Formation Developpeur Intégrateur Web

Jaafar Elalamy

Ingénieur en Informatique & Entrepreneur

Fondateur & CEO @ Street4fit, Bonapart

UTC Compiègne - HEC Paris - UC Berkeley

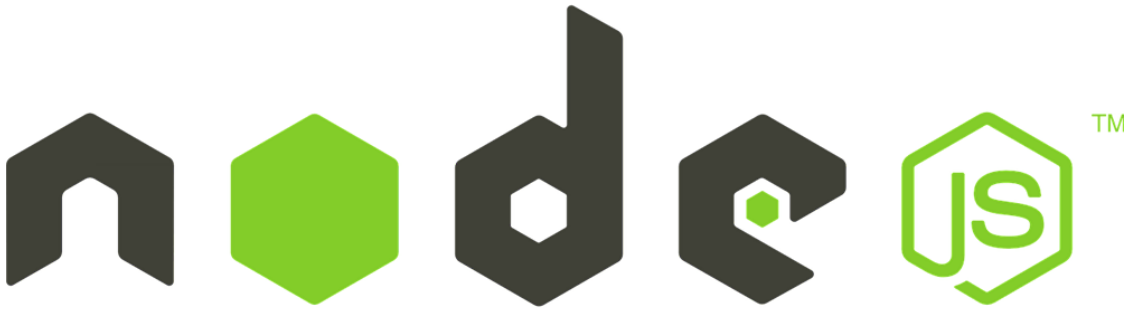
Formation 3 jours

Objectif : Réaliser une application web à l'aide du Framework Angular

- Introduction à Node et NPM
- Présentation de Angular
- Le système composants/modules
- TypeScript
- Data Binding et Two Way
- Les directives ngfor et ngif
- Gestion d'évènements avec (click)
- Ajout propriétés avec @input
- Discussion autour des services
- Présentation d'un stockage de données
- Gestion du routing
- Hébergement sur Heroku

Création d'une application web

Introduction à Node & NPM



Qu’est ce que Node ?

C’est environnement **open-source**, **multi-plate-forme** d'exécution pour le développement **côté serveur** et les **applications de réseau**

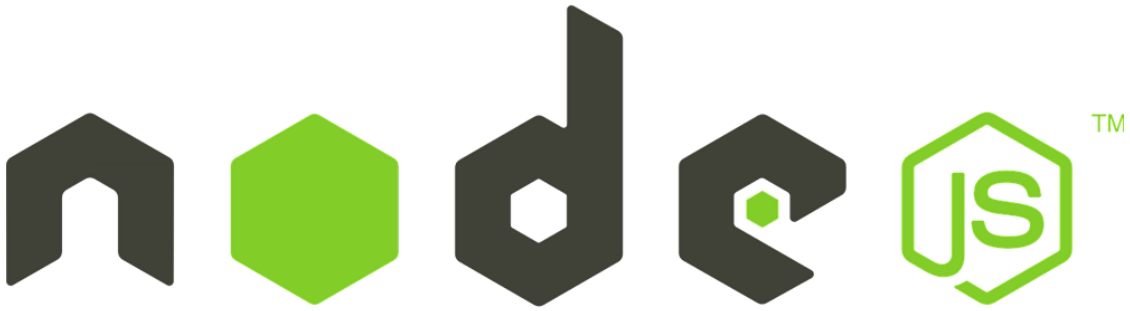
Les Particularités

Asynchrone, scalable, déclenchement d’événements, beaucoup de modules disponibles, traitement côté serveur

NPM : Node Packaged Modules

http (serveur et client web)
net (serveur et client TCP)
cradle (base de données en cache)
xml2js, crud-file-server
Nodepress (MVC pour dev de blogs)
Session (gestionnaire de sessions)
Librairies de gestion BDD (postgre, mysql, sqlite…)
Everyauth (connexions à plusieurs API de type Facebook, Github, Twitter, Instagram …)
Chatio : simple chat

```
$ npm install everyauth
```

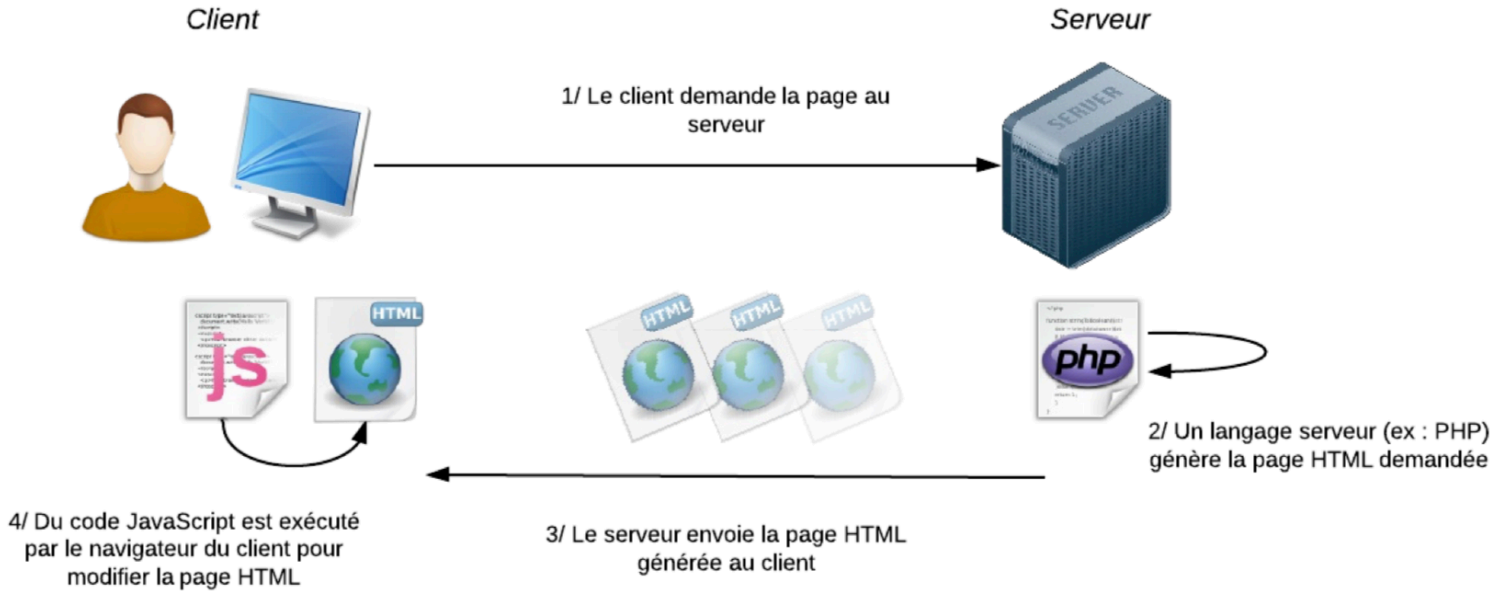


Les performances de Node

- Avec une simple utilisation, les performances sont les mêmes qu'Apache/PHP
- En mode multi-serveurs, les performances sont supérieures
- Evolution avec Socket I/O très efficace, grâce à la scalability et le fonctionnement asynchrone
- NodeJS casse les barrières de langage entre le client et le serveur : Node.js permet d'utiliser le langage JS sur le serveur - en dehors du navigateur !

Un langage basé "évènement"

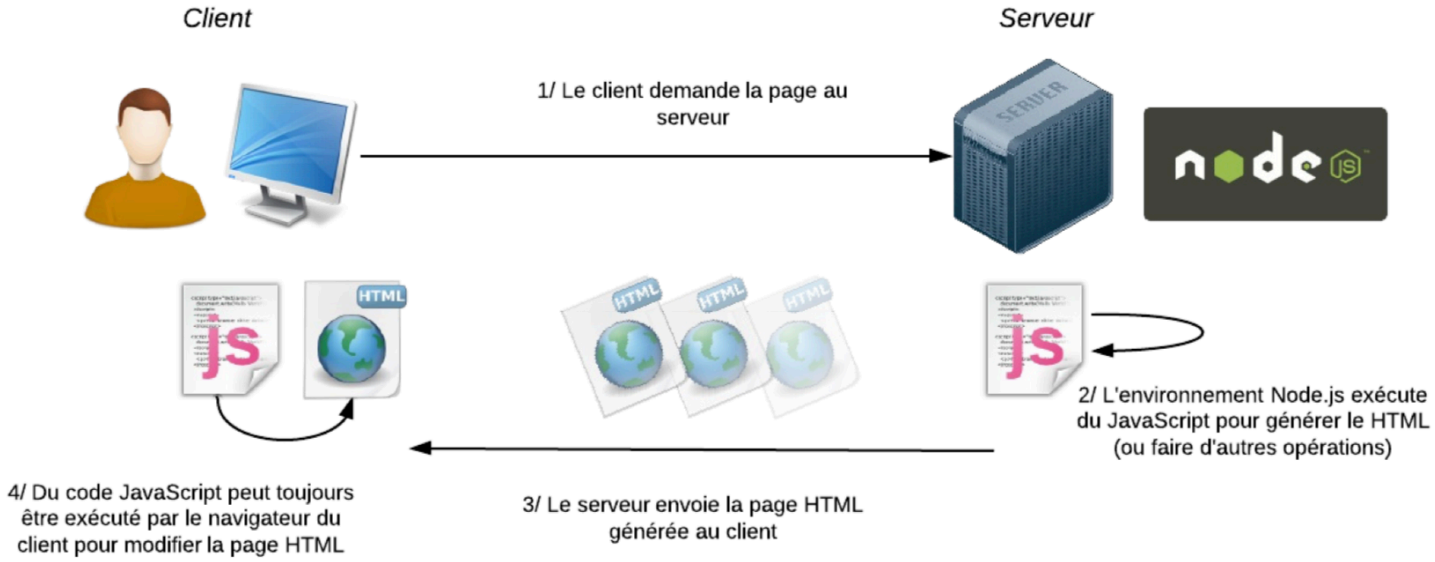
- JS est un langage rapide et puissant
- Node.js est adapté pour les uploads rapides
- Node.js est adapté pour les serveurs de chat
- Idéal pour les applications aux nombreuses requêtes
- Efficace pour les applications Temps Réel
- Ce n'est pas un Framework mais un environnement
- Express est un Framework basé Node
- Rapidité : moteur V8 Google + système non bloquant



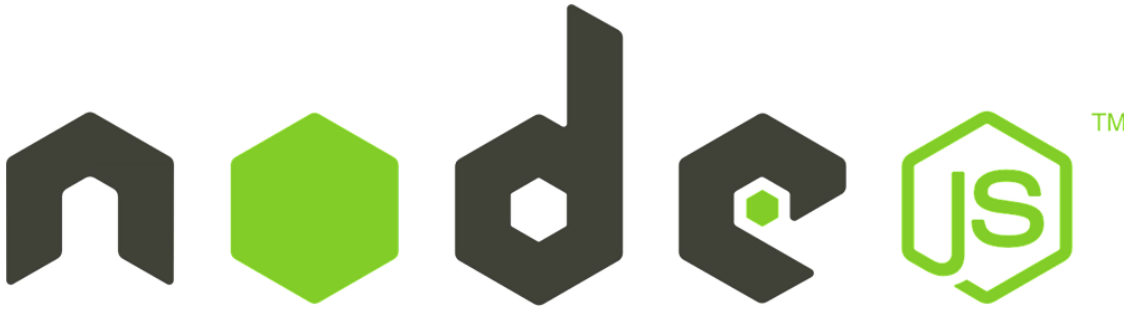
Le schéma classique : PHP sur le serveur, JavaScript chez le client

Cas classique

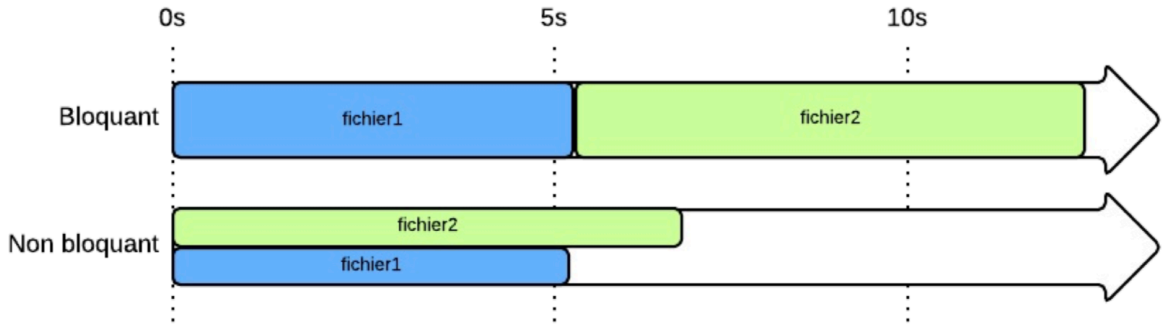
Avec Node.js



Avec Node.js, on peut aussi utiliser du JavaScript sur le serveur !



Système non bloquant




```
var callback = function (error, response, body) {
    console.log("Fichier téléchargé !");
});

request('http://www.site.com/fichier.zip', callback);
request('http://www.site.com/autrefichier.zip', callback);
```

PayPal

"The Node.js app was built almost **twice as fast** with **fewer people**, in **33% fewer lines of code** and **40% fewer files** (in comparison with previous Java-based application)"

"When compared with the previous Ruby on Rails based version, the new mobile app is **up to 20 times faster and uses only a fraction of resources – servers were cut from 30 to 3.** The development was fast"

The team decided to use Node.js to achieve lightweight, modular and fast application. As a result, the startup time of their new app has been reduced by 70%.

YAHOO!

"You can make it **scale, and it's very performant**, and every property that we've moved over to the Node.js stack has seen an increase in performance"

According to Uber, this technology has three core strengths: **processes lots of information quickly**; programs can be inspected and **errors can be addressed on the fly** – without requiring a restart, so developers can publish and deploy new code constantly; active open source community continuously optimizes the technology, so it **gets better all the time**, practically on its own.

Installation de NODE

LINUX / UNIX

Code : Console

```
sudo apt-get install python-software-properties python g++ make
sudo add-apt-repository ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install nodejs
```

Verification : node -v

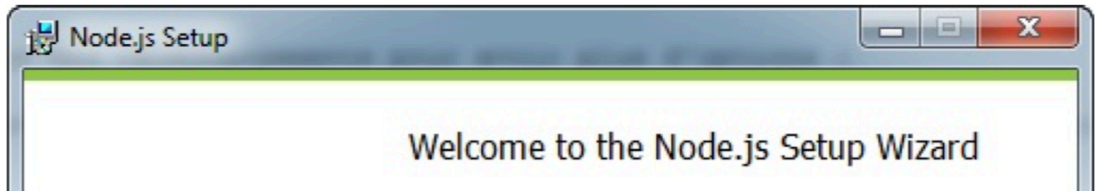
```
test.js : console.log('Bienvenue dans Node.js !');
```

node test.js

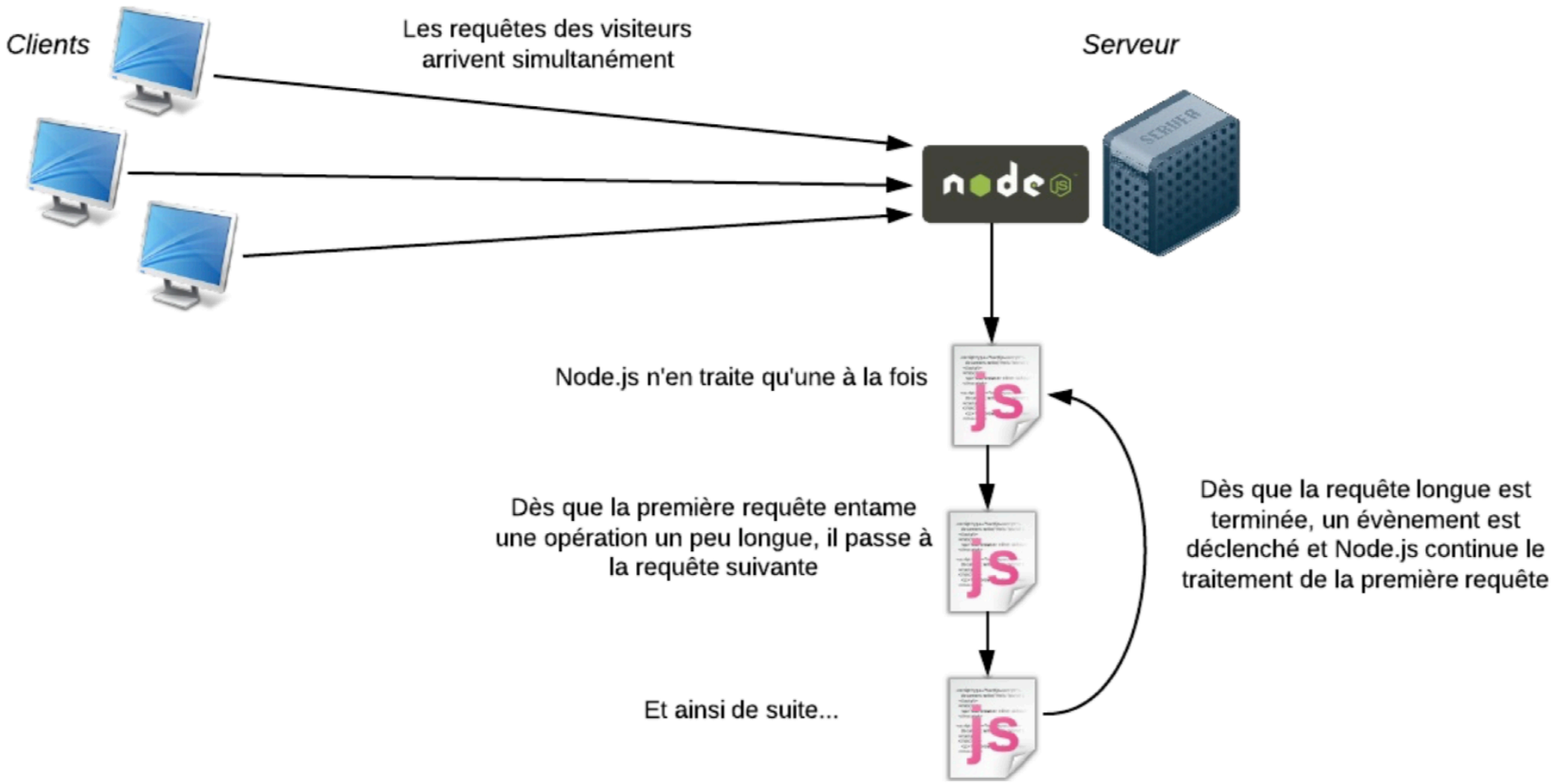
WINDOWS : aller sur nodejs.org

Vous pouvez télécharger soit le .msi, soit le .exe (le résultat sera le même). Prenez la version 64 bits si vous avez un Windows 64 bits (cas de la plupart des PC récents).
Dans le doute, prenez la version 32 bits.

Lancez ensuite l'installeur :



NODE est mono thread mais orienté évènements



Node.js est monothread, mais il est souple grâce aux évènements

Construction du serveur HTTP (hypertext transfer Protocol) : communication client-serveur

- Code minimal a placé dans un fichier serveur.js - quelque soit la page demandé il envoie “salut tout le monde”

```
var http = require('http');

var server = http.createServer(function(req, res) {
  res.writeHead(200);
  res.end('Salut tout le monde !');
});
server.listen(8080);
```

//Appel à la bibliothèque issue de NPM

//On lance le serveur web

//Fonction a exécuté quand le user se connecte.
//C’est une Fonction callback

//req : requête user - page, params, champs forms
// res : resultat - souvent code HTML à envoyer.

