



FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo para Alumnos con Experiencia Profesional

TÍTULO: Implementación de un programa de fidelidad

AUTOR/A: Franco Emanuel Liptak

DIRECTOR/A ACADÉMICO: Claudia Banchoff - Matías Pagano

DIRECTOR/A PROFESIONAL: Rodrigo Martín López Gregorio

CODIRECTOR/A ACADÉMICO: -

CARRERA: Licenciatura en Sistemas

RESUMEN

Esta tesina presenta la participación del autor en el desarrollo del programa de fidelidad de Despegar, desde su implementación inicial hasta sus iteraciones más recientes. El trabajo aborda la elección de tecnologías para el front-end y back-end, una propuesta de refactoring orientada a estandarizar componentes reutilizables, y la posterior adaptación del programa al modelo de white labels utilizado por el sitio.

Palabras Claves

- Programa de fidelidad
- Front-end
- Back-end
- Componente reutilizable
- A/B testing
- White label
- Microservicios

Conclusiones

Esta tesina fue una oportunidad para repasar una experiencia clave en la carrera de quien escribe.

El programa de fidelidad desarrollado sigue generando valor, tanto para Despegar como para empresas asociadas, gracias a una solución tecnológica sólida y fácil de escalar.

Trabajos Realizados

Basada en la experiencia del autor, la tesina recorre los aspectos técnicos y decisiones más importantes del desarrollo del programa de fidelidad. Se incluyen gráficos e imágenes para facilitar la comprensión.

Trabajos Futuros

1. Servir componentes desde **s-commons-vr** para simplificar dependencias.
2. Usar **s-commons-vr** en el flujo principal para manejar lógica desde backend.
3. Evaluar tecnologías más desacopladas como **Stencil**.

Índice general

Capítulo 1. Introducción.....	5
1.1 Introducción.....	5
1.2 Motivación.....	7
1.3 Objetivos.....	9
1.4 Aporte del autor.....	10
Capítulo 2. Descripción funcional del sistema de fidelidad.....	12
2.1. Visión general del programa.....	12
2.2. Funcionamiento desde la perspectiva del usuario.....	13
2.3 Despegar: contexto y navegación.....	14
2.4 Aspectos generales de la implementación del programa.....	19
Capítulo 3. Estado del arte: tecnologías y metodología en Despegar.....	22
3.1. Arquitectura de microservicios.....	22
3.2. Stack tecnológico predominante.....	25
3.3. Componentes compartidos en el Front-End.....	26
3.4. Metodología de desarrollo.....	27
3.5. Estrategia de experimentación y pruebas A/B.....	28
Capítulo 4. Primera implementación del programa de fidelidad.....	30
4.1. Requerimientos, objetivos y organización del proyecto.....	30
4.2. Integración del core de Loyalty.....	32
4.4. Componente Front-End compartido.....	34
4.4.1. Lógica funcional del componente.....	34
4.4.2. Arquitectura del componente y parámetros.....	35
4.4.3. Evaluación de tecnologías posibles.....	36
4.5. Back-End del componente Front-End.....	37
4.5.1. Elección de tecnología: ¿por qué Node.js?.....	37
4.5.2. Alternativas consideradas.....	38
4.6. Integración del programa en Alojamientos.....	39
4.7. Distribución e integración del componente compartido.....	39

4.8. Trackeo y medición.....	41
4.9. Pruebas y puesta en producción.....	42
4.10. Alarmas y monitoreo en la nueva aplicación.....	43
4.11. A/B Testing y análisis de resultados.....	43
4.11.1. Cómo se hacen los A/B tests en Despegar.....	43
4.11.2. A/B tests sincronizados.....	44
4.11.3. Hipótesis y criterios de evaluación.....	44
4.11.4. Trackeos y análisis de resultados.....	44
4.11.5. Qué se decidió a partir de esto.....	45
4.12. Conclusiones de la primera implementación.....	45
4.13. Iteraciones posteriores y evolución del programa.....	46
Capítulo 5. Unificación técnica y liderazgo en la implementación de componentes	
Angular.....	48
5.1. Contexto y motivación.....	48
5.2. La propuesta: una historia técnica transversal.....	49
5.3. Arquitectura de la solución.....	49
5.3.1. Componentes Angular unificados.....	50
5.3.2. Wrapper del loyalty-redemption-switch.....	50
5.3.3. Servicio Angular de estado compartido.....	50
5.3.4. Nuevo rol de s-commons-vr como back-end unificado.....	50
5.4. Coordinación y ejecución.....	52
5.5. Impacto y aprendizajes.....	53
Capítulo 6. Adaptación del programa de puntos al modelo de white labels.....	55
6.1. Particularidades del modelo white label.....	55
6.2. Reutilización de la implementación base.....	55
6.3. Configuración centralizada de comportamiento.....	56
6.4. El componente “switch” en white labels.....	56
6.5. Personalización por marca.....	56
6.6. Beneficios de la mejora técnica previa.....	57
6.7. Comparativa entre el sitio de Despegar y las white labels.....	57
6.8. Ejemplo visual de personalización en white labels.....	58
Capítulo 7. Conclusiones y próximos pasos.....	61

Referencias bibliográficas..... 64

Agradecimientos

Llegar al final de esta carrera universitaria representa para mí un momento muy importante, en el que quiero destacar a todas las personas que han formado parte de este recorrido.

En primer lugar, agradezco profundamente a mis padres, quienes, con gran esfuerzo, me brindaron la oportunidad de realizar los primeros años de esta carrera con un foco exclusivo que me permitió incorporar muchísimo conocimiento, el cual luego sería importante para poder insertarme en el mercado laboral y avanzar en paralelo hacia la finalización de mis estudios. Por supuesto, una mención especial también para mi hermano, que siempre está para acompañarme.

