Estrategia de Pruebas – Presupuesto Final

Link del video: https://drive.google.com/file/d/1q-9Hr8uqWTp-e_eT7r3aF4irnTwJouUL/view?usp=sharing

1. Aplicación Bajo Pruebas

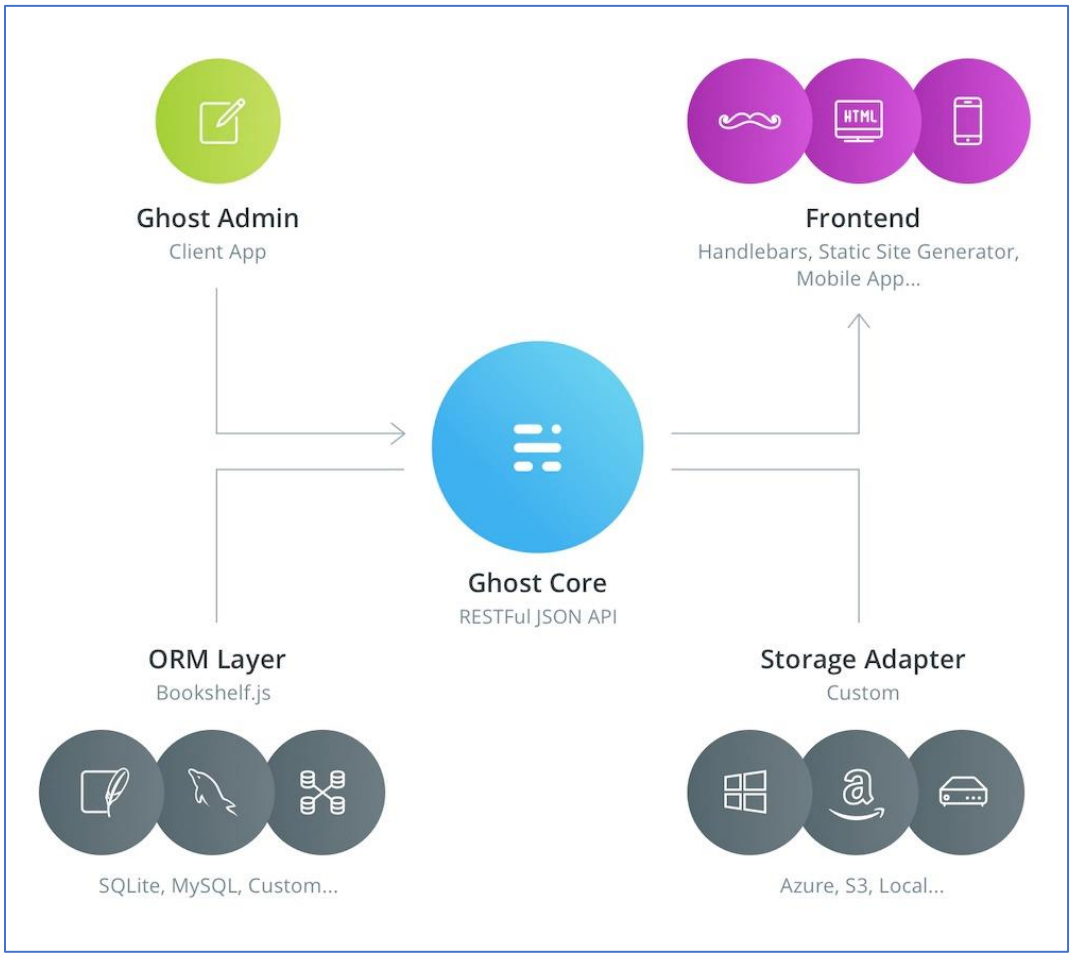
1.1. Nombre Aplicación: GHOST
1.2. Versión: 5.18.0
1.3. Descripción:

Ghost es un sistema de gestión de contenidos (CMS), open source, desarrollado principalmente en JavaScript para la parte lógica y para maquetación visual en HTML y CSS. Utiliza una base de datos SQLite de forma local para la gestión de datos del mismo. Diseñada para equipos que necesitan poder, flexibilidad y performance.

1.4. Funcionalidades Core:

- Crear post
- Editar post
- Crear página
- Editar página
- Eliminar página
- Crear miembro
- Crear tag
- Editar tag
- Editar configuración del sitio
- Ver RSS del sitio
- Editar diseño del sitio

1.5. Diagrama de Arquitectura:



1.6. Diagrama de Contexto:

Diagrama Contexto.png

Se sugiere la descarga de la imagen y verla desde un visor en el equipo.

1.7. Modelo de Datos:

Modelo de Datos.png

Se sugiere la descarga de la imagen y verla desde un visor en el equipo.

1.8. Modelo de GUI:

ModeloGUI.png

Se sugiere la descarga de la imagen y verla desde un visor en el equipo.

2. Contexto de la estrategia de pruebas

URL Repositorio Actualizado: https://github.com/slozano95/pruebas_semana5

2.1. Objetivos:

1. Realizar la ejecución de pruebas manuales y exploratorias que nos permitan dar una visión funcional del aplicativo Ghost.
2. Implementar pruebas E2E de las funcionalidades principales del aplicativo GHOST con las herramientas KRAKEN y Cypress
3. Implementar pruebas VRT para comparar los diferentes cambios que se han presentado desde la versión 3.x a la versión 5.x

2.1.1. Duración de la iteración de pruebas

2.1.2. Estimación de tiempos

Iteración 1 (Día 11 octubre – 21 octubre)	Iteración 2 (Día 24 octubre – 11 noviembre)	Iteración 3 (Día 15 noviembre - 2 diciembre)
Fase de implementación de pruebas manuales y exploratorios **Cada Ingeniero dispondrá de 8 horas al día durante las 2 semanas para llevar a cabo esta implementación**	Fase de ejecución de pruebas 2E2 y pruebas de reconocimiento **Cada Ingeniero dispondrá de 8 horas al día durante las 3 semanas para llevar a cabo esta implementación**	Fase de ejecución de pruebas VRT y escenarios de Datos **Cada Ingeniero dispondrá de 8 horas al día durante las 3 semanas para llevar a cabo esta implementación**

2.2. Presupuesto de pruebas:

2.2.1. Recursos Humanos

Ingeniero	Experiencia	Tiempo Disponible
#1	- 5 años elaborando pruebas con diferentes herramientas de automatización e integración continua. - 3 años de desarrollo de pruebas unitarias en javascripts. - Certificaciones en calidad de software vigentes - 4 meses desarrollando pruebas con jasmine para framework angular. - Experiencia en desarrollo de pruebas de aplicaciones móviles usando Appium	Lunes – Viernes 8 A.M. - 12 M 2 P.M. - 6 P.M.
# 2	- 4 años desarrollo de arquitectura y diseño de software - 5 meses de desarrollo de pruebas unitarias en javascripts. - 6 meses desarrollando pruebas con jasmine para framework angular. - Conocimiento profundo de bases de datos relacionales (por ejemplo, PostgreSQL, MySQL) y bases de datos NoSQL (por ejemplo, MongoDB).	Lunes – Viernes 8 A.M. - 12 M 2 P.M. - 6 P.M.
# 3	- 5 años elaborando pruebas en VTR y - 2 años amplia experiencia en desarrollo de software, scripting y gestión de proyectos. - Certificaciones en calidad de software vigentes - 4 meses desarrollando pruebas con jasmine para framework angular. - Conocimiento de lenguajes de programación seleccionados (por ejemplo, Python, C ++) y la plataforma Java / J2EE	Lunes – Viernes 8 A.M. - 12 M 2 P.M. - 6 P.M.
# 4	- 4 año elaborando pruebas con cypress - 3 meses de desarrollo de pruebas unitarias en javascripts.	Lunes – Viernes 8 A.M. - 12 M 2 P.M. - 6 P.M.

	- 6 meses desarrollando pruebas con jasmine para framework angular. - Experiencia en el uso de herramientas de monitoreo del sistema (New Relic) y marcos de prueba automatizados	

2.2.2. Recursos Computacionales

Recurso	Descripción
Servidor EC2 Familia T3 Tamaño micro	2 vCPU 8 GB Ram 12 Creditos de CPU por hora
Computador portátil ingeniero senior	Intel Core i5 16 GB Ram Linux
Computador portátil ingeniero senior	Intel Core i5 16 GB Ram Linux
Computador portátil ingeniero senior	Intel Core i5 16 GB Ram Linux
Computador portátil ingeniero senior	Intel Core i5 16 GB Ram Linux

2.2.3. Recursos Económicos para la contratación de servicios/personal:

Esta estrategia no presenta recursos económicos para la elaboración de las pruebas, ya que se realizan directamente desde los recursos maquinas asignados y con herramientas de libre distribución.

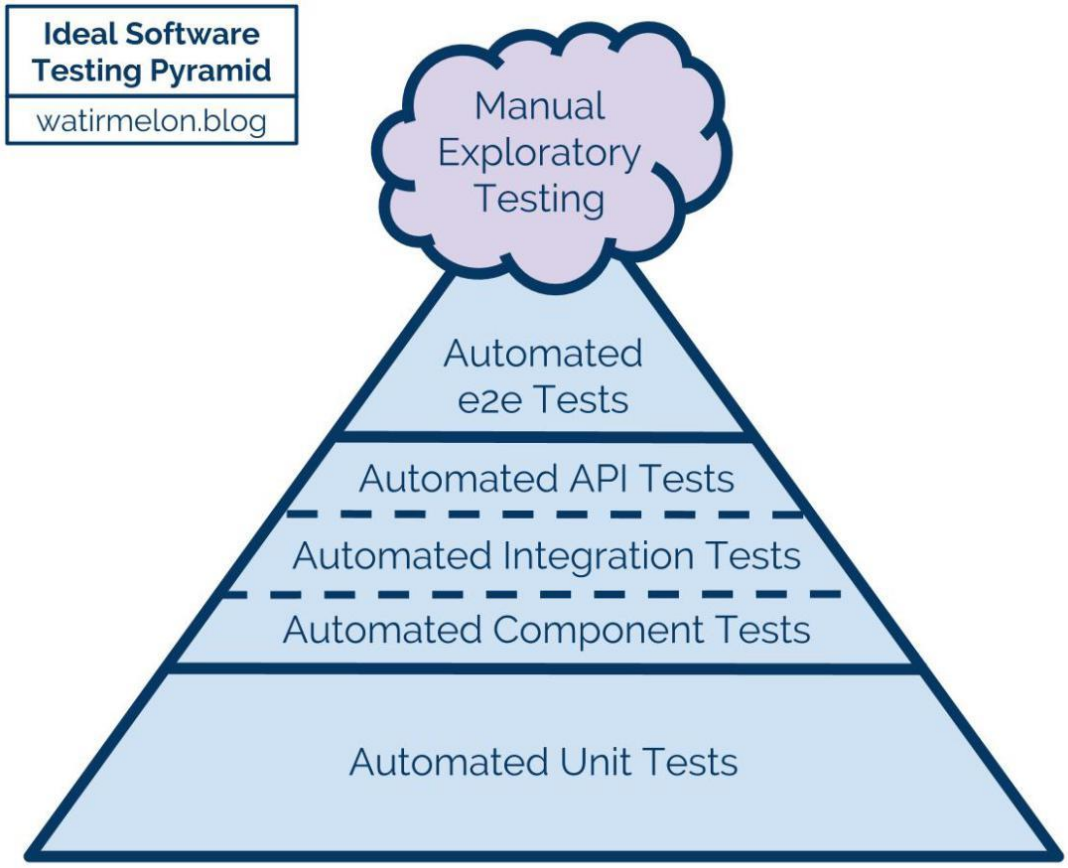
2.3. TNT (Técnicas, Niveles y Tipos) de pruebas:

NIVEL	TIPO	TÉCNICA	OBJETIVO
Pruebas de Sistema	Caja Negra	Pruebas Manuales	Realizar la ejecución de pruebas manuales y exploratorias que nos permitan dar una visión funcional del aplicativo Ghost.
Aceptación	Caja Blanca	Pruebas Automatizadas (API's de Automatizacion)	Implementar pruebas E2E de las funcionalidades principales del aplicativo GHOST con las herramientas KRAKEN y Cypress
Aceptación	Funcionales	Pruebas Automatizadas	Implementar pruebas VRT para comparar los

		(API's de Automatizacion)	diferentes cambios que se han presentado desde la versión 3.x a la versión 5.x

2.4. Distribución de Esfuerzo

Se propone una distribución de pruebas tipo pirámide con la adopción de pruebas unitarias y de integración principalmente para este caso.



