BACKBONE.JS

Build spaghetti-proof web apps

JAVASCRIPT ?

JAVASCRIPT ?

JAVASCRIPT ?

jQuery

do more.

LA PHILOSOPHIE

LA PHILOSOPHIE

- Application web monopage

LA PHILOSOPHIE

- Application web monopage
- Architecture MV(C?)

LA PHILOSOPHIE

- Application web monopage
- Architecture MV(C?)
- Framework léger

LA PHILOSOPHIE

- Application web monopage
- Architecture MV(C?)
- Framework léger
- Agnostique

LA PHILOSOPHIE

- Application web monopage
- Architecture MV(C?)
- Framework léger
- Agnostique
- Communication RESTful

LA PHILOSOPHIE

- Application web monopage
- Architecture MV(C?)
- Framework léger
- Agnostique
- Communication RESTful
- Dépendance Underscore.js et jQuery

LA WEB-APP CLASSIQUE

Serveur

- Logique business
- Gestion des données
- Templating
- i18n

HTML

Client

- Requête
- Parsing HTML et affichage

LA WEB-APP CLASSIQUE

Serveur

- Logique business
- Gestion des données
- Templating
- i18n

HTML

Client

- Requête
- Parsing HTML et affichage

x nClients

LA WEB-APP CLASSIQUE

Serveur

- Logique busin
- Gestion des d
- Templating
- i18n

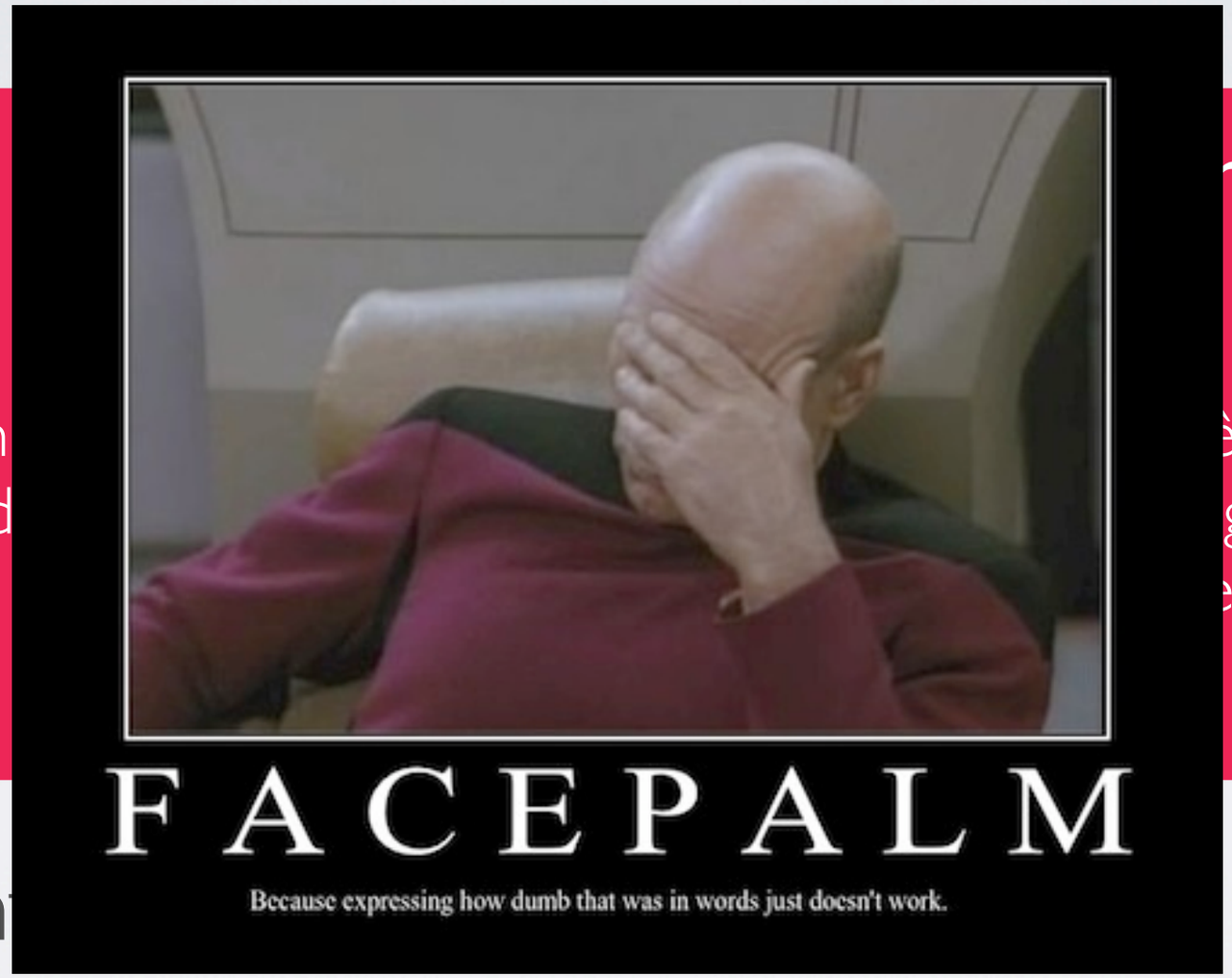
nt

ète

g HTML et

e

x nClien



LA WEB-APP CLASSIQUE

Serveur

- Logique business
- Gestion des données
- Templating
- i18n

HTML

Client

- Requête
- Parsing HTML et affichage

x nClients

LA WEB-APP CLASSIQUE

Serveur

- Logique business
- Gestion des données
- Templating
- i18n

HTML

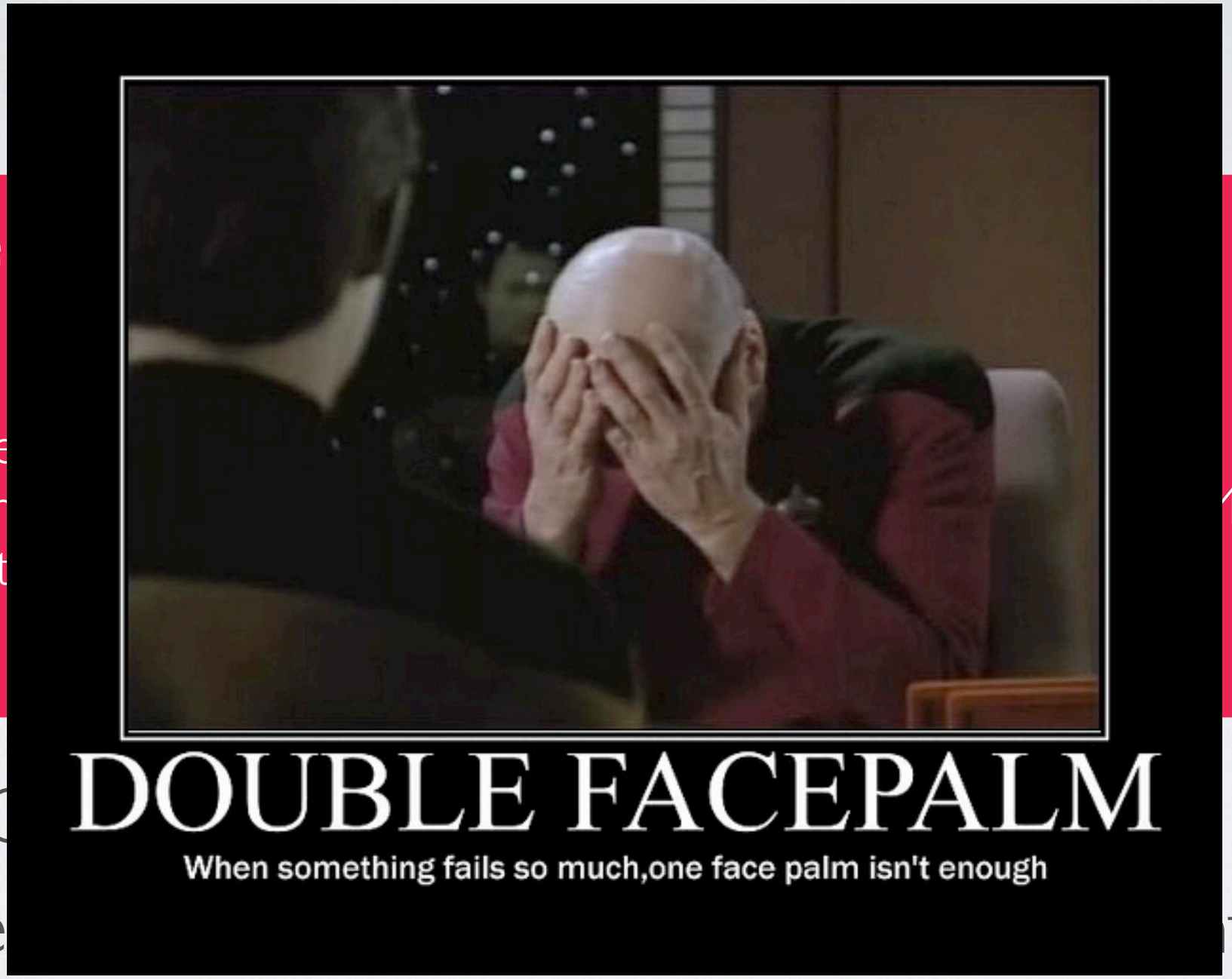
Client

- Requête
- Parsing HTML et affichage

x nClients

+ Templating complet à chaque changement de page

LA WEB-APP CLASSIQUE



Serve

- Logique
- Gestion
- Templat
- i18n

1L et

x nC

+ Te

t de page

LA WEB-APP BACKBONE

Serveur

- Logique business
- Gestion des données

JSON

Client

- Requête
- Templating
- i18n
- Récupération des données

LA WEB-APP BACKBONE

Serveur

- Logique business
