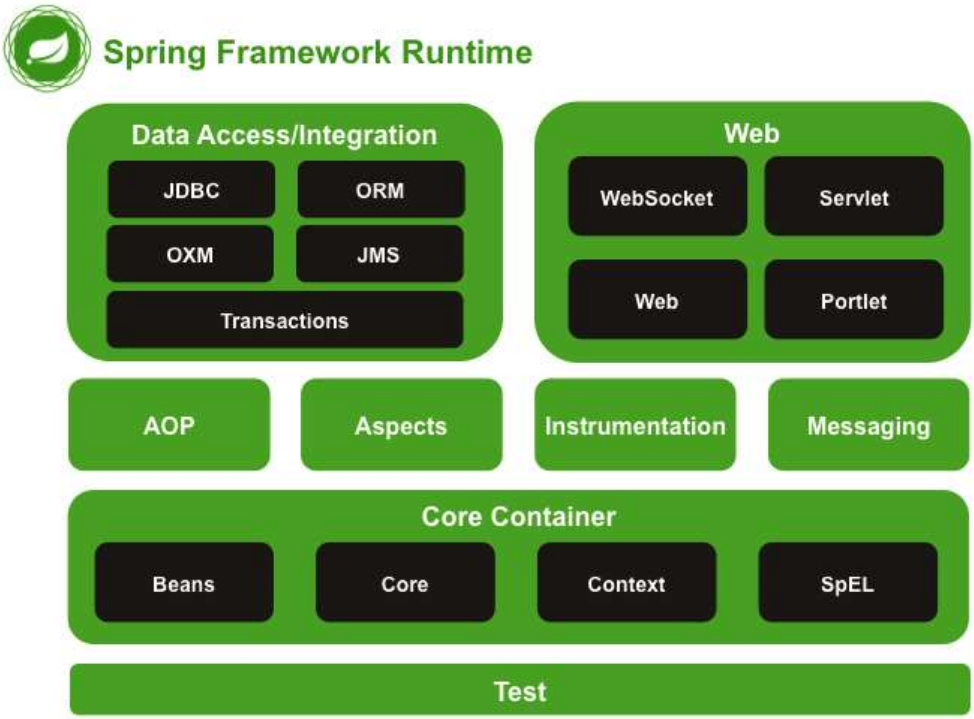
DESARROLLO WEB CON SPRING



UNIDAD 05
USO DE THYMELEAF

Eric Gustavo Coronel Castillo
gcoronelc.github.io
I N S T R U C T O R

CONTENIDO

¿QUÉ ES THYMELEAF?

Thymeleaf es un motor de procesamiento de plantillas muy flexible, adaptado a las necesidades actuales, poco intrusivo en el marcado y muy rápido.

El uso de Thymeleaf facilita el uso de recursos HTML y el amplio uso diferentes estilos. Permite trabajar con varios tipos de expresiones:

- Expresiones variables: Son quizás las más utilizadas, como por ejemplo ${…}
- Expresiones de selección: Son expresiones que nos permiten reducir la longitud de la expresión si prefijamos un objeto mediante una expresión variable, como por ejemplo *{…}
- Expresiones de mensaje: Que nos permiten, a partir de ficheros properties o ficheros de texto, cargar los mensajes e incluso realizar la internalización de nuestras aplicaciones, como por ejemplo #{…}
- Expresiones de enlace: Nos permiten crear URL que pueden tener parámetros o variables, como por ejemplo @{…}
- Expresiones de fragmentos: Nos van a permitir dividir nuestras plantillas en plantillas más pequeñas e ir cargándolas según las vayamos necesitando, como por ejemplo ~{…}

Por defecto, cuando se trabaja con Thymeleaf se suele hacer con el lenguaje de expresiones OGNL (Object-Graph Navigation Language), aunque cuando trabajamos conjuntamente con Spring MVC podemos utilizar el SpEL (Spring Expression Language).

