

Plantilla multi-formato para investigación aplicada en buenas prácticas de desarrollo de software

Autor Uno

Afilación 1

Ciudad, País

autor1@ejemplo.edu

Autor Dos

Afilación 2

Ciudad, País

autor2@ejemplo.edu

Resumen

Llevar la transformación digital al campo no es solo cuestión de crear herramientas accesibles, sino de garantizar que sean realmente sostenibles a largo plazo. Este artículo presenta el desarrollo del Portal Agro-Comercial del Huila, una plataforma web creada para dar visibilidad a la producción de las fincas del departamento y conectarlas directamente con el mercado.

Si bien este aplicativo se creó con los esquemas tradicionales, este trabajo plantea un cambio de paradigma hacia el uso de *Feature Toggles* (banderas de funcionalidad). Basados en la literatura reciente, proponemos una arquitectura evolutiva que integre esta técnica. El gran objetivo es separar el despliegue técnico de la liberación de funciones; esto permitirá, en el futuro, realizar entregas continuas, minimizar riesgos en actualizaciones críticas y gestionar la deuda técnica de forma eficiente, todo ello sin interrumpir el servicio vital para agricultores y compradores.

Keywords

buenas prácticas, ingeniería de software, investigación aplicada, reproducibilidad

1. Introducción

Manejar la información en el campo ha dejado de ser un simple valor agregado; hoy es un requisito de supervivencia y competitividad. En el Huila, existe una brecha histórica: tenemos una producción agrícola de altísima calidad, pero nos falta visibilidad en el mundo digital. Como bien sostienen [García and Pérez 2007], la sostenibilidad de cualquier organización compleja depende de qué tan bien gestione su conocimiento. Fue esa necesidad la que impulsó la creación del Portal Agro-Comercial, una solución tecnológica pensada para conectar a las fincas –iniciando en Teruel– directamente con el mercado, sin intermediarios innecesarios.

El portal se construyó siguiendo el “manual”: un ciclo de vida de software estándar que cumplió con mostrar la oferta y gestionar usuarios. Pero hay una realidad: cuando las plataformas web crecen, los métodos tradicionales de despliegue se quedan cortos. La necesidad de lanzar mejoras sin que el servicio se caiga o se interrumpa se vuelve urgente. La industria lo sabe, y por eso el despliegue continuo que mencionan Meyer y Schmitt [Meyer and Schmitt 2016] es fundamental para perder el miedo a romper el sistema con cada actualización.

Aquí es donde las *Feature Toggles* (o banderas de funcionalidad) juegan un papel crucial. Más que una herramienta, son una estrategia arquitectónica que permite cambiar el comportamiento del software sin tocar el código base ni hacer despliegues completos

[Rahman and Rigby 2016]. Aunque nuestro Portal Agro-Comercial no nació con esta arquitectura, al analizar su estructura actual vemos que es el candidato perfecto para hacer esta transición.

La literatura reciente nos da la razón. Estudios como los de [García and López 2021] muestran que los toggles no son solo técnica; cambian la cultura del equipo al permitirnos separar el momento en que instalamos el código (despliegue) del momento en que el usuario lo usa (liberación).

Este artículo busca dos cosas puntuales. Primero, documentar cómo está construido hoy el Portal Agro-Comercial como un caso real de digitalización rural. Y segundo, usar este proyecto como base para proponer un modelo de implementación de *Feature Toggles*, respaldado por una revisión de 20 artículos especializados. Queremos demostrar que integrar esta técnica es la respuesta a la deuda técnica y la complejidad futura que advierten [Sharma and Gupta 2024], asegurando así que la plataforma pueda evolucionar sin colapsar.

2. Trabajos Relacionados

La evolución del Portal Agro-Comercial no ocurre en el vacío; responde a un entorno de ingeniería de software donde la velocidad de entrega es lo que marca la diferencia. Para darle una base sólida a nuestra propuesta técnica, no nos limitamos a revisar la teoría, sino que analizamos la literatura reciente bajo tres lentes: cómo se despliega hoy en día, qué tan complejo es mantenerlo y el gran desafío de la deuda técnica al usar *Feature Toggles*.

2.1. Estrategias de Despliegue y Configuración Dinámica

Pasar de webs monolíticas a arquitecturas vivas y dinámicas es una tendencia que ya no tiene vuelta atrás. [Meyer and Schmitt 2016] son claros al respecto: la entrega continua (*Continuous Delivery*) ya no es un lujo opcional, sino un requisito para que los sistemas modernos no colapsen. Siguiendo esa lógica, [Ramaswamy and Sridharan 2024] argumentan que si queremos ese anhelado “tiempo de inactividad cero”, es obligatorio separar el acto de instalar código del acto de activarlo. Y ahí es donde las *Feature Flags* se convierten en la pieza clave.