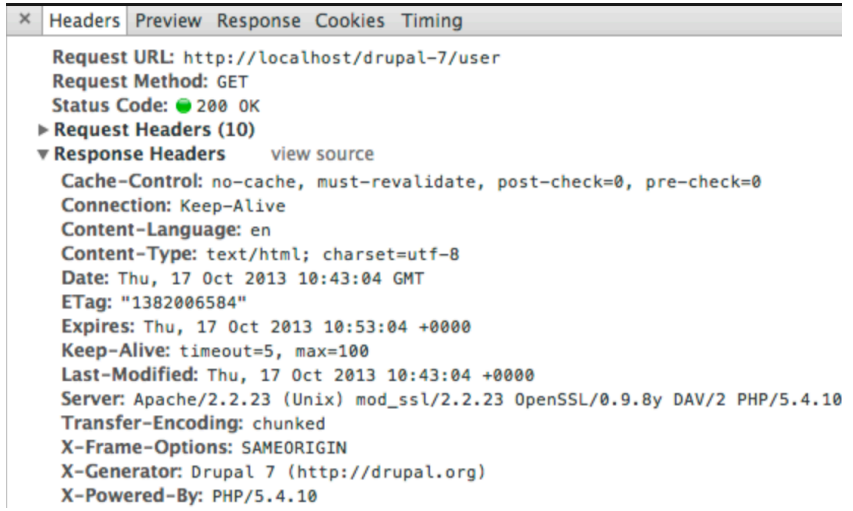
On lance **node serveur.js** et on va sur **localhost:8080** puis tester **localhost:8080/mapage**

- On peut envoyer du contenu HTML en réponse !

```
var http = require('http');

var server = http.createServer(function(req, res) {
    res.writeHead(200, {"Content-Type": "text/html"});
    res.end('<p>Voici un paragraphe <strong>HTML</strong> !</p>');
});
server.listen(8080);
```

Les systèmes de Templates permettent de séparer HTML du code JS
On vera ça plus tard

HEADER REQUETE HTTP



Quelle page est recherchée par le visiteur et quels paramètres ?

Il suffit de parser la requête pour savoir quelle page est demandée

```
var http = require('http');
var url = require('url');

var server = http.createServer(function(req, res) {
    var page = url.parse(req.url).pathname;
    console.log(page);
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.write('Bien le bonjour');
    res.end();
});
server.listen(8080);
```

```
[Jaafars-MacBook-Pro:
/mapage1
/favicon.ico
```

Introduire des conditions d'affichage en fonction de la page appelée

```
var http = require('http');
var url = require('url');

var server = http.createServer(function(req, res) {
    var page = url.parse(req.url).pathname;
    console.log(page);
    res.writeHead(200, {"Content-Type": "text/plain"});
    if (page == '/') {
        res.write('Vous êtes à l\'accueil, que puis-je pour vous ?
');
    }
    else if (page == '/sous-sol') {
        res.write('Vous êtes dans la cave à vins, ces bouteilles
sont à moi !');
    }
    else if (page == '/etage/1/chambre') {
        res.write('Hé ho, c\'est privé ici !');
    }
    res.end();
});
server.listen(8080);
```

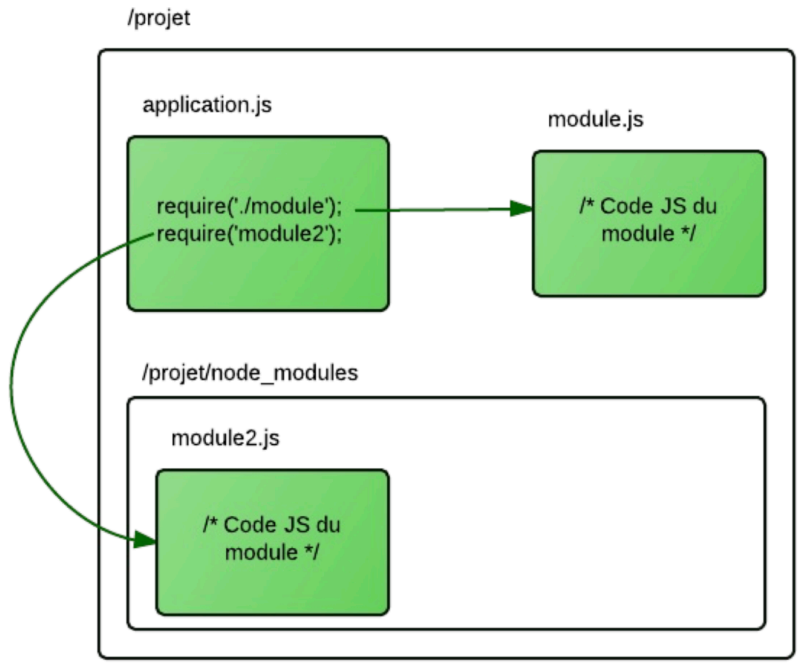
Récupérer les params : /mapage?nom=bob

==> Parser, récupérer params, découper la chaine grâce à querystring

```
var http = require('http');
var url = require('url');
var querystring = require('querystring');

var server = http.createServer(function(req, res) {
    var params = querystring.parse(url.parse(req.url).query);
    res.writeHead(200, {"Content-Type": "text/plain"});
    if ('prenom' in params && 'nom' in params) {
        res.write('Vous vous appelez ' + params['prenom'] + ' ' +
params['nom']);
    }
    else {
        res.write('Vous devez bien avoir un prénom et un nom, non ?
');
    }
    res.end();
});
server.listen(8080);
```

NPM : Node Packaged Modules - npm install nomdumodule

Un moyen formidable d'utiliser des modules développés par la communauté ! Principe D.R.Y + Gestion automatique des dépendances. Voir nodetoolbox.com modules par thématique ! Installation locale dans le sous-dossier node_modules. Ajouter -g pour installation globale/ console



Créer son propre module

Code : JavaScript

```
var direBonjour = function() {
    console.log('Bonjour !');
}

var direByeBye = function() {
    console.log('Bye bye !');
}

exports.direBonjour = direBonjour;
exports.direByeBye = direByeBye;
```

Maintenant, dans le fichier principal de votre application (ex : app.js), vous pouvez faire appel à ces fonctions issues du module !

Code : JavaScript

```
var monmodule = require('./monmodule');

monmodule.direBonjour();
monmodule.direByeBye();
```

GOOD TO KNOW

- npm update
- Si plusieurs modules, installer un à un est fastidieux : package.json - npm install pour les dépendances
- Le “tilde” autorise la mise à jour !
- Publier vos propres modules “npm adduser” + package.json + README.md + “npm publish”

```
{
    "name": "mon-app",
    "version": "0.1.0",
    "dependencies": {
        "markdown": "~0.4"
    }
}
```

Node.js est un langage bas niveau, donc il faut coder beaucoup !
Comment aller plus vite ? Utiliser des Frameworks (super bibliothèques) tels que Express.js
Express.js fournit les outils de base pour créer son application NODE plus vite ! **npm install express**

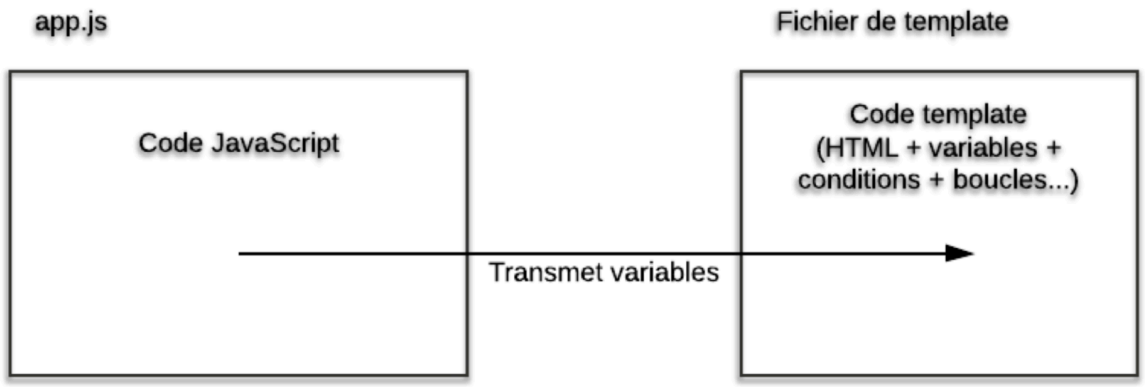
Exemple de simplification : les routes ! Au lieu des IF on a :

```
app.get('/', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.end('Vous êtes à l\'accueil, que puis-je pour vous ?');
});

app.get('/sous-sol', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.end('Vous êtes dans la cave à vins, ces bouteilles sont à
moi !');
});

app.get('/etage/1/chambre', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hé ho, c\'est privé ici !');
});
```

```
// ... Tout le code de gestion des routes (app.get) se trouve au-
dessus

app.use(function(req, res, next){
    res.setHeader('Content-Type', 'text/plain');
    res.send(404, 'Page introuvable !');
});

app.listen(8080);
```

Les Templates

Exemples de Templates : Twig, Smarty, Haml, Jade, EJS …



Les templates avec Node.js

Plus besoin d'écrire du HTML en plein milieu du code JS

```
app.get('/etage/:etagenum/chambre', function(req, res) {
    res.render('chambre.ejs', {etage: req.params.etagenum});
});
```

Ce code fait appel à un fichier chambre.ejs **qui doit se trouver dans un sous-dossier appelé "views"**. Créez donc un fichier `/views/chambre.ejs` et placez-y le code suivant :

Code : Autre

```
<h1>Vous êtes dans la chambre</h1>

<p>Vous êtes à l'étage n°<%= etage %></p>
```

La balise `<%= etage %>` sera remplacée par la variable `etage` que l'on a transmise au template avec `{etage: req.params.etagenum}` !

Formation Angular 5

Qu’est ce que Angular ?

- C’est un framework open-source front-end : côté client
- Il a été créé et est toujours maintenu par Google
- Interfaces de type **monopage** ou “**single page**” qui fonctionnent Sans rechargement de la page : Single Page Applications (SPA)

Les avantages d’Angular

- Experience utilisateur plus agréable, navigation fluidifiée.
- Rapidité d’execution conséquente - les calculs partie client
- Responsiveness
- Interfaces haut de gamme - solution aux interfaces complexes
- Composants réutilisables et re-testable : gain de temps !
- Facilité de prise en main des projets
- Forte maintenabilité

Popularité d’Angular

Angular (2,4,5 ou 6) est un des **framework front-end (Javascript)** les plus utilisés du moment ! Très different de AngularJS (limité)
ReactJS (Facebook) et VueJS sont des alternatives –
Il reste le préféré pour les applications solides et complexes.

Code : Langage TypeScript

Angular repose sur **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.

Nouveautés : POO et Typage strict des variables.

Le navigateur ne comprenant que HTML, CSS, JS natif, il faut traduire !

TSC : TypeScript Compiler (tourne dans un environnement Node.Js)

En savoir plus **: https://www.typescriptlang.org/**

Les 3 concepts clés

I - Les composants : **les briques de base** constituant une application.
Ex : formulaire, liste de clients
Composant = Classe + Template
Fonctionnalités simples à réutiliser et à maintenir.

II - Les modules : **Un groupement de composants** mis ensemble.
Facile à réutiliser dans d'autres applications.
Tel qu'un package (JAVA) ou Namespace(C#)

III - Les services : **Ils assurent la communication** inter-composants
+ la communication avec le monde extérieur
Des tâches pour le bon fonctionnement de l'app.

Lancement de l’app
Angular en pratique

- **Installation de l’environnement de développement**
- **Creation d’un Workspace et initialisation de l’app**
- **Serveur de l’application**
- **Appliquer des modifications**

1) Console et Sublime Text (ou autre éditeur) ready !
2) Node et NPM installés
3) **npm install -g @angular/cli**
4) **ng new my-app**

Angular CLI procède à l’installation des packages npm nécessaires et des autres dépendances.
- Nouvel espace de travail avec dossier racine “my-app”
- Une arborescence du projet dans src
- e2e tests end-to-end
- Des fichiers de configurations

5) **cd my-app** puis **ng serve —open** (cette partie ouvre directement le browser à localhost:4200)

Les Composants

I - Une classe TypeScript

Données (propriétés)et comportements (méthodes)

```typescript
@Component({
 selector: 'app-clients-list', // Ce composant pourra être appelé avec la balise <app-clients-list>
 templateUrl: './clients.component.html', // Ce composant affichera le HTML qui se trouve dans ce fichier
 styleUrls: ['./clients.component.css'] // Ce composant utilisera le style CSS qui est dans ce fichier
})
export class ClientsListComponent {
  // La propriété clients est un tableau d'objets qui représente des clients
  clients = [
    { prenom: 'Lior', nom: 'Chamla' },
    { prenom: 'Eliott', nom: 'Smith' },
    { prenom: 'Joseph', nom: 'Dupont'}
  ];

}
```

II - Un Template HTML

Affichage des données

```html
<h2>Liste de clients !</h2>
<table>
  <thead>
    <tr>
      <th>Nom</th>
      <th>Prénom</th>
    </tr>
  </thead>
  <tbody>
    <!-- ici on explique que l'on veut que ce <tr> s'affiche pour
         chaque client au sein du tableau clients qui se trouve dans la classe typescript
         grâce à la directive *ngFor (voir plus loin)
    -->
    <tr *ngFor="let client of clients">
      <!-- lorsque l'on utilise la syntaxe {{ variable }} on demande à Angular d'afficher
           d'une variable (ici le nom du client)
      -->
      <td class="important">{{ client.nom }}</td>
      <td>{{ client.prenom }}</td>
    </tr>
  </tbody>
</table>
```

III - Feuille de style CSS

Style du composant

```css
td.important {
    font-weight: bold;
}

h2 {
    font-size: 2em;
}
```

Les Composants

AppComponent: Le composant “coquille” contenant tous les autres, représentant globalement l’app Angular

@Component *Un décorateur - une fonction qui permet de donner des informations supplémentaires par rapport à une classe, une fonction ou même une variable. Angular est capable de lire ces décorateurs. Ici cela indique que ce n’est pas une simple classe.*

Template HTML *Il puise dans les données qui sont contenus dans la classe TypeScript du composant. Il en affiche les propriétés et les méthodes (comportements).*

- **La directive *ngFor** : permet de boucler sur les éléments d’un tableau afin de répéter un bout de code plusieurs fois
- **{{ variable }}** : affiche le contenu d’une variable
- **<app-client-list>** appelé au sein de notre application, affiche un tableau contenant une liste de clients.

<app-clients-list></app-clients-list> : Angular remplacera cette balise par le HTML dans le template du composant.

Comment le Template arrive t’il à puiser les données de la classe TypeScript ?

Le phénomène liant des données de la classe TypeScript à des éléments HTML du template est: **Le Data Binding (la liaison de données).**

Les Composants

Le Data Binding - La liaison des données : communication intra-composant

1) La syntaxe d'**interpolation : {{ variable }}** - affichage du contenu d'une propriété ou le retour d'une fonction (données primitives)

2) La syntaxe de **property binding : ** - attributs à la valeur de données dans la classe TypeScript.

```
<!-- Cette notation est correcte (interpolation) -->
<img src="{{ imageUrl }}" />
<!-- Cette notation est aussi correcte et conseillée (property binding) -->
<img [src]="imageUrl" />
```

Le fait d'encadrer un attribut par des crochets permet indique à Angular que la valeur est une variable /une fonction issue de la classe.

3) La syntaxe d'**event binding** : **<button (click)='myFunction()'>Click me !</button>** - Réaction aux actions utilisateurs

On va souvent appeler un comportement qui est codé au sein de la classe TypeScript. On utilise alors l'Event Binding

```
<!-- Lorsque l'utilisateur va cliquer sur le bouton, on va appeler
     la fonction myFunction() au sein de la classe TypeScript -->
<button (click)="myFunction()">Click me !</button>
```

```
<!-- Lorsque l'utilisateur va taper une lettre au sein de cet input
     on va appeler la fonction search() qui se trouve dans la classe
     TypeScript en lui passant $event qui représente les détails
     de l'événement -->
<input type="text" (keyup)="search($event)" />
```

Un peu de Pratique

1) Ouvrir ./src/app/app.component.ts.
2) Changer “title” —> Mise à jour automatique

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Ma premiere application!';
}
```

3) Améliorer le Style du composant src/app/app.component.css

```
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
```

Création d’une application SIRH de gestion des ressources De l’entreprise AMAZOGLE

Ne gardez que ce qu’il y a ci dessous en remplaçant le texte

Bienvenue dans SIRH de AMAZOGLE!

Appliquer le style général suivant dans src/style.css

```
/* Application-wide Styles */
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
h2, h3 {
  color: #444;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
}
body, input[type="text"], button {
  color: #888;
  font-family: Cambria, Georgia;
}
/* everywhere else */
* {
  font-family: Arial, Helvetica, sans-serif;
}
```

Des outils supplémentaires

I - Le routeur : **Plusieurs vues sous des URLs differentes**

II - Les requêtes HTTP : **Communication avec serveurs distants**

III - Les directives :**Nouveaux attributs HTML ! On peut aussi en créer**

La CLI d'Angular : Command Line Interface

Au sein d'Angular, on va créer beaucoup de fichiers, 3 fichiers minimum ne serait-ce que pour un seul composant, sachant qu'une application peut contenir des dizaines de composants.

