Application Architecture Guide - Chapter 9 - Layers and Tiers

From Guidance Share

Note - The patterns & practices Microsoft Application Architecture Guide, 2nd Edition is now live at http://msdn.microsoft.com/en-us/library/dd673617.aspx.

- J.D. Meier , Alex Homer, David Hill, Jason Taylor , Prashant Bansode , Lonnie Wall, Rob Boucher Jr, Akshay Bogawat

Contents

Objectives

- Learn how to divide your applications into separate physical and logical parts.
- Learn the difference between logical layers and physical tiers.
- Learn about services that you can use to expose logic on layers.
- Learn about the components commonly encountered in layers and tiers.
- Learn about applications that support multiple client types.
- Learn how to choose an appropriate functional layout for your applications.

Overview

This chapter discusses the overall structure for applications, in terms of the logical grouping of components into separate layers or tiers that communicate with each other and with other clients and applications. Layers are concerned with the logical division of components and functionality, and take no account of the physical location of components on different servers or in different locations. Tiers are concerned with the physical distribution of components and functionality on separate servers, computers, networks, and remote locations. Although both layers and tiers use the same set of names (presentation, business, service, and data), remember that only tiers imply a physical separation. It is quite common to locate more than one layer on the same physical machine. You can think of the term “tier” as referring to physical distribution patterns such as two-tier, three-tier, and n-tier.

Layers

Layers are the logical groupings of the software components that make up the application or service. They help to differentiate between the different kinds of tasks performed by the components, making it easier to create a design that supports reusability of components. Each logical layer contains a number of discrete component types grouped into sublayers, with each sublayer performing a specific type of task. By identifying the generic types of components that exist in most solutions, you can construct a meaningful map of an application or service, and then use this map as a blueprint for your design.

Splitting an application into separate layers that have distinct roles and functionalities helps you to maximize maintainability of the code, optimize the way that the application works when deployed in different ways, and provide a clear delineation between locations where certain technology or design decisions must be made.

Presentation, Business, and Data Services

At the highest and most abstract level, the logical architecture view of any system can be considered to be a set of cooperating services grouped into the following layers, as shown in Figure 1.

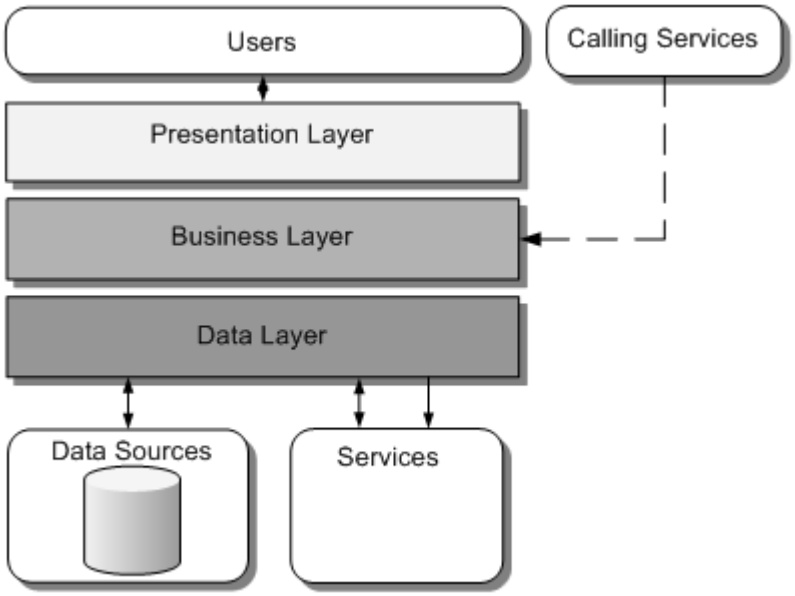

Figure 1 The logical architecture view of a layered system The sections of the application design shown in Figure 1 can be thought of as three basic sets of services:

- **Presentation services**. These are the user-oriented services responsible for managing user interaction with the system, and generally consist of components located within the presentation layer. They provide a common bridge into the core business logic encapsulated in the business services.
- **Business services**. These services implement the core functionality of the system, and encapsulate the relevant business logic. They generally consist of components located within the business layer, which may expose service interfaces that other callers can use.
- **Data services**. These services provide access to data that is hosted within the boundaries of the system, and data exposed by other back-end systems; perhaps accessed through services. The data layer exposes data to the business layer through generic interfaces designed to be convenient for use by business services.