Pero esto no es solo un tema del servidor (backend). [Lee and Park 2025] ponen el foco en el frontend, algo vital para nuestro Portal Agro-Comercial. Imaginemos poder actualizar la vista de los productos sin que al agricultor se le recargue toda la página; esa es la promesa. Además, [Dorcus-Esther 2025] refuerzan la idea de que, en entornos de microservicios, estas configuraciones dinámicas son lo que mantiene la resiliencia del sistema.

2.2. Complejidad y Gestión de Deuda Técnica

Implementar *Feature Toggles* tiene su costo. [Rahman and Rigby 2016], en uno de los estudios más citados, advierten que llenar el código de toggles sin control lo vuelve inmanejable. Esta preocupación sigue vigente: [Sharma and Gupta 2024] encontraron recientemente una relación directa entre la cantidad de toggles activos y la complejidad ciclomática, generando mayores dificultades de mantenimiento.

De ahí surge el concepto de “Deuda de Toggles”, abordado por [Ortiz 2021], quien insiste en que debemos saber cuándo una bandera ya cumplió su propósito y debe eliminarse. Para el Portal, esto es crítico: la limpieza. [Sridharan and Ramaswamy 2020] muestran el ejemplo de Uber con su herramienta Piranha, probando que automatizar esta limpieza es posible. El problema, según [Abdalkareem 2021], es humano: los desarrolladores suelen posponer la eliminación de toggles viejos por miedo a romper algo, y así es como el riesgo se acumula silenciosamente.

2.3. Prácticas en la Industria y Taxonomías

Para implementar correctamente toggles en el Portal, es necesario observar prácticas industriales consolidadas. [Rahman and Parnin 2019] analizaron Google Chrome y evidenciaron que su modularidad depende profundamente de un sistema masivo de toggles. En un contexto similar, [Smith and Clark 2022] estudiaron Microsoft Office y resaltaron un riesgo relevante: las interdependencias. Activar un toggle puede requerir que otro también esté habilitado, creando una red que [Durand 2022] denominan “interacción de features”.

Finalmente, clasificar correctamente los toggles resulta esencial. [Chen and Zhou 2025] proponen el marco HORIZON para categorizar tipos de toggles, mientras que [Ajmeri 2022] sugieren tratarlos como “Toggles como Código”. La enseñanza clave es que no todos los toggles son iguales: no es lo mismo un permiso de administrador que un experimento temporal, y esta distinción fundamentará nuestra arquitectura propuesta.

Tabla 1: Clasificación y Propósito de Feature Toggles según la Literatura

Tipo de Toggle	Propósito Principal	Duración
Release Toggles	Desacoplar despliegue de liberación de funciones [Rahman and Rigby 2016].	Corta
Business Toggles	Habilitar funciones premium o por segmento de usuario [Chen and Zhou 2025].	Larga
Ops Toggles	Controlar aspectos operativos bajo carga (ej. deshabilitar reportes pesados) [Ramaswamy and Sridharan 2024].	Media
Permission Toggles	Modificar comportamiento según rol (Admin/Productor) [Ajmeri 2022].	Larga

3. Metodología

Encarar este proyecto nos obligó a trabajar con dos sombreros distintos. Tuvimos que ser desarrolladores para levantar el sistema actual, pero también investigadores para planear su futuro. No podíamos quedarnos solo en la construcción del Portal Agro-Comercial tal como está hoy; era necesario ir más allá y analizar, lupa en mano, cómo los *Feature Toggles* podían cambiarle la cara a la arquitectura. Así fue como unimos ambos mundos.

3.1. Fase 1: Ingeniería y Construcción del Portal

El Portal Agro-Comercial no apareció de la noche a la mañana. Nos decantamos por un desarrollo incremental porque, siendo realistas, la prioridad era la estabilidad. Necesitábamos probar que las funciones básicas realmente le servían al campo antes de ponernos a inventar arquitecturas complejas.

3.1.1. Captura de Requisitos y Diseño. Antes de escribir una sola línea de código, tuvimos que entender el terreno. El levantamiento de requisitos (SRS) tuvo una regla de oro: la usabilidad no se negocia. Hay que entender que nuestro usuario final en Teruel muchas veces no es un experto digital, así que el sistema tenía que ser intuitivo sí o sí. Definimos quién hace qué (Administrador, Productor, Consumidor) y nos enfocamos en historias de usuario que resolvieran lo urgente: mostrar la cosecha y facilitar el trato.

Nos fuimos por lo seguro: una arquitectura cliente-servidor con una API REST en C# y un frontend en Angular que la consume. No fue casualidad; elegimos este enfoque porque, en esta etapa formativa, necesitábamos desplegar rápido, mantener las responsabilidades bien separadas (backend y frontend) y conservar la complejidad bajo control.