También agradezco especialmente a mi novia, cuyo apoyo fue muy importante a lo largo de toda la carrera.

Me gustaría destacar el trabajo de mis directores académicos, Claudia Banchoff y Matías Pagano, quienes me aconsejaron sobre cómo avanzar con la finalización de mis estudios, y también a mi director profesional, Rodrigo Martín López Gregorio. Los tres siempre estuvieron disponibles para ayudarme en este período.

A mis amigos, entre los que naturalmente destaco a quienes cursaron la carrera conmigo y a quienes compartieron equipo conmigo en Despegar. Además de su amistad, todos hicieron aportes a mi desarrollo profesional de manera directa o indirecta.

Y por último, si bien no están presentes físicamente, me gustaría hacer una mención especial a mis abuelos, quienes sin duda estarían muy contentos por este objetivo cumplido.

Capítulo 1. Introducción

1.1 Introducción

En los últimos años se puede observar cómo los programas de fidelidad han ganado gran relevancia en el ámbito empresarial, convirtiéndose en una estrategia importante para fomentar la retención de clientes y la recurrencia de compras. Según un informe de SEMrush (2024), el mercado global de gestión de fidelización alcanzó un valor estimado de 5.570 millones de dólares en 2023, y se proyecta que crecerá hasta 28.650 millones de dólares en 2030, evidenciando un crecimiento muy significativo.

Este crecimiento está relacionado con el interés de las empresas en establecer un vínculo de largo plazo con el cliente, en un contexto cada vez más competitivo. Además, de acuerdo con datos de Pretii (2024), el 90% de las compañías que implementan programas de lealtad reportan un retorno positivo de la inversión, con una rentabilidad promedio de 4,8 veces la inversión inicial, lo que refuerza su efectividad como estrategia de negocio.

Pero, ¿qué es un programa de fidelidad? En resumen, un programa de fidelidad es una estrategia de marketing diseñada para incentivar a los clientes a repetir compras o interacciones con una misma empresa, ofreciéndole algún beneficio como por ejemplo descuentos o puntos acumulables. Según HubSpot (2024), el objetivo principal de estos programas es fortalecer la lealtad del cliente hacia la marca y fomentar relaciones a largo plazo.

En el año 2018 la empresa Despegar comenzó a diseñar el proyecto que daría vida a su programa de fidelidad. Antes de la implementación de este programa Despegar siempre había tenido una estrategia mayormente orientada a atraer a nuevos clientes, con mucho foco en las campañas de marketing.

Una vez que los clientes ingresaban al sitio, la expectativa era (y sigue siendo) que encuentren una propuesta de valor atractiva basada en distintos pilares, entre los cuales podemos destacar: precio y financiación, confiabilidad en la marca y la posibilidad de adquirir todo lo necesario para el viaje en una única plataforma (ya que a través del sitio se pueden comprar pasajes de avión, reservar hoteles, actividades, autos, entre otras cosas).

Sin embargo, no se hacía demasiado foco en usuarios que ya habían comprado en el sitio. Si bien se enviaban mails con promociones generales y también se ofrecía alguna oferta durante un determinado periodo de tiempo posterior a la compra, cada vez que el usuario decidía realizar un nuevo viaje no tenía ningún incentivo diferencial para elegir nuevamente a Despegar por sobre la competencia.

Esto último hizo notar que se estaba desperdiciando una oportunidad de negocio, y este programa nació con el objetivo de explotar dicha oportunidad: ofrecer un incentivo a volver a comprar en el sitio a aquellos usuarios que previamente habían realizado alguna compra. ¿Cómo? Gracias a la implementación del programa **Pasaporte Despegar**, con cada compra los usuarios acumulan puntos que pueden canjear en el sitio para un futuro viaje.

Es importante destacar también que, más allá de que los usuarios pueden utilizar este beneficio en compras futuras, la posibilidad de acumular puntos desde la primera compra genera que también nuevos usuarios encuentren valor en el programa.

La implementación de Pasaporte Despegar requirió esfuerzo de equipos de distintas áreas de la empresa. Podemos mencionar algunos ejemplos de tareas que realizó cada sector:

- Legales:
 - Revisión de los términos y condiciones del programa.
 - Definición de cláusulas contractuales para alianzas con terceros (por ejemplo, bancos que comenzaron a emitir las tarjetas de crédito que permiten acumular puntos con cada compra, canjeables en Despegar).
 - Evaluación de riesgos legales por otorgar y permitir canjear puntos.
- Producto (en algunos casos referido como “Negocio”):
 - Diseño del modelo de fidelización (por ejemplo, sistema de acumulación y canje de puntos).
 - Definición de métricas clave de éxito (engagement¹ de usuarios, conversión², impacto en revenue³).
 - Coordinación con IT y UX para establecer requisitos funcionales del sistema.
- Experiencia de usuario (UX):
 - Diseño de la experiencia de usuario, adaptando la plataforma de Despegar para incluir componentes asociados al programa.
 - Pruebas de usabilidad.
- IT:
 - Desarrollo de la plataforma tecnológica que soporta el programa.
 - Integración con sistemas existentes.

El enfoque principal de esta tesina es el trabajo realizado por distintos equipos IT en el desarrollo e implementación del programa de fidelidad. Se expone el trabajo realizado para la primera iteración del programa, y también otras iteraciones que se hicieron luego.

¹ Grado de involucramiento o interacción sostenida de un usuario con una marca o producto.