Las tareas propuestas se realizarán en un ambiente de desarrollo de acuerdo al siguiente cronograma por nivel de prueba:

ITERACION	NIVEL	HORAS	Cronograma	Alcance
1	Pruebas de Sistema	320 Horas	Día 11 octubre – 21 octubre	Realizar la ejecución de pruebas manuales y exploratorias que nos permitan dar una visión funcional del

				aplicativo Ghost.
2	Aceptación	480 horas	Día 24 octubre – 11 noviembre URL: https://github.com/slozano95/pruebas_semana5/wiki/Semana-6	Implementar pruebas E2E de las funcionalidades principales del aplicativo GHOST con las herramientas KRAKEN y Cypress
3	Aceptación	480 horas	Día 15 noviembre - 2 diciembre URL: https://github.com/slozano95/pruebas_semana5/wiki/Semana-7	Implementar pruebas VRT para comparar los diferentes cambios que se han presentado desde la versión 3.x a la versión 5.x

2.5. Especificación Pruebas

2.5.1. Funcionalidades Bajo pruebas

- Funcionalidades Posts: Permite la creación de artículos que pueden ser publicados, editados dentro de la Herramienta.
- Funcionalidades Paginas: Permite la creación de Paginas que pueden ser publicada o editadas.
- Funcionalidades Tags: Son etiquetas que describen el contenido de un post mediante palabras claves las cuales se pueden manipular de diferentes maneras
- Funcionalidades Miembros: Permite la Creación de miembro del ghost publicadores, lectore ect.
- Funcionalidades Login: Este es el inicio de Ghost es importante hacer log-in en el portal.

2.5.2. Pros y Contras Herramientas de Testing

2.5.2.1. Kraken

Al realizar el ejercicio de pruebas E2E con la herramienta kraken, se tuvieron muchos inconvenientes con la instalación de la herramienta presenta muchas incompatibilidades al momento de su ejecución por ejemplo el hecho de instalar android studio solo para poder usar la opción de ejecución web, es bastante engorroso, por otra parte la poca documentación que se encuentra de la herramienta no permite superar errores que al parecer son comunes en la herramienta.

2.5.2.2. Cypress

Las pruebas realizadas con esta herramientas fueron bastantes mas fáciles de implementar ya que tanto la instalación de la herramienta fue mucho mas fácil y la construcción de la prueba igualmente mas fácil, se encontró buena documentación de la herramienta que permitió corregir o adecuar ciertas sentencias necesarias para la interacción con el entorno de ghost, la construcción es mucho mas técnica que la hecha en kraken pero ya que no se uso el lenguaje Gherkin, también se pudo observar que cypress es mas sensible en cuanto al búsquedas en el dom algunas instrucciones que servían perfectamente en kraken la realizarlas en cypress, no funcionaban y se debían especificar mas la búsqueda de la etiqueta. Esto a su vez es una ventaja ya que muchas web como es el ejemplo de ghost las etiquetas html no se rigen por un identificador único si por convenciones diferentes que hace mas complicado el manejo.

Dentro de la estrategia propuesta se usaron Diferentes técnicas que permitieron cumplir los objetivos planteados, por lo tanto, a continuación, detallamos cada una de ellas especificando el contexto en que se usaron y uso que se les dio.

2.5.3. Pruebas Manuales Pruebas

Versión de Software requerido

- Ghost: 5.22.10
- Node: 14.20.1
- npm: 6.14.17
- Google Chrome: 107.0.5304.107 64 bits
- Windows: Windows 10 64 bits

#	Proceso	Descripción	Link Demostración
1	Crear Post	Iniciar sesión -> Click en new post -> Crear post con titulo y descripción ->Click post ->Verificar que este el post.	https://drive.google.com/file/d/1hWSzyBZZ0UL9jiswbpCp-GbsgxoTVJHd/view?usp=sharing
2	Editar Post	Iniciar sesión -> Click en post -> seleccionamos el post que queremos modificar -> editamos -> guardamos	https://drive.google.com/file/d/12fpOJ-AVS5nHMxuzpzo-_Xf6-TQppCSF/view
3	Crear Publicación Post	Iniciar sesión -> Click en new post -> Crear post con titulo y descripción ->Click Publish-> Volver a la lista de publised -> Verificar que este el post en la lista	https://drive.google.com/file/d/14SDSkYGFTMKCcLXpILGVjUuHZFuT8BML/view
4	Despublicar Post	Iniciar sesión -> Click en new post -> Crear post con titulo y descripción ->Click post ->Verificar que este el post->Click Published->Seleccionar el post-> Click Unpublish	https://drive.google.com/file/d/19AprMrfTQ3DH1tDArbUk5e2DGQihCxvO/view
5	Crear Pagina	Iniciar sesión -> Ingresar a sección páginas -> click para agregar página "New page" -> ingresar titulo -> ingresar cuerpo -> click en botón "Publish" -> click en "Continue" -> click en "Publish page" -> verificar "Boom"	https://uniandes-my.sharepoint.com/personal/oa_sanchez2_uniandes_edu_co/_layouts/15/stream.aspx?id=%2Fpersonal%2Foa%5Fsanchez2%5Funiandes%5Fedu%5Fco%2FDocuments%2FPruebas%20Automatizadas%2FPruebas%20Exploratorias%20Ghost%2FVideos%20Evidencias%20Pruebas%20Exploratorias%2FPE007%2Emp4&ga=1
6	Editar Pagina	Iniciar sesión -> Ingresar a sección páginas -> click para agregar página "New page" -> ingresar titulo -> ingresar cuerpo -> click en botón "Publish" -> click en "Continue" -> click en "Publish page" -> verificar "Boom"	https://uniandes-my.sharepoint.com/personal/oa_sanchez2_uniandes_edu_co/_layouts/15/stream.aspx?id=%2Fpersonal%2Foa%5Fsanchez2%5Funiandes%5Fedu%5Fco%2FDocuments%2FPruebas%20Automatizadas%2FPruebas%20Exploratorias%20Ghost%2FVideos%20Evidencias%20Pruebas%20Exploratorias%2FPE008%2Emp4&ga=1

7	Crear Tag	Iniciar Sesión -> Ingresar modulo Tags ->Ingresar Crear un nuevo Tag con nombre con un string generado, un slug string generado y una descripción string generada -> click en el boton de guardar -> Validar que el tag se haya creado mediante el boton "Saved"	https://uniandes-my.sharepoint.com/personal/oa_sanchez2_uniandes_edu_co/_layouts/15/stream.aspx?id=%2Fpersonal%2Foa%5Fsanchez2%5Funiandes%5Fedu%5Fco%2FDocuments%2FPruebas%20Automatizadas%2FPruebas%20Exploratorias%20Ghost%2FVideos%20Evidencias%20Pruebas%20Exploratorias%2FPE013%2Emp4&ga=1
8	Editar Tag	Iniciar Sesión -> Ingresar modulo Tags ->Ingresar Crear un nuevo Tag con nombre string generado, un slug string generado y una descripción string generada -> click en el boton de guardar -> se valida que halla creado -> Se devuelve a la lista de tags -> se ingresa nuevamente a editar tag -> Se ingresar editar un nuevo Tag con nombre string generado, ingresa nuevo slug string generado y una nueva descripción string generada -> click en el boton de guardar -> Validar que el tag se halla guardado mediante el boton "Saved".	https://uniandes-my.sharepoint.com/personal/oa_sanchez2_uniandes_edu_co/_layouts/15/stream.aspx?id=%2Fpersonal%2Foa%5Fsanchez2%5Funiandes%5Fedu%5Fco%2FDocuments%2FPruebas%20Automatizadas%2FPruebas%20Exploratorias%20Ghost%2FVideos%20Evidencias%20Pruebas%20Exploratorias%2FPE014%2Emp4&ga=1
9	Eliminar un tag	Iniciar Sesión -> Ingresar modulo Tags ->Ingresar Crear un nuevo Tag con nombre string generado, un slug string generado y una descripción string generada -> click en el botón de guardar -> se valida que halla creado -> Se devuelve a la lista de tags -> se ingresa nuevamente a editar tag -> se da clic en el botón eliminar y luego en confirmar eliminar tag	https://uniandes-my.sharepoint.com/personal/oa_sanchez2_uniandes_edu_co/_layouts/15/stream.aspx?id=%2Fpersonal%2Foa%5Fsanchez2%5Funiandes%5Fedu%5Fco%2FDocuments%2FPruebas%20Automatizadas%2FPruebas%20Exploratorias%20Ghost%2FVideos%20Evidencias%20Pruebas%20Exploratorias%2FPE011%2Emp4&ga=1
10	Crear Miembro	Iniciar Sesión -> Crear un miembro con nombre con un string generado y un email generado -> click en el botón de guardar -> Validar que el miembro se haya creado mediante el botón "Saved"	https://uniandes-my.sharepoint.com/personal/oa_sanchez2_uniandes_edu_co/_layouts/15/stream.aspx?id=%2Fpersonal%2Foa%5Fsanchez2%5Funiandes%5Fedu%5Fco%2FDocuments%2FPruebas%20Automatizadas%2FPruebas%20Exploratorias%20Ghost%2FVideos%20Evidencias%20Pruebas%20Exploratorias%2FPE012%2Emp4&ga=1

2.5.4. Pruebas de reconocimiento

Las pruebas de reconocimientos nos permiten generar de forma automática casos de prueba sobre una ABP que permitan identificar defectos o comportamiento inesperados mediante la utilización de un robot automático que realiza operaciones aleatorias sobre el sistema.

Para ejecutar las pruebas de reconocimiento se utilizará la herramienta Cypress que nos permitirá generar eventos aleatorios sobre el ABP.

Como requerimiento tenemos que tener instalado NodeJS version 14.0 o superior

Para comenzar instalaremos Cypress mediante el comando

npm install cypress

Ahora clonaremos el repositorio con el comando

git clone https://github.com/slozano95/pruebas_semana5

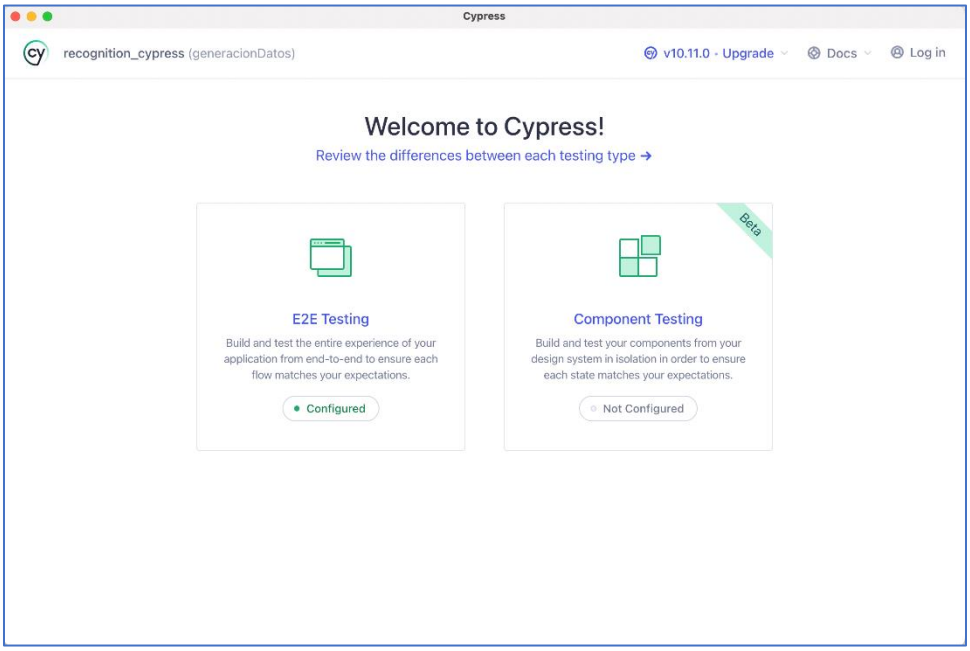
Navegamos a la carpeta ***recognition_cypress*** con el comando

