



# AngularJS

*Jour 4 - Les promises, les providers et le serveur*

# Au programme

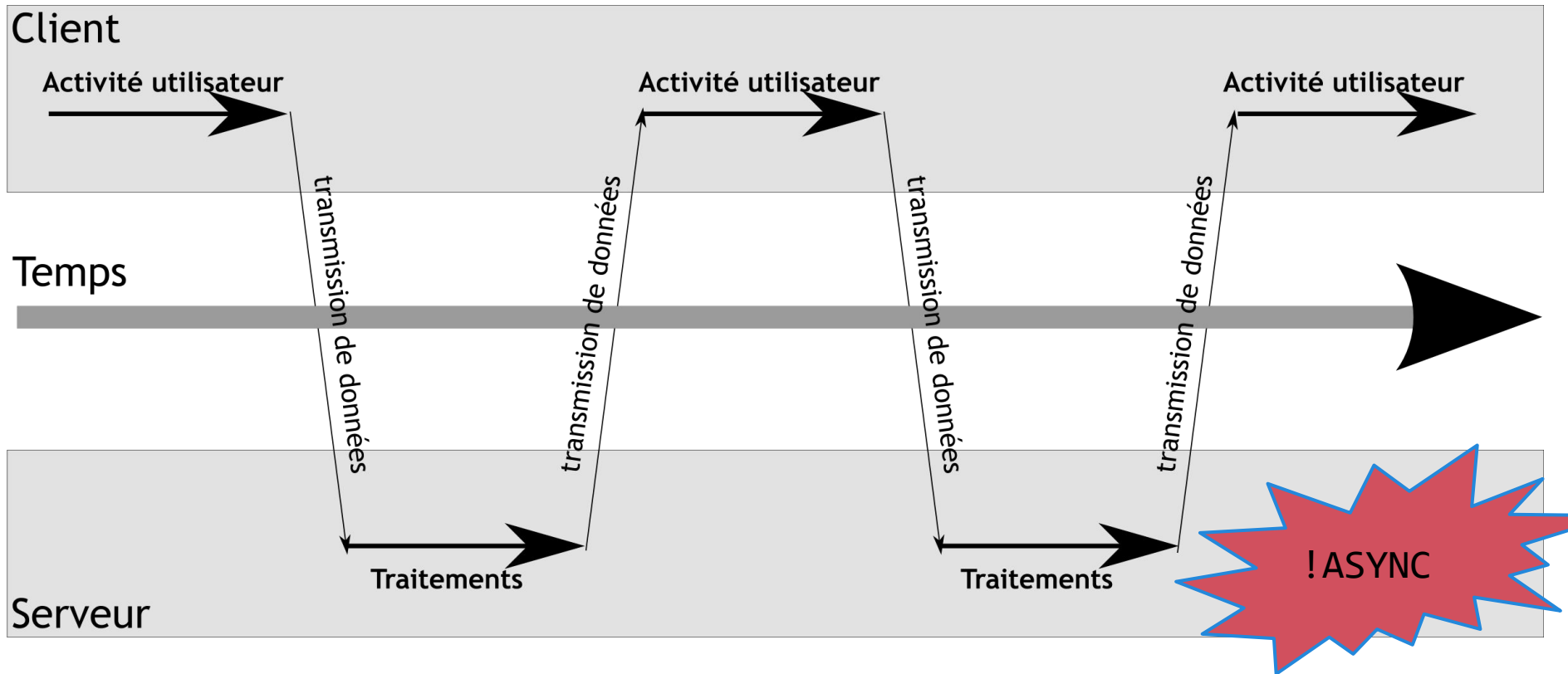
- Les Promises (ou Promesses)
- Les providers AngularJS  
*service, factory, provider, value, constant, decorators*
- La communication avec le server  
*\$http, \$resource*
- TP : Démarrer un projet

