

# TP2: Testing Automático

## Responsive Design



Tecnología: Galen Framework

Curdi Francisco  
Dellanzo Antonella  
Martinez Federico  
Rossi Lucas

# Galen

- *Framework* utilizado para generar *layout* y *functional testing* de aplicaciones web.
- Originalmente pensado para testear el *look & feel* de páginas web *responsives*.
- Actualmente contiene herramientas para hacer tests funcionales.
- Utiliza Selenium para manejar las interacciones con el browser.
- Permite hacer tests en Java, Javascript y Galen Specs, un lenguaje que provee el mismo *framework*.



# Galen

- Permite ejecutar múltiples tests en paralelo.
- Permite generar reglas y/o parametrizar información para utilizarla en varios tests distintos y evitar repetir código.
- Dado que utiliza Selenium de fondo, permite realizar interacciones con los distintos elementos de la página y navegar por el sitio mientras testeamos el layout.
- Utiliza selectores css para identificar los elementos del layout que queremos chequear.
- Es un proyecto open source.



# ¿Qué nos permite testear?

- Ubicación, dimensión y cantidad de elementos.
- Si el elemento es visible o está ausente.
- Si el elemento está dentro de otro o cerca de otro elemento, y a qué distancia.
- Si el elemento está por encima, por debajo, a la izquierda o derecha de otro.
- Cantidad de elementos de un determinado tipo.
- Distribución de colores en un área (por medio de un screenshot).
- Imágenes y la distribución de sus colores.
- Testing de compatibilidad cross-browser.



# Layout Testing

- Los tests de layout respetan la siguiente operatoria:
  - Mediante Selenium, se abre la página a testear en el browser que se haya configurado (firefox, chrome, internet explorer, edge o phantomjs).
  - Se redimensiona la ventana al tamaño especificado en el test (ej: 800x600). Es importante destacar que el tamaño incluye el espacio ocupado por la barra de navegación, marcadores, etc. Debido a esto, puede variar dependiendo del SO en el que se corra.
  - Se ejecutan los tests del archivo \*.gspec especificado con las distintas aserciones sobre los elementos de la página.



# Reportes

- Galen genera automáticamente algunos reportes durante y después de la ejecución de los tests:
  - HTML Reports
  - Screenshots
  - Image comparison
- Estos reportes permiten visualizar fácilmente el resultado de los tests y en especial las causas de los errores, en caso de haberlos.



# HTML Reports

## Galen Test Report

Tests								
Groups								
Test	Passed	Failed	Warnings	Total	Groups	Started	Duration	
<a href="#">Chat page on C-Mobile device</a>	10	3	0	13		08-06-2018 22:42:15	10s	<div><div></div></div>
<a href="#">Chat page on C-Mobile-L device</a>	11	2	0	13		08-06-2018 22:42:25	11s	<div><div></div></div>
<a href="#">Chat page on Desktop device</a>	12	1	0	13		08-06-2018 22:42:48	11s	<div><div></div></div>
<a href="#">Chat page on Desktop-FHD device</a>	12	1	0	13		08-06-2018 22:43:10	10s	<div><div></div></div>
<a href="#">Chat page on Desktop-HD device</a>	12	1	0	13		08-06-2018 22:42:59	10s	<div><div></div></div>
<a href="#">Chat page on Tablet device</a>	12	1	0	13		08-06-2018 22:42:37	10s	<div><div></div></div>
<a href="#">Home page on C-Mobile device</a>	3	0	0	3		08-06-2018 22:41:39	5s	<div><div></div></div>
<a href="#">Home page on C-Mobile-L device</a>	3	0	0	3		08-06-2018 22:41:45	5s	<div><div></div></div>
<a href="#">Home page on Desktop device</a>	3	0	0	3		08-06-2018 22:41:56	5s	<div><div></div></div>
<a href="#">Home page on Desktop-FHD device</a>	3	0	0	3		08-06-2018 22:42:09	6s	<div><div></div></div>
<a href="#">Home page on Desktop-HD device</a>	3	0	0	3		08-06-2018 22:42:02	7s	<div><div></div></div>
<a href="#">Home page on Tablet device</a>	3	0	0	3		08-06-2018 22:41:50	6s	<div><div></div></div>
<a href="#">Login page on C-Mobile device</a>	9	0	0	9		08-06-2018 22:41:03	6s	<div><div></div></div>

