Enhance Web Development

Angular 2

Les formulaires

- Kezako
- Template driven forms
- Model driven forms
- Validations

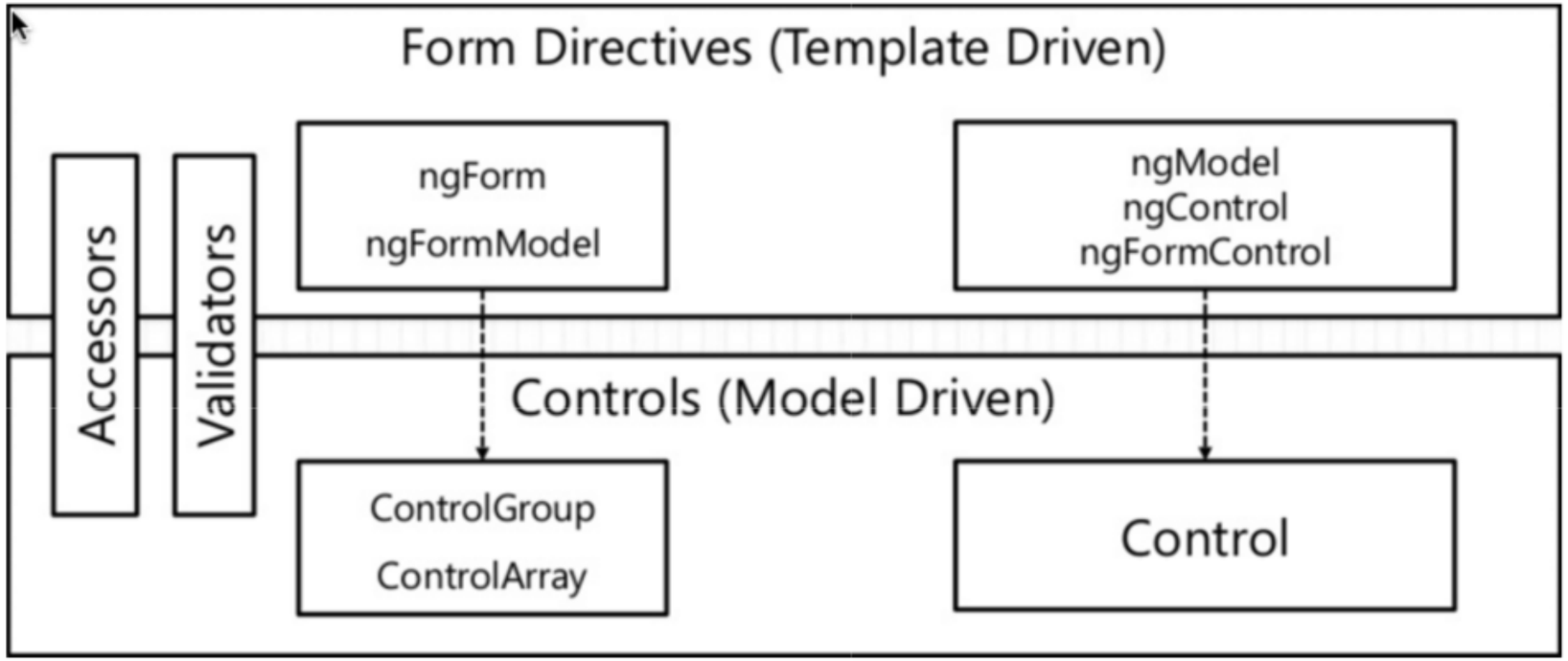
Kezako

- Les formulaires sont la première forme d'intéraction avec l'utilisateur pour mettre à jour des données
- Les formulaires angular 2 (model driven forms) remplacent ceux de angular 1 (template driven forms)

