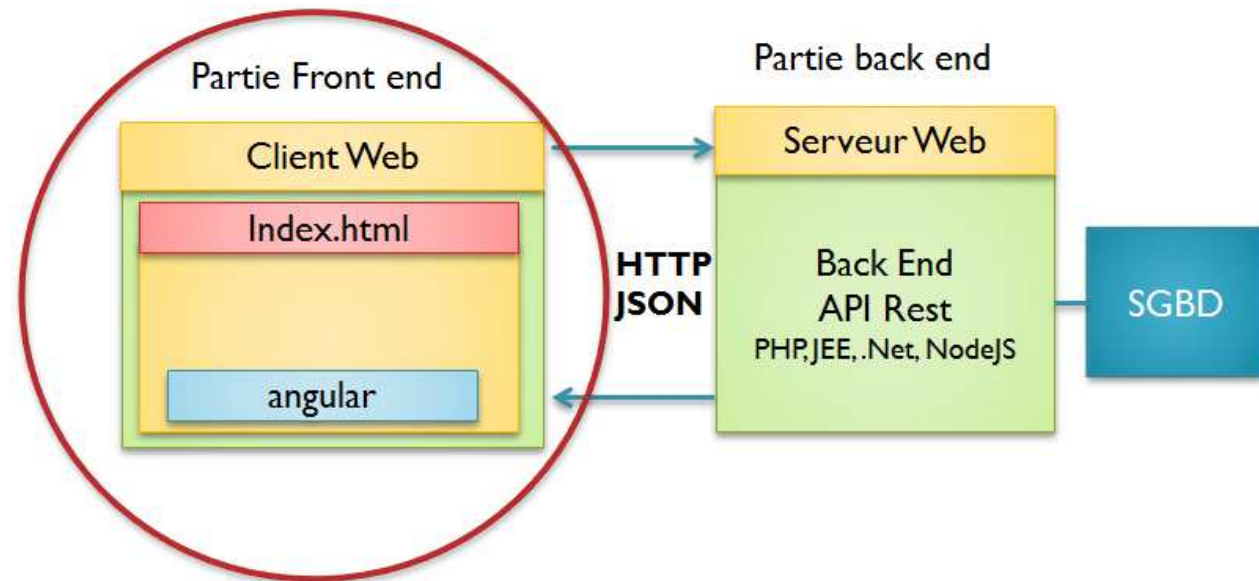


Développement Web FrontEnd

Framework Angular 2, 4, 5, 6



Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

Email : med@yousfsi.net

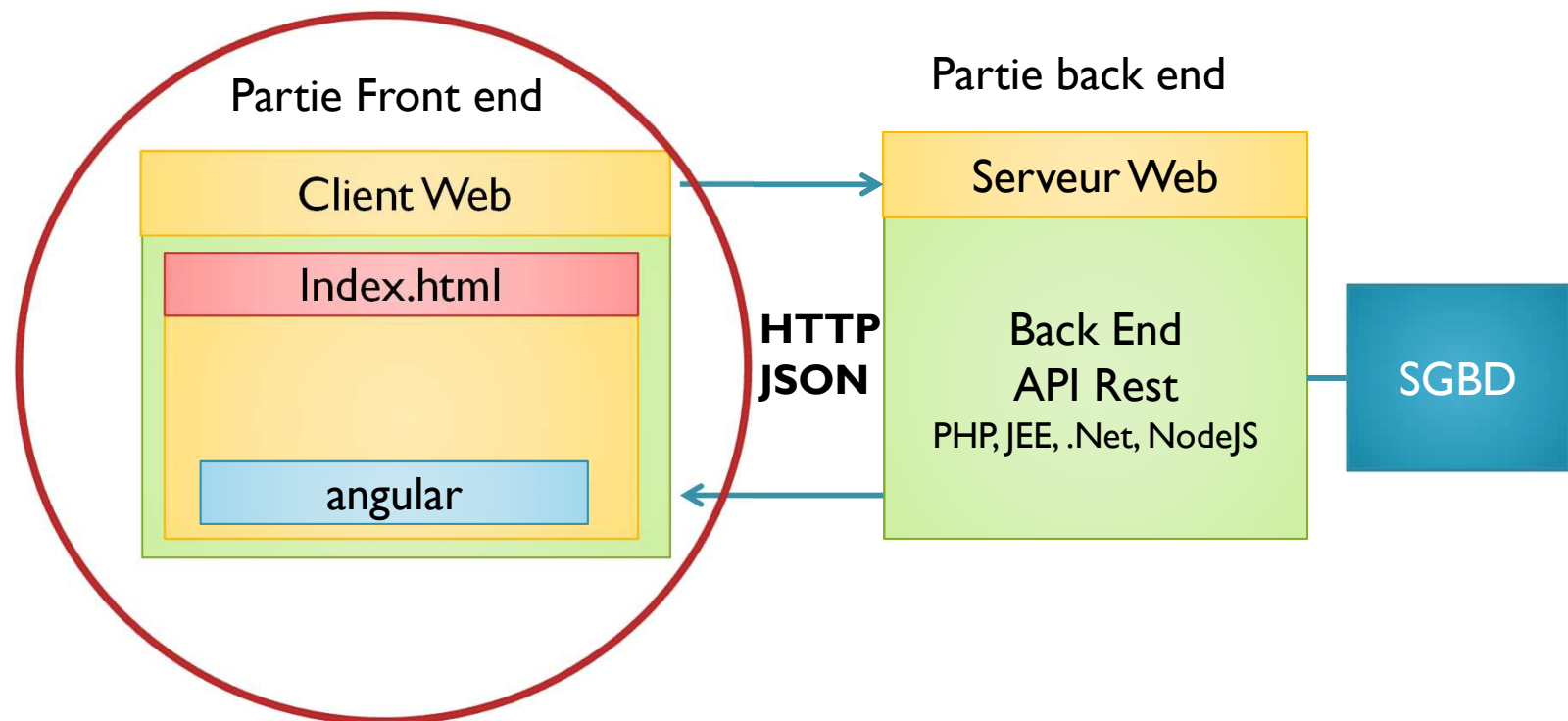
Supports de cours : <http://fr.slideshare.net/mohamedyousfsi9>

Chaîne vidéo : <http://youtube.com/mohamedYousfsi>

Recherche : http://www.researchgate.net/profile/Yousfsi_Mohamed/publications

Angular

- Angular Permet de créer la partie front end des applications web de type SPA (Single Page Application réactive)
- Une SPA est une application qui contient une seule page HTML (index.html) récupérée du serveur.
- Pour naviguer entre les différentes partie de cette application, Java Script est utilisé pour envoyer des requêtes http (AJAX) vers le serveur pour récupérer du contenu dynamique généralement au format JSON.
- Ce contenu JSON est ensuite affiché côté client au format HTML dans la même page.

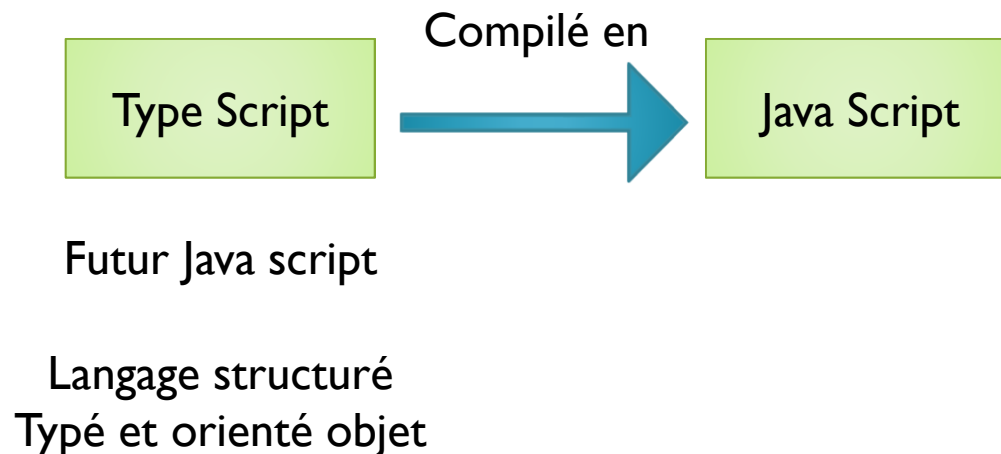


Angular 1, 2, 4, 5

- Angular 1 (**Angular JS**) :
 - Première version de Angular qui est la plus populaire.
 - Elle est basé sur une architecture MVC coté client. Les applications Angular 1 sont écrite en Java Script.
- Angular 2 (**Angular**) :
 - Est une réécriture de Angular 1 qui est plus performante, mieux structurée et représente le futur de Angular.
 - Les applications de Angular2 sont écrites en Type Script qui est compilé et traduit en Java Script avant d'être exécuté par les Browsers Web.
 - Angular 2 est basée sur une programmation basée sur les Composants Web (Web Componenet)
- Angular 4, 5, 6 sont de simples mises à jour de Angular 2 avec des améliorations au niveau performances.

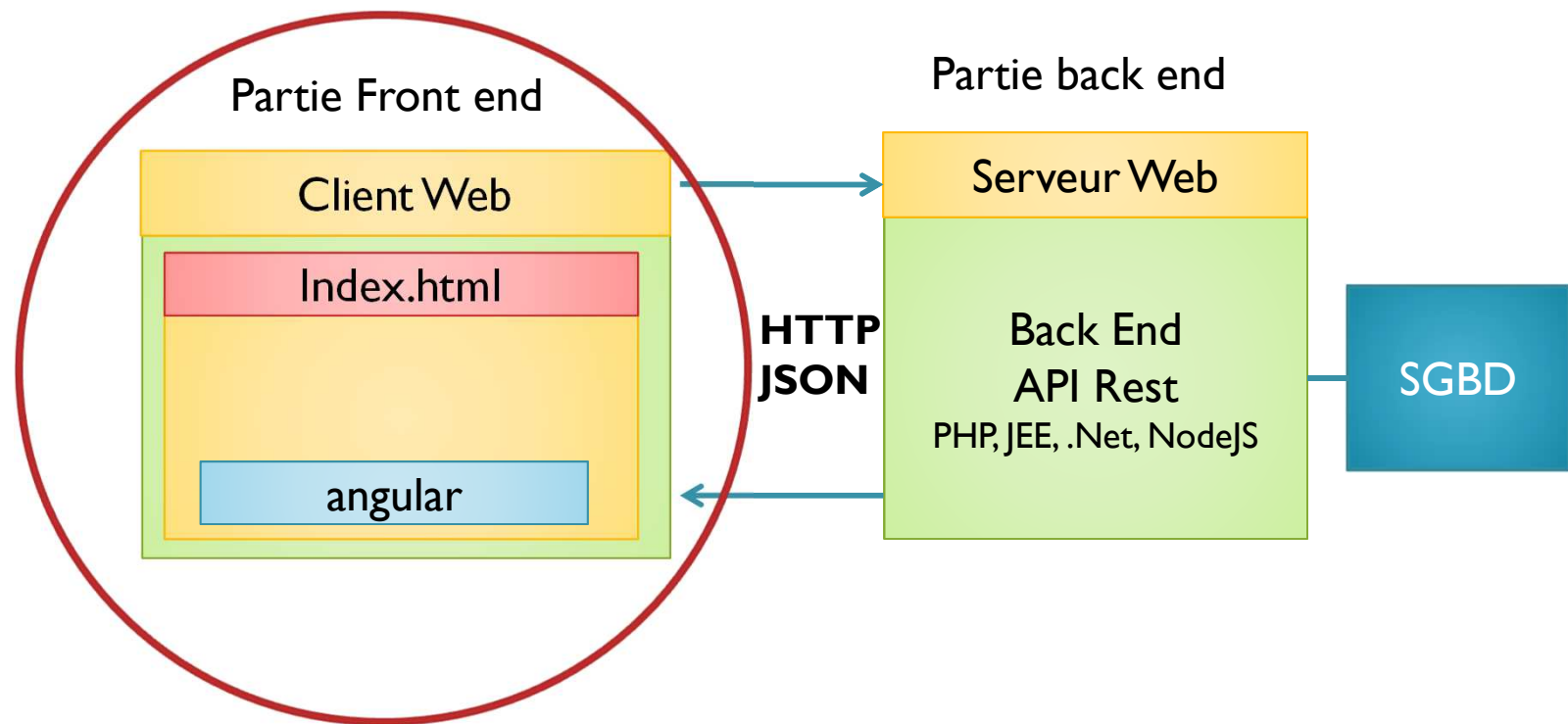
Angular avec Type Script

- Pour développer une application Angular il est recommandé d'utiliser Type Script qui sera compilé et traduit en Java Script.
- Type Script est un langage de script structuré et orienté objet qui permet de simplifier le développement d'applications Java Script



Single Page Application: SPA

- Angular Permet de créer la partie front end des applications web de type SPA (Single Page Application)



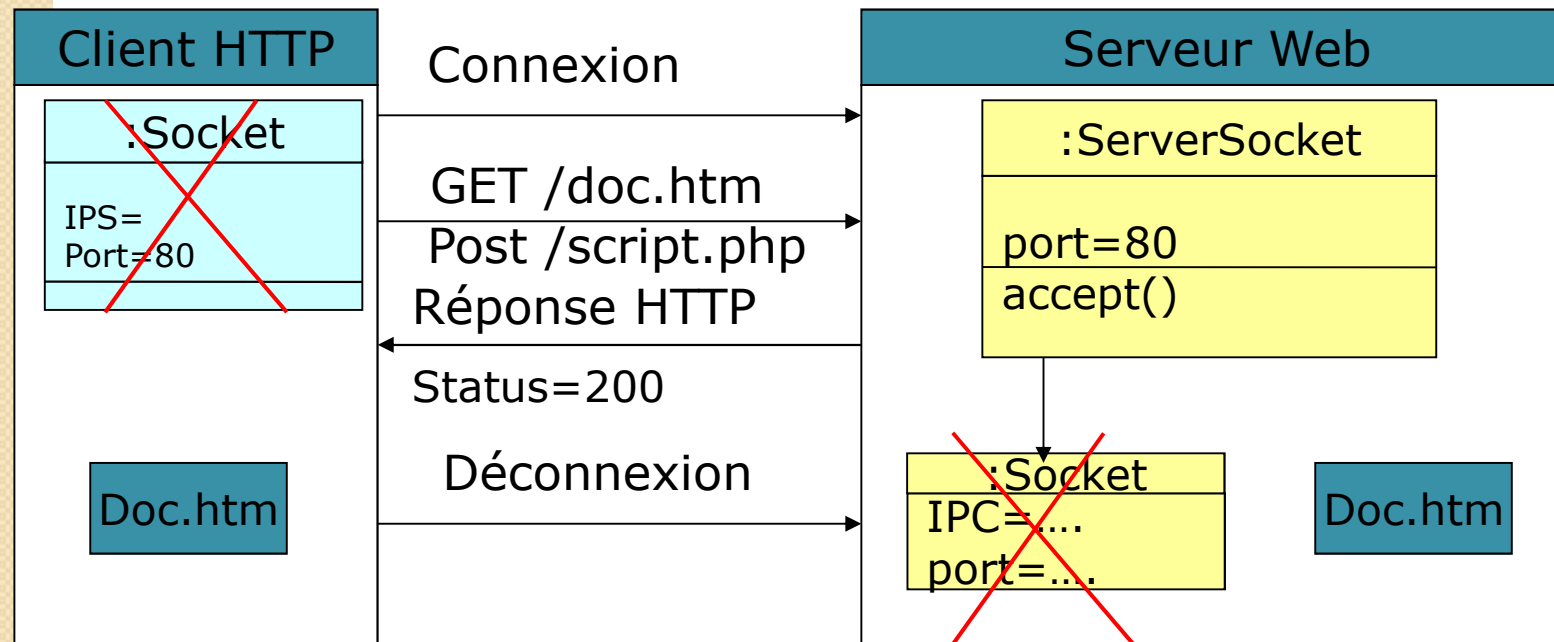


RAPPELS SUR HTTP

LE PROTOCOLE HTTP

- **HTTP :HyperText Tranfert Protocol**
 - Protocole qui permet au client de récupérer des documents du serveur
 - Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP,JSP,ASP...)
 - Ce protocole permet également de soumissionner les formulaires
- **Fonctionnement (très simple en HTTP/1.0)**
 - Le client se connecte au serveur (Créer une socket)
 - Le client demande au serveur un document : Requête HTTP
 - Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
 - Déconnexion

Connexion



Méthodes du protocole HTTP

- Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:
 - **GET** : Pour récupérer le contenu d'un document
 - **POST** : Pour soumissionner des formulaires (Envoyer, dans la requête, des données saisies par l'utilisateur)
 - **PUT** pour envoyer un fichier du client vers le serveur
 - **DELETE** permet de demander au serveur de supprimer un document.
 - **HEAD** permet de récupérer les informations sur un document (Type, Capacité, Date de dernière modification etc...)

