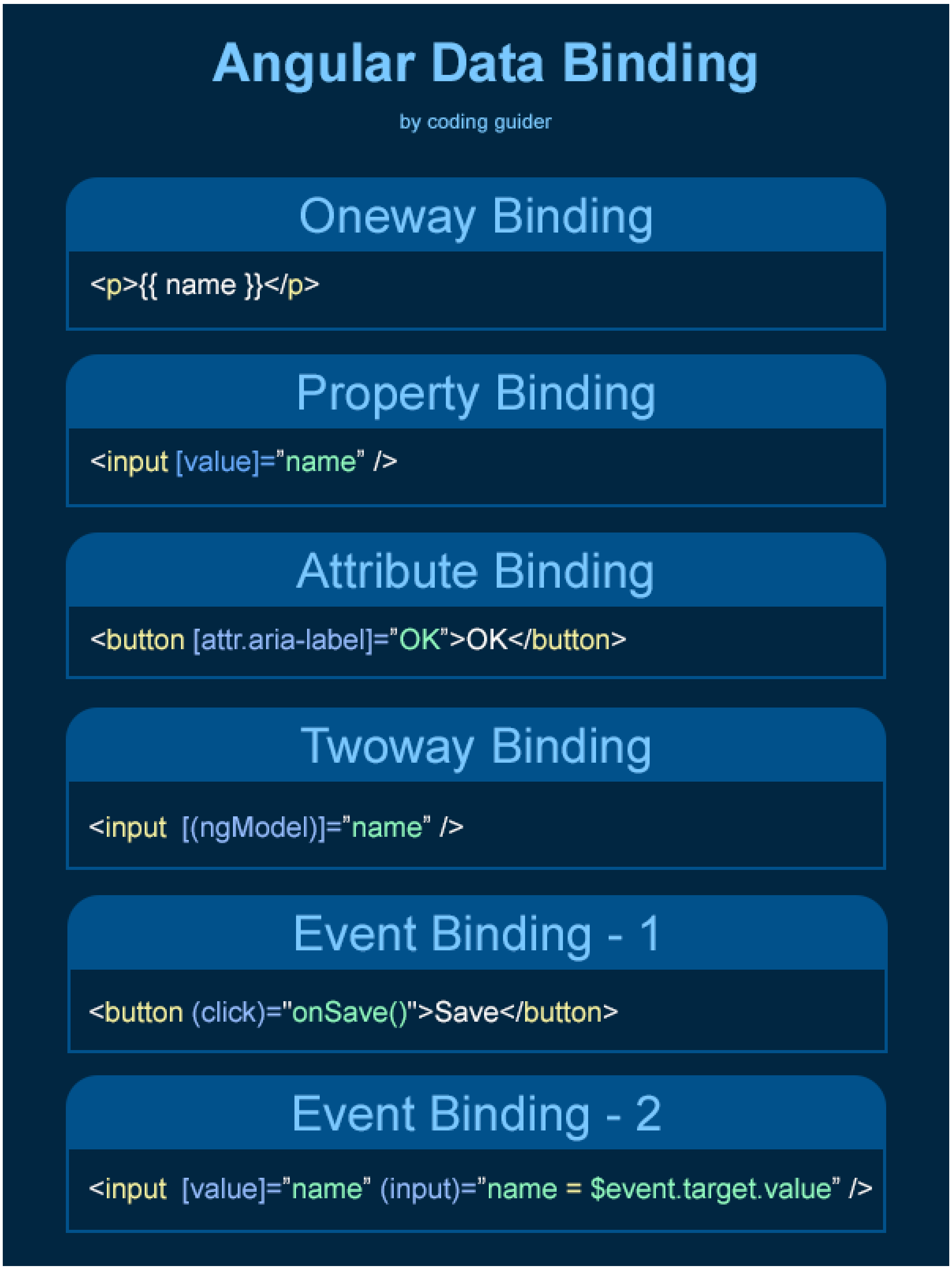
Forms

So far we've only looked at static website design and reading data. Forms allow users to input data. There are two different ways to do forms in Angular Template-driven and Reactive.

