

# Angular

# Angular

- Typescript
- Компонентный подход
- Управляемый Change detection
- Поддерживается любой платформой
- DI
- Структура проекта



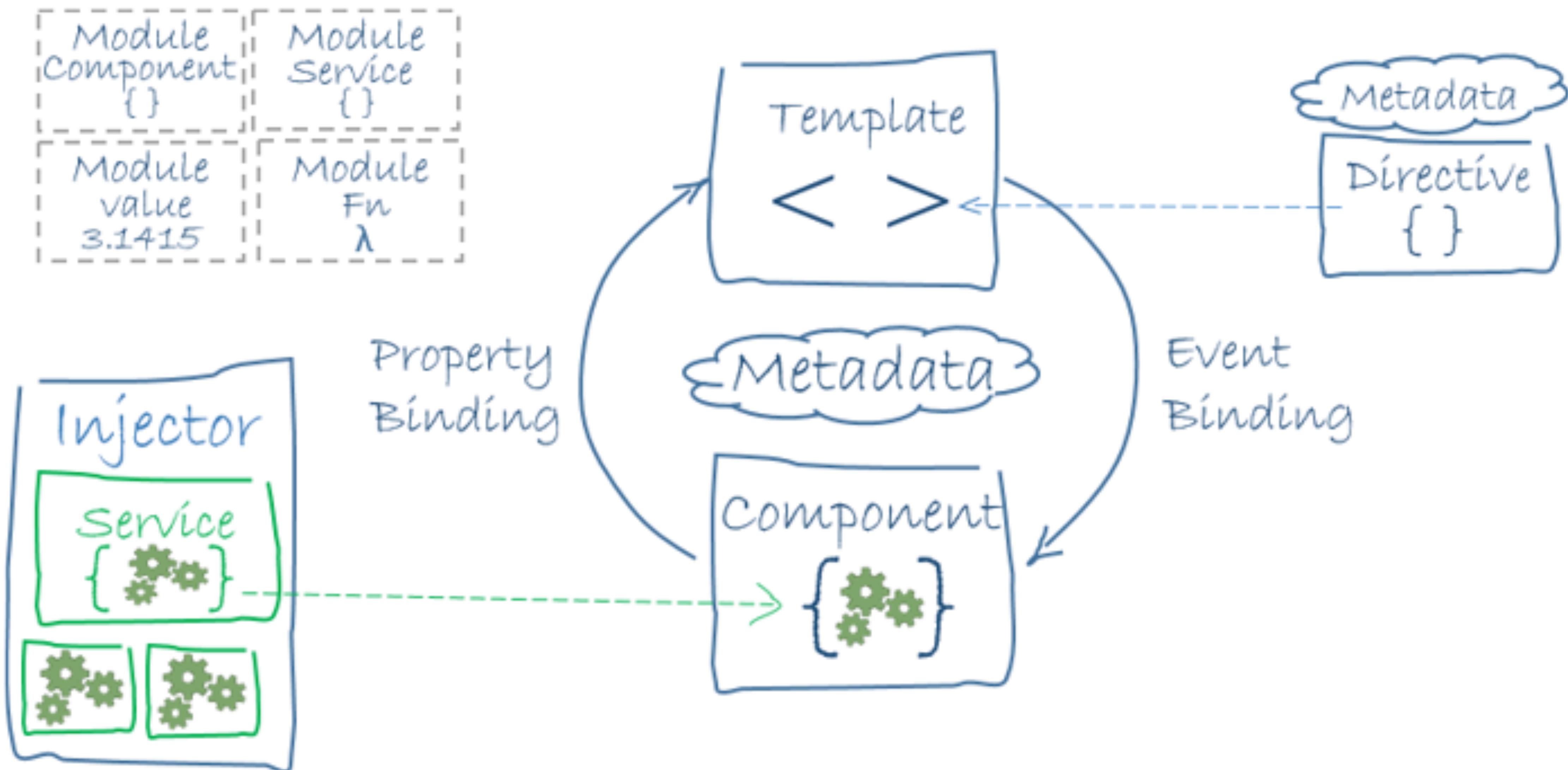
# Конфигурация проекта

- Удобный и гибкий CLI
- angular.json – скелет приложения
- Очень гибкая конфигурация проекта
- tsconfig.json
- tslint.json
- Karma, Jasmine для тестов
- Кастомизация сборки

# Как создать проект

1. Install node js (<https://nodejs.org/en/download/>)
2. npm install -g @angular/cli
3. ng new project-name
4. cd my-app
5. ng serve --open

# Архитектура Angular приложения



# Компонент

Компонент - обособленная часть функционала со своей логикой, HTML-шаблоном и CSS-стилями. Компонент управляет отображение представление на экране.  
За объявление компонента отвечает декоратор `@Component()`

Основные свойства объекта, который принимает декоратор:

- `selector` — название компонента;
- `template` (или `templateUrl`) — HTML-разметка в виде строки (или путь к HTML-файлу);
- `providers` — список сервисов, поставляющих данные для компонента;
- `styles` (или `styleUrl`) — массив путей к CSS-файлам, содержащим стили для создаваемого компонента (или путь к css-файлу).
- `standalone`

# Декораторы свойств

@Input() - декоратор, используемый для получения данных

```
@Component({...})  
export class CustomSlider {  
  @Input() value = 0;  
}
```

# Декораторы свойств

@Input() - декоратор, используемый для получения данных

```
@Component({...})
export class CustomSlider {
  @Input() value = 0;
}

export class CustomSlider {
  @Input()
  set value(newValue: number) {
    this.internalValue = newValue;
  }

  private internalValue = 0;
}
```

# Декораторы свойств

@Input() - декоратор, используемый для получения данных

```
@Component({...})
export class CustomSlider {
  @Input() value = 0;
}

export class CustomSlider {
  @Input()
  set value(newValue: number) {
    this.internalValue = newValue;
  }

  private internalValue = 0;
}

<custom-slider [value]="50" />
```

