4
N-Tier Architecture

In the previous chapters we have seen, through code samples, how 1-tier 2-layer and 1-tier 3-layer solutions work in our ASP.NET web projects, and the advantages of going for a 3-layered architecture to create scalable and maintainable web applications. In all of the high-level architectural configurations we have studied up to now, we were dealing with the basic structuring and coding of the main ASP.NET application code—the Visual Studio solution, to be more precise. We had not considered the database and the client browser as separate physical tiers. We did this because we wanted to focus on how we can structure our main application solution in terms of layers and tiers. However, from this chapter onwards, we will include the physical database and the browser as distinct tiers being a part of the whole application. The reason for this change is because from now on we will be breaking our 1-tier application into multiple physical tiers (and not simply layers) and see how the entire distributed system works in collaboration. The term *distributed system* involves:

- The main ASP.NET application code, which will be broken down further into separate physical tiers so that each tier or assembly can be used independently of the others
- The physical database (an external RDMBS such as MS SQL Server, or any external storage such as XML files), also called the Data Tier
- The client browser (Internet Explorer or Firefox where the HTML will be rendered) also called the Presentation Tier

In this chapter we will learn:

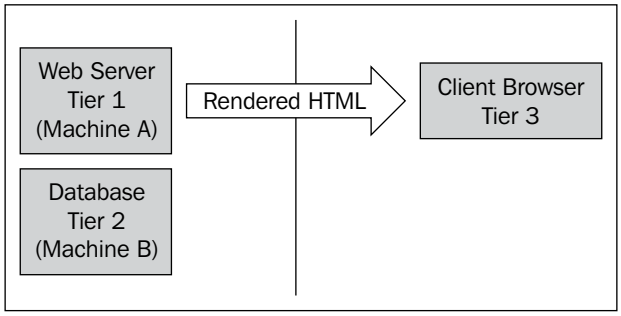
- Why we need N-tier based systems
- How to create a 4-tier architecture
- How to create a 5-tier architecture
- What Data Transfer Objects are
- What a Lazy Loading design pattern is

Here, we will move from the basic 3-tier client-server model described in the previous chapters, to a four, five, or higher tiered architectures—in short, n-tier systems. Before moving ahead, let me emphasize an important point: it is very crucial to understand that there is no perfect architecture. Each application is unique, and therefore there can be different ways to implement an n-tier architecture. Hence, we will learn and understand the basic fundamentals of the n-tier system, and implement it in one particular style. The concepts discussed in this chapter will be generic enough to help you learn and apply your own customized n-tier style suited to the unique need of each project.

Why N-Tier?

"**N-tier**" is a team that almost every software developer knows, and a team that has been hugely debated across forums, blogs and offline discussion groups. During my early years as a programmer, I was so impressed with n-tier architecture that I thought every application should be n-tier, without even understanding the high-level view, which I eventually realized comes later with experience! To n-tier or not is the question for which we will try to find an answer in this chapter.

We have already seen 1-tier architectures, and if we keep the database on a separate machine with its own CPU, we will have a rudimentary 3-tier architecture in our web projects, as shown here:



We have already seen how to break the main application tier in the above 3-tier application into logical layers. Now, the first question that comes to one's mind is why, exactly, do we need to break these logical layers into their own, separate, physical assemblies as tiers.

The answer is that **n-tiered development** allows a component-based approach to software design, allowing developers to make updates and changes to individual tiers without breaking other code. Let me explain this further.