**component** - отвечает за часть web-страницы и включает в себя HTML-шаблон, CSS-стили и логику поведения;

**service** - поставщик данных для component;

**directive** - преобразует определенную часть DOM заданным образом.

Все вышеперечисленное собирается в корневой модуль, который общепринято называется **AppModule.**

**exports** - компоненты, сервисы, директивы и фильтры, доступные для использования разработчикам, которые будут использовать ваш модуль в своих разработках;

**bootstrap** - имя главного компонента приложения (как правило, называется AppComponent).

Сервисы нужны для предоставления данных компонентам. Это могут быть не только запросы к серверу, но и **функции**, преобразующие исходные данные по заданному алгоритму. Считается хорошей практикой все обращения к серверу и **функции, возвращающие данные**, выносить в сервисы.

**Контроллер** - экземпляр класса компонента.

**Некоторые DOM свойства:**

x.innerHTML - текстовое значение x

x.nodeName - название (имя) узла x

x.nodeValue - значение x

x.parentNode - родительский узел x

x.childNodes - дочерние узлы x

x.attributes - атрибутивные узлы x

Название свойства элемента DOM должно быть заключено в **квадратные скобки** или иметь **префикс bind-**, иначе значением будет строка с названием свойства компонента.

<p [class.red-text]="isRed">Some text.</p>

Здесь **red-text** - название класса, а **isRed** - переменная компонента типа boolean, значение которой определяет, будет ли в Angular шаблоне установлен класс или нет.

Каждое возникающее событие передает о себе всю информацию в объекте **$event**, который может быть передан методу класса в качестве аргумента.

<button on-click="showContacts($event)">Show Contacts List</button>

**Свойства объекта $event**:

**target** - элемент DOM, выступивший инициатором;

**target.value** - значение элемента DOM (справедливо для полей формы);

**keyCode** - код клавиши (справедливо для keyup).

Перечисленные выше свойства имеются только у стандартных событий браузера.

Для удобства в Angular предусмотрено так называемое псевдо событие **keyup.{keyCode}**, которое будет инициировано только в том случае, если будет нажата определенная клавиша. Например, для отслеживания нажатия Enter используется keyup.enter.

Пишем «new Event Emitter<any>», когда создаем свои события. Для стандартных событий типа “click” этого писать не нужно.

Метод **emit()** инициирует возникновение события.

Класс становится Angular компонентом, если его объявлению предшествует **декоратор @Component()** с объектом конфигурации.

**OnChanges** - устанавливаются или изменяются значения входных свойств класса компонента;

**OnInit** - устанавливаются "обычные" свойства; вызывается единожды вслед за первым вызовом OnChanges();

**DoCheck** - происходит изменения свойства или вызывается какое-либо событие;

**AfterContentInit** - в шаблон включается контент, заключенный между тегами компонента;

**AfterContentChecked** - аналогичен DoCheck(), только используется для контента, заключенного между тегами компонента;

**AfterViewInit** - инициализируются компоненты, которые входят в шаблон текущего компонента;

**AfterViewChecked** - аналогичен DoCheck(), только используется для дочерних компонентов;

**OnDestroy** - компонент "умирает", т.е. удаляется из DOM-дерева.

**<ng-content></ng-content>** отображает контент родительского компонента в дочернем.

**по поводу синхронности и асинхронности для наглядности примеры:**

==============

Синхронное выполнение:

console(1);

console(2);

console(3);

Выведет в консоль:

1

2

3

То есть код выполняется последовательно, строка за строкой.

==============

Асинхронное выполнение:

console.log(1);

setTimeout(() => console.log(2));

console.log(3);

Выведет:

1

3

2

То есть вывод цифры 2 происходит с задержкой, но это не остановило вывод цифры 3.

