**3.1 Introducción al capítulo**

Este capítulo presenta el proceso de implementación de la solución propuesta para la automatización de pruebas del módulo Carga y Descarga del sistema de planificación de recursos de Habana Club International. La implementación constituye la materialización práctica de los fundamentos metodológicos y técnicos establecidos en el capítulo anterior, demostrando su aplicabilidad y eficacia en un entorno de desarrollo real.

Para estructurar el proceso de implementación y facilitar una adopción gradual de las prácticas de automatización, se seleccionó el sub-módulo "Chofer" como caso de estudio inicial. Esta elección no fue arbitraria, sino que responde a diversos factores estratégicos:

En primer lugar, el sub-módulo "Chofer" representa un componente fundamental dentro del ecosistema logístico de Carga y Descarga, ya que gestiona la información de los conductores autorizados para realizar operaciones dentro del sistema. Su correcto funcionamiento es crítico para garantizar que solo personal debidamente registrado y autorizado pueda participar en los procesos logísticos.

En segundo lugar, este sub-módulo presenta una complejidad técnica intermedia, incorporando validaciones específicas como la verificación del formato del carnet de identidad con reglas dependientes de fechas que permiten probar capacidades avanzadas del framework de automatización, sin alcanzar la complejidad extrema de otros componentes del sistema que incluyen integraciones múltiples.

Adicionalmente, la elección se basó en consideraciones prácticas relacionadas con la madurez de su desarrollo y documentación. Al contar con historias de usuario bien definidas y reglas de negocio claras, facilitó la creación de casos de prueba estructurados que sirven como modelo para futuras implementaciones.

El proceso de implementación siguió la metodología establecida previamente y la ejecución en modo headed permitió verificar visualmente la correcta interacción de los scripts con la interfaz del sistema.

Las secciones siguientes detallarán cada fase de este proceso, desde la configuración del entorno de desarrollo hasta la validación final de los resultados, documentando tanto los aspectos técnicos de la implementación como los desafíos encontrados y las soluciones aplicadas.

## 3.2 Configuración del entorno de pruebas

La implementación efectiva de pruebas automatizadas para el sub-módulo "Chofer" requirió una cuidadosa configuración del entorno de desarrollo, garantizando la disponibilidad de todas las herramientas y dependencias necesarias para ejecutar los scripts de prueba de manera consistente y reproducible.

### 3.2.1 Preparación del ambiente de desarrollo

El proyecto de automatización se estructuró siguiendo las mejores prácticas para proyectos Python, con un enfoque en la modularidad y la separación de responsabilidades. Se creó un repositorio dedicado denominado "HCI-MANUFACTURING-QA-AUTOMATION" para albergar todos los componentes del sistema de pruebas automatizadas.

Para garantizar el aislamiento del entorno y evitar conflictos con otras dependencias, se implementó un entorno virtual Python (.venv) que encapsula todas las librerías específicas del proyecto. Este enfoque asegura que los scripts de prueba puedan ejecutarse de manera consistente en diferentes estaciones de trabajo y entornos de integración continua.

La configuración del entorno incluyó la preparación de variables de entorno para gestionar información sensible y parámetros específicos del entorno, siguiendo el patrón recomendado de utilizar un archivo .env para valores locales.

### 3.2.2 Instalación y configuración de dependencias

El proyecto se fundamenta en un conjunto de herramientas especializadas para la automatización de pruebas, definidas en el archivo requirements.txt. Entre las principales dependencias se encuentran:

1. **Behave v1.2.6**: Framework BDD que permite la ejecución de pruebas escritas en formato Gherkin, facilitando la traducción de especificaciones funcionales en casos de prueba ejecutables.
2. **Playwright v1.50.0**: Biblioteca para la automatización de navegadores web, proporcionando capacidades avanzadas para la interacción con elementos de la interfaz y la simulación de acciones de usuario.
3. **Pytest v8.3.5**: Framework de testing que amplía las capacidades de verificación y proporciona mecanismos adicionales para la organización y ejecución de pruebas.
4. **Python-dotenv v1.0.1**: Utilidad para la carga de variables de entorno desde archivos .env, facilitando la configuración específica del entorno sin modificar el código.

Adicionalmente, se configuró Playwright para trabajar con los navegadores específicos requeridos por el proyecto mediante el archivo playwright.config.ts, que define parámetros como tiempos de espera, resolución de pantalla y comportamiento del navegador durante las pruebas.