```
# La commande ng new [nom de l'application] permet de créer un projet Angular
# Elle est essentielle car un projet Angular possède beaucoup de fichiers
# et surtout beaucoup de configurations qu'il est presque impossible de mettre
# en place à la main
ng new NomDeLApplication

# La commande ng serve permet de lancer un serveur local qui contiendra
# notre application Angular (l'adresse sera souvent http://localhost:4200)
# L'option "-o" permet aux flemmards de demander à la CLI d'ouvrir automatiquement
# notre navigateur à cette adresse
ng serve -o

# La commande ng generate permet de générer des blocs fonctionnels tels que
# des composants ou des services, des directives etc
ng generate component clients-list
ng generate service clients

# La commande ng build permet de construire l'application de façon à la mettre
# en production
ng build

# La commande ng seule permet d'avoir la liste des commandes possibles et
# leur description
ng
```

Un peu de Pratique

Chez AMAZOGLE, nous valorisons beaucoup les employés. Ce sont des talents. Ce sont des héros.
Nous allons donc générer un composant “heroes” **: ng generate component heroes**
New folder : src/app/heroes/

```
CREATE src/app/heroes/heroes.component.css (0 bytes)
CREATE src/app/heroes/heroes.component.html (25 bytes)
CREATE src/app/heroes/heroes.component.spec.ts (628 bytes)
CREATE src/app/heroes/heroes.component.ts (269 bytes)
UPDATE src/app/app.module.ts (475 bytes)
```

Dans heroes.component.ts (hero property) : hero = ‘Xavier’;
Dans heroes template file : heroes.component.html —> {{hero}}
Le faire apparaitre dans la vue en dessous du titre : app.component.html

```
<h1>{{title}}</h1>

<app-heroes></app-heroes>
```

Un employé est bien plus qu’un simple nom ! Créons donc une classe : src/app/hero.ts

```
export class Hero {

  id: number;

  name: string;

}
```

Importer la classe Hero dans HeroesComponent

Hero property est à initialiser avec id à 1 et name à Xavier

src/app/heroes/heroes.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  hero: Hero = {
    id: 1,
    name: 'Xavier'
  };

  constructor() { }

  ngOnInit() {
  }

}
```

```
<h2>Informations de {{hero.name}}</h2>
<div><span>id: </span>{{hero.id}}</div><br/>
<div><span>name: </span>{{hero.name}}</div>
```

Utilisation de Pipe pour le formatting

```
<h2>{{hero.name | uppercase}} Details</h2>
```

Les Directives

Au sein d'Angular, il existe ce qu'on appelle des **directives** :
elles permettent d'appliquer un comportement à des balises HTML ou à des composants qu'on utilise.
Il en existe **deux sortes** :

1) **Les directives structurelles** : ayant un effet sur la structure de la page. Par ex : *ngFor ou *ngIf (toujours un astérisque)

```
<ul>
  <!-- Ce <li> va s'afficher 4 fois vu qu'il y a 4 nombres dans le tableau -->
  <li *ngFor="let nombre of [10, 20, 30, 40]">
    Le nombre est {{ nombre }}
  </li>
</ul>

<div *ngIf="2 + 2 === 5">
  <p>Cette div ne s'affichera jamais car la condition ne sera jamais respectée</p>
</div>

<!-- En imaginant que la variable isAuthenticated contient réellement true
     alors ce <h1> s'affichera -->
<h1 *ngIf="isAuthenticated === true">
  Bienvenue !
</h1>
```

2) **Les directives d'attributs** (non structurelles) : Elles enrichissent les éléments avec du style ou des comportements

ngClass : Préciser qu'un élément HTML devra avoir certaines classes sous certaines conditions
ngStyle : Préciser des règles de styles CSS à un élément HTML en tenant compte de valeurs de variables
ngModel : Permet d'attacher un <input> ou autre contrôle de formulaire à une variable qui se trouve dans notre classe TypeScript

Un peu de Pratique

Permettons aux responsables des ressources humaines de modifier le nom des collaborateurs grace à un input !
2 choses arrivent en même temps que le RH modifie le nom :

La propriété dans la classe est mise à jour **EN MEME TEMPS** que l'information est renseignée à l'écran

On appelle cela le **Two-Way Data Binding** entre l'**<input>** et la propriété **hero.name.**
La Syntaxe Angular du Two-Way Binding est [(ngModel)]

```
<div>
  <label>name:
    <input [(ngModel)]="hero.name" placeholder="name">
  </label>
</div>
```

Les données circulent dans deux directions *:* de la propriété `hero.name` à l'input ,et de l'input à `hero.name`.
Erreur

```
Uncaught Error: Template parse errors:
Can't bind to 'ngModel' since it isn't a known property of 'input'. ("
```

ngModel est une directive (non structurelle) appartenant au module FormsModule. Elle n'y est pas par défaut !!

Les Modules

Un module Angular est un ensemble de composants que l'on met en relation au sein d'un package (contenant un ensemble de fonctionnalités)

Il existe une multitude de **modules pré-développés et qui sont à utiliser !**

- **FormsModule** : le module qui contient toutes sortes de fonctionnalités et de directives qui ont trait à la gestion des formulaires
- **RouterModule** : le module qui contient toutes les fonctionnalités liées au routage dans l'application
- **HttpClientModule** : le module qui contient toutes les fonctionnalités liées aux appels http distants

Mais il existe un module qui est particulier : **AppModule**. C'est le chef d'orchestre connaissant tous les composants de l'application.

```
/* the AppModule class with the @NgModule decorator */
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

The `@NgModule` decorator identifies `AppModule` as an `NgModule` class. `@NgModule` takes a metadata object that tells Angular how to compile and launch the application.

- ***declarations***—this application's lone component.
- ***imports***—import `BrowserModule` to have browser specific services such as DOM rendering, sanitization, and location.
- ***providers***—the service providers.
- ***bootstrap***—the *root* component that Angular creates and inserts into the `index.html` host web page.

The default application created by the Angular CLI only has one component, `AppComponent`, so it is in both the `declarations` and the `bootstrap` arrays.

Un peu de Pratique

Ajoutons donc FormsModule dans **src/app/app.module.ts**

```
import { FormsModule } from '@angular/forms'; // <-- NgModel lives here

imports: [
  BrowserModule,
  FormsModule
],
```

WARNING
Tous les composants doivent être déclarés dans ngModule.
HeroesComponent a été déclaré
Par défaut grâce à Angular CLI

```
import { HeroesComponent } from './heroes/heroes.component';
```

```
<h2>Informations de {{hero.name}}</h2>
<div><span>id: </span>{{hero.id}}</div><br/>
<div><span>name: </span>{{hero.name}}</div>

<div><br/><br/><br/>
  <label>name:
    <input [(ngModel)]="hero.name" placeholder="name">
  </label>
</div>
```


```
@NgModule({
  declarations: [
    AppComponent,
    HeroesComponent
  ],
```

Faites de même en ajoutant les propriétés suivantes des employés :
- Current Position : Engineer, Business Developer, Manager, Principal
- Age
- Gender : M - W radio button
- Sector : selection entre Computer Science, Business, Marketing, Operations
- Adress

Un peu de Pratique

Le DRH a demandé à avoir accès à la liste de tous les employés sur le SIRH ! Allons-y faisons apparaître les héros / talents de AMAZOGLE !

A défaut d'avoir accès tout de suite à la base de données des employés de l'entreprise, utilisons un fichier data : **mock-heroes.ts** dans src/app ! (Initialisez les autres propriétés si vous en avez crée en plus…)

mock-heroes.ts | app.component.ts | styles.css | hero

```
import { Hero } from './hero';

export const HEROES: Hero[] = [
  { id: 11, name: 'Anas Yousfi' },
  { id: 12, name: 'Assa Traore' },
  { id: 13, name: 'Hakim Belfkih' },
  { id: 14, name: 'Julie Buisson' },
  { id: 15, name: 'Magali Piszczyglowa' },
  { id: 16, name: 'Mathieu Berrah' },
  { id: 17, name: 'Mehdi Traore' },
  { id: 18, name: 'Mohamed Ba' },
  { id: 19, name: 'Mohamed Aarous' },
  { id: 21, name: 'Sonia Bougamha' },
  { id: 20, name: 'Yvette Toukam' }
];
```

FIRST - importons nos données dans src/app/heroes/heroes.component.ts

```
import { HEROES } from '../mock-heroes';
```

Ajout de la propriété :

```
export class HeroesComponent implements OnInit {
heroes = HEROES;
```

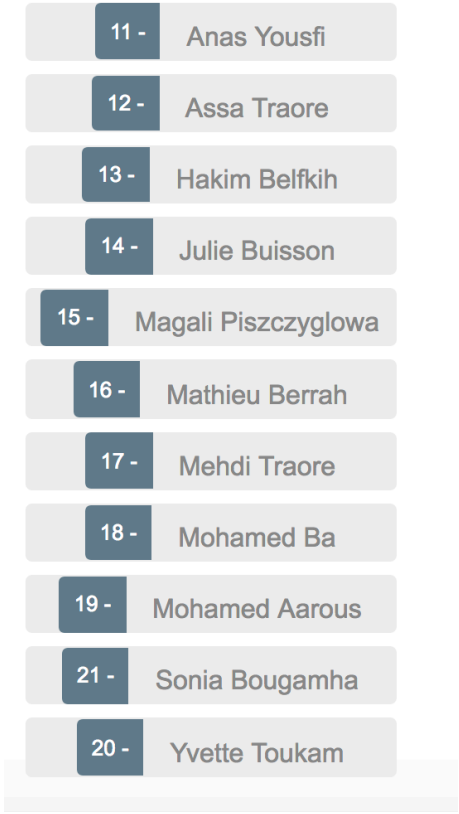
AFFICHAGE - utilisons la directive *ngFor

```
<br/><br/>
<h2>La liste des employés :</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes">
    <span class="badge">{{hero.id}} - </span> {{hero.name}}
  </li>
</ul>
```

Un peu de Pratique

Améliorons le Style

Bienvenue dans SIRH de AMAZOGLE!




```css
/* HeroesComponent's private CSS styles */
.selected {
  background-color: #CFD8DC !important;
  color: white;
}
.heroes {
  margin: 0 0 2em 0;
  list-style-type: none;
  padding: 0;
  width: 15em;
}
.heroes li {
  cursor: pointer;
  position: relative;
  left: 0;
  background-color: #EEE;
  margin: .5em;
  padding: .3em 0;
  height: 1.6em;
  border-radius: 4px;
}
.heroes li.selected:hover {
  background-color: #BBD8DC !important;
  color: white;
}
.heroes li:hover {
  color: #607D8B;
  background-color: #DDD;
  left: .1em;
}
.heroes .text {
  position: relative;
  top: -3px;
}
.heroes .badge {
  display: inline-block;
  font-size: small;
  color: white;
  padding: 0.8em 0.7em 0 0.7em;
  background-color: #607D8B;
  line-height: 1em;
  position: relative;
  left: -1px;
  top: -4px;
  height: 1.8em;
  margin-right: .8em;
  border-radius: 4px 0 0 4px;
}
```

Un peu de Pratique

Le DRH a souhaite désormais que l'on obtienne le détail des employés.
Lorsque l'on **click** sur un employé, ses détails s'affichent.

On ajoute un event binding :

```
<li *ngFor="let hero of heroes" (click)="onSelect(hero)">
```

() indique à Angular d'être à l'écoute des évènements qui ont lieu pour li

La méthode onSelect() est dans HeroesComponent

```
selectedHero: Hero;
onSelect(hero: Hero): void {
  this.selectedHero = hero;
}
```

Assignation de l'element cliqué à selectedHero

```
Affichage de l'élément sélectionné :
<h2>{{selectedHero.name | uppercase}} Details</h2>
<div><span>id: </span>{{selectedHero.id}}</div>
<div>
  <label>name:
    <input [(ngModel)]="selectedHero.name" placeholder="name">
  </label>
</div>
```

```
ERROR TypeError: Cannot read property 'name' of undefined
```

selectedHero n'est pas défini avant une 1ere sélection !!

Ajoutons donc la condition SI IL Y A SELECTION :

***ngIf="selectedHero" : Disparait du DOM (non caché)**

```
<div *ngIf="selectedHero">
  <h2>{{selectedHero.name | uppercase}} Details</h2>
  <div><span>id: </span>{{selectedHero.id}}</div>
  <div>
   <label>name:
   <input [(ngModel)]="selectedHero.name" placeholder="name">
   </label>
  </div>
</div>
```

Vrai ou Faux

- Angular permet un développement par modules et composants

- Angular permet un développement avec de très bonnes performances

- Angular est un langage Front-End

- Angular permet un meilleur référencement

- Pour créer un nouveau composant, j'utilise : Angular Generator, Angular CDN, Angular CLI ou Angular Creator

- Angular permet un meilleur référencement

- Que se passe t'il avec ces code ?

```
<li [class.good]="eleve.note > 15">
   <span>{{eleve.note}}</span> {{eleve.nom}}
</li>
```

```
<button (click)="person = male" [disabled]="person.sex=='m'">Male</button>
```

```
<button (click)="person = female" [disabled]="person.sex=='f'">Female</button>
```

```
 <p><img [src]="person.photo" [alt]="person.name" [title]="person.name"></p>
```

- Angular est un framework NPM, CDN, serveur, client

- Angular utilise : des composants, des moteurs, des modules, des directions

- Comment afficher le contenu de la variable hero dans la balise h1:

<h1>((titre))</h1>
<h1>$titre</h1>
<h1>{{titre}}</h1>
{{<h1>titre</h1>}}

- Dans le decorator Component on a :
selector, templateUrls, templateUrl, styleUrls, styleUrl, javascriptUrl, jsUrls

- Soit le composant suivant : comment l'utiliser dans un document html

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

@Component({
selector: 'my-list',
templateUrl: './my-list.component.html',
styleUrls: ['./my-list.component.css']
})
export class MyListComponent {
list = [1,2,3];
}
```

- Qu'est ce que le Two-Way Binding ? Que faut-il importer pour que cela fonctionne ?

- Quelles sont les trois types de Data Binding ?

- Qu'est ce que AppModule, NgModule, FormsModule ?

- Quelle est la bonne syntaxe de *ng-for ?

<li *ngFor(let eleve of eleves)>
<li *ngFor="let eleve of eleves">
<li *ngFor(let eleve=0 ;eleve<eleves.lenght();eleves ++)>
<li *ngFor="let {{eleve}} of [[eleves]]">

- Quels sont les deux concurrents de Angular ?

- Qui a crée Angular ? Qui a crée TypeScript ?

- Quels sont les deux éléments que TypeScript a de plus que Javascript ?

- Quels sont les 3 concepts clés d’Angular ?

La communication entre les composants

Le cœur d’Angular, c’est de pouvoir faire communiquer les composants les uns avec les autres

Pour cela on utilise différents moyens dont deux sont les plus simples :

1- Envoyer des données à un composant
2 -Ecouter les événements qui ont lieu dans un composant.

- Quels sont les 2 décorateurs que l’on utilise ?
- Qui est le composant coquille ?
- Qu’est ce qu’une directive ? Les deux types ? Difference de syntaxe ?
- Que fait la directive *ngIf et que s’est il passé à son utilisation hier ?
- A quoi sert Package.json
- Node est il synchrone ou asynchrone ?
- Qu’est ce qu’une callback function ?
- Qu’est ce que SPA et quel avantage ?
- Qu’est ce qu’un pipe ? Donner un exemple ..
- Qu’est ce que NgModule ?

NgModules	`import { NgModule } from '@angular/core';`

`@NgModule({ declarations: ..., imports: ...,`
`exports: ..., providers: ..., bootstrap: ...})`
`class MyModule {}`
	Defines a module that contains components, directives, pipes, and providers.

`declarations:` `[MyRedComponent, MyBlueComponent, MyDatePipe]`	List of components, directives, and pipes that belong to this module.
`imports:` `[BrowserModule, SomeOtherModule]`	List of modules to import into this module. Everything from the imported modules is available to `declarations` of this module.
`exports:` `[MyRedComponent, MyDatePipe]`	List of components, directives, and pipes visible to modules that import this module.
`providers:` `[MyService, { provide: ... }]`	List of dependency injection providers visible both to the contents of this module and importers of this module.
`entryComponents:` `[SomeComponent, OtherComponent]`	List of components not referenced in any reachable template, for example dynamically created from code.
`bootstrap:` `[MyAppComponent]`	List of components to bootstrap when this module is bootstrapped.

- Une application Angular est un ensemble de :

- **Modules**
- **Component**
- **Template**
- **Directives**
- **Data Binding**
- **Services**
- **Dependency Injection**
- **Routing**

La communication entre deux composants Père/fils

@input dans le fils : la donnée est transmise du Père au fils

@output dans le fils, l'event est transmis du fils au père

La difference entre composant et directive

- Le composant est une partie constitutive de l'application AppComponent

- La directive ajoute un comportement à un élément DOM existant

Template syntax	
`<input **[value]**="firstName">`	Binds property `value` to the result of expression `firstName`.
`<div **[attr.role]**="myAriaRole">`	Binds attribute `role` to the result of expression `myAriaRole`.
`<div **[class.extra-sparkle]**="isDelightful">`	Binds the presence of the CSS class `extra-sparkle` on the element to the truthiness of the expression `isDelightful`.
`<div **[style.width.px]**="mySize">`	Binds style property `width` to the result of expression `mySize` in pixels. Units are optional.
`<button **(click)**="readRainbow($event)">`	Calls method `readRainbow` when a click event is triggered on this button element (or its children) and passes in the event object.
`<div title="Hello **{{ponyName}}**">`	Binds a property to an interpolated string, for example, "Hello Seabiscuit". Equivalent to: `<div [title]="'Hello ' + ponyName">`
`<p>Hello **{{ponyName}}**</p>`	Binds text content to an interpolated string, for example, "Hello Seabiscuit".
`<my-cmp **[(title)]**="name">`	Sets up two-way data binding. Equivalent to: `<my-cmp [title]="name" (titleChange)="name=$event">`

`<video #movieplayer ...>`
`<button (click)="movieplayer.play()">`
`</video>`
	Creates a local variable `movieplayer` that provides access to the `video` element instance in data-binding and event-binding expressions in the current template.

`<p *myUnless="myExpression">...</p>`	The * symbol turns the current element into an embedded template. Equivalent to: `<ng-template [myUnless]="myExpression"><p>...</p></ng-template>`
`<p>Card No.: {{cardNumber | myCardNumberFormatter}}</p>`	Transforms the current value of expression `cardNumber` via the pipe called `myCardNumberFormatter`.
`<p>Employer: {{employer?.companyName}}</p>`	The safe navigation operator (`?`) means that the `employer` field is optional and if `undefined`, the rest of the expression should be ignored.
`<svg:rect x="0" y="0" width="100" height="100"/>`	An SVG snippet template needs an `svg:` prefix on its root element to disambiguate the SVG element from an HTML component.

`<svg>`
`<rect x="0" y="0" width="100" height="100"/>`
`</svg>`
	An `<svg>` root element is detected as an SVG element automatically, without the prefix.

Built-in directives	`import { CommonModule } from '@angular/common';`
`<section *ngIf="showSection">`	Removes or recreates a portion of the DOM tree based on the `showSection` expression.
`<li *ngFor="let item of list">`	Turns the li element and its contents into a template, and uses that to instantiate a view for each item in list.

`<div [ngSwitch]="conditionExpression">`
`<ng-template [ngSwitchCase]="case1Exp">...</ng-template>`
`<ng-template ngSwitchCase="case2LiteralString">...</ng-template>`
`<ng-template ngSwitchDefault>...</ng-template>`
`</div>`
	Conditionally swaps the contents of the div by selecting one of the embedded templates based on the current value of `conditionExpression`.

`<div [ngClass]="{'active': isActive, 'disabled': isDisabled}">`	Binds the presence of CSS classes on the element to the truthiness of the associated map values. The right-hand expression should return {class-name: true/false} map.
`<div [ngStyle]="{'property': 'value'}">` `<div [ngStyle]="dynamicStyles()">`	Allows you to assign styles to an HTML element using CSS. You can use CSS directly, as in the first example, or you can call a method from the component.

Forms	`import { FormsModule } from '@angular/forms';`
`<input [(ngModel)]="userName">`	Provides two-way data-binding, parsing, and validation for form controls.

Class decorators	`import { Directive, ... } from '@angular/core';`
`@Component({...})` `class MyComponent() {}`	Declares that a class is a component and provides metadata about the component.
`@Directive({...})` `class MyDirective() {}`	Declares that a class is a directive and provides metadata about the directive.
`@Pipe({...})` `class MyPipe() {}`	Declares that a class is a pipe and provides metadata about the pipe.
`@Injectable()` `class MyService() {}`	Declares that a class has dependencies that should be injected into the constructor when the dependency injector is creating an instance of this class.