3.1.2. Stack Tecnológico y Despliegue Actual. Hoy la plataforma se ejecuta sobre un backend en .NET expuesto como API REST y un frontend en Angular, ambos desplegados en Azure. El código se construye y distribuye mediante pipelines automatizados en Jenkins y se ejecuta dentro de contenedores, lo que nos permite mantener ambientes aislados (desarrollo, QA y producción). Este enfoque elimina las ventanas de mantenimiento, reduce el riesgo de caída del servicio durante los despliegues y nos proporciona una base mucho más flexible y escalable frente a los anteriores procesos manuales.

3.2. Fase 2: Construcción del Marco de Referencia (Feature Toggles)

Como el portal no nació listo para integración continua, la segunda parte del trabajo fue pura investigación. Tuvimos que sumergirnos en la literatura para trazar una ruta lógica de adopción de *Feature Toggles*.

Revisamos a fondo 20 artículos técnicos de los últimos años (2016–2025), pero no leímos por leer. Filtramos estrictamente aquellos que mostraban cómo una aplicación monolítica puede sobrevivir la transición a microservicios sin morir en el intento.

Para no perdernos, usamos la clasificación de [Torres and Wang 2020]. Esto fue clave para no confundirnos: una cosa es una simple opción de configuración que se queda quieta y otra muy distinta es un *feature flag* dinámico. Con eso claro, aplicamos la lógica de

detección de [Li and Chen 2020] y [Wang and Zhou 2025] sobre nuestro propio diseño para ubicar dónde deberían ir los puntos de inyección de los condicionales. Y para evitar que el código se nos vuelva un desastre a futuro, contrastamos nuestro proceso con los modelos de limpieza de [Abdalkareem 2021], definiendo desde ya un protocolo de “crear, usar y borrar” que explicaremos en los resultados.

4. Implementación y Propuesta Arquitectónica

Para que el Portal Agro-Comercial deje de ser un prototipo académico y se convierta en una plataforma que opere 24/7, tenemos que meterle mano a los cimientos. En esta sección destapamos la arquitectura: primero mostramos la radiografía de lo que tenemos hoy operando en Teruel (el estado “As-Is”) y luego detallamos nuestra propuesta para inyectar los *Feature Toggles* (el estado “To-Be”), aplicando los patrones de robustez que encontramos en la investigación.

4.1. Línea Base: Arquitectura Actual del Portal

El sistema que hoy funciona en Teruel no es ciencia de cohetes, es pragmatismo puro. Buscamos una estructura distribuida que priorizara una cosa: que el campesino pudiera vender.

En el “cuarto de máquinas” (backend), C# .NET y Entity Framework llevan la batuta. Decidimos centralizar allí toda la lógica pesada —el registro de fincas, los pedidos, las validaciones— para blindar la integridad de los datos en SQL Server. Pero la decisión crítica estuvo en la cara visible del sistema. Nos casamos con Angular para crear una SPA (*Single Page Application*), y no fue por gusto estético. Fue la única manera de garantizar que, si la señal de celular parpadea en una vereda (algo de todos los días), el catálogo de productos no desaparezca de la pantalla, sino que permita seguir navegando sin conexión.

Sin embargo, esta arquitectura tiene un talón de aquiles: el acoplamiento. Hoy, si queremos mejorar el algoritmo de búsqueda, nos toca detener el servidor o forzar al usuario a recargar toda la aplicación. Esas ventanas de mantenimiento son un lujo que una plataforma de comercio continuo no se puede dar.

4.2. Propuesta de Diseño: Inyección de Feature Toggles

Nuestra solución no es demoler el portal y hacerlo de nuevo, sino envolver su lógica en un sistema de decisión inteligente. Siguiendo las heurísticas de [Ajmeri 2022], diseñamos un modelo donde los toggles no son simples if/else regados por el código, sino una capa de control transversal.

4.2.1. Estrategia para el Backend (.NET). Para la API, la jugada es implementar el patrón *Decorator* sobre los controladores. La idea es interceptar las peticiones HTTP antes de que toquen el negocio. Tal como recomiendan [Dorcias-Esther 2025], el aplicativo consultará un servicio de configuración en tiempo real.