cd recognition_cypress

Ahora deberá ejecutar cypress mediante el comando

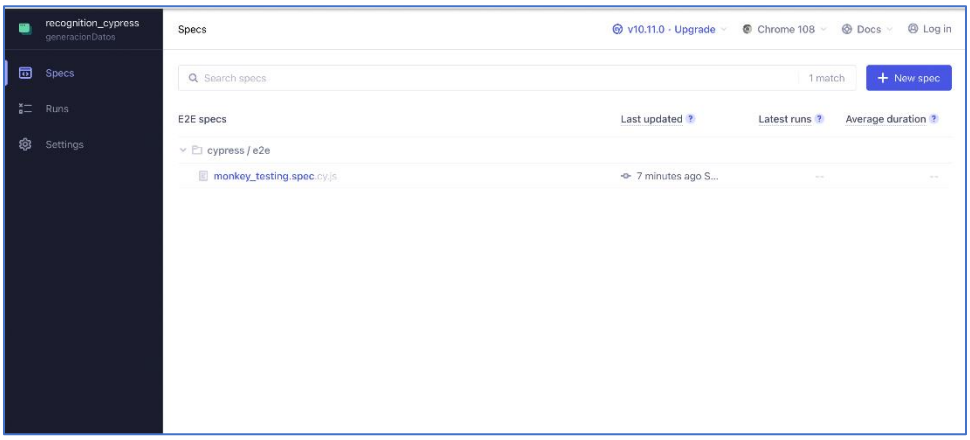
npx cypress open

Este comando abrirá una pantalla como la que ve a continuación.

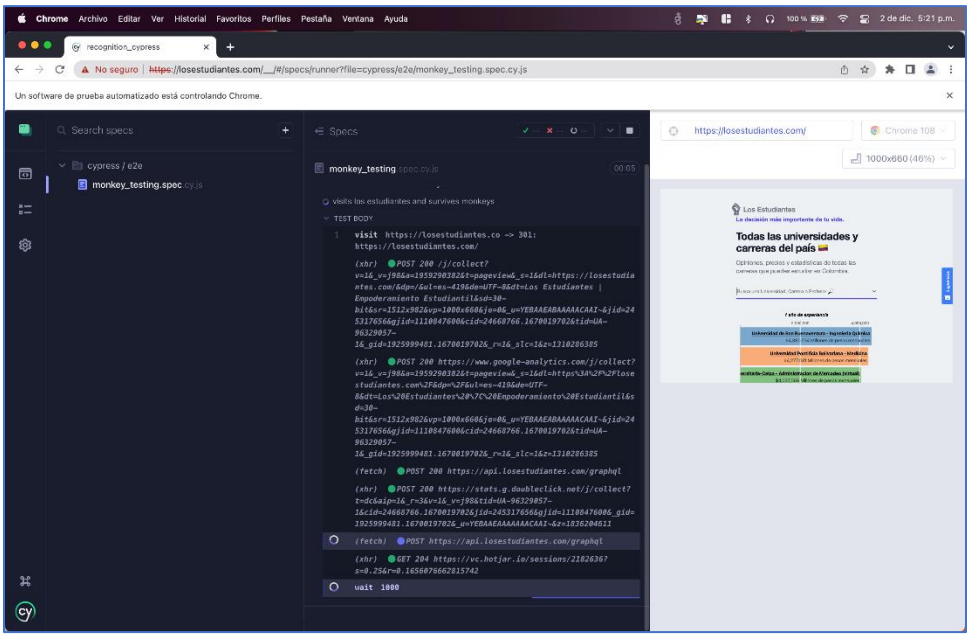


Haga click en la sección "E2E Testing", seleccione el navegador de pruebas, como Chrome y haga click en “Start testing in Chrome”

Verá la pantalla principal de cypress en Chrome como se muestra a continuación



Haga click sobre el archivo “monkey_testing.spec” y la ejecución del monkey iniciará



La pantalla está dividida en dos secciones principales, en la parte derecha verá la ejecución de la prueba automática en el navegador y en la parte izquierda verá las acciones que el script va ejecutando sobre el mismo.

En este caso se observa que se generan 30 eventos de forma aleatoría sobre la página "losestudiantes.co”

Para cambiar la página sobre la cual se generan los eventos, abrá el archivo “monkey_testing.spec.cy.js” en un editor de texto y reemplace la URL que encontrará en la línea 3 del archivo

Para complementar las pruebas de reconocimiento se puede utilizar un Ripper que permita obtener un mapa de conexiones de la ABP

Para esto cambiaremos a la carpeta “Ripper” y ejecutaremos el comando

node index.js url

Donde la url representa la página sobre la cual queremos realizar la ejecución del ripper

2.5.5. Pruebas E2E

Para la realización de las pruebas E2E, se han elegido las herramientas KRAKEN y CYPRESS. Estas herramientas nos permiten ejecutar pruebas a nivel de interfaz gráfica con una serie de scripts que ejecutan las diferentes acciones dentro del navegador, para al final realizar una validación del contenido esperado en la pantalla con el contenido que se muestra de cara al usuario.

Antes de iniciar el paso a paso de como ejecutar estas pruebas, cabe resaltar que el aplicativo GHOST se encuentra desplegado en la siguiente dirección URL para facilidad de las pruebas que se ejecutan en las herramientas.

URL GHOST: https://pruebasautomatizadas.digitalpress.blog/ghost/#/signin

Para llevar a cabo esta ejecución debemos cumplir con las siguientes premisas e instalaciones en nuestra máquina:

1. **Pruebas de Ghost en KRAKEN**

Por favor asegúrese de instalar las dependencias requeridas, las puede consultar aca.

Es importante que adicional a las dependencias mencionadas, tenga instalado:

Android Studio + SDK Appium, este se instala con el comando

npm install -g appium

Pasos de ejecución de pruebas

a. Descargue el repositorio con el comando

git clone https://github.com/slozano95/pruebas_semana5

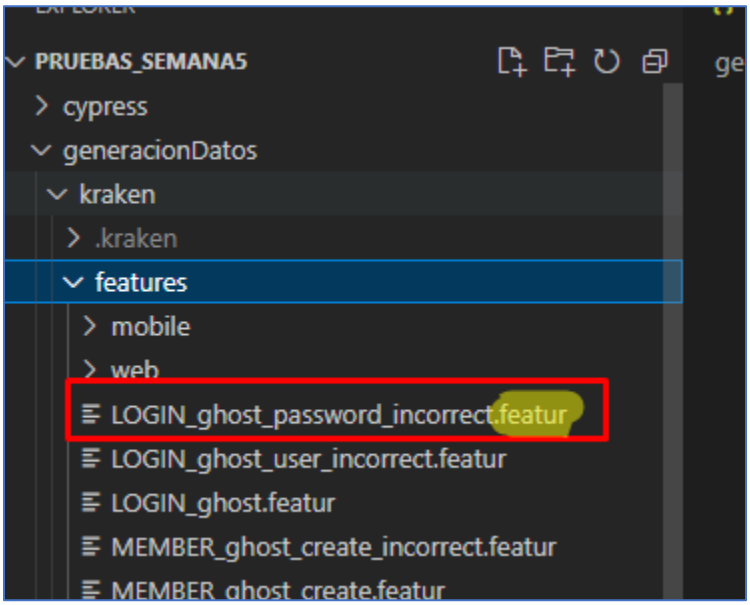
b. Navegue en la terminal hasta la carpeta 'kraken' con el comando

   cd kraken

c. Note que dentro de la carpeta de features existen varios archivos que NO tienen la extensión .feature.
d. Dado que Kraken no permite la ejecución de varios tests en serie, usted deberá modificar la extensión de la prueba que requiera ejecutar y asegurarse que sea '.feature'.

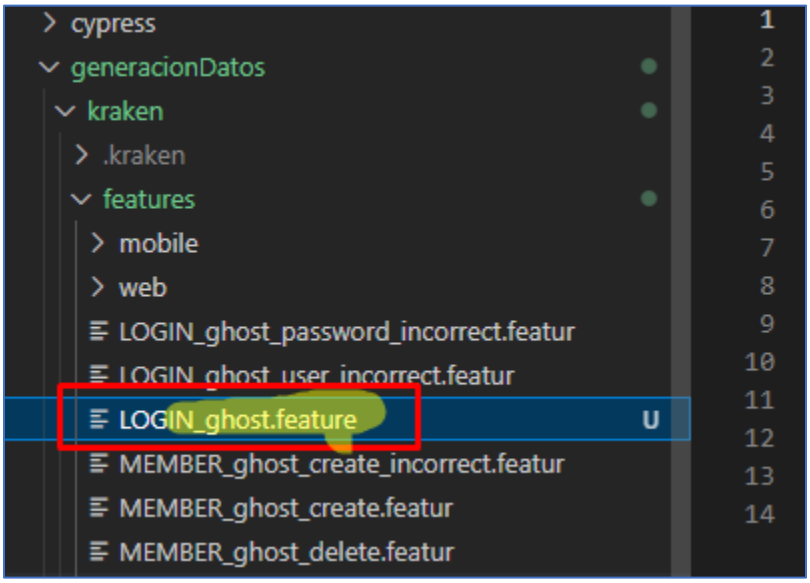
   Ejemplo:

   Nombre **original** del archivo: MEMBER_ghost_create.featur



   Nombre que **debe quedar** para poder ejecutar la prueba:

   - MEMBER_ghost_create.feature



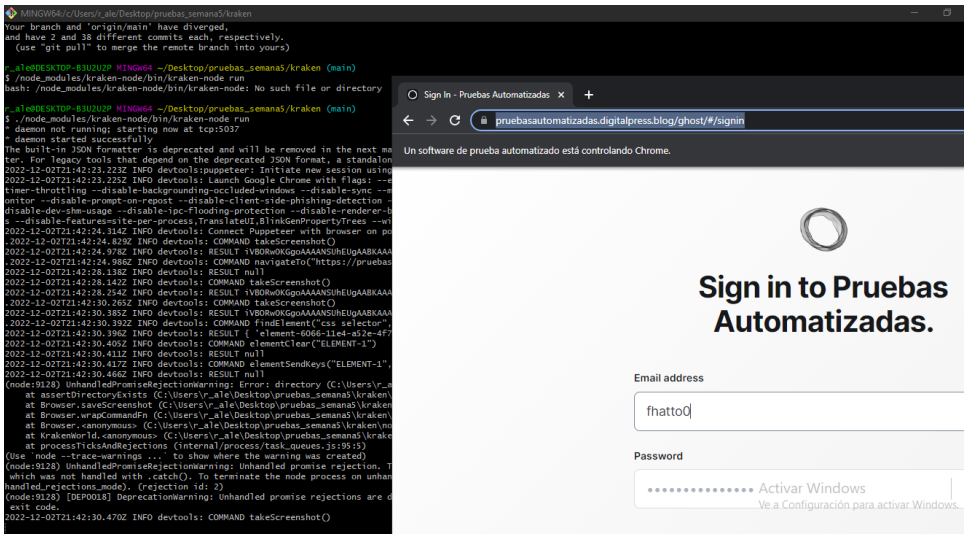
Es importante, por lo mencionado anteriormente, que exista un único archivo con la extensión '.feature', por lo tanto, si está probando múltiples archivos no olvide ir ajustando la extensión de los archivos.

