Business Intelligence et KPI

3. Développer des Tableaux de Bord avec R Shiny

Salim Lardjane
Université de Bretagne Sud

A. HTML et Shiny

HTML et Shiny

- Shiny est un cadre de développement Web basé sur R permettant de créer des applications Web interactives.

- Shiny peut être utilisé directement comme une application Web ou pour construire des tableaux de bord à l'aide de R Markdown.

- Il permet de lire, explorer et analyser les données à l'aide d'un navigateur Web.

- Les applications Shiny peuvent être étendues à l'aide de CSS, JavaScript et à l'aide un package dédié d'éléments graphiques (widgets) HTML.

HTML et Shiny

- Shiny est basé sur HTML mais permet de programmer une interface complète en R et Shiny sans se préoccuper du HTML sous-jacent.

- On peut mettre à jour l'interface en utilisant des fonctions natives Shiny ou via du code HTML en utilisant la fonction tags.

- Shiny permet également de programmer une interface entière uniquement en HTML.

- Pour travailler avec Shiny, il est préférable d'avoir quelques notions de HTML et CSS. Nous les introduirons au fur et à mesure dans ce cours.

HTML et Shiny

- HTML (HyperText Markup Language) est un langage de programmation basé sur des balises.
- Il est utilisé pour générer des pages Web statiques.
- Les principales balises HTML sont les suivantes :

HTML et Shiny

- p : utilisée pour créer un paragraphe.

- h1-h6 : Styles de titres utilisés pour créer des titres et des sous-titres ; h1 est le plus grand visuellement et h6 le plus petit.

- a : utilisée pour créer des liens. Elle est associée à href qui correspond à l'URL de la page Web visée.

- br : utilisée pour générer un saut à la ligne.

- div : utilisée pour définir une section ayant un style particulier.

HTML et Shiny

- span : utilisée pour définir un style correspondant à une chaîne de caractères.

- pre : utilisée pour formater des sections de code ou de commandes sous la forme de blocs de texte.

- code : utilisée pour formater des sections de code.

- img : utilisée pour afficher une image.

- strong : sert à mettre le texte en gras.

- em : utilisée pour mettre le texte en italique.

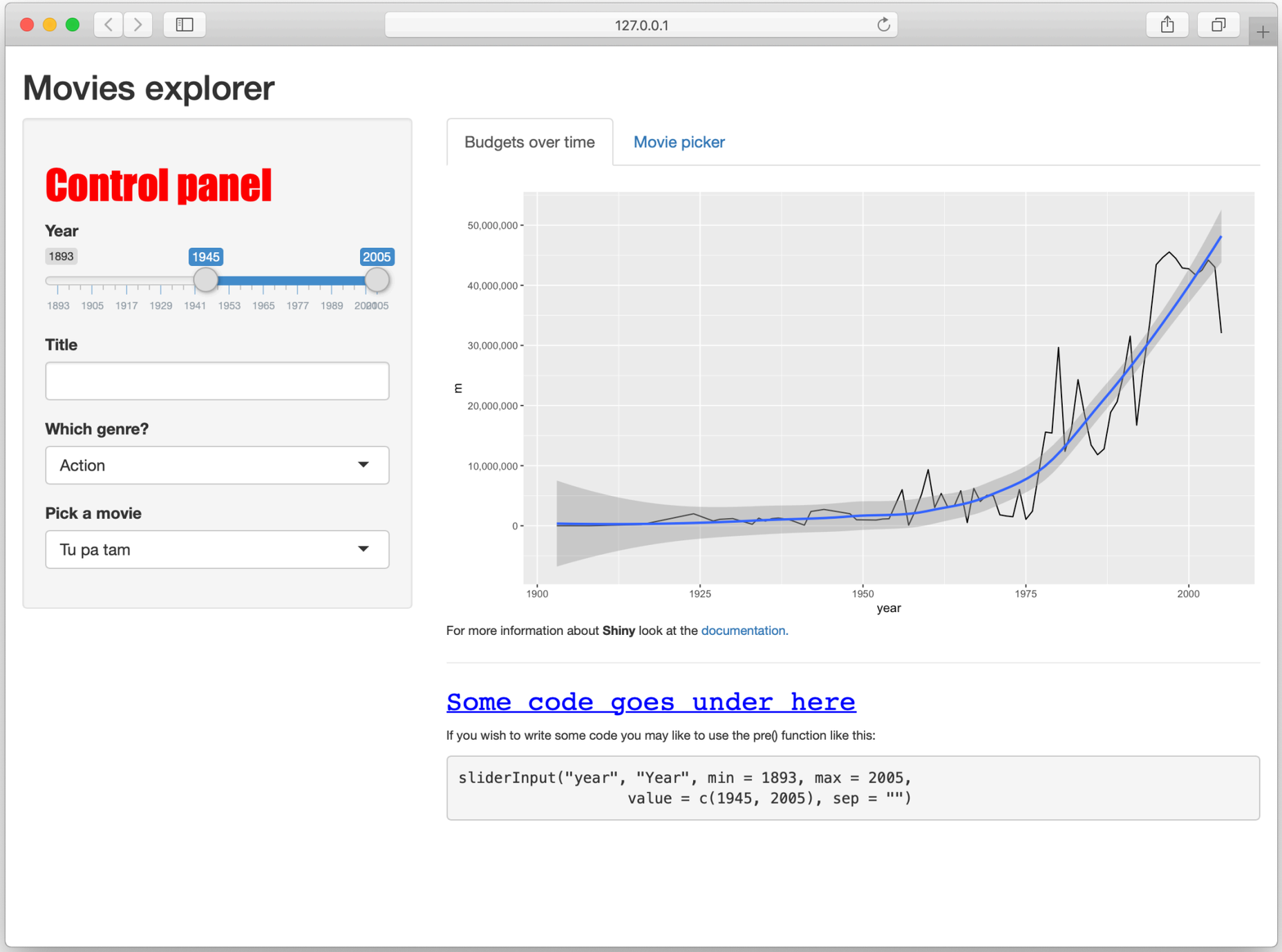
- hr : utilisée pour insérer une ligne horizontale.

HTML et Shiny

- Voici un exemple d’application Web générée avec Shiny.

- On donne à la suite le code Shiny/HTML correspondant au cadre bas droit de l’interface.

HTML et Shiny



HTML et Shiny

```
mainPanel(
    tabsetPanel(
      tabPanel("Budgets over time", plotOutput("budgetYear"),
               p("For more information about ", strong("Shiny"), " look at the ",
                 a(href = "http://shiny.rstudio.com/articles/",
                   "documentation.")),
               hr(),
               h3("Some code goes under here"),
               p("If you wish to write some code you may like to use the pre()
                 function like this:",
                 pre('sliderInput("year", "Year", min = 1893, max = 2005,
                 value = c(1945, 2005), sep = "")'))),
      tabPanel("Movie picker", tableOutput("moviePicker"))
)
```

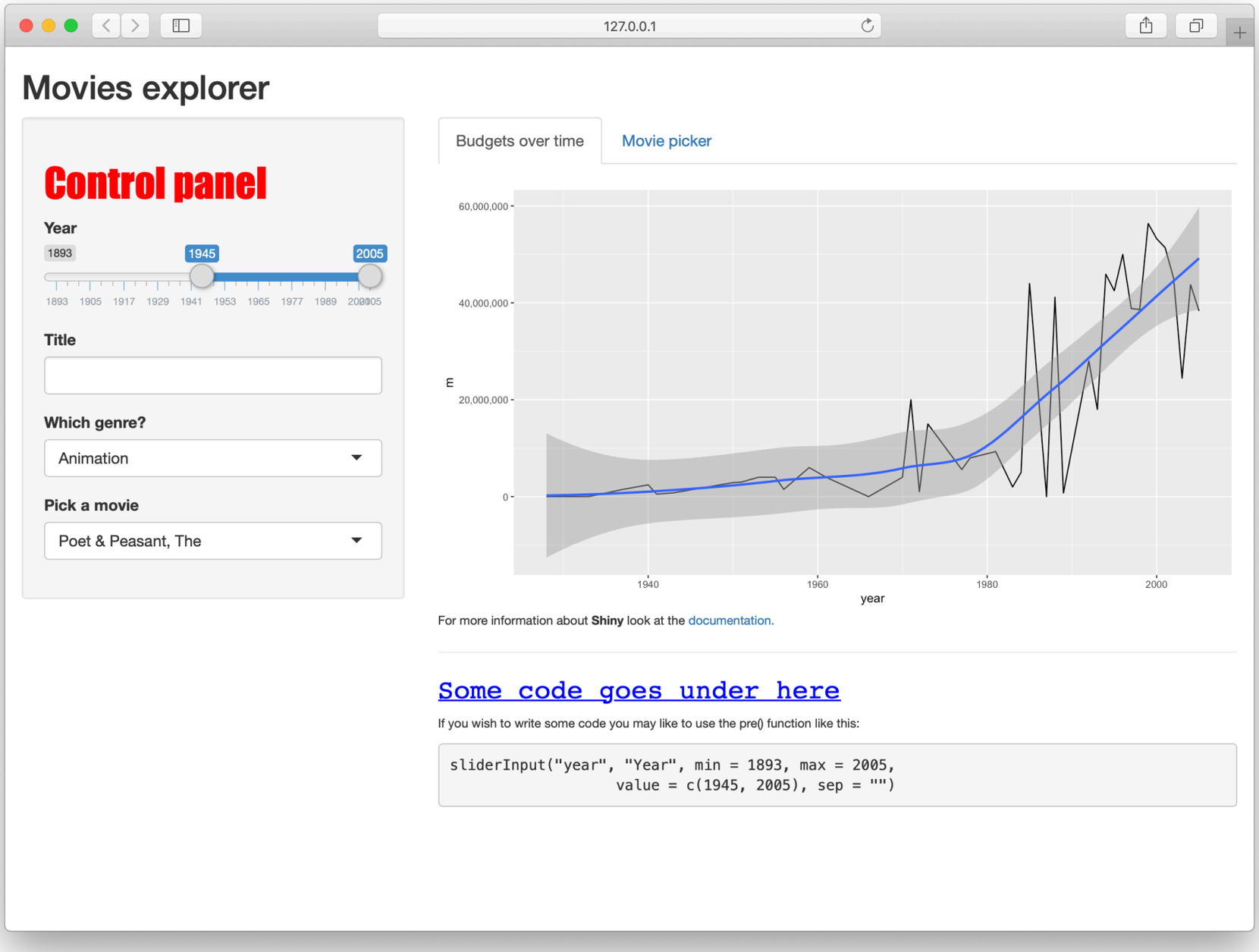
Créer une IU avec Shiny/HTML

- On a vu quelques unes des balises HTML dans ce qui précède.

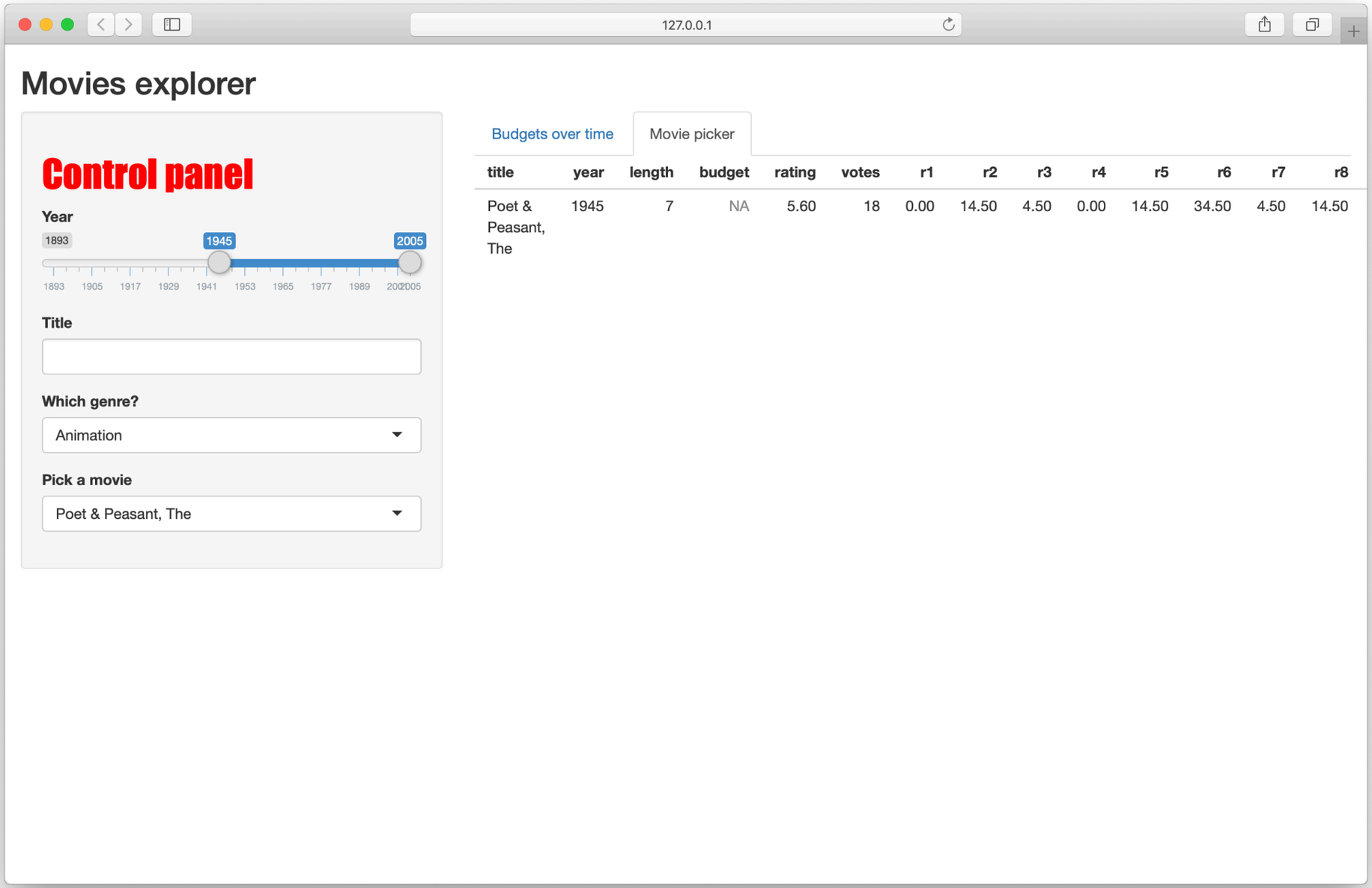
- Nous allons à présent voir comment créer une application Web avec Shiny/HTML.

- Reprenons l’exemple précédent.

Créer une IU avec Shiny/HTML



Créer une IU avec Shiny/HTML



Créer une IU avec Shiny/HTML

- On constate sur les fenêtres précédentes qu'il y a deux menus déroulants : un pour sélectionner le titre du film et l'autre pour sélectionner son genre.

- Il y a également un champ texte pour donner un titre au graphique.

- On va voir dans la suite comment créer les pages précédentes avec du code HTML.

- On verra comment rajouter la règle glissante avec Shiny car c'est beaucoup plus simple qu'en HTML.

Créer une IU avec Shiny/HTML

- On commence par inclure trois liens dans la section head.
- Ils sont requis pour que les pages s’affichent correctement. On peut également inclure des liens additionnels JavaScript ou CSS si on veut les utiliser dans l’application :

Créer une IU avec Shiny/HTML

```
<html>

<head>

  <title>HTML movies application</title>

  <script src="shared/jquery.js" type="text/javascript"></script>

  <script src="shared/shiny.js" type="text/javascript"></script>

  <link rel="stylesheet" type="text/css" href="shared/shiny.css"/>

  <link href="shared/bootstrap/css/bootstrap.min.css" rel="stylesheet">

</head>
```

Créer une IU avec Shiny/HTML

- On inclus le CSS Bootstrap qui permet aux colonnes et lignes de la class div d'être affichées correctement.

- On introduira davantage de CSS dans la suite.

- Une fois les liens rajoutés, on passe au corps du HTML :

Créer une IU avec Shiny/HTML

```
<h1>Minimal HTML UI</h1>


  <div class="container-fluid">

    <div class="row">


<div class="col-sm-4">

  <h3>Control panel</h3>


  <div id="listMovies" class="shiny-html-output"></div>


  Title:<input type="text" name="title"><br>
```

Créer une IU avec Shiny/HTML

```
<select name = "genre" class = "form-control">

        <option value="Action">Action</option>

        <option value="Animation">Animation</option>

        <option value="Comedy">Comedy</option>

        <option value="Drama">Drama</option>

        <option value="Documentary">Documentary</option>

        <option value="Romance">Romance</option>

        <option value="Short">Short</option>

      </select>


</div>
```

Créer une IU avec Shiny/HTML

- Tout d'abord, on mentionne un titre pour la page dans la balise h1.

- Ensuite, on crée notre premier champ de saisie. C'est un champ de type inhabituel car il est affiché de façon dynamique. On lui affecte la classe shiny-html-output qui permet de l'afficher dynamiquement.

- Cela signifie que la saisie est transmise au serveur qui la renvoie pour affichage à l'IU, ce qui permet à l'élément affiché de changer en fonction de la saisie de l'utilisateur.

Créer une IU avec Shiny/HTML

- Ensuite, on crée le champ de saisie du titre de film, auquel on donne le nom « title ».

- A la suite, on crée le menu déroulant « genre ».

- Vient ensuite le code qui permettra l'affichage du graphique :

Créer une IU avec Shiny/HTML

```
<div class="col-sm-8">
  <div id="budgetYear" class="shiny-plot-output"
  style="width: 100%; height: 400px"></div>
```

Créer une IU avec Shiny/HTML

- On assigne l'identifiant « budetYear » à la balise div et on lui affecte la classe shiny-plot-output qui permettra d'afficher le graphique dynamiquement. On spécifie également la largeur et la hauteur du graphique.

