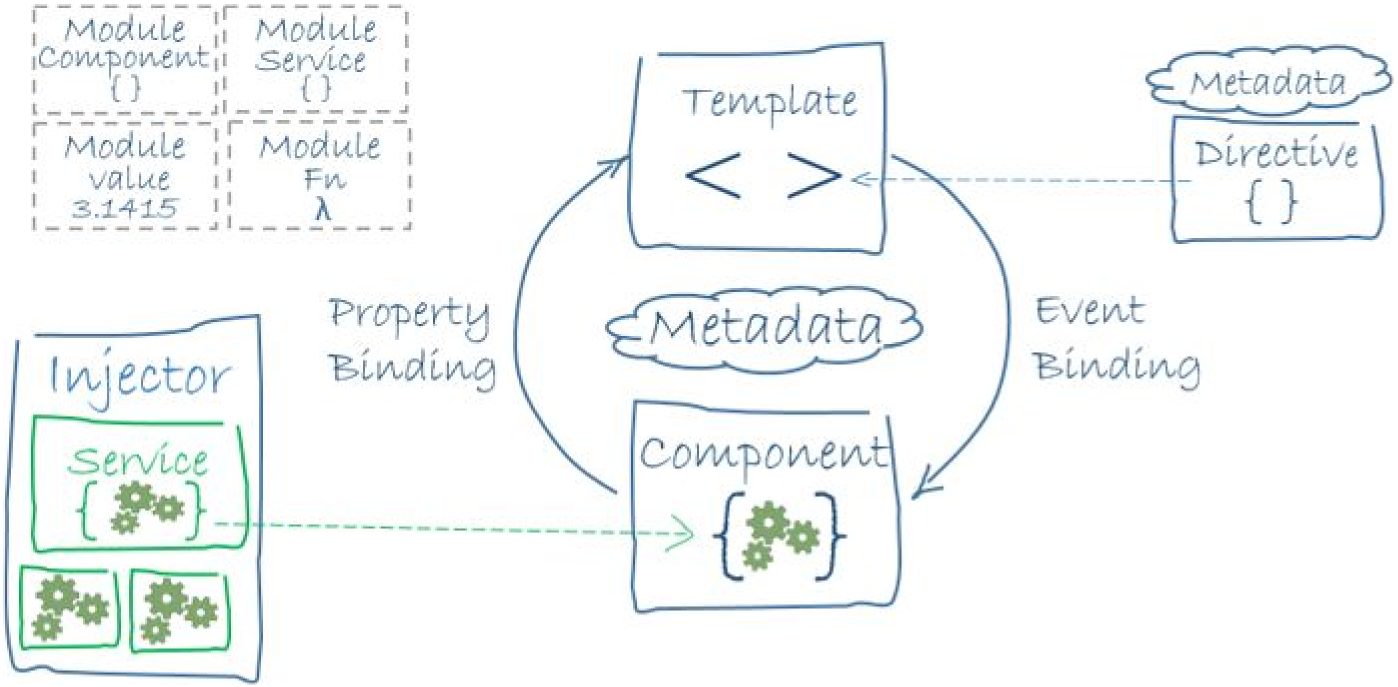
Angular Architecture

architecture

Angular architecture is made from the following parts

- **Modules**
- **Components / Directives**
- **Templates**
- **Services**
- **DI**

