

# 1

## Building an ASP.NET Website

In this book we are going to build a content-based ASP.NET website. This website will consist of a number of modules, which will all fit together to produce the finished product.

We will build each module in a standard order:

- ❑ Identify the **problem** – What do we want to do? What restrictions or other factors do we need to take into account?
- ❑ Produce a **design** – Decide what features we need to solve the problem. Get a broad idea of how the solution will work.
- ❑ Build the **solution** – Produce the code, and any other material, that will realize the design.

This book focuses on programming. When we talk about design, we generally mean designing the software – we will not be looking at graphic or user interface design.

Your website will not be solving all of the same problems as ours, but many of the modules we build – and the programming techniques we use – are very transferable.

In this chapter we will take a high-level look at the whole site – what it needs to do, and how it will do it.

### The Problem

We will be building a website for DVD and book enthusiasts. In outlining the site's problem, we need to consider the purpose and audience. In real life this stage would be business oriented – taking into account things like advertising demographics, competition, and availability of funding. These processes need to be analyzed rigorously, but we will leave all that to the managers.

Our site will cater for lovers of books and DVDs. It will provide useful content and try to build community. Our visitors will want to read about these things, and contribute their opinions, but each visit will be fairly short – this will not be a huge database in the style of the Internet Movie Database ([www.imdb.com](http://www.imdb.com)). It will be funded by advertising, and will rely on repeated (but fairly short) visits from its readers.

We also need to consider constraints. These are more practical. One of the major constraints that this site faced was the development team – the members would never meet, because they were on opposite sides of the world. This meant that the design must allow one developer to work on sections of the site without interfering with other developers working on different sections. But all of the sections needed to eventually work together smoothly. In most cases the separation between developers will be less extreme, but giving each developer the ability to work independently is very useful. We need to design and build methods to enable this.

Site development never really finishes – sites tend to be tweaked frequently. Another key to successful websites is to design them in a way that makes modification easy. We will need to find ways to do this.

**We will call our site [ThePhile.com](http://ThePhile.com), because it is a site for lovers of books (bibliophiles) and DVDs (DVD-philes). It's also a play on the word 'file', because our website will be a definitive source of information.**

## The Design

We have outlined what our site needs – now let's look at how we can provide it. The main points raised in the problem section were:

- Enable developers to work from many different locations
- Build a maintainable, extendable site
- Build community
- Provide interesting content
- Provide revenue through advertising
- Encourage frequent visits

Let's discuss each of these in turn.

### Working from Different Locations

Our developers need to work on sections of the site with relatively little communication. Our developers are in different countries so face-to-face meetings are impossible. Telephone conversations can be expensive, and different time zones cause problems.

We need to design the system so that developers can work on their own section of the site, knowing that they will not damage the work of others.

A good way to solve this is to develop the site as a series of **modules**, with each module being fairly independent. Of course there will be shared components, but changes to these will be rare and can be done in a controlled way. In this book, we work in modules. We also make frequent use of **controls**. This means that components for a page can be developed independently, and easily 'dropped in' as needed – changes to the actual pages of the site are kept to a minimum.

## A Maintainable, Extendable Site

Most websites have new features added quite frequently. This means that from the start the site needs to be designed to make that easy.

Working in modules and using controls already goes some way towards this. Particularly, using controls means that non-programmers can edit the pages of our site more easily – nearly all they see is HTML code. A control just looks like another HTML tag.

Working in modules means that new modules can be added to the site at any time, with minimum disruption. All modules are fairly independent, so new ones can be added – and changes made – pretty easily.

Each individual module needs to be easy to change. A good way to do this is to work in layers, or 'tiers'. We will be using a three-layer design for most modules. We have a data layer, a business layer, and a presentation layer. Data passes from **data layer** to **business layer**, and from business layer to **presentation layer**, and back again. Each layer has a job to do. Underneath the data layer is a data source, which it is the data layer's job to access.

The data layer obtains fairly raw data from the database (for example, "-10"). The business layer turns that data into information that makes sense from the perspective of business rules (for example, "-10 degrees centigrade"). The presentation layer turns this into something that makes sense to users (for example, "Strewth! It's freezing!").

It's useful to do this, because each layer can be modified independently. We can modify the business layer, and provided we continue to accept the same data from the data layer, and provide the same data to the presentation layer, we don't need to worry about wider implications. We can modify the presentation layer to change the look of the site, without changing the underlying business logic.

This means we can provide versions of the site for different audiences. We just need new presentation layers that call the same business objects. For example, providing different languages: "Zut alors! Comme il fait froid!", "Allora, è freddo!", and so on.

We need methods to get changes we make onto the live site. This could be through FTP uploads, but in many circumstances it is better to work through a web interface.

We will also need tools to administer the other sections – ban problem users, add news articles, and so on. This is all part of providing a maintainable site.

## Community

Sites generally benefit from allowing readers to contribute. Because our site is not intended for users to spend hours looking at, our community features must not require a lot of users' time.

There are two ways that we will build our community: through polls and forums. Polls give users the opportunity to give their opinion in a single click – so they require very little time from the user, but can make a site seem far more alive.

Forums enable users to discuss topics with other users. Messages remain in the system, and replies are posted. Readers can leave a post, and then come back later to see if there are replies. This is more appropriate for our purposes than a chat room, which requires the reader to concentrate on the site for the whole duration of the chat.

Community can really give a site a life of its own. Over time, strong characters, heroes, and villains emerge. Many sites depend entirely on community, and become extremely popular – for example [www.plastic.com](http://www.plastic.com).

