Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Informática

Navegación Web Automatizada con simulación de Comportamiento Humano

Trabajo fin de estudio presentado por:	Manuel Di Nucci
Línea de investigación:	Ingeniería en la Web, servicios Web y Seguridad en la Web.
Director/a:	Bermejo Gonzalez Belen
Fecha:	19/03/2023

Resumen

Desde el origen de los tiempos, el ser humano se ha visto envuelto en la tarea de automatizar sus procesos manuales, de modo tal que su interacción se vea reducida y por ende así su esfuerzo, pero para que los resultados de un proceso automatizado sean lo más equivalente posible al proceso manual existe un requisito indispensable a saber: un importante set de pruebas debe ejecutarse antes de dar por finalizada la tarea de creación de la automatización.

El mundo del software no es para nada ajeno a esta premisa de automatización y es quizás una de las áreas que llevan estos conceptos de automatización y pruebas a su máxima expresión, es decir, cada vez que creamos una aplicación (sea cual sea su naturaleza) necesariamente debe estar acompañada de una buena cantidad de pruebas que nos otorguen al menos una alta confiabilidad respecto de que nuestra aplicación hace correctamente las operaciones para las que fue creada.

Mediante un conjunto de herramientas informáticas he creado una plataforma que permite codificar diferentes tipos de pruebas que simulan la operativa humana de navegación en un sitio web, incrementando así la calidad de las mismas, así como las áreas de evaluación.

Palabras clave: WebDriver, Automatización de Aplicaciones Web, interacción Humano-Computador, Pruebas Automatizadas, Simulación

Abstract

Since the beginning of time, humans have been involved in the task of automating their manual processes so that their interaction is reduced and thus their effort. However, to ensure that the results of an automated process are as equivalent as possible to the manual process, an indispensable requirement must be met: a significant set of tests must be executed before the task of creating automation is considered complete.

The world of software is not at all unfamiliar with this premise of automation and is perhaps one of the areas that take these concepts of automation and testing to the maximum expression. Every time we create an application (whatever its nature may be), it must necessarily be accompanied by a good number of tests that give us at least a high degree of confidence that our application performs the operations for which it was created correctly.

Through a set of computer tools, I have created a platform that allows for the coding of different types of tests that simulate human navigation operations on a website, thereby increasing their quality as well as the areas of evaluation.

Keywords: WebDriver, Web Application Automation, Human Computer Interaction, Automated Test, Simulation

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

Hoy en día más del 90% de las operaciones que se realizan en internet empiezan con una búsqueda en Google y derivados. Google controla cerca del 92% de la cuota de mercado de los motores de búsqueda en todo el mundo. Eso incluye el 72% del mercado de ordenadores de escritorio y el 92% del mercado de motores de búsqueda móviles (CepymeNews, 2022).

Dicho esto, pasamos a la parte del Testing Automatizado o Automatización de Pruebas Funcionales, actualmente, herramientas tales como Selenium[1] o Puppeteer[2] entre otras, nos permiten ejecutar una serie de comandos de manera automatizada en un explorador Web (sea cual sea el elegido), estas herramientas actúan como una cierta capa de abstracción sobre las operaciones comunes que realiza cualquier ser humano (Navegar, Mover el Mouse, Hacer Click, etc.), pero con la salvedad de que estos "Frameworks[3]" de automatización nos permiten justamente ejecutar una serie de operaciones de manera y/o utilizando un sistema informático. De esto surge su potencial en la ejecución de pruebas en los entornos de programación y/o compañías. Supongamos por un momento que somos dueños de una empresa que tiene una aplicación web en la cual queremos realizar pruebas varias directamente desde la interfaz productiva (en un entorno controlado claramente), simulando lo mejor que se pueda un usuario real, de hecho, ese es el principio básico de toda prueba, "Si una prueba sea cual sea se aleja demasiado de la realidad, por ende no es una prueba factible o útil en principio", volviendo a nuestro ejemplo, vamos a realizar una serie de pruebas simulando el comportamiento humano como objetivo básico y elegimos algunos de estos "Frameworks" de automatización para escribir y llevar a cabo dichas pruebas. Por ende, elegimos una de estas herramientas, codificamos nuestras pruebas y por ultimo las ejecutamos. Lo primero que vamos a notar es que, todas las operaciones llevadas a cabo se alejan muchísimo de como las llevaría a cabo un ser humano real, estos "Frameworks" no tipean caracteres en los campos de búsqueda como lo haría una persona normal en términos

[1] https://www.selenium.dev/projects/
[2] Librería Node.js que proporciona una API de alto nivel para controlar Chrome/Chromium sobre el protocolo DevTools. Fuente: https://pptr.dev/
[3] Esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software. Fuente: https://www.edix.com/es/instituto/framework/

de digamos velocidad, no mueven el puntero del ratón y demás operaciones como las haría un ser humano, el tipado carece de una frecuencia humana, el movimiento del mouse sigue coordenadas o realiza movimientos robóticos sin los retardos de tiempo entre coordenadas X_0Y_0 e X_1Y_1 propios de un ser humano y demás operaciones que básicamente radican en un solo inconveniente, no se ejecutan con los retardos, frecuencias, aleatoriedad, etc. propias de un ser humano. En resumen, las trayectorias del ratón son completamente diferentes para los humanos y para los bots. Estos últimos en general pueden mover el mouse de la manera más "efectiva", es decir, evitando giros o curvas innecesarias. Es casi imposible para los humanos mover un ratón de la misma manera "directa". En la siguiente imagen puede verse una gráfica del movimiento humano (A) y la de un bot (B).

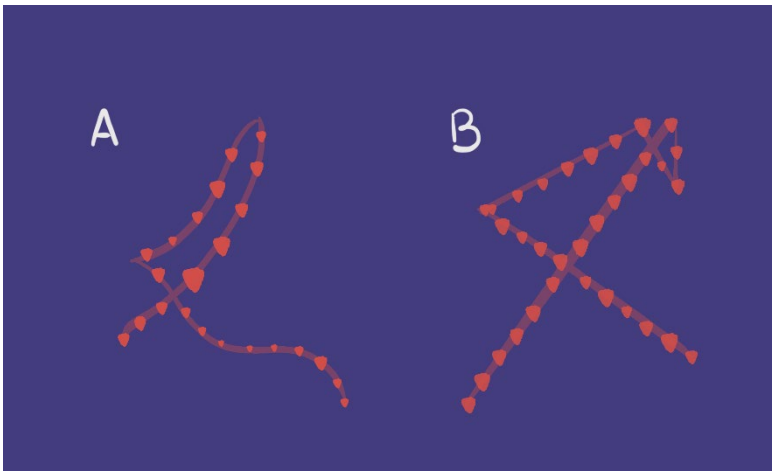


Ilustración 1 - Movimiento Humano vs Bot. Fuente: https://www.digica.com/blog/improving-bot-detection-with-ai.html

Además, tenemos otro inconveniente, como dijimos anteriormente, hoy en día la mayoría de los seres humanos al momento de navegar en la web, primeramente se dirigen a un buscador como Google, por ende, recordando nuestra premisa básica “simular el comportamiento de navegación humana”, si queremos ejecutar nuestras pruebas de la manera más representativa posible, deberíamos previamente dirigir nuestras pruebas a estos buscadores, buscar nuestro sitio en cuestión en los resultados, ingresar en él (si el resultado esta previamente indexado) y luego realizar las operaciones predefinidas en los casos de prueba. Ahora bien, suponiendo que todo lo anteriormente mencionado ya lo tenemos resuelto y podemos dirigir nuestra aplicación de pruebas a los buscadores y buscar nuestra web, surge

uno de los mayores inconvenientes que tienen estos buscadores, así como también la mayoría de los sitios y este es: la detección de bots[4].

Estas herramientas de pruebas automatizadas y en base a sus "movimientos robóticos" son detectadas como bots, siendo este el principal objetivo a resolver a la hora de realizar este tipo de pruebas y no solo para evitar ser detectados como bots sino también para dotar de "humanismo" a las operaciones realizadas en dichas pruebas, lo cual nos dé como resultado maximizar la calidad de las pruebas en cuestión.

1.1. Justificación del tema elegido

La programación en los últimos tiempos viene sufriendo cambios extremadamente acelerados, tomando como ejemplo y como cierto punto de partida el famoso caso de AngularJS[5] a Angular 2 o el caso de Angular 2 a Angular 4, es cuando todo ciertamente cambio, empezó a ser mucho más frecuente y disruptivo, en aquel momento, lo que nos valía para AngularJS dejo de valer para Angular 4, es decir, uno podía ser un experto en AngularJS y no así para Angular 4. El anterior caso, es simplemente un ejemplo, nadie sabe a ciencia cierta cuando comenzó esta actualización como patrón común en todas las áreas de la tecnología y ciertamente se entiende que esta frecuencia del cambio vino para quedarse.

El testing también fue otra de las víctimas de este cambio voraz (y aún lo es y lo sufre), aunque el área de pruebas siempre fue preponderante en todo desarrollo, estas estaban sujetas a ciertas vicisitudes tales como: pruebas unitarias, pruebas de integración, regresiones manuales y demás. Un porcentaje alto de pruebas, así como el de Code Coverage[6] era casi una utopía y peor aún, todas estas pruebas estaban supeditas a lenguajes específicos de programación. Con lo cual y potenciado por esta vorágine de cambios mencionada anteriormente, lo que antiguamente había sido programado digamos en AngularJS como

[4] Aféresis de robot, programa informático que efectúa automáticamente tareas reiterativas mediante Internet a través de una cadena de comandos o funciones autónomas previas para asignar un rol establecido; y que posee capacidad de interacción, cambiando de estado para responder a un estímulo. (Tsvetkova, García-Gavilanes, Floridi, & Yasseri, 2017)
[5] Framework MVC (Modelo Vista Controlador), desarrollado por Google para el Desarrollo Web Front End que permite crear aplicaciones SPA (Single-Page Applications) (Galán, 2020).
[6] Método de análisis que determina qué partes de nuestro código han sido cubiertas, es decir, han sido ejecutadas por las pruebas unitarias, y qué partes no lo han sido.

parte de las pruebas, con la aparición de Angular4 debía ser reprogramado, incurriendo así en una baja reutilización de código, perdida de recursos, etc. Es importante mencionar que, si bien los altos porcentajes de pruebas siempre fueron vistos como excelencia en la calidad de código, la realidad era que no solo eran realmente una utopía (ya que todos destacaban su importancia pero pocos lo llevaban a cabo con excelencia) sino que además eran muy limitadas en su alcance, todos los tipos de prueba existentes se ocupaban en su mayoría de probar porciones de código o de funcionabilidad, relegando la prueba de la aplicación y su ecosistema completo a pruebas manuales o poco automatizadas.

En resumen, hoy en día y usando de ejemplo una página Web, la cual es un denominador común para la mayoría de las personas, lo que tenemos debajo de esa página Web, el lenguaje con el que se desarrolló esa página Web, sus operaciones y programación pueden estar realizadas en una gran cantidad de lenguajes de los más variopintos, lo cual claramente dificulta tanto su concreción como por ende la codificación de sus pruebas sea cual sea su tipo.

Ahora bien, siguiendo con este ejemplo, una página Web es claramente una cierta cantidad de recursos alojados en la web los cuales se muestran de cara al usuario con una determinada interfaz, sea cual sea los lenguajes utilizados para su concreción a fin de cuentas no es más que eso, una interfaz que realiza y expone información ajustada a un determinado tópico y acá es donde las pruebas automatizadas se hacen fuertes. Una prueba automatizada no se centra en cómo o cuando o frente a que entradas genera que salidas, una prueba automatizada desde el punto de vista de quien la escribe podríamos decir que es una suerte de caja negra, la página web, la aplicación web es simplemente una caja negra, para el caso de prueba en cuestión no le interesa en que lenguaje fue codificada, no le interesa que la función “A” realice los pasos “1” y “2”, sino que simplemente se centra en decir: dado la siguiente operación y siempre y cuando este en determinado lugar del sitio, debo recibir tal resultado. Es decir, las pruebas automatizadas son en cierto punto agnósticas de lo que existe debajo y ahí radica su potencia.

Dicho esto, el objetivo del presente trabajo busca una forma de dotar de “humanismo” a las operaciones realizadas en las pruebas automatizadas, es decir, busca evitar que las pruebas automatizadas utilizando Selenium sean detectados como bots, pero siempre enfocadas al siguiente patrón y de manera completamente automatizada:

1. Web previamente indexada en Google
2. Búsqueda en Google de la Web en cuestión
3. Redirección a la Web en cuestión
4. Navegación concreta y esperada

Por otro lado, mediante esta propuesta, se busca separarse del lenguaje de programación utilizado para el desarrollo de la aplicación web obteniendo así una plataforma de desarrollo de pruebas automatizadas que evitan ser detectadas como bot y permite potenciar la calidad de las pruebas en cuestión, además de incrementar la operación crítica y objetiva de cada una de las pruebas. Esto permitirá dar un paso más en la mejoría de las pruebas y reducir la brecha producto de la voracidad de los cambios antes mencionados, así como incrementar el grado de generalización de las herramientas de pruebas automatizadas para entornos web.

Como aporte personal, para mí que llevo años en el ámbito del desarrollo en multiplicidad de compañías y dada las características de la evolución informática en donde todo se espera que sea cada vez más WEB en vez de escritorio, todo más DISTRIBUIDO en vez de LOCAL, este trabajo representa una aportación práctica y valiosa a la comunidad participe en querer generar cada vez más, software de mejor calidad.

1.2. Problema y finalidad del trabajo

Cuando hablamos de pruebas automatizadas en general nos referimos a codificar acciones que posteriormente van a ser ejecutadas de manera automatizada, cuando estas incluyen Selenium nos referimos a codificar acciones utilizando este “Framework” que posteriormente van a ser ejecutadas en un navegador web de manera automatizada. Un ejemplo de esto es simplemente cualquier navegación web cuyas acciones no estén comandadas por un ser humano en tiempo real y que permitan ser ejecutadas a posteriori y de forma repetitiva. A priori la diferencia más radical es la codificación previa de las acciones en un cierto lenguaje de programación y su posterior ejecución. Ahora bien, si simplemente, tomamos un entorno IDE de desarrollo, elegimos www.google.com como url destino y la palabra “UNIR” como termino a buscar, codificamos algunas acciones simples utilizando Selenium en alguno de los múltiples lenguajes aceptados, ejecutamos estas acciones y vemos los resultados, estos serán a grandes rasgos y de manera automática los siguientes:

1) La aplicación mediante Selenium:
   a) Abre el navegador elegido
   b) Ingresa www.google.com en la sección URL del navegador
   c) Navega a www.google.com
   d) Busca y selecciona el campo de búsqueda de Google
   e) Ingresa (Tipea) “Unir” en el campo de búsqueda
   f) Busca y presiona el botón “Buscar”
   g) La página de detección de bot de Google aparece con la opción para resolver un captcha[i]

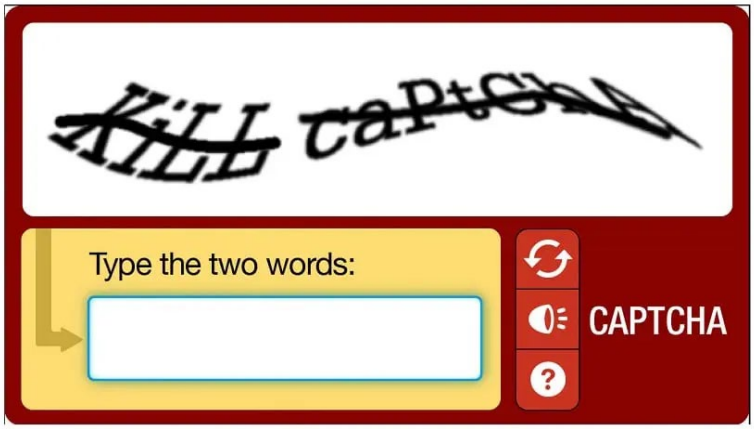


Ilustración 2 - Primer Nivel de Captcha. Fuente: https://www.windowsnoticias.com/cs/captcha-a-recaptcha%2C-co-jsou-za%C4%8D-a-jak-se-li%C5%A1%C3%AD/

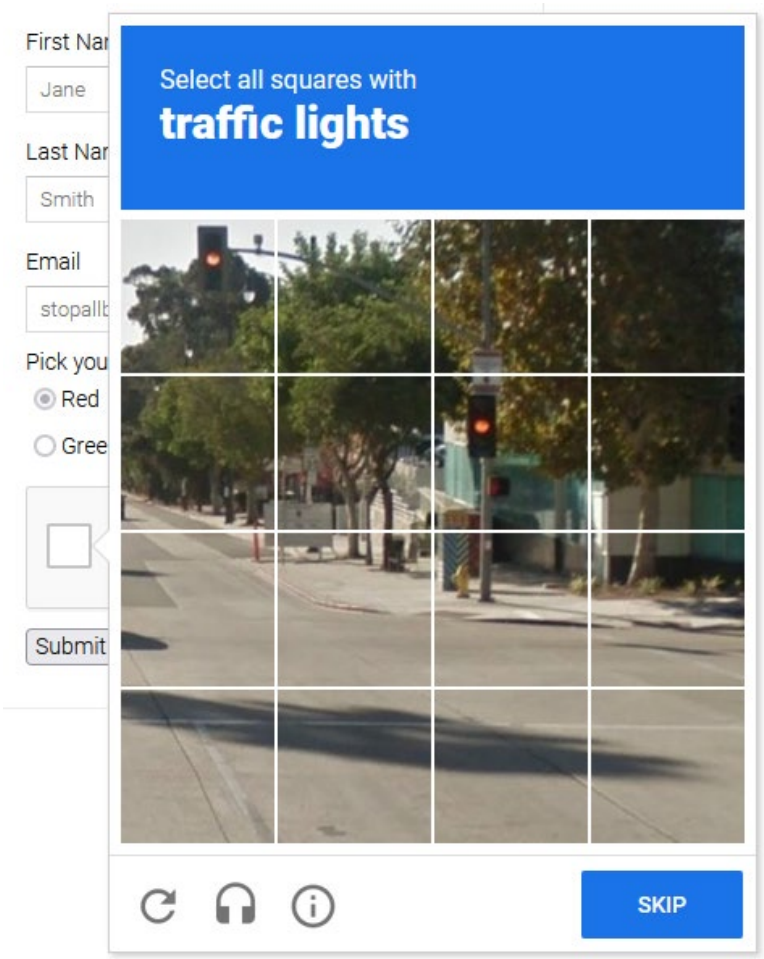


Ilustración 3 - Segundo Nivel de Captcha. Fuente: https://east-ee.com/2022/03/09/1367/

Este resultado claramente no es el esperado, es decir, si lo que buscamos es realizar pruebas lo más similares a un humano entonces hemos fallado radicalmente, si es cierto que este ejemplo se aplica simplemente a Google, pero a efectos prácticos sirve para explicar el punto. Google, siendo uno de los motores de búsqueda y Web más potentes al momento de la detección de bots sirve como punto de partida para nuestro objetivo. Google y la mayoría de las webs que poseen un mecanismo de detección de bots serio y reactivo, es decir, mecanismos que no están activos todo el tiempo sino que se activan al detectar comportamiento sospechoso, se basan en analizar el comportamiento, es decir, antes de mostrar el "Captcha" verifican el comportamiento previo, basándonos en nuestro ejemplo, Google analiza como fue el movimiento del mouse y con qué frecuencia y espacialidad se presionaron las teclas en el campo de búsqueda y demás acciones.

Por ende, lo interesante es tratar no de crear un mecanismo que permita evitar los controles que tienen los sitios para detección de bots sino generar o mejorar las formas en las que se realizan las pruebas automatizadas de manera que podamos aumentar la calidad de estas. De esta manera poder contar con herramientas agnósticas del lenguaje de programación del sitio Web en cuestión que nos aproximen cada vez más a pruebas cercanas a quizás cierta aleatoriedad producto de la naturaleza humana, otorgándonos una mayor calidad, reutilización de código, generalización y demás beneficios.

1.3. Objetivos del TFE

1.3.1. Objetivo General

Diseñar una herramienta que nos permita desarrollar una serie de operaciones web evitando ser detectados como bot y que se ajusten a ciertas características del comportamiento humano, tales como:

- Movimiento del ratón
    - Duración de los movimientos directos o número y cantidad de movimientos del ratón
    - Velocidad del cursor
- Frecuencia y espacialidad de la presión de las teclas del teclado
    - Ritmo y la velocidad de tecleo

Esta herramienta no estará basada en grabaciones previas como es ofrecido por ciertas herramientas de Screen Recorder[7] y dispondrá de operaciones excluyentes y operaciones aleatorias, siendo las operaciones excluyentes las consideradas “necesarias” para realizar la prueba y las “aleatorias” las consideradas como “relleno” para dotar de un comportamiento más aleatorio que nos alejen del comportamiento robótico o automatizado que claramente arrojan un patrón repetitivo y son posteriormente detectadas como bot.

[7] Software que permite grabar operaciones realizadas por un humano y luego generar código de prueba en algunos casos y en otros reproducir a posteriori esos pasos grabados previamente.

1.3.2. Objetivos específicos

- Crear dos tipos de servicios del tipo Worker[8] utilizando C# a saber:
    - Feeder: encargado de proveer las navegaciones y acciones requeridas y/o excluyentes, actuando de Publisher/Productor (encargados de escribir mensajes) de un tópico de Kafka[9] enviando por cada mensaje el destino y la cantidad de permutaciones de acciones (sino las acciones en concreto) que se quieren realizar.
    - View Sender: encargado de actuar de Consumer/Subscriptor (encargados de leer y procesar los mensajes) de este tópico, y mediante la implementación de patrones tales como Builder (creación de la receta de navegación aleatoria), Chain of Responsability (orquestador de los tipos de operaciones), etc. y utilizando algoritmos tales como Bezier o variaciones de key rollover y además mediante el framework de Selenium, nos permitirá enviar ordenes de navegación que simulen el comportamiento humano superando así las restricciones de detección de bots
- Generar una navegación automatizada que busque un sitio en Google y no sea detectado como bot
    - Navegar en el antedicho sitio de manera automatizada
- Generar operaciones de movimiento de mouse que persigan movimientos curvilíneos utilizando el algoritmo de Bezier
- Generar operaciones de escritura o presión de teclas de teclado con retardos aleatorios obviando patrones repetitivos
- Generar un archivo de Docker Compose que permita su instalación y ejecución en contenedores
- Utilizar un tópico de mensajería Kafka que permita él envió de mensajes entre Feeder y View Sender

[8] Servicios multiplataforma en segundo plano. se ejecuta sobre el concepto de host, que mantiene la vida de la aplicación https://learn.microsoft.com/es-es/dotnet/core/extensions/workers

[9] Sistema de mensajería y una plataforma completa de streaming y de procesamiento de datos en tiempo real. Fuente: https://datademia.es/blog/que-es-apache-kafka#:~:text=Apache%20Kafka%20es%20un%20sistema,de%20la%20Apache%20Software%20Foundation.

2. Marco teórico

Este trabajo se basa en dos grandes fuentes teóricas todas contextualizadas o tomando como punto de partida y referencia la navegación web.

En primer lugar, tenemos el contexto de detección de bots o también llamado análisis y comprensión de los comportamientos automatizados que se centra en determinar las bases o reglas para decidir si cierto conjunto de operaciones y/o acciones corresponden a un humano o no, es una suerte de extensión de la prueba de Turing aplicado a navegación Web, en este apartado se incluye también la comprensión de los comportamientos humanos, es decir cómo y de qué forma se comporta o en base a que reglas podemos determinar que cierta navegación es ciertamente producto de un ser humano.

En segundo nivel, tenemos el contexto puramente técnico, en donde incluimos las herramientas, patrones, algoritmos y demás cuestiones informáticas que pueden dotarnos de los recursos necesarios para llevar a cabo ciertas acciones de manera automatizada evitando caer dentro de las reglas resultantes del contexto de detección de bots y acercándonos el máximo posible a los resultados obtenidos de nuestro análisis del comportamiento humano.

Si bien es cierto que existen múltiples trabajos enfocados en estudiar y limitar el accionar malicioso de los bots, también es cierto que la mayoría de los trabajos se enfocan en las características nocivas de estos y no centran su visión en la potencia que estos ofrecen a las áreas de testing automatizado. Este trabajo como anteriormente mencionamos se centra en utilizar estas herramientas con el fin de potenciar la calidad y fiabilidad de las pruebas automatizadas de páginas web.

2.1. Conceptos Generales

Antes de avanzar, es importante hablar de algunos conceptos que se manejan en este trabajo. Esto es importante ya que nos permite centrarnos en profundidad en los detalles de implementación más allá de invertir tiempo en la explicación conceptual de cada termino y/o idea, herramienta, etc.