² Acción clave del usuario que cumple un objetivo del negocio (como comprar o registrarse).

³ Ingreso bruto que una empresa obtiene por ventas, sin descontar gastos.

Para dar mayor contexto, también se mencionan aspectos asociados al programa de fidelidad y cómo era la arquitectura de aplicaciones de Despegar (en términos generales), para que se pueda entender mejor el desafío que representó implementar este programa, y cuáles fueron los requerimientos no sólo de la primera iteración sino también de las siguientes.

1.2 Motivación

Como se mencionó anteriormente, en los últimos años se ha vuelto cada vez más común que las empresas desarrollen programas de fidelización. En un mercado competitivo, no alcanza con atraer nuevos clientes: también es clave lograr que aquellos clientes que ya compraron vuelvan a elegir la empresa en el futuro. Contar con un programa de fidelidad puede resultar muy beneficioso, tanto para destacarse frente a la competencia como para no quedar en desventaja si otras empresas del mismo rubro ya tienen uno.

Esto se debe a que la fidelización de clientes no solo impacta en la recurrencia de compras, sino también en la percepción de marca y la relación a largo plazo con los consumidores. Un cliente fidelizado tiene más probabilidades de seguir eligiendo una empresa incluso cuando existen opciones similares en el mercado.

Este tipo de estrategias se pueden ver en distintas industrias en Argentina. Algunos ejemplos conocidos incluyen:

- Retail y supermercados:
 - Coto → Coto Más
 - Carrefour → Mi Carrefour
 - Día % → Club Día
- Bancos y tarjetas de crédito:
 - Banco Galicia → Quiero!
 - Banco Santander → SuperClub
 - Banco Macro → Macro Premia
- Combustibles y estaciones de servicio:
 - YPF → YPF ServiClub
 - Shell → Shell LATAM Pass

Más allá de las diferencias entre cada programa, es evidente que la fidelización de clientes es una estrategia cada vez más usada. Pero, ¿por qué tantas empresas eligen implementarla? ¿Realmente es necesario?

Reichheld y Scheffer (2000), en su artículo *E-Loyalty: Your Secret Weapon on the Web* publicado en la Harvard Business Review, afirman que las empresas que logran construir relaciones de lealtad con sus clientes tienden a ser más rentables y sostenibles en el tiempo que aquellas enfocadas únicamente en adquirir nuevos clientes. Según los autores, los consumidores leales no sólo repiten sus compras con mayor frecuencia, sino que además tienden a gastar más a lo largo del tiempo y a recomendar la marca a otros. Esto genera un efecto multiplicador en la adquisición de nuevos clientes sin necesidad de invertir grandes sumas en publicidad. Además, las relaciones duraderas permiten a las empresas reducir los costos asociados al servicio al cliente, ya que los usuarios recurrentes suelen requerir menos soporte y conocen mejor el funcionamiento de los productos o servicios. En contextos digitales, donde la competencia está a solo un clic de distancia, lograr ese tipo de compromiso es especialmente valioso y puede convertirse en una verdadera ventaja competitiva.

En el caso de Despegar, la implementación de un programa de fidelización se convirtió en un paso estratégico clave. En la industria del turismo y los viajes, donde la competencia es alta y los clientes tienen múltiples opciones para reservar vuelos, alojamientos y paquetes, lograr que un usuario vuelva a elegir la plataforma es fundamental. En este contexto, contar con un sistema que recompense la recurrencia no solo permite mejorar la experiencia del usuario, sino que también fortalece la posición de la empresa frente a otras plataformas del sector.

Sin embargo, más allá del impacto comercial, implementar un programa de fidelidad presenta desafíos técnicos significativos. Integrar este tipo de sistemas dentro de una arquitectura ya existente requiere considerar múltiples factores, como la gestión de puntos y beneficios, la comunicación con otras plataformas, la seguridad de los datos y la escalabilidad de la solución.

Es importante considerar que, previo a la implementación de este programa, Despegar ya contaba con una gran cantidad de componentes y microservicios interconectados entre sí, los cuales en conjunto permiten la reserva de los distintos tipos de productos anteriormente mencionados. Integrar los nuevos componentes y microservicios relacionados al programa a este stack tecnológico ya existente, mientras se buscaba garantizar una experiencia fluida y agradable para los usuarios, representó un gran desafío debido a que demandó esfuerzo de muchos equipos de IT, UX y Producto.

El objetivo principal al momento de implementar el programa era diseñar e implementar una solución tecnológica que permitiera la gestión eficiente del programa, asegurando su escalabilidad, seguridad y mantenimiento. Para ello, fue necesario:

- Desarrollar un ecosistema de aplicaciones que permitiera la acumulación y canje de puntos.

- Garantizar la seguridad y la consistencia de los datos, evitando fraudes o inconsistencias en el saldo de los puntos.
- Integrar dicho ecosistema dentro del conjunto de aplicaciones de Despegar, asegurando compatibilidad con los sistemas ya existentes.
- Implementar incentivos al usuario para que se registrara en Pasaporte Despegar, ya que inicialmente el usuario debía iniciar sesión y adherirse al programa de manera activa.
- Implementar herramientas de tracking para medir el rendimiento del programa, no solo en el lanzamiento sino también a largo plazo.
- Desarrollar los componentes visuales que permitan al usuario interactuar con el programa, como la visualización de puntos acumulados en cada compra y los descuentos aplicables al canjear puntos. Estos componentes debían integrarse en aplicaciones con distintas tecnologías de base.

En los próximos capítulos, se analizará cómo fue este proceso en Despegar, qué decisiones técnicas fueron necesarias y qué aprendizajes surgieron a lo largo del desarrollo.

