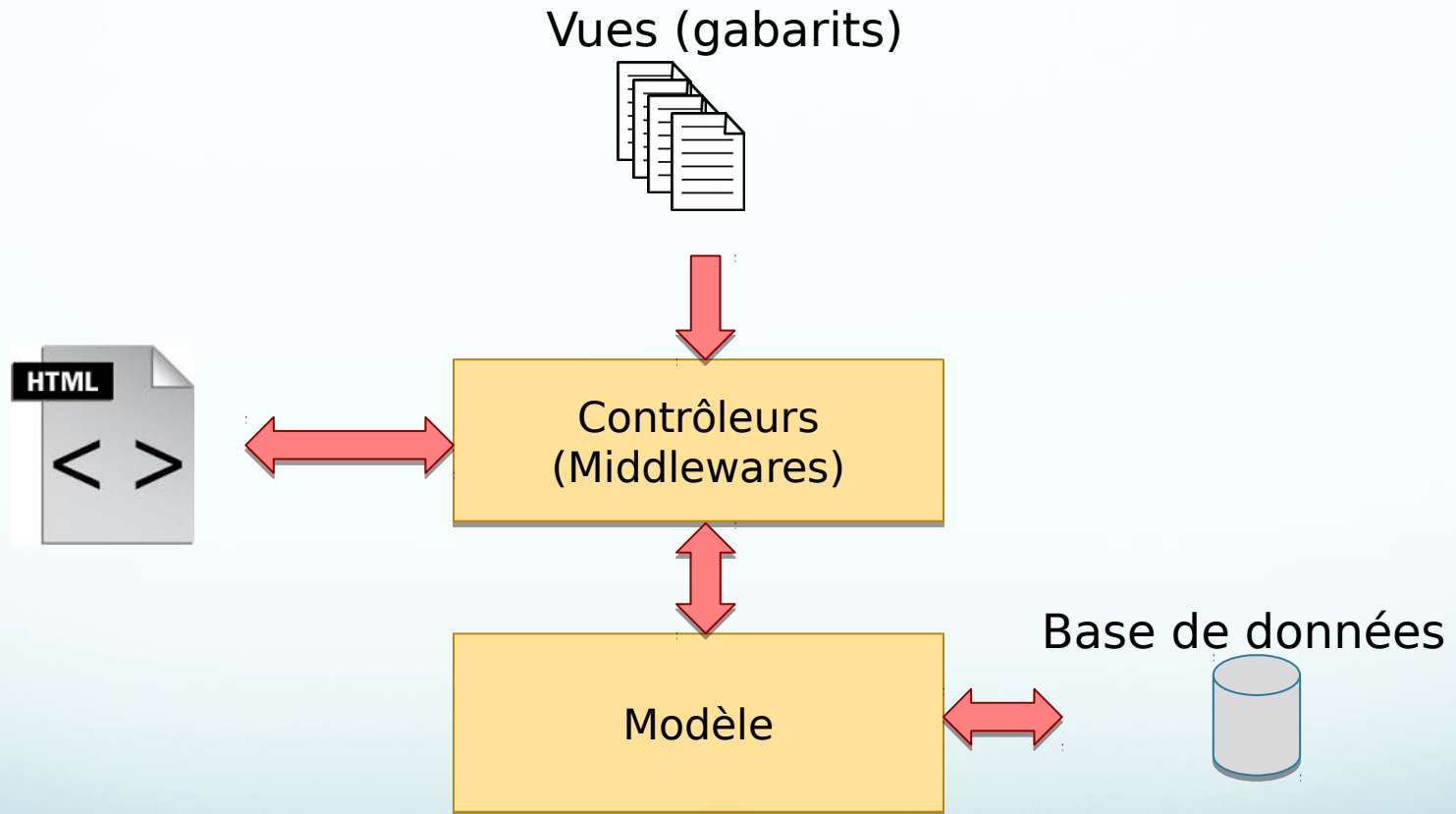


# MVC sur le client avec Angular

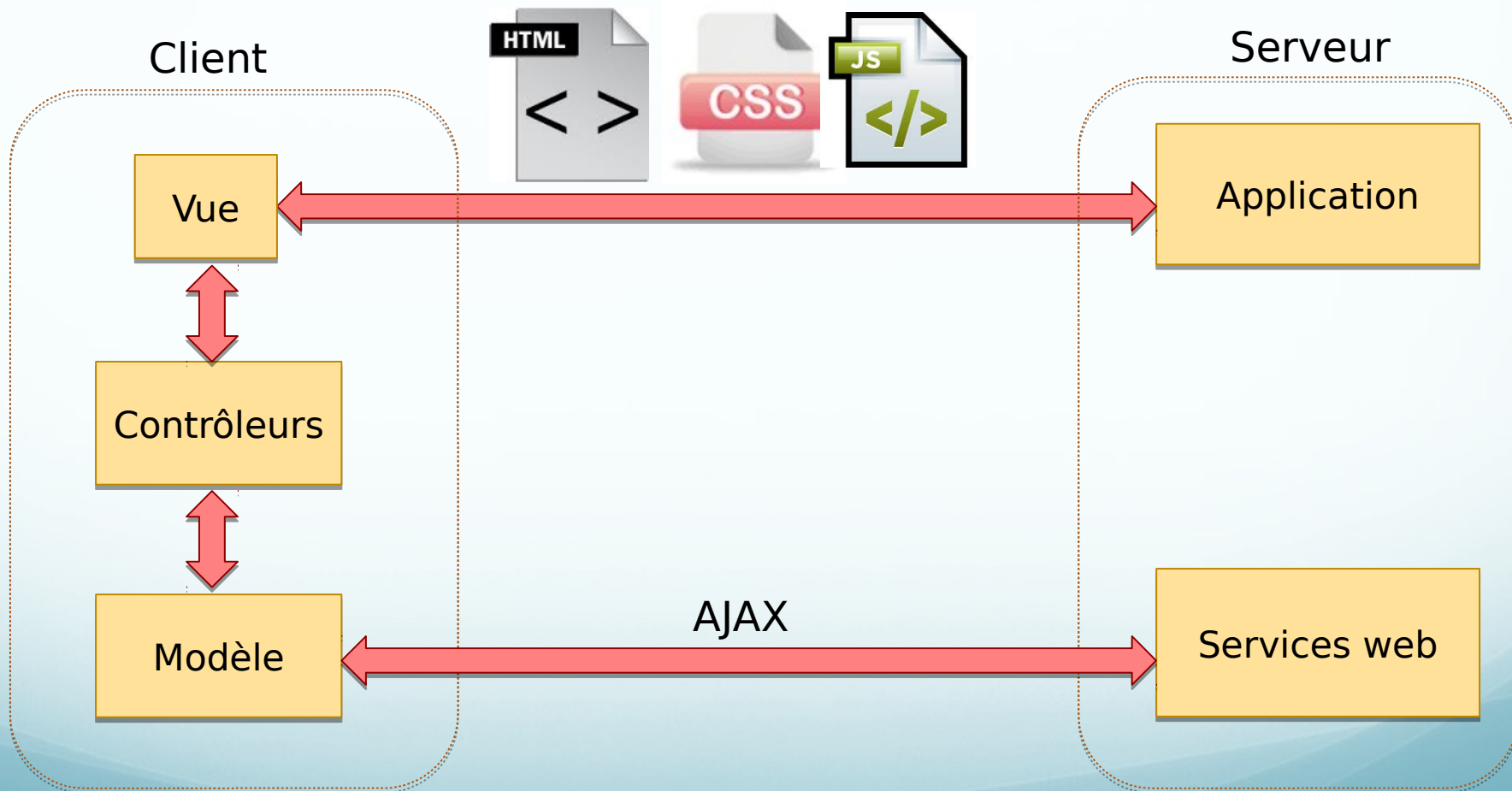
Michel Gagnon  
Konstantinos Lambrou-Latreille  
École polytechnique de Montréal