- Les lignes suivantes fournissent des exemples de liens, formatage de texte et de blocs de code avec les balises introduites précédemment :

Créer une IU avec Shiny/HTML

```
<p>For more information about <strong>Shiny</strong> look at the

    <a href="http://shiny.rstudio.com/articles/">documentation.</a></p>

    <hr>

    <p>If you wish to write some code you may like to use

    the pre() function like this:</p>

      <pre>sliderInput("year", "Year", min = 1893, max = 2005,

          value = c(1945, 2005), sep = "")</pre>


      <div id = "moviePicker" class = "shiny-html-output"></div>

    </div>

  </div>

 </div>

</body>

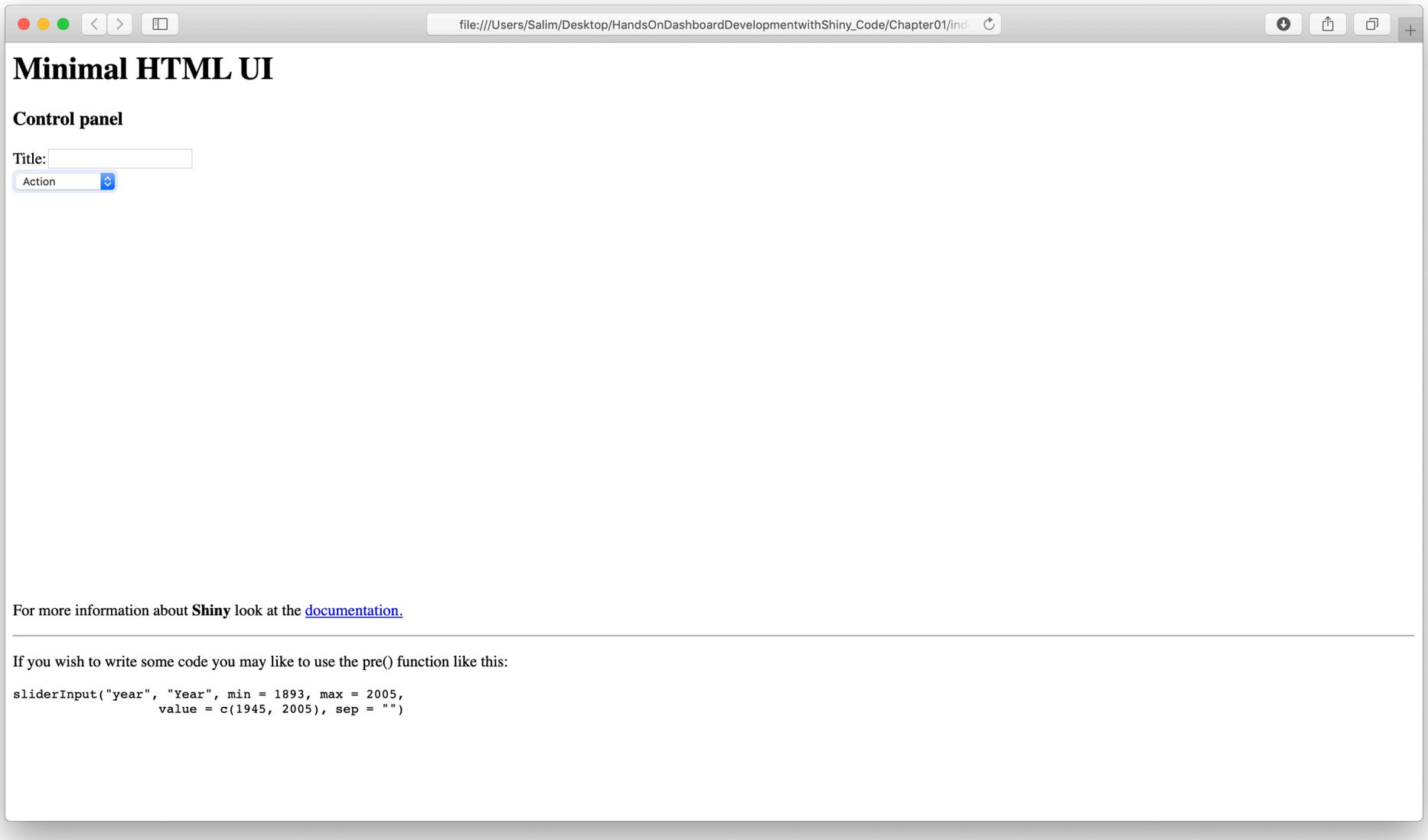
</html>
```

Créer une IU avec Shiny/HTML

- La dernière balise div permettra d'obtenir l'affichage dynamique d'une table.

- Au stade actuel, l'exécution du code précédent (fichier index.html sur le forum) par un navigateur donne le résultat suivant :

Créer une IU avec Shiny/HTML



La fonction tags( )

- Shiny fournit des fonctions permettant d'utiliser certaines balises HTML de base pour créer une application.

- Pour les balises non supportées par cette fonction, on peut utiliser la fonction tags.

- Pour utiliser cette fonction, il suffit d'utiliser la syntaxe tags$name, où name est la balise requise.

La fonction tags( )

- Exemple :

tags$a(href = ‘’www.example.com'', ‘’This is an example”).

- Résultat :

This is an exemple .

- Les arguments non nommés sont utilisés comme corps de la balise.

La fonction tags( )

- On peut utiliser des fonctions tags emboîtées si nécessaire.

- Exemple :

La fonction tags( )

```
tags$head(

    tags$title(« HTML movies applications »),

    tags$script(src =  « shared/jquery.js », type = « text/javascript »),

    tags$script(src = « shared/shiny.js », type = « text/javascript »),

    tags$link(rel = « stylesheeet » , type = « text/css », href = « shared/shiny.css »),

)
```

La fonction tags( )

- On a utilisé tags$head et d’autres tags séparés par des virgules.

- En sortie, on obtient le code HTML suivant :

La fonction tags( )

```
<head>
  <title>HTML movies application</title>
  <script src="shared/jquery.js" type="text/javascript"></script>
  <script src="shared/shiny.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="shared/shiny.css"/>
</head>
```

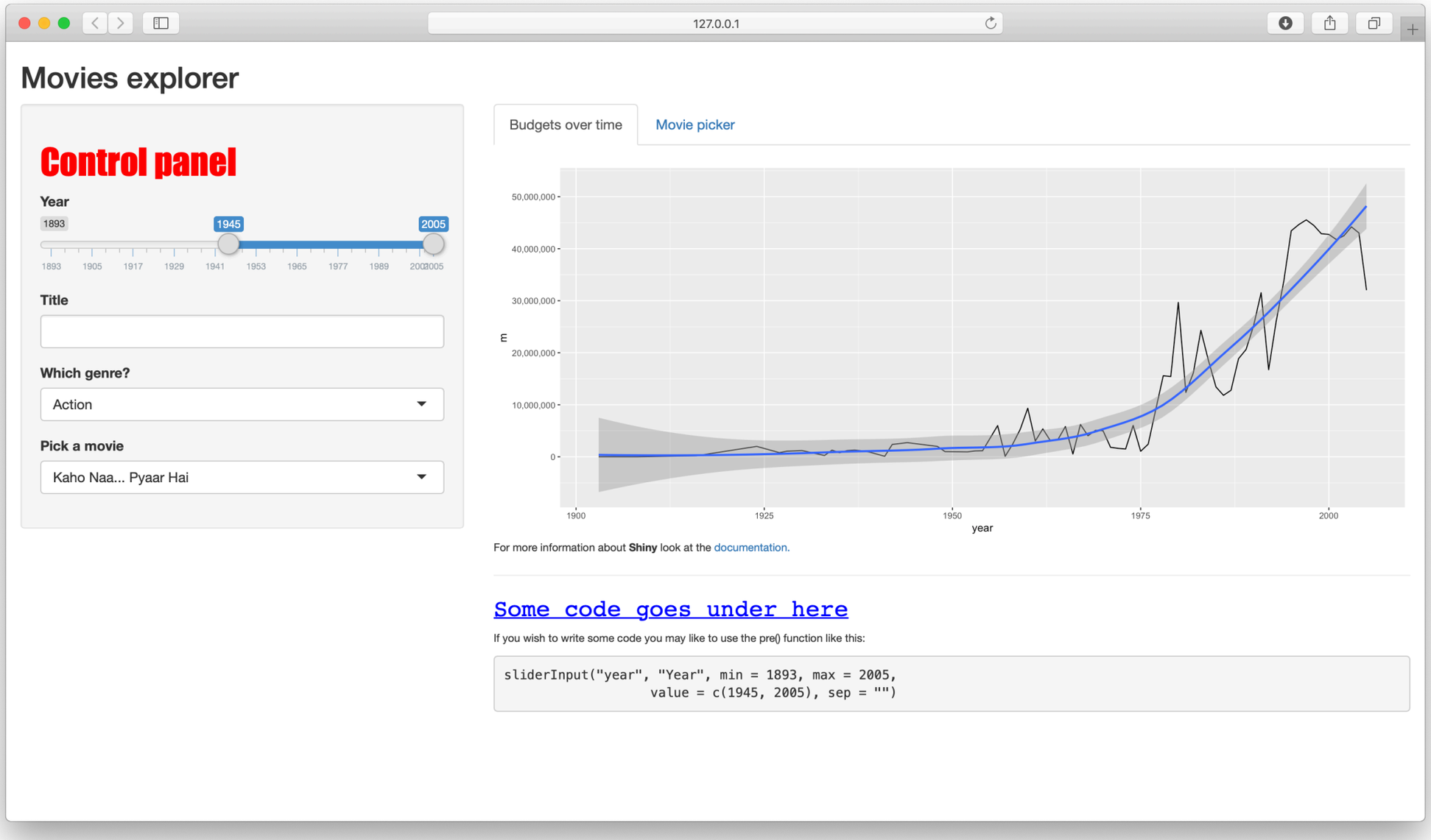
CSS

- On a vu à présent les bases de HTML.

- On va voir maintenant comment modifier les styles d’affichage de l’application en utilisant CSS et une feuille de style.

- Voici la sortie finale que nous obtiendrons :

CSS


127.0.0.1
Movies explorer
Control panel
Year
1893
1945
2005
1893 1905 1917 1929 1941 1953 1965 1977 1989 2001 2005
Title
Which genre?
Action
Pick a movie
Kaho Naa... Pyaar Hai
Budgets over time
Movie picker
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
0
1900
1925
1950
1975
2000
year
For more information about Shiny look at the documentation.
Some code goes under here
If you wish to write some code you may like to use the pre() function like this:
sliderInput("year", "Year", min = 1893, max = 2005,
value = c(1945, 2005), sep = "")

CSS

- La meilleure façon d’inclure CSS est d’insérer un style dans l’élément qu’on veut modifier.

- Par exemple, pour le tag h1 affichant « Control panel », on procède de la façon suivante :

h1(« Control panel », style = « color:red ; font-family:Impact; Charcoal, sans-serif; »)

CSS

- Lorsqu'on utilise CSS sous Shiny, les règles de bases des CSS sont appliquées.

- Par exemple, on utilise un point-virgule pour séparer les éléments.

- L'autre façon d'utiliser les CSS est de les inclure dans la section head à l'aide de la commande tags :

CSS

```
tags$head(

tags$style(HTML(« h3 {
color: blue; font-family: courier; text-decoration:
underline;} »

))

)
```

CSS

- Dans ce cas, toutes les spécifications CSS sont incluses dans les tags head et style.

- La commande HTML est utilisée pour spécifier que les instructions sont formatées en HTML et ne doivent pas être traduites.

CSS

- La dernière façon d’utiliser CSS sous Shiny est de prévoir une feuille de style séparée.

- A cet effet, on utilise la commande include :

includeCSS(« styles.css »)

CSS

- On peut utiliser cette command n’importe où dans le programme, mais il est conseillé de la positionner en tête de la définition de l’IU.

- Le fichier style.css soit se trouver **dans le même répertoire** que le fichier ui.r et pas dans le répertoire www.

CSS

- Ecrire du CSS et assez simple, mais la méthode utilisée dépend de la quantité de spécifications à fournir.

- Pour éviter de répéter de grandes portions de code, il est conseillé d’utiliser une feuille de style séparée.

Rapports dynamiques

- On va a présent voir comment utiliser la fonction Shiny downloadhandler pour lire et afficher des rapports.

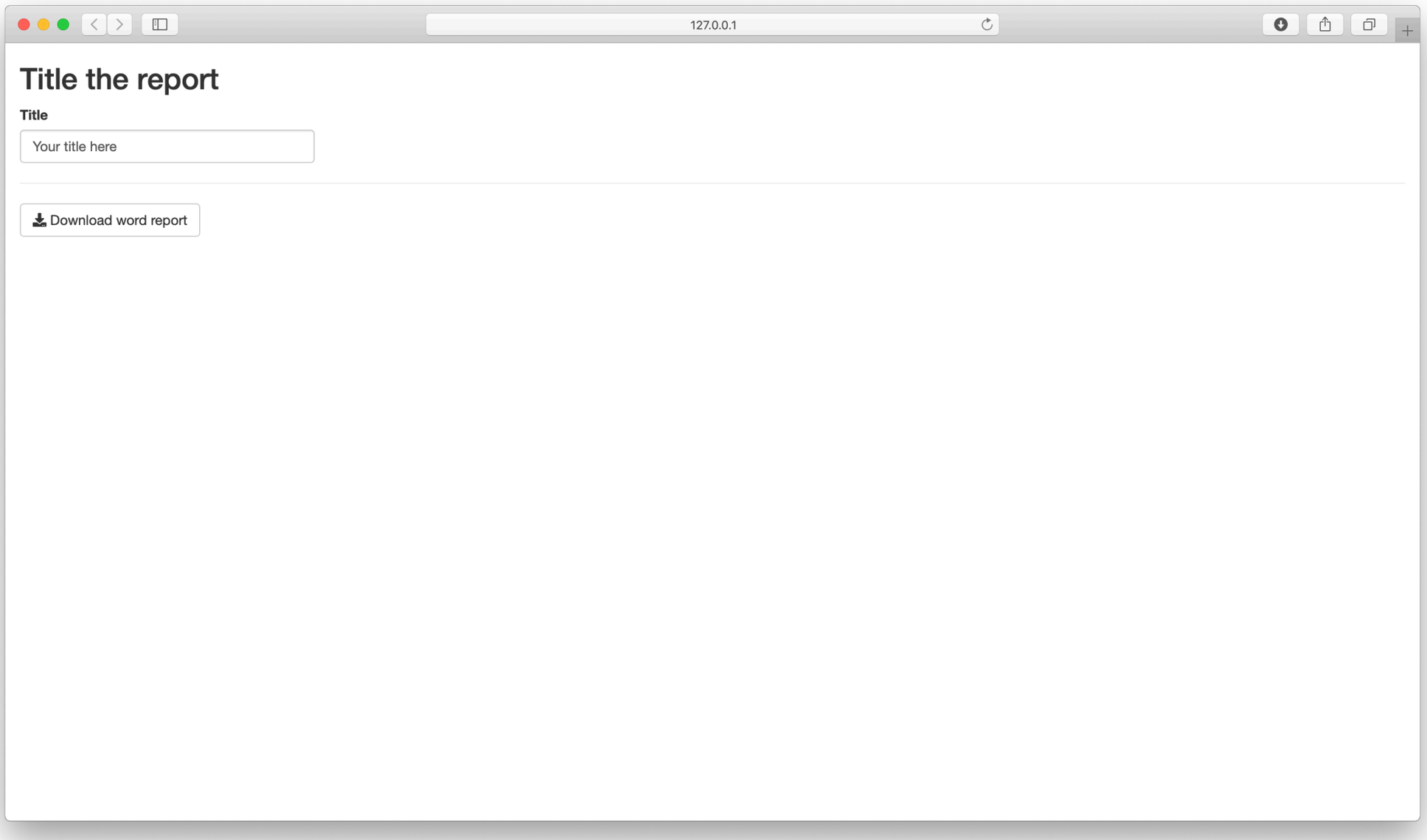
- On va également apprendre à écrire un document R Markdown et à le rendre dynamique.

Rapports dynamiques

- Commençons par essayer de comprendre le fonctionnement de l’application qu’on va créer.

- Voici la page principale de l’application, qui propose un champ texte pour modifier le titre du rapport et un bouton pour le télécharger et le sauvegarder au format Word.

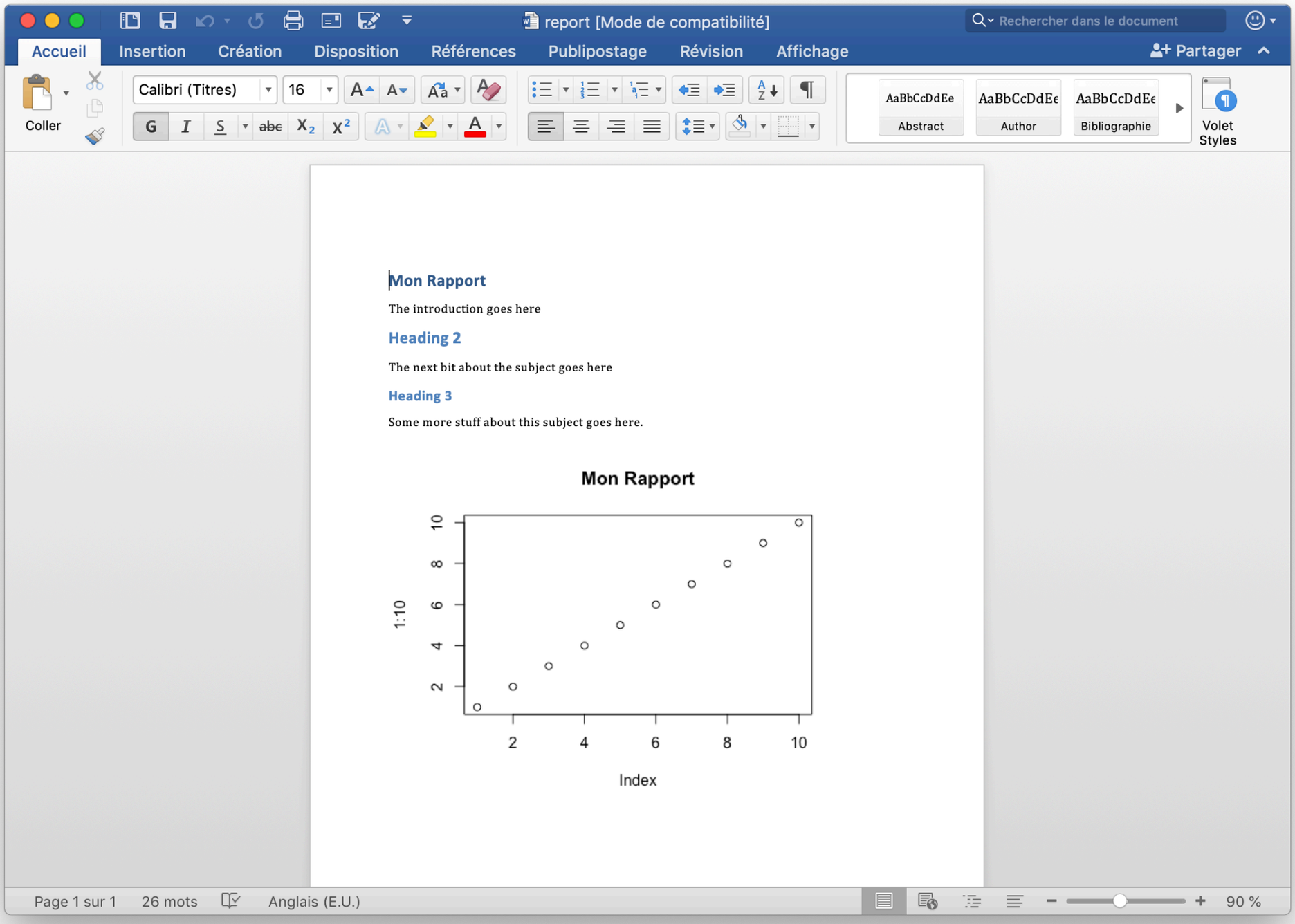
Rapports dynamiques



Rapports dynamiques

- En chargeant le rapport généré au format Word, on remarque que le titre du document et du graphique correspondent à ce qu’on a saisi dans le champ teste « Title » :

Rapports dynamiques



Rapports dynamiques

- Le fichier ui.R, qui définit l'interface utilisateur, contient le code suivant :

```
fluidPage(

  titlePanel("Title the report"),

  textInput("title", "Title", value = "Your title here"),

  hr(),

  downloadButton("downloadWordReport", "Download word report")

)
```

Rapports dynamiques

- Le code précédent définit le titre de la page, un champ texte pour spécifier le titre du rapport et un bouton.