2.1.1. Receta

Llamamos receta a un comportamiento o deseo humano o automatizado que se ve satisfecho por un conjunto de operaciones tales como una navegación a un sitio web, como ejemplos tenemos:

- Navegar hacia Google, buscar un concepto, encontrar un sitio web que satisfaga el criterio de búsqueda y navegar hacia el sitio, encontrando posteriormente lo buscado

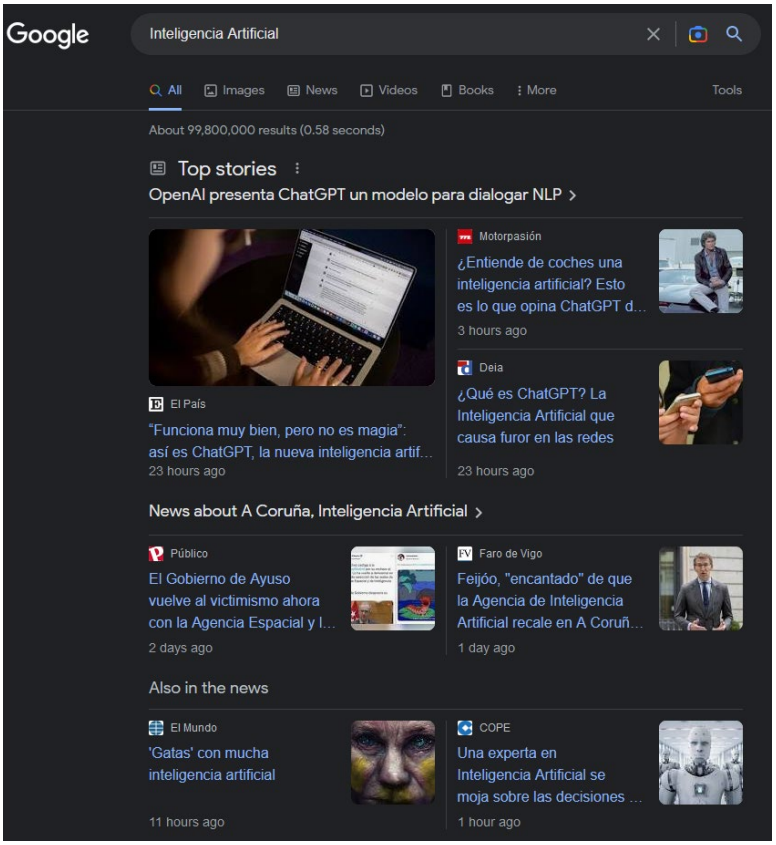


Ilustración 4 - Resultado de un Comportamiento/Deseo. Fuente: Elaboración Propia

2.1.2. Operaciones

Son aquellas que poseen un objetivo específico dentro de una receta, esta operación es un paso más para concretar nuestro deseo. Para satisfacer un comportamiento o deseo necesitamos de una serie de operaciones dispuestas en un determinado orden, las cuales como mencionamos anteriormente poseen un objetivo específico, como ejemplos tenemos:

- Mover el ratón de la posición P1 a la posición P2 (Ver imagen debajo)
- Hace clic en un botón/HYPERLINK, etc.

- Hacer scroll
- Esperar o introducir una demora

Cada uno de los puntos de la imagen debajo son operaciones dentro de una operación mayor que es la de “Desplazamiento” marcada en amarillo entre el punto P1 hasta el punto P2. Cada operación definida puede ser tratada como una unidad atómica o la mínima expresión de un accionar o bien puede ser tomada como un conjunto de estas dando como resultado un grupo de operaciones.

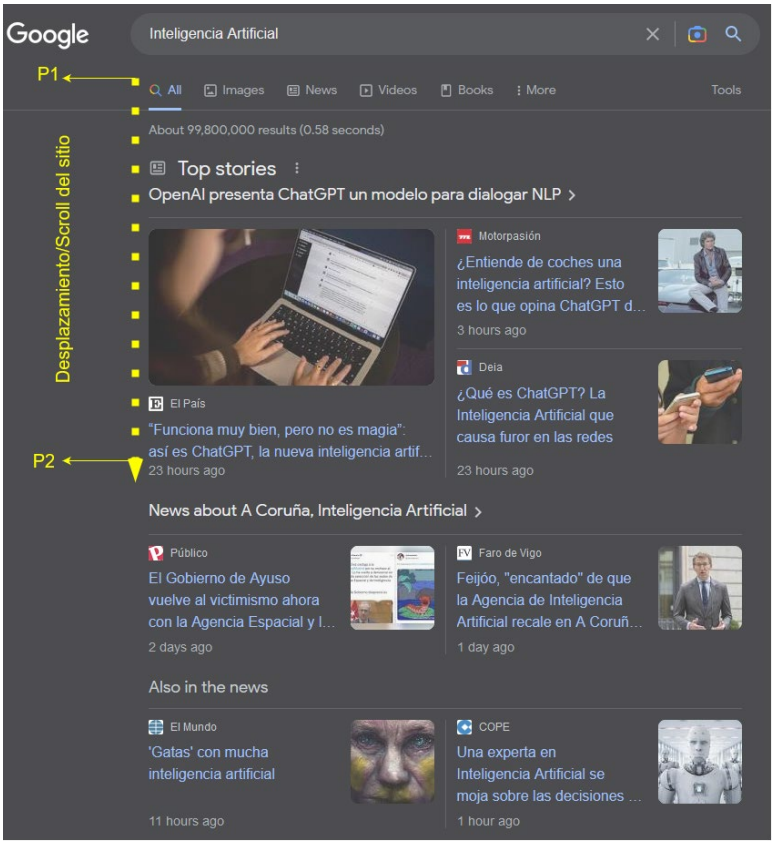


Ilustración 5 - Operación de Desplazamiento. Fuente: Elaboración Propia

Podemos decir que un ser humano para satisfacer un “deseo” hace uso de una “receta” en la cual se indican las “operaciones” necesarias para ver visto satisfecho ese deseo.

2.1.3. Operaciones requeridas (MA)

Son aquellas operaciones que si no existen en la “receta” o no son llevadas a cabo entonces no veríamos satisfecho nuestro “deseo”, por ejemplo: sino presionamos la tecla “enter” en

Google o no presionamos el botón de “búsqueda” nunca confirmaremos la acción de búsqueda en el sitio.

En la siguiente imagen vemos las acciones requeridas marcadas en “Rojo” definidas como MA1/2 (Mandatory Actions en inglés).

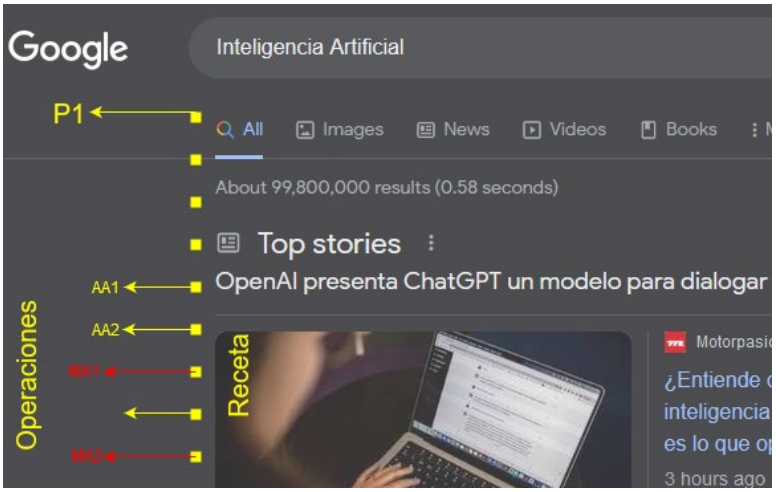


Ilustración 6 - Operaciones Requeridas. Fuente: Elaboración Propia

2.1.4. Operaciones de relleno (AA)

Son aquellas operaciones que pueden o no existir en la “receta” pero su importancia radica en que otorgan o favorecen a que cuando se lleve a cabo la “receta” el comportamiento simule el de un ser humano.

En la siguiente imagen vemos las acciones requeridas marcadas en “Amarillo” definidas como AA1/2 (Aleatory Actions en inglés).

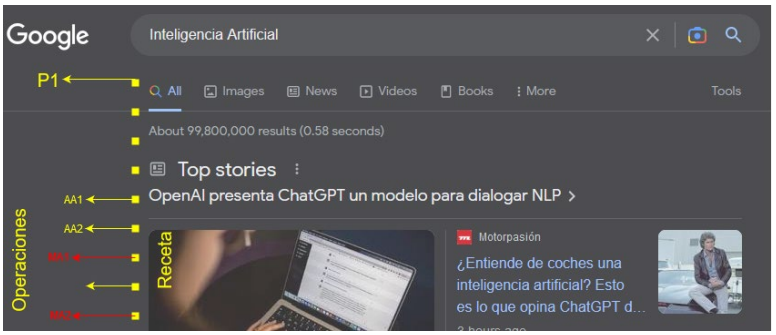


Ilustración 7 - Operaciones de Relleno. Fuente: Elaboración Propia

2.1.5. Micro/InterOperaciones (Oox)

Son de la misma y exacta naturaleza que las operaciones tanto requeridas como de relleno, es decir, son simplemente operaciones, pero utilizamos este concepto para destacar aquellas operaciones que pueden o no ejecutarse dentro de una operación en concreto, podría decirse que son la mínima expresión de un accionar. Si una operación padre dada no tiene ninguna micro/Inter operación desde su comienzo a su fin, podemos decir que esa operación padre es igual o se ve descripta por su micro/interoperación.

2.1.6. Operaciones Prohibidas (Oe)

Son de la misma y exacta naturaleza que las operaciones tanto requeridas como de relleno, es decir, son simplemente operaciones, pero utilizamos este concepto para destacar aquellas operaciones que no deben ni pueden ejecutarse ya que romperían la posibilidad de seguir ejecutando la aplicación o las diferentes operaciones, un ejemplo de estas operaciones es:

- Presionar el botón derecho del ratón durante una navegación abre el menú contextual y redirige el foco de la aplicación
- Abrir el panel de impresión del explorador, abre un popup de impresión el cual redirige el foco de la aplicación

2.1.7. Template de Operaciones u Operaciones Fijas

Al igual que las anteriores operaciones estas no presentan diferencia alguna, pero tienen dos particularidades, la primera es que siempre están asociadas a un dominio particular y la segunda es que usualmente no sufren cambios. Es un concepto que usamos para aquellas operaciones que necesariamente tienen que repetirse para una URL en particular ya que de no llevarse a cabo no permitirían que las operaciones siguientes se ejecuten correctamente, algunos ejemplos de este tipo de operaciones son:

- Aceptar el popup de cookies

2.2. Análisis de Comportamiento

El análisis de conducta o comportamiento aplicado es una rama de la ciencia que busca encontrar un marco que permita definir cómo se comporta un humano. El término "análisis conductual aplicado" o "análisis de conducta aplicado" hace referencia a un tipo de procedimiento que utiliza los principios y técnicas de la psicología del aprendizaje. Esto aplicado al área de computación y en especial a "como" los humanos interactúan con el computador busca principalmente entender como una persona por ejemplo navega por internet, con qué frecuencia hace clic, de qué manera mueve el ratón, etc. Es decir, busca anticiparse de manera proactiva a catalogar y categorizar un accionar conductual determinado, predefiniendo grupos de acciones y áreas de comportamiento, hasta llegar a poder establecer un patrón representativo de la mayoría de la población o la totalidad de la muestra.

Existen en la actualidad dos grandes áreas distintivas entre otras que nos permiten detectar cuando una operación web se está realizando por un individuo humano y estas son: basadas en características fisiológicas (Biométrico) y características del comportamiento (Accionar)

El reconocimiento biométrico o características fisiológicas se refiere al reconocimiento automatizado de individuos basado en sus características fisiológicas (huellas dactilares, ojos, rostro, etc.), siendo estos en general técnicas menos invasivas que el resto (Desbloqueo del teléfono móvil mediante el rostro, etc.) (Alejandro Acien, 2021).

Cuando hablamos de características del comportamiento (pulsación de teclas, la forma de mover el ratón, etc.), nos referimos a los rasgos que revelan comportamientos y accionares distintivos de los usuarios humanos cuando interactúan con los dispositivos. (Alejandro Acien, 2021)

En nuestro caso vamos a estar enfocados en las características del comportamiento, no vamos a centrarnos en explicar en detalle a que se dedica el Análisis de conducta, pero si es importante mencionar que, a grandes rasgos, este análisis de conducta arroja (y por obvias razones ya que son las formas de ingreso o manipulación del computador) dos tipos específicos de subáreas a saber: Análisis Conductual del Ratón y Análisis Conductual del Teclado.

Por lo tanto, para poder realizar una herramienta que nos permita realizar pruebas automatizadas lo más cercanas al comportamiento humano, lo primero que debemos lograr es entender “como” se comporta un humano, reduciendo nuestro campo de estudio a estas dos grandes áreas en la medida que interactúa con los dispositivos Ratón y Teclado al momento de realizar una navegación web.

2.2.1. Análisis Conductual: Ratón

Uno de los principales modos de interactuar con un computador es el uso del Ratón, dada las características propias del dispositivo tendremos 3 áreas a analizar:

- Movimientos del Mouse/Ratón
- Velocidad del Cursor
- Distancia o cantidad de movimientos directos.

La mayoría de los estudios arrojan que el movimiento del ratón persigue una suerte de puntero de atención, es decir, donde está el puntero es donde se centra la visión del usuario. Esto habla de una transmutación de la intención humana plasmada en un accionar determinado forzando estos dispositivos a que sigan un comportamiento puntual. Es decir, tomemos un ejemplo mundano, si estamos pasando una situación de stress o de nervios lo usual es que el movimiento del ratón, así como su trayectoria sea claramente veloz y mucho más rápido que si por ejemplo estamos frente a una lectura de ocio.

2.2.1.1. Movimientos del Ratón

Las trayectorias del ratón son completamente diferentes para los humanos y para los bots. Los robots suelen mover el ratón de la manera más "efectiva", es decir, evitando cualquier giro o curva o movimiento extra innecesario. Para los humanos es casi imposible mover el ratón de la misma manera "directa".

Así mismo, investigaciones recientes han demostrado que los movimientos del cursor se correlacionan con la mirada, por lo que pueden ser un indicador eficaz de la atención del usuario (Jeff Huang).

Estos estudios arrojan como resultados que estos dispositivos se convierten en una especie de extensión de las intenciones de la mente humana con lo cual resulta en una prueba más de que la trayectoria necesariamente sigue una cierta irregularidad, así como nuestros pensamientos ya que, dicho de otro modo, trasladamos nuestras intenciones a estos elementos, dando como resultado, estos "movimientos" irregulares.

Las siguientes imágenes muestran la trayectoria del ratón frente a un caso de decisión, las mismas son un extracto del experimento de Spivey, indistintamente de los resultados obtenidos en este estudio lo que vale la pena destacar es esa tendencia o patrón irregular humano al momento de mover el mouse.

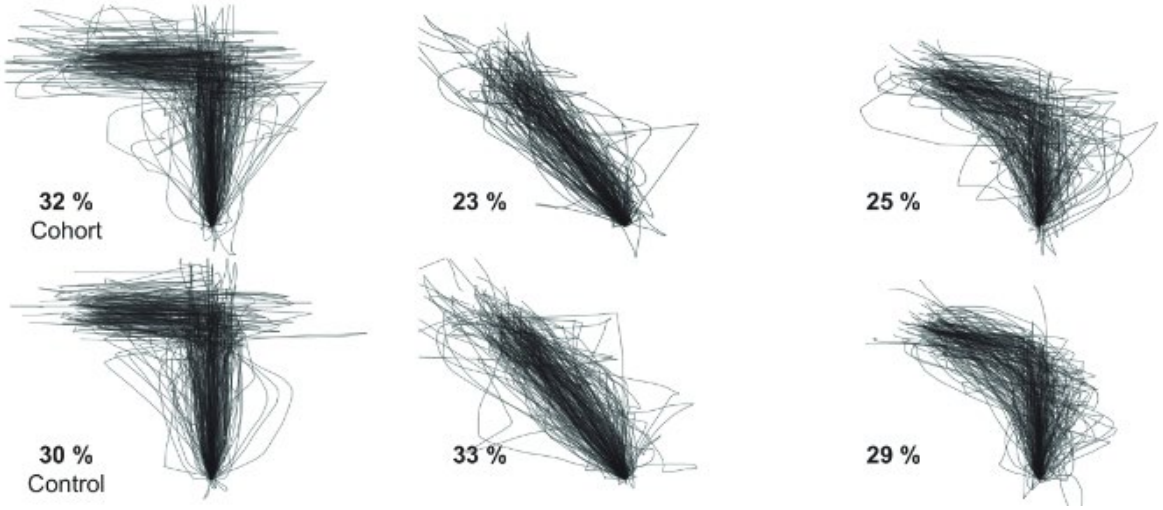


Ilustración 8 - Agrupaciones de trayectorias extraídas de los datos del experimento de Spivey et al. (2005). (Dirk U. Wulff)

Como puede observarse, las líneas no son exactas, sino que persiguen una cierta curvatura la cual, repitiendo el experimento múltiples veces no se sobrepone a las anteriores, sino que agrega más y más líneas lo cual claramente decanta en este comportamiento irregular.

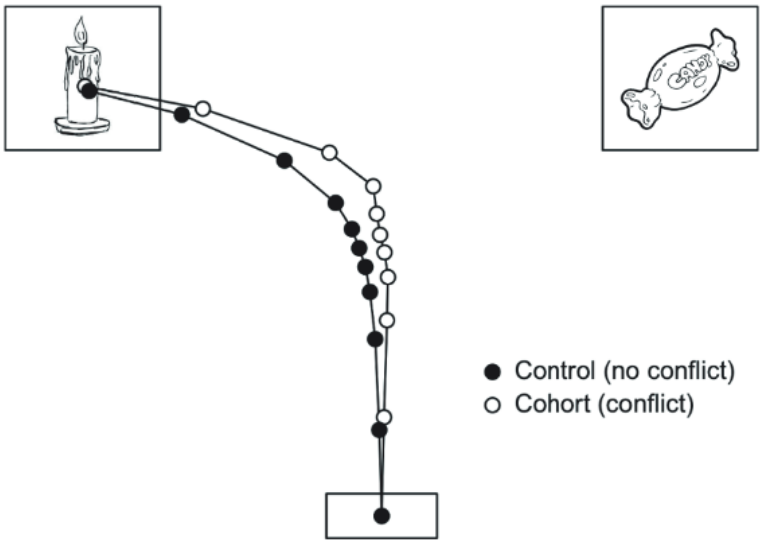


Ilustración 9 - Diseño y resultados de Spivey et al. (2005). Las líneas representan las trayectorias medias para las condiciones de control y cohorte cuando el objetivo estaba en el lado izquierdo. Los puntos marcan 10 puntos temporales equidistantes. (Dirk U. Wulff)

En la imagen anterior y solamente para contextualizar, se trata de una de las etapas del experimento de Spivey, en su estudio, les pedía a los participantes que utilizaran el ratón para seleccionar una de las dos imágenes presentadas en las esquinas superiores en base a una palabra que se les hacía oír por unos auriculares.

Aquí tenemos otro ejemplo, pero esta vez de autoría propia, la siguiente imagen capturada y generada con el software iographica (Zenkov, n.d.) muestra una gráfica del movimiento con los clics incluidos de mi mouse/ratón generado luego de 10 navegaciones en diferentes sitios de mi interés personal con búsquedas en Google incluidas.

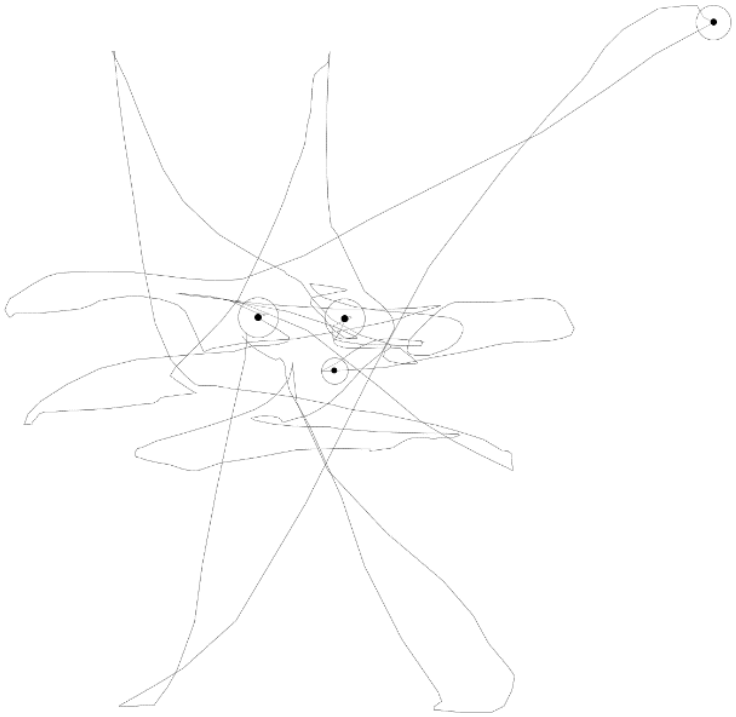


Ilustración 10 - Ilustración de los Movimientos y Clics del Ratón con iographica. Fuente: Elaboración Propia

Por lo anteriormente mencionado podemos concluir que un ser humano moviendo el ratón raramente se mueve en línea recta, de hecho, es casi imposible, por ende, se presentan variaciones significativas en la dirección elegida de traslado.

Desde la perspectiva de un programa que recibe información sobre el movimiento del ratón, un mouse/ratón en movimiento se representa como una serie de actualizaciones de ubicación a intervalos relativamente fijos, normalmente de unos 10 ms (100 Hz). Cualquiera de estas dos velocidades es suficiente para que el movimiento normal del ratón parezca fluido. Cada actualización de ubicación proporcionará sólo la ubicación del cursor (en píxeles) en el momento de la actualización. Aunque la información proporcionada por las actualizaciones del ratón es limitada, para cada paso es posible calcular:

- Tiempo transcurrido desde el último paso
- Cambio de posición desde el último paso
- A partir de la distancia y el tiempo, la velocidad

Todas estas métricas podrían recogerse fácilmente y compararse con las distribuciones típicas de un ser humano para identificar movimientos automatizados. Las pruebas empíricas sugieren que los humanos mueven el ratón entre 5 y 10 px/ms, pero el valor exacto aquí depende de la resolución de la pantalla, los PPP de la pantalla, la configuración del ratón del

usuario y la tarea exacta que se esté realizando. Basta con decir que la velocidad del ratón debe ser una entrada para cualquier algoritmo de movimiento del ratón, y debe decidirse caso por caso.

Una interpolación lineal simple que satisfaga cualquier suposición de velocidad humana del ratón sería fácilmente identificable como no humana cuando se observe una distribución angular[10], ya que se encontrarían segmentos con ángulos discretos para cada línea. La reproducción de un conjunto finito de movimientos reales del ratón también sería identificable observando estas distribuciones angulares. Por lo tanto, es fundamental contar con algún algoritmo aleatorio que cree una distribución angular aceptable de cada uno de los pasos, sin dejar de dirigirse hacia un destino o unas coordenadas determinadas en el sitio web. La distribución temporal es relativamente sencilla de satisfacer, pero la generación de trayectorias con cualidades suficientemente humanas es la parte más compleja y es para donde utilizaremos el algoritmo de WindMouse (Land, WindMouse, an algorithm for generating human-like mouse motion, 2021).

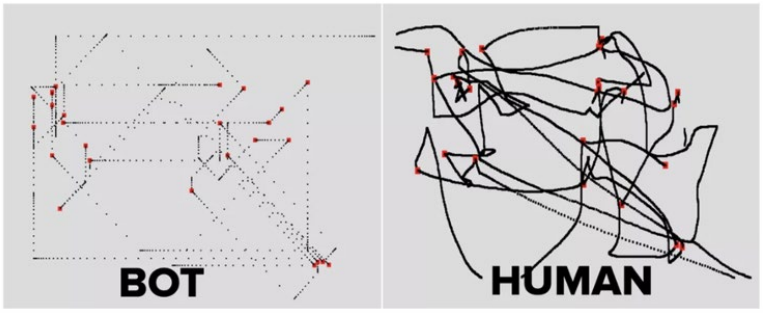


Ilustración 11 - Comparativa de las trayectorias del mouse/ratón entre un bot y un humano. Fuente: https://securityboulevard.com/2021/07/how-attackers-use-request-bots-to-bypass-your-bot-mitigation-solution-2/

De esto obtenemos el primer parámetro a respetar si queremos dar una simulación considerable:

"Tenemos que respetar movimientos y desplazamientos del ratón que no sigan una suerte de línea "recta", estos desplazamientos digamos del punto 1 al 2 dentro de un sitio web tienen que ser como poco irregulares y curvilíneos"

[10] Función de distribución de los ángulos que forman con una dirección prefijada las trayectorias de las partículas emitidas. Fuente: https://www.sne.es/diccionario-nuclear/distribucion-angular/

En términos técnicos hablamos de introducir una suerte de curvatura utilizando algún algoritmo que nos permita realizar movimientos curvilíneos obviando así la ruta optima de un punto de inicio a un punto final.

