

---

# ***Informe de Ambientes***

**Proyecto: Experiencias Significativas**

# ÍNDICE

1. Introducción
2. Objetivos del documento
3. Estructura de Ambientes en GitHub
  - 3.1. ¿Qué es un ambiente en control de versiones?
  - 3.2. Ambientes utilizados en Experiencias Significativas
4. Flujo de trabajo por ambientes
  - 4.1. Ambiente dev
  - 4.2. Ambiente qa
  - 4.3. Ambiente staging
  - 4.4. Ambiente main
5. Uso de ambientes por repositorio
6. Problemas encontrados y soluciones implementadas
7. Ejemplos con capturas
  - 7.1. Ambientes principales configurados en GitHub
  - 7.2. Vista de ramas y flujo de commits
  - 7.3. Visualización del gráfico de commits (GRAPH)
8. Conclusiones

## 1. Introducción

El proyecto *Experiencias Significativas* implementa una arquitectura de control de versiones basada en múltiples ambientes, con el objetivo de asegurar la estabilidad del código, mejorar el flujo de trabajo del equipo y garantizar que cada funcionalidad pase por un proceso adecuado de revisión, pruebas y validación.

El uso de ambientes separados permite evitar que cambios no probados impacten directamente en la versión estable del sistema. Cada ambiente (dev, qa, staging y main) está diseñado para cumplir una función específica en el ciclo de desarrollo, desde la creación inicial del código hasta su despliegue final.

Este informe explica en detalle cada uno de estos ambientes, los flujos de trabajo utilizados, los procesos aplicados por el equipo, ejemplos visuales del repositorio y problemas detectados durante la ejecución del proyecto.

## **2. Objetivos del documento**

- Describir los ambientes utilizados durante el desarrollo.
- Explicar el flujo de trabajo aplicado en GitHub.
- Documentar el proceso aplicado en frontend, backend y móvil.
- Presentar ejemplos reales con comandos y ramas creadas.
- Proveer una guía clara para futuros desarrolladores del proyecto.

## 1. Estructura de Ambientes en GitHub

### 3.1 ¿Qué es un Ambiente de control de versions?

Un ambiente es un espacio lógico que representa una etapa del software dentro del ciclo de vida del Desarrollo. Permite separar Código experimental de Código estable.

### 3.2 Ambiente utilizados en Experiencias Significativas

#### Ambiente | Propósito

dev | Desarrollo diario y nuevas funcionalidades.

qa | Validación funcional y pruebas internas.

staging | Simulación de producción antes del despliegue.

main | Versión final estable del Proyecto.

## 2. Flujo de trabajo por ambientes

### 4.1 Ambiente dev

Primer entorno de desarrollo donde se crean y prueban nuevas funcionalidades.

#### Flujo estándar:

- git pull origin dev
- git switch dev
- git switch -c HU-01-dev
- realizar Desarrollo
- crear MR de HU-01-dev → dev
- Nota: El desarrollo se cierra con la rama dev update.

**Propósito del ambiente dev:**

- Permitir iteraciones rápidas sin afectar las ramas principales.
- Facilitar la integración continua de nuevas funcionalidades.
- Detectar errores tempranos antes de pasar a QA.
- Asegurar que cada cambio esté documentado en commits individuales.

**4.2 Ambiente qa**

Ambiente para pruebas de calidad, donde se valida que las funcionalidades que vienen de dev cumplen lo requerido.

**Flujo estándar:**

- `git pull origin qa`
- `git switch qa`
- `git switch -c HU-01-qa`
- realizar desarrollo QA
- crear MR de HU-01-qa → qa
- Nota: El desarrollo se cierra con la rama qa update.

**Propósito del ambiente qa:**

- Realizar pruebas manuales y automáticas.
- Validar reglas de negocio.
- Asegurar consistencia en el sistema.
- Prevenir que cambios defectuosos lleguen a staging

### 4.3 Ambiente staging

Ambiente previo a producción. Aquí se valida la integración completa del sistema.