Ejecute el comando

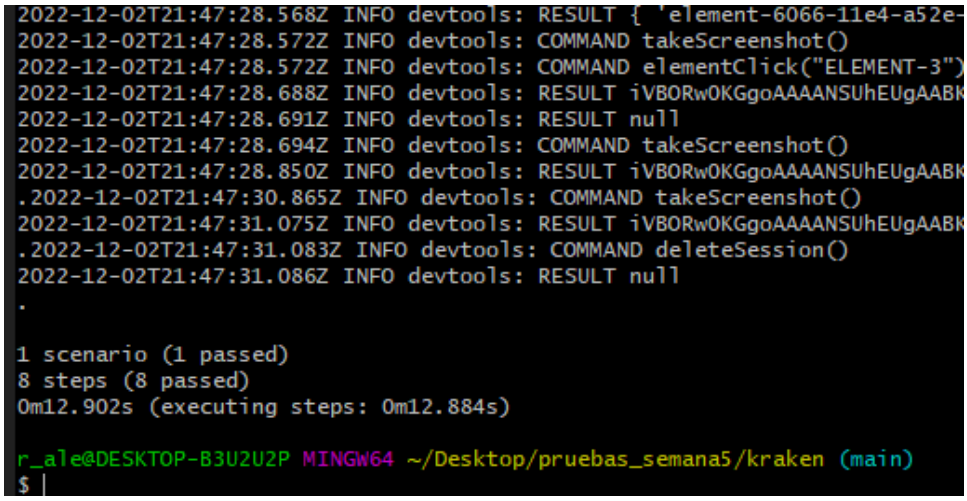
./node_modules/kraken-node/bin/kraken-node run

La ejecución de la prueba con ese comando se verá de la siguiente manera:

1. Se ejecutará una ventana de navegador donde se podrá visualizar el paso a paso:



2. Cuando la prueba finalice se visualizará el resultado en la consola.



2.5.6. Pruebas de Ghost en CYPRESS

Por favor asegúrese de instalar las dependencias requeridas, las puede consultar aca.

Pasos de ejecución de pruebas

a. Descargue el repositorio con el comando

   git clone https://github.com/slozano95/pruebas_semana5

b. Navegue en la terminal hasta la carpeta monkey-cypress con el comando

   cd monkey-cypress

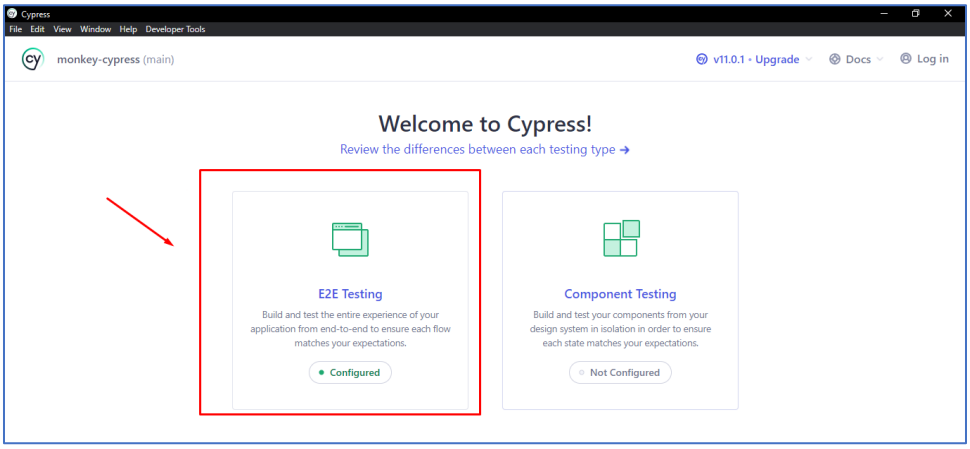
c. Ejecute el comando

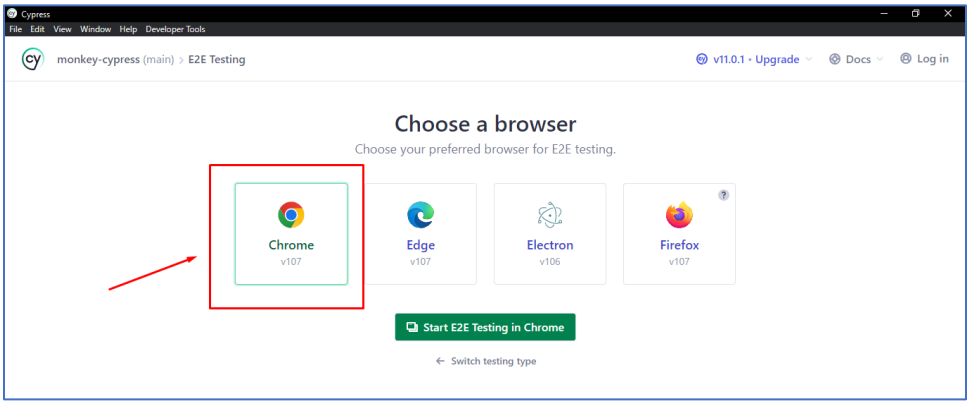
   npm install

d. Verifique que se realizó la instalación de todas las versiones mencionadas previamente
e. Ejecute las pruebas con el siguiente comando:

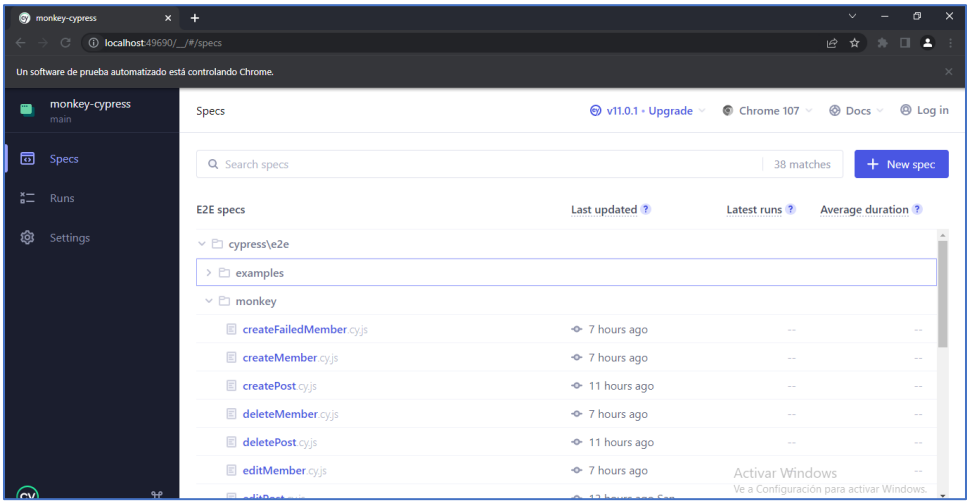
   cypress open

f. Se desplegará la siguiente ventana, lo cual seleccionaremos la opción de pruebas e2e y el navegador Google Chrome

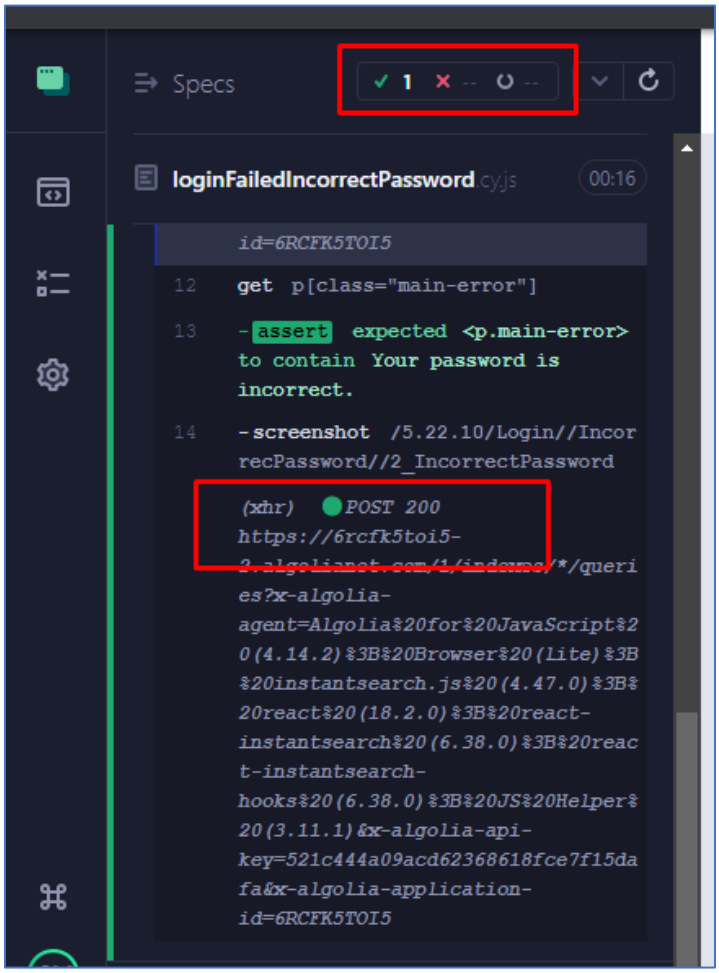




Y posteriormente se visualizarán en la ruta e2e/monkey, los diferentes archivos .cy que contienen las pruebas para cypress.



Los resultados de las pruebas se podrán obtener al dar click en un archivo .cy de la carpeta monkey, el cual ejecutará el paso a paso de los scripts y por ultimos nos reportará si la prueba fue exitosa o no como se ve en la siguiente imagen.



2.5.7. Pruebas VRT

A continuación, presentamos una tabla que muestra algunas pros y contras de las herramientas que se pueden utilizar para realizar regresiones visuales.

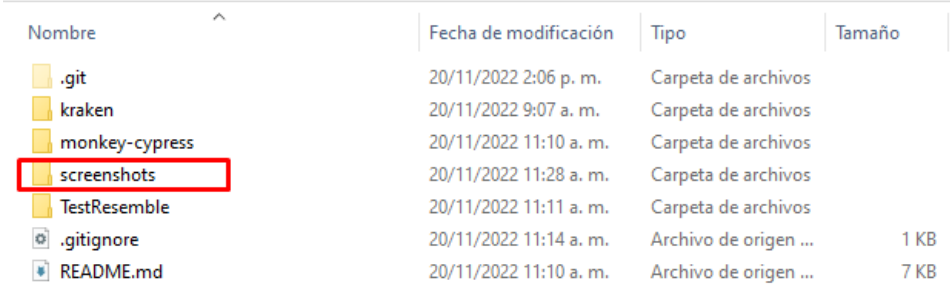
ResembleJS	BackStopJS
PRO Facilidad de generar reportes, los genera en formato estandar HTML	**CONTRA** Implementación más larga
PRO Utilización de escenarios	**CONTRA** No los incluye por defecto
CONTRA Basado en Puppetter, solo utiliza Chrome	**PRO** Permite extensibilidad a navegadores
CONTRA Comparación visual obtenida a partir de la posición absoluta de los pixeles	**CONTRA** Comparación visual obtenida a partir de la posición absoluta de los pixeles
CONTRA No se tiene en cuenta contexto alguno	**CONTRA** No se tiene en cuenta contexto alguno

Con el fin de realizar la comparación visual de los screenshot que fueron tomados en cada uno de los pasos de los escenarios generados en las 2 versiones en prueba, se ha utilizado la herramienta como es Resemblejs, la cual es posible realizar la comparación visual de 2 imágenes indicadas y mostrar sus diferencias, inicialmente por defecto esta herramienta utiliza otra que permite generar la prueba E2E como es playwright, pero para nuestro efecto usamos herramientas diferentes como lo es Cypress y Kraken, estas 2 fueron modificadas para generar los screenshot y dejarlos en una ruta especifica.