# **1 - Les Promises (ou Promesses)**

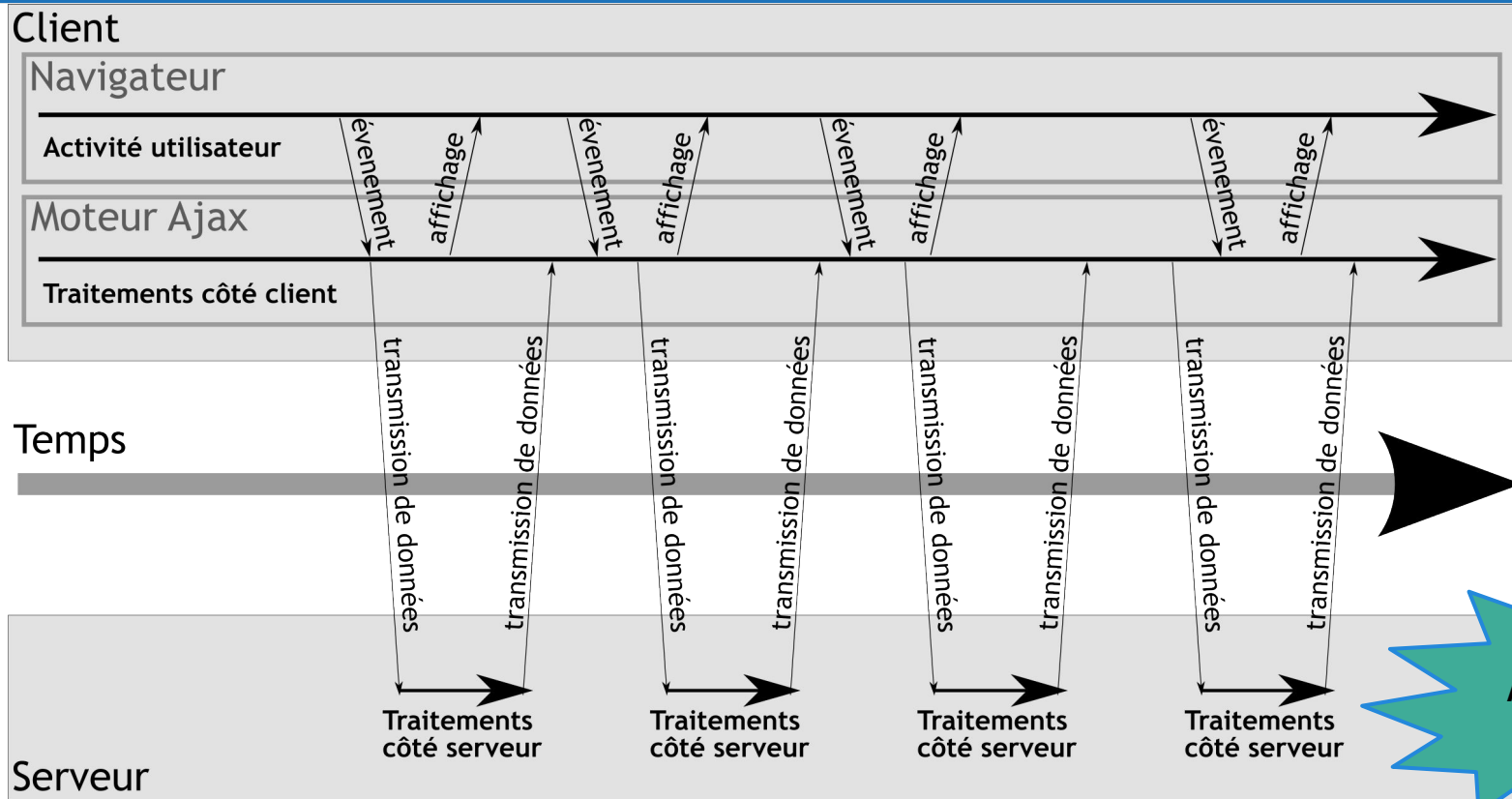
# Les Promises - Kezako

- JavaScript est un langage **asynchrone**
  - à l'inverse du séquentiel

# Les Promises - Kezako



# Les Promises - Kezako



# Les Promises - Kezako

- JavaScript est un langage **asynchrone** (à l'inverse du séquentiel)
- Les promises permettent d'écrire à la façon **séquentielle** tout en gérant de l'**asynchrone**