1.3 Objetivos

El objetivo de este trabajo es compartir una experiencia profesional concreta en el desarrollo de software dentro de una empresa tecnológica de gran escala. Puntualmente, se va a detallar cómo fue el desarrollo e implementación de la primera versión y varias iteraciones siguientes del programa de fidelidad de Despegar, y cuál fue el rol del autor de este documento en las distintas etapas del proyecto.

En un primer momento, el desarrollo estuvo enfocado exclusivamente en la marca Despegar, pero con el tiempo se extendió a las distintas *white labels*⁴ que la empresa opera en la región. Esta evolución trajo consigo nuevos desafíos técnicos, organizativos y de coordinación entre equipos, que también se abordarán en el trabajo.

Además del recorrido del proyecto, se expondrán aspectos propios del proceso de desarrollo utilizado en la empresa, como la estrategia de despliegue progresivo de funcionalidades (features habilitadas por contexto), la medición del impacto mediante pruebas A/B (A/B testing) y el uso de trackeos⁵ para registrar el comportamiento de los usuarios.

También se abordarán conceptos clave relacionados con el diseño de arquitecturas basadas en microservicios, así como aspectos específicos del desarrollo de componentes front-end. En ese sentido, se presentarán comparaciones entre tecnologías (por ejemplo, Angular y

⁴ Producto o servicio desarrollado por una empresa (fabricante o proveedor) que otras empresas comercializan bajo su propia marca, como si fuera propio.

⁵ Registro y seguimiento del comportamiento de los usuarios dentro de una aplicación o sitio web

otras herramientas), y se describirán decisiones tomadas para facilitar la evolución del producto, como el diseño de componentes reutilizables y desacoplados.

Uno de los puntos destacados será cómo se logró reducir la dependencia de múltiples equipos en las iteraciones del front-end, logrando una solución más autónoma que permitió escalar y mantener el desarrollo de forma más eficiente.

Además de repasar los aspectos técnicos y organizativos, el trabajo busca reflexionar sobre los aprendizajes que surgieron a lo largo del proceso, y cómo esa experiencia aportó a la formación profesional de quien escribe.

1.4 Aporte del autor

Como se mencionó anteriormente, este proyecto tuvo impacto en múltiples equipos de distintas áreas. Parte de lo que se explicará en este documento incluye aspectos relacionados con la arquitectura de aplicaciones de Despegar y desarrollos asociados a esta funcionalidad en los que el autor no participó directamente, pero que aportan el contexto necesario para entender el alcance y los desafíos del proyecto.

Ahora bien, ¿cuál fue el aporte concreto en el desarrollo inicial y cuál fue el rol asumido en las iteraciones posteriores?

Al momento de trabajar en la primera versión del programa de fidelización, quien escribe ya formaba parte del equipo encargado del front-end del flujo de Alojamientos. En ese entonces, se desempeñaba como desarrollador junior, y actualmente ocupa el rol de Engineering Manager dentro del mismo equipo. El desarrollo inicial representó una gran oportunidad de crecimiento, ya que permitió asumir un rol protagónico desde el comienzo.

En una primera etapa, como cualquier equipo de front-end, fue necesario establecer acuerdos con los equipos responsables de los servicios consumidos. Estos, a su vez, realizaron la integración con el *core* de Loyalty (ecosistema de aplicaciones centrales del programa de fidelidad, que implementan su lógica principal y reglas de negocio). Es decir, el equipo del autor no trabajó directamente sobre el nuevo stack, sino que la información generada por el *core* de Loyalty llegó a través de integraciones ya existentes. La responsabilidad en esta instancia fue mapear el nuevo modelo de datos y aplicar las reglas específicas correspondientes a su capa. En esta parte del desarrollo, la participación fue parcial.

Adicionalmente, el equipo de quien escribe asumió una responsabilidad mayor en relación con otros equipos de front-end, ya que se decidió centralizar en él el desarrollo de los componentes visuales reutilizables que se usarían en todo el sitio, con el objetivo de optimizar tiempos de implementación. En esta etapa la participación fue activa y completa, incluyendo el desarrollo del componente visual principal del programa, denominado *loyalty-redemption-switch*, y también un back-end llamado *s-commons-vr* encargado de centralizar la lógica de negocio relacionada al front-end. Para avanzar con esto fue necesario abordar discusiones técnicas clave, como la elección de tecnologías para el componente front-end (teniendo en cuenta su integración en aplicaciones con bases

tecnológicas distintas), la definición técnica del back-end, y la integración de todos estos elementos en los diferentes flujos de producto del sitio.

Luego de esa primera versión, se continuó participando en varias iteraciones posteriores, entre las que se destacan:

- La adaptación del programa de puntos al negocio de White Labels.
- El desarrollo de nuevos componentes front-end asociados al programa.
- Mejoras técnicas sobre la solución existente, tanto a nivel visual como de arquitectura.

A lo largo de los próximos capítulos se profundizará en estos desarrollos y en cómo fueron abordados por el equipo de front-end de quien escribe en cada etapa.

Capítulo 2. Descripción funcional del sistema de fidelidad

2.1. Visión general del programa

Pasaporte Despegar es el programa de fidelidad de Despegar, que entró en vigencia en 2019. La idea es simple: los usuarios acumulan puntos por cada compra que hacen en la plataforma, ya sea de vuelos, alojamientos, paquetes, autos, actividades, etcétera. Es gratis, y cualquier persona con cuenta puede participar.

