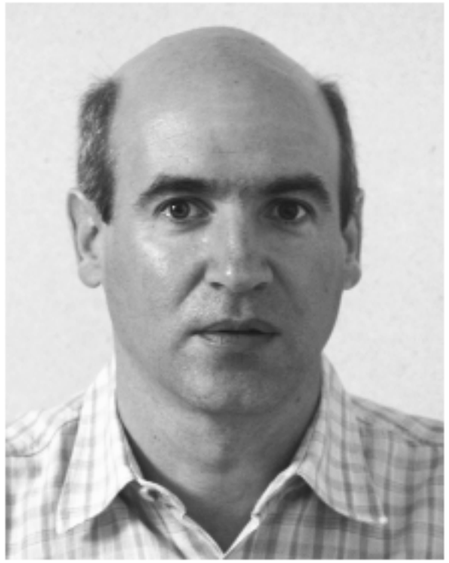
“Au secours,

mon code AngularJS est pourri !”

MethoTIC Conseil

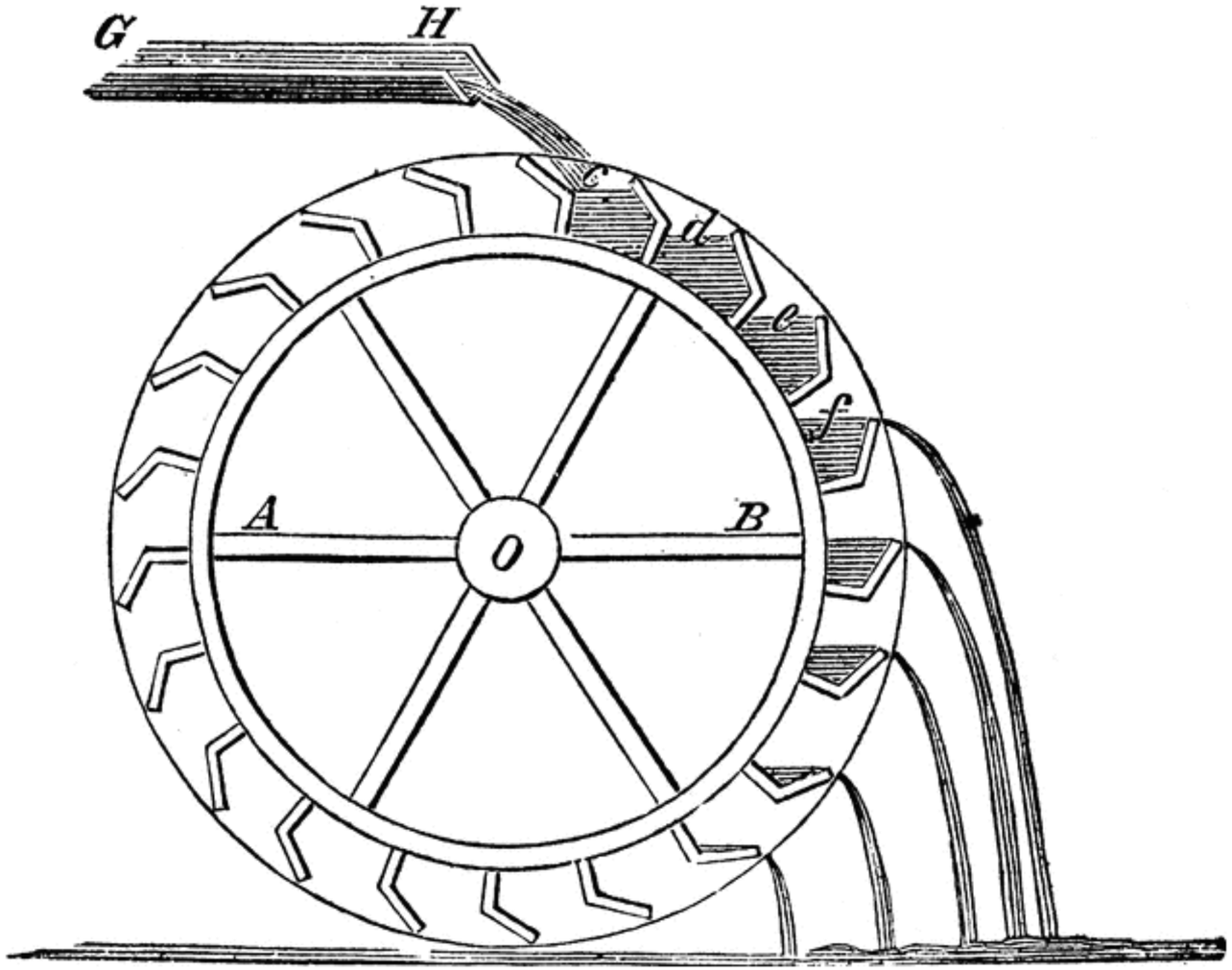


Thierry Chatel

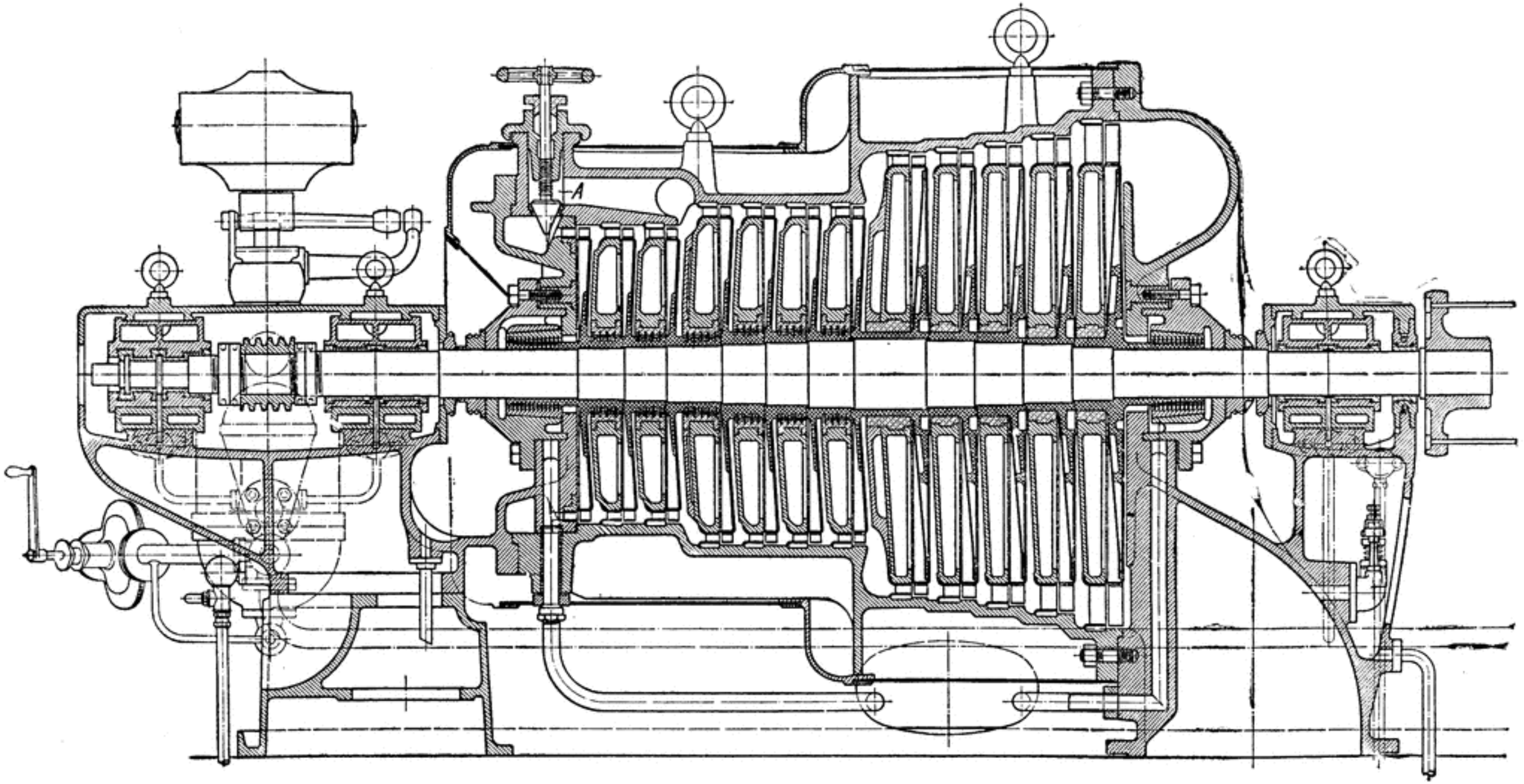