# Декораторы свойств

@Input() - декоратор, используемый для получения данных

@Output() - декоратор, используемый для отправки данных, выставляя их в качестве производителей событий

```
@Component({...})
export class CustomSlider {
  @Input() value = 0;
}

export class CustomSlider {
  @Input()
  set value(newValue: number) {
    this.internalValue = newValue;
  }

  private internalValue = 0;
}
```

```
<custom-slider [value]="50" />
```

```
@Component({...})
export class ExpandablePanel {
  @Output() panelClosed = new EventEmitter<void>();
}
```

# Декораторы свойств

@Input() - декоратор, используемый для получения данных

@Output() - декоратор, используемый для отправки данных, выставляя их в качестве производителей событий

```
@Component({...})
export class CustomSlider {
  @Input() value = 0;
}

export class CustomSlider {
  @Input()
  set value(newValue: number) {
    this.internalValue = newValue;
  }

  private internalValue = 0;
}
```

```
<custom-slider [value]="50" />
```

```
@Component({...})
export class ExpandablePanel {
  @Output() panelClosed = new EventEmitter<void>();
}
```

```
<expandable-panel (panelClosed)="savePanelState()" />
```

# Декораторы свойств

@Input() - декоратор, используемый для получения данных

@Output() - декоратор, используемый для отправки данных, выставляя их в качестве производителей событий

```
@Component({...})
export class CustomSlider {
  @Input() value = 0;
}

export class CustomSlider {
  @Input()
  set value(newValue: number) {
    this.internalValue = newValue;
  }

  private internalValue = 0;
}
```

```
<custom-slider [value]="50" />
```

```
@Component({...})
export class ExpandablePanel {
  @Output() panelClosed = new EventEmitter<void>();
}

<expandable-panel (panelClosed)="savePanelState()" />

this.panelClosed.emit();
```

# Декораторы свойств

- Required inputs

```
@Component({...})
export class CustomSlider {
  @Input({required: true}) value = 0;
}
```

# Декораторы свойств

- Required inputs

```
@Component({...})
export class CustomSlider {
  @Input({required: true}) value = 0;
}
```

- Input transforms

```
@Component({
  selector: 'custom-slider',
  ...
})
export class CustomSlider {
  @Input({transform: trimString}) label = '';
}

function trimString(value: string | undefined) {
  return value?.trim() ?? '';
}
```

# Декораторы свойств

- Type checking

```
@Component({...})
export class CustomSlider {
  @Input({transform: appendPx}) widthPx: string = '';
}

function appendPx(value: number) {
  return `${value}px`;
}
```

# Декораторы свойств

- Type checking

```
@Component({...})
export class CustomSlider {
  @Input({transform: appendPx}) widthPx: string = '';
}

function appendPx(value: number) {
  return `${value}px`;
}
```

- Built-in transformations

```
import {Component, Input, booleanAttribute, numberAttribute} from '@angular/c

@Component({...})
export class CustomSlider {
  @Input({transform: booleanAttribute}) disabled = false;
  @Input({transform: numberAttribute}) number = 0;
}
```

# Проекция контента

- Для проекции разных контентов в компоненте

```
@Component({
  selector: 'custom-card',
  template: `
    <div class="card-shadow">
      <ng-content />
    </div>
  `,
})
export class CustomCard /* ... */
```

# Проекция контента

- Для проекции разных контентов в компоненте

```
@Component({
  selector: 'custom-card',
  template: `
    <div class="card-shadow">
      <ng-content />
    </div>
  `,
})
export class CustomCard /* ... */
```

```
<!-- Using the component -->
<custom-card>
  <p>This is the projected content</p>
</custom-card>
```

# Проекция контента

- Для проекции разных контентов в компоненте

```
@Component({
  selector: 'custom-card',
  template: `
    <div class="card-shadow">
      <ng-content />
    </div>
  `,
})
export class CustomCard /* ... */
```

```
<!-- Using the component -->
<custom-card>
  <p>This is the projected content</p>
</custom-card>
```

```
<!-- The rendered DOM -->
<custom-card>
  <div class="card-shadow">
    <p>This is the projected content</p>
  </div>
</custom-card>
```

# Проекция контента

- Для проекции разных контентов в компоненте

```
@Component({
  selector: 'custom-card',
  template: `
    <div class="card-shadow">
      <ng-content select="card-title"></ng-content>
      <div class="card-divider"></div>
      <ng-content select="card-body"></ng-content>
    </div>
  `,
})
export class CustomCard /* ... */
```

```
<!-- Using the component -->
<custom-card>
  <p>This is the projected content</p>
</custom-card>
```

```
<!-- The rendered DOM -->
<custom-card>
  <div class="card-shadow">
    <p>This is the projected content</p>
  </div>
</custom-card>
```

# Проекция контента

- Для проекции разных контентов в компоненте

```
@Component({
  selector: 'custom-card',
  template: `
    <div class="card-shadow">
      <ng-content />
    </div>
  `,
})
export class CustomCard /* ... */
```

<!-- Component template -->

```
<div class="card-shadow">
  <ng-content select="card-title"></ng-content>
  <div class="card-divider"></div>
  <ng-content select="card-body"></ng-content>
</div>
```

<!-- Using the component -->

```
<custom-card>
  <card-title>Hello</card-title>
  <card-body>Welcome to the example</card-body>
</custom-card>
```

<!-- Using the component -->

```
<custom-card>
  <p>This is the projected content</p>
</custom-card>
```

<!-- The rendered DOM -->

```
<custom-card>
  <div class="card-shadow">
    <p>This is the projected content</p>
  </div>
</custom-card>
```

# Проекция контента

- Для проекции разных контентов в компоненте

```
@Component({
  selector: 'custom-card',
  template: `
    <div class="card-shadow">
      <ng-content />
    </div>
  `,
})
export class CustomCard /* ... */
```

<!-- Component template -->

```
<div class="card-shadow">
  <ng-content select="card-title"></ng-content>
  <div class="card-divider"></div>
  <ng-content select="card-body"></ng-content>
</div>
```

<!-- Using the component -->

```
<custom-card>
  <card-title>Hello</card-title>
  <card-body>Welcome to the example</card-body>
</custom-card>
```

<!-- The rendered DOM -->

```
<custom-card>
  <div class="card-shadow">
    <p>This is the projected content</p>
  </div>
</custom-card>
```