Components

Each layer of an application will contain a series of components that implement the functionality for that layer. These components should be cohesive and loosely coupled to simplify reuse and maintenance. Figure 2 shows the types of components commonly found in each layer.

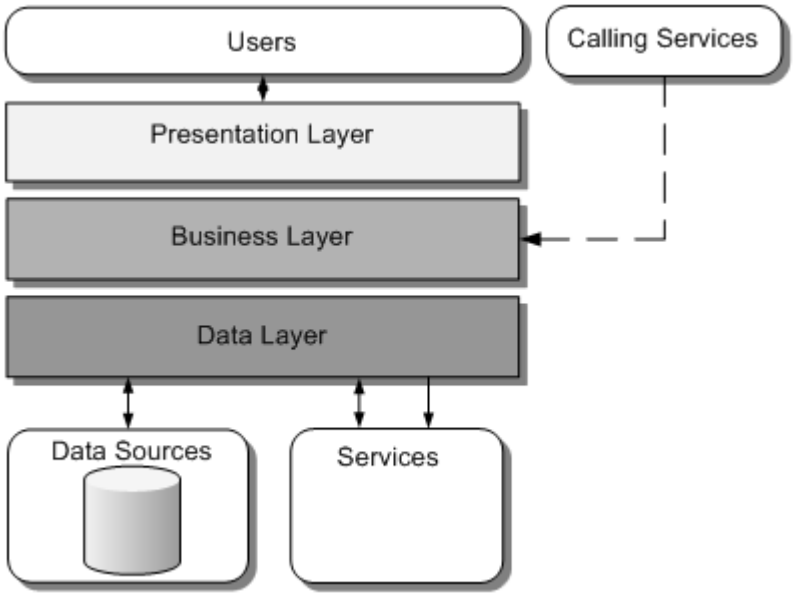

Figure 2 Types of components commonly found in each layer The components shown in Figure 2 are described in the following sections.

Presentation Layer Components

Presentation layer components implement the functionality required to allow users to interact with the application. The following types of components are commonly found in the presentation layer:

- **User interface (UI) components**. These components provide the mechanism for users to interact with the application. They format data and render it for display, and acquire and validate data entered by users.
- **UI process components**. To help synchronize and orchestrate user interactions, it can be useful to drive the process using separate UI process components. This prevents the process flow and state management logic from being hard-coded into the UI elements themselves, and allows you to reuse the same basic user interaction patterns in other user interfaces.

Business Layer Components

Business layer components implement the core functionality of the system, and encapsulate the relevant business logic. The following types of components are commonly found in the business layer:

- **Application façade.** This is an optional feature that you can use to combine multiple business operations into single messaged-based operations. This feature is useful when you locate the presentation layer components on a separate physical tier from the business layer components, allowing you to optimize use of the communication method that connects them.
- **Business components**. These components implement the business logic of the application. Regardless of whether a business process consists of a single step or an orchestrated workflow, your application is likely to require components that implement business rules and perform business tasks.

- **Business workflows**. After the UI components collect the required data from the user and pass it to the business layer, the application can use this data to perform a business process. Many business processes involve multiple steps that must be performed in the correct order, and may interact with each other through an orchestration. Business workflow components define and coordinate long-running, multi-step business processes, and can be implemented using business process management tools.
- **Business entity components**. Business entities are used to pass data between components. The data represents real-world business entities, such as products or orders. The business entities that the application uses internally are usually data structures, such as DataSets, DataReaders, or Extensible Markup Language (XML) streams, but they can also be implemented using custom object-oriented classes that represent the real-world entities that your application will handle.

Data Layer Components

Data layer components provide access to data that is hosted within the boundaries of the system, and data exposed by other back-end systems. The following types of components are commonly found in the data layer:

- **Data access components**. These components abstract the logic required to access the underlying data stores. Doing so centralizes data access functionality and makes the application easier to configure and maintain.
- **Data helper and utility components**. Most data access tasks require common logic that can be extracted and implemented in separate reusable helper components. This helps to reduce the complexity of the data access components and centralizes the logic, which simplifies maintenance. Other tasks that are common across data layer components, and not specific to any set of components, may be implemented as separate utility components. Both helper and utility components can often be reused in other applications.
- **Service agents**. When a business component must use functionality provided by an external service, you might need to implement code to manage the semantics of communicating with that particular service. Service agents isolate the idiosyncrasies of calling diverse services from your application, and can provide additional services such as basic mapping between the format of the data exposed by the service and the format your application requires.

