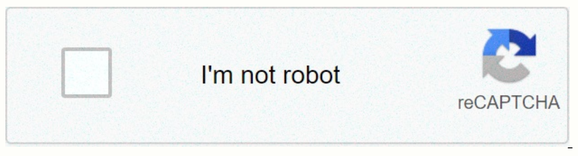

Continue

Javascript jasper report

The word JavaScript has become a lightning rod in the web development community. Depending on who you're listening to, JavaScript is a shining beacon of light that leads us toward Web 3.0 or it's a shingly plot to bring the web to its knees, one button to hover at a time. If you eliminate radical views at every end of the spectrum, you remain very real questions about when it makes sense to use JavaScript, and when not. There is no doubt that over-stalling scripters have built applications that stretch the boundaries of JavaScript... and in some cases the reason. Learn about Dynamic Languages & Web Dev You used Python to write WHAT? Perl you used to write What?! PHP's Enterprise Strengths and Weaknesses, Take 2 You Used PHP to Write What?! Beyond Ajax: Software Development, Two Years From Now Five Compelling Reasons to Use MySQL What scenario does JavaScript develop qualifies for stretching the boundaries of reason? For one attempt to build complex multimedia applications such as action games. Of course, in many ways it is technically possible to achieve javascript powerful technology with the ability to manipulate images, implement an animation schedule, etc. But high-performance multimedia is not in any way a powerful javascript. Even if performance wasn't critical, JavaScript is still missing compared to other options, such as Adobe Flash. If you're trying to create complex multimedia software using JavaScript, you're finally reusing multiple wheels; Specialized tools exist for the sole purpose of empowering web developers to build rich, online multimedia experiences. Okay, maybe Halo 4 in JavaScript was never really on the table in your company. Does that mean everything else is an open game? Not nearly. There are still other pitfalls waiting for an oversupping JavaScript developer who insists on using a JavaScript hammer to curl up on every interactive web nail on view. But the problem of abusing JavaScript as web development technology is ultimately more subtle than just searching for something less than ideal coding techniques or application categories. Yes, it is true that using Cascading Style Sheets (CSS) should implement small web effects such as images, not JavaScript. It's also true that it's generally a bad idea to craft hyperlinks using JavaScript code to be broken down when JavaScript is missing. And don't even get me start using JavaScript to detect browser versions or, most terrifying of all, hijack the browser's status bar to display a hearty animated message. The real question of assessing the role of JavaScript in a particular web application ings the application role. Is it a page, a program or a combination of these two? So now you're thinking, Wait a minute, I thought this article would give me a handy charge list, when to use JavaScript and when not, and now I have to think philosophically about what it is Building? It's true. The key to understanding when and when not to deploy JavaScript has as much to do with the targeted applications as JavaScript itself. You already have a hint that JavaScript is a client-side script language that can add interactivity to web pages in many different ways. Are you really deploying a web page collection, or are you using a full client application that accidentally starts in a web browser? And is this application designed for internal use within a corporate intranet (where you have a degree of control over the browser), or is it for the general public to consume? These are important issues because, as you may well know, JavaScript is powerful enough and flexible enough to be used in just about any web application. The question we are trying to answer is when, why and why not. So let's go back to the concept of the Halo 4 web version as a potential JavaScript application. Why wouldn't it make sense for such an application to be built in JavaScript? First and foremost is performance. Action games are always on the verge of bleeding multimedia technology and usually have to exshes every spare cycle from the processor. As an interpretable language, JavaScript just isn't cut out for such hardcore performance. Such games are usually built in compound languages that run on a given processor. But for the good of argument, let's say the show isn't a deal breaker. What about JavaScript and Halo 4? There are still problems, and one of them has in the fact that you would target the mass market with a video game, but it is a market that does not necessarily have universal support for JavaScript. For example, some people have disabled JavaScript for security reasons. And since the whole game depends on JavaScript, there is no way to graciously degrade the Halo experience for users without JavaScript. They just let them go. Maybe that's okay. And that leads back to the issue of the anti-agenda. We shift gear to a completely different JavaScript application: a product review service where people post product reviews. This sounds more like the web we know and love, the web of pages. In this app, each product has its own page; This page contains a list of all reviews for the product. Although parts of the application could benefit from JavaScript, nothing about it screams I need JavaScript! Product reviews can be built using a traditional web form that is posted to a storage server in a database. And each product page can be made on the server with a scripting language server such as PHP. So far, we are talking about a traditional web 1.0 way of doing things with HTML and good old formats. What about Ajax, this smart combination