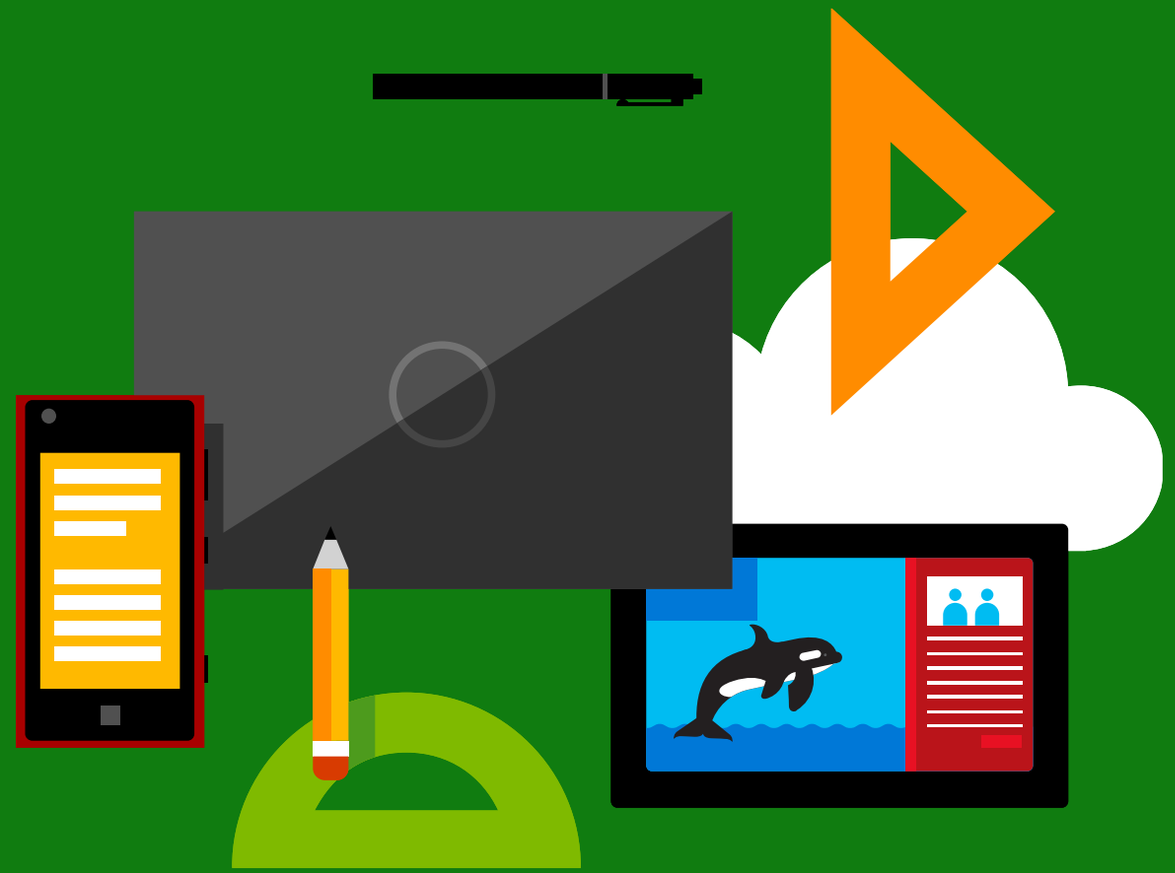
Desenvolvimento Web

Typescript e Angular

Instrutor: Júlio Pereira Machado (julio.machado@pucrs.br)