Directive and component change detection and lifecycle hooks	(implemented as class methods)
`constructor(myService: MyService, ...) { ... }`	Called before any other lifecycle hook. Use it to inject dependencies, but avoid any serious work here.
`ngOnChanges(changeRecord) { ... }`	Called after every change to input properties and before processing content or child views.
`ngOnInit() { ... }`	Called after the constructor, initializing input properties, and the first call to `ngOnChanges`.
`ngDoCheck() { ... }`	Called every time that the input properties of a component or a directive are checked. Use it to extend change detection by performing a custom check.
`ngAfterContentInit() { ... }`	Called after `ngOnInit` when the component's or directive's content has been initialized.
`ngAfterContentChecked() { ... }`	Called after every check of the component's or directive's content.
`ngAfterViewInit() { ... }`	Called after `ngAfterContentInit` when the component's views and child views / the view that a directive is in has been initialized.
`ngAfterViewChecked() { ... }`	Called after every check of the component's views and child views / the view that a directive is in.
`ngOnDestroy() { ... }`	Called once, before the instance is destroyed.

COMMAND	ALIAS	DESCRIPTION
add		Adds support for an external library to your project.
build	b	Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory.
config		Retrieves or sets Angular configuration values in the angular.json file for the workspace.
doc	d	Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword.
e2e	e	Builds and serves an Angular app, then runs end-to-end tests using Protractor.
generate	g	Generates and/or modifies files based on a schematic.
help		Lists available commands and their short descriptions.
lint	l	Runs linting tools on Angular app code in a given project folder.
new	n	Creates a new workspace and an initial Angular app.
run		Runs an Architect target with an optional custom builder configuration defined in your project.
serve	s	Builds and serves your app, rebuilding on file changes.
test	t	Runs unit tests in a project.
update		Updates your application and its dependencies. See https://update.angular.io/
version	v	Outputs Angular CLI version.
xi18n		Extracts i18n messages from source code.

Les Composants

@Input et @Output : des décorateurs pour la communication entre composants.
Envoi de données et Ecoute d'évènements dans d'autres composants.

@Input : Envoi de données depuis l'extérieur

```
@Component({
 selector: 'app-clients-list', // Ce composant pourra être appelé avec la balise <app-clients-list>
 templateUrl: './clients.component.html', // Ce composant affichera le HTML qui se trouve dans ce fichier
 styleUrls: ['./clients.component.css'] // Ce composant utilisera le style CSS qui est dans ce fichier
})
export class ClientsListComponent {
  // La propriété clients est un tableau d'objets qui représente des clients
  // Ici le décorateur @Input() permet de dire à Angular que la variable clients peut être spécifiée
  // depuis l'extérieur sous la forme d'un attribut de la balise <app-clients-list>
  @Input() clients = [];

}
```

```
<!-- Cette syntaxe ne fonctionnera pas car Angular pensera qu'on envoi à la
     propriété clients une simple chaine de caractères -->
<app-clients-list clients="[{ prenom: 'Lior', nom: 'Chamla' }, { prenom: 'Joseph', nom: 'Dupont' }]"></app-clients-list>

<!-- Cette syntaxe fonctionnera car Angular comprendra qu'on passe à la propriété
     "clients" un véritable tableau javascript (syntaxe de property binding) -->
<app-clients-list [clients]="[{ prenom: 'Lior', nom: 'Chamla' }, { prenom: 'Joseph', nom: 'Dupont' }]"></app-clients-list>
```

@Output : Ecoute d'évènement dans un composant

```
@Component({
 selector: 'app-clients-list', // Ce composant pourra être appelé avec la balise <app-clients-list>
 templateUrl: './clients.component.html', // Ce composant affichera le HTML qui se trouve dans ce fichier
 styleUrls: ['./clients.component.css'] // Ce composant utilisera le style CSS qui est dans ce fichier
})
export class ClientsListComponent {
  // La propriété clients est un tableau d'objets qui représente des clients
  @Input() clients = [];
  // On déclare ici un émetteur d'événement en tant qu'Output() qui pourra donc sortir du composant
  @Output() clientClicked = new EventEmitter();

  onClickClient(client) {
    // On demande à l'émetteur d'événement d'émettre un événement porteur de la donnée
    // client
    this.clientClicked.emit(client)
  }

}
```

```
<!-- On explique ici que lorsque l'événement clientClicked aura lieu sur ce composant,
     on veut afficher dans la console le client en question (qui est contenu dans
     la variable $event) -->
<app-clients-list
   [clients]="[{ nom: 'Chamla', prenom: 'Lior' }, { nom: 'Dupont', prenom: 'Joseph' }]"
   (clientClicked)="console.log($event)">
</app-clients-list>
```

Un peu de Pratique

Ajout d'une classe avec condition : **si c'est l'employé sélectionné**
Selected étant un style défini en css

```
<li *ngFor="let hero of heroes"
  [class.selected]="hero === selectedHero"
  (click)="onSelect(hero)">
```

Remarque : Nous n'avons pas fait les choses proprement jusqu'ici

Notre composant fait l'équivalent de 2 fonctionnalités au lieu d'une seule !

1) Affichage de la liste des employés
2) Affichage des détails suite à un click sur un des employés de la liste.

Il faut adopter la philosophie 1 to 1 pour plus de **maintenabilité et reutilisation**
Divisons nos "super" composants **en sub-components** pour des tâches spécifiques

HeroComponent : Liste des employés
HeroDetailComponent : Détail d'un employé sélectionné

```
ng generate component hero-detail
```

Hero detail template HTML est le suivant :

```
<div *ngIf="hero">
  <h2>{{hero.name | uppercase}} Details</h2>
  <div><span>id: </span>{{hero.id}}</div>
  <div>
    <label>name:
   <input [(ngModel)]="hero.name" placeholder="name"/>
    </label>
  </div>
</div>
```

IL FAUT IMPORTER HERO: hero-detail.component.ts

```
import { Hero } from '../hero';
```

Un peu de Pratique

Pour afficher proprement les détails, nous feront appel à HeroDetailComponent de cette façon, dans HeroesComponent (appel externe)

```
Nous aurons donc besoin d'un décorateur @Input()
Donc dans hero-detail.component.ts

import { Component, OnInit, Input } from '@angular/core';
@Input() hero: Hero;

HeroesComponent délègue une partie des tâches initiales.
Il a une relation parent/child avec HeroDetailComponent

<app-hero-detail [hero]="selectedHero"></app-hero-detail>

==> [hero]="selectedHero" est un property binding. On change la propriété hero définie dans HeroDetailComponent
```

Un peu de Pratique

hero-detail.component.ts

```ts
import { Component, OnInit, Input } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-hero-detail',
  templateUrl: './hero-detail.component.html',
  styleUrls: ['./hero-detail.component.css']
})
export class HeroDetailComponent implements OnInit {
  @Input() hero: Hero;

  constructor() { }

  ngOnInit() {
  }

}
```

hero-detail.component.html

```html
<div *ngIf="hero">

  <h2>{{hero.name | uppercase}} Details</h2>
  <div><span>id: </span>{{hero.id}}</div>
  <div>
    <label>name:
      <input [(ngModel)]="hero.name" placeholder="name"/>
    </label>
  </div>

</div>
```

heroes.component.html

```html
<br/><br/>
<h2>La liste des employés :</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes" [class.selected]="hero === selectedHero" (click)="onSelect(
  hero)">
    <span class="badge">{{hero.id}} - </span> {{hero.name}}
  </li>
</ul>
<br/><br/>

<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

Un peu de Pratique

app.module.ts

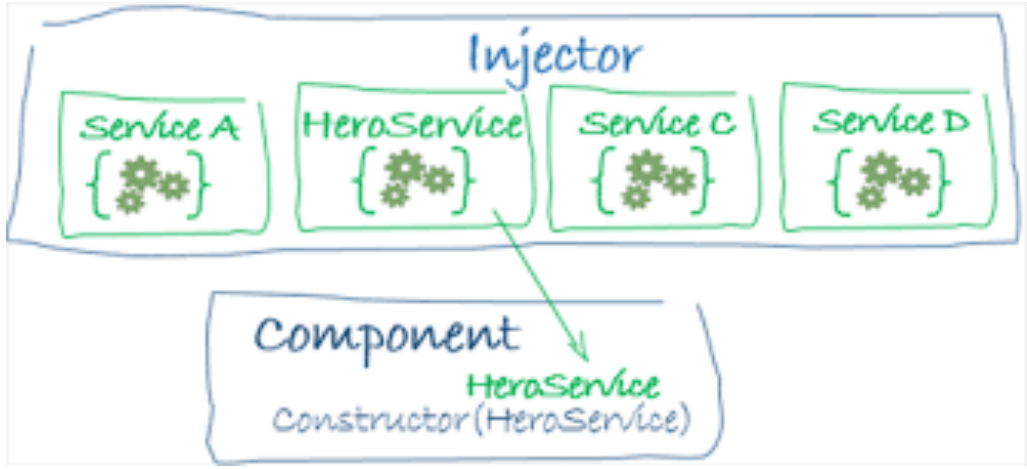
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { HeroesComponent } from './heroes/heroes.component';
import { HeroDetailComponent } from './hero-detail/hero-detail.component';

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

Les Services

La communication entre les différents composants (entre eux mais aussi entre eux et le monde extérieur) est fondamentale. C'est à ça que servent l**es services**, que l'on peut créer à sa guise pour offrir des outils supplémentaires à nos composants.

Why services

Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. They should focus on presenting data and delegate data access to a service.



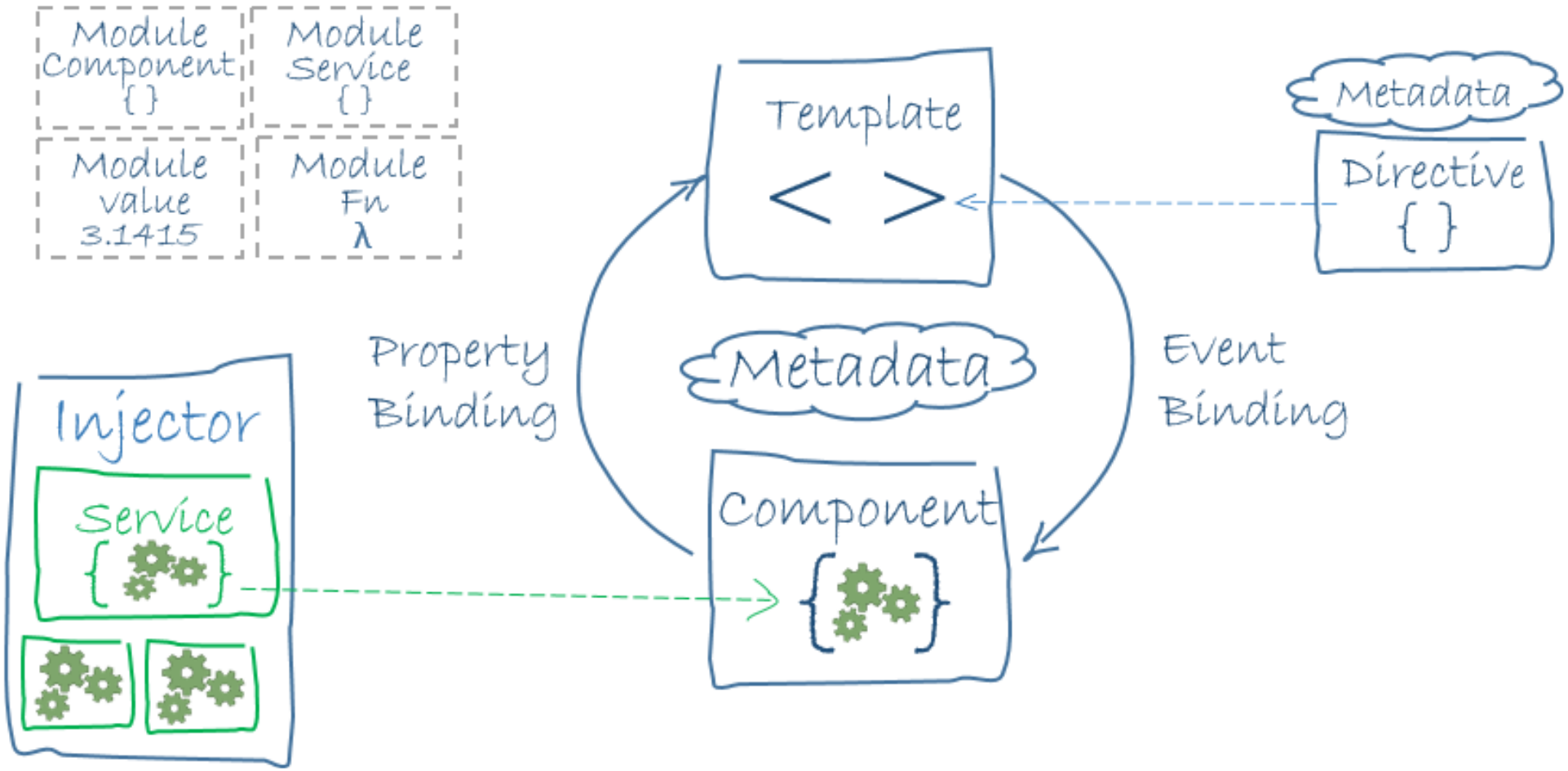
Les Services

Injection de dépendance : DI - Dependency Injection
L'injection de dépendance est une notion importante à bien comprendre et à bien implémenter

"Si une classe a besoin d'une instance d'une autre classe, que ce soit dans son constructeur ou dans une autre méthode (un *setter* par exemple), alors elle prend cette instance directement en paramètre et ne s'occupe certainement pas de l'instancier elle-même"

`@Injectable()` est un **décorateur** un peu particulier. Il ne permet pas l'injection à proprement parlé, mais plutôt d'initialiser un contexte détectabilité. Si vous injectez dans un de vos services (sans ce décorateur) un autre service, le moteur d'injection retournera une erreur.

Vue
Globale


Module
Component
{ }
Module
Service
{ }
Module
value
3.1415
Module
Fn
λ
Template
< >
Metadata
Directive
{ }
Property
Binding
Metadata
Event
Binding
Injector
Service
{ }
Component
{ }

Un peu de Pratique

Jusqu'ici nous avons utilisé des données "hard-codée", non issue d'un serveur/base de données.

RAPPEL : Le rôle des composants n'est certainement pas de s'occuper de l'accès au données : c'est **le rôle des services.**
Les composants ont pour but de "présenter" les données **récupérées grâce aux services.**

Créons le service HeroService pour récupérer les données relatives aux employés de AMAZOGLE.
Au lieu de créer le service avec la commande CLI "new" utilisons l'injection de dependance Angular.
Ainsi nous l'injectons dans le constructeur de HeroesComponent.

Les services sont d'excellent moyens d'échanger de l'information entre classes inconnues l'une de l'autre.

Le service MessageService sera injecté dans **HeroService** qui l'utilise pour envoyer un message
Le service MessageService sera injecté dans **MessagesComponent** qui l'utilise pour afficher le message

**Le provider indique que*
Le service est utilisable et
Injectable partout. S'il n'est
Plus utile, c'est un bon moyen
De le "disable" : optimisation ++

1- Création de **HeroService**

```
ng generate service hero
Provider* c'est root !
```

src/app/hero.service.ts (new service)

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

@Injectable({
  providedIn: 'root',
})
export class HeroService {

  constructor() { }

}
```

2- **HeroService** prend la donnée et la méthode getHeroes pour récupérer

```
import { Hero } from './hero';
import { HEROES } from './mock-heroes';
```

```
getHeroes(): Hero[] {
  return HEROES;
}
```

3- **HeroesComponent** à mettre à jour

```
import { HeroService } from '../hero.service';

heroes: Hero[];

constructor(private heroService: HeroService) { }

getHeroes(): void {
  this.heroes = this.heroService.getHeroes();
}
```

```
ngOnInit() {
  this.getHeroes();
}
```

Un peu de Pratique

Attention : ce fonctionnement marche car les données sont récupérable de manière “synchrone”
Dans une vrai application cela ne marcherait pas …

```
this.heroes = this.heroService.getHeroes();
```

`HeroService.getHeroes()` DOIT ETRE ASYNCHRONE : “callback”, “promise” ou “observable”
On choisit “observable” car la méthode HttpClient.get() retourne un “observable”
Observable est une classe clé de la librairie : RxJS library
On simulera l’obtention de la donnée via un serveur grâce à la fonction of() de RxJS.

APPROCHE ASYNCHRONE

Dans hero.service.ts :

```
import { Observable, of } from 'rxjs';
```

Remplacer getHeroes() par :

```
getHeroes(): Observable<Hero[]> {
  return of(HEROES);
}
```

`of(HEROES)` retourne un `Observable<Hero[]>` émettant une valeur unique :la liste “hardcodée” de nos employés

On passe de :

```
getHeroes(): void {
  this.heroes = this.heroService.getHeroes();
}
```

À:

```
getHeroes(): void {
  this.heroService.getHeroes()
      .subscribe(heroes => this.heroes = heroes);
}
```

Observable.subscribe() fait toute la différence !
Observable fait la recuperation de la données (il peut prendre son tps) puis subscribe passe les données en callback, mettant à jour la propriété heroes

Un peu de Pratique

Nous avons vu l'envoie asynchrone des données. QUID de l'affichage des données.

Créons MessageComponent : **ng generate component message**

Dans app.component.html cela devient plus simple:

```
<h1>{{title}}</h1>
<app-heroes></app-heroes>
<app-messages></app-messages>
```

==> MessageComponent permet d'afficher des messages en bas de page

Créons MessageService : **ng generate service message**

Message.services.ts doit avoir le contenu suivant ——————————————————>

Maintenant Injectons MessageService dans HeroService **(service in service)**

Dans hero.service.ts:

```
import { MessageService } from './message.service';
constructor(private messageService: MessageService) { }
```

MessageService ==> HeroService ==> HeroesComponent

Ajout et suppression du cache "message"
Dans message.service.ts

```
1.  import { Injectable } from '@angular/core';
2.
3.  @Injectable({
4.    providedIn: 'root',
5.  })
6.  export class MessageService {
7.    messages: string[] = [];
8.
9.    add(message: string) {
10.     this.messages.push(message);
11.   }
12.
13.   clear() {
14.     this.messages = [];
15.   }
16. }
```

Un peu de Pratique

1 - Envoyer un message depuis **HeroService** une fois que les employés sont récupérés:

```
getHeroes(): Observable<Hero[]> {
  // TODO: send the message _after_ fetching the heroes
  this.messageService.add('HeroService: fetched heroes');
  return of(HEROES);
}
```

2 - Afficher le message depuis **HeroService**

MessageComponent est une fonctionnalité d'affichage des messages.
Les messages envoyés par HeroService aussi !