Cross-Cutting Components

Many tasks carried out by the code of an application are required in more than one layer. Cross-cutting components implement specific types of functionality that can be accessed from components in any layer. The following are common types of cross-cutting components:

- **Components for implementing security**. These may include components that perform authentication, authorization, and validation.
- **Components for implementing operational management tasks**. These may include components that implement exception handling policies, logging, performance counters, configuration, and tracing.
- **Components for implementing communication**. These may include components that communicate with other services and applications.

Services and Layers

From a high-level perspective, a service-based solution can be seen as being composed of multiple services, each communicating with the others by passing messages. Conceptually, the services can be seen as components of the overall solution. However, internally, each service is made up of software components, just like any other application, and these components can be logically grouped into presentation, business, and data services. Other applications can make use of the services without being aware of the way they are implemented. The principles discussed in the previous sections on the layers and components of an application apply equally to service-based solutions.

Services Layer

When an application will act as the provider of services to other applications, as well as implementing features to support clients directly, a common approach is to use a services layer that exposes the functionality of the application, as shown in Figure 3.

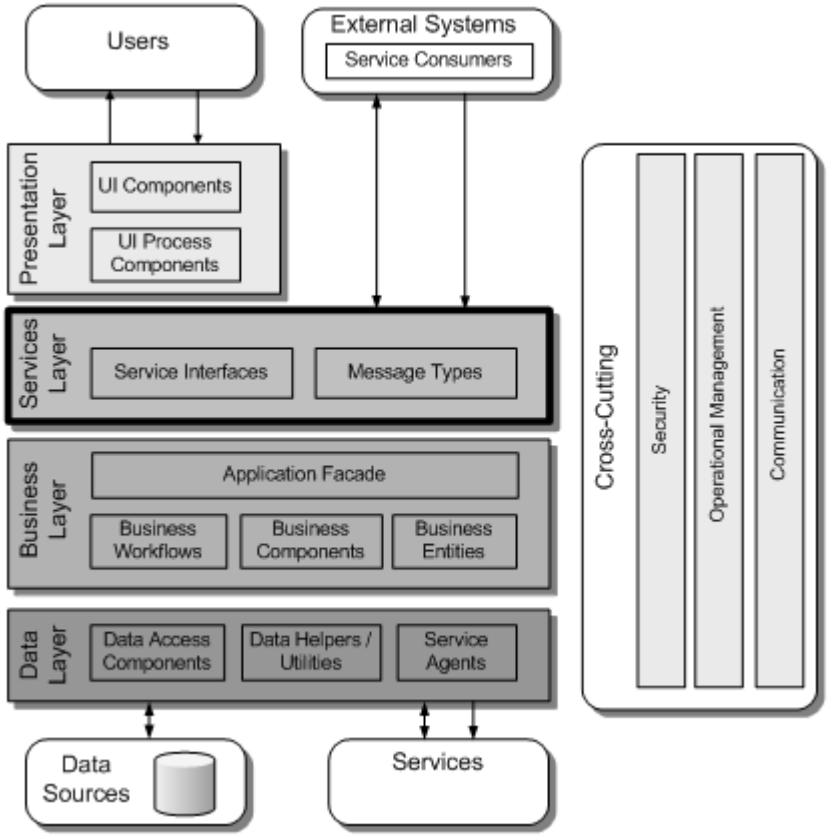

Figure 3 Incorporating a services layer in an application The following section describes the components usually found in the services layer.

Services Layer Components

Services layer components provide other clients and applications with a way to access business logic in the application, and make use of the functionality of the application by passing messages to and from it over a communication channel. The following types of components are commonly found in the services layer:

- **Service interfaces**. Services expose a service interface to which all inbound messages are sent. The definition of the set of messages that must be exchanged with a service in order for the service to perform a specific business task constitutes a contract. You can think of a service interface as a façade that exposes the business logic implemented in the service to potential consumers.
- **Message types**. When exchanging data across the service layer, data structures are wrapped by message structures that support different types of operations. For example, you might have a Command message, a Document message, or another type of message. These message types are the “message contracts” for communication between service consumers and providers.

Multi-Client Application Scenario

Applications often must support different types of clients. In this scenario, the application will usually expose services to external systems, as well as directly supporting local clients, as shown in Figure 4.