==============

Таймаут это самая простая имитация асинхронности. В жизни хороший пример это данные с сервера. Ты можешь сделать несколько запросов на сервер, а они могут выполняться за разное время и ты не ждёшь выполнения их по очереди, они происходят асинхронно и не зависят друг от друга.

**ControlValueAccessor** используется, когда мы создаем свой элемент формы, возможно, какой –нибудь необычный (например, светофор).

Назначение свойства **pathMatch**:

**const routes**:Routes=[

{**path**: **'login'**, **component**:LoginRouteComponent},

{**path**: **'home'**, **component**:HomeRouteComponent, **children**:[

{**path**: **'profile'**, **component**:ProfileRouteComponent}

]},

{**path**: **'contacts'**, **redirectTo**: **'/home'**, **pathMatch**: **'full'**, **children**:[

{**path**: **'director'**, **component**:DirectorContactsRouteComponent}

]},

{**path**: **'\*\*'**, **component**:LoginRouteComponent}

];

**@NgModule**({

**imports**:[RouterModule.**forRoot**(**routes**)],

**exports**:[RouterModule]

})

**export class** AppRoutingModule{}

У нас имеется маршрут, например, «contacts», прямой переход на который вызвал бы ошибку, если бы не было определено свойство redirectTo, поскольку для «contacts» не определен компонент.

Теперь представим, что redirectTo указан, а pathMatch - нет. Переход по маршруту /contacts/director отобразит компонент, указанный в redirectTo для contacts. Это происходит потому что при навигации на дочерний маршрут, Router проходит по дереву иерархии все его родительские URL и при попадании на /contacts сработает редирект.

Чтобы избежать этого в Angular routing указывается свойство pathMatch со значением full. И тогда перенаправление сработает только когда точно будет запрошен URL /contacts**.**

**LoadChildren:**

{path: 'order', loadChildren: './modules/order/order.module#OrderModule'}

Когда будет запрошен маршрут в состав которого входит order, тогда и загрузится OrderModule. Но в момент старта приложения он не включается в загрузку.

Использование **@ViewChild()** в родительском Angular component позволяет получить все свойства указанного дочернего компонента.

**Чтобы передавать доп. информацию при рутинге на другой стейт**:

import { Router, NavigationExtras } from '@angular/router';

private \_router: Router

const navigationExtras: NavigationExtras = {

state: {

isProfileFinished: true

}

};

this.\_router.navigate(['main/launch-plan'], navigationExtras).then();

получать:

this.\_router.getCurrentNavigation().extras.state.isProfileFinished

Если **не** устанавливается **priming**:

npm update @angular/cli @angular/cdk rxjs

npm install -S @angular/material @angular/cdk @angular/animations

npm uninstall @angular/core

npm install -S @angular/core

**Как создать Resolver**:

<https://www.youtube.com/watch?v=fBJ-ziQdPuM>

**Методы и операторы Observable**:

1. **next()** - принимает значение, которое будет возвращено функции-обработчику (т. У. используется для передачи значений). Иными словами – *«Observable, вызови эту функцию, когда у тебя есть новое значение для меня»*;
2. **error()** - принимает значение, возвращаемое функции-обработчику ошибок. Иными словами – *«Observable, вызови эту функцию, когда у тебя есть новая error для меня»*;
3. **complete()** - вызывается для уведомления "подписчиков" об окончании рассылки. Иными словами – *«Observable, вызови эту функцию, когда закончишь свою работу»*;
4. Все операторы преобразования данных объявляются в методе **pipe()** через запятую. Сам метод pipe() импортировать не надо и он должен быть вызван перед методом subscribe().

**Операторы**— это функции, которыми можно преобразовывать данные между моментом, когда Observable их отправил, и моментом, когда подписчик их получил. Т.е. преобразовывать данные во время движения.

Oператор «**of»** используется для преобразования примитивных типов в объект Observable.

Если необходимо, чтобы обработчик вместо всего массива сразу получал каждый его элемент в отдельности, используется оператор «**from**».