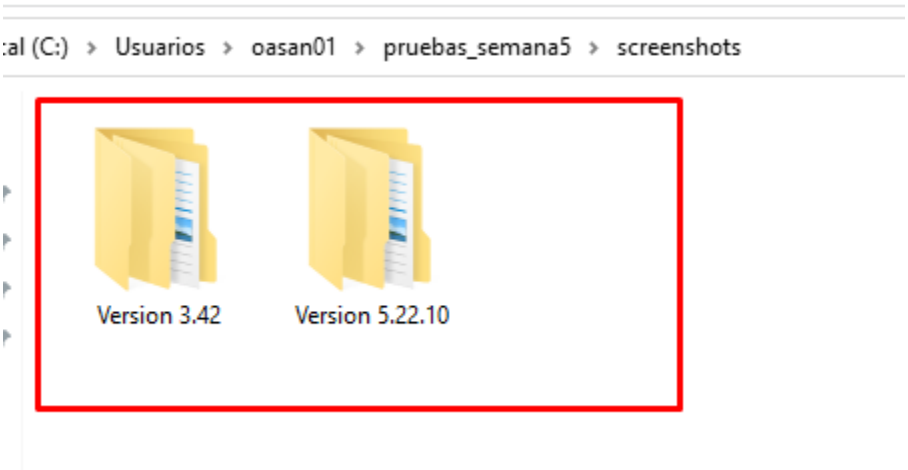
Por tanto, se realizó una modificación para dada las 2 rutas de las imágenes a comparar, la aplicación lee la estructura de carpetas y archivos, realiza el recorrido de cada directorio con este la construcción del reporte html, comparando cada escenario y cada paso realizado.

Para llevar a cabo esta ejecución debemos cumplir con las siguientes premisas.

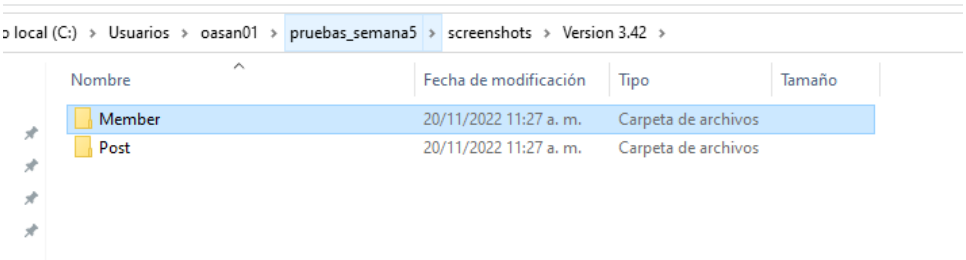
I. Ubicar el directorio donde se han generado las imágenes tomadas de cada versión. Como se ha indicado en pasos anteriores para ejecución de los escenarios de cada herramienta. un ejemplo de esta estructura es la siguiente.



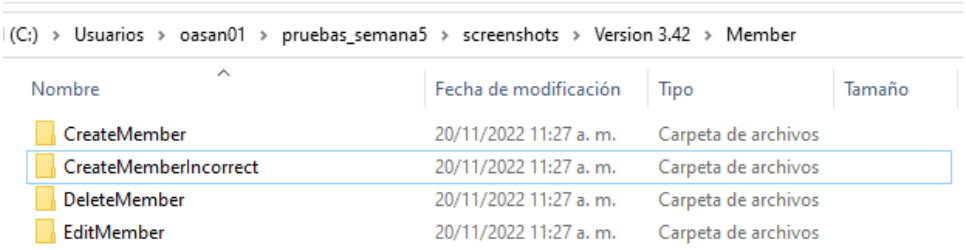
Esta imagen podemos identificar la carpeta "screenshots" en la raíz del proyecto allí es donde la ejecución de las pruebas hechas por ejemplo con "Kranken" a dejado depositada nuestras screenshots de cada paso, identificado con la versión en la aplicación objeto de las pruebas en este caso Ghost versiones (3.42) y (5.22.10).



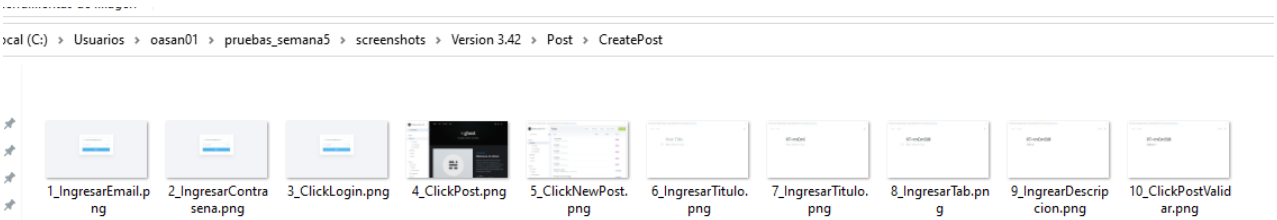
Si ingresamos a una de las versiones encontraremos los subdirectorios con las funcionalidades en pruebas. Ejemplo (Member, Post, Page, Tags ..)



Y dentro de cada una de estas encontramos los escenarios generados para cada funcionalidad. Ejemplo (CreateMember, CreateMemberIncorrect)

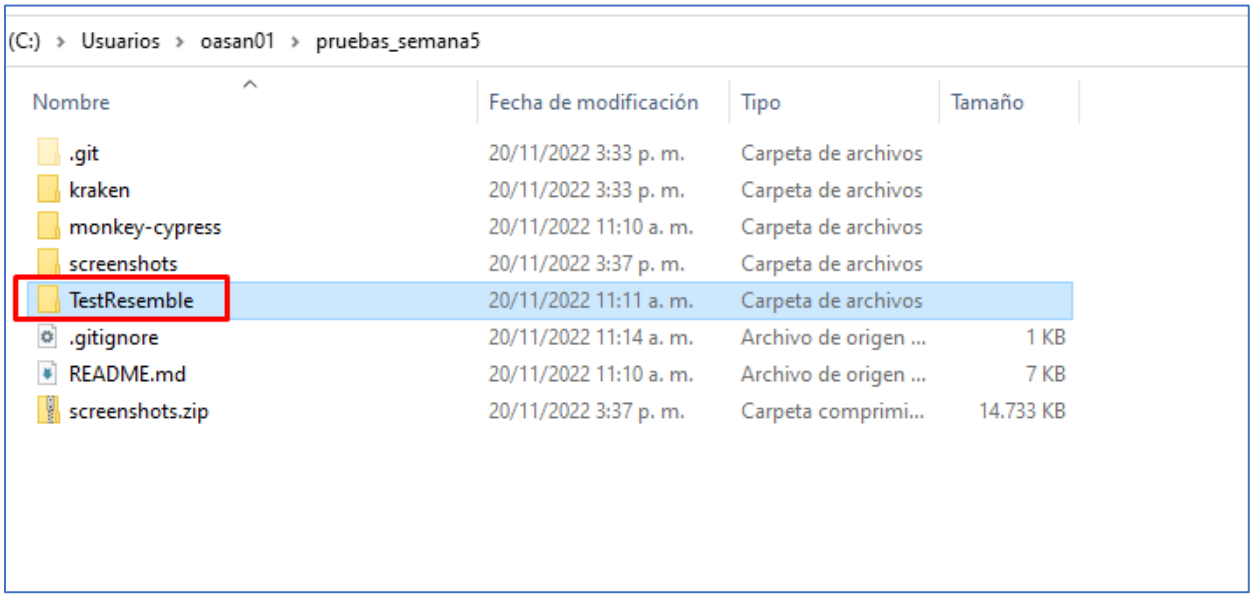


Por ultimo, dentro de estos encontramos las imágenes que se van a comparar.

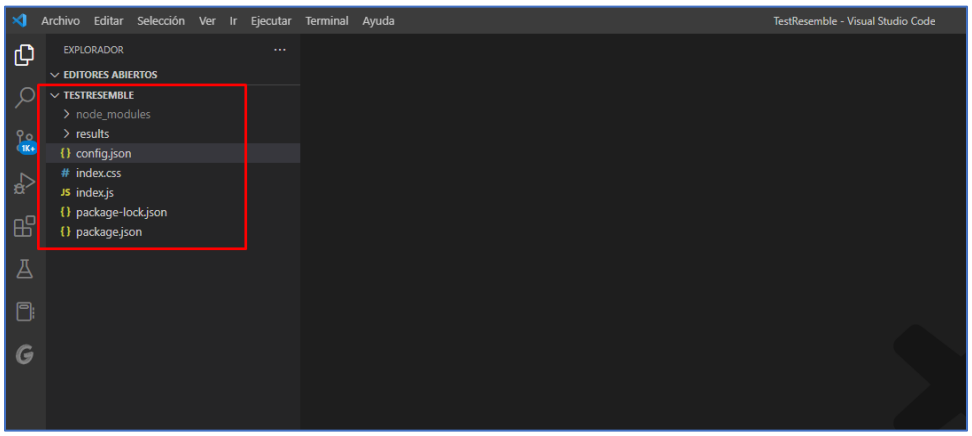


II. Una vez identificados los directorios con las imágenes a comparar procedemos a ejecutar nuestro reporte VRT de la siguiente manera

Para esto una vez descargado el repositorio y generadas las pruebas para cada versión ya sea con Kraken o Cypress. Debemos abrir con la aplicación VSCode la carpeta del proyecto "TestResemble".



Una vez cargado el proyecto en visual studio code.



Ubicamos el archivo config.json

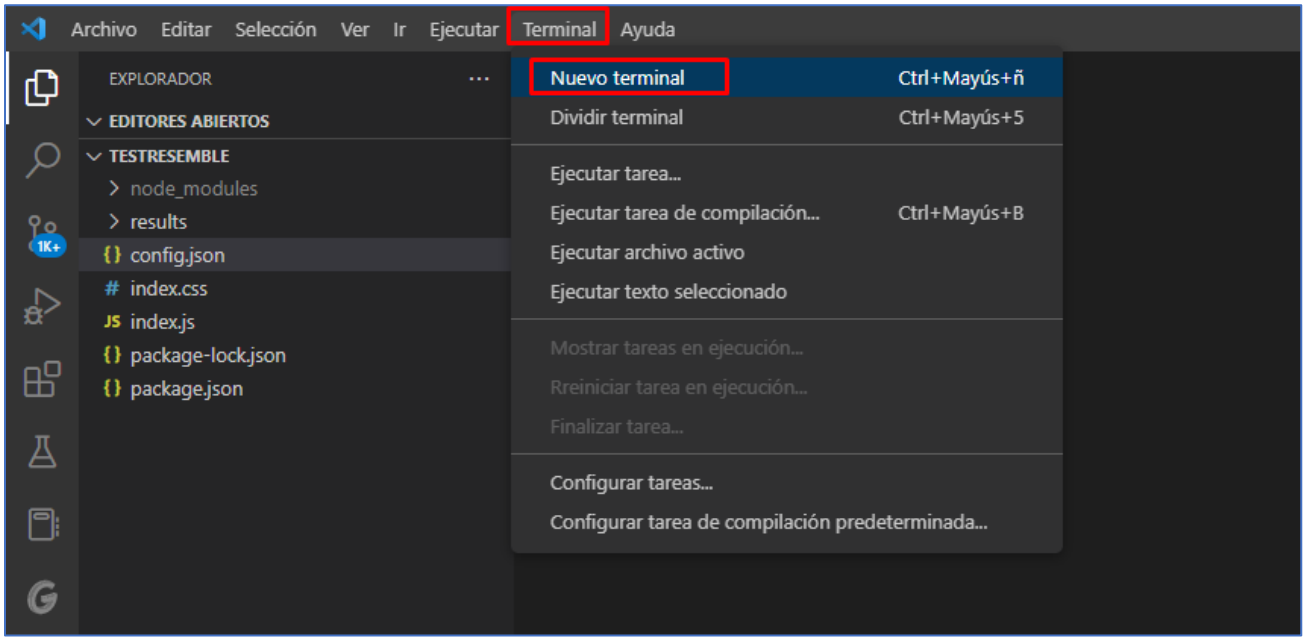
En este debemos cambiar tres parámetros

- **título**: En este podemos colocar el título que tendrá nuestro reporte al momento de generarlo. Ejemplo (Ghost)

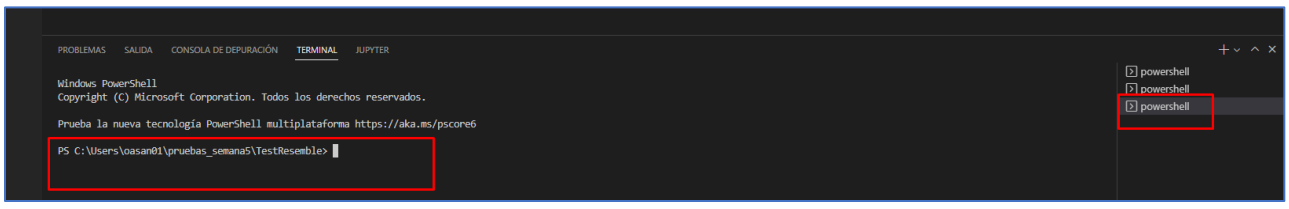
- **versionBefore**: Esta se configura la ruta donde encontramos los pantallazos de la versión uno a comparar. Ejemplo (../screenshots/Version 5.22.10), como vemos en este caso es una ruta relativa ya que la carpeta raíz donde se encuentran las imágenes esta un nivel anterior donde está el archivo de configuración.
- **versionAfter**: Esta se configura la ruta donde se encuentra los pantallazos de la versión dos a comparar. Ejemplo (../screenshots/Version 3.42), como vemos en este caso es una ruta relativa ya que la carpeta raíz donde se encuentran las imágenes esta un nivel anterior donde está el archivo de configuración.