Angular
e Forms



Entrada de Dados

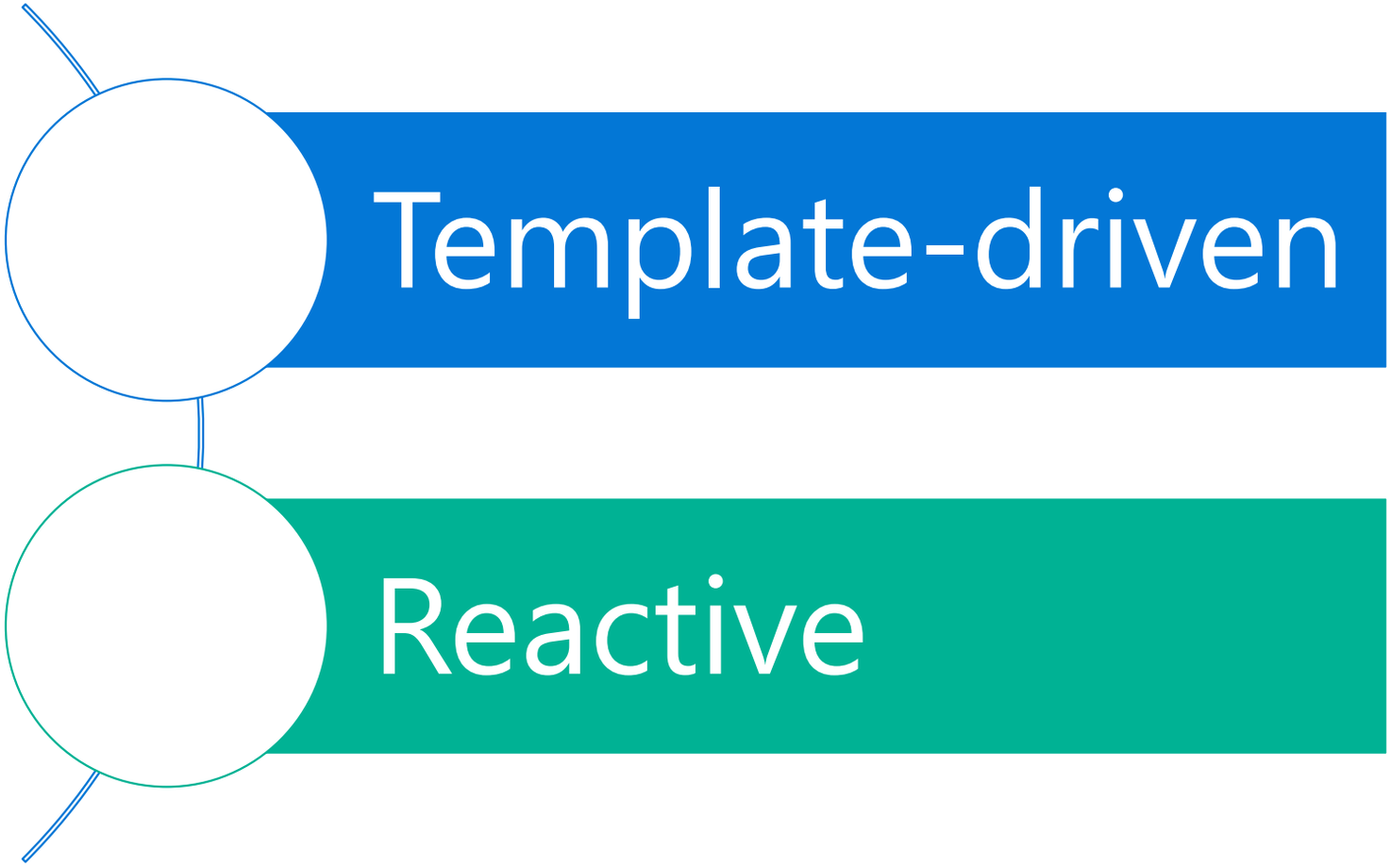
Controle manual

- Manipular referências a elementos do template
- Manipular eventos

Suporte a formulários

- *Two-way databinding*
- Validação de dados
- Controle do estado dos elementos do formulário
- Tratamento de erros

Formulários



Formulários

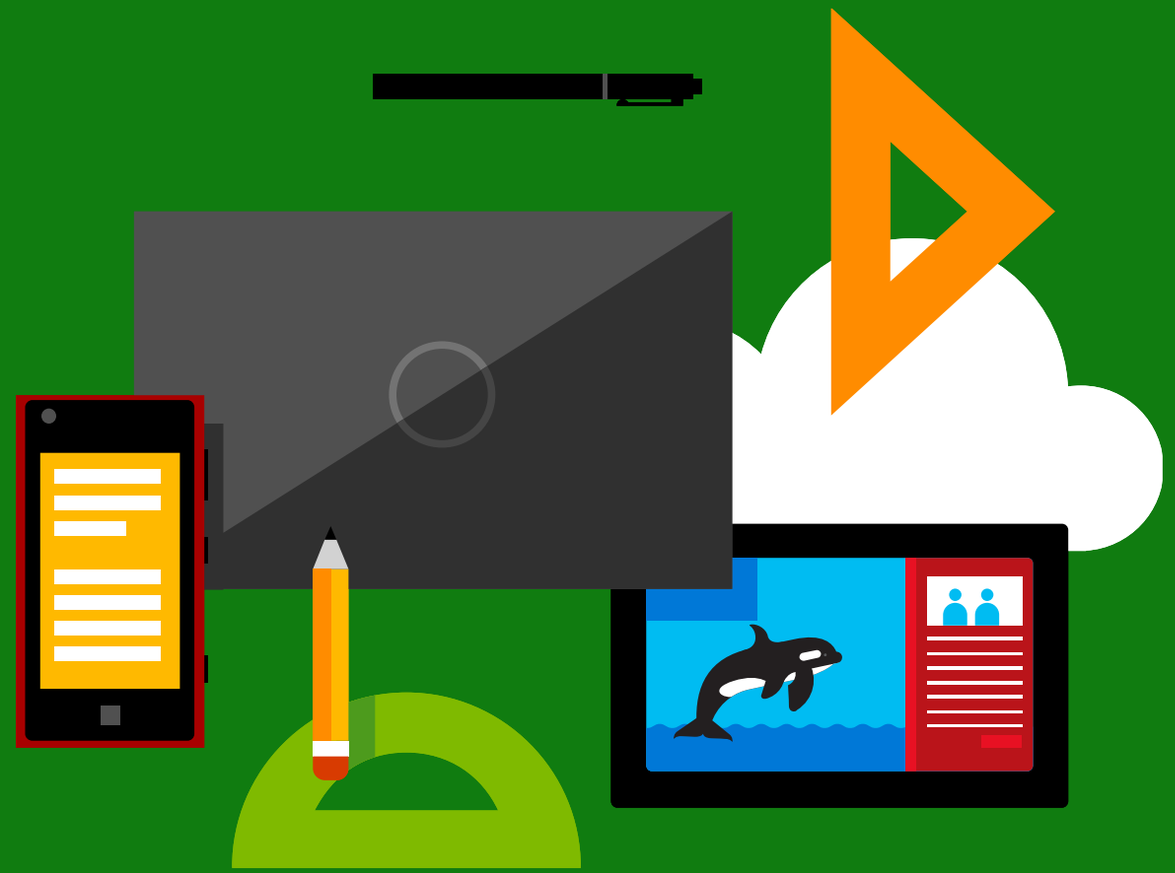
	Reactive	Template-driven
Processo de configuração	Explícito, criado no componente	Implícito, criado por diretivas na view
Modelo de dados	Estruturado e imutável	Não-estruturado e mutável
Fluxo de dados	Síncrono	Assíncrono
Validação de formulário	Funções	Diretivas
Escalabilidade	Reutilizáveis, facilidades para testar	Não-reutilizáveis, dificuldades para testar
Cenário	Aplicação com muitos formulários com componentes reutilizáveis	Aplicações com poucos formulários (uma ou duas páginas básicas)