consultant formateur AngularJS

crealead

INTRODUCTION



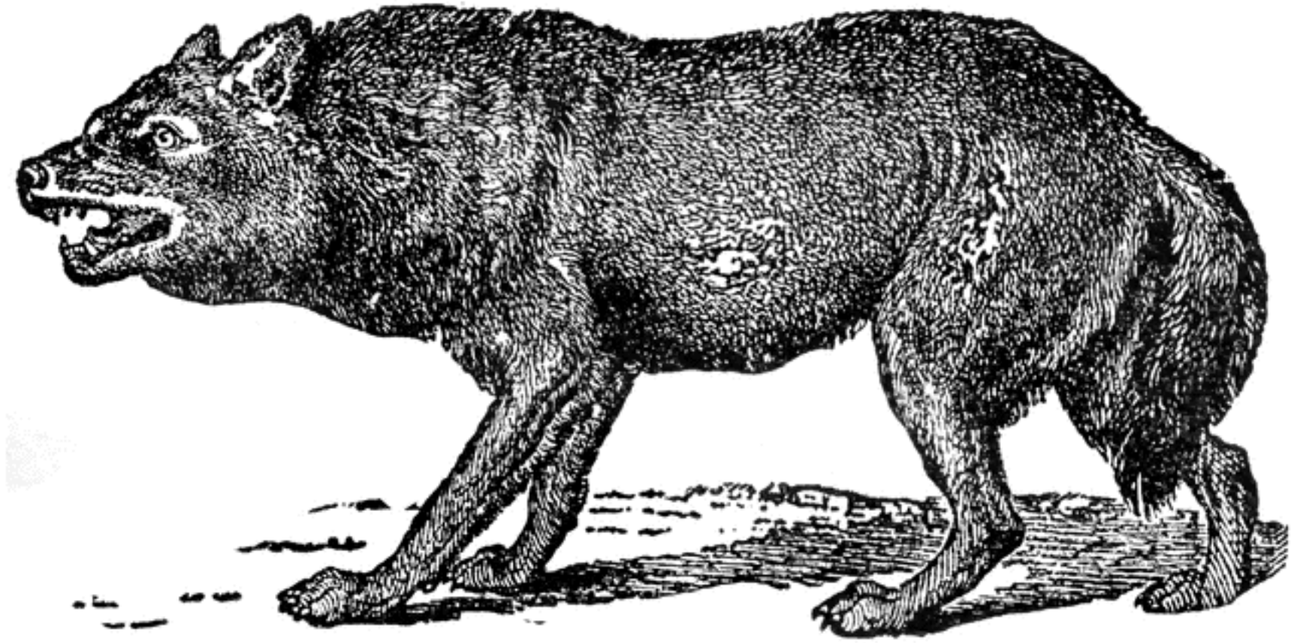
les débuts avec AngularJS



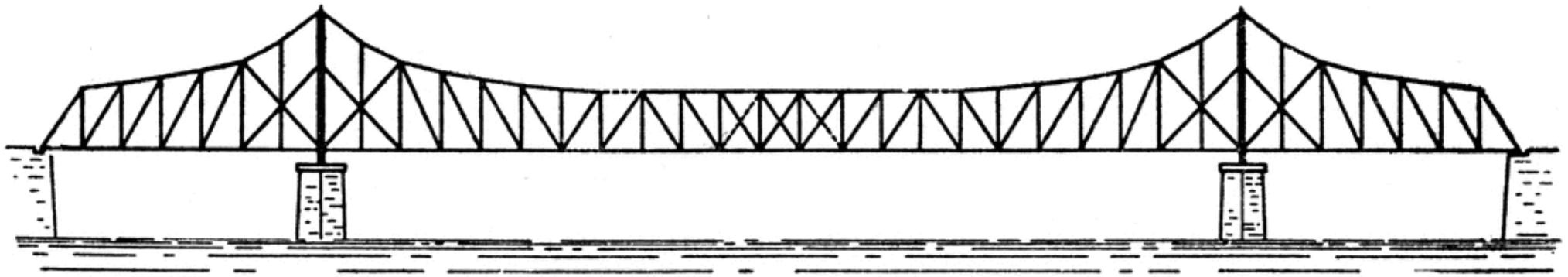
mais après...

