Ocsigen

Vincent Balat

Epita — 12 mars 2014

Affiliation

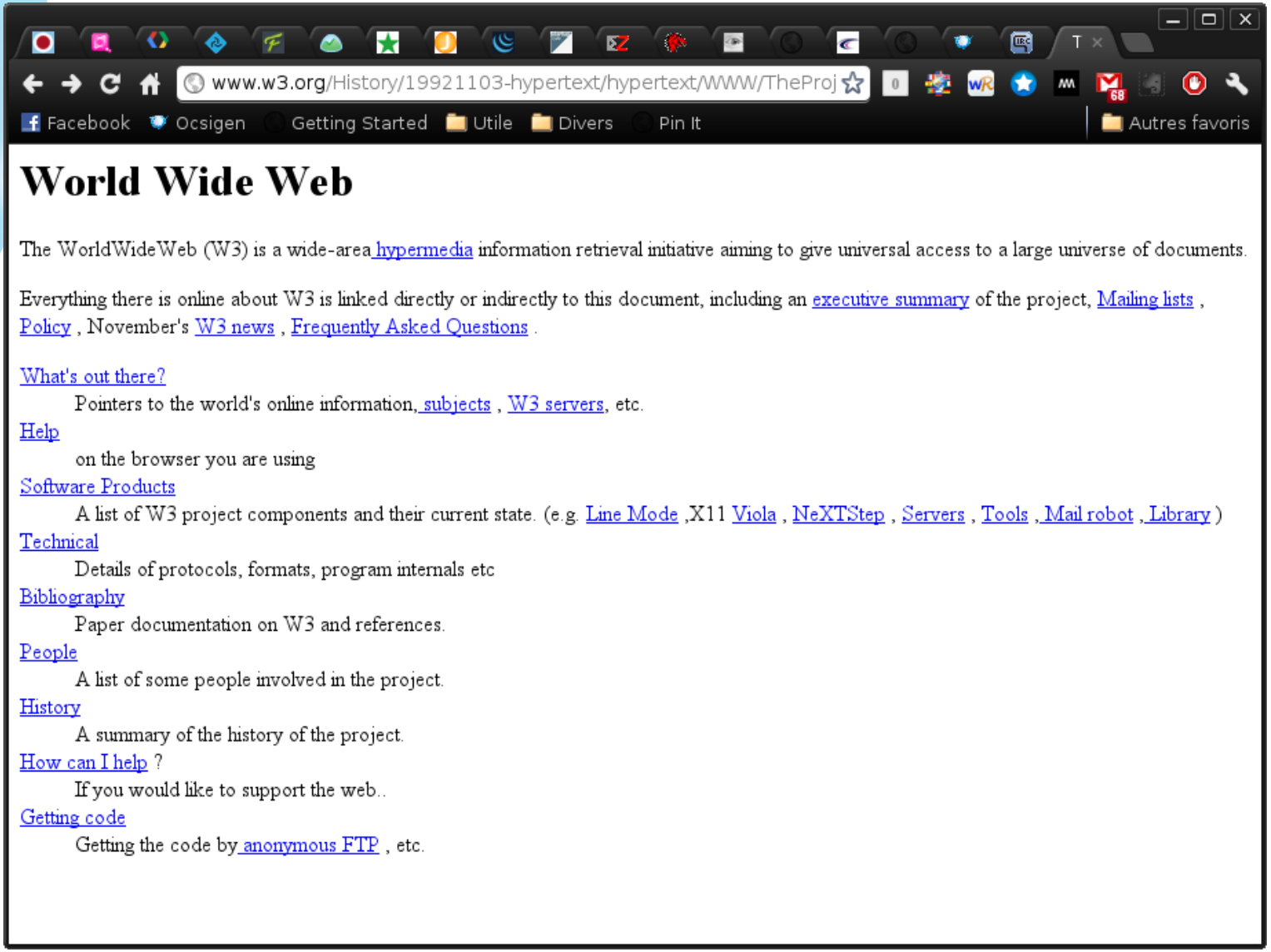
Ocsigen est logiciel libre issu d'un projet de recherche (Univ Paris Diderot, CNRS et Inria).

2 projets ANR

Collaborations avec UPMC, Inria Sophia Antipolis et Université Paris Nord.

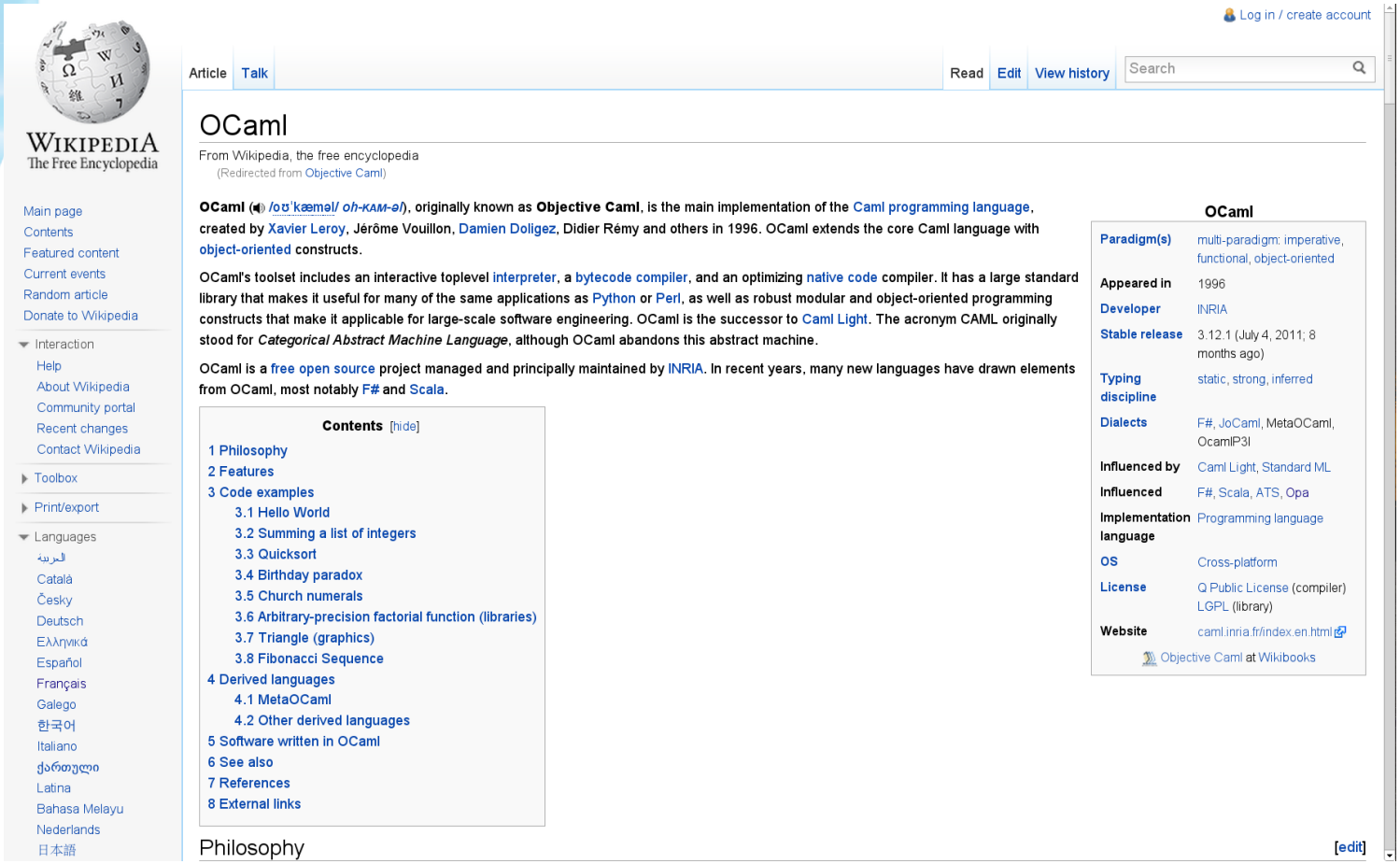
ocsigen
fresh air in web programming

L'évolution du Web



ocsigen
fresh air in web programming

L'évolution du Web



Log in / create account

WIKIPEDIA
The Free Encyclopedia

Article | Talk | Read | Edit | View history | Search

OCaml

From Wikipedia, the free encyclopedia
(Redirected from Objective Caml)

OCaml (/ɒkˈæməl/ *oh-кам-əl*), originally known as **Objective Caml**, is the main implementation of the Caml programming language, created by Xavier Leroy, Jérôme Vouillon, Damien Doligez, Didier Rémy and others in 1996. OCaml extends the core Caml language with object-oriented constructs.

OCaml's toolset includes an interactive toplevel interpreter, a bytecode compiler, and an optimizing native code compiler. It has a large standard library that makes it useful for many of the same applications as Python or Perl, as well as robust modular and object-oriented programming constructs that make it applicable for large-scale software engineering. OCaml is the successor to Caml Light. The acronym CAML originally stood for *Categorical Abstract Machine Language*, although OCaml abandons this abstract machine.

OCaml is a free open source project managed and principally maintained by INRIA. In recent years, many new languages have drawn elements from OCaml, most notably F# and Scala.

Contents [hide]

1 Philosophy
2 Features
3 Code examples
  3.1 Hello World
  3.2 Summing a list of integers
  3.3 Quicksort
  3.4 Birthday paradox
  3.5 Church numerals
  3.6 Arbitrary-precision factorial function (libraries)
  3.7 Triangle (graphics)
  3.8 Fibonacci Sequence
4 Derived languages
  4.1 MetaOCaml
  4.2 Other derived languages
5 Software written in OCaml
6 See also
7 References
8 External links

OCaml

Paradigm(s)	multi-paradigm: imperative, functional, object-oriented
Appeared in	1996
Developer	INRIA
Stable release	3.12.1 (July 4, 2011; 8 months ago)
Typing discipline	static, strong, inferred
Dialects	F#, JoCaml, MetaOCaml, OcamlP3l
Influenced by	Caml Light, Standard ML
Influenced	F#, Scala, ATS, Opa
Implementation language	Programming language
OS	Cross-platform
License	Q Public License (compiler) LGPL (library)
Website	caml.inria.fr/index.en.html

Objective Caml at Wikibooks

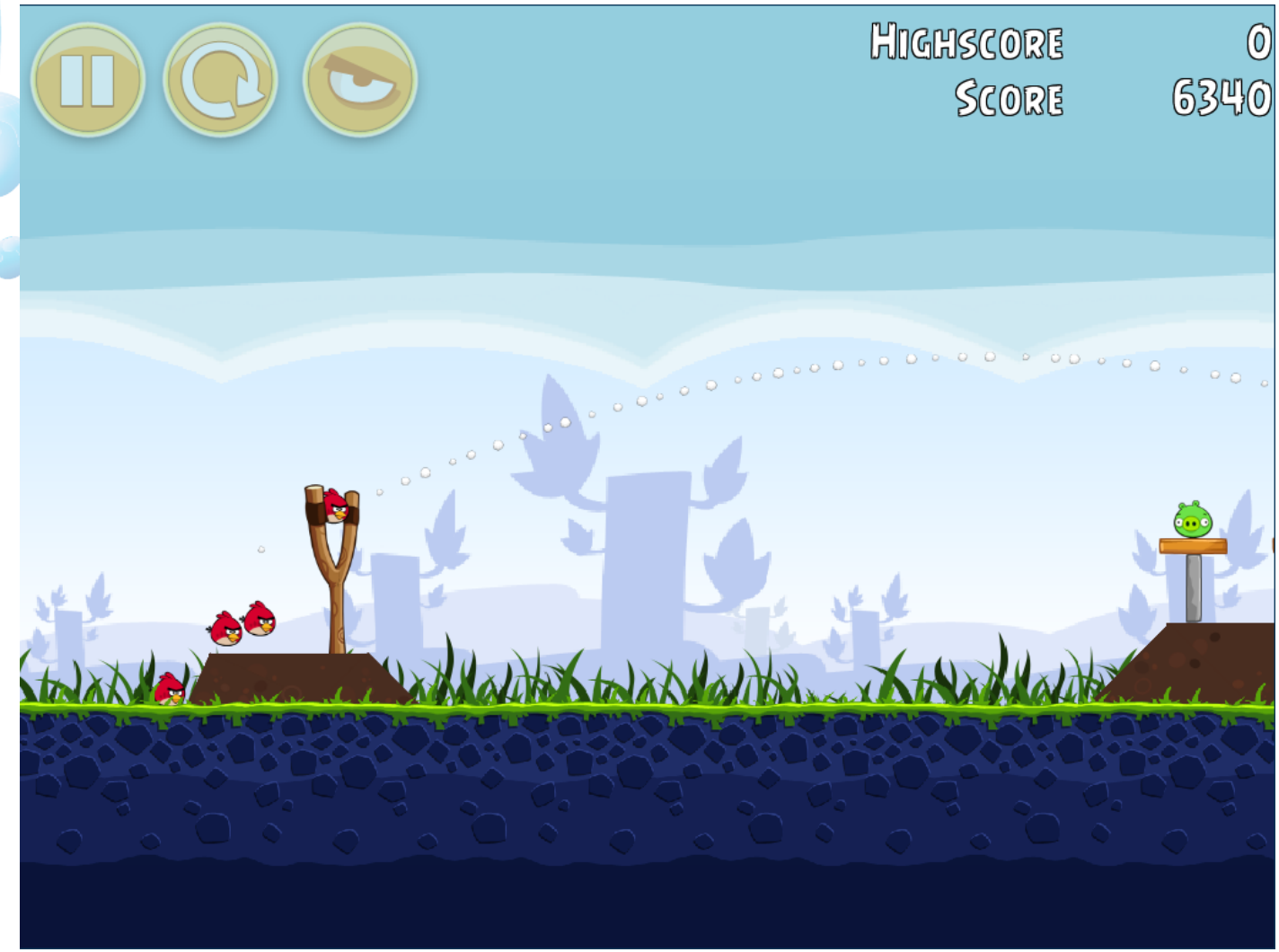
Philosophy [edit]

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export
Languages
العربية
Català
Česky
Deutsch
Ελληνικά
Español
Français
Galego
한국어
Italiano
ქართული
Latina
Bahasa Melayu
Nederlands
日本語