#### Flujo estándar:

- `git pull origin staging`
- `git switch staging`
- `git switch -c HU-01-staging`
- realizar desarrollo staging
- crear MR de HU-01-staging → staging
- Nota: El desarrollo se cierra con la rama staging update.

#### Propósito del ambiente staging:

- Simular el comportamiento final del sistema.
- Validar la interacción entre múltiples módulos.
- Verificar la compatibilidad antes del despliegue final.
- Aprobar o rechazar el release temporal.

### 4.4 Ambiente main

Ambiente principal donde solo se almacenan releases aprobados. No se trabaja directamente sobre esta rama.

#### Flujo estándar:

- `git pull origin main`

- `git switch main`
- `git switch -c release.1.1`  
`HU-01-release.1.1`  
`HU-02-release.1.1`
- crear MR `release.1.1` → `main`

#### **Propósito del ambiente main:**

- Representar la versión estable del proyecto.
- Asegurar que el código en `main` siempre sea funcional.
- Evitar cambios directos y mantener control total con releases.
- Servir como base para las versiones de producción.

## **5. Uso de ambientes por repositorio**

- **Frontend Web** – Implementación completa y correcta del flujo de ambientes.
- **Backend** – Implementación completa, con control adecuado de merges y conflictos.
- **Aplicación móvil** – Flujo implementado con éxito, siguiendo la misma estructura.

Cada repositorio mantiene independencia en su ciclo de versionamiento, pero todos comparten la misma estructura de ambientes para mantener uniformidad y control.

## **6. Problemas encontrados y soluciones implementadas**

Durante el ciclo de desarrollo se presentaron dificultades que permitieron fortalecer el flujo de trabajo.

✓ **Conflictos entre ramas**

**Solución:**

- Estandarizar commits.
- Mantener ramas actualizadas con dev antes de trabajar.

✓ **Fallos de control de versiones en mobile**

**Solución:**

- Documentar flujos.
- Crear ramas más ordenadas.

✓ **Dependencias rotas en backend**

**Solución:**

- Unificación de versiones.
- Revisión del package.json / dependencias internas.

✓ **Ramas abiertas sin cerrar**

**Solución:**

- Política estricta de cierre de ramas.
- Limpieza mensual del repositorio.

## 7. Ejemplos con capturas

### 7.1 Ambientes principales configurados en GitHub.

Aquí se muestran las cuatro ramas base del flujo de desarrollo:

- **main**: versión estable y publicada (producción).
- **dev**: ambiente destinado al desarrollo activo de funcionalidades.
- **qa**: ambiente donde se realizan pruebas internas de validación.
- **staging**: ambiente de preproducción donde se simula el despliegue

Branches

Overview Active Stale All

Q Search branches...

Default

Branch	Updated	Check status	Behind	Ahead	Pull request
main	last week			Default	

Active branches

Branch	Updated	Check status	Behind	Ahead	Pull request
qa	last week		0	0	
dev	last week		0	0	
HU-10-dev	last week		3	0	
staging	last week		12	16	
HU-21-dev	2 weeks ago		7	0	

[View more branches >](#)

final.