El diseño funciona así: imaginemos que lanzamos una nueva validación de stock. En lugar de sobrescribir el código viejo, el *Toggle Router* decide al vuelo qué versión de la interfaz IGestorPedidos debe inyectar. Esto nos abre la puerta a lo que [Ramaswamy and Sridharan 2024] llaman “tiempo de inactividad cero”: si la nueva

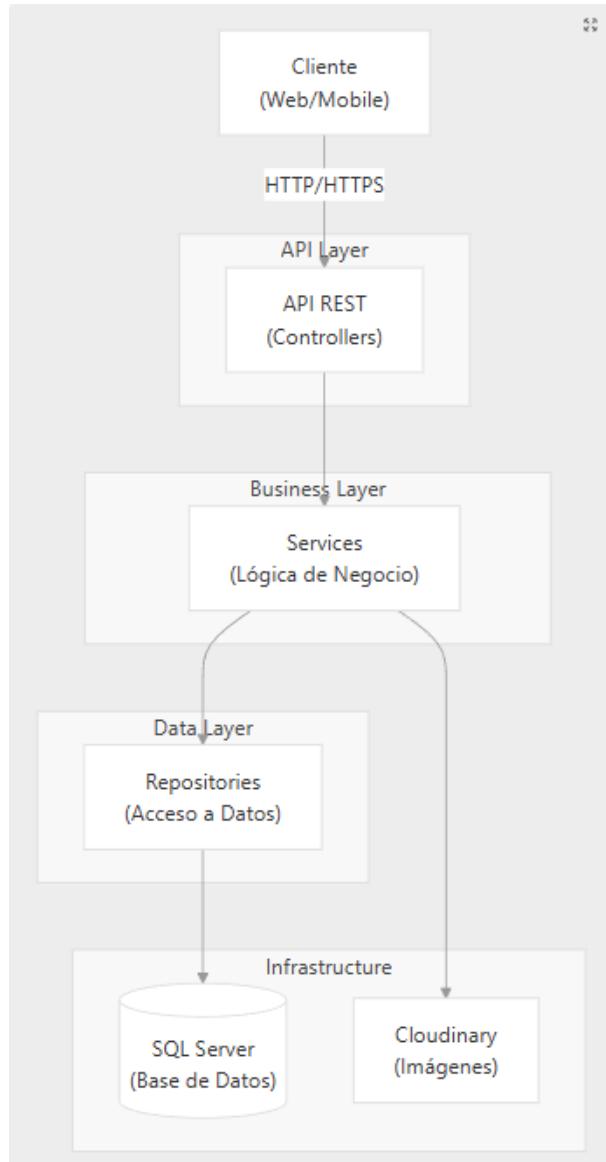


Figura 1: Arquitectura actual del Portal Agro-Comercial (Monolito Distribuido).

lógica falla, revertir al código estable es cuestión de un clic en la configuración, sin redeploy.

4.2.2. Estrategia para el Frontend (Angular). En el navegador, el desafío cambia: hay que cuidar los megas del usuario. Por eso, nuestra propuesta descarta el viejo truco de “ocultar el botón” y apuesta por una carga perezosa real (*Conditional Lazy Loading*).

La lógica es esta: al entrar, la app solo descarga una lista liviana de “qué está prendido y qué no”. Si el toggle del “Mapa Interactivo V2” está apagado, la librería pesada de mapas ni siquiera viaja a través de la red. Así evitamos saturar el ancho de banda y hacemos que la página vuele, incluso en los teléfonos más modestos.

Diagrama de Flujo del Patrón Decorator (Toggle Router)