# MVC sur le serveur



# MVC sur le client



# Angular

- Enrichit le HTML, par le biais de directives implémentées sous forme d'attributs dans les balises HTML
- Nouvelle version majeure de AngularJS (pas compatible)
- Les directives minimales pour définir une application:
  - **@NgModule**: Définition et amorçage de **@NgModule**: définit l'ensemble des dépendances de votre application Angular
  - **@Component**: Définition d'une ou plusieurs composantes principales (classes qui gèrent une partie de la vue)

# Interpolation

- Toute variable qui est dans la portée d'une composante peut être utilisée dans la vue (interpolation)
- L'interpolation se fait avec la syntaxe **{{expression}}**
- L'expression peut être une variable ou toute expression Javascript, avec les contraintes suivantes:
  - Le contexte est la classe qui représente la **composante**, pas l'objet **window**
  - Pas de structure de contrôle (conditionnels, boucles, exceptions)
  - On peut utiliser des filtres avec les **pipes** -> |

# Exemple simple

## main.js

```
import { platformBrowserDynamic } from "@angular/platform-browser-dynamic";
import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";
import { Component } from "@angular/core";

@Component({
  selector: "mon-app",
  template: `
    <h1>{{cours}}</h1>
    <p>{{salutation.texte}} à tous</p>
  `
})
class AppComponent {
  cours = "LOG4420";
  salutation = { texte: "Bonjour" };
}

@NgModule({ imports: [BrowserModule], declarations: [AppComponent], bootstrap:
[AppComponent] })
class AppModule { }

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

# Exemple simple

## index.html

```
<html>
<head>
  <meta charset="utf-8">
  // Définir les dépendances ici
</head>
<body>
  <mon-app>Loading</mon-app>
</body>
</html>
```



## main.js

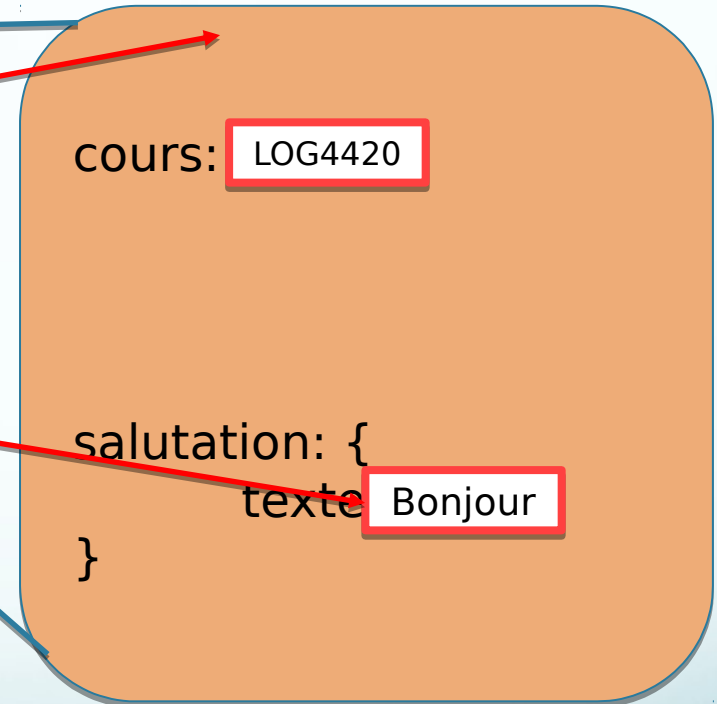
```
@Component({
  selector: "mon-app",
  template: `
    <h1>{{cours}}</h1>
    <p>{{salutation.texte}} à tous</p>
  `
})
```

# Exemple simple

main.js