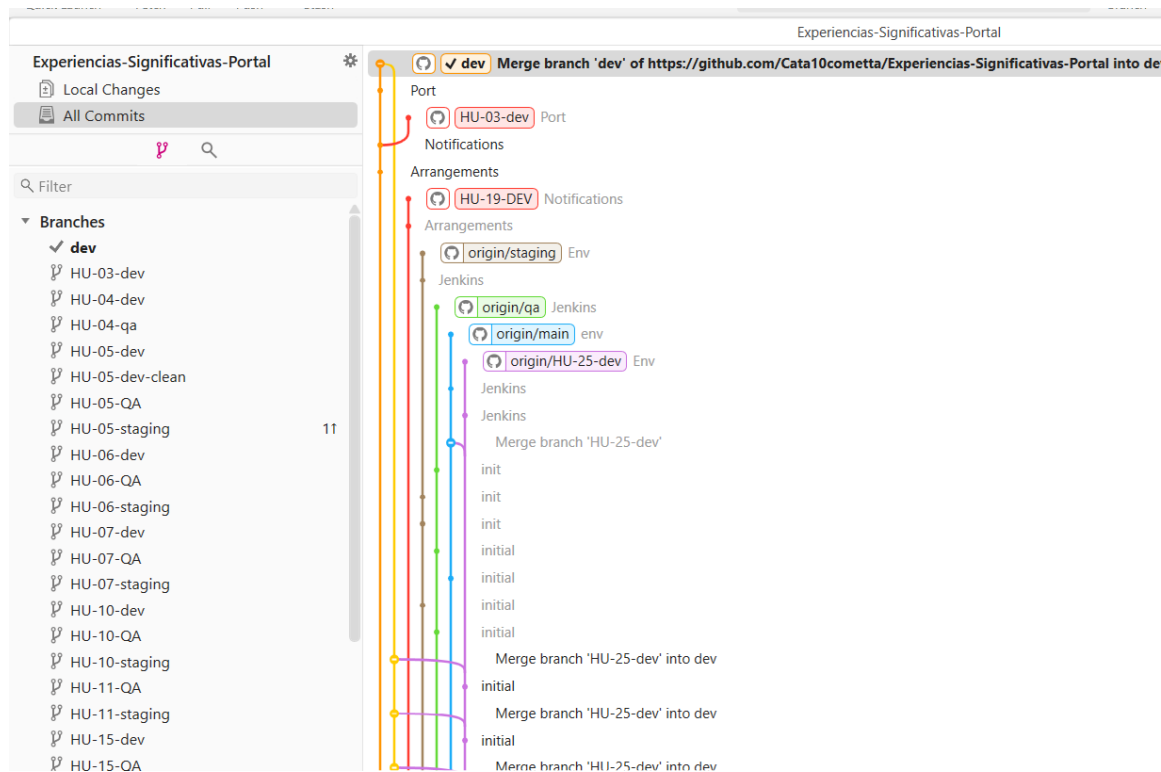
También se visualizan ramas individuales de Historia de Usuario, como HU-10-dev, lo que demuestra la aplicación correcta del flujo GitFlow adaptado al Proyecto.

## 7.2 Vista de ramas y flujo de commits del Proyecto

En esta imagen se observa la estructura completa de las ramas utilizadas en el proyecto *Experiencias Significativas*, incluyendo las ramas por Historia de Usuario en sus diferentes ambientes (*dev*, *qa*, *staging*).

Esta vista permite visualizar :

- El recorrido de las HU en dev, qa y staging.
- Los cherry-picks realizados para mover cambios puntuales.
- Los merges hacia dev y staging.
- Los commits del equipo organizados por cada funcionalidad.



Además, este es el espacio donde se realizan los *cherry-picks* para traer commits específicos desde una rama hacia otra.

### 7.3 Ramas de Historias de Usuario en el ambiente dev.

Esta vista muestra todas las ramas utilizadas para el desarrollo activo.

Cada rama corresponde a una Historia de Usuario específica y permite trabajar funciones de forma aislada.

Branch	Updated	Check status	Behind	Ahead	Pull request
dev	last week		0	0	...
HU-18-dev	last week		3	0	...
HU-30-dev	2 weeks ago		7	3	...
HU-25-dev	2 weeks ago		12	1	...
HU-24-dev	2 weeks ago		12	1	...
HU-21-dev	2 weeks ago		7	0	...
HU-06-dev	2 weeks ago		8	0	#7  ...
HU-05-dev	2 weeks ago		12	1	...
HU-01-dev	2 weeks ago		11	0	#1  ...
HU-33-dev	2 weeks ago		12	0	...
HU-05-dev	2 weeks ago		12	0	...

### 7.4 Merge Request desde HU-XX-dev hacia dev.

El Merge Request es un paso obligatorio para integrar código.

Aquí se revisa la calidad del código y se validan los cambios antes de aprobarlos.

## Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).

base: Hu-05-dev

compare: main

✓ Able to merge. These branches can be automatically merged.