2.2.1.2. Velocidad y aceleración del Cursor

Según varios estudios, el rango de la velocidad de los movimientos del cursor realizados por los humanos cae dentro de distribuciones de probabilidad normales, como se muestra a continuación.

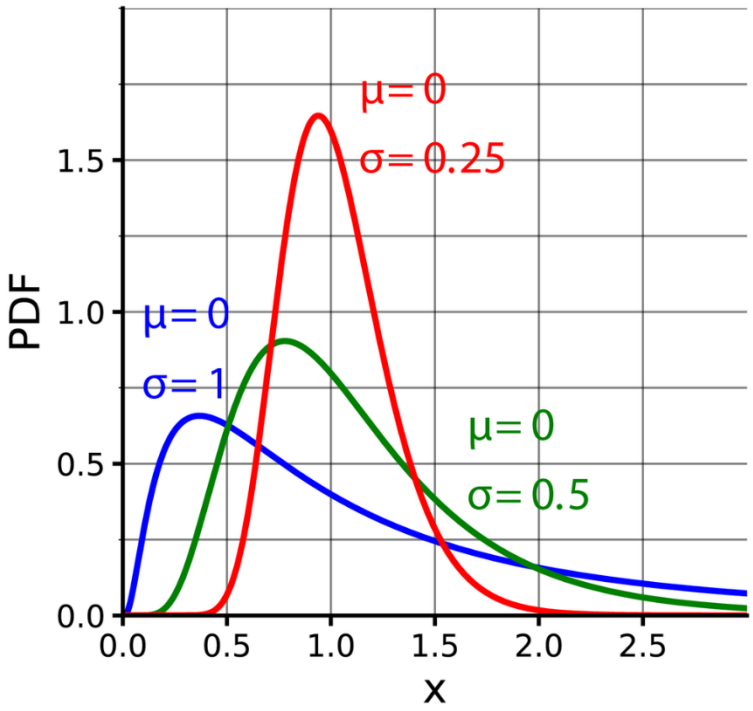


Ilustración 12 - Distribución Normal aplicada al comportamiento humano. Fuente:
https://en.wikipedia.org/wiki/Log-normal_distribution#/media/File:Log-normal-pdfs.png

Si tomamos una velocidad X cuya aceleración es Y, cuando utilizamos sistemas automatizados estos valores se mantienen constantes mientras se realizan las operaciones para dirigirse del punto P1 al punto P2, dibujando como resultante una distribución lineal.

Basándonos en diferentes estudios tales como “How design factors of the mouse-tracking procedure impact the inference from action to cognition” (Scherbaum, 2019) y “Symbiotic Interaction” (Luciano Gamberini, 2016), si bien en ellos se estudian cuestiones diferentes todos ellos coinciden y demuestran una hipótesis claramente esperada la cual postula que, la

velocidad y la aceleración del puntero del mouse/ratón basados en una trayectoria determinada en los seres humanos persigue un patrón irregular, no constante y variable, tomados en perspectiva todos ellos caen claramente en una distribución normal pero lo importante aquí es el segundo parámetro a valorar:

"Las velocidades y aceleraciones al mover un mouse/ratón de un punto A hacia un punto B tienen que ser ejecutadas con retardos variables de manera que podamos simular una variabilidad considerable en términos de velocidad y aceleración."

Esto en términos técnicos se traduce como introducir retardos aleatorios, operaciones de relleno o micro/interoperaciones en las ordenes de movimiento del ratón.

2.2.1.3. Distancia o cantidad de movimientos directos

Otra métrica es comprobar cuántos movimientos realiza un usuario para desplazarse a un determinado botón o incluso una sección vacía del sitio web. Un bot suele estar programado para llegar a nuevas coordenadas "saltando" directamente a esas coordenadas desde las coordenadas actuales, pero un humano necesita un mayor número de desplazamientos del ratón, más cortos y menos directos, para moverse a las nuevas coordenadas desde las coordenadas actuales.

En otras palabras, la combinación de manos/cerebro humano desplaza el ratón en zig-zag, un pequeño paso cada vez, acercándose gradualmente a la nueva ubicación. En cambio, un bot tiende a ir directamente a la nueva ubicación, y en muchos menos desplazamientos, lo que significa que el seguimiento de tales movimientos es perfecto para detectar bots. En otras palabras, si tomamos el concepto de "receta" mencionado anteriormente, podemos decir que un sistema automatizado por defecto realiza una única operación sin intermedios u "operaciones de relleno" intermedias, su "receta" contiene una única operación. Esto claramente se aplica a cierto tipo de operaciones, en las cuales su naturaleza permita agregar variaciones entre su comienzo y su final, es decir, tomando como ejemplo hacer clic en un botón, su mera naturaleza no admite realizar algo desde que se comienza la operación hasta que se finaliza, pero si hablamos del desplazamiento del mouse/ratón, sí que podríamos agregar operaciones hasta que el punto llegue al punto de destino.

Dado:

- O_x: toda operación que pueda realizarse en una navegación web
- O_{ti}: punto o momento de inicio de O_x
- O_{tf}: punto o momento de final de O_x
- O_{ox}: todas las micro operaciones que se suceden entre O_{ti} y O_{tf}

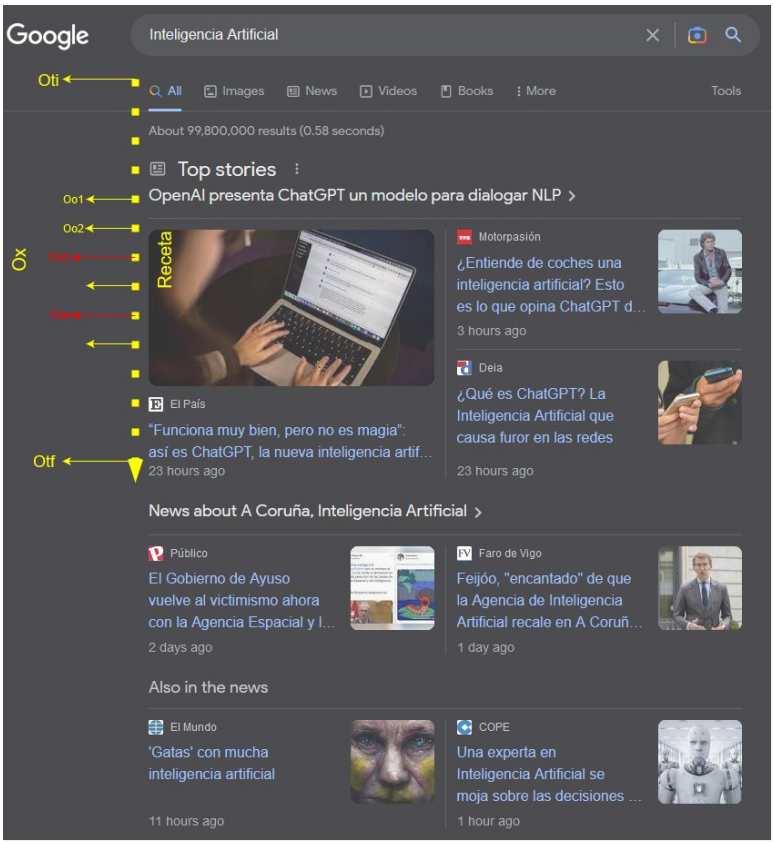


Ilustración 13 - Operaciones Vs Ejemplo de Pagina Web. Fuente: Elaboración Propia

Ejemplo:

- O_x: Operación de desplazamiento del ratón
- O_{ti}: Momento de inicio del desplazamiento
- O_{tf}: Momento de fin del desplazamiento
- O_{ox}: micro/interoperaciones
    - O_{o1}: operación de "espera"
    - O_{o2}: operación de "presionar la rueda media" del ratón

De esto se obtiene el tercer y último parámetro para tener en cuenta dentro de las cuestiones a considerar en este dispositivo de entrada entendiendo este como:

"Toda operación automatizada (siempre y cuando sea posible) tiene que contener micro/interoperaciones que doten de variabilidad a la operación padre"

A nivel de aplicación esta regla se puede conseguir agregando "pasos" extras u micro/interoperaciones a una operación en cuestión.

2.2.2. Análisis Conductual: Teclado

El seguimiento de la dinámica de pulsación de teclas también es un parámetro para analizar de cara a obtener un patrón de comportamiento (o quizás la ausencia de él), en este caso, la idea es capturar y analizar datos sobre:

- Duración o el tiempo que se pulsa una tecla
- Intervalo de salida/llegada o el tiempo transcurrido entre la pulsación de una tecla y la siguiente

Esto en comparación con un sistema automatizado arrojan diferencias radicales tales como que la duración es constante al igual que los intervalos de salida/llegada.

Si bien es cierto que existen múltiples trabajos que explican y estudian los fenómenos humanos aplicados a cómo y de qué manera usamos el teclado, por las características propias de este dispositivo, por su simplicidad conceptual y el fácil entendimiento de su funcionamiento, así como su ejemplificación, no vamos a desgranar estos conceptos en el trabajo y nos limitaremos a exponer las dos mayores cuestiones para tener en cuenta las cuales son:

- Presionar una única tecla carece de estudio, ni para un humano ni para un bot tiene sentido en un entorno de navegación web (se excluyen los juegos) tener para una tecla T una duración D media que este en un orden mayor del segundo.
- Un ser humano al momento de tener que escribir palabras, frases, párrafos y demás mantiene el orden de duración D mencionado en el punto anterior pero tomado como un promedio, es decir, la duración D difícilmente sea constante e invariable, dicho de otro modo, aunque se quiera, un ser humano no puede sostener una duración D invariable durante todo el proceso de escritura y para todas las teclas que participan del proceso.

- Lo mismo pasa con los intervalos de salida/llegada I, por más que se intente, difícilmente un ser humano pueda mantener una tendencia invariable, un valor de intervalo I constante durante todo el proceso de escritura.

Si bien es cierto que los valores de duración (D) e intervalo (I) en los seres humanos persiguen una cierta distribución estadística y una suerte de tendencia si analizamos los modos de escritura, es decir, existen estudios que pueden otorgarnos el patrón de comportamiento humano al momento de la escritura, simplemente nos basta con destacar que si bien no son completamente variables ciertamente no son para nada constantes.

De estos postulados se obtienen los parámetros a respetar dentro el análisis conductual del teclado:

"Toda operación automatizada debe tener unos valores de duración D e intervalos I variables desde su concepción"

A nivel de aplicación esta regla se puede conseguir del mismo modo que vimos para la sección "Distancia o cantidad de movimientos directos", es decir agregando "pasos" extras u micro/intra-operaciones a una operación de escritura en cuestión, pero la salvedad es que en este caso las micro/intra-operaciones son simplemente operaciones de "retardo".

2.3. Contexto Técnico

En esta sección vamos a hablar de los conceptos y herramientas técnicas (sin entrar en detalles de la implementación) a utilizar en el trabajo a fin de dotar de una contextualización en términos teóricos de los diferentes temas que se van a abordar en el actual trabajo. En esta primera parte nos centraremos en las herramientas y/o frameworks de terceros que vamos a utilizar.

2.3.1. Kafka

Apache Kafka es una plataforma distribuida para la transmisión de datos que permite no solo publicar, almacenar y procesar flujos de eventos de forma inmediata, sino también suscribirse a ellos. Está diseñada para administrar los flujos de datos de varias fuentes y enviarlos a distintos usuarios. En pocas palabras, transfiere cantidades enormes de datos, no solo desde

el punto A hasta el B, sino también del punto A al Z y a cualquier otro lugar que necesite, y todo al mismo tiempo (RedHat, 2022).

Kafka tiene tres componentes fundamentales: Los productores, los consumidores y los brokers.

- Los productores son los encargados de escribir mensajes en Kafka.
- Los consumidores los pueden leer y procesar.
- Los brokers son los nodos que forman parte del clúster de Kafka y almacenan y distribuyen los datos.

Los mensajes de Kafka se almacenan en tópicos, y se reparten en particiones para hacer el sistema escalable y tolerante a fallos (Datademia, s.f.).

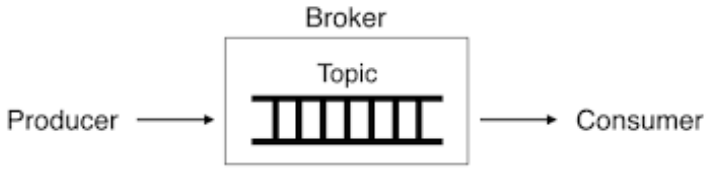


Ilustración 14 - Resumen de Conceptos Apacha Kafka. Fuente: https://datademia.es/blog/que-es-apache-kafka

2.3.2. Selenium Web Driver

Entorno de pruebas que se utiliza para comprobar si el software que se está desarrollando funciona correctamente. Esta herramienta permite: grabar, editar y depurar casos de pruebas que se pueden automatizar, así como editar acciones o crearlas desde cero. También ayuda mucho en las pruebas de regresión porque consigue pruebas automatizadas que luego se pueden reutilizar cuando se necesite. Algunas de sus características son (Digital55, 2019):

- Las acciones serán ejecutadas punto a punto, si así se considera.
- A la hora de escribir el código tiene la opción de autocompletar.
- Se puede referenciar a objetos DOM: nombre, ID o con XPath.
- Ejecutar test complejos que ahorran muchas horas de trabajo.
- Gran depuración y puntos de verificación

- Almacenamiento en varios formatos las pruebas realizadas

2.3.3. Docker Compose

Docker Compose es una herramienta para definir y ejecutar aplicaciones Docker de varios contenedores. En Compose, se usa un archivo YAML para configurar los servicios de la aplicación. Después, con un solo comando, se crean y se inician todos los servicios de la configuración. Dentro de los elementos que caracterizan a la herramienta de Docker Compose (o docker-compose), se encuentra su capacidad para contener una diversidad de entornos dentro de una misma máquina de host, así como su labor de mantener la preservación de los datos de volumen una vez se crean los diferentes contenedores (Keepcoding, 2022).

2.3.4. Worker Services

Es básicamente un conjunto de operaciones que se realizan en segundo plano, en general no requieren de una intervención humana y están pensados para ejecutar operaciones que no necesitan de una interfaz o similar. En un Worker Service no se espera que se genere una salida por pantalla ni nada parecido. Algunos ejemplos de ellos son:

- Realizar operaciones repetitivas en segundo plano dentro de un horario determinado
- Actuar frente a alguna situación o evento que no requiera manipulación humana

Alguna de las fuentes que explican su funcionamiento en detalle son:

- https://learn.microsoft.com/es-es/dotnet/core/extensions/workers
- https://www.roundthecode.com/dotnet/use-dotnet-worker-service-run-background-services
- https://www.arsys.es/blog/service-worker

2.4. Algoritmia

Se entiende por algoritmo a un conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas (Google, s.f.).

Si miramos la definición de la Real Academia Española, nos dice que la definición de algoritmo es: “Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema”.

Si bien los algoritmos generalmente se asocian al ámbito matemático, no necesariamente implica que sean exclusivos de esta área. Se puede entender un algoritmo como una secuencia de pasos finitos bien definidos que resuelven un problema.

Desde el punto de vista informático un algoritmo es cualquier procedimiento computacional bien definido que parte de un estado inicial y un valor o un conjunto de valores de entrada, a los cuales se les aplica una secuencia de pasos computacionales finitos, produciendo una salida o solución. Se puede considerar al algoritmo como una herramienta para resolver un cálculo computacional bien especificado (Garate, s.f.).

2.4.1. Curva de Bezier

Es una serie de fórmulas matemáticas para describir dibujos y curvas que se basan en ecuaciones polinómicas. Se desarrolló en 1960 para el trazado de dibujos técnicos. El método de descripción de la curva, denominada con ese nombre, en honor a Pierre Bezier. Se comenzó a utilizar con éxito en los programas de CAD. Estas curvas son conocidas como trazados vectoriales, que ahora dan uso los diseñadores. Lo primero que podemos notar, es que estas curvas deben tener un punto de comienzo y un punto de final. además de ello, se tiene un tercer y cuarto punto, los cuales son llamados los puntos de control y manejadores (Mott Glosario, s.f.).

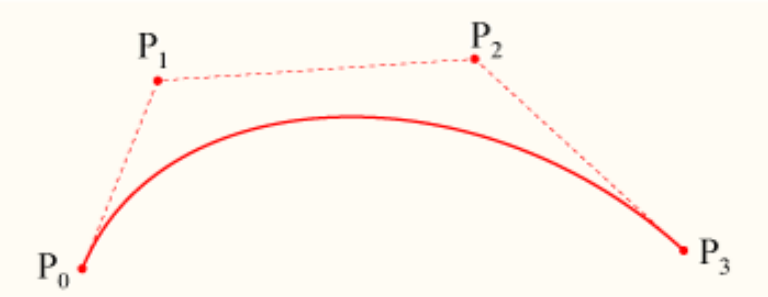


Ilustración 15 - Construcción de una curva de Bézier. Fuente:
https://es.wikipedia.org/wiki/Curva_de_B%C3%A9zier

2.4.2. Windmouse

Es un tipo de algoritmo inspirado en los postulados físicos en donde el cursor se modela como un objeto con cierta inercia (masa) sobre el que actúan dos fuerzas (Land, WindMouse, an algorithm for generating human-like mouse motion, 2021):

- La gravedad, que es constante en magnitud (un parámetro configurable) y siempre apunta hacia el destino final.
- El viento, que ejerce una fuerza aleatoria en una dirección aleatoria, y cambia suavemente tanto en magnitud como en dirección con el tiempo.

2.5. Patrones

Se dice de patrones a soluciones habituales a problemas comunes en el diseño de software. Cada patrón es como un plano que se puede personalizar para resolver un problema de diseño particular del código. Los patrones de diseño varían en su complejidad, nivel de detalle y escala de aplicabilidad. Además, pueden clasificarse por su propósito y dividirse en tres grupos a saber (Refactoring, s.f.).:

- Los patrones creacionales proporcionan mecanismos de creación de objetos que incrementan la flexibilidad y la reutilización de código existente.
- Los patrones estructurales explican cómo ensamblar objetos y clases en estructuras más grandes a la vez que se mantiene la flexibilidad y eficiencia de la estructura.
- Los patrones de comportamiento se encargan de una comunicación efectiva y la asignación de responsabilidades entre objetos.

2.5.1. Builder

Patrón de diseño creacional que nos permite construir objetos complejos paso a paso. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción (Refactoring, s.f.).

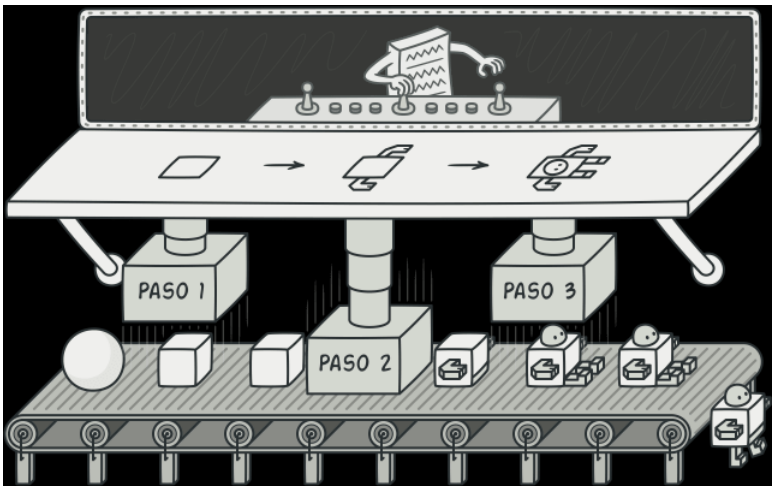


Ilustración 16 - Ejemplificación Patrón Builder. Fuente: https://refactoring.guru/es/design-patterns/builder

2.5.2. Factory

Es un tipo de patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán (Refactoring, s.f.).

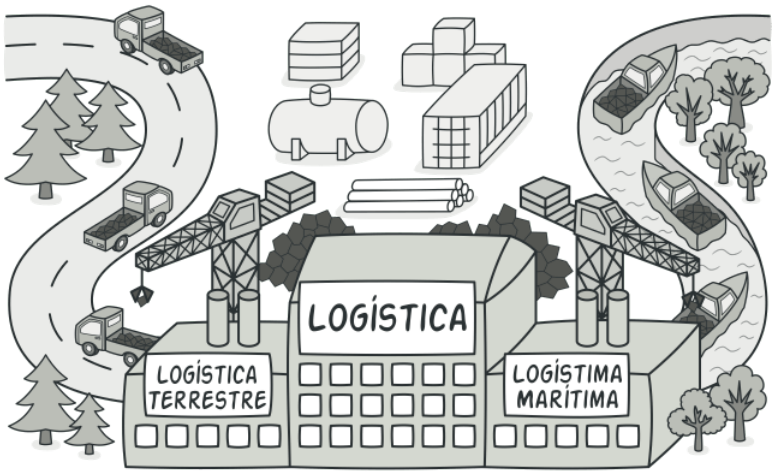


Ilustración 17 - Ejemplificación Patrón Factory. Fuente: https://refactoring.guru/es/design-patterns/factory-method

2.6. Pruebas Automatizadas

Las pruebas automatizadas enfocadas al área de software son esencialmente la ejecución consecutiva de pasos de manera automática en pos de un objetivo, como toda prueba su búsqueda principal es encontrar una falla, es decir uno de los principales objetivos es detectar

fallas sin la intervención humana, evitando errores que en algunos casos las personas no pueden percibir con la misma eficacia. Así mismo, permite ejecutar un gran número de escenarios la cantidad de veces que sea necesario, optimizando el tiempo en la identificación de errores.

Su comprensión es bastante clara desde su inicio, básicamente simulan los flujos que realizaría un usuario al utilizar el sistema, para así asegurar que cumpla con las exigencias para las que fue desarrollado el mismo.

2.6.1. Beneficios

Existen multiplicidad de beneficios o ventajas que podemos mencionar al momento de referirnos o de hablar del concepto de pruebas, pero a modo de resumen nos centraremos en solo algunos.

- Mayor precisión al diagnosticar fallas y mejoras: Las pruebas automatizadas entregan evidencia de cómo funcionaría el software en cada una de las situaciones que los usuarios lo usarían. Al utilizar distintos escenarios automatizados de prueba es posible realizar una mayor cobertura y ser muy certero en cada uno de los flujos que pudieran representar un problema (Verity, 2022).
- Efectividad en tiempo y costo: Las pruebas automatizadas de software optimizan el tiempo de ejecución de pruebas en comparación con las manuales. Si bien tienen un costo inicial, en el tiempo dicho valor se amortigua de manera idónea respecto a las ganancias y beneficios que otorga automatizar dichos procesos. Hacer estas pruebas de forma manual es dispendioso, costoso y consume un tiempo considerable en comparación a las pruebas automatizadas que pueden hacerse de manera inmediata y mucho más rápido (Verity, 2022).
- Aseguramiento de la calidad en el desarrollo del software: Las pruebas de software deben ser realizadas durante distintas etapas de desarrollo para asegurar que todos los aspectos inherentes al sistema tengan la calidad esperada. Cada vez que se hacen algunas modificaciones en el sistema, las pruebas automatizadas de regresión se convierten en el apoyo a las pruebas manuales, ya que tienen en cuenta todas las

funcionalidades y requerimientos que a lo mejor han sido modificados anteriormente (Verity, 2022).

3. Contextualización

Ya habiendo hablado en términos generales de los diferentes conceptos y herramientas que vamos a manejar durante el trabajo, ahora es momento de centrarnos en las cuestiones más detalladas y arquitectónicas de la solución planteada, así como de la metodología utilizada, la cual será el primer tema que abordar ya que nos define un marco de ejecución formal y dota de orden a como fue realizado el trabajo.

3.1. Metodología

Dada la naturaleza de la aplicación y por sobre todo como surgieron las necesidades, podríamos decir que el proceso metodológico fue una mixtura entre SCRUM[11] y Waterfall[12], más adelante haremos una breve explicación de ambas metodologías, pero antes vale la pena destacar la razón de esta mixtura.

Inicialmente los requerimientos fueron descubiertos personalmente de uno en uno, es decir, la idea o el objetivo inicial fue mutando a medida que iban surgiendo más y más dudas y/o preguntas al respecto. Por lo tanto, la idea de Waterfall se asocia al concepto de que los requerimientos fueron tomados uno en uno en una etapa inicial de relevamiento y su desarrollo tanto de pruebas de concepto como el desarrollo final, no comenzó hasta no tener formada una idea concreta, así como su comprobación inicial de que la solución podía ser llevada a cabo desde el punto de vista técnico. Esta comprobación fue realizada principalmente haciendo una suerte de revisión del estado del arte, pero técnico, buscando herramientas que me permitan ir satisfaciendo estos objetivos tentativos en un principio y más concretos al final.