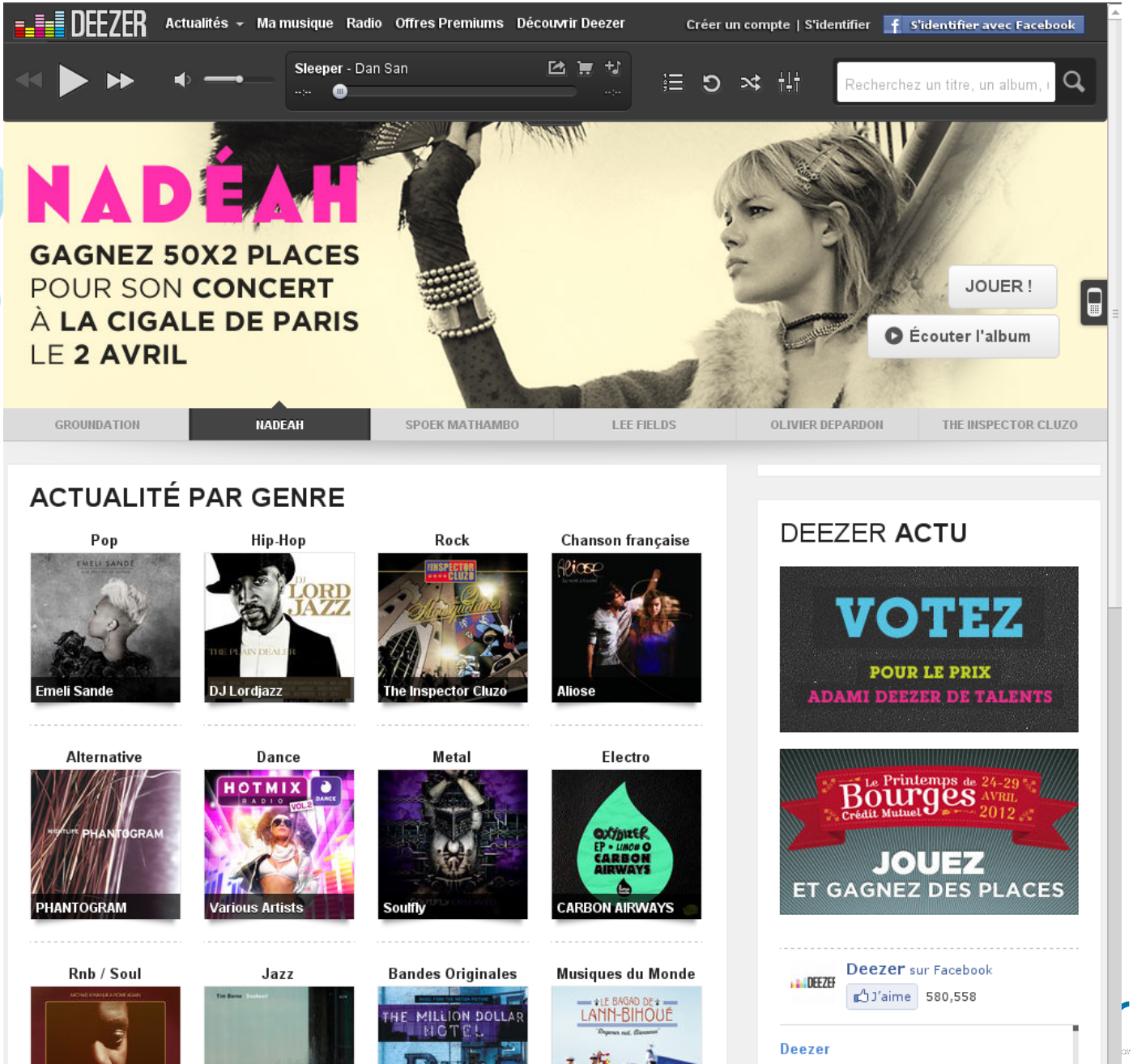
ocsigen
fresh air in web programming

L'évolution du Web

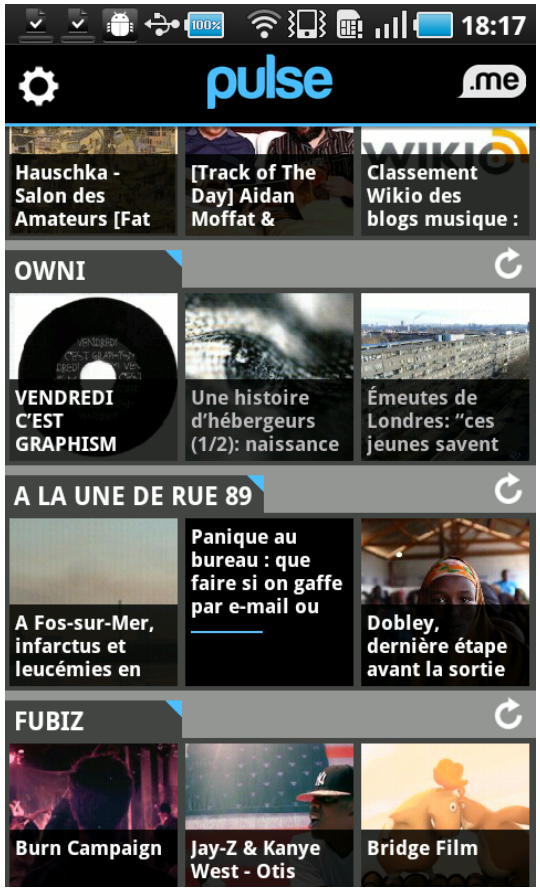
L'évolution du Web



L'évolution du Web

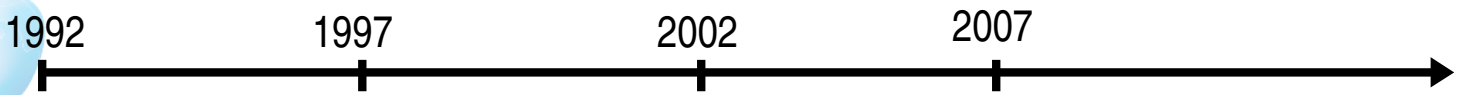


L'évolution du Web



ocsigen
fresh air in web programming

L'évolution du Web



Slashdot
linuxfr
Yahoo mail
CaraMail
Amazon

phpBB
Wikipedia

Twitter
Facebook
Google+
Farmville
Angry birds
Google documents
Pearltrees
Mindmeister
Pivotal Tracker

Servlets
PHP
JSP
ASP
Ruby on rails

Javascript
Flash
HTML5

Qu'est-ce qu'Ocsigen ?

Un framework Web pour :

- les applications HTML5 avec beaucoup d'interactions client / serveur
- les sites Web traditionnels (signets, bouton back, liens, etc)
- Les deux ensemble (le programme client ne s'arrête pas pendant la navigation)

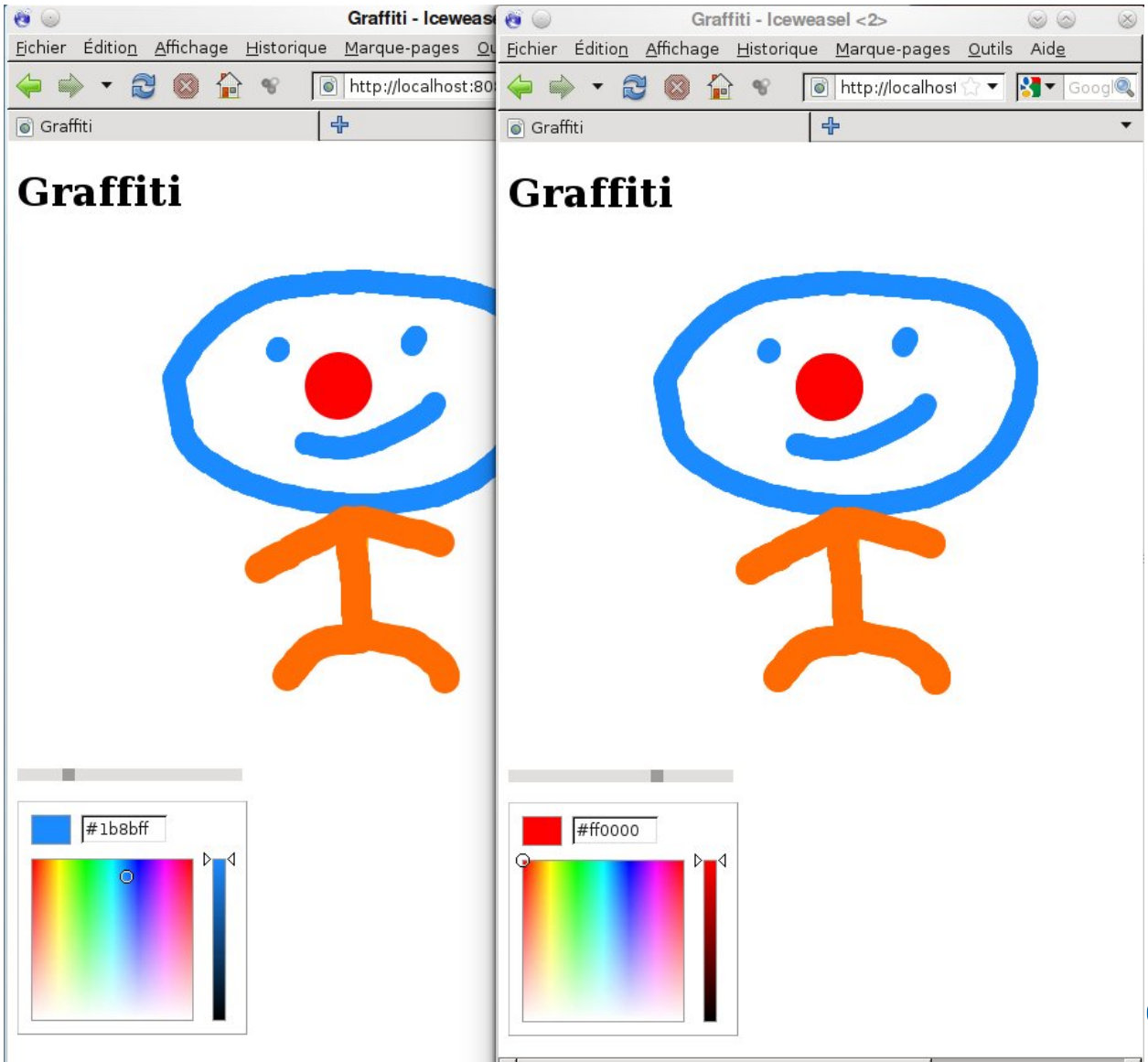
Exemple

http://ocsigen.org/graffiti

Exemple :
une application de dessin

ocsigen
fresh air in web programming

Exemple :
une application de dessin collaborative



csigen
fresh air in web programming

Le code de *Graffiti* en entier

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}
module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}
```

```
let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

ocsigen
fresh air in web programming

Le code de *Graffiti* en entier

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}
module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}
```

```
let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

- Court

Le code de *Graffiti* en entier

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}
module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}
```

```
let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

- Court
- Un seul code

ocsigen
fresh air in web programming

Le code de *Graffiti* en entier

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

- Court
- Un seul code
  - côté server
  - code partagé
  - code client , compilé vers JS

ocsigen
fresh air in web programming

Une application Web client/server comme un seul programme

Avantages :

- Même language
- Mêmes structures de données — pas besoin de conversions
- Code partagé
- Générer des portions de la page soit sur le server (indexation...) soit le client (mises à jour)
- Appels de fonctions distants
- Utiliser des variables serveur depuis le côté client
- …

Les buts d'Ocsigen

Expressivité

- Expressivité du langage lui-même
- Concepts de haut niveau adaptés aux besoins des développeurs Web.

Securité

- Garanties de sécurité offertes par le langage ou par Ocsigen
  (pas d'injection de code possible,...)

Améliorer la fiabilité, Réduire le temps de débogage

Grâce à un **langage compilé, avec un typage statique puissant**

- Le HTML respecte les recommandations du W3C (Vérifié à la compilation !)
- URLs générées dynamiquement $\Rightarrow$ pas de liens morts
- ...

Plan

OCaml

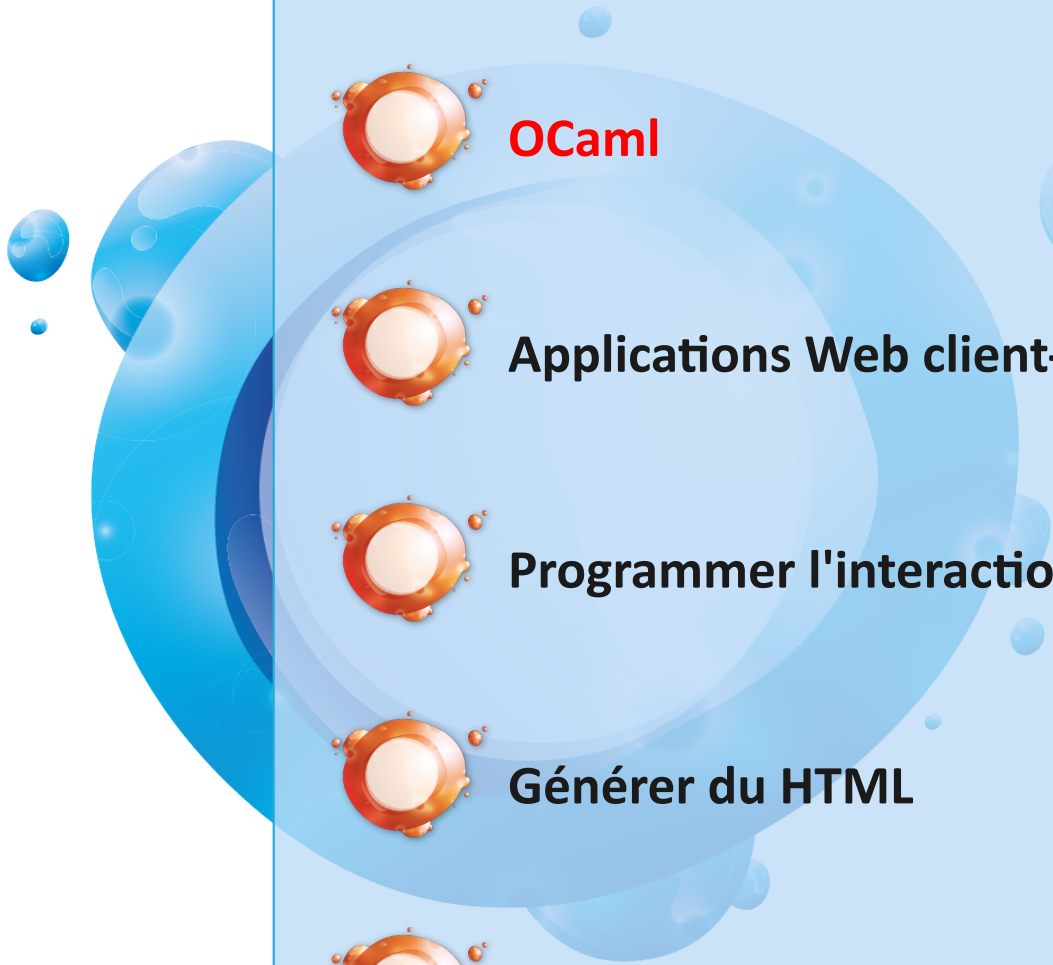
Applications Web client-serveur

Programmer l'interaction Web traditionelle

Générer du HTML

Conclusion

ocsigen
fresh air in web programming



- OCaml
- Applications Web client-serveur
- Programmer l'interaction Web traditionelle
- Générer du HTML
- Conclusion

OCaml

- Très expressif: fonctionnel, orienté objet, modules paramétrés …
- Système de types très riche (typage statique !)
- Langage compilé (rapide)
- Syntaxe extensible
- Nombreuses bibliothèques et bindings
- Communauté d'utilisateurs, utilisateurs industriels

Quelques utilisateurs d'OCaml

Jane Street Capital Transactions financières

Xen -- Citrix 15% des machines virtuelles, dont Amazon

Airbus

OCaml Labs (Cambridge UK)

Ocamlpro (Paris)

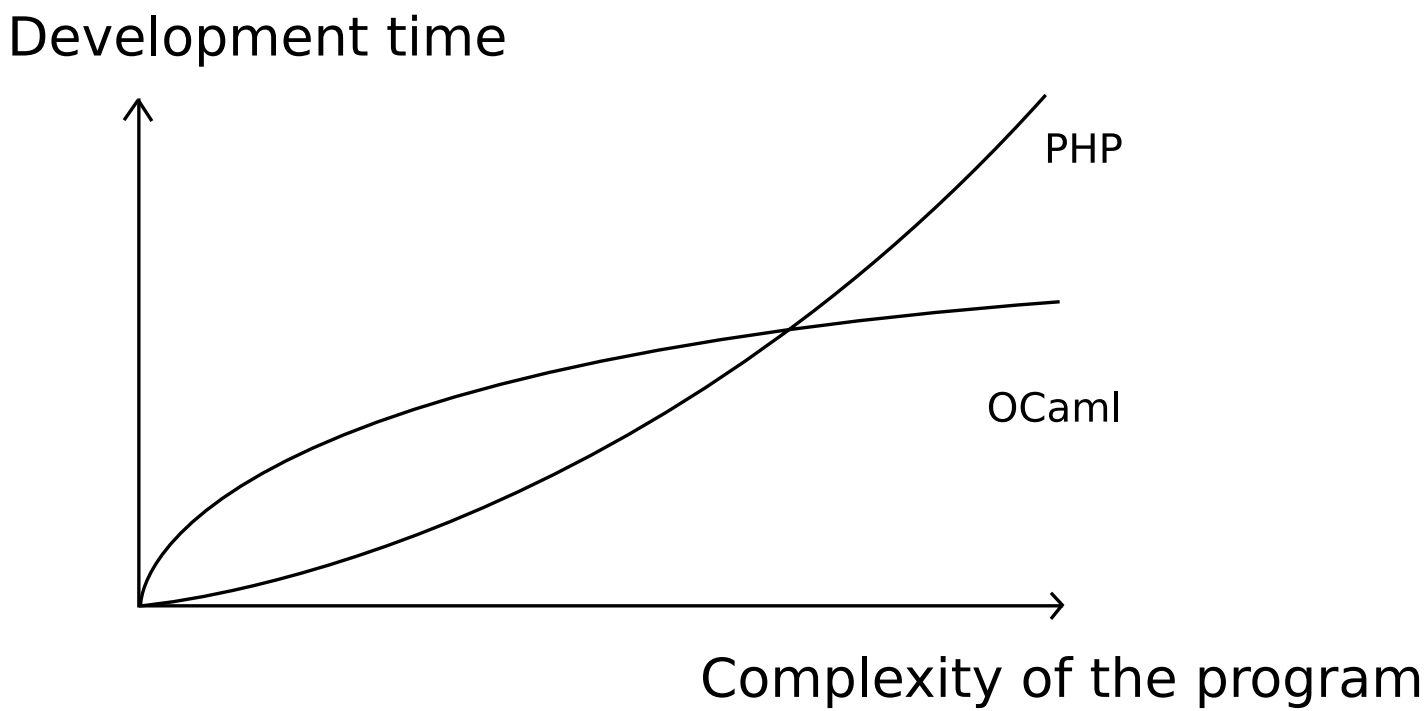
La plupart des **prouveurs de programmes**

Microsoft (F#), Dassault,...