- Luego de ejecutar los tests, se genera un reporte en HTML con la información de todos los tests que se ejecutaron, que incluye cuáles fueron los tests que pasaron, cuáles fallaron y cuánto tiempo tardaron en correr.

# Screenshots

inside: welcome-block ~30px left  
height: > 20px  
above: login-button 10 to 50 px

## login-button

inside: welcome-block ~30px left  
height: ~ 45px  
text is: Login  
above: text-block-3 10 to 50px

## text-block-3

inside: welcome-block ~30px left

## greeting

above: text-block-1 10 to 50 px  
 inside: welcome-block ~ 50px left  
- "greeting" is 30px left which is

## greeting

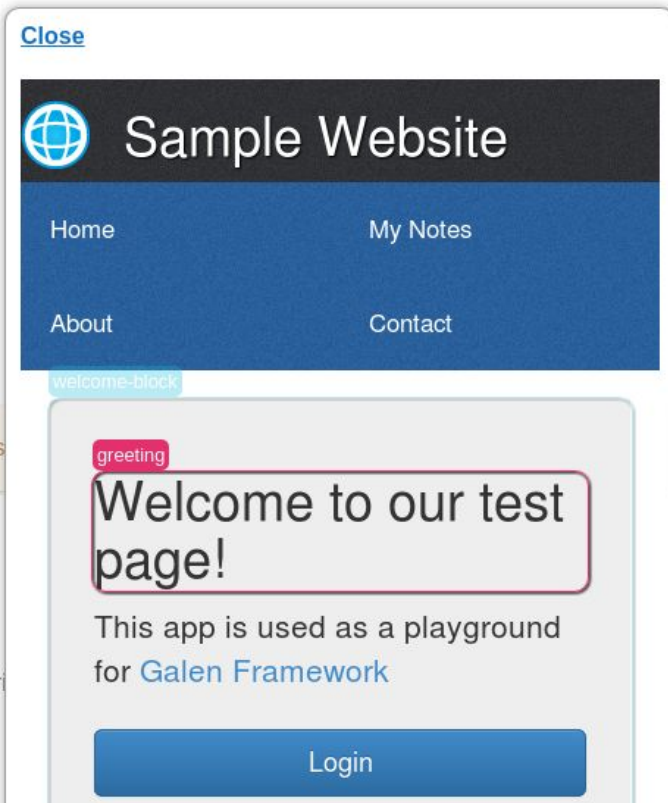
height: ~ 78px  
inside: welcome-block ~ 50 px top

## login-button

inside: welcome-block ~ 30px left

## Footer

o Screenshot



Al acceder a un test en particular, se muestran los resultados del mismo.

En caso de falla, se detalla el error ocurrido y se puede acceder para visualizar una screenshot, en la cual se encuentra resaltado el error.



# Image comparison

## Welcome page on tablet device

11:34:00  **ERROR** Check layout: specs/welcomePage.spec included tags: ta



### Header

#### header


inside: screen 0px top  
centered horizontally inside: screen 1px

 **height: ~ 70px**

- "header" height is 109px which is not in range

#### header-logo

inside: header 5 to 15px top, 0 to 10px left  
near: header-text 5 to 30px left

 **image: file header-logo.png, error 10px** [Show diff](#)

- Element does not look like "specs/header-logo.png". There are 335 mismatching pixels but max allowed is 10

[Close](#)

☐ Put images on top of each other

#### Actual Image



#### Expected



#### Comparison Map



- Cuenta también con una herramienta de *image comparison* para observar la diferencia entre la imagen esperada y la obtenida.