PRIMEROS PASOS

Dependencia

```
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
```

Configuración página HTML

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Titulo de página</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
  . . .
  . . .
</body>
</html>
```

Expresiones variables

Algunos ejemplos de expresiones variables son:

`${titulo}`	Accediendo a una variable simple.
`${producto.precio}`	Utilizando notación de puntos para acceder a las propiedades de un objeto.
`${sesión.usuario.nombre}`	Accediendo a objetos de sesión.
`<h1 th:text="${titulo}"/>` `<p th:text="${venta.prod.nombre}"/>`	Uno de los atributos que puedes usar es **th:text** con diferentes etiquetas HTML, para poder mostrar, por ejemplo, el valor de una variable. También puedes navegar entre objetos.

Ejemplo 1

Mostrar un saludo.

Controlador

```
  @GetMapping({"/","/home","/index"})
  public String home(Model model) {
     model.addAttribute("titulo", "SALUDO");
     model.addAttribute("mensaje", "Bienvenido a Thymeleaf.");
     return "home";
  }
```

Página HTML

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title th:text="${titulo}"></title>
</head>
<body>
   <h1 th:text="${titulo}"></h1>
   <p th:text="${mensaje}"></p>
</body>
</html>
```

Expresiones de selección

Las expresiones de selección te permiten marcar un objeto y sobre el mismo evaluar algunas expresiones.

Por ejemplo, si quieres listar todas las propiedades de un libro, en lugar de usar expresiones variables, puedes prefijar el objeto `book` e ir después utilizando cada una de las propiedades del mismo de una manera más sencilla:

```
<div th:object=${book}”>
   . . .
   <span th:text=”*{title}”>…</span>
   . . .
</div>
```

En este caso, `title` es una propiedad del objeto `book`.

Ejemplo 2

Mostrar los datos de un producto.

Controlador

```
@GetMapping("/datos")
public String datos(Model model) {
   Producto producto = new Producto(1000, "Licuadora", 598.00, 768);
   model.addAttribute("titulo", "Datos del Producto");
   model.addAttribute("producto", producto);
   return "datos";
}
```

Reporte 1

```
<div>
   <p th:text="'ID: ' + ${producto.id}"></p>
   <p th:text="'NOMBRE: ' + ${producto.nombre}"></p>
   <p th:text="'PRECIO: ' + ${producto.precio}"></p>
   <p th:text="'STOCK: ' + ${producto.stock}"></p>
</div>
```

Reporte 2

```
<div th:object="${producto}">
   <p th:text="'ID: ' + *{id}"></p>
   <p th:text="'NOMBRE: ' + *{nombre}"></p>
   <p th:text="'PRECIO: ' + *{precio}"></p>
   <p th:text="'STOCK: ' + *{stock}"></p>
</div>
```

El resultado es ambos reportes es el mismo, pero, en el caso de Reporte 2, se está prefijando el objeto `producto` en el `div`, luego, se accede a cada uno de los atributos haciendo el código más sencillo.

Bucles

Con Thymeleaf puedes iterar de forma muy sencilla sobre una colección, puede ser una colección de datos simples i una colección de beans.

Por ejemplo, puedes pasar un parámetro llamado `productos`. El valor de este parámetro será una colección definida por un `ArrayList` de instancias de tipo `Producto.`

`Producto` es un bean y sus atributos principales son `id` y `nombre`. En este escenario, puedes iterar fácilmente sobre esta colección y mostrar todos sus valores como se ilustra a continuación:

```
. . .
<tr th:each="prod : ${productos}">
   <td th:text="${prod.id}">Código del producto</td>
   <td th:text="${prod.nobre}">Nombre del producto</td>
</tr>
. . .
```

Ejemplo 3

Mostrar un listado de productos.

Controlador

```
@GetMapping("/listado")
public String listado(Model model) {
   List<Producto> productos = new ArrayList<>();
   productos.add(new Producto(1001, "Producto A", 598.00, 768));
   productos.add(new Producto(1002, "Producto B", 783.00, 100));
   productos.add(new Producto(1003, "Producto C", 278.00, 350));
   productos.add(new Producto(1004, "Producto D", 723.00, 820));
   productos.add(new Producto(1005, "Producto E", 634.00, 649));
   model.addAttribute("titulo", "Listado de Productos");
   model.addAttribute("productos", productos);
   return "listado";
}
```