[11] Proceso para llevar a cabo un conjunto de tareas de forma regular con el objetivo principal de trabajar de manera colaborativa, es decir, para fomentar el trabajo en equipo (APD, 2022).
[12] También denominada en “cascada”, es el método que se ha utilizado tradicionalmente. Consiste en desarrollar un proyecto de forma secuencial, comenzando con las fases de análisis y diseño y terminando con las de testeo y puesta en producción (deloitte, s.f.).

Luego de tener una suerte de borrador de la idea o del objetivo, comenzó una fase más de pruebas de concepto, subdividiendo esta idea inicial en partes exclusivamente técnicas y creando pequeñas pruebas de concepto, las cuales a su vez volvieron a modificar ese objetivo y sirvieron como retroalimentación que permitió darle una noción más formal al objetivo principal, tanto como en su concreción como en su alcance, modificándolo claramente.

La parte SCRUM, aparece al final, es decir, una vez que tenía la idea concreta y ya formada, apuntalada por las pequeñas pruebas de concepto, las cuales dotaron de veracidad al objetivo final, el siguiente paso fue preparar una especie de lista de tareas a realizar y en este punto intervino puntualmente SCRUM, ayudándome a entregar o finalizar pequeños conjuntos de funcionabilidades que al final de cada periodo de trabajo (predefinido con una duración de 10 días) me acercaban más y más al objetivo final. Vale destacar que opte por SCRUM ya que, entre el trabajo y la vida diaria, el tiempo debía ser organizado al detalle en pos de poder cumplir con los entregables y sus fechas.

Dicho esto, vamos a explicar las cuestiones generales de cada una de las metodologías y al final, denotar que sección de cada una de ellas fueron utilizadas durante el trabajo

3.1.1. Scrum

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. El marco de trabajo de scrum es heurístico. Se basa en el aprendizaje continuo y en la adaptación a los factores fluctuantes. Reconoce que el equipo no lo sabe todo al inicio de un proyecto y evolucionará a través de la experiencia. Scrum está estructurado para ayudar a los equipos a adaptarse de forma natural a las condiciones cambiantes y a los requisitos de los usuarios, con el cambio de prioridades integrado en el proceso y ciclos de lanzamiento breves para que tu equipo pueda aprender y mejorar constantemente (Drumond, s.f.). Es importante destacar que en SCRUM se utilizan periodos de tiempo predefinidos llamados Sprints, cada uno de ellos tiene un inicio, duración y final concretos en donde, según sea el momento, se deben llevar a cabo ciertas ceremonias o reuniones y se deben entregar ciertos artefactos o entregables.

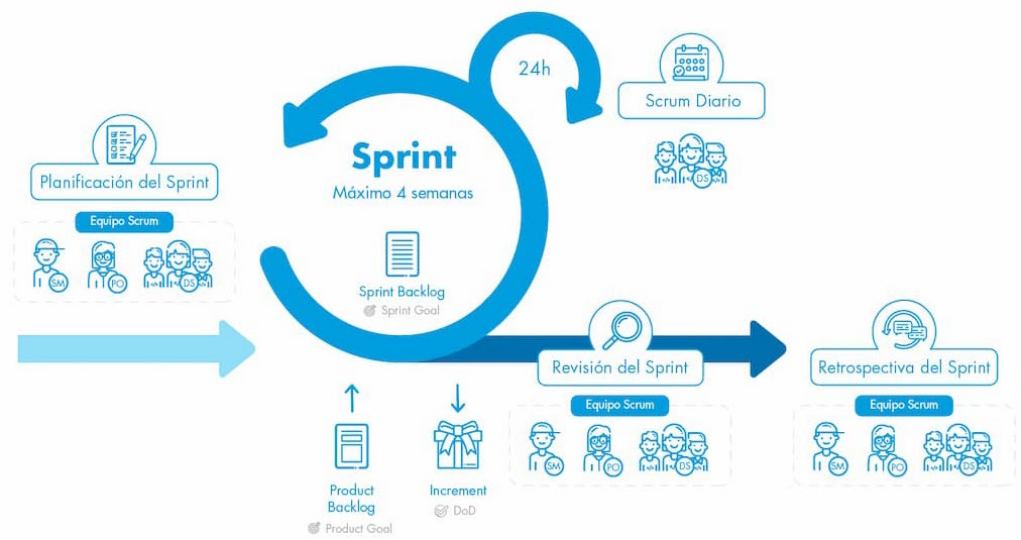


Ilustración 18 - Metodología SCRUM. Fuente: https://scroolling.net/digital/scrum-503

3.1.1.1. Ceremonias

Las ceremonias son reuniones en donde están predefinidas tanto su duración y objetivo como los participantes, estas sirven a un objetivo parcial del objetivo global de la metodología que es el de entregar trabajo. Estas ceremonias son:

- PI Planning: reunión en donde se agrupan todas las tareas que se piensan que se pueden entregar en un periodo más extenso de tiempo que un simple Sprint, usualmente incluye lo que debe hacerse dentro de un conjunto de 6 Sprints
- Sprint Planning: reunión en donde se vuelve a revisar las tareas que se piensan que se pueden entregar en el Sprint inmediato por venir, la mayor diferencia con la ceremonia de PI Planning, es que, en este caso, se evalúan las tareas a realizar en un único Sprint
- Refinement: Reunión en donde se analizan las tareas que van a venir en el siguiente Sprint y se relevan que tengan todo lo necesario para llevarse a cabo.
- Daily: Es una reunión diaria en donde los miembros comentan que hicieron el día anterior, que van a hacer el día en curso y si tienen algún bloqueo.
- Demo: Reunión en donde se le presenta al Cliente o Interesados en el proyecto, una prueba de todas las tareas realizadas durante el Sprint pasado, estas tareas deberían haber sido previamente estipuladas en la Sprint Planning
- Retrospective: reunión en donde los participantes comparten sus ideas y sensaciones respecto de las cuestiones que creen que se realizaron correctamente (en pos de

continuar haciéndolas) y las que se hicieron de mala forma o pueden mejorarse, de manera que se obtengan planes de acción de mejoras.

3.1.1.2. Roles

El equipo de scrum se compone de tres cargos específicos: el propietario del producto, el experto en scrum y el equipo de desarrollo. Y, puesto que los equipos de scrum son interdisciplinares, el equipo de desarrollo está formado por evaluadores, diseñadores, especialistas en experiencia de usuario e ingenieros de operaciones, además de desarrolladores (Drumond, s.f.).

3.1.1.3. Ventajas

Las ventajas aquí mencionadas son simplemente algunas y a carácter meramente informativo.

- Claridad en los objetivos y alcances: Al inicio del proyecto el cliente establece los requisitos, prioridades y la fecha en la que requiere que esté completado, luego de manera regular se realizan entregas para comprobar que efectivamente se estén cumpliendo completamente los objetivos (pueden ser con demos funcionales) y se brinda feedback a los equipos para realizar ajustes si se requieren.
- Flexibilidad: La metodología es flexible, adaptable y con alta capacidad de reacción ante los cambios de requerimientos generados por el cliente, ya que, de requerirse algún ajuste, este sólo se aplica a una pequeña parte y no al proyecto completo.
- Mayor control de imprevistos: El cliente conoce o comienza a usar las funcionalidades más importantes del proyecto o de su producto antes que esté finalizado por completo, esto reduce las probabilidades de sorpresas.
- Equipos altamente productivos:
- Predictibilidad
- Facilidad de implementación: la metodología SCRUM es simple de aprender e implementar

3.1.1.4. Desventajas

- Se aplica a equipos reducidos: Scrum es exitosa cuando se trabaja con grupos de pocos colaboradores (wearedrew, 2019).
- Requiere una exhaustiva definición de las tareas y sus plazos: Scrum funciona correctamente cuando tanto las tareas como el tiempo en que se ejecutará cada una se encuentran definidos. La esencia de esta metodología reside en la división del trabajo de cada etapa y de sus tareas específicas (wearedrew, 2019).
- Requiere de perfiles senior en su aplicación: Quienes aplican Scrum cuentan con una alta cualificación, por lo que no es una modalidad de gestión propia de grupos junior o que estén en formación (wearedrew, 2019).
- Difícil escalabilidad: Aplicar un enfoque Scrum para grandes proyectos se establece un reto ya que puede fallar la coordinación precisa, por lo que no garantiza que sea escalable a largo plazo (wearedrew, 2019).
- Puede necesitar de transformaciones dentro de la organización: En ocasiones, para trabajar con Scrum la empresa debe pasar por ciertas transformaciones organizativas en sus departamentos y áreas. Es la empresa quien debe gestionar y organizarse para que las colaboraciones sean exitosas (wearedrew, 2019).
- No se integra fácilmente con enfoque clásico de gestión de proyectos: El enfoque de Scrum no suele ser el adecuado para proyectos que requieren previsibilidad y un plan bien definido (wearedrew, 2019).

3.1.2. Waterfall

La metodología waterfall también es conocida como modelo de desarrollo en cascada. Consiste en el desarrollo de un proyecto de manera secuencial. Se redacta una lista de requisitos que el producto final debería tener. Tras esta intervención en la fase inicial por parte de la organización, no sería necesario que volviera a participar en el proceso de diseño. Una vez el proyecto queda en manos del equipo de desarrollo, este realizaría las tareas de manera secuencial. Una tarea no dará comienzo hasta que no se haya finalizado la inmediatamente anterior (ticportal, 2022). Quizás el punto más destacable es que las fases se suceden en secuencia y no en paralelo, es decir nada se comienza hasta no tener una lista de requisitos, a su vez, los requisitos raramente sufren cambios y el cliente no se ve involucrado hasta la fase final, lo cual claramente presenta ventajas y desventajas que veremos más adelante.

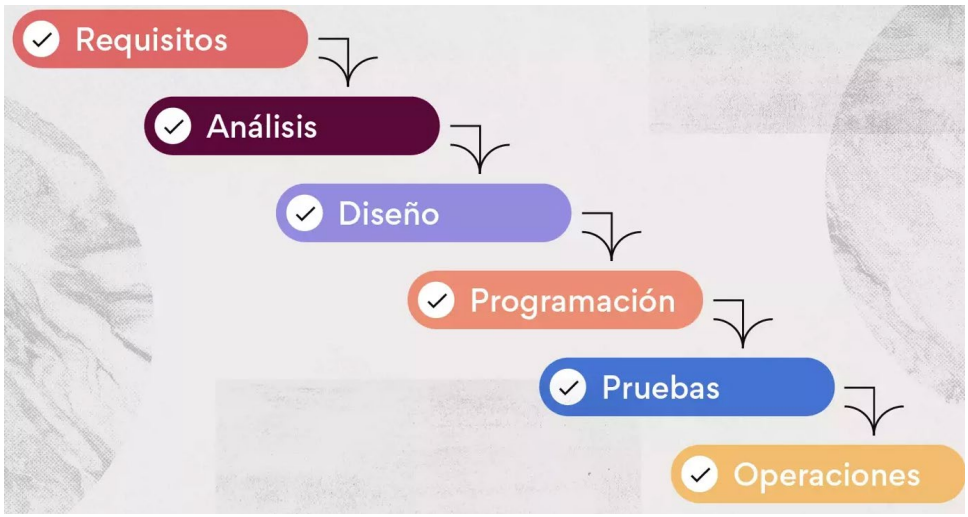


Ilustración 19 - Metodología Waterfall y sus etapas. Fuente: https://asana.com/es/resources/waterfall-project-management-methodology

3.1.2.1. Etapas

A diferencia de SCRUM, en esta metodología usualmente no hablamos de ceremonias sino de fases o etapas, estas etapas son:

- Fase de requerimientos: Es el proceso de planificación inicial en el que los miembros del equipo reúnen toda la información posible para garantizar el éxito del proyecto. Como las tareas del método waterfall dependen de los pasos anteriores, hay que prever todo en detalle antes de empezar (asana, 2022). Esta fase incluye todo, es decir, desde qué recursos se necesitan a qué miembros específicos del equipo trabajarán. Por lo general, a este registro se lo llama documento de requerimientos del proyecto
- Etapa de diseño del sistema: En un proceso de desarrollo de software, la fase de diseño implica que el equipo que trabajará en el proyecto especifique qué hardware usará, además de cualquier otro detalle, como los lenguajes de programación y la interfaz de usuario.
- Etapa de implementación: Esta es la fase en que todo entra en acción. Según los documentos de requerimientos del paso uno y del proceso de diseño del sistema del paso dos, el equipo inicia un proceso de desarrollo pleno para elaborar el software que se ha previsto tanto en la fase de requerimientos como en la de diseño del sistema (asana, 2022).

- Etapa de pruebas: En esta etapa el equipo de Desarrollo entrega el proyecto al equipo de Calidad para que realice las pruebas pertinentes. Los 'QA testers' buscan cualquier error que deba repararse antes de la implementación del proyecto. Los encargados de las pruebas documentan con claridad todos los problemas que encuentran al realizar el control de calidad (asana, 2022).
- Fase de desarrollo: En los proyectos de desarrollo, esta es la etapa en la que se implementa el software para los usuarios finales (asana, 2022). En otros casos, es el momento en que se lanza el entregable definitivo a los clientes finales.
- Fase de mantenimiento: Una vez que el proyecto se ha lanzado para su implementación, puede haber instancias en las que se descubra algún error nuevo o en las que sea necesario realizar alguna actualización del software. A esto se lo conoce como fase de mantenimiento y es muy común, en el desarrollo de software, que el trabajo de esta etapa sea continuo (asana, 2022).

3.1.2.2. Roles

Vamos a mencionar simplemente los roles principales, dentro de estos tenemos:

- Project Manager
- Analistas de Negocio
- Analistas de Técnicos
- Desarrolladores
- Testers

3.1.2.3. Ventajas

Algunas de las ventajas de esta metodología son:

- Facilidad a la hora de evaluar u obtener métricas respecto del progreso del proyecto.
- Evita involucrar al cliente lo cual en ciertos casos es una ventaja ya que pueden existir casos en donde el mismo no tenga el tiempo disponible para participar activamente.
- A nivel de costes en general se dice que es de "Presupuesto Cerrado", en general esto se asocia al concepto de que, dado su carácter secuencial, el proyecto no sufre cambios por ende sus costes son cuasi constantes.

- Al tener la suerte de que los requerimientos no sufren cambios, la documentación es extensa y coherente.

3.1.2.4. Desventajas

Ahora bien, sus desventajas no son pocas y vale destacar que actualmente hay una tendencia a no utilizar este tipo de metodologías y se vio reemplazada por metodologías agiles, de todas formas, algunas de sus desventajas son:

- Es una metodología que no está preparada para trabajar con cambios frecuentes
- Por el punto anterior decimos que no se adapta a la realidad ya que la frecuencia de los cambios en cualquier proyecto generalmente está a la orden del día
- Al no involucrar activamente al cliente incurrimos en un riesgo muy grande que es el hecho de que podamos no haber entendido al cliente y sus requisitos y entreguemos al final del proyecto (y con el presupuesto ya consumido) algo lo cual el cliente no esperaba, es decir, las expectativas del cliente pueden no ser satisfechas
- En conjunción con el punto anterior, tenemos que las evaluaciones de calidad llegan en un momento muy tardío del proyecto, es decir, cuando ya tenemos bastante desarrollado por ende los costes si surgen errores de calidad pueden ser muy altos.

3.1.3. Aplicación al actual trabajo

Una vez presentado todos los conceptos de estas dos metodologías y habiendo ofrecido una breve explicación de su funcionamiento y demás entramos en la etapa de la aplicación individual. Es claro que estas metodologías están pensadas para ser implementadas en equipos de trabajo, es decir, raramente hayan sido pensadas para un individuo, aun así, la idea de utilizar estas metodologías era la de obtener conceptos y/o marcos de trabajo que me permitieran estructurar el "¿qué hacer?" frente al día a día y ofrecerme una suerte de mapa de acción coordinado y definido. Por lo tanto, tomando ciertos criterios sumados a la opinión periódica y frecuente del tutor pude armar una gama de entregables predefinidos, no tanto siguiendo las fechas de entregas formales sino más bien plazos personales de realización de partes del trabajo. Esta estructuración puede ejemplificarse de la siguiente manera:

- Establecí que mis bloques de trabajo se iban a estructurar en periodos de 10 días naturales y al final de cada periodo de 10 días iba a tener un título de la memoria finalizado, pero solo cuando los Objetivos Generales y Específicos estuviesen terminados, caso contrario el periodo de 10 días no iba a aplicarse.
    - Este periodo de 10 días era mi Sprint
    - Metodología usada: SCRUM
- El primer paso fue relevar las necesidades, es decir, formular mi idea de necesidad en una lista extensa de cuestiones a resolver
    - Este relevamiento extenso de tareas fue mi fase de requerimientos
    - Metodología usada: Waterfall
- Acto seguido se buscó convertir esa suerte de lista de tareas y/o entregables, en un párrafo concreto y claro, de esto se extrajo el Objetivo General
    - Metodología usada: Waterfall
- Posteriormente, se optó filtrar o agrupar esa lista de tareas extensa en objetivos más concretos desde donde se obtuvo la lista de Objetivos Específicos.
    - Metodología usada: SMART[13] con Waterfall
- Mi fase de Demo era cada fin de par de Sprints en vez de 1 como indica la metodología, en esta demo le mostraba a mi tutor los avances.
    - Metodología usada: SCRUM
- En cada Sprint realizaba una parte técnica de puro desarrollo y en el siguiente me encargaba de trabajar sobre la memoria en base a lo desarrollado previamente, por eso cada sesión con mi tutor era cada dos sprints.
- Existió una fase de Demo interna o personal en donde al final de cada Sprint puramente Técnico me encargaba de realizar una prueba completa de todo lo desarrollado, pero únicamente para mi persona.

Respecto a los roles podemos decir que fui parte de todos ellos, indistintamente de la metodología y mi tutor actuó de Cliente, pero más como evaluador general que como usuario de la solución planteada.

[13] Acrónimo que define una forma de definir objetivos de forma que sean específicos (Specific), medibles (Measurable), alcanzables (Reachable), relevantes (Relevants) y a tiempo (on Time)

Como comentario extra y meramente asociado a la parte técnica, todos los cambios y agregados al código se hacían a través del concepto de Pull Request[14] de manera que al finalizar cada "Entrega" al final de cada 2 Sprints, mis cambios eran confirmados y la rama principal de código (Master[15]) era actualizada.

3.2. Introducción a Arquitectura y Diseño

Vale aclarar que la solución está pensada para utilizar multithreading[16] pero a efectos prácticos y de simpleza, la explicación y ejemplificación radicara en un único Thread/Hilo[17].

Partiendo de la base de que todos los elementos ya están desplegados y funcionado tenemos que el punto inicial es la intención de navegación, es decir, como usuario de esta aplicación deseo que se genere una navegación automatizada (con acciones aleatorias o no) a una determinada URL o bien que se busque en algún buscador web (Google) una serie de palabras clave o Keywords[18] y que se navegue a través de los resultados hasta encontrar una URL en concreto, luego se ingrese a la antedicha URL y nuevamente se genere una navegación automatizada.

Esta intención se sistematiza utilizando el View Feeder, luego entraremos en detalles en explicar que es y que contiene el Worker Service llamado View Feeder así como los otros componentes, pero de momento manejemos la idea de que la intención del usuario se traduce en un mensaje que debe ser parametrizado y enviado a través del View Feeder, este, al recibir esta intencionalidad parametrizada, encapsula la misma en un mensaje de Kafka y lo publica (Produce) en el Tópico asignado, finalizando allí su trabajo, es decir, el trabajo del View Feeder es proveer y traducir intenciones que se encolan en forma de mensajes dentro de un Tópico de Kafka.

---

[14] Petición o manifestación de la intención de cambio, para integrar cambios en el código fuente de un proyecto
[15] Rama con la que se comienza en cualquier proyecto, y es la que se utiliza como rama principal donde se encuentra el proyecto en su estado final (Lázaro, 2018).
[16] Técnica computacional que permite a un computador ejecutar múltiples procesos (uno por cada Thread/Hilo) simultáneamente.
[17] Unidad Atómica de Proceso dentro de un microprocesador.
[18] Es un término, palabra, concepto, etc. que un usuario cualquiera introduce en un buscador (Google, etc) para dar solución a sus dudas

Recordando los conceptos de mensajería vistos previamente y tomados en su expresión más básica, teníamos que para un Tópico existen subscriptores que son notificados cada vez que un mensaje llega a un Tópico, por lo tanto, nuestro servicio View Sender es un cliente, un subscriptor de este Tópico el cual se ve notificado o alterado cada vez que un mensaje llega al Tópico al que está subscripto.

Una vez que nuestro Worker Service View Sender tiene en su posesión un mensaje el siguiente paso es analizar qué tipo de mensaje es, ya que no son las mismas las acciones a realizar para ejecutar una navegación directamente a un sitio web o si tenemos que dirigirnos previamente a un buscador web.

Una vez definido el tipo de mensaje el siguiente paso es obtener un Proxy según el país desde el que se quiera simular una navegación.

Acto seguido tenemos que armar la "Receta", la misma tendrá acciones requeridas (MA), acciones aleatorias o de relleno (AA) y del tipo Template, de todo esto se encargará el servicio llamado Builder, que según ciertas reglas propias de las operaciones que surjan del motor aleatorio va creando diferentes tipos de operaciones (Atómicas o no). Como resultante del Builder tendremos una "Receta" llena de Operaciones que servirán como una suerte de guía para nuestro último servicio llamado Executor.

Este último servicio se encarga de configurar el controlador del explorador elegido (Chrome) y de indicarle al framework Selenium que operaciones de la receta deben ejecutarse a continuación, respetando ciertas reglas, que fueron definidas por el Builder previamente.

A grandes rasgos este es el flujo completo de operación, claramente que existen infinidad de detalles que iremos viendo más en detalle a medida que se avance en el trabajo.

3.3. Arquitectura y Diseño de Alto Nivel

La arquitectura a alto nivel persigue una suerte de atomicidad de componentes, una especie de microservicios, pero no en su sentido estricto sino tomado desde el punto de vista de sus responsabilidades. Podemos definir 3 grandes áreas de responsabilidades a saber:

- Grupo de Manipulación de Navegación y Vistas
    - View Feeder

- View Sender

- Grupo Coordinador de Comunicación
  - Kafka
  - Docker
- Grupo de Prestación de Servicios varios
  - SQL Server
  - Proxy Provider API
  - Cache
  - Common

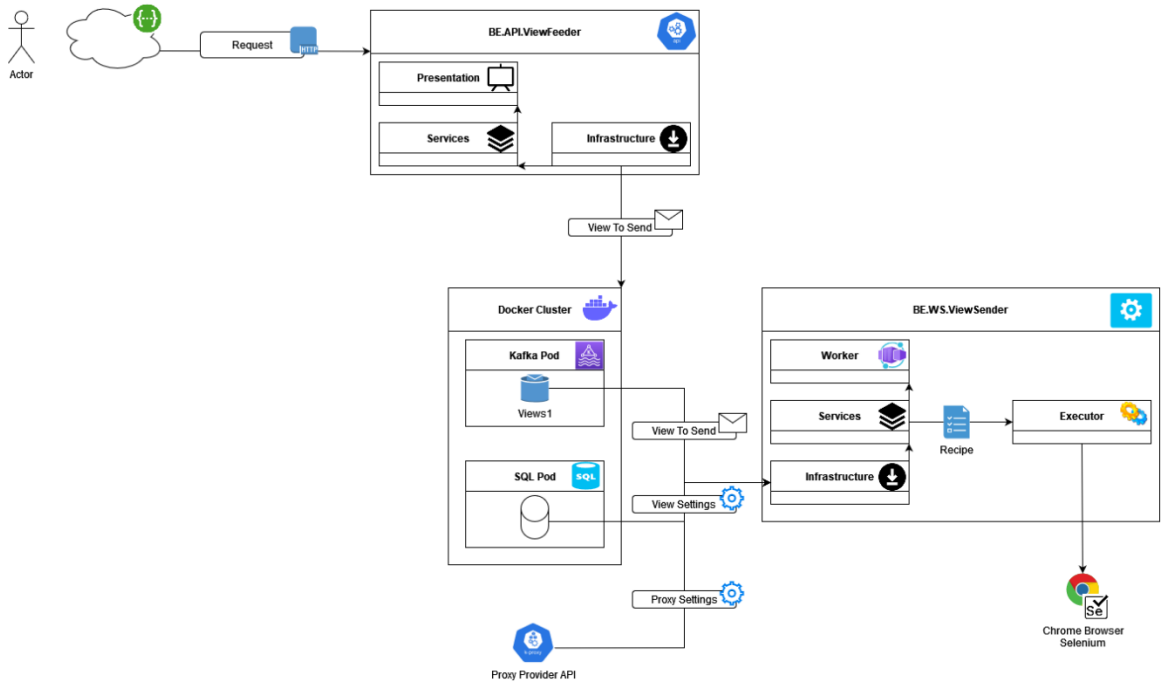


Ilustración 20 - Arquitectura de Alto Nivel. Fuente: Elaboración Propia

Como podemos ver en la imagen de arriba, cada componente o grupo de componentes persigue un objetivo en concreto, siendo capaz estos de funcionar de manera aislada siempre y cuando sus interfaces se respeten. A continuación, vamos a explicar cada uno de estos componentes y sus objetivos, así como su arquitectura de alto nivel.

