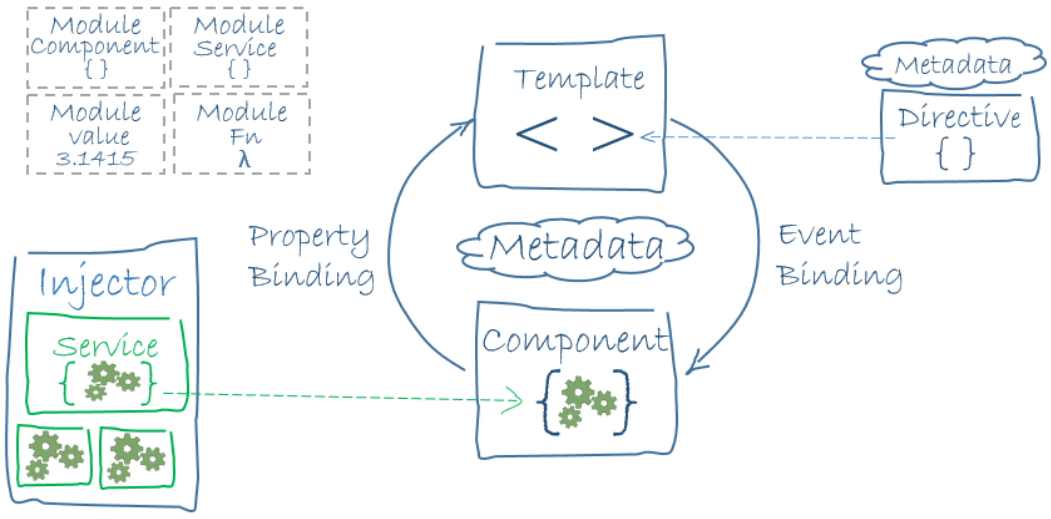
Agenda: Introduction

- Basic Building Blocks of Angular Applications

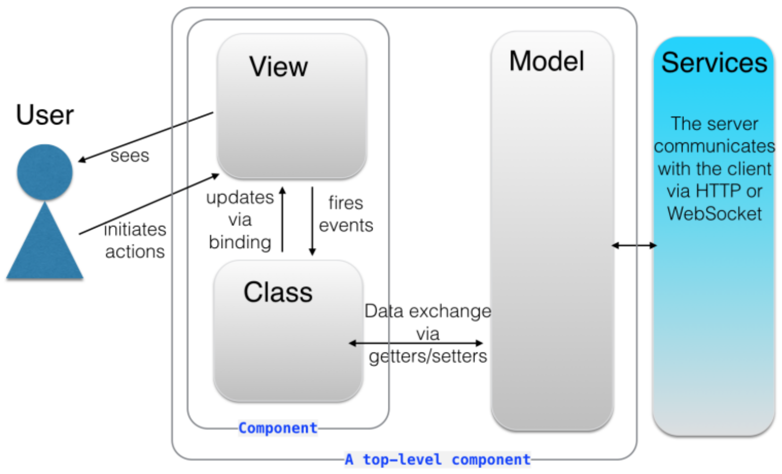
Angular Architecture

- Angular is a framework for building client applications in HTML and either JavaScript or a language like TypeScript that compiles to JavaScript.
- The framework consists of several libraries, some of them core and some optional.
- You write Angular applications by composing HTML templates with Angularized markup, writing component classes to manage those templates, adding application logic in services, and boxing components and services in modules.
- Then you launch the app by *bootstrapping* the *root module*. Angular takes over, presenting your application content in a browser and responding to user interactions according to the instructions you've provided.



In **Angular2 component** is a centerpiece of the architecture.

The Figure shows a high-level diagram of a sample Angular application:



The architecture diagram identifies the eight main building blocks of an Angular application:

1. Modules
2. Components
3. Templates
4. Metadata
5. Data binding
6. Directives
7. Services
8. Dependency injection

Modules:

Angular apps are modular and Angular has its own modularity system called *Angular modules* or *NgModules.* Every Angular app has at least one Angular module class**, the *root module***, conventionally named AppModule with @NgModule decorator.

Every angular moudle has @NgModule decorator. @NgModule is a decorator function that takes a single metadata object whose properties describe the module.

- **imports** - other modules whose exported classes are needed by component templates declared in *this* module. Only NgModule classes go in the imports array.
- **declarations** - the *view classes* that belong to this module. Angular has three kinds of view classes: components, directives, and pipes.
- **bootstrap** - the main application view, called the ***root component***, that hosts all other app views. Only the *root module* should set this bootstrap property.

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';

@NgModule({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

- Declarations will have components which we are using in the application, we can declare 'N' number of declarations but before declaring it should be imported from the respective module.
- In angular4 we need to bootstrap the app manually, this will tell the browser what are the components that should be visible, we can add 'N' number of components

Launch an application by *bootstrapping* its root module. During development you're likely to bootstrap the AppModule in a main.ts file like this one.

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

Note: Angular modules are completely different and unrelated to JavaScript modules.

Angular module — a class decorated with @NgModule — is a fundamental feature of Angular

JavaScript module – a file is a module and all objects defined in the file belong to that module. Some are public having export keyword.