<!-- Rendered DOM -->

```
<custom-card>
  <div class="card-shadow">
    <card-title>Hello</card-title>
    <div class="card-divider"></div>
    <card-body>Welcome to the example</card-body>
  </div>
</custom-card>
```

# Жизненный цикл компонента

**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;

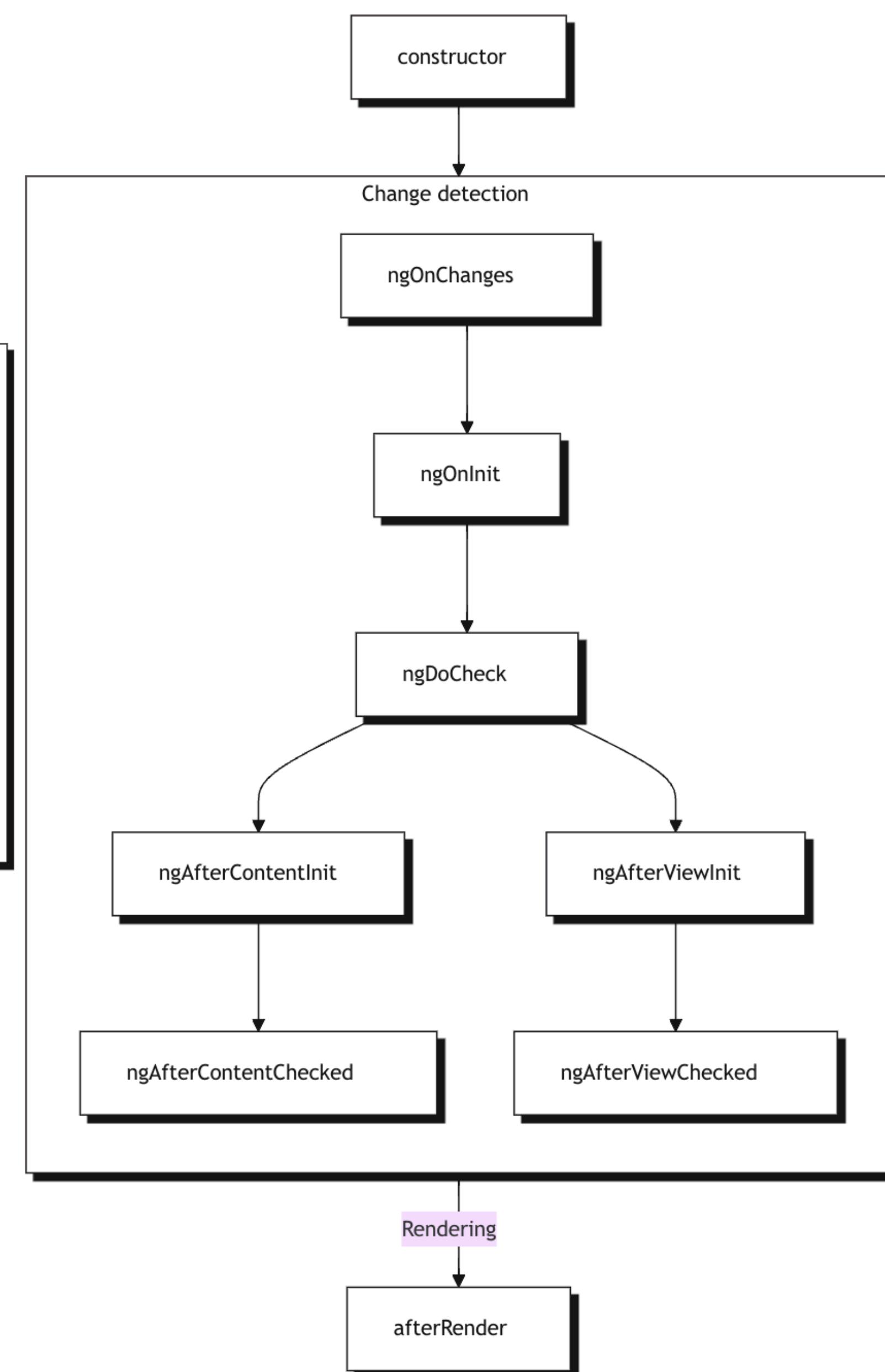
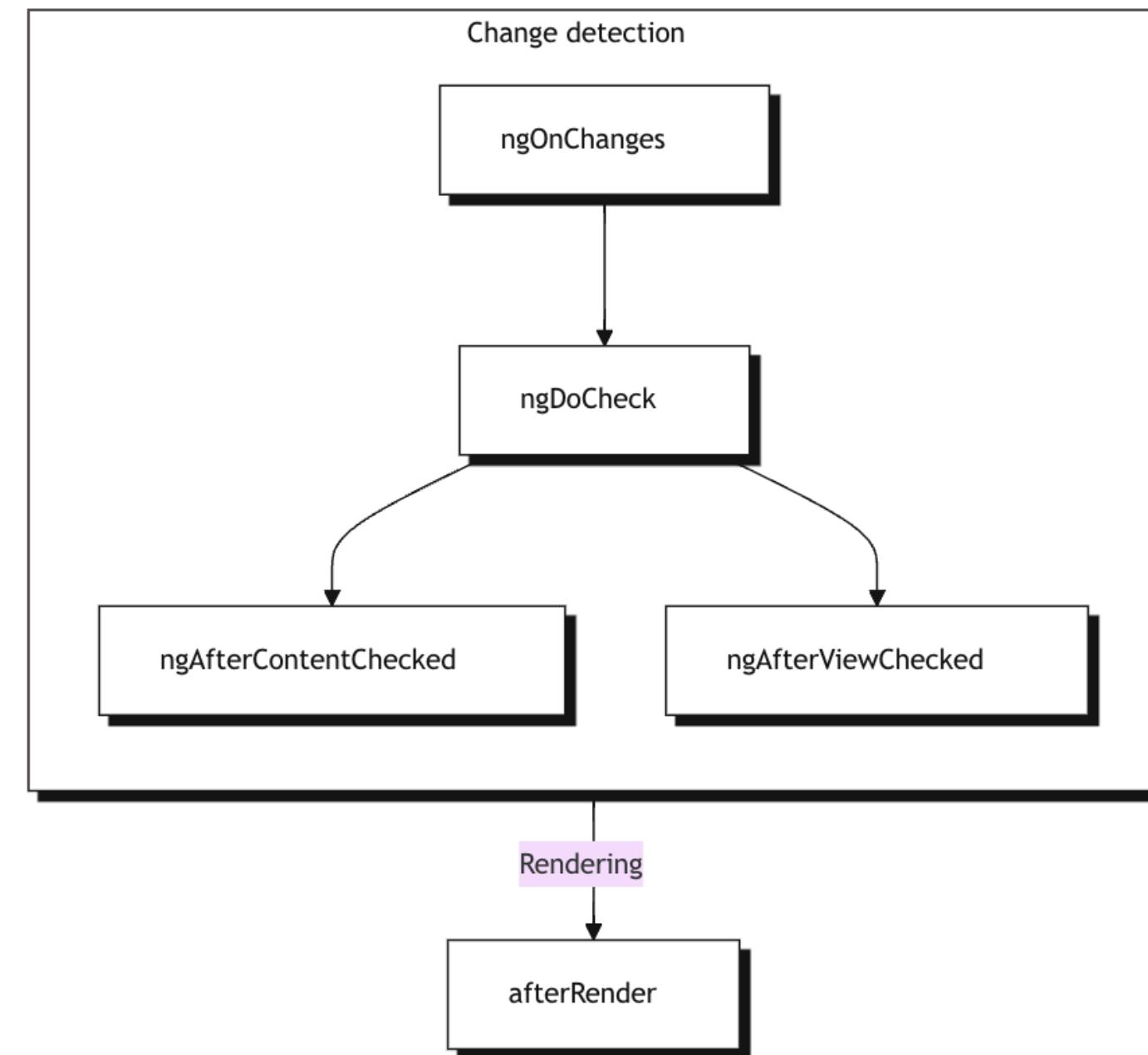
**AfterContentInit** - в шаблон включается контент, заключенный между тегами компонента;