For any of this to work, we need to identify users and provide them with unique logons. So our system will need some form of user accounts system.

## Interesting Content

The content most relevant to our users will be movie- and-book-related news and reviews. This content tends to be highly relevant for a short period of time: after a story has broken, or immediately after a release. Our site will need tools to manage news in this way.

Another way to provide interesting content is to get somebody else to provide it! This is part of what we're doing with our community section. Part of the purpose of building community is to get people contributing content.

## Advertising

Advertising generates revenue (or in some cases it is used to exchange banners with other sites). We need to display adverts, and record data about how often each advert has been displayed and clicked on.

We also need to gather information about what the users of the site like, so we can target our advertising content. Polls and forums can provide us with useful information when finding products to advertise.

The biggest sites target individual users based on their demographic and any other information gathered about them (for example, Yahoo! and Amazon.com target advertising and product recommendations to the demographic and buying habits of each user). Our site already has a fairly narrow target demographic, and is not particularly big, so we don't need to do this.

## Frequent Visits

A good site will make people want to return. If the content is compelling, and there's plenty of discussion going on, then people visit again and again.

It's still a good idea to remind users from time to time. We want to draw attention back to the site, even when the user isn't viewing it. One way we'll be doing this is through an e-mail newsletter, which gives users useful information and subtly reminds them to visit the site.

We will also build a Windows application that acts as a news ticker, with automatically updating news headlines. Users can click a headline to view the full story on the site.

## The Solution

We've seen what we want the site to do, and sketched out some rough ideas of how we might provide it. Now we'll look at how to build our solution. This really encompasses the whole of the book. Here we'll look at how each chapter relates to our initial problem and design.

## Working from Different Locations

In the next two chapters, we will provide a framework for development. This will lay down coding standards, and a framework for organizing the modules into folders and Visual Studio .NET projects.

We will decide what namespaces we will use for each module, and all the other things that will make team working as hassle free as possible. We will also develop some initial UI features to use across the site, promoting a unified feel. These include a header, footer, and navigation control, and stylesheets.

## Building a Maintainable, Extendable Site

Chapters 2 and 3 will also set us on the road to a maintainable site. We will develop base classes, giving each new module a solid foundation to build on.

We will develop a web-based file manager in Chapter 4. Through this we can download and upload files, create new ones, move them, change their attributes, and even edit files online with a built in, web-based text editor. If you've ever wanted to provide file upload facilities, offer source code for download, or provide online editing tools then this is the place to look!

Most of the modules we develop will have administration features. For these to be useful, we need to identify administrators. In Chapter 5 we will develop a user accounts system. Using this, we can collect user information and give different users different privileges. Our final site will support full role-based security, with login details stored in a SQL Server database.

## Providing Interesting Content

In Chapter 6 we create a news management system. This will enable our administrators to add and edit news articles, receive and approve suggested articles from readers, and place new articles in categories, and of course, it lets users read the news. We will create a control so that we can easily display headlines on any page that we like.

The news system will be flexible enough to also cover reviews, which will eventually form the core of our site.

### **Managing Adverts**

Advertising will be covered in Chapter 7. We will develop a system to display adverts, and log impressions (when an ad is displayed) and hits (when an ad is clicked). This will allow us to create reports from this data to give to advertisers.

There will be admin facilities to create adverts, select how frequently they should be displayed, and start and end campaigns.

### **Encouraging Community**

Chapter 8 will cover our voting system, and forums will be covered in Chapter 10. The voting system will allow administrators to create new questions to vote on. Answers will be recorded and displayed, and an archive of old results maintained – accessible from a standalone Windows application. We guard against multiple votes from the same user by using cookies and IP number.

The forums system will let each user choose an avatar image to represent them, and start posting. Discussion will be organized into categories, and within them there will be various topics. Users can post new topics, and reply to existing topics. We use regular expressions to allow formatting tags in messages, but prevent images or JavaScript.

### **Getting Repeat Visitors**

As well as providing all this great content, we will include two features specifically for getting visitors back to the site.

The first is covered in Chapter 6 where we look at news. We will develop a web service that exposes our news headlines. We will then build a Windows client that displays the headlines, updating itself regularly. Clicking a headline will open a browser on the correct page for the full story.

The second is covered in Chapter 9. We will create the facility for visitors to subscribe to receive e-mail updates from us. Once they are subscribed, we send a mail out regularly to encourage repeat visits. This mail will include highlighted news and features, and links back to the site. We will develop a system that enables administrators to create plain text and HTML messages. We then develop a mailing list admin module for creating subscription forms for new mailing lists, administering list members, adding newsletters, and managing subscriptions. Messages can include custom tags so that each list member receives an e-mail tailored to their own details.

### **Deploying the Site**

Although we haven't mentioned it before, we will eventually need to move the site from our production machine to the live server. This can be a complex task, because we need to separate the files needed for the site to run from the source code files that we only need for development. We will look at this in Chapter 11, and see how Visual Studio .NET gives us tools to make the process easy.

## Summary

We're now ready to look at the site in detail. Before reading the following chapters, it's worth getting hold of the code download and seeing how the final site fits together. This book does not describe every detail of the website, and it will be a lot clearer if you look at the final site first.

**The code and database are available from [www.wrox.com](http://www.wrox.com). Once you've downloaded and unzipped it, look at the readme file to see how to get it working in Visual Studio .NET. You will get far more from the book if you look at the project *before* reading on.**

In the next chapter we will start to build the foundations for the rest of the site.