A web browser is like a shadow puppet theater

With behind the scene the puppet master
Using a headless browser

Be the stage director...

... and fully decide what should be done...

... but in the dark! 🙈

No visual interface to see the result of your actions
Chrome Headless mode and the Devtools protocol

**Full control of Chrome** using Node.js modules like puppeteer, chrome-remote-interface

https://chromedevtools.github.io/devtools-protocol/

Interact with the protocol using json messages exchange through **websockets**.
You may already know other R related work

- **RSelenium** (@ropensci) client for Selenium WebDriver, requires a Selenium server (Java).

- **webshot** (@wch), **webdriver** (@rstudio) relies on the abandoned PhantomJS library.

- **htmlunit** (@hrbrmstr) uses the **HtmlUnit Java library**.

- **splashr** (@hrbrmstr) works with the **Splash JavaScript Rendering Service**

- **decapitated** (@hrbrmstr) uses headless Chrome command-line instructions or the **Node.js gepetto module** (built-on top of the puppeteer Node.js module)

- **chradle** (@MilesMcBain), first tests for driving Chromium/Chrome from R using a websocket connection. An inspiration for crrri
What is different with the crrri package?

https://github.com/RLesur/crrri/

Have the full control of from without Java, NodeJS or any server

Low-level API inspired by the chrome-remote-interface JS module gives access to 500+ functions to control Chrome

Dedicated to advanced uses / R packages developers

Also compatible with Opera, EdgeHtml and Safari

Only on github for now: remotes::install_github("rlesur/crrri")
What is different with the crrri package?

https://github.com/RLesur/crrri/

Have the full control of 🛡️ from 🏆 without Java, NodeJS or any server

Low-level API inspired by the chrome-remote-interface JS module gives access to 500+ functions to control Chrome

Dedicated to advanced uses / R packages developers

Also compatible with Opera, EdgeHtml and Safari

Only on github for now: remotes::install_github("rlesur/crrri")
How to interact from R with Chrome?

Headless Chrome can be controlled using the **Chrome DevTools Protocol (CDP)**

1. Launch Chrome in headless mode
2. Connect R to Chrome through websockets
3. Build an **asynchronous** function that
   - sends CDP commands to Chrome
   - listens to CDP events from Chrome
4. Execute this async flow with R

The goal of `{crrri}` is to **ease these steps**.

**Requirement:** You need to install chrome or **chromium**
library(crrri)
# launch headless Chrome
chrome <- Chrome$new()
# Inspect Headless chrome inside Rstudio viewer
client <- chrome$connect(callback = ~.x$inspect())
First step: Go to a page

Use a domain and its commands or event listeners

```r
# extract a domain from the protocol to work with
Page <- client$Page
# Send the 'Page.navigate' command from the protocol
Page$navigate(url = "http://user2019.r-project.org/")
```
One example: Web Scraping

Using promises package to build asynchronous function to perform with chrome

An API close to Javascript

```r
# Build an asynchronous flow - the puppet library (crrri)
dump_DOM <- function(client) {
  Page <- client$Page
  Runtime <- client$Runtime
  Page$enable() %>% {
    Page$navigate(
      url = 'http://user2019.r-project.org/talk_schedule/
    )} %>% {
      Page$loadEventFired()
    } %>% wait(3) %>% {
      Runtime$evaluate(
        expression = 'document.documentElement.outerHTML'
      )
    } %>% {
      writeLines(.result$value, "users2019-talks.html")
    }
  }
}

# and execute it using chrome - be the puppet master
perform_with_chrome(dump_DOM)
```
Using promises package to build asynchronous function to perform with chrome

An API close to Javascript

```r
# Build an asynchronous flow - the puppet library(crrri)
dump_DOM <- function(client) {
  Page <- client$Page
  Runtime <- client$Runtime
  Page$enable() %>% {
    Page$navigate(
      url = 'http://user2019.r-project.org/talk_schedule/'
    ) %>% {
      Page$loadEventFired()
    } %>% wait(3) %>% {
      Runtime$evaluate(
        expression = 'document.documentElement.outerHTML'
      )
    } %>% {
      writeLines(.result$value, "users2019-talks.html")
    }
  }
}

# and execute it using chrome - be the puppet master
perform_with_chrome(dump_DOM)
```

One example: Web Scraping
What is also possible?

Print PDF

```r
print_pdf <- function(client) {
  Page <- client$Page
  Page$enable() %>% {
    Page$navigate(
      url = "https://r-project.org/"
    )
    # await the load event
    Page$loadEventFired()
  } %>% {
    Page$printToPDF()
  } %>% # await PDF reception
    write_base64("r_project.pdf")
} %>% # To modify depending on the page
  # content (JS libraries...)
  perform_with_chrome(print_pdf)
```

Screenshot and Device emulation

```r
Emulation$setDeviceMetricsOverride(
  width = 375, height = 667,
  mobile = TRUE,
  deviceScaleFactor = 2
)
```

Screencast

- `Page$screencastFrame`
- `Page$startScreencast`
- `Page$stopScreencast`

Exemple on YouTube
Questions?

See also uRos2019 talk by R.Lesur

We welcome feedbacks, issues and ideas!
Tell us how you would use crrri

https://rlesur.github.io/crrri/

@cderv
@chrisderv
@RLesur
@RLesur
Questions?

See also uRos2019 talk by R.Lesur

We welcome feedbacks, issues and ideas!
Tell us how you would use crrri

https://rlesur.github.io/crrri/

@cderv  @chrisderv  @RLesur  @RLesur