Sharepoint 2010 Interview Questions And Answers For Experienced Developer

Mastering the SharePoint 2010 Developer Interview: Key Questions and Expert Answers for Experienced Professionals

Landing a senior developer role often hinges on demonstrating deep, practical knowledge of the platform you claim to master. For many organizations, SharePoint 2010 remains a critical part of their infrastructure, even in the age of cloud solutions. Experienced developers are expected to do more than write code; they must understand the architecture, troubleshoot complex issues, and design robust solutions. This article compiles a comprehensive set of SharePoint 2010 interview questions and answers for experienced developer candidates. The focus is not on rote memorization but on the underlying concepts and real-world scenarios that separate a novice from a seasoned professional. Whether you are preparing for an interview or brushing up on your skills, these insights will help you demonstrate genuine expertise.

Core Architecture and Solution Design

Interviewers often start with foundational architecture questions to gauge your understanding of how SharePoint 2010 operates at a systems level. Experienced developers should be able to explain not just what something does, but why it works that way.

What is the role of the Timer Service in SharePoint 2010, and how would you troubleshoot a failing timer job?

The SharePoint 2010 Timer Service (**SPTimerV4**) is the backbone of asynchronous operations. It executes background tasks like workflow timeouts, alerts, and search crawls. An experienced developer will understand that timer jobs are not single-threaded; they run on all servers in the farm, but typically only execute on one server unless specifically designed otherwise. Troubleshooting a failing timer job requires a methodical approach: check the ULS logs for the specific job ID, verify the service account has sufficient permissions on the content database, and ensure the job is not disabled or in a stuck state. You can use PowerShell with `Get-SPTimerJob` and `Start-SPTimerJob` to manually trigger jobs for testing. A common mistake is assuming the timer service is running when it is actually hung; restarting the service or the entire server is sometimes necessary, but always check the logs first.

Explain the differences between a Farm Solution and a Sandboxed Solution in SharePoint 2010. When would you choose one over the other?

This is a classic question that tests your understanding of deployment and security context. A **Farm Solution** is a fully trusted, server-side package (WSP) that can deploy assemblies to the GAC, run code with full trust, and access any resource on the server. It requires farm-level administrator permissions to deploy and can impact the entire farm if it malfunctions. A **Sandboxed Solution**, introduced in SharePoint 2010, runs in a partially trusted environment with limited permissions. It cannot access the file system, call unmanaged code, or write to the registry. The key advantage is that site collection administrators can deploy sandboxed solutions without farm admin privileges, and a failing sandboxed solution will only affect the site collection, not the entire farm. For an experienced developer, the choice is pragmatic: use farm solutions for central administration features, custom timer jobs, and any code that requires full trust. Use sandboxed solutions for site-level customizations like custom fields, content types, and simple web parts, especially in multi-tenant or hosting environments where isolation is critical.

How does SharePoint 2010 handle caching, and what strategies would you implement for a high-traffic publishing site?

SharePoint 2010 offers several caching mechanisms: output caching, object caching, and BLOB caching. Output caching stores the rendered HTML of a page for a specified duration, reducing the load on the database and application server. Object caching caches data retrieved from SharePoint lists and libraries, such as navigation and site data. BLOB caching stores binary large objects (like images and documents) on the front-end web server, significantly reducing database queries. For a high-traffic publishing site, an experienced developer would enable BLOB caching for all image and document libraries, configure output caching with appropriate durations for anonymous users (since authenticated users often need fresh data), and ensure the object cache is populated for anonymous access. Additionally, using a reverse proxy like **ARR (Application Request Routing)** can offload static content. The key is to test caching under load; overly aggressive caching can lead to stale content, while insufficient caching defeats the purpose. Always monitor cache hit ratios using performance counters.

Development, Debugging, and Deployment

Practical development skills are where experienced candidates truly shine. Interviewers will probe your familiarity with tools, debugging techniques, and deployment strategies specific to SharePoint 2010.

Walk me through your process for debugging a custom web part that is causing a yellow screen of

death (YSOD) in SharePoint 2010.

The YSOD in SharePoint 2010, unlike classic ASP.NET, often shows a generic error page unless custom errors are turned off. An experienced developer knows the first step is to modify the **web.config** file for the web application to enable detailed error messages: set `<customErrors mode="Off" />` and `<SafeMode MaxControls="200" CallStack="true" />`. Next, check the ULS logs, which provide the most granular error information. For a web part specifically, you can attach the Visual Studio debugger to the **w3wp.exe** process (the IIS application pool for the SharePoint web application). Since multiple w3wp.exe processes may be running, you need to identify the correct one by its Process ID (PID), which can be found using `iisapp.vbs` or Task Manager. Once attached, set breakpoints in your code. A common issue in SharePoint 2010 is permission problems: if the web part tries to access a resource beyond its current user context, it will fail. Always impersonate using `SPSecurity.RunWithElevatedPrivileges` only when absolutely necessary, and never run entire web part logic in an elevated context.

