Integration Tests

Components or modules that should be tested independently are following

1) Browser Compatibility.

2) Hardware compatibility.

3) Interface Testing.

4) User Input Testing.

System-Level Tests

In system level tests we will use black-box testing since all independent modules or functionalities are already tested before. System level test will start after unit and integration testing gets completed.

Unstructured Tests

As user input to the system is not drastically varying, it will not require Unstructured Tests. Input related modules would be tested with set of test data, which covers whole data set of user input.

Volume Tests

In other scenario we will test system against large amount of transactions that will help in testing system to function properly at high stress levels.

Performance Tests

Systems have certain features that highly affect their takeaway value. These features need to be tested against its performance.

- User Response Time - Should be moderate. It should be in range of 1-5 seconds.
- Searching Time - Should be in range of 3-10 seconds.
- System Load Time - Should be in range of 1-5 seconds as is a web-based system.

3.6.3 CONFIGURATION AND COMPATIBILITY TESTING

Application Tests

- Tests were conducted to find errors and accuracy when requested by more user agents.
- Test response time to users by IIS application server.
- These tests were focused on operating large amount of data by MS SQL Server database server, and also data fetch time for response to user.

Browser Tests

- Tests were conducted to find errors on different versions of Internet Explorer.
- Tests were focused on the user interface on the client side. It was found that the means provided gull functionality when Internet Explorer 5.0 or later was used.
- All the processing is to be done on the server side so, on client side only some JavaScript's are to be executed and most of the commonly available browsers will have no problem.

Operating System Tests

- This test was conducted to find errors on different versions of operating system.
- Tests were focused on the user interface using different operating system like Window XP and Linux.
- In this we tested all the JavaScript's.

3.7 TEST CASES

Table 3.1 Test case for Candidate Login

Purpose	Required Input	Expected Result
Username and password both blank	Neither Username nor Password	Invalid Credentials. Redirected to login page.
Username or Password blank	Either Username or Password but not both	Invalid Credentials. Redirected to login page.
Username correct but Password incorrect	Correct username and incorrect password	Invalid Credentials. Redirected to login page.
Username incorrect but Password correct	Incorrect username and correct password	Invalid Credentials. Redirected to login page.
Username and Password both correct	Correct username and Correct password	Valid Credentials. Redirected to home page based on the user.

Table 3.2 Test case for Buying coupons(Guest User)

Purpose	Required Input	Expected Result
If guest user clicks on 'Buy' button.	Login to the system is required.	User will be prompted with a message to register and then log in.

Table 3.3 Test Case for Registration Page

Purpose	Required Input	Expected Result
Name of the company or affiliate or customer registering on our portal is compulsory.	Name is to be entered and it should not have wild characters in it.	User will be prompted a message to enter a valid name.
Email of the user must be in the correct format	Email address of the user.	If not entered in the desired format the user will be

		prompted a message "Invalid email".
Phone number should be a valid mobile number.	Phone number of the user in the correct format of Indian mobile number.	If not entered in the desired format the user will be prompted a message "Invalid Phone number".
Username must be unique in the system.	Username of the system.	If the username will not be unique a prompt message