A module defines the application functionality that is applied to the entire HTML page using the ng-app directive.

You can think of a module as a container for the different parts of your app – controllers, services, filters, directives, etc.

At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's **HTML compiler** ($compile) to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children.

AngularJS comes with a set of these directives built-in, like ngBind, ngModel, and ngClass. Much like how you create controllers and services, you can create your own directives for AngularJS to use.

When AngularJS bootstraps your application, the HTML compiler compares the DOM matching directives against the DOM elements.

Scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events.

Scopes provide APIs ($watch) to observe model mutations.

Scopes provide APIs ($apply) to propagate any model changes through the system into the view from outside of the "AngularJS realm" (controllers, services, AngularJS event handlers).

Scopes can be nested to limit access to the properties of application components while providing access to shared model properties. Nested scopes are either "child scopes" or "isolate scopes". A "child scope" inherits properties from its parent scope. An "isolate scope" does not.

Scopes provide context against which expressions are evaluated. For example {{username}} expression is meaningless, unless it is evaluated against a specific scope which defines the username property.

Expressions are JavaScript-like code snippets that are mainly placed in bonds such as {{ textBinding }}, but also used directly in directive attributes such as ng-click="functionExpression()".

Services are substitutable objects that are wired together using dependency injection (DI). You can use services to organize and share code across your app.

To use an AngularJS service, you add it as a dependency for the component (controller, service, filter or directive) that depends on the service.