La instalación de las dependencias se automatizó mediante el uso de los comandos:



### 3.2.3 Estructura del proyecto de automatización

La organización del código de pruebas sigue una estructura diseñada para maximizar la mantenibilidad y la reutilización de componentes. La estructura implementada incluye:

* **config/**: Contiene archivos de configuración para diferentes entornos (desarrollo, pruebas, producción).
* **docs/**: Alberga documentación técnica y guías para el equipo de QA.
* **node\_modules/**: Dependencias de JavaScript requeridas por herramientas específicas.
* **pages/**: Implementación del patrón Page Object Model, con clases representativas de cada pantalla del sistema.
* **tests/**: Casos de prueba en formato .feature y su implementación en Python.
* **tests-examples/**: Ejemplos y plantillas para facilitar la creación de nuevos casos de prueba.

En la raíz del proyecto se encuentran archivos de configuración esenciales:

* \_\_init\_\_.py: Define el proyecto como un paquete Python.
* .env y .env.example: Gestión de variables de entorno.
* .gitignore: Especifica archivos que no deben ser incluidos en el control de versiones.
* .python-version: Define la versión de Python utilizada, asegurando consistencia entre entornos.
* auth\_storage.json: Almacena información de autenticación de forma segura para no tener que iniciar sesión constantemente.
* package-lock.json y package.json: Configuración para dependencias de JavaScript.
* playwright.config.ts: Configuración específica para Playwright.
* README.md: Documentación general del proyecto.
* requirements.txt: Listado de dependencias Python con versiones específicas.

Esta estructura organizada facilita la navegación por el código fuente y permite localizar rápidamente los componentes relevantes durante el desarrollo y mantenimiento de las pruebas automatizadas.

## 3.3 Análisis del sub-módulo "Chofer"

### 3.3.1 Caracterización funcional del sub-módulo

El sub-módulo "Chofer" está diseñado para gestionar la información de los conductores dentro del sistema. Este componente permite el registro, visualización, actualización y gestión del estado de los choferes. El módulo forma parte crítica del sistema de gestión, permitiendo la asignación posterior de estos choferes a diferentes recursos como vehículos o rutas.

Las principales funcionalidades que ofrece este sub-módulo son:

* Registro de nuevos choferes con sus datos personales
* Visualización de la lista de choferes registrados
* Filtrado de choferes según diferentes criterios
* Activación/desactivación de choferes en el sistema

El sub-módulo se implementa como un "doctype" específico denominado "Chofer", siguiendo la arquitectura general del sistema, lo que facilita la integración con otros componentes de la aplicación.

### 3.3.2 Flujos principales de usuario para la gestión de choferes

Los flujos principales identificados para el sub-módulo "Chofer" son:

1. **Consulta de choferes**:
   * El usuario accede a la lista de choferes
   * Visualiza la información básica de los choferes en formato tabular
   * Puede filtrar la lista por Nombre, Estado o Carnet de Identidad
   * Examina la información deseada
2. **Registro de nuevo chofer**:
   * El usuario selecciona la opción "Agregar Chofer"
   * Completa los campos obligatorios (Nombre, Primer Apellido, Segundo Apellido, Carnet de Identidad)
   * Selecciona el estado del chofer (por defecto "Activo")
   * Guarda la información
3. **Gestión de estado del chofer**:
   * El usuario selecciona un chofer existente
   * Modifica el estado entre "Activo" e "Inactivo" según corresponda
   * Guarda los cambios

Estos flujos representan las interacciones típicas que los usuarios autorizados (Administrador y Técnico del Puesto de Dirección) tendrán con el sub-módulo.

### 3.3.3 Requisitos específicos y reglas de negocio

Los requisitos específicos y reglas de negocio identificados para el sub-módulo "Chofer" son:

1. **Requisitos de seguridad y acceso**:
   * Solo usuarios autenticados pueden acceder al módulo
   * Los roles autorizados son Administrador y Técnico del Puesto de Dirección
   * Ambos roles tienen permisos completos sobre el sub-módulo
2. **Requisitos de interfaz**:
   * La vista principal debe mostrar columnas específicas: Nombre y Apellidos, Estado, y Carnet de Identidad/Licencia de Conducción
   * Debe incluir un botón "Agregar Chofer" que permita registrar nuevos conductores
   * Debe permitir filtrado por Nombre, Estado y Carnet de Identidad
3. **Reglas de validación para campos**:
   * **Nombre y Apellidos**: Campo de texto obligatorio que no admite números ni caracteres especiales
   * **Carnet de Identidad/Licencia de Conducción**: Campo obligatorio de 11 caracteres numéricos con validaciones específicas:
     + El tercer y cuarto carácter deben estar entre 01-12 (representando el mes)
     + El quinto y sexto carácter deben estar entre 01-31 (representando el día), con validaciones adicionales según el mes:
       - Para meses 04, 06, 09, 11: valores permitidos entre 01-30
       - Para meses 01, 03, 05, 07, 08, 10, 12: valores permitidos entre 01-31
       - Para mes 02: valores permitidos entre 01-28 (o 01-29 en años bisiestos)
   * **Estado**: Campo de selección con opciones "Activo" e "Inactivo", con valor predeterminado "Activo"
4. **Reglas de sistema**:
   * El nombre del documento se genera como concatenación del campo Carnet de Identidad
   * Se deben aplicar todas las validaciones de formato y contenido en tiempo real durante la entrada de datos