# Galen Specs

- Galen define su propio lenguaje, *Galen Specs*, para describir cómo debe verse la página web en distintos tamaños del browser.
- Está pensado para que sea fácil de escribir y a su vez fácil de leer, ya que el enfoque está en que los tests sean una especie de especificación que se puede leer como prosa. Tiene un alto nivel expresivo.
- Los tests de una página cuentan esencialmente con dos partes:
  - Object definitions.
  - Object specs.



# Object definitions

- Define los elementos de la página que son de interés para el test. Sobre estos elementos vamos a realizar las distintas aserciones.
- Se utiliza **@objects** para indicar que comienza la definición de los mismos.
- Provee tres formas de establecer el mapeo entre estos objetos y los elementos en el html:
  - **Id**: busca el elemento por id en el DOM.
  - **css**: busca el elemento en el dom a partir de un selector css.
  - **xpath**: soporte para expresiones XPath.



# Object definitions

```
<body>
  <div id='search-bar'>
    <input type='text' name='search' value='' />
    <a href='#' class='search-button'>Search</a>
  </div>
</body>
```

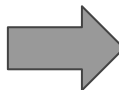


```
@objects
search_panel      id      search-bar
search_panel_input  xpath  //div[@id='search-bar']/input[@type='text']
search_panel_button css   #search-bar a
```

- Se puede omitir el indicador de qué tipo de selector es (columna del medio).

# Object definitions

```
<ul id='menu'>
  <li><a href='#'>Home</a></li>
  <li><a href='#'>Blog</a></li>
  <li><a href='#'>Categories</a></li>
  <li><a href='#'>About</a></li>
</ul>
```



```
@objects
menu_item-*      css      #menu li a
```

- Galen permite realizar definiciones para objetos que se repiten y luego los numera. En este caso, a cada **li** se le asigna un número como menu\_item-1, menu\_item-2, menu\_item-3 y menu\_item-4.

# Ranges

- Mediante el uso de rangos, se puede definir que el valor esperado sea exactamente igual, menor, mayor, entre otros. Por ejemplo:

```
# Exact range  
width 100px
```

```
# The value between range  
width 50 to 200 px
```

- Se puede definir que el valor esperado sea relativo al *screen* (área de la página del browser, incluso la no visible), al *viewport* (área visible de la página) o a objetos que puedan ser referenciados mediante id, css y xpath. Estos valores pueden ser combinados con los rangos.

```
width ~ 95 % of screen/width  
height > 40 % of screen/height  
width 30 to 100 % of screen/width  
width 50 % of screen/width
```

# Variables

- Galen permite definir variables de manera de poder utilizarlas en los distintos tests que hagamos sobre los elementos sin necesidad de andar repitiendo el mismo valor una y otra vez.
- Para definir variables se utiliza la nomenclatura **@set**. Para utilizarlas, **\${variable}**.

```
@set
  commonHeaderMargin  10 to 20px
  contentMargin ~ 20px

= Header =
  header_icon:
    inside header ${commonHeaderMargin} top left

= Content =
  article-description:
    inside main ${contentMargin} left right
```

# Tagging and sections

- Se pueden definir secciones para estructurar mejor el código.
  - Esto se hace colocando el símbolo = al comienzo y fin de la línea.
  - Se pueden tener secciones encadenadas.
- Se usan además *tags* para manejar las distintas condiciones necesarias para el *layout testing*, como lo son los distintos dispositivos:
  - Se define por un lado el *tag* junto con su especificación.
  - Luego, se invoca con la sentencia **@on** seguido del *tag*.

```
= Header section =  
  = Icons and text =  
    header.icon:  
      inside header 10px top left  
  
    header.caption:  
      text is "Greetings!"
```

```
@on *  
  menu:  
    height 70px
```