- Ce bouton correspond à une fonction, downloadWordReport, qui chargera le fichier mentionné dans le programme server.R en argument de la fonction downloadHandler.

Rapports dynamiques

- Dans le programme server.R, on commence par charger la libraire rmarkdown, qui est utilisée pour effectuer la conversion du document.

- On y spécifie également ce que fait la fonction downloadWordReport mentionnée dans le programme ui.R.

- Le code correspondant est le suivant :

Rapports dynamiques

```
library(rmarkdown)

function(input, output){

  output$downloadWordReport =
    downloadHandler(filename = "report.docx",
                    content = function(file){

                      render("report.Rmd", output_format = "word_document",
                             output_file = file,
                             quiet = TRUE)
                    })
}
```

Rapports dynamiques

- La fonction downloadHandler a la structure suivante :

- La première partie définit le nom de fichier (filename) assigné au rapport. Celui-ci peut prendre n'importe quelle forme. On peut par exemple utiliser la date courante dans celui-ci ou un nom quelconque avec une extension.

- La deuxième partie, la fonction content, est utilisée pour écrire le contenu du rapport dans le fichier passé en argument (file), sous le nom spécifié par filename.

Rapports dynamiques

- Dans notre application, la fonction content convertit le document report.Rmd en document Word et le sauvegarde sous le nom report.docx.

- Considérons à présent le document R Markdown initial :

Rapports dynamiques

```
---

output: word_document

---

# `r input$title`

The introduction goes here

## Heading 2

The next bit about the subject goes here

### Heading 3

Some more stuff about this subject goes here.

```{r, echo = FALSE}

plot(1:10, main = input$title)

```
```

Rapports dynamiques

- Dans le document R Markdown, on commence par spécifier le format de sortie.

- C'est optionnel car on mentionne déjà ce format dans la fonction content du fichier server.R.

- On utilise les dièses # pour spécifier le niveau des titres de section.

- **De plus, le graphique et le premier titre récupèrent le titre dynamique spécifié dans le champ texte de l'IU pour l'afficher.**

Gabarits HTML

- Les **templates** (gabarits, modèles) HTML sont des gabarits clefs-en-main, combinant dans les mêmes programmes du code HTML et du code Shiny.

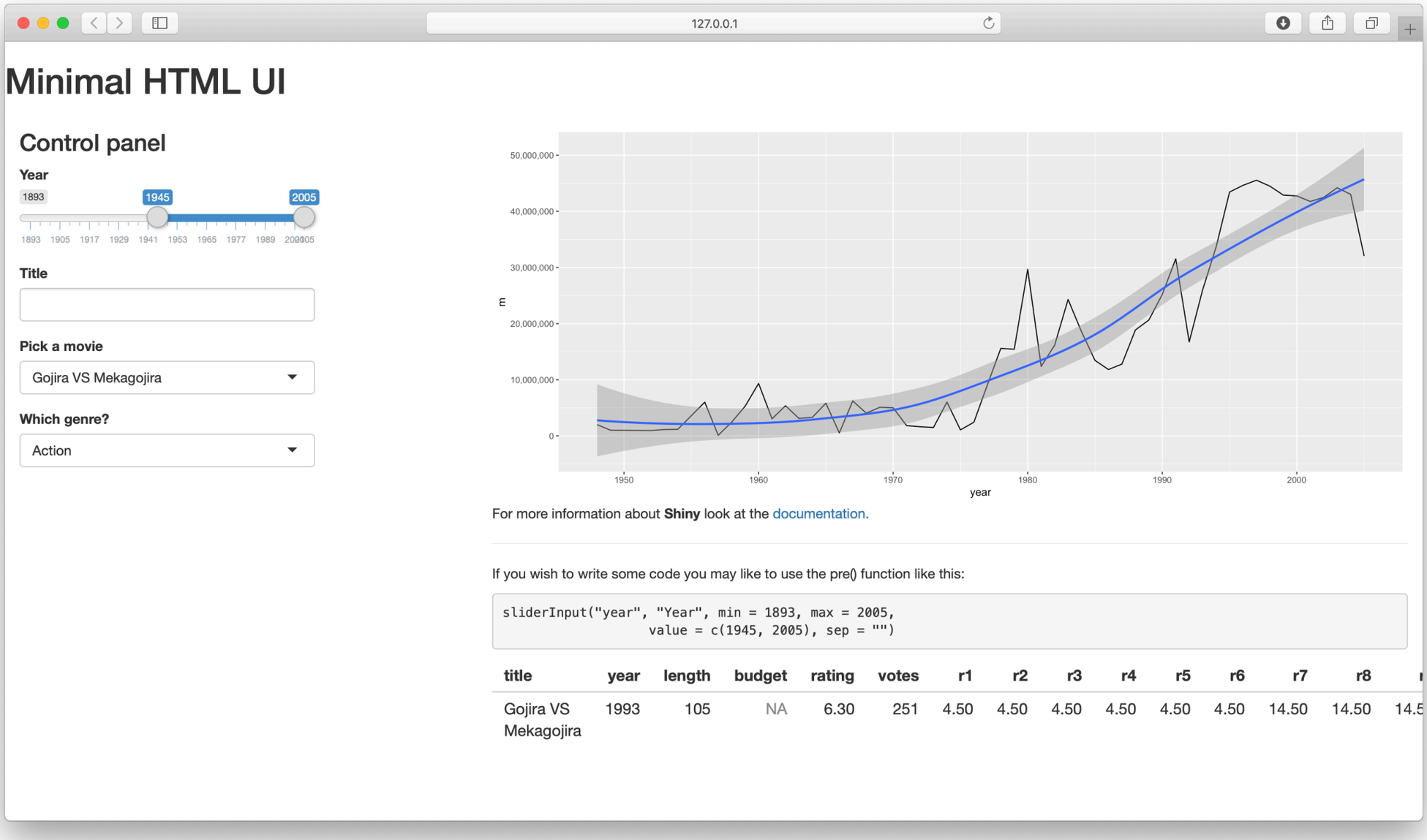
- On va voir dans la suite comment inclure du code R directement dans un template HTML et comment utiliser du code R dans un template à partir d'un fichier séparé.

Gabarits HTML

- Nous allons créer la première application dotée d’une règle coulissante.

- C’est beaucoup plus facile en utilisant du code Shiny que du HTML brut.

Gabarits HTML



Gabarits HTML

- Il y a deux façons d'utiliser R avec les gabarits HTML.
- Le premier consiste à utiliser une fonction R insérée dans le gabarit.
- Le deuxième consiste à référer dans le gabarit à des variables définies dans un fichier séparé.
- Dans les deux cas, on a besoin de trois fichiers : server.R, ui.R et un fichier .html.

Gabarits HTML

- Dans notre cas, nous utilisons un fichier nommé template.html, mais tout autre nom est possible.

- On pourrait l'appeler par exemple apps.html.

- Ce fichier doit être positionné dans le même répertoire que server.R et ui.R.

Gabarits HTML

- Le fichier server.R est analogue à ceux créés précédemment et le fichier ui.R contient la ligne :

```
htmlTemplate(« template.html »)
```

- Cette ligne a pour effet d'informer l'application Shiny qu'on va utiliser un template HTML.

- Avec la première approche, le fichier .html contient le code suivant :

Gabarits HTML

```
<html>
  <head>
    {{ headContent() }}
    {{ bootstrapLib() }}
  </head>
  <body>
    <h1>Minimal HTML UI</h1>

    <div class="container-fluid">
  <div class="row">
    <div class="col-sm-4">
      <h3>Control panel</h3>

      {{ sliderInput("year", "Year", min = 1893, max = 2005, value = c(1945, 2005), sep = "") }}

      {{ textInput("title", "Title") }}

      <div id="listMovies" class="shiny-html-output"></div>

      {{ selectInput("genre", "Which genre?",  c("Action", "Animation", "Comedy", "Drama", "Documentary", "Romance", "Short")) }}

    </div>
```

Gabarits HTML

```
<div class="col-sm-8">

        {{ plotOutput("budgetYear") }}

        <p>For more information about <strong>Shiny</strong> look at the
        <a href="http://shiny.rstudio.com/articles/">documentation.</a></p>

        <hr>

        <p>If you wish to write some code you may like to use the pre() function like this:</p>
          <pre>sliderInput("year", "Year", min = 1893, max = 2005,
                value = c(1945, 2005), sep = "")</pre>

        <div id = "moviePicker" class = "shiny-html-output"></div>

      </div>

    </div>
  </div>
 </body>
</html>
```

Gabarits HTML

- Ce code HTML est analogue à la définition brute utilisée précédemment, excepté en ce qui concerne les inputs et les outputs.

- Les contrôles d'input et les outputs sont du code Shiny au lieu d'être du code HTML.

- Intéressons-nous au **fonctions** apparaissant dans le code Shiny.

Gabarits HTML

- Au début de la section head, deux fonctions interviennent : {{ headContent() }} et {{ bootstrapLib() }}.

- {{ headContent() }} a pour effet de générer le code HTML vu précédemment, qui définit les liens vers JavaScript et CSS.

- {{ bootstrapLib() }} permet de charger la librairie Bootstrap.

Gabarits HTML

- On définit le slider, le champ texte et le champ de saisie en utilisant des commandes Shiny au sein du code HTML.

- Toutes les commandes Shiny sont entourée de **doubles accolades**.

- Le seul inconvénient de cette approche est qu'on ne peut pas utiliser de **sauts de ligne**, car il ne sont pas traités comme en HTML brut.

Gabarits HTML

- Considérons à présent **la deuxième façon** d'utiliser les gabarits HTML.

- La structure de fichiers est analogue à celle vue précédemment : on a trois fichiers, server.R, ui.R et un fichier .html (ici template.html).

- On commence par créer le fichier HTML :

Gabarits HTML

```
<html>
  <head>
    {{ headContent() }}
    {{ bootstrapLib() }}
  </head>
  <body>
    <h1>Minimal HTML UI</h1>
    <div class="container-fluid">
      <div class="row">
        <div class="col-sm-4">
          <h3>Control panel</h3>
          {{ slider }}
          {{ text }}
          <div id="listMovies" class="shiny-html-output"></div>
           {{ comboBox }}
          </div>
          <div class="col-sm-8">
          {{ thePlot }}
          <p>For more information about <strong>Shiny</strong> look at the <a href="http://shiny.rstudio.com/articles/">documentation.</a></p>
          <hr>  <p>If you wish to write some code you may like to
          use the pre() function like this:</p>
          <pre>sliderInput("year", "Year", min = 1893, max = 2005,
                  value = c(1945, 2005), sep = "")</pre>
        <div id = "moviePicker" class = "shiny-html-output"></div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Gabarits HTML

- On retrouve les deux fonctions standard en début de code ; **la principale différence est qu'on définit des noms de variable entre doubles accolades** plutôt que de définir les fonctions utilisées complètement, comme dans le code précédent.

- Plus précisément, on définit les variables {{ spider }}, {{ text }}, {{ comboBox }} et {{ thePlot }}.

Gabarits HTML

- On passe à présent au fichier ui.R.

- On commence par spécifier le nom du gabarit HTML puis on définit les variables utilisées dans le fichier HTML.

- Le code du fichier ui.R final est le suivant :

Gabarits HTML

```
htmlTemplate(
  "template.html",
  slider = sliderInput("year", "Year", min = 1893, max = 2005,
                       value = c(1945, 2005), sep = ""),
  text = textInput("title", "Title"),
  thePlot = plotOutput("budgetYear"),
  comboBox = selectInput("genre", "Which genre?",
                         c("Action", "Animation", "Comedy", "Drama",
                           "Documentary", "Romance", "Short"))
)
```

Gabarits HTML

- On a définit la variable spider à l'aide de la fonction sliderInput, text à l'aide de textInput, thePlot à l'aide de plotOutput et comboBox à l'aide de selectInput.

- Ainsi, on peut définir toutes les variables dans le fichier ui.R au lieu de les définir dans le fichier HTML.

- Par exemple, on peut définir une chaîne de caractère qui contient un nom d'auteur, utilisable partout dans le document HTML.

- Si on doit changer le nom d'auteur, il suffit alors de le faire une fois dans le fichier ui.R.

B. Les dispositions Shiny

Dispositions

- On a vu dans la première partie les bases de la programmation Shiny et diverses commandes Shiny.

- Dans cette partie, nous allons nous intéresser aux fonctions de disposition (ang. layout functions) de Shiny.

- Plus précisément, nous allons :

Dispositions

1. Expliquer comment fonctionne Bootstrap pour Shiny

2. Expliquer comment produire des dispositions en lignes et en colonnes

3. Présenter les différentes fonctions de disposition disponibles sous Shiny

4. Présenter les barres de navigation et les listes de navigation sous Shiny

5. Présenter des éléments d'IU conditionnels et comment contrôler l'IU

Bootstrap

- Bootstrap est une boîte à outils open source basée sur HTML, CSS et JavaScript.

- Elle est particulièrement intéressante pour les développeurs Web car elle permet de réaliser des sites qui s'adaptent à l'écran sur lequel ils sont affichés, que ce soit sur Ordinateur de Bureau, tablette ou téléphone mobile.

- Shiny fait appel par défaut à la version 3 de Bootstrap. Toutefois, la version 4 est d'ores et déjà disponible.

- La principale fonctionnalité de Bootstrap utilisée est son **système de grille** (ang. grid system).

Bootstrap

- Le système de grille est une façon d'organiser le contenu en lignes et en colonnes.

- Apprendre Shiny, c'est aussi apprendre divers composantes de Bootstrap, comme les barres de navigations, les panneaux et les fenêtres d'alerte.

Bootstrap

- Bootstrap peut être utilisé avec des **gabarits HTM**L ou avec du HTML brut et on peut utiliser nos propres CSS.

- La fonction bootstrapPage( ) fournit une IU Shiny minimale.

- Cette fonction charge les CSS Bootstrap et JavaScript mais ne fait rien d'autre en termes de HTML.

- Pour plus d'infos sur cette fonction, on pourra consulter le site Shiny : https://shiny.rstudio.com.

Bootstrap

- Bootstrap propose un thème par défaut mais il existe de nombreux thèmes Bootstrap libres et open source sur Internet.

- Pour les utiliser, il suffit d'enregistrer le thème dans un répertoire public www, **de même niveau que l'application** et de spécifier un nom de fichier.

- Par exemple, le thème sandstone.css peut être utilisé comme argument des fonctions fluidPage, fixedPage, bootstrapPage ou navbarPage.

Bootstrap

- Shiny peut utiliser le système de grille de Bootstrap pour disposer du contenu.

- Il y a plusieurs façon de le faire, mais toutes font intervenir des fonctions Shiny. Les différences entre ces fonctions son mineures et la démarches générale est toujours la même.

- Dans la suite, on va s'intéresser au fonctions suivantes : fluidPage( ), bootstrapPage( ) et fixedPage( ).

fluidPage( )

- C’est la façon la plus commune d’utiliser le système de grille de Bootstrap.

- Le code suivant illustre l’utilisation de fluidPage( ) :

fluidPage( )

```
server = function(input, output) {

# server code

}

ui = fluidPage(

     fluidRow(

          column(2, wellPanel(p(« Column width 2 »))),

          column(10, wellPanel(p(« Column width 10 »)))),

     fluidRow(

          column(4, wellPanel(p(« Column width 4 »))),

          column(4, wellPanel(p(« Column width 4 »))),

          column(4, wellPanel(p(« Column width 4 »))))

)