3.3.1. Kafka Cluster y SQL en Docker

Puede considerarse la capa más abstracta de toda la aplicación, ya que no ofrece ni está configurada para ofrecer nada distinto a su propia naturaleza, es decir, si cambiásemos de

aplicación completamente, esta capa en general terminaría realizando lo mismo, es decir, actuar de soporte y puente de comunicación entre los distintos elementos.

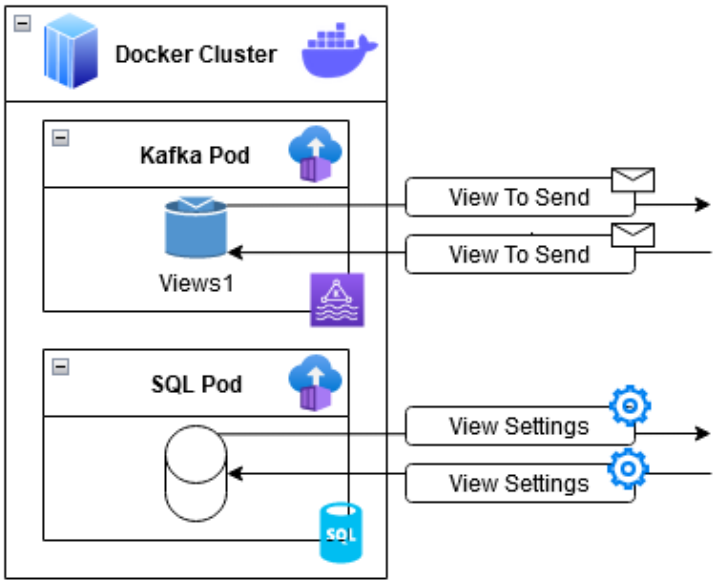


Ilustración 21 - Arquitectura de Alto Nivel - Docker, Kafka y SQL. Fuente: Elaboración Propia

3.3.1.1. Capa de Ejecución – Docker Cluster

Para empezar lo más conveniente es ver una rápida explicación de la Arquitectura de Docker.

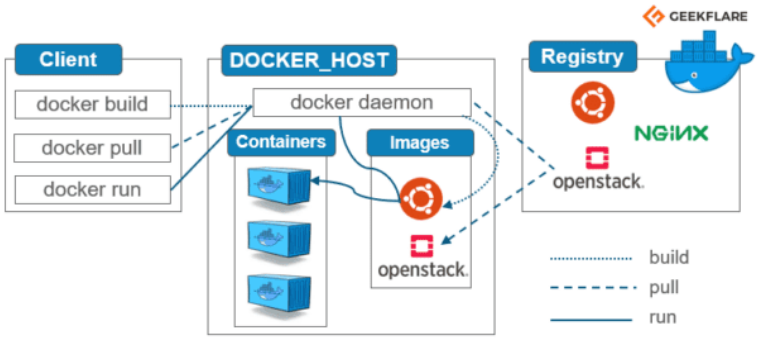


Ilustración 22 - Arquitectura Docker. Fuente: https://geekflare.com/es/docker-architecture/

- Docker Host: Es la parte central de todo el sistema Docker, es una aplicación que sigue una arquitectura cliente-servidor. Está instalado en la máquina host. Posee tres componentes:
  - Servidor
  - API
  - Interfaz de línea de comandos (CLI)

- Cliente Docker: Los usuarios de Docker pueden interactuar con Docker a través de un cliente. Cuando se ejecuta cualquier comando de docker, el cliente los envía al demonio dockerd, que los ejecuta. Los comandos de Docker utilizan la API de Docker.
- Registros de Docker: Es la ubicación donde se almacenan las imágenes de Docker.
- Imágenes: plantillas de solo lectura con instrucciones para crear un contenedor de Docker.
- Contenedores: Entorno de ejecución de una imagen.

Ahora bien, vamos a enfocarnos en Docker y sus beneficios los cuales son claramente la independencia y flexibilización que nos ofrece en términos generales, pero por mencionar algunos otros más particulares tenemos:

- Menor sobrecarga: ya que requieren menos recursos del sistema que los entornos de máquinas virtuales tradicionales o de hardware porque no incluyen imágenes del sistema operativo.
- Portabilidad: Las aplicaciones que se ejecutan en contenedores son multiplataforma.
- Mayor eficiencia: Los contenedores permiten poner en marcha, aplicar parches o escalar las aplicaciones con mayor rapidez.
- Mejor desarrollo de aplicaciones: Los contenedores respaldan los esfuerzos ágiles y de DevOps para acelerar los ciclos de desarrollo, prueba y producción.

En términos de objetivos con Docker buscamos: obtener una simpleza a la hora de la ejecución de aplicaciones, así como portabilidad.

3.3.1.2. Capa de Gestión Tópicos – Kafka

Ahora bien, habiendo hablado de los beneficios de los contenedores y lo que buscamos en ellos, es momento de hablar de la mensajería (Kafka), en este punto carece de sentido explicar en detalle todos los beneficios de este tipo de comunicación por lo que nos vamos a centrar en explicar que cosas buscamos puntualmente en este trabajo con el uso de este mecanismo.

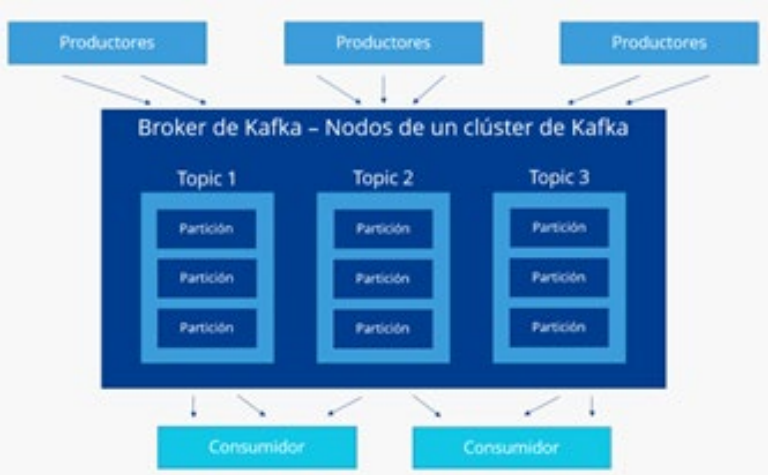


Ilustración 23 - Arquitectura Alto Nivel Kafka. Fuente: https://www.ionos.es/digitalguide/servidores/know-how/que-es-apache-kafka/

Apache Kafka se ejecuta como un clúster (red de ordenadores) en uno o más servidores que pueden encontrarse en centros de datos diferentes. Los nodos o puntos de intersección del clúster, denominados brokers[19], almacenan los flujos de datos entrantes, clasificándolos en los llamados tópicos. Los datos se dividen en particiones y se replican y distribuyen en el clúster, recibiendo un sello de tiempo. De esta manera, la plataforma de transmisión asegura una gran disponibilidad y un acceso de lectura rápido (Ionos, 2019).

Sumado al conjunto de bondades propias de este mecanismo en sí mismo, en nuestro caso nos interesa, además:

- Secuencialidad en términos de que el propio mecanismo se ocupa de asegurarnos que los mensajes siguen una suerte de orden (el cual a modo informativo puede definirse según ciertos criterios)
- Persistencia de los datos desde el punto de vista de que los mensajes jamás se pierden una vez que ingresan al tópico (inclusive una vez que son procesados siguen vigentes en el sistema)
- Integridad y tolerancia a fallos ya que el propio mecanismo ofrece una forma de asegurarse de que los datos no se corrompan.
- Soporte multiplataforma

[19] Cada uno de los servicios de Kafka que conforman el clúster. Almacenan y distribuyen los datos, que se organizan en tópicos

- Soporte de entornos concurrentes y posterior coordinación, como mencionamos al principio, la aplicación está preparada para ejecutarse en entornos concurrentes.

En cuanto a objetivos con Kafka buscamos: obtener un mecanismo fiable y escalable de comunicación.

3.3.1.3. Capa de Configuración – SQL Server

Por último, tenemos el motor de base datos, en este sentido, SQL Server no es utilizado más que como repositorio de configuraciones y algunas métricas útiles. Como beneficios tenemos:

- Reduce el tiempo dedicado a la gestión de datos.
- Analiza datos de diversas formas.
- Promueve un enfoque disciplinado para la gestión de datos.
- Convierte información dispar en un recurso valioso.
- Mejora la calidad y consistencia de la información.

En términos de objetivos tenemos los mismos que para el punto anterior, es decir: obtener un mecanismo fiable y escalable de comunicación.

3.3.2. View Feeder

Recordando lo que mencionamos anteriormente "…el trabajo del View Feeder es proveer y traducir intenciones que se encolan en forma de mensajes dentro de un Tópico de Kafka", este componente es una API en toda su extensión, su arquitectura persigue los componentes normales de un Microservicio tanto desde el concepto del dominio que maneja hasta como está compuesta. Ahora bien, internamente View Feeder persigue una arquitectura en capas como se puede ver en la siguiente imagen.

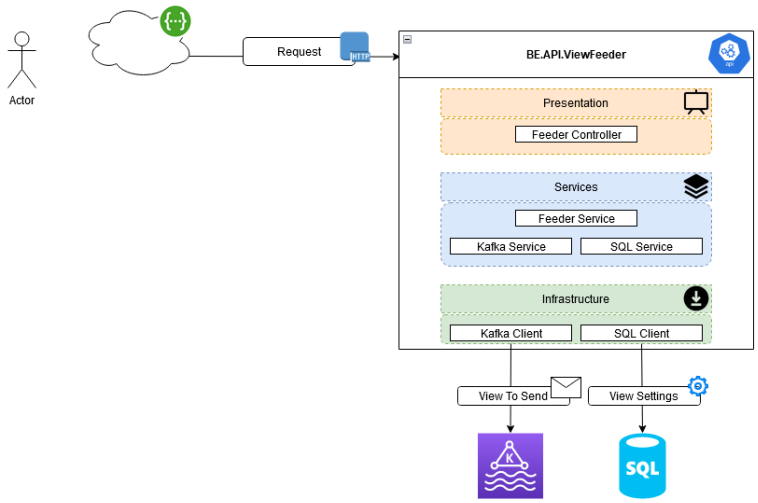


Ilustración 24 - Arquitectura Alto Nivel View Feeder. Fuente: Elaboración Propia

A grandes rasgos tenemos 3 grandes capas a saber:

- Presentación: es el nivel más alto o abstracto de toda la aplicación, podría decirse también que es la capa más cercana al usuario de la aplicación, se ocupa de controlar la exposición de la aplicación, en nuestro caso, se ocupa de definir las interfaces con el mundo exterior que tendrá la API.
- Servicios: consiste en la lógica que realiza las funciones principales de la aplicación, podría entenderse como la capa que encapsula las reglas de negocio de la aplicación.
- Infraestructura: es la capa más baja de toda la aplicación y es la encargada de manejar las cuestiones de comunicación con servicios de terceros o clientes externos. Dicho de otro modo, tomando como ejemplo una conexión a una DB, en esta capa encontraremos la gestión de las conexiones y demás, carece de lógica de aplicación y posee lógica pura y exclusivamente asociada a la infraestructura de la aplicación.

Para entender un poco más a que nos referimos con capas veamos algunas definiciones.

Cuando aumenta la complejidad de las aplicaciones, una manera de administrarla consiste en dividir la aplicación según sus responsabilidades o intereses. Este enfoque sigue el principio de separación de intereses y puede ayudar a mantener organizado un código base que crece para que los desarrolladores puedan encontrar fácilmente dónde se implementa una función determinada. Pero la arquitectura en capas ofrece una serie de ventajas que van más allá de la simple organización del código. Al organizar el código en capas, la funcionalidad común de bajo nivel se puede reutilizar en toda la aplicación. Esta reutilización es beneficiosa ya que

significa escribir menos código y puede permitir que la aplicación se estandarice en una sola implementación, siguiendo el principio Una vez y solo una (DRY). Con una arquitectura en capas, las aplicaciones pueden aplicar restricciones sobre qué capas se pueden comunicar con otras capas. Esta arquitectura permite lograr la encapsulación. Cuando se cambia o reemplaza una capa, solo deberían verse afectadas aquellas capas que funcionan con ella. Mediante la limitación de qué capas dependen de otras, se puede mitigar el impacto de los cambios para que un único cambio no afecte a toda la aplicación. Las capas (y la encapsulación) facilitan considerablemente el reemplazo de funcionalidad dentro de la aplicación. Por ejemplo, es posible que una aplicación use inicialmente su propia base de datos de SQL Server para la persistencia, pero más adelante podría optar por usar una estrategia de persistencia basada en la nube, o situada detrás de una API web. Si la aplicación ha encapsulado correctamente su implementación de persistencia dentro de una capa lógica, esa capa específica de SQL Server se podría reemplazar por una nueva que implementara la misma interfaz pública (Microsoft, 2022).

Surge a simple vista el principal motivo de la elección de una arquitectura en capas y esta es la separación de intereses.

Ahora bien, vamos a adentrarnos en cada capa en concreto respecto del componente API View Feeder.

3.3.2.1. Capa de Presentación – Presentation Layer

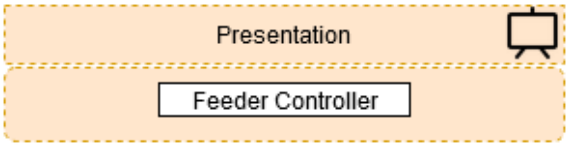


Ilustración 25 - View Feeder Capa de Presentación. Fuente: Elaboración Propia

Esta capa se encarga de exponer una API a modo de ofrecer una interfaz genérica de comunicación la cual se utilizará para encolar las intenciones en el Tópico en cuestión. Inicialmente fue pensada para utilizarse sin ningún tipo de autenticación (AllowAnonymous) ya que a efectos prácticos carece de sentido limitar tales acciones. La forma de exponer los endpoints es a través de un controlador tradicional.

A su vez, esta capa está basada en el estándar openAPI, el cual nos ofrece una manera común de exponer los contratos para su posterior interacción indistintamente cual sea la tecnología de su consumidor, la manera de adoptar el estándar es a través de las etiquetas ProduceResponse[20] y la posterior configuración de Swagger [21].

Tipificando sus interacciones tenemos que:

- Tipo de Entrada: HTTP Request en formato JSON
- Tipo de Salida: HTTP Reponse Status tradicional
- Tipo de Interfaz de Interacción: Open API + Swagger

Elementos:

- FeederController: es un controlador[22] que expone al usuario una serie de endpoints con los cuales se puede encolar una intención, obtener estadísticas y demás operaciones. Actúa como puerta de entrada del usuario y de la aplicación en general

A alto nivel lo más importante a destacar es que la manera de interactuar con este componente es a través de peticiones HTTP las cuales deben ser serializables a formato Json y sus respuestas serán estados tradicionales HTTP.

3.3.2.2. Capa de Servicios - Services Layer

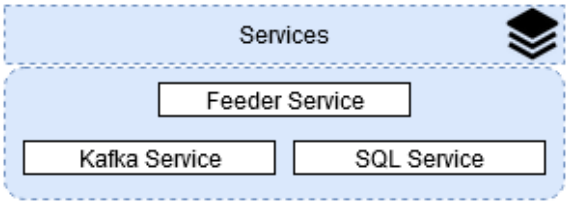


Ilustración 26 - View Feeder Capa de Servicios. Fuente: Elaboración Propia

Esta capa separa y encapsula las funcionabilidades de negocio especificas según el dominio de aplicación que precisemos, dicho de otro modo, esconde y aísla las implementaciones de negocio particulares de cada dominio. En nuestro caso no tenemos múltiples dominios de

[20] Etiqueta de C# que indica los tipos conocidos y los códigos de estado HTTP que debe devolver una acción.
[21] La especificación OpenAPI, originalmente conocida como la especificación Swagger, es una especificación para archivos de interfaz legibles por máquina para describir, producir, consumir y visualizar servicios web RESTful
[22] Objetos que se ejecutan al recibir una petición

aplicación por ende las clases Service son asociadas a una tecnología a nivel de convención de nombres en vez de a un dominio de aplicación tradicional. En el siguiente ejemplo se ve como el servicio KafkaService esconde las cuestiones particulares del método Consume(), de esta forma, cualquier componente que requiera consumir un mensaje de un Tópico, le bastara con inyectar este servicio en su implementación, siendo este agnóstico de cómo o que realiza el KafkaService internamente para llevar a cabo dicha funcionabilidad.

```
internal class KafkaService : IDisposable, IKafkaService
{
    public string Queue_Name { get { return _queueSetting.Queue_Name; } }

    public bool IsKafkaServerAliveAndTopicCreated { get { return _kafkaClient.IsKafkaServerAliveAndTopicCreated; } }

    private readonly ILogger _logger;
    private readonly QueueSetting _queueSetting;
    private readonly IKafkaClient _kafkaClient;

    public KafkaService(IServiceProvider serviceProvider, ILogger logger, QueueSetting queueSetting)
    {
        _queueSetting = queueSetting;

        _logger = logger;

        _kafkaClient = serviceProvider.GetRequiredService<IEnumerable<IKafkaClient>>().FirstOrDefault(x => x.Queue_Name == queueSetting.Queue_Name);
    }

    public DirectViewViewToSend ConsumeDirectViewViewToSend(bool simulateDesktop, bool simulateMobile, bool simulateTablet, CancellationToken cancellationToken)
    {
        if (simulateDesktop && !simulateMobile && !simulateTablet)
            return KafkaServiceHelper.DirectViewViewToSend_SimulateConsumeForDesktop();

        if (!simulateDesktop && simulateMobile && !simulateTablet)
            return KafkaServiceHelper.DirectViewViewToSend_SimulateConsumeForMobile();

        if (!simulateDesktop && !simulateMobile && simulateTablet)
            return KafkaServiceHelper.DirectViewViewToSend_SimulateConsumeForTablet();

        _logger.LogInformation(GetCommonLoggerMessage(message: "Started"));

        var consumeResult = _kafkaClient.Consume(cancellationToken);

        if (consumeResult is null)
            return null;

        return JsonSerializer.Deserialize<DirectViewViewToSend>(consumeResult.Message.Value);
    }
```

Ilustración 27 - Ejemplo Kafka Service. Fuente: Elaboración Propia

Es importante destacar que esta capa solo se ocupa de consumir un mensaje de un tópico en cuestión aplicando las reglas de negocios pertinentes, es decir, se ocupa de ejecutar las reglas de negocio frente a cada mensaje consumido del tópico de Kafka.

Elementos:

- FeederService: es un servicio del tipo “pasamanos”, en el sentido de que no implementa ningún tipo de lógica de negocio, sino que hace de punto de entrada para la capa de Servicios todas las peticiones que llegan al FeederController, es decir, redirige las peticiones que llegan al FeederController a los diferentes servicios que se requieren para exponer o encolar una intención o mensaje.
- KafkaService: Como mencionamos antes este servicio se ocupa de ocultar las implementaciones y reglas de negocio asociadas con la manipulación de Mensajes de Kafka y su posterior consumo o publicación de los tópicos. Su interfaz expone métodos para consumir dos tipos de mensajes específicos tipificados, el método dispose y un método para encolar mensajes.

```
namespace BE.WS.ViewSender.Services.Interfaces
{
    internal interface IKafkaService
    {
        string Queue_Name { get; }

        bool IsKafkaServerAliveAndTopicCreated { get; }

        DirectViewViewToSend ConsumeDirectViewViewToSend(bool simulateDesktop, bool simulateMobile, bool simulateTablet, CancellationToken cancellationToken);
        GoogleCtrViewToSend ConsumeGoogleCtrViewToSend(bool simulateDesktop, bool simulateMobile, bool simulateTablet, CancellationToken cancellationToken);
        void Dispose();
        Task<PersistenceStatus> ProduceAsync(bool isSimulation, BaseViewToSend baseViewToSend, CancellationToken cancellationToken);
    }
}
```

Ilustración 28 - Interfaz de exposición Kafka Service. Fuente: Elaboración Propia

- SQLService: es un servicio que se ocupa de gestionar las implementaciones de negocio asociadas al motor de base de datos de SQL Server. No se expone su interfaz ya que el sistema está preparado para utilizar un motor de base de datos, pero a efectos prácticos no se utiliza dentro de la aplicación.

3.3.2.3. Capa de Infraestructura – Infrastructure Layer

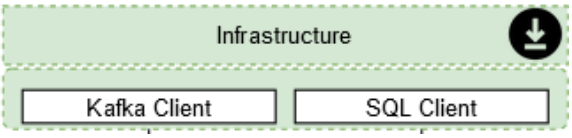
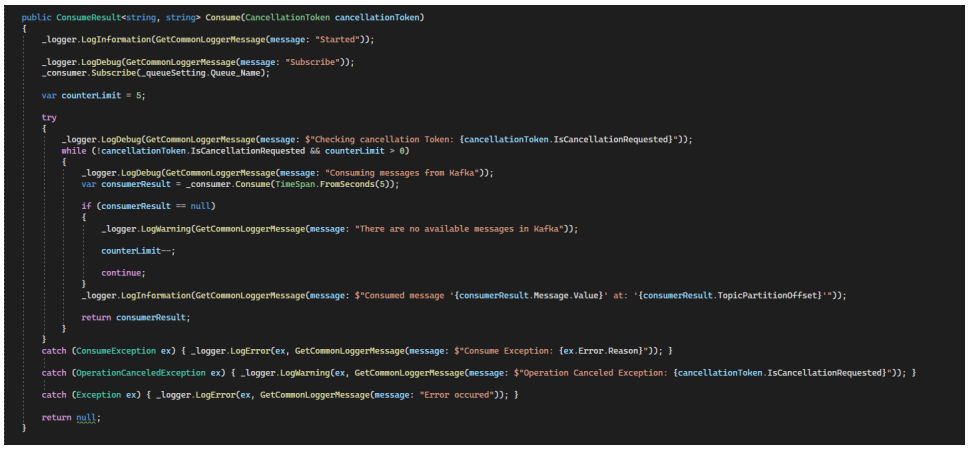


Ilustración 29 - View Feeder Capa de Infraestructura. Fuente: Elaboración Propia

Como mencionamos antes, esta capa se ocupa de manipular y gestionar los requisitos asociados a las dependencias externas que tiene la aplicación, en nuestro caso, nuestras dependencias para el View Feeder son a grandes rasgos dos a saber: Kafka y SQL.

Siguiendo el ejemplo mostrado anteriormente, en la imagen a continuación podemos ver como el Kafka Client esconde las implementaciones del método consume, por ende, si algún componente quiere consumir un mensaje de un tópico particular bastara con que inyecte en su propia implementación el servicio KafkaClient, vale destacar que a diferencia del KafkaService, esta capa ofrece simplemente los requisitos para su conexión con el tópico de Kafka, es decir, no implementa ni tiene en cuenta reglas de negocio, dicho de otro modo: “Solo se ocupa de la gestión de comunicación entre la API y el Kafka Broker”.



```
public ConsumeResult<string, string> Consume(CancellationToken cancellationToken)
{
    _logger.LogInformation(GetCommonLoggerMessage(message: "Started"));

    _logger.LogDebug(GetCommonLoggerMessage(message: "Subscribe"));
    _consumer.Subscribe(_queueSetting.Queue_Name);

    var counterLimit = 5;

    try
    {
        _logger.LogDebug(GetCommonLoggerMessage(message: $"Checking cancellation Token: {cancellationToken.IsCancellationRequested}"));
        while (!cancellationToken.IsCancellationRequested && counterLimit > 0)
        {
            _logger.LogDebug(GetCommonLoggerMessage(message: "Consuming messages from Kafka"));
            var consumerResult = _consumer.Consume(TimeSpan.FromSeconds(5));

            if (consumerResult == null)
            {
                _logger.LogWarning(GetCommonLoggerMessage(message: "There are no available messages in Kafka"));

                counterLimit--;

                continue;
            }
            _logger.LogInformation(GetCommonLoggerMessage(message: $"Consumed message '{consumerResult.Message.Value}' at: '{consumerResult.TopicPartitionOffset}'"));

            return consumerResult;
        }
    }
    catch (ConsumeException ex) { _logger.LogError(ex, GetCommonLoggerMessage(message: $"Consume Exception: {ex.Error.Reason}")); }

    catch (OperationCanceledException ex) { _logger.LogWarning(ex, GetCommonLoggerMessage(message: $"Operation Canceled Exception: {cancellationToken.IsCancellationRequested}")); }

    catch (Exception ex) { _logger.LogError(ex, GetCommonLoggerMessage(message: "Error occured")); }

    return null;
}
```

Ilustración 30 - Ejemplo Kafka Client. Fuente: Elaboración Propia

Si bien en este caso la responsabilidad de esta capa puede parecer simple, realmente no lo es, ya que dentro de sus responsabilidades se encuentran algunas reglas asociadas al establecimiento y uso del recurso, como, por ejemplo, al momento de querer consumir un mensaje esta capa impone un contador del tipo back off exponencial en donde si luego de 5 intentos seguimos sin tener mensajes disponibles, retorna el manejador al contexto actual de ejecución, liberando así recursos.

