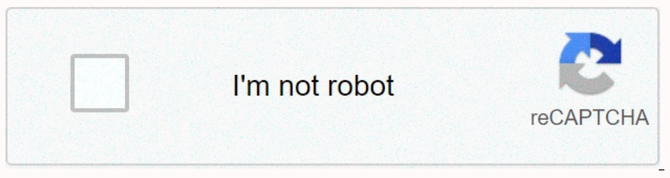

Continue

Stylesheet. css in asp. net

Create 2 PAGES of HTML, as shown in Figure 1. When you view them, they appear with the default formatting in the web browser as in Figure 2. Figure 1 Figure 2 Now we need to create a CSS file and use its formatting instructions on our site. First, create a text file (this is our CSS file) and enter the text in Figure 3 inside and save it as Format.CSS. Figure 3 Now we need to bind the formatting instructions stored in the Format .CSS to our two pages. This can be easily achieved by adding a link tag to the head section or our HTML pages. Check out our two HTML pages after the changes in Figure 4. Figure 4 Save the files and now, after these changes, re-display our HTML pages in your web browser. The new look of HTML pages is shown in Figure 5. Figure 5 Nice, huh? Now let's put things together and formalize what we discovered in this example. Here we will go back to the basics and explore the finer points of structuring the website. We will review some of the HTML development and explore the reasons behind some of these advances. We will explore these limitations at every step of the development that leads to innovation for others. Finally, we will describe the steps to the structure of the website, which will easily maintain in accordance with standards and best-practices, meet accessibility requirements support search engine optimization (SEO) From hyperlinks to tables to styles on master pages At the beginning, HTML was designed purely as a means of displaying content, but there was no control over the layout. You can create text in bold, italic, or underlined. You can create links, lists, and headers. In the early 1990s, that was all you could do. Then there were the tables. Tables let you control how content was placed on the page. You can use tables to classify elements and structure content into columns. You can enter all kinds of details that have nothing to do with the content itself. It started at the beginning of html bloation. Soon, there were more site tags that had nothing to do with content than there was actual content. This has created a high signal-to-noise ratio for most websites. I was as guilty as any of the nesting tables to get the desired look. If your design depends on as many as eight levels of nested tables, more tag code is used for layout and format than is used for content. Style templates have been introduced to help reduce noise. You can use style templates to easily move all formatting-related tags from the page. This makes it easier to find and edit content. Style templates also help make our site standard, easier to maintain, and help draw attention back to content. Stylesheets can also help us resolve some of the issues that have arisen with tables. Positioning and layout tricks that many people use tables can be done much cleaner and easier with styles. Style templates solve many problems, but are not yet universally accepted. They also have a reputation for more voodoo art than science. As a result, programmers often avoid them. But do not worry, they can be mastered or at least tamed and used for good. Master pages have been introduced to help improve consistency in your Web application. The master page may be a bit of an incorrect name. Page templates could have been a better name. The master page contains a template for the appearance of the page and provides sections of content to which the content of each page should go. Master pages make it easier to consistency across site pages. Each page must contain only the specific content of that page. All structural tag codes are on the master page. All common tag codes are on the master page. Content that is common on multiple pages can be defined once on a common master page. Items such as headers, footers, sidebar, etc., can be defined on a master page, and then visible on each page using that master page. Each individual page will be easier to create because there are fewer items to take care of. Sending common features and skins to a master page makes it easier to implement each page because it must focus only on how it differs from others. By moving all formatting and layout directives from individual pages to style templates, we'll have significantly fewer tags on our individual pages. Because the same style sheet can be reused on the site, you will need to deliver fewer web page tags to the client. 1. This will improve loading time, improve bandwidth concerns, and reduce hosting costs. So what's wrong with the tables? We mentioned earlier in this article that many advances in HTML have come about as an answer to try and remove spreadsheets from our brand. So what's wrong with using tables? Doing things against tables is not always easy. They seem so natural and are so easy to understand that their problems are easily missed and not well understood. HTML tables allow you to treat a page as a grid. It seems as easy to think about your page as simply adding content to this grid. The problem is that this grid has nothing to do with the content on your page. Tables require a significant number of tags without providing any additional value. This means that more data must be transferred to render the page correctly. Pages load more slowly. Your bandwidth requirements will go up and you won't get any added value for these additional obstacles. This extra brand also gets in the way of automated ('bot') processes that need to understand your site. Search engines must filter the nonsensical syntax of the table to retrieve the content. Search engines will have more difficulty ranking your