shinyapp(ui = ui, server = server)
```

fluidPage( )

- La fonction fluidPage organise l'affichage en ligne.

- Dans le code précédent, celui-ci est organisé en deux lignes, identifiées par les fonctions fluidRow.

- Les lignes sont composées de colonnes : la première ligne est composée de deux colonnes, l'une de 2 unités en largeur et l'autre de 10 unités. **Les largeurs de colonnes doivent toujours être de somme égale à 12**. La deuxième ligne est composée de trois colonnes, chacune de largeur 4.

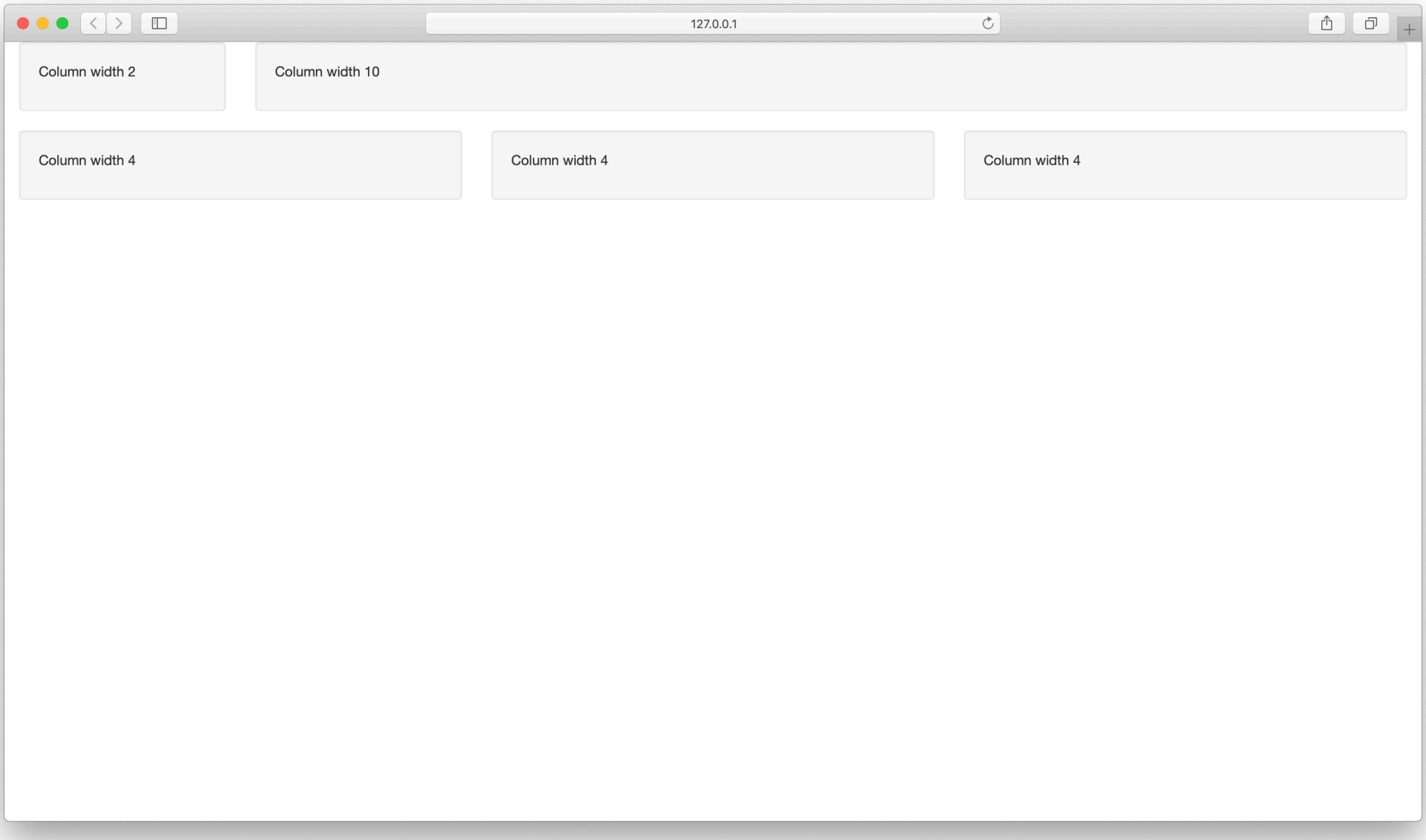
fluidPage( )

- Les colonnes contiennent des panneaux (ang. panels) et ceux-ci peuvent contenir des objets tels que des contrôles, des graphiques, des tableaux, etc.

- C’est Bootstrap qui se charge de dimensionner les éléments sur l’écran utilisé.

- Le code précédent (disponible sur le forum sous le nom app.R) affiche la sortie suivante :

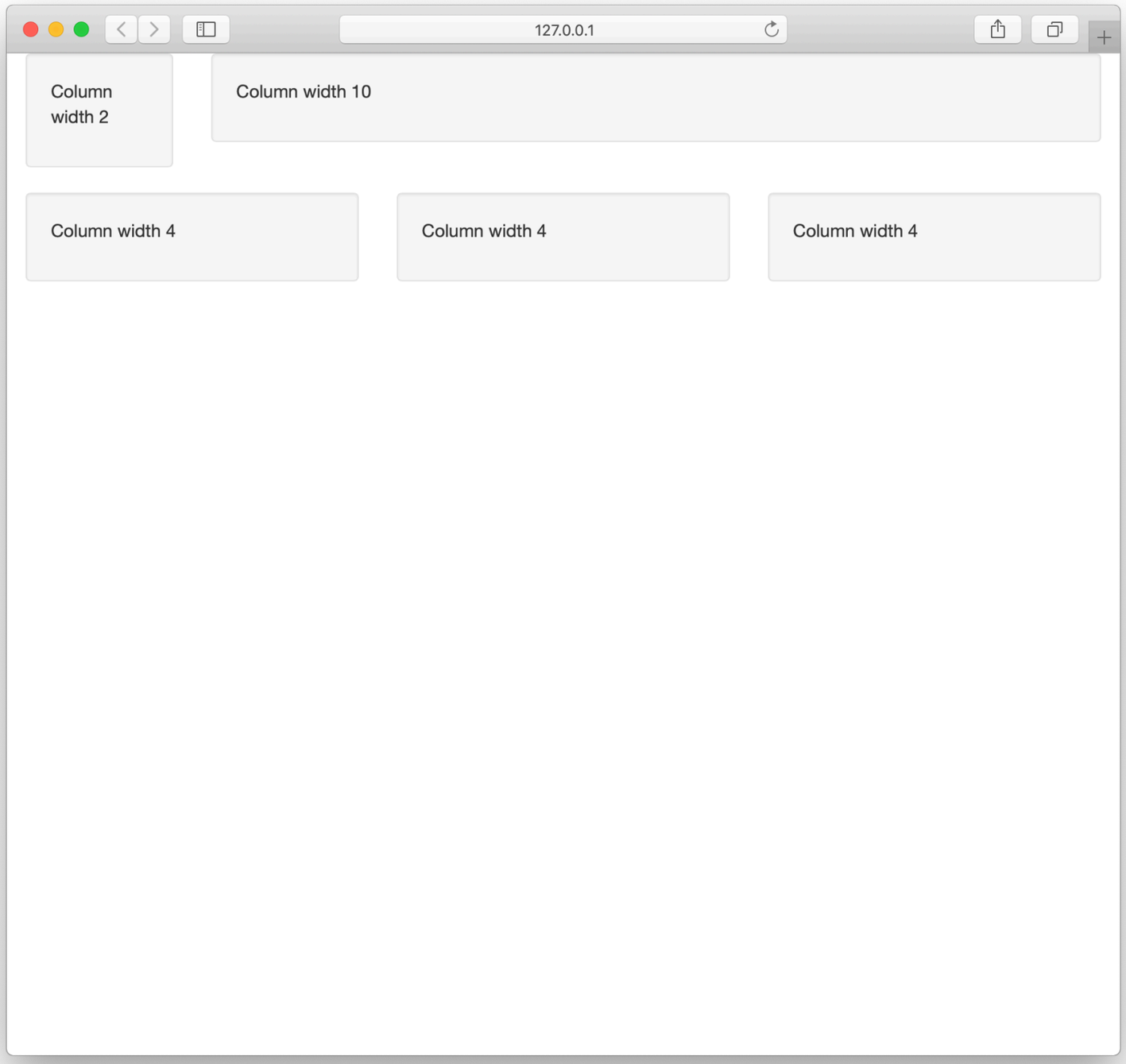
fluidPage( )


127.0.0.1
Column width 2
Column width 10
Column width 4
Column width 4
Column width 4

fluidPage( )

- Si on redimensionne le fenêtre, les éléments sont redimensionnés automatiquement :

fluidPage( )


127.0.0.1
Column width 2
Column width 10
Column width 4
Column width 4
Column width 4

fluidPage( )

- On peut également imbriquer des colonnes, comme c’est fait dans le code suivant :

fluidPage( )

```
server = function(input, output){

    # server code

}
```

fluidPage( )

```
ui = fluidPage(


  fluidRow(
    column(2, wellPanel(p("Column width 2"))),
    column(10, wellPanel(p("Column width 10")))),
  fluidRow(
    column(2, wellPanel(p("Column width 2"))),
    column(10, wellPanel(p("Column width 10")),
           fluidRow(
             column(6, wellPanel(p("Column width 6"))),
             column(6, wellPanel(p("Column width 6")))
           )
    )
  )
)
```

fluidPage( )

```
shinyApp(ui = ui, server = server)
```

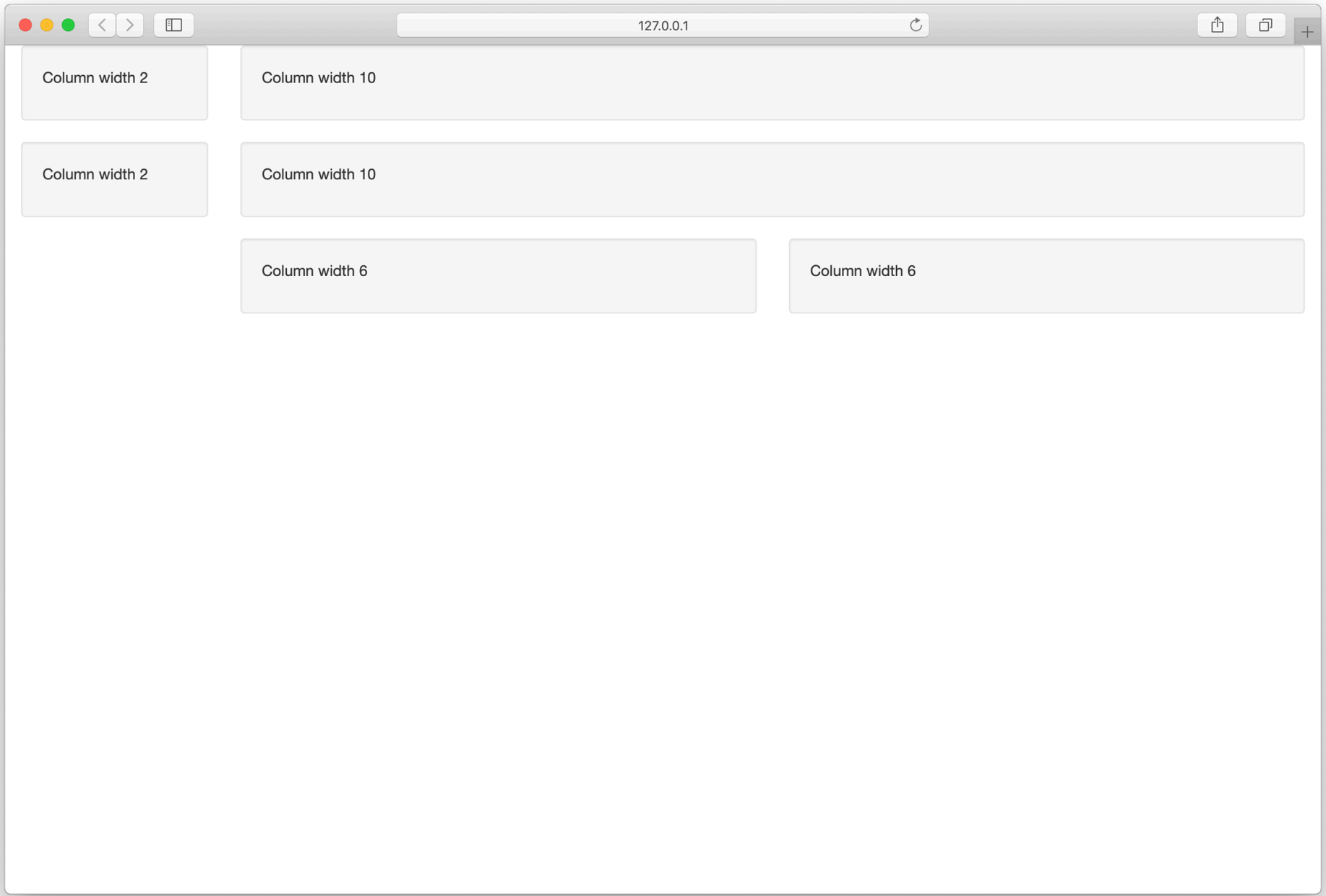
- Les lignes peuvent également être imbriquées dans des colonnes, comme on peut le voir dans le code précédent.

- La dernière contient deux lignes qui contiennent chacune deux colonnes.

- Les colonnes de cette ligne ont des largeurs de somme 12, bien qu'elles ne correspondent qu'à une partie de la largeur de la page.

- Le résultat à l'affichage est le suivant :

fluidPage( )


127.0.0.1
Column width 2
Column width 10
Column width 2
Column width 10
Column width 6
Column width 6

bootstrapPage( )

- La fonction bootstrapPage( ) ne fait qu'une seule chose : charger la boîte à outils Bootstrap.

- Une fois la boîte à outil chargée, les panneaux se terminent abruptement à droite de la page, mais leur taille s'ajuste quand même automatiquement.

- On règle le problème en rajoutant un container fluide « div », ce qui a pour effet de générer le même code HTML que dans l'exemple précédent. Le code correspondant est le suivant :

bootstrapPage( )

```
server = function(input, output){

  # server code

}
```

bootstrapPage( )

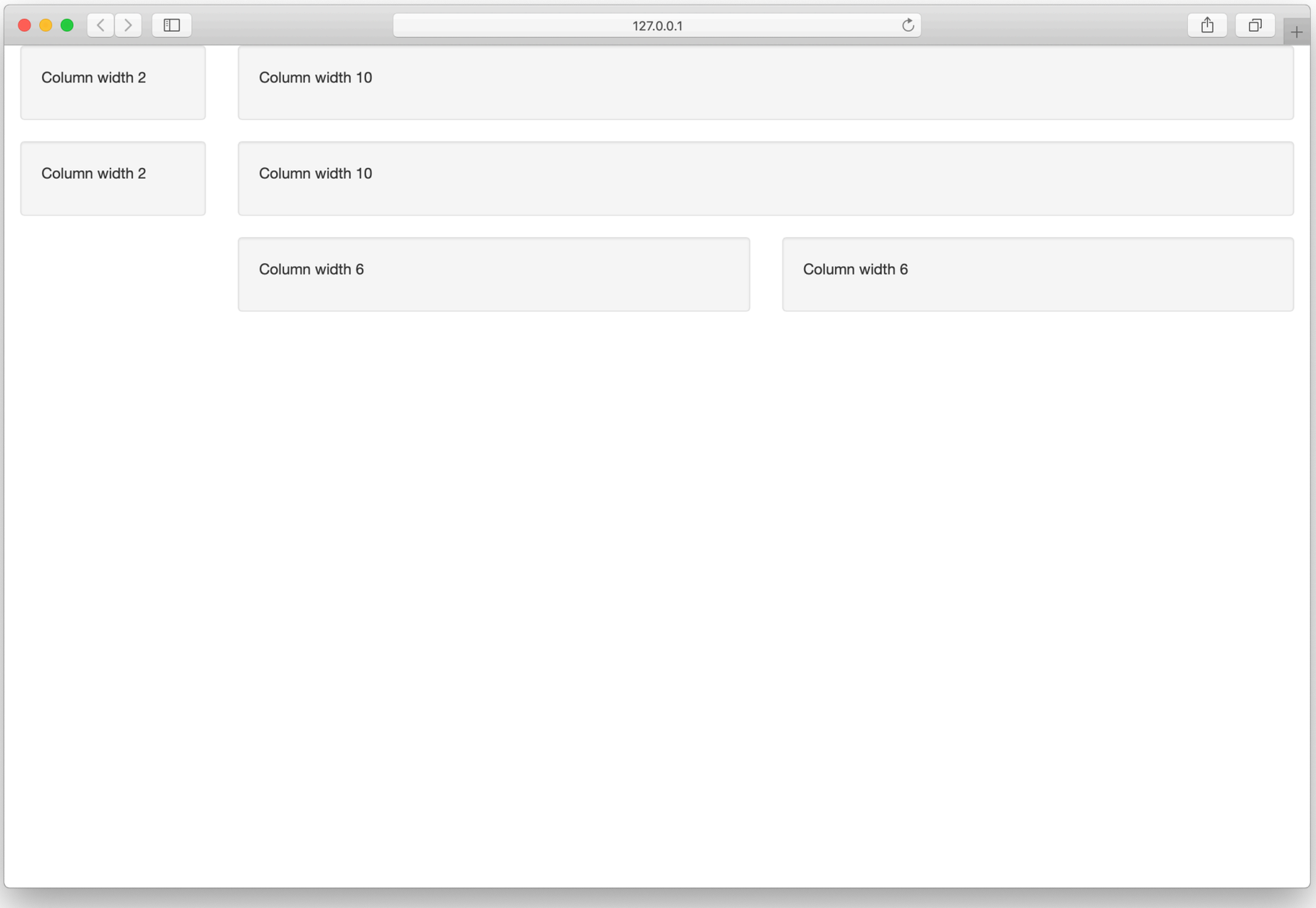
```
ui = bootstrapPage(
  div(class = "container-fluid",
      fluidRow(
        column(2, wellPanel(p("Column width 2"))),
        column(10, wellPanel(p("Column width 10")))),
      fluidRow(
        column(2, wellPanel(p("Column width 2"))),
        column(10, wellPanel(p("Column width 10")),
               fluidRow(
                 column(6, wellPanel(p("Column width 6"))),
                 column(6, wellPanel(p("Column width 6")))
               )
        )
      )
  )
)
```

bootstrapPage( )

```
shinyApp(ui = ui, server = server)
```

- L’affichage obtenu en sortie est le suivant :

bootstrapPage( )



fixedPage( )

- fixedPage( ) est utilisée avec des lignes de largeur fixée, comme l’illustre le code suivant :

fixedPage( )

```
server = function(input, output){

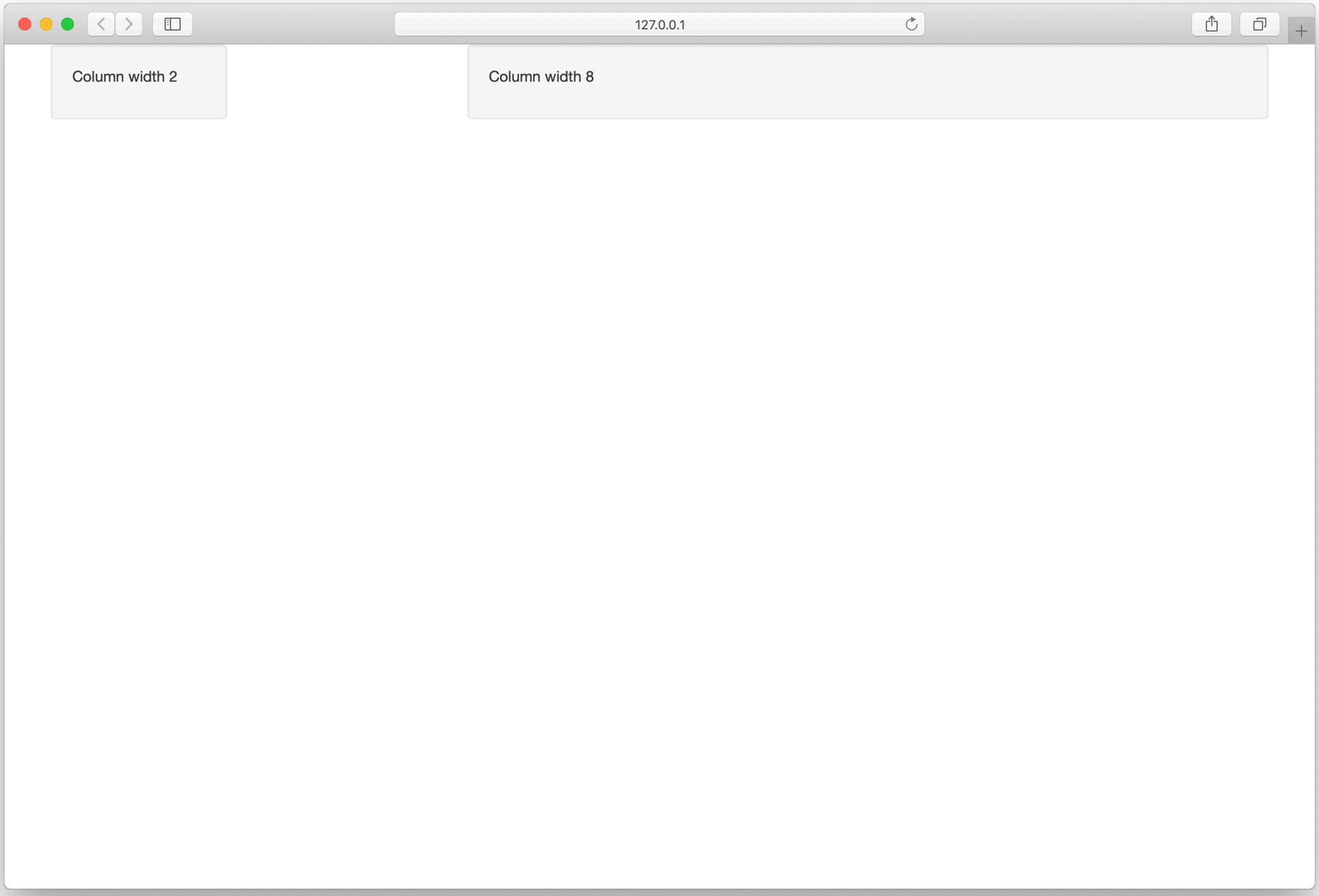
  # server code
}

ui = fixedPage(

  fixedRow(
    column(2, wellPanel(p("Column width 2"))),
    column(8, wellPanel(p("Column width 8")), offset = 2))
)