“No silver bullet”

Fred Brooks, 1986



dans cette présentation…



GENERALITÉS

attention aux composants

ne pas optimiser

sauf quand c'est VRAIMENT nécessaire

bindings sur des fonctions

sans stocker le résultat

travaillez le modèle

View-Model

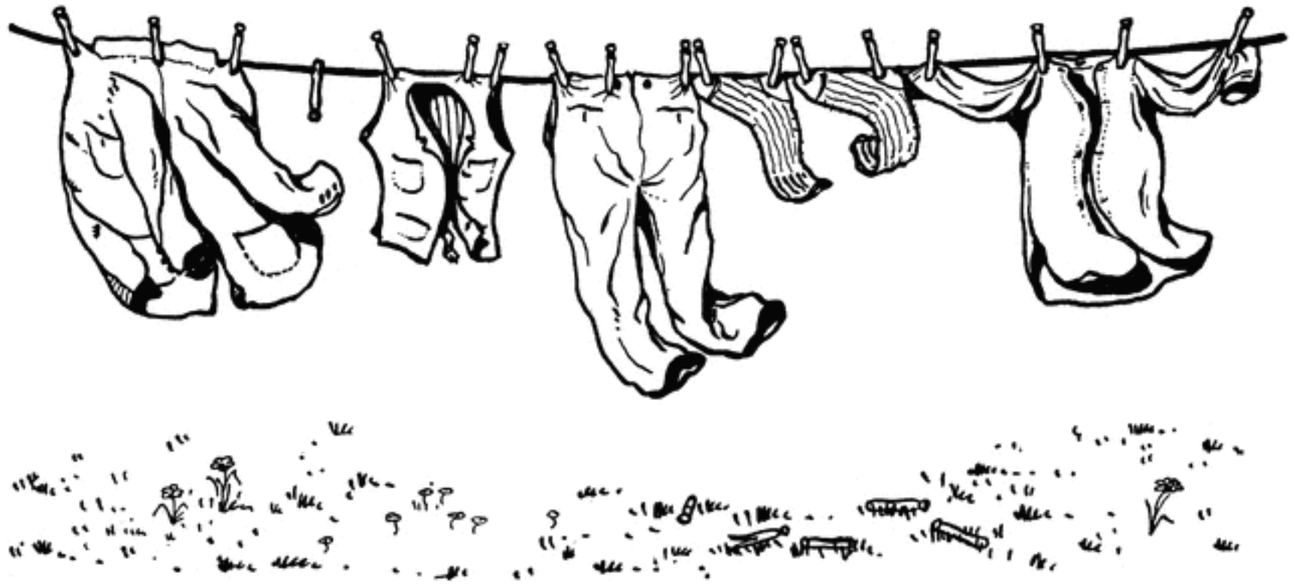
vrai modèle objet, pas juste du JSON

service dans le scope

ou une façade

STRUCTURE

organiser par type ?



