Documentación proyecto integrado
Covid Data Ilustration

Indice

1 - Idea del proyecto

La idea del proyecto viene dada por el cliente, quien encarga que, mediante un proyecto Katalon, leer un archivo de extensión .csv y que llame a una api, también aportada por el cliente la url de la misma, y que finalmente Muestre los datos en forma de gráfica. Por otra parte se planteó este mismo proceso mediante un proyecto web para, a parte de añadir contenido, poder ver y entender las diferencias en cuanto a funcionalidad que ofrece Katalon.
La idea en sí estaba motivada por el aprendizaje de Katalon, un framework de automatización de pruebas, el cual permite poder ejecutar varios procesos en cadena de forma rápida y sencilla.
Y en cuanto la temática, se debe a la actualidad que estamos viviendo en todo el mundo, y la necesidad de adquirir información sobre cómo avanzan los acontecimientos.

Cuando se empezó el proyecto Katalon, la idea en sí no era algo novedosa, pero se pretendía crear un proceso de automatización que fuese sencillo, práctico y útil al mismo tiempo, mediante el cual, quien hiciese uso del programa, pudiera acceder a los datos requeridos fácilmente y sin largas esperas. Katalon permite esto

En cuanto al proyecto web, aunque tampoco era novedoso, pero permitía acceder a los datos en un instante, sin marear al usuario. Aprovechando el uso de una api, se evita tener que estar buscando la información directamente y, además, se consigue dicha información formateada, lo que agiliza el trabajo de la aplicación y consigue unos resultados más rápidos.

2 - Descripción de la aplicación:

La aplicación *Covid data ilustration* tiene como fin el mostrar datos del Covid representados mediante gráficas.
En primer lugar, el usuario podrá decidir cómo insertar los nombres de los países de los cuales quiere conocer la información referente al Covid, esto podrá realizarlo o bien mediante la inserción manual de los nombres de los países en un formulario, o bien mediante un archivo de extensión ".csv". Una vez la aplicación haya recogido los nombres de los países, llamará a una api para conseguir los datos de cada uno de los países, los cuales sacará representados en gráficas.

3 Tecnologías empleadas:

-Lenguajes: Java EE, JavaScript
-Librerías HTML - JavaScript: Bootstrap, JQuery
-Librerías Java : JfreeChart, Jcommon, Jasperreports, Json (Java)
-Framwork de automatización: Katalon
-Api: https://api.covid19api.com

Diagrama de vistas:

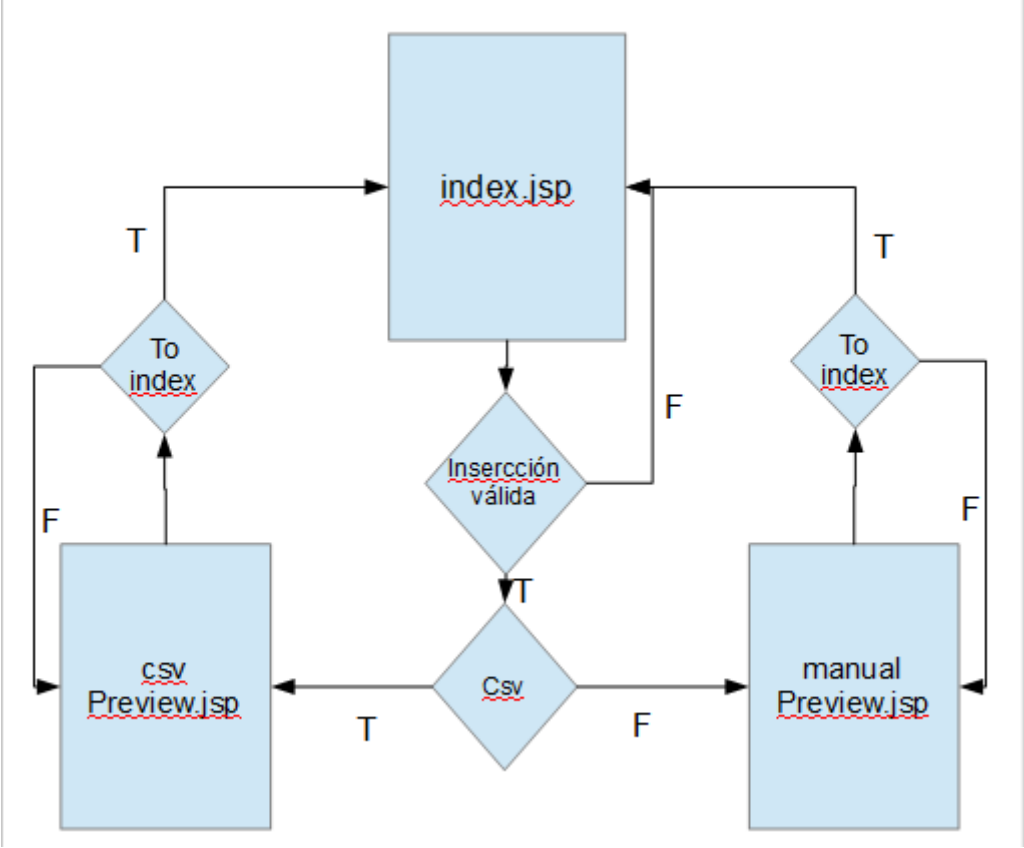
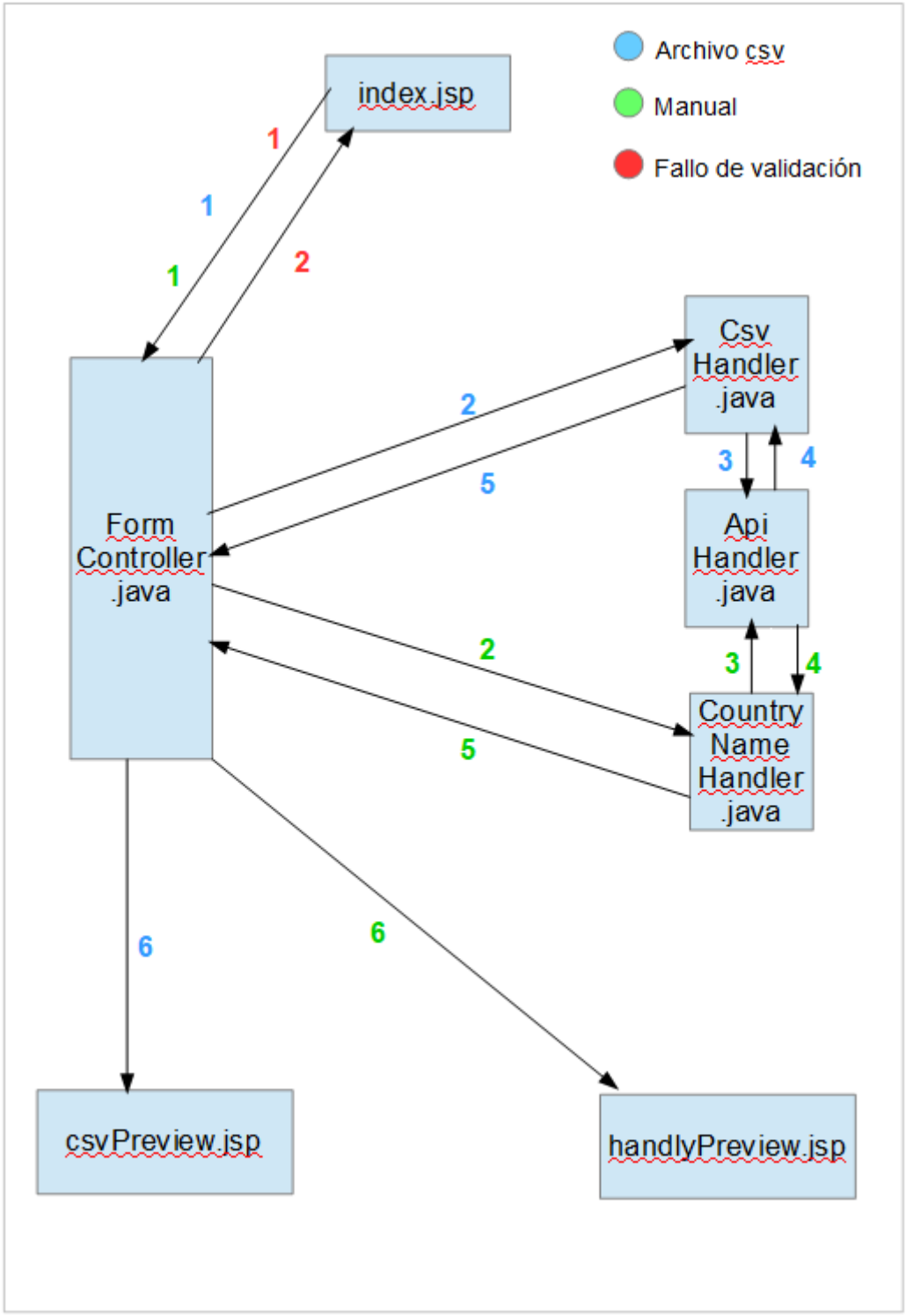