shinyApp(ui = ui, server = server)
```

fixedPage( )



fixedPage( )

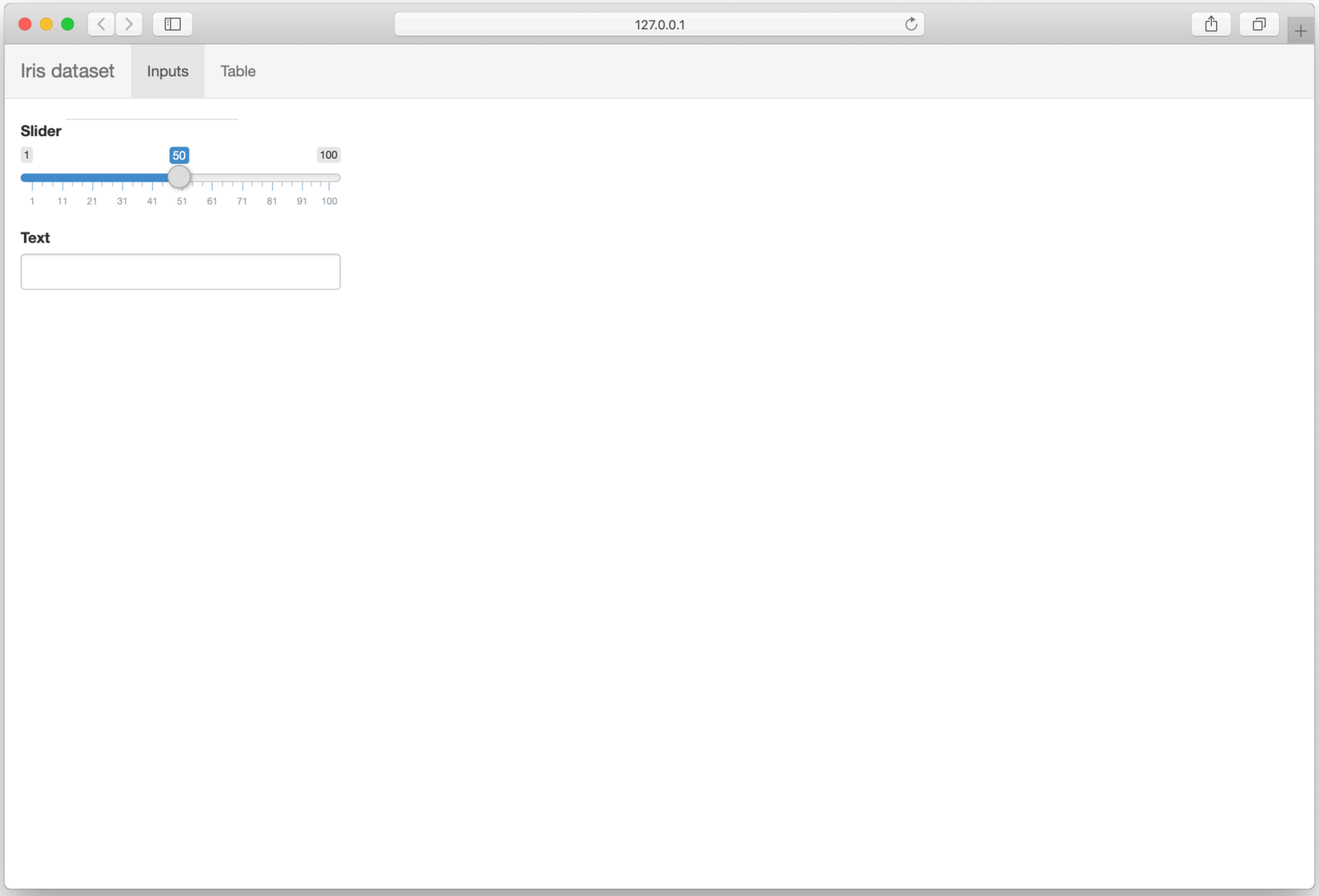
- On voit qu’un espace a été inséré entre les deux colonnes.

- Lors du redimensionnement, l’espace est maintenu tant que possible.

Barre de navigation

- Sur l’écran suivant, dans le premier onglet (Inputs), on dispose des contrôles du pourcentage d’observations et titre de l’application « Iris dataset » :

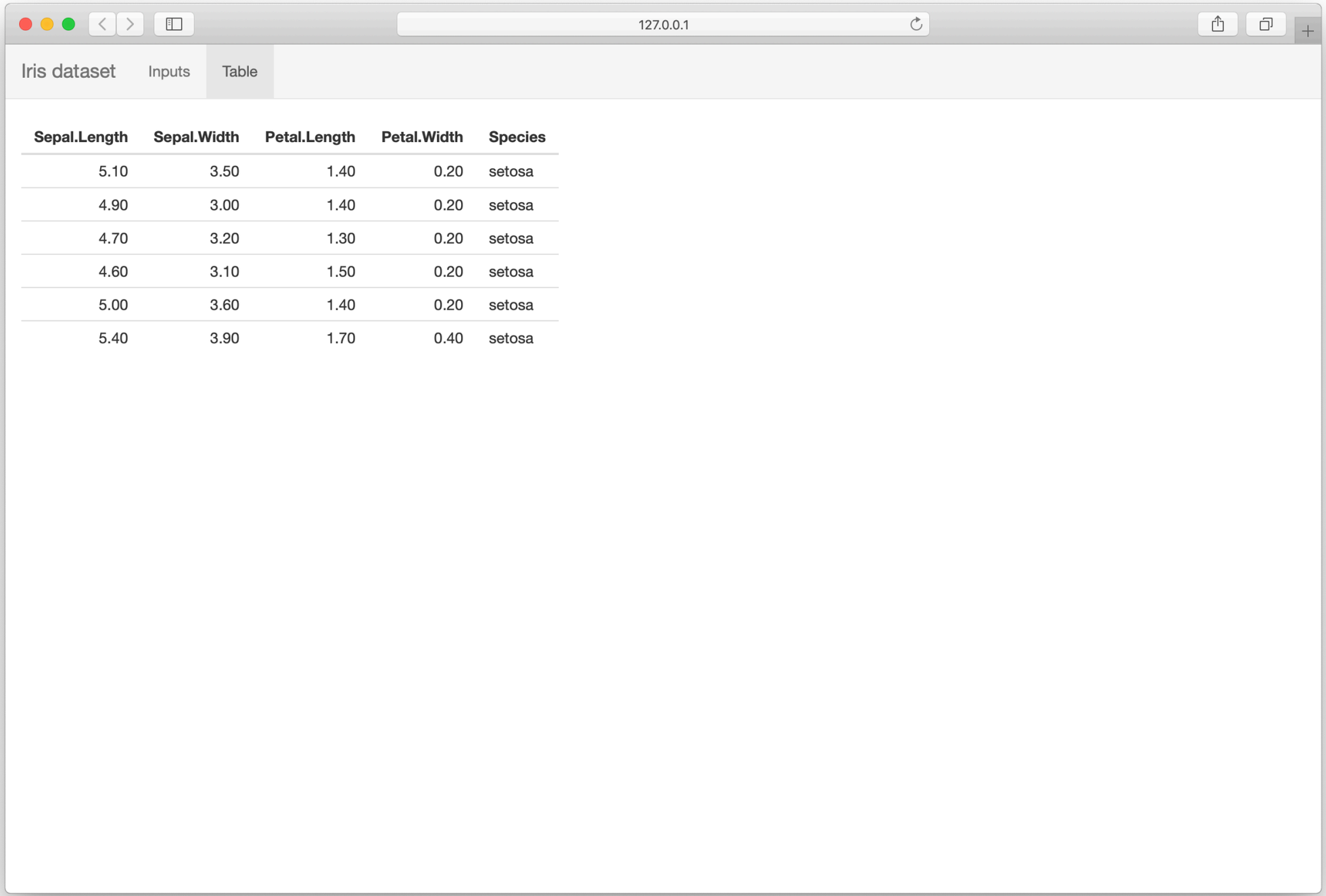
Barre de navigation



Barre de navigation

- Dans le deuxième onglet (Table), on a une table contenant les dix premières lignes du data frame « iris » :

Barre de navigation



Barre de navigation

- Le code correspondant est le suivant :

Barre de navigation

```
server = function(input, output) {


  output$table = renderTable({

    head(iris)

  })


}
```

Barre de navigation

```
ui = navbarPage("Iris dataset",
                id = "navBar",
                tabPanel("Inputs",
                         sliderInput("slider", "Slider",
                                     min = 1, max = 100, value = 50),
                         textInput("text", "Text")),
                tabPanel("Table", tableOutput("table"))
)
shinyApp(ui, server)
```

Barre de navigation

- Au lieu d'être configurée avec fluidPage( ), la barre de navigation est configurée à l'aide de la fonction navbarPage( ).

- On lui a assigné un identifiant (« navBar »), qu'on peut utiliser dans input$navBar.

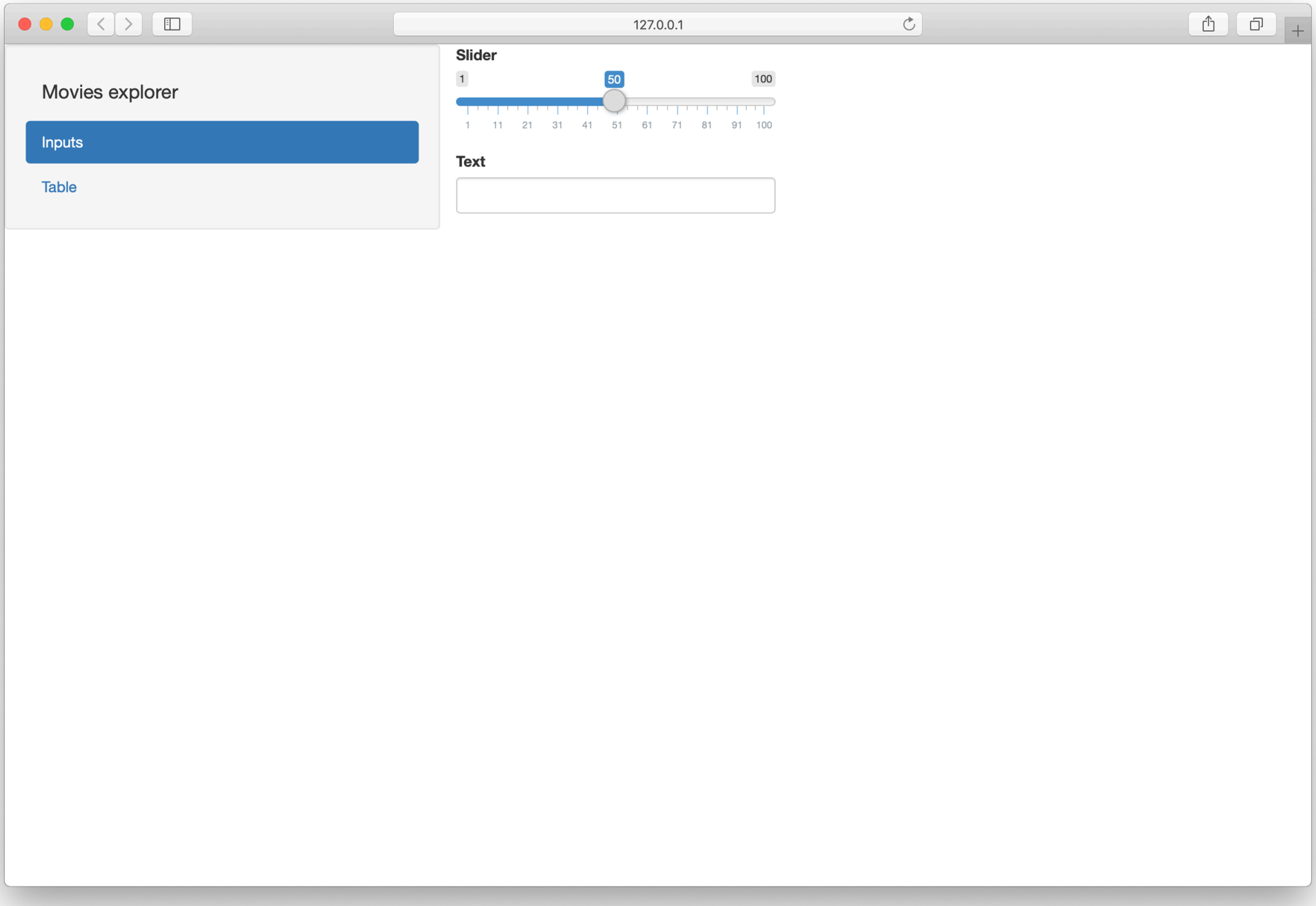
- L'input navBar prend la valeur du contenu correspondant à l'onglet sélectionné.

Liste de navigation

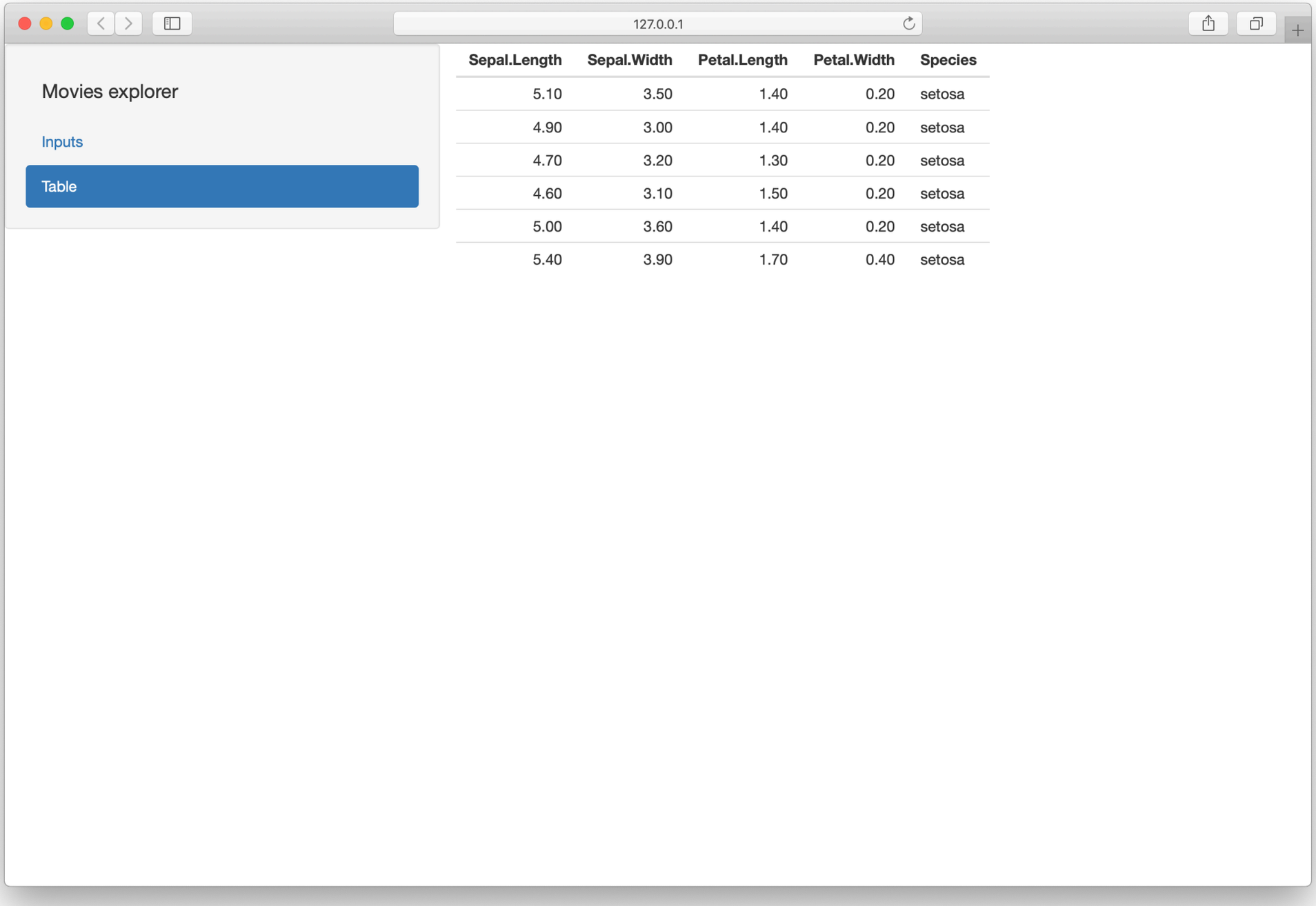
- La **liste de navigation** est similaire à la barre de navigation, à ceci près qu'elle propose les différentes options sous la forme d'une liste, positionnée sur le côté de la page affichée.

- La fonction de disposition utilisée est navlistPanel.

Liste de navigation



Liste de navigation



Liste de navigation

- Le code correspondant est le suivant :

Liste de navigation

```
server = function(input, output) {

  output$table = renderTable({


    head(iris)

  })


}
```

Liste de navigation

```
ui = fluidPage(


  navlistPanel(
    "Iris dataset",
    tabPanel(
      "Inputs",
      sliderInput("slider", "Slider", min = 1, max = 100, value = 50),
      textInput("text", "Text")),
    tabPanel("Table", tableOutput("table"))
  )
)

