Rarely Used Salesforce Apex Annotations You Should Know

In Salesforce annotations seems like a very small thing but it has big impact on your apex code. These annotations were symbols prefixed with @ before their names. These Annotations works like an sticky notes on your apex code defining or giving special instructions change code behaviour. They don't actually do the work themselves, but they tell Salesforce how to handle your code. Once you understand them, they'll save you time, prevent headaches, and open up new ways to solve problems. Let's walk through the most useful ones.

@InvocableMethod:

Have you ever built something in Apex and thought, I wish the person who isn't a developer could use this by just clicking something in Flow? That's exactly what @InvocableMethod does. It turns your fancy Apex code into a building block that anyone can use in Process Builder or Flow, no code needed.

Here's a real example from my work, Our support team wanted to automatically create a bunch of follow-up tasks when a case was marked as Escalated. The logic needed data from several different places, which Flow couldn't do easily. So I wrote the logic in Apex and put @InvocableMethod right above it. Suddenly, that complex code appeared as a simple action in the Flow builder. The support manager could drag and drop it like any other action. It builds a perfect bridge between the code world and the drag and drop world.

@InvocableVariable:

If @InvocableMethod builds a bridge, @InvocableVariable puts clear labels on the boxes moving across that bridge. When your Apex method needs information from a Flow (like a Case ID), and needs to send information back (like a success message), you use this annotation to label exactly what's coming and going.

I learned why this matters the hard way. I made a method for Flow, but without @InvocableVariable, Salesforce just sent all the data it could find. The Flow became confusing because too many fields appeared. When I added @InvocableVariable to just the two specific pieces of data the method needed, everything became clean and clear. It's like giving someone a form with specific blanks to fill in, instead of telling them to just write something down.

@TestVisible:

In Saleforce private methods or variables were hidden from test classes. So in apex classes if a method is marked as private class it was hidden even from test class. But we will face scenarios where we need to test core logic of apex code with private methods.

So most people make methods public so that test class can work it makes things messy. @TestVisible is the elegant fix. It says, Hey, this is still private to everyone else, but let the test classes take a look. For instance, if you have a private method that calculates a discount, your test can check that calculation directly without running the whole sales process. It keeps your code clean and your tests thorough.

@JsonAccess:

When we integrate salesforce with external systems we will send or receive data mostly in JSON format. By default, when Apex creates JSON from an object, it includes everything. This can be a problem if you have sensitive data or just extra stuff the other system doesn't need.

@JsonAccess lets you be the boss of your JSON. You can tell it, Always include this field when we send data out, or Never let outside data fill in this field. It's like having a security guard for your data moving in and out. I used this for an API that sent order confirmations. We needed to send the order total and status, but never the internal profit margin. @JsonAccess made that easy and safe.

@RemoteAction:

This is a bit of an older tool, but if you work with Visualforce pages, it's a lifesaver. Back before Lightning, we used @RemoteAction to make Visualforce pages act like modern web pages, where you could click something and see new data without the whole page reloading.

You add @RemoteAction to a method, and suddenly your JavaScript on the Visualforce page can call it. It's like giving your page a direct phone line to the server. I still use it to update parts of a page based on a user's selection. While newer tools exist, for maintaining thousands of existing Visualforce pages, @RemoteAction is the straightforward way to add quick, dynamic features.

@Deprecated:

As a salesforce developer we will encounter situation where our old code gets outdated and will be replaced by new code. But some people might still use that code somewhere else. @Deprecated is how you tell them, nicely, to stop.

When you put @Deprecated above an old method, two things happen. First, it's a bright red flag for any developer who sees it. Second, if someone writes new code that uses it, they'll get a warning. It's not an error that breaks things, just a gentle remainder. We used this when we replaced an old way of calculating commissions with a new one. We marked the old one as deprecated, and over the next few months, everyone had time to move their reports and processes to the new method without any panic.

@ReadOnly:

In Salesforce the governor limit for retrieving the data was 50,000 records so salesforce wont allow pulling data more than that limit. But if we want more data for making reports, salesforce wont allow us retrieving data unless you use @ReadOnly Annotation.

Putting @ReadOnly on a Visualforce page action tells Salesforce, This page is only looking at data, not changing it. In return, Salesforce gives you a much bigger bucket of up to 1,000,000 rows. The trade-off is you can't save anything on that page. I use this for read-only dashboards or data export tools. It's a specific solution for a specific problem, but when you need it, nothing else will do.

@NamespaceAccessible:

This last one is important if you build apps to sell on the AppExchange (managed packages). By default, the code in your package is locked away so customers can't mess with it. But sometimes, you want to give them a way to use some of your code in their own customizations.

@NamespaceAccessible creates a safe sharing window. It lets you say, This particular class or method can be used by code outside my package, inside this customer's org. It's like letting your neighbor borrow your lawnmower but not your car. They get something useful, but your important stuff stays safe. It's how professional app builders make their apps flexible and powerful for their customers

In the end, these annotations are all about communication. They're how you, the developer, leave clear instructions for Salesforce, for other developers, and for the people who use what you build. They take a few minutes to learn but will save you hours of work and frustration. Start using them, and you'll find your code becomes cleaner, more powerful, and much easier to work with.