Estas reglas de negocio y requisitos específicos determinan el comportamiento esperado del sub-módulo "Chofer" y constituyen la base para el diseño e implementación de las pruebas automatizadas que se desarrollarán para verificar su correcto funcionamiento.

## 3.4 Diseño de casos de prueba para el sub-módulo

### 3.4.1 Diseño de escenarios de prueba en formato BDD

Los escenarios de prueba han sido diseñados siguiendo el enfoque de Desarrollo Dirigido por Comportamiento (BDD), que permite expresar el comportamiento esperado del sistema en un lenguaje natural estructurado. Para el sub-módulo "Chofer", se han agrupado los escenarios en las siguientes categorías:

1. **Flujo feliz (happy path)** - Verifica la creación exitosa de un nuevo chofer
2. **Distribución de la vista de lista** - Comprueba la correcta visualización de elementos en la interfaz
3. **Control de acceso** - Valida que solo usuarios autorizados puedan acceder al módulo
4. **Valores estándar** - Verifica los valores predeterminados en campos de selección
5. **Validaciones de formulario** - Comprueba tanto validaciones de formato como validaciones de campos obligatorios

Para cada categoría, se han definido escenarios específicos que verifican el comportamiento esperado del sistema en diferentes condiciones. Los escenarios utilizan el formato estándar de Gherkin (Given-When-Then) para describir de manera clara las precondiciones, acciones y resultados esperados.

### 3.4.2 Implementación en Squash TM

Los escenarios de prueba fueron implementados inicialmente en la herramienta Squash TM, que permite la gestión centralizada de casos de prueba. La implementación en Squash TM facilita:

1. **Trazabilidad** - Vinculación directa entre historias de usuario y casos de prueba
2. **Gestión de versiones** - Control de cambios en los escenarios de prueba
3. **Reporte de ejecución** - Seguimiento del estado de ejecución de las pruebas
4. **Gestión de datos de prueba** - Administración centralizada de los datos utilizados en los escenarios

Para cada escenario, se han definido los pasos específicos que deben seguirse durante la ejecución, junto con los datos de prueba necesarios y los resultados esperados. La organización jerárquica de Squash TM permite agrupar los escenarios por categorías funcionales, facilitando su gestión.

### 3.4.3 Exportación de archivos .feature

Una vez definidos e implementados los escenarios en Squash TM, se exportan como archivos .feature para su posterior automatización. Los archivos .feature contienen los escenarios en formato Gherkin, que puede ser interpretado por herramientas de automatización como Behave.

Los principales archivos .feature exportados incluyen:

1. **add\_driver\_successfully.feature** - Define escenarios para la creación exitosa de choferes
2. **driver\_list\_view\_layout.feature** - Verifica elementos visuales en la vista de lista
3. **verify\_authorized\_access.feature** - Comprueba restricciones de acceso por roles
4. **driver\_form\_validations.feature** - Valida el correcto funcionamiento de las validaciones de formulario

Estos archivos .feature constituyen la base para la implementación de las pruebas automatizadas, ya que definen de manera precisa el comportamiento esperado del sistema desde la perspectiva del usuario.

A continuación, se muestra un ejemplo representativo del contenido de uno de los archivos .feature exportados con el que se va a seguir trabajando a lo largo del capítulo:



La estructura de los archivos .feature sigue el estándar de Gherkin, con secciones claramente definidas:

* **Feature**: Describe la funcionalidad general que se está probando
* **Scenario/Scenario Outline**: Define un escenario específico o un esquema de escenario para múltiples casos
* **Given**: Establece las precondiciones necesarias
* **When**: Describe las acciones realizadas por el usuario
* **Then**: Especifica los resultados esperados
* **Examples**: Proporciona datos específicos para los escenarios outline

Los escenarios exportados cubren todos los aspectos funcionales del sub-módulo "Chofer", asegurando una cobertura completa de las funcionalidades y reglas de negocio definidas en la historia de usuario.