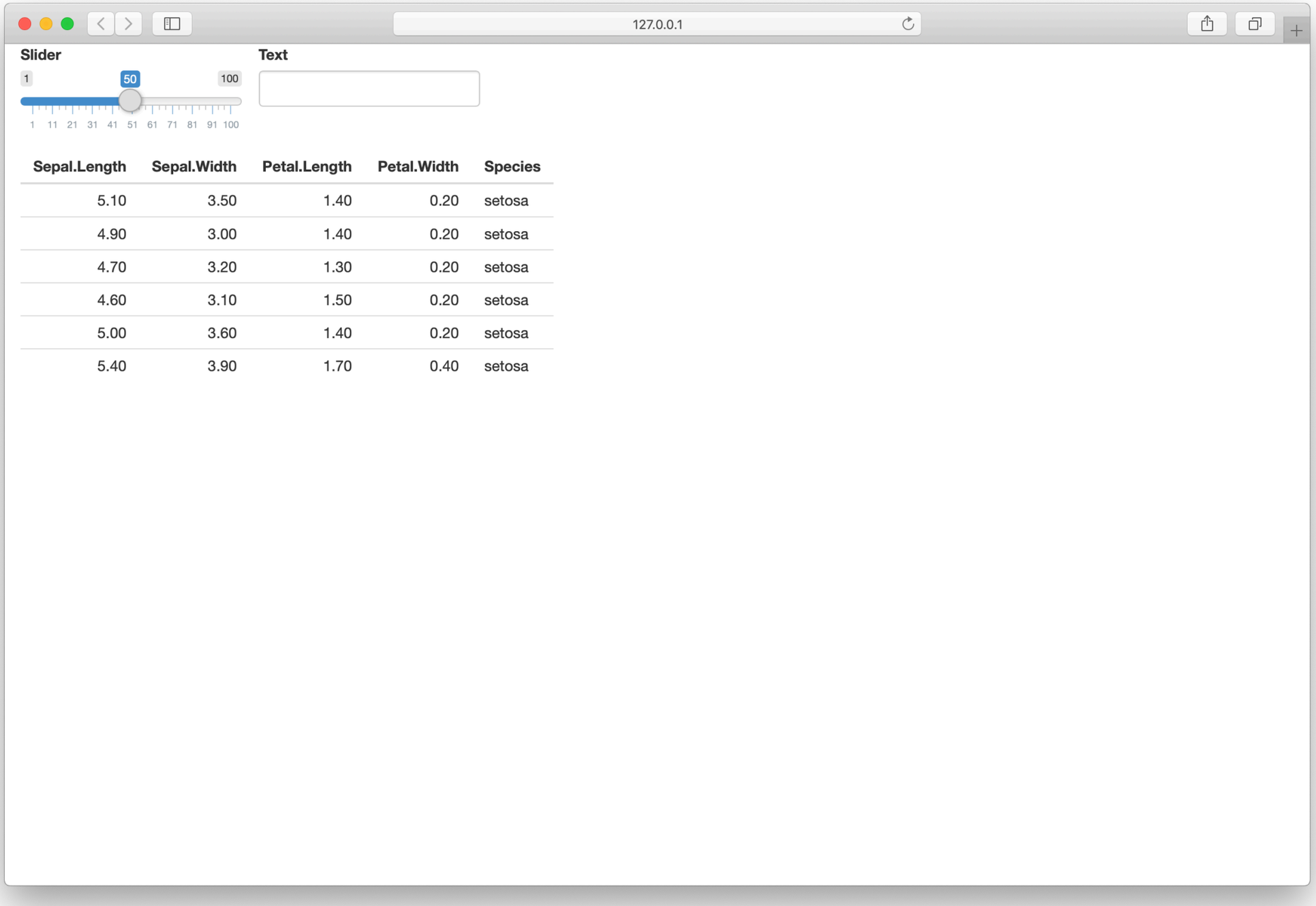

shinyApp(ui, server)
```

Fonctions de disposition

- **flowLayout** : les éléments sont ordonnés de gauche à droite et de haut en bas.

- Un exemple est disponible sur le forum.

flowLayout



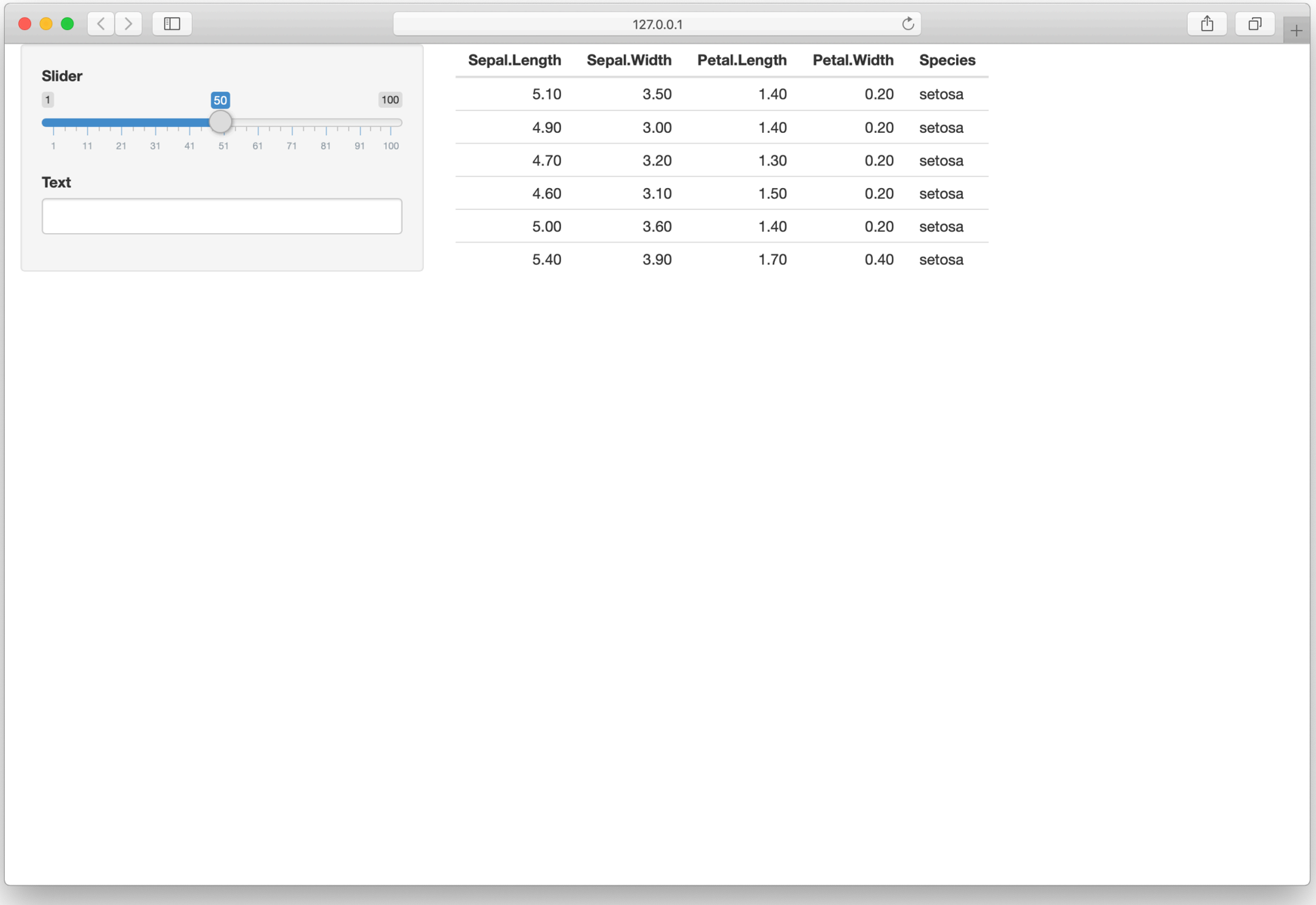
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.10	3.50	1.40	0.20	setosa
4.90	3.00	1.40	0.20	setosa
4.70	3.20	1.30	0.20	setosa
4.60	3.10	1.50	0.20	setosa
5.00	3.60	1.40	0.20	setosa
5.40	3.90	1.70	0.40	setosa

Fonctions de disposition

- **sidebarLayout** : Les input sont positionnés à gauche de l’écran et les outputs à droite.

- Un exemple est disponible sur le forum.

sidebarLayout



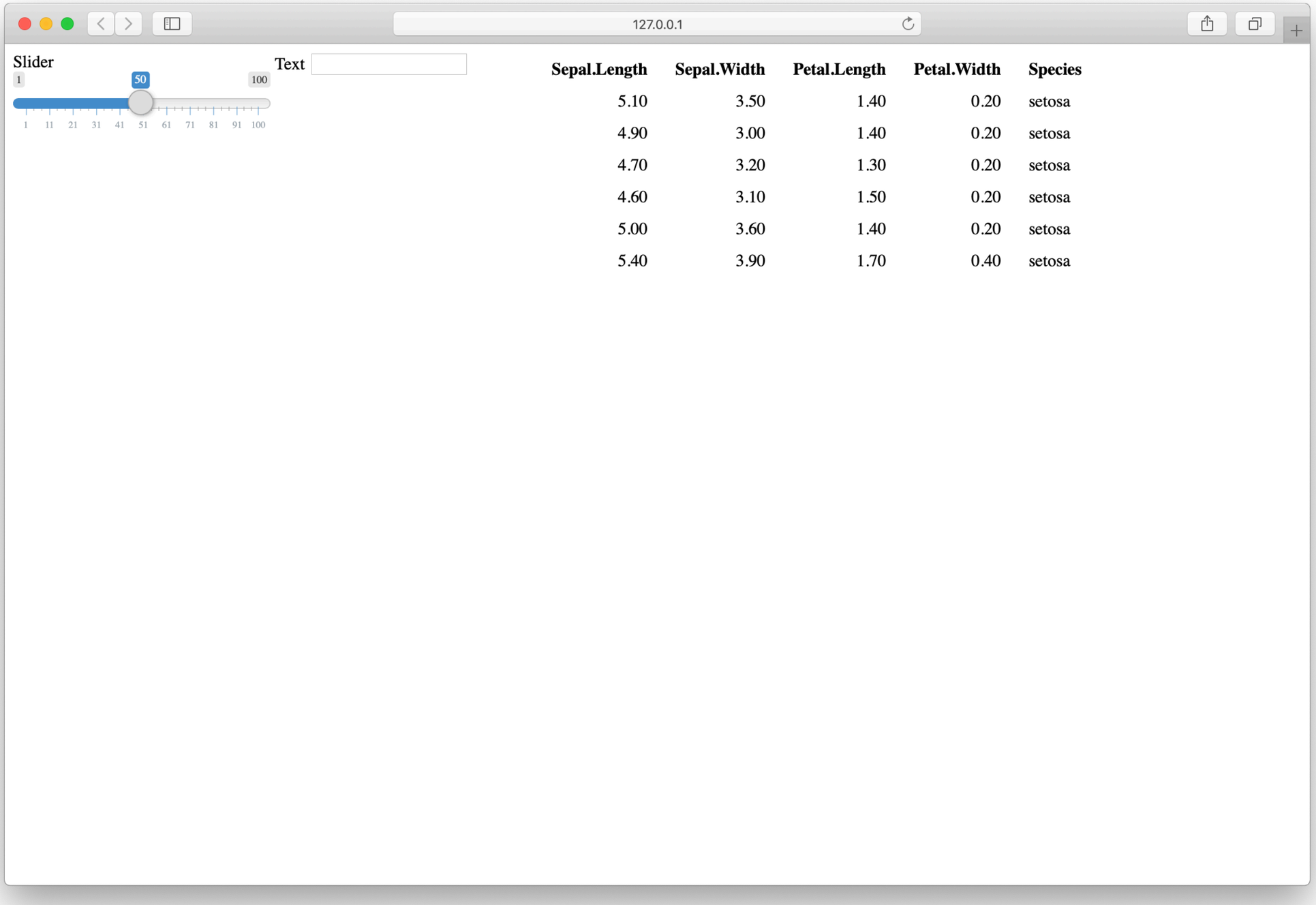
Fonctions de disposition

- **splitLayout** : elle prend tous les éléments passés en argument et les positionne de gauche à droite sur la page.

- Chaque colonne a par défaut la même largeur, mais celle-ci peut être ajustée manuellement.

- Un exemple est donné sur le forum.

splitLayout

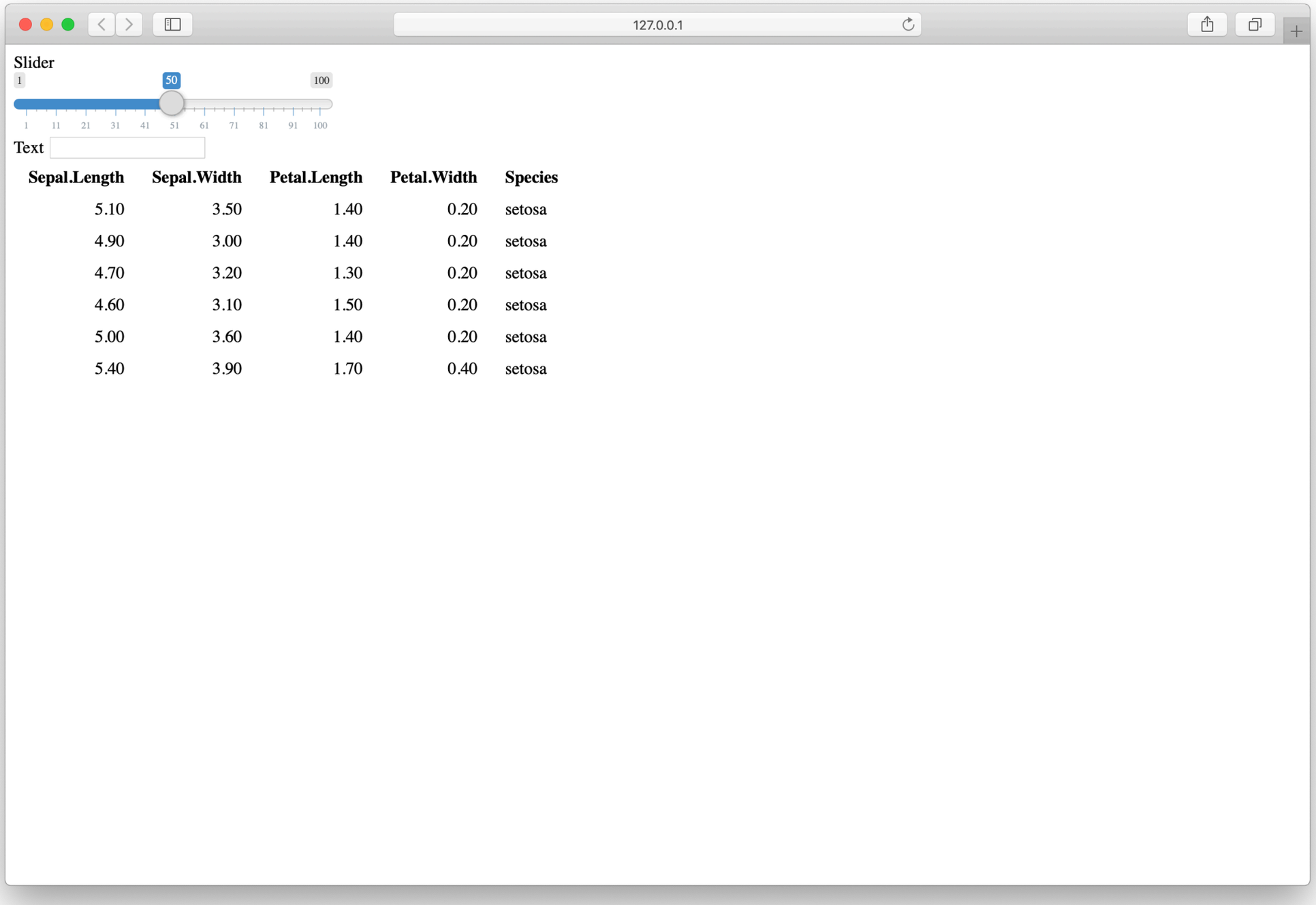


Fonctions de disposition

- **verticalLayout** : elle positionne verticalement les éléments passés en argument.

- Un exemple est donné sur le forum.

verticalLayout



Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.10	3.50	1.40	0.20	setosa
4.90	3.00	1.40	0.20	setosa
4.70	3.20	1.30	0.20	setosa
4.60	3.10	1.50	0.20	setosa
5.00	3.60	1.40	0.20	setosa
5.40	3.90	1.70	0.40	setosa

Fonctions de disposition

- On a vu également les fonctions fluidRow, navBar et navList.

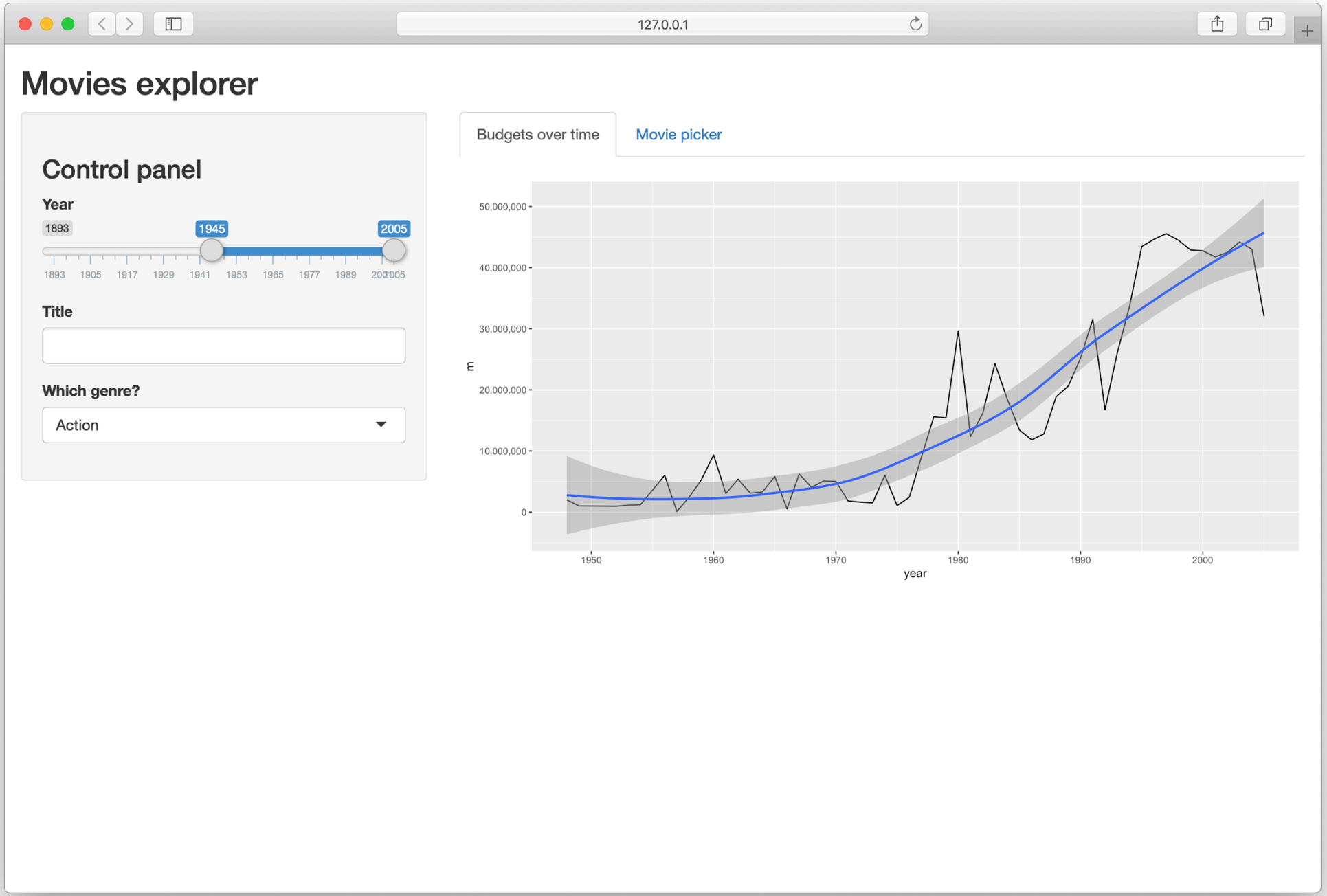
IU conditionnelle

- Dans la suite, on va voir comment utiliser la fonction IU conditionnelle, contrôler l'interface utilisateur avec observe( ) et utiliser des fonctions modales pour transmettre des messages à l'utilisateur.

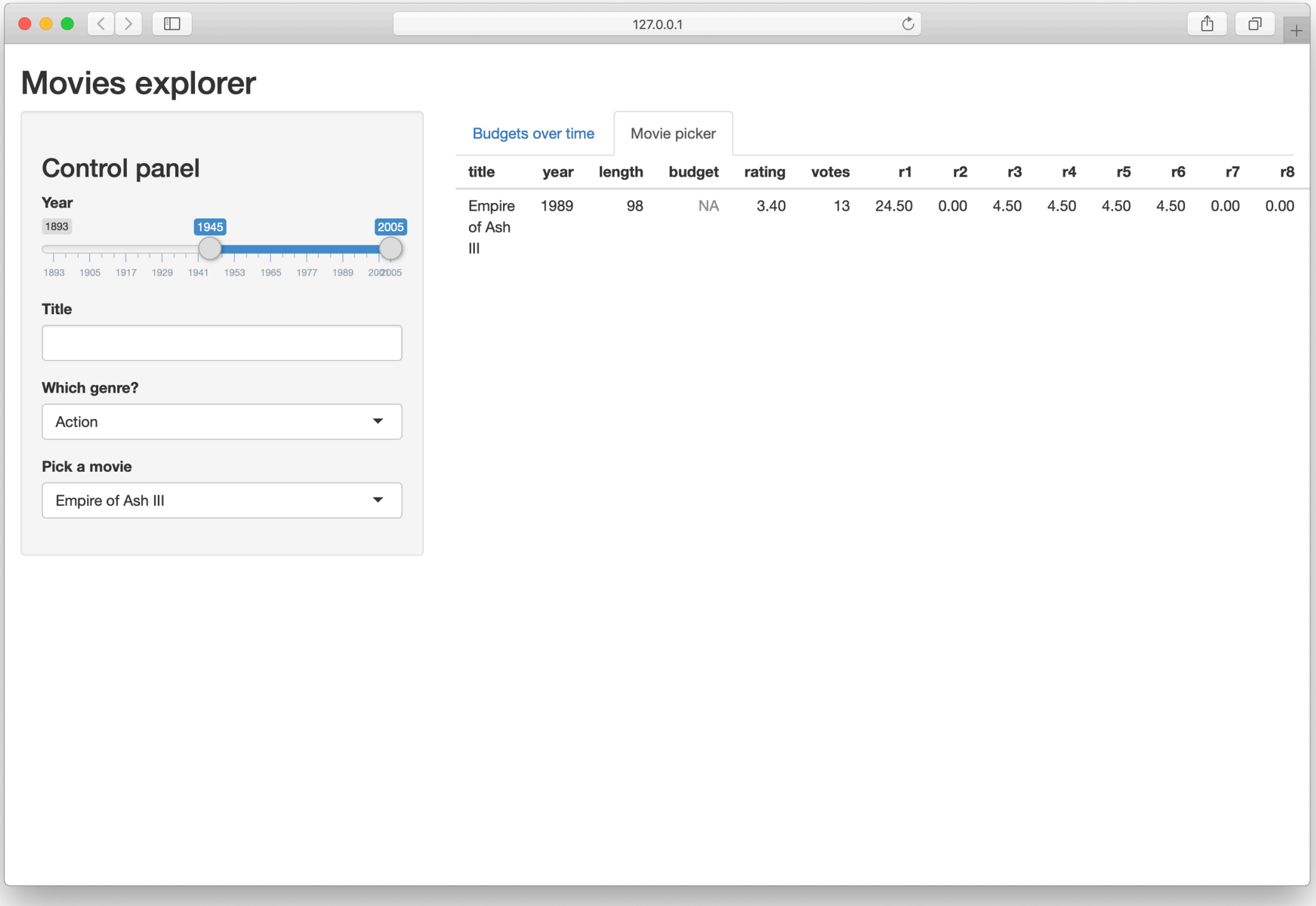
IU conditionnelle

- Commençons par noter que le sélecteur de films n’apparaît pas sur l’écran avec le graphique, comme le montre la fenêtre suivante :

IU conditionnelle



IU conditionnelle



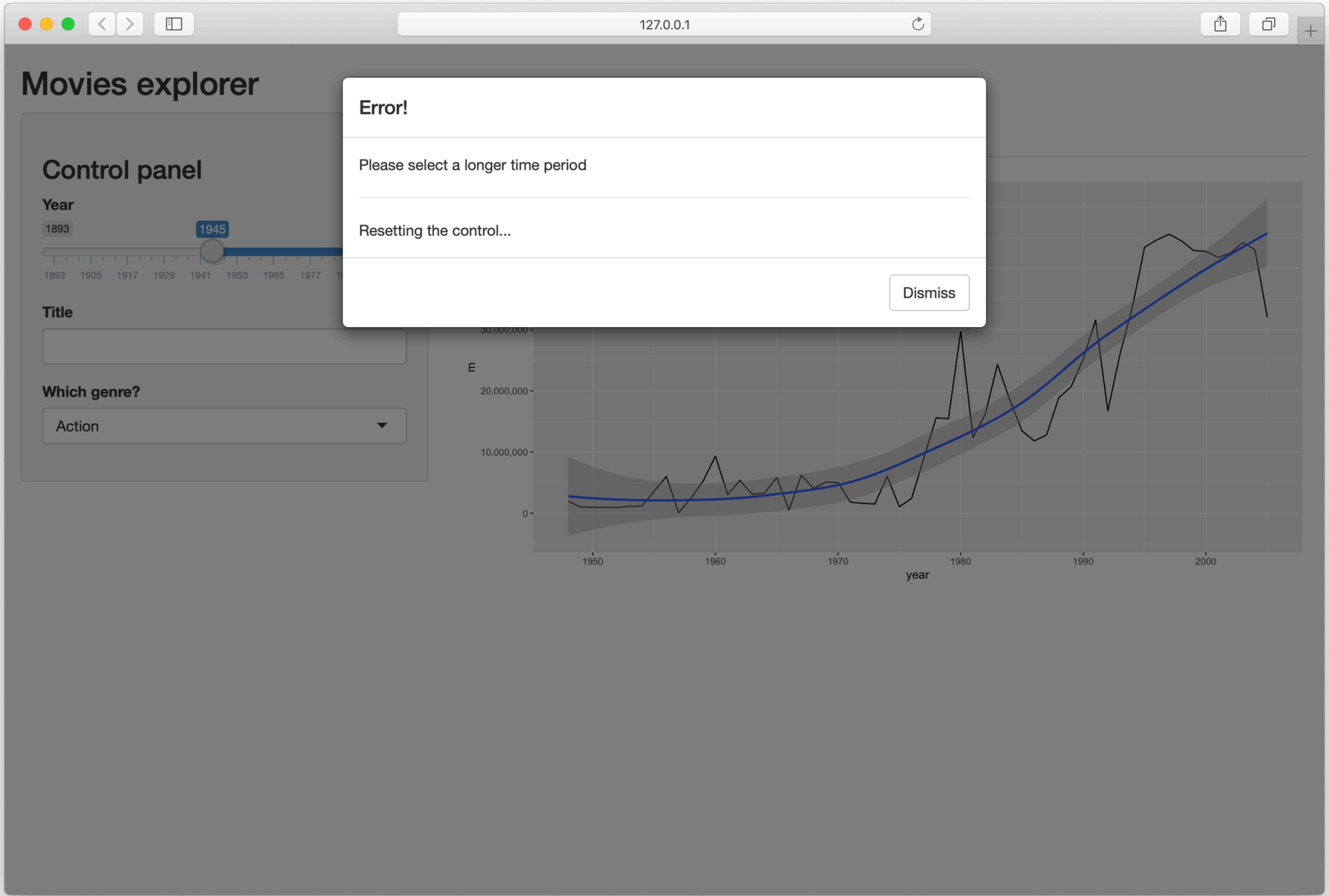
IU conditionnelle

- En effet, il est utile de ne montrer à l'utilisateur que les contrôles qui peuvent être utilisés.

- Cela permet d'éviter de surcharger l'IU et de mieux montrer comment elle fonctionne.

- De plus, si l'utilisateur choisit une période de temps insuffisante (inférieure à dix ans) sur la règle glissante, le contrôle est réinitialisé et un message lui est envoyé sous la forme d'une fenêtre pop-up :

IU conditionnelle



IU conditionnelle

- Deux fonctions permettent de générer une telle fenêtre : l'une sert à la créer et l'autre à l'afficher.

- Elles sont combinées avec la fonction observe( ) qui détecte que la condition d'affichage est vérifiée.

- Ces fonctions sont : modalDialog et showModal.

- Un exemple de code est donné sur ls forum (répertoire : Partie B, fichier : server.R).

C. Tableaux de bord

Tableaux de bord

- Afin de réaliser des tableaux de bord Shiny, il faut commencer par installer le package shinydashboard.

- Il y a deux façons de structurer le code pour un tableau de bord Shiny et cela requiert dans les deux cas la configuration de trois composantes : dashboardHeader, dashbordSidebar et dashboardBody.

Tableaux de bord

- Dans le cadre de la première approche, les trois composantes sont passées en arguments de la fonction dashboardPage :