Le client envoie la requête : Méthode POST

Entête de la requête

Post /Nom_Script HTTP/1.0

host: www.intra.net

ACCTEPT_LANGUAGE : fr

User-Agent : Mozilla/4.0

Méthode, chemin, version

Nom de domaine

Code de la langue

Type et version du navigateur

***** saut de ligne *****

**login=Value1& pass=Value2
& Var3=Value3**

Paramètres des différents champs du formulaire.

corps de la requête

Le client envoie la requête : Méthode GET

Entête de la requête

GET /Nom_Script?login=val1&pass=val2&.... HTTP/1.0
host: www.intra.net
ACCEPT_LANGUAGE : fr
User-Agent : Mozilla/4.0

corps de la requête est vide

Le Serveur retourne la réponse :

Entête de la réponse

HTTP/1.0 200 OK

Date : Wed, 05Feb02 15:02:01 GMT

Server : Apache/1.3.24

Last-Modified : Wed 02Oct01 24:05:01 GMT

Content-Type : Text/html

Content-length : 4205

Ligne de Status

Date du serveur

Nom du Serveur

Dernière modification

Type de contenu

Sa taille

***** saut de ligne *****

<HTML><HEAD>

....

</BODY></HTML>

*Le fichier que le client
va afficher*

corps de la réponse

Code de status

- Lorsque le serveur renvoie un document, il lui associe un code de statut renseignant ainsi le client sur le résultat de la requête (requête invalide, document non trouvé...).
- Les principales valeurs des codes de statut HTTP sont détaillées dans le tableau ci-après.
- Information 1xx :
 - 100 (Continue) : Utiliser dans le cas où la requête possède un corps.
 - 101 (Switching protocol) : Demander au client de changer de protocole. Par exemple de Http1.0 vers Http 1.1
- Succès 2xx :
 - 200 (OK) : Le document a été trouvé et son contenu suit
 - 201 (Created) : Le document a été créé en réponse à un PUT
 - 202 (Accepted) : Requête acceptée, mais traitement non terminé
 - 204 (No response) : Le serveur n'a aucune information à renvoyer
 - 206 (Partial content) : Une partie du document suit

Code de status

- Redirection 3xx :
 - 301 (Moved) : Le document a changé d'adresse de façon permanente
 - 302 (Found) : Le document a changé d'adresse temporairement
 - 304 (Not modified) : Le document demandé n'a pas été modifié
- Erreurs du client 4xx :
 - 400 (Bad request) : La syntaxe de la requête est incorrecte
 - 401 (Unauthorized) : Le client n'a pas les privilèges d'accès au document
 - 403 (Forbidden) : L'accès au document est interdit
 - 404 (Not found) : Le document demandé n'a pu être trouvé
 - 405 (Method not allowed): La méthode de la requête n'est pas autorisée
- Erreurs du serveur 5xx :
 - 500 (Internal error) : Une erreur inattendue est survenue au niveau du serveur
 - 501 (Not implemented) : La méthode utilisée n'est pas implémentée
 - 502 (Bad gateway) : Erreur de serveur distant lors d'une requête proxy

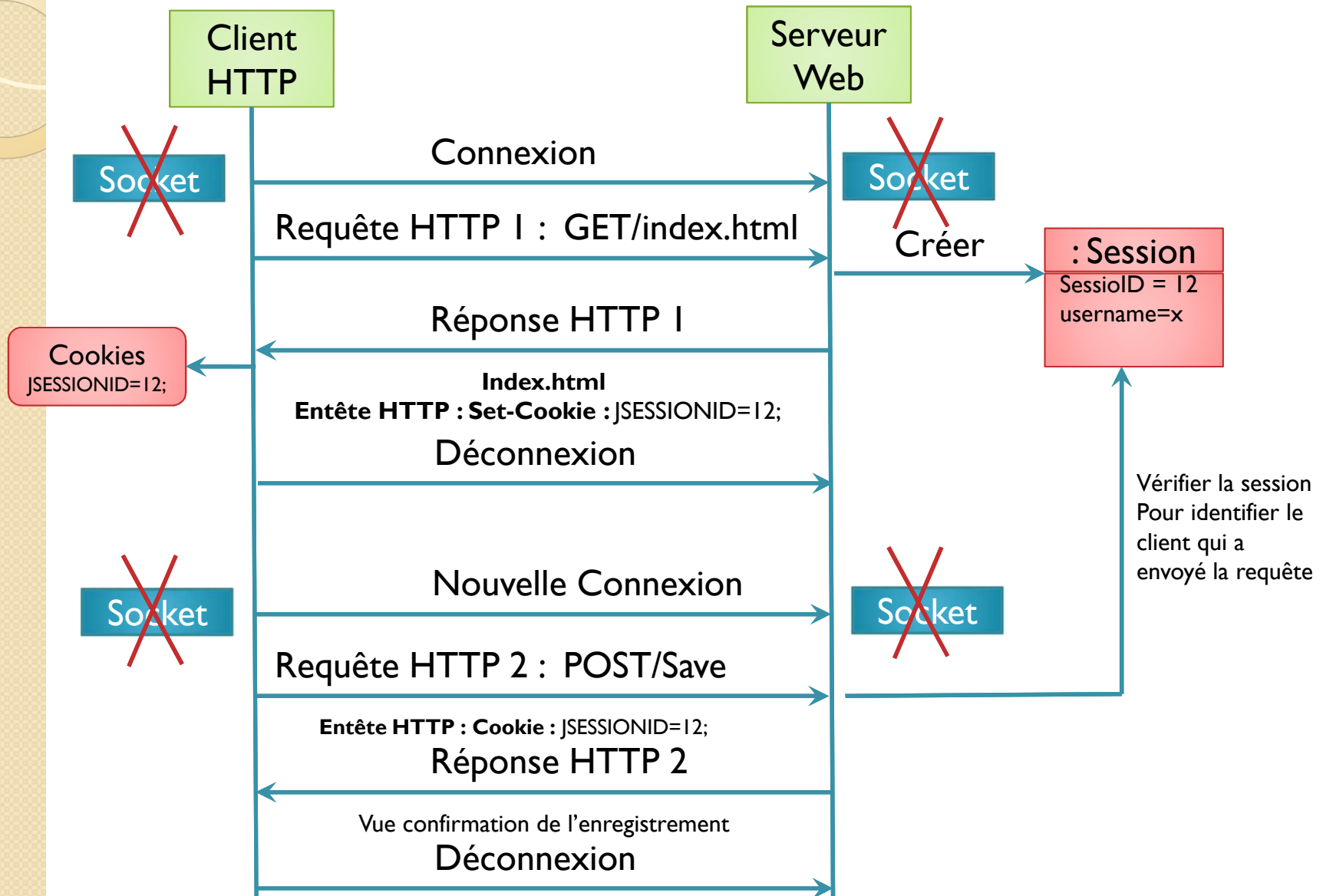
Entêtes HTTP

- Entêtes HTTP génériques :
 - Content-length : Longueur en octets des données suivant les en-têtes
 - Content-type : Type MIME des données qui suivent
 - Connection : Indique si la connexion TCP doit rester ouverte (Keep-Alive) ou être fermée (close)
- Entêtes de la requête :
 - Accept : Types MIME que le client accepte
 - Accept-encoding: Méthodes de compression supportées par le client
 - Accept-language: Langues préférées par le client (pondérées)
 - Cookie : Données de cookie mémorisées par le client
 - Host : Hôte virtuel demandé
 - If-modified-since : Ne retourne le document que si modifié depuis la date indiquée
 - If-none-match : Ne retourne le document que s'il a changé
 - Referer : URL de la page à partir de laquelle le document est demandé
 - User-agent : Nom et version du logiciel client

Entêtes de réponse

- Allowed : Méthodes HTTP autorisées pour cette URI (comme POST)
- Content-encoding : Méthode de compression des données qui suivent
- Content-language : Langue dans laquelle le document retourné est écrit
- Date : Date et heure UTC courante
- Expires : Date à laquelle le document expire
- Last-modified : Date de dernière modification du document
- Location : Adresse du document lors d'une redirection
- Etag : Numéro de version du document
- Pragma : Données annexes pour le navigateur (par exemple, no.cache)
- Server : Nom et version du logiciel serveur
- Set-cookie : Permet au serveur d'écrire un cookie sur le disque du client

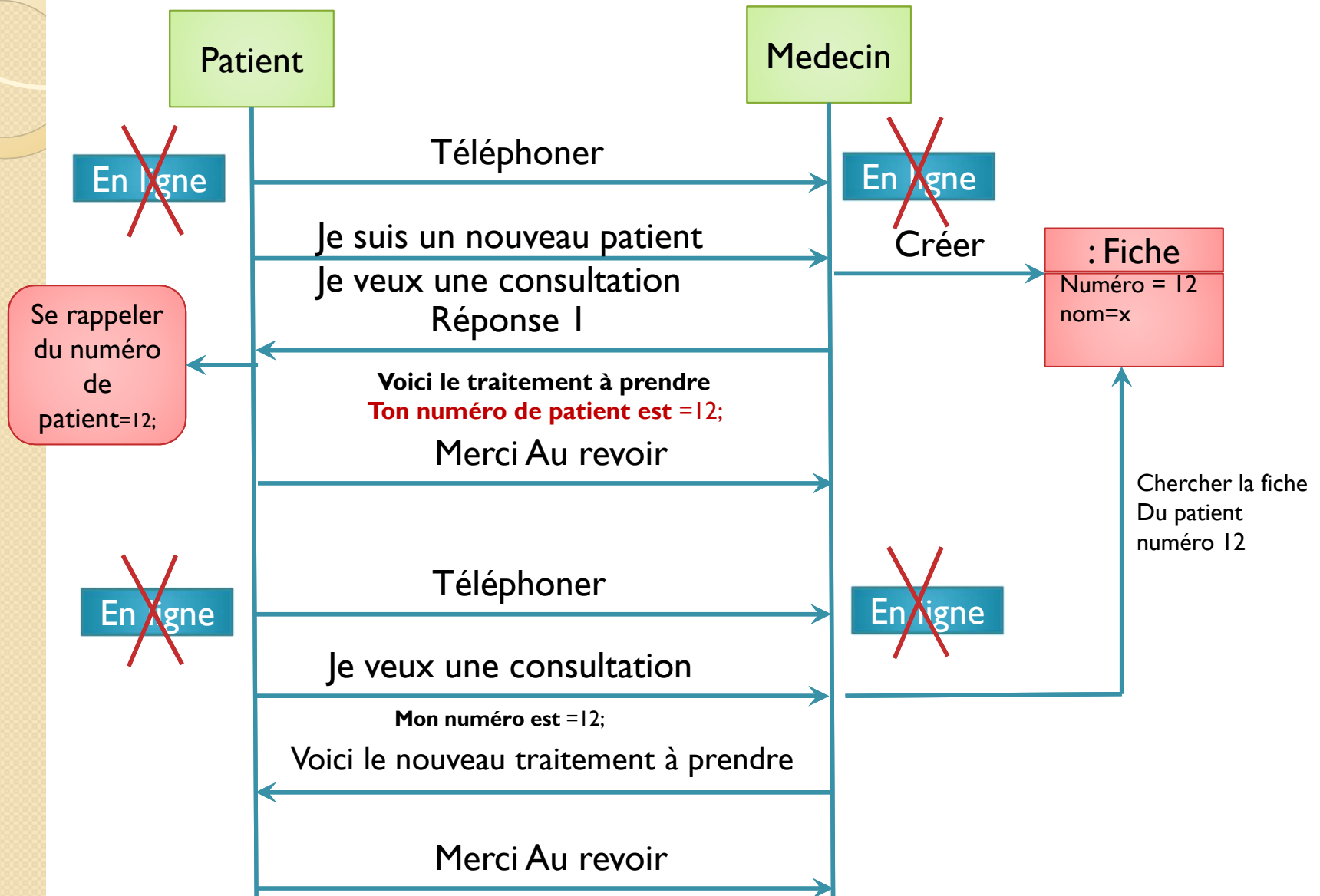
Session et Cookies



Utilisation des sessions et des cookies

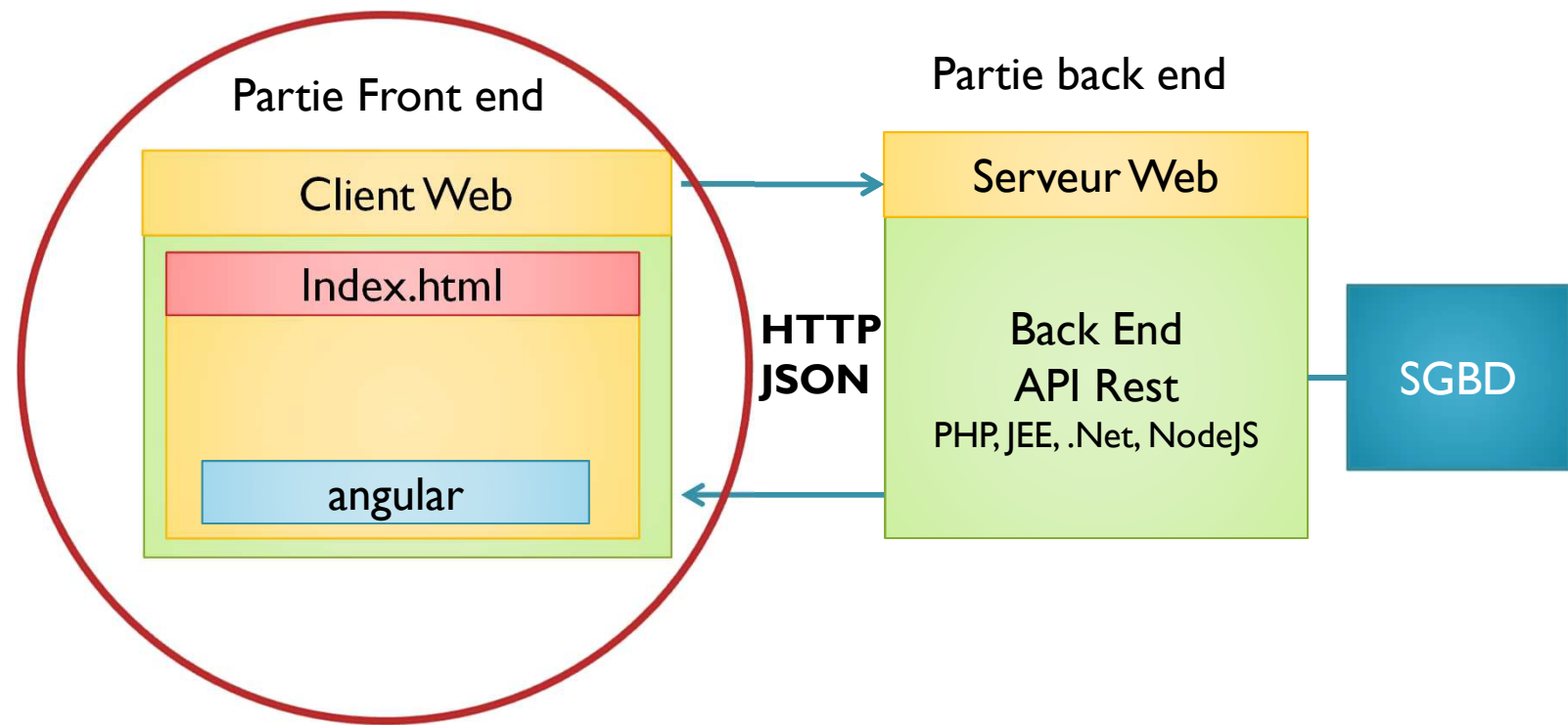
- Généralement quand un client HTTP envoie sa première requête, le serveur web crée une session pour ce client.
- Une session est un objet stocké dans la mémoire du serveur qui peut servir pour stocker des informations relatives au client.
- Le serveur attribut un SessionID unique à chaque session.
- Ce SessionID est ensuite envoyé dans la réponse http en sousforme d'un cookie en utilisant l'entête de la réponse HTTP :
 - **Set-Cookie** : JSESSIONID=F84DB7B959F76B183DBF05F999FAEE11;
- Ce qui signifie que le serveur demande au client d'enregistrer ce SESSIONID dans un fichier stocké dans la machine du client appelé COOKIE.
- Une fois que le client reçoit la réponse HTTP, la connexion est fermée.
- A chaque fois que le client envoie une requête HTTP vers le serveur, il envoie toujours les données des cookies dont le SESSIONID.
- Les cookies sont envoyés dans la requête HTTP en utilisant une entête COOKIE:
 - **Cookie**: JSESSIONID=F84DB7B959F76B183DBF05F999FAEE11
- Grace à cette information, le serveur peut savoir de quel client s'agit-il même s'il s'agit d'une nouvelle connexion.