**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



Angular hooks реализованы в виде интерфейсов, реализующих функцию, совпадающую по названию с названием интерфейса + префикс `ng`.

```
1 | export class ContactsItemComponent implements OnInit {  
2 |   //  
3 |   ngOnInit() {  
4 |     console.log('OnInit');  
5 |   }  
6 |   //  
7 | }
```

# Директива

Предназначение директив — преобразование DOM заданным образом, наделение элемента поведением.

По своей реализации директивы практически идентичны компонентам, компонент — это директива с HTML-шаблоном, но с концептуальной точки зрения они различны.

Компоненты являются директивами. (Директивы с собственным template)

Есть два вида директив:

- структурные — добавляют, удаляют или заменяют элементы в DOM;
- атрибуты — задают элементу другое поведение.

Они создаются с помощью декоратора `@Directive()` с конфигурационным объектом.

В Angular имеется множество встроенных директив (`*ngFor`, `*ngIf`), но зачастую их недостаточно для больших приложений, поэтому приходится реализовывать свои.

# Примеры директив

## Структурные встроенные (предваряются символом \*)

\*ngIf добавляет или удаляет элемент из DOM-дерева в зависимости от истинности переданного выражения

```
<app-item-detail *ngIf="isActive" [item]="item"></app-item-detail>
```

# Примеры директив

## Структурные встроенные (предваряются символом \*)

\*ngIf добавляет или удаляет элемент из DOM-дерева в зависимости от истинности переданного выражения

```
<app-item-detail *ngIf="isActive" [item]="item"></app-item-detail>
```

\*ngFor используется для визуализации массива данных. Директива применяется к блоку HTML-кода, определяющему, как должны отображаться данные элемента массива. Далее Angular использует этот HTML как шаблон для всех последующих элементов в массиве.

```
<div *ngFor="let item of items">{{ item.name }}</div>
```

# Примеры директив

## Структурные встроенные (предваряются символом \*)

\*ngIf добавляет или удаляет элемент из DOM-дерева в зависимости от истинности переданного выражения

```
<app-item-detail *ngIf="isActive" [item]="item"></app-item-detail>
```

\*ngFor используется для визуализации массива данных. Директива применяется к блоку HTML-кода, определяющему, как должны отображаться данные элемента массива. Далее Angular использует этот HTML как шаблон для всех последующих элементов в массиве.

```
<div *ngFor="let item of items">{{ item.name }}</div>
```

ngSwitch эмулирует работу оператора switch применительно к шаблонам.

# Примеры директив

`@if` добавляет или удаляет элемент из DOM-дерева в зависимости от истинности переданного выражения

```
@if (a > b) {  
    {{a}} is greater than {{b}}  
} @else if (b > a) {  
    {{a}} is less than {{b}}  
} @else {  
    {{a}} is equal to {{b}}  
}
```

}

# Примеры директив

**@if** добавляет или удаляет элемент из DOM-дерева в зависимости от истинности переданного выражения

```
@if (a > b) {  
  {{a}} is greater than {{b}}  
} @else if (b > a) {  
  {{a}} is less than {{b}}  
} @else {  
  {{a}} is equal to {{b}}  
}
```

**@for** используется для визуализации массива данных. Директива применяется к блоку HTML-кода, определяющему, как должны отображаться данные элемента массива. Далее Angular использует этот HTML как шаблон для всех последующих элементов в массиве.

```
@for (item of items; track item.name) {  
  <li> {{ item.name }}</li>  
} @empty {  
  <li aria-hidden="true"> There are no items. </li>  
}
```

# Примеры директив

**@switch** эмулирует работу оператора `switch` применительно к шаблонам.

```
@switch (userPermissions) {  
  @case ('admin') {  
    <app-admin-dashboard />  
  }  
  @case ('reviewer') {  
    <app-reviewer-dashboard />  
  }  
  @case ('editor') {  
    <app-editor-dashboard />  
  }  
  @default {  
    <app-viewer-dashboard />  
  }  
}
```