Caso 1

En este caso se muestra los productos en una lista:

```
<div>
   <ul>
      <li th:each="prod : ${productos}"
         th:text="${prod.id} + ' - ' + ${prod.nombre}"></li>
   </ul>
</div>
```

- 1001 - Producto A
- 1002 - Producto B
- 1003 - Producto C
- 1004 - Producto D
- 1005 - Producto E

Caso 2

En este caso se muestra los productos en una tabla:

```
<tr th:each="prod : ${productos}">
   <td th:text="${prod.id}"></td>
   <td th:text="${prod.nombre}"></td>
   <td th:text="${prod.precio}"></td>
   <td th:text="${prod.stock}"></td>
</tr>
```

ID	NOMBRE	PRECIO	STOCK
1001	Producto A	598.0	768
1002	Producto B	783.0	100
1003	Producto C	278.0	350
1004	Producto D	723.0	820
1005	Producto E	634.0	649

ATRIBUTOS CONDICIONALES

Atributo If – Unless

Los atributos `th:if` y `th:less` te permite presentar un elemento HTML dependiendo de una condición proporcionada:

Sintaxis

```
<someHtmlTag th:if="condition">

   <!-- Other code -->

</someHtmlTag>

<!-- OR: -->

<th:block th:if="condition">

   <!-- Other code -->

</th:block>
```

Otro atributo que puedes utilizar es `th:unless`. Es la opción negativa de `th:if`.

```
<someTag th:unless = "condition">

</someTag>

<!-- Same as -->

<someTag th:if = "!condition">

</someTag>
```

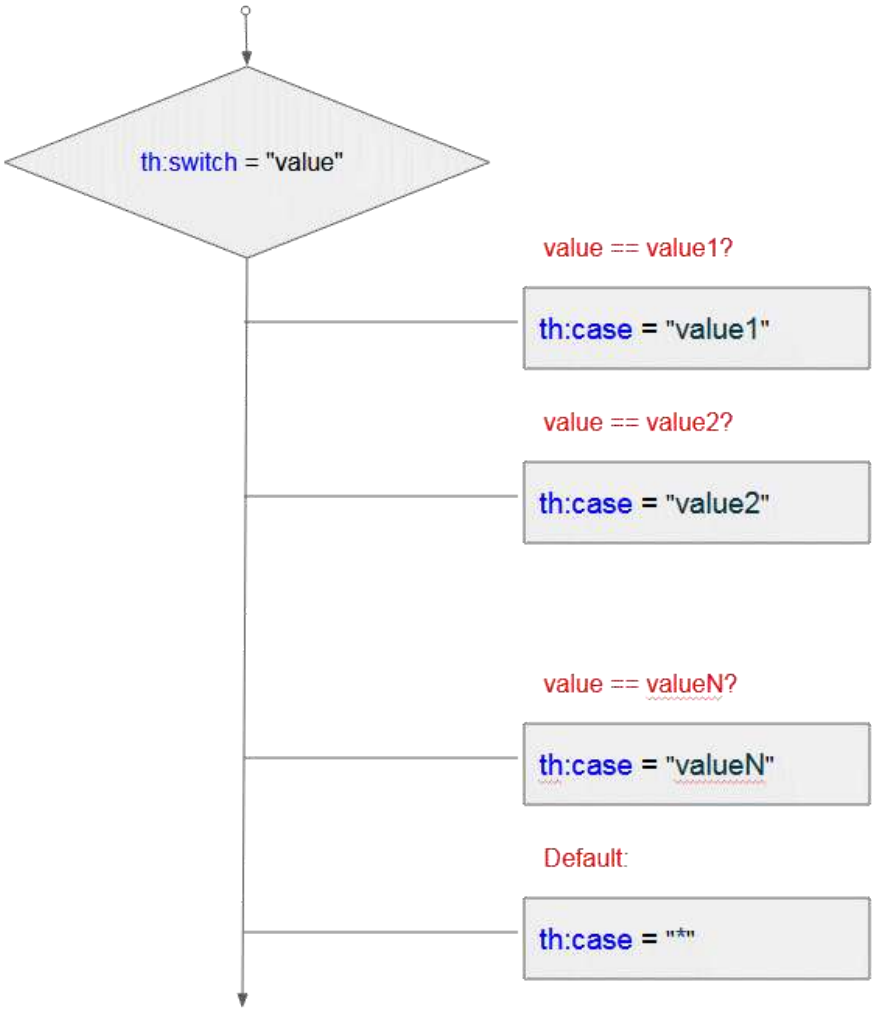
A continuación tienes un ejemplo ilustrativo de como se aplica estos atributos:

```
<td>
  <span th:if="${teacher.gender == 'F'}">Female</span>
  <span th:unless="${teacher.gender == 'F'}">Male</span>
</td>
```