Facebook

...

Influence du typage sur le temps de développement



js_of_ocaml

Un compilateur d'OCaml vers JS.

js_of_ocaml

Un compilateur de bytecode OCaml vers JS.

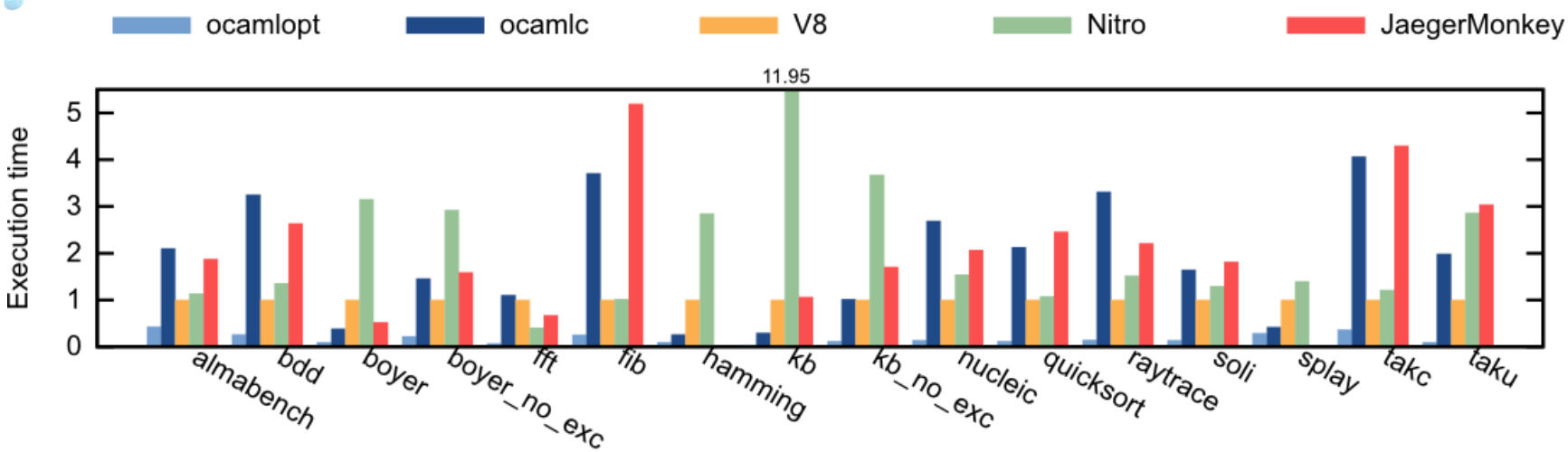
Un compilateur de bytecode OCaml vers JS.

Très facile à utiliser

Possibilité d'utiliser des bibliothèques déjà compilées (même non libres)

js_of_ocaml

Un compilateur de bytecode OCaml vers JS.

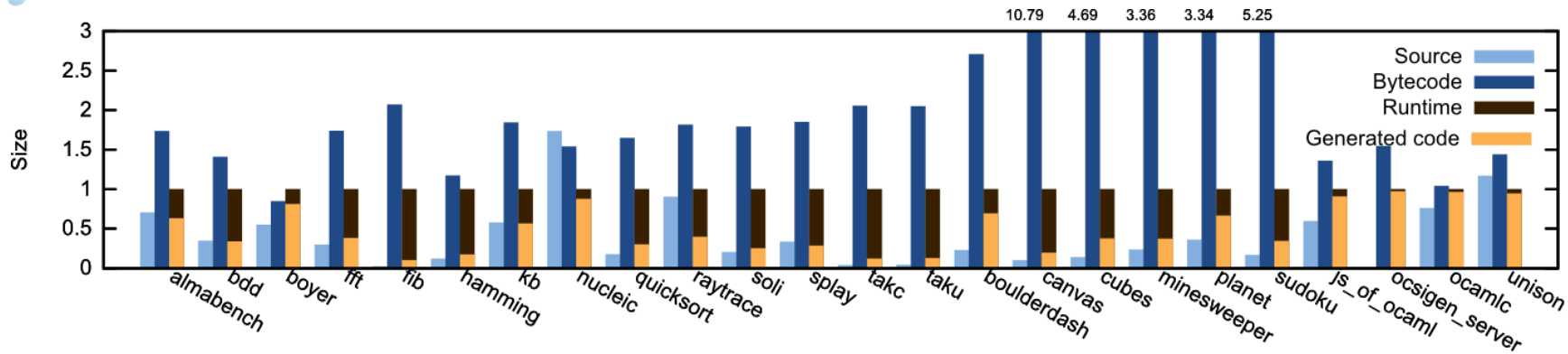


Rapide !

ocsigen
fresh air in web programming

js_of_ocaml

Un compilateur de bytecode OCaml vers JS.



Concis !

ocsigen
fresh air in web programming

Js_of_ocaml : Appeler des fonctions JS

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

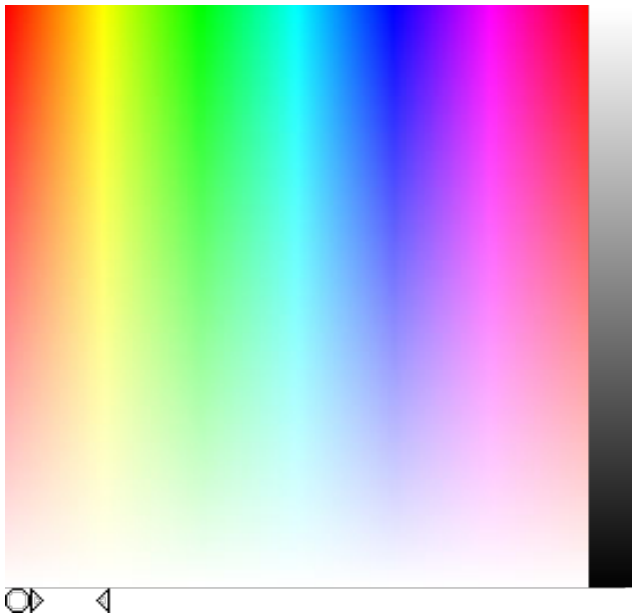
ocsigen
fresh air in web programming

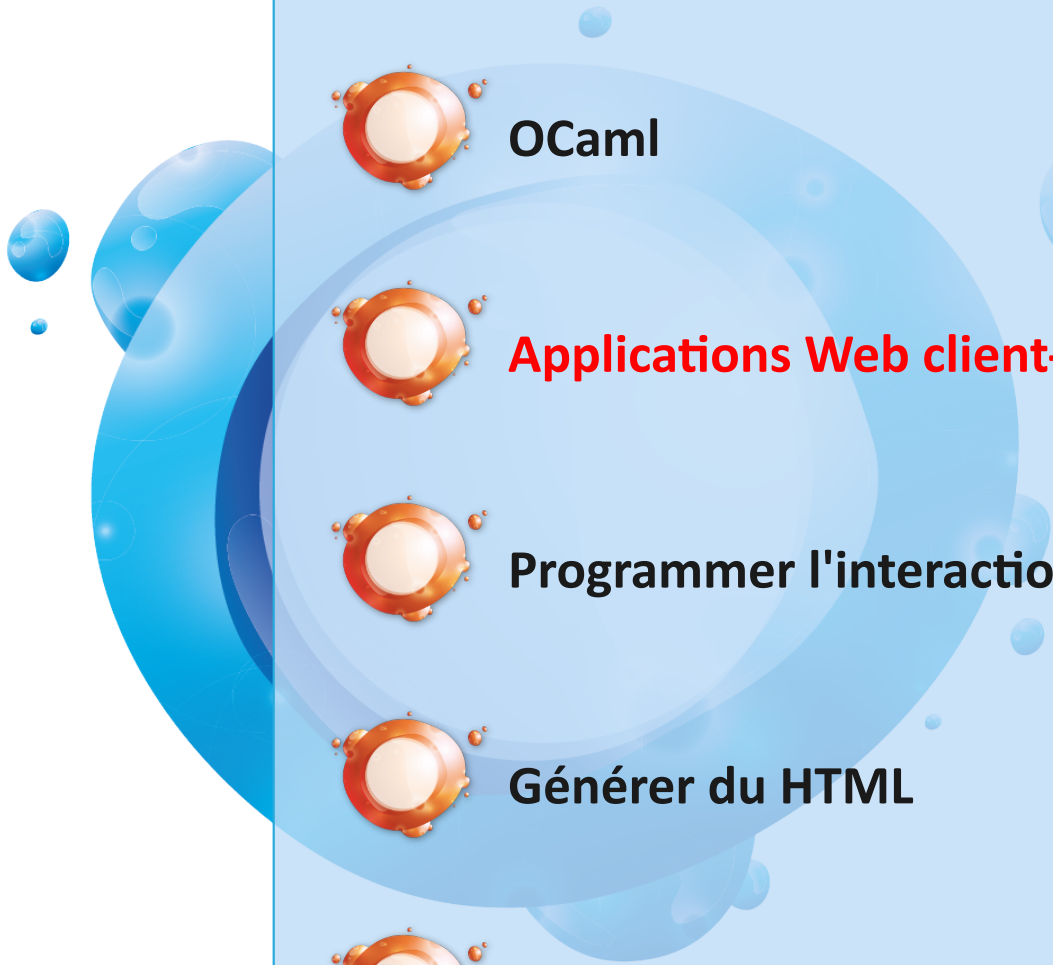
Js_of_ocaml : Appeler des fonctions JS

```
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- color;
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1);
  ctx##lineTo(float x2, float y2);
  ctx##stroke()
```

Dessiner des lignes

O'Closure: un binding pour la bibliothèque de widgets Google Closure





- OCaml
- Applications Web client-serveur
- Programmer l'interaction Web traditionelle
- Générer du HTML
- Conclusion

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}


let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

open, constantes, type

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

open, constantes, type

Initialisation de l'appl

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

open, constantes, type

Initialisation de l'appl

Bus

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

open, constantes, type

Initialisation de l'appl Bus

fonction `draw`

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

open, constantes, type

Initialisation de l'appl Bus

fonction `draw`

Service

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

HTML5 page

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

Code spécifique à la page

HTML5 page

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

Canvas

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

Canvas

Slider

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

Canvas

Slider

Palette de couleurs

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

Canvas

Slider

Palette de couleurs

Fonction
de calcul des coordonnées
et envoi au serveur

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

Canvas

Slider

Palette de couleurs

Fonction
de calcul des coordonnées
et envoi au serveur

Réactions aux évé. serveur

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

Canvas

Slider

Palette de couleurs

Fonction
de calcul des coordonnées
et envoi au serveur

Réactions aux évé. serveur

Évé. souris

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

open, constantes, type

Initialisation de l'appl
Bus

fonction draw

Service

Canvas

Slider

Palette de couleurs

Fonction
de calcul des coordonnées
et envoi au serveur

Réactions aux évé. serveur
Évé. souris

HTML5 page

ocsigen
fresh air in web programming

Ex de com. client-serveur : le bus

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_mod...
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
```

```
let b =
  Eliom_bus.create ~name:"graff" Json.t<messages>
```

Ex de com. client-serveur : le bus

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_mod...(struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1...
       slider##render(Js.so...

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
```

```
let b =
  Eliom_bus.create ~name:"graff" Json.t<messages>
```

```
Eliom_bus.write %b v
```

ocsigen
fresh air in web programming

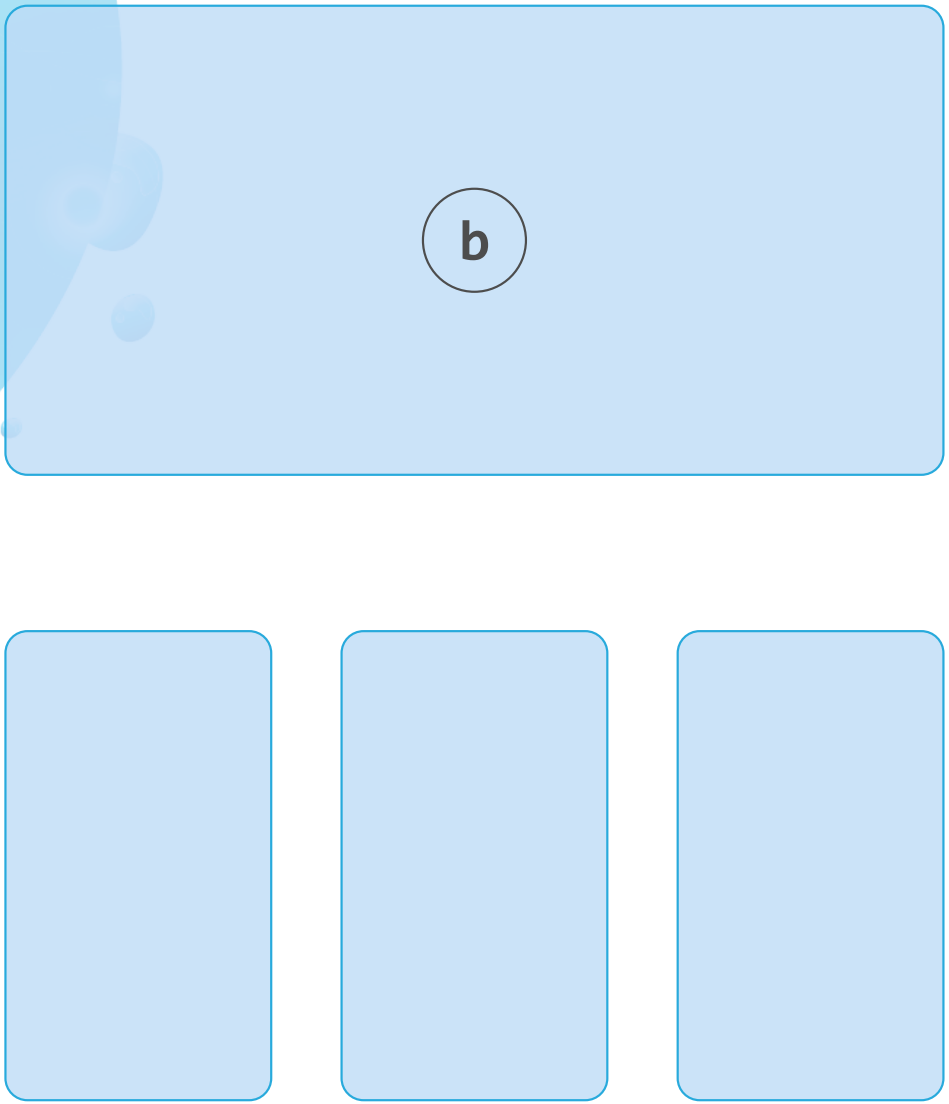
Ex de com. client-serveur : le bus

```
let b =
  Eliom_bus.create ~name:"graff" Json.t<messages>
```