# Примеры директив

## Атрибуты встроенные

[ngStyle] принимает объект, в котором ключами служат наименования CSS-свойств, а их значениями — возможные значения соответствующих CSS-свойств.

```
setCurrentStyles() {  
    // CSS styles: set per current state of component properties  
    this.currentStyles = {  
        'font-style': this.canSave ? 'italic' : 'normal',  
        'font-weight': !this.isUnchanged ? 'bold' : 'normal',  
        'font-size': this.isSpecial ? '24px' : '12px',  
    };  
}
```

[ngClass] также принимает объект, но ключами здесь служат наименования классов, а значениями — выражения типа Boolean. Если выражение истинно, класс будет добавлен к списку уже имеющихся классов.

```
<!-- toggle the "special" class on/off with a property -->  
<div [ngClass]="isSpecial ? 'special' : ''>This div is special</div>
```

# Пайп

- Позволяют осуществлять преобразование формата отображаемых данных (например, дат или денежных сумм) прямо в шаблоне.
- Фильтры можно объединять в последовательности (pipe chains).
- Фильтры могут принимать аргументы.

```
<p>The event will occur on {{ scheduledOn | date | uppercase }}.</p>
```

```
// kebab-case.pipe.ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'kebabCase',
  standalone: true,
})
export class KebabCasePipe implements PipeTransform {
  transform(value: string): string {
    return value.toLowerCase().replace(/\ /g, '-');
  }
}
```

# Сервис

Сервис - это класс, который является поставщиком данных. Сервисы инкапсулируют бизнес логику приложения. Сервисы могут предоставлять интерфейс взаимодействия между отдельными не связанными друг с другом компонентами.

- Реализуются в виде отдельных классов в соответствии с принципами ООП.
- Компонент может делегировать какие-либо из своих задач сервисам.
- Доступ компонентов к сервисам реализуется с помощью DI.

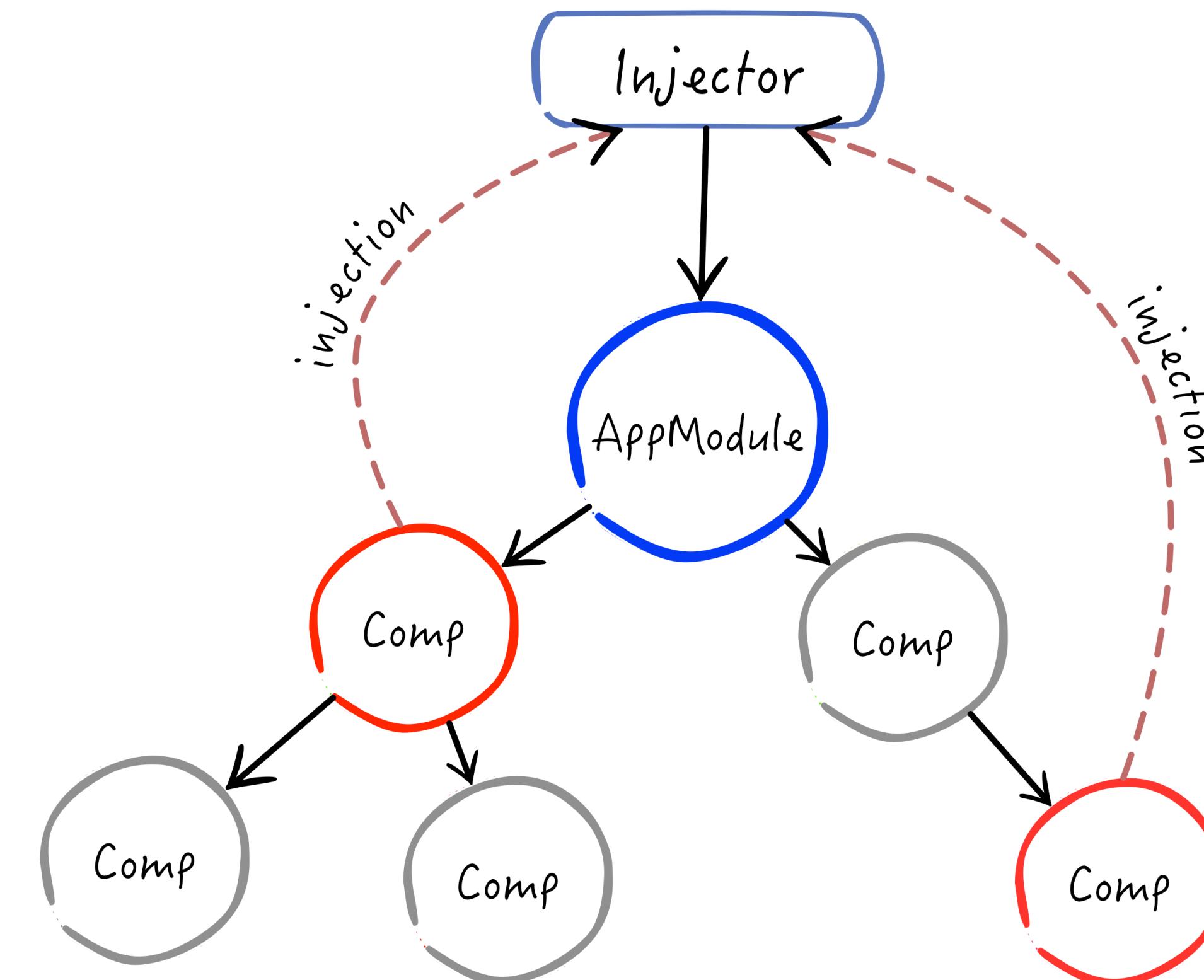
# Пример сервиса

```
1  @Injectable({ providedIn: 'root' })
2  export class StatesService {
3      private _filtersState: any = {
4          accounts: {
5              all: true,
6              opened: false,
7          },
8          deposits: {
9              all: true,
10             opened: false,
11         },
12     };
13
14    getFilters(): any {
15        return this._filtersState;
16    }
17 }

1  @Injectable({ providedIn: 'root' })
2  export class AccountsHttpService {
3      constructor(private http: HttpClient) {}
4
5      getUsers(): Observable<any> {
6          return this.http.get('/api/users');
7      }
8  }
```

# Dependency injection

- Компоненты могут использовать сервисы с помощью DI.
- Для того, чтобы класс можно было использовать с помощью DI, он должен содержать декоратор `@Injectable()`.