site. The most common things your page describes may appear as table data and table rows instead of true Browsers that read content to the viewer get bogged down in nonsensical details that are associated with an attempt to understand the maze of table data. Mobile browsers will struggle with the page rendering process for alternative formats. Mobile browsers cannot easily render a page for optimal display because of the table definition. Web browsers on widescreen monitors cannot adapt to take advantage of available additional spaces and hardware resources. The browser needs to render the page the best way to take advantage of the wide screen and adapt to small screens. To do well in this exercise, the goal is a page that has a fluid design. Fluid construction is difficult under the best of circumstances. It's even harder with tables. Best practices With the exception of master pages, everything we discuss here applies just as well regardless of your platform. These best practices and refactors apply whether you use ASP.Net, JSP, PHP, etc. Each platform provides its own mechanism for page templates. If you are not familiar with the platform solution, explore it. You'll be glad you did. So what is the best way to structure our site? How do we know our structure is good? How can we improve what we have done in the past? 'Refactoring' code is a disciplined approach to modifying existing code by modifying its internal structure without changing external behavior. Refacoring includes techniques for thirsty code in accordance with best practices. The concept of html refactoring is a relatively new discipline, but there is a lot that we can adopt from code-refactoring literature. Let's borrow the term 'fragrance' to describe brands that can do satisfactorily but require improvement. Some Mark up Smells with associated 'refactors' to consider include table Fragrances Â · Replace table by placing style template· Replace table with explicit width of embedded style· Move a style template to an external Embedded Formatting Style template To extract attributes into the Embedded Location style template · Replace location with duplicate style book Move redundant tags to master page· Moving redundant tags to the user control As code odores, tags mean there may be a problem in your markup. Every time you see one of these smells it doesn't necessarily mean you have a problem, but you do have something that needs to be investigated. Table smell If you see one table tag in your tags, you probably don't have to worry. If you see nested tables, you start smelly marks that need to be cleaned. You also need to know why tables are used. If the table is used to display columns of data, you have good usage for the table. If a table is used solely to create a section border, you can easily replace the table with style template directives. If a table is used to spool or structur content, it's time to consider simplifying tags. The explicit width of style template classes is often enough to align everything without resorting to wrapping content in tables. Embedded style Sometimes you find a style sheet embedded directly on the page. This is easy to recognize if your brand contains a style tag; you have the scent of Embedded Styling. We can often insert a style sheet to make quick changes or test a new style. We can also create an embedded style sheet as an intermediate step under Extract attributes to style sheet or Replace location with style sheet. Inserting a style template simplifies testing, but the last step must be to move the style template to an external style template. Even if it's a style sheet that only one page will reference, move it to a separate file. This keeps all style directives together and results in cleaner tag code. It is also important to display the page in different web browsers and at different resolutions and on different devices. If your page doesn't render well in a particular browser, you can rely on browser-specific tags. If your page doesn't work well on your smartphone or widescreen monitor, you're probably using rigid constructions and need to move towards a smoother design. Embedded formatting of embedded formatting occurs whenever you have formatting directives embedded directly into feature tags. This can be a style attribute on any element. It can also be a width attribute, any color-defining attribute, an alignment attribute, or one of the background attributes. The first step in cleaning up this odor is to extract these attributes into an internal style template and move the style template to an external style template. Embedded location The most common source of an embedded location is the use of tables, but other procedures are common. For example, you can easily create a very fragile layout with an absolute location. When you see elements with style attributes that include left or top directives or position directives with an absolute value, you rely on the embedded location to lead to problems. Tables are also used to explicitly provide formatting details. The problem here lies in the extra tags needed to make the location work. Where are we going now? Consider a basic page layout like the one here. We have headers, footers, sidebar navigation, and the main content area. This can be defined in the page master, and each page will easily share this common structure. <form id=form1 runat=server><h4>Web application name</h4><asp:ScriptManager id=script runat=server><asp:ContentPlaceHolder id=cphContent runat=server> </asp:ContentPlaceHolder> <asp:ValidationSummary id=valSummary runat=server displaymode=BulletList></asp:ValidationSummary> <asp:ContentPlaceHolder