```
dashboardPage(
    dashboardHeader( ),
    dashboardSidebar( ),
    dashboardBody( )
)
```

Tableaux de bord

- Dans le cadre de la deuxième approche, on définit des variables header, sidebar et body puis on les passe en argument de dashboardPage :

```
header = dashboardHeader( )

sidebar = dashboardSidebar( )

body = dashboardBody( )

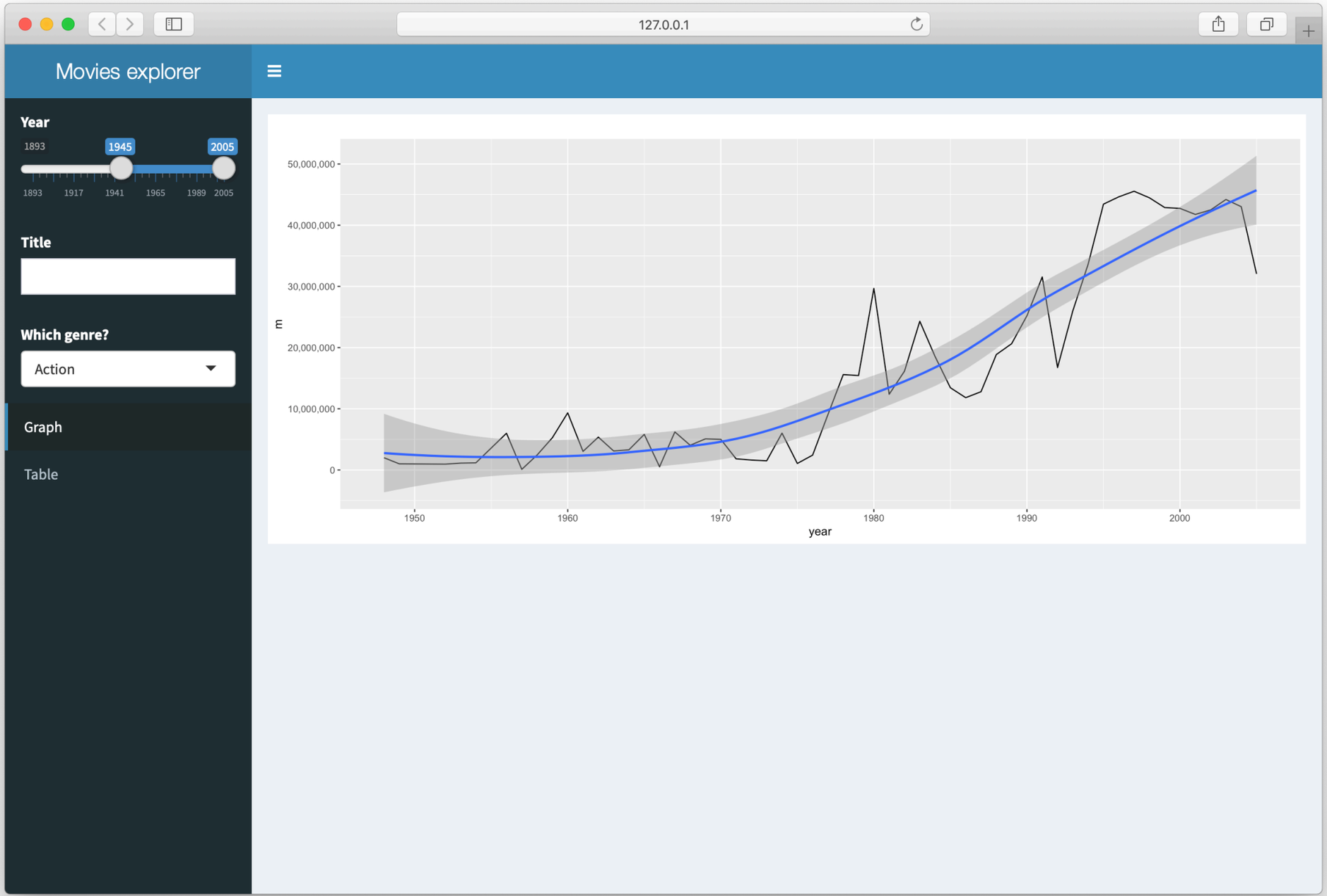
dashboardPage(header, sidebar, body)
```

Tableaux de bord

- La deuxième approche est préférable car elle simplifie la structure du code.

- Examinons à présent notre premier tableau de bord Shiny. Celui-ci correspond à l'application « Movies Explorer » qu'on a vu précédemment :

Tableaux de bord



Tableaux de bord

- Le programme server.R est le même que celui utilisé précédemment. La différence se fait au niveau du programme ui.R.

- Le code correspondant est la suivant :

Tableaux de bord

```
library(shinydashboard)



header = dashboardHeader(title = "Movies
explorer")
```

Tableaux de bord

```
sidebar = dashboardSidebar(
  sliderInput("year", "Year", min = 1893, max = 2005,
              value = c(1945, 2005), sep = ""),
  textInput("title", "Title"),
  selectInput("genre", "Which genre?",
              c("Action", "Animation", "Comedy", "Drama",
                "Documentary", "Romance", "Short")),

  sidebarMenu(
    menuItem("Graph", tabName = "graph"),
    menuItem("Table", tabName = "table")
  )
)
```

Tableaux de bord

```
body = dashboardBody(
  tabItems(
    tabItem(tabName = "graph",
            plotOutput("budgetYear")
    ),
    tabItem(tabName = "table",
            tableOutput("moviePicker"),
            uiOutput("listMovies")
    )
  )
)
```

Tableaux de bord

`dashboardPage(header, sidebar, body)`

- Ce code fait appel aux fonctions usuelles, mais également à sidebarMenu et menuItem et à tabItems et tabItem. Les premières servent à afficher un menu dans la barre latérale et les deuxièmes à définir les onglets affichant les objets correspondants aux différentes options du menu.

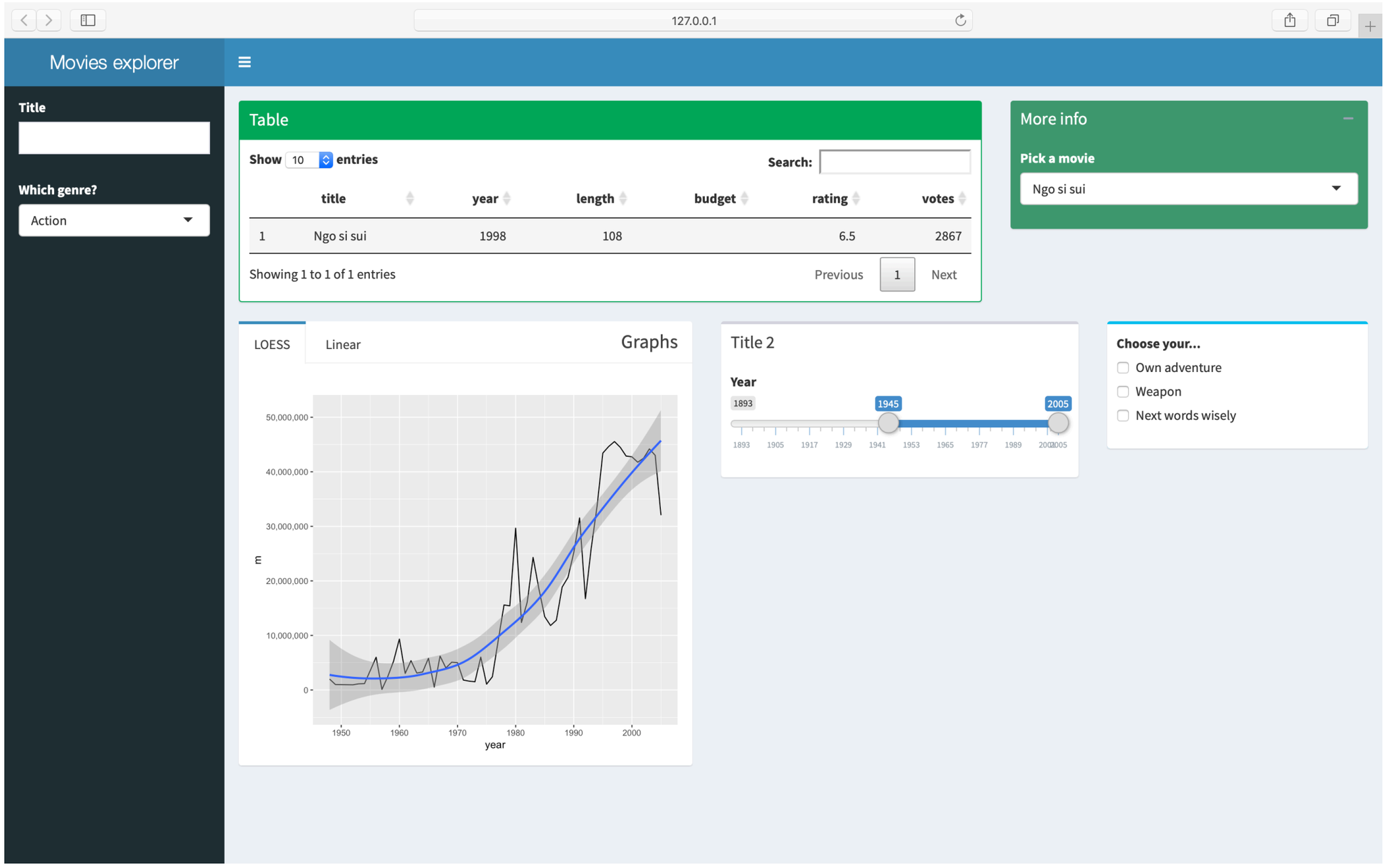
- Le code est disponible sur le forum dans le répertoire Partie C.

Tableaux de bord

- On va voir à présent comment améliorer la présentation d'un tableau de bord à l'aide de la fonction fluidRow( ), des boîtes (ang. boxes) et du package DT, qui améliore l'affichage des tables.

- L'interface qui sera obtenue est la suivante :

Tableaux de bord



Tableaux de bord

- Le code correspondant est le suivant :

Tableaux de bord

server.R