# Основные принципы реализации DI

- Приложение содержит как минимум один глобальный Injector, который занимается DI.
- Injector создаёт зависимости и передаёт их экземпляры контейнеру (container).
- Провайдер (provider) -- это объект, который сообщает Injector'у, как получить или создать экземпляр зависимости.
- Обычно провайдером сервиса является сам его класс.
- Зависимости компонентов указываются в качестве параметров их конструкторов

# Провайдеры (providers) для сервисов

Для каждого сервиса должен быть зарегистрирован как минимум один провайдер.

Способы задания провайдера:

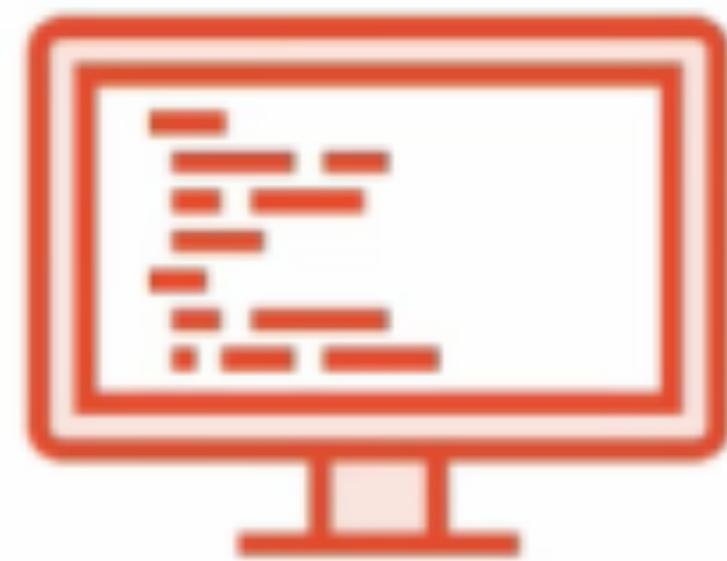
- в метаданных самого сервиса

```
1 | @Injectable({providedIn: 'root'})
```

- в метаданных компонента

```
1 | @Component({
2 |   selector: 'accounts-list',
3 |   providers: [AccountsHttpService],
4 |   template: `<div>My accounts</div>`
5 | })
```

# Angular Bindings / Привязка данных



DOM



Component

`{{expression}}`

Interpolation

`[property] = “expression”`

One Way Binding

`(event) = “statement”`

Event Binding

`[(ngModel)] = “property”`

Two Way Binding

# Two way binding

```
1 | <contacts-item [(name)]="contactPerson"></contacts-item>
```

Привет

Привет



# Change detection

**ChangeDetection** - это механизм в Angular, который отвечает за изменение выражений в шаблонах при их изменении в моделях.

**ChangeDetectionStrategy:** OnPush, Default.

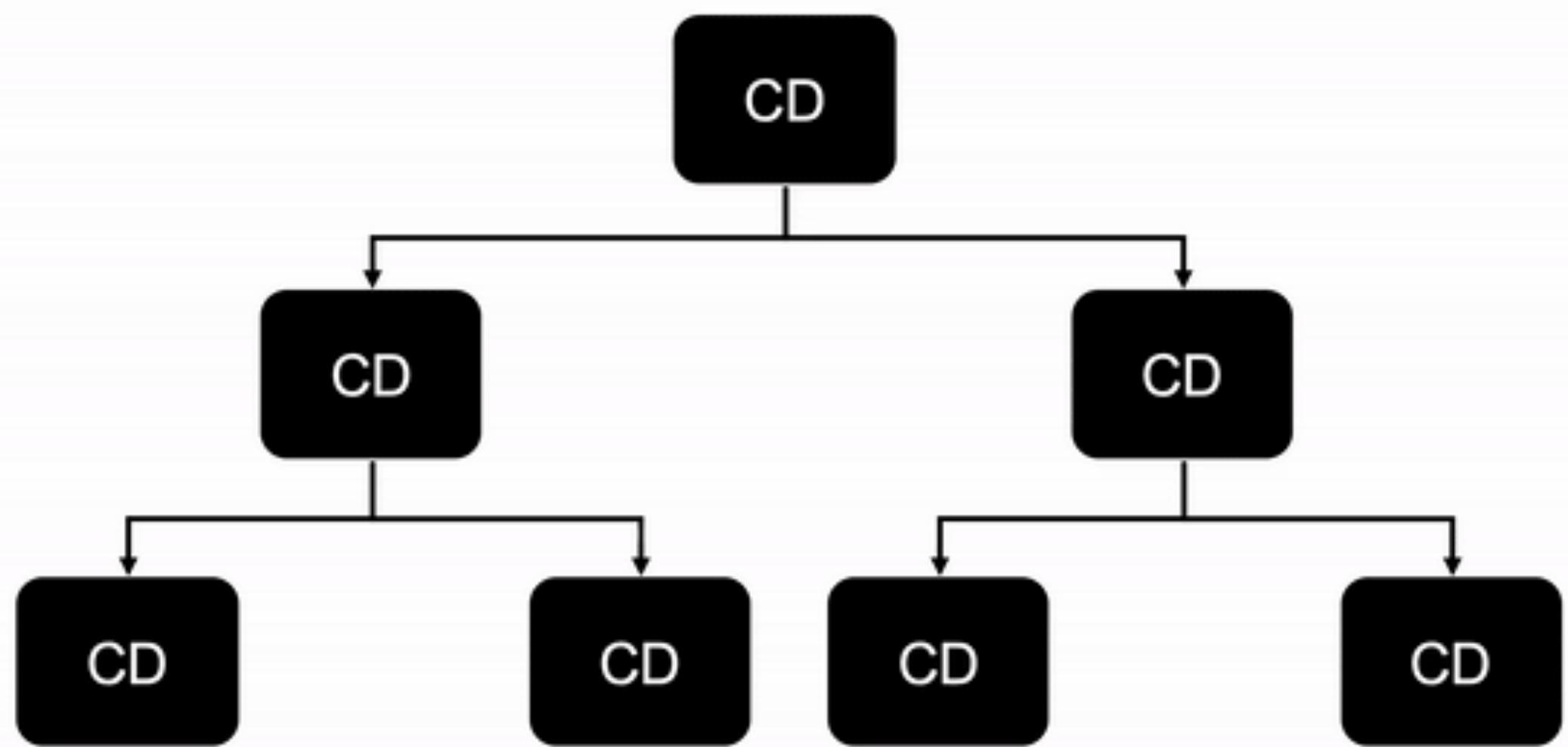
Сервис **ChangeDetectorRef** позволяет взять управление механизмом отслеживания изменений полностью под свой контроль.