Formulários

Componentes comuns:

- FormControl – mantém o valor e estado da validação de um controle individual de formulário
- FormGroup – mantém o valor e estado da validação de uma coleção de controles de formulário
- FormArray – mantém o valor e estado da validação de um array de controles de formulário de forma dinâmica
- ControlValueAccessor – cria uma ponte entre uma instância de um FormControl e os elementos nativos do DOM

Reactive Forms



Formulários - Reactive

- Características:
  - É a forma mais robusta para a criação de formulários
  - Modelo programático reativo
  - Componente é responsável pelo gerenciamento do fluxo de dados entre controles do formulário e *models*
  - Não utiliza vinculação de mão-dupla via NgModel
- Configuração:
  - Importar *ReactiveFormsModule* disponível no módulo JavaScript *@angular/forms*
  - Configurar dependência a *ReactiveFormsModule* no componente ou módulo
- Documentação:
  - https://angular.dev/guide/forms/reactive-forms
  - https://angular.dev/guide/forms/typed-forms
  - https://angular.dev/guide/forms/form-validation

Formulários - Reactive

- Exemplos: configuração em módulo

```
import { ReactiveFormsModule } from '@angular/forms';
…
@NgModule({
  imports: [ BrowserModule, ReactiveFormsModule ],
  declarations: [ AppComponent, FavoriteColorComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

Formulários - Reactive

- Exemplos: configuração em componente

```
import {Component} from '@angular/core’;
import {ReactiveFormsModule} from '@angular/forms’;
…
@Component({
  standalone: true,
  imports: [ReactiveFormsModule],
  …
})
export class FavoriteColorComponent {
  …
}
```

Formulários – Reactive (Construção)

- Exemplos:

```
import {Component} from '@angular/core’;
import {ReactiveFormsModule} from '@angular/forms’;
import {FormControl} from '@angular/forms';
@Component({
  standalone: true,
  imports: [ReactiveFormsModule],
  selector: 'app-reactive-favorite-color',
  template: `
    Favorite Color: <input type="text" [formControl]="favoriteColorControl">
  `,
})
export class FavoriteColorComponent {
  favoriteColorControl = new FormControl('');
}
```

Formulários – Reactive (Construção)

- Exemplos: acesso ao valor

```
<p>Favorite Color: {{ favoriteColorControl.value }} <p>
```

Formulários – Reactive (Construção)

- Classe **`FormControl`** é o elemento central dos formulários reativos
  - Controles são criados no componente de forma programática como instâncias de *FormControl*
  - Controle Angular é associado a um controle HTML no template da *view* via diretiva de atributo `[formControl]`
- Classe **`FormGroup`** organiza um agrupamento de controles em um formulário
  - *FormGroup* é associado a um formulário HTML no template da *view* via diretiva de atributo `[formGroup]`
  - Controles são criados no componente de forma programática como instâncias de *FormControl* associadas a um *FormGroup*
  - Cada controle HTML do formulário é associado a um controle no componente via diretiva de atributo `formControlName`
- Serviço **`FormBuilder`** facilita a criação dos controles no componente através de métodos auxiliares

Formulários – Reactive (Construção)

- Exemplos:

```
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'app-profile-editor',
  templateUrl: './profile-editor.component.html',
  styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent {
  profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
  });
}
```

Formulários – Reactive (Construção)

- Exemplos:

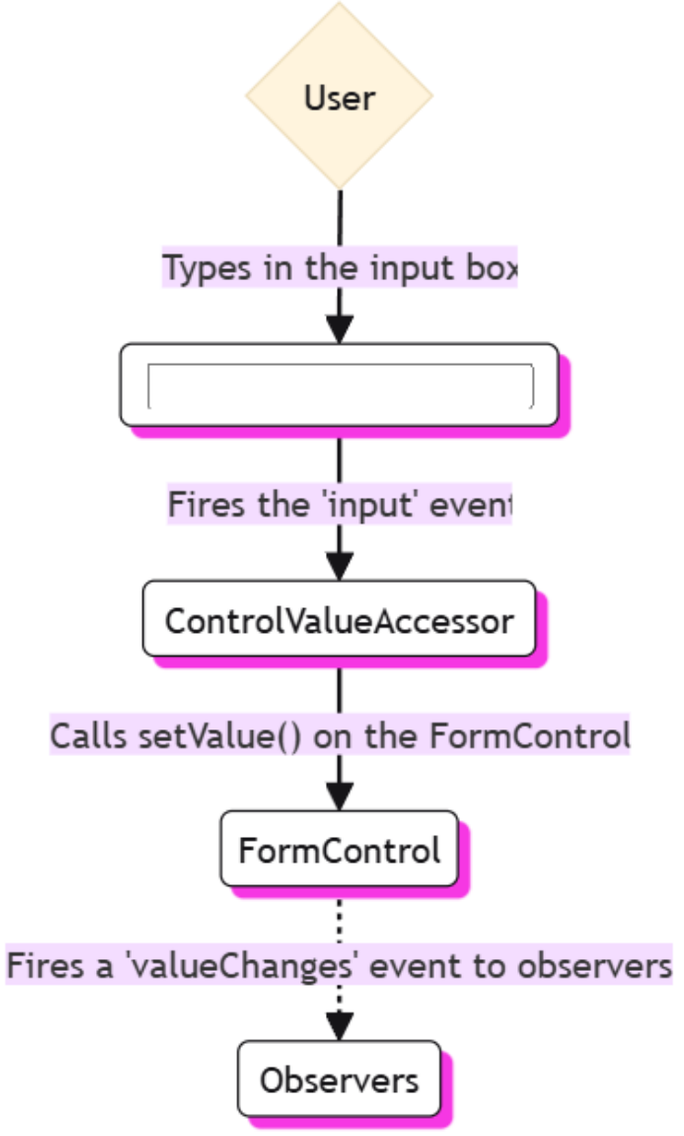
```
<form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
  <label>
    First Name:
    <input type="text" formControlName="firstName" required>
  </label>