# Les Promises - callbacks

## - Avant les promises : Les callbacks

```
var myAsyncFunction = function (id, callback) {  
  // Do some stuff  
  setTimeout(function() {  
    callback(); // Retour asynchrone  
  }, 1000);  
  return true; // Retour séquentielle  
};
```



# Les Promises - callbacks

- Avant les promises : Les callbacks

```
asncFunction1(function(err, result) {  
  asncFunction2(function(err, result) {  
    asncFunction3(function(err, result) {  
      asncFunction4(function(err, result) {  
        asncFunction5(function(err, result) {  
          // do something useful  
        })  
      })  
    })  
  })  
})
```

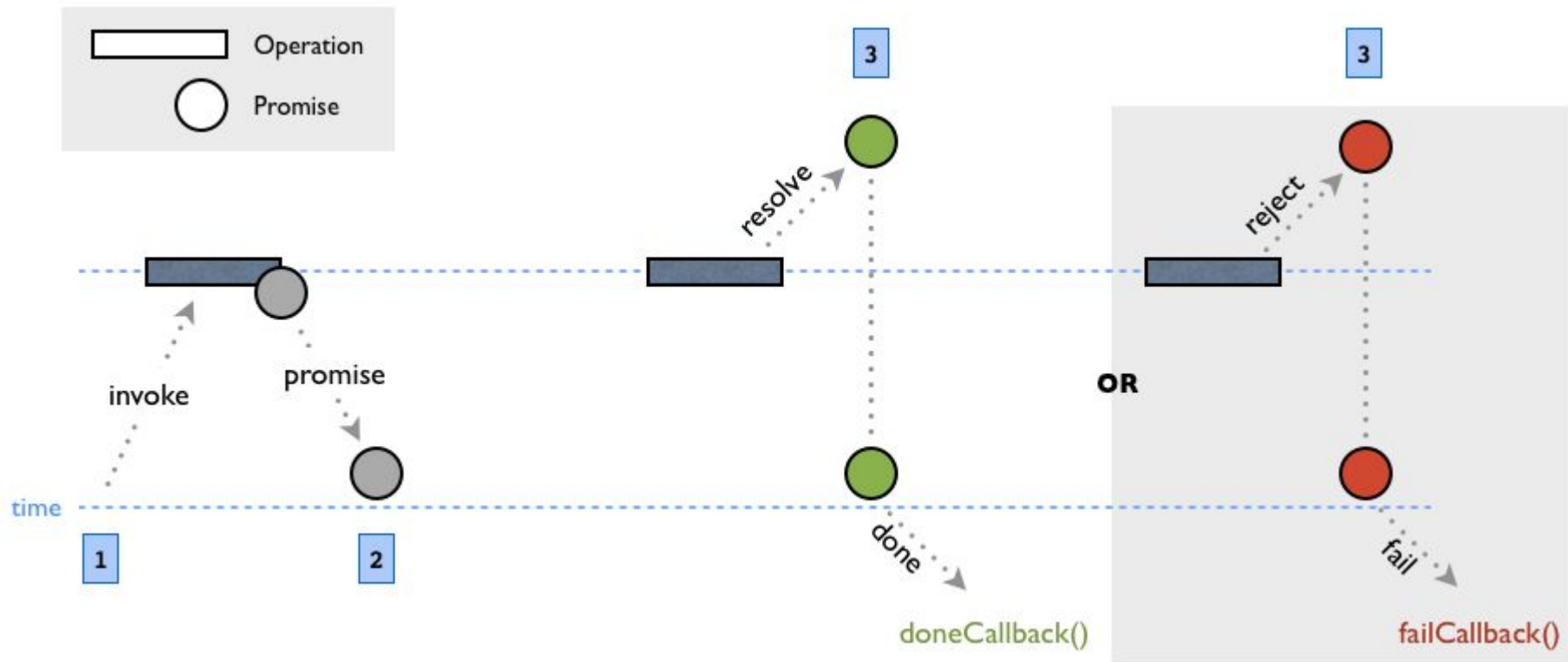


Illisible!