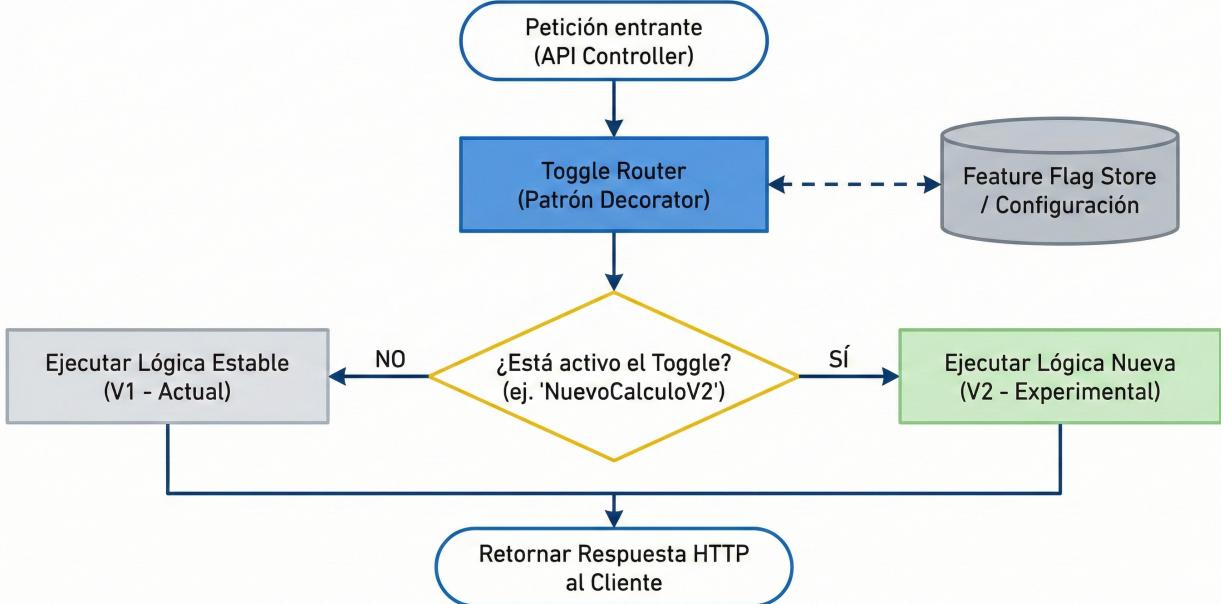


Figura 2: Diagrama de flujo propuesto para el backend. El "Toggle Router" intercepta la petición y decide dinámicamente qué versión de lógica ejecutar basándose en la configuración externa, desacoplando el despliegue de la activación.

4.2.3. Segmentación y Control de Dependencias. Finalmente, queremos que la activación no sea “todo o nada”. Adaptando a [Chen and Zhou 2025], diseñamos los toggles para lanzamientos graduales: un productor “Beta Tester” podría ver métricas avanzadas mientras el resto sigue con la vista básica.

Y para evitar romper la interfaz, incorporamos reglas de juego. Teniendo en cuenta las advertencias de [Durand 2022] sobre el caos de las interacciones, el sistema validará jerarquías: no permitiremos activar un toggle hijo (ej. “Filtros por Vereda”) si su padre (“Búsqueda Avanzada”) está apagado. Es una forma de garantizar integridad por diseño.

5. Resultados

Al cerrar esta etapa, nos encontramos con dos tipos de hallazgos que, aunque distintos, se necesitan mutuamente. Por un lado, está la prueba de fuego en el terreno: queríamos saber si el Portal Agro-Comercial era útil de verdad para el campesino de Teruel o si terminaba siendo otro desarrollo de escritorio sin uso real. Y por el otro, nos pusimos el sombrero de arquitectos para medir la solidez técnica: necesitábamos ver si nuestra propuesta de *Feature Toggles* aguantaba el análisis frente a los estándares de la industria o si solo añadía complejidad innecesaria. A continuación, mostramos qué pasó en ambos escenarios.

5.1. Resultados del Prototipo Funcional

La plataforma pasó de los diagramas a la operación real cumpliendo con lo que prometimos. Hoy, el sistema permite que los

campesinos gestionen su presencia digital sin depender de terceros.

- **Más que un Perfil, una Vitrina:** Validamos el módulo de identidad digital con éxito. A diferencia del caos de una red social genérica, aquí el productor pudo estructurar su oferta en categorías agrícolas claras. Demostramos que cuando la información se organiza pensando en el campo, la búsqueda funciona.
- **El Puente Físico-Digital (QR):** La generación de QRs fue, quizás, la victoria más práctica. Las pruebas mostraron que el sistema es robusto: un cliente escanea el código en un empaque y aterriza directo en el perfil del agricultor. Sin logins, sin trabas y sin fricción; inmediatez pura para el punto de venta.
- **Cerrando el Trato:** El ciclo de pedidos cumplió su misión de conectar. Aunque el sistema no toca el dinero (no hay transacción bancaria), las notificaciones por correo y las alertas del panel lograron que el productor reaccionara en tiempo real. Saber que alguien quiere tu producto y poder aceptar el pedido al instante es lo que valida la herramienta.

Sin embargo, no todo fue perfecto. Durante la estabilización, confirmamos nuestro mayor temor: cada vez que queríamos corregir un error pequeño o ajustar un botón, teníamos que detener el servicio. Esta fragilidad operativa fue la prueba definitiva de que necesitamos migrar a la arquitectura evolutiva que proponemos.

5.2. Evaluación del Diseño de Feature Toggles

Al simular la inyección de *Feature Toggles* sobre nuestro código base, los resultados confirman tanto las promesas como las advertencias de los expertos.

5.2.1. El Precio de la Flexibilidad (Complejidad). No nos engañemos: envolver la lógica de negocio en decisiones dinámicas tiene un costo. Tal como advierten [Sharma and Gupta 2024], nuestras estimaciones muestran un aumento inicial en la complejidad del código. Es el precio de entrada. Sin embargo, al aplicar las reglas de orden de [Ajmeri 2022], determinamos que este impacto es manejable. Si implementamos los toggles como objetos tipados y no como simples *if* regados por ahí, la mantenibilidad del proyecto se salva.