- Gestion des données

JSON

Client

- Requête
- Templating
- i18n
- Récupération des données

Requêtes légères

LA WEB-APP BACKBONE

Serveur

- Logique business
- Gestion des données

JSON

Client

- Requête
- Templating
- i18n
- Récupération des données

Requêtes légères

Travaille pour nous

L'EXEMPLE

Plixee

Ecrire un tweet

Votre nom

Hugoch

Votre message

Hey, I'm presenting #Backbonejs at #NWXTech5 conference.

ENVOYER

Derniers tweets

marcsimoncini

Les américains parient que dans les 2/3 ans une des 10 + grandes villes US bannira les voitures particulières du centre ville. #autopartage

---

da_Rouen

Recensement des zones de circulation apaisée en HN (zones 30, double sens cyclable, zones de rencontre..) | DREAL HN dlvr.it/2qklF1

---

newsycombinator

Who Needs Starbucks? Dwolla Gets Into Gov't With Iowa Tax-Paying Plan

LES MODÈLES

```coffeescript
# Création de la classe d'un Tweet
Tweet = Backbone.Model.extend
    urlRoot: "/api/tweets"
    validate: (attributes)->
        if attributes.author == "" then return "Invalid author"
        if attributes.text == "" or attributes.text.length > 140 then return
"Invalid tweet"

# Instanciation d'un nouveau Tweet
aTweet = new Tweet(
    author: "Hugoch"
    text: "Hey, I'm presenting #Backbonejs at #NWXTech5 conference."
)
```

LES MODÈLES

```coffeescript
# Création de la classe d'un Tweet
Tweet = Backbone.Model.extend
    urlRoot: "/api/tweets"
    validate: (attributes)->
        if attributes.author == "" then return "Invalid author"
        if attributes.text == "" or attributes.text.length > 140 then return
"Invalid tweet"

# Instanciation d'un nouveau Tweet
aTweet = new Tweet(
    author: "Hugoch"
    text: "Hey, I'm presenting #Backbonejs at #NWXTech5 conference."
)

aTweet.save()
```

POST /api/tweets

LES MODÈLES

```coffeescript