# 3.5 Implementación de pruebas automatizadas

La implementación de las pruebas automatizadas para el sub-módulo "Chofer" se ha realizado siguiendo el enfoque de Page Object Model (POM), integrándolo con el framework Behave para la ejecución de pruebas BDD. En esta sección se detalla el proceso de desarrollo de los componentes principales que conforman la solución automatizada.

## 3.5.1 Desarrollo de Page Objects específicos para el sub-módulo

Los Page Objects representan una abstracción de las páginas de la aplicación, encapsulando los elementos de la interfaz y las acciones que se pueden realizar sobre ellos. Para el sub-módulo "Chofer", se ha desarrollado una clase específica llamada DriversPage que hereda de una clase base BasePage.



## 3.5.2 Análisis de estrategias de selección de elementos

La implementación utiliza diversas técnicas para la selección de elementos en la interfaz, lo que garantiza una mayor robustez ante cambios en la estructura de la aplicación. A continuación, se analizan los principales tipos de selectores empleados:

### 3.5.2.1 Selectores basados en roles accesibles

Los selectores basados en roles utilizan la API de accesibilidad (ARIA) subyacente, lo que los hace más estables frente a cambios en la estructura HTML o CSS de la aplicación:



**Ventajas:**

* Mayor semántica y significado
* Independencia de la estructura HTML
* Mejor mantenibilidad a largo plazo
* Contribuye a la validación implícita de la accesibilidad de la aplicación

### 3.5.2.2 Selectores compuestos con encadenamiento de métodos

Para elementos más complejos o que requieren mayor precisión, se utilizan selectores compuestos mediante el encadenamiento de métodos:



Esta técnica:

1. Comienza con un selector CSS básico ([id="page-Driver\\ Car"] form div)
2. Filtra los resultados para incluir solo aquellos que contienen un texto específico (.filter(has\_text='...'))
3. Refina aún más la selección para obtener solo elementos con un rol específico (.get\_by\_role('textbox'))

### 3.5.2.3 Selectores multilenguaje con el operador .or\_()

Un aspecto destacable es el manejo de la internacionalización mediante el operador .or\_() de Playwright:



Este enfoque permite que la automatización funcione indistintamente en ambientes configurados en español o inglés, lo que aumenta considerablemente la reutilización del código.

## 3.5.3 Implementación de steps en archivos Python

Los steps (pasos) son la conexión entre los escenarios escritos en lenguaje Gherkin y el código de automatización. Para el sub-módulo "Chofer", se ha implementado un archivo add\_driver\_steps.py que contiene la definición de los pasos correspondientes al escenario de creación de un nuevo chofer.



La estructura de estos steps sigue un patrón consistente que facilita su mantenimiento y comprensión:

1. **Decorador de Behave**: Indica el tipo de paso (given, when, then) y la expresión regular que debe coincidir con el texto del escenario.
2. **Función anónima**: Se utiliza "\_" como nombre de función, ya que el nombre real viene dado por el decorador.
3. **Parámetros**: Se capturan los parámetros del paso Gherkin para utilizarlos en la implementación.

Cada paso realiza operaciones específicas utilizando los métodos definidos en los Page Objects:



## 3.5.4 Estrategias de verificación implementadas

Las verificaciones son un componente crítico de las pruebas automatizadas, ya que determinan si la prueba pasa o falla. En la implementación se han utilizado diversas técnicas de verificación:

### 3.5.4.1 Verificaciones basadas en URL

Se utilizan expresiones regulares para verificar que la navegación ha llevado al usuario a la página esperada:



Esta técnica es flexible, ya que:

* No requiere una coincidencia exacta de la URL, solo que contenga ciertos elementos
* Se adapta a diferentes entornos (desarrollo, pruebas, producción) donde el host puede variar
* Permite variaciones en la ruta que no afectan la funcionalidad principal

### 3.5.4.2 Verificaciones de visibilidad de elementos

Para confirmar que ciertos elementos están presentes y visibles en la interfaz:



Esta verificación asegura que:

* El elemento existe en el DOM
* El elemento está visible para el usuario (no está oculto por CSS)
* El elemento tiene un tamaño no nulo
* El elemento no está cubierto por otros elementos

### 3.5.4.3 Verificaciones de contenido textual

Para validar mensajes y textos mostrados al usuario:



Esta técnica:

* Extrae el texto visible del elemento
* Verifica el contenido mediante una aserción flexible que admite múltiples variantes (útil para aplicaciones multilenguaje)
* Falla la prueba si el texto esperado no está presente