Importons alors MessageService dans MessageComponent

```
import { MessageService } from '../message.service';

constructor(public messageService: MessageService) {}
```

On notera qu'Angular ne permet le BINDING que des composants **public**

Dans messages.component.html

```
<div *ngIf="messageService.messages.length">

  <h2>Messages</h2>
  <button class="clear"
          (click)="messageService.clear()">clear</button>
  <div *ngFor='let message of messageService.messages'> {{message}} </div>

</div>
```

*ngif n'affiche des messages que s'il y en a

`Ajouter le style dans messages.component.css`

Retour sur le code : hero.service.ts

```
1. import { Injectable } from '@angular/core';
2. import { Observable, of } from 'rxjs';
3.
4. import { Hero } from './hero';
5. import { HEROES } from './mock-heroes';
6. import { MessageService } from './message.service';
7.
8. @Injectable({
9.   providedIn: 'root',
10. })
11. export class HeroService {
12.
13.   constructor(private messageService: MessageService) { }
14.
15.   getHeroes(): Observable<Hero[]> {
16.     // TODO: send the message _after_ fetching the heroes
17.     this.messageService.add('HeroService: fetched heroes');
18.     return of(HEROES);
19.   }
20. }
```

Retour sur le code :

message.service.ts

```
1. import { Injectable } from '@angular/core';
2.
3. @Injectable({
4.   providedIn: 'root',
5. })
6. export class MessageService {
7.   messages: string[] = [];
8.
9.   add(message: string) {
10.    this.messages.push(message);
11.  }
12.
13.  clear() {
14.    this.messages = [];
15.  }
16. }
```

heroes.service.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { Hero } from '../hero';
3. import { HeroService } from '../hero.service';
4. @Component({
5.   selector: 'app-heroes',
6.   templateUrl: './heroes.component.html',
7.   styleUrls: ['./heroes.component.css']})
8. export class HeroesComponent implements OnInit {
9.  selectedHero: Hero; heroes: Hero[];
10.  constructor(private heroService: HeroService) { }
11.  ngOnInit() {
12.    this.getHeroes(); }
13.  onSelect(hero: Hero): void {
14.    this.selectedHero = hero;
15.  }
16.  getHeroes(): void {
17.    this.heroService.getHeroes()
18.        .subscribe(heroes => this.heroes = heroes);
19.  }}
```

Retour sur le code :

messages.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { MessageService } from '../message.service';
3.
4. @Component({
5.   selector: 'app-messages',
6.   templateUrl: './messages.component.html',
7.   styleUrls: ['./messages.component.css']
8. })
9. export class MessagesComponent implements OnInit {
10.
11.   constructor(public messageService: MessageService) {}
12.
13.   ngOnInit() {
14.   }
15.
16. }
```

messages.component.html

```
<div *ngIf="messageService.messages.length">

  <h2>Messages</h2>
  <button class="clear"
   (click)="messageService.clear()">clear</button>
  <div *ngFor='let message of messageService.messages'>
 {{message}}
</div>

</div>
```

Retour sur le code : messages.component.css

```
/* MessagesComponent's private CSS styles */
h2 {
  color: red;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
}
body, input[text], button {
  color: crimson;
  font-family: Cambria, Georgia;
}
```

```
button.clear {
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
  cursor: hand;
}
```

```
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee;
  color: #aaa;
  cursor: auto;
}
button.clear {
  color: #888;
  margin-bottom: 12px;
}
```

Retour sur le code

app.module.ts :

```
1.  import { BrowserModule } from '@angular/platform-browser';
2.  import { NgModule } from '@angular/core';
3.  import { FormsModule } from '@angular/forms';
4.  import { AppComponent } from './app.component';
5.  import { HeroesComponent } from './heroes/heroes.component';
6.  import { HeroDetailComponent } from './hero-detail/hero-detail.component';
7.  import { MessagesComponent } from './messages/messages.component';
8.  @NgModule({
9.    declarations: [
10.     AppComponent,
11.     HeroesComponent,
12.     HeroDetailComponent,
13.     MessagesComponent
14.   ],
15.   imports: [
16.     BrowserModule,
17.     FormsModule
18.   ],
19.   providers: [
20.     // no need to place any providers due to the `providedIn` flag...],
21.   bootstrap: [ AppComponent ]})
22. export class AppModule { }
```

app.component.html

```
<h1>{{title}}</h1>
<app-heroes></app-heroes>
<app-messages></app-messages>
```

Angular Review

Binding	
One Way Binding	`<h1>{{pageTitle}}</h1>`
Two Way Binding	`<input [(ngModel)]="customer.FirstName">`
Property Binding	`<img [src]="customer.imageUrl">`
Attribute Binding	`<button [attr.aria-label]="ok">Ok</button>`
Class Binding	`<div [class.Selected]="Selected">Selected</div>`
ngClass	`<div [ngClass]="setClasses()"> {{customer.name}} </div>`
Style Binding	`<button [style.color]="isSelected ? 'red' : 'white'">`
ngStyle	`<div [ngStyle]="setStyles()"> {{customer.name}} </div>`
Component Binding	`<customer-detail [customer]="currentCustomer"> <customer-detail>`
Directive Binding	`<div [ngClass] = "{selected: isSelected}">Customer</div>`
Event Binding	`<button (click)="save()">Save</button>`
$event	`<input [value]="customer.name" (input)="customer.name=$event.target.value">`

Structural Directives	
*ngIf	`<div *ngIf="currentCustomer"> Selected {{currentCustomer.Name}} </div>`
*ngFor	`<ul> <li *ngFor="let customer of customers"> {{ customer.name }} </li> </ul>`
*ngSwitch	`<div [ngSwitch]="orderStatus"> <template [ngSwitchCase]="purchased"></template> <template [ngSwitchCase]="shipped"></template> <template [ngSwitchDefault]></template> </div>`

Component with Inline Template

```
import { Component } from '@angular/core';
@Component({
  moduleId: module.id,
  selector: 'customer',
  template: `
    <h3>{{customer.name}}</h3>
  `
})
```

Component with Inline Template (cont)

```
export class CustomerComponent {
  customer = { id: 100, name: 'John Smith' };
}
```

Component Linked Template

```
import { Component } from '@angular/core';
@Component({
  moduleId: module.id,
  selector: 'customer',
  templateUrl: 'customer.component.html',
  styleUrls: ['customer.component.css']
})
export class CustomerComponent {
  customer = { id: 100, name: 'John Smith' };
}
```

Pipes	
Upper Case	`<p>{{customer.name | uppercase}}</p>`
Lower Case	`<p>{{customer.name | lowercase}}</p>`
Date	`<p>{{orderDate | date:'medium'}}</p>`
Date Format	`<p>{{orderDate | date:'yMMMd'}}</p>`
Currency	`<p>{{price | currency}}</p>`
Percent	`<p>{{taxes | percent:'1.1-1'}}</p>`
Number	`<p>value | number:'1.1-2'}}</p>`
JSON Debugging	`<pre>{{Customer | json}}</pre>`

Forms Angular Review

Introduction

A form creates a cohesive, effective, and compelling data entry experience. An Angular form coordinates a set of data-bound user controls, tracks changes, validates input, and presents errors.

Form

```
<form>
  .... tags that include all input elements
</form>
```

All forms are placed within the HTML form tags

Standard Input Types

Text Input	<input type="text">
Email Input	<input type="email">
Password Input	<input type="password">
Dropdown Selection	<select> <option value="volvo">Volvo</option> <option value="saab">Saab</option> <option value="opel">Opel</option> <option value="audi">Audi</option> </select>
Multi Selection	<select multiple> <option value="volvo">Volvo</option> <option value="saab">Saab</option> <option value="opel">Opel</option> <option value="audi">Audi</option> </select>
Checkbox	<input type="checkbox">
Radio Control	<input type="radio">
Numeric Input	<input type="number">
Date	<input type="date">
Multiline Input	<textarea rows="4" cols="50"></textarea>

Angular 2 Form - Elements

FormGroup	A FormGroup aggregates the values of each child FormControl into one object, with each control name as the key
FormControl	Tracks the value and validation status of an individual form control. It is one of the three fundamental building blocks of Angular forms
FormArray	Tracks the value and validity state of an array of FormControl instances. A FormArray aggregates the values of each child FormControl into an array
FormBuilder	Creates an AbstractControl from a user-specified configuration. It is essentially syntactic sugar that shortens the new FormGroup(), new FormControl(), and new FormArray() boilerplate that can build up in larger form

Requires use of FormModule

Reactive Form Names

formGroupName	Used to reference a group of elements
formControlName	Similar to ngModel reference to a name but simpler from a naming convention perspective
formArrayName	Syncs a nested FormArray to a DOM element.

Requires the use of the ReactiveFormsModule Module

Handling Submission Event

```
<form (ngSubmit)="onSubmit()">
...
</form>
```

Standard Validation

Mandatory	Validators.required
Minimum Length	Validator.minLength(size)
Maximum Length	Validators.maxLength(size)
Pattern Match	Validators.pattern("regEx")

Custom Validators

```
function {name}(control : FormControl) : {[s: string] : boolean} {

.... function body....
pass return a null
fail return an object of type {key : true}
}
```

Displaying Validator Failures

```
<label for="name">Name</label>
        <input type="text" class="form-control" id="name"
               required
               [(ngModel)]="model.name" name="name"
               #name="ngModel" >
        <div [hidden]="name.valid || name.pristine"
             class="alert alert-danger">
          Name is required
        </div>
```

Pipes Angular Review

Overview

currency	Supported
date	Supported
uppercase	Supported
json	Supported
limitTo	Supported
lowercase	Supported
number	Not
orderBy	Not
filter	Not
async	Supported
decimal	Supported
percent	Supported

A Pipe is a filter applied over a angular expression. This is denoted by the | (pipe symbol)

Example Code

```
<!-- Sep 1, 2015 -->
<p>{{date | date:'mediumDate'}}</p>
<!-- September 1, 2015 -->
<p>{{date | date:'yMMMMd'}}</p>
<!-- 3:50 pm -->
<p>{{date | date:'shortTime'}}</p>
```

Currency

Usage	expression | currency[:currencyCode[:symbolDisplay[:digitInfo]]]
currencyCode	ISO 4217 Compliant, eg USD, EUR

```
class CurrencyPipe {
transform(value: any, currencyCode?: string, symbolDisplay?: boolean, digits?: string) : string
}
```

Date

Usage	expression | date[:format]
Year	y or yy
Month	M or MM
Day	d or D
Weekday	EEE or EEEE

Date (cont)

Hour	j or jj
Hour 12	h or hh
Hour 24	H or HH
Minute	m or mm
Second	s or ss
Timezone	z

```
class DatePipe {
transform(value: any, pattern?: string) : string
supports(obj: any) : boolean
}
```

Uppercase

Usage	item | uppercase

Json

Usage	item | json

Transforms any input value using JSON.stringify. Useful for debugging.

lowercase

Usage	item| lowercase

async

Usage	item | async

The async pipe subscribes to an Observable or Promise and returns the latest value it has emitted. When a new value is emitted, the async pipe marks the component to be checked for changes.

decimal

Usage	expression | number[:digitInfo]
digitInfo	{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}

minIntegerDigits is the minimum number of integer digits to use. Defaults to 1. minFractionDigits is the minimum number of digits after fraction. Defaults to 0. maxFractionDigits is the maximum number of digits after fraction. Defaults to 3.

percent

Usage	expression | percent[:digitInfo]
	see decimal for usage

Custom Pipes

```
import {Component, View, bootstrap, Pipe, PipeTransform} from 'angular2/angular2';
@Pipe({
  name: 'tempConvert'
})
class TempConvertPipe implements PipeTransform {
  transform(value: number, args: any[]) {
    if(value && !isNaN(value) && args[0] === 'celsius') {
      var temp = (value - 32) * 5/9;
      var places = args[1];
      return temp.toFixed(places) + ' C';
    }
    return;
  }
}
```

Using non standard pipes

import	import as normal
component usage	pipes: [ExponentialStrengthPipe]

TypeScript Review

Types

String	let customerName: string= "John Doe";
Number	let price: number = 19.95;
Boolean	let shipped: boolean = false;
Date	let orderDate: Date = new Date(2017, 2, 9);
Any	let something: any = "Can be anything";
Enum	enum Color {Red, Green, Blue};
Array	let cards: string[] = ['Visa', 'MasterCard'];
Null	let orderId: number = null;
Tuple	let stateTaxRates: [string, number];

Void	function log(msg: string): void {
	console.log(msg);
	}

Const	const lives: number = 99;

Classes

```
class OrderLogic {
    constructor(public order: IOrder) { }
    getOrderTotal(): number {
        let sum: number = 0;
        for (let orderDetail of
this.order.orderDetails)
        {
            sum += orderDetail.price;
        }
        return sum;
    }
}
```

Abstract Classes

```
abstract class Person {
  name: string;
  monthlySalary: number;
  monthlyBenefits: number;
  abstract calcSalary(): number;
}
```

Inheritance and Implementing Interfaces

```
interface IGPS {
  getLocation() number;
}
interface ISelfDrive extends IGPS {
  drive(latitude: number, longitude: number,
elevation: number) : void;
}
class Vehicle {
  make: string;
  model: string;
  year: number;
}
class FlyingCar extends Vehicle implements ISelfDrive
{
   hasGps: boolean;
  drive(latitude: number, longitude: number,
elevation: number) {
  }
  getLocation(): number {
  }
}
```

Usage

Installing TypeScript npm	npm install -g typescript
Compiling TypeScript	tsc somefile.ts
TypeScript Docs	TypeScriptLang.org
Type Definition Files	DefinatelyTyped.org

Scope/Modifiers

Public (default)	public firstName: string;
Protected	protected inventory: number;
Private	private outOfStock: boolean;
Read Only	readonly pi: number = 3.14159;
Static	static log(msg: string) { console.log(msg) };

NodeJs Review

Getting Started

Node.JS is evented I/O for V8 JavaScript. It is asynchronous in nature, with handlers to I/O and other events being function callbacks. It is particularly suited to distributed computing environments with high concurrency.

node script.js	Run script
npm install <package>	Install package with npm

Globals

var variable	Initialize variable local to module
process	Properties & methods for current process
console	Used to print to stdout & stderr
require()	To require modules
require.resolve	Lookup location of module
require.paths	Paths to search when requiring modules
_filename	File name of script being executed
_dirname	Directory name of script being executed
module	Reference to current module
setTimeout(), clearTimeout()	
setInterval(), clearInterval()	

Modules

stdio

Object for printing to stdout and stderr, like in a browser.

console.log(string)	Print to stdout with newline
console.error(string)	Same as console.log() but to stderr
console.time(label)	Set time marker
console.timeEnd(label)	Finish timer, record output
console.trace()	Print stack trace to stderr of current position

Process

Global object. Instance of *EventEmitter*

Events:

process.on(SIGNAL, callback)	Signal events emitted when process receives a signal
exit	Process is about to exit
uncaughtException	Exception bubbled back to event loop

Properties: **process.stdout**, **processs.stderr**, **process.stdin**, **process.argv**, **process.env**, **process.pid**

Modules (continued)

util

Useful methods:

util.debug(message)	Synchronous console.error(message)
util.log(message)	Print timestamped message to stdout

events

Callback functions executed when events occur are *listeners*. *emitter* is an instance of *EventEmitter*.

emitter.on(event, listener)	Add a listener for *event*
emitter.once(event, listener)	Fire listener once
emitter.removeListener(event,listener)	Remove a listener
emitter.removeAllListeners(event)	Remove all listeners
emitter.emit(event, [[arg1], [arg2], [...]])	Execute listeners for this event with supplied args

net

Asynchronous network wrapper for creating streams.

net.Server:

net.createServer([options], [connectionListener])	Create TCP server. Returns *net.Server*
server.listen(port, [host], [callback])	Bind on host:port. *listener* is executed when bound
server.listenFD(fd)	Listen on file descriptor *fd*
server.close()	Stop accepting new connections

net.Socket:

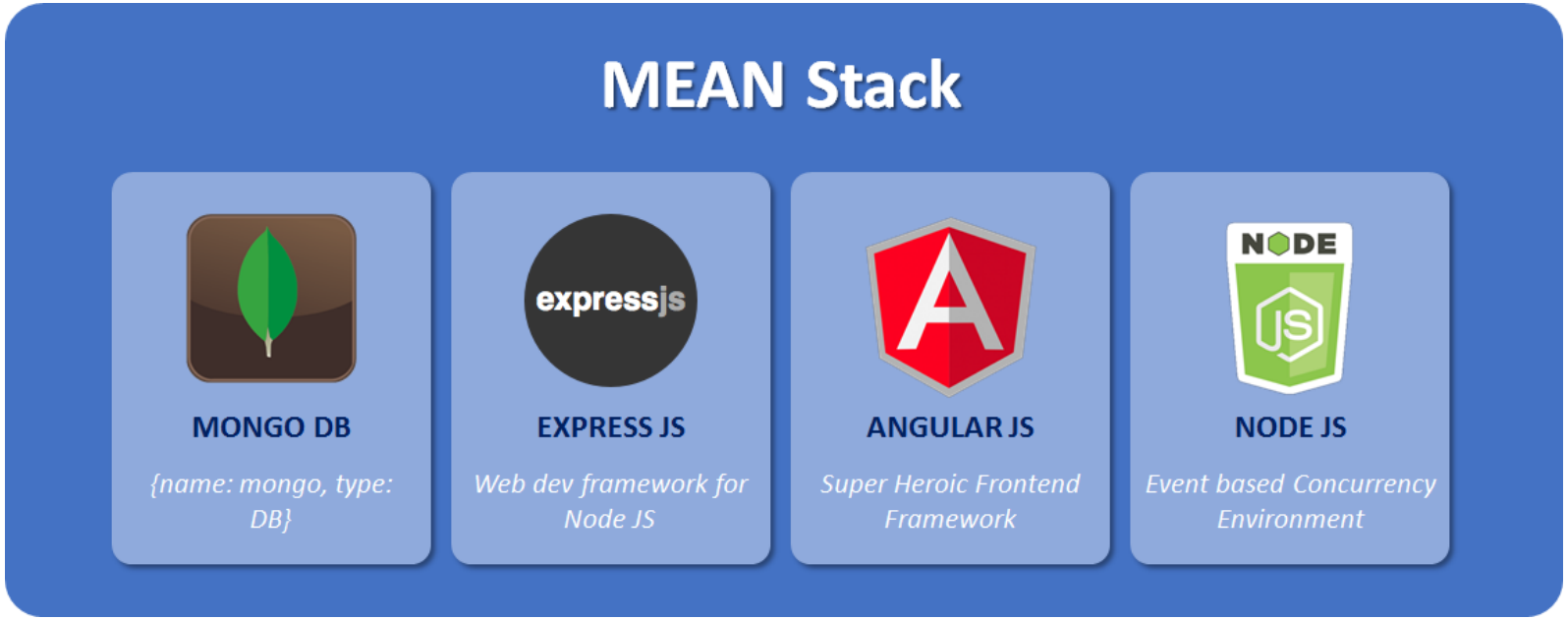
new net.Socket([{fd: file descriptor, type: socket type, allowHalfOpen: bool}])	Construct new socket object
socket.connect(port, [host], [callback])	Open connection to socket
socket.bufferSize	Number of characters in internal write buffer
socket.write(data, [encoding], [callback])	Send data on socket
socket.end()	Send FIN packet
socket.pause()	Pause reading of data

event

Emitted when:

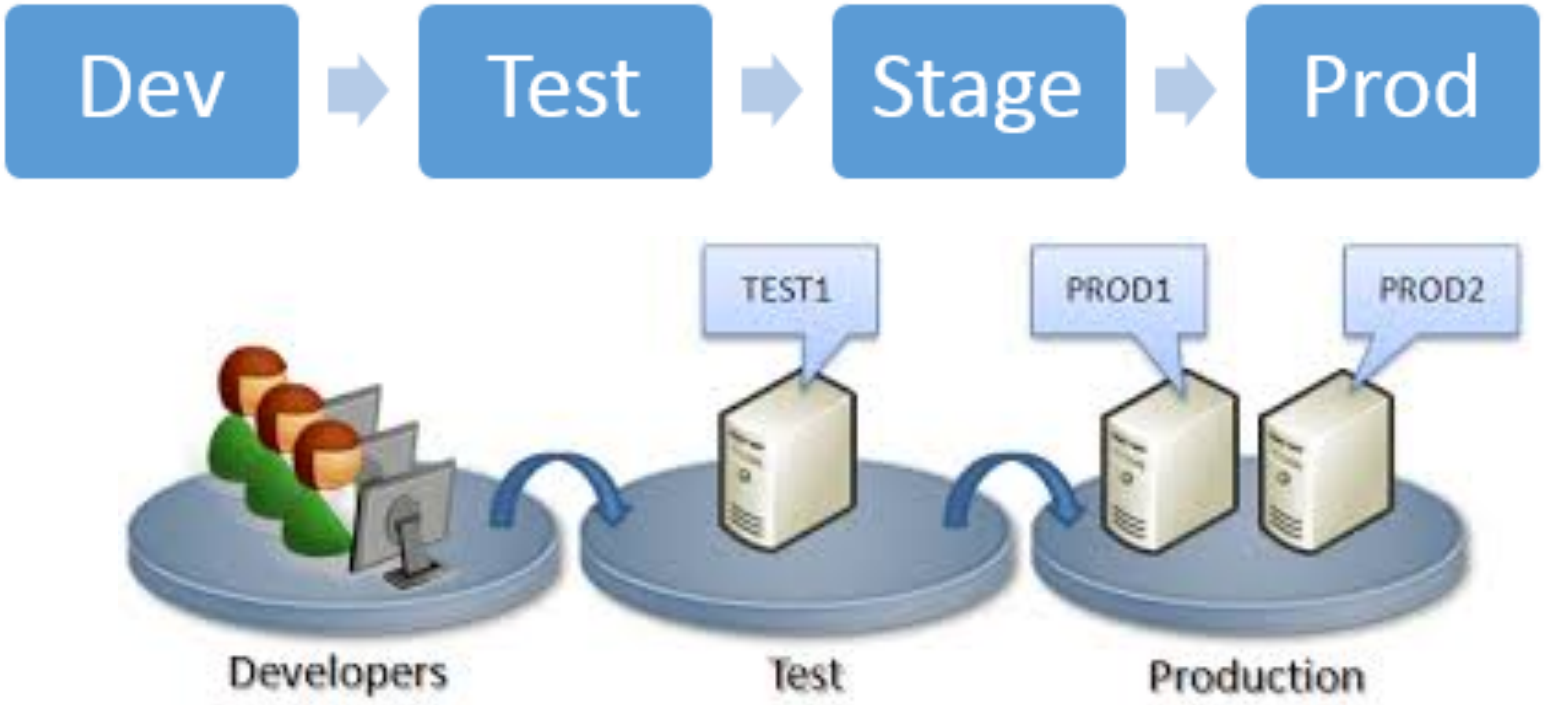
connect	Socket connection established
data	Data is received
end	Other end of socket sent FIN packet
timeout	Timed out from inactivity
drain	Write buffer has become empty
error	Error has occurred. close event emitted after
close	Socket fully closed

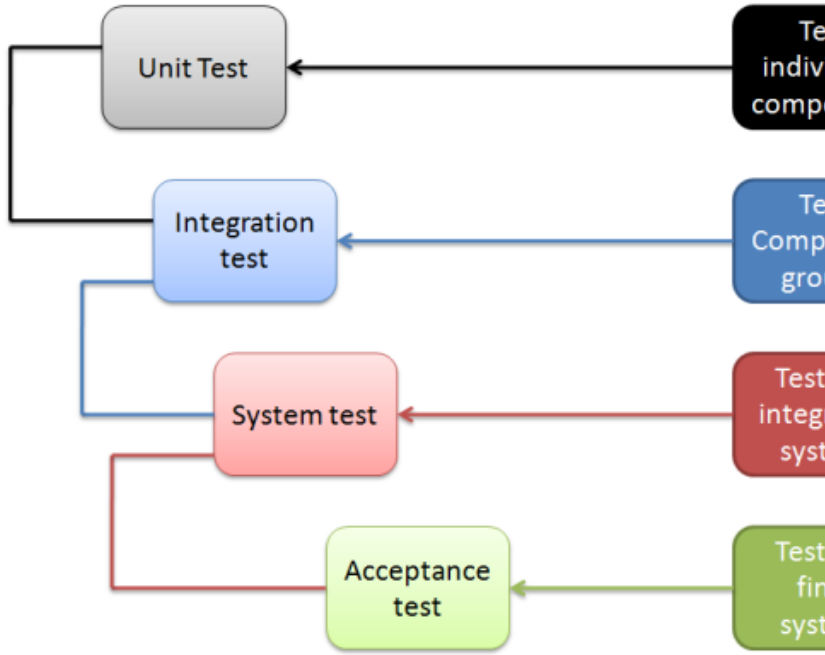
MEAN Stack
Review



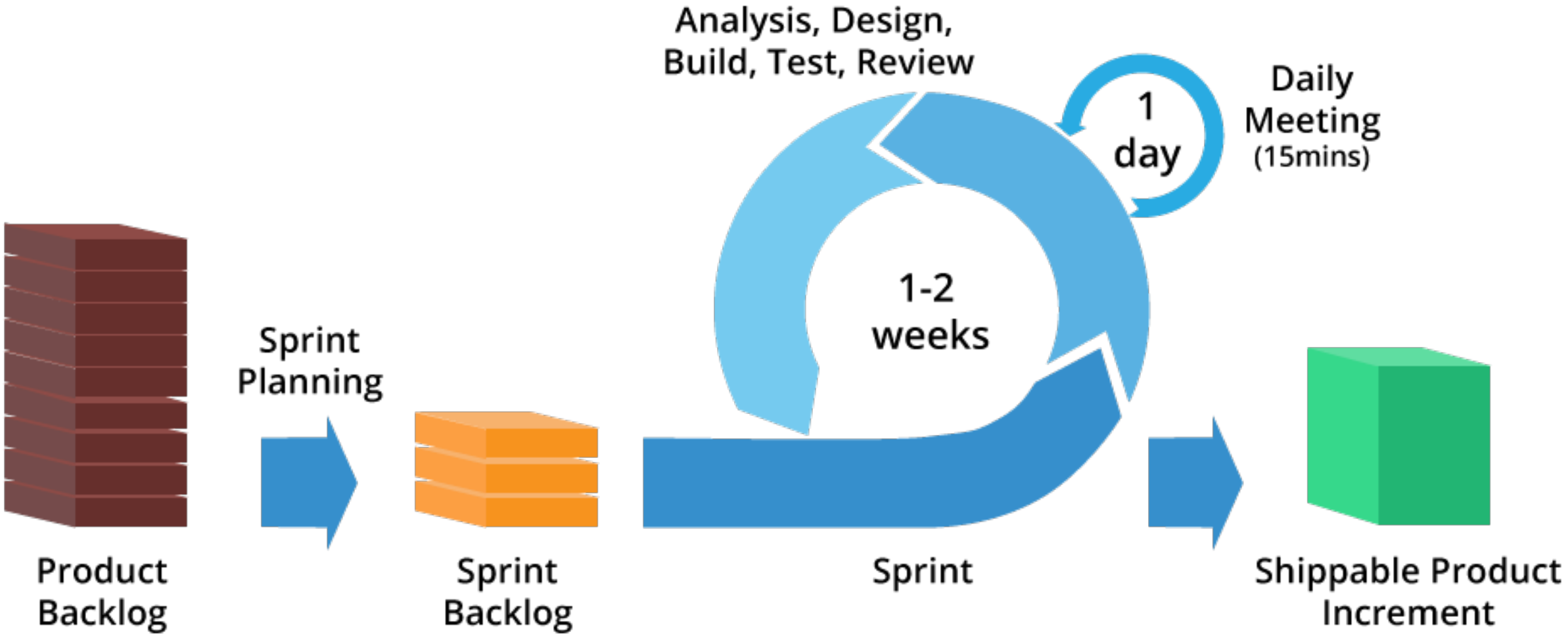
Environnements

Review

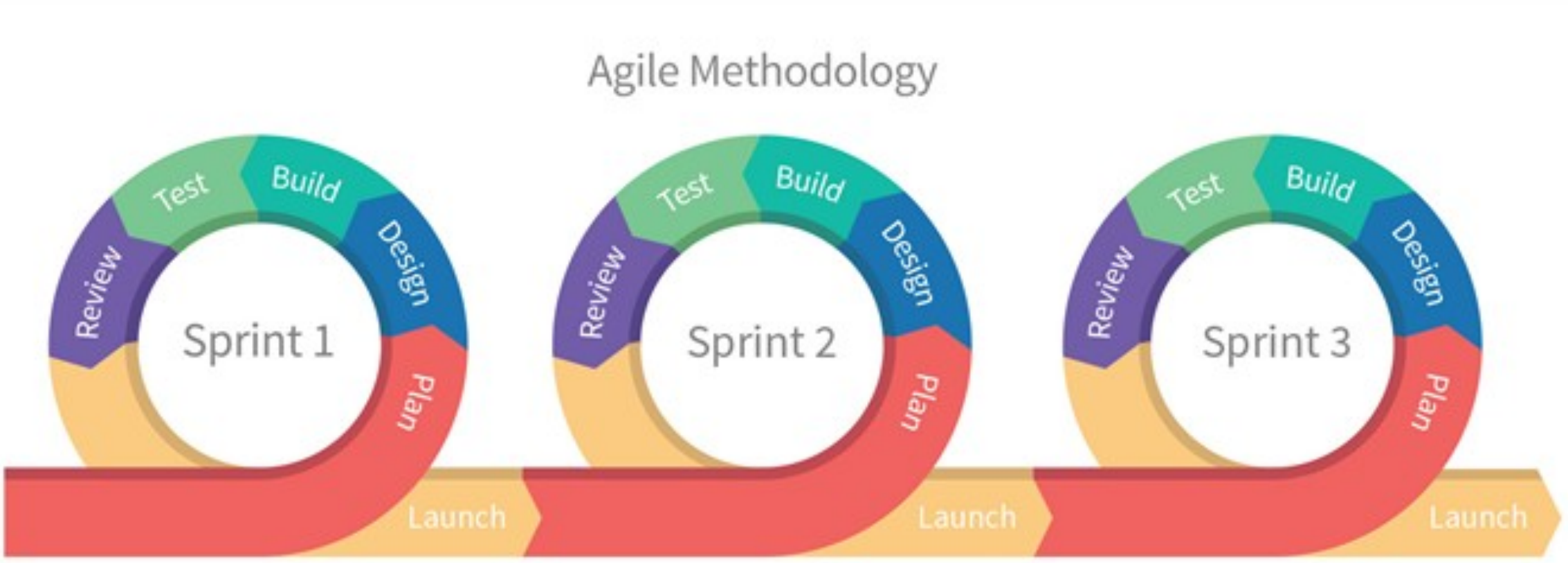




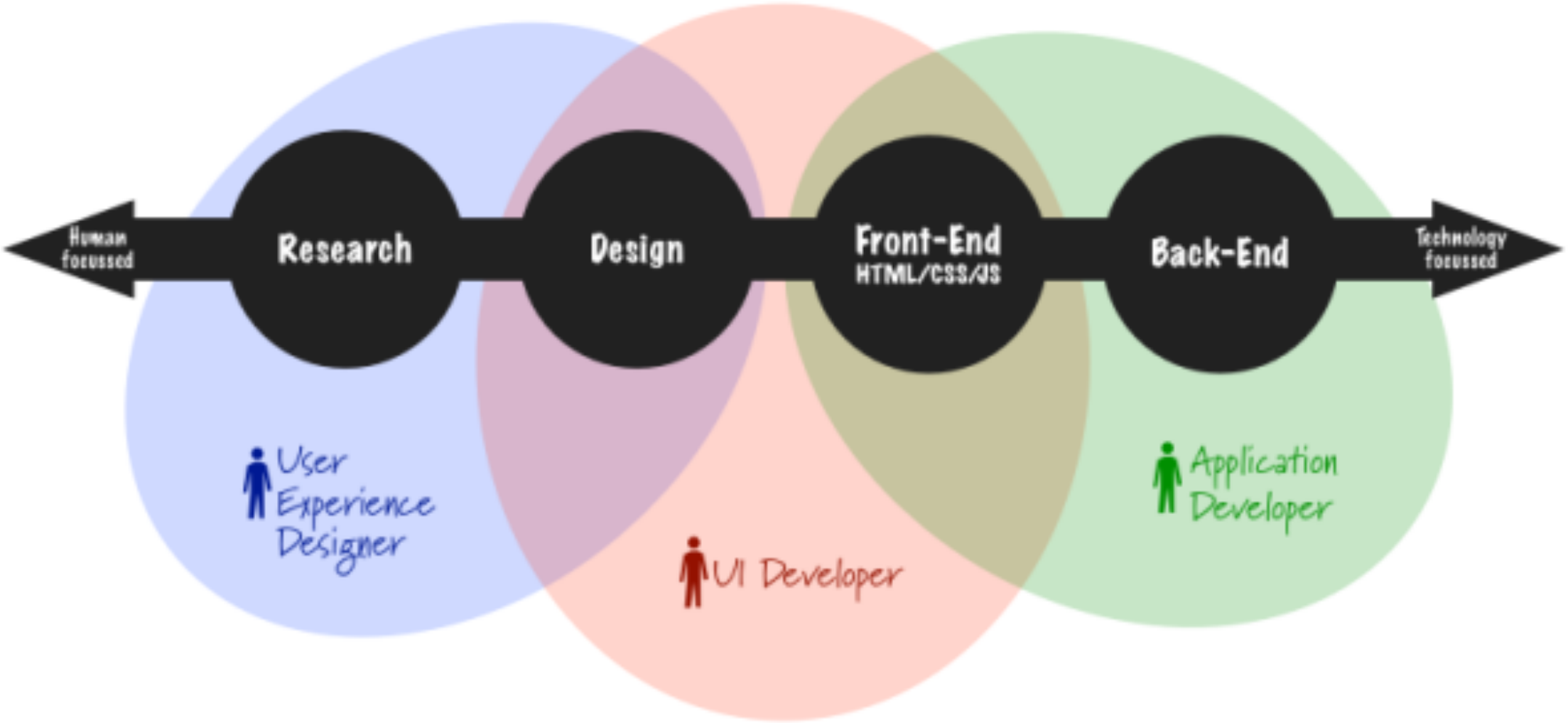
Agile Development



Agile
Development



De l’Homme à
La Machine



Parenthèse: les design patterns

En informatique, et plus particulièrement en développement logiciel, **un patron de conception** (souvent appelé **design pattern**) est un arrangement caractéristique de modules, reconnu comme **bonne pratique** en réponse à un p**roblème de conception d'un logiciel**. Il décrit une solution standard, utilisable dans la conception de différents logiciels. *Exemple : Dependency Injection*

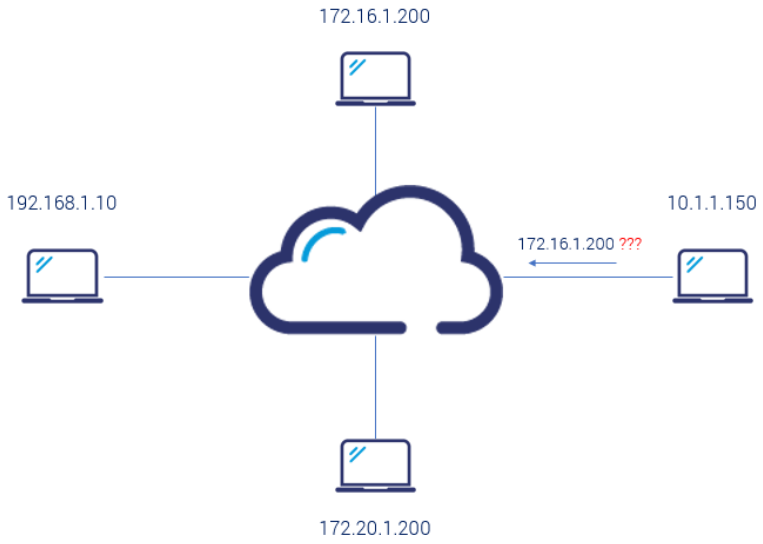
Creational patterns [edit]

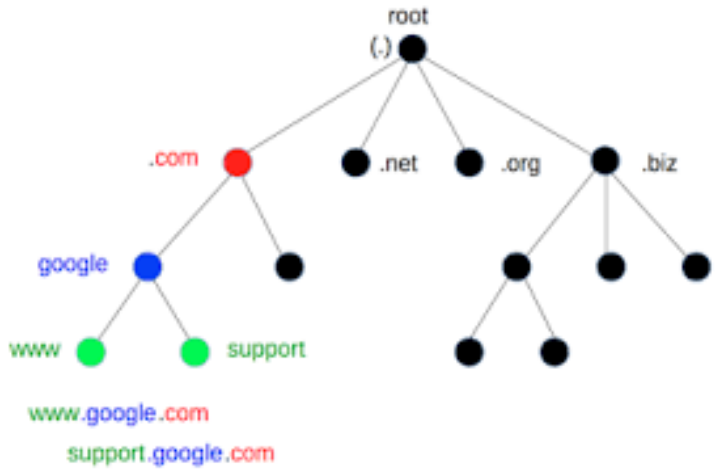
Name	Description
Abstract factory	Provide an interface for creating *families* of related or dependent objects without specifying their concrete classes.
Builder	Separate the construction of a complex object from its representation, allowing the same construction process to create various representations.
Dependency Injection	A class accepts the objects it requires from an injector instead of creating the objects directly.
Factory method	Define an interface for creating a *single* object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
Lazy initialization	Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed. This pattern appears in the GoF catalog as "virtual proxy", an implementation strategy for the Proxy pattern.
Multiton	Ensure a class has only named instances, and provide a global point of access to them.
Object pool	Avoid expensive acquisition and release of resources by recycling objects that are no longer in use. Can be considered a generalisation of connection pool and thread pool patterns.
Prototype	Specify the kinds of objects to create using a prototypical instance, and create new objects from the 'skeleton' of an existing object, thus boosting performance and keeping memory footprints to a minimum.
Resource acquisition is initialization (RAII)	Ensure that resources are properly released by tying them to the lifespan of suitable objects.
Singleton	Ensure a class has only one instance, and provide a global point of access to it.

