neatComponents: Guiding Principles & Technical Choices

Introduction

neatComponents is a comprehensive website development platform and management system.

This document provides a brief overview of the underlying architecture, along with some of the reasoning behind the architectural principles, to explain the neatComponents 'mindset'.

neatComponents is all about hiding the technical aspects of web-based systems away from the user, and indeed, away from the system designer, in order to make the process efficient and profitable.

Some guiding principles

The system is based around some fundamental guiding principles, designed to facilitate the objectives of making website design easier (ie less technical) and quicker (to be more profitable). These are defined to mean that:

- No coding should be needed
- No FTPing should be needed
- No direct access to the server should be needed

'No coding' means that the system needs to provide, through a graphical interface, options and settings that allow all the usual business logic found in websites to be configured, without the use of Java or VB or any other coding or scripting language by the designer.

'No FTPing' and 'No direct access' go together; as they serve to both save the designer from technical aspects, and equally importantly, protect the server from inadvertent damage. As an example of how this affects the functionality of the system, it means that the system builds in much of the admin interface found in the Web server (IIS in this case), and presents this in a controlled fashion to the site designers.

Another guiding principle that underlines the depth of the system is that the self-imposed limitations that the above principles impose shouldn't impede the designer in their quest to create unique and distinctive sites. Thus the system can't simply present a few 'templates' designed outside of the system, and force the designer to choose from them; rather it must provide a rich graphical layer that can be styled and controlled.

All that is focused on the designer creating individual sites. However the system has a wider scope to leverage some extra cost savings for the designer (or their employer). So it includes a principle of 'zero-administration', meaning that any task on the business administration side that could be automated should be included. In practice this includes two areas: The first is client

billing – where the system manages a credit balance along with low balance warning emails, and interfaces with bank credit-card systems to handle payments. The second is client sign-up – not appropriate in all situations, but useful where a provider is offering template sites that the designer has created, and you need the client to be able to sign up, maybe get a 30 day free trial, and then be moved on to the client billing system.

It is notable that the system is server-based, rather than running on individual designers' machines. Whilst it's true that some popular design tools are based on designers' machines, and then upload finished pages to a web server, they have difficulty when it comes to anything but the most trivial of content management. That approach also seems to create an artificial distinction between site designers and site users, whereas in reality a large system with fine-grained delegation may have thousands of users with permission to make changes. You would not want the expense of individual developer licenses for each of them, nor the administrative burden of keeping them all up to date. In scenarios where you start opening data manipulation up to clients, either in an extranet or even in an e-commerce system where you invite feedback on products, the distinction between a designer and a visitor becomes blurred, and its best to equip everyone with the tools to make any change. Since with neatComponents the only tool that is needed is a web browser, this is cost-effective and scalable.

The lack of any requirement to have coding skills, and its emphasis on business process optimization mean neatComponents is most suited to graphic web designers, who want to create sites that don't just look good, but also include technical functionality; and directors of web design companies who want to grow without taking on extra staff, and thus become (more) profitable.

Platforms, languages and databases

There are numerous open source projects that can be combined to produce a basic content management system; however they do not offer an attractive foundation for a commercial offering, with the main issues being related to code maintainability and intra-module interfacing and most importantly - timely Support and Directed Innovation (that's problem solving, bug-fixing and new features). neatComponents is closed-source, and you benefit every step of the way[Q].

So, with two exceptions, the system is entirely coded in-house. The exceptions are the database layer, where the architecture supports MySQL (with SQLServer and Oracle support to follow), and the Text Editor.

neatComponents supports the Microsoft IIS web server, running on Windows 2003 Server, however the architecture is designed to be portable to Linux or NetBSD. This means that the system does not use Microsoft's dot Net programming layer, and is coded in VB (compiled to DLLs), which is relatively easy to migrate to Java. The system does not use J2EE as the processor overhead imposed by that architecture would have compromised the scalability of the system on affordable hardware.

At the client-side, modern browsers enable AJAX principles to be used to deliver a smooth and interactive design environment, and the system has embraced these technologies wherever possible, using XML, background HTTP transfers and DOM manipulation.

Web pages – and how they are made

Whilst neatComponents does many things besides, the basic fabric is a content management system (CMS), which is to say a system where the content that appears on pages is stored in a database, can be modified by suitably authorized people, and is then displayed to users.

The architectural choice here is whether to get the CMS to write HTML files to disk in public facing web-space, and then allow visitors to browse that: updating the pages as needs be, or whether the system should create the pages on-the-fly each and every time the user visits the page.

From a performance standpoint it seems straightforward. Pages are viewed far more often than they are changed, and it takes longer to construct the pages from the database than to simply deliver them from pre-constructed files on disk. So why does the system do the opposite, and create a system where the pages are constructed each and every time? The answer is 'users'.

The static file approach is fine if you have just two sets of users: admins, who can change things, and visitors, who see them. However, if you have another user-group, say 'clients', then you need two sets of the pages, as the same page may look different depending on who you are logged in as. In fact, since an individual can be a member of more than one user-group, if you have two groups, he could be in one of three states (one group, the other, or both). And if you have more groups, the problem gets worse, exponentially. Before long, the permutations mean the hard disk space required becomes impractical. With just ten user groups that means 3,628,800 (10 x 9 x 8 x ...) different copies of the site would be needed to be stored - and updated whenever a change is made.

Building pages from the database each time a copy is requested still leaves the performance issue to solve, so the system implements a multi-level caching system. Thus the first time a page is requested after a data change it might take a bit longer than normal, but subsequent requests will be swifter as intermediate results can be obtained 'for free'. As responsiveness is always important, the system monitors how long it takes to construct pages, and logs any particularly slow ones for further analysis and optimization.

Storing data

The basic fabric of neatComponents is a content management system, where individual pages are edited online, and the changes are stored in a database. Five years ago that might have sounded advanced, but today that is standard functionality across the sector. What sets neatComponents apart is that that is not the goal; it is just the starting point.

The true value is apparent when you start working with structured data. A product catalogue, or a list of employee profiles. Scenarios where you would traditionally utilize a programmer familiar

with a relational database (MS Access, Oracle, etc), and integrate their back-end systems with a web front-end.

Reviewing neatComponents' guiding principles: 'no code' and 'no access to the machine', this is not a trivial extension. However, the system includes the ability to add special pages that encapsulate the essence of various relational database concepts. (Since these pages are going to do far more that just display text, they are called 'Components' instead of pages).

Thus where a database is made up of Tables, neatComponents provides a component (called a Form) that allows you to define a record structure, and fill in forms (hence the 'Form' name) to populate those records.

Where a database has Queries that allow you to retrieve grids of records matching certain criteria, so does neatComponents. These queries can then be embedded on to pages to display structured information in context, formatted appropriately.

There is also a series of components that provide specialized views of the data, and ways to query it: query-by-example forms, search engines, category views, etc, and numerous advanced features to make building websites that encapsulate all this functionality into a rapid and precise development process.

Behind the scenes, the system maps all the data manipulation on to the underlying database, creating tables and fields on the fly, and protecting the user and the site designer from any of the technical complexity that is normally required whenever databases are involved.

Further information

For more information, including a complete description of what neatComponents functionality, see the neatcomponents.com website.