Una vez terminados los cambios guardamos y cerramos el archivo.

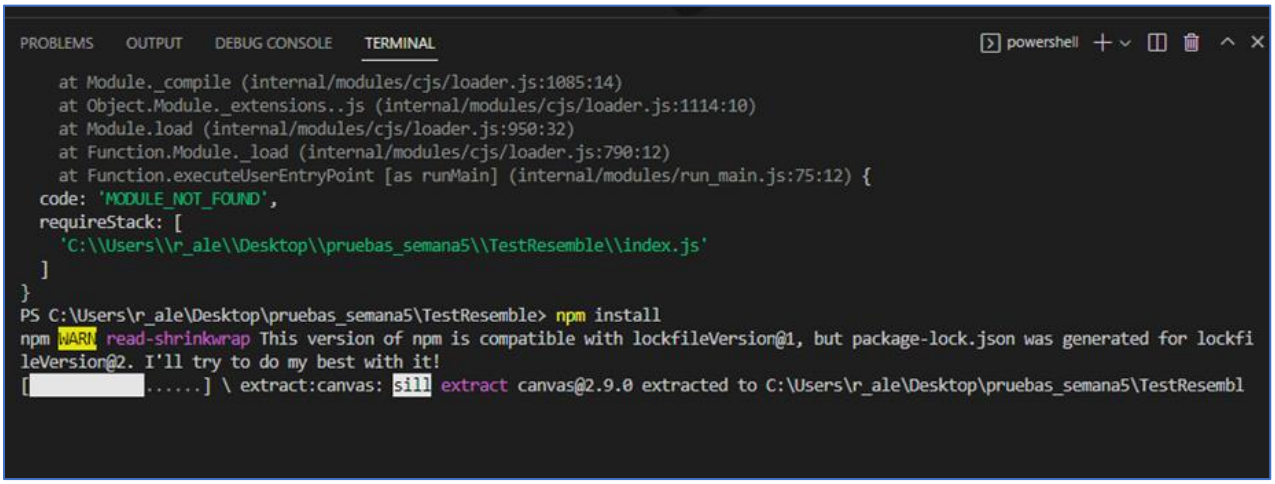
Teniendo lo anterior listo ahora simplemente corremos nuestro reporte para que sea generado. para esto abrimos una venta de comandos desde visual studio code.



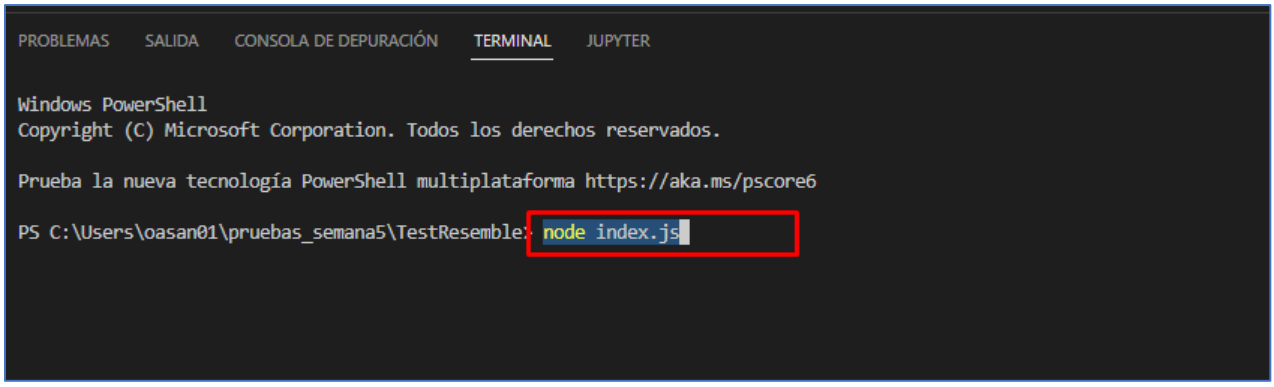
Esto nos desplegara la consola de comandos



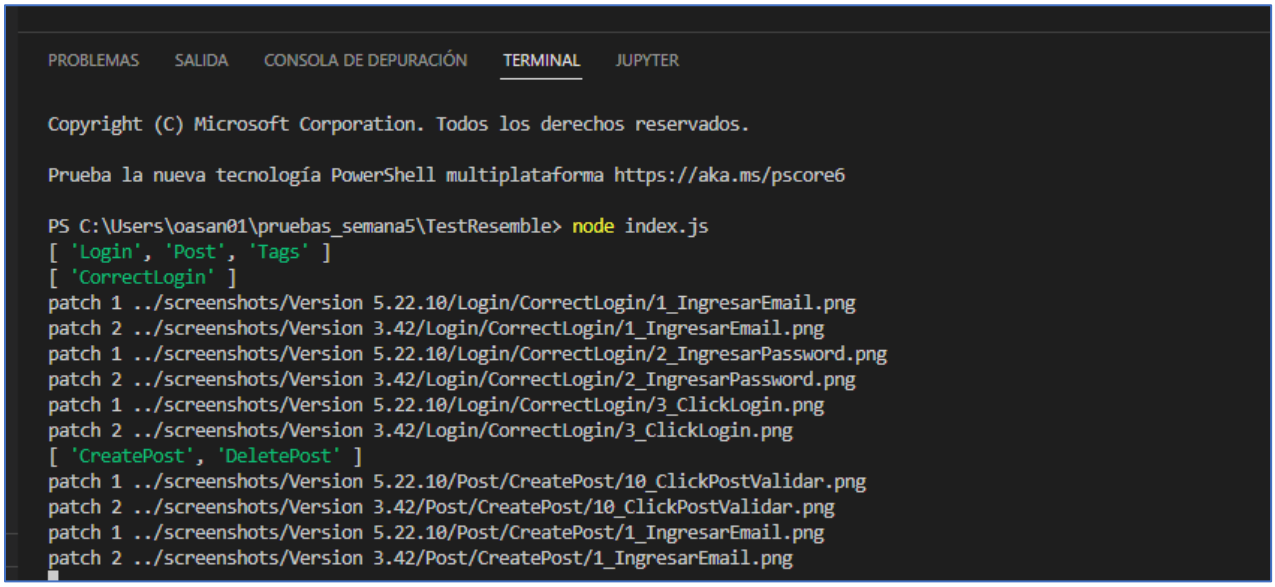
Si es la primera vez que ejecuta estas pruebas del proyecto, se deben de instalar las dependencias con el comando NPM INSTALL



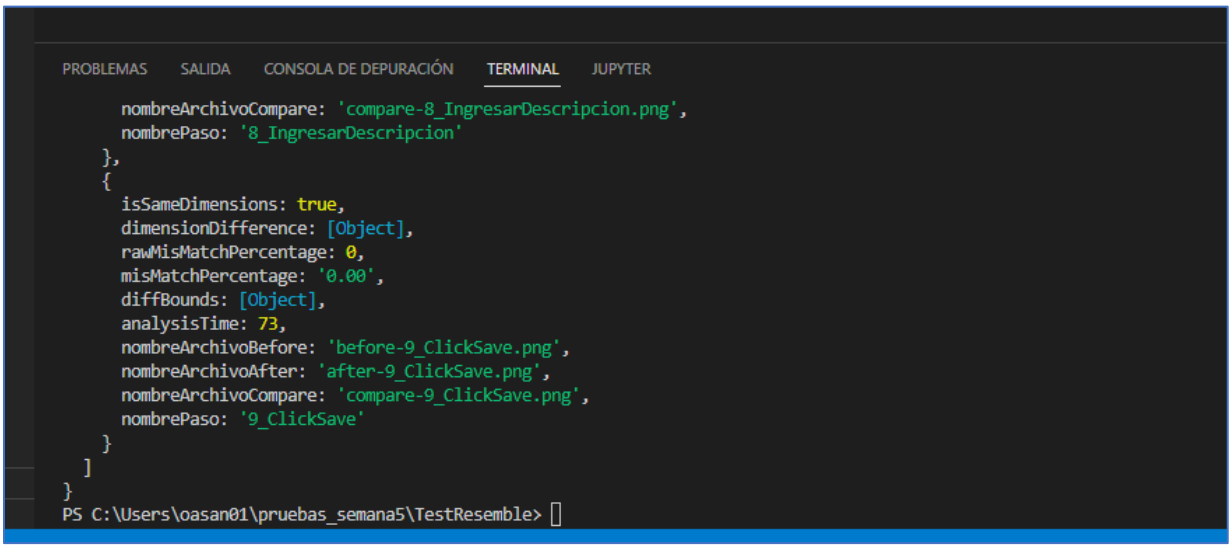
En donde ingresamos el comando "node index.js"



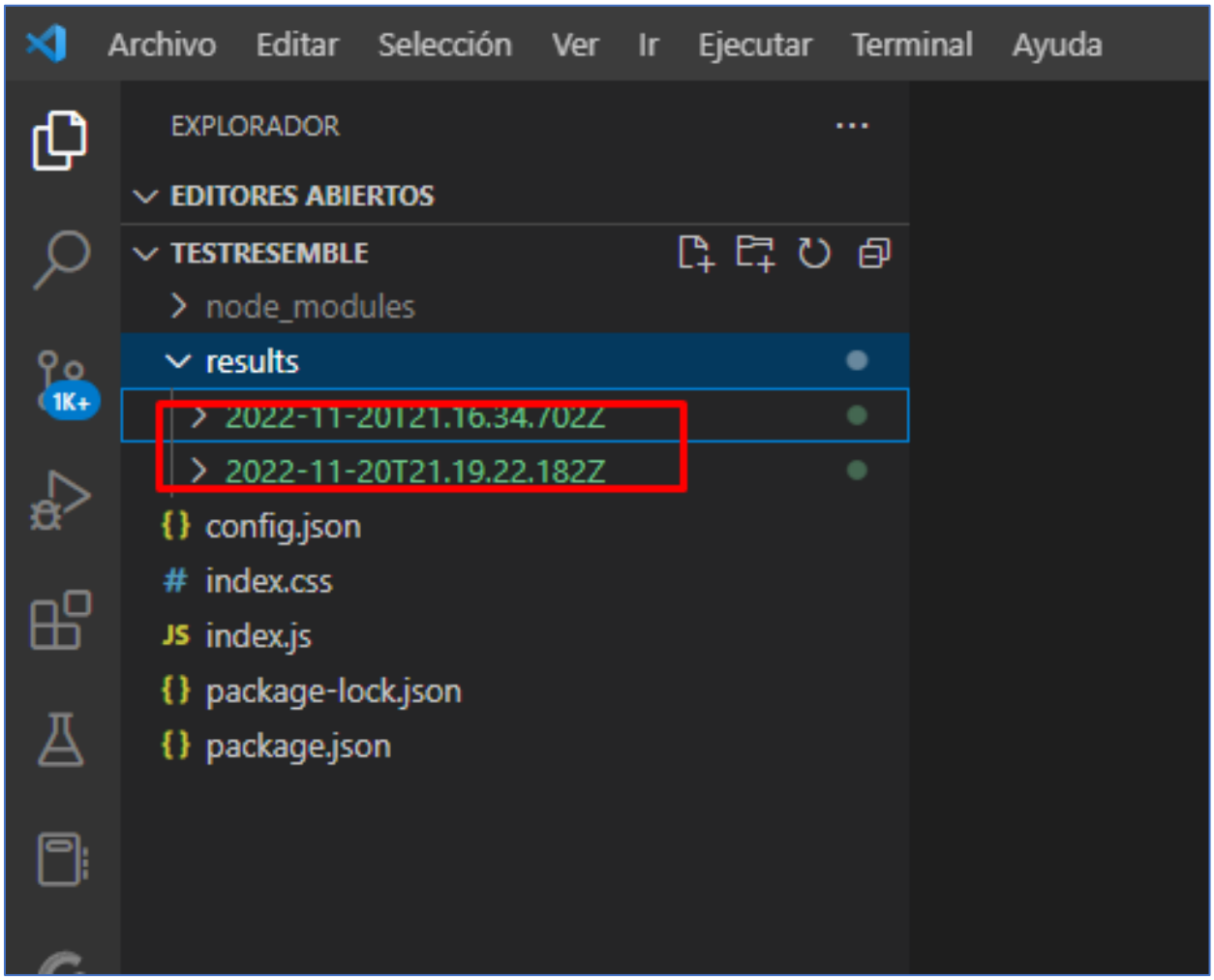
Esto iniciara el proceso de generación del reporte.