```
@Component({
  selector: "mon-app",
  template: `
    <h1>{{cours}}</h1>
    <p>{{salutation.texte}} à tous</p>
  `
})
class AppComponent {
  cours = "LOG4420";
  salutation = { texte: "Bonjour" };
```

Variables visibles  
de la composante



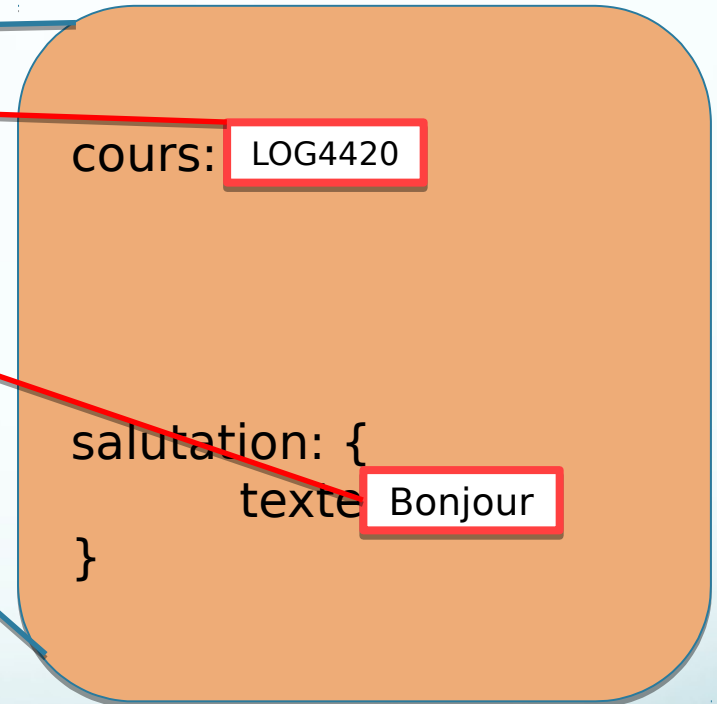


# Exemple simple

**main.js**

```
@Component({
  selector: "mon-app",
  template: `
    <h1>LOG4420</h1>
    <p>Bonjour à tous</p>
  `
})
class AppComponent {
  cours = "LOG4420";
  salutation = { texte: "Bonjour" };
}
```

Variables visibles  
de la composante



# Exemple simple

**main.js**

```
Component({
  selector: "mon-app",
  template: `
    <h1>{{cours}}</h1>
    <p>{{ "*** " + message(cours) + " ***" }} </p>
  `
})

class AppComponent {
  cours = "LOG4420";
  message = c => "Bienvenue au cours " + c;
```

Un exemple  
d'interpolation avec une  
expression Javascript  
complexe

# Exemple simple

**main.js**

```
@Component({
  selector: "mon-app",
  template: `
    <h1>LOG4420</h1>
    <p>*** Bienvenue au cours LOG4420 *** </p>
  `
})
class AppComponent {
  cours = "LOG4420";
  message = c => "Bienvenue au cours " + c;
}
```

Un exemple  
d'interpolation avec une  
expression Javascript  
complexe

# Liaison bidirectionnelle

- Si la valeur change dans la vue, elle est immédiatement reflétée dans le modèle
- Si la valeur change dans le modèle, elle est immédiatement reflétée dans la vue
- On utilise la directive **ngModel**: associe un item input du HTML avec une variable du modèle
- On peut utiliser la directive **ngChangeModel** pour associer une fonction qui sera exécutée chaque fois que la valeur change (on verra un exemple plus loin)
- Il faut importer le module **FormsModule**

# Liaison bidirectionnelle

## Exemple

### main.js

```
Autres imports ...
import { FormsModule } from "@angular/forms"

Component({
  selector: "mon-app",
  template: `
    <h1>{{cours}}</h1>
    Qui êtes-vous?
    <input [(ngModel)]="salutation.nom">
    <p>{{salutation.texte}} {{salutation.nom}}!</p>
  `,
})

class AppComponent {
  cours = "LOG4420";
  salutation = { texte: "Bonjour" };
```



# Liaison bidirectionnelle

## Exemple

### index.html

```
// Autres imports ...
import { FormsModule } from "@angular/forms"

@Component({
  selector: "mon-app",
  template: `
    <h1>{{cours}}</h1>
    Qui êtes-vous?
    <input [(ngModel)]="salutation.nom">
    <p>{{salutation.texte}}
  {{salutation.nom}}!</p>
    <button
  (click)="init()">Initialiser</button>
  `
})
class AppComponent {
  cours = "LOG4420";
  salutation = { texte: "Bonjour" };
  init = function() { this.salutation.nom = ""; }
}
```



# Filtres

- Avant de transmettre une valeur à la vue, on peut la faire passer par des filtres
- Exemple: `{{ expression | filtre1 | filtre2 | ... }}`
- Un filtre peut prendre des arguments: `{{ expression | filtre:arg1:arg2 }}`
- Filtres pré-définis:
  - **currency**
  - **date**
  - **json**
  - **lowercase** et **uppercase**
  - **limitTo**
  - **async**
  - **decimal**
  - **percent**
- On peut définir nos propres filtres

# Filtres - Exemple

## index.html

```
// Autres imports ...
import { FormsModule } from "@angular/forms"

@Component({
  selector: "mon-app",
  template: `
    <h1>{{cours}}</h1>
    Qui êtes-vous?
    <input [(ngModel)]="salutation.nom">
    <p>{{salutation.texte}} {{salutation.nom | uppercase}}!</p>
  `
})
class AppComponent {
  cours = "LOG4420";
  salutation = { texte: "Bonjour" };
}
```





# Filtres - Exemple

## index.html

```
// Autres imports ...
import { FormsModule } from "@angular/forms"

@Component({
  selector: "mon-app",
  template: `
    <h1>{{cours}}</h1>
    <p>{{note | percent: ".2" }}</p>
    <p>{{score | number: "2.1-2" }}</p>
  `
})
class AppComponent {
  cours = "LOG4420";
  note: number;
  score: number;

  constructor() {
    this.note = 0.99;
    this.score = 9.12345
  }
}
```

# Filtres - Exemple

## index.html

```
// Autres imports ...
import { FormsModule } from "@angular/forms"