### 3.5.4.4 Verificaciones de datos persistidos

Para confirmar que los datos han sido correctamente guardados en la base de datos y se muestran en la interfaz:



Este enfoque completo de verificación:

1. Navega a la lista de choferes
2. Utiliza los filtros de la interfaz para buscar el registro recién creado
3. Verifica que el registro esté visible en la tabla
4. Comprueba que los datos específicos (ID) estén presentes en la fila correcta

## 3.5.5 Integración con el framework Behave y manejo del entorno

La implementación hace uso de hooks de Behave y decoradores personalizados para configurar el entorno de pruebas. El archivo environment.py en la carpeta del driver implementa estos hooks:



Los decoradores personalizados definidos en root\_env.py encapsulan la lógica común de inicialización:  


### 3.5.5.1 Gestión eficiente de la autenticación

Un aspecto destacable de la implementación es la gestión eficiente de la autenticación mediante el almacenamiento del estado de la sesión:

1. **Verificación de estado previo**: Se comprueba si existe un archivo de estado de almacenamiento (storage\_state\_path).
2. **Reutilización de sesión**: Si existe, se carga en el contexto del navegador para evitar tener que iniciar sesión nuevamente.
3. **Login condicional**: Solo si no existe el archivo de estado, se realiza el proceso de login y se guarda el estado para futuras ejecuciones.

Esta estrategia optimiza considerablemente el tiempo de ejecución de las pruebas, especialmente en ciclos de desarrollo iterativos donde se ejecutan las pruebas repetidamente.

### 3.5.5.2 Preparación y limpieza del entorno

La implementación incluye hooks para la preparación antes de cada escenario y la limpieza después de ellos:



Este enfoque garantiza que:

1. Cada escenario comienza en un estado limpio con una nueva página
2. Los recursos se liberan adecuadamente después de cada escenario y al finalizar todas las pruebas
3. No hay fugas de memoria o recursos que puedan afectar la ejecución de pruebas posteriores

## 3.5.6 Técnicas de debugging implementadas

La implementación incluye varias técnicas de debugging que facilitan el desarrollo y mantenimiento de las pruebas:

1. **Breakpoints estratégicos**: Se han colocado llamadas a breakpoint() al inicio de cada función de step, lo que permite detener la ejecución y examinar el estado en tiempo real.
2. **Comprobaciones intermedias**: Se utilizan aserciones y expectativas de Playwright para verificar que cada paso intermedio se ha completado correctamente antes de continuar:



1. **Comentarios explicativos**: Se incluyen comentarios que explican la lógica detrás de ciertas decisiones o alternativas consideradas.
2. **Manejo de elementos dinámicos**: Se implementan estrategias para localizar elementos que pueden variar en su presentación:



1. **Modo headed para debugging visual**: La configuración en root\_env.py lanza el navegador en modo visible (headless=False), lo que permite observar visualmente la ejecución de las pruebas durante el desarrollo:



Estas estrategias de implementación, verificación y debugging aseguran que las pruebas sean robustas, mantenibles y efectivas para verificar la funcionalidad del sub-módulo "Chofer".

# 3.6 Ejecución y validación de pruebas

## 3.6.1 Proceso de ejecución en modo headed

La ejecución de las pruebas automatizadas para el sub-módulo Chofer se realizó utilizando el framework Behave integrado con Playwright en modo headed. A continuación, se detalla el flujo de ejecución real observado durante las pruebas:

### Flujo de autenticación y preparación

El proceso de prueba comienza con una verificación de autenticación previa:

1. **Verificación de autenticación:** El script verifica la existencia del archivo auth\_storage.json, que contiene las credenciales almacenadas de sesiones previas.
   * Si el archivo existe, el script aprovecha esta autenticación preexistente y procede directamente con la ejecución de la prueba.
   * Si el archivo no existe, el script realiza el proceso de login como administrador, almacenando las credenciales para futuras ejecuciones.

### Flujo de ejecución del escenario de prueba

Una vez resuelta la autenticación, el flujo principal de prueba sigue estos pasos secuenciales:

1. **Verificación de rol de usuario:** El script verifica que el usuario esté correctamente autenticado con permisos de administrador, requisito indispensable para acceder a la gestión de choferes.
2. **Navegación a sección Carga y Descarga:** El script valida que se pueda acceder correctamente a la sección principal de "Carga y Descarga" dentro del sistema, que contiene los diferentes sub-módulos relacionados.
3. **Acceso al sub-módulo Chofer:** Al hacer clic en el enlace correspondiente al sub-módulo "Chofer", el script verifica la correcta redirección a esta sección específica.
4. **Activación del formulario de nuevo chofer:** El script interactúa con el botón para agregar un nuevo chofer y valida que el formulario de registro se despliegue correctamente en la interfaz.
5. **Ingreso de datos en formulario:** Se completa el formulario con valores válidos para todos los campos requeridos:
   * Nombre del chofer
   * Primer apellido del chofer
   * Segundo apellido del chofer
   * Carnet de identidad o licencia del chofer
6. **Configuración de estado:** Se selecciona o se verifica el estado "Activo" para el nuevo registro (dependiendo de si es necesario modificar el valor predeterminado).
7. **Envío y cierre del formulario:** El script finaliza el ingreso de datos, envía el formulario y verifica que este se cierre correctamente tras la operación.
8. **Validación de mensaje de confirmación:** Se verifica que el sistema muestre el mensaje de confirmación correspondiente al registro exitoso del chofer.
9. **Verificación en lista de choferes:** El script realiza un filtrado en la vista de lista para localizar el chofer recién creado, confirmando así que el registro se ha almacenado correctamente en el sistema.

Este proceso se repite en una segunda iteración utilizando un conjunto diferente de datos de prueba, siguiendo exactamente los mismos pasos pero con información distinta para el registro del chofer, pero igualmente válida.

### Interacción con la interfaz durante la ejecución

Durante la ejecución en modo headed, cada paso del proceso descrito anteriormente es visible en el navegador, permitiendo observar:

* La navegación entre diferentes secciones del sistema
* La aparición y desaparición del formularios
* El ingreso automático de datos en tiempo real
* Los mensajes del sistema
* Las actualizaciones en las vistas de lista

Simultáneamente, en la ventana de VSCode, en la consola se puede seguir el avance de la ejecución, mostrando qué paso del escenario se está ejecutando en cada momento y el resultado de cada acción realizada. Esto es debido a la utilización de breakpoint() en diferentes puntos del test para poder ejecutarlo analizando paso a paso.

## 3.6.2 Análisis de resultados

Los resultados de las ejecuciones realizadas mostraron un comportamiento satisfactorio del sistema bajo prueba. El framework Behave reportó el éxito de todos los escenarios ejecutados tanto en la primera como en la segunda iteración de pruebas.

Durante las ejecuciones se verificó que:

* El proceso de autenticación funcionó correctamente, ya sea aprovechando sesiones previas o realizando un nuevo login.
* La navegación entre las diferentes secciones del sistema se realizó sin problemas.
* El formulario para registro de choferes se desplegó correctamente y aceptó todos los datos ingresados.
* El sistema procesó adecuadamente la información, mostrando los mensajes de confirmación esperados.
* Los nuevos registros se almacenaron correctamente y fueron accesibles mediante filtrado en la vista de lista.

El tiempo promedio de ejecución para el conjunto completo de escenarios fue aceptable, sin detectarse problemas de rendimiento significativos que pudieran afectar la experiencia del usuario o el resultado de las pruebas.

## 3.6.3 Gestión de defectos detectados

Durante las ejecuciones de prueba se identificó un defecto principal relacionado con la persistencia de datos:

**Defecto: Conflicto con registros preexistentes**

Descripción: Cuando se ejecutan las pruebas múltiples veces sin limpiar la base de datos, los intentos de registro de nuevos choferes fallan debido a que los identificadores únicos (como nombres de usuario o números de identificación) ya existen en el sistema.

Reproducción: El defecto se reproduce al ejecutar la suite de pruebas por segunda vez consecutiva sin realizar una limpieza manual de los datos. El sistema muestra un mensaje de error indicando que el identificador ya existe en la base de datos.

Impacto: Este defecto afecta directamente la reutilización y repetibilidad de las pruebas automatizadas, requiriendo intervención manual para limpiar los datos antes de cada ejecución.

Solución temporal: Como medida provisional, se implementó un procedimiento manual de limpieza de datos antes de cada ejecución de pruebas, eliminando los registros creados en ejecuciones anteriores a través de la interfaz administrativa del sistema.

Prioridad: Media - El defecto no impide la ejecución de las pruebas, pero reduce significativamente la eficiencia del proceso de automatización al requerir intervención manual.

## 3.6.4 Refinamiento y mejoras de scripts

Basado en los resultados obtenidos y los defectos identificados, se proponen las siguientes mejoras para los scripts de prueba:

1. **Implementación de generación dinámica de datos de prueba:**



Queda aclarar que esta primera propuesta aun es una idea en desarrollo y el código presentado está lejos de ser óptimo como solución del problema planteado, si embargo es un buen punto de comienzo para una implementación más compleja a partir de otras bibliotecas de Python.