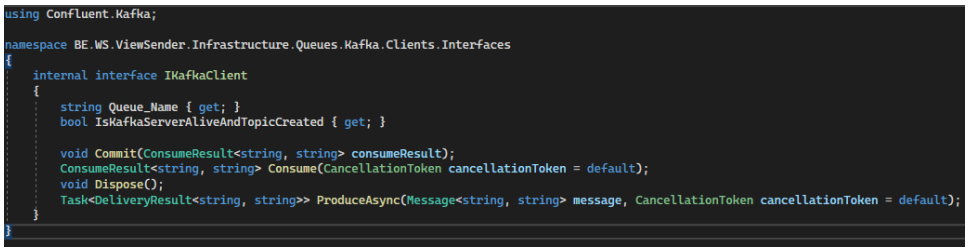
```
_logger.LogDebug(GetCommonLoggerMessage(message: $"Checking cancellation Token: {cancellationToken.IsCancellationRequested}"));
while (!cancellationToken.IsCancellationRequested && counterLimit > 0)
{
    _logger.LogDebug(GetCommonLoggerMessage(message: "Consuming messages from Kafka"));
    var consumerResult = _consumer.Consume(TimeSpan.FromSeconds(5));

    if (consumerResult == null)
    {
        _logger.LogWarning(GetCommonLoggerMessage(message: "There are no available messages in Kafka"));

        counterLimit--;

        continue;
    }
    _logger.LogInformation(GetCommonLoggerMessage(message: $"Consumed message '{consumerResult.Message.Value}' at: '{consumerResult.TopicPartitionOffset}'"));

    return consumerResult;
}
```

Ilustración 31 - Back off Exponencial - Kafka Client. Fuente: Elaboración Propia

Elementos:

- Kafka Client: es una clase del tipo “gestión” de recurso, se ocupa de establecer, gestionar, definir, etc. Es decir, todas las cuestiones pertinentes al establecimiento de la conexión con un Broker de Kafka, no impone reglas de aplicación o de negocio, sino que únicamente se ocupa a nivel de infraestructura. Su interfaz expone métodos para gestionar las operaciones con el tópico a saber: Consumir un mensaje (sin tipificación), encolar un mensaje (sin tipificación), confirmar un mensaje y el método Dispose[23]

[23] Se utiliza para cerrar o liberar correctamente sentencias que crean subprocesos en el sistema operativo



```
using Confluent.Kafka;

namespace BE.WS.ViewSender.Infrastructure.Queues.Kafka.Clients.Interfaces
{
    internal interface IKafkaClient
    {
        string Queue_Name { get; }
        bool IsKafkaServerAliveAndTopicCreated { get; }

        void Commit(ConsumeResult<string, string> consumeResult);
        ConsumeResult<string, string> Consume(CancellationToken cancellationToken = default);
        void Dispose();
        Task<DeliveryResult<string, string>> ProduceAsync(Message<string, string> message, CancellationToken cancellationToken = default);
    }
}
```

Ilustración 32 - Interfaz de exposición Kafka Client. Fuente: Elaboración Propia

- SQL Client: al igual que Kafka Client esta clase también es del tipo “gestión” de recurso, se ocupa de establecer, gestionar, definir, etc. Es decir, expone todas las cuestiones pertinentes al establecimiento de la conexión con una base de datos SQL Server, nuevamente, no impone reglas de aplicación o de negocio, sino que únicamente se ocupa a nivel de infraestructura. No se expone su interfaz ya que el sistema está preparado para utilizar un motor de base de datos, pero a efectos prácticos no se utiliza dentro de la aplicación.

3.3.3. View Sender

Tomando el criterio definido anteriormente como objetivo específico de este Worker Service “…enviar ordenes de navegación que simulen el comportamiento humano…” es a partir de donde iniciamos la explicación de este servicio, a diferencia del componente View Feeder, este elemento no es una API tradicional, ni siquiera espera la interacción (directa) del usuario, sino que es un servicio que se ejecuta detrás de escena. La arquitectura tanto del View Feeder como del Sender responde a la misma naturaleza, es decir, una arquitectura multicapa en donde cada capa responde a un dominio de aplicación, la mayor diferencia radica en que en este caso no existe una capa de Presentación, sino que la misma es de aplicación, por el motivo que explicamos antes de que no existe una espera de un accionar de usuario de manera directa para su ejecución.

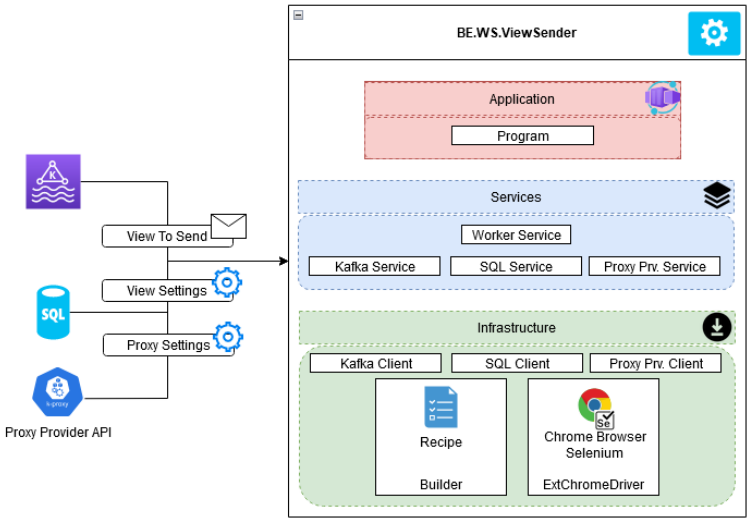


Ilustración 33 - Arquitectura Alto Nivel View Sender. Fuente: Elaboración Propia

A grandes rasgos nuevamente tenemos 3 grandes capas, pero a efectos prácticos solamente nos centraremos en la capa que es diferente respecto del View Feeder que es la de más alto nivel, esta es:

- Aplicación: es el nivel más alto o abstracto de toda la aplicación, se ocupa de controlar la ejecución del servicio, en este caso, se ocupa de definir las interfaces con el sistema operativo, así como de gestionar los hilos de ejecución

3.3.3.1. Capa de Aplicación – Application Layer

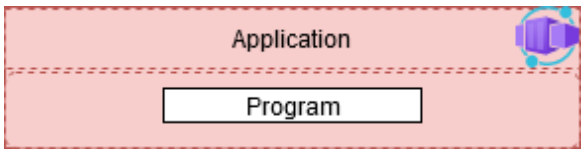


Ilustración 34 - View Sender Capa de Aplicación. Fuente: Elaboración Propia

Esta capa se encarga de gestionar y coordinar el Worker Service en comunicación con el sistema operativo, define entre otras cosas los hilos de ejecución que se van a utilizar y los elementos que se van a inyectar. En este caso la explicación en detalle carece de sentido ya que esta capa es la más simple de todo el componente, contiene una sola clase, que como mencionamos anteriormente se ocupa de definir cómo y con qué elementos se va a ejecutar

el servicio. En la siguiente imagen podemos ver su única clase en donde se definen mediante métodos de extensión[24] los elementos a utilizar a lo largo de la aplicación.

```
using BE.LIB.Common.Cache;
using BE.WS.ViewSender.Infrastructure;
using BE.WS.ViewSender.Services;

IHost host = Host
    .CreateDefaultBuilder(args)
    .ConfigureHostConfiguration(configHost =>
    {
        configHost.AddJsonFile("appsettings.json", optional: false, reloadOnChange: true);
        configHost.AddJsonFile("appsettings.Development.json", optional: true, reloadOnChange: true);
    })
    .ConfigureServices((hostContext, services) =>
    {
        services.AddMemoryCache();
        services.AddSingleton<ICoreMemoryCache, CoreMemoryCache>();

        services.AddInfrastructure(hostContext.Configuration);
        services.AddServices();
    })
    .Build();

host.Run();
```

Ilustración 35 - View Sender Clase única de la Capa de Aplicación. Fuente: Elaboración Propia

Tipificando sus interacciones tenemos que:

- Tipo de Entrada: Mensaje de Kafka en tópico
- Tipo de Salida: Navegación Web Automatizada
- Tipo de Interfaz de Interacción: Consola

Como mencionamos antes, para esta capa carece de sentido ahondar en detalles ya que es muy simple y más aun tomando su explicación a alto nivel.

3.3.3.2. Capa de Servicios - Services Layer

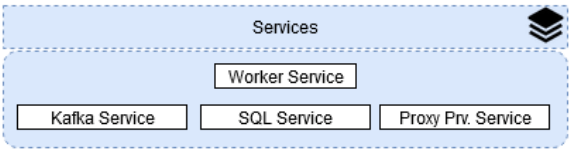


Ilustración 36 - View Sender Capa de Servicios. Fuente: Elaboración Propia

Elementos:

- WorkerService: es un servicio del tipo “pasamanos”, en el sentido de que no implementa ningún tipo de lógica de negocio, sino que hace de punto de entrada para la capa de Servicios todas las peticiones que llegan al FeederController, es decir,

[24] Los métodos de extensión permiten "agregar" métodos a los tipos existentes sin crear un nuevo tipo derivado, recompilar o modificar de otra manera el tipo original. Los métodos de extensión son métodos estáticos, pero se les llama como si fueran métodos de instancia en el tipo extendido.

redirige las peticiones que llegan al FeederController a los diferentes servicios que se requieren para exponer o encolar una intención o mensaje. Su interfaz expone dos simple métodos para iniciar y parar el servicio.

```
namespace BE.WS.ViewSender.Services.Interfaces
{
    internal interface IWorkerService
    {
        string Queue_Name { get; }
        Task StartAsync(CancellationToken cancellationToken);
        Task StopAsync(CancellationToken cancellationToken);
    }
}
```

Ilustración 37 - Interfaz de exposición Worker Service. Fuente: Elaboración Propia

- KafkaService: Como mencionamos antes este servicio se ocupa de ocultar las implementaciones y reglas de negocio asociadas con la manipulación de Mensajes de Kafka y su posterior consumo o publicación de los tópicos. Su interfaz expone métodos para consumir dos tipos de mensajes específicos tipificados, el método dispose y un método para encolar mensajes.

```
namespace BE.WS.ViewSender.Services.Interfaces
{
    internal interface IKafkaService
    {
        string Queue_Name { get; }

        bool IsKafkaServerAliveAndTopicCreated { get; }

        DirectViewViewToSend ConsumeDirectViewViewToSend(bool simulateDesktop, bool simulateMobile, bool simulateTablet, CancellationToken cancellationToken);
        GoogleCtrViewToSend ConsumeGoogleCtrViewToSend(bool simulateDesktop, bool simulateMobile, bool simulateTablet, CancellationToken cancellationToken);
        void Dispose();
        Task<PersistenceStatus> ProduceAsync(bool isSimulation, BaseViewToSend baseViewToSend, CancellationToken cancellationToken);
    }
}
```

Ilustración 38 - Interfaz de exposición Kafka Service. Fuente: Elaboración Propia

- ProxyProviderService: es un servicio encargado de interactuar con la API que provee los proxies según ciertos criterios tales como código de país, tipo de proxy y demás. Su interfaz expone dos métodos, uno para encolar y otro para desencolar proxis.

```
using BE.WS.ViewSender.Infrastructure.ExternalClients.BE.API.ProxyProvider.Implementation;
using BE.WS.ViewSender.Infrastructure.ExternalClients.BE.API.ProxyProvider.Models;

namespace BE.WS.ViewSender.Services.Interfaces
{
    internal interface IProxyProviderService
    {
        Proxy DequeueProxy(string countryCode, ProxyLevels proxyLevel, ProxyType proxyType);
        void QueueProxy(Proxy proxy);
    }
}
```

Ilustración 39 - Interfaz de exposición Proxy Provider Service. Fuente: Elaboración Propia

3.3.3.3. Capa de Infraestructura – Infrastructure Layer

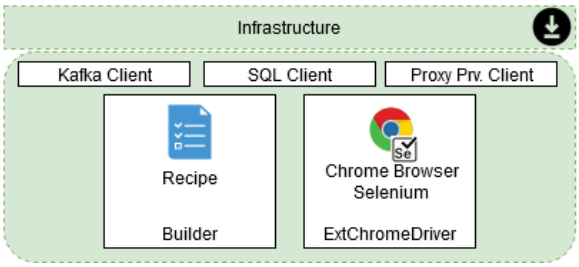


Ilustración 40 - View Sender Capa de Infraestructura. Fuente: Elaboración Propia

Elementos:

- Kafka Client: es una clase del tipo "gestión" de recurso, se ocupa de establecer, gestionar, definir, etc. Es decir, todas las cuestiones pertinentes al establecimiento de la conexión con un Broker de Kafka, no impone reglas de aplicación o de negocio, sino que únicamente se ocupa a nivel de infraestructura.

```
using Confluent.Kafka;

namespace BE.WS.ViewSender.Infrastructure.Queues.Kafka.Clients.Interfaces
{
    internal interface IKafkaClient
    {
        string Queue_Name { get; }
        bool IsKafkaServerAliveAndTopicCreated { get; }

        void Commit(ConsumeResult<string, string> consumeResult);
        ConsumeResult<string, string> Consume(CancellationToken cancellationToken = default);
        void Dispose();
        Task<DeliveryResult<string, string>> ProduceAsync(Message<string, string> message, CancellationToken cancellationToken = default);
    }
}
```

Ilustración 41 - Interfaz de exposición Kafka Client. Fuente: Elaboración Propia

- SQL Client: al igual que Kafka Client esta clase también es del tipo "gestión" de recurso, se ocupa de establecer, gestionar, definir, etc. Es decir, todas las cuestiones pertinentes al establecimiento de la conexión con una base de datos SQL Server, nuevamente, no impone reglas de aplicación o de negocio, sino que únicamente se ocupa a nivel de infraestructura.
- Proxy Provider Client: esta capa es una capa del tipo abstracción del contrato (OpenAPI) que ofrece el api de proxis, la cual es un servicio de 3ros gratuito llamado FreeProxies[25] que nos permite obtener proxis según ciertos criterios tales como código de país y demás. Ofrece todas las posibles operaciones que se pueden realizar con el controlador expuesto en la API de terceros.

[25] https://public.freeproxyapi.com/

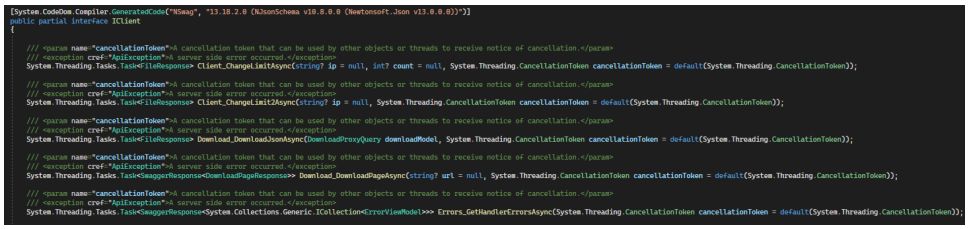


Ilustración 42 - Interfaz de exposición Proxy Provider Client. Fuente: Elaboración Propia

- Builder: es un tipo de componente especial, el cual no es inyectado en el motor de DI[26] sino que sus instancias son creadas de manera tradicional y manual por cada hilo de ejecución + mensaje que se encuentra en la aplicación o es consumido, este componente a su vez no expone ninguna interfaz. Actúa de creador de la receta y es agnóstico de la tecnología del navegador a utilizar. Es donde radica el esqueleto de ejecución de las operaciones de la receta.
- ExtChromeDriver: es un componente especial también, en nuestro caso solamente utilizamos este componente, pero tipificado al navegador Chrome pero permite generalizar y crear un componente por cada tipo de navegador, de manera que podamos tener una implementación especifica por cada tecnología de navegador que queramos utilizar, además, participa en la creación de la receta pero únicamente gestionando las configuraciones y valores pertinentes a operaciones que se deban ejecutar en un navegador del tipo Chrome. Su interfaz expone simplemente dos métodos a saber: SendView utilizado para ejecutar la navegación automática y su método Dispose.

```
using BE.WS.ViewSender.Infrastructure.Models;

namespace BE.WS.ViewSender.Infrastructure.WebDrivers.Chrome.Interfaces
{
    internal interface IExtChromeDriver
    {
        void Dispose();
        void SendView<TSendView>(TSendView viewToSend, string proxyValue) where TSendView : BaseViewToSend;
    }
}
```

Ilustración 43 - Interfaz de exposición ExtChromeDriver. Fuente: Elaboración Propia

[26] Motor de Inyección de Dependencias global de .NET

4. Diseño de la propuesta

En la siguiente sección haremos una suerte de relación 1 a 1 respecto de cuales fueron los objetivos planteados al comienzo del trabajo y como estos tienen su correlativa implementación, a efectos práctico lo que se busca es la relación estricta de resolver un OBJETIVO mediante una CONCRECION.

4.1. Objetivos

Los objetivos como destacamos al principio del trabajo se dividen en dos grandes secciones a saber: Generales y Específicos

Los primeros buscan resumir y presentar la idea central del trabajo, claramente en un sentido más amplio, también buscan contener la hipótesis o el problema en cuestión, así como la delimitación del tema

Los segundos buscan presentar de forma detallada los resultados que se pretenden alcanzar a través del trabajo, así como describir las etapas de la búsqueda en la secuencia de la ejecución, otra cosa que se busca en los objetivos detallados es la de relacionar el objeto del trabajo con sus particularidades, con una mayor delimitación.

4.1.1. Objetivo General

Como ya se ha indicado en la introducción, el objetivo general del trabajo es diseñar una herramienta que nos permita desarrollar una serie de operaciones web evitando ser detectados como bot y que se ajusten a ciertas características del comportamiento humano.

La concreción a grandes rasgos, se logró primeramente detectando que operaciones realiza en general un ser humano al interactuar con un computador, luego a través de un marco formal y a partir del diseño y tipificación de acciones frente a la antedicha interacción humano-ordenador se obtuvieron una suerte de pasos a intentar evitar o a realizar forzosamente en cada acción o grupo de ellas, luego se plasmaron esas acciones en un generador de operaciones, para acabar por último diseñando un ejecutor de operaciones.

4.1.2. Objetivos específicos

Basándonos en los objetivos específicos definidos anteriormente, ahora vamos a explicar que componentes satisfacen cada objetivo específico.

- Objetivo: Crear un tipo de servicio del tipo Worker y una API que permita su interacción utilizando C#
    - Concreción: Se creo un Worker Service llamado View Sender que se ocupa de actuar de Consumer/Subscriptor (encargados de leer y procesar los mensajes) de este tópico, y nos permite enviar ordenes de navegación que simulen el comportamiento humano superando así las restricciones de detección de bots
- Objetivos: Generar una navegación automatizada que busque un sitio en Google y no sea detectado como bot
    - Concreción: mediante Selenium podemos enviar comandos u operaciones a los navegadores web de manera automatizada y gracias al componente Builder y las tipificaciones de las acciones podemos evitar ser detectados como Bot y así simular mejor el comportamiento humano mejorando así la calidad de la prueba
- Objetivo: Navegar en el antedicho sitio de manera automatizada
    - Concreción: gracias al componente Builder podemos tener una lista de operaciones a realizar exclusivas para un dominio de un sitio concreto que nos permite navegar utilizando Selenium de manera automatizada.
- Objetivo: Generar operaciones de movimiento de mouse que persigan movimientos curvilíneos utilizando el algoritmo de Bezier
    - Concreción: mediante la Curva de Bezier y el algoritmo WindMouse, tenemos múltiples operaciones tipificadas de movimiento de mouse las cuales, mediante las coordenadas y el tamaño de resolución del navegador, nos permite simular un movimiento del ratón más aceptable a un comportamiento humano
- Objetivo: Generar operaciones de escritura o presión de teclas de teclado con retardos aleatorios obviando patrones repetitivos
    - Concreción: mediante dos tipos de operaciones tipificadas y agregando algunos retardos aleatorios basados en demoras de presionado de teclas que son

estadísticamente normales para un ser humano, podemos simular la escritura humana.

- Objetivo: Generar un archivo de Docker Compose que permita su instalación y ejecución en contenedores
    - Concreción: toda la aplicación se ejecuta y configura con un único archivo Docker Compose
- Objetivo: Utilizar un tópico de mensajería Kafka que permita él envió de mensajes entre Feeder y View Sender
    - Concreción: Tanto el View Feeder como el Sender se comunican mediante un tópico de Kafka.

4.2. Tipificación de Vistas y Operaciones

Como mencionamos antes, las operaciones son aquellas acciones que nos permiten interactuar con el explorador a través del framework Selenium, las mismas pueden ser atómicas o incluir otras operaciones, es decir, son un paso más hacia la concreción del objetivo.

Dentro de estas operaciones (sean cual sean su tipo) tenemos algunas que siempre deben ser ejecutadas en conjunto con otras de manera excluyente, esta suerte de "nueva" categoría, solo aplica a nivel de código y le llamamos "Bundle"[27].

Por otro lado, tenemos lo que llamamos Vistas, que es simplemente una forma de diferenciar cuando la intención es navegar hacia un sitio Web en concreto (DirectViewViewToSend) o bien cuando previo a esta navegación, debemos buscar ciertos Keywords en Google que nos permitan obtener dentro de sus resultados la dirección del sitio web en concreto (GoogleCtrViewToSend). Vale mencionar que estas "Vistas" simplemente son un modelo que permiten almacenar la información requerida o los parámetros necesarios para orientar la receta hacia el objetivo buscado.

[27] Paquete en ingles

4.2.1. Vistas

Existen dos tipos de vistas tal cual mencionamos en el punto anterior, estas a saber son: DirectViewViewToSend y GoogleCtrViewToSend.

Este tipo de vistas son meras clases que se utilizan en el código, simples modelos que nos permiten almacenar la información necesaria para realizar las navegaciones o intenciones del usuario. Es importante destacar que estos modelos actúan como interfaz común para los mensajes enviados por el componente View Feeder hacia el View Sender, siendo estos gestionados por el componente Kafka.

Otra cuestión importante, es que ambos modelos heredan de una vista o modelo base llamado BaseViewToSend, el cual simplemente actúa de elemento padre a fin de reducir la cantidad de código requerido y evita código duplicado.