Realsee.1.0.0 #14

No description available

View pull request

Create another pull request to discuss and review the changes again. [Learn about pull requests](#)

Create pull request

12 commits

16 files changed

1 contributor

Commits on Nov 24, 2025

Implement Helper Password

Mari2303 committed 2 weeks ago

92d0720

<>

Merge pull request #1 from Mari2303/HU-01-dev

Verified

cda6bc7

<>

Mari2303 authored 2 weeks ago

Update grades experience

Mari2303 committed 2 weeks ago

5696201

<>

Update population grade

Mari2303 committed 2 weeks ago

d78f16e

<>

## 7.5 Historial de Commits del Proyecto

En esta imagen se visualiza el registro completo de todos los commits realizados.

Permite identificar quién realizó cada cambio, cuándo y en qué rama.

### Se analizan:

- Mensajes descriptivos
- Secuencia de trabajo
- Frecuencia de Desarrollo
- Calidad de los commits

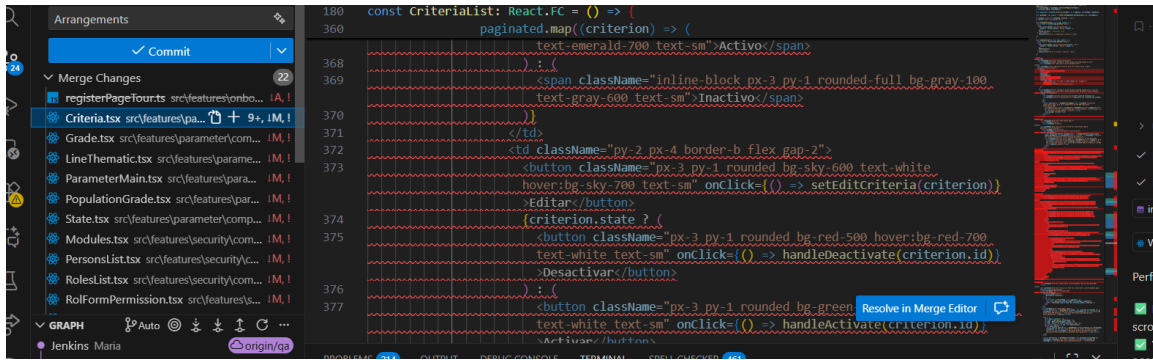
## Commits

main	All users	All time
Commits on Dec 1, 2025		
Merge branch 'dev' of <a href="https://github.com/Mari2303/Experiencia-significativa-API">https://github.com/Mari2303/Experiencia-significativa-API</a> into dev Mari2303 committed last week	1f5d13a	<>
Commits on Nov 28, 2025		
FROM EDIT Mari2303 committed last week	eaacaf0	<>
Aupdate Mari2303 committed 2 weeks ago	9ccb143	<>
Commits on Nov 27, 2025		
Initial data Mari2303 committed 2 weeks ago	ddb5d8d	<>
Update register Mari2303 committed 2 weeks ago	25a5936	<>
Commits on Nov 26, 2025		
Commit Correction DeveloTime Mari2303 committed 2 weeks ago	5044fdd	<>

### 7.6 Conflicto generado durante un cherry-pick realizado desde Fork, visualizado en Merge Changes dentro de Visual Studio Code.

En esta captura se observa el proceso en el cual se realizó un **cherry-pick** desde la aplicación Fork. Al intentar aplicar un commit específico desde otra rama, la herramienta detectó que el cambio no podía integrarse automáticamente debido a diferencias con el código actual.

Fork notificó el conflicto y, al abrir el proyecto en Visual Studio Code, los archivos afectados se mostraron dentro de la sección **“Merge Changes”**, lo cual indica claramente que existen líneas ocultas o incompatibles entre ambas versiones del archive



Este escenario es común cuando se trabaja con varios ambientes (dev, qa, staging) y múltiples ramas activas, ya que los cambios pueden haberse modificado de manera diferente en cada contexto.