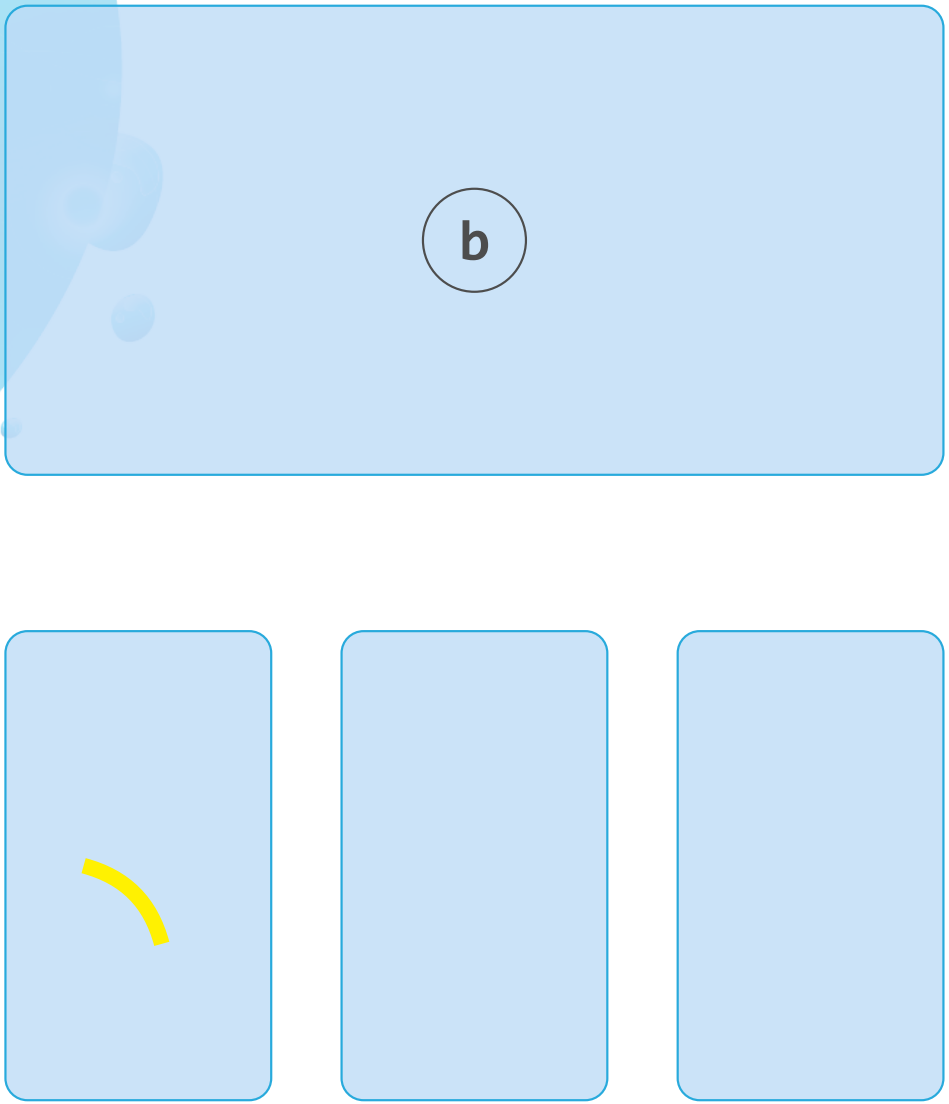
```
Eliom_bus.write %b v
```

```
Lwt_stream.iter (draw ctx) (Eliom_bus.stream %b)
```

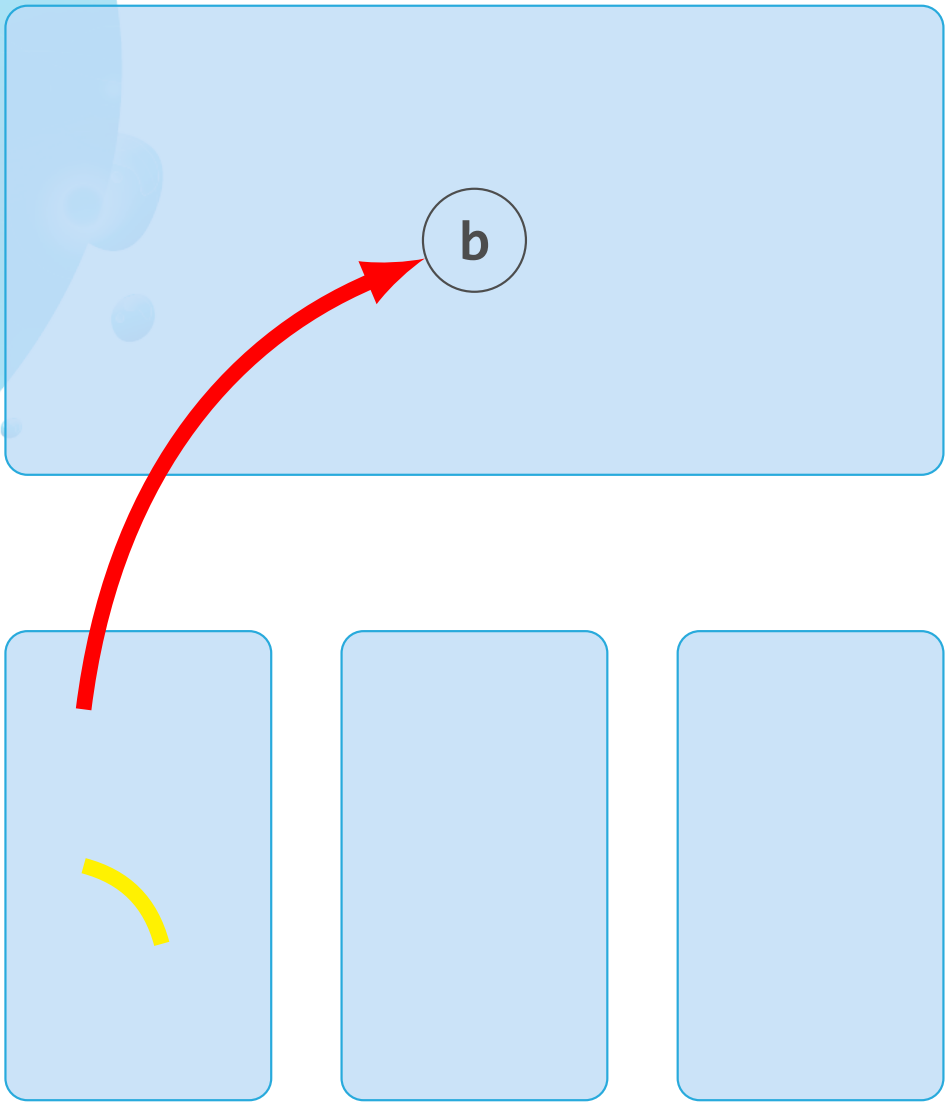
Bus client-server



Bus client-server

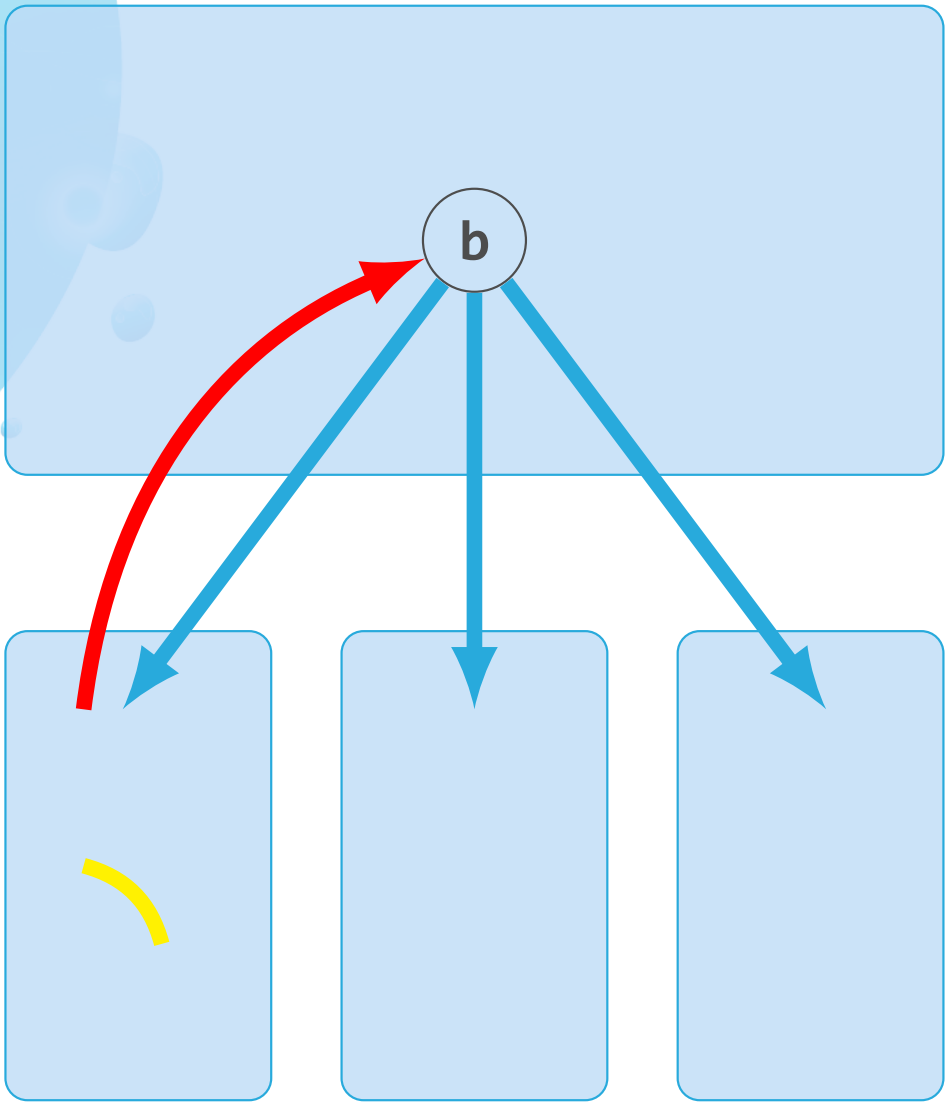


Bus client-server



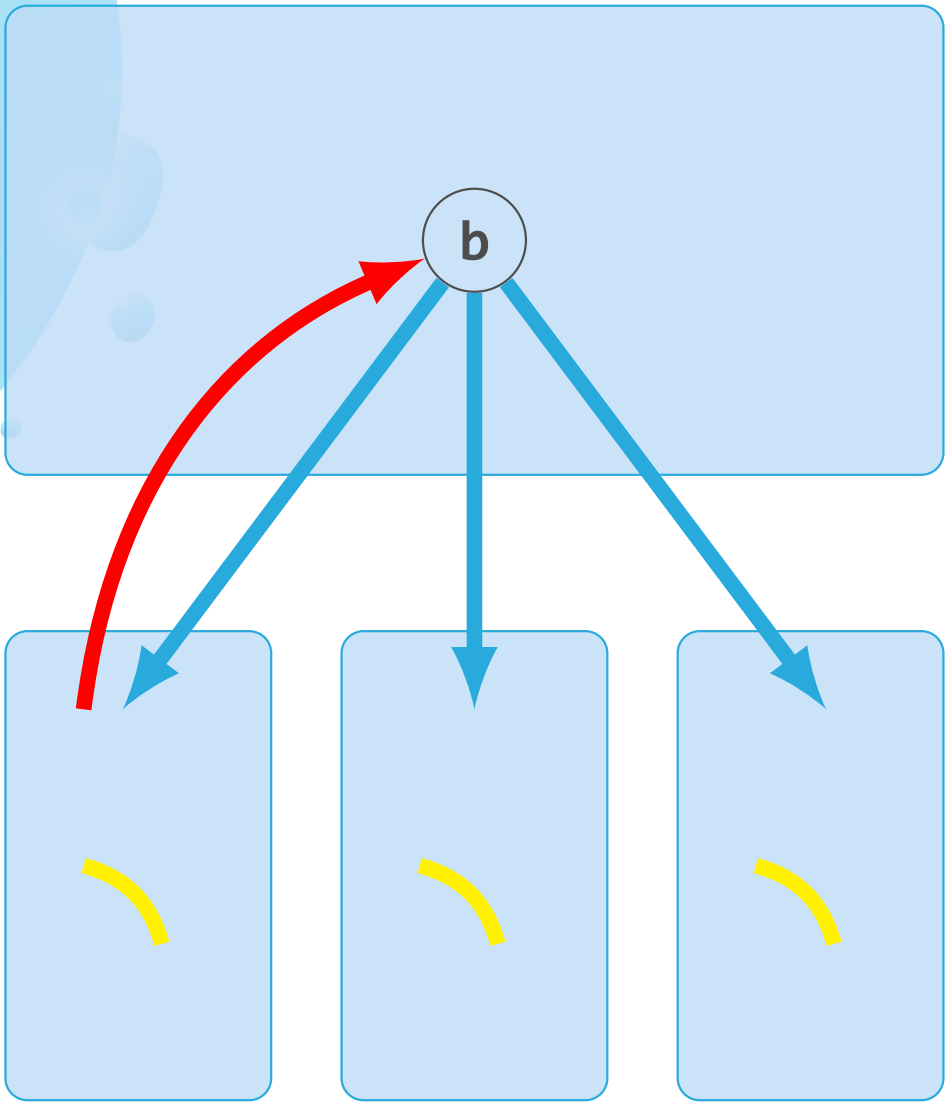
call `write`

Bus client-server

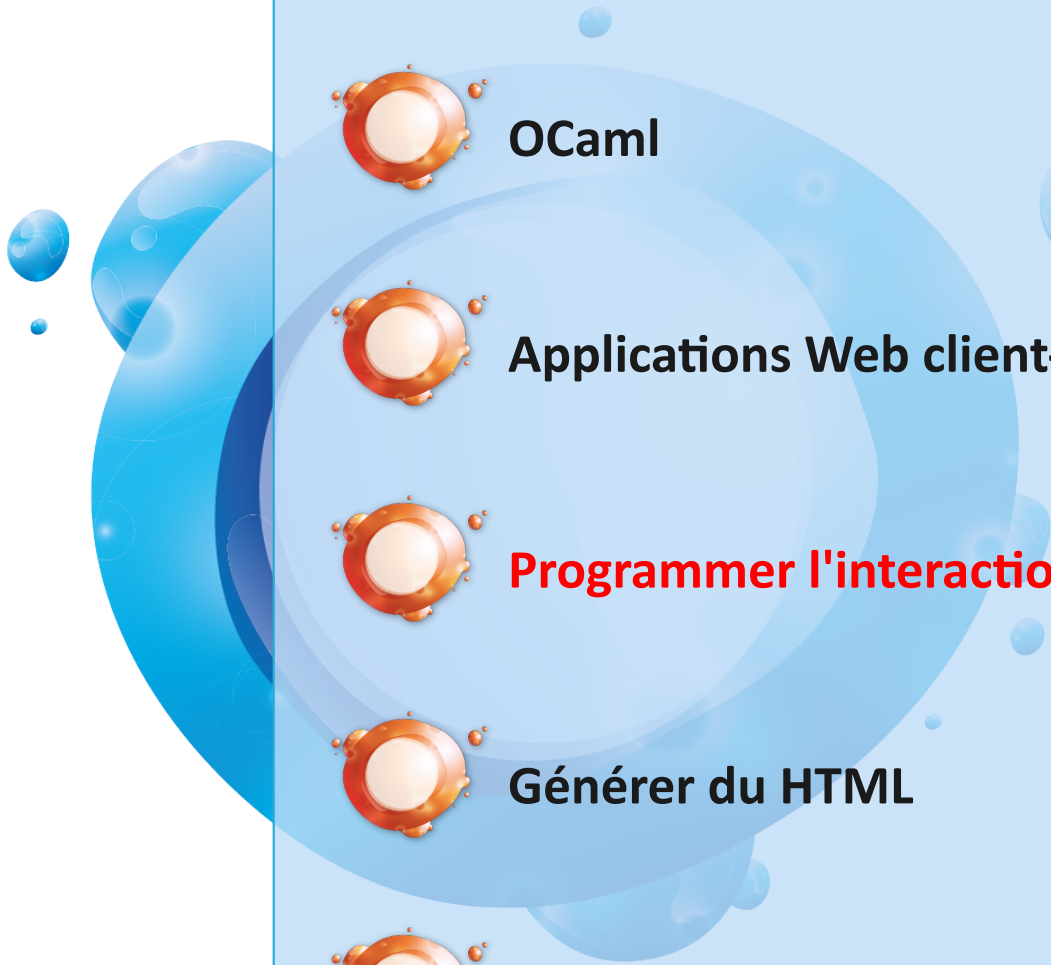


listen on bus b

Bus client-server



listen on bus b



- OCaml
- Applications Web client-serveur
- Programmer l'interaction Web traditionelle
- Générer du HTML
- Conclusion