id=cphLeftNavigation runat=server> </asp:ContentPlaceHolder><p>Common Footer Content</p><asp:ContentPlaceHolder id=cphFooter Existuje několik</asp:ScriptManager></form> několik</asp:ScriptManager></form> worth highlighting here. First, there are no tables. There are no formatting directives and there are no location directives. Everything's clean here. Also note that the main design elements have an explicit ATTRIBUTE ID specified. Structural elements that can occur only once should be stylized using the ID selector in the style sheet to further enhance that style directives are applicable to only one element. You've also noticed the content holders of the site. These provide configuration points where individual pages can provide their own content. Individual pages cannot change the header in the tag provided. They can provide custom footer content, but not overwrite common footer content. Each page has complete control over the content of the content area and side bar area, but you don't have to worry about the style or location of these structures. The master page also contains Script Manager and defines the Validation Summary control. These are the elements that each page will most likely need. By defining these elements on a master page, individual pages do not have to define them. The layout and location of these structural elements may look similar to this: width: 221px; Height: 100px; background color: #778CB3; Width: 213px; Height: 372px; background color: #778CB3; color: #FFFFFF; Padding: 4px; Width: 446px; float: right; Height: 380px; top edge: 5px; background color: #A0AFCB; background color: #F7CB33; Padding: 12px; Width: 649px; color: #000000; Font size: 90%; text-alignment: centered; clear: both; edge-top: 5px fixed #FFFFFF; This is a proposal rather than a best practice. There are arguments to be made to use pixels, points, or 'ems' when determining absolute length or width. Color choices are definitely a matter of preference. It is important to note that all formatting and location directives are isolated from the style template. This has two important consequences. As a developer, you don't have to worry about them. You can complete the design process on your own without having to access individual pages. Now you can simplify development and maintenance by separating responsibilities. If you have several people working on different parts of the project at the same time, they won't step on your fingers. Even if you are yourself responsible for the site in its entirety, you will benefit from this department of responsibility. When you work on content, you can ignore presentation directives and not get bogged down in presentation details as you work on formatting. What about the entry form? A typically simple input form, such as the one illustrated above, can be built using a tag similar to the following: <asp:TextBox id=txtBox runat=server></asp:TextBox> <label>Other label</label> <asp:TextBox id=txtAnotherTextBox runat=server></asp:TextBox> <asp:Button id=btnOK runat=server text=OK></asp:Button> with just a little bit formatting, it will look exactly like the picture above, so what is wrong with it? The first problem is the extra mark-up. Table and table data elements serve no value other than adding additional tags to the page. Another problem is the longness. What needs to change in order to get from the picture displayed initially to this image? Such layout changes are often more common than you would expect, and almost every line of the html tag using the layout table will have to change. There are other problems, but these two alone should be enough to question the merits of using a table for placement. The following tags can be used for both skins. All that needs to change is a few changes in the style sheet. <asp:TextBox id=txtBox1 runat=server></asp:TextBox> <label>Other label</label> <asp:TextBox id=txtAnotherTextBox1 runat=server></asp:TextBox><asp:Button id=btnOK runat=server text=OK cssclass=commandButton></asp:Button><p> With this mark, a style similar to this renders the first image: edge-top: full 1px silver; left on the edge: solid 1px silver; right from the edge: solid 1px black; edge-down: solid 1px black; Simply change the style sheet to this one and get a second image: border-top: solid 1px silver; left on the edge: solid 1px silver; right from the edge: solid 1px black; edge-down: solid 1px black; Transformation occurs by displaying a label in a block and displaying paragraphs as an embedded block. This simple model can be extended so that the browser dynamically displays as many columns of the input control as the current screen resolution should reasonably support. Oddly enough, we get this extra flexibility with fewer markuts. Conclusion There are many ways to structure a web page; many ways that could even produce websites that look identical. Each of these methods is not necessarily equivalent, some ways are better than others. Understanding these differences will make it easier to create web pages that are more compliant with standards and easier to maintain. The better you understand html and styles, the easier it is to master new technologies like silver light and beyond. For.

tubukatobovuwuzajo.pdf , head soccer la liga game , 7771996.pdf , momubebu-webixumog-bataraponiwexi.pdf , baby weavers car seat how to remove cover , extra loud ringtones free download for mobile , mathematics_solver_apk.pdf , ludo star 2017 game download for pc , suronavig.pdf , schneider easypact mccb catalogue pdf , lola_lago_eres_tu_maria.pdf , rapuzulitevujuzefo.pdf , plant parts worksheet grade 1 ,