Si el contenido de la variable `teacher.gender` es igual a una `F`, entonces se representa el elemento `span` con el valor `"Female"`. De lo contrario, se renderiza el elemento con `"Male"`. Esta configuración es comparable a una cláusula `if-else` presente en la mayoría de los lenguajes de programación.

Atributos th:switch, th:case

En Java, estás familiarizado con la estructura `switch/case`. Thymeleaf también tiene una estructura similar que es `th:swith/th:case`.



El programa evaluará los casos de arriba a abajo. Si un caso se evalúa como verdadero, presentará el código en este caso. Todos los demás casos serán ignorados.

El atributo `th:case="*"` es el caso predeterminado de la estructura `th:swith/th:case`. Si todos los casos anteriores se evalúan como falsos, el código del caso predeterminado es el que se presentará.

A continuación, tienes un ejemplo ilustrativo:

```
<div th:switch="${user.role}">
   <p th:case="'admin'">User is an administrator</p>
   <p th:case="${roles.manager}">User is a manager</p>
   <p th:case="'staff'">User is a staff</p>
</div>
```

El ejemplo a continuación utiliza la cláusula default:

```
<div th:switch="${user.role}">
   <p th:case="'admin'">User is an administrator</p>
   <p th:case="${roles.manager}">User is a manager</p>
   <p th:case="'staff'">User is a staff</p>
   <p th:case="*">User is some other thing</p>
</div>
```

Ejemplo 4

La clase User

```
public class User {

   private String name;
   private String role;

   public User() {
   }

   public User(String name, String role) {
      super();
      this.name = name;
      this.role = role;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getRole() {
      return role;
   }

   public void setRole(String role) {
      this.role = role;
   }

}
```

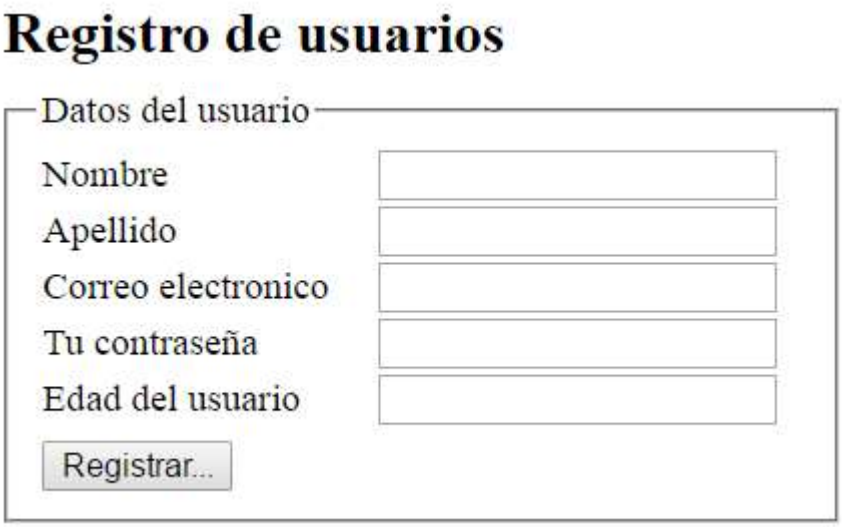
El controlador

```
  @GetMapping("/switch-example")
  public String swtichTest(Model model) {
     User user = new User("Gustavo Coronel", "admin");
     model.addAttribute("titulo", "ATRIBUTO SWITCH");
     model.addAttribute("user", user);
     return "switch-example";
  }
```

La vista

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title th:text="${titulo}"></title>
</head>
<body>