Services

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       (html
          (head
             (title (pcdata "Graffiti"))
             [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
              script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
          (body [h1 [pcdata "Graffiti"]])))
```

```
let main_service = register_service
                     ~path:[""]
                     ~get_params:unit
                     (fun () () -> ... )
```

Services

```
let main_service = register_service
                     ~path:[""]
                     ~get_params:unit
                     (fun () () -> ... )
```

- HTML
- Fichier
- Redirection
- Action
- Application
- ...

ocsigen
fresh air in web programming

Programmer l'interaction Web traditionelle avec les services d'Ocsigen

1. Les services sont des valeurs de première classe
2. Liens et formulaires sont vérifiés statiquement
3. Les données générées sont vérifiés statiquement
4. Création dynamique de nouveaux services
5. Mécanisme puissant d'identification de services, basé sur :
   - L'URL ou un identifiant de service (comme paramètre)
   - Le méthode HTTP (GET or POST)
   - Le nom des paramètres
   - La session (navigateur)
   - L'onglet qui fait la requête
   - …

ocsigen
fresh air in web programming

Services

service : (GET and POST parameters) $\longrightarrow$ page

Ocsigen vérifie la présence des paramètres
et les traduit automatiquement vers des types OCaml
(integers, booleans,... et même des ensembles ou listes !)

Liens et formulaires

Les services sont des *valeurs de première classe*

Liens et formulaires

Les services sont des *valeurs de première classe*

<a> ne prend pas l'*URL* comme paramètre,
mais le *service* !

⇒ *Pas de liens cassés !*

Liens et formulaires

Les services sont des *valeurs de première classe*

<a> ne prend pas l'*URL* comme paramètre,
mais le *service* !

⇒ *Pas de liens cassés !*

Conformité des formulaires par rapport aux services vérifiée à la compilation

Mécanisme d'identification de services

Ocsigen choisit le service automatiquement

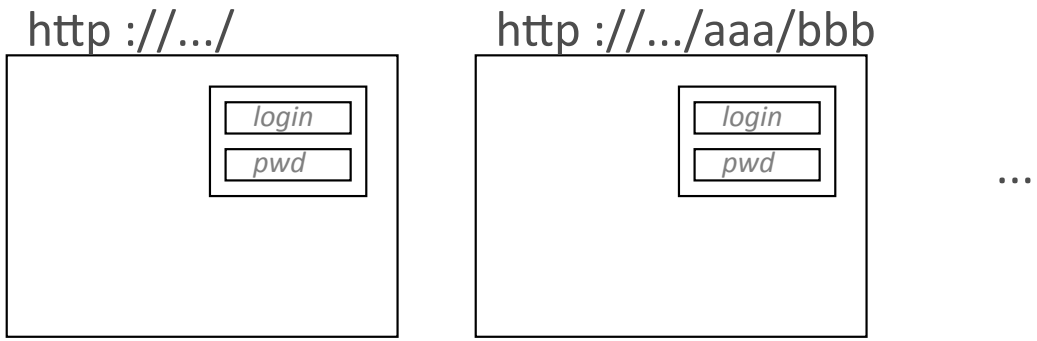
en fonction :

- De l'URL et/ou d'un paramètre interne
- De la méthode HTTP (GET ou POST)
- Du nom des paramètres
- La session de l'utilisateur
- …

⟶ Le programmeur choisit le style d'identification dont il a besoin.

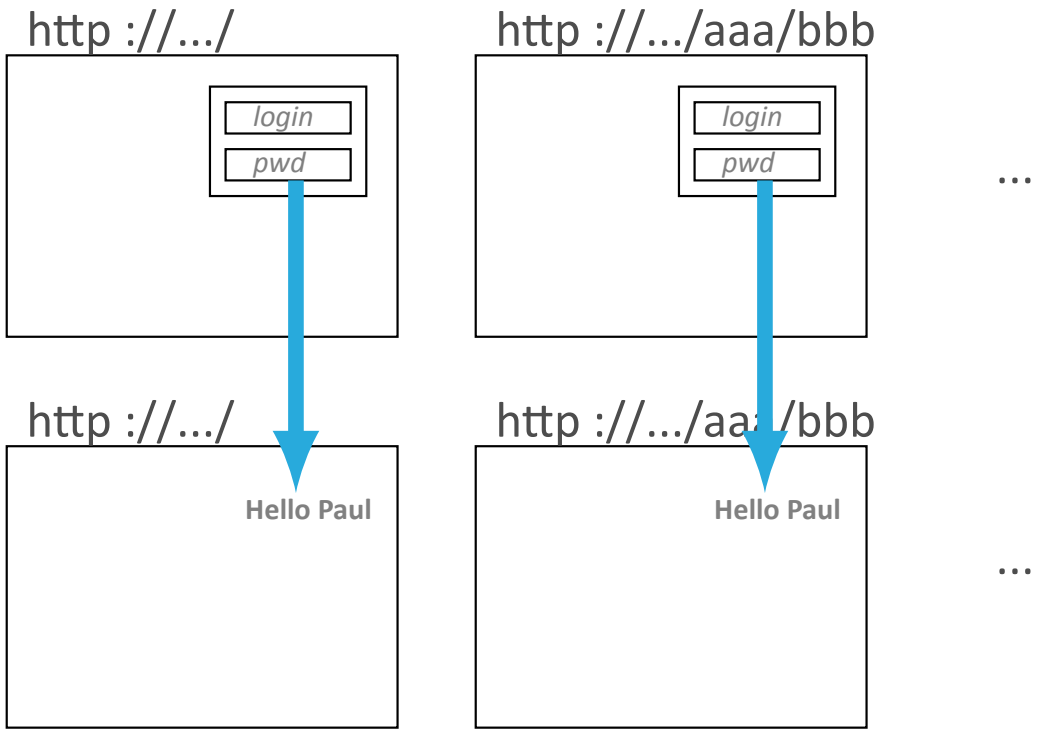
Mécanisme d'identification de services

Exemple : connection d'utilisateur depuis n'importe quelle page



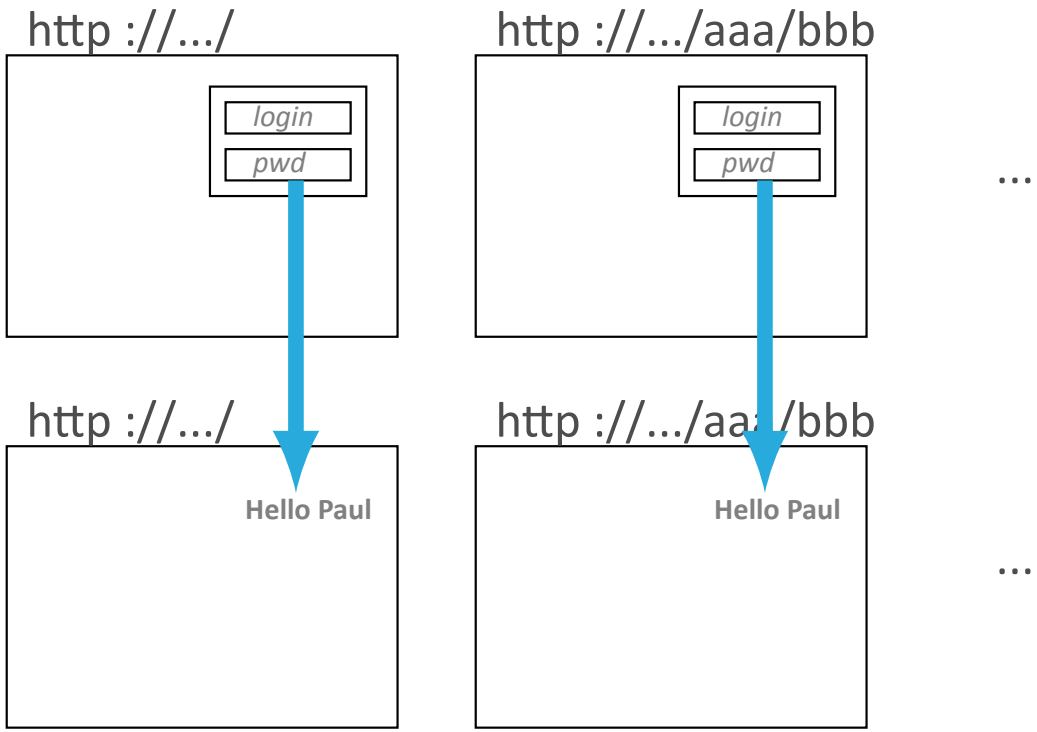
Mécanisme d'identification de services

Exemple : connection d'utilisateur depuis n'importe quelle page



Mécanisme d'identification de services

Exemple : connection d'utilisateur depuis n'importe quelle page

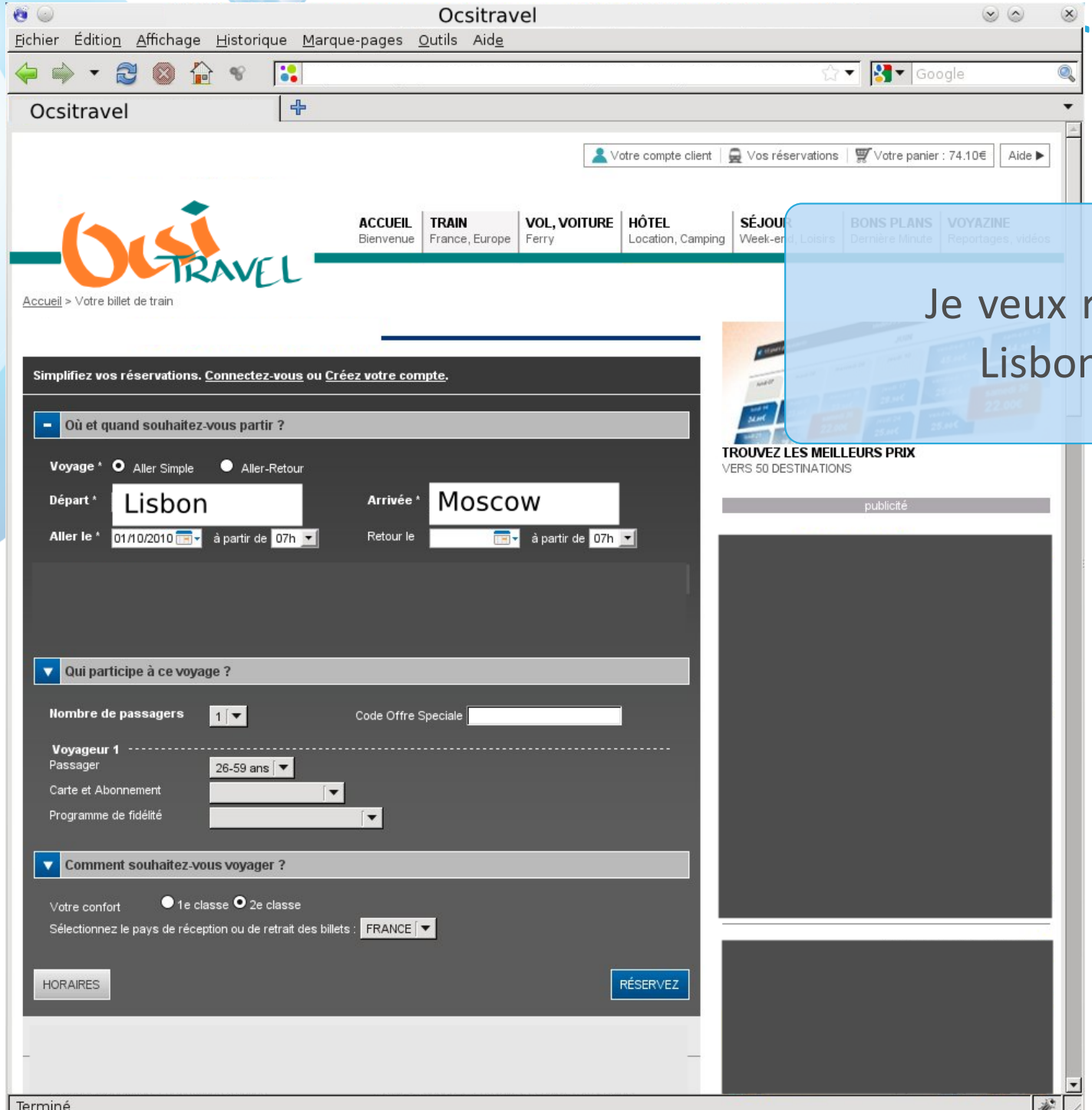


L'*action* de « se connecter » est un service *caché* disponible pour toutes les URLs.

Deux sortes d'interaction Web

Exemple : réserver un billet d'avion

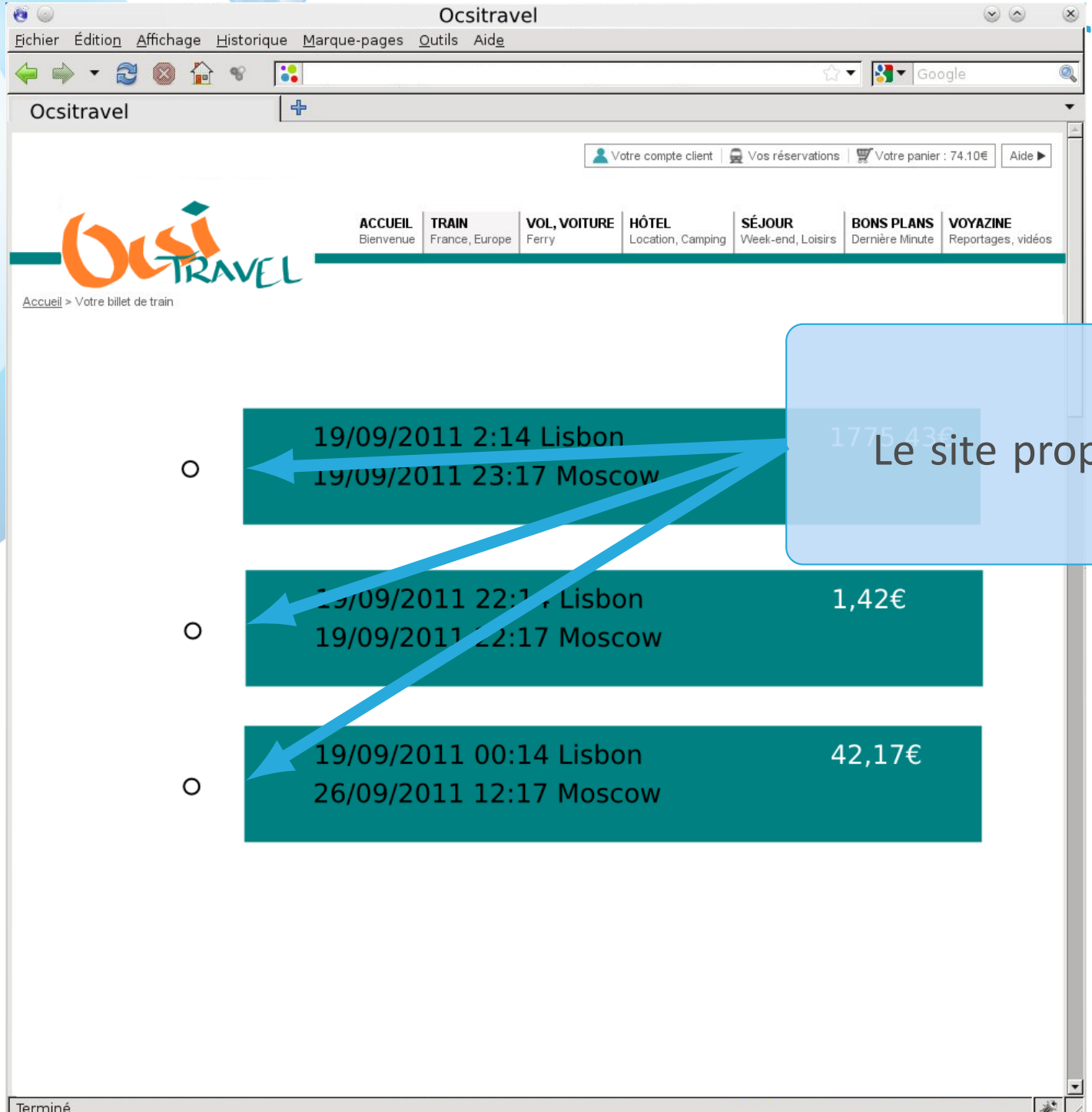
raction Web



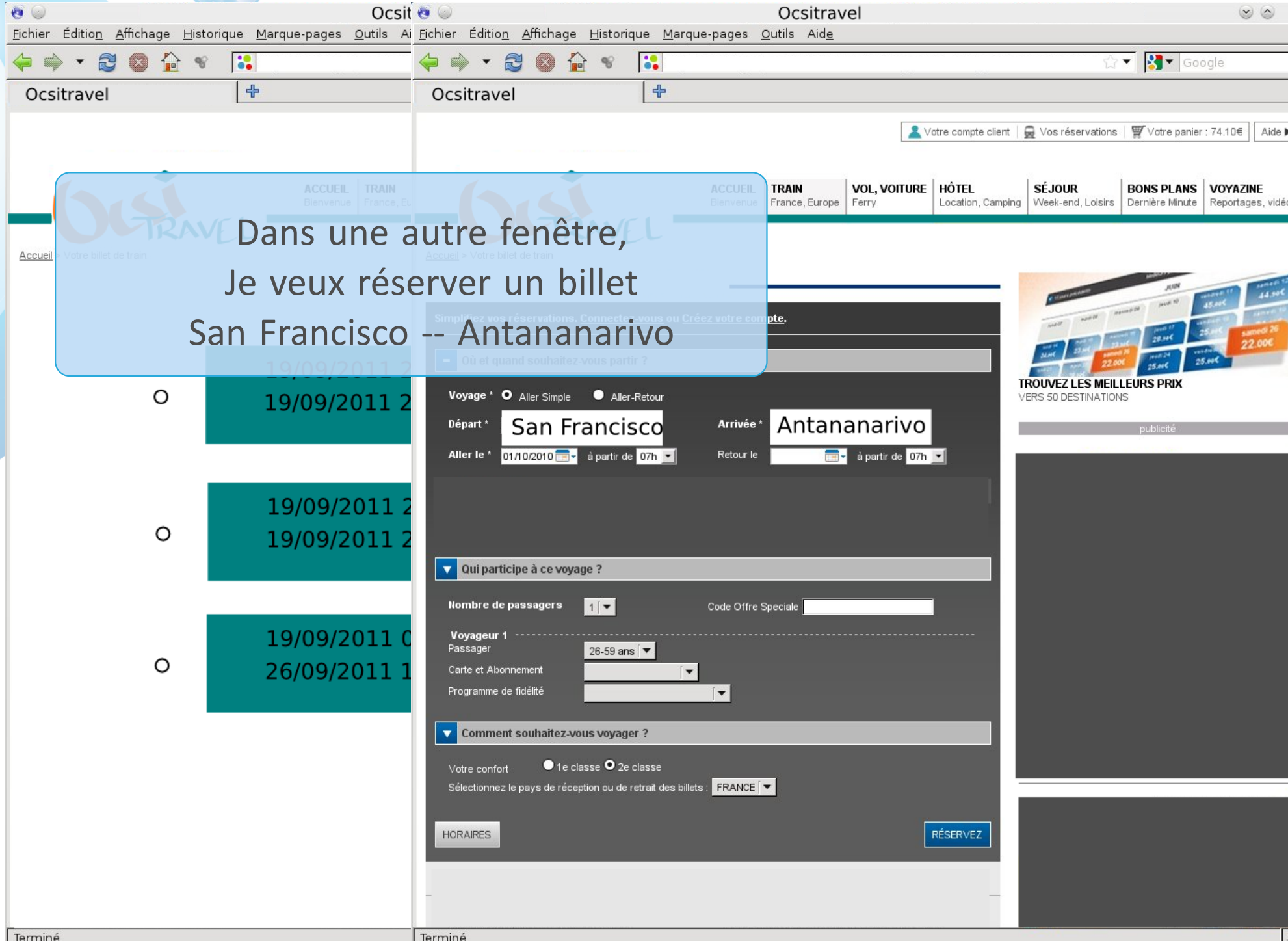
Je veux réserver un billet
Lisbonne -- Moscou

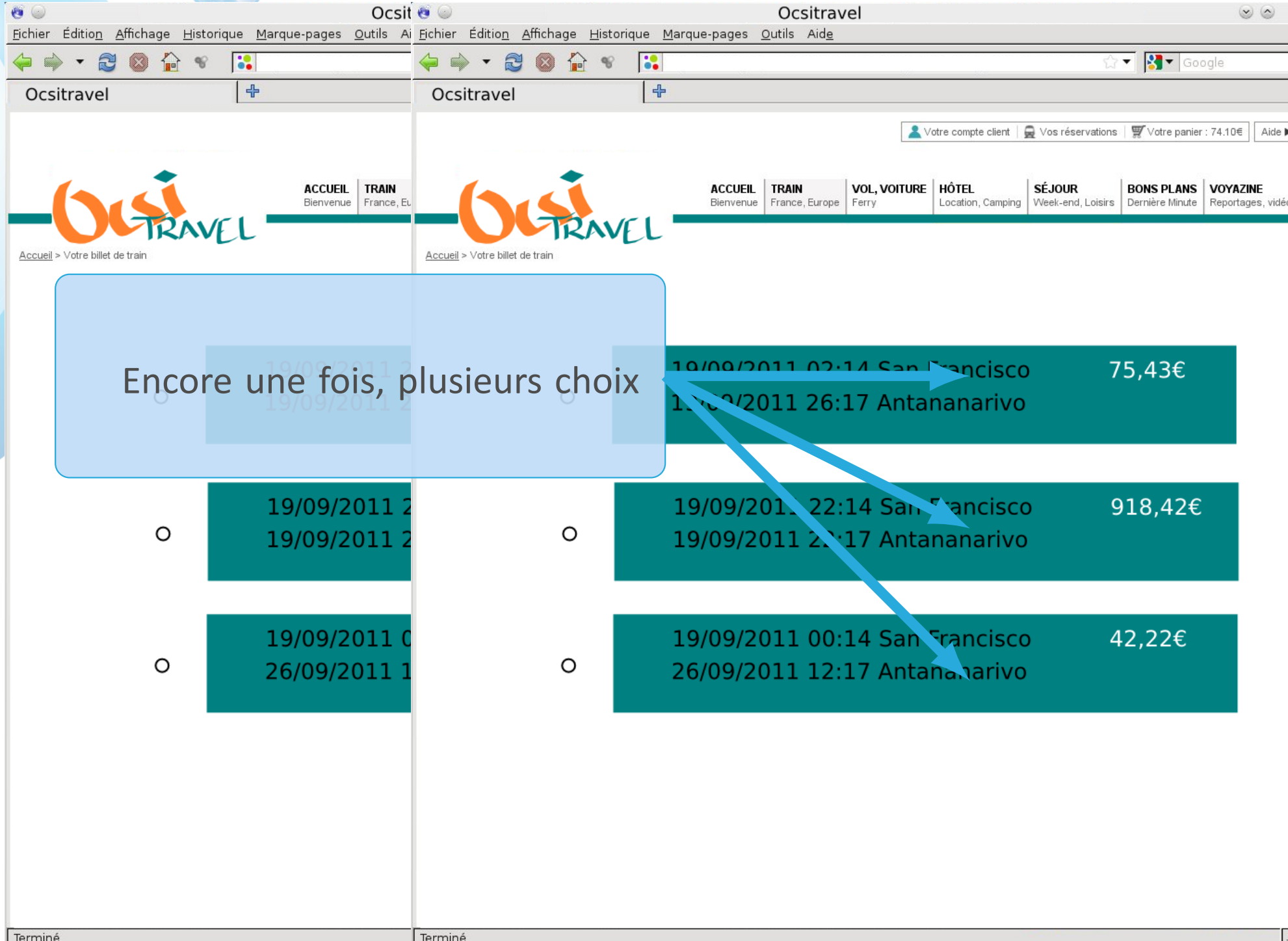
ocsigen
fresh air in web programming

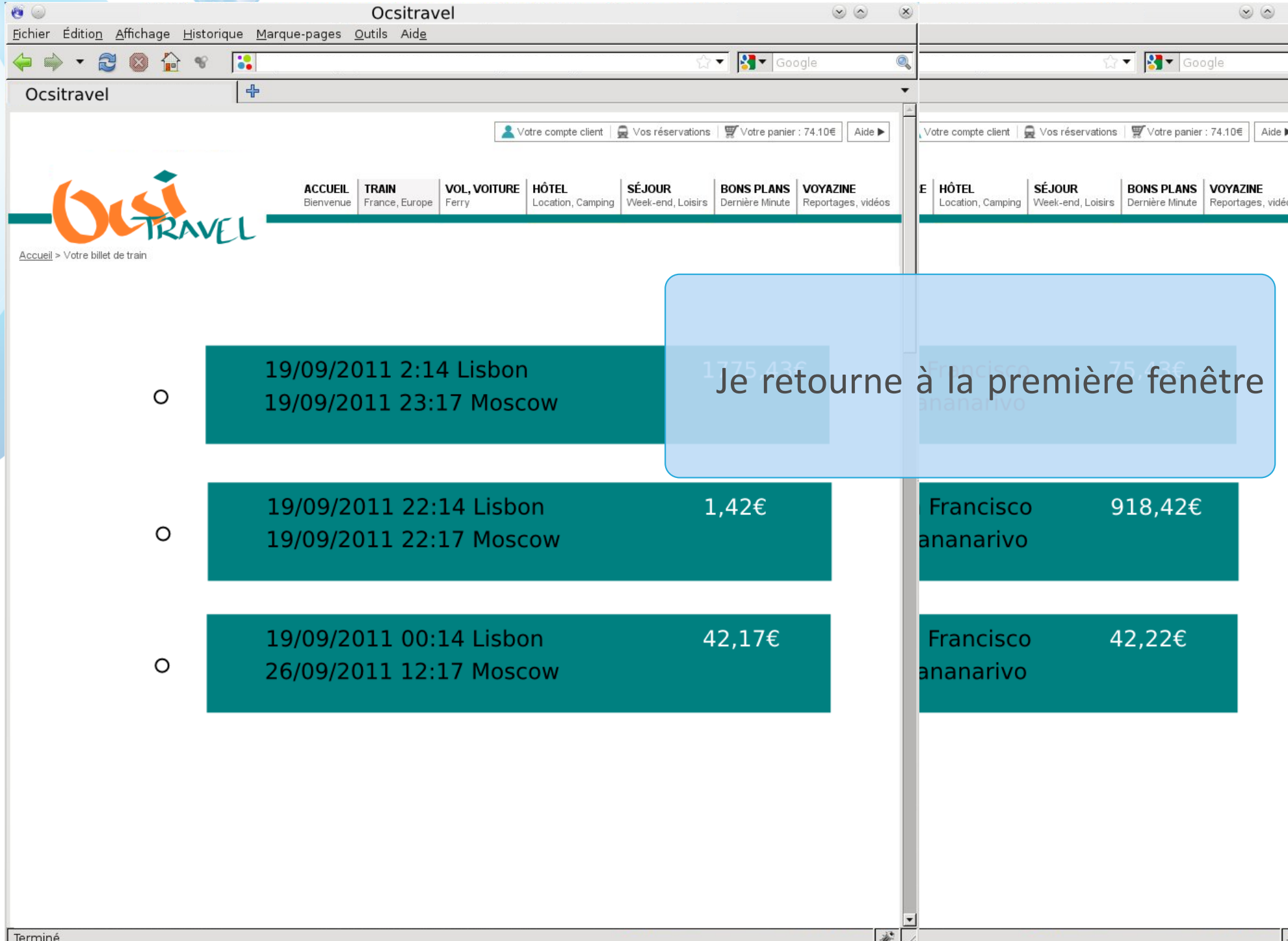
raction Web

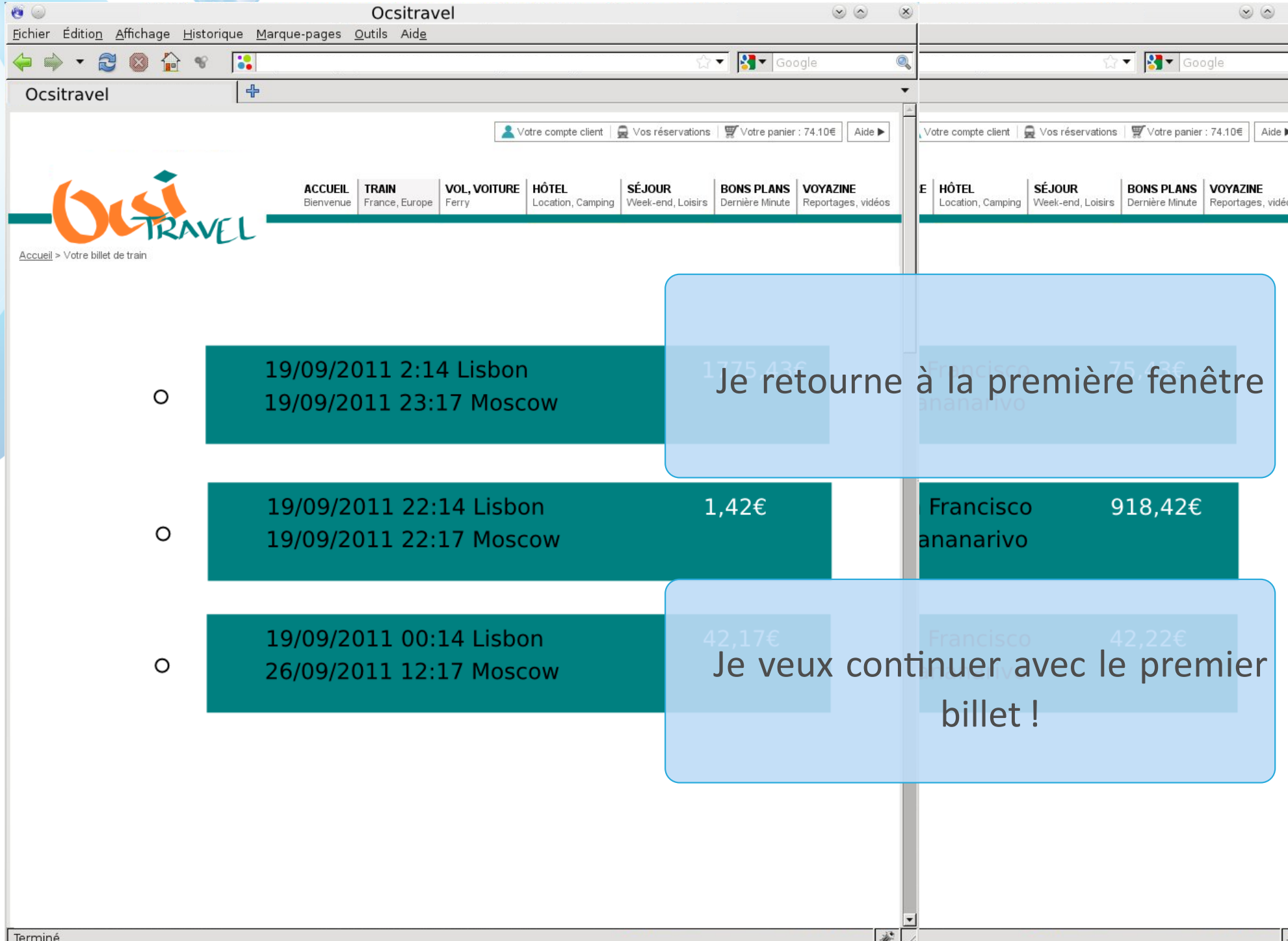


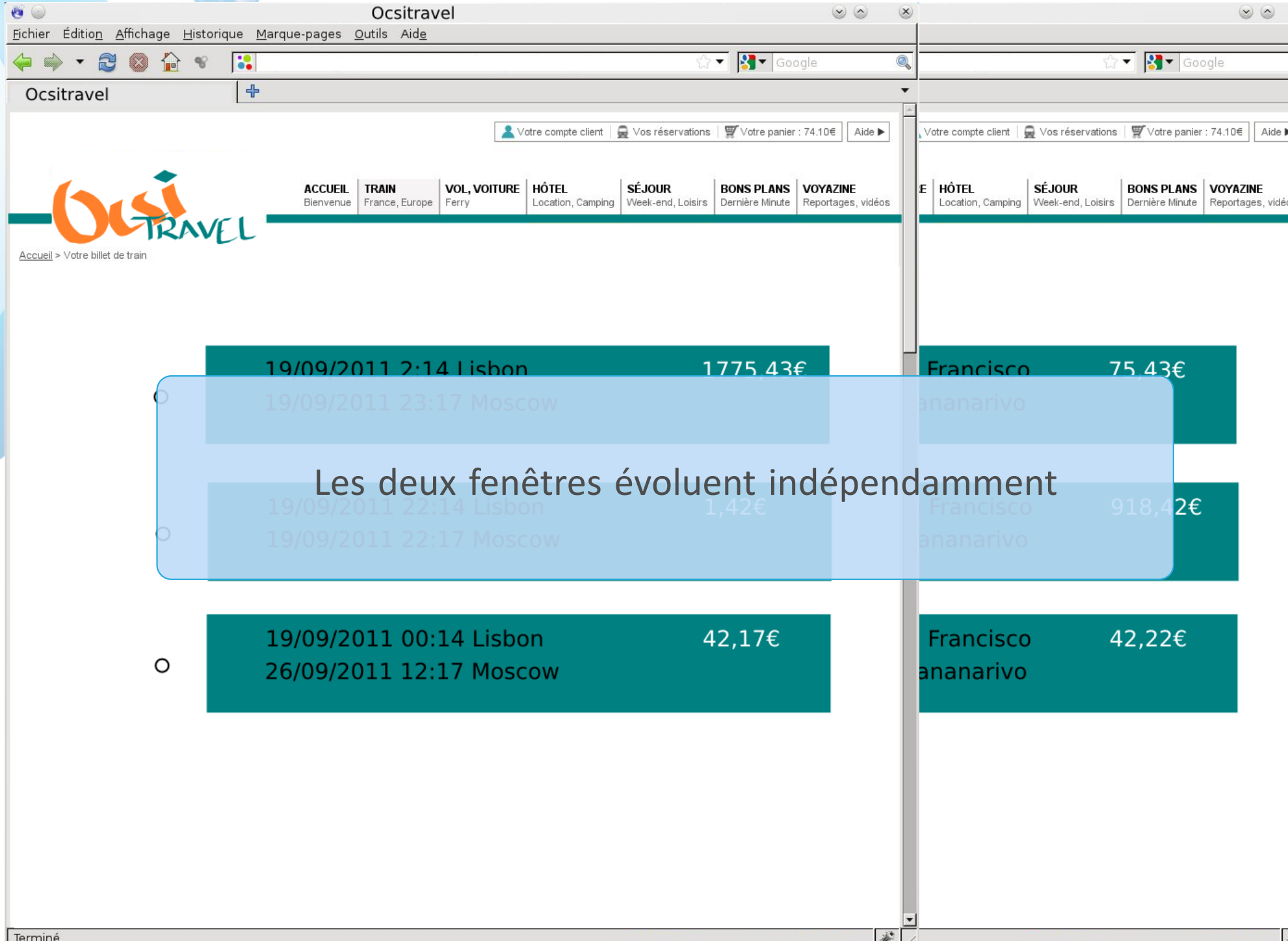
Le site propose plusieurs choix

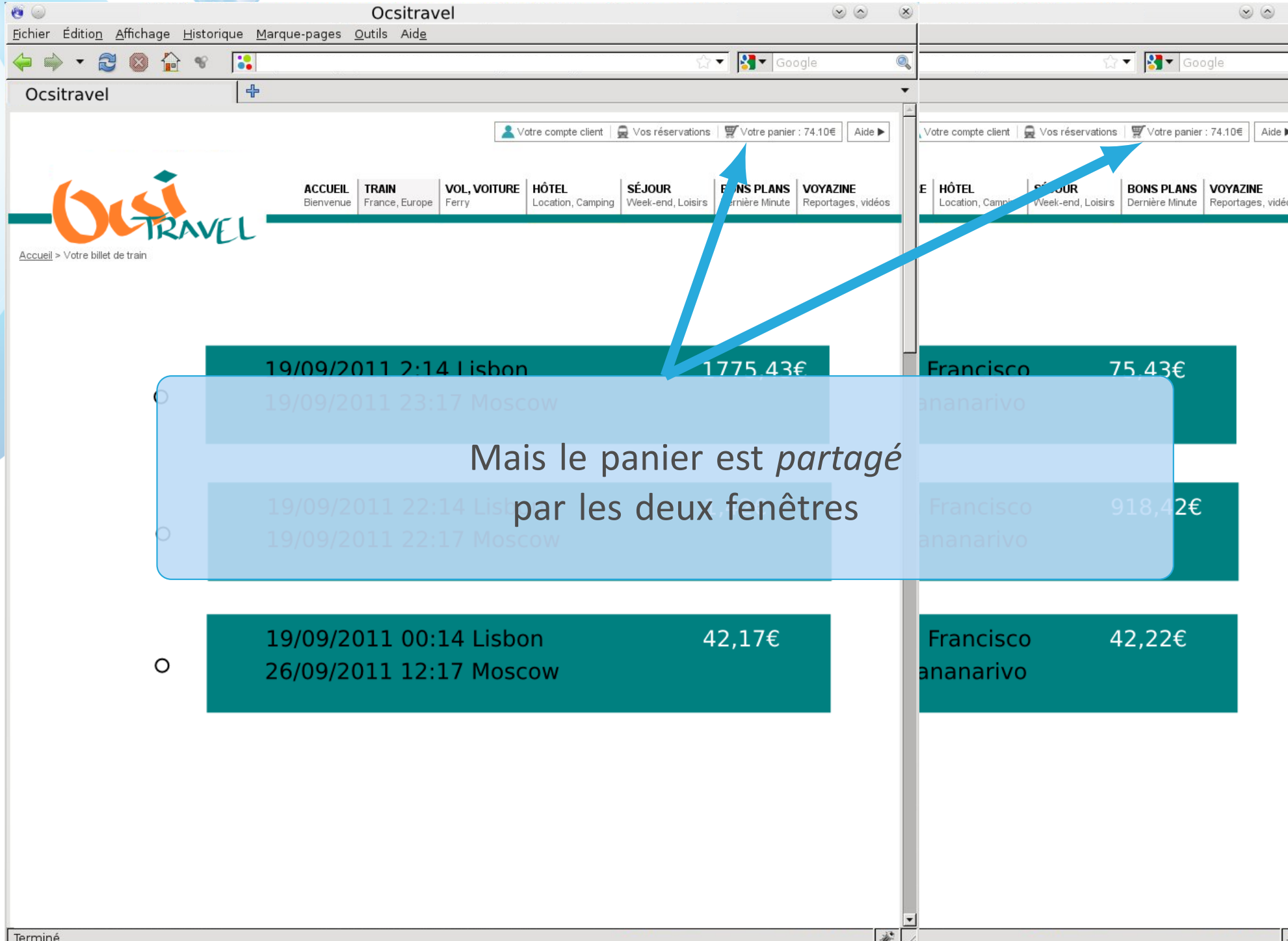

Ocsitravel
Fichier Édition Affichage Historique Marque-pages Outils Aide
Ocsitravel
Votre compte client
Vos réservations
Votre panier : 74.10€