Ahora bien, centrándonos en las vistas en concreto, tenemos que la vista base BaseViewToSend nos ofrece las siguientes propiedades:

- Id: es un entero que sirve de identificador único de la Vista
- UserAgent:
    - Id: es un entero que sirve de identificador único del Navegador elegido
    - UserAgentsGlobalName: cadena de caracteres que determina el nombre comercial del navegador elegido (Chrome, Firefox, etc)
    - DeviceId: entero que hace referencia a la enumeración utilizada para determinar el dispositivo elegido para simular la navegación (Mobile, Desktop o Tablet)
    - String: cadena de caracteres que responde al estándar que se utiliza para determinar el navegador para un determinado dispositivo
    - Width: es un entero que tiene relación directa con DeviceId, define el ancho máximo en pixels del dispositivo, es decir, cual es el máximo ancho en pixeles que puede usarse para simular la navegación
    - Height: es un entero que tiene relación directa con DeviceId, define el alto máximo en pixels del dispositivo, es decir, cual es el máximo alto en pixeles que puede usarse para simular la navegación

    - PixelRatio: es un entero que tiene relación directa con DeviceId, define la relación que existe entre las propiedades Width y Height para un dispositivo elegido
- Country
    - Id: es un entero que sirve de identificador único del país elegido
    - Iso: cadena de caracteres que responde al ISO del país elegido
    - Name: cadena de caracteres que responde al nombre del país elegido
    - Iso3: cadena de caracteres que responde al ISO3 del país elegido
    - LanguageCode: cadena de caracteres que responde al código de idioma del país elegido
- SendViewProxy: es una cadena de caracteres que tiene una asociación directa con las propiedades Latitude y Longitude la cual almacena el Proxy que se va a utilizar para simular la navegación desde una determinada parte del mundo
- Latitude: es una cadena de caracteres que nos indica la Latitud desde donde se quiere simular la navegación
- Longitude: es una cadena de caracteres que nos indica la Longitud desde donde se quiere simular la navegación
- UrlToSend: es una cadena de caracteres que nos indica la url del sitio a la que queremos navegar (no responde a la URL del buscador)

El caso de la vista DirectViewViewToSend no agrega propiedades extras, pero es la vista utilizada para denominar a las intenciones de navegación que solo contienen una simulación de navegación web directa, es decir, no utilizan un buscador web ni keywords para encontrar la url del sitio en cuestión, simplemente, albergan una url puntual y una intención concreta y simple que es: “navegar simulando una operativa humana hacia tal o cual sitio”

El caso de la vista GoogleCtrViewToSend agregan algunas propiedades extras a saber:

- KeywordsToFind: cadena de caracteres que almacena los keywords requeridos para poner en el buscador de Google para obtener como resultados la url del sitio web en cuestión, el cual se ve identificado por la propiedad base UrlToSend
- SearchEngineURL: es una cadena de caracteres que tiene una propiedad de solo lectura que almacena la url base del buscador Google "https://www.google.com";

Esta vista es la utilizada para denominar a las intenciones de navegación que contienen una simulación de navegación web directa pero buscando previamente mediante ciertas keywords definidas en la propiedad KeywordsToFind el sitio en los resultados ofrecidos por Google, es decir, utilizan un buscador web y ciertas keywords para encontrar la url del sitio en cuestión, su intención concreta es: "navegar simulando una operativa humana hacia tal o cual sitio que debe ser previamente encontrado en los resultados ofrecidos por Google según ciertas keywords utilizadas como criterio de búsqueda".

Además, alberga otras propiedades de solo lectura que sirven de ayuda para buscar ciertos elementos en el sitio de Google, esas propiedades exclusivas de esta vista son:

- GoogleInputSearchXPathSelector: selector predefinido que busca por XPATH el campo de busqueda de Google
- GoogleInputSearchCSSSelector: selector predefinido que busca por CSS Selector el campo de busqueda de Google

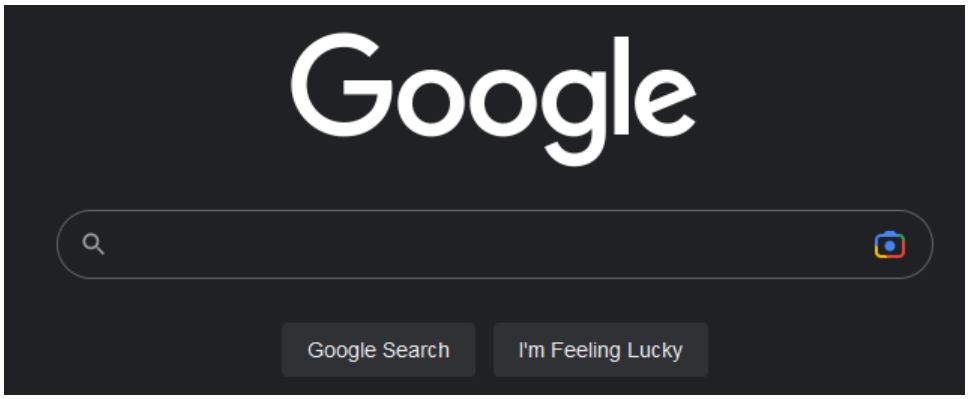


Ilustración 44 - Campo de búsqueda de Google. Fuente: Elaboración Propia

- GoogleAvailableResultsXPathSelector: selector predefinido que almacena el template de resultados según cada XPATH

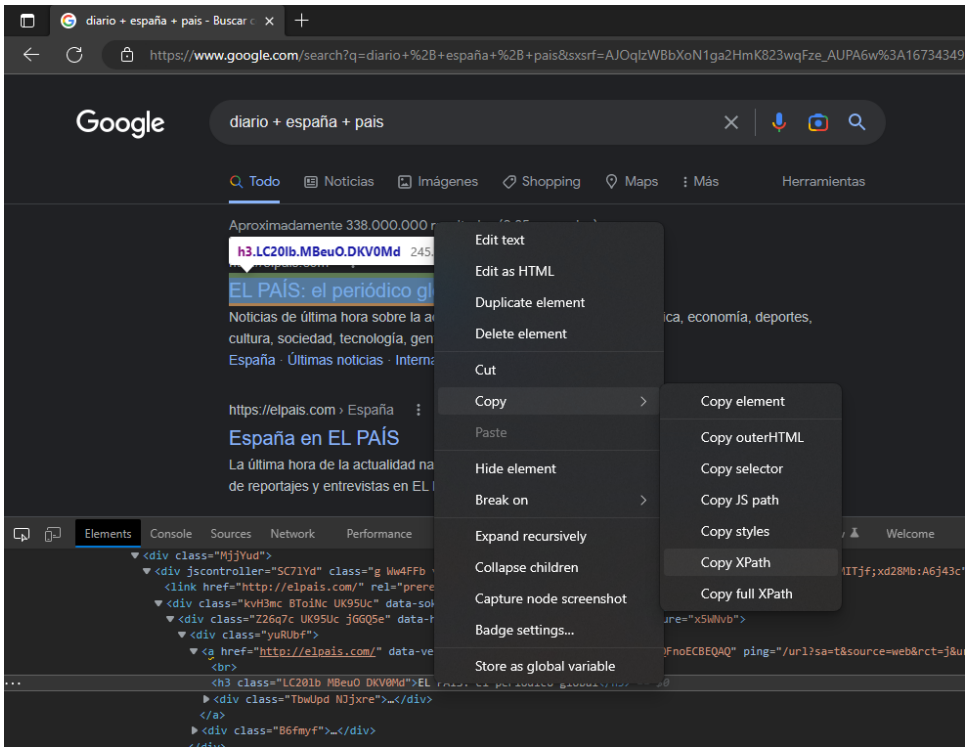


Ilustración 45 - Selector XPATH del Resultado Obtenido. Fuente: Elaboración Propia

- GoogleAvailableResultsNextButtonXPathSelector: selector predefinido que busca por XPATH el botón siguiente página de resultados de Google
- GoogleAvailableResultsNextButtonCSSSelectorSelector: selector predefinido que busca por CSSSelector el botón siguiente página de resultados de Google

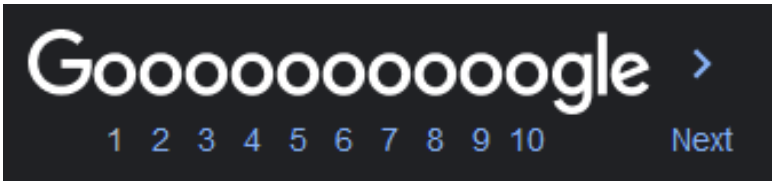


Ilustración 46 - botón siguiente de Google. Fuente: Elaboración Propia

4.2.2. Tipos de Operaciones

A modo informativo, detallaremos cada tipo de operación que se utiliza en la solución, así como su objetivo y el grupo/herencia al que pertenece, a modo de detallar todas las operaciones posibles que se pueden realizar en la solución.

Tabla 1 - Tipos de Operaciones en Código. Fuente: Elaboración Propia

Hereda	Grupo	Nombre	Objetivo
*	Ninguno	ExtAction	Clase base abstracta que almacena la estructura de ejecución general de todas las acciones.
ExtAction	Random Actions	BundleActions. InternalSiteNavigationRandomBundleAction	Busca y Navega de manera aleatoria en los enlaces hijos que presente una página web
ExtAction	Random Actions	MouseMoveRandomAction	Mueve el Ratón de manera aleatoria basado en las coordenadas retornadas por el algoritmo de Bezier
ExtAction	Random Actions	MouseScrollRandomAction	Mueve el Scroll del Ratón de manera aleatoria
ExtAction	Random Actions	MouseScrollTopBottomRandomAction	Mueve el Scroll del Ratón de manera aleatoria, pero al

			límite superior o inferior del sitio
ExtAction	Random Actions	PressKeywoardKeyRandomAction	Presiona ciertas teclas del teclado de manera aleatoria
ExtAction	Random Actions	WaitRandomAction	Introduce una espera aleatoria
ExtAction	Typed Actions	Bundle. GoogleCTRBundleAction	Incluye todas las operaciones necesarias para escribir ciertas Keywords dadas en Google y luego navegar por las páginas de resultados hasta que una cierta URL dada este presente, en ese caso, navega al sitio en cuestión caso contrario arroja un error
ExtAction	Typed Actions	Bundle. SendTextOnElementAction	Escribe un texto dado en el elemento elegido de manera secuencial con

			retardos, es decir, por cada carácter
ExtAction	Typed Actions	ClickOnElementAction	Realiza un clic en un elemento o selector dado
ExtAction	Typed Actions	IFrameAction	Pone el foco en un elemento IFrame dado
ExtAction	Typed Actions	MouseScrollIntoViewAction	Hace scroll hasta el lugar donde se encuentra un elemento o selector dado
ExtAction	Typed Actions	MouseScrollToAction	Hace scroll hasta las coordenadas dada
ExtAction	Typed Actions	MouseScrollTopBottomAction	Hace scroll hasta el límite superior o inferior del sitio
ExtAction	Typed Actions	MoveToElementAction	Mueve el cursor hasta el lugar donde se encuentra un elemento o selector dado

ExtAction	Typed Actions	NavigateAction	Hace una redirección a una url dada
ExtAction	Typed Actions	WaitAction	Introduce un retardo en base a un tiempo dado
ExtAction	Typed Actions	WindowAction	Pone el foco en un elemento Window dado

4.3. Armado de Receta

En esta sección vamos a hablar de quizás los dos módulos o componentes más importantes de la aplicación, ellos son los llamados ExtDriver (tipificado a ExtChromeDriver) y el componente Builder. En las siguientes secciones hablaremos más en detalle, pero a modo de resumen el primero de estos componentes se ocupa de preparar el explorador (y posteriormente ejecutar la receta) en el que luego se ejecutara la receta armada por el builder. El segundo se ocupa esencialmente de armar y configurar las operaciones que se incluirán en la receta.

4.3.1. ExtDriver

Es el componente encargado de configurar el navegador o el controlador que va a ejecutar el navegador posteriormente, hay muchísimos parámetros que el navegador Chrome nos permite configurar, para una descripción detallada recomendamos ir a la documentación oficial ya que aquí solo mencionaremos los más importantes.

La documentación oficial se encuentra en chromium.googlesource.

Lo más claro a la hora de explicar su funcionamiento es dividir su ejecución en etapas las cuales son:

1) Definir parámetros de Proxy: según la vista obtenida del tópico de Kafka (en la cual se nos indica desde que localización se quiere simular la navegación) y los proxies disponibles (provistos por la aplicación de terceros llamada FreeProxy), el siguiente paso es indicarle al navegador que proxy vamos a utilizar.

```
private void SetProxy(string proxyValue)
{
    _logger.LogInformation(GetCommonLoggerMessage(message: "Started"));

    _logger.LogDebug(GetCommonLoggerMessage(message: $"Proxy Value:{proxyValue}"));
    if (!string.IsNullOrEmpty(proxyValue))
        ChromeOptions.Proxy = new Proxy()
        {
            IsAutoDetect = false,
            HttpProxy = proxyValue,
            SslProxy = proxyValue,
            Kind = ProxyKind.Manual
        };
}
```

Ilustración 47 - Definición de Proxy - ExtDriver. Fuente: Elaboración propia

2) El siguiente paso consiste en definir los parámetros del navegador que corresponden a el tipo de navegador a utilizar, basados en los estándares de user agents, estos son “La cadena User-Agent (UA) está contenida en las cabeceras HTTP y tiene por objeto identificar los dispositivos que solicitan contenidos en línea. El User-Agent indica al servidor cuál es el dispositivo visitante (entre otras muchas cosas) y esta información puede utilizarse para determinar qué contenido devolver. (deviceatlas, s.f.)”, en base al user agent obtenido del mensaje del tópico de Kafka procedemos a definir el explorador a simular y su idioma, el cual recordemos tiene que estar asociado al proxy definido en el punto 1.

```
private void SetCommonArgumentsAsInputs<TSendView>(TSendView viewToSend, IEnumerable<string> argumentsAsInputs)
    where TSendView : BaseViewToSend
{
    _logger.LogInformation(GetCommonLoggerMessage(message: "Started"));

    if (argumentsAsInputs.Any(x => x.Trim().ToUpper() == "--USER-AGENT"))
        ChromeOptions.AddArgument($"{argumentsAsInputs.FirstOrDefault(x => x.Trim().ToUpper() == "--USER-AGENT")}={viewToSend.UserAgent.String}");

    if (argumentsAsInputs.Any(x => x.Trim().ToUpper() == "--LANG"))
        ChromeOptions.AddArgument($"{argumentsAsInputs.FirstOrDefault(x => x.Trim().ToUpper() == "--LANG")}={viewToSend.Country.LanguageCode}");

    ChromeOptions.AddUserProfilePreference($"intl.accept_languages", viewToSend.Country.LanguageCode);
}
```

Ilustración 48 - Definición de UA - ExtDriver. Fuente: Elaboración propia

3) El próximo paso es definir la resolución máxima que vamos a ofrecer, esto se hace en base al User Agent utilizado en el paso previo, ya que como sabemos, cada combinación de User Agent + Dispositivo tiene una resolución máxima. Este paso realmente solo aplica si el dispositivo a simular es una Tableta o un Movil, caso contrario utilizaremos la resolución común por defecto que es la de 1024x768.

Ilustración 49 - Definición de Resolución - ExtDriver. Fuente: Elaboración propia

4) Este es uno de los últimos pasos previos a generar la receta, en este caso se definen las configuraciones comunes según sea el tipo de Dispositivo y aquellas configuraciones más generalistas, por mencionar las más importantes indistintamente del dispositivo tenemos:
   a) disable-popup-blocking[28]: deshabilita todas las ventanas emergentes, pero solo aquellas propias del navegador no así del sitio.
   b) enable-automation[29]: usualmente cuando el navegador es manipulado automáticamente, se genera una ventana emergente que indica que el navegador está siendo controlado automáticamente (además incluye esta información en cada petición que realiza el mismo), con esta configuración quitamos ese comportamiento
   c) Incognito[30]: ejecutamos el navegador como incognito[31]
   d) allow-running-insecure-content[32]: permitimos que se ejecute contenido inseguro en el navegador
   e) start-maximized[33]: le indicamos al navegador que se ejecute de forma maximizada.
   f) acceptInsecureCerts[34]: Siempre que trabaje con certificados auto firmados o algún servidor con un certificado obsoleto o que no sea de confianza, la mayoría de los navegadores modernos muestran una advertencia de seguridad o errores de certificado no válido. Para evitar estas advertencias y errores mientras ejecutas tus pruebas automatizadas en BrowserStack, utiliza la siguiente capacidad.
5) El penúltimo paso es agregar a un listado de operaciones (que servirán de entrada para el componente builder) las operaciones requeridas (MA) y operaciones Template (TA), estas

[28] Detalles en https://www.browserstack.com/docs/automate/selenium/handle-permission-pop-ups#c_sharp
[29] Detalles en https://peter.sh/experiments/chromium-command-line-switches/#disable-blink-features
[30] Detalles en https://peter.sh/experiments/chromium-command-line-switches/#disable-blink-features
[31] Modo de navegación por internet en el cual el navegador no guarda ningún tipo de información sobre las páginas web visitadas (Gobierno de Argentina, s.f.)
[32] Detalles en https://peter.sh/experiments/chromium-command-line-switches/#disable-blink-features
[33] Detalles en https://chromedriver.chromium.org/capabilities
[34] Detalles en https://www.browserstack.com/docs/automate/selenium/accept-insecure-certificates

operaciones dependen estrictamente de la vista a utilizar según el mensaje obtenido del tópico de Kafka

```
if (baseViewToSend.GetType() == typeof(GoogleCtrViewToSend))
{
    var googleCtrViewToSend = (GoogleCtrViewToSend)Convert.ChangeType(baseViewToSend, typeof(GoogleCtrViewToSend));

    requiredActions.Add(new NavigateAction(ChromeDriver, googleCtrViewToSend.SearchEngineURL, LoggerFactory));

    requiredActions.AddRange(GetTemplateActionsByDomain(googleCtrViewToSend.SearchEngineURL, googleCtrViewToSend.UserAgent.DeviceId));

    requiredActions.Add(new GoogleCTRBundleAction(ChromeDriver, ChromeDriver, ScriptsModels, LoggerFactory, googleCtrViewToSend));
}
else
{
    var directViewViewToSend = (DirectViewViewToSend)Convert.ChangeType(baseViewToSend, typeof(DirectViewViewToSend));

    requiredActions.Add(new NavigateAction(ChromeDriver, directViewViewToSend.UrlToSend, LoggerFactory));

    requiredActions.AddRange(GetTemplateActionsByDomain(directViewViewToSend.UrlToSend, baseViewToSend.UserAgent.DeviceId));
}
```

Ilustración 50 - Definición de Operaciones Requeridas y Template - ExtDriver. Fuente: Elaboración propia

6) El último paso es crear una instancia del Builder otorgándole todas las configuraciones pertinentes, así como los mensajes de Kafka tipificados y demás para así, finalmente ejecutar la receta que obtengamos como respuesta.

```
var builder = new Builder(
                    ChromeDriver,
                    baseViewToSend.UrlToSend,
                    ChromeDriver,
                    ScriptsModels,
                    requiredActions,
                    LoggerFactory,
                    NavigateAction.NavigationType.GoTo,
                    (DeviceEnum._DeviceEnum)Enum.Parse(typeof(DeviceEnum._DeviceEnum), baseViewToSend.UserAgent.DeviceId.ToString()));

builder.Execute();

foreach (var error in builder.ErrorList)
    _logger.LogWarning(GetCommonLoggerMessage(message: $"Error Title: {error.Item1}. Error Message: {error.Item2}"));
```

Ilustración 51 - Creación de Builder y Posterior Ejecución - ExtDriver. Fuente: Elaboración propia

4.3.2. Builder

Este quizás es el componente que más algoritmia y lógica aplicada posee, pero de una forma indirecta, su objetivo es básicamente armar la receta que contiene las operaciones que luego se ejecutaran en el navegador, estas operaciones que ira seleccionando o creando para formar parte de la receta a su vez tienen condiciones varias e incluso sub-operaciones asociadas como explicamos anteriormente. Como decíamos, su lógica y aplicación se ve derivada de sus operaciones, la idea de administrar la algoritmia de una forma dividida fue en principio la mayor complejidad, es más, si se observa en detalle este componente posee simplemente dos grandes métodos a saber: Execute y BuildRandomActions.

El método BuildRandomActions contiene quizás una gran parte del corazón de la aplicación, el mismo se encarga de cargar operaciones aleatorias en base a un patrón builder aleatorio, en la imagen debajo se ve como se itera según una variable que es obtenida aleatoriamente

entre un máximo y mínimo, y por cada iteración que a su vez están enumeradas, crea e inyecta en la lista de operaciones a ejecutar cada una de las operaciones aleatorias que fue obteniendo.

```
private void BuildRandomActions(int maxQuantityOfRandomActions)
{
    _logger.LogInformation(GetCommonLoggerMessage(message: "Started"));

    var randomActions = new List<IExtAction>();

    _logger.LogDebug(GetCommonLoggerMessage(message: $"maxQuantityOfRandomActions: {maxQuantityOfRandomActions}"));
    for (int i = 0; i < maxQuantityOfRandomActions; i++)
    {
        var randomType = _availableRandomActionTypes[_random.Next(_availableRandomActionTypes.Count)];

        while (i > 0 && (randomType == randomActions[i - 1].GetType() || randomActions.Any(x => x.GetType() == randomType)))
            randomType = _availableRandomActionTypes[_random.Next(_availableRandomActionTypes.Count)];

        var createdAction = GetRandomActionInstanceByType(randomType);

        _logger.LogDebug(GetCommonLoggerMessage(message: $"Adding Random Action {createdAction.GetType().Name} as Required"));
        randomActions.Add(createdAction);
    }

    _requiredActions.AddRange(randomActions);
}
```

Ilustración 52 - Generación de Operaciones Aleatorias (AA) - Builder. Fuente: Elaboración propia

Por otro lado, tenemos el método Execute, el mismo simplemente se encarga de iterar sobre cada operación que tenga la receta y mediante la interfaz común que utilizan todas las operaciones (IExtAction) se ocupa de ejecutar cada una de ellas. Recordemos que la lógica de ejecución está repartida entre todas las operaciones de la receta, es decir, cada operación en tanto y en cuanto se le ofrezca lo que precisa, es lo suficientemente inteligente como para ejecutar lo que tenga que ejecutar, de esta manera se logra una separación de conceptos, dominios y funcionabilidades.

```
public void Execute()
{
    _logger.LogInformation(GetCommonLoggerMessage(message: "Started"));

    foreach (var actions in _requiredActions)
        try
        {
            _logger.LogDebug(GetCommonLoggerMessage(message: $"Executing Action:{actions.GetType().Name}"));

            actions.ExecuteAction();

            ErrorList.AddRange(actions.ErrorList);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, GetCommonLoggerMessage());

            ErrorList.Add((ex.GetType().Name, $"{nameof(Builder)} - " + ex.Message));
        }
}
```

Ilustración 53 - Método Execute - Builder. Fuente: Elaboración propia

Dicho de otro modo y a modo de resumen, según orden de atomicidad de ejecución tenemos que para una intencionalidad de navegación simulando la interacción humana, encolada en un tópico de Kafka y generada por un usuario:

- Cada operación: sabe exactamente QUE ejecutar según su naturaleza

- Cada receta: sabe exactamente CUALES operaciones deben ser ejecutadas según el tipo de vista
- El componente Builder: sabe exactamente CUAL receta ejecutar y COMO dotarla de operaciones aleatorias (AA)
- El componente ExtDriver: sabe exactamente SOBRE QUE navegador y COMO configurar el antedicho navegador para otorgarle al componente BUILDER los requisitos

4.4. Evaluación

A la hora de evaluar este proyecto, vamos a analizar si la herramienta desarrollada nos ha permitido conseguir nuestro objetivo general, pero con dos enfoques diferentes a saber: búsquedas en Google con posterior navegación a sitio web y navegación a sitio web

4.4.1. ¿Qué vamos a evaluar?

En principio se evaluará el éxito o fracaso de la solución, el mismo se encuentra definido en la sección 4.5.4. Criterios de Éxito/Fracaso además también evaluaremos:

- En 10 búsquedas (incluye navegación a los sitios en cuestión) en Google que porcentaje de estas es detectada como bot
- En 10 navegaciones en diferentes sitios (con detección de bots) que porcentaje es detectada como bot.

Además, como otras métricas a evaluar tomaremos según cada caso, cantidad de operaciones requeridas, cantidad de operaciones aleatorias, cantidad de operaciones Template y tiempo requerido, cada navegación tendrá un identificador único compuesto por el hash[35] generado por la combinación de KeyWords[36] + URL.

[35]Cadena única de longitud fija, también llamado valor de “resumen de mensaje”, para cualquier dato o “mensaje” dado.
[36] Filtros de búsqueda a través de los cuales queremos mostrar anuncios si se usan ciertos términos de búsqueda.

4.4.2. Representatividad de las muestras y valores

La elección de 10 iteraciones de búsqueda según cada caso, no fueron resultados seleccionados arbitrariamente, se partió de una base que suponía evaluar o traducir numéricamente los 4.5.4. Criterios de Éxito/Fracaso de esto se obtuvo que simplemente una sola navegación que falle ya era tomada como fracaso, por lo tanto, si tomamos como ejemplo un usuario normal y según varios estudios tenemos que:

"*…Nuestros datos muestran que el usuario medio de banda ancha navega 138,1 páginas al día. En el caso de la conexión telefónica, la cifra es de 136,4*. (Mitchell, 2019)"

Haciendo una aproximación estadística y teniendo en cuenta que estos estudios son del 2019, podríamos aplicar un incremento considerando una tendencia alcista, es decir que en promedio un usuario normal visita entre unas 140/160 páginas distintas por día, dicho de otro modo, un usuario normal navega en unos 140/160 sitios web al día. De este promedio de navegaciones decidí hacer dos cosas, por un lado, ajustar ese promedio a unas 100 navegaciones diarias y por el otro, de este muestreo "ajustado" tomar solo el 10%.

El antedicho valor del 10% sumado al criterio de fracaso el cual es en extremo estricto, considere que unas 10 iteraciones de navegación por tipo eran más que suficientes para demostrar que mi solución funciona.

4.4.3. ¿Cómo lo vamos a evaluar?