Session et Cookies > Patient et médecin



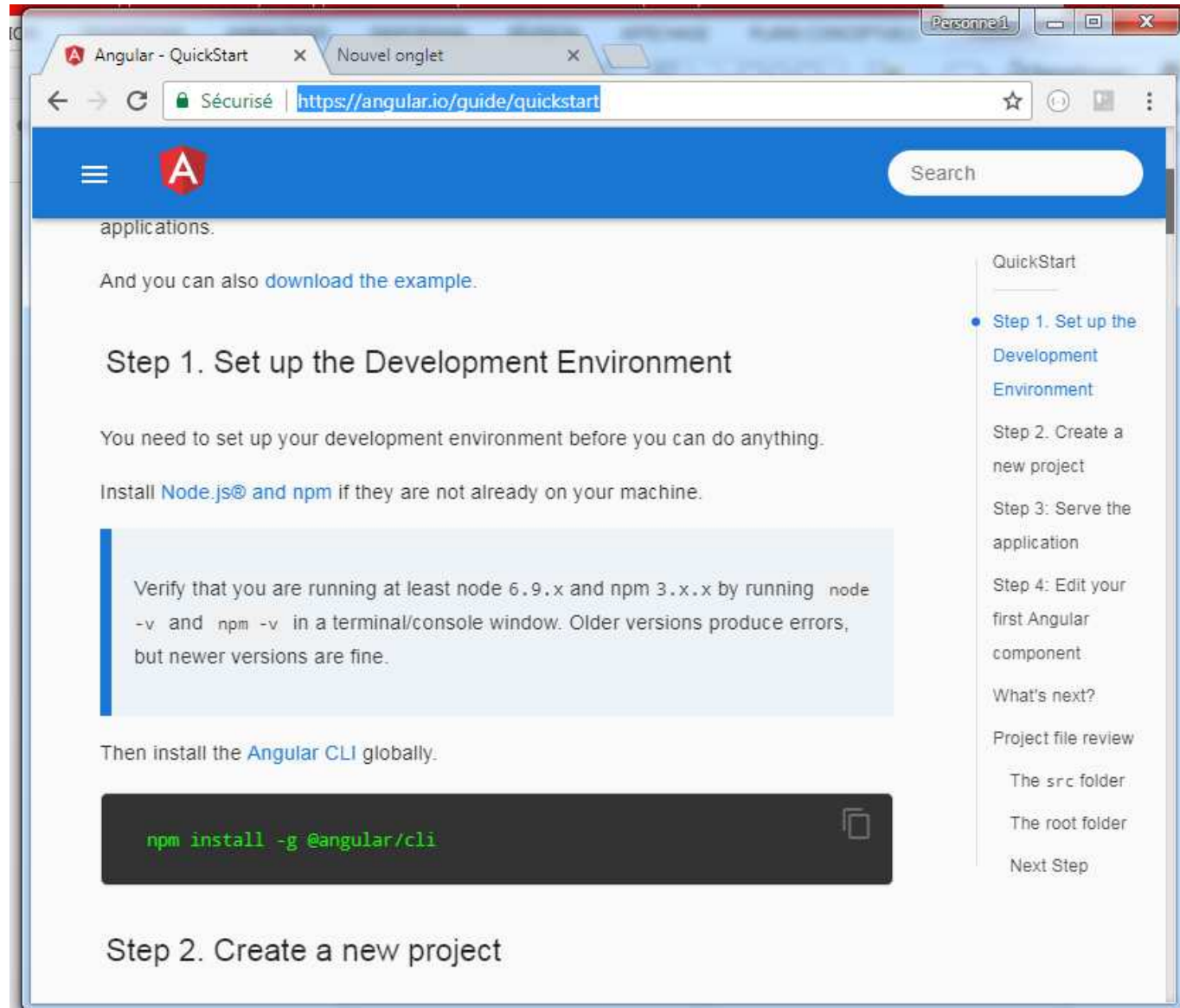
Single Page Application: SPA

- Angular Permet de créer la partie front end des applications web de type SPA (Single Page Application)



Démarre avec Angular

- <https://angular.io/guide/quickstart>



The screenshot shows a web browser window with the Angular QuickStart guide. The browser has two tabs: 'Angular - QuickStart' and 'Nouvel onglet'. The address bar shows the URL 'https://angular.io/guide/quickstart'. The page has a blue header with the Angular logo and a search bar. The main content area is titled 'Step 1. Set up the Development Environment'. It includes instructions on setting up the development environment, including installing Node.js and npm, and verifying the versions. A code block shows the command 'npm install -g @angular/cli'. The right sidebar contains a table of contents with links to 'QuickStart', 'Step 1. Set up the Development Environment', 'Step 2. Create a new project', 'Step 3: Serve the application', 'Step 4: Edit your first Angular component', 'What's next?', 'Project file review', 'The src folder', 'The root folder', and 'Next Step'.

applications.

And you can also [download the example](#).

Step 1. Set up the Development Environment

You need to set up your development environment before you can do anything.

Install [Node.js@](#) and [npm](#) if they are not already on your machine.

Verify that you are running at least node 6.9.x and npm 3.x.x by running `node -v` and `npm -v` in a terminal/console window. Older versions produce errors, but newer versions are fine.

Then install the [Angular CLI](#) globally.

```
npm install -g @angular/cli
```

Step 2. Create a new project

QuickStart

- [Step 1. Set up the Development Environment](#)
- [Step 2. Create a new project](#)
- [Step 3: Serve the application](#)
- [Step 4: Edit your first Angular component](#)
- [What's next?](#)
- [Project file review](#)
- [The src folder](#)
- [The root folder](#)
- [Next Step](#)

Installation des outils

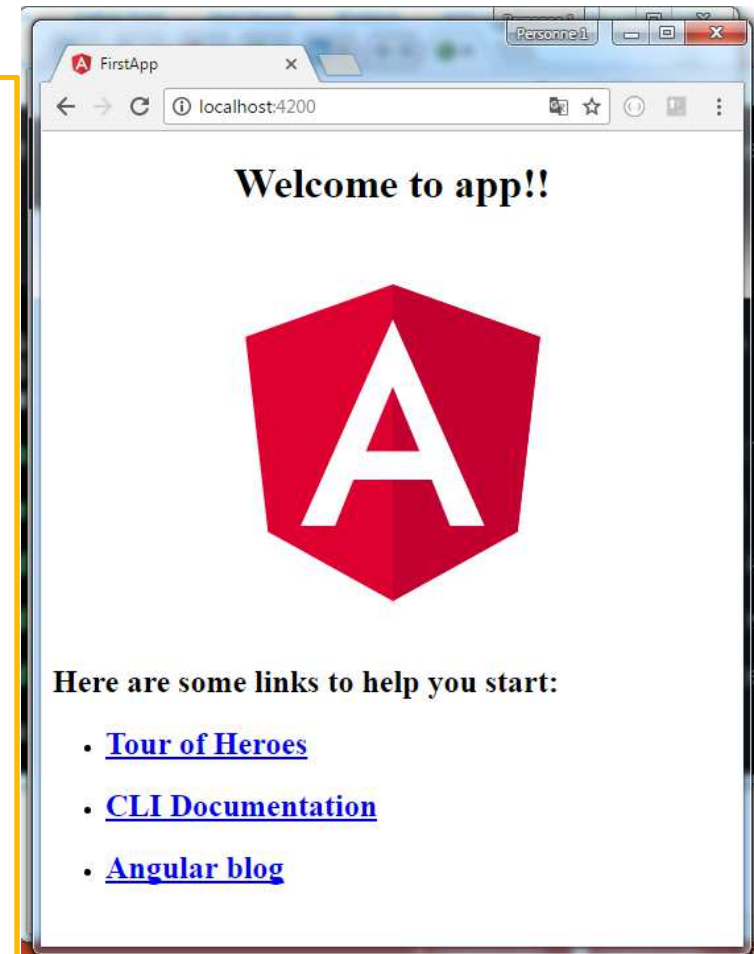
- Pour faciliter le développement d'une application Angular, les outils suivant doivent être installés :
 - Node JS : <https://nodejs.org/en/download/>
 - Node JS installe l'outil npm (Node Package Manager) qui permet de télécharger et installer des bibliothèques Java Script.
 - Installer ensuite Angular CLI (Command Line Interface) qui vous permet de générer, compiler, tester et déployer des projets angular (<https://cli.angular.io/>) :
 - **npm install -g @angular/cli**

Création d'un nouveau projet Angular

- Pour générer la structure d'un projet Angular, on utilise Angular CLI via sa commande `ng` suivie des options `new` et le nom du projet.
 - `ng new FirstApp`
- Cette commande génère les différents fichiers requis par une application basique Angular et installe aussi toutes les dépendances requise par ce projet.

Exécuter un projet Angular

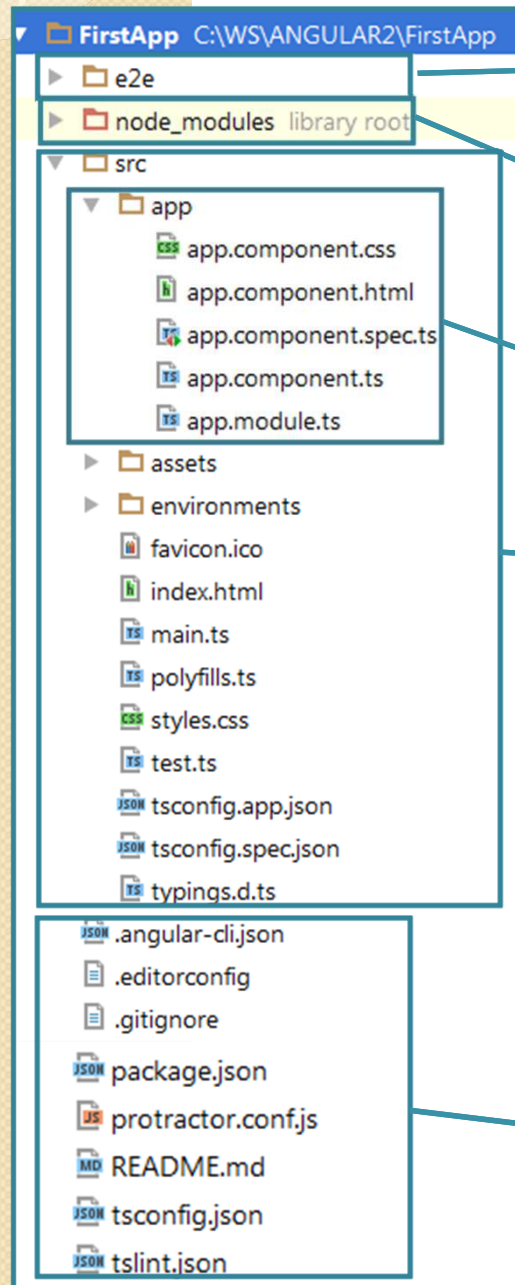
- Pour excuter un projet Angular, on exécuter la commande suivante à partir de la racine du projet
 - **ng serve**
- Cette commande compile le code source du projet pour transpiler le code TypeScript en Java Script et en même temps démarre un serveur Web local basé sur Node JS pour déployer l'application localement.
- Pour tester le projet généré, il suffit de lancer le Browser et taper l'url : http:// localhost:4200
- Dans l'étape suivante, nous allons regarder la structure du projet généré par Angular CLI.



Edition du projet

- Plusieurs IDE professionnels peuvent être utilisé pour éditer le code:
 - Web Storm, PHP Storm
 - Visual Studio Code
 - Eclipse avec plugin Angular
- D'autres éditeurs classiques peuvent être aussi utilisé :
 - Atom
 - Sublime Text
 - Etc ...

Structure du projet Angular



Fichier de configuration du projet

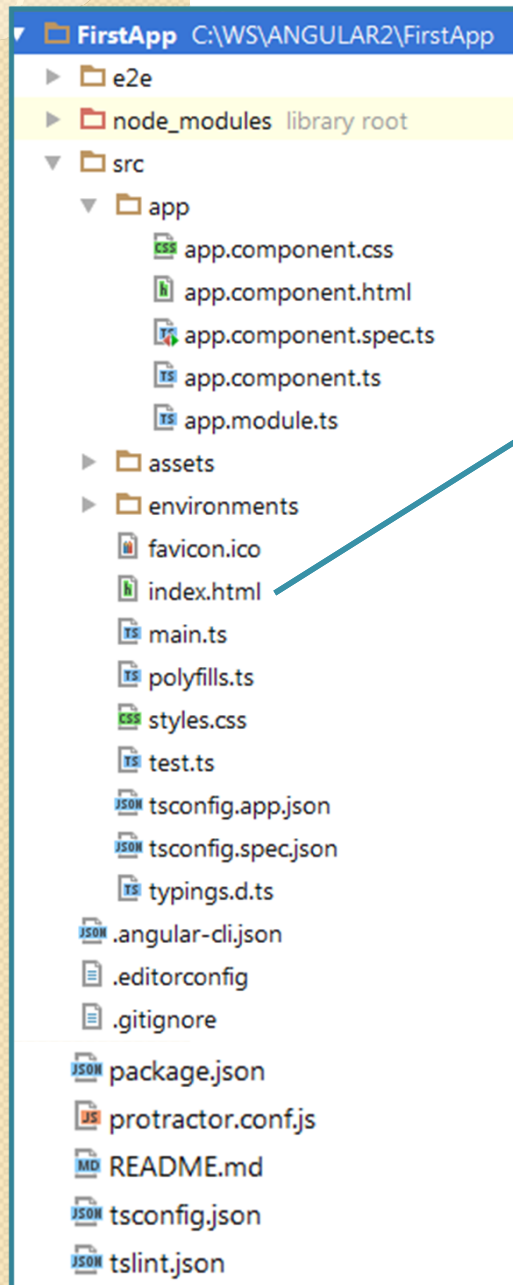
Dépendances externes du projets (Librairies Java Script et CSS)

Logique applicative de votre projet : Les composants, Les services ..
C'est dans ce dossier app ou vous allez passer votre temps de dev

Code source relatif à votre projet

Fichier de configuration du projet

Structure du projet Angular



index.html

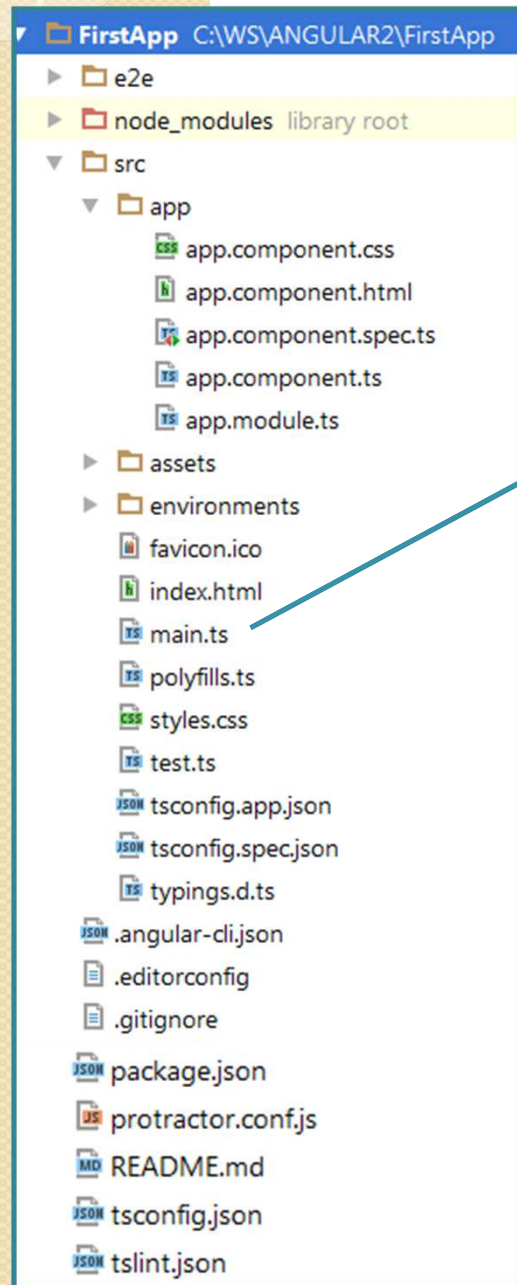
```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>FirstApp</title>
  <base href="/">

  <meta name="viewport" content="width=device-
width, initial-scale=1">
  <link rel="icon" type="image/x-icon"
href="favicon.ico">
</head>
<body>

  <app-root> </app-root>

</body>
</html>
```

