ScalaJS + React = ♥

Présentation des GO

François LAROCHE

Lead Dev Scala chez MAKE. ORG

fl@make.org

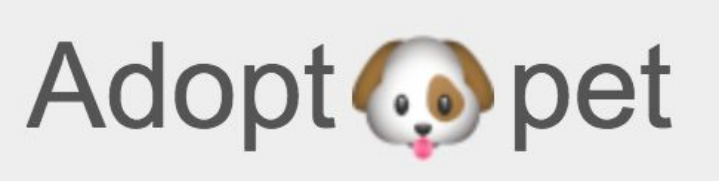
Colin SALMON-LEGAGNEUR

Dev Scala chez MAKE. ORG

cs@make.org

LUNATECH tinyclues NeoLynk Link to success Norauto iAdvize Teads lizeo group ebiznext BLECKWEN DEEP TRUTH REVEALED criteo labs

Objectif final:



Site d’adoption d’animaux de compagnie

Plan

- Présentation ScalaJS
- Construire son application
- ReactJS
- Tests unitaires
- ScalaCSS
- Les façades
- Router
- Communication avec le backend

Du JS … en Scala

ScalaJS:

- Web apps robustes
- Fortement typé
- Compatibles avec Javascript

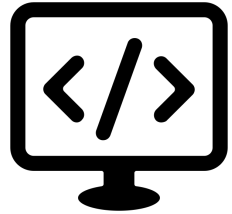
Tooling

sbt et webpack à la rescousse !

- compilation / transpilation incrémentale avec sbt
- le serveur dev webpack pour servir le tout !
- Le plugin scalajs-bundler pour lier le tout

À vos claviers !!

Les mains dans le camboui !



TODO:

- Clone du projet
- Lancement du dev-server en version incrémentale
- Jouer avec le js

URL du repo: tinyurl.com/workshop-scalajs

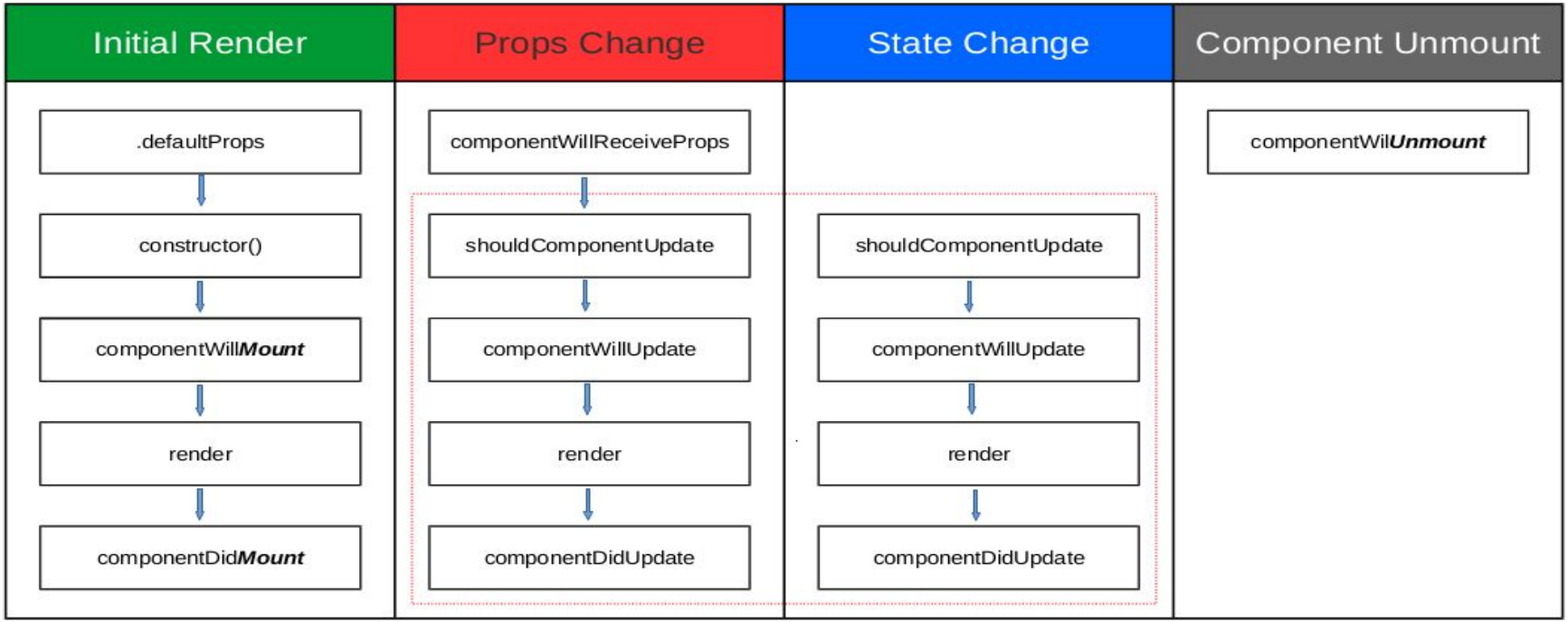
React

Lib JS qui fonctionne par **composants**.