How do you handle feature stapling in SharePoint 2010, and what are its limitations?

Feature stapling is a technique used to automatically activate a feature when a site is created from a specific site template. It is configured declaratively in the **Feature Stapling** element within a feature definition (usually a farm-level feature). An experienced developer understands that feature stapling is powerful but has limitations. It only works for site templates defined in the farm; custom site definitions must be explicitly targeted. Also, feature stapling does not support deactivation or removal cleanly—once a staple is applied, you cannot easily remove it without custom code. Another limitation is that you cannot conditionally staple based on properties of the site being created; it is all or nothing for a given template. For more complex scenarios, consider using **Site Provisioning Providers** or event receivers on the `SPWebEventReceiver.SiteProvisioned` event, which give you greater control and flexibility.

Describe your approach to creating a custom field type in SharePoint 2010. What assemblies are required?

Creating a custom field type requires at minimum a field class (inheriting from `SPField` or one of its derived classes) and a field control (a server control inheriting from `BaseFieldControl`). The field class is defined in an assembly deployed to the GAC, while the field control can be deployed to the GAC or the web application's bin directory. You also need an **FldTypes_*.xml** file (e.g., `FldTypes_MyCustomField.xml`) that registers the field type with SharePoint. This XML file must be placed in the `Template\Xml` folder of the 14 hive. The field type definition includes the class name, assembly, field control class, and rendering template. For experienced developers, the challenge is often handling validation, databinding, and rendering correctly across list views, display forms, and edit forms. Testing is critical because field types are loaded at startup and any errors can prevent the web application from functioning.

Event Receivers, Workflows, and Business Logic

Business logic integration is a core responsibility for any SharePoint developer. Interviewers will want to see that you understand the event model, workflow architecture, and performance implications.

Explain the difference between synchronous and asynchronous event receivers in SharePoint 2010. Give an example of when you would use each.

Synchronous event receivers run before or after an event (e.g., `ItemAdding` vs. `ItemAdded`) and block the user until they complete. They are ideal for validation logic where you must prevent the action from occurring. For example, in an `ItemAdding` event, you can check if a user has permission to add an item and cancel the event if not. Asynchronous event receivers run after the event is committed and do not block the user. They are suitable for logging, notifications, or any non-critical post-processing. For instance, after an item is added (`ItemAdded`), you can send an email notification to a manager. Experienced developers know that synchronous events are riskier because they can cause timeouts if the logic is too slow, and they cannot be relied upon for auditing since the event can be canceled. Asynchronous events, on the other hand, are more reliable for logging because they execute after the transaction is committed, but they do not block the user interface.

How do you ensure a custom workflow in SharePoint 2010 does not degrade farm performance?

Workflows in SharePoint 2010 run using the .NET Framework 3.5 Windows Workflow Foundation runtime. A common performance pitfall is creating workflows that are too complex or have long-running operations that hold timers hostage. An experienced developer will design workflows to be as stateless as possible, using simple logic and avoiding excessive custom code activities. One effective strategy is to use the **Workflow Timer Service** judiciously: set appropriate delays and avoid very short intervals. Also, ensure that workflows are not accidentally triggered in large batches, which can overwhelm the timer service. For high-volume scenarios, consider using a custom event receiver instead of a workflow, as event receivers are generally lighter. Monitoring the **Workflow Instance** count in the Central Admin and setting appropriate throttling limits (like the number of queued workflow instances) is essential for maintaining farm health.

Explain the concept of item-level security in event receivers. How would you implement a custom permission scheme that checks a user's role before allowing an item to be edited?

Item-level security in SharePoint 2010 is typically managed through list item permissions, but event receivers can enforce custom rules. An experienced developer would use the `ItemUpdating` event to check the current user's role membership using `SPWeb.CurrentUser` or `SPWeb.AllUsers`. For example, you can verify if the user belongs to the "Editors" group by checking

`SPWeb.CurrentUser.Groups`. If the user is not in the appropriate group, you set `properties.ErrorMessage` and perform `properties.Cancel = true`. However, this approach has limitations: item-level security is not reflected in the UI automatically, so the user may see the edit link but get an error when they click it. A more robust solution is to combine the event receiver with custom UI that hides the edit button based on the same permission logic. Also, be careful with elevated privileges: if your event receiver runs with elevated permissions, it will bypass the security check entirely, which is a common mistake. Always use the user context from the event properties.

Performance, Scalability, and Production Considerations

Experienced developers are expected to think beyond development and consider production stability, scalability, and operational health.

What are the most common causes of memory leaks in SharePoint 2010 custom code, and how do you prevent them?

Memory leaks in SharePoint 2010 often stem from improper disposal of **SPRequest** objects and other unmanaged resources. The most common culprit is failing to call `Dispose()` on objects that implement `IDisposable`, such as `SPSite`, `SPWeb`, and `SPListItem`. An experienced developer always uses the `using` block in C# to ensure proper disposal. Another common cause is holding references to large SharePoint objects in static variables or session state, which prevents garbage collection. Also,