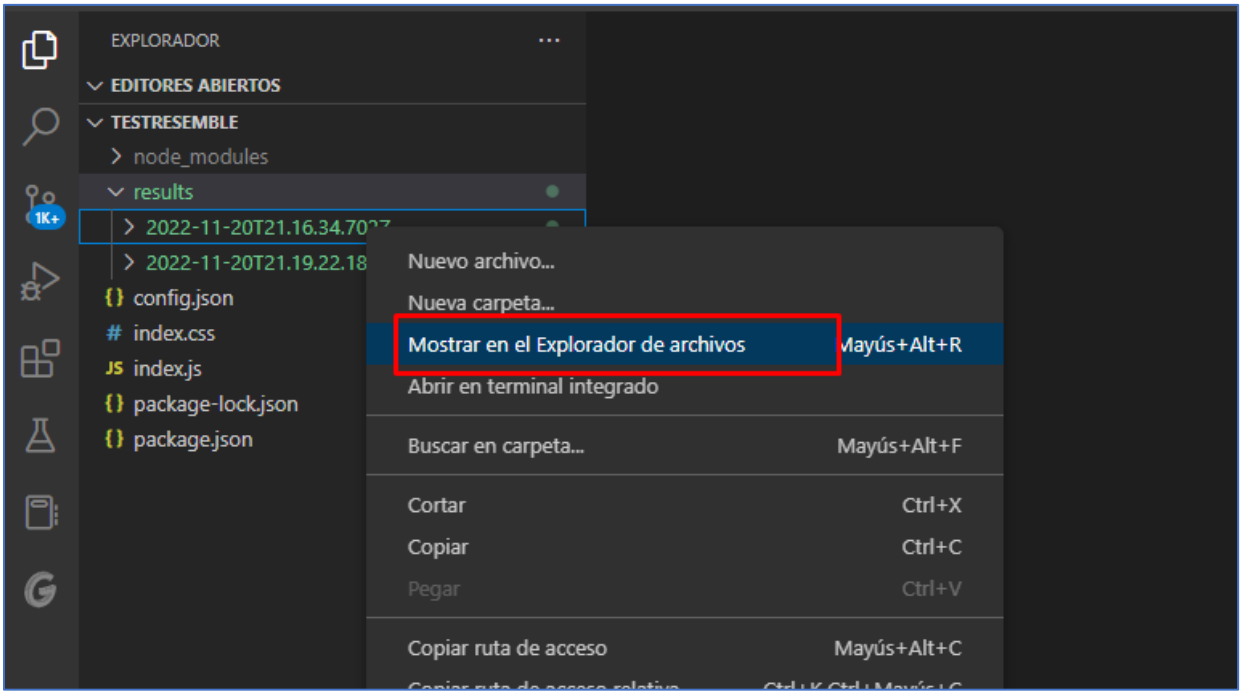
Una vez terminado el proceso.



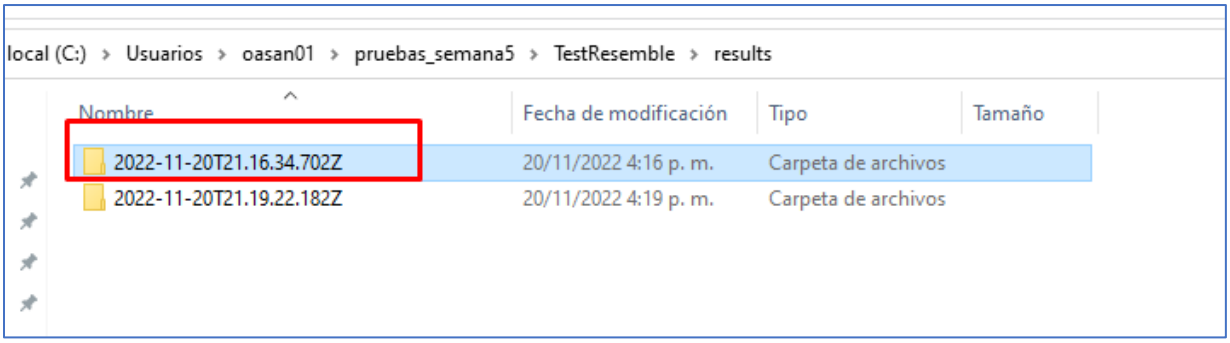
Generara una carpeta en el directorio "result" del proyecto, generara una por cada ejecución que hagamos.



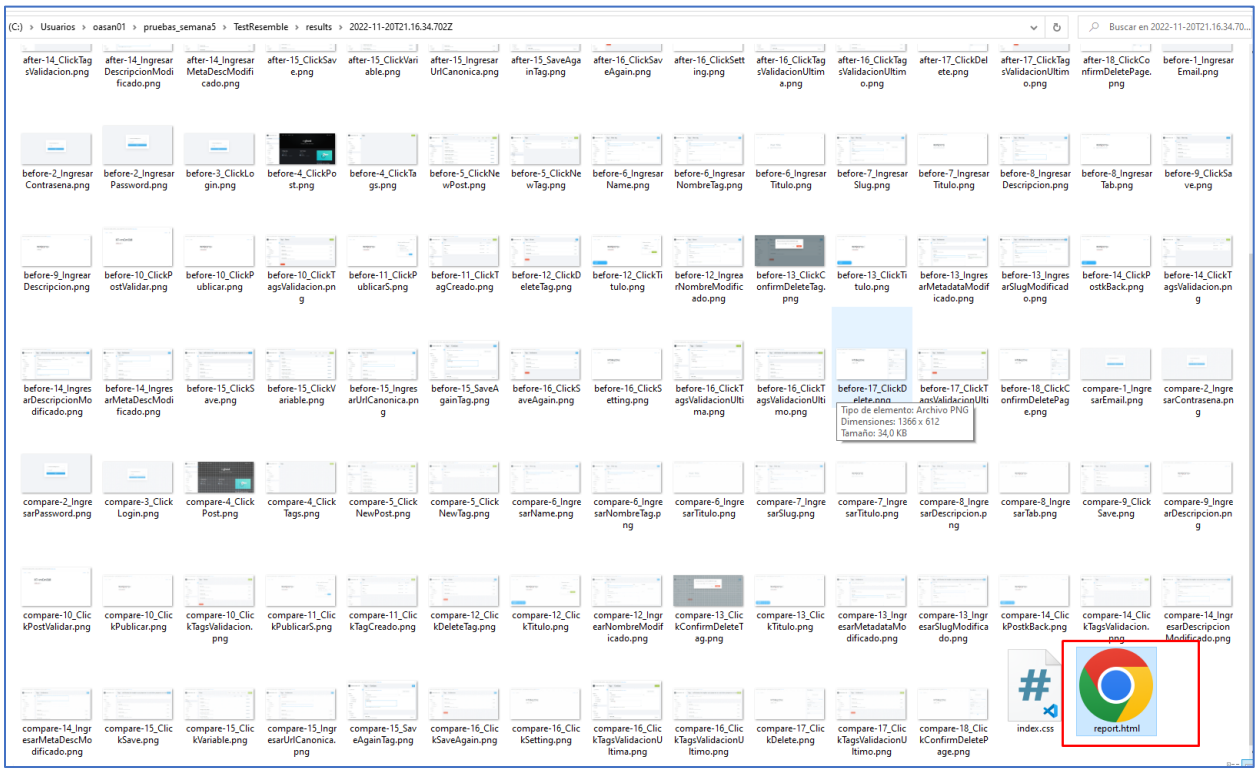
Una manera fácil de ir a la ruta del reporte es dando clic derecho a la carpeta del reporte generado y luego dando clic en la opción "Mostrar en el explorador de archivos".



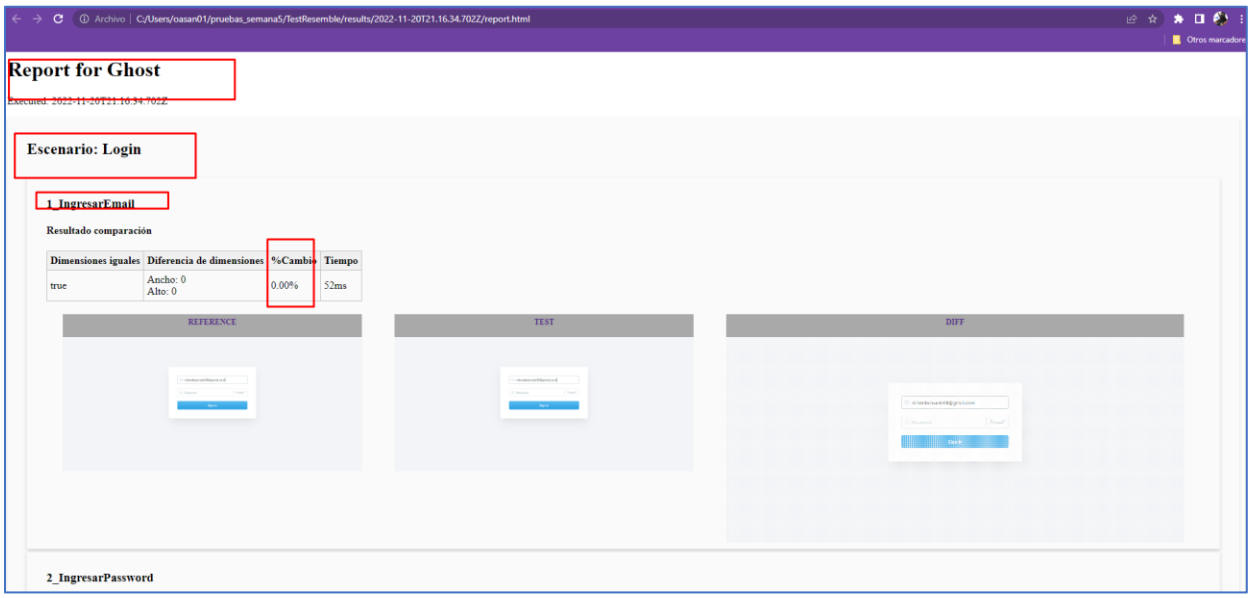
Ingresamos a la carpeta.



Allí encontraremos las imágenes que han sido comparadas y un archivo con nombre "report.html" al cual damos doble clic y se abrirá en el navegador.



En este reporte podemos navegar en la comparación de los pantallazos de cada uno de los pasos generados en los escenarios de pruebas. También podemos determinar el porcentaje de diferencia entre una y otra y en qué lugar de esta las podemos ver.