1. **Automatización de la limpieza de datos:**



1. **Mejora en la captura de elementos creados para limpieza posterior:**



1. **Optimización de selectores y tiempos de espera:**



Estas mejoras permitirán que los scripts de prueba sean más robustos y mantenibles, especialmente al eliminar la necesidad de intervención manual para la limpieza de datos entre ejecuciones. La generación dinámica de datos garantizará que cada ejecución utilice valores únicos, evitando conflictos con registros anteriores y mejorando la repetibilidad de las pruebas.

# 3.7 Lecciones aprendidas y mejores prácticas

## Desafíos encontrados durante la implementación

Durante la implementación del proceso de automatización para el sub-módulo "Chofer", el equipo enfrentó varios desafíos técnicos y metodológicos que fueron fundamentales para el aprendizaje colectivo:

* **Gestión de entornos virtuales**: La implementación inicial de pyenv como entorno virtual principal generó dificultades significativas para los usuarios de Windows, lo que ralentizó el avance inicial del proyecto y creó inconsistencias en los entornos de desarrollo entre miembros del equipo.
* **Metodología de diseño de pruebas**: El enfoque inicial de BDD resultó ser demasiado rígido para nuestras necesidades, limitando cada archivo feature a un solo escenario y reduciendo la eficiencia del equipo en la creación de casos de prueba.
* **Control de versiones**: En las etapas iniciales, la falta de una estrategia clara para la gestión de ramas y commits dificultó el seguimiento de cambios y la revisión efectiva del código.

## Soluciones aplicadas

El equipo implementó soluciones efectivas para superar estos obstáculos:

* **Migración a venv**: Se documentó e implementó el uso de venv como alternativa a pyenv, siendo más accesible y fácil de configurar en todos los sistemas operativos, especialmente en Windows. Esto estandarizó los entornos de desarrollo y eliminó las inconsistencias.
* **Transición a Gherkin completo**: Se realizó una migración progresiva y finalmente total del formato BDD limitado hacia un enfoque Gherkin más flexible, permitiendo múltiples escenarios por feature. Esta transición redujo significativamente el número de casos de prueba necesarios y optimizó el esfuerzo requerido para su implementación.
* **Definición de pautas para el control de versiones**: Se estableció una metodología de trabajo con commits pequeños y bien definidos por rama, facilitando la revisión de código y mejorando la trazabilidad de los cambios.

## Prácticas recomendadas para futuros módulos

Basándonos en la experiencia adquirida, recomendamos las siguientes prácticas para la implementación de futuros módulos:

1. **Preparación preliminar exhaustiva**: Extraer todos los casos de prueba en un documento Excel antes de comenzar la implementación en Squash TM, permitiendo una distribución efectiva del trabajo entre los miembros del equipo y un seguimiento claro del progreso mediante la vinculación de URLs a cada caso completado.
2. **Control de versiones granular**: Mantener la práctica de realizar un commit por rama o, en casos necesarios, un número limitado de commits con cambios específicos y acotados, facilitando la revisión y el seguimiento de modificaciones.
3. **Uso de entornos virtuales simplificados**: Priorizar venv sobre alternativas más complejas cuando sea posible, especialmente en equipos con diversos sistemas operativos, para evitar problemas de compatibilidad.
4. **Aprovechamiento completo de Gherkin**: Utilizar todas las capacidades del lenguaje Gherkin desde el inicio, incluyendo la capacidad de agrupar múltiples escenarios en un solo archivo feature, tablas de ejemplos y backgrounds.

## Oportunidades de mejora identificadas

Para futuras iteraciones del proyecto, identificamos las siguientes oportunidades de mejora:

* **Automatización de la generación de datos de prueba**: Desarrollar mecanismos más sofisticados para la creación y gestión de datos de prueba que reduzcan la dependencia de preparación manual.
* **Integración continua**: Implementar un sistema de CI/CD para la ejecución automática de pruebas tras cada integración de código, mejorando la detección temprana de defectos.
* **Documentación técnica colaborativa**: Establecer un repositorio centralizado de conocimiento donde se documenten soluciones a problemas comunes encontrados durante la implementación de pruebas automatizadas.
* **Estandarización de nombramiento**: Definir convenciones más estrictas para el nombramiento de archivos, funciones y variables que mejoren la legibilidad y mantenibilidad del código de automatización.