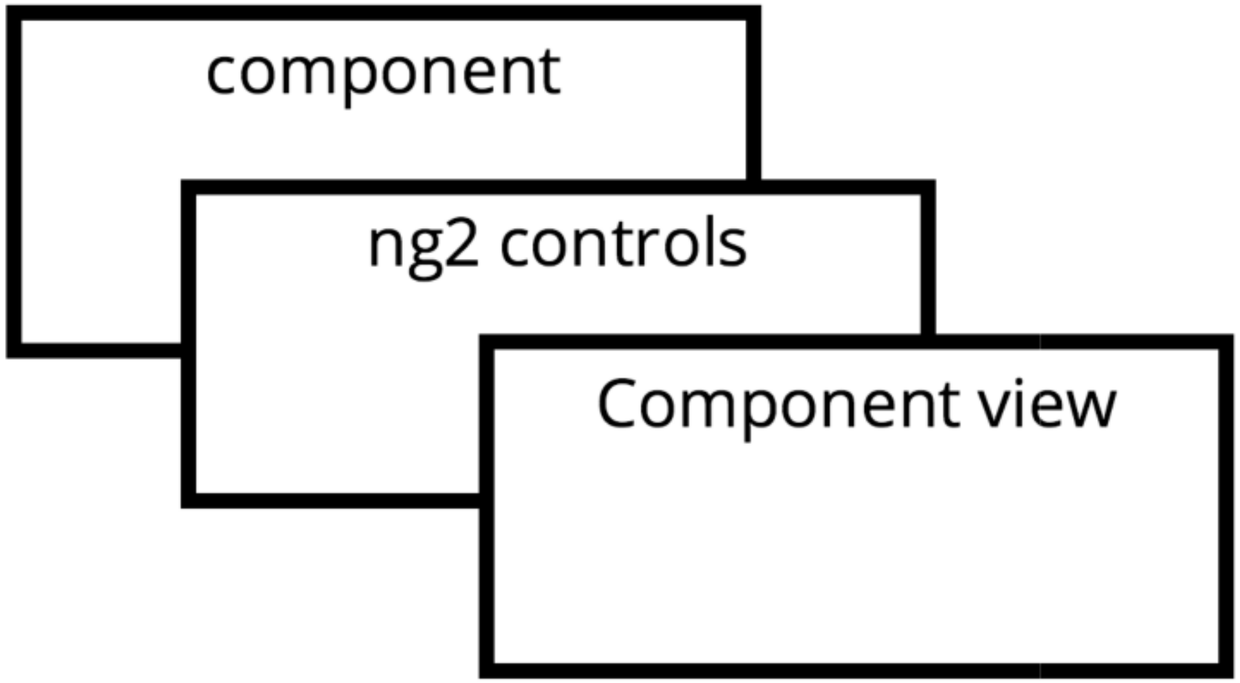
Configuration

- Les classes de formulaire se trouve dans :

  import { FORM_DIRECTIVES } from 'angular2/common'



Processus



- pristine
- dirty

- touched
- untouched

- errors
- valid

Template driven forms

ngModel

- Utilise la notation du two-way bindings : [(ngModel)]

  [(ngModel)]
- Une des seules directives à utiliser le two-way binding
- Il permet de lier un champs input avec un model

```
import { FORM_DIRECTIVES } from '@angular/common'

@Component({
  selector: "my-form",
  directives: [FORM_DIRECTIVES],
  template: `<input type="text" [(ngModel)]="name">`
})
class MyForm {
  name: string
}
```

ngModel

Validations

- Comme Angular 1, ngModel nous donne accès à l'état du formulaire et du champs

```
@Component({
  selector: "my-form",
  template: `
    <input type="text" [(ngModel)]="name" #field="ngModel">
    <pre>
      {{ field.valid }}
    </pre>
  `
})
class MyForm {
  name: string
}
```

ngModel

Model driven forms

Formulaires simples

- Utilisation de FormBuilder et de Validators dans le composant
- Utilisation des directives ngFormModel et ngControl dans la vue

```
<form [ngFormModel]="loginForm" (submit)="doLogin($event)">
  <input ngControl="email" type="email" placeholder="Your email">
  <input ngControl="password" type="password" placeholder="Your password">
  <button type="submit">Log in</button>
</form>
```