@Component({
  selector: "mon-app",
  template: `
    <h1>{{titre}}</h1>
    <p> {{ promesse | async }} </p>
  `
})
class AppComponent {
  titre = "Async";
  promesse = Promise<string>;

  constructor() {
    this.promesse = new Promise(function(resolve,
reject) {
      setTimeout(function() {
        resolve("Hé! Je viens d'une promesse");
      }, 5000);
    }
  )
}
```

# Filtre personnalisé

- Étapes
  1. Importer les modules de déclarations de filtres
  2. Enregistrer le nom du filtre avec **@Pipe**
  3. Hériter de la classe **PipeTransform**
    - Implémentation de la méthode **transform**
    - Premier argument: la valeur passée dans le filtre
    - Deuxième argument: tableau d'arguments du filtre
  4. Inclure notre nouveau filtre dans le tableau de déclarations de notre @NgModule
- Les arguments passés en paramètre dans notre filtre seront transmis par la syntaxe monfiltre : arg1 : arg2 : ... : argn

# Filtre personnalisé - Exemple

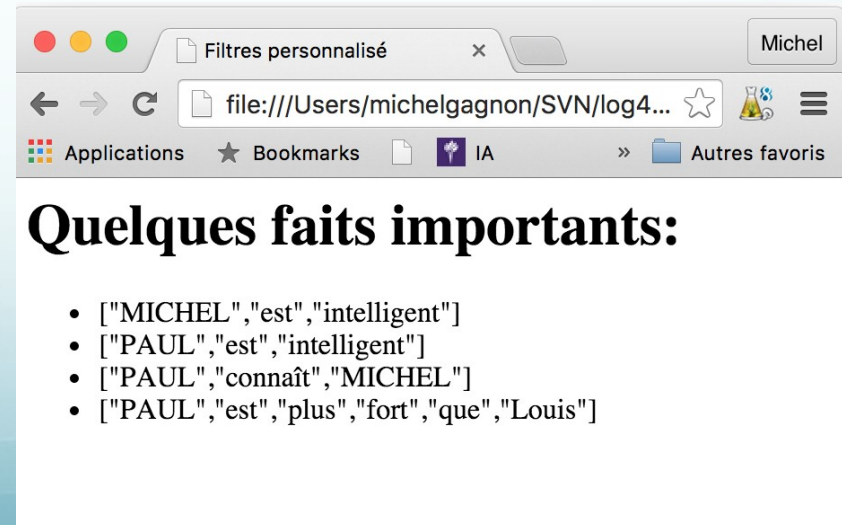
```
import { Pipe, PipeTransform } from "@angular/core"

@Pipe({ name: "filtrerNoms" })
class FiltrerNomsPipe implements PipeTransform {
  transform(value: string, noms: string[]): string[] {
    let listeMots = value.split(' ');
    let listeMotsFiltres = listeMots.map(item => {
      if (noms.some(n => n === item))
        return item.toUpperCase()
      else
        return item;
    });
    return listeMotsFiltres;
  }
}

@Component({
  selector: "mon-app",
  template: `
    <h1>Quelques faits importants:</h1>
    <ul>
      <li *ngFor="let item of liste">
        {{ item | filtrerNoms: ['Michel', 'Paul'] }}
      </li>
    </ul>
  `
})
```

```
class AppComponent {
  liste = [ "Michel est intelligent",
            "Paul est intelligent",
            "Paul connaît Michel",
            "Paul est plus fort que Louis" ]
}

@NgModule({
  imports: [...],
  declarations: [AppComponent,
                 FiltrerNomsPipe],
  bootstrap: [ ... ] })
class AppModule { }
```



# Filtres – Exemple complexe

localhost:3000

Rechercher

Par défaut Les tâches de Michel

Texte filtre

Montrer seulement les tâches complétées ☐

### Quelques faits importants:

- Michel - passer aspirateur - true
- Michel - laver la vaisselle - false
- Eliana - regarder un film - true
- Tomas - faire son lit - false
- Michel - ramasser les feuilles - true
- Michel - faire le souper - true
- Tomas - étudier - false
- Felipe - faire son lit - false
- Felipe - étudier - false

# Filtres – Exemple complexe

```
@Pipe({ name: "filtrerNotes" })
```

```
class FiltrerNotesPipe implements PipeTransform {  
  transform(value, option: string, done: boolean): string[] {  
    let filteredVals = value.filter(option);  
    return done ? filteredVals.filter(v => v.done === done) : filteredVals;  
  }  
}
```

```
@NgModule({  
  imports: [ BrowserModule, FormsModule ],  
  declarations: [ AppComponent, FiltrerNotesPipe ],  
  bootstrap: [ AppComponent ]  
})  
class AppModule { }
```

```
const platform = platformBrowserDynamic();  
platform.bootstrapModule(AppModule);
```

# Filtres – Exemple complexe

```
@Component({
  selector: 'my-app',
  template: `
    <button (click)="filtreActuel=filtreOptions['chaîne']
(chaîne)">
      Par défaut
    </button>
    <button (click)="filtreActuel=filtreOptions['objet']">
      Les tâches de Michel
    </button>

    <label>Texte filtre</label>
    <input [ngModel]="chaîne"
(ngModelChange)="fc($event)" />

    <label for="done">Montrer seulement les tâches
complétées</label>
    <input id="done"
      type="checkbox"
      [(ngModel)]="done" />

    <h1>Quelques faits importants:</h1>
    <ul>
      <li *ngFor="let note of notes |
filtrerNotes:filtreActuel:done">
        {{ note.personne + " - " + note.tache + " - " +
note.done }}
      </li>
    </ul>
  `
})
```