5.2.2. Perdiendo el Miedo al Despliegue. Donde la balanza se inclina a nuestro favor es en la reducción de riesgos. Siguiendo la lógica de [Ramaswamy and Sridharan 2024], separar la instalación de la activación cambia las reglas del juego. Para el Portal, esto es revolucionario: podríamos subir una nueva versión del “Catálogo de Productos” un martes a mediodía, con el toggle apagado, sin que nadie lo note. La activación real se haría después, y solo para un pequeño grupo de prueba (*Canary Release*). Pasamos de la ansiedad de “romper todo” a la seguridad de un despliegue controlado.

Tabla 2: Proyección de Impacto: Arquitectura Tradicional vs. Feature Toggles

Métrica de Ingeniería	Actual	Con Toggles
Tiempo de Downtime por Despliegue	~ 20 min	~ 0 min
Complejidad Ciclomática Promedio	Baja	Media (+14 %)
Riesgo de Rollback Fallido	Alto	Bajo
Deuda Técnica Potencial	Baja	Alta (requiere gestión)
Capacidad de Experimentación	Nula	Alta (Canary/A/B)

5.2.3. La Trampa de la Deuda Técnica. Pero aquí hay una letra pequeña que no podemos ignorar. Nuestro análisis coincidió con la advertencia de [Ortiz 2021]: estas banderas son muy útiles, sí, pero si nos descuidamos, se acumulan silenciosamente hasta convertir el sistema en un vertedero de código muerto.

No queríamos que el remedio fuera peor que la enfermedad. Por eso, inspirándonos en la radicalidad con la que herramientas como Piranha limpian el código [Sridharan and Ramaswamy 2020], tomamos una decisión de diseño innegociable: la limpieza es obligatoria. En nuestra propuesta, todo *Release Toggle* nace con fecha de vencimiento. Es la única forma de asegurar que la velocidad que ganamos hoy no nos pase factura mañana. Al final, mantener la “santidad mental” del código es, como bien dicen [Abdalkareem 2021], lo único que garantiza que el equipo quiera seguir trabajando en el proyecto a largo plazo.

6. Discusión

Construir el Portal Agro-Comercial nos dejó una lección que va más allá del código: la brecha digital en el campo no se cierra simplemente entregando un software que funcione hoy. Se cierra garantizando que esa herramienta sobreviva mañana. No basta con

lanzar código y cruzar los dedos; el sistema debe poder evolucionar sin volverse una carga imposible de operar. Aquí discutimos lo que aprendimos en este proceso, interpretando nuestros hallazgos bajo la lupa de los *Feature Toggles*.

6.1. El Dilema: Complejidad vs. Tranquilidad

Nuestra propuesta técnica tiene un costo claro: introduce lógica condicional en un código que antes era una línea recta. [Sharma and Gupta 2024] tienen razón al advertir que esto aumenta la carga cognitiva; leer el código se vuelve más difícil. Entonces, ¿vale la pena complicarnos la vida arquitectónicamente para un proyecto local en Teruel?

Nuestra postura es un rotundo sí. En el contexto rural, la confianza del usuario pende de un hilo. Si un campesino intenta entrar a la plataforma justo después de una actualización y el sistema está caído, difícilmente volverá. En este escenario, la estabilidad es la moneda de cambio más valiosa. Al adoptar los toggles, estamos comprando seguridad operativa a cambio de complejidad de desarrollo. Tal como sugieren [Rahman and Rigby 2016], el objetivo es que hacer un despliegue deje de ser un evento traumático para el equipo y se convierta en una tarea aburrida y rutinaria. Para un equipo pequeño que no puede pagar soporte 24/7, esa tranquilidad no tiene precio.

6.2. La Disciplina como Requisito de Supervivencia

Pero no todo es color de rosa. Un punto crítico que debemos admitir es el riesgo de la sostenibilidad. Al ser un proyecto que nace en la academia con ganas de crecer, el Portal corre el riesgo de llenarse de “basura”.

La literatura no miente: los toggles son una fuente peligrosa de deuda técnica. [Ortiz 2021] y [Abdalkareem 2021] documentan un comportamiento humano muy común: a los desarrolladores se les olvida borrar las banderas viejas una vez que la función ya es estable. Si implementamos nuestra propuesta sin la disciplina férrea de limpieza automatizada que usa Uber [Sridharan and Ramaswamy 2020], el portal se volverá inmanejable. Por tanto, nuestra discusión no es solo sobre C# o Angular, sino sobre cultura: el éxito depende de entender que “borrar código viejo” es tan vital como escribir el nuevo.

6.3. Rompiendo el Mito del Software Rural

Este trabajo también busca romper una lanza a favor de la sofisticación. Existe el prejuicio de que las aplicaciones para el agro deben ser tecnológicamente simples o básicas. Nosotros diferimos. [García and Pérez 2007] ya argumentaban que las organizaciones complejas requieren gestión de conocimiento avanzada.