Основные методы: detach, detectChanges, reattach

# Change Detection

Проще представлять его в виде дерева компонентов, где на самом верху находится корневой компонент. Похожие механизмы реализованы не только в Angular, но и в других современных инструментах.

Важно понимать этот механизм, так как он влияет на производительность, да и в целом дает представление о том, как работает ваше приложение.



# Change detection

- **ChangeDetectionStrategy.Default** – механизм отслеживания изменений запускается при любом действии пользователя или изменения состояния приложения
- **ChangeDetectionStrategy.OnPush** – подразумевает, что механизм отслеживания изменений запустится только в момент создания компонента (стадия жизненного цикла OnChanges) и при каждом последующем изменении его входных свойств (предваряются декоратором @Input() и передаются извне).
- <https://jeanmeche.github.io/angular-change-detection/>

# Роутинг



ANGULAR  
ROUTER

# Маршруты

При определении маршрута можно указать ряд свойств:

- **path** — наименование маршрута;
- **component** — компонент для отображения при переходе на URL, совпадающий с path;
- **children** — одно из дополнительных свойств, объединяющее в себе группу маршрутов относительно текущего;
- **redirectTo** — перенаправляет на указанный URL при попадании на маршрут, указанный в path;

```
1 const routes: Routes = [
2   { path: 'login', component: LoginRouteComponent },
3   {
4     path: 'home',
5     component: HomeRouteComponent,
6     children: [
7       {
8         path: 'profile',
9         component: ProfileRouteComponent,
10        },
11      ],
12    },
13    {
14      path: 'contacts',
15      redirectTo: '/home',
16      pathMatch: 'full',
17      children: [
18        {
19          path: 'director',
20          component: DirectorContactsRouteComponent,
21        },
22      ],
23    },
24    { path: '**', component: LoginRouteComponent },
25  ];
26
27 @NgModule({
28   imports: [RouterModule.forRoot(routes)],
29   exports: [RouterModule],
30 })
31 export class AppRoutingModule {}
```

# Маршруты

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

```
1 | @NgModule({
2 |   imports: [
3 |     AppRoutingModule
4 |   ],
5 |   declarations: [
6 |     AppComponent,
7 |     LoginRouteComponent,
8 |     HomeRouteComponent,
9 |     ProfileRouteComponent,
10 |     DirectorContactsRouteComponent
11 |   ],
12 |   providers: [],
13 |   bootstrap: [AppComponent],
14 |   exports: []
15 | })
```

# Router outlet

```
1 <div class="wrapper">
2   <app-nav></app-nav>
3
4   <main>
5     <router-outlet></router-outlet>
6   </main>
7
8   <app-footer></app-footer>
9 </div>
```

# Router

navigate(). В качестве первого параметра он принимает массив, где задается URL, а в качестве второго – объект с дополнительными параметрами запрашиваемого маршрута:

```
1 this.router.navigate(['profile', 3], {  
2   queryParams: { id: 3 },  
3   fragment: 'address',  
4 });
```

```
1 <li>  
2   <a routerLink="profile" routerLinkActive="active-link"  
3     >Profile</a  
4   >  
5 </li>
```