  <label>
    Last Name:
    <input type="text" formControlName="lastName">
  </label>
…
</form>
```

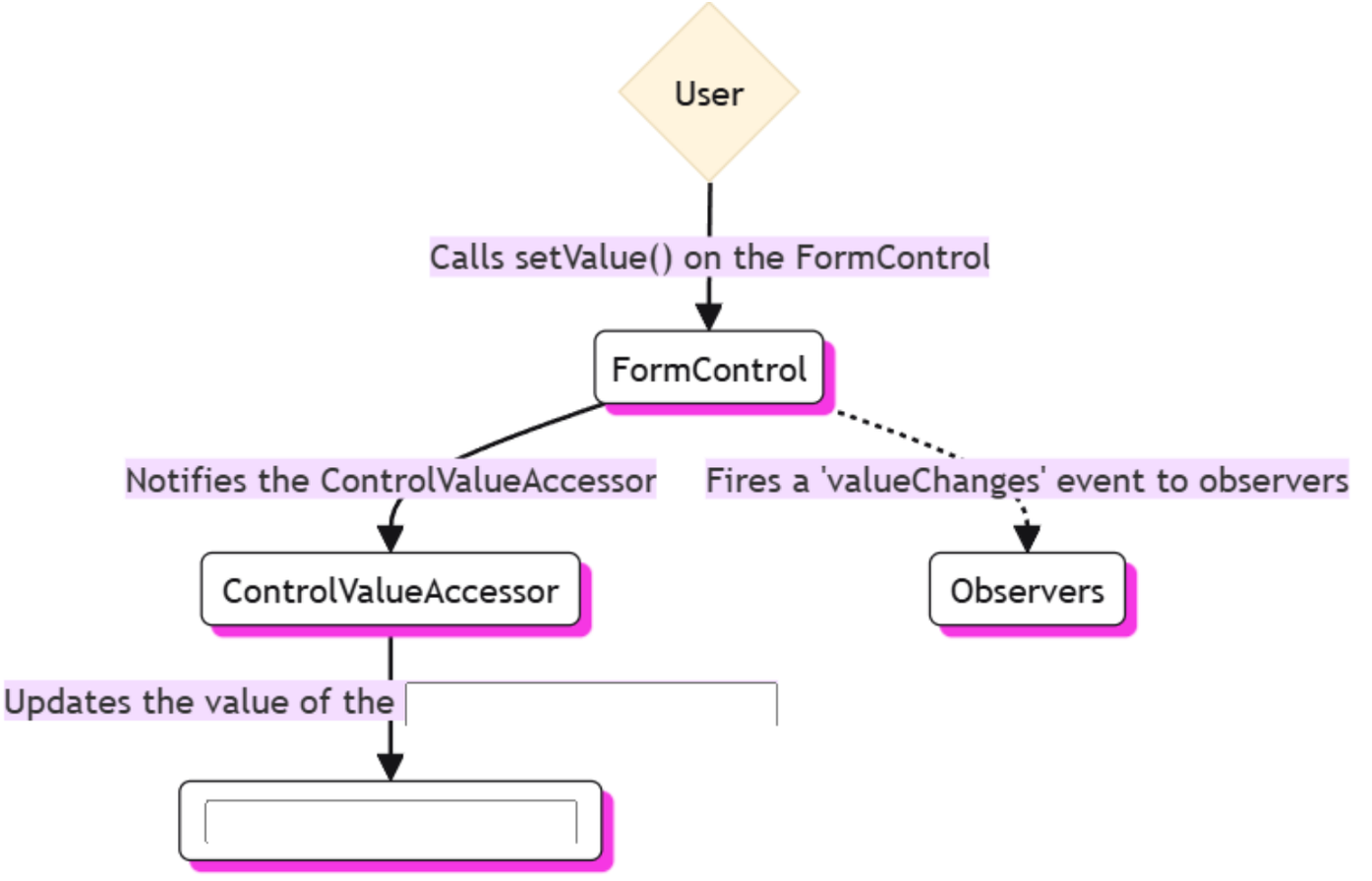
Formulários – Reactive (Construção)

- Exemplos:

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

@Component({
  selector: 'app-profile-editor',
  templateUrl: './profile-editor.component.html',
  styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent {
  constructor(private fb: FormBuilder) { }
  profileForm = this.fb.group({
  firstName: [''],
  lastName: [''],
});
}
```

Formulários – Reactive (Fluxo)





Formulários – Reactive (Fluxo)

- Valores de controles são lidos de duas maneiras:
  - Propriedade `valueChanges` expõe um *Observable*
    - Consumir no template via *AsyncPipe* e no componente via *subscribe()*
  - Propriedade `value` expõe valor atual
- Valores de controles são alterados programaticamente através de diversos métodos:
  - Método `setValue()` permite alterar o valor de um único controle programaticamente no lado do componente
  - Método `patchValue()` permite alterar qualquer coleção de propriedades no objeto que sofreu alteração

Formulários – Reactive (Validação)