```
library(ggplot2movies)

library(tidyverse)

library(scales)

library(rlang)

library(DT)
```

Tableaux de bord

```
function(input, output, session) {


  moviesSubset = reactive({

    movies %>% filter(year %in% seq(input$year[1], input$year[2]),

                      UQ(sym(input$genre)) == 1)


})
```

Tableaux de bord

```
output$budgetYear = renderPlot({


    budgetByYear = summarise(group_by(moviesSubset(), year),
                             m = mean(budget, na.rm = TRUE))

  ggplot(budgetByYear[complete.cases(budgetByYear), ],
         aes(x = year, y = m)) +
    geom_line() +
    scale_y_continuous(labels = scales::comma) +
    geom_smooth(method = "loess") +
    ggtitle(input$title)


})
```

Tableaux de bord

```
output$budgetYearLinear = renderPlot({

    budgetByYear = summarise(group_by(moviesSubset(), year),
                             m = mean(budget, na.rm = TRUE))
  ggplot(budgetByYear[complete.cases(budgetByYear), ],
         aes(x = year, y = m)) +
    geom_line() +
    scale_y_continuous(labels = scales::comma) +
    geom_smooth(method = "lm") +
    ggtitle(input$title)

})
```

Tableaux de bord

```
output$listMovies = renderUI({

    selectInput("pickMovie", "Pick a movie",
                choices = moviesSubset() %>%
                  sample_n(10) %>%
                  select(title)
    )
})
```

Tableaux de bord

```
output$moviePicker = renderDataTable({

    filter(moviesSubset(), title == input$pickMovie)[, 1:6]
  }, extensions = "Responsive")

}
```

Tableaux de bord

ui.R

```
library(shinydashboard)

library(DT)

header = dashboardHeader(title = "Movies explorer")


sidebar = dashboardSidebar(
  textInput("title", "Title"),
  selectInput("genre", "Which genre?",
              c("Action", "Animation", "Comedy", "Drama",
                "Documentary", "Romance", "Short"))
)
```

Tableaux de bord

```
body = dashboardBody(
  fluidRow(
    box(
      title = "Table", status = "success", width = 8, solidHeader = TRUE,
      DT::dataTableOutput("moviePicker")
    ),
    box(
      title = "More info", background = "olive", width = 4, collapsible = TRUE,
      uiOutput("listMovies")
    )
  ),
```

Tableaux de bord

```
fluidRow(
    tabBox(width = 5, title = "Graphs",
      tabPanel("LOESS", plotOutput("budgetYear")),
      tabPanel("Linear", plotOutput("budgetYearLinear"))
),
box(
  title = "Title 2", width = 4,
  sliderInput("year", "Year", min = 1893, max = 2005,
              value = c(1945, 2005), sep = "")
),
```

Tableaux de bord

```
box(
      width = 3, status = "info",
      checkboxGroupInput("checkbox", "Choose your...",
                         choices = c("Own adventure",
                                     "Weapon",
                                     "Next words wisely"))
    )
  )
)

dashboardPage(header, sidebar, body)
```

Tableaux de bord

- Le package DT est chargé à la fois dans le programme ui.R et dans le programme server.R.

- C’est un package qui utilise javaScript pour améliorer l’affichage des tables.

- Au sein du programme server.R, dans la fonction renderDataTable, on a spécifié extension = « responsive » pour que l’affichage s’adapte à l’écran.

Tableaux de bord

- Dans le fichier ui.R, on utilise des boîtes (ang. boxes) pour améliorer l'affichage.

- Parmi leurs paramètre, l'un des plus utiles est status, qui définit leur couleur : Info correspond à bleu, success à vert. Pour une liste complète, faire ?validStatuses à la console.

Tableaux de bord

- L’option solidHeader = TRUE améliore l’affichage du titre de la boîte.

- On peut également sélectionner une couleur d’arrière-fond. Pour plus d’information, faire ?validColors dans la console.

- On peut également spécifier qu’une boîte est réductible an utilisant l’option collapsible = TRUE.

Tableaux de bord

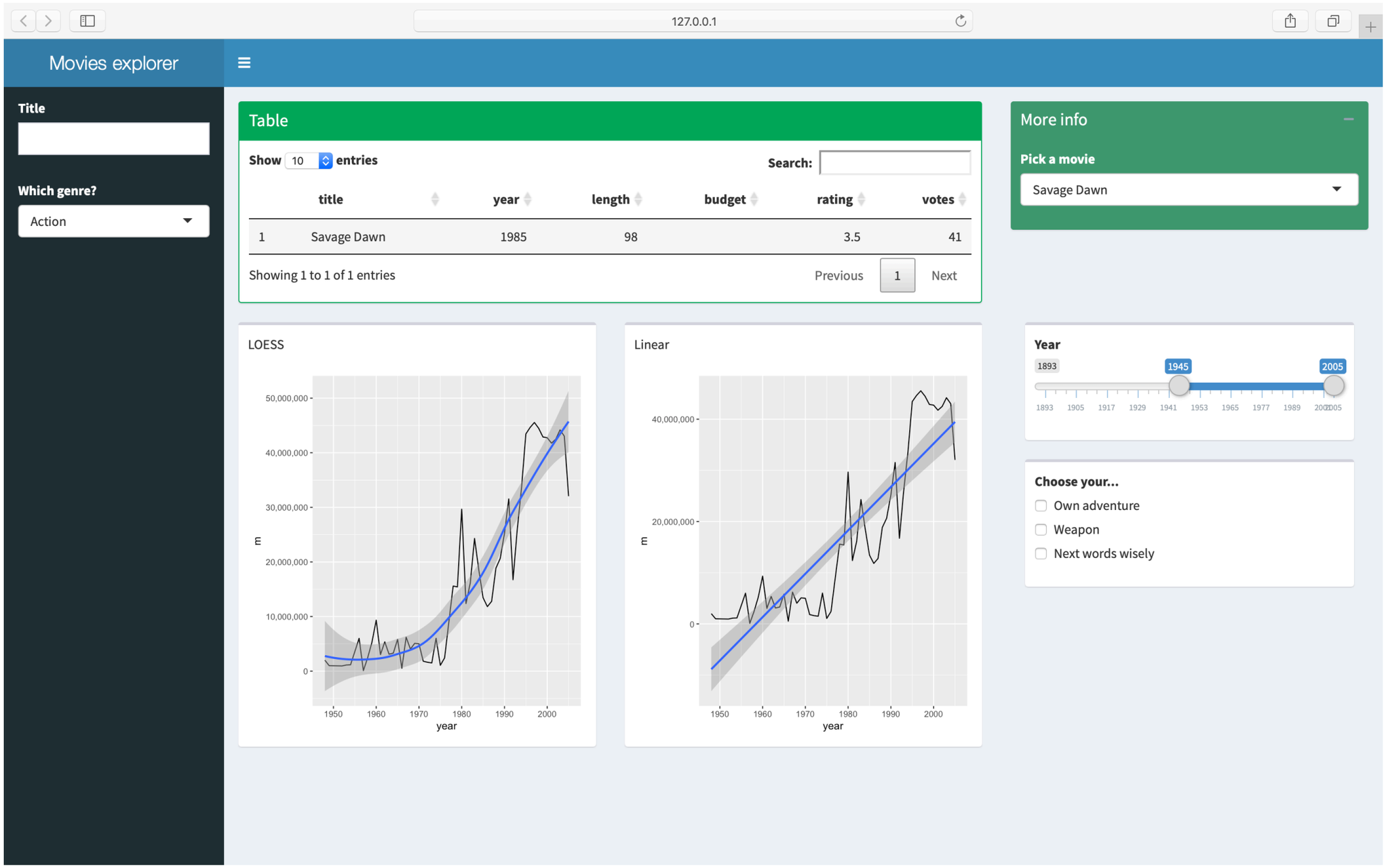
- Enfin, on remarque que la boîte d'affichage des graphiques comprend deux onglets.

- Pour celà, on utilise la fonction tabBox.

- L'affichage des deux graphes simultanément peut être fait à l'aide de la fonction column dans fluidRow.

- Le résultat est le suivant (code disponible sur le forum) :

Tableaux de bord

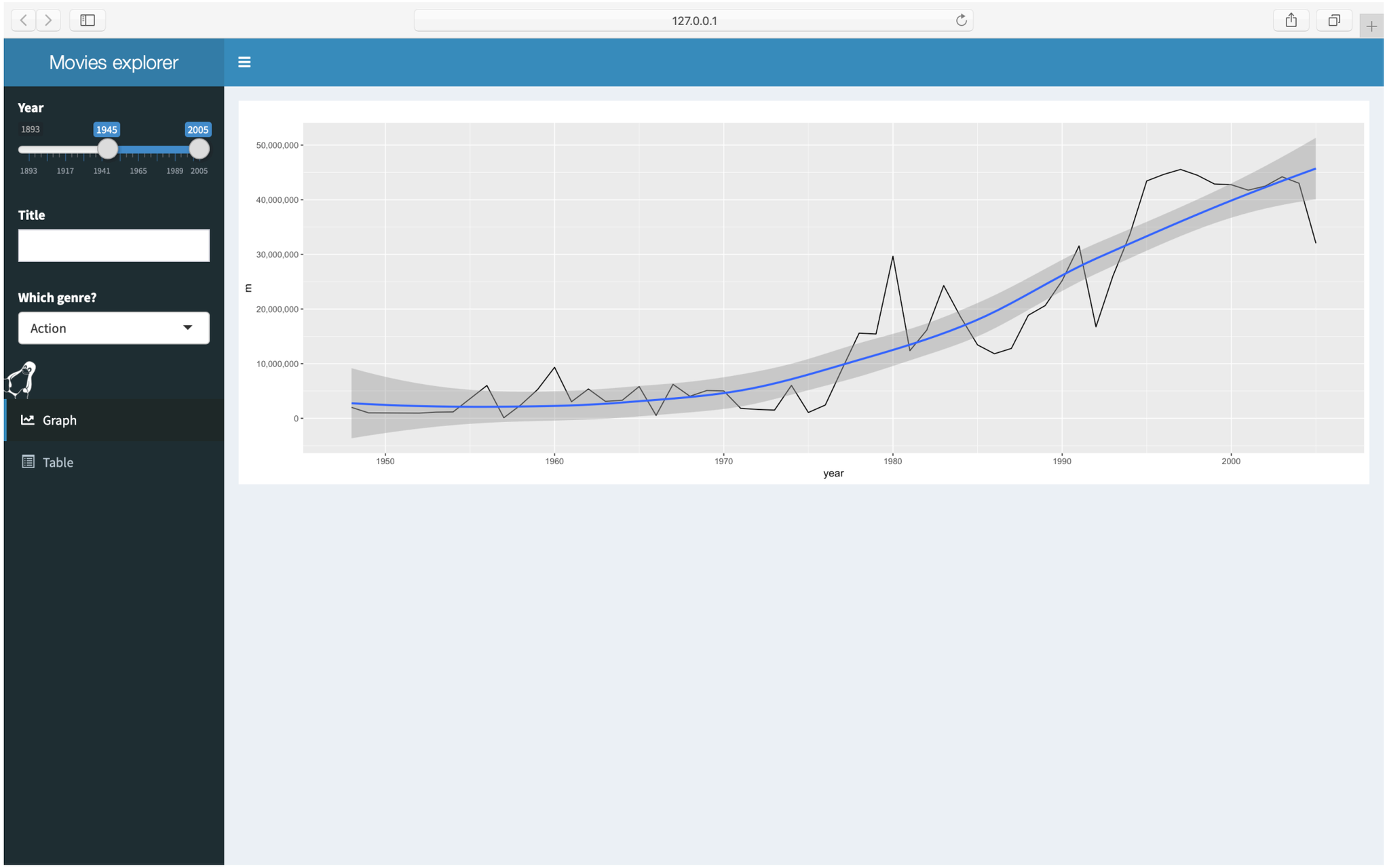


Icônes

- On va voir à présent comment rajouter des icônes à un tableau de bord ou à toute application Shiny.

- L’interface qu’on obtiendra est la suivante (répertoire Partie C, sous-répertoire main) :

Icônes


127.0.0.1
Movies explorer
Year
1893
1945
2005
1893
1917
1941
1965
1989
2005
Title
Which genre?
Action
Graph
Table
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
0
m
1950
1960
1970
1980
1990
2000
year

Icônes

- L’application contient trois icônes.

- L’icône « graph » est issue de la librairie d’icônes « Font Awesome », les icônes « table » et « linux » de la librairie « glyphicon ».

- Ce sont les deux librairies accessibles à partir de Shiny.

- Les sites correspondants sont les suivants :

https://fontawesome.com/icons?from=io

https://getbootstrap.com/docs/3.3/components/

Icônes

- L’utilisation des icônes est très simple.
- Pour accéder à la librairie Font Awesome, il suffit d’utiliser la commande icon avec le nom de l’icône souhaitée.
- Pour accéder à gliphycon, il suffit d’utiliser la commande icon avec l’argument lib = glyphicon.

Icônes

- Le formatage des icônes est également simple.
- Pour Font Awesome, la taille peut être spécifiée en utilisant l'argument class de la commande icon : fa-lg divise la taille par trois, fa-2x la multiplie par deux, fa-3x par trois.
- Rajouter une rotation (comme pour l'icône « linux ») se fait en rajoutant fa-spin dans l'argument class de la commande icon.

Icônes

- Exemple pour l'icône « linux » (penguin) : la commande correspondante est

```
icon(« linux », class = « fa-spin fa-3x »)
```

- On peut également associer des icônes à des onglets.

- Exemple (répertoire Partie C, sous-répertoire sidebarLayout) :

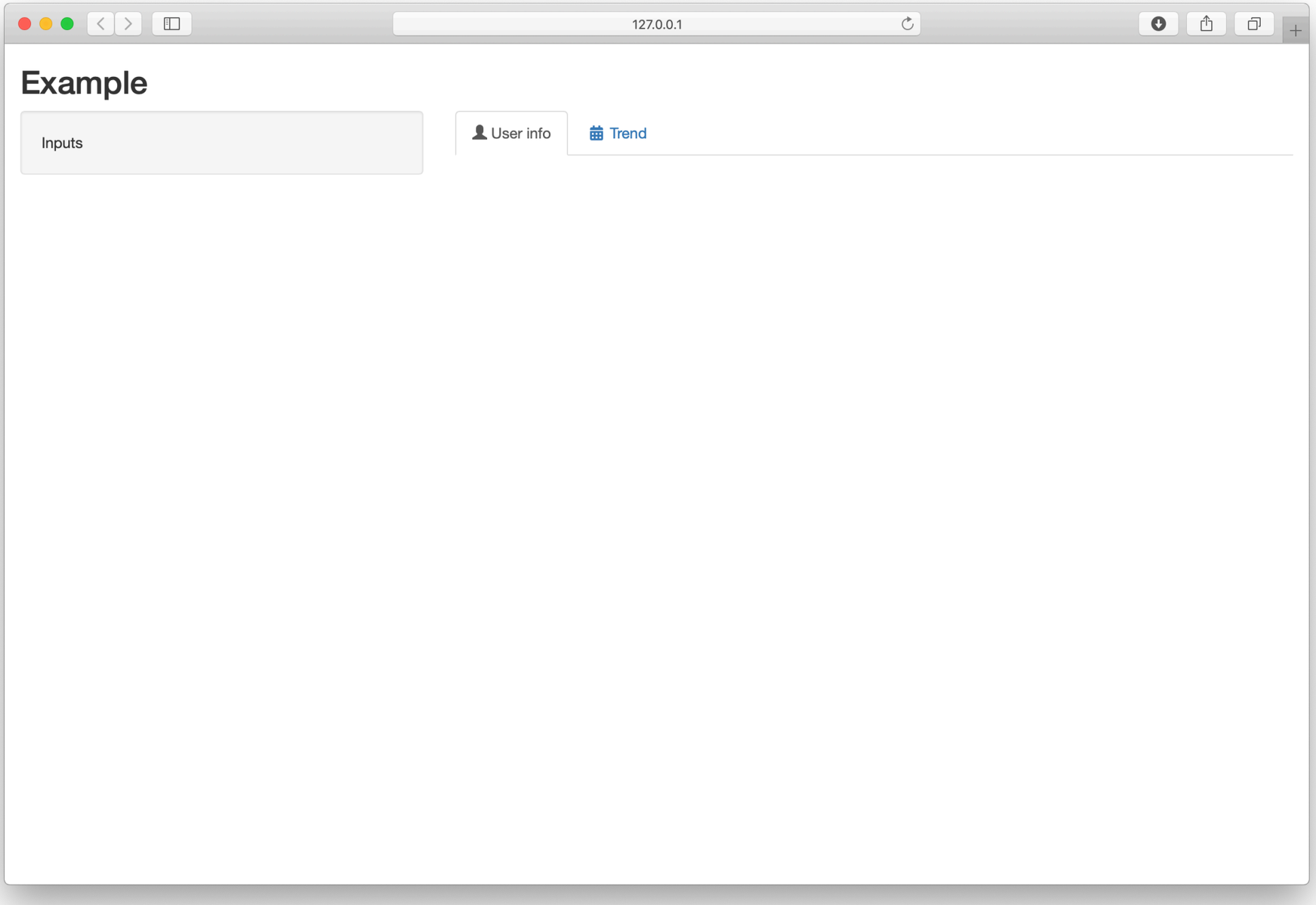
Icônes

```
tabPanel(« User info », icon = icon(« user »,
lib = « glyphicon »)

tabPanel(« Trend », icon = icon(« calendar »))
```

- Résultat :

Icônes



Compléments

- Il y a également possibilité, sous Shiny, de rajouter des notifications, des messages, ainsi que des graphiques Google (ang. Google Visualizations).

- Ce sont des notions plus avancées. Des exemples de programmes sont disponibles sur Internet. Voir : https://rstudio.github.io/shinydashboard/structure.html et le package googleVis.

Références

- *Hands-On Dashboard Development with Shiny*, C. Beeley, Packt 2018.

- *Web Application Development with R using Shiny*, 2nd Edition, C. Beeley, Packt 2016.

- *Learning Shiny*, H. G. Resnizky, Packt 2015.