découpage fonctionnel

et par type au plus bas niveau

```
front/
    profiles/
    repositories/
    activity/
    contributions/

back/
    statistics/

common/
```

1 fichier JS = 1 module

```
front/
    front.js
    profiles/
        profiles.js
        profiles-controller.js
        profiles-service.js
        profiles-directive.js
    repositories/
        repositories.js
        ...
```

```
common/
        common.js
        common-service.js
        common-filter.js
        common-directive.js
```

+ templates, tests, images...

routes :

dans chaque branche

- ajouter des données à la définition de la route
- récupérables dans `$route.current`

un service `constant()`

utilisable dans `config()`

pour définir le chemin des templates

CONTRÔLEURS

contrôleur

=

initialisation du scope

pas de traitements

sur les données

lien

entre services

et scope

contrôleurs légers

contrôleur local

pour une zone de la vue

contrôleur sur un scope répété

pour calculer des valeurs, ou mettre un watch

```
<ul>
  <li ng-repeat="item in list"
      ng-controller="ItemCtrl">
      ...
  </li>
</ul>
```

syntaxe “controller as …” ?

préconisée par Google

- mieux pour l'héritage entre les scopes
- plus explicite dans les templates
- mais plus verbeux

SERVICES

chaque fonctionnalité

isolée dans un service

- rien qu'une fonctionnalité
- mais dans son intégralité

tout le code métier

- pas de traitements dans les contrôleurs
- pas de règles métier dans les templates

```
ng-class="{alert: quantity(row) >= 100}"
```

plutôt :

```
ng-class="{alert: orderSrv.isAlert(row)}"
```

conserver des données

service = données + traitements

services asynchrones

renvoyer des promesses

découper les services

en couches

erreurs et notifications

- un intercepteur $http
- surcharge du service $exceptionHandler
- un service pour les erreurs
- un service de log serveur
- un service de notification

utilisateur connecté

1. créer un service

pour injecter dans des services, des contrôleurs

utilisateur connecté

2. publier le service dans le scope

intégralement, partiellement, ou une façade

uniquement pour les templates

PROMESSES

même code

pour traiter des résultats

synchrones / asynchrones

en attente / résolus

appeler

then()

ne conserver

que des promesses

(jamais les résultats !)

en cas d'échec

renvoyer une promesse en erreur

ou exception

FILTRES

ne jamais modifier

les données en entrée

parser / évaluer des expressions

plutôt que de prendre des noms de propriétés

```
users | orderBy:'profile.name':false
```

$parser

```
var getter = $parser(expr);

var value = getter(context);
```

```
item in list | filter:search

(list | filter:search).length
```

plutôt :

```
item in filtered = (list | filter:search)

filtered.length
```

DIRECTIVES

privilégier les bindings

aux manipulations de DOM

s'appuyer sur le HTML

plutôt que de le remplacer

(syndrome JSF)

attributs HTML

=

interface de la directive

pas d'héritage implicite

(données du scope parent)

contrôleur de directive

lien directives parent / enfants

expressions dans les attributs

(éviter les nombres

et les noms de propriétés)

scope isolé

lien “=” : comme nommer un argument

pas de pollution d'un scope externe

directive sur plusieurs éléments

syntaxe `-start` et `-end`

(`ng-repeat-start="..."` / `ng-repeat-end`)

aussi valable pour vos propres directives

élément HTML standard

directives `input`, `form`, `script`

plutôt que de rechercher les éléments dans la page

TESTS

soigner le code

des tests ?

il faut le maintenir

imbriquer un *describe*

pour factoriser dans un *beforeEach*

Protractor :

factoriser des ElementFinder et des fonctions

```
rows = element.all(
    by.repeater('row in rows')
);

getQuantityOnFirstRow = function () {
    var firstRow = rows.get(0);
    var input = firstRow.$('input.quantity');
    return input.getAttribute('value');
}
```

CONCLUSION

faire simple

- partir du HTML
- profiter de la souplesse du JS
- bonnes pratiques de programmation objet

bonne chance...