Aide
TRAIN France, Europe
VOL, VOITURE Ferry
HÔTEL Location, Camping
SÉJOUR Week-end, Loisirs
BONS PLANS Dernière Minute
VOYAZINE Reportages, vidéo
Accueil > Votre billet de train
Dans une autre fenêtre,
Je veux réserver un billet
San Francisco -- Antananarivo
Voyage * Aller Simple Aller-Retour
Départ * San Francisco
Arrivée * Antananarivo
Aller le * 01/10/2010 à partir de 07h
Retour le à partir de 07h
Qui participe à ce voyage ?
Nombre de passagers 1
Code Offre Speciale
Voyageur 1
Passager 26-59 ans
Carte et Abonnement
Programme de fidélité
Comment souhaitez-vous voyager ?
Votre confort 1e classe 2e classe
Sélectionnez le pays de réception ou de retrait des billets : FRANCE
HORAIRES
RÉSERVEZ
TROUVEZ LES MEILLEURS PRIX
VERS 50 DESTINATIONS
publicité
Terminé


Ocsitravel
Fichier Édition Affichage Historique Marque-pages Outils Aide
Ocsitravel
Votre compte client
Vos réservations
Votre panier : 74.10€
Aide
ACCUEIL Bienvenue
TRAIN France, Europe
VOL, VOITURE Ferry
HÔTEL Location, Camping
SÉJOUR Week-end, Loisirs
BONS PLANS Dernière Minute
VOYAZINE Reportages, vidéo
Accueil > Votre billet de train
Encore une fois, plusieurs choix
75,43€
918,42€
42,22€
Terminé


Ocsitravel
Fichier Édition Affichage Historique Marque-pages Outils Aide
Ocsitravel
Votre compte client | Vos réservations | Votre panier : 74.10€ | Aide
ACCUEIL Bienvenue
TRAIN France, Europe
VOL, VOITURE Ferry
HÔTEL Location, Camping
SÉJOUR Week-end, Loisirs
BONS PLANS Dernière Minute
VOYAZINE Reportages, vidéos
Accueil > Votre billet de train
19/09/2011 2:14 Lisbon
19/09/2011 23:17 Moscow
Je retourne à la première fenêtre
19/09/2011 22:14 Lisbon 1,42€
19/09/2011 22:17 Moscow
Francisco 918,42€
ananarivo
19/09/2011 00:14 Lisbon 42,17€
26/09/2011 12:17 Moscow
Francisco 42,22€
ananarivo
Terminé


Ocsitravel
Fichier Édition Affichage Historique Marque-pages Outils Aide
Ocsitravel
Votre compte client
Vos réservations
Votre panier : 74.10€
Aide
ACCUEIL Bienvenue
TRAIN France, Europe
VOL, VOITURE Ferry
HÔTEL Location, Camping
SÉJOUR Week-end, Loisirs
BONS PLANS Dernière Minute
VOYAZINE Reportages, vidéos
Accueil > Votre billet de train
19/09/2011 2:14 Lisbon
19/09/2011 23:17 Moscow
19/09/2011 22:14 Lisbon
1,42€
19/09/2011 22:17 Moscow
Francisco
ananarivo
918,42€
19/09/2011 00:14 Lisbon
26/09/2011 12:17 Moscow
Je retourne à la première fenêtre
Je veux continuer avec le premier billet !
Terminé


Ocsitravel
Fichier Édition Affichage Historique Marque-pages Outils Aide
Ocsitravel
Votre compte client
Vos réservations
Votre panier : 74.10€
Aide
ACCUEIL Bienvenue
TRAIN France, Europe
VOL, VOITURE Ferry
HÔTEL Location, Camping
SÉJOUR Week-end, Loisirs
BONS PLANS Dernière Minute
VOYAZINE Reportages, vidéos
Accueil > Votre billet de train
19/09/2011 2:14 Lisbon
1775,43€
19/09/2011 23:17 Moscow
Francisco
75,43€
ananarivo
Les deux fenêtres évoluent indépendamment
19/09/2011 22:14 Lisbon
1,42€
19/09/2011 22:17 Moscow
Francisco
918,42€
ananarivo
19/09/2011 00:14 Lisbon
42,17€
26/09/2011 12:17 Moscow
Francisco
42,22€
ananarivo
Terminé


Ocsitravel
Fichier Édition Affichage Historique Marque-pages Outils Aide
Ocsitravel
Votre compte client
Vos réservations
Votre panier : 74.10€
Aide
ACCUEIL Bienvenue
TRAIN France, Europe
VOL, VOITURE Ferry
HÔTEL Location, Camping
SÉJOUR Week-end, Loisirs
BONS PLANS Dernière Minute
VOYAZINE Reportages, vidéos
Accueil > Votre billet de train
19/09/2011 2:14 Lisbon
1775,43€
19/09/2011 23:17 Moscow
Francisco
75,43€
ananarivo
19/09/2011 22:14 Lisbon
19/09/2011 22:17 Moscow
Francisco
918,42€