Ces composants se rendent dans le DOM via une méthode **render**.

Ils sont configurable par leur **props** et suivent un cycle de vie défini. Ils sont dynamiques grâce à leur **state**.

React cycle de vie des composants



Scalajs-React (version David Barry)

- Lib ScalaJS
- Façade ReactJS
- Composants React typés en Scala

Scalajs-React (version David Barry)

composant stateless :

```
object HelloWorld {

 private val component = ScalaComponent
   .builder[Unit]("HelloWorld")
   .renderStatic(<.p("Hello World!"))
   .build

 def apply() = component()

}
```

composant avec backend :

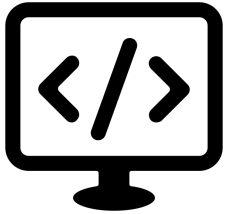
```
object HelloScalaist {

 case class HelloScalaistProps(name: String)

 class Backend($ : BackendScope[HelloScalaistProps,
Unit]) {
   def render(props: HelloScalaistProps): VdomElement =
     <.p(s"Hello ${props.name}!")
 }

 private val component = ScalaComponent
   .builder[HelloScalaistProps]("HelloScalaist")
   .renderBackend[Backend]
   .build

 def apply(name: String) =
component(HelloScalaistProps(name))
}
```

LUNATECH

tinyclues

NeoLyn
Link to success

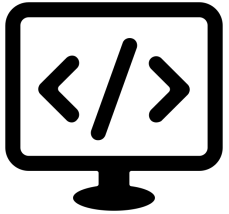
Gogogogogogogogogogogo !



TODO:

- Checkout sur la branche *next*
- Créer un composant statique
- Créer un composant avec *Backend*
- *Bonus:* créer un composant avec modification de state

Gogogogogogogogogogogo !

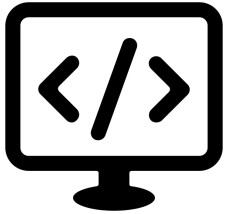


TODO (1/2):

Créer un composant statique:

- *Description*
- Affiche un lorem ipsum

Gogogogogogogogogogogo !

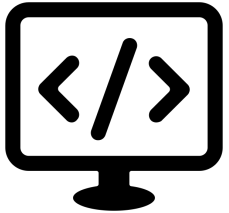


TODO (2/2):

Créer un composant avec Backend:

- *HelloPet*
- Prend un `*name*` en props
- Affiche “Hello $name!”

Gogogogogogogogogogogo !



TODO (Bonus):

Créer un composant avec modification de state:

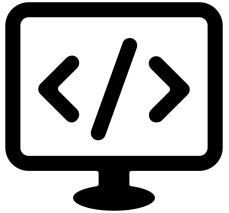
- *Counter*
- *CounterProps(initialCount: Int)*
- *CounterState(count: Int)*
- Affiche le `*count*` et un bouton pour l'incrémenter

Tests unitaires

- Dom émulé par JSDom
- Façade du testkit React (simule clicks, state …)
- [scalajs-react testing]

```
ReactTestUtils.withRenderedIntoDocument(MyComponent()) { component =>
  val link = ???
  Simulate.click(link)
  // Assert your assumptions
}
```

Tests unitaires



TODO:

- TU du DOM du composant *HelloPet*
- *Bonus:* test du compteur

ScalaCSS

Inline :

```
import scalacss.ScalaCssReact._
import scalacss.DevDefaults._

object Component {
 def render: VdomElement =
   React.Fragment(
     <.p(ComponentStyles.cyanText, "This is a cyan text"),
     ComponentStyles.render
   )
 /* ... */
}

object ComponentStyles extends StyleSheet.Inline {
 import dsl._

 val cyanText: StyleA = style(color.cyan)
}
```