```
class AppComponent {
  notes = [
    {personne: 'Michel', tache: 'passer aspirateur', done: true},
    {personne: 'Michel', tache: 'laver la vaisselle', done: false},
    {personne: 'Eliaana', tache: 'regarder un film', done: true},
    {personne: 'Tomas', tache: 'faire son lit', done: false},
    {personne: 'Michel', tache: 'ramasser les feuilles', done:
true},
    {personne: 'Michel', tache: 'faire le souper', done: true},
    {personne: 'Tomas', tache: 'étudier', done: false},
    {personne: 'Felipe', tache: 'faire son lit', done: false},
    {personne: 'Felipe', tache: 'étudier', done: false}
  ];

  filtreOptions = {
    chaîne: c => note => note.tache.indexOf(this.chaîne) != -1,
    objet: note => note.personne === "Michel"
  };

  chaîne = "";
  done = undefined;
  filtreActuel = this.filtreOptions["chaîne"];

  fc = function(nouvelleVal) {
    this.chaîne = nouvelleVal;
    this.filtreActuel = this.filtreOptions['chaîne'](this.chaîne)
  }
}
```

# Itérateurs

- Comme dans les gabarits utilisés sur le serveur, Angular2 permet d'exécuter des itérateurs
- On utilise la directive **ngFor**
- Pour chaque item de la liste indiquée, on génère une copie de l'élément qui contient la directive **ngFor**
- Une variable **index**, incrémentée à chaque itération, peut être utilisée
- On a aussi les variables booléennes **first**, **middle**, **last**, **odd** et **even**

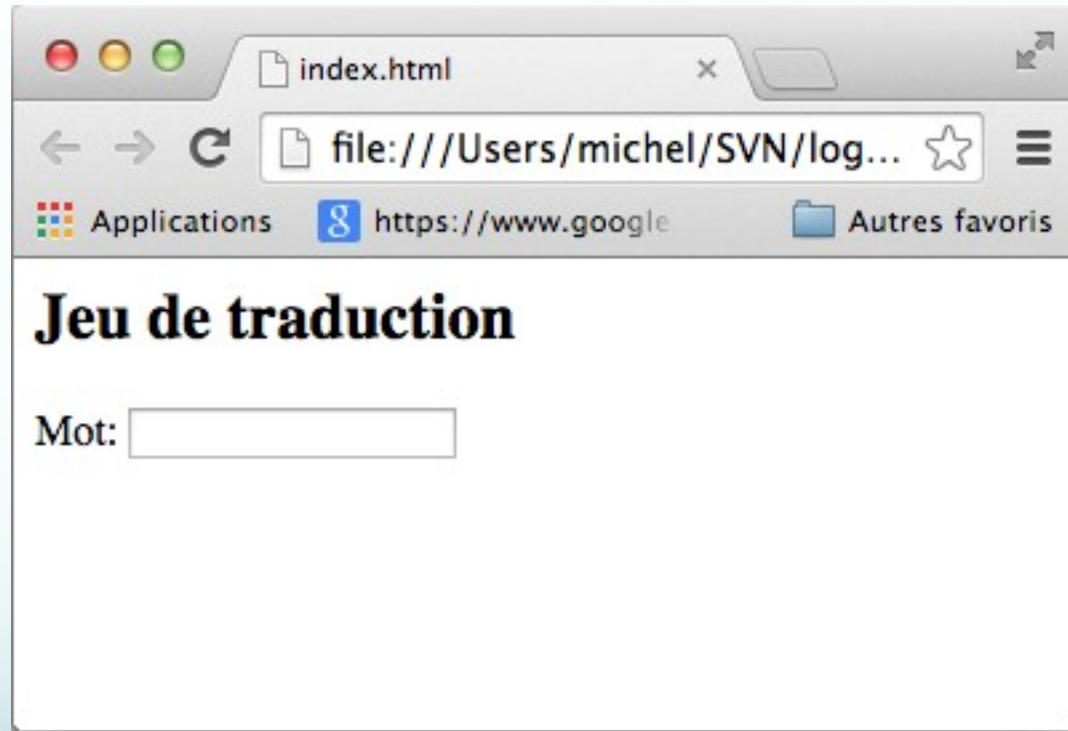


# Conditionnels

- Comme dans les gabarits utilisés sur le serveur, Angular2 permet l'utilisation de conditionnels
- On utilise la directive **ngIf**
- On passe en attribut une expression Javascript
- Si la valeur de cette expression est **true**, l'élément qui contient la directive **ngIf** sera affiché

# Itérateur et conditionnel

## Exemple



# Itérateur et conditionnel

## Exemple



A screenshot of a web browser window. The title bar shows 'index.html'. The address bar contains 'file:///Users/michel/SVN/log4...'. The browser's toolbar includes back, forward, and refresh buttons, along with a search bar containing 'https://www.google' and a folder icon labeled 'Autres favoris'. The main content area displays the title 'Jeu de traduction' in bold. Below it, a label 'Mot:' is followed by a text input field containing the word 'chien'. Underneath the input field, the text 'Le mot CHIEN existe. Traduis-le' is shown. At the bottom, there are three radio button options: 'dog', 'cat', and 'rabbit'.

**Jeu de traduction**

Mot:

Le mot CHIEN existe. Traduis-le

☐ dog  
☐ cat  
☐ rabbit

# Itérateur et conditionnel

## Exemple



# Itérateur et conditionnel

## Exemple



# Module

- Les modules offrent beaucoup de flexibilité:
  - Peuvent contenir des composantes, des services, des fabriques, les routes, etc.
  - Établissent les dépendances
  - Une application Angular2 doit contenir au moins 1 module
- Pour définir un module:
  - Créer une classe qui va représenter le module
  - Déclarer le décorateur @NgModule sur votre classe
- Pour lancer votre module:
  - La plateforme qui lance votre application doit être initialisée
  - Utiliser cette plateforme pour amorcer votre module

# Module

- Code minimal pour lancer sur le navigateur

## **app.module.ts**

```
import { NgModule }    from '@angular/core';
import { BrowserModule } from '@angular/platform-
browser';

@NgModule({
  imports:    [ BrowserModule ]
})
export class AppModule { }
```

## **main.ts**

```
import { platformBrowserDynamic } from '@angular/platform-
browser-dynamic';

import { AppModule } from './app.module';

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

# Séparation de responsabilités

- Pourquoi séparer *main.ts*, *app.module.ts* et *app.component.ts*?
- Responsabilités différentes
  - *main.ts* - L'amorçage de l'application
  - *app.module.ts* - Création d'un module
  - *app.component.ts* - Présentation de la vue
- Pour amorcer sur un navigateur, on utilise la fonction *platformBrowserDynamic*.
- Dans une application mobile, on peut créer un module avec *Apache Cordova* ou *NativeScript* et l'amorcer avec une fonction spécifique à cette plateforme.