# Création de la classe d'un Tweet
Tweet = Backbone.Model.extend
    urlRoot: "/api/tweets"
    validate: (attributes)->
        if attributes.author == "" then return "Invalid author"
        if attributes.text == "" or attributes.text.length > 140 then return
"Invalid tweet"

# Instanciation d'un nouveau Tweet
aTweet = new Tweet(
    author: "Hugoch"
    text: "Hey, I'm presenting #Backbonejs at #NWXTech5 conference."
)

aTweet.save()

aTweet.save(
    author: "Chuck Norris"
)
```

POST /api/tweets

PUT /api/tweets/42

LES MODÈLES

```coffeescript
# Création de la classe d'un Tweet
Tweet = Backbone.Model.extend
    urlRoot: "/api/tweets"
    validate: (attributes)->
        if attributes.author == "" then return "Invalid author"
        if attributes.text == "" or attributes.text.length > 140 then return
"Invalid tweet"

# Instanciation d'un nouveau Tweet
aTweet = new Tweet(
    author: "Hugoch"
    text: "Hey, I'm presenting #Backbonejs at #NWXTech5 conference."
)

aTweet.save()

aTweet.save(
    author: "Chuck Norris"
)

aTweet.destroy()
```

→ POST /api/tweets

→ PUT /api/tweets/42

→ DELETE /api/tweets/42

LES COLLECTIONS

```coffeescript
# Création d'une collection de tweets
Tweets = Backbone.Collection.extend
    model: Tweet
    url: "/api/tweets"

# Instanciation de la collection
someTweets = new Tweets()

# Récupération des Tweets sur le serveur
someTweets.fetch()
```

```json
[
    {
        "id": 544102,
        "author": "N_W_X",
        "text": "Conférence #nwxtech5 du 24
           janvier, avec @GrieuL @nautilebl
           eu @romainlouvet @zigazou @hugoc
           h et @moebius_eye : amiando.com/
           nwxtech5"
    },
    {
        "id": 24454,
        "author": "N_W_X",
        "text": "Conférence dédiée aux tech
           nos web #nwxtech5 à Rouen le 24
           janvier :  amiando.com/nwxtech5"
    }
]
```

