In our last blog post, we talked about the basics of AngularJS and how to get started with apps that use one or more modules. In this blog post, we'll continue that discussion by talking about the Module Pattern. This pattern is a fundamental strategy for managing complexity in software applications and uses a separate part of the framework to handle all module interactions. Common scenarios where you might use a module pattern include: Separating different kinds of modules from one another, such as UI modules and business logic modules. Allowing for "isolation" of modules from one another, such as having a user login module and a user logout module. A special form of modularity called "Asynchronous Module Definition". Read on to see how AngularJS uses the Module Pattern to provide all these benefits. The Module Pattern: An Overview

Module Scenario #1 - Separate UI and Business Logic Modules. In this scenario, we have two modules: a login module and a logout module. These two modules may be coupled together, but they can also be separated from one another to reduce coupling between the modules. Module Scenario #2 - User Login and User Logout. In this second scenario, we have two modules: a login module and a logout module. We want to be able to log out the user after the user logs in. To accomplish this, we can use an isolated router that will take care of routing through our app when someone's logged in (using the login module) or when they're logged out (using the logout module). Module Scenario #3 - Separate "user" Module and "auth" Module. This scenario is especially useful for separating authentication logic from your application logic. This is a common scenario in web applications. In this scenario, we have a user module and an auth module. We set something called "isolation" on the auth module, which means that it can't see anything from outside of itself. It totally trusts the user module, but nothing else. As a result, if we want to redirect a user after they login or logout, we can do that by using router-outlet directives in the user module without having to worry about whether or not they're authenticated at that time. Module Scenario #4 - Asynchronous Module Definition (AMD). In this last scenario, modules are being used to define dependencies asynchronously from where those dependencies are being used. This is a special form of modularity that makes modules asynchronous using the System.js module loader. Module Dependencies are defined using AMD. Let's look at how AngularJS uses the Module Pattern to provide all these benefits. Using AngularJS to Simplify Your Life

Module Scenario #1 - Separate UI and Business Logic Modules. This module pattern is used when you want to separate different kinds of modules from one another, such as UI modules and business logic modules.

658eeb4e9f3251