# Module

- Modules vont être plus complexes
- Le champ *imports* contient les modules externes qui seront utilisés dans l'application
  - *BrowserModule, FormsModule, RouterModule, HttpClientModule, etc.*
- Le champ *declarations* contient les classes Angular2 que vous avez déclaré (composantes, filtres, etc.).
- Le champ *bootstrap* contient la ou les composantes principales qui seront utilisées pour l'amorçage.

```
app.module.ts  
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
  
@NgModule({  
  imports: [ BrowserModule ],  
  declarations: [ AppComponent, FilterNomsPipe ],  
  bootstrap: [ AppComponent ]  
})  
export class AppModule { }
```

# Gérer plusieurs composants

```
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { Component, Input } from '@angular/core'
```

```
@Component({
  selector: 'message',
  template: `
    <p>Bienvenue au cours {{cours}}</p>
  `
})
```

```
class MessageComponent {
  @Input()
  cours: string;
}
```

```
@Component({
  selector: 'mon-app',
  template: `
    <label>Choisir un sigle de cours:</label><input
    [(ngModel)]="cours" />
    <message [cours]= "cours"></message>
  `
})
```

```
class AppComponent {
  cours = "LOG4420";
}
```

```
@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ AppComponent,
MessageComponent ],
  bootstrap: [ AppComponent ]
})
class AppModule { }
```

```
const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

# Formulaires

- On peut associer chaque élément `<input>` à une portion du modèle par le biais de la directive **ngModel**
- Dans le CSS, on peut utiliser des classes prédéfinies par Angular2, par exemple:
  - **ng-valid** et **ng-invalid** (le modèle est valide/invalid)
  - **ng-pristine** et **ng-dirty** (il y a eu/n'y a pas eu d'interaction avec le contrôle)
  - **ng-touched** et **ng-untouched** (le contrôle a perdu/n'a pas perdu le focus)
  - <https://angular.io/docs/ts/latest/guide/forms.html#!#track-change-state-and-validity-with-ngmodel->
- On a aussi les variables booléennes
  - **valid** et **invalid** : les entrées et les contrôles sont valides/invalides)
  - **pristine**: aucune interaction avec le formulaire n'a encore eu lieu
  - **dirty** : il y a déjà eu une interaction avec le formulaire
  - **submitted** : le formulaire a été soumis
- On peut associer une fonction à l'événement de soumission, par le biais de la directive **ngSubmit**

# Formulaires (suite)

- **ngModelChange** (l'expression ou la fonction associée est exécutée lorsque le contenu change)
- On peut utiliser la propriété **disabled** pour indiquer qu'un contrôle est désactivé tant que l'expression associée est évaluée à **true**
- On a aussi des directives pour la validation de formulaires réactifs:
  - **minLength** (longueur minimale du texte)
  - **maxLength** (longueur maximale du texte)
  - **required** (ajoute l'attribut **required** si l'expression associée est évaluée à **true**)

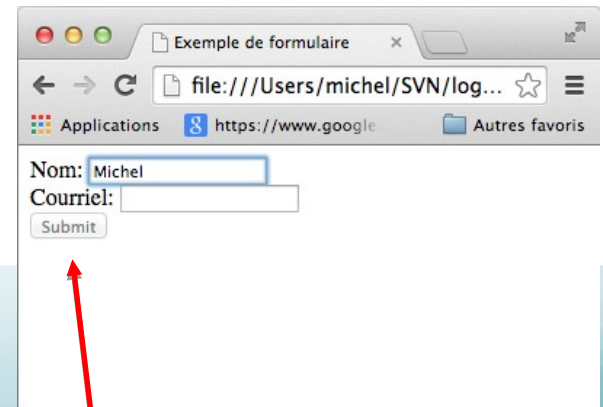
# Formulaires (suite)

- La directive **ngClass** permet d'ajouter une classe à un élément selon un prédicat
  - `<p [ngClass]="c1 c2">...</p>`
  - `<p [ngClass]="['c1', 'c2']">...</p>`
  - `<p [ngClass]="{'c1': true, 'c2': true, 'c3': false}">...</p>`

# Exemple de formulaire

```
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { Component, Input } from '@angular/core'
```

```
@Component({
  selector: 'mon-app',
  template: `
    <form (ngSubmit)="submit()" #monFormulaire="ngForm">
      Nom: <input [(ngModel)]="user.nom" name="nom" required minlength="4"><br>
      Courriel: <input type="email" [(ngModel)]="user.courriel" name="courriel" pattern="^[a-z]+@[a-z]+$"
required><br>
      <input type="submit" value="Submit" [disabled]="!monFormulaire.valid" />
    </form>
  `,
})
class AppComponent {
  user = {};
  submit = function() {
    console.log("Voici ce qui a été soumis: ", this.user);
  }
}
```

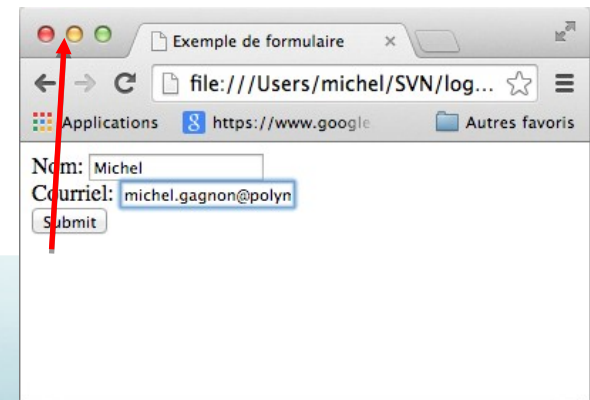


*Désactivé*

# Exemple de formulaire