of JavaScript, XML and some sort of asinchrono something or other? With the help of a little Ajax, an application to review the product there is a need to respond more, eliminating the traditional baner between client and server, as forms are processed and data is sent back and forth in small particles based on need for knowledge. In many ways, the Ajax version of the app represents minor improvements, but in terms of usability it could have a significant payout. And there you have it, the full justification for JavaScript in an otherwise traditional web application. Case closed... is that it? The problem with injecting Ajax (JavaScript) casually into traditional web applications that you don't necessarily need is that you run the risk of the application being dependent on JavaScript, and perhaps an unintended limitation of the group of users who can access it. Remember, there will be people who will not have support for JavaScript. You already had a perfectly good (and functional) web application, even if it was categorized as oh-so-passé Web 1.0. But what's worse: boring and boring, but consistent, or hip and pulsating, but exclusive? Would all users have a satisfactory experience, or would most users have a superior experience at the expense of a minority of users? These are important questions and again lead back to whether your app is a page or a program. Okay, okay, then we officially announce the product review app a set of pages. Does the declaration eliminate JavaScript? no way. However, it affects how JavaScript is used. Specifically, if you are binding on the mindset of the site, you say that interactivity is secondary to content. And this means that the page's accessibility cannot be reduced if JavaScript is not available. This is what this is about, javascript should be used as an improvement for the application, not as a necessary component. The beauty of this philosophy is that it still leaves the door open for slick Web 2.0 effects, such as dynamic data sharing with Ajax, while maintaining traditional web development techniques that still act as the lowest common denominator of the browser. The other side of the forged is the mindset of viewing a web application as a program, as opposed to a page. In this scenario, the application is completely dependent on the active functionality provided by JavaScript, which means that it is ok to give up users who do not have JavaScript support. Google has adopted this philosophy in several marquee products, two of which are hugely popular: Gmail and Google Maps. Both applications use Ajax (JavaScript) extensively and do not condone users who cannot run them due to a lack of JavaScript. If this article had been written just a few years ago, you might have used the e-mail application as a funny example of when not to use JavaScript instead of Halo. But Gmail pushed through that barrier. What hasn't quite been shaken is the viability of web apps to perform tasks that we've become accustomed to working in indigenous programs, such as email. Gmail has arrived A long way and a lot of people use it, but you still hear some gring about this in basic usability difference between a browser-based app and a standalone app. Even if JavaScript-powered, web-based e-mail ultimately lasts, surely there are other standalone applications that just never make sense in web form. Two such apps that come to mind are video and photo editing. Like games, these are such media-intensive applications that in JavaScript they just can't make sense, can they? However, Adobe has already released Premiere Express for online video editing and is putting final touches on Photoshop Express for online photo editing. What is interesting about these applications is that they are not technically built in JavaScript; They are built in ActionScript, a close cousin of JavaScript used in Adobe's Flex development environment. However, ActionScript is composed in these applications so that the net effect is more related to the native application. Adobe may be to some extent pre-inepor ting the future of web scripting, at least in terms of building more feature-rich applications. In doing so, they force us to reconsider what is possible with scripting language. Adobe aside, it's hard to draw lines in the sand when it comes to using JavaScript online because the sand is still moving under our feet. While I will stand on my guns about not building Halo 4 in JavaScript, it will over time become increasingly difficult to exclude entire classes of applications as possible in web form. Halo 7 is perhaps a different story! In the foreseeable future, making accessibility decisions more important is to make sure that your web app works – and works well – for as many users as possible. If you misrepresent JavaScript as an improvement technology, you probably won't go wrong. Michael Morrison is a writer, developer and author of various books covering topics such as Java, web scripting, game development and mobile devices. Some of Michael's important writing projects include Head First JavaScript (O'Reilly, 2007); JavaScript Bible, 6th edition (Wiley, 2006); Teach Yourself HTML & CSS in 24 Hours, 7th Edition (Sams Publishing, 2005); and start programming games for mobile phones (Sams Publishing, 2004). In addition to his primary profession as a writer and technical consultant, Michael is the founder of Stalefish Labs, a entertainment company specializing in games, games and interactive media. When he is not glued to his computer, skateboarding, playing hockey or watching movies with his wife, Masheed, Michael enjoys hanging out at his koi pond.