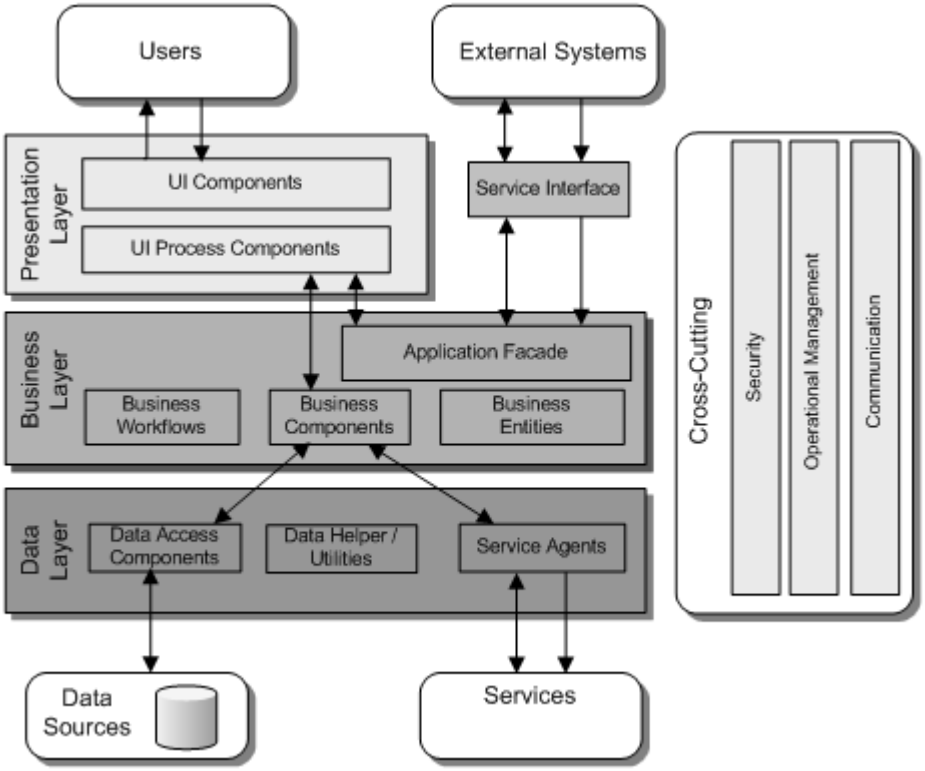

Figure 4 The multi-client application scenario In this scenario, local and known client types can access the application through the presentation layer, which communicates either directly to the components in the business layer or through an application façade in the business layer if the communication methods require composition of functionality. Meanwhile, external clients and other systems can treat the application as an "application server" and make use of its functionality by communicating with the business layer through service interfaces.

Business Entities Used by Data and Business Services

There are many cases where business entities must be accessible to components and services in both the business layer and the data layer. For example, business entities can be mapped to the data source and accessed by business components. However, you should still separate business logic from data access logic. You can achieve this by moving business entities into a separate assembly that can be shared by both the business services and data services assemblies, as shown in Figure 5. This is similar to using a dependency inversion pattern, where business entities are decoupled from the business and data layer so that both business and data layers are dependent on business entities as a shared contract.