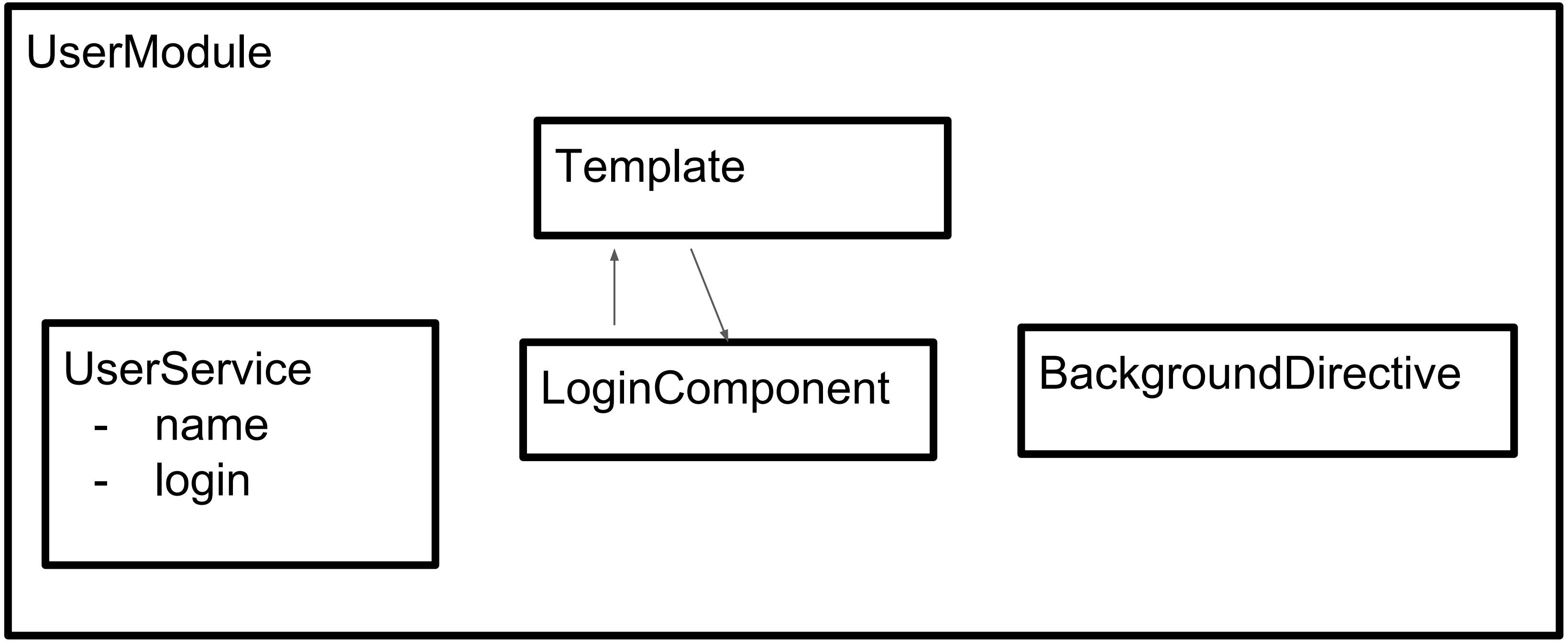
architecture



Architecture

- Architecture elements are represented by classes with decorators
- every element will usually be in a separate file
- we won’t create instances of those classes (this will be the job of the DI)
- Let’s start a new @angular/cli project and go over each element in the architecture.
- While going over the elements in the architecture we will learn how to create them by adding user authentication

What we will create



Module

- Some of the items we create in one app we might need to use them in another
- Angular is modular meaning you can wrap sections in your app with a module
- In angular app there is a single module connected to the DOM which is referred to **RootModule**
- There is at least one module in an angular app
- Component/Directives/Services should belong to a single module
- a module can use features from other modules
- a module can export certain component/directives
- @angular/cli creates a root module for us: **src/app/app.module.ts**
- Let’s go over the module created and create another module of our own
- module can use other modules by including them in the **imports** array of the metadata
- The root module have to include **BrowserModule** in imports

Component

- Components represent a block of UI
- components are connected to a template
    - properties and methods from the class can be connected to the template
    - inputs and events from the template can be connected to the class
- the template is built with angular templating language and it’s based on HTML
- component are classes decorated with **@Component**
- components has a **selector** which creates a new tag/class/attribute that when angular sees it in the template it will be replaced with our compnent
- components belong to a single module and can be included in the module by adding them to the **declarations** array in the **@NgModule** decorator
- In our user module we have a login component containing a form with a name input and a login button
- Let’s create our component...