ananarivo
19/09/2011 00:14 Lisbon
42,17€
26/09/2011 12:17 Moscow
Francisco
42,22€
ananarivo
Mais le panier est *partagé*
par les deux fenêtres
Terminé

Solutions

Panier :

Sessions

⟶ « État » côté serveur pour un navigateur

⟶ Gestion automatiques des cookies de session

⟶ Possibilité de créer des services *de session* !

Solutions

Panier :

Sessions

⟶ « État » côté serveur pour un navigateur

⟶ Gestion automatiques des cookies de session

⟶ Possibilité de créer des services *de session* !

Pages dépendant d'interactions précédentes :

Solutions

Panier :

Sessions

⟶ « État » côté serveur pour un navigateur

⟶ Gestion automatiques des cookies de session

⟶ Possibilité de créer des services *de session* !

Pages dépendant d'interactions précédentes :

Création dynamique de services

⟶ L'histoire de l'interaction est gardée automatiquement en mémoire (côté serveur)

⟶ Vous pouvez retourner dans le passé (bouton ``back'')
ou changer de fenêtre de navigateur

Sessions revisitées

Données de session sauvegardées dans des *références* avec une **portée (scope)**.

Portée :

- Site
- Session de navigateur (cookie)

Sessions revisitées

État côté serveur mis en œuvre grâce à des *références* avec une **portée (scope)**.

Portée :

- Site
- Session de navigateur (cookie)
- **Processus client (onglet)**

Exemple : Si vous avez plusieurs instances d'un jeu dans plusieurs onglets, le score est une référence de portée « onglet ».

Sessions revisitées

État côté serveur mis en œuvre grâce à des *références* avec une **portée (scope)**.

Portée :

- Site
- **Groupes de sessions (utilisateur)**
- Session de navigateur (cookie)
- Processus client (onglet)

Exemple : partager le panier entre plusieurs terminaux !

Sessions revisitées

État côté serveur mis en œuvre grâce à des *références* avec une **portée (scope)**.

Portée :

- Site
- Groupes de sessions (utilisateur)
- Session de navigateur (cookie)
- Processus client (onglet)
- **Requête**

Garder de l'information pendant la génération d'une page.

Sessions revisitées

État côté serveur mis en œuvre grâce à des *références* avec une **portée (scope)**.

Portée :

- Site
- Groupes de sessions (utilisateur)
- Session de navigateur (cookie)
- Processus client (onglet)
- Requête

Les *services* ont aussi une portée.

Web 1.0 + Web 2.0 = ?

Web 1.0 + Web 2.0 = ?

Avec Ocsigen, le programme côté client ne s'arrête pas quand vous cliquez sur un lien !

Web 1.0 + Web 2.0 = ?

Avec Ocsigen, le programme côté client ne s'arrête pas quand vous cliquez sur un lien !

⟶ Les applications Ocsigen client-server sont 100% compatibles avec l'interaction Web traditionnelle (signets, bouton back) !

⟶ Vous pouvez garder un état côté client.

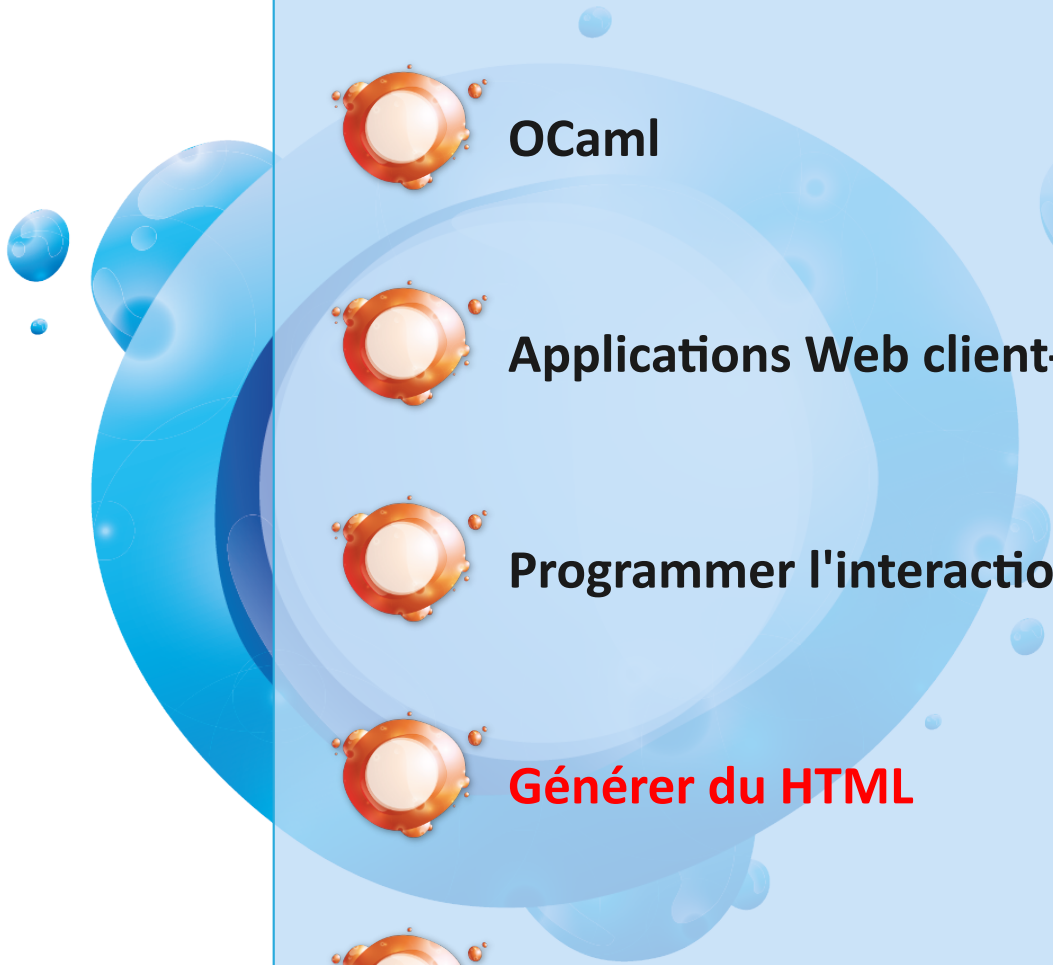
⟶ Certaines portions de la page peuvent rester après avoirchangé de page.

⟶ La musique ou les vidéos ne s'arrêtent pas.

Exemple : Site de streaming vidéo

(continuer sa navigation pour choisir d'autres albums sans arrêter la musique)

ocsigen
fresh air in web programming



- OCaml
- Applications Web client-serveur
- Programmer l'interaction Web traditionelle
- **Générer du HTML**
- Conclusion

Génération de HTML

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        (html
           (head
              (title (pcdata "Graffiti"))
              [link ~rel:[ `Stylesheet ] ~href:(uri_of_string"./css/style.css") ();
               script ~a:[a_src (uri_of_string "./graffiti_oclosure.js")] (pcdata "");])
           (body [h1 [pcdata "Graffiti"]])))
```