Los puntos no se acreditan en el momento, sino después de que el viaje se completa. Por ejemplo, si se compra un vuelo o una reserva de alojamiento, los puntos se suman una vez que se realiza efectivamente ese viaje. Esto ayuda a evitar fraudes o acumulaciones por servicios que después se cancelan.

Además de las compras en Despegar, en algunos países también se puede sumar puntos con tarjetas de crédito co-brandeadas que se lanzaron en conjunto con distintos bancos. Con estas tarjetas, cualquier compra suma puntos, incluso si no tiene nada que ver con viajes. Esto hace que los usuarios puedan avanzar más rápido en el programa.

Los puntos se pueden usar como parte de pago en futuras compras dentro de la misma plataforma. Se pueden aplicar para obtener un descuento sobre el precio total de un vuelo, un alojamiento o cualquier otro producto. También hay promociones exclusivas, pensadas especialmente para quienes forman parte del programa.

En resumen, es un programa de fidelidad que busca premiar a quienes eligen Despegar con frecuencia, ofreciéndoles beneficios concretos que impactan directamente en el precio de sus próximos viajes.

Desde el punto de vista del negocio, el programa persigue varios objetivos. El principal es fomentar la recurrencia de compra, es decir, que los usuarios vuelvan a utilizar la plataforma para futuras reservas. Esto se logra ofreciendo una recompensa concreta —los puntos— que pueden transformarse en descuentos reales en próximas compras. En un mercado con tanta competencia y comparación de precios como el del turismo online, este tipo de incentivos puede marcar la diferencia al momento de elegir una plataforma por sobre otra.

Otro de sus objetivos principales es fortalecer la relación con el usuario. La fidelización no solo tiene que ver con que alguien compre más de una vez, sino también con generar una conexión más estable con la marca. A través del programa, Despegar busca construir esa relación ofreciendo beneficios exclusivos y un trato diferencial para quienes usan la plataforma con más frecuencia.

También hay un objetivo más estratégico vinculado a la eficiencia: retener usuarios es más económico que adquirir nuevos. Entonces, si el programa logra que una persona vuelva a

comprar en Despegar sin tener que ser alcanzada por una campaña de marketing o una promoción puntual, hay un ahorro directo en costos de adquisición.

Desde el lado del usuario, el atractivo principal es el ahorro. Acumular puntos por cada compra y poder usarlos después para pagar parte de un viaje genera una sensación de beneficio tangible, ya que es un descuento directo, visible, que se acumula y se puede usar cuando el usuario quiera. Esto hace que muchas personas se interesen en seguir sumando puntos y prefieran concentrar sus compras en un solo lugar.

2.2. Funcionamiento desde la perspectiva del usuario

Desde la perspectiva del usuario, el programa está pensado para que su funcionamiento sea simple y fácil de entender.

Primero, ¿cómo se accede al programa? Durante los primeros meses de vida de Pasaporte Despegar, era necesario que los usuarios, además de iniciar sesión en Despegar, completaran un formulario específico para adherirse al programa. Si bien el proceso no era complejo, esta condición agregaba un paso adicional que, en la práctica, generaba fricción. Aunque durante el flujo de compra se mostraban incentivos visuales y mensajes promocionando el programa, muchos usuarios no llegaban a completar ese paso, lo que impactaba negativamente en la cantidad de miembros activos y, por lo tanto, en el alcance real del programa.

Con el tiempo, y a partir del análisis de estas métricas, se entendió que este requerimiento inicial era innecesario. Como respuesta, se decidió modificar el comportamiento del sistema para que cualquier usuario que haya iniciado sesión quede automáticamente adherido al programa, sin tener que hacer ninguna acción extra. Esta lógica —que sigue vigente hasta el día de hoy— no solo simplificó la experiencia del usuario, sino que también mejoró significativamente las tasas de adopción del programa.

El programa tiene tres categorías de usuarios. Al momento de adherirse, cada usuario comienza en la categoría **Pasaporte Viajero**, que es la base del programa. Actualmente (abril de 2025), esta categoría ofrece, además de la posibilidad de acumular y canjear puntos, los siguientes beneficios:

- 10% de descuento en alojamientos seleccionados.
- 15% de descuento en actividades.
- Acumulación de hasta 10 puntos por cada 10 dólares gastados en la plataforma.

A medida que el usuario realiza compras, puede avanzar de categoría. La siguiente es **Pasaporte Explorador**, que se alcanza al superar los 1.000 dólares gastados en Despegar. Los beneficios de esta categoría, al día de hoy, son:

- 15% de descuento en alojamientos seleccionados.

- 15% de descuento en actividades.
- Atención telefónica personalizada en el call center de Despegar.
- Acumulación de hasta 20 puntos por cada 10 dólares gastados.

Finalmente, la categoría más alta es **Pasaporte Global**, a la cual se accede luego de superar los 5.000 dólares en compras acumuladas. En este caso, los beneficios (vigentes en abril de 2025) son:

- 20% de descuento en alojamientos seleccionados.
- 15% de descuento en actividades.
- Línea telefónica exclusiva en el call center.
- Mejora de categoría de habitación en alojamientos seleccionados.
- Acumulación de hasta 30 puntos por cada 10 dólares gastados.

Los beneficios crecen de forma progresiva a medida que se avanza de categoría, lo cual premia especialmente a los usuarios que más compran (o realizan compras de mayor valor). Esta estructura apunta a brindar una experiencia diferenciada para los clientes más valiosos para la empresa, incentivando la fidelidad y el uso continuo de la plataforma.

Ahora bien, ¿cómo es la experiencia de usuario relacionada a este programa, durante el flujo de venta? Para responder a esa pregunta, primero es necesario explicar el sitio web de Despegar.