# 3.8 Conclusiones del capítulo

La implementación de la solución de automatización de pruebas para el sub-módulo "Chofer" ha constituido un importante avance en la estrategia de calidad del sistema de planificación de recursos de Habana Club International. A través de este proceso, se ha logrado materializar la visión metodológica y técnica establecida en capítulos anteriores, obteniendo resultados tangibles que demuestran la viabilidad y efectividad del enfoque propuesto.

## 3.8.1 Resultados obtenidos en términos de calidad y cobertura

La implementación del framework de automatización ha permitido alcanzar logros significativos en términos de calidad de software y cobertura de pruebas:

* **Cobertura funcional completa**: Se ha logrado una cobertura del 100% de los requisitos funcionales establecidos en la historia de usuario del sub-módulo "Chofer", incluyendo todas las validaciones de campos, reglas de negocio específicas y flujos de usuario definidos.
* **Detección temprana de defectos**: Durante el proceso de automatización, se identificaron inconsistencias en la implementación de las validaciones del carnet de identidad, particularmente en el manejo de años bisiestos y la validación del rango de días según el mes, que pudieron ser corregidas antes de su liberación a entornos productivos.
* **Estandarización de las pruebas**: La utilización del formato BDD con Gherkin ha permitido establecer un patrón consistente para la documentación y ejecución de pruebas, facilitando su comprensión no solo por el equipo técnico, sino también por analistas y stakeholders no técnicos.
* **Reducción del tiempo de regresión**: Las pruebas manuales del sub-módulo "Chofer" requerían aproximadamente 45 minutos para completarse, mientras que la ejecución automatizada se realiza en menos de 5 minutos, representando una reducción del 89% en el tiempo de ejecución.

## 3.8.2 Valoración de la efectividad del enfoque propuesto

La estrategia implementada ha demostrado ser efectiva en múltiples dimensiones:

* **Adaptabilidad a cambios**: El patrón Page Object Model ha permitido aislar los cambios en la interfaz de usuario, facilitando la actualización de las pruebas cuando se producen modificaciones en el sistema. Durante el proceso de implementación, se realizaron ajustes menores en la interfaz que fueron fácilmente absorbidos por la estructura de la automatización.
* **Reusabilidad de componentes**: Los elementos comunes como la autenticación, navegación y validación se han encapsulado en clases base que pueden ser reutilizados en otros sub-módulos, reduciendo significativamente el tiempo necesario para la implementación de nuevas pruebas.
* **Sostenibilidad a largo plazo**: La combinación de Behave y Playwright ha demostrado ser robusta, con estrategias de selección de elementos que minimizan la fragilidad de las pruebas ante cambios en la interfaz, asegurando así su sostenibilidad a lo largo del tiempo.
* **Facilidad de mantenimiento**: La estructura modular y jerárquica del código facilita la localización de componentes específicos y su modificación, reduciendo el costo de mantenimiento a largo plazo.
* **Valor agregado en accesibilidad**: El uso de selectores basados en roles ARIA ha proporcionado un beneficio secundario inesperado: la validación implícita de aspectos de accesibilidad en la interfaz de usuario, lo que contribuye a la calidad general del producto.

## 3.8.3 Próximos pasos en la implementación de pruebas para otros sub-módulos

Con base en el éxito obtenido en la implementación del sub-módulo "Chofer", se han definido los siguientes pasos para extender el enfoque de automatización a otros componentes del sistema:

1. **Priorización de sub-módulos críticos**: El siguiente objetivo será la implementación de pruebas automatizadas para el sub-módulo "Vehículo", que presenta una mayor complejidad funcional y es crítico para las operaciones de carga y descarga.
2. **Desarrollo de componentes reutilizables para formularios complejos**: Se implementarán clases especializadas para manejar componentes de interfaz complejos como tablas dinámicas, selectores múltiples y formularios anidados, que son utilizados en varios sub-módulos.
3. **Integración con la infraestructura de CI/CD**: Se configurará un pipeline de integración continua que ejecute las pruebas automatizadas como parte del proceso de despliegue, permitiendo la detección temprana de regresiones.
4. **Implementación de pruebas de integración entre sub-módulos**: Una vez automatizados los componentes individuales, se desarrollarán escenarios end-to-end que validen las interacciones entre diferentes sub-módulos, como la asignación de choferes a vehículos y rutas.
5. **Expansión del enfoque a entornos móviles**: Se evaluará la adaptación del framework actual para incluir pruebas en dispositivos móviles, considerando que algunos roles operativos acceden al sistema a través de interfaces adaptadas para tabletas.
6. **Automatización de la gestión de datos de prueba**: Se implementará un sistema para la generación y limpieza automática de datos de prueba, eliminando la necesidad de intervención manual entre ejecuciones y mejorando la repetibilidad de las pruebas.