ocsigen
fresh air in web programming

Génération de HTML

```
(html
  (head
    (title (pcdata "Graffiti"))
    [link ~rel :[ `Stylesheet ] ~href :(uri_of_str
    script ~a :[a_src (uri_of_string "./graffiti
  (body [h1 [pcdata "Graffiti"]])))
```

Générer du HTML

La validité du HTML est vérifiée à la compilation !

Produire du HTML valide

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</html>
```

Produire du HTML valide

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</html>
```

Produire du HTML valide

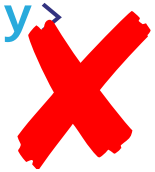
```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</html>
```

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</hmtl>
```

Produire du HTML valide

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</html>
```

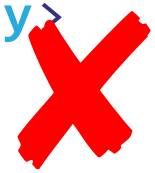
```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</hmtl>
```



Produire du HTML valide

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</html>
```

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</hmtl>
```

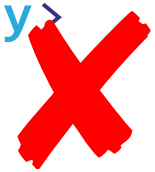


```
<html>
  <head><title>Hello</title></head>
  <body><title>Hello</title></body>
</html>
```

Produire du HTML valide

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</html>
```

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</hmtl>
```

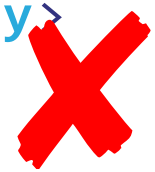


```
<html>
  <head><title>Hello</title></head>
  <body><title>Hello</title></body>
</html>
```

Produire du HTML valide

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</html>
```

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</hmtl>
```



```
<html>
  <head><title>Hello</title></head>
  <body><title>Hello</title></body>
</html>
```

→ rejetés *à la compilation !*

Produire du HTML valide

Les programmes pouvant générer des pages invalides sont rejetés par le compilat

$f : () \rightarrow$ block list

<body> $f()$ </body>

Produire du HTML valide

Les programmes pouvant générer des pages invalides sont rejetés par le compilat

$f : () \rightarrow$ block list

<body> $f()$ </body>

Produire du HTML valide

Les programmes pouvant générer des pages invalides sont rejetés par le compilat

$f : () \rightarrow$ block list

<body> $f()$ </body>

 $f()$

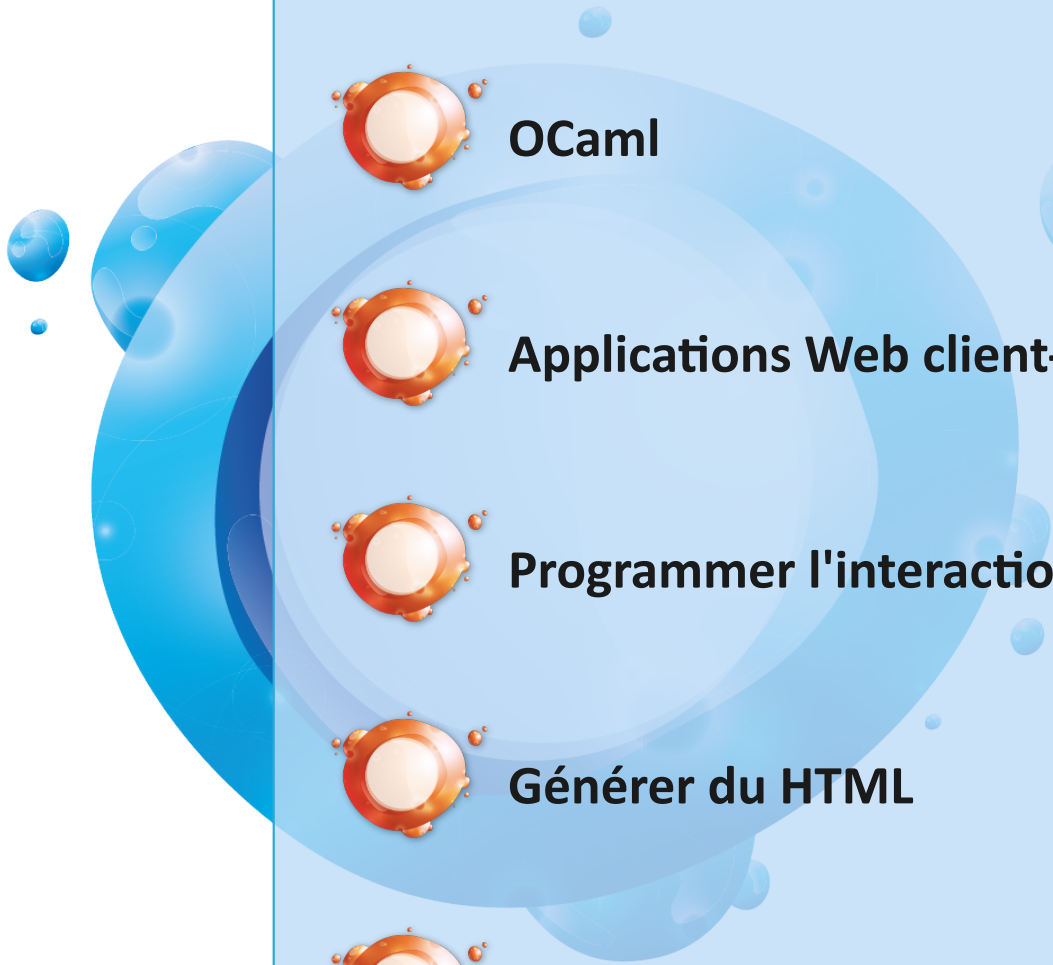
Produire du HTML valide

Les programmes pouvant générer des pages invalides sont rejetés par le compilat

$f : () \rightarrow$ block list

<body> $f()$ </body>

 $f()$



- OCaml
- Applications Web client-serveur
- Programmer l'interaction Web traditionelle
- Générer du HTML
- Conclusion

Beaucoup de fonctionnalités uniques

Beaucoup de fonctionnalités uniques

Identification de services sophistiquée

Beaucoup de fonctionnalités uniques

Identification de services sophistiquée

Typages des liens, formulaires, paramètres

Beaucoup de fonctionnalités uniques

Identification de services sophistiquée

Typage du HTML

Typages des liens, formulaires, paramètres

Beaucoup de fonctionnalités uniques

Services dynamiques

Identification de services sophistiquée

Typage du HTML

Typages des liens, formulaires, paramètres

Beaucoup de fonctionnalités uniques

Services dynamiques

Identification de services sophistiquée

Typage du HTML

Services de session

Typages des liens, formulaires, paramètres

ocsigen
fresh air in web programming

Beaucoup de fonctionnalités uniques

Services dynamiques

Identification de services sophistiquée

Typage du HTML

Services de session

Portée

Typages des liens, formulaires, paramètres

ocsigen
fresh air in web programming

Beaucoup de fonctionnalités uniques

Services dynamiques

Identification de services sophistiquée

Typage du HTML

Programmation client-server unifiée

Services de session

Portée

Typages des liens, formulaires, paramètres

ocsigen
fresh air in web programming

Beaucoup de fonctionnalités uniques

Services dynamiques

Persistence du programme client

Identification de services sophistiquée

Typage du HTML

Programmation client-server unifiée

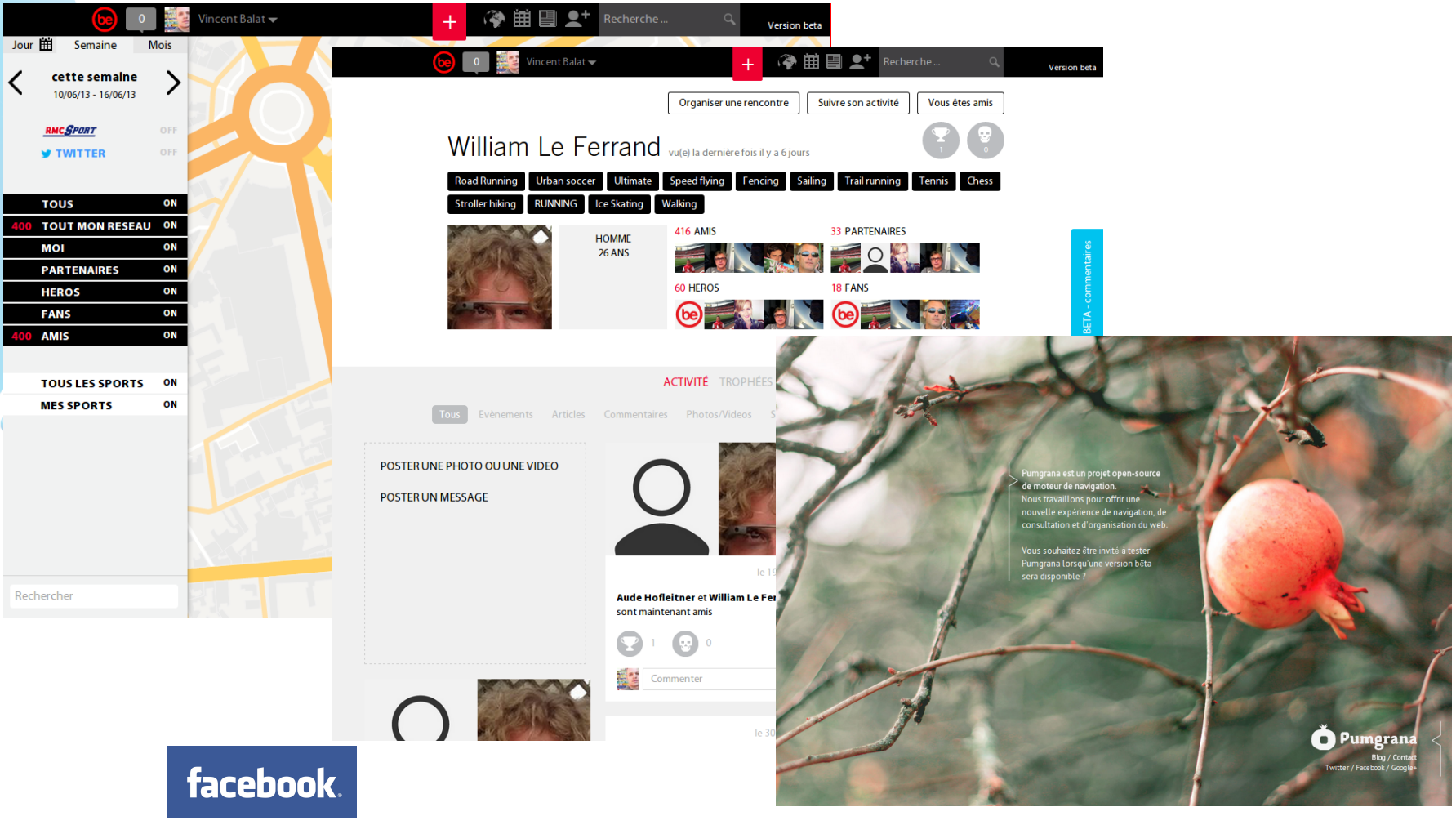
Services de session

Portée

Typages des liens, formulaires, paramètres

ocsigen
fresh air in web programming

Users



Some companies and open source projects :

BeSport, NYU CGSB Genomics Core, Pumgrana, Facebook, Life.tl, Ashima Arts, Metaweb/Freebase, Hypios, Ocamlcore, Ocamlpro, Baoug, Nleyten...

ocsigen
fresh air in web programming

Quelques-uns de nos projets

ocsigen eliom

More information

Write client/server Web applications in very few lines of OCaml code!

ocsigen js_of_ocaml

More information

An OCaml to Javascript compiler.

ocsigen lwt

More information

A cooperative threading library for OCaml.

ocsigen server

More information

A full-featured and extensible Web server.

> And many other projects ...

ocsigen
fresh air in web programming

Le projet

ocsigen.org

Logiciel libre --- Version 4.0 ce mois-ci !

Auteurs et contributeurs :

Vincent Balat, Jérôme Vouillon, Pierre Chambart, Grégoire Henry, Benedikt Becker, Raphaël Proust, Benjamin Canou, Boris Yakobowski, Jérémie Dimino

Charly Chevalier, Gabriel Radanne, Jacques-Pascal Deplaix, Stéphane Glondu, Gabriel Kerneis, Arnaud Parant, Christophe Lecointe, Denis Berthod, Gabriel Cardoso, Piero Furiesi, Jaap Boender, Thorsten Ohl, Gabriel Scherer, Séverine Maingaud, Simon Castellan, Jean-Henri Granarolo, Archibald Pontier, Nataliya Guts, Cécile Herbelin, Charles Oran, Jérôme Velleine, Pierre Clairambault …

ocsigen
fresh air in web programming