```
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { Component, Input } from '@angular/core'
```

```
@Component({
  selector: 'mon-app',
  template: `
    <form (ngSubmit)="submit()" #monFormulaire="ngForm">
      Nom: <input [(ngModel)]="user.nom" name="nom" required minlength="4"><br>
      Courriel: <input type="email" [(ngModel)]="user.courriel" name="courriel" pattern="^[a-z]+@[a-z]+$"
required><br>
      <input type="submit" value="Submit" [disabled]="!monFormulaire.valid" />
    </form>
  `,
})
class AppComponent {
  user = {};
  submit = function() {
    console.log("Voici ce qui a été soumis: ", this.user);
  }
}
```



Activé

# Exemple de formulaire réactif

```
import { BrowserModule } from '@angular/platform-browser'
import { ReactiveFormsModule, FormGroup,  
       FormBuilder, Validators } from '@angular/forms'
import { Component, Input } from '@angular/core'
```

```
@Component({
  selector: 'mon-app',
  template: `
    <form (ngSubmit)="submit()"
[formGroup]="monFormulaire">
      Nom: <input [(ngModel)]="user.nom"
formControlName="nom"><br>
      Courriel: <input type="email"
                  [(ngModel)]="user.courriel"
                  formControlName="courriel"><br>
      <input type="submit" value="Submit" [disabled]="!
monFormulaire.valid" />
    </form>
  `
})
```

```
class AppComponent {
  user = { nom: "", courriel: "" };
  monFormulaire: FormGroup;
```

```
  constructor(private builder: FormBuilder) {  
    this.monFormulaire = this.builder.group({  
      "nom": [ "", [Validators.required,  
Validators.minLength(4)],  
      "courriel": [ "", [Validators.required,  
RegexValidator.regex("...")]  
    });  
  }  
}
```

```
class RegexValidator {  
  static regex(pattern: string): Function {  
    return (control: AbstractControl): {[key: string]:  
any} => {  
      return control.value.match(pattern) ? null :  
{ pattern: true };  
    }  
  }  
}
```

```
@NgModule({
  imports: [ BrowserModule, ReactiveFormsModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
class AppModule { }
```

```
const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```



# Services

- Classes qui peuvent être partagés par plusieurs composantes ou applications
- Responsabilités typiques d'un service: communication avec le serveur, validation, stockage local, calculs et processus qui peuvent être nécessaires à différentes applications ou contrôleurs
- Services pré-définis de Angular:
  - **WindowRef** (pour interagir avec l'objet **window**)
  - **Location** (pour interagir avec l'URL qui se trouve dans la fenêtre de navigation)
  - **Http** (pour effectuer les requêtes AJAX)
  - **Log** (service de journalisation)

# Services (suite)

- L'implémentation des services est basée sur l'injection de dépendances
- Un service est indépendant de la vue
- Pour créer un service, il suffit de créer une classe qui sera injectable avec @Injectable.

```
@Injectable  
class MonService {  
    getAll = ... // Fonction qui récupère des résultats  
}
```

# Exemple d'utilisation de service

```
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { NgModule, Component, OnInit, Injectable } from '@angular/core'
import { Pipe, PipeTransform } from '@angular/core'
```

```
@Pipe({ name: "orderBy" })
class OrderByPipe implements PipeTransform {
  transform(values, option) {
    values.sort((obj1,obj2) => {
      let a = obj1[option];
      let b = obj2[option];
      if (a < b) return -1;
      else if (a > b) return 1;
      else return 0;
    });
    return values;
  }
}
```

```
@Injectable()
class UserService {
  users = [
    { num: 4, nom: 'Michel' },
    { num: 7, nom: 'Ana' }
  ]
  getAll = () => this.users;
  add = (user) => this.users.push(user);
}
```