Ejecutando la aplicación 10 veces según cada grupo de keywords los cuales deberán ser previamente introducidos y buscados en Google y acto seguido buscando en los resultados obtenidos la URL o Dominio del sitio en cuestión y luego ejecutando la aplicación 10 pero esta vez directamente navegando al sitio web o al dominio en concreto.

4.4.4. Contexto de evaluación

Se ejecutará la aplicación de manera secuencial, es decir, sin múltiples hilos de ejecución en un entorno Windows. Para ello deberemos primeramente definir algunos datos requeridos. Para el primer caso deberemos definir 10 grupos de keywords (usualmente 3 o más keywords)

para cada búsqueda y 1 sitio a navegar según cada grupo de keywords y para el segundo directamente usaremos los 10 sitios definidos en el punto anterior.

4.4.5. Criterios de Éxito/Fracaso

Partiendo de la premisa definida en los objetivos generales y específicos, llamamos fracaso a los siguientes puntos:

- "Dado un dominio concreto y una navegación especifica (sin importar su tipo GVTS/DVTS), tenga más del 15% de sus operaciones (indistintamente su tipo) detectadas como bot"
- "Dado un dominio concreto y una navegación especifica (sin importar su tipo GVTS/DVTS), tenga una duración de ejecución de más de 25 minutos"
- "Dado un dominio concreto y una navegación especifica (sin importar su tipo GVTS/DVTS), alguna de sus operaciones no pueda ejecutarse (sin importar el motivo)"
- "Dado un dominio concreto y una navegación especifica (sin importar su tipo GVTS/DVTS), la aplicación se ejecuta con algún tipo de error o no se ejecuta de forma normal"

En contrapartida llamamos éxito a:

- "Dado un dominio concreto y una navegación especifica (sin importar su tipo GVTS/DVTS), tenga menos del 15% de sus operaciones (indistintamente su tipo) detectadas como bot"
- "Dado un dominio concreto y una navegación especifica (sin importar su tipo GVTS/DVTS), tenga una duración de ejecución de menos de 25 minutos"
- "Dado un dominio concreto y una navegación especifica (sin importar su tipo GVTS/DVTS), todas sus operaciones puedan ejecutarse sin errores"
- "Dado un dominio concreto y una navegación especifica (sin importar su tipo GVTS/DVTS), la aplicación se ejecuta sin error alguno y de forma normal"

4.4.6. Exclusiones de evaluación

No se evaluará la conexión y rendimiento de las conexiones a Kafka ni a SQL Server ya que carece de sentido su prueba incluso de agregarlos incurriríamos en errores y deberíamos tener en cuenta otros retardos.

El tipo de sitio a buscar serán siempre sitios web tradicionales, es decir se excluyen redes sociales, youtube, etc.

Además, no vamos a realizar búsquedas que estén fuera del territorio español, ni en un idioma distinto al español. Como último, las restricciones horarias no van a ser tenidas en cuenta.

Por otro lado, no se implementarán restricciones frente a las cantidades máximas y mínimas de Operaciones Aleatorias

4.4.7. Especificaciones Técnicas de evaluación

- Processor: Intel(R) Core (TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
- Memory RAM: 32.0 GB (31.8 GB usable)
- Sistema Operatívo: Windows 11 64-bit operating system, x64-based processor
- Edición: Windows 11 Pro

4.4.8. Datos de entrada de evaluación

A continuación, se definen los datos en concreto, los cuales van a ser tomados como entradas para el sistema.

- País origen de la Búsqueda: España
- Horario de la Búsqueda: sin limitaciones
- Idioma de la Búsqueda: español
- Tipo de sitio: sitio web tradicional
- Tipos de Navegaciones y Vistas utilizadas: GoogleCTRViewToSend y DirectViewViewToSend

Tabla 2 - Resumen de datos de URL y Keywords para evaluación. Fuente: Elaboración Propia

ID	Grupo de Keywords	URL del Sitio Por Navegar
1	diario + españa + pais	https://elpais.com
2	deportes + novedades	https://www.marca.com
3	economia + último momento	https://www.eleconomista.es/economia/
4	vacaciones + playas + economico	https://www.booking.com/es/
5	vacaciones + playas + lujoso	https://www.smartbox.com/es
6	universidad + online	https://www.ilerna.es/formación/profesional
7	pinturas + oleo	https://desenio.es/láminas
8	avion + renting + seguros	https://www.mapfre.es/empresas/seguro-para-aviones
9	patinetes + alquiler	https://www.getyourguide.es/actividades
10	buceo + tiburones	https://aquariumlanzarote.com/es/producto/buceo-con-tiburones

5. Conclusiones y trabajo futuro

La obtención de las métricas de tiempo se realizó con un stopwatch[37] nativo de C# como puede verse en la imagen debajo, estos sirvieron para medir exactamente el tiempo de ejecución neto despreciando el tiempo requerido por la aplicación para ponerse en marcha, reservar recursos, etc.

[37] Es una clase del tipo temporizador que se usa en el lenguaje C# para obtener mediciones de tiempo.

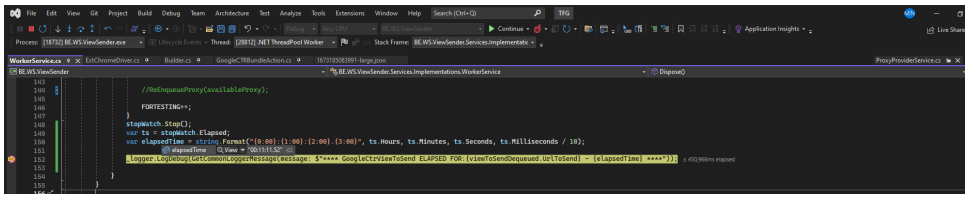


Ilustración 54 - Ejemplo Stopwatch en C#. Fuente: Elaboración Propia

Como mencionamos antes, la prueba se realizó haciendo 10 iteraciones del tipo GoogleCTRViewToSend y 10 Iteraciones del tipo DirectViewViewToSend, utilizando los datos que se especifican en la sección "4.6.6. Datos de entrada de evaluación".

5.1. Resultados y Conclusiones

A continuación, tendremos una tabla resumen con cada iteración y sus resultados, previamente hablaremos de los resultados a nivel general, separados en cada una de las métricas analizadas y haciendo una breve explicación de los valores vistos tomados en dos perspectivas diferentes a saber: Funcional y Técnica.

Por último encontraremos la perspectiva Personal, que es donde hare una apreciación personal del trabajo y lo que genero en mi persona.

A modo de tener una tabla más estrecha, se utilizaron algunos acrónimos para denotar las Vistas, en el caso de las vistas GoogleCTRViewToSend se utilizó el acrónimo GVTS y en el caso de las vistas DirectViewViewToSend se utilizó DVTS.

Tabla 3 - Resumen de Resultados. Fuente: Elaboración Propia

ID	Vista	Grupo de Keywords	URL del Sitio Por Navegar	Duración (hh:mm:ss.ms)	MA	AA
1	GVTS	diario + españa + pais	https://elpais.com	00:11:12.41	15	12
2	GVTS	deportes + novedades	https://www.marca.com	00:15:37.17	15	15
3	GVTS	economia + ultimo momento	https://www.eleconomista.es/economia/	00:06:33.52	15	11
4	GVTS	vacaciones + playas + economico	https://www.booking.com/es/	00:03:28.33	15	13
5	GVTS	vacaciones + playas + lujoso	https://www.smartbox.com/es	00:04:27.09	15	12
6	GVTS	universidad + online	https://www.ilerna.es	00:03:13.03	15	9

7	GVTS	pinturas + oleo	https://desenio.es/láminas	00:03:26.11	15	13
8	GVTS	avion + renting + seguros	https://www.mapfre.es/empresas/seguro-para-aviones	00:02:19.56	15	17
9	GVTS	patinetes + alquiler	https://www.getyourguide.es/actividades	00:03:37.44	15	12
10	GVTS	buceo + tiburones	https://aquariumlanzarote.com/es/producto/buceo-con-tiburones	00:07:52.19	15	11
11	DVTS	No utilizado	https://elpais.com	00:01:22.45	5	13
12	DVTS	No utilizado	https://www.marca.com	00:02:07.11	5	8
13	DVTS	No utilizado	https://www.eleconomista.es/economia/	00:02:23.29	5	11
14	DVTS	No utilizado	https://www.booking.com/es/	00:03:45.17	5	12
15	DVTS	No utilizado	https://www.smartbox.com/es	00:01:12.27	5	11
16	DVTS	No utilizado	https://www.ilerna.es	00:03:09.56	5	18
17	DVTS	No utilizado	https://desenio.es/láminas	00:02:43.48	5	16
18	DVTS	No utilizado	https://www.mapfre.es/empresas/seguro-para-aviones	00:02:33.23	5	15
19	DVTS	No utilizado	https://www.getyourguide.es/actividades	00:01:56.24	5	15
20	DVTS	No utilizado	https://aquariumlanzarote.com/es/producto/buceo-con-tiburones	00:02:47.14	5	13

5.1.1. Perspectiva Funcional

Desde este enfoque los resultados fueron más que exitosos, al punto de que los mismos fueron:

- Navegaciones detectadas como bot
  - De 10 iteraciones del tipo GVTS: 0
  - De 10 iteraciones del tipo DVTS: 0
- Tiempo requerido (Duración) promedio por tipo de Vista (hh:mm:ss.ms)
  - Para GVTS: 00:06:03:00
  - Para DVTS: 00:02:27:00

De estos dos resultados podemos concluir que en ninguna iteración indistintamente del tipo de vista empleado, ninguno de los dominios visitados detecto nuestra simulación como bot, lo cual arroja un porcentaje de éxito del 100%. Por otro lado si bien en algunos casos el tiempo empleado por la solución alcanzo umbrales que rozaban el máximo de los 25 minutos, también

la realidad es que la cantidad de recursos empleados en cada sitio web así como su construcción influyeron directamente en el tiempo requerido por cada operación para poder ser ejecutada, ya que, el framework Selenium posee un retardo de ejecución de operaciones que está asociado al tiempo que le toma al explorador "dibujar" todo el sitio web, esto es así ya que de no existir el antedicho retardo, podríamos incurrir en ejecutar acciones sobre elementos que aún no han sido dibujados o están en proceso de renderizado[38].

Respecto al tiempo podemos observar otra cuestión que vale la pena mencionar, además de las características obvias de mayor cantidad de operaciones requeridas que posee la navegación en donde primeramente se busca en Google (Vistas GVTS), el tiempo empleado en estas siempre es mayor que en las simulaciones a dominio directo (Vistas DVTS).

La conclusión que tenemos es que la aplicación en términos de éxito responde en un 100% a los objetivos planteados inicialmente, dicho esto, la solución puede ser empleada prácticamente en su totalidad para dotar de mayor realismo y calidad a las pruebas automatizadas, dejando en manos del usuario/desarrollador/tester los criterios de evaluación específicos de cada prueba que se desee realizar con esta herramienta.

5.1.2. Perspectiva Técnica

Desde esta perspectiva lo que buscamos es detallar cuestiones netamente técnicas y obtener algunas conclusiones en base a sus resultados. En la tabla debajo se resumen los resultados de las operaciones llevadas a cabo, sus cantidades, promedios, etc. Hay que recordar que, según los criterios definidos, los siguientes resultados son ejecutando la aplicación en un solo hilo.

Tabla 4 - Resumen de Resultados de Métricas

Métrica	MA	AA	TA	Total	Ite. Tot	Pro. MA	Pro. AA	Pro. TA	Pro. Total
GVTS	150	125	100	375	10	15	12.5	10	37.5

[38] El término renderizado (representación gráfica) es un anglicismo que viene de rendering y se considera un vulgarismo en español, que se aplica en informática en aquellos casos en los que el ordenador dibuja, pinta o representa algo en la pantalla (Saquete, s.f.)

DVTS	50	132	40	222	10	5	13.2	4	22.2
Total	200	257	140	597	20	20	25.7	14	59.7

Lo que podemos resaltar de los resultados obtenidos es que en general la cantidad de operaciones requeridas para las navegaciones de vistas del tipo GVTS son casi el triple de las utilizadas en DVTS, esto es por obvias razones ya que se requieren una mayor cantidad de operaciones en la receta producto de que la simulación debe pasar por más etapas.

Los promedios de operaciones del tipo Template (TA) así como las Requeridas (MA) necesariamente deben tener una variabilidad menor en contraste con las Aleatorias (AA) ya que el motor de generación de operaciones “Builder” no entra en participación en estos casos.

Por el lado de los recursos consumidos tenemos que hacer una salvedad de distinguir cuales son los recursos consumidos por el entorno para ejecutar la solución en contraste de los recursos consumidos por la solución en cuestión, para obtener esta métrica lo que se hizo fue:

1. Distinguir los nombres y PID de los procesos que incurren en la solución teniendo:
    a. Visual Studio: entorno de desarrollo
    b. Docker Daemon: entorno de ejecución de los contenedores
    c. Chrome Driver: proceso de ejecución del navegador utilizado por Selenium para ejecutar las operaciones
    d. View Sender: Worker Service encargado de ejecutar la solución en concreto
2. Promediar los recursos consumidos por el entorno de desarrollo en reposo es decir sin ejecutar nada
    a. Tenemos como resultado unos 940MB en promedio

Ilustración 55 - Recursos consumidos por el entorno de desarrollo en reposo. Fuente: Elaboración propia

3. Promediar los recursos consumidos por el entorno de ejecución en reposo activo es decir sin procesar nada, pero con los contenedores ejecutándose
    a. Tenemos como resultado unos 200MB en promedio

Ilustración 56 - Recursos consumidos por el entorno de ejecución en reposo. Fuente: Elaboración propia

4. Promediar los recursos consumidos por el entorno de desarrollo en ejecución es decir ejecutando la aplicación
    a. Tenemos como resultado unos 1300MB en promedio

Ilustración 57 - Recursos consumidos por el entorno de desarrollo en ejecución. Fuente: Elaboración propia

5. Promediar los recursos consumidos por el entorno de ejecución activo
    a. Tenemos como resultado unos 230MB en promedio

Ilustración 58 - Recursos consumidos por el entorno de ejecución activo. Fuente: Elaboración propia

6. Promediar los recursos consumidos por el proceso de aplicación en ejecución
    a. Tenemos como resultado unos 30MB en promedio

Ilustración 59 - Recursos consumidos por el proceso de aplicación en ejecución. Fuente: Elaboración propia

7. Promedio de los recursos consumidos por el proceso de ejecución del navegador en tiempo de ejecución
    a. Tenemos como resultado unos 200MB en promedio

Ilustración 60 - Recursos consumidos por el proceso de ejecución del navegador en tiempo de ejecución.

Fuente: Elaboración propia

Los resultados obtenidos nos indican que en total la aplicación en reposo requiere una arquitectura de 64Bits y una cantidad de memoria RAM del orden de los 4GB para funcionar correctamente por hilo de ejecución, de esta medida hay que quitarle el consumo del entorno de desarrollo y sumarle los recursos del entorno de despliegue una vez que la aplicación está desplegada. Por lo tanto, se hace una estimación de unos 3GB a la cual se le suma un "agregado" por precaución de 1GB en caso de que los sitios o dominios a navegar requieran más recursos para ser renderizados.

5.1.3. Perspectiva Personal

A carácter personal vale destacar que ver como algo que empezaba como una idea iba tomando forma con un marco cada vez más estricto era una sensación de realización difícil de explicar, la dedicación y lo aprendido a lo largo del proceso de codificación fue algo que también vale la pena destacar, quizás la parte más compleja fue la de la tipificación y prueba de las acciones, así como también la integración con Docker con todos los YML[39] de configuración y claramente su posterior puesta en marcha. Mención aparte la parte de la evaluación la cual requirió muchísimo tiempo para preparar los datos y obtener métricas que tengan un sentido.

Otra de las cuestiones que creo que como profesional me enriquecen es que obligar a la escritura a dotarla de un carácter científico, conciso y claro no es algo menor, cada párrafo tenía que ser pensado y escrito con una orientación determinada a explicar un concepto

[39] Lenguaje de declaración de datos que facilita la legibilidad y la capacidad de escritura del usuario

especifico, tenía que tener una orientación y estructura definida lo cual claramente dotaba de una complejidad mayor.

Mas allá de todas estas complejidades creo que puedo decir que fui uno al comienzo del trabajo y uno al final, mi juicio es mucho más crítico y estructurado y creo que más allá de la experiencia técnica obtenida a lo largo de todo este tiempo, la parte más rica es esta suerte de capacidad de estructuración del pensamiento.

5.2. Trabajos Futuros

Definitivamente y como todo trabajo, existen infinidad de áreas y variables a mejorar de modo que puedan obtenerse mejores conclusiones y resultados, por mencionar algunas mejoras a considerar tendríamos:

- Análisis Conductual Ratón: en el presente trabajo se analizaron 3 métricas a saber: Movimiento, Velocidad/Aceleración y Distancia para determinar y tipificar el comportamiento, esto se podría mejorar analizando además la combinación de esas 3 métricas e incluso combinar esas métricas con la operativa del teclado.
- Análisis Conductual Teclado: en este caso se analizaron Duración o el tiempo que se pulsa una tecla y el Intervalo de salida/llegada o el tiempo transcurrido entre la pulsación de una tecla y la siguiente quizás podrían por un lado unirse para obtener un análisis conductual más completo o bien analizar todas esas micro operaciones en un rango de tiempo determinado logrando así analizar el comportamiento de manera contextual en vez de simplemente el análisis de un accionar aislado.
- Uso de inteligencia artificial: mediante de una IA entrenada podrían lograrse mejoras considerables a la hora de realizar las operaciones aleatorias, sobre todo aquellas que involucran movimiento, así en vez de basarnos en algoritmos tales como WindMouse y Bezier que enfocan los resultados a parametrizaciones de factores físicos como ser la gravedad o el viento, podríamos basarnos en una AI entrenada en seres humanos operando con un computador de manera que en vez de obtener unas lista de coordenadas mediante un algoritmo, sea la AI la que nos otorgue este conjunto de coordenadas de modo que los resultados sean más certeros o más “humanos”.

- Utilización de multithreading: si bien el sistema está preparado para operar con múltiples hilos, quizás se podría optimizar su rendimiento generando un contenedor de Docker por acá aplicación y relegando la gestión operativa de los múltiples hilos a obtener un clúster de contenedores en donde cada contenedor ejecutara una única instancia de la aplicación.
- Utilización de Múltiples Frameworks: en el presente trabajo solamente utilizamos y tipificamos un accionar en base al framework Selenium, pero podría mejorarse la navegación y dotar de mayor utilidad la combinación con otros frameworks tales como Puppeteer
- Acciones con Herencia extendida: actualmente nuestras acciones estan tipificadas y son estáticas, es decir, solo pueden actuar según un cierto rango de opciones limitadas, por lo tanto, tenemos una amplia gama de opciones en donde se podría por ejemplo hacer que cada acción puede ser heredada de manera de que las acciones sean solo "bloques" controlados de acciones que sirvan de base para acciones más complejas.
- Implementar el Motor de DI a modo global: actualmente el motor de DI no toma partido en la creación del Builder y del Driver (EXTChromeDriver) sino que estos elementos son creados de forma manual. Esta implementación puede permitirnos gestionar mejor la vida útil de los objetos inclusive puede permitirnos la reutilización de instancias de modo que se ahorren recursos y tiempo.
- Múltiples Navegadores: la aplicación está preparada para realizar las pruebas con otros navegadores, pero ciertas cosas deben ser programadas manualmente ya que no se encuentran presentes, esto nos permitiría controlar el comportamiento del objeto a probar en múltiples navegadores lo cual arrojaría una mejora de calidad considerable.

Las áreas de mejora mencionadas anteriormente son solo a modo de ejemplo de la infinidad de cambios a considerar en pos de incrementar la calidad de la solución planteada.

Referencias bibliográficas

Alejandro Acien, A. M.-R. (2021). *BeCAPTCHA-Mouse Synthetic Mouse Trajectories and Improved Bot Detection.* Madrid.

APD. (2022, January 13). *apd*. Retrieved from apd: https://www.apd.es/metodologia-scrum-que-es/

asana. (2022, September 29). *asana*. Retrieved from asana: https://asana.com/es/resources/waterfall-project-management-methodology

CepymeNews. (2022, 01 19). *CepymeNews*. Retrieved from CepymeNews: https://cepymenews.es/estadisticas-busqueda-google/

Datademia. (n.d.). *Datademia*. Retrieved from Datademia: https://datademia.es/blog/que-es-apache-kafka

deloitte. (n.d.). *deloitte*. Retrieved from deloitte: https://www2.deloitte.com/es/es/pages/technology/articles/waterfall-vs-agile.html

deviceatlas. (n.d.). Retrieved from https://deviceatlas.com/blog/list-of-user-agent-strings

Digital55. (2019, July 11). *Digital55*. Retrieved from Digital55: https://digital55.com/blog/herramientas-testing-introduccion-selenium/

Dirk U. Wulff, J. M.-M. (n.d.). *Detecting Types in Movement Trajectories* (Vol. 9). Retrieved from https://www.researchgate.net/publication/335586482_Mouse-tracking_Detecting_types_in_movement_trajectories

Drumond, C. (n.d.). *atlassian*. Retrieved from Atlassian: https://www.atlassian.com/es/agile/scrum

Galán, D. (2020, March 11). Retrieved from https://ifgeekthen.nttdata.com/es/que-es-angularjs-y-por-que-deberias-usarlo

Garate, I. G. (n.d.). Retrieved from https://ude.edu.uy/que-son-algoritmos/

Gobierno de Argentina. (n.d.). Retrieved from https://www.argentina.gob.ar/justicia/convosenlaweb/situaciones/para-que-sirve-navegar-de-modo-incognito

Google. (n.d.). Retrieved from https://languages.oup.com/google-dictionary-es/

Ionos. (2019, June 17). *Ionos*. Retrieved from Ionos: https://www.ionos.es/digitalguide/servidores/know-how/que-es-apache-kafka/

Jeff Huang, R. W. (n.d.). *No Clicks, No Problem: Using Cursor Movements to Understand and Improve Search.* University of Washington.

Keepcoding. (2022, August 23). *Keepcoding*. Retrieved from Keepcoding: https://keepcoding.io/blog/que-es-docker-compose/

Land, B. J. (2021, September 20). *WindMouse, an algorithm for generating human-like mouse motion*. Retrieved from https://ben.land/post/2021/04/25/windmouse-human-mouse-movement/

Land, B. J. (2021, Abril 25). *WindMouse, an algorithm for generating human-like mouse motion*. Retrieved from WindMouse, an algorithm for generating human-like mouse motion: https://ben.land/post/2021/04/25/windmouse-human-mouse-movement/

Lázaro, D. (2018). Retrieved from https://diego.com.es/ramas-y-uniones-en-git

Luciano Gamberini, A. S. (2016). Symbiotic Interaction. *5th International Workshop, Symbiotic 2016 Padua, Italy, September 29–30, 2016.* Padua.

Microsoft. (2022, November 01). *Microsoft Learn*. Retrieved from Microsoft Learn: https://learn.microsoft.com/es-es/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

Mitchell, J. (2019, October 9). *MetaMuse*. Retrieved from https://kickstand.typepad.com/metamuse/2007/10/how-many-web-pa.html

Mott Glosario. (n.d.). Retrieved from https://glosario.mott.pe/diseno/palabras/curva-bezier

RedHat. (2022, October 10). *redhat*. Retrieved from redhat: https://www.redhat.com/es/topics/integration/what-is-apache-kafka

Refactoring. (n.d.). *Refactoring*. Retrieved from Refactoring: https://refactoring.guru/es/design-patterns

Saquete, R. (n.d.). *Human Level Communications*. Retrieved from Human Level Communications: https://www.humanlevel.com/diccionario-marketing-online/renderizado-rendering-o-representacion-grafica-de-la-pagina

Scherbaum, T. G. (2019). *How design factors of the mouse-tracking procedure impact the inference from action to cognition.*

ticportal. (2022, March 14). *ticportal*. Retrieved from ticportal: https://www.ticportal.es/glosario-tic/waterfall-metodologia-desarrollo-secuencial

Tsvetkova, M., García-Gavilanes, R., Floridi, L., & Yasseri, T. (2017, 02 23). Even good bots fight: The case of Wikipedia. *Journals Plos*. Retrieved from https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0171774

Verity. (2022, September 06). *Verity*. Retrieved from Verity: https://www.verity.cl/beneficios-de-la-automatizacion-de-pruebas-de-software/

wearedrew. (2019, December 3). Retrieved from http://blog.wearedrew.co/productividad/-ventajas-y-desventajas-de-la-metodologia-scrum

Zenkov, A. (n.d.). *https://iographica.com/*. Retrieved from https://iographica.com/: https://iographica.com/

Índice de acrónimos