LES VUES

- Une vue = un élément du DOM
- Une vue représente un modèle
- Possibilité de mettre à jour une portion de page

```
# Création d'une vue de tweet
TweetView = Backbone.View.extend
    tagName: "li"
    className: "tweet"
    render: ()->
        tpl = _.template("""<h2><%-author%></h2>
        <p><%-text%></p>""")
        @$el.append(tpl(@model.toJSON()))
        return @
```

```
# Instanciation de la collection
someTweets = new Tweets()

# Affichage des tweets lorsqu'ils sont ajoutés à ma collection
someTweets.on("add",(tweet)->
    view = new TweetView(
        model: tweet
    )
    $("#tweet-list").append(view.render().el)
)
```

LES ROUTEURS

- Gèrent la navigation au sein de la web-app avec des URL transparentes (History API des navigateurs)
- Permettent aux utilisateurs de bookmarker des vues de l'application

```
# On crée la classe de routeur
TweetApp = Backbone.Router.extend
    routes:
        "last/:num":    "showLast"
        "*path":        "home"

    showLast: (num)->
        # Récupération des nums
        derniers Tweets sur le serveur
        someTweets.fetch(
            update: true
            data:
                limit: num
        )

    home: ()->
        someTweets.fetch(
            update: true
        )
# Instanciation du routeur
app = new TweetApp()

# Démarrage de l'app
Backbone.history.start({pushState: true})
```

L'APPLICATION

```
<html>
<head>
    <script src="jquery.js"></script>
    <script src="underscore.js"></script>
    <script src="backbone.js"></script>
    <script src="examples.js"></script>
    <link href="stylesheets/screen.css" media="screen, projection" rel="stylesheet" type="text/
css" />
</head>
<body>
    <div id="container">
        <h1>Ecrire un tweet</h1>
        <div class="form-row">
            <label for="new-tweet">Votre message</label>
        </div>
        <div class="form-row">
            <textarea id="new-tweet"></textarea>
        </div>
        <div class="form-row right">
            <button id="send-tweet">Envoyer</button>
        </div>

        <h1>Derniers tweets</h1>
            <ul id="tweet-list">
            </ul>
    </div>
</body>
</html>
```

Insertion des vues de tweets

L'APPLICATION

```coffee
# Création de la classe d'un Tweet
Tweet = Backbone.Model.extend
    urlRoot: "/api/tweets"
    validate: (attributes)->
        if attributes.author == "" then return "Invalid author"
        if attributes.text == "" or attributes.text.length > 14
0 then return "Invalid tweet"

# Création d'une collection de tweets
Tweets = Backbone.Collection.extend
    model: Tweet
    url: "/api/tweets"

# Instanciation de la collection
someTweets = new Tweets()

# Affichage des tweets lorsqu'ils sont ajoutés à ma collection
someTweets.on("add",(tweet)->
    view = new TweetView(
        model: tweet
    )
    $("#tweet-list").prepend(view.render().el)
)

# Création d'une vue de tweet
TweetView = Backbone.View.extend
    tagName: "li"
    className: "tweet"
    render: ()->
        tpl = _.template("""<h2><%-author%></h2>
        <p><%-text%></p>""")
        @$el.append(tpl(@model.toJSON()))
        return @
```

```coffee
# On écoute le clic sur le bouton de création de tweet
$(()->
    $("#send-tweet").click(()->
        tweet = new Tweet()
        if tweet.save(
            author: $("#author").val()
            text: $("#new-tweet").val()
        )
            someTweets.add(tweet)
            $("#new-tweet").val("").focus()
    )
)

# On crée la classe de routeur
TweetApp = Backbone.Router.extend
    routes:
        "last/:num":    "showLast"
        "*path":        "home"

    showLast: (num)->
        # Récupération des Tweets sur le serveur
        someTweets.fetch(
            update: true
            data:
                limit: num
        )

    home: ()->
        someTweets.fetch(
            update: true
        )

# Instanciation du routeur
app = new TweetApp()
# Démarrage de l'app
Backbone.history.start({pushState: true})
```

EXEMPLE

nwxtech.herokuapp.com

CONVAINCU ?

- 117 lignes de code (serveur + javascript)
- Temps réel
- 1 chargement des ressources (serveur statique), puis 300 octets par tweet
- Plugins : Backbone.Relational, LocalStorage

Utilisé par :

foursquare

airbnb

Trello

diaspora*

Plixee

jolicloud

SIGINT

Plus d'infos

- Backbone.js ➔ backbonejs.org
- GitHub ➔ github.com/documentcloud/backbone
- Exemple ➔ github.com/Plixee/backbone-nwx-example

Contact

hugo@plixee.com

@hugoch

Plixee

www.plixee.com