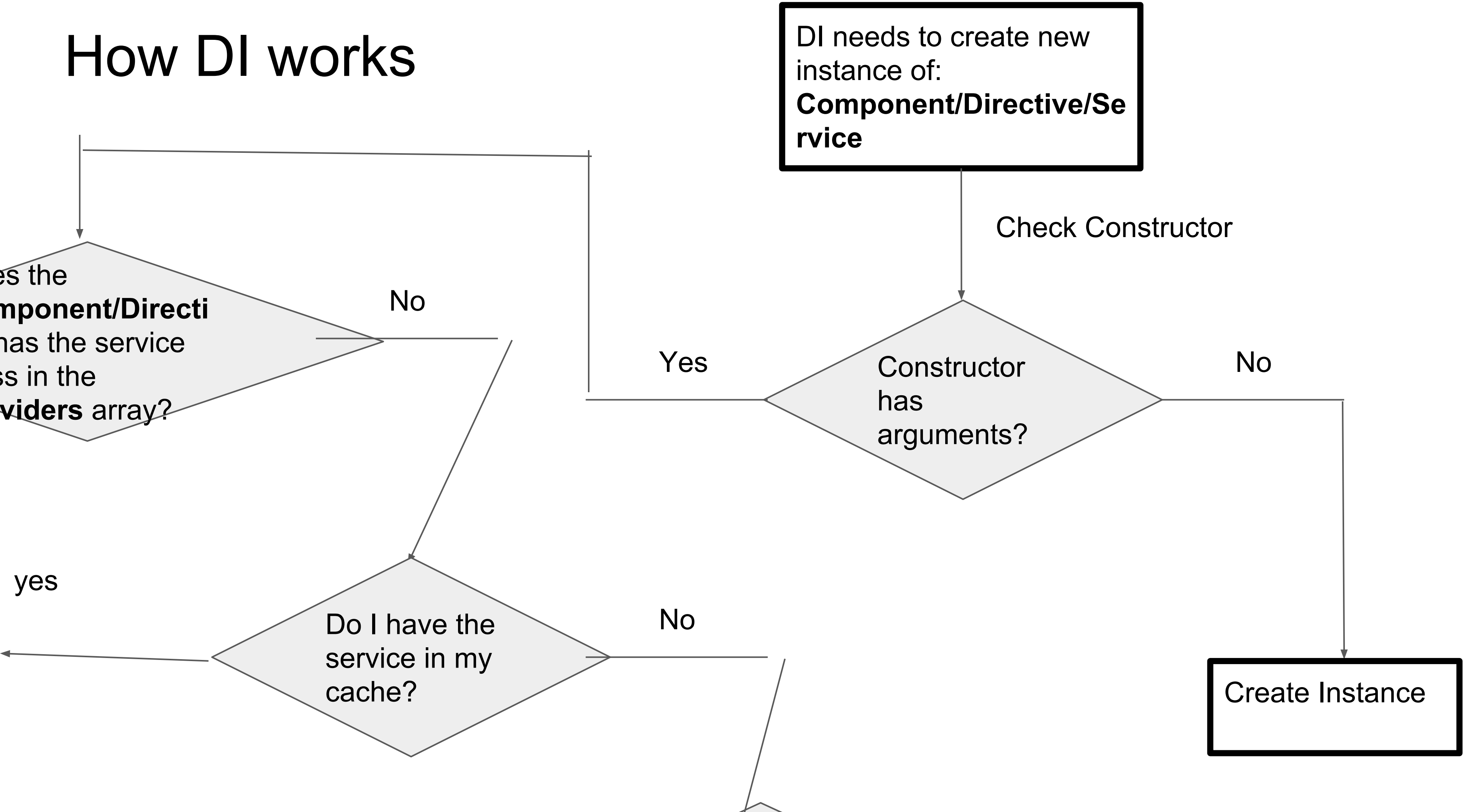
Service

- Business logic of our app
- service supplies the data to our **components/directives**
- we can use services to transfer data between compnents/directives
- **components/directives** usually need to be slim and most of the calculation should be in services
- communication with server will usually be done in services
- a service can use other services
- a service needs to belong to a module and we need to put the service class in the **providers** array of the module it belongs to
- lets create a user service which holds the name of the logged in user

DI

- Dependency Injector is in charge of creating the instances of our classes
- it can supply **components/directives** with our services
- it can supply services with other services
- DI is lazy
- DI injecting items in the constructor
- If we want to inject services to other services, we have to decorate the services with the **Injectable** decorator
- The DI holds instances of the classes in a cache and it needs to decide will i create new instance or grab from cache

How DI works


DI needs to create new instance of: **Component/Directive/Se rvice**
Check Constructor
Constructor has arguments?
No
Create Instance
Yes
es the **mponent/Directi** has the service ss in the **viders** array?
No
Do I have the service in my cache?
yes
No

Directives

- directives are similar to components but they don’t have a template
- directive usually add functionality to their hosting element
- they also has a selector and add tag/class/attribute to the templating language
- when angular sees that tag/class/attribute it will create the instance of our directive
- directive belongs to a module and we need to put the directive class in the **declarations** array of the module
- let’s add a directive that change the background of element

Summary

- Angular is now more ui component based
- similar to how directives were in angular 1
- now we create our entire application from blocks of UI
- angular is in charge on rendering the UI of the component and keep the templates and components in sync.
-