Diagrama de flujos:



Descripción interna de la aplicación:

<table>
<tr><td colspan="2">La aplicación se compone de tres vistas (index.jsp, csvPreview.jsp y handlyPreview.jsp), cuatro clases (ApiHandler.java, CountryNameHandle.java, CsvHandler.java y Utils.java) y un servlet (FormController.java).</td></tr>
<tr><td colspan="2"><u>Las vistas</u></td></tr>
<tr><td>index.jsp</td><td>Es la vista principal de la aplicación, en ella viene el formulario mediante el cual el usuario podrá ingresar los nombres de los países, ya bien sea mediante un archivo csv o mediante una entrada de texto. Los datos que introduce el usuario son enviados al controlador FormController.java.</td></tr>
<tr><td>csvPreview.jsp</td><td>Es la vista que se encarga de mostrar los datos obtenidos tras la lectura del archivo csv que el usuario introduce en el formulario. Representa los datos en gráficas.
Va extrayendo las cadenas de la lista de cadenas que le llega desde FormController.java y va recorriendo cada una para extraer los datos y montar las gráficas.</td></tr>
<tr><td>handlyPreview.jsp</td><td>Es la vista que se encarga de mostrar los datos obtenidos a raíz de la cadena que ha introducido el usuario en el formulario. Representa los datos en gráficas.
Recorre la cadena que le llega desde FormController.java para extraer los datos y montar la gráfica.</td></tr>
<tr><td colspan="2"><u>Las clases</u></td></tr>
<tr><td>CountryNameHandler.java

Es la encargada de procesar la cadena que introduce el usuario en el formulario</td><td>BuildJson():
Recibe una cadena del controlador FormController.java la cual envía en la llamada al método apiCall() de ApiHandler.java para así recibir los datos relativos a esa cadena, en formato json, y posteriormente mandar tales datos al controlador FormController.java.</td></tr>
<tr><td>CsvHandler.java

Es la encargada de procesar el archivo csv que introduce el usuario en el formulario.</td><td>ProcessCsvFile():
Recibe un objeto tipo BufferedReader, en el cual está almacenado un documento csv, el cual lee línea a línea y saca todas las cadenas que tenga y las guarda en una lista que envía a buildJsonList(), en la misma clase, para que este consiga una lista de json (como cadenas), con los datos referentes a las cadenas de la lista enviada, y se la entregue. Finalmente manda al controlador FormController.java la lista de json recibida.</td></tr>
</table>