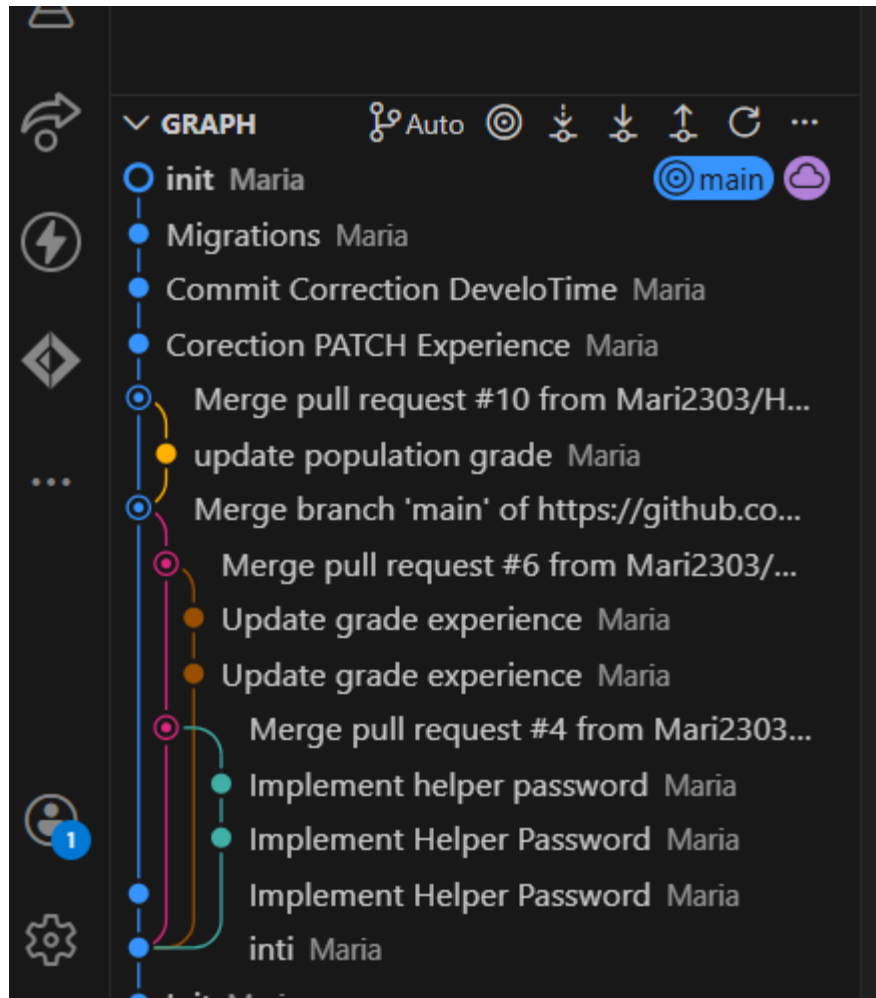
## 7.7 Visualización del gráfico de commits (GRAPH) que muestra el historial de ramas y movimientos del proyecto.

En esta imagen se observa la representación gráfica del flujo de trabajo Git utilizado en el proyecto *Experiencias Significativas*.

**El GRAPH permite visualizar:**

- La línea de tiempo de todos los commits realizados.
- La separación entre ramas como *dev*, *qa*, *staging* y *main*.
- Las ramas individuales creadas para cada Historia de Usuario (HU-XX-dev, HU-XX-qa, HU-XX-staging).
- Los merges entre ramas y el recorrido que siguen las funcionalidades.
- Los puntos específicos donde se realizan **cherry-picks**, permitiendo traer commits puntuales desde otras ramas sin necesidad de hacer un merge completo.

- Se aprecia la estabilidad en main al no recibir cambios directos.
- 



Esta vista es fundamental para entender cómo evoluciona el proyecto y cómo se integran los cambios entre los diferentes ambientes, asegurando trazabilidad y orden en el ciclo de desarrollo.

## **8. Conclusiones**

- El manejo de ambientes permitió mantener un flujo organizado.
- El frontend fue el repositorio con implementación completa.
- Backend y móvil aplicaron el flujo parcialmente.
- El esquema garantiza calidad, orden y trazabilidad.