**@on \***  
Aplica para todos los  
*tags*

```
@on mobile, desktop  
  menu:  
    height 300 px
```

Aplica para  
múltiples *tags*  
(mobile y desktop)

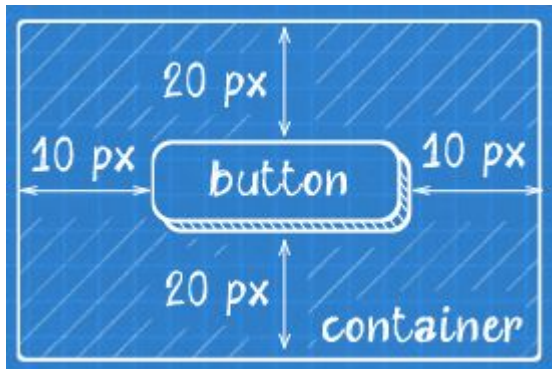
```
@on mobile  
  login-button:  
    width 100px
```

Aplica para el *tag*  
mobile



# Object specs

- Galen provee distintos tipos de specs que se pueden utilizar para realizar las aserciones sobre los objetos. Algunos de ellos son:
  - **Below & Above:** verifican que un elemento esté abajo o arriba de otro objeto.
  - **Inside:** verifican que un objeto esté dentro total o parcialmente de otro.
  - **Absent:** verifican que un elemento no esté en la página o no esté visible.
  - **Color-scheme:** verifican la distribución de colores en el área del objeto.
  - **Width & Height:** verifican el ancho y alto del objeto.



```
button:  
  inside container 10px left right, 20px top bottom
```

# Custom rules

- Galen nos permite definir reglas parametrizadas, las cuales permiten un mayor nivel de expresión de lo que se está testeando.
- Se definen con la nomenclatura @rule.
- Permiten aplicar distintos parámetros según desde donde se llamen.

```
@rule elemento esta centrado en relacion a %{element}  
  aligned vertically centered ${element}  
  
= Login =  
  login:  
    contains loginTitle, loginField, loginButton  
  
    @on desktop, desktop-hd, desktop-fhd, tablet  
      height ${loginMaxHeight} px  
      width ${loginMaxWidth} px  
    @on c-mobile  
      height ${loginMaxHeight} px  
      width 320 px  
    @on c-mobile-l  
      height 320 px  
      width ${loginMaxWidth} px  
  
  loginTitle:  
    | elemento esta centrado en relacion a login  
  
  loginButton:  
    text is "Ingresar"  
    | elemento esta centrado en relacion a login  
    visible  
  
  loginField:  
    | elemento esta centrado en relacion a login  
    visible
```

# Galen Test Suite Syntax

- Galen define un formato para especificar las Test Suites.
- Brinda gran flexibilidad a la hora de parametrizar las corridas de las suites con distintos dispositivos y tamaños.
- La estructura que propone es:

```
Home page on a small mobile device
  http://example.com/home      320x600
    check homepage.gspec --include "mobile,all" --exclude "nomobile"
```

# Galen Test Suite Syntax

```
@@ table devices
| deviceName | tags | size |
| C-Mobile | c-mobile | 320x600 |
| C-Mobile-L | c-mobile-l | 600x394 |
| Tablet | tablet | 800x600 |
| Desktop | desktop | 1024x800 |
| Desktop-HD | desktop-hd | 1366x768 |
| Desktop-FHD | desktop-fhd | 1920x1080 |

@@ parameterized using devices
Login page on ${deviceName} device
http://localhost:3000/ ${size}
  check galen_test/login.gspec --include "${tags}"

@@ parameterized using devices
Home page on ${deviceName} device
http://localhost:3000/ ${size}
  run galen_test/login.js '{username: "testname"}'
  check galen_test/home.gspec --include "${tags}"

@@ parameterized using devices
Chat page on ${deviceName} device
http://localhost:3000/ ${size}
  run galen_test/chat.js '{username: "testname", groupname: "testgroup"}'
  check galen_test/chat.gspec --include "${tags}"
```

- Utilizamos **@@ table devices** para especificar los dispositivos y tamaños en los que vamos a correr nuestras test suites.
- **@@ parameterized using devices** nos permite iterar sobre los dispositivos definidos y correr las suites con cada uno.
- El **run {script} {parametros}** nos permite ejecutar un archivo javascript antes de hacer el chequeo de los specs de Galen. Más de esto en próxima diapo.