2.3 Despegar: contexto y navegación

Para entender cómo y dónde se insertaron los distintos componentes del programa de fidelidad dentro del flujo de venta, es importante primero tener una visión general del negocio de Despegar y de cómo está estructurado su sitio web. En esta sección se va a describir, por un lado, el modelo general de la empresa y los productos que comercializa, y por otro lado, las principales pantallas que componen el recorrido típico de un usuario al momento de realizar una compra, tomando de ejemplo el flujo de Alojamientos. Esta información va a servir como base para contextualizar las decisiones de diseño e implementación que se tomaron en las distintas etapas del proyecto.

Despegar fue registrada como marca en agosto de 1999. Su fundador, Roberto Souviron, identificó una oportunidad de negocio al notar que en América del Norte estaban empezando a surgir plataformas online que facilitaban la reserva de vuelos, mientras que en América Latina no existía todavía una propuesta similar. A partir de esa observación, nació Despegar, con el objetivo inicial de digitalizar el proceso de reserva de vuelos en la región.

Con el paso del tiempo, la empresa fue sumando nuevos productos a su oferta, como alojamientos, actividades, seguros, alquiler de autos, entre otros. Esta diversificación se

convirtió en una parte central de su propuesta de valor: permitir que los usuarios puedan organizar todos los aspectos de su viaje desde una única plataforma. Si bien Despegar también cuenta con otros canales de venta —como call centers y algunas agencias físicas— su canal principal siempre fue el sitio web, tanto por volumen de transacciones como por estrategia de negocio. Actualmente, la empresa tiene presencia en más de 20 países de América, incluyendo toda América Latina y el mercado hispano de Estados Unidos.

En cuanto al sitio web, el *funnel*⁶ de venta típico comienza en la **home**, a la que los usuarios acceden directamente mediante el dominio o a través de motores de búsqueda como Google o Bing. En esta pantalla se destacan distintas promociones, campañas activas y, sobre todo, se permite seleccionar un producto puntual para comenzar una reserva (por ejemplo, vuelos, alojamientos, paquetes o actividades). A partir de ahí, se inicia el recorrido dentro del flujo de compra.

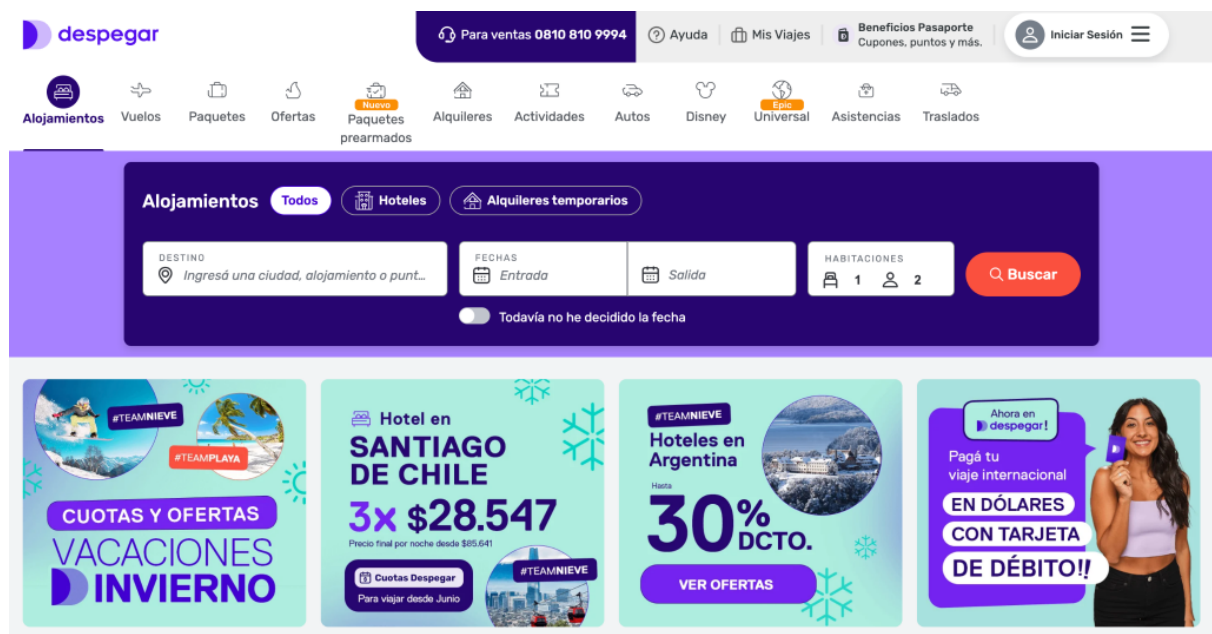


Figura 2.1. Captura de la página principal del sitio web (*home*).

Como se puede ver en la Figura 2.1, el usuario tiene acceso directo a los distintos productos que ofrece la plataforma:

- **Alojamientos**
- **Vuelos**
- **Paquetes** (combinación de Vuelos + Alojamientos u otros productos)
- **Alquileres Vacacionales** (similar a Alojamientos, pero orientado a propiedades completas como casas, cabañas, departamentos, etcétera; diferente a hoteles)

⁶ Modelo que representa las etapas que atraviesa un usuario desde el primer contacto hasta completar una acción (como una compra o registro).

tradicionales)

- **Actividades**
- **Autos**
- **Asistencias** (seguros de viaje)
- **Traslados**

Además de estos productos principales, en la imagen también se pueden observar accesos destacados como **Ofertas**, **Paquetes prearmados** o **Disney**. Si bien no se trata de productos distintos en términos técnicos, estos accesos se presentan de forma diferenciada en la interfaz porque representan focos estratégicos para la compañía. En general, estos accesos redirigen al usuario a flujos ya existentes, pero con filtros o configuraciones predefinidas.