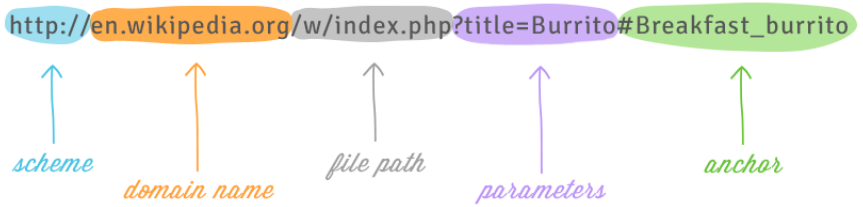
Behavioral patterns [edit]

Name	Description
Blackboard	Artificial intelligence pattern for combining disparate sources of data (see blackboard system)
Chain of responsibility	Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
Command	Encapsulate a request as an object, thereby allowing for the parameterization of clients with different requests, and the queuing or logging of requests. It also allows for the support of undoable operations.
Interpreter	Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.
Iterator	Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
Mediator	Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it allows their interaction to vary independently.
Memento	Without violating encapsulation, capture and externalize an object's internal state allowing the object to be restored to this state later.
Null object	Avoid null references by providing a default object.
Observer or Publish/subscribe	Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically.
Servant	Define common functionality for a group of classes. The servant pattern is also frequently called helper class or utility class implementation for a given set of classes. The helper classes generally have no objects hence they have all static methods that act upon different kinds of class objects.
Specification	Recombinable business logic in a Boolean fashion.
State	Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
Strategy	Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Le Routage





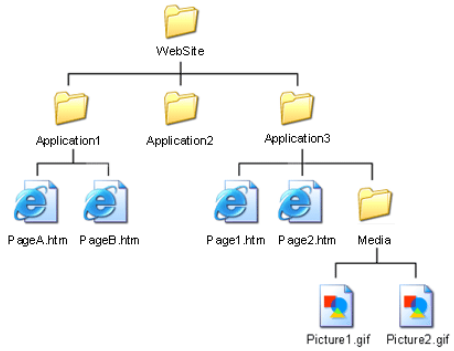


Les moyens classiques de navigation entre pages :

- Entrer URL dans la barre de navigation ex: www.angular.io et être redirigé vers la page correspondante
- Cliquer sur des liens dans une page et être redirigé vers la page correspondante
- Cliquer sur les boutons “précédent” et “suivant” que le navigateur propose

Le Routeur Angular s’inspire de ces 3 moyens pour gérer le routage

Le Routage



Il est possible de créer une application sur laquelle **différents affichages seront vus sous des URLs différentes** (donc on peut simuler le fait que notre application possède différentes pages sous différentes adresses web). C'est à ça que sert **le routage dans une application Angular** et tous les outils sont offerts pour gérer cet aspect.

En fonction du contenu de l'URL on va afficher les views correspondantes ! Pas besoin d'aller demander au serveur de nous envoyer la bonne Vue ==> SPA. Au clique sur les liens dans l'application, on affichera les bonnes vues. On peut afficher la vue (view, pas page) après event click.. Ou un autre stimulus …

Le routeur renseigne dans l'historique du navigateur les mouvements de page en page : **Browser's History Journal**
Ceci va permettre d'utiliser les boutons "précédent" et "suivant". On mimique le fonctionnement classique. En effet, l'utilisateur n'a pas à savoir SPA. Sentiment de réactivité et efficacité du site ++ La user exp (UX) est donc favorisée.

ROUTER IMPORTS

Dans app.module.ts ! Angular Router est optionnel donc
Nob Inclut par défaut

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

Le Routage : configuration

Une application Angular routée ne peut avoir qu'**une seule et unique instance** (singleton) du **service ROUTER dans app.module**

```
const appRoutes: Routes = [
 { path: 'hero/:id',
   component: HeroDetailComponent
  },
 {
   path: 'heroes',
   component: HeroListComponent,
   data: { title: 'Heroes List' }
 },
 { path: '',
   redirectTo: '/heroes',
   pathMatch: 'full'
 },
 { path: '**', component: PageNotFoundComponent }
];

@NgModule({
 imports: [
   RouterModule.forRoot(
     appRoutes,
     { enableTracing: true } // <-- debugging purposes only
   )
   // other imports here
 ],
 ...
})
export class AppModule { }
```

appRoutes indique le PLAN de Navigation

Pour configurer ce router (plan de navigation)
Il va falloir le passer en paramètre de la fonction
forRoot qui se trouve dans le RouterModule !
Cette fonction est mise en imports !!

1 URL —> 1 Component

'hero/:id' —> ex hero/12 —> employé d'identifiant 12

data property : stockage de données tels que
Titre de la page, static data …

The ** path in the last route is a wildcard.

L'ORDRE EST IMPORTANT CAR
STRATEGIE : first-match wins !!

Déclarer Les plus spécifiques d'abord puis les moins spécifiques !!

In the configuration, routes with a static path are listed first,
followed by an empty path route, that matches the default route. The wildcard route comes last because it matches every URL and should be selected only if no other routes are matched first.

enableTracing : historique de navigation pour debug après

Le Routage

Router-outlet : Un Placeholder pour l'endroit ou le routeur doit injecter les composants routés

```
<router-outlet></router-outlet>
<!-- Routed components go here —>
```

when the browser URL for this application becomes /heroes, the router matches that URL to the route path /heroes
and displays the HeroListComponent as a sibling element to the RouterOutlet that you've placed in the host component's template.

Configuration faite comment se fait la navigation ? **Router links**

src/app/app.component.html

```
<h1>Angular Router</h1>
<nav>
  <a routerLink="/dashboard" routerLinkActive="active">Dashboard</a>
  <a routerLink="/employees" routerLinkActive="active">Employees</a>
</nav>
<router-outlet></router-outlet>
```

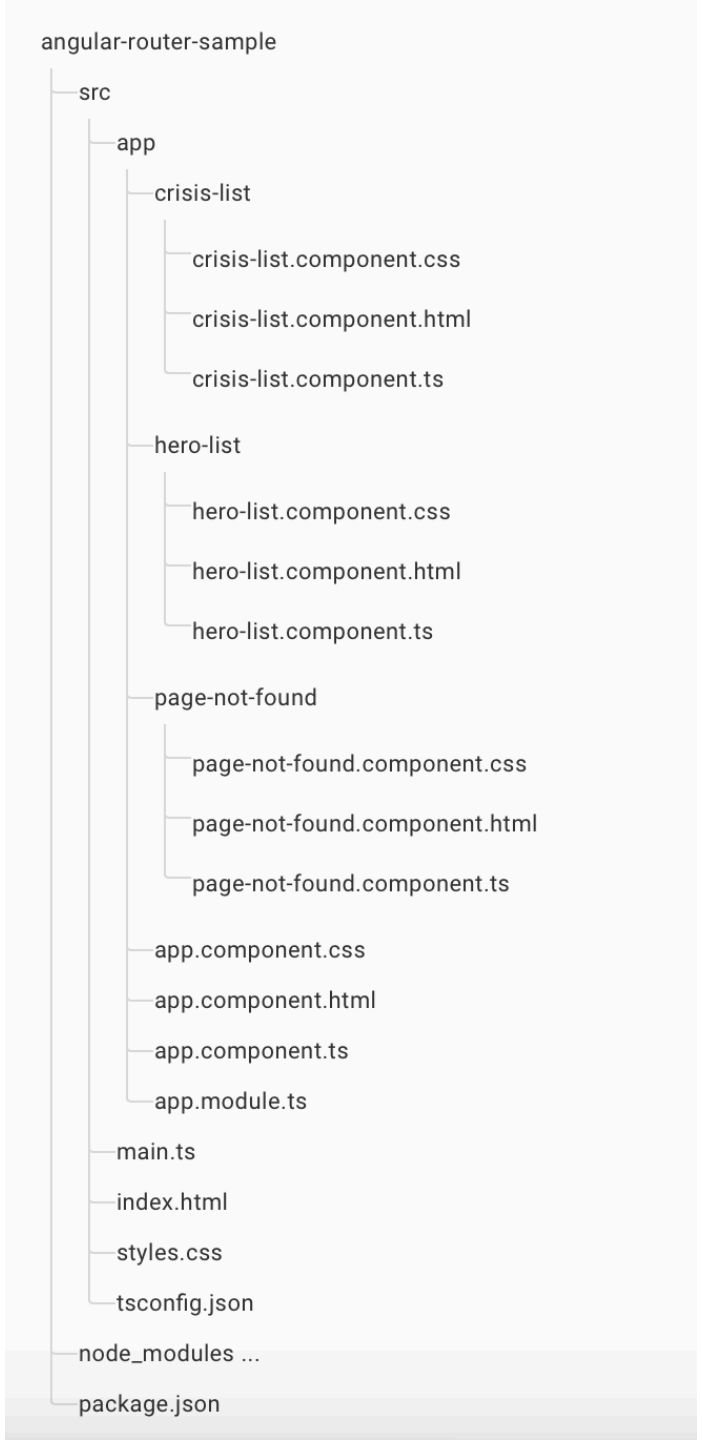
The navigation paths are fixed, so you can assign a string to the routerLink (a "one-time" binding)

Each **ActivatedRoute** in the **RouterState** provides methods to traverse up and down the route tree
to get information from parent, child and sibling routes : history is kept (btw : Machine Learning ..)

Le Routage

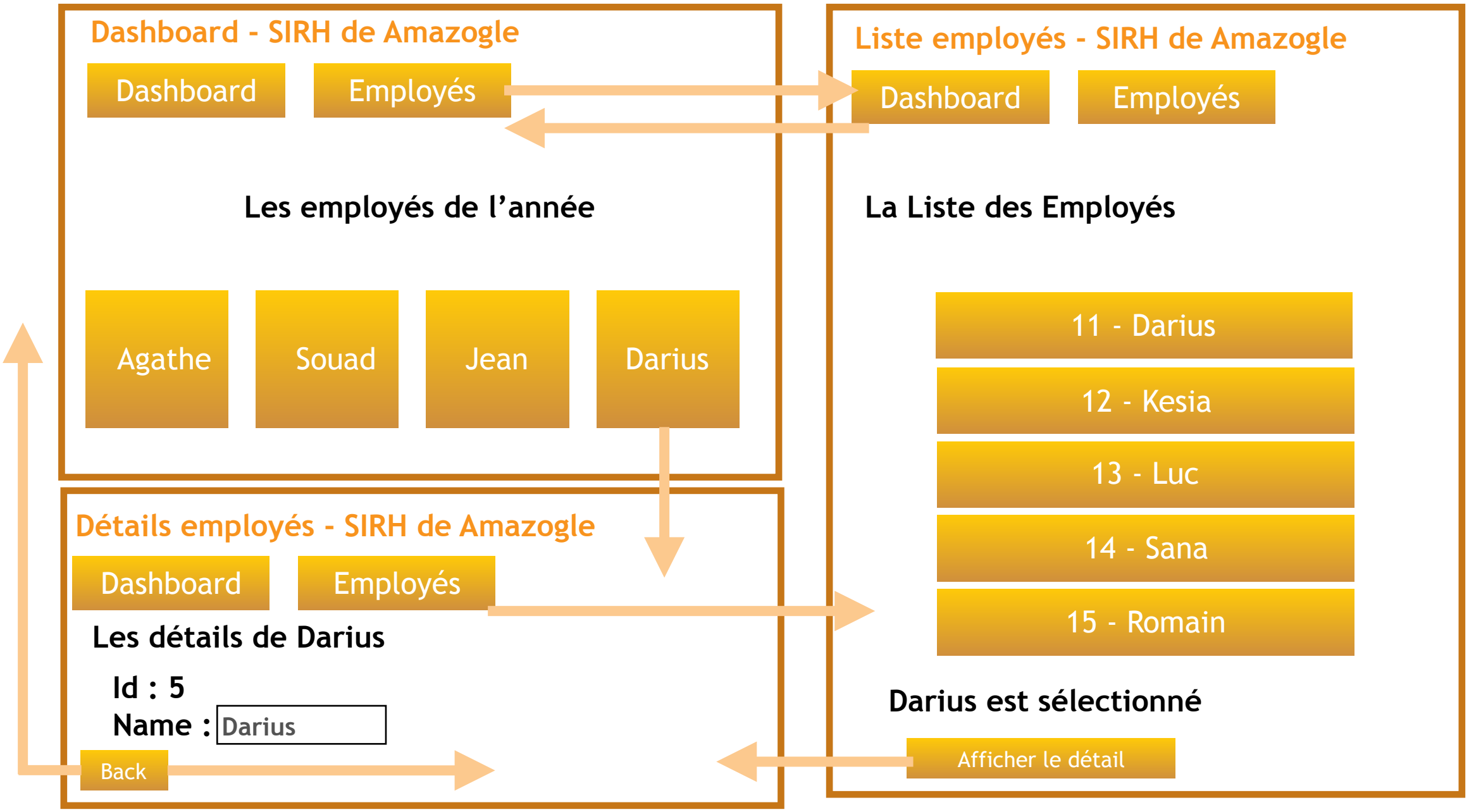
`Router`	Displays the application component for the active URL. Manages navigation from one component to the next.
`RouterModule`	A separate NgModule that provides the necessary service providers and directives for navigating through application views.
`Routes`	Defines an array of Routes, each mapping a URL path to a component.
`Route`	Defines how the router should navigate to a component based on a URL pattern. Most routes consist of a path and a component type.
`RouterOutlet`	The directive (`<router-outlet>`) that marks where the router displays a view.
`RouterLink`	The directive for binding a clickable HTML element to a route. Clicking an element with a `routerLink` directive that is bound to a *string* or a *link parameters array* triggers a navigation.
`RouterLinkActive`	The directive for adding/removing classes from an HTML element when an associated `routerLink` contained on or inside the element becomes active/inactive.
`ActivatedRoute`	A service that is provided to each route component that contains route specific information such as route parameters, static data, resolve data, global query params, and the global fragment.
`RouterState`	The current state of the router including a tree of the currently activated routes together with convenience methods for traversing the route tree.
Link parameters array	An array that the router interprets as a routing instruction. You can bind that array to a `RouterLink` or pass the array as an argument to the `Router.navigate` method.
Routing component	An Angular component with a `RouterOutlet` that displays views based on router navigations.

Le Routage


angular-router-sample
src
app
crisis-list
crisis-list.component.css
crisis-list.component.html
crisis-list.component.ts
hero-list
hero-list.component.css
hero-list.component.html
hero-list.component.ts
page-not-found
page-not-found.component.css
page-not-found.component.html
page-not-found.component.ts
app.component.css
app.component.html
app.component.ts
app.module.ts
main.ts
index.html
styles.css
tsconfig.json
node_modules ...
package.json

En Pratique

#E0AF44



Comprendre notre workspace

WORKSPACE CONFIG FILES	PURPOSE
`.editorconfig`	Configuration for code editors. See EditorConfig.
`.gitignore`	Specifies intentionally untracked files that Git should ignore.
`angular.json`	CLI configuration defaults for all projects in the workspace, including configuration options for build, serve, and test tools that the CLI uses, such as TSLint, Karma, and Protractor. For details, see Angular Workspace Configuration.
`node_modules`	Provides npm packages to the entire workspace.
`package.json`	Configures npm package dependencies that are available to all projects in the workspace. See npm documentation for the specific format and contents of this file.
`package-lock.json`	Provides version information for all packages installed into `node_modules` by the npm client. See npm documentation for details. If you use the yarn client, this file will be yarn.lock instead.
`tsconfig.json`	Default TypeScript configuration for apps in the workspace, including TypeScript and Angular template compiler options. See TypeScript Configuration.
`tslint.json`	Default TSLint configuration for apps in the workspace.
`README.md`	Introductory documentation.

Comprendre notre workspace

`app/`	Contains the component files in which your app logic and data are defined. See details in App source folder below.
`assets/`	Contains image files and other asset files to be copied as-is when you build your application.
`environments/`	Contains build configuration options for particular target environments. By default there is an unnamed standard development environment and a production ("prod") environment. You can define additional target environment configurations.
`browserlist`	Configures sharing of target browsers and Node.js versions among various front-end tools. See Browserlist on GitHub for more information.
`favicon.ico`	An icon to use for this app in the bookmark bar.
`index.html`	The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any `<script>` or`<link>` tags here manually.
`main.ts`	The main entry point for your app. Compiles the application with the JIT compiler and bootstraps the application's root module (AppModule) to run in the browser. You can also use the AOT compiler without changing any code by appending the `--aot` flag to the CLI `build` and `serve` commands.
`polyfills.ts`	Provides polyfill scripts for browser support.
`styles.sass`	Lists CSS files that supply styles for a project. The extension reflects the style preprocessor you have configured for the project.
`test.ts`	The main entry point for your unit tests, with some Angular-specific configuration. You don't typically need to edit this file.
`tsconfig.app.json`	Inherits from the workspace-wide `tsconfig.json` file.
`tsconfig.spec.json`	Inherits from the workspace-wide `tsconfig.json` file.
`tslint.json`	Inherits from the workspace-wide `tslint.json` file.

Comprendre notre workspace

App source folder

Inside the `src/` folder, the `app/` folder contains your app's logic and data. Angular components, templates, and styles go here. An `assets/` subfolder contains images and anything else your app needs. Files at the top level of `src/` support testing and running your app.

APP SOURCE FILES	PURPOSE
`app/app.component.ts`	Defines the logic for the app's root component, named `AppComponent`. The view associated with this root component becomes the root of the view hierarchy as you add components and services to your app.
`app/app.component.html`	Defines the HTML template associated with the root `AppComponent`.
`app/app.component.css`	Defines the base CSS stylesheet for the root `AppComponent`.
`app/app.component.spec.ts`	Defines a unit test for the root `AppComponent`.
`app/app.module.ts`	Defines the root module, named `AppModule`, that tells Angular how to assemble the application. Initially declares only the `AppComponent`. As you add more components to the app, they must be declared here.
`assets/*`	Contains image files and other asset files to be copied as-is when you build your application.

Dependencies

The packages listed in the `dependencies` section of `package.json` are essential to *running* applications.

The `dependencies` section of `package.json` contains:

- Angular packages: Angular core and optional modules; their package names begin `@angular/`.
- Support packages: 3rd party libraries that must be present for Angular apps to run.
- Polyfill packages: Polyfills plug gaps in a browser's JavaScript implementation.

To add a new dependency, use the `ng add` command.

Angular packages

The following Angular packages are included as dependencies in the default `package.json` file for a new Angular workspace. For a complete list of Angular packages, see the API reference.

PACKAGE NAME	DESCRIPTION
@angular/animations	Angular's animations library makes it easy to define and apply animation effects such as page and list transitions. For more information, see the Animations guide.
@angular/common	The commonly-needed services, pipes, and directives provided by the Angular team. The `HttpClientModule` is also here, in the `@angular/common/http` subfolder. For more information, see the HttpClient guide.
@angular/compiler	Angular's template compiler. It understands templates and can convert them to code that makes the application run and render. Typically you don't interact with the compiler directly; rather, you use it indirectly via `platform-browser-dynamic` when JIT compiling in the browser. For more information, see the Ahead-of-time Compilation guide.
@angular/core	Critical runtime parts of the framework that are needed by every application. Includes all metadata decorators, `Component`, `Directive`, dependency injection, and the component lifecycle hooks.
@angular/forms	Support for both template-driven and reactive forms. For information about choosing the best forms approach for your app, see Introduction to forms.
@angular/http	Angular's legacy HTTP client, which was deprecated in version 5.0 in favor of @angular/common/http.
@angular/platform-browser	Everything DOM and browser related, especially the pieces that help render into the DOM. This package also includes the `bootstrapModuleFactory()` method for bootstrapping applications for production builds that pre-compile with AOT.
@angular/platform-browser-dynamic	Includes providers and methods to compile and run the app on the client using the JIT compiler.
@angular/router	The router module navigates among your app pages when the browser URL changes. For more information, see Routing and Navigation.

Angular : JiT vs AoT

Angular propose deux types de compilation de templates : **JiT (Just-in-Time)** et **AoT (Ahead-of-Time)**.

Question d'optimisation ==> AOT —> Rendering + rapide, meilleure sécurité, pré-compilation en JS pré-affichage client

What is AOT?

Ahead-of-Time Compilation works Just like JIT (**Just-in-Time** Compilation) in fact there is only one Compiler in Angular.