Structure du projet Angular



main.ts

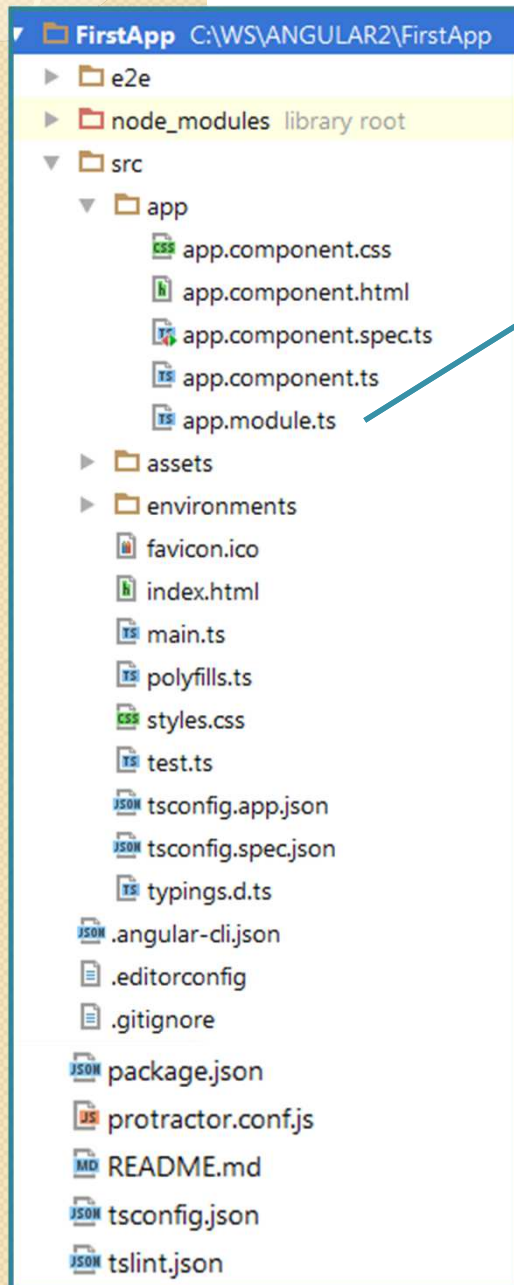
```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from
'@angular/platform-browser-dynamic';

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

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

Structure du projet Angular



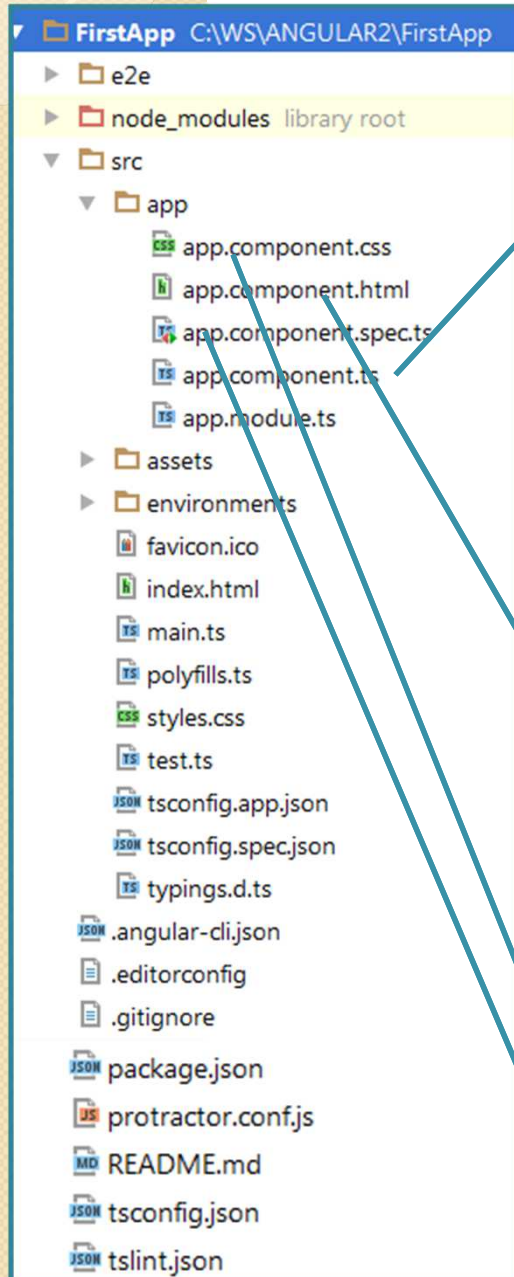
app.module.ts

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

import { AppComponent } from './app.component';

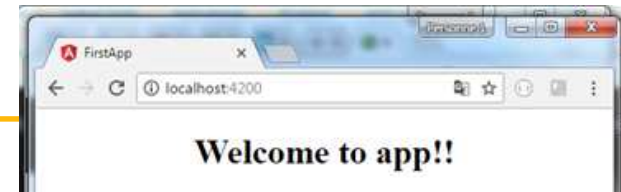
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```


Structure du projet Angular



app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
}
```



app.component.html

```
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!!
  </h1>
</div>
```

app.component.css

Les fichiers spec sont des tests unitaires pour vos fichiers source. La convention pour les applications Angular2 est d'avoir un fichier .spec.ts pour chaque fichier .ts. Ils sont exécutés à l'aide du framework de test javascript Jasmine via le programme de tâches Karma lorsque vous utilisez la commande '**ng test**'.

Plan de cours

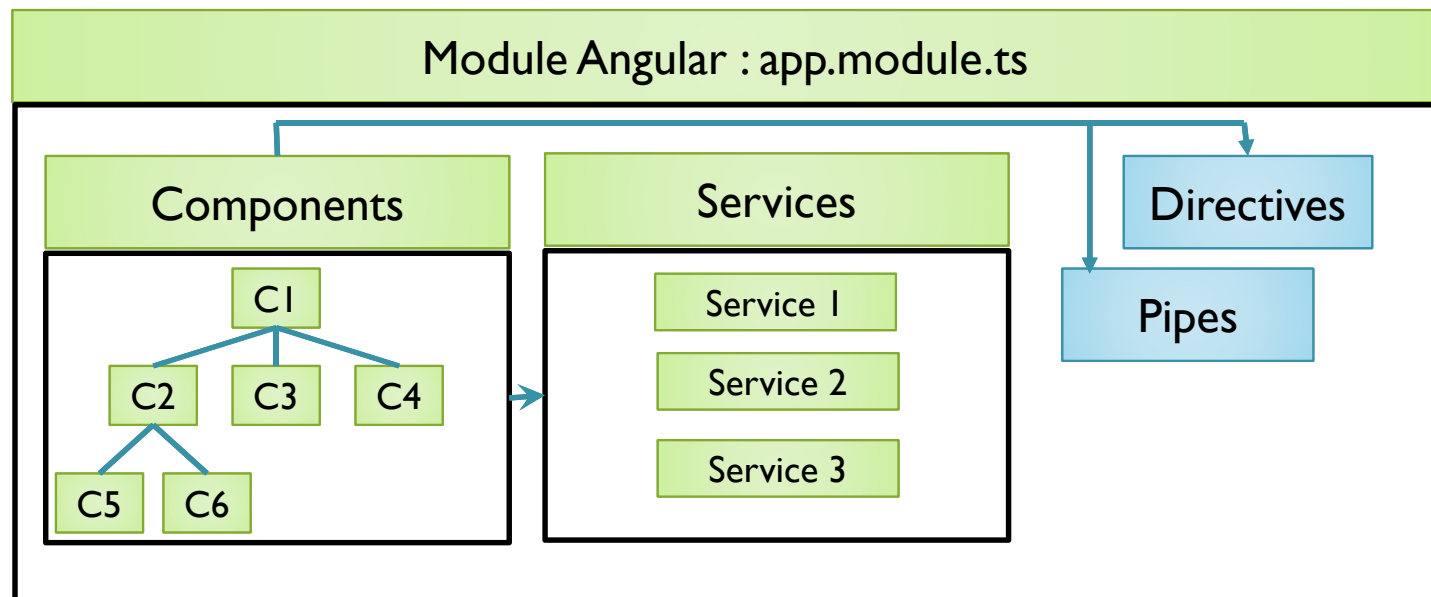
- Démarrage de Angular
- Le bases de Angular
- Components and Data Binding
- Directives
- Services et Injection de dépendances
- Routage
- Observables
- Forms
- Pipes
- http
- Authentification
- Optimisation et NgModules
- Deployment
- Animations
- Tests

Type Script

- **TypeScript** est un langage de programmation libre et open source développé par Microsoft qui a pour but d'améliorer et de sécuriser la production de code JavaScript.
- C'est un sur-ensemble de JavaScript (c'est-à-dire que tout code JavaScript correct peut être utilisé avec TypeScript).
- Le code TypeScript est transcompilé en JavaScript, pouvant ainsi être interprété par n'importe quel navigateur web ou moteur JavaScript.
- Il a été cocréé par Anders Hejlsberg, principal inventeur de C#
- TypeScript permet un typage statique optionnel des variables et des fonctions, la création de classes et d'interfaces, l'import de modules, tout en conservant l'approche non-contraignante de JavaScript.
- Il supporte la spécification ECMAScript 6.

Architecture de Angular

- Angular est un Framework pour créer la partie Front End des applications web en utilisant HTML et JavaScript ou TypeScript compilé en JavaScript.
- Une application Angular se compose de :
 - Un à plusieurs modules dont un est principal.
 - Chaque module peut inclure :
 - Des composant web : La partie visible de la 'application Web (IHM)
 - Des services pour la logique applicative. Les composants peuvent utiliser les services via le principe de l'injection des dépendances.
 - Les directives : un composant peut utiliser des directives
 - Les pipes : utilisés pour formater l'affichage des données dans els composants.



Modules

- Les applications angulaires sont modulaires
- Angular possède son propre système de modularité appelé modules angulaires ou NgModules.
- Chaque application Angular possède au moins une classe de module angulaire: le module racine, appelé classiquement **AppModule**.
- Un module angulaire est une classe avec un décorateur **@NgModule**.
- Les décorateurs sont des fonctions qui modifient les classes JavaScript.
- Angular possède de nombreux décorateurs qui attachent des métadonnées aux classes pour configurer et donner le sens à ces classes.

src/app/app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports: [ BrowserModule ],
  providers: [ Logger ],
  declarations: [ AppComponent ],
  exports: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

@NgModule

- @NgModule est un décorateur qui prend en parameter un objet javascript qui contient des métadonnées dont les propriétés décrivent le module. Les propriétés les plus importantes sont:
 - **declarations** : la classe représentant le module. Angular a trois types de classes de modules : **components**, **directives**, and **pipes**.
 - **exports** – Pour exporter les classes utilisables dans d'autres modules.
 - **imports** – Pour importer d'autres modules.
 - **providers** – Pour déclarer les fabriques de services.
 - **bootstrap** – Pour déclarer le composant Racine du module. Seul le module racine doit définir cette propriété.

src/app/app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports: [ BrowserModule ],
  providers: [ Logger ],
  declarations: [ AppComponent ],
  exports: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Démarrage de l'application

- Le module racine est démarré dans le fichier main.ts
- Par défaut le module racine s'appelle AppModule

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

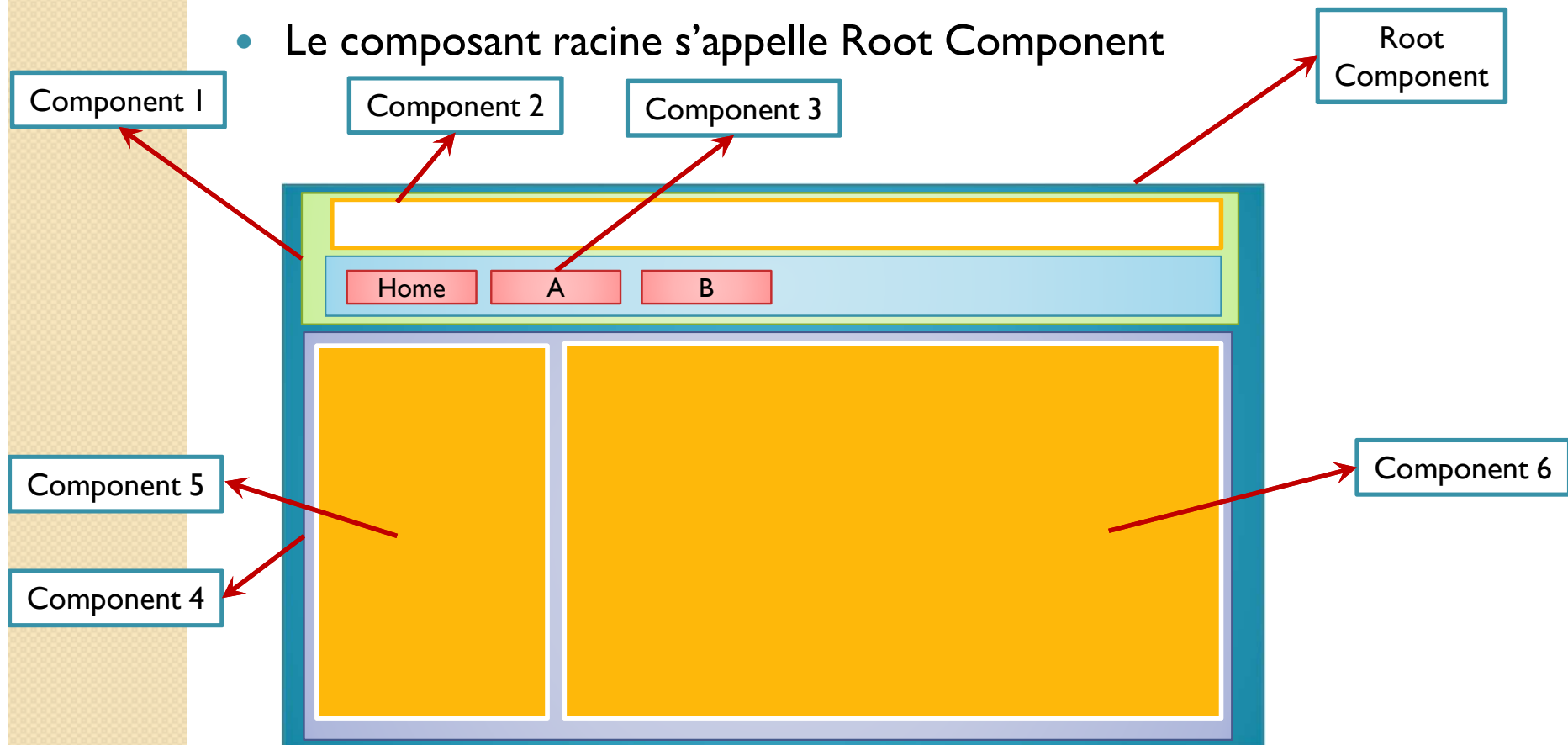
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

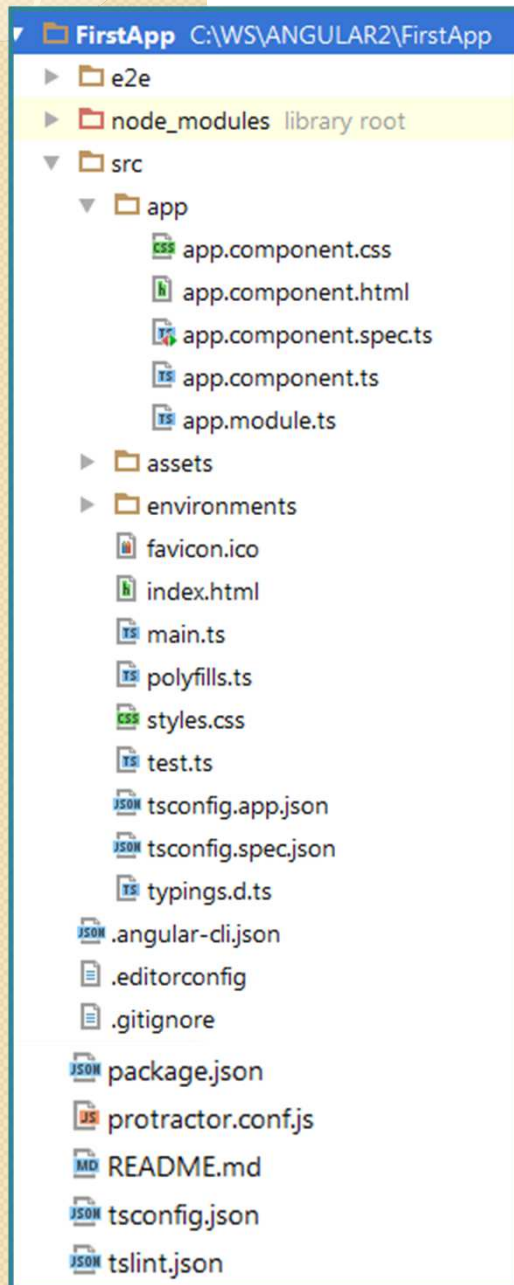

Components

- Les composants sont des éléments importants dans Angular.
- L'application est formée par un ensemble de composants.
- Chaque composant peut imbriquer d'autres composants définissant ainsi une structure hiérarchique.
- Le composant racine s'appelle Root Component



Components

- Chaque composant se compose principalement des éléments suivants:
 - HTML Template : représentant sa vue
 - Une classe représentant sa logique métier
 - Une feuille de style CSS
 - Un fichier spec sont des tests unitaires Ils sont exécutés à l'aide du framework de test javascript Jasmine via le programme de tâches Karma lorsque vous utilisez la commande '**ng test**'.
- Les composants sont facile à mettre à jour et à échanger entre les différentes parties des applications.



app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
}
```

app.component.html