2.5.8. Escenarios de validación de datos.

Con el fin de ampliar nuestro radio de acción en la ejecución de las pruebas se utilizaron 3 estrategias en la generación de datos con el fin de evaluar escenarios de pruebas de las diferentes funcionalidades de la aplicación, esto permitirá generar una mayor cobertura de escenarios posibles de pruebas y validar en correcto funcionamiento en varios ámbitos.

Nuestra estrategia de generación de Datapool se basó en la abstracción del proceso que permitiera con una solo entrada y parámetros poder definir una estrategia de generación de datos (a-priori, Pseudo-Aleatorio y Aleatorio).

Para el manejo del repositorio de datos se implementó una clase *DataPool* que permite el manejo de las diferentes fuentes disponibles para generar datos.

La clase se encarga de recibir un origen de datos y la carga util del mismo. Siendo el origen de datos un valor del enumerador **PoolOrigin** y la carga útil una url o un archivo json.

A partir de estos parámetros se gestionará un arreglo en memoria al cual se accede para obtener los datos según son solicitados utilizando la expresión **DataPool.get("fieldName")** donde **fieldName** corresponde al nombre del campo cargado desde el payload o carga útil.

Si el conjunto de datos contiene más de un registro, la clase escogerá de manera aleatoria un registro de los disponibles y este será el utilizado para entregar datos relacionados al escenario de prueba.

- enum PoolOrigin

Describe los posibles valores del origen del pool de datos.

```
export const PoolOrigin = {
             APriori: 0,
                 Pseudo: 1,
             Random: 2
      }
```

- Método prepare(origin, payload)

```
Parámetros:
               - origin: PoolOrigin
              - payload: carga útil para como entrada de datos, puede representar una URL o un
      a                   archivo de tipo JSON
```

Ejemplo de uso: `DataPool.prepare(PoolOrigin.APriori, fileData)`

- Método get(fieldName)

```
Parámetros:
     - fieldName: string, representa el nombre del campo a obtener desde el conjunto de datos
```

Ejemplo de uso: `DataPool.get("MetaDataTitle")`

Para implementar el **DataPool**, debemos invocar la preparación del mismo a partir de la expresión **DataPool.prepare(origin, payload)** con los parámetros correspondientes según sea el caso. Esta invocación debe ser realizada en el paso *beforeEach* de un escenario de pruebas de Cypress

Para utilizar un campo debemos usar la expresión **DataPool.get("fieldName")** donde **fieldName** corresponde al nombre del campo cargado desde el payload o carga útil.

Estrategia de Generación de datos

A-priori

Esta técnica busca obtener un conjunto de datos antes de la ejecución de los escenarios mediante la carga de archivos de tipo JSON.

Para la generación de estos archivos utilizamos Mockaroo, que nos permite crear schemas de datos y luego generar registros a partir de este schema.

En la imagen se muestra un schema de datos que contiene 5 campos con diferentes tipos de datos asociados a cada uno.

```
[{
  "title": "DCK",
  "url": "1",
  "body": "Dahl Creek Airport",
  "meta_title": "Z21x4mk205TD0JAtxh3Im1478l533E500351jc7z683099rqC9764618B1h17",
  "meta_description": "EE0jRX1yam1kZ00FJhH6YOI30ns2fg6B85XkD653O11jW4O4mA57M14q71quGP06KyU2y4H9089Ow20SOD7H3136K2vJ2u1
}, {
  "title": "ILP",
  "url": "test",
  "body": "Île des Pins Airport",
  "meta_title": "3N9y5E629Z37A1A31XiF0Irv5Y563f87QK08A52459zb3L0Ob6N5T268R820P",
  "meta_description": "55g6280560N9638Y8j128rq03qHK1H5S02X4v8NF85208994jOD3620ziJ5878471x2398pm26r8I36MeZk2JA3E8r4yLq2
}, {
  "title": "CEW",
  "url": "בְּרֵאשִׁית, בָּרָא אֱלֹהִים, אֵת הַשָּׁמַיִם, וְאֵת הָאָרֶץ",
  "body": "Bob Sikes Airport",
  "meta_title": "ArE81u18unUR750WP675uj9z3122h5OSmAhy313fSh9x4HQPX1Lb19ZVx74n6",
  "meta_description": "4102V060DP4v58oae269PtV6N325WlL53h2b463VI7y90j44431j7NmkL455OD95bO0STCV585CQ925T6V9GbJ9S64449pDM
}, {
  "title": "PJG",
  "url": "ᵬ",
  "body": "Panjgur Airport",
  "meta_title": "9lXBtm7f7NqJT1n8vC11ktZ68520e578195272P02uy30c751FMWYLht9mRe3",
  "meta_description": "53A383jTOxGIPC224aba3RZ62M68x16W42n3e2k94966IqQ25521nG439nR6X10J6556UjXgq282X4Ehes1XO5dikv4n692M
}, {
  "title": "VAA",
  "body": "Vaasa Airport",
  "meta_title": "08p4S34vAO99Rn74736p6mjp4La7Sy01288G3191226a37u8J5F2cN5972o38",
  "meta_description": "60y4089U5Y4RoNJB57D18Cg28NY1wP9a553p54K701CykwK5rshZ6SJBi4Pz2968I21Za2fhb82YS72d3Bw7AI08d5iaS9D
}, {
  "title": "KUN",
  "url": "-1.00",
  "body": "Kaunas International Airport"
```

En la imagen se muestra un conjunto de datos generados a partir del schema presentado anteriormente.

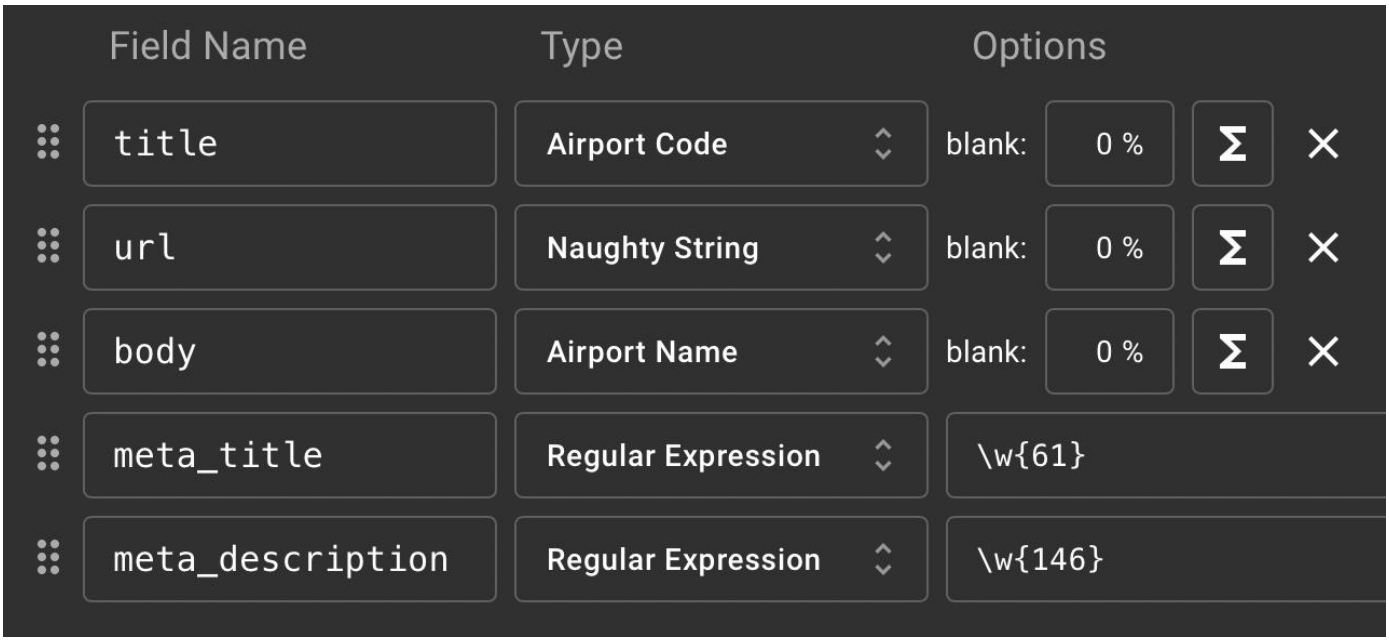
Implementación desde DataPool

Debemos utilizar la expresión *cy.readFile* para leer el archivo desde Cypress y luego entregarlo al DataPool con la expresión **DataPool.prepare**

```
cy.readFile('./mocks/13.json').then((fileData) => {
     DataPool.prepare(PoolOrigin.APriori, fileData);
})
```

Pseudo-aleatorio dinámico

Esta técnica busca obtener un conjunto de datos de forma dinámica y aleatoria, para este caso utilizamos Mockaroo para generar un schema y obtenemos los datos directamente desde la URL expuesta del schema creado. Esto nos permite realizar ajustes sobre el schema sin necesidad de descargar un nuevo archivo JSON.




```
curl "https://api.mockaroo.com/api/d88abf50?count=100&key=94e8ade0" > "cypress_39.json"
```

La imagen muestra un comando curl que permite obtener los datos desde la URL, para nuestro caso, no se utiliza el comando curl sino unicamente la URL generada por Mockaroo

Implementación desde DataPool

Debemos utilizar la expresión **DataPool.prepare** con el **PoolOrigin.Pseudo** y la URL de Mockaroo de la cual se obtendrán los datos.

```
var mockarooUrl = "https://my.api.mockaroo.com/users.json?key=79ad7410";
await DataPool.prepare(PoolOrigin.Pseudo, mockarooUrl);
```

Aleatorio

Esta técnica busca generar datos de forma aleatoria mediante la definición de un schema que representa los tipos de datos a generar y su tamaño

Un schema de ejemplo es:

```
[{
    "title": "string_3",
    "url": "string_3",
    "body": "string_10",
    "twitter_title": "string_301"
}]
```

Donde el nombre del campo se mantiene, y el contenido representa un tipo de dato separado por un _ del tamaño del dato a generar. Para el caso anterior, tenemos 4 campos, todos de tipo string y sus tamaños serían respectivamente: 3 caracteres, 3 caracteres, 10 caracteres y 301 caracteres.

Cada tipo de dato se mapea dentro del DataPool a una expresión de generación de datos de la librería faker-js como se muestra a continuación:

```
static async prepareRandomOrigin(basicStruct, f) {
    let json = JSON.parse(JSON.stringify(basicStruct));
    json.forEach(element => {
        var data = {}
        Object.entries(element).forEach(entry => {
            const [key, dataType] = entry;
            var components= dataType.split("_");
            var size = components[1] ?? 10;
            var randomData = "";
            switch(components[0]) {
                case "string":
                    randomData = faker.random.alphaNumeric(size)
                    break;
                case "email":
                    randomData = faker.internet.email()
                    break;
                case "integer":
                    randomData = faker.random.number(size)
                    break;
                case "wierd":
                    randomData = faker.datatype.string(size).replaceAll("{", "")
                    break
                case "naughty":
                    var possible = "(ノಠ益ಠ)ノ彡┻━┻ 1 2";
                    for (var i = 0; i < size; i++)
                      randomData += possible.charAt(Math.floor(Math.random() * possible.length));
                    break
                case "emoji":
                    var possible = "😀😃😄😊😆😜😅😂🤣🥲🙃🙂😇☺😊";
                    for (var i = 0; i < size; i++)
                        randomData += possible.charAt(Math.floor(Math.random() * possible.length));
                break
                case "url":
                    randomData = faker.internet.url()
                    break
                default:
                    randomData = dataType
                    break;
            }
            data[key] = randomData;
        });
        this.data.push(data);
    });
    cy.log("RANDOM DATAPOOL READY TO USE");
    cy.log("Fields available: "+JSON.stringify(this.data[0]));
}
```

Implementación desde DataPool

Debemos utilizar la expresión **cy.readFile** para leer el archivo desde Cypress y luego entregarlo al DataPool con la expresión **DataPool.prepare**

```
cy.readFile('./mock_structs/90.json').then((fileData) => {
    DataPool.prepare(PoolOrigin.Random, fileData);
})
```