Angular Libraries:

- Angular ships as a collection of JavaScript modules. You can think of them as library modules.
- Each Angular library name begins with the @angular prefix.
- You install them with the **npm** package manager and import parts of them with JavaScript import statements.

Eg:

```
import { Component } from '@angular/core'; //In app.component.ts
import { BrowserModule } from '@angular/platform-browser'; //In app.module.ts
```

Angular Root Module / application module (app.module.ts) needs material from BrowserModule. To access that material, add it to the @NgModule metadata imports like this.

- To watch the output in the browser **BrowserModule** is to be imported from **@angular/platform-browser** library.

```
imports: [ BrowserModule ],
```

Components:

- A component controls a patch of screen called a View

- You define a component's application logic—what it does to support the view—inside a class. The class interacts with the view through an API of properties and methods.
- Angular creates, updates, and destroys components as the user moves through the application. Your app can take action at each moment in this lifecycle through optional lifecycle hooks, like ngOnInit().

```
import { Component } from '@angular/core';


@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`,
})
export class AppComponent  { name = 'Angular'; }
```

Templates:

- You define a component's view with its companion **template**. A template is a form of HTML that tells Angular how to render the component.
- A template looks like regular HTML, except for a few differences.

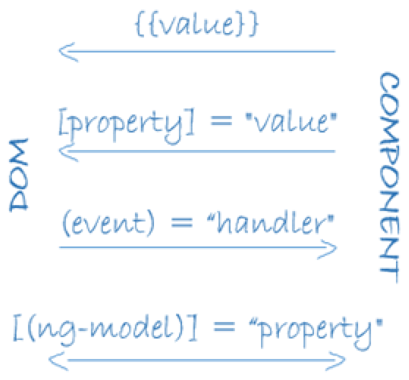
Metadata:

- Metadata tells Angular how to process a class.
- In TypeScript, you attach metadata by using a **decorator**.

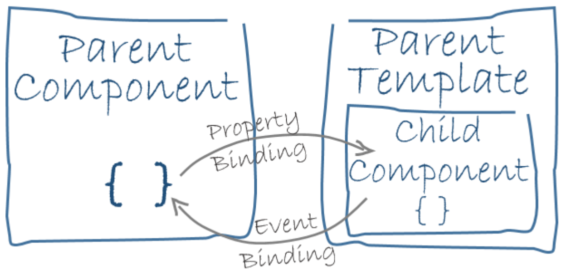
Note: The template, metadata, and component together describe a view.

Data Binding:

- A mechanism for coordinating parts of a template with parts of a component. Add binding markup to the template HTML to tell Angular how to connect both sides.
- There are four forms of data binding syntax.



- Data binding is also important for communication between parent and child components.



Directives:

Angular templates are *dynamic*. When Angular renders them, it transforms the DOM according to the instructions given by **directives**.

A directive is a class with a @Directive decorator. A component is a *directive-with-a-template*; a @Component decorator is actually a @Directive decorator extended with template-oriented features.

Services:

Service is a broad category encompassing any value, function, or feature that your application needs.

Almost anything can be a service. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

Examples include:

- logging service
- data service
- message bus
- tax calculator
- application configuration

There is nothing specifically *Angular* about services. Angular has no definition of a service. There is no service base class, and no place to register a service. Yet services are fundamental to any Angular application. Components are big consumers of services.

A component's job is to enable the user experience and nothing more. It mediates between the view (rendered by the template) and the application logic (which often includes some notion of a *model*). A good component presents properties and methods for data binding. It delegates everything nontrivial to services.

Dependency Injection:

- Dependency injection is wired into the Angular framework and used everywhere.
- The *injector* is the main mechanism.
    - An injector maintains a *container* of service instances that it created.

    - An injector can create a new service instance from a *provider*.
- A *provider* is a recipe for creating a service.
- Register *providers* with injectors.

Other important features:

a) Animations
b) Change detection
c) Events
d) Forms
e) HTTP
f) Lifecycle hooks
g) Pipes
h) Router
i) Testing

Bootstrap in *main.ts*

There are many ways to bootstrap an application. The variations depend upon how you want to compile the application and where you want to run it.

In the beginning, you will compile the application dynamically with the *Just-in-Time (JIT)* compiler and you'll run it in a browser. You can learn about other options later.

The recommended place to bootstrap a JIT-compiled browser application is in a separate file in the app folder named app/main.ts

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app.module';
platformBrowserDynamic().bootstrapModule(AppModule);
```

This code creates a browser platform for dynamic (JIT) compilation and bootstraps the AppModule.

The *bootstrapping* process sets up the execution environment, digs the *root* AppComponent out of the module's bootstrap array, creates an instance of the component and inserts it within the element tag identified by the component's selector.

Remember that Angular can run on different platforms and so when we say platformBrowser-Dynamic() we're telling Angular that we're using a browser and we want to compile Angular "dynamically" (e.g. when we open the page in our browser vs. ahead-of-time compiling.

Once the application is bootstrapped, the AppComponent will be rendered where the <my-app></my-app> snippet is on the index.html file.

```
<my-app>Loading AppComponent content here ...</my-app>
```

Note: This file is very stable. Once you've set it up, you may never change it again.