```
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!!
  </h1>
</div>
```

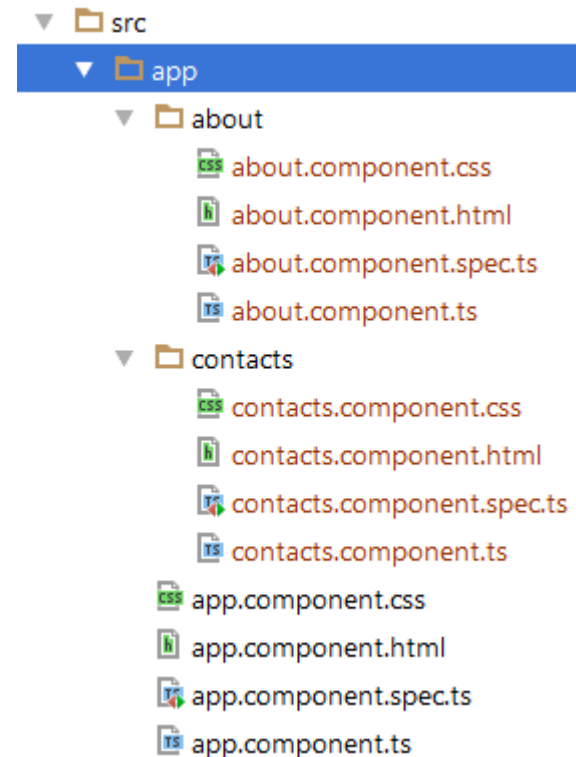
app.component.css

Création de nouveaux composants

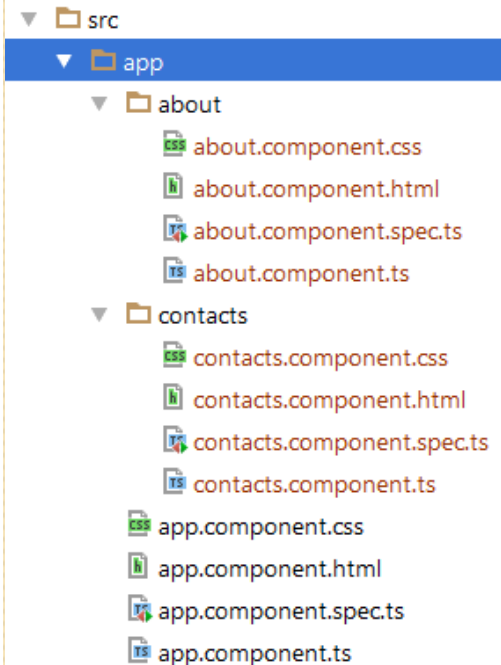
- Pour créer facilement des composants Angular, on peut utiliser à nouveau la commande ng comme suit :
 - **ng** generate component NomComposant
- Dans notre exemple, nous allons créer deux composants : about et contacts

```
Terminal
+ C:\WS\ANGULAR2\FirstApp>ng generate component about
X installing component
  create src\app\about\about.component.css
  create src\app\about\about.component.html
  create src\app\about\about.component.spec.ts
  create src\app\about\about.component.ts
  update src\app\app.module.ts
```

```
Terminal
+ C:\WS\ANGULAR2\FirstApp>ng generate component contacts
X installing component
  create src\app\contacts\contacts.component.css
  create src\app\contacts\contacts.component.html
  create src\app\contacts\contacts.component.spec.ts
  create src\app\contacts\contacts.component.ts
  update src\app\app.module.ts
```

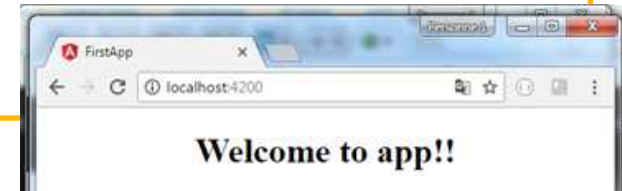


Structure du composant about



about.component.ts

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.css']
})
export class AboutComponent implements OnInit {
  constructor() { }
  ngOnInit() {
  }
}
```



about.component.html

```
<p>
  about works!
</p>
```

about.component.css

@Component

- Un composant est une classe qui possède le décorateur @Component
- Ce décorateur possède les propriétés suivantes :
 - **selector** : indique la déclaration qui permet d'insérer le composant dans le document HTML. Cette déclaration peut être :
 - Le nom de la balise associé à ce composant
 - **selector** : `app-about`
 - Dans ce cas le composant sera inséré par : `<app-about></app-about>`
 - Le nom de l'attribut associé à ce composant :
 - **selector** : `[app-about]`
 - Dans ce cas le composant sera inséré par : `<div app-about></div>`
 - Le nom de la classe associé à ce composant :
 - **selector** : `.app-about`
 - Dans ce cas le composant sera inséré par : `<div class="app-about"></div>`
 - **template ou templateUrl** :
 - **template** : permet de définir dans à l'intérieur du décorateur le code HTML représentant la vue du composant
 - **templateUrl** : permet d'associer un fichier externe HTML contenant la structure de la vue du composant
 - **styleUrls** : spécifier les feuilles de styles CSS associées à ce composant

Déclaration du composant

- Pour utiliser un composant, ce dernier doit être déclaré dans le module :

App.module.ts

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

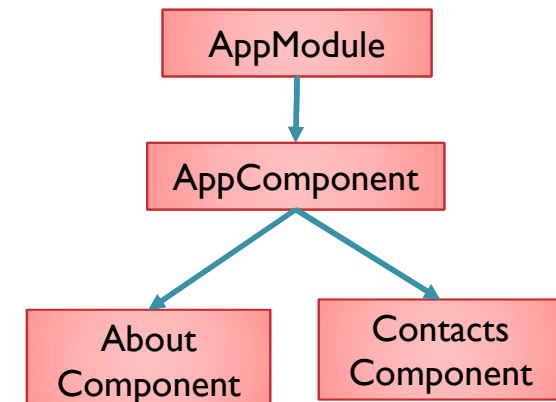
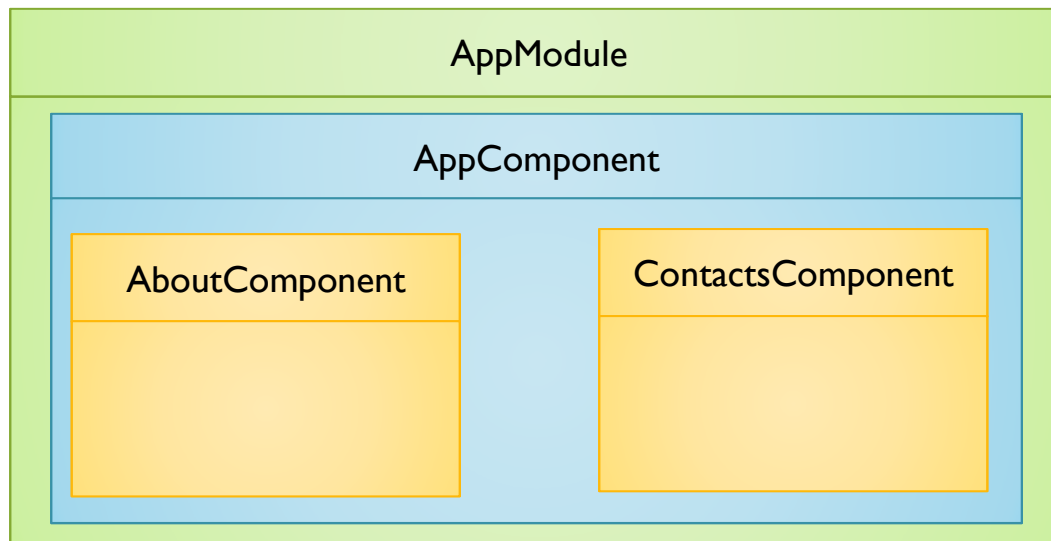
import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { ContactsComponent } from './contacts/contacts.component';

@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    ContactsComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Utilisation composant

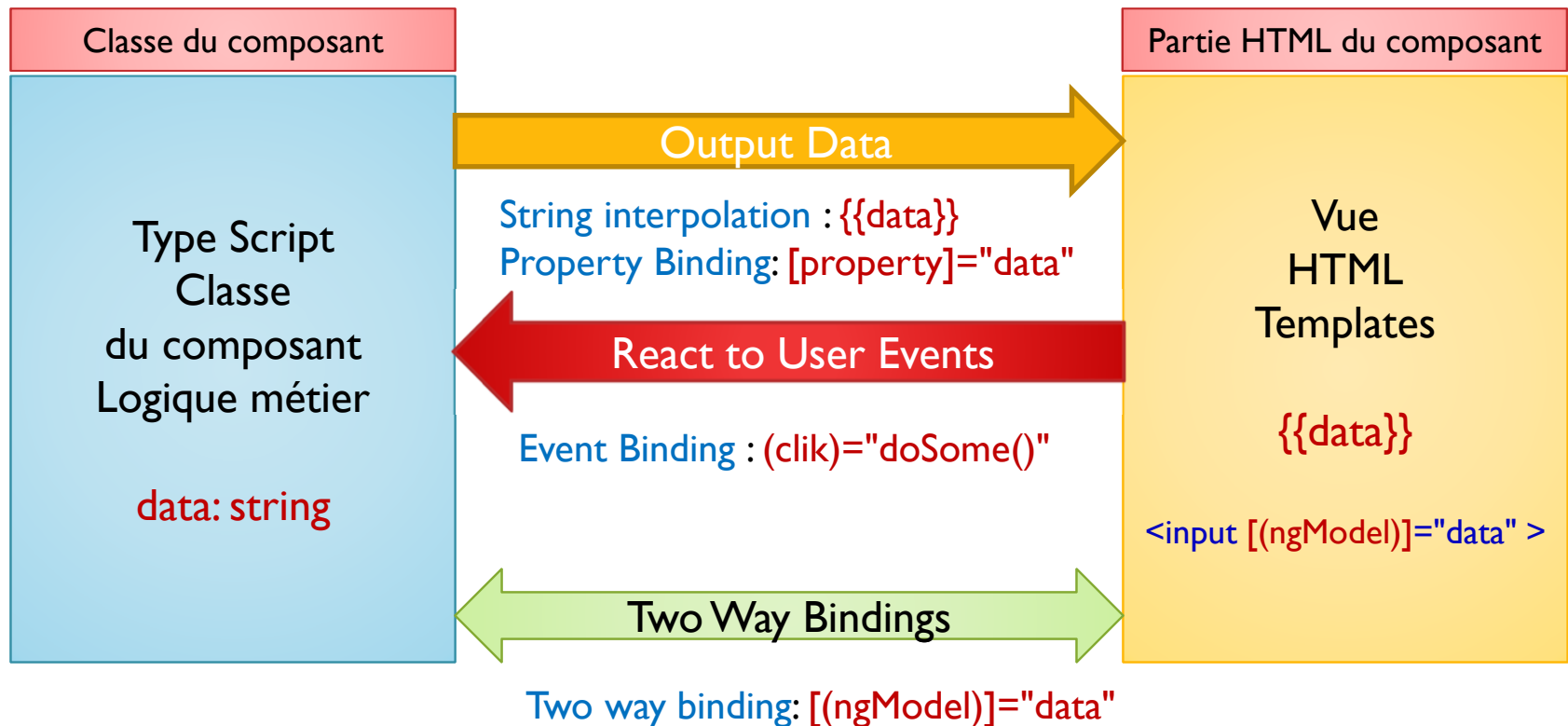
- Un composant peut être inséré dans n'importe que partie HTML de l'application en utilisant son selecteur associé.
- Dans cet exemple les deux composants générés sont insérés à l'intérieur du composant racine AppComponent

```
<div style="text-align:center">  
  <h1>  
    Welcome to {{title}}!!  
  </h1>  
  <app-about></app-about>  
  <div app-contacts></div>  
</div>
```



Data Binding

- Pour insérer dynamiquement des données de l'applications dans les vues des composants, Angular définit des techniques pour assurer la liaison des données.
- Data Binding = Communication



Exemples de Data Binding :



localhost:4200

about.component.ts

```
export class AboutComponent {  
  info={ nom:"Mohamed",  
    email:"med@youssfi.net",  
    tel:"0661326837"  
  };  
  comments=[];  
  
  comment={ id:0,message:'' };  
  
  newComment=false;  
  
  addComment () {  
    if (this.comment.message!='') {  
      this.comment.id=this.comments.length+1;  
      this.comments.push({  
        id:this.comment.id,  
        message:this.comment.message  
      });  
      this.comment.message='';  
    }  
  }  
}
```

Welcome to app!!

- Nom : Mohamed
- Email : med@youssfi.net
- Tel : 0661326837

Liste des messages :

- 1 : aaaaa
- 2 : bbbb
- 3 : cccc

contacts works!

```
</ul>  
</div>  
<ng-template #noComments>  
  <p> Liste des commentaires est vide</p>  
</ng-template>
```

Exemples de Data Binding :

about.component.ts

```
export class AboutComponent {  
  info={ nom:"Mohamed",  
    email:"med@yousfsi.net",  
    tel:"0661326837"  
  };  
  comments=[];  
  comment={ date:null, message:'' };  
  newComment=false;  
  
  addComment () {  
    if(this.comment.message!='') {  
      this.comment.date=new  
Date();  
      this.comments.push({  
        date:this.comment.date,  
        message:this.comment.message  
      });  
      this.comment.message='';  
    }  
  }  
}
```

about.component.html

```
<ul>  
  <li>Nom      : {{info.nom}}      </li>  
  <li>Email    : {{info.email}}    </li>  
  <li>Tel      : {{info.tel}}      </li>  
</ul>  
<div>  
  <input  
    type="text"  
    [(ngModel)]="comment.message" >  
  <button  
    (click)="addComment () "  
    [disabled]="newComment">  
    Add comment  
  </button>  
</div>  
<div *ngIf="comments.length>0; else  
noComments">  
  <h3>Liste des messages :</h3>  
  <ul>  
    <li *ngFor="let c of comments">  
      {{c.date}} : {{c.message}}</li>  
  </ul>  
</div>  
<ng-template #noComments>  
  <p> Liste des commentaires est vide</p>  
</ng-template>
```

Importer FormsModule pour ngModel

- Pour utiliser la directive ngModel, il est nécessaire d'importer le module FormsModule de Angular.

app.module.ts

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

import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';

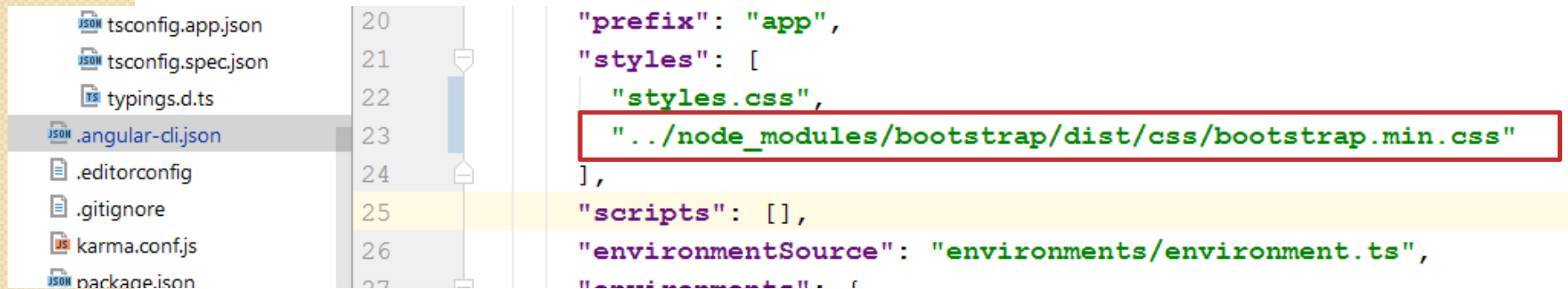
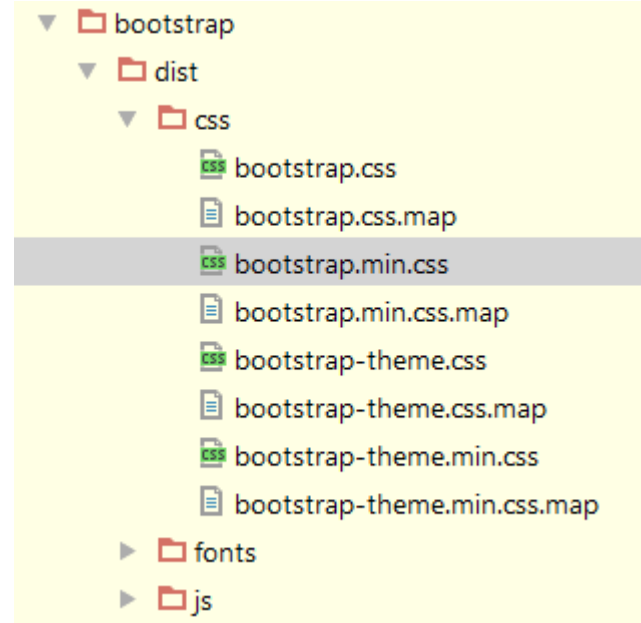
import { ContactsComponent } from './contacts/contacts.component';

import {FormsModule} from "@angular/forms";

@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    ContactsComponent
  ],
  imports: [
    BrowserModule, FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Ajouter le Framework bootstrap css au projet

- L'un des moyen les plus élégant pour ajouter de nouveau liens vers des fichiers css à index.html est de le faire à l'aide Angular CLI.
- Il faut d'abord installer bootstrap avec npm:
 - `npm install --save bootstrap@3`
- Ensuite, ajouter le chemin de bootstrap.min.css dans le fichier `.angular-cli.json`
- Ensuite démarre le serveur avec `ng serve`



Nouveau Look avec Bootstrap

```
<div class="container">
<div class="panel panel-primary">
  <div class="panel-heading">About Component</div>
  <div class="panel-body">
    <ul class="">
      <li>Nom : {{info.nom}}</li>
      <li>Email : {{info.email}}</li>
      <li>Tel : {{info.tel}}</li>
    </ul>
    <div class="form-group">
      <label>Message:</label>
      <input type="text" [(ngModel)]="comment.message">
      <button (click)="addComment()"
[disabled]="newComment" class="btn btn-primary">Add
comment</button>
    </div>
    <div *ngIf="comments.length>0; else noComments">
      <h3>Liste des messages :</h3>
      <ul class="list-group">
        <li *ngFor="let c of comments" class="list-group-
item">
          {{c.message}}<span
class="badge">{{c.date | date:'HH:mm:ss'}}</span></li>
      </ul>
    </div>
    <ng-template #noComments>
      <p>Liste des commentaires est vide</p>
    </ng-template>
  </div>
</div>
</div>
```

localhost:4200

Welcome to app!!