# Page Actions

- Las *page actions* son acciones que podemos realizar antes de que Galen proceda a ejecutar los tests sobre la página del browser abierta.
- Se definen 3 tipos distintos:
  - **Cookie Handling:** nos permite inyectar cookies que dentro de la página utilizamos para mostrar o esconder contenido.
  - **Javascript Injection:** nos permite inyectar un archivo javascript que realiza cambios sobre el html de la página.
  - **Selenium Interaction:** es similar al anterior pero con más funcionalidad ya que nos permite utilizar selenium para completar campos y realizar interacciones como loguearnos al sistema.



# Selenium Interaction

- En nuestros tests utilizamos *Selenium Interaction* ya que nos permitió poder realizar el logueo a la app de chat y navegar por la misma para poder testear cada una de las secciones de la app.
- De fondo utiliza WebDriver.js para poder seleccionar campos de texto y llenarlos, o para presionar botones.

```
var username = arg.username;

driver.findElement(By.cssSelector("#login-box #user")).sendKeys(username);
driver.findElement(By.cssSelector("#login-box #submit")).click();

function pageIsLoaded() {
    return driver.findElement(By.id("navbar")) != null;
}

function waitFor(func) {
    var timeout = 10;

    while (timeout > 0 && !func()) {
        timeout = timeout - 1;
        Thread.sleep(1000);
    }

    if (!func()) {
        throw new Error("Wait timeout");
    }
}

waitFor(pageIsLoaded);
```

# IDE Support

- Hay plugins de *syntax highlighting* para los editores mas populares:
  - **Sublime Text**
  - **Vim**
  - **Visual Studio Code**
  - **Atom**
- Existe un plugin para **IntelliJ**, pero no parece andar del todo bien y no tiene mucho soporte.



# Integraciones

- Galen se puede integrar con distintas tecnologías que permiten automatizar el proceso de *testing*:
  - **Jenkins**
    - Automatizar las corridas de tests en pull requests, durante los builds y demas.
  - **Selenium Grid**
    - Distribución de tests en distintas máquinas, pudiendo así correr las suites de test en paralelo.
    - Manejar distintos browsers y sistemas operativos.
  - **BrowserStack / Sauce Labs**
    - Configurar los tests para ejecutarlos en la nube pudiendo así testear en distintos emuladores y también dispositivos reales.





# Alternativas tecnológicas: Applifools

- Está diseñado para probar todos los elementos que aparecen en una pantalla.
- Ofrece:
  - **Cross Browser Testing:** soporta los principales navegadores y versiones de los mismos para hacer pruebas en paralelo.
  - **Cross Device Testing:** permite verificar la aplicación ya sea en celulares, tablets, notebooks o computadoras de escritorio.
  - **Responsive Design Testing:** evalúa que el diseño que se muestra por pantalla coincida con el tamaño de la misma.



Cross Browser  
Testing



Cross Device  
Testing



Responsive Design  
Testing

# Bibliografía

<http://galenframework.com/>

<http://mindengine.net/category/galen/>

<https://github.com/galenframework/galen>

<https://applitools.com/features/test-automation>

<https://github.com/davidrv87/syntax-sublime-galen2>

<https://github.com/galenframework/galen.vim>

<https://marketplace.visualstudio.com/items?itemName=simonhdickson.galen>

<https://atom.io/packages/language-galen-v2>

<https://plugins.jetbrains.com/plugin/8302-galen-specs-language-support>