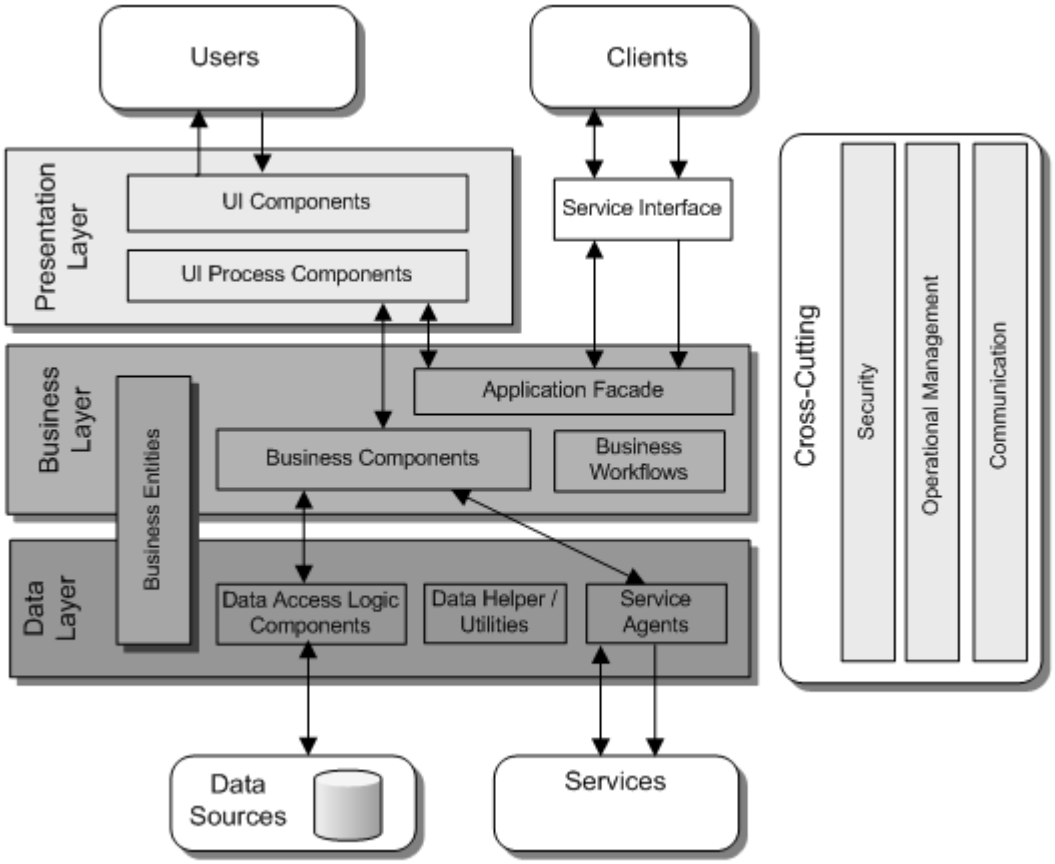

Figure 5 Business entities used by data and business services

Choosing Layers for Your Application

Use a layered approach to improve the maintainability of your application and make it easier to scale out when necessary to improve performance. Keep in mind that a layered approach adds complexity and can impact your initial development time. Be smart about adding layers, and don't add them if you don't need them. Use the following guidelines to help you decide on the layering requirements for your application:

- If your application does not expose a UI, such as a service application, you do not require a presentation layer.
- If your application does not contain business logic, you may not require a business layer.
- If your application does not expose services, you do not require a services layer.
- If your application does not access data, you do not require a data layer.
- Only distribute components where this is necessary. Common reasons for implementing distributed deployment include security policies, physical constraints, shared business logic, and scalability.
- In Web applications, deploy business components that are used synchronously by user interfaces or user process components in the same physical tier as the user interface to maximize performance and ease operational management, unless there are security implications that require a trust boundary between them.

- In rich client applications where the UI processing occurs on the desktop, you may prefer to deploy components that are used synchronously by UIs or user process components in a separate business tier for security reasons, and to ease operational management.
- Deploy service agent components on the same tier as the code that calls the components, unless there are security implications that require a trust boundary between them.
- Deploy asynchronous business components, workflow components, and business services on a separate physical tier where possible.
- Deploy business entities on the same physical tier as the code that uses them.

Tiers

Tiers represent the physical separation of the presentation, business, services, and data functionality of your design across separate computers and systems. Common tiered design patterns are two-tier, three-tier, and n-tier. The following sections explore each of these scenarios.

Two-Tier

The two-tier pattern represents a basic structure with two main components, a client and a server. In this scenario, the client and server may exist on the same machine, or may be located on two different machines. Figure 6 illustrates a common Web application scenario where the client interacts with a Web server located in the client tier. This tier contains the presentation layer logic and any required business layer logic. The Web application communicates with a separate machine that hosts the database tier, which contains the data layer logic.

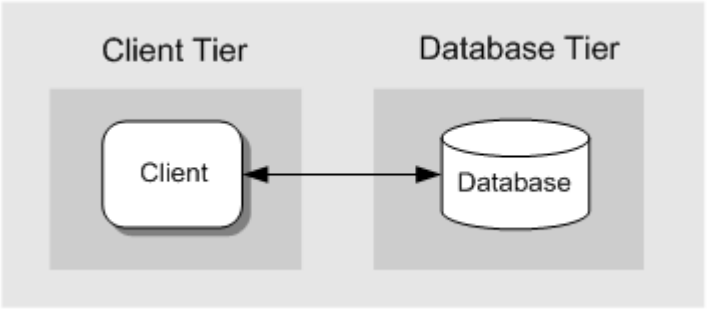

Figure 6 The two-tier deployment pattern

Three-Tier

In a three-tier design, the client interacts with application software deployed on a separate server, and the application server interacts with a database that is also located on a separate server. This is a very common pattern for most Web applications and Web services. Figure 7 illustrates the three-tier deployment pattern.

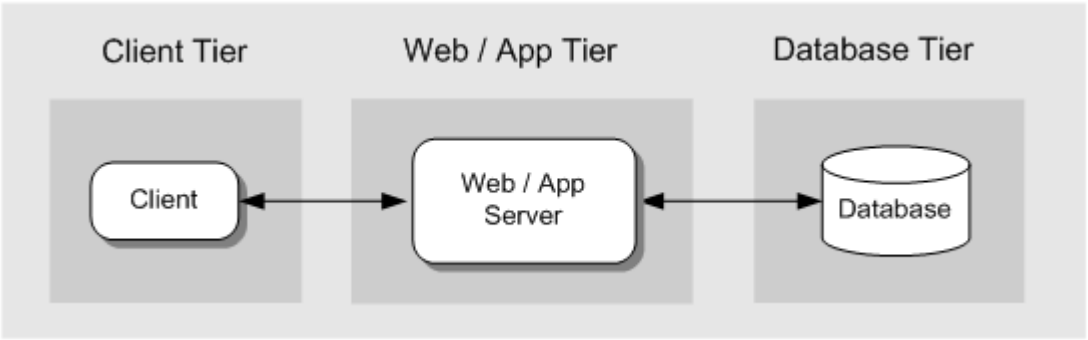

Figure 7 The three-tier deployment pattern

N-Tier

In this scenario, the Web server (which contains the presentation layer logic) is physically separated from the application server that implements the business logic. This usually occurs for security reasons, where the Web server is deployed within a perimeter network and accesses the application server located on a different subnet through a firewall. It is also common to implement a firewall between the client and the Web tier. Figure 8 illustrates the n-tier deployment pattern.

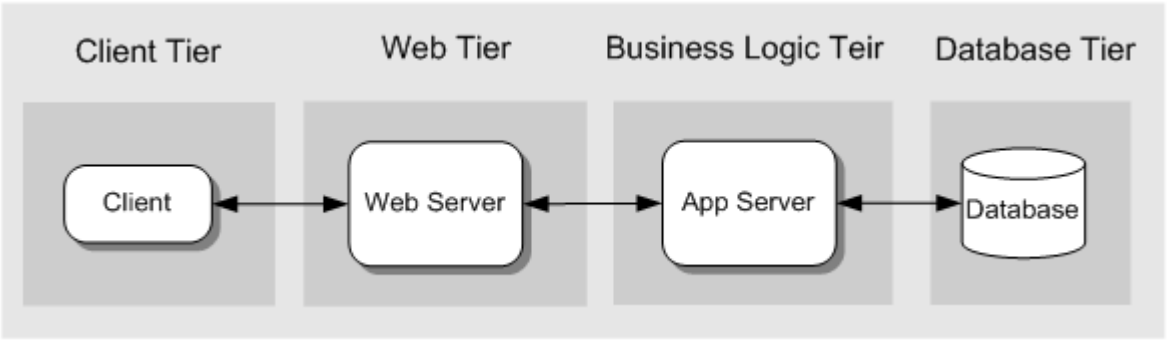

Figure 8 The n-tier deployment pattern

Choosing Tiers for Your Application

Placing your layers on separate physical tiers can help performance by distributing the load across multiple servers. It can also help with security by segregating more sensitive components and layers onto different networks or on the Internet versus an intranet. Keep in mind that adding tiers increases the complexity of your deployment, so don’t add more tiers than you need.

In most cases, you should locate all of the application code on the same server, using a single-tier approach. Whenever communications must cross physical boundaries, performance is affected because the data must be serialized. However, in some cases you might need to split functionality across servers, because of security or organizational constraints. To mitigate the effects of serialization, depending on where servers are located, you can usually choose communication protocols that are optimized for performance.

Consider the client/server or two-tier pattern if:

- You are developing a client that must access an application server.
- You are developing a stand-alone client that accesses an external database.

Consider the three-tier pattern if:

- You are developing an intranet-based application, where all servers are located within a private network.
- You are developing an Internet-based application, and security requirements do not restrict implementation of business logic on the public-facing Web or application server.

Consider the N-tier pattern if:

- Security requirements dictate that business logic cannot be deployed to the perimeter network.
- You have application code that makes heavy use of resources on the server, and you want to offload that functionality to another server.