NgModule API

At a high level, NgModules are a way to organize Angular applications and they accomplish this through the metadata in the `@NgModule` decorator. The metadata falls into three categories:

- **Static:** Compiler configuration which tells the compiler about directive selectors and where in templates the directives should be applied through selector matching. This is configured using the `declarations` array.
- **Runtime:** Injector configuration using the `providers` array.
- **Composability/Grouping:** Bringing NgModules together and making them available using the `imports` and `exports` arrays.

```
@NgModule({
  // Static, that is compiler configuration
  declarations: [], // Configure the selectors

  // Runtime, or injector configuration
  providers: [], // Runtime injector configuration

  // Composability / Grouping
  imports: [], // composing NgModules together
  exports: [] // making NgModules available to other parts of the app
})
```

`@NgModule` metadata

The following table summarizes the `@NgModule` metadata properties.

```
<th>
  Property
</th>

<th>
  Description
</th>
```

```
<td style="vertical-align: top">
  <code>declarations</code>
</td>

<td>

  A list of [declarable](guide/ngmodule-faq#q-declarable) classes,
  (*components*, *directives*, and *pipes*) that _belong to this module_.

  <ol>
    <li>When compiling a template, you need to determine a set of selectors which should be
    <li>
      The template is compiled within the context of an NgModule&mdash;the NgModule within w
      <ul>
        <li>All selectors of directives listed in `declarations`.</li>
        <li>All selectors of directives exported from imported NgModules.</li>
      </ul>
    </li>
  </ol>

  Components, directives, and pipes must belong to _exactly_ one module.
  The compiler emits an error if you try to declare the same class in more than one module.
  directly or indirectly from another module.

</td>

<td style="vertical-align: top">
  <code>providers</code>
</td>

<td>

  A list of dependency-injection providers.

  Angular registers these providers with the NgModule's injector.
  If it is the NgModule used for bootstrapping then it is the root injector.

  These services become available for injection into any component, directive, pipe or servi

  A lazy-loaded module has its own injector which
  is typically a child of the application root injector.

  Lazy-loaded services are scoped to the lazy module's injector.
  If a lazy-loaded module also provides the `UserService`,
  any component created within that module's context (such as by router navigation)
  gets the local instance of the service, not the instance in the root application injector.

  Components in external modules continue to receive the instance provided by their injector

  For more information on injector hierarchy and scoping, see [Providers](guide/providers) a

</td>

<td style="vertical-align: top">
  <code>imports</code>
</td>

<td>
```

```
  A list of modules which should be folded into this module. Folded means it is
  as if all the imported NgModule's exported properties were declared here.

  Specifically, it is as if the list of modules whose exported components, directives, or pi
  are referenced by the component templates were declared in this module.

  A component template can [reference](guide/ngmodule-faq#q-template-reference) another comp
  when the reference is declared in this module or if the imported module has exported it.
  For example, a component can use the `NgIf` and `NgFor` directives only if the
  module has imported the Angular `CommonModule` (perhaps indirectly by importing `BrowserMo

  You can import many standard directives from the `CommonModule`
  but some familiar directives belong to other modules.
  For example, you can use `[(ngModel)]` only
  after importing the Angular `FormsModule`.

</td>

<td style="vertical-align: top">
  <code>exports</code>
</td>

<td>

  A list of declarations&mdash;*component*, *directive*, and *pipe* classes&mdash;that
  an importing module can use.

  Exported declarations are the module's _public API_.
  A component in another module can [use](guide/ngmodule-faq#q-template-reference) _this_
  module's `UserComponent` if it imports this module and this module exports `UserComponent`

  Declarations are private by default.
  If this module does _not_ export `UserComponent`, then only the components within _this_
  module can use `UserComponent`.

  Importing a module does _not_ automatically re-export the imported module's imports.
  Module 'B' can't use `ngIf` just because it imported module 'A' which imported `CommonModu
  Module 'B' must import `CommonModule` itself.

  A module can list another module among its `exports`, in which case
  all of that module's public components, directives, and pipes are exported.

  [Re-export](guide/ngmodule-faq#q-reexport) makes module transitivity explicit.
  If Module 'A' re-exports `CommonModule` and Module 'B' imports Module 'A',
  Module 'B' components can use `ngIf` even though 'B' itself didn't import `CommonModule`.
```

```
</td>

<td style="vertical-align: top">
  <code>bootstrap</code>
</td>

<td>

  A list of components that are automatically bootstrapped.

  Usually there's only one component in this list, the _root component_ of the application.

  Angular can launch with multiple bootstrap components,
  each with its own location in the host web page.

</td>
```

More on NgModules

You may also be interested in the following: * Feature Modules. * Entry Components. * Providers. * Types of Feature Modules.