About Component

- Nom : Mohamed
- Email : med@yousfi.net
- Tel : 0661326837

Message: Add comment

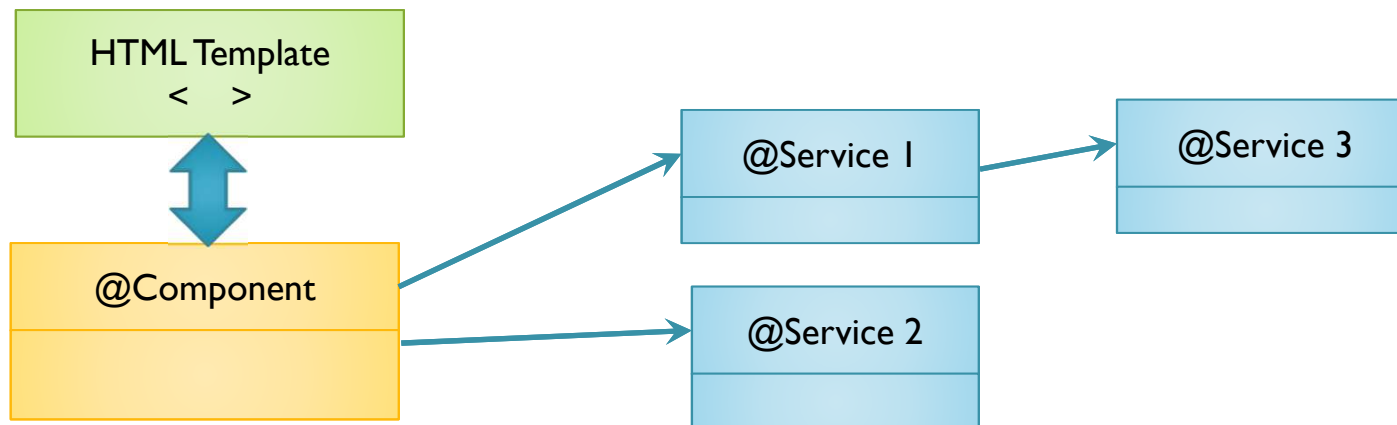
Liste des messages :

aa	18:42:08
bb	18:42:12

contacts works!

Services

- *Un service est une catégorie large qui englobe toute valeur, fonction ou fonctionnalité dont votre application a besoin.*
- *Un service est généralement une classe avec un but étroit et bien défini.*
- *Généralement, les composants se limite à l'affichage et à la gestion des événements utilisateurs dans la vue du composant. L'exécution des traitements en local ou en back end sont attribués aux services.*
- *Quand un événement survient dans la vue, le composant fait appel à des fonctions dans les services pour effectuer des traitements et fournir des résultats.*
- Généralement, c'est les service qui interagissent avec la partie back end de l'application en envoyant des requêtes HTTP.
- Généralement c'est les composants qui consomme les services, toutefois, un service peut consommer d'autres services.
- l'utilisation d'un service se fait via le principe de l'injection des dépendances.



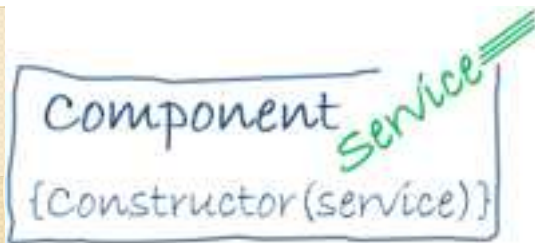
Exemple de service

exemple.service.ts

```
import { Injectable } from '@angular/core';
@Injectable()
export class ExempleService {
  constructor() { }
  saveData(data) {
    console.log('saving data at back end....');
  }
  getData() {
    console.log('gettig data from back end ...');
  }
}
```

exemple.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ExempleService } from "../exemple.service";
@Component({
  selector: 'app-exemple', templateUrl: './exemple.component.html',
  styleUrls: ['./exemple.component.css']
})
export class ExempleComponent {
  constructor(private exempleService: ExempleService) { }
  onSave(data) {
    this.exempleService.saveData(data);
  }
  onGetData() {
    return this.exempleService.getData();
  }
}
```



Injection des dépendances

- L'injection de dépendance est une façon de fournir une nouvelle instance d'une classe avec les dépendances entièrement formées dont elle a besoin.
- La plupart des dépendances sont des services.

```
import { Injectable } from
 '@angular/core';

@Injectable()
export class AboutService {
  info={
    nom:"Mohamed",
    email:"med@yousfsi.net",
    tel:"0661326837"
  };
  comments=[];
  constructor() { }
  getInfos() {
    return this.info;
  }
  addComment(c) {
    c.date=new Date();
    this.comments.push(c);
  }
  getAllComments() {
    return this.comments;
  }
}
```

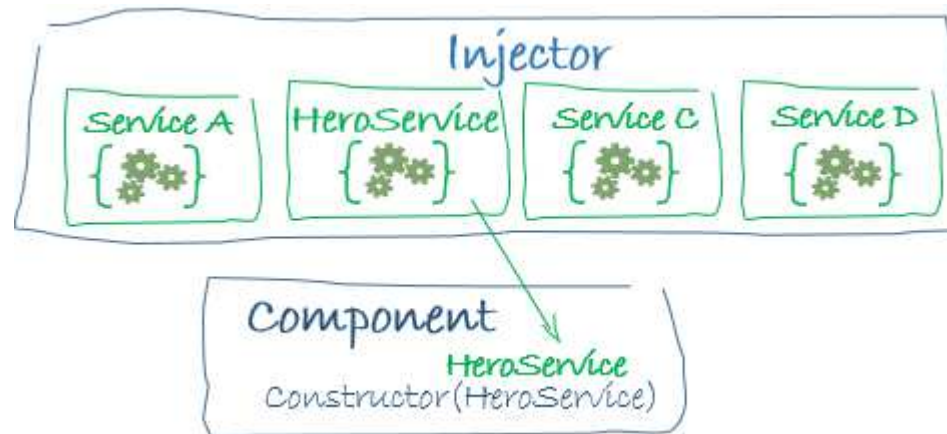
```
export class AboutComponent {
  constructor(private
    aboutService:AboutService) { }

  info=this.aboutService.getInfos();
  comments=this.aboutService.getAllComments();

  comment={id:0,message:'',date:null};
  newComment=false;
  addComment() {
    if(this.comment.message!='') {
      this.aboutService.addComment({
        message:this.comment.message});
      this.comments=this.aboutService.getAllComments();
      this.comment.message='';
    }
  }
}
```

Injection des dépendances

- Lorsque Angular crée un composant, il demande d'abord à un injecteur les services requis.
- Un injecteur maintient un conteneur d'instances de service qu'il a créé précédemment.
- Si une instance de service demandée n'est pas dans le conteneur, l'injecteur en fait une et l'ajoute au conteneur avant de renvoyer le service à Angular.
- Lorsque tous les services demandés ont été résolus et retournés, Angular peut appeler le constructeur du composant avec ces services comme arguments.
- Il s'agit d'une injection de dépendance.



Enregistrement des services

- Pour utiliser un service, il faut préalablement enregistrer un fournisseur de ce service avec l'injecteur.
- Un fournisseur de service est une fabrique qui permet de gérer l'instanciation des services.
- Vous pouvez enregistrer les fournisseurs en modules ou en composants.
- En général, ajoutez des fournisseurs au module racine afin que la même instance d'un service soit disponible partout.

app.module.ts

```
imports: [  
  BrowserModule, FormsModule  
],  
providers: [AboutService, ExempleService],  
bootstrap: [AppComponent]
```

Sinon, enregistrez-vous à un niveau de composant dans la propriété des fournisseurs des métadonnées `@Component`. Dans ce cas le service est instancié pour chaque nouvelle instance du composant.

```
@Component ({  
  selector: 'app-about',  
  templateUrl: './about.component.html',  
  styleUrls: ['./about.component.css'],  
  providers: [AboutService]  
})
```

Routage et Navigation

- Le routeur angulaire permet la navigation d'une vue à l'autre lorsque les utilisateurs exécutent des tâches d'application.
- Le routeur angulaire est un service facultatif qui présente une vue de composant particulière pour une URL donnée.
- Il ne fait pas partie du noyau angulaire.
- C'est dans son propre paquet de bibliothèque, [@angular/router](#).
- Importez ce dont vous avez besoin comme vous le feriez à partir de tout autre paquet Angular.

src/app /app.module.ts (import)

```
import { RouterModule, Routes } from '@angular/router';
```

Configuration des routes

```
import ...
import {RouterModule, Routes} from '@angular/router';

const appRoutes: Routes = [
  { path: 'about', component: AboutComponent },
  { path: 'contacts', component: ContactsComponent },
  { path: '',
    redirectTo: '/about',
    pathMatch: 'full'
  }
];

@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    ContactsComponent,
    ExempleComponent
  ],
  imports: [
    BrowserModule, FormsModule, RouterModule.forRoot(appRoutes)
  ],
  providers: [AboutService, ExempleService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Router outlet

- Etant donnée cette configuration,
 - Quand l'utilisateur tape : <http://localhost:4200/about> ,
 - le routeur cherche et charge le composant AboutComponent et l'affiche dans un élément **<router-outlet>** **</router-outlet>** .
 - Cet élément est sensé se trouver dans la vue du composant racine.

app.component.html

```
<div class="container spacer">
  <div class="container">
    <button routerLink="/about" class="btn btn-primary">About</button>
    <button routerLink="/contacts" class="btn btn-primary">Contacts</button>
  </div>
  <div class="container spacer">
    <router-outlet></router-outlet>
  </div>
</div>
```

← → ↻ ⓘ localhost:4200/about

About Contacts

About Component

- Nom : Mohamed
- Email : med@youssfi.net
- Tel : 0661326837

Message: Add comment

Liste des messages :

aa 21:12:50

bb 21:12:53

← → ↻ ⓘ localhost:4200/contacts

About Contacts

contacts works!

Routage et Navigation

- Il est également possible de naviguer entre les différentes routes en utilisant la méthode `navigate()` du service Router. Pour cela le service Router doit être injecté dans la classe composant.
- L'exemple suivant montre le code de la classe du composant AppComponent : `app.component.ts`

```
import { Component } from '@angular/core';
import { Router } from "@angular/router";

@Component ({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
  constructor(private router:Router) {}
  onAbout () {
    this.router.navigate(['about']);
  }
}
```

```
<button (click)="onAbout()" class="btn btn-primary">About</button>
```

Créer un module séparé pour la configuration des routes

app/app-routing.module.ts

```
import {AboutComponent} from "../about/about.component";
import {ContactsComponent} from "../contacts/contacts.component";
import {Routes, RouterModule} from "@angular/router";
import {NgModule} from "@angular/core";
const appRoutes: Routes = [
  { path: 'about', component: AboutComponent },
  { path: 'contacts', component: ContactsComponent },
  { path: '',
    redirectTo: '/about',
    pathMatch: 'full'
  }
];
@NgModule({
  imports: [RouterModule.forRoot(appRoutes)],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

app.module.ts

- Il est plus utile de créer un module séparé pour configurer les routes au de le faire dans le module principale.
- Il suffit de créer un fichier app/app-routing.module.ts
- Ensuite importer ce module dans le module principal

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

HTTP: Interaction avec la partie back end

- La plupart des applications Frontend communiquent avec les services backend via le protocole HTTP.
- Les navigateurs modernes prennent en charge deux API différentes pour effectuer des requêtes HTTP :
 - l'interface XMLHttpRequest
 - et l'API fetch ().
- Le HttpClient du module @angular/common/http offre une API HTTP client simplifiée pour les applications Angular qui repose sur l'interface XMLHttpRequest exposée par les navigateurs.
- Les avantages supplémentaires de HttpClient incluent des fonctionnalités de test, des objets de requête et de réponse typés, un mécanisme d'interception de requêtes et de réponses, des apis observables et une gestion simplifiée des erreurs.

HttpClientModule

- Avant de pouvoir utiliser **HttpClient**, vous devez importer le module **HttpClientModule** angulaire.
- La plupart des applications le font dans le module racine AppModule.

```
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  declarations: [
    AppComponent, LoginComponent, TasksComponent, NewTaskComponent,
    RegistrationComponent
  ],
  imports: [
    BrowserModule, RouterModule.forRoot(appRoutes), FormsModule, HttpClientModule
  ],
  providers: [AuthenticationService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Exemple de Service avec HttpClientModule

```
import {Injectable} from '@angular/core';  
import {HttpClient, HttpHeaders} from '@angular/common/http';
```

```
@Injectable()  
export class AuthenticationService{  
  private host:string="http://localhost:8080";
```

```
  constructor(private http:HttpClient){ }
```

```
  getTasks() {  
    return this.http.get(this.host+"/tasks");  
  }
```

```
  login(user) {  
    return this.http.post(this.host+"/login",user,{observe:'response'});  
  }
```

```
  getTasksV2() {  
    return this.http.get(this.host+"/tasks",  
      {headers:new HttpHeaders({'Authorization':'my-token'})});  
  }
```

```
  saveTask(task) {  
    return this.http.post(this.host+"/tasks",task,{headers:new  
      HttpHeaders({'authorization':'my-token'})});  
  }  
}
```

Exemple de composant

```
export class TasksComponent implements OnInit {  
  tasks;  
  constructor(public authService:AuthenticationService,private  
router:Router) { }  
  ngOnInit() {  
    this.authService.getTasks()  
      .subscribe(data=>{  
        this.tasks=data;  
      },err=>{  
        this.authService.logout();  
        this.router.navigateByUrl('/login')  
      })  
  }  
}
```

Exemple de composant

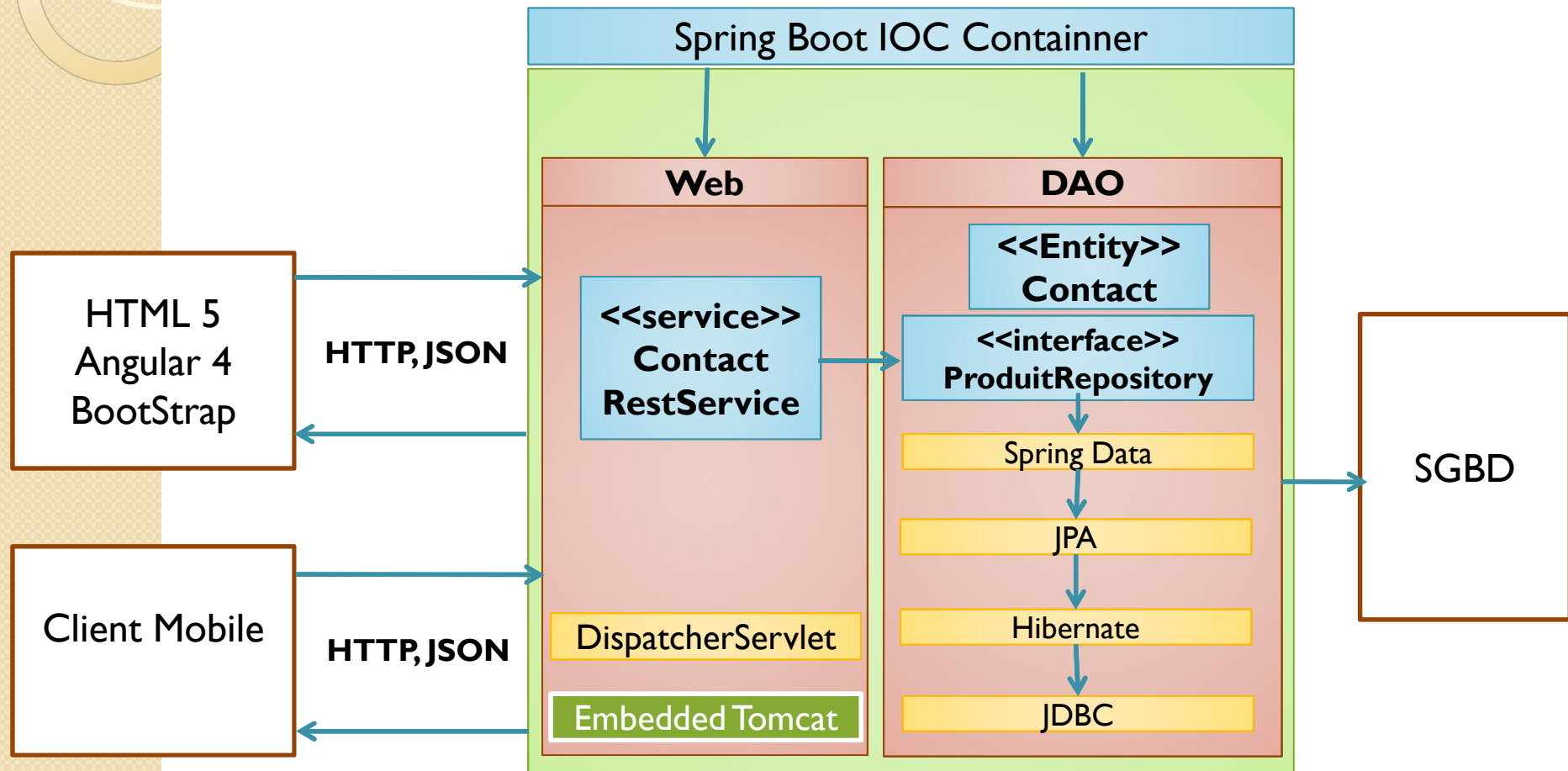
```
export class LoginComponent implements OnInit {  
  mode:number=0;  
  constructor(private authService:AuthenticationService,private  
router:Router) { }  
  
  ngOnInit() {  
  
    onLogin(user) {  
      this.authService.login(user)  
        .subscribe(resp=>{  
          let jwt=resp.headers.get('Authorization');  
          // console.log(resp.headers.get('Authorization'));  
          this.authService.saveToken(jwt);  
          this.router.navigateByUrl('/tasks');  
        },  
        err=>{  
          this.mode=1;  
        })  
    }  
  
  }  
}
```