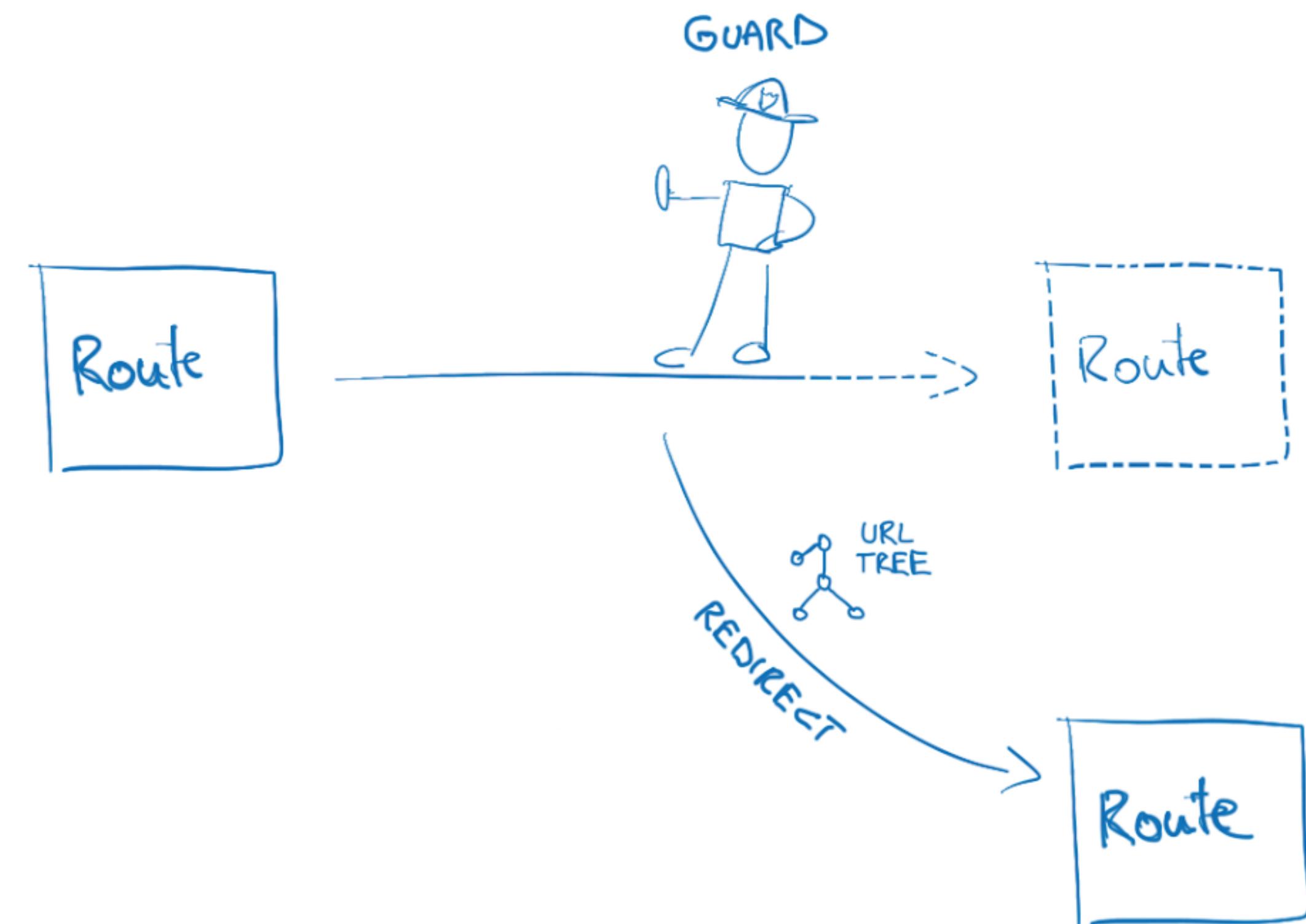
# ActivatedRoute

- url — наименование маршрута;
- params — параметры Angular маршрутизации и их значения, указываемые при определении маршрута, например, id в /profile/:id;
- queryParams — параметры строки запроса, например, id в /profile?id=3;
- fragment — значение hash, например, address в /profile#address;
- data — объект одноименного свойства, указываемого при определении маршрута.

```
@Component({
  selector: 'app-profile',
  templateUrl: './profile.component.html',
  styleUrls: ['./profile.component.scss'],
})
export class ProfileComponent {
  constructor(private route: ActivatedRoute) {
    console.log(this.route);
  }
}
```

## Ограничение доступов к маршруту

Route Guards позволяют ограничить доступ к маршрутам на основе определенного условия, например, только авторизованные пользователи с определенным набором прав могут просматривать страницу.



# Route Guards

Различают следующие виды guard-ов:

- CanActivate – разрешает/запрещает доступ к маршруту;
- CanActivateChild – разрешает/запрещает доступ к дочернему маршруту;
- CanDeactivate – разрешает/запрещает уход с текущего маршрута;
- Resolve – выполняет какое-либо действие перед переходом на маршрут, обычно ожидает данные от сервера;
- CanLoad – разрешает/запрещает загрузку модуля, загружаемого асинхронно.

# Создание Guards

```
1 @Injectable()
2 export class AuthGuard
3   implements CanActivate, CanActivateChild {
4     constructor(
5       @Inject(AuthService) private auth: AuthService
6     ) {}
7
8     canActivate(
9       next: ActivatedRouteSnapshot,
10      state: RouterStateSnapshot
11    ): boolean {
12       return this.auth.isLoggedIn;
13     }
14
15     canActivateChild(
16       next: ActivatedRouteSnapshot,
17       state: RouterStateSnapshot
18     ): boolean {
19       return this.canActivate(next, state);
20     }
21 }
```

```
1 const routes: Routes = [
2   { path: 'login', component: LoginComponent },
3   {
4     path: 'pages',
5     component: PagesComponent,
6     canActivate: [AuthGuard],
7     canActivateChild: [AuthGuard],
8     children: [
9       { path: 'about', component: AboutComponent },
10      {
11        path: 'contacts',
12        component: ContactsComponent,
13      },
14    ],
15  },
16];
17
18 @NgModule({
19   imports: [RouterModule.forChild(routes)],
20   exports: [RouterModule],
21 })
```

# Создание Guards

```
@Injectable()
class UserToken {}

@Injectable()
class PermissionsService {
  canActivate(currentUser: UserToken, userId: string): boolean {
    return true;
  }
  canMatch(currentUser: UserToken): boolean {
    return true;
  }
}

const canActivateTeam: CanActivateFn = (
  route: ActivatedRouteSnapshot,
  state: RouterStateSnapshot,
) => {
  return inject(PermissionsService).canActivate(inject(UserToken), route.params['id']);
};

bootstrapApplication(App, {
  providers: [
    provideRouter([
      {
        path: 'team/:id',
        component: TeamComponent,
        canActivate: [canActivateTeam],
      },
    ]),
  ],
});
```

# Взаимодействие с back-end'ом

- Реализуется с помощью сервиса HttpClient.
- Для его использования необходимо импортировать модуль HttpClientModule в свой AppModule.
- После импорта модуля можно заинжектить HttpClient

```
1 @Injectable()
2 export class DataService {
3     constructor(private http: HttpClient) {}
4 }
```

# Пример взаимодействие с back-end'ом

```
1 //GET-запрос на получение списка счетов
2 getAccounts(){
3     return this.http.get('http://example.com/api/accounts');
4 }
5
6 //GET-запрос на получение счета по id, id передается как GET-параметр
7 getAccountByID(id: number | string){
8     return this.http.get('http://example.com/api/accounts', {
9         params: new HttpParams().set('id', id)
10    });
11 }
```

# Где почитать

Официальная документация

<https://angular.io/>

<https://v18.angular.dev/overview>

На русском

<https://metanit.com/web/angular2/>

Zone js in angular

<https://medium.com/@lukaonik/what-is-ngzone-in-angular-762580ae37f6>

Angular architecture

<https://angular-academy.com/angular-architecture-best-practices/>

Angular platform

<https://medium.com/nuances-of-programming/%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D1%8B-angular-%D0%B2-%D0%B4%D0%B5%D1%82%D0%B0%D0%BB%D1%8F%D1%85-%D1%87%D0%B0%D1%81%D1%82%D1%8C-1-%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D1%8B-angular-bc5002b46e52>

Doc на русском

<https://webdraftt.com/>

Doc RxJs

<https://rxjs-dev.firebaseio.com/>

Ngrx – redux for angular

<https://ngrx.io/>