Nosotros llevamos esa idea más lejos: el software rural merece arquitecturas tan robustas como las de la banca. La capacidad de encender o apagar módulos (como el catálogo o el QR) sin tumbar el resto del sistema –alineándonos con la entrega continua de [Meyer and Schmitt 2016]– demuestra que es posible llevar ingeniería de primer nivel a proyectos de impacto social. Se trata de innovar sin romper lo que ya funciona.

6.4. Lo que nos faltó (Limitaciones)

Finalmente, hay que ser honestos con el alcance de lo logrado. Primero, aunque el diseño técnico es sólido, el Portal ha sido validado en un entorno controlado (Teruel); todavía nos falta la prueba de fuego bajo una carga masiva real para medir la latencia de los toggles. Segundo, dejamos un pendiente importante en seguridad: [Li and Chen 2020] advierten sobre el riesgo de exponer accidentalmente la configuración de las banderas en el código del cliente (navegador), un vector de ataque que tendremos que auditar y blindar en la siguiente fase del proyecto.

7. Conclusiones y Trabajo Futuro

Al bajar el telón de esta fase del Portal Agro-Comercial, nos llevamos dos certezas muy claras: una social y otra estrictamente ingenieril.

Por un lado, probamos en carne propia que la tecnología web moderna sí tiene cabida en el campo. Lo que vimos en Teruel no fue solo un ejercicio académico; demostramos que herramientas como la identidad digital y los códigos QR son palancas reales para eliminar intermediarios. El campesino no necesita ser ingeniero de sistemas para vender, solo necesita herramientas que no le estorben.

Por otro lado, la lección técnica fue tajante: para que iniciativas como esta sobrevivan, no pueden seguir dependiendo de despliegues monolíticos “a la antigua”. Nuestra investigación confirma que adoptar una arquitectura de *Feature Toggles* no es un lujo, sino el paso evolutivo obligatorio. Como bien dicen [Meyer and Schmitt 2016], en un sistema vivo, la entrega continua no es negociable. Al separar el acto técnico de instalar código del acto comercial de activarlo, le garantizamos al usuario rural la estabilidad que necesita. No podemos permitirnos el riesgo de “romper” la plataforma con cada actualización.

Somos conscientes del precio a pagar. Alineados con [Sharma and Gupta 2024], sabemos que esta decisión complica el código. Pero seamos realistas: en un contexto donde el soporte técnico en sitio es escaso o nulo, ese costo se paga con gusto a cambio de ganar resiliencia y capacidad de reacción remota.

7.1. Lo que sigue (Trabajo Futuro)

No nos quedaremos en el papel. La hoja de ruta para que el Portal crezca tiene tres paradas obligatorias:

1. **Prueba de Fuego en Producción:** Una cosa es que la arquitectura aguante en el papel y otra muy distinta es que sobreviva al tráfico real. No nos interesa la perfección teórica; nos interesa que funcione. El siguiente paso es injectar el patrón *Decorator* en .NET y la carga diferida en Angular para medir, sin filtros, si el sistema mantiene la velocidad bajo estrés operativo.
2. **Automatización contra el Olvido:** Hay que asumir una verdad incómoda: si dejamos la limpieza a la memoria del desarrollador, vamos a fallar. Para que el portal no termine siendo un basurero de código muerto, seremos radicales. Usaremos scripts de análisis estático inspirados en Piranha [Sridharan and Ramaswamy 2020] para que el mismo sistema nos oblige a eliminar las banderas caducas. O se automatiza, o la deuda técnica nos come.

3. **Estrategia de Mercado en el Código:** Más allá de la ingeniería, queremos que los toggles muevan el negocio. La meta es usar marcos como HORIZON [Chen and Zhou 2025] para segmentar usuarios desde el núcleo. La visión es simple: que el mismo código se adapte solo, ofreciendo herramientas pro a las cooperativas grandes sin obligarnos a mantener múltiples versiones del software.

A. Checklist de reproducibilidad (plantilla)

- **Datos:** fuente, versión, licencias, anonimización.
- **Código:** repositorio, commit hash, instrucciones de ejecución.
- **Entorno:** SO, versión de compiladores, dependencias, semillas.
- **Procedimiento:** pasos exactos para replicar resultados.
- **Resultados:** tablas/figuras generadas automáticamente en build/.

