**Módulo de diseño:**

**Caso 1: Búsqueda válida de vuelo ida y vuelta**

**Objetivo:** Validar que el sistema muestre resultados de vuelos al ingresar correctamente todos los datos.

**Pasos:**

1. Seleccionar tipo de vuelo: Ida y vuelta.
2. Ingresar origen: Bogotá, BOG - Colombia
3. Ingresar destino: Cali, CLO - Colombia
4. Seleccionar fecha de ida y regreso válidas.
5. Hacer clic en “Buscar vuelos”

**Resultado esperado:** Se deben mostrar opciones de vuelo disponibles.

**Resultado no esperado:** Encontrar un vuelo en fechas menores a la actual, o desde y hacia ciudades diferentes

**Caso 2: Error al seleccionar una fecha de regreso anterior a la de ida**

**Objetivo:** Validar que el sistema detecte y bloquee una fecha inválida.

**Pasos:**

1. Seleccionar tipo de vuelo: Ida y vuelta.
2. Ingresar origen y destino válidos.
3. Ingresar fecha de ida: 15 de abril
4. Ingresar fecha de regreso: 10 de abril (antes de la ida)
5. Clic en “Buscar vuelos”

**Resultado esperado:** Debe mostrarse un mensaje de validación o impedir continuar.

**Resultado no esperado:** Muestre resultados donde la fecha de salida sea mayor a la de regreso.

**Caso 3: Búsqueda sin completar todos los campos**

**Objetivo:** Validar los mensajes de error por campos obligatorios vacíos.

**Pasos:**

1. No ingresar ningún dato.
2. Hacer clic directamente en “Buscar vuelos”

**Resultado esperado:** Deben aparecer mensajes de error indicando campos obligatorios (por ejemplo: origen, destino, fechas).

**Resultado NO esperado:** El sistema permite Buscar vuelos sin llenar datos obligatorios.

**Modulo Arquitectura**

## **Patrones propuestos**

### 1. **Page Object Model (POM)**

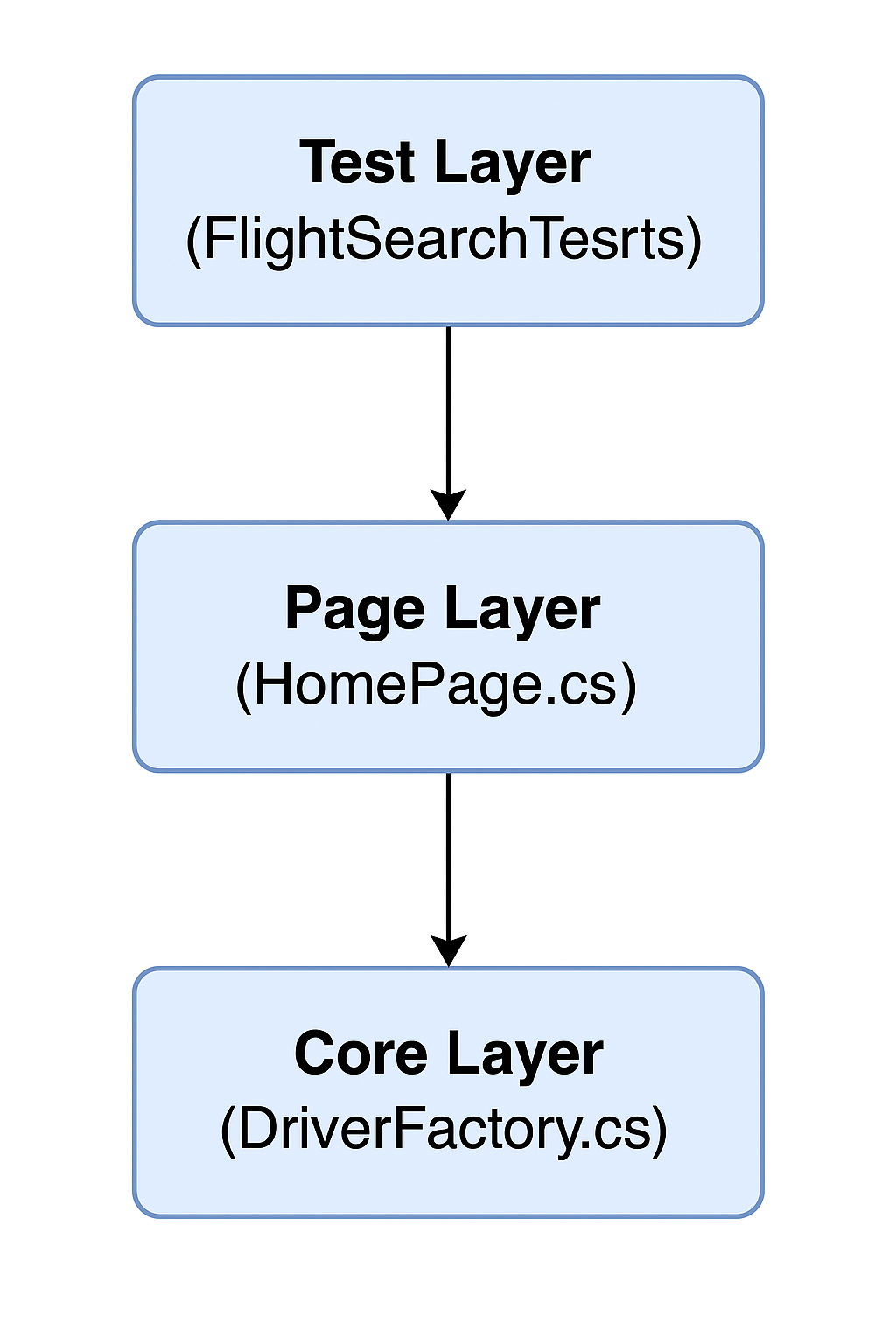
* Separa la lógica de interacción con la UI del código de los tests
* Cada página se representa con una clase
* Mejora el mantenimiento y la legibilidad