# Les Promises

- Permet d'écrire en séquentielle pour gérer de l'asynchrone
- Eviter l'arbre de noel des callbacks
- Conserve le context d'exécution

# Les Promises



# Les Promises - La gestion du résultat

- Les promises sont protégées (try/catch)
- Quatres retours possible:
  - resolve
  - reject
  - error
  - notify

# Les Promises - La gestion du résultat

```
function fetchData(id){  
    return getDataFromServer(id)  
        .then(transformData, handleError)  
        .then(saveToIndexDB);  
}  
  
function transformData() {}  
  
function saveToIndexDB() {}  
  
function getDataFromServer() {} // return promise  
  
function handleError() {}
```

# Les Promises - La gestion de l'état

```
promise = new Promise()  
promise.resolve(data) // any type  
promise.reject(msg) // rejection error message  
promise.notify(data) // any type
```

# Les Promises - Dans AngularJS

- Utilise un service \$q  
<https://goo.gl/QaTwwp>
- DEMO
  - [http://localhost:8080/day\\_04/step\\_01](http://localhost:8080/day_04/step_01)

## 2 - Les providers

# AngularJS



# Les providers - Kezako

## - Six providers AngularJS

Provider	Singleton	Instantiable	Configurable
Constant	Yes	Yes	No
Value	Yes	No	No
Service	Yes	No	No
Factory	Yes	Yes	No
Decorator	Yes	No?	No
Provider	Yes	Yes	Yes

# Les providers - Constant

- Une ***constant*** est **immutable**

```
angular.module('app', []);  
app.constant('API_URL', 'http://api.twitter.com');  
app.controller('MyController', function (API_URL) {  
    console.log(API_URL === 'http://api.twitter.com');  
});
```

# Les providers - Value

- Une ***value*** est **valeur** injectable (tout type)

```
angular.module('app', []);  
app.value('myApiUrl', 'http://api.twitter.com');  
app.controller('MyController', function (myApiUrl) {  
    console.log(myApiUrl === 'http://api.twitter.com');  
});
```

# Les providers - Service

- Un ***service*** est un singleton (*new*)

```
angular.module('app', []);
app.service('userService', function($http) {
  this.getUser = function () {
    return $http.get('url.com/users');
  };
});
app.controller('MyController', function (userService) {
  var self = this;
  userService.getUser().then(function(res) {
    self.users = res;
  });
});
```

# Les providers - Factory

- Un ***factory*** est un singleton (sans *new*)

```
angular.module('app', []);
app.factory('userService', function ($http) {
  var _privateUser = [];
  return {
    getUser: function () {
      return $http.get('url.com/users').then(function(res) {
        _privateUser = res.data;
        return _privateUser;
      });
    }
  });
});
app.controller('MyController', function (userService) {
  var self = this;
  userService.getUser().then(function(res) {
    self.users = res;
  });
});
```

# Les providers - Decorator

- Un ***decorator*** modifie ou encapsule d'autres providers

```
angular.module('app', []);
app.value('apiUrl', "api.com");
app.config(function ($provide) {
    $provide.decorator('apiUrl', function ($delegate) {
        return $delegate + '/v2';
    });
});
app.controller('MyController', function (apiUrl) {
    console.log(apiUrl, 'api.com/v2');
});
```

# Les providers - Provider

- La plus complexe
- Un ***provider*** donne la possibilité d'avoir une phase de configuration.

# Les providers - Provider

```
angular.module('app', []);
app.provider('apiUrl', function () {
  var version;
  return {
    setVersion: function (v) {
      version = v;
    },
    $get: function () {
      return url = "api.com/" + version;
    }
  }
});
app.config(function (apiUrlProvider) {
  apiUrlProvider.setVersion('v2');
});
app.controller('MyController', function (apiUrl) {
  console.log(apiUrl, 'api.com/v2');
});
```



# Les providers

- DEMO
  - [http://localhost:8080/day\\_04/step\\_02](http://localhost:8080/day_04/step_02)

# **3 - La communication avec le serveur**

# Le server - Kezako

- AJAX au coeur des **Single Page Application**
- Application client à (pratiquement) **toujours** besoin d'information du server
  - Authentification
  - Liste d'information
  - etc.