# Exemple d'utilisation de service

```
@Component({
  selector: 'mon-app',
  template: `
    <h1>Exemple utilisant un service</h1>
    <button (click)="open(0)">En ordre
    alphabétique</button>
    <button (click)="open(1)">En ordre
    numérique</button>
    <div [ngSwitch]="tab">
      <div *ngSwitchCase="0">
        <h3>En ordre alphabétique</h3>
        <table>
          <tr *ngFor="let user of users | orderBy:'nom'">
            <td>{{ user.nom }}</td>
            <td>{{ user.num }}</td>
          </tr>
        </table>
      </div>
      <div *ngSwitchCase="1">
        <h3>En ordre alphabétique</h3>
        <table>
          <tr *ngFor="let user of users | orderBy:'num'">
            <td>{{ user.num }}</td>
            <td>{{ user.nom }}</td>
          </tr>
        </table>
      </div>
    </div>
    Nom: <input [(ngModel)]="nom" /><br>
    Numéro: <input [(ngModel)]="num" /><br>
    <button (click)="add()">Ajouter</button>
  `,
  providers: [UserService]
})
```

```
class AppComponent implements OnInit {
  tab = 1;
  users = [];

  nom = "";
  num = "";

  open = tab => this.tab = tab;
  add = () => this.userService.add({ num: parseInt(this.num),
    nom: this.nom });

  constructor(private userService: UserService) { }

  ngOnInit(): void {
    this.users = this.userService.getAll();
  }
}

@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, OrderByPipe ],
  bootstrap: [ AppComponent ]
})
class AppModule { }

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

# Exemple d'utilisation de service



The screenshot shows a web browser window with a single tab titled 'index.html'. The address bar displays the file path 'file:///Users/michel/SVN/log4420/...'. Below the address bar, there are links to 'Applications', 'https://www.google', and 'Autres favoris'. The main content area has a title 'Exemple utilisant un service'. Below the title, there are two buttons: 'En ordre numérique' and 'En ordre alphabétique'. Underneath these buttons, the text 'En ordre numérique d'identificateur' is displayed. This is followed by a list of entries: '4 Michel' and '7 Ana'. Below the list, there are two input fields labeled 'Nom:' and 'Numéro:', and an 'Ajouter' button.

index.html

file:///Users/michel/SVN/log4420/...

Applications <https://www.google> Autres favoris

## Exemple utilisant un service

**En ordre numérique d'identificateur**

4 Michel  
7 Ana

Nom:   
Numéro:

# Exemple d'utilisation de service



# Exemple d'utilisation de service



A screenshot of a web browser window displaying a local file named 'index.html'. The address bar shows the file path: 'file:///Users/michel/SVN/log4420/...'. Below the address bar, there are links for 'Applications', 'https://www.google', and 'Autres favoris'. The main content area features a heading 'Exemple utilisant un service' and two buttons: 'En ordre numérique' and 'En ordre alphabétique'. The 'En ordre alphabétique' button is selected. Below these buttons, a list of names and numbers is displayed: 'Ana 7', 'Michel 4', and 'Zoé 5'. At the bottom, there is a form with two input fields: 'Nom: Zoé' and 'Numéro: 5', followed by an 'Ajouter' button.

index.html

file:///Users/michel/SVN/log4420/...

Applications https://www.google Autres favoris

## Exemple utilisant un service

En ordre numérique En ordre alphabétique

### En ordre alphabétique

Ana 7  
Michel 4  
Zoé 5

Nom: Zoé  
Numéro: 5

Ajouter

# Exemple d'utilisation de service



A screenshot of a web browser window. The title bar shows 'index.html'. The address bar contains 'file:///Users/michel/SVN/log4420/...' with a star icon and a menu icon. Below the address bar, there are links for 'Applications', 'https://www.google', and 'Autres favoris'. The main content area has the heading 'Exemple utilisant un service'. Below this, there are two buttons: 'En ordre numérique' (selected) and 'En ordre alphabetique'. Under the selected button, the text 'En ordre numérique d'identificateur' is displayed. Below this, there is a list of names with numbers: '4 Michel', '5 Zoé', and '7 Ana'. At the bottom, there are two input fields labeled 'Nom:' and 'Numéro:', followed by an 'Ajouter' button.

index.html

file:///Users/michel/SVN/log4420/...

Applications https://www.google Autres favoris

## Exemple utilisant un service

**En ordre numérique d'identificateur**

4 Michel  
5 Zoé  
7 Ana

Nom:   
Numéro:



# Communication avec le serveur

- On utilise le service **Http**, qui offre une méthode pour chaque méthode HTTP
  - `http.get`, `http.post`, `http.put`, `http.delete`, etc.
- Retourne la classe *Observable*, mais peut être convertit en « promesse » avec *.toPromise()*, c'est-à-dire un objet qui contient la méthode **then(successCallback)** et **catch errorCallback)**
  - Accès à *toPromise()* : `import 'rxjs/add/operator/toPromise';`

# Exemple d'utilisation de service

## Serveur NodeJS

```
var users = [  
  {num: 4, nom: 'Michel'},  
  {num: 7, nom: 'Ana'}  
]
```

```
router.get('/api/users', function(req, res) {  
  res.json(users);  
});
```

**// Initialiser le body-parser**

```
router.post('/api/users', function(req, res) {  
  var user = req.body;  
  users.push(user);  
  res.send(user);  
})
```

```
import { BrowserModule } from '@angular/platform-browser'  
import { FormsModule } from '@angular/forms'  
import { NgModule, Component, OnInit, Injectable } from  
'@angular/core'  
import { Pipe, PipeTransform } from '@angular/core'  
import { HttpModule, Headers, Http } from '@angular/core'
```

```
@Pipe({ name: "orderBy" })  
class OrderByPipe implements PipeTransform {  
  transform(values, option) {  
    if (values == null) return null;  
    values.sort((obj1,obj2) => {  
      let a = obj1[option];  
      let b = obj2[option];  
      if (a < b) return -1;  
      else if (a > b) return 1;  
      else return 0;  
    });  
    return values;  
  }  
}
```

```
@Injectable()  
class UserService {  
  url = "/api/users";  
  getAll: Promise<Object[]> {  
    return this.http.get(this.url).toPromise().then(res =>  
    res.json())  
  }  
  add (user): {  
    return this.http  
    .post(this.url, JSON.stringify(user), {headers:  
    this.headers})  
    .toPromise().then(res => res.json())  
  }  
}
```

# Exemple d'utilisation de service

```
@Component({
  selector: 'mon-app',
  template: `
    <h1>Exemple utilisant un service</h1>
    <button (click)="open(0)">En ordre
    alphabétique</button>
    <button (click)="open(1)">En ordre
    numérique</button>
    <div [ngSwitch]="tab">
      <div *ngSwitchCase="0">
        <h3>En ordre alphabétique</h3>
        <table>
          <tr *ngFor="let user of users | async |
orderBy:'nom'">
            <td>{{ user.nom }}</td>
            <td>{{ user.num }}</td>
          </tr>
        </table>
      </div>
      <div *ngSwitchCase="1">
        <h3>En ordre alphabétique</h3>
        <table>
          <tr *ngFor="let user of users | async |
orderBy:'num'">
            <td>{{ user.num }}</td>
            <td>{{ user.nom }}</td>
          </tr>
        </table>
      </div>
    </div>
    Nom: <input [(ngModel)]="nom" /><br>
    Numéro: <input [(ngModel)]="num" /><br>
    <button (click)="add()">Ajouter</button>
  `
})
```

```
class AppComponent implements OnInit {
  tab = 1;
  users: Promise<Object[]>;

  nom = "";
  num = "";

  open = tab => this.tab = tab;
  add = () => {
    this.userService.add({num: parseInt(this.num), nom: this.nom});
    this.users = this.userService.getAll();
  }

  constructor(private userService: UserService) { }

  ngOnInit(): void {
    this.users = this.userService.getAll();
  }
}

@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, OrderByPipe ],
  bootstrap: [ AppComponent ]
})
class AppModule { }

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```