- Angular fornece funções de validação
  - Angular permite criar validações customizadas
- Validadores são de dois tipos:
  - Síncronos – imediatamente retornam erros de validação ou null; configurados como segundo argumento de FormControl
  - Assíncronos – retornam objetos Promise ou Observable com erros de validação ou null; configurados como terceiro argumento de FormControl
- Em formulários reativos, o controle deve ser configurado com um validador associado
  - Angular fornece a classe `Validators` com diversas propriedades estáticas que retornam validadores
- Documentação:
  - https://angular.dev/guide/forms/form-validation#validating-input-in-reactive-forms
  - https://angular.dev/api/forms/Validators

Formulários – Reactive (Validação)

- Exemplos:

```
heroForm = new FormGroup({
  'name': new FormControl(this.hero.name, [
      Validators.required,
      Validators.minLength(4)
  ]),
  'power': new FormControl(this.hero.power, Validators.required)
});

get name() { return this.heroForm.get('name'); }

get power() { return this.heroForm.get('power'); }
```

Formulários – Reactive (Validação)

- Exemplos:

```
<input id="name" class="form-control"
      formControlName="name" required >

<div *ngIf="name.invalid && (name.dirty || name.touched)"
    class="alert alert-danger">

  <div *ngIf="name.errors.required">
    Name is required.
  </div>
  <div *ngIf="name.errors.minlength">
    Name must be at least 4 characters long.
  </div>
</div>
```

Formulários – Reactive (Submissão)

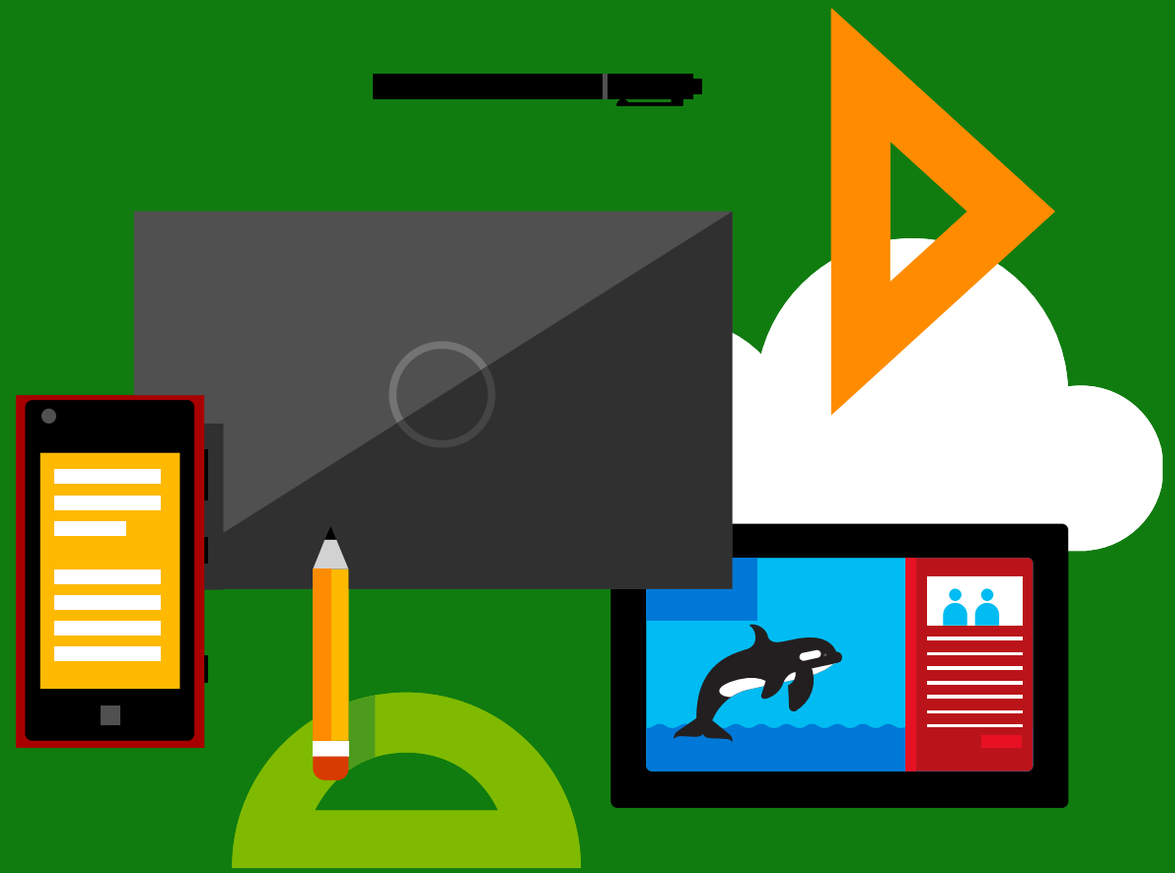
- Controle de botão do tipo *submit* é o elemento HTML responsável por submeter um formulário
  - Para vincular uma ação do componente ao evento de submissão de um formulário, usar `(ngSubmit)="tratadorEvento()"` no elemento `<form>`
- É usual controlar o estado do botão de submissão vinculado ao estado de validação do formulário
  - Impedir a submissão de formulários inválidos

Formulários – Reactive (Submissão)

- Exemplos:

```
<form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
…
  <button type="submit"
  [disabled]="!profileForm.valid">Submit</button>
</form>
```

Template-driven Forms



Formulários - Template

- Características:
  - É a forma mais simples de criação de formulários
  - É baseado em templates das views para a construção de formulários
  - Utiliza diretivas para conectar os elementos do formulário HTML e os objetos de model fornecidos pelos componentes
- Configuração:
  - Importar *FormsModule* disponível no módulo JavaScript *@angular/forms*
  - Configurar dependência a *FormsModule* no componente ou no módulo que conterá os componentes
- Documentação:
  - https://angular.dev/guide/forms/template-driven-forms
  - https://angular.dev/guide/forms/form-validation