Una vez que el usuario selecciona un producto y completa los datos iniciales de búsqueda (fechas, destino, cantidad de pasajeros, entre otros), se accede al **flujo de resultados** o **search** (ambos términos se utilizan internamente para referirse a esta etapa del *funnel*), donde se listan las opciones disponibles de acuerdo con los criterios ingresados.

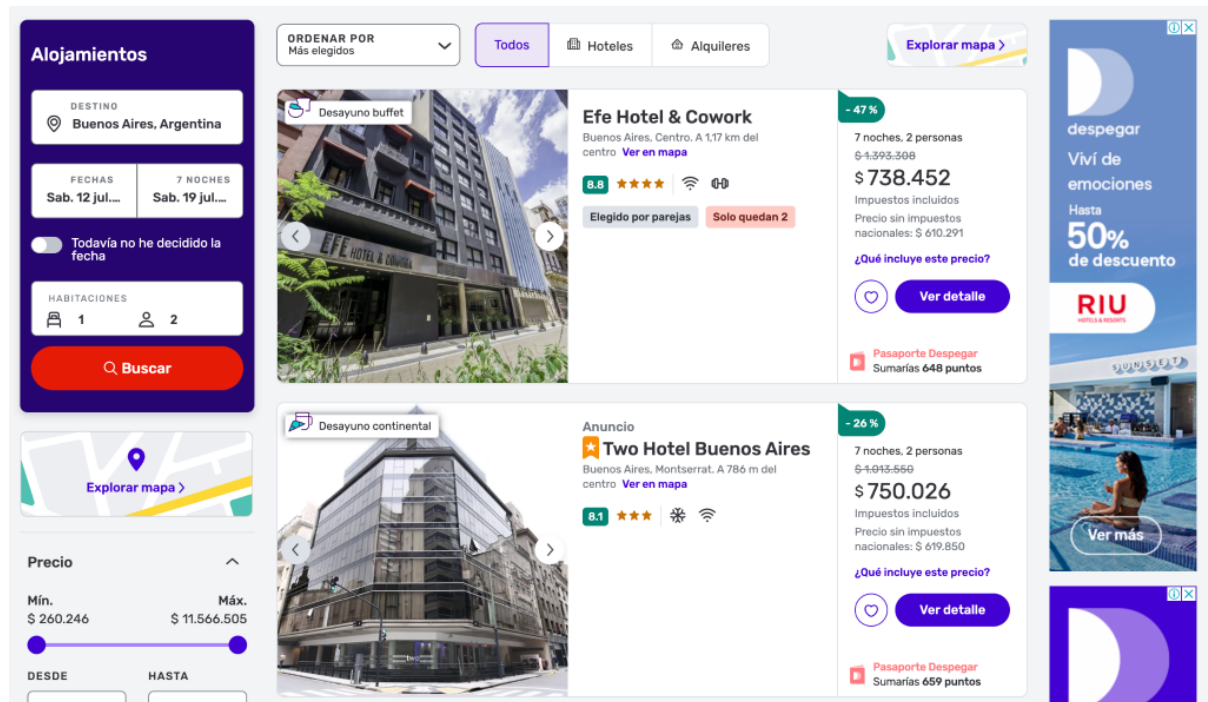


Figura 2.2. Captura de la página de resultados de Alojamiento.

En esta parte del *funnel*, ejemplificada en la Figura 2.2 con el caso de Alojamiento, el usuario accede a un listado de opciones disponibles según el destino, las fechas y la cantidad de pasajeros ingresados en la búsqueda. Esta pantalla permite comparar alternativas, aplicar filtros y ordenar los resultados según distintas preferencias (precio, puntuación, ubicación, etcétera).

Luego del paso de **search**, algunos productos ofrecen una página de **detail**, donde se amplía la información sobre una opción puntual seleccionada por el usuario.