	BuildJsonList(): Recibe una lista de cadenas de *processCsvFile() en la misma clase*, la cual procesa y envía cadena a cadena a *apiCall()* de *ApiHandler.java* para ir recibiendo los json (en forma de cadena) con los datos referentes a cada cadena enviada. Estos json los guarda en otra lista (de cadenas) que enviará a *processCsvFile()*. **CsvToFormat():** Recibe una cadena que consiste en una línea de un archivo csv, la cual formatea para que las distintas cadenas estén separadas por comas (,).
Utils.java **Se trata de una clase auxiliar que alberga métodos útiles para la aplicación.**	**StringFormat**(): Recibe una cadena que formatea para que no tenga espacios en los extremos y para que sus caracteres se conviertan a minúsculas. Después del formateo devuelve la cadena formateada.
ApiHandler.java Se encarga de realizar las llamadas a la api para conseguir los datos requeridos	**ApiCall**(): Recibe una cadena que usa para hacer la llamada apropiada a la api y recibir los datos deseados, los cuales guarda en otra cadena que manda allá donde fue invocado el método.
Los servlets	
FormController.java	Se encarga de controlar la aplicación. Recibe los datos que el usuario introduce en el formulario en *index.jsp* y se encarga de mandarlos a los métodos convenientes (en clases ajenas al servlet) para ser procesados, cuya respuesta dirigirá a las vistas adecuadas.

*

Dificultades encontradas

La primera dificultad encontrada fue con el lenguaje de programación *Groovy,* debido a que usar dicho lenguaje junto a sus frameworks era un quebradero de cabeza ya que cuando encontraba cómo montar el proyecto no me detectaba las palabras claves o el framework, aunque tuviese la estructura y las tecnologías instaladas en mi ordenador. Es por ello que decidí cambiar el proyecto de *GSP* a *JSP* después de varios días intentándolo.

El siguiente problema con el que me topé fue con la librería de *Java OpenCsv,* que después de aprender a manejarla me di cuenta de que el servidor no tiraba, y era debido a que la clase que detecta las excepciones me daba un error de falta de jdoc. Pedí ayuda a varias personas por webs donde hacían tutoriales, pero no recibí respuesta. Al final después de buscar, encontré un tutorial en el que se usaba *BufferReader* para leer el documento csv, la solución a mi problema era súper sencilla y Google solo me mostraba tutoriales de *OpenCsv.*

Después de solucionar mi problema para leer los documentos con extensión .csv, me quedé bloqueado con JfreeChart (para las gráficas). Encontré muy pocos tutoriales que tratasen JSP, y los que encontré usaban un archivo jsp como servlet (un poco feo) y no usaban las gráficas que yo necesitaba. Gracias a la documentación de la librería fui sacando el cómo crear la gráfica, pero me empezó a dar un error que me decía que los archivos importados estaban mal, esto me dio bastantes quebraderos de cabeza. Al final resultó que debía añadir los jar de *Jasperreports.*

En cuanto a Katalon, era una tecnología bastante novedosa para mí, con lo que no comprendía muy bien como funcionaban o encajaban las cosas. Me ha llevado bastante tiempo adaptarme a dicho framework, pero al menos ya he conseguido entender la lógica de funcionamiento de Katalon y saber como realizar pruebas un poco más complejas mediante scripts.

Historial de versiones:

V0.0 – Preparación

-Se ha generado la documentación
    -Se ha incluido una descripción de la aplicación
    -Se ha incluido un listado provisional con las tecnologías que se van a utilizar

-Se ha incluido un historial de versiones

V0.1 – Preparación

-Se ha modificado la documentación
    -Se ha incluido en un documento aparte un boceto de las vistas de la aplicación
    -Se ha incluido un boceto del diagrama de vistas de la aplicación

-Se han modificado el listado de las tecnologías
    -Se han descartado los frameworks Gaelyk y Spring-Boot

-Se ha añadido una breve descripción de la finalidad de la aplicación a la descripción de la misma