Application

- Créer une application Web JEE qui permet de gérer des contacts (id, nom, prénom, date de naissance, email, tel, photo):
 - Saisir et ajouter des contacts
 - Chercher des contacts
 - Éditer et modifier des contacts
 - Supprimer des contacts
- L'application se compose de deux parties :
 - La partie BackEnd basée un Spring, Spring Data JPA et Hibernate. (API Restful)
 - La partie FrontEnd est basée sur Angular 4

Architecture

Micro Service



Application

← → ↻ ⓘ localhost:4200/contacts

About

Contacts

New Contact

Liste des contacts

ID	Nom	Prénom		
1	ibrahimi	Hassan	Detail	Delete
2	alaoui	mehdi	Detail	Delete
3	zahir	amal	Detail	Delete
4	ibrahimi	Hassan	Detail	Delete
5	alaoui	mehdi	Detail	Delete
6	zahir	amal	Detail	Delete

Application

← → ↻ ⓘ localhost:4200/detailContact/1

About

Contacts

New Contact

Détail Contact

ID: 1

Nom: ibrahimi

Prénom: ibrahimi

Date Naissance: 12/11/1970

Email: hassan@gmail.com

Tel: 632546587

Photo:



Application

← → ↻ ⓘ localhost:4200/newContact

About

Contacts

New Contact

Nouveau Contact

Nom:

aaaa

Prénom:

bbbb

Date naissance:

15/06/2017

Email:

med@gmail.com

Tel:

06523254|

Save

Application



Confirmation

Id: 17

Nom: aaaa

Prénom: bbbb

Date Naissance: 1497484800000

Email: med@gmail.com

Tel: 6523254

OK

Module de routage : app/app-routing.ts

```
import {AboutComponent} from "../about/about.component";
import {ContactsComponent} from "../contacts/contacts.component";
import {Routes, RouterModule} from "@angular/router";
import {NgModule} from "@angular/core";
import {DetailContactComponentComponent} from "../detail-contact-
component/detail-contact-component.component";
import {NewContactComponent} from "../new-contact/new-
contact.component";
const appRoutes: Routes = [
  { path: 'about', component: AboutComponent },
  { path: 'contacts', component: ContactsComponent },
  { path: 'detailContact/:id', component:
DetailContactComponentComponent },
  { path: 'newContact', component: NewContactComponent },
  { path: '',
    redirectTo: '/about',
    pathMatch: 'full'
  }
];
@NgModule({
  imports: [RouterModule.forRoot(appRoutes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```


Service ContactService: `app/service/contact.service.ts`

```
import {Injectable} from "@angular/core";
import {Http, Response} from "@angular/http";
import "rxjs/add/operator/map";
import "rxjs/add/operator/catch"
import {Observable} from "rxjs";
@Injectable()
export class ContactService {
  constructor(private http:Http) {}

  getAllContacts():Observable<any>{
    return this.http.get("http://localhost:8080/contacts")
      .map(resp=>resp.json());
  }

  getContact(id:number):Observable<any>{
    return this.http.get("http://localhost:8080/contacts/"+id)
      .map(resp=>resp.json());
  }

  saveContact(contact) {
    return this.http.post("http://localhost:8080/contacts",contact)
      .map(resp=>resp.json());
  }

  deleteContact(id:number) {
    return this.http.delete("http://localhost:8080/contacts/"+id)
      .map(resp=>resp);
  }
}
```

Composant ContactsComponent: **contacts.component.ts**

```
import { Component, OnInit } from '@angular/core';
import { ContactService } from '../services/contacts.service';
import { Router } from '@angular/router';
@Component({
  selector: '[app-contacts]',
  templateUrl: './contacts.component.html',
  styleUrls: ['./contacts.component.css']
})
export class ContactsComponent implements OnInit {
  contacts=[];
  constructor(
    private contactService:ContactService,
    private router:Router
  ) { }
  ngOnInit() {
    this.contactService.getAllContacts()
      .subscribe(data=>this.contacts=data);
  }
  detailContact(id:number){
    this.router.navigate(["/detailContact",id]);
  }
  deleteContact(id:number){
    this.contactService.deleteContact(id)
      .subscribe(data=>{this.ngOnInit();});
  }
}
```

Composant ContactsComponent: **contacts.component.html**

```
<div class="panel panel-primary">
  <div class="panel-heading">Liste des contacts</div>
  <div class="panel-body">
    <table class="table table-striped">
      <tr>
        <th>ID</th><th>Nom</th><th>Prénom</th><th></th><th></th><th></th>
      </tr>
      <tr *ngFor="let c of contacts">
        <td>{{c.id}}</td>
        <td>{{c.nom}}</td>
        <td>{{c.prenom}}</td>
        <!--
        <td><a routerLink="/detailContact/{{c.id}}">Detail</a></td>
        -->
        <td><a class="clickable" (click)="detailContact(c.id)">Detail</a></td>
        <td><a class="clickable" (click)="deleteContact(c.id)">Delete</a></td>
      </tr>
    </table>
  </div>
</div>
```

Composant DetailContactComponent: detail-contact.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from "@angular/router";
import { ContactService } from "../services/contacts.service";

@Component ({
  selector: 'app-detail-contact-component',
  templateUrl: './detail-contact-component.component.html',
  styleUrls: ['./detail-contact-component.component.css']
})
export class DetailContactComponentComponent implements OnInit {

  constructor(
    private route:ActivatedRoute,
    private router:Router,
    private contactService:ContactService) { }
  contact={};
  id:number;
  ngOnInit () {
    this.id+=this.route.snapshot.params['id'];
    this.contactService.getContact(this.id)
      .subscribe (data=>this.contact=data);
  }
}
```

Composant DetailContactComponent: detail-contact.component.html

```
<div class="panel panel-primary">
  <div class="panel-heading">Détail Contact</div>
  <div class="panel-body">
    <div class="form-group">
      <label>ID:</label>
      <label>{{contact.id}}</label>
    </div>
    <div class="form-group">
      <label>Nom:</label>
      <label>{{contact.nom}}</label>
    </div>
    <div class="form-group">
      <label>Prénom:</label>
      <label>{{contact.nom}}</label>
    </div>
    <div class="form-group">
      <label>Date Naissance:</label>
      <label>{{contact.dateNaissance | date: 'dd/MM/yyyy'}}</label>
    </div>
    <div class="form-group">
      <label>Email:</label>
      <label>{{contact.email}}</label>
    </div>
    <div class="form-group">
      <label>Tel:</label>
      <label>{{contact.tel}}</label>
    </div>
    <div class="form-group">
      <label>Photo:</label>
      
    </div>
  </div>
</div>
```

Composant NewContactComponent: new-contact.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ContactService } from '../services/contacts.service';

@Component ({
  selector: 'app-new-contact',
  templateUrl: './new-contact.component.html',
  styleUrls: ['./new-contact.component.css']
})
export class NewContactComponent implements OnInit {
  contact={nom:"",prenom:"",email:"",dateNaissance:null,tel:""};
  mode="new";
  constructor(private contactService:ContactService) { }

  ngOnInit () {

  }

  saveContact () {
    this.contactService.saveContact(this.contact)
      .subscribe(data=>{this.mode='confirm';this.contact=data;});
  }

  newContact () {
    this.contact={nom:"",prenom:"",email:"",dateNaissance:null,tel:""};
    this.mode='new';
  }
}
```

Composant NewContactComponent: new-contact.component.html

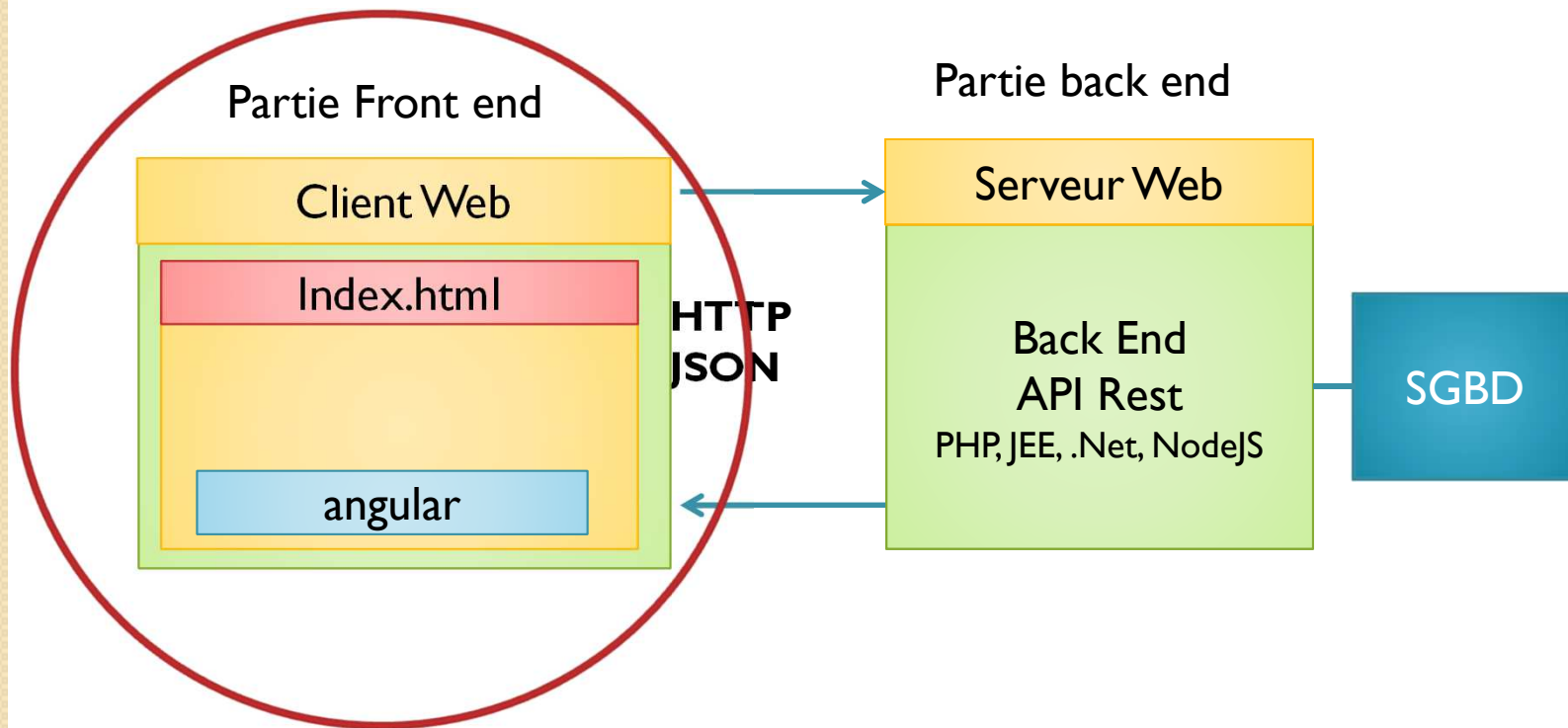
```
• <div class="panel panel-primary" *ngIf="mode=='new'">
  <div class="panel-heading">Nouveau Contact</div>
  <div class="panel-body">
    <div class="form-group">
      <label class="control-label">Nom:</label>
      <input type="text" [(ngModel)]="contact.nom" class="form-control">
    </div>
    <div class="form-group">
      <label class="control-label">Prénom:</label>
      <input type="text" [(ngModel)]="contact.prenom" class="form-control">
    </div>
    <div class="form-group">
      <label class="control-label">Date naissance:</label>
      <input type="date" [(ngModel)]="contact.dateNaissance" class="form-
control">
    </div>
    <div class="form-group">
      <label class="control-label">Email:</label>
      <input type="email" [(ngModel)]="contact.email" class="form-control">
    </div>
    <div class="form-group">
      <label class="control-label">Tel:</label>
      <input type="tel" [(ngModel)]="contact.tel" class="form-control">
    </div>
    <button (click)="saveContact()" class="btn btn-primary">Save</button>
  </div>
</div>
```


Composant NewContactComponent: new-contact.component.html (suite)