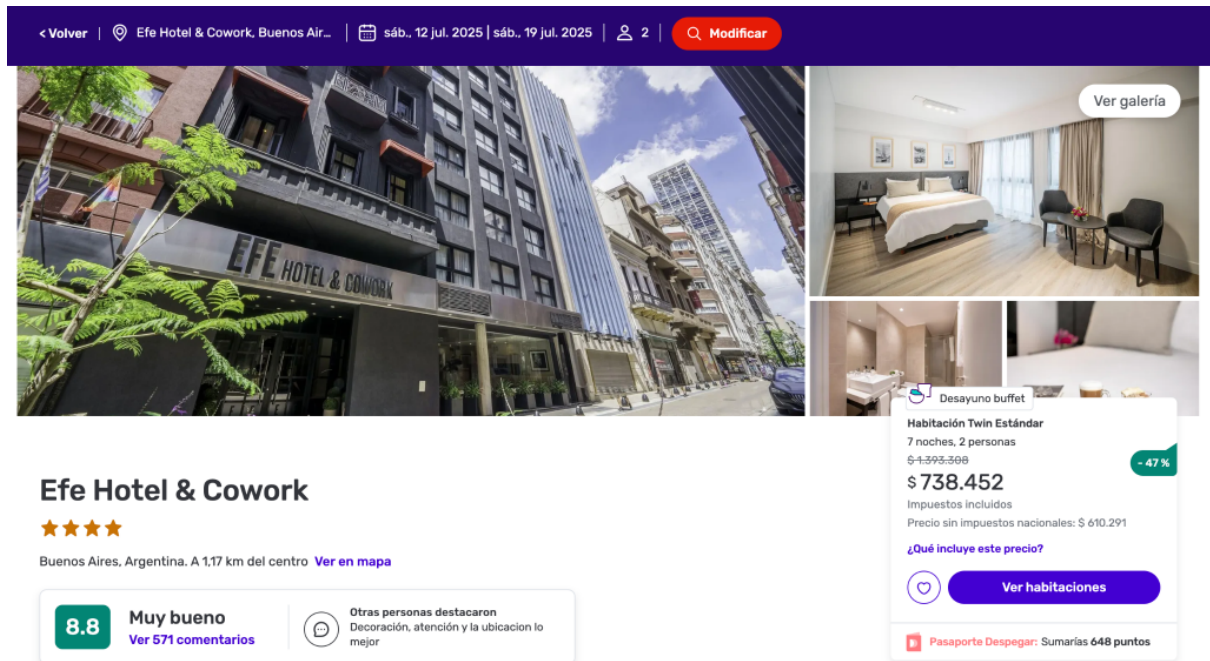


Figura 2.3. Captura de pantalla de la página de detalle de un alojamiento.

La página de **detalle**, mostrada en la Figura 2.3, tiene como objetivo brindar información más completa sobre el ítem seleccionado, para que el usuario pueda tomar una decisión informada y elegir la opción que mejor se adapte a sus necesidades y presupuesto.

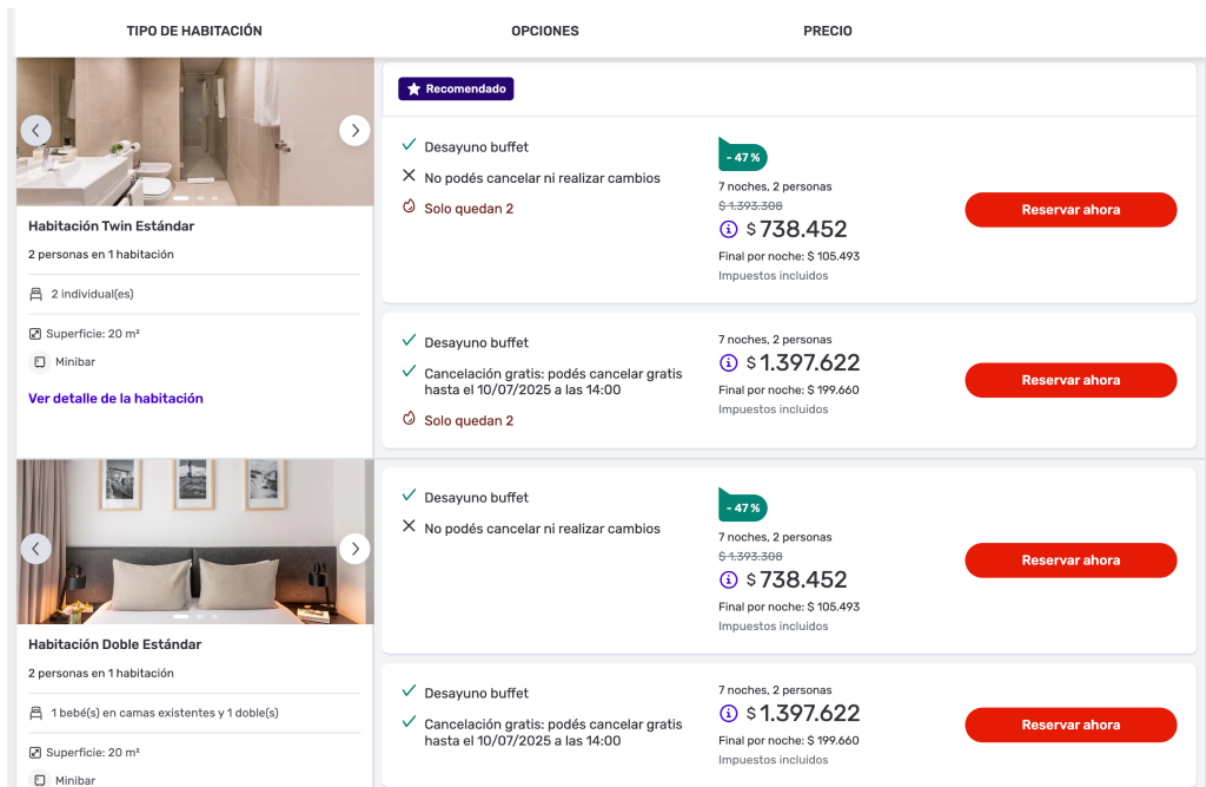


Figura 2.4. Captura del módulo de selección de habitación.

En la Figura 2.4 —correspondiente a un alojamiento— se puede ver cómo el usuario puede elegir entre distintas alternativas para su reserva: opciones con o sin cancelación, diferentes tipos de habitación, y otros elementos variables según el caso. Si bien no todas las opciones se reflejan en la captura, es común que se ofrezcan variantes en cuanto a planes de comida (por ejemplo, sin comidas, solo desayuno, media pensión o todo incluido), entre otras diferencias que impactan en el precio y en la experiencia general del viaje.

Luego de la página de detalle, en algunos casos —dependiendo del destino y el producto— se presenta la llamada **Página Intermedia**. Esta instancia aparece antes del **checkout** y tiene como objetivo ofrecer al usuario productos adicionales que podrían complementar su viaje. Por ejemplo, si se está reservando un alojamiento, es común que en esta etapa se sugieran alquileres de autos, actividades o asistencias.

La lógica detrás de esta pantalla está alineada con la estrategia de negocio de Despegar: al tratarