## *Enterprise Application Development*

An enterprise application is a business application, obviously. As most people use the term, it is a *big* business application. In today’s corporate environment, enterprise applications are complex, scalable, distributed, component-based, and mission-critical. They may be deployed on a variety of platforms across corporate networks, intranets, or the Internet. They are data-centric, user-friendly, and must meet stringent requirements for security, administration, and maintenance. In short, they are highly complex systems.

Designing and developing such enterprise applications means satisfying hundreds or thousands of separate requirements. What’s more, every development decision you make to satisfy each requirement affects many other requirements, often in ways that are difficult to understand or predict — and the failure to meet *any* of these requirements can mean the failure of the entire project!

The Enterprise Application Model introduced in this document gives you a way to look at the enterprise application "whole cloth," to bring some order out of this complexity. It organizes an application’s requirements into a small set of distinct but interdependent categories, and shows how each requirement interacts with the others. By balancing the effects of each design choice against all the other requirements, you can avoid the nasty shock of discovering too late that you’ve overlooked or underestimated some important design consideration. Later chapters will use this model to break up the complex task of enterprise development into a small set of more manageable sub-tasks.

## *Definition*

An enterprise application is the phrase used to describe applications (or software) that a business would use to assist the organization in solving enterprise problems. When the word "[enterprise](https://www.webopedia.com/TERM/E/enterprise.html)" is combined with "[application](https://www.webopedia.com/TERM/A/application.html)," it usually refers to a [software](https://www.webopedia.com/TERM/S/software.html) platform that is too large and too complex for individual or [small business](https://www.webopedia.com/TERM/S/SMB.html) use.

## *Integration and Deployment*

Enterprise applications are typically designed to interface or integrate with other enterprise applications used within the organization, and to be deployed across a variety of [networks](https://www.webopedia.com/TERM/N/network.html) ([Internet](https://www.webopedia.com/TERM/I/Internet.html), [Intranet](https://www.webopedia.com/TERM/I/Intranet.html) and corporate networks) while meeting strict requirements for [security](https://www.webopedia.com/TERM/S/security.html) and administration management.

## *Proprietary Enterprise Apps*

[Proprietary](https://www.webopedia.com/TERM/P/proprietary.html) enterprise applications are usually designed and deployed in-house by a specialized IT development team within the organization. However, an enterprise may [outsource](https://www.webopedia.com/TERM/O/outsource.html) some or all of the development of the application, and bring it back in-house for deployment.

## *Application Service Providers (ASP)*

Today, using enterprise **a***pplication***s***ervice***p***roviders* ([ASP](https://www.webopedia.com/TERM/A/application_service_provider.html)) is more prevalent. Here, the enterprise application is designed by a third-party application service provider and leased to the enterprise, as an on-premise or hosted service. This is also often referred to **s***oftware-***a***s-***a***-***s***ervice* ([SaaS](https://www.webopedia.com/TERM/S/SaaS.html)) or Web-based applications.

## *Trends in Enterprise Apps*

Another trend in enterprise applications is the move to [cloud computing](https://www.webopedia.com/TERM/C/cloud_computing.html), where the enterprise moves some or its entire infrastructure to the cloud -- a type of Internet-based computing, where services are delivered to an organization's computers and devices through the Internet as an on-demand service. Some enterprises may also choose a hybrid solution where cloud applications are integrated with on-premise systems.

Some of the more common types of enterprise applications include the following:

* automated billing systems
* [payment processing](https://www.webopedia.com/quick_ref/ecommerce_shopping_process.asp)
* [email marketing systems](https://www.webopedia.com/quick_ref/small_business_marketing.asp)
* content management
* call center and customer support
* [Customer Relationship Management](https://www.webopedia.com/TERM/C/CRM.html) (CRM)
* [Enterprise Resource Planning](https://www.webopedia.com/TERM/E/ERP.html) (ERP)
* Business Intelligence
* [Business Continuity Planning](https://www.webopedia.com/TERM/B/Business_Continuity_Planning_BCP.html) (BCP)
* HR Management
* [Enterprise Application Integration](https://www.webopedia.com/TERM/E/EAI.html) (EAI)
* enterprise search
* messaging and collaboration systems.

## *Enterprise Application Requirements*

Like any modern application, an enterprise application must be reliable, perform well, provide an intuitive and efficient user interface, and so on. But beyond these common qualities, it can be characterized by three specific attributes.

An enterprise application is:

**Large**

A multi-user, multi-developer, multi-machine, multi-component application that can manipulate massive data and utilize extensive parallel processing, network distributed resources, and complex logic. It can be deployed across multiple platforms and inter-operate with many other applications, and it is long lived.

**Business Oriented**

Its purpose is to meet specific business requirements. It encodes business policies, processes, rules, and entities, is developed in a business organization, and is deployed in a manner responsive to business needs.

**Mission Critical**

An enterprise application must be robust enough to sustain continuous operation. It must be extremely flexible for scalability and deployment, and allow for efficient maintenance, monitoring, and administration.

These qualities clearly make the task of enterprise development extraordinarily challenging, and the trend is toward rapidly increasing demands. The rapid improvement of computer hardware and software, combined with global economic competition — and opportunities — have created an environment in which business systems must respond quickly and deliver unparalleled levels of performance. As these demands continue, developers must automate even more of their businesses, build their software even faster, serve more and more users, and process a rapidly growing mass of data.

Aside from these challenges, the power, complexity, and rate of change of the technology used in building these corporate solutions makes efficient development ever more difficult.

To design an enterprise application you must consider and balance an enormous array of application requirements, such as:

* Its business goals.
* How soon it must be delivered.
* Its budget.
* How many people will develop, test, and maintain it.
* How many concurrent users it must support.
* The importance of performance and ease of use.
* The hardware it must run on.
* Where it will be deployed.
* What security is required.
* How long you expect to use it.

Without a systematic way to understand the relationships among these complex and often conflicting requirements, it’s hard to know where to begin. A simpler model can help reduce this complexity, and provide an organized way to design and build applications that chart an optimum course among the many requirements. This model is introduced in the section, Enterprise Application Model.

**2.Challenges of Enterprise Application Development**

Timing has always been a critical factor when organizations adopt new technologies, and the accelerated pace of the information-driven business model puts greater emphasis on response times. Organizations need to be able to project enterprise systems into various client channels, and to do so in a way that’s reliable, productive, and capable of sustaining frequent updates to both information and services. The principal issue is how to keep up with today’s business challenges—whatever they may be—while maintaining and leveraging the value of existing information assets. In this environment, timeliness, productivity, security, and predictability are all absolutely critical to building and maintaining momentum. A number of factors can enhance or impede an organization’s ability to deliver custom enterprise applications quickly and to maximize their value over their lifetime.

Programming ProductivityThe ability to develop and deploy applications is key to success in the information  
economy. Applications must go quickly from prototype to production and must continue to evolve even after they are deployed.  
Productivity is thus vital to responsive application development. Providing  
application development teams with standard means to access the services  
required by multitier applications and standard ways to support a variety of clients  
can contribute to both responsiveness and flexibility.  
The current divergence of technologies and programming models is a destabilizing factor in Internet and other distributed computing applications. Traditional  
Web technologies such as HTML and Common Gateway Interface (CGI) have  
provided a mechanism for distributing dynamic content, while back-end systems  
such as transaction processors and database management systems have provided  
controlled access to the data to be presented and manipulated. These technologies  
present a diversity of programming models: some based on well-defined standards; others on more ad-hoc standards; and others still on proprietary architectures.  
With no single application model, it can be difficult for teams to communicate  
application requirements effectively and productively. As a result, architecting  
applications becomes more complex. What’s more, the skill sets required to integrate these technologies aren’t well organized for effective division of labor. For  
example, CGI development requires coders to define both content and layout of a  
dynamic Web page.  
Another complicating factor in application development time is the choice of  
clients. While many applications can be distributed to Web browser clients  
through static or dynamically generated HTML, others may need to support a specific type of client or to support several types of clients simultaneously. The programming model needs to support a variety of client configurations, with  
minimum effect on basic application architecture and on the application’s core  
business logic.

Integration with Existing SystemsMuch of the data of value to organizations has been collected over the years by  
existing enterprise information systems. Much of the programming investment  
resides in applications on those same systems. The challenge for developers of  
enterprise applications is how to reuse and commoditize these existing information  
assets.  
To achieve this goal, application developers need standard ways to access  
middle-tier and back-end services such as database management systems and  
transaction monitors. They also need systems that provide these services consistently, so that new programming models or styles aren’t required as integration  
expands to encompass various systems within an enterprise.

Freedom of ChoiceApplication development responsiveness requires the ability to mix and match solutions to come up with the optimum configuration for the task at hand. Freedom of  
choice in enterprise application development should extend from servers to tools to  
components. The wide range of J2EE compatible solutions available today and in  
the future ensures the maximum freedom of choice.  
The availability of choices among server products gives an organization the  
ability to select configurations tailored to their application requirements. It also  
provides the ability to move quickly and easily from one configuration to another  
as internal and external demand requires.  
Access to the appropriate tools for the job is another important choice. Development teams should be able to adopt new tools as new needs arise, including  
tools from server vendors and third-party tool developers. What’s more, each  
member of a development team should have access to tools that are most appropriate to their skill set and contribution.  
Finally, developers should be able to choose from a ready market of off-theshelf application components to take advantage of external expertise and to  
enhance development productivity

Response to DemandWhen designing large-scale distributed applications, both availability and scalability  
are key considerations. The more easily and automatically that an application can  
handle changes in use patterns and system configurations, the better. Systems that

require any redesign, recoding, or redeployment to achieve either availability or  
scalability will limit flexibility and diminish expected performance.  
To scale effectively, systems need to be designed to handle multiple client  
interactions with ease. They need mechanisms for efficient management of system  
resources and services such as database connections and transactions. For highest  
availability, they need access to features such as automatic load balancing and  
failover, without any effort on the part of the application developer. Applications  
should be able to run on any server configuration appropriate to anticipated client  
volumes and to easily switch configurations when the need arises. Support for  
clustered application deployment environments contributes to achieving many of  
these goals

Maintaining SecurityMore than ever, information systems security is on the minds of IT managers and  
system architects. That’s because protecting information assets to maximize their  
value can jeopardize that very value. Traditionally, IT departments have been able to  
maintain a relatively high level of control over the environment of both servers and  
clients. When information assets are exposed in less-protected environments, it  
becomes increasingly important to maintain tight security over the most sensitive  
assets, while allowing seemingly unencumbered access to others.  
One of the difficulties in integrating disparate systems is providing a unified  
security model. Single sign on across internal application and asset boundaries is  
important to creating a positive user experience with the applications. Security  
needs to be compatible with existing mechanisms. In cases where customers need  
to access secure information, the mechanisms need to maintain high security (and  
user confidence) while remaining as unobtrusive and transparent as possible.

The Client Tier

FROM a developer’s point of view, a J2EE application can support many types of  
clients. J2EE clients can run on laptops, desktops, palmtops, and cell phones. They  
can connect from within an enterprise’s intranet or across the World Wide Web,  
through a wired network or a wireless network or a combination of both. They can  
range from something thin, browser-based and largely server-dependent to something rich, programmable, and largely self-sufficient.  
From a user’s point of view, the client *is* the application. It must be useful,  
usable, and responsive. Because the user places high expectations on the client,  
you must choose your client strategy carefully, making sure to consider both technical forces (such as the network) and non-technical forces (such as the nature of  
the application

**Client Considerations**Every application has requirements and expectations that its clients must meet, constrained by the environment in which the client needs to operate.  
Your users and their usage patterns largely determine what type of client or  
interface you need to provide. For example, desktop Web browser clients are popular for e-mail and e-shopping because they provide a familiar interface. For  
another example, wireless handheld clients are useful for sales force automation  
because they provide a convenient way to access enterprise resources from the  
field in real time. Once you have decided what type of interface you need, you  
should design your client configuration with network, security, and platform considerations in mind.

**Network Considerations**J2EE clients may connect to the enterprise over a wide array of networks. The  
quality of service on these networks can vary tremendously, from excellent on a  
company intranet, to modest over a dialup Internet connection, to poor on a wireless  
network. The connectivity can also vary; intranet clients are always connected,  
while mobile clients experience intermittent connectivity (and are usually online for  
short periods of time anyway).  
Regardless of the quality of service available, you should always keep in mind  
that the client *depends* on the network, and the network is imperfect. Although the  
client *appears* to be a stand-alone entity, it cannot be programmed as such because  
it is part of a distributed application. Three aspects of the network deserve particular mention:  
• Latency is non-zero.  
• Bandwidth is finite.  
• The network is not always reliable.  
A well-designed enterprise application must address these issues, starting  
with the client. The ideal client connects to the server only when it has to, transmits only as much data as it needs to, and works reasonably well when it cannot  
reach the server

**Security Considerations**Different networks have different security requirements, which constrain how  
clients connect to an enterprise. For example, when clients connect over the Internet,  
they usually communicate with servers through a firewall. The presence of a firewall  
that is not under your control limits the choices of protocols the client can use. Most  
firewalls are configured to allow Hypertext Transfer Protocol (HTTP) to pass across, but not Internet Inter-Orb Protocol (IIOP). This aspect of firewalls makes Webbased services, which use HTTP, particularly attractive compared to RMI- or  
CORBA-based services, which use IIOP.  
Security requirements also affect user authentication. When the client and  
server are in the same security domain, as might be the case on a company intranet, authenticating a user may be as simple as having the user log in only once to  
obtain access to the entire enterprise, a scheme known as *single sign on*. When the  
client and server are in different security domains, as would be the case over the  
Internet, a more elaborate scheme is required for single sign on, such as that proposed by the Liberty Alliance, an industry collaboration spearheaded by Sun  
Microsystems.  
The authentication process itself needs to be confidential and, usually, so does  
the client-server communication after a user has been authenticated

**Platform Considerations**Every client platform’s capabilities influence an application’s design. For example, a  
browser client cannot generate graphs depicting financial projections; it would need  
a server to render the graphs as images, which it could download from the server. A  
programmable client, on the other hand, could download financial data from a server  
and render graphs in its own interface.  
Another aspect of the platform to consider is form factor. Desktop computers  
offer a large screen, a keyboard, and a pointing device such as a mouse or trackball. With such clients, users are willing to view and manipulate large amounts of  
data. In contrast, cell phones have tiny screens and rely on button-based interactions (usually thumb-operated!). With such clients, users can’t (and don’t want to)  
view or manipulate large amounts of data.  
Applications serving multiple client platforms pose additional challenges.  
Developing a client for each platform requires not only more resources for implementation, testing, and maintenance but also specialized knowledge of each platform. It may be easier to develop one client for all platforms (using a browser- or  
a Java technology-based solution, for example), but designing a truly portable  
client requires developers to consider the lowest common denominator. Consequently, such a client implementation cannot take advantage of the various capabilities unique to each platform.

**General Design Issues and Guidelines**

**Presenting the user interface***—*Although a client presents the views to a user,  
the logic for the views may be programmed on the client or downloaded from  
a server.  
• **Validating user inputs***—*Although the EIS and EJB tier must enforce constraints on model data (since they contain the data), a client may also enforce  
data constraints by validating user inputs.  
• **Communicating with the server***—*When a user requests functionality that resides on a server, the user’s client must present that request to the server using  
a protocol they both understand.  
• **Managing conversational state***—*Applications need to track information as a  
user goes through a workflow or process (effectively conversing with the application). The client may track none, some, or all of this information, known  
as conversational state.  
How you handle these responsibilities on your client can significantly impact  
your development efforts, your application’s performance, and your users’ experience. Generally, the more responsibilities you place on the client, the more  
responsive it will be.

**Design Issues and Guidelines for Browser Clients***Browsers* are the thinnest of clients; they display data to their users and rely on  
servers for application functionality.  
From a deployment perspective, browser clients are attractive for a couple of  
reasons. First, they require minimal updating. When an application changes, erver-side code has to change, but browsers are almost always unaffected.  
Second, they are ubiquitous. Almost every computer has a Web browser and many  
mobile devices have a microbrowser.  
This section documents the issues behind designing and implementing  
browser clients

**Validating User Inputs**Consider an HTML form for completing an order, which includes fields for credit  
card information. A browser cannot single-handedly validate this information, but it  
can certainly apply some simple heuristics to determine whether the information is  
invalid. For example, it can check that the cardholder name is not null, or that the  
credit card number has the right number of digits. When the browser solves these  
obvious problems, it can pass the information to the server. The server can deal with  
more esoteric tasks, such as checking that the credit card number really belongs to  
the given cardholder or that the cardholder has enough credit.

<form name="account\_form" method="POST"  
action="default\_test.aspx"**onSubmit="return  
checkFamilyName();"**>  
<p>Family name: <input type="text" name="family\_name"></p>  
<!- ... -->  
<p><input type="submit" value="Send it!" /></p>  
</form>  
**Code Example 3.1** HTML Form Calling a JavaScriptValidation Function  
Code Example 3.2 shows how the JavaScript validation function might be  
implemented.  
<script language="JavaScript">  
<!--  
function checkFamilyName() {  
var familyName =  
window.document.account\_form.family\_name.value;  
if (familyName == "") {  
alert("You didn't enter a family name.");  
return false;  
}  
else {  
return true;  
}  
}

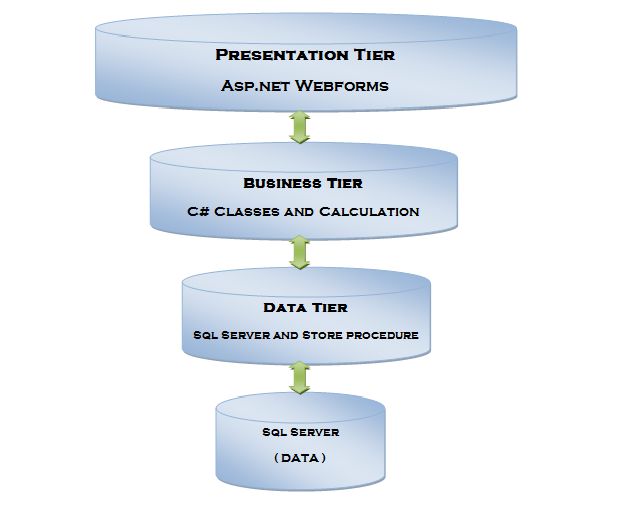
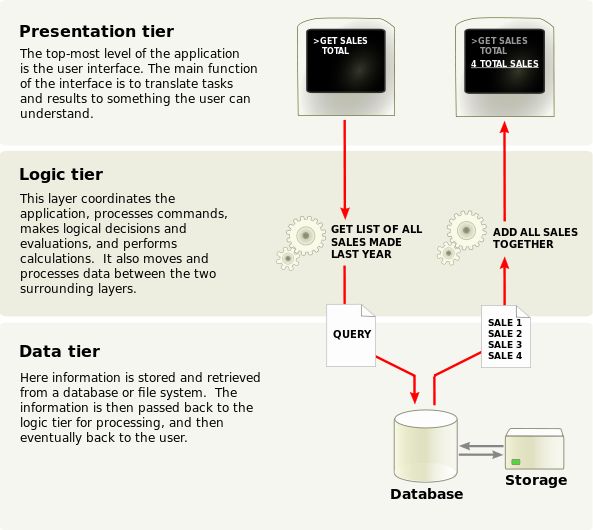
</script>

**Communicating with the Server**Browser clients connect to a J2EE application over the Web, and hence they use  
HTTP as the transport protocol.  
When using browser interfaces, users generally interact with an application by  
clicking hyperlinked text or images, and completing and submitting forms.  
Browser clients translate these gestures into HTTP requests for a Web server,  
since the server provides most, if not all, of an application’s functionality.  
User requests to retrieve data from the server normally map to HTTP GET  
requests. The URLs of the requests sometimes include parameters in a query  
string that qualify what data should be retrieved. For example, a URL for listing  
all dogs might be written as follows:  
http://javapetstore.sun.com/product.screen?category\_id=DOGS  
User requests to update data on the server normally map to HTTP POST  
requests. Each of these requests includes a MIME envelope of type  
application/x-www-form-urlencoded, containing parameters for the update. For  
example, a POST request to complete an order might use the URL

**Managing Conversational State**Because HTTP is a request-response protocol, individual requests are treated  
independently. Consequently, Web-based enterprise applications need a mechanism for identifying a particular client and the state of any conversation it is  
having with that client.  
The *HTTP State Management Mechanism* specification introduces the notion  
of a *session* and *session state*. A session is a short-lived sequence of service  
requests by a single user using a single client to access a server. Session state is  
the information maintained in the session across requests. For example, a shopping cart uses session state to track selections as a user chooses items from a catalog. Browsers have two mechanisms for caching session state: cookies and URL  
rewriting.  
• A *cookie* is a small chunk of data the server sends for storage on the client.  
Each time the client sends information to a server, it includes in its request the  
headers for all the cookies it has received from that server. Unfortunately,  
cookie support is inconsistent enough to be annoying: some users disable  
cookies, some firewalls and gateways filter them, and some browsers do not  
support them. Furthermore, you can store only small amounts of data in a  
cookie; to be portable across all browsers, you should use four kilobytes at  
most.  
• *URL rewriting* involves encoding session state within a URL, so that when the  
user makes a request on the URL, the session state is sent back to the server.  
This technique works almost everywhere, and can be a useful fallback when  
you cannot use cookies. Unfortunately, pages containing rewritten URLs consume much bandwidth. For each request the server receives, it must rewrite

# **3-Tier Architecture in ASP.Net**

This topic explains how to create and implement a 3-tier architecture for our project in ASP.Net.

This article explains how to create and implement a 3-tier architecture for our project in ASP.Net.  
  


## **What is a Layer?**

A layer is a reusable portion of code that performs a specific function.  
  
In the .NET environment, a layer is usually set up as a project that represents this specific function. This specific layer is in charge of working with other layers to perform some specific goal.  
   
Let’s briefly look at the latter situation first.  
   
**Data Layer**

A DAL contains methods that helps the Business Layer to connect the data and perform required actions, whether to return data or to manipulate data (insert, update, delete and so on).  
 

## **Business Layer**

A BAL contains business logic, validations or calculations related to the data.  
  
Though a web site could talk to the data access layer directly, it usually goes through another layer called the Business Layer. The Business Layer is vital in that it validates the input conditions before calling a method from the data layer. This ensures the data input is correct before proceeding, and can often ensure that the outputs are correct as well. This validation of input is called business rules, meaning the rules that the Business Layer uses to make “judgments” about the data.  
 

## **Presentation Layer**

The Presentation Layer contains pages like .aspx or Windows Forms forms where data is presented to the user or input is taken from the user. The ASP.NET web site or Windows Forms application (the UI for the project) is called the Presentation Layer. The Presentation Layer is the most important layer simply because it’s the one that everyone sees and uses. Even with a well structured business and data layer, if the Presentation Layer is designed poorly, this gives the users a poor view of the system

## **Advantages of a 3-Tier Architecture**

The main characteristic of a Host Architecture is that the application and databases reside on the same host computer and the user interacts with the host using an unfriendly dumb terminal. This architecture does not support distributed computing (the host applications are unable to connect to a database of a strategically allied partner). Some managers find that developing a host application takes too long and it is expensive. These disadvantages consequently led to a Client-Server architecture.  
  
A Client-Server architecture is a 2-Tier architecture because the client does not distinguish between Presentation Layer and Business Layer. The increasing demands on GUI controls caused difficulty in managing the mixture of source code from a GUI and the Business Logic (Spaghetti Code). Further, the Client Server Architecture does not support enough the Change Management. Let us suppose that the government increases the Entertainment tax rate from 4% to 8 %, then in the Client-Server case, we need to send an update to each client and they must update synchronously on a specific time otherwise we may store invalid or incorrect information. The Client-Server Architecture is also a burden to network traffic and resources. Let us assume that about five hundred clients are working on a data server. Then we will have five hundred ODBC connections and several ruffian record sets, that must be transported from the server to the clients (because the Business Layer remains in the client side). The fact that Client-Server does not have any caching facilities like in ASP.NET causes additional traffic in the network. Normally, a server has better hardware than the client, therefore it is able to compute algorithms faster than a client, so this fact is also an additional argument in favor of the 3-Tier Architecture. This categorization of the application makes the function more reusable easily and it becomes too easy to find the functions that have been written previously. If a programmer wants to make further updates in the application then he can easily understand the previous written code and can update it easily.