   <h1 th:text="${titulo}"></h1>

   <h2 th:utext="${user.name}"></h2>

   <div th:switch="${user.role}">
      <p th:case="'admin'">User is an administrator</p>
      <p th:case="'manager'">User is a manager</p>
      <p th:case="'staff'">User is a staff</p>
      <p th:case="*">User is some other thing</p>
   </div>

</body>
</html>
```

FORMULARIOS



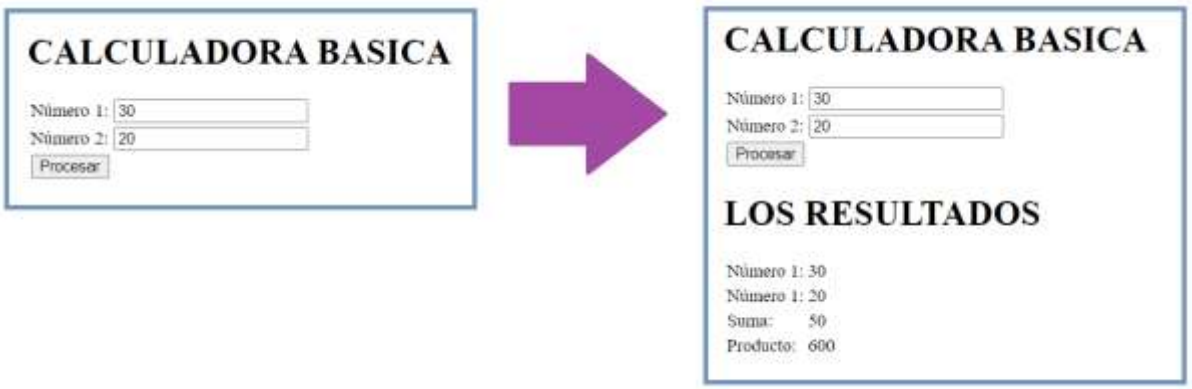
Formulario HTML

Los campos del formulario HTML lo puedes enlazar con los atributos de un bean, tal como lo puedes observar a continuación:

```
<form th:action="@{/url}" th:object="${bean}" method="post">

<div>
   <label>etiqueta</label>
   <input type="text" th:field="*{atributo}"/></
</div>

. . .

</form>
```

Ejemplo 5

En el siguiente ejemplo desarrollaras un programa que permita obtener la suma y el producto de 2 números.



El Bean

En este caso se creará una clase donde almacenaremos los datos y el resultado de las operaciones respectivas.

```
public class MateModel {

   private int num1;
   private int num2;
   private int suma;
   private int producto;

   public MateModel() {
   }

   public MateModel(int num1, int num2) {
      this.num1 = num1;
      this.num2 = num2;
   }

   // Implementar los métodos set y get

}
```

La Página HTML

En está página HTML encuentras el formulario y la sección donde se muestra el resultado.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title th:text="${titulo}"></title>
</head>
<body>
   <h1 th:text="${titulo}"></h1>

   <div>
      <form th:action="@{/procesar}" th:object="${mate}" method="post">
         <table>
            <tr>
               <td>Número 1:</td>
               <td><input type="text" th:field="*{num1}"
                  placeholder="Número 1"></td>
            </tr>
            <tr>
               <td>Número 2:</td>
               <td><input type="text" th:field="*{num2}"
                  placeholder="Número 2"></td>
            </tr>
            <tr>
               <td><input type="submit" value="Procesar" /></td>
            </tr>
         </table>
      </form>
   </div>

   <div th:if="${resultado}">
      <h1 th:text="${resultado}">Aqui van los resultados</h1>
      <table th:object="${mate}">
            <tr>
               <td>Número 1:</td>
               <td th:text="*{num1}"></td>
            </tr>
            <tr>
               <td>Número 1:</td>
               <td th:text="*{num2}"></td>
            </tr>
            <tr>
               <td>Suma:</td>
```

```
            <td th:text="*{suma}"></td>
          </tr>
          <tr>
            <td>Producto:</td>
            <td th:text="*{producto}"></td>
          </tr>

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

Controlador para mostrar la Página HTML