Also here's a cheatsheet on types of databinding in Angular, which i will go into more detail on later:



```
<p>{{ name }}</p>
```

```
<input [value]="name" />
```

```
<button [attr.aria-label]="OK">OK</button>
```

```
<input  [(ngModel)]="name" />
```

```
<button (click)="onSave()">Save</button>
```

```
<input  [value]="name" (input)="name = $event.target.value" />
```

Template-Driven Forms

- Template driven forms are the easier of the 2 to understand. The idea is pretty straightforward:

    - We use an HTML element, such as <input> and create variables in the backing typescript file to bind to the elements.
    - So as the value in an input changes the typescript variable is automatically updated
    - We don't actually submit the form to a server, but instead have a submit button which will be bound to a method in the typescript file that can read the variables to get the form's values
- Import the `FormsModule` in app.module.ts to interact with forms in different template driven ways

Template Form Validation

- Adding the HTML required tag will enable dynamic class processing on the field for some nice CSS visual cues. Ex: `<input type="text" class="form-control" id="name" placeholder="user name" [(ngModel)]="formUser.name" name="name" required>` Now when loading the page we see the following classes on that field when it is left blank: 4225e7e04d64a2cb1b977f203b437cca.png
- There are 2 styles of CSS Angular uses on form validation, ng-valid and ng-invalid, to indicate whether the form element passes validation.
- Other control styles:
    - ng-touched / ng-untouched - Tells whether user has touched the element
    - ng-pristine / ng-dirty - Tells whether the value has been edited
- We can change the behavior of these css styles in the css file

```
// When the input box is invalid
// and has been touched change border to red
input.ng-invalid.ng-touched {
  border: 1px solid #f00;
}
```

- We can also add template references so we can check for errors across a form or model

```
<form #userForm="ngForm">
...
<button type="submit" class="btn btn-primary" (click)="onSubmit()"
[disabled]="userForm.invalid">Save</button>
```

Reactive Forms

- Requires `ReactiveFormsModule` imported in app.module.ts

- We create an Object in the typescript that is bound to the HTML

```
 roomForm = new FormGroup({
   roomName : new FormControl('roomName')
 });
```

- We have an HTML form that has some controls

```
<form [formGroup]="roomForm">
  <div class="form-group">
    <label for="name">Name</label>
        <!-- Notice below the formControlName doesn't need to be bound with [] bc the formGroup has already
been applied -->
    <input type="text" class="form-control" id="name" placeholder="room name"
[formControlName]="roomName">
    <div class="alert alert-danger"></div>
  </div>
  <button type="button" class="btn btn-primary" (click)="onSubmit()">Save</button>
</form>
```

- To get the data into the forms we use a patch value which allows us to take each of the labels and provide a value for it

```
 ngOnInit(): void {
   this.roomForm.patchValue({
     roomName : this.room.name,
     location : this.room.location
   });
 }

 onSubmit(): void {
   this.room.name = this.roomForm.controls['roomName'].value;
   this.room.location = this.roomForm.value['location'];
   // TODO: Call a method in the dataService to save the room
 }
```

- We can inject the FormBuilder dependency into the constructor and use it to remove the patch values and form controls we added manually.

```
 constructor(private formBuilder: FormBuilder) {
}
```

```
ngOnInit(): void {
  this.roomForm = this.formBuilder.group({
    roomName : this.room.name,
    location : this.room.location
  });

  for (const layout of this.layouts) {
    const layoutCapacity = this.room.capacities.find( (lc) => lc.layout === Layout[layout]);
    const initialCapacity = layoutCapacity == null ? 0: layoutCapacity.capacity;
    this.roomForm.addControl(`layout${layout}`, this.formBuilder.control(initialCapacity));
  }
}
```

- And then the HTML can be simplified

```
 <div class="form-group" *ngFor="let layout of layouts">
  <label for="layout{{layout}}">{{ layoutEnum[layout] }}</label>
    <input type="number" class="form-control" id="layout{{layout}}" formControlName="layout{{layout}}">
</div>
```

- And validation is very simple in reactive forms. Angular gives us an object type called a validator

```
   this.roomForm = this.formBuilder.group({
    roomName : [this.room.name , Validators.required],
    location : [this.room.location, [Validators.required, Validators.minLength(2)]]
  });
```

Revision #2
Created 16 April 2022 23:24:24 by Elkip
Updated 16 April 2022 23:40:55 by Elkip