Formulários - Template

- Exemplos: configuração em módulo

```
import { FormsModule } from '@angular/forms';
…
@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, FavoriteColorComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

Formulários - Template

- Exemplos: configuração em componente

```
import {Component} from '@angular/core’;
import {FormsModule} from '@angular/forms';
…
@Component({
  standalone: true,
  imports: [FormsModule],
  …
})
export class FavoriteColorComponent {
  …
}
```

Formulários – Template (Construção)

- Exemplos:

```
import {Component} from '@angular/core’;
import {FormsModule} from '@angular/forms’;
@Component({
  standalone: true,
  imports: [FormsModule],
  selector: 'app-reactive-favorite-color',
  template: `
    Favorite Color: <input type="text" [(ngModel)]="favoriteColor">
  `,
})
export class FavoriteColorComponent {
  favoriteColor = '';
}
```

Formulários – Template (Construção)

- Exemplos: acesso ao valor

```
<p>Favorite Color: {{ favoriteColor }} <p>
```

Formulários – Template (Construção)

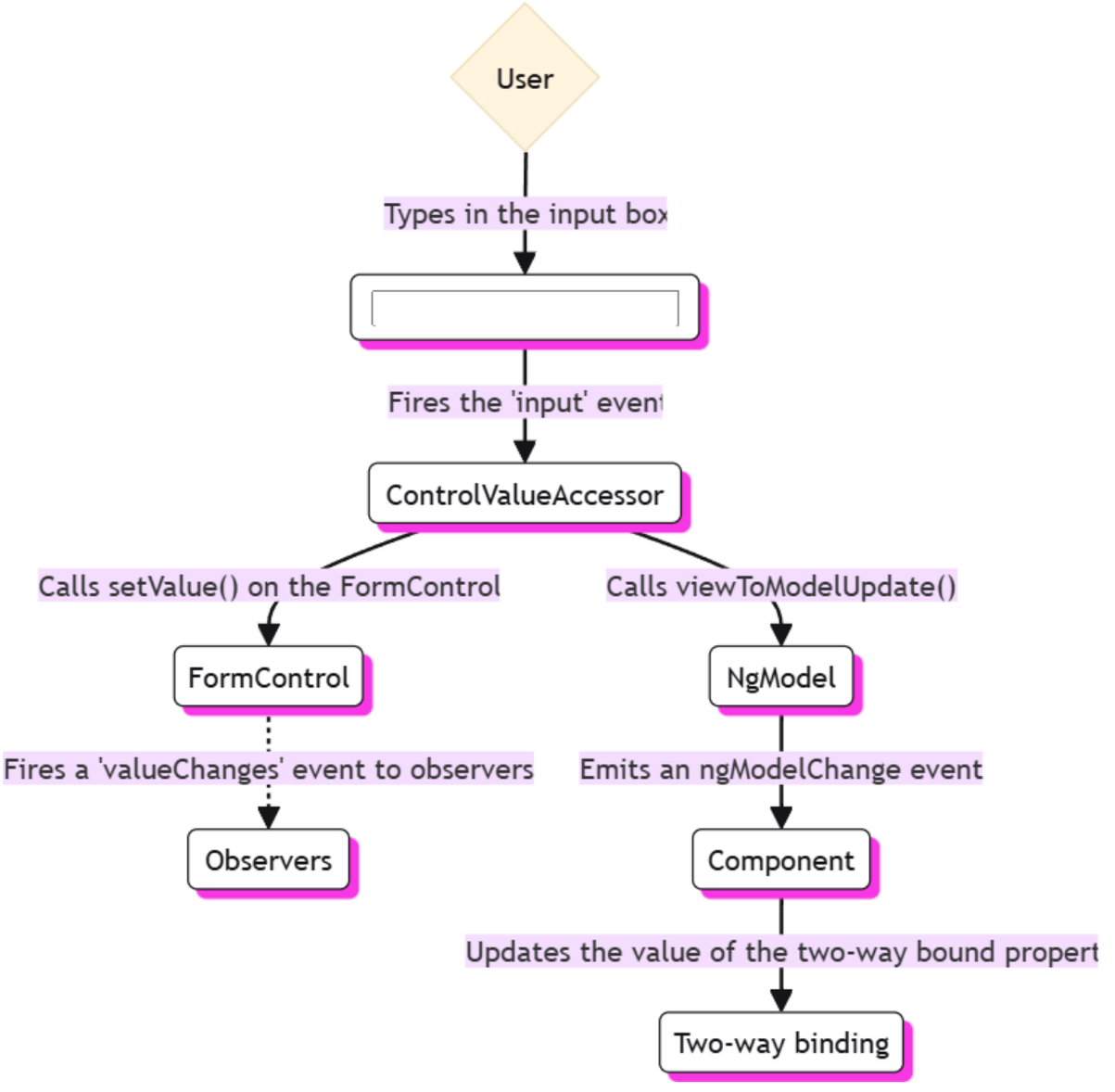
- Um formulário é construído dentro do template da view pela composição dos elementos HTML
  - <form>, <input>, <select>, <button>, etc
- Diretiva **`NgForm`** é aplicada pelo Angular ao elemento **`<form>`**
  - Fornece mecanismos de controle do estado do formulário, controle do estado de validação dos elementos do formulários e registro dos campos do formulário
  - Para acessar as propriedades da diretiva, definir uma variável de template `#nomeDoFormulario="ngForm"`

Formulários – Template (Construção)

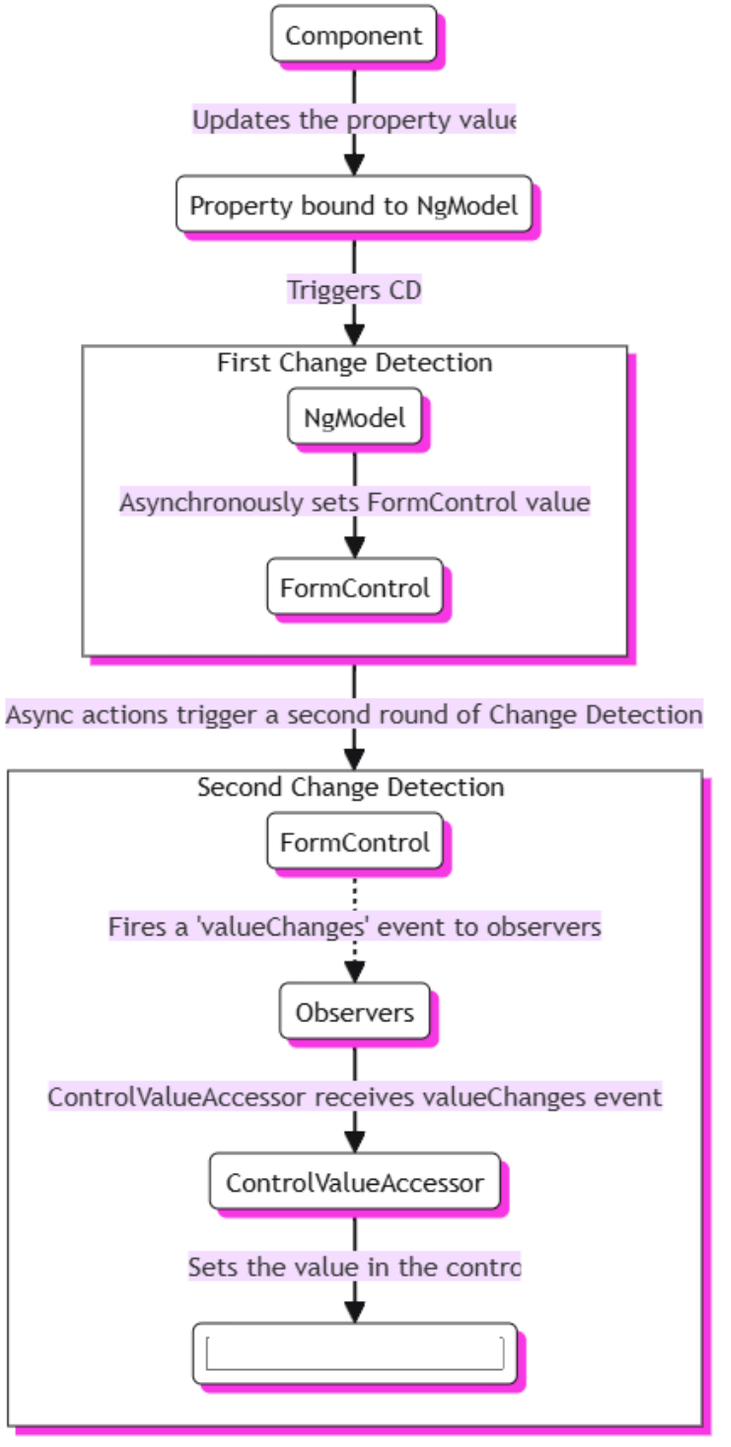
- Exemplos:

```
<h1>Hero Form</h1>
<form #heroForm="ngForm">
      <label for="name">Name</label>
      <input type="text" id="name" required>

      <label for="alterEgo">Alter Ego</label>
      <input type="text" id="alterEgo">

      <button type="submit">Submit</button>
</form>
```

Formulários – Template (Fluxo)

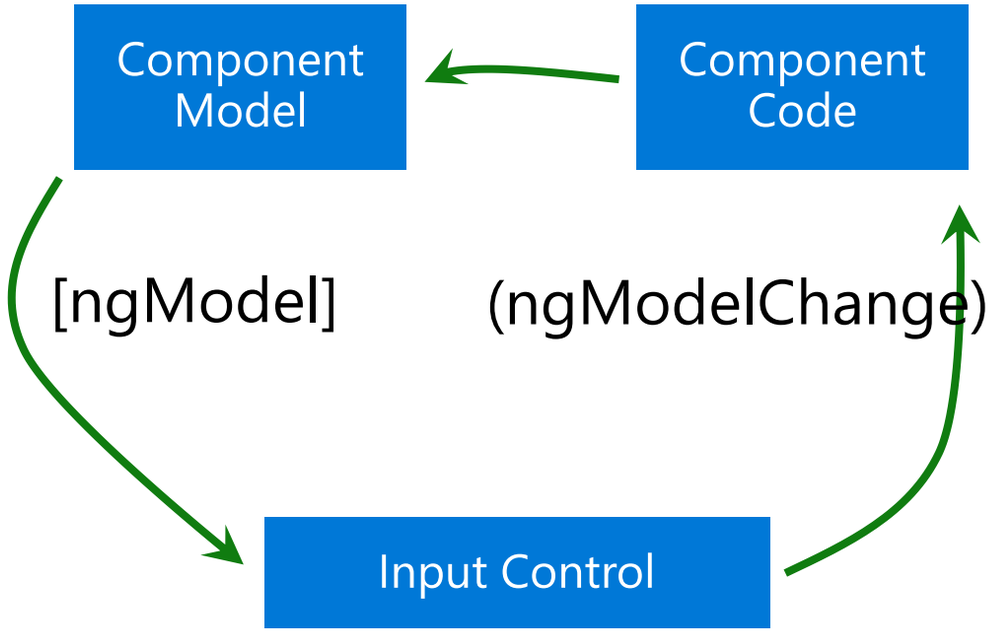


Formulários – Template (Fluxo)

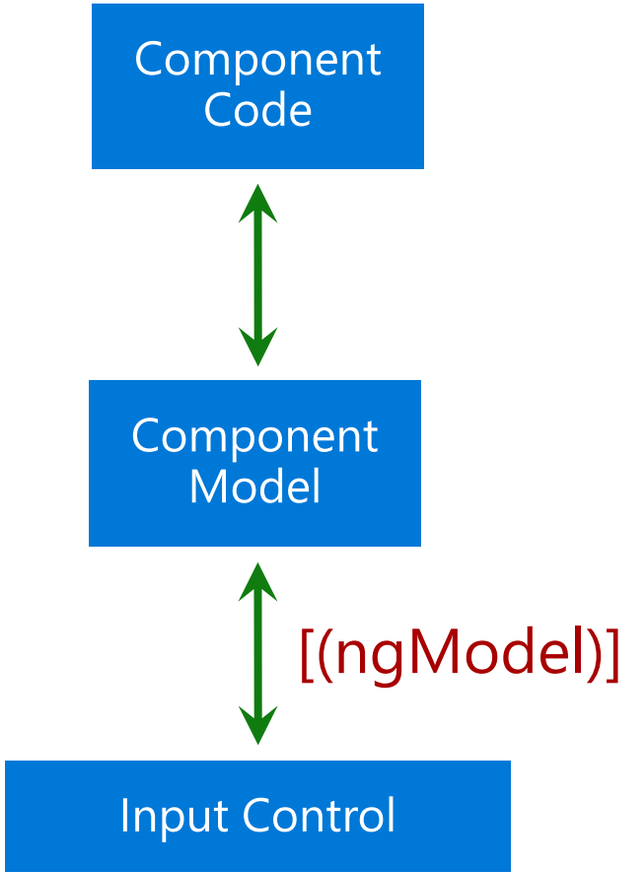


Formulários – Template (Fluxo)

- A diretiva **`NgModel`** conecta a propriedade de um *model* a um controle do formulário



One-Way Data Binding



Two-Way Data Binding

Formulários – Template (Fluxo)

- Exemplos:

```
<h1>Hero Form</h1>
<form #heroForm="ngForm">
        <label for="name">Name</label>
        <input type="text" id="name" required
        [(ngModel)]="model.name" name="name">
…
</form>
```

Importante!
Diretiva *ngModel* vem junto com atributo *name* para registrar o elemento HTML no formulário Angular

```
@Component({
…
})
export class HeroFormComponent {
  model: Hero;
}
```

Formulários – Template (Controle de Estado)

- Diretiva **`NgModel`**, além da vinculação de dados, fornece o controle do estado de cada controle do formulário e permite manipular o estilo CSS em função do estado
  - Para acessar as propriedades da diretiva, definir uma variável de template `#nomeDoControle="ngModel"`

ESTADO	CLASSE - VERDADEIRO	CLASSE - FALSO
Controle foi visitado touched, untouched	ng-touched	ng-untouched
Valor do controle foi modificado dirty, pristine	ng-dirty	ng-pristine
Valor do controle é válido valid, invalid	ng-valid	ng-invalid

Formulários – Template (Controle de Estado)

- Exemplos:

```
<h1>Hero Form</h1>
<form #heroForm="ngForm">
        <label for="name">Name</label>
        <input type="text" id="name" required
        [(ngModel)]="model.name" name="name"
        #name="ngModel">
        <div [hidden]="name.valid || name.pristine">
          Name is required
        </div>
…
</form>
```

Formulários – Template (Validação)

- A validação de formulários utiliza a configuração das restrições do HTML
  - Veja exemplos em https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation
- Angular fornece diretivas "casadas" aos atributos do HTML
  - Angular permite criar validações customizadas
- Documentação: https://angular.dev/guide/forms/form-validation#validating-input-in-template-driven-forms

Formulários – Template (Validação)

- Validação ocorre no nível do controle e o status de validação é controlado pelo Angular e agregado ao status do formulário ou grupo de controles
  - Ou seja, se um único controle é inválido, todo o formulário é inválido
- Se necessário desabilitar validação do HTML5, aplicar atributo **`novalidate`** ao elemento <form>

Formulários – Template (Validação)

- Exemplos:

```
<h1>Hero Form</h1>
<form #heroForm="ngForm">
  <label for="name">Name</label>
  <input type="text" id="name" required minlength="4"
  [(ngModel)]="model.name" name="name" #name="ngModel">
  <div *ngIf="name.invalid && (name.dirty || name.touched)"
    class="alert alert-danger">
    <div *ngIf="name.errors.required">
      Name is required.
    </div>
    <div *ngIf="name.errors.minlength">
      Name must be at least 4 characters long.
    </div>
  </div>
…
</form>
```

Formulários – Template (Submissão)

- Controle de botão do tipo *submit* é o elemento HTML responsável por submeter um formulário
  - Para vincular uma ação do componente ao evento de submissão de um formulário, usar `(ngSubmit)="tratadorEvento()"` no elemento `<form>`
- É usual controlar o estado do botão de submissão vinculado ao estado de validação do formulário
  - Impedir a submissão de formulários inválidos

Formulários – Template (Submissão)

- Exemplos:

```
<h1>Hero Form</h1>
<form #heroForm="ngForm" (ngSubmit)="onSubmit()">
…
  <button type="submit"
  [disabled]="!heroForm.form.valid">Submit</button>
</form>
```