- ```
<div class="panel panel-primary" *ngIf="mode=='confirm'">
 <div class="panel-heading">Confirmation</div>
 <div class="panel-body">
 <div class="form-group">
 <label class="control-label">Id:</label>
 <label class="control-label">{{contact.id}}</label>
 </div>
 <div class="form-group">
 <label class="control-label">Nom:</label>
 <label class="control-label">{{contact.nom}}</label>
 </div>
 <div class="form-group">
 <label class="control-label">Prénom:</label>
 <label class="control-label">{{contact.prenom}}</label>
 </div>
 <div class="form-group">
 <label class="control-label">Date Naissance:</label>
 <label class="control-label">{{contact.dateNaissance}}</label>
 </div>
 <div class="form-group">
 <label class="control-label">Email:</label>
 <label class="control-label">{{contact.email}}</label>
 </div>
 <div class="form-group">
 <label class="control-label">Tel:</label>
 <label class="control-label">{{contact.tel}}</label>
 </div>
 <button (click)="newContact()" class="btn btn-primary">OK</button>
 </div>
</div>
```

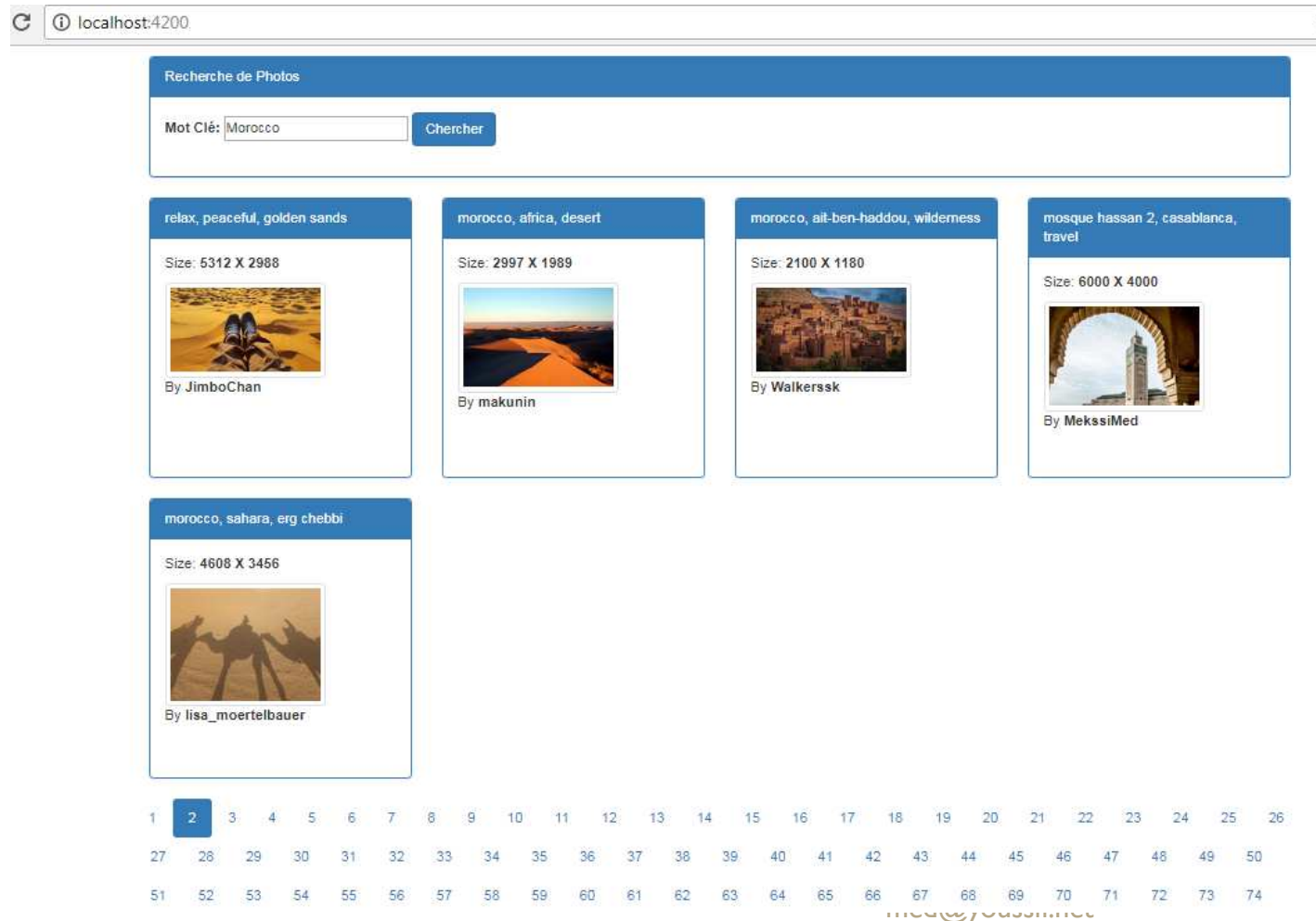
# Interaction avec La partie Backend

- Angular dispose d'un service http qui permet d'envoyer des requêtes http Ajax vers les serveur Web
- La partie serveur exécute des traitement et renvoi à la partie Front des données généralement au format JSON.
- Ces données sont généralement affichées en utilisant des composants Web



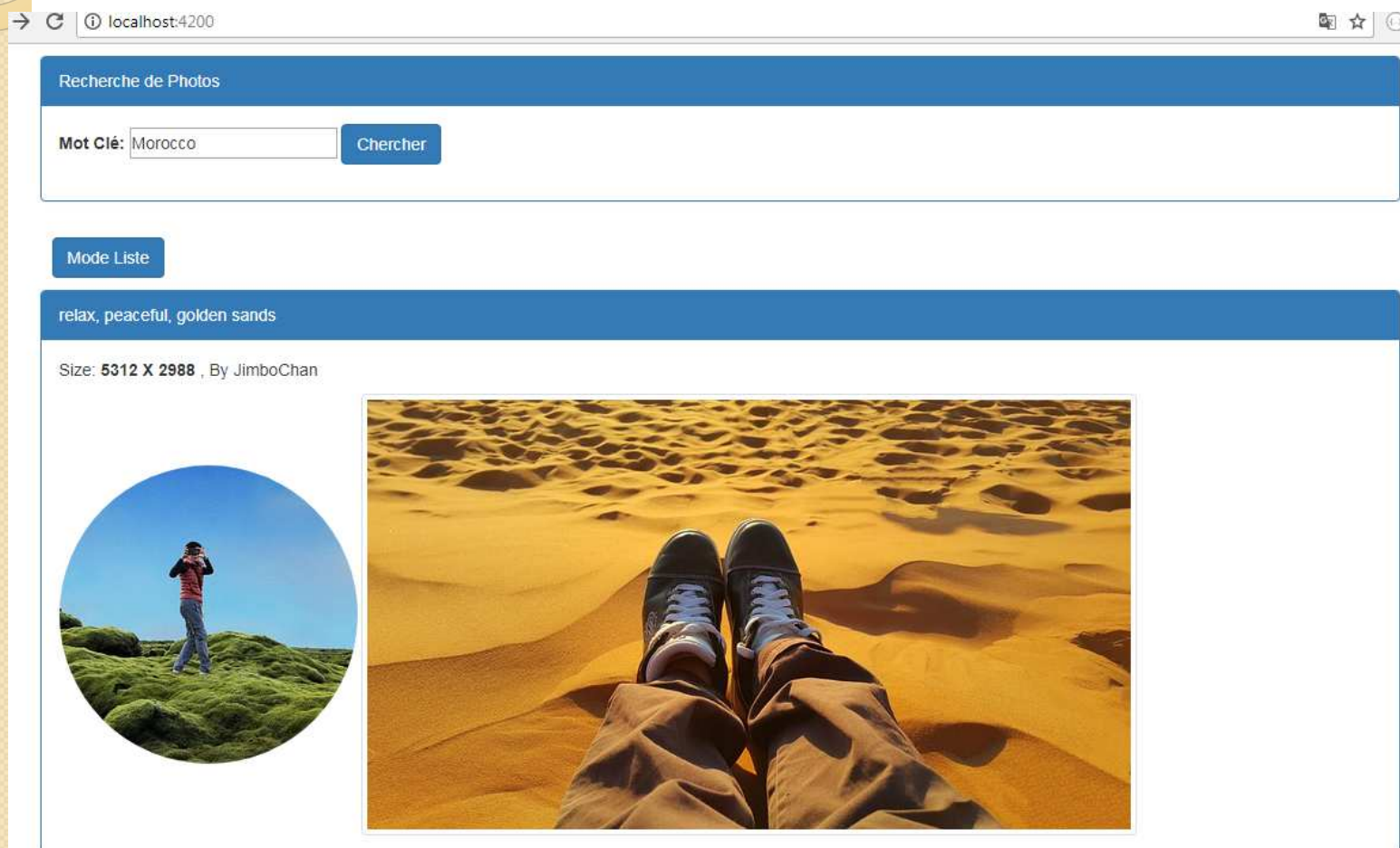
# Application

- Application permettant de chercher des photos en interagissant avec une API Rest
- [https://pixabay.com/api/?key=5832566-81dc7429a63c86e3b707d0429&q=casablanca&per\\_page=10&page=1](https://pixabay.com/api/?key=5832566-81dc7429a63c86e3b707d0429&q=casablanca&per_page=10&page=1)



# Application

- Application permettant de chercher des photos en interagissant avec une API Rest :
  - [https://pixabay.com/api/?key=5832566-81dc7429a63c86e3b707d0429&q=casablanca&per\\_page=10&page=1](https://pixabay.com/api/?key=5832566-81dc7429a63c86e3b707d0429&q=casablanca&per_page=10&page=1)



# API REST

- Voir <https://pixabay.com/api/docs/>

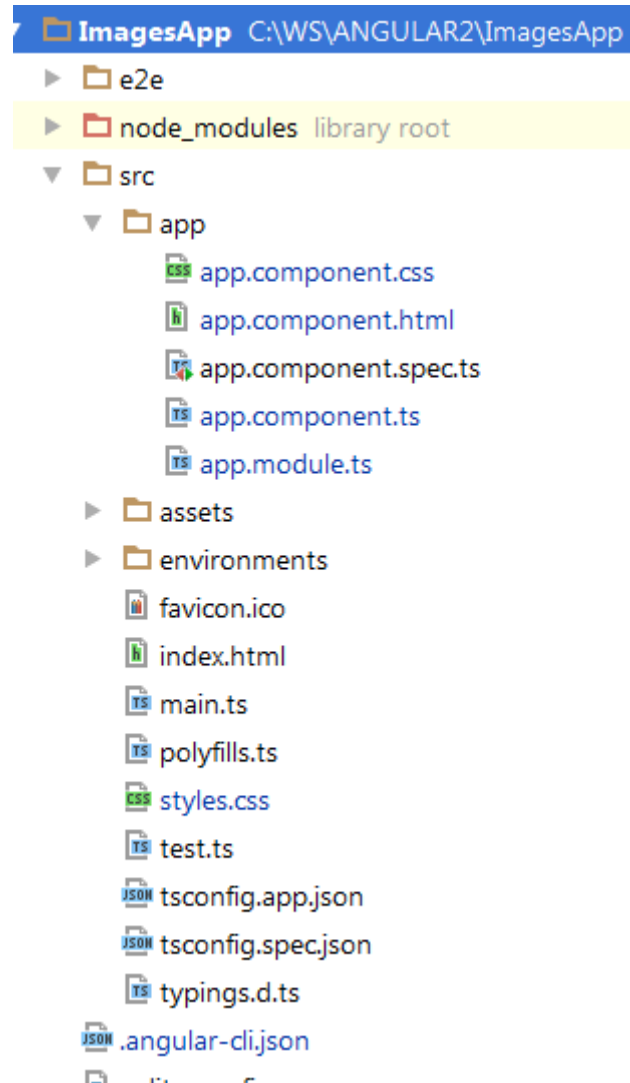


The screenshot shows a web browser window with the address bar displaying the URL: [https://pixabay.com/api/?key=5832566-81dc7429a63c86e3b707d0429&q=casablanca&per\\_page=10&page=1](https://pixabay.com/api/?key=5832566-81dc7429a63c86e3b707d0429&q=casablanca&per_page=10&page=1). The browser's developer tools are open, showing the JSON response from the API. The response is a JSON object with a 'totalHits' property set to 62 and a 'hits' array containing two image objects. The first object in the array represents a photo of a mosque in Casablanca, and the second object represents a building in Morocco.

```
{
 "totalHits": 62,
 "hits": [
 {
 "previewHeight": 99,
 "likes": 1,
 "favorites": 1,
 "tags": "mosque hassan 2, casablanca, travel",
 "webformatHeight": 426,
 "views": 149,
 "webformatWidth": 640,
 "previewWidth": 150,
 "comments": 0,
 "downloads": 84,
 "pageURL": "https://pixabay.com/en/mosque-hassan-2-casablanca-travel-2458314/",
 "previewURL": "https://cdn.pixabay.com/photo/2017/06/30/13/51/mosque-hassan-2-2458314_150.jpg",
 "webformatURL": "https://pixabay.com/get/eb31b4072bf5053ed95c4518b7494e95e172e4d304b0144195f2c578a2ebbc_640.jpg",
 "imageWidth": 6000,
 "user_id": 2927579,
 "user": "MekssiMed",
 "type": "photo",
 "id": 2458314,
 "userImageURL": "https://cdn.pixabay.com/user/2017/04/24/10-56-55-348_250x250.jpg",
 "imageHeight": 4000
 },
 {
 "previewHeight": 150,
 "likes": 3,
 "favorites": 4,
 "tags": "morocco, mosque, building",
 "webformatHeight": 640,
 "views": 469,
 "webformatWidth": 427,
 "previewWidth": 100,
 "comments": 0,
 "downloads": 285,

```

# Structure du projet





# app.module.ts

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

import { AppComponent } from './app.component';
import {FormsModule} from "@angular/forms";
import {HttpModule} from "@angular/http";

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



# app.component.ts

```
import { Component } from '@angular/core';
import { Http } from '@angular/http';
import "rxjs/add/operator/map"
@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 motCle:string=""; images:{hits:null}; pageSize:number=5; currentPage:number=1;
 totalPages:number; pages:Array<number>=[]; mode='LIST'; currentImage=null;

 getImages() {
 this.http.get("https://pixabay.com/api/?key=5832566-81dc7429a63c86e3b707d0429&q="+this.motCle+" &per_page="+this.pageSize+" &page="+this.currentPage)
 .map(resp=>resp.json())
 .subscribe(data=>{
 this.images=data;
 this.totalPages=this.images['totalHits'] / this.pageSize;
 if(this.images['totalHits'] % this.pageSize !=0)
 this.totalPages+=1;
 this.pages=new Array(this.totalPages);
 });
 }
 gotoPage(i:number) { this.currentPage=i; this.getImages(); }
 detailImage(im) { this.mode='DETAIL'; this.currentImage=im; }

 constructor(private http:Http) {
 }
}
```

# app.component.html

```
<p></p>
<div class="container">
 <div class="panel panel-primary">
 <div class="panel-heading">Recherche de Photos</div>
 <div class="panel-body">
 <div class="form-group">
 <label>Mot Clé:</label>
 <input type="text" [(ngModel)]="motCle">
 <button class="btn btn-primary"
(click)="getImages()">Chercher</button>
 </div>
 </div>
 </div>
</div>
```

# app.component.html

```
<div *ngIf="mode=='LIST'">
 <div class="row">
 <div *ngFor="let im of images.hits" class="col-md-3 col-xs-12">
 <div class="panel panel-primary hauteur">
 <div class="panel-heading">{{im.tags}}</div>
 <div class="panel-body">
 <p>Size: {{im.imageWidth}} X
{{im.imageHeight}}</p>

 <p>By {{im.user}}</p>
 </div>
 </div>
 </div>
 </div>
 <div class="row">
 <ul class="nav nav-pills">
 <li [ngClass]="{'active':currentPage==(i+1)}" *ngFor="let p of
pages; let i=index" class="clickable">
 <a (click)="gotoPage(i+1)">{{i+1}}

 </div>
</div>
</div>
```

# app.component.html

```
<div *ngIf="mode=='DETAIL'" class="container">
 <div class="container padding">
 <button class="btn btn-primary" (click)="mode='LIST'">Mode
 List</button>
 </div>
 <div class="panel panel-primary">
 <div class="panel-heading">{{currentImage.tags}}</div>
 <div class="panel-body">
 <p>
 Size: {{currentImage.imageWidth}} X
 {{currentImage.imageHeight}}
 , By {{currentImage.user}}
 </p>
 <div>

 </div>
 </div>
 </div>
</div>
```

## app.component.css

```
.padding {
 padding: 5px;
 margin: 5px;

}

.border {
 border: 1px dotted gray;
}

.hauteur {
 height: 280px;
}
```

# Développement Mobile

## IONIC Framework



Mohamed Youssefi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

Email : [med@youssefi.net](mailto:med@youssefi.net)

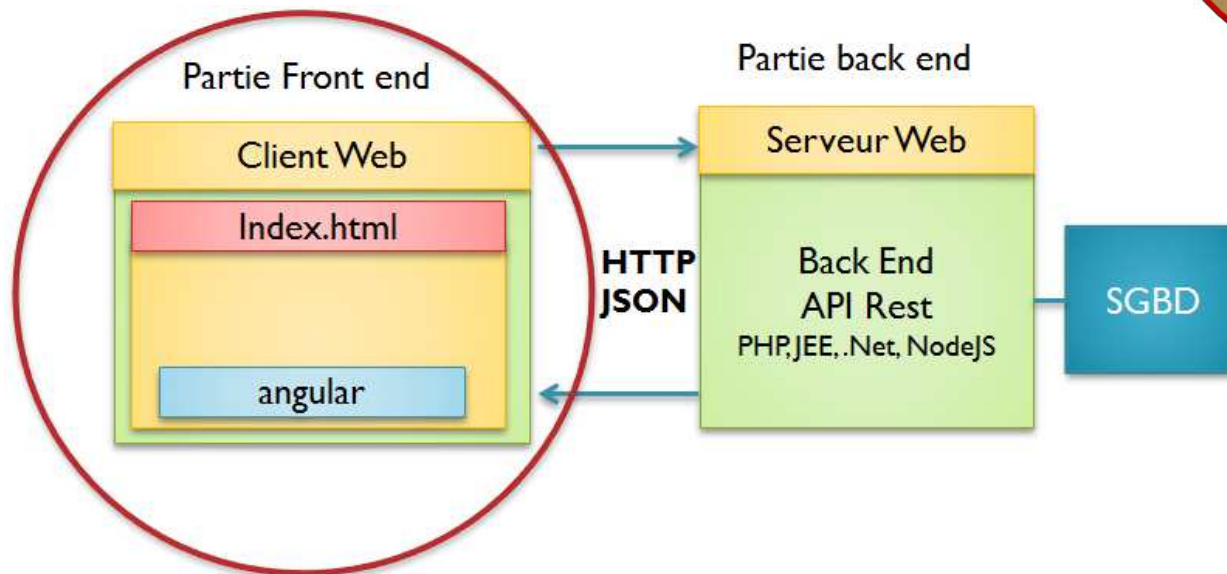
Supports de cours : <http://fr.slideshare.net/mohamedyoussefi9>

Chaîne vidéo : <http://youtube.com/mohamedYoussefi>

Recherche : [http://www.researchgate.net/profile/Youssefi\\_Mohamed/publications](http://www.researchgate.net/profile/Youssefi_Mohamed/publications)

# Développement Web

- Back End : Spring Boot
- Front End : Angular 4



Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

Email : [med@youssfi.net](mailto:med@youssfi.net)

Supports de cours : <http://fr.slideshare.net/mohamedyoussfi9>

Chaîne vidéo : <http://youtube.com/mohamedYoussfi>

Recherche : [http://www.researchgate.net/profile/Youssfi\\_Mohamed/publications](http://www.researchgate.net/profile/Youssfi_Mohamed/publications)