Formulaire simple : la vue

```
import { Component } from '@angular/core';
import { FormBuilder, Validators } from '@angular/common';

@Component({
  selector: 'login-page',
  templateUrl: 'login-page.html'
})
export class LoginPage {
  constructor(private _formBuilder: FormBuilder) {
    this.loginForm = _formBuilder.group({
      email: ["", Validators.required],
      password: ["", Validators.required]
    });
  }
  doLogin(event) {
    console.log(this.loginForm.value);
    event.preventDefault();
  }
}
```

Formulaire simple : le composant

ControlGroup

- Le ControlGroup nous permet de créer plusieurs Controls.
- Il est créer par le FormBuilder et le code suivant serait identique au précédent :

```
this.loginForm = new ControlGroup({
  email: new Control("email", Validators.required),
  password: new Control("password", Validators.required)
});
```

Directive form

- Plus aucun ngModel
- Le bind se fait au travers du FormBuild ou du ControlGroup

```
function containsTroll(c: Control) {
  if(c.value.indexOf('troll') >= 0) {
    return {
      noTroll: true
    }
  }
  return null
}
```

```
this.loginForm = _formBuilder.group({
  email: ['', containsTroll]
  password: ['', Validators.required],
});
```

Custom validator

Validations

Validateurs natifs

- Angular 2 propose 4 validations natives :
  - Validators.required
  - Validators.minLength
  - Validators.maxLength
  - Validators.pattern

```
// Component
this.name = new Control('', Validators.minLength(4));

// View
<input required type="text" ngControl="name" />
<div [hidden]="name.dirty && !name.valid">
   <p [hidden]="name.errors.minlength">
      Your name needs to be at least 4 characters.
   </p>
</div>
```

Validateurs natifs

Validateurs personnalisés

- Nos validateurs personnalisées doivent respecter :
  - retourner null si valide (ok ok ... bizarre)
  - Respecter l'interface suivante

```
interface ValidationResult {
 [key:string]:boolean;
}
```

```
// Component
function containsTroll(c: Control) {
  if(c.value.indexOf('troll') >= 0) {
    return {
      noTroll: true
    }
  }
  return null
}
this.name = new Control('', containsTroll);

// View
<input required type="text" ngControl="name" />
<div [hidden]="name.dirty && !name.valid">
   <p [hidden]="name.errors.noTroll">
      There is "troll" in your name ...
   </p>
</div>
```

Validateurs personnalisés

Validateurs asynchrone

- Fonctionne de la même façon qu'un validateur personnalisé, mais retourne une Promise
- S'initialise avec le troisième paramètre de

Control(name, validators, asyncValidators)

```
// Component
function exists(c: Control): Promise<ValidationResult> {
  let q = new Promise((resolve, reject) => {
    setTimeout(() => {
      if (c.value === 'George') {
        resolve({"usernameExists": true})
      } else {
        resolve(null)
      }
    }, 1000)
  })
  return q
}
this.name = new Control('', Validators.required, exists);
```

```
// View
<input required type="text" ngControl="name" />
<p [show]="name.pending">
  Fetching data from server!
</p>
<div [hidden]="name.dirty && !name.valid && !name.pending">
  <p [hidden]="name.errors.required">
    Your name is required!
  </p>
  <p [hidden]="name.errors.usernameExists">
    This username already exists! sad ;-(
  </p>
```