ScalaCSS

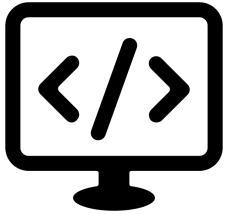
GlobalRegistry :

```
import scalacss.ScalaCssReact._
import scalacss.DevDefaults._
import scalacss.internal.mutable.GlobalRegistry

object App {
 GlobalRegistry.register(new ComponentStyles)

 def main() = {
   /* ... */
   GlobalRegistry.addToDocumentOnRegistration()
 }
}

object Component {
 val componentStyles: ComponentStyles = GlobalRegistry[ComponentStyles].get

 def render(): VdomElement =
   <.p(componentStyles.cyanText, "This is a cyan text")
}
```

LUNATECH

tinyclues NeoLynk Link to success Norauto iAdvize Teads liveo group ebiznext BLECKWEN criteo labs

ScalaCSS



TODO:

- Création d'une *Stylesheet*
- Register stylesheet dans la *GlobalRegistry*
- Utiliser cette Stylesheet
- …

Les façades

- Choisir la lib à façader
- Les annotations ScalaJS
- Le *js.Object*, *js.native* et autres types JS
- Interactions Scala ⇔ JS

Les façades : Scala ⇔ JS

Interoperabilité:

- `js.FunctionN` ⇔ `scala.FunctionN`
- `js.Array[T]` ⇔ `mutable.Seq[T]`
- `js.Dictionary[T]` ⇔ `mutable.Map[String, T]`
- `js.UndefOr[T]` ⇔ `Option[T]` (/!\ : null)
- `js.Promise[T]` ⇔ `Future[T]`

Les façades : Types JS

Statics:

```
@js.native
@JSImport("./images/dog.png", "default")
object dog extends js.Object
```

Libs:

```
@js.native
@JSImport("react-i18nify", "I18n")
object I18n extends js.Object {
 def setTranslations(
       translations: js.Any,
       rerenderComponents: Boolean = true): Unit =
   js.native
}
```

Les façades : Types JS

Instancier les objets JS:

```
@js.native
trait GradientColor extends js.Object {
 val from: String
 val to: String
}

object GradientColor {
 def apply(from: String, to: String): GradientColor = {
   js.Dynamic.literal(from = from, to = to).asInstanceOf[GradientColor]
 }
}
```

Les façades

TODO:

- Lib [react-i18nify]
- Façade *I18n.t*() + utilisation
- Façade de statics (json de traductions)
- Façade composant Translate
- Composant de bouton pour traduire le site

Router

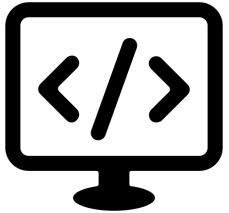
Router custom

- Pas une façade de ReactRouter
- *RouterCtl*, gestion URL ...
- Permet de typer ses pages
- Layout

Router

```
val routerConfig = RouterConfigDsl[PetPages].buildConfig { dsl =>
 import dsl._

 val home = staticRoute(root, HomePage) ~> renderR(Home(_))
 val petDetails = {
  dynamicRouteCT("#" / "pet" / string(".*").caseClass[PetDetailsPage]) ~> dynRenderR {
    (page: PetDetailsPage, routerCtl: RouterCtl[PetPages]) =>
      PetDetails(page.id, routerCtl)
  }
 }

 (trimSlashes
    | home
    | petDetails
 ).notFound(HomePage)
  .renderWith(layout)
}
```

Router



TODO:

- Layout
  - composant *Header*
  - composant *Footer*
  - `*main*` doit render le composant *Description*
- Route *#/pet/<pet-id>*
  - `*main*` rendre le composant *HelloPet*

Communication avec le backend

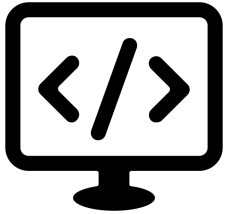
Client HTTP:

- Cross build
- Json ⇔ Scala avec circe
- Lib http au choix

Services:

- Par entité/feature (auth)
- Types Scala

Calls API



TODO:

- Appeler l’endpoint */pets/<pet-id>*

Créer un composant avec backend:

- *PetDetails*
- Affiche les informations d’un animal

Show must go on

TODO:

- Composant *PetTile*
- Liste d'animaux sur la home
- Router *SearchPets* + composants
- Formulaire d'adoption
- Validation du formulaire
- ...

Liens intéressants :

- ScalaJS
- ScalaJS-React
- ScalaCSS
- ScalaJS-Bundler
- tinyurl.com/workshop-scalajs
- React DevTools (chrome)
- scalajs-react testing
- scalajs-react router

En plus: les challenges de scalajs

Objectif production

- taille du JS
- compliqué à débugger

En plus: un mot sur le packaging

- fullOptJS vs. fastOptJS
- mode library vs. application
- webpack, minification ...

Merci à tous !

Hâte de voir vos apps ScalaJS en prod