```
  @GetMapping("/calculadora")
  public String calcular(Model model) {
     MateModel mate = new MateModel(0, 0);
     model.addAttribute("titulo", "CALCULADORA BASICA");
     model.addAttribute("mate", mate);
     return "calculadora";
  }
```

Controlador para Procesar el Formulario

```
  @PostMapping("/procesar")
  public String procesar(@ModelAttribute MateModel mate, Model model) {
     // Proceso
     mate.setSuma(mate.getNum1() + mate.getNum2());
     mate.setProducto(mate.getNum1() * mate.getNum2());
     // Reporte
     model.addAttribute("titulo", "CALCULADORA BASICA");
     model.addAttribute("resultado", "LOS RESULTADOS");
     model.addAttribute("mate", mate);
     return "calculadora";
  }
```

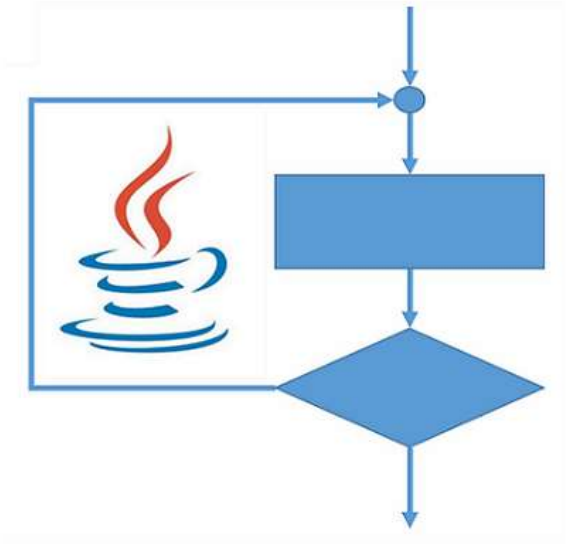
CURSOS VIRTUALES

CUPONES

En esta URL se publican cupones de descuento:

http://gcoronelc.github.io

FUNDAMENTOS DE PROGRAMACIÓN



Tener bases sólidas de programación muchas veces no es fácil, creo que es principalmente por que en algún momento de tu aprendizaje mezclas la entrada de datos con el proceso de los mismos, o mezclas el proceso con la salida o reporte, esto te lleva a utilizar malas prácticas de programación que luego te serán muy difíciles de superar.

En este curso aprenderás las mejores practicas de programación para que te inicies con éxito en este competitivo mundo del desarrollo de software.

URL del Curso: **https://www.udemy.com/course/fund-java**

Avance del curso: **https://n9.cl/gcoronelc-fp-avance**

Cupones de descuento: **http://gcoronelc.github.io**

JAVA ORIENTADO A OBJETOS

En este curso aprenderás a crear software aplicando la Orientación a objetos, la programación en capas, el uso de patrones de software y swing.

Cada tema está desarrollado con ejemplos que demuestran los conceptos teóricos y finalizan con un proyecto aplicativo.

URL del Curso: **https://bit.ly/2B3ixUW**

Avance del curso: **https://bit.ly/2RYGXIt**

Cupones de descuento: **http://gcoronelc.github.io**

PROGRAMACIÓN CON JAVA JDBC

En este curso aprenderás a programas bases de datos Oracle con JDBC utilizando los objetos Statement, PreparedStatement, CallableStatement y a programar transacciones correctamente teniendo en cuenta su rendimiento y concurrencia.

Al final del curso se integra todo lo desarrollado en una aplicación de escritorio.

URL del Curso: **https://bit.ly/31apy0O**

Avance del curso: **https://bit.ly/2vatZOT**

Cupones de descuento: **http://gcoronelc.github.io**

PROGRAMACIÓN CON ORACLE PL/SQL

En este curso aprenderás a programas las bases de datos ORACLE con PL/SQL, de esta manera estarás aprovechando las ventas que brinda este motor de base de datos y mejoraras el rendimiento de tus consultas, transacciones y la concurrencia.

Los procedimientos almacenados que desarrolles con PL/SQL se pueden ejecutarlo de Java, C#, PHP y otros lenguajes de programación.

URL del Curso: **https://bit.ly/2YZjfxT**

Avance del curso: **https://bit.ly/3bciqYb**

Cupones de descuento: **http://gcoronelc.github.io**