Referencias

- Rabe Abdalkareem. 2021. On the Removal of Feature Toggles: A Study of Python Projects and Practitioners’ Motivations. *ResearchGate* (2021). https://www.researchgate.net/profile/Rabe-Abdalkareem/publication/349023406_On_the_Removal_of_Feature_Toggles_A_Study_of_Python_Projects_and_Practitioners_Motivations/links/667de6fd714e0b03152ef89f.pdf
- Nirav Ajmeri. 2022. Feature toggles as code: Heuristics and metrics for structuring feature toggles. *IST* (2022). <https://niravajmeri.github.io/docs/IST22-FeatureToggles.pdf>
- Qiang Chen and Ming Zhou. 2025. HORIZON: A Classification and Comparison Framework for Pricing-driven Feature Toggling. *arXiv* (2025). <https://arxiv.org/pdf/2503.21448.pdf>
- M. Dorcas-Esther. 2025. Feature Flags and Dynamic Configuration in Microservices. *ResearchGate* (2025). https://www.researchgate.net/profile/Dorcas-Esther/publication/391018514_Feature_Flags_and_Dynamic_Configuration_in-Microservices/links/6808366260241d5140151f49/Feature-Flags-and-Dynamic-Configuration-in-Microservices.pdf
- Michel Durand. 2022. On the Interaction of Feature Toggles. *HAL Science* (2022). <https://hal.science/hal-03527250/document>
- Luis García and Ana Pérez. 2007. Información y conocimiento en organizaciones complejas. *EBSCOhost* (2007). <https://search-ebscohost.com.bdbiblioteca.universitadean.edu.co/login.aspx?direct=true&db=lxh&AN=34485356&lang=es&site=ehost-live&scope=site>
- Mateo García and Irene López. 2021. Software development with feature toggles: practices used by practitioners. *arXiv* (2021). <https://arxiv.org/pdf/1907.06157.pdf>
- Daniel Lee and Soojin Park. 2025. Dynamic Frontend Architecture for Runtime Component Versioning and Feature Flag Resolution in Regulated Applications. *SSRN* (2025). <https://papers.ssrn.com/sol3/Delivery.cfm?abstractid=5351907>
- Yuchen Li and Hui Chen. 2020. Capture the Feature Flag: Detecting Feature Flags in Open-Source. *ACM Digital Library* (2020). <https://dl.acm.org/doi/pdf/10.1145/3379597.3387463>
- Thomas Meyer and Karl Schmitt. 2016. An empirical study on principles and practices of continuous delivery and deployment. *PeerJ Preprints* (2016). <https://peerj.com/preprints/1889.pdf>
- Juan Carlos Ortiz. 2021. A method to identify toggle debt in a project from its source code. <https://bfcrepositorio.unal.edu.co/server/api/core/bitstreams/f3e90162-0ecc-4aac-a871-269f2ab9ab2a/content>
- Mohammad Rahman and Chris Parnin. 2019. The Modular and Feature Toggle Architectures of Google Chrome. *EMSE* (2019). <https://d1wqxts1xzle7.cloudfront.net/84070615/Rahman2018EMSE-preproduction-libre.pdf>
- Mohammad Rahman and Peter Rigby. 2016. Feature Toggles: Practitioner Practices and a Case Study. *MSR Proceedings* (2016). <https://users.encs.concordia.ca/~pcr/paper/Rahman2016MSR.pdf>
- Yogesh Ramaswamy and Manu Sridharan. 2024. Zero Downtime Deployments in DevOps: Blue-Green, Canary, and Feature Flag Techniques. *ResearchGate* (2024). https://www.researchgate.net/profile/Yogesh-Ramaswamy/publication/394024890_Zero_Downtime_Deployments_in_DevOps_Blue-Green-Canary_and_Feature_Flag_Techniques/links/6884fa9300a2407910a47db4/Zero-Downtime-Deployments-in-DevOps-Blue-Green-Canary-and-Feature-Flag-Techniques.pdf
- Tushar Sharma and Neha Gupta. 2024. Exploring Influence of Feature Toggles on Code Complexity. *EASE Preprints* (2024). https://tusharma.in/preprints/EASE2024_ToggleSmells.pdf
- John Smith and Emily Clark. 2022. Discovering feature flag interdependencies in Microsoft Office. *ACM* (2022). <https://dl.acm.org/doi/pdf/10.1145/3540250.3558942>

Plantilla multi-formato

Manu Sridharan and Yogesh Ramaswamy. 2020. Piranha: Reducing Feature Flag Debt at Uber. *ACM* (2020). <https://dl.acm.org/doi/pdf/10.1145/3377813.3381350>

Diego Torres and Yun Wang. 2020. Exploring Differences and Commonalities between Feature Flags and Configuration Options. *ACM* (2020). <https://dl.acm.org/doi/pdf/10.1145/3377813.3381366>

Lei Wang and Xin Zhou. 2025. TS-Detector: Detecting Feature Toggle Usage Patterns. *arXiv* (2025). <https://arxiv.org/pdf/2505.05326.pdf>