**RESUMO**

**Ciclo de vida básico de uma aplicação angular:**

* **ngOnInit** = utilizado quando o componente é iniciado;
* **ngAfterViewInit** = utilizado quando a view do componente é carregada;
* **ngOnDestroy** = utilizado antes do componente ser destruído (tirado da memória);

**Change Detection:**

* Responsável por detectar mudanças na aplicação e refletir esse novo estado para o usuário;
* Processo de detecção ocorre primeiro nos componentes pais e vai para os componentes filhos. Por padrão todos os componentes da aplicação são verificados. Processo unidirecional;

**Zones:** áreas de código que são monitoradas para identificar quando uma ação assíncrona é realiza e assim chamar o change dection.

**Processo change detection:**

|  |  |
| --- | --- |
| **Processo** | **Local** |
| Atualiza os binds com o ChildComponent | ChildComponent |
| Chama ngOnInit, ngDoCheck e ngOnChanges | ChildComponent |
| Atualiza a view atual | ParentComponent |
| Roda Change Detection no ChildComponent | ChildComponent |
| Chama ngAfterViewChecked e ngAfterViewInit | ChildComponent |

**Presenter**

* Se preocupa em como as coisas são mostradas;
* Recebe informações do componente pai via Input;
* Emite eventos para o componente pai via Output;
* Não guardam estado;
* Podem ter persenters e containers dentro deles;
* Podem ser mais performáticos com o OnPush;

**Container**

* Se preocupa em como as coisas funcionam;
* Consomem e geram informação de serviços;
* Disparam ações com base nos eventos do componente filho;
* Tem noção de estado;
* Podem ter presenters e containers dentro deles;

**Serviços**

* Servem para criar uma única instância de uma classe e a compartilha por todo modulo ou toda aplicação;

**Testes**

* Karma: serve para fazer testes no angular, de componentes, métodos e simular o uso do usuário;
* Jasmine: pacote que incorpora uma bilblioteca que permite o realizar testes sobre os effects do Ngrx;
* Cypress: serve parar testes e2e, end-to-end, para testes de caixa preta para simular um usuário real. Tem sido visto melhor que o Protector que vem por padrão no angular;
* Jest: usado pelo Facebook, usa typescript e é mais rápido que jasmine. Usa o virtual DOM;

**Control Value Access**

* Serve para que um componente filho tenha acesso aos valores de um formulário presente no componente pai sem a necessidade de @Inputs. Quando há alteração de um control do formulário, sua aplicação é replicada no filho. Para realizar isso deve-se injetar um formControl no filho e implementar a interface ControlValueAccessor;

**Directivas**

* São tags de marcação que não são exibidas quando o angular roda a aplicação. Servem como locais para pegar um contexto vindo de um componente pai e de acordo com uma verificação lógica (true ou false) exibe o que está dentro da diretiva. Você pode criar suas próprias directivas se necessário. EX: \*ngIf, \*ngFor e \*ngSwitch, que originalmente é definido como “<ng-template [ngIf]=”true”><modulo-exibido-aqui></ng-template>;

**Injeção de Dependências**

* Padrão de design;
* Dependências são serviços ou objetos que uma classe precisa para executar alguma ação. No angular isso permite que um componente use uma dependência (serviço por exemplo) sem instanciá-lo manualmente;
* O injetor de dependência funciona através de tokens (conjunto chave: valor). Esse token pode ser definido dentro dos providers de maneira manual, ou usando a sintaxe sugar;
* **Os injetores** de dependência do angular são responsáveis por armazenar os tokens, instanciar as classes e compartilhar essas classes para os componentes que dependem desse injetor. Se comunicam de maneira hierárquica sempre de baixo para cima;
* Para cada Modulo o Angular cria um injetor do tipo ModuleInjector e para cada componente cria um injetor ElementInjector;
* **Providers:** determinam o valor de um dado token; normalmente instanciam uma classe do injector, ou valor literal;