### 2. **Factory Pattern (DriverFactory)**

* Se encarga de crear y cerrar instancias del navegador
* Evita duplicación y facilita reutilización

### 3. **Separación por Capas (Layered Test Architecture)**

* **Test Layer**: Donde están los casos (GSQA.Tests)
* **Page Layer**: Objetos de página (GSQA.Automation.Pages)
* **Core Layer**: Utilidades como DriverFactory, configuración, esperas, etc.



### **Modulo Devops:** Herramientas propuestas:

|  |  |
| --- | --- |
| Elemento | Herramienta sugerida |
| Repositorio | GitHub |
| Automatización CI/CD | GitHub Actions o Azure DevOps |
| Empaquetado | Docker (opcional para entorno aislado) y debido al uso que se hará dentro de la contenirizacion |
| Navegadores | Google Chrome |
| Resultados/Reportes | ReportPortal, Allure, HTML Reports |

**Pipelines recomendados:**

### 1. **Pipeline de Desarrollo**

* Construye el proyecto
* Ejecuta pruebas **unitarias**
* Análisis estático (SonarQube, CodeQL)

**Se ejecuta en cada push / PR**

### 2. **Pipeline de QA o Validación**

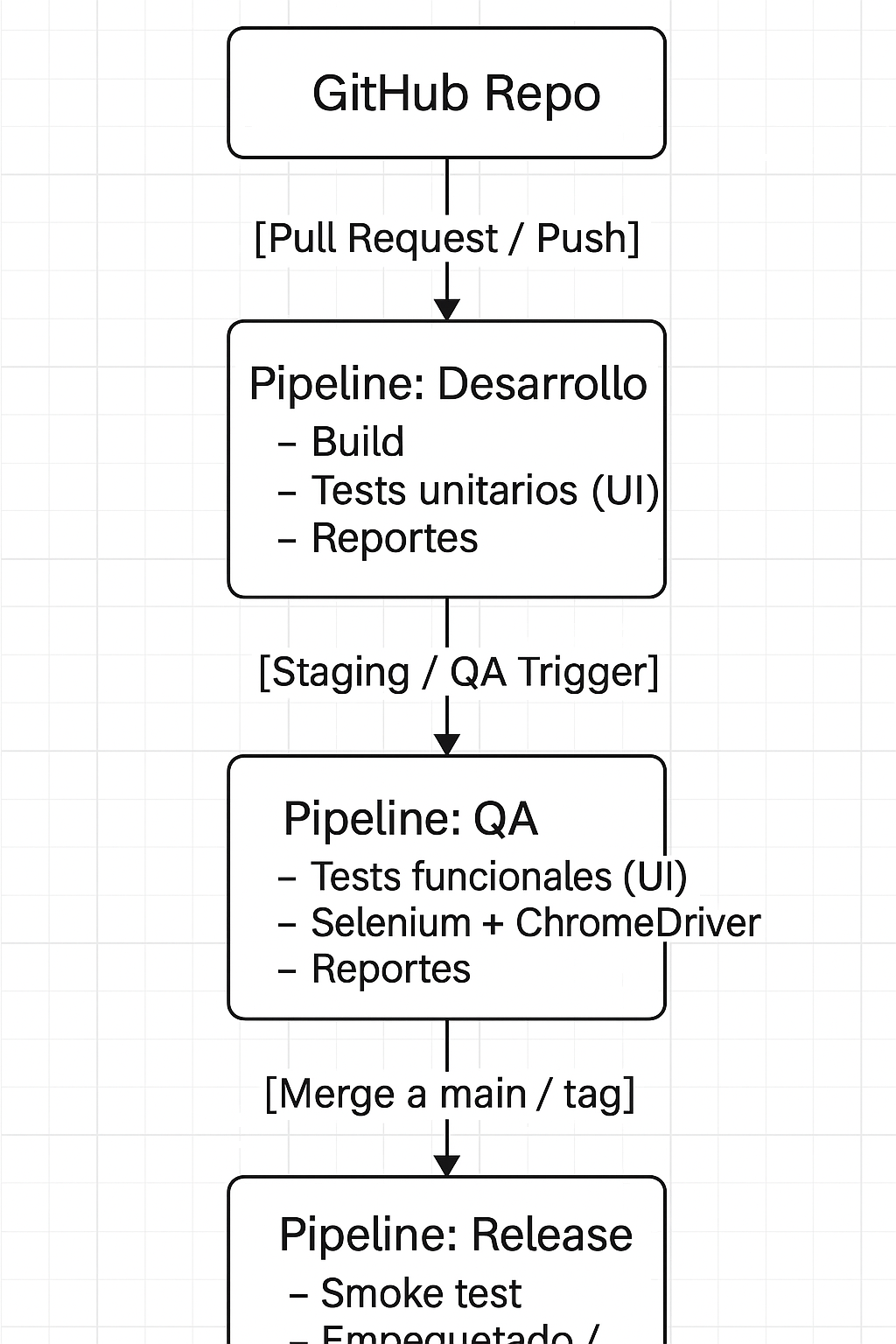
* Ejecuta pruebas **funcionales UI** (Selenium)
* Puede usarse con chromedriver o Selenium Grid
* Genera reporte de resultados
* Puede ejecutarse en paralelo con matriz de navegadores

**Se ejecuta en ambiente de staging, nightly o por solicitud manual**

### 3. **Pipeline de Release**

* Corre pruebas finales
* Empaqueta aplicación
* Despliega a producción (si aplica)

**Se ejecuta al hacer merge a main o etiquetar release**



## ¿Dónde corre cada tipo de prueba?

|  |  |
| --- | --- |
| Tipo de prueba | Pipeline recomendado |
| Unitarias | Desarrollo |
| UI (Selenium) | QA/Validación |
| Pruebas contractuales | QA/Validación o Release |
| Pruebas smoke final | Release |

 Tests se desarrollan localmente (Mac, haciendo uso de visual studio code y entornos .net diferentes a visual studio code)

 En CI, se puede usar:

* Contenedor Docker con .NET + Chrome
* Selenium Grid
* GitHub Actions con runners hospedados