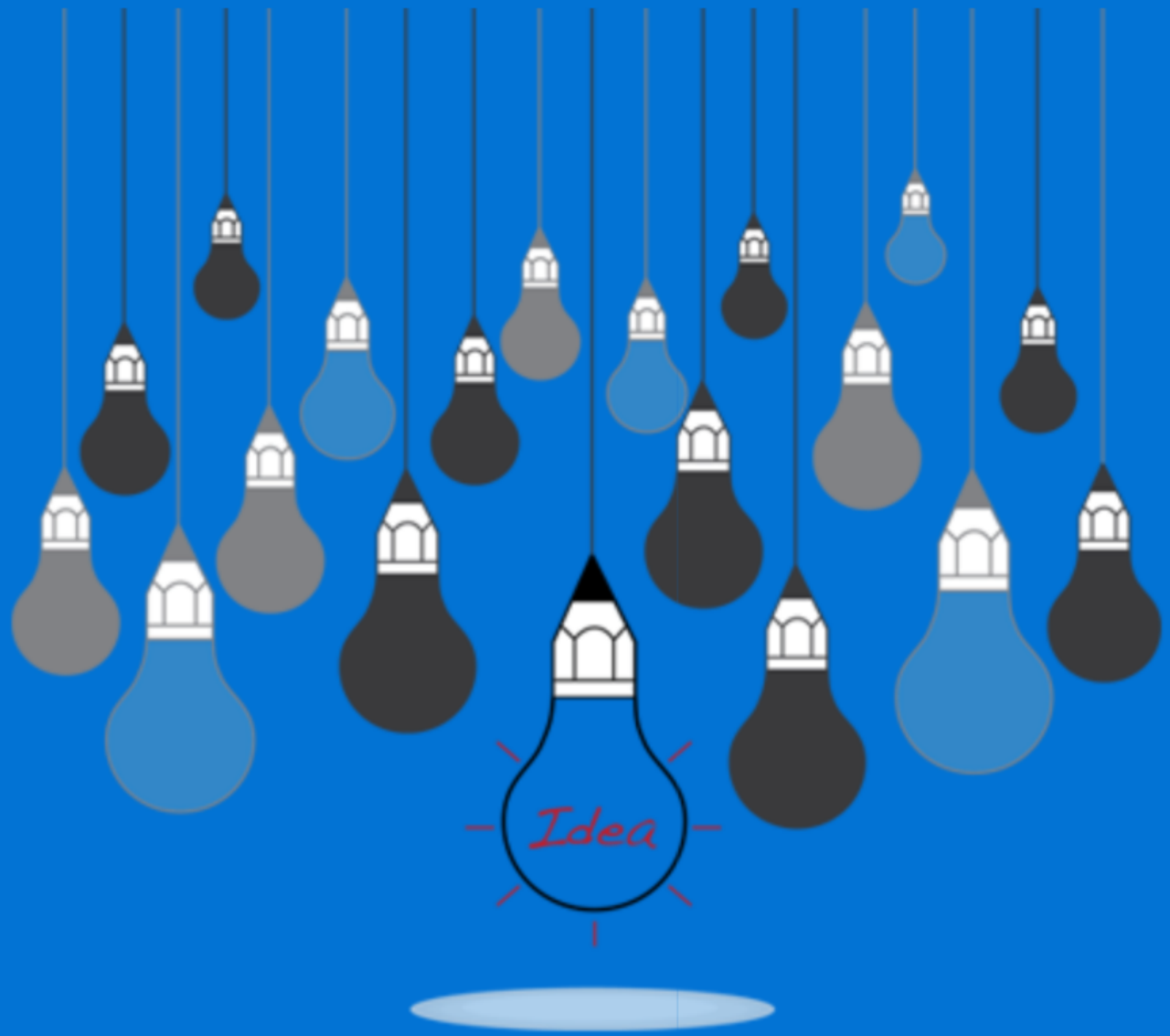
Validateurs asynchrones

```
let validators = Validators.compose(Validators.required, containsTroll)
let asyncValidators = Validators.composeAsync(exists, exists1)

this.name = new Control('', validators, validatorsAsync);
```

Validateurs combinés

Demo time !



Défi

- Créer un nouveau composant **widget-detail** qui prend en input un widget
- Dans ce composant, afficher un formulaire pour éditer ses valeurs
- Créer un formulaire pour créer un **widget**