-Se ha añadido un manual de usuario

V1.0 – Desarollo

-Se ha creado el formulario de inserción de datos
    -Se han generado dos subformularios, uno para cargar el archivo csv y otro manualmente

V1.1 – Desarollo

-Se ha modificado el sistema de formularios
    -Ahora ambos formularios están en el index, pero por separado

-Se han modificado el diagrama de vistas y el boceto del diseño de las mismas

V1.2 – Desarrollo (framework)

-El proyecto se ha mudado a Grails
    -Se empleará Groovy Server Pages (GSP)

V1.3 – Desarrollo (cambio tecnología)

-Debido a los problemas encontrados, el proyecto ahora usará JSP en lugar de GSP

-El formulario para la introducción de los países será el mismo para ambos casos

-Se ha modificado el archivo de las vistas.

V1.4 – Desarrollo

-El formulario está terminado
    -Funciona perfectamente, se valida que solo se use una de las dos opciones

-Las vistas han sido cambiadas

-Se usará OpenCsv para gestionar el archivo csv que envía el usuario

V1.5 – Desarollo

-Se ha prescindido de la librería OpenCsv para la lectura del archivo .csv al dar problemas con la captura de excepciones
    -A partir de ahora se usa BufferReader

-La aplicación toma los nombres de los países y hace la llamada a la api correctamente
    -Se obtinen los json correspondientes

V1.6 – Desarrollo

-Las vistas han sido cambiadas

-El diagrama de vistas ha cambiado
    -Ahora hay una vista para los diagramas de cada sistema de ingreso de los nombres de los países

-Se ha agregado a la documentación un diagrama de flujo de datos.

V1.7 – Desarrollo

-Las vistas han sido cambiadas a su forma definitiva
    -Ahora disponen de un botón para regresar a inicio

V1.8 – Desarollo

-Se usará JFreeChart y Jasperreports para la generación de las gráficas

V1.9 – Desarrollo

-Se ha agregado a la documentación un apartado para exponer las dificultades encontradas durante el proyecto

-Se ha agregado a la documentación una explicación de las clases y sus métodos y de las vistas

-Se ha agregado a la documentación una webgrafía

-La aplicación está terminada
    -Se pasa a la fase de pruebas

V2.0 – Pruebas

-Se ha encontrado un fallo derivado de un problema de la propia api a la que se llama
-El problema está solventado

V2.1 – Explotación

-La aplicación está lista.

Webgrafía

Oficiales

Bootstrap:

https://getbootstrap.com/

JQuery:

https://jquery.com/

Katalon:

https://www.katalon.com/

Java EE:

https://jakarta.ee/release/9/?utm_term=%2Bjava%20%2Bee&utm_campaign=Brand&utm_source=adwords&utm_medium=ppc&hsa_acc=7706012104&hsa_cam=11475443642&hsa_grp=118940440784&hsa_ad=485403291773&hsa_src=g&hsa_tgt=kwd-322177491042&hsa_kw=%2Bjava%20%2Bee&hsa_mt=b&hsa_net=adwords&hsa_ver=3&gclid=CjwKCAiAoOz-BRBdEiwAyuvA6wKRjWpIaT4if6NBVqYS1IjWKO1-byndtQ3m4Wl8FFUdfb6RSbbyQxoCZOwQAvD_BwE

JfreeChart:

https://www.jfree.org/jfreechart/

Jasperreports:

https://community.jaspersoft.com/project/jasperreports-library

Json (Java):

https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html

No oficiales consultados (solo las que han servido de ayuda)

Tutorial JfreeChart:

https://ingeniods.wordpress.com/2010/05/05/diagrama-de-barras-en-jsp/

Tutorial JfreeChart

https://www.javascan.com/

Respuesta en StackOverflow sobre el uso de Json en Java:

https://es.stackoverflow.com/questions/154596/c%C3%B3mo-recorrer-un-json-con-java