The difference between AOT and JIT is a matter of timing and tooling

Angular offers two ways to compile your application:

1. Just-in-Time (JIT), which compiles your app in the browser at runtime.
2. Ahead-of-Time (AOT), which compiles your app at build time

JIT compilation is the default when you run the `ng build` (build only) or `ng serve` (build and serve locally) CLI commands:

```
ng build
ng serve
```

For AOT compilation, include the `--aot` option with the `ng build` or `ng serve` command:

```
ng build --aot
ng serve --aot
```

Déploiement dans Heroku

Setup Your Angular Application

```
ng new my-app
cd my-app
ng serve
```

Our basic angular app is ready and running — locally

Create GitHub Repo and Push it

Login to Github and create new repository.
TERMINAL

git remote add origin <new_github_repository_url>
git add .
git commit -m "initial commit"
git push -u origin master

L'application est sur GitHub

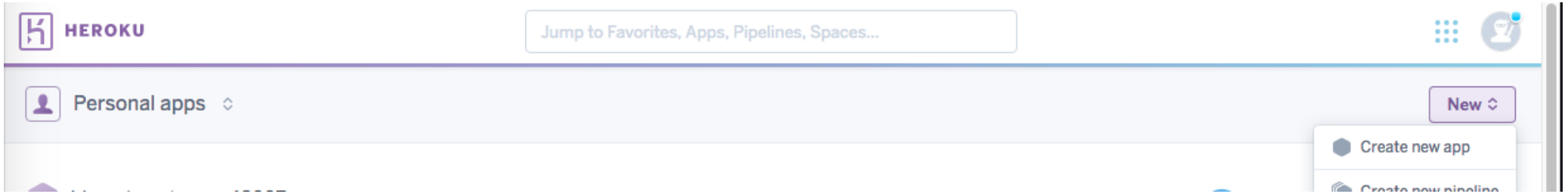
https://medium.com/@hellotunmbi/how-to-deploy-angular-application-to-heroku-1d56e09c5147

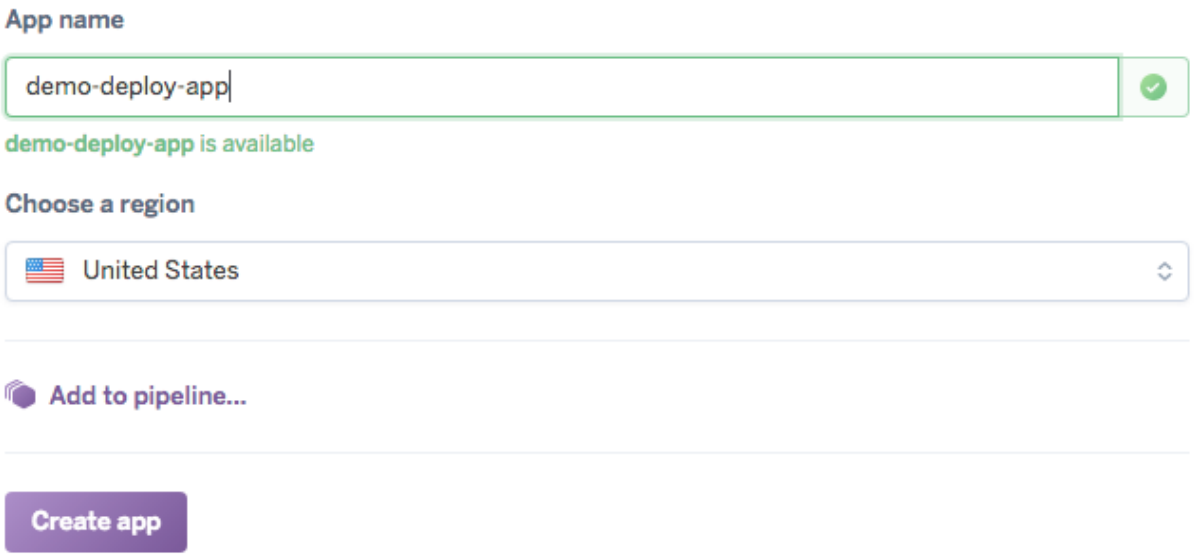
Déploiement dans Heroku

Setup Automatic Deployment from GitHub to Heroku

Advantage of this step is so that, once you push a change to your Github repository,
it automatically pushes the change to your codebase on heroku, which then takes effect live on the web.
This means, you'll only have to push your changes to Github and its done.

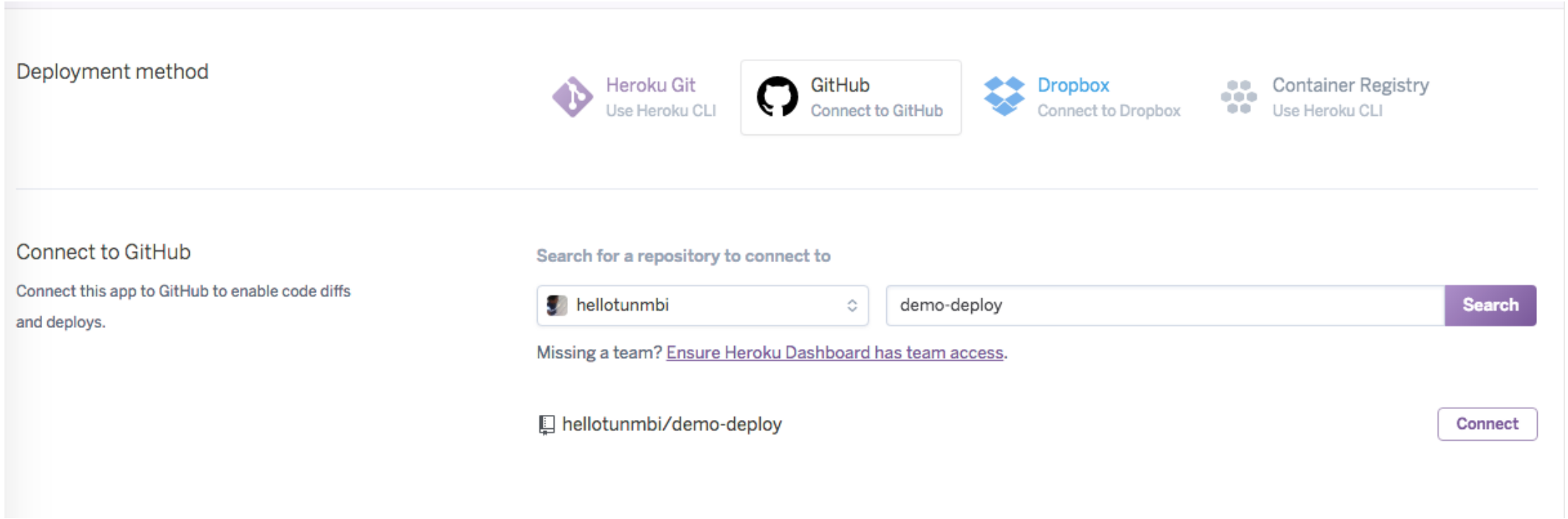
Create account on HEROKU it is free !





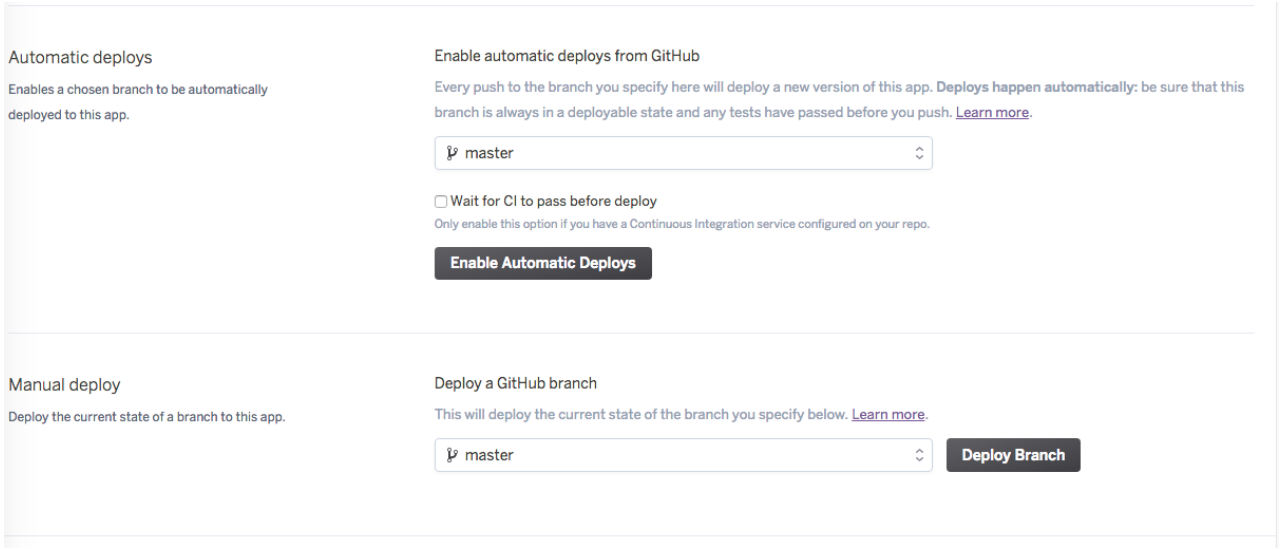
Déploiement dans Heroku

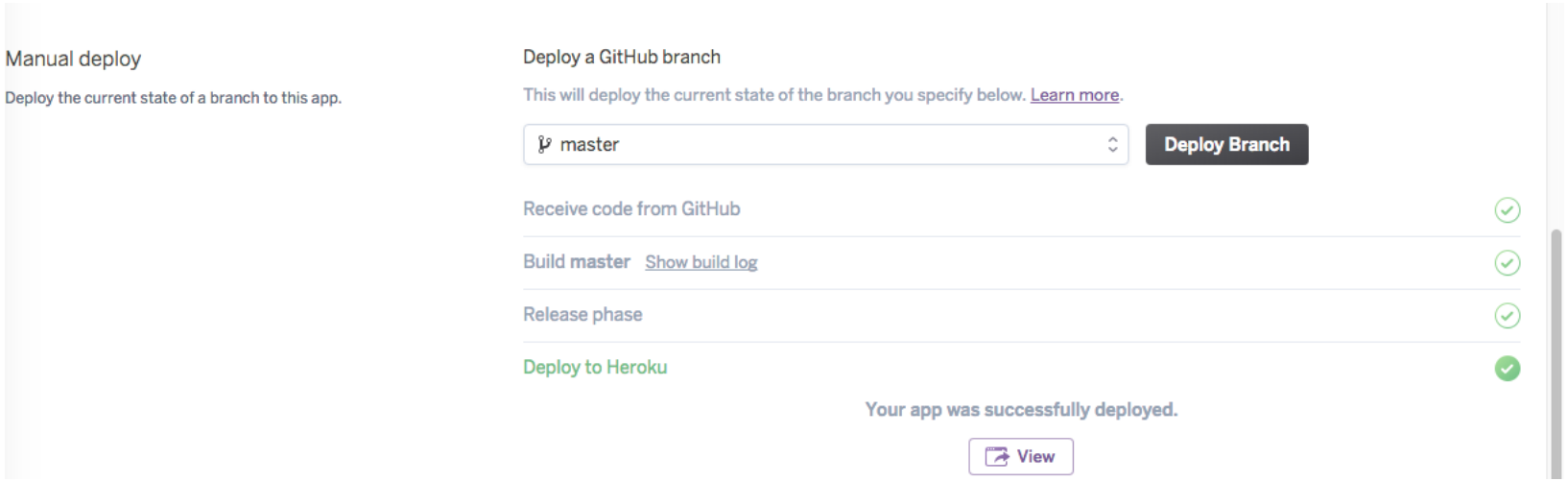
Click Create app
In the Deploy menu, under Deployment method, select GitHub



Two more simple steps
- Under Automatic Deploys, select the master branch and click Enable Automatic Deploys.
- Under Manual Deploys, click Deploy Branch. This is to push our fresh code to heroku.

Déploiement dans Heroku





Configure Your Angular App to Deploy Properly on Heroku

The following are production-ready steps to easily and properly deploy your app without hitches.

Ensure you have the latest version of angular cli and angular compiler cli.

Install them into your application by running this commands in your terminal:

```
npm install @angular/cli@latest @angular/compiler-cli --save-dev
```

In your package.json, copy

```
"@angular/cli": "1.4.9",
"@angular/compiler-cli": "^4.4.6",
```

from devDependencies to dependencies

Create postinstall script in package.json

Under "scripts", add a postinstall command like so:

```
"postinstall": "ng build --aot -prod"
```

This tells Heroku to build the application using Ahead Of Time (AOT) compiler and make it production-ready. This will create a `dist` folder where all html and javascript converted version of our app will be launched from.

Add Node and NPM engines

You will need to add the Node and NPM engines that Heroku will use to run your application. Preferably, it should be same version you have on your machine. So, run `node -v` and `npm -v` to get the correct version and include it in your package.json file like so:

```
1  "engines": {
2    "node": "6.11.0",
3    "npm": "3.10.10"
4  }
```

gistfile1.txt hosted with ❤ by GitHub view raw

Copy typescript to dependencies.

Copy `"typescript": "~2.3.3"` from devDependencies to dependencies to also inform Heroku what typescript version to use.

Install Enhanced Resolve 3.3.0

Run the command `npm install enhanced-resolve@3.3.0 --save-dev`

Install Server to run your app

Locally we run `ng serve` from terminal to run our app on local browser. But we will need to setup an Express server that will run our production ready app (from dist folder created) only to ensure light-weight and fast loading.

Install Express server by running:

```
npm install express path --save
```

Create a server.js file in the root of the application and paste the following code.

```
1  //Install express server
2  const express = require('express');
3  const path = require('path');
4
5  const app = express();
6
7  // Serve only the static files form the dist directory
8  app.use(express.static(__dirname + '/dist/<name-of-app>'));
9
10 app.get('/*', function(req,res) {
11
12 res.sendFile(path.join(__dirname+'/dist/<name-of-app>/index.
13 });
14
15 // Start the app by listening on the default Heroku port
16 app.listen(process.env.PORT || 8080);
```

gistfile1.txt hosted with ❤ by GitHub view raw

Change start command

In package.json, change the "start" command to `node server.js` so it becomes:

```
"start": "node server.js"
```

Here's what the complete package.json looks like. Yours may contain more depending on your application-specific packages.

```
{
  "name": "demo-deploy",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "node server.js",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "postinstall": "ng build --aot -prod"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^4.2.4",
    "@angular/cli": "^1.4.9",
    "@angular/common": "^4.2.4",
    "@angular/compiler": "^4.2.4",
    "@angular/compiler-cli": "^4.4.6",
    "@angular/core": "^4.2.4",
    "@angular/forms": "^4.2.4",
    "@angular/http": "^4.2.4",
    "@angular/platform-browser": "^4.2.4",
    "@angular/platform-browser-dynamic": "^4.2.4",
    "@angular/router": "^4.2.4",
    "core-js": "^2.4.1",
    "express": "^4.16.2",
    "rxjs": "^5.4.2",
    "typescript": "~2.3.3",
    "zone.js": "^0.8.14"
  },
  "devDependencies": {
    "@angular/cli": "^1.4.9",
    "@angular/compiler-cli": "^4.4.6",
    "@angular/language-service": "^4.2.4",
    "@types/jasmine": "~2.5.53",
    "@types/jasminewd2": "~2.0.2",
    "@types/node": "~6.0.60",
    "codelyzer": "~3.2.0",
    "enhanced-resolve": "^3.3.0",
    "jasmine-core": "~2.6.2",
    "jasmine-spec-reporter": "~4.1.0",
    "karma": "~1.7.0",
    "karma-chrome-launcher": "~2.1.1",
    "karma-cli": "~1.0.1",
    "karma-coverage-istanbul-reporter": "^1.2.1",
    "karma-jasmine": "~1.1.0",
    "karma-jasmine-html-reporter": "^0.2.2",
    "protractor": "~5.1.2",
    "ts-node": "~3.2.0",
    "tslint": "~5.7.0",
    "typescript": "~2.3.3"
  },
  "engines": {
    "node": "6.11.0",
    "npm": "3.10.10"
  }
}
```

gistfile1.txt hosted with ❤ by GitHub view raw

Push changes to GitHub:

```
git add .
git commit -m "updates to deploy to heroku"
git push
```

At this point, your application on Heroku will automatically take the changes from GitHub and update itself.

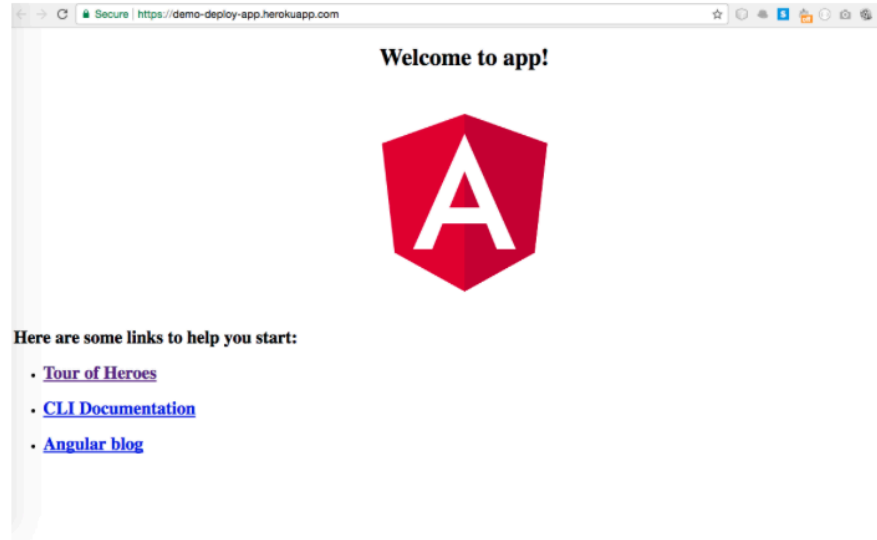
Also, it'll look into your package.json and install packages.

It will run the postinstall and then, `node server.js` to spin up your application.

You can check Activity tab and open Build log t o see how it actually runs.

You should not run into any issue. I followed through while writing this post also and.

Viola!! Our Angular app is Ready and LIVE!



- REVOIR LES QUESTIONS GLOBALES ANGULAR

- REVOIR NODE

- SI IL RESTE DU TEMPS ENTAMMER HTTP ANGULAR

Angular 5

Angular est effectivement un **framework très puissant**, relativement **innovant** tant dans sa **philosophie** et son organisation que par les **technologies** et les concepts qu'il utilise.

Néanmoins **la courbe d'apprentissage peut être relativement complexe** et il demande énormément de **pratique** et de **documentation** avant d'être complètement maitrisé.

il existe de nombreux tutoriels dont le meilleur est tout simplement servi **par Google sur le site officiel d'Angular**.

MERCI

Formation Cordova & Ionic

APACHE CORDOVA™

IONIC

Formation 2 jours

Objectif : **Réaliser une application web à l'aide de Cordova et Ionic**

Cordova

- Introduction à Node et NPM
- Présentation de Cordova
- Gestion des plateformes
- Ajout de Plugins
- App caméra mobile

Ionic

- Introduction à Node et NPM
- Présentation de Ionic
- composants & architecture
- Navigation et workflow
- App mobile complète

Déploiement

- Présentation Ionic
- Plateforme IOS et Android
- Performance, vigilance avec les listes
- Déploiement Cordova
- Tester et Publier son application