# Le server - Communiquer avec \$http

- Service AngularJS **\$http**
- Facilite la communication HTTP (XMLHttpRequest ou JSONP)
- Compatible verbes HTTP
  - GET, POST, HEAD, DELETE, PUT, JSONP
- Utilise des **promises**

# Le server - Communiquer avec \$http

```
$http.get('url/someUrl')  
  .success(function(data, status, headers, config) { })  
  .error(function(data, status, headers, config) { })
```

```
$http.post('/someUrl', {msg: 'Message sent to the server!'})  
  .success(function(data, status, headers, config) { })  
  .error(function(data, status, headers, config) { })
```

// Disponible : \$http.get, \$http.head, \$http.post, \$http.put, \$http.delete, \$http.jsonp, \$http.patch

- Aller plus loin

[https://docs.angularjs.org/api/ng/service/\\$http](https://docs.angularjs.org/api/ng/service/$http)

- DEMO

- [http://localhost:8080/day\\_04/step\\_03](http://localhost:8080/day_04/step_03)

# Le server - Communiquer avec \$resource

- REST : <http://goo.gl/FvfGMV>
- Convention de nommage avec le server
- Méthode générique pour TOUT les serveur de type *RESTful*
  - get, save, query, remove & delete

# Le server - Communiquer avec \$resource

## - Utiliser \$resource

```
angular.factory("userService", function($resource) {  
    return $resource('/users/:userId', {userId: '@id'});  
});
```

1. Service **/users** appelé (le nom de la ressource)
2. **:userId** sera remplacé lors d'un appel get avec ID
3. **@id** permet d'automatiser la mise à jours

# Le server - \$resource : utilisation

- Récupérer une liste (**query**)
  - Appel statique sans \$resource

```
$scope.users = [userObject1, etc.]; // Statique
```

- Appel dynamique avec \$resource

```
$scope.users = userService.query(); // dynamique
```



# Le server - \$resource : utilisation

- Récupérer une entrée (**get(id)**)

```
$scope.user = userService.get({userId: 5});
```

- Créer une entrée (save)

```
var user = new userService();  
user.firstname = 'Benjamin';  
user.lastname = 'Longearet';  
user.$save();
```

# Le server

- Modifier une entrée (**update**)
  - A créer car *update* n'existe pas par défaut

```
angular.factory("userService", function ($resource) {  
  return $resource(  
    '/users/:userId',  
    {userId: '@id'},  
    {update: {method: 'PUT'}}  
  });  
$scope.user = userService.get({userId: 5});  
$scope.user.name = 'new name';  
$scope.user.$update();
```

# Le server - \$resource

- DEMO AngularJS (tutorial step 11)
  - <https://github.com/angular/angular-phonecat>
  - <http://angular.github.io/angular-phonecat/step-11/app/#/phones>

**4 - TP**

# Objectifs

- Mettre en pratique tout les concepts
- A minima
  - Une architecture orienté composant
  - Du routing (au moins deux modules)
  - Deux directives personnalisées (avec un scope isolé)
  - Un filter
  - Un provider qui s'occupe de récupérer les données d'un fichier json
  - L'utilisation des directives angularJS les plus connues  
ng-repeat, ng-model

# Helps :)

## - Architecture composant

**projet/**

- index.html

- **vendors/**

--- angular.js

--- jquery.js

--- etc.

- **app/**

--- app.js  $\Rightarrow$  *angular.module('myApp', ['myApp.home', 'myApp.user'])*

--- **components/**

---- **home/**

----- home.js (controller)  $\Rightarrow$  *angular.module('myApp.home', [])*

----- home.html (template)

----- homeFilter.js

----- etc.

---- **user/**

----- user.js (controller)  $\Rightarrow$  *angular.module('myApp.user', [])*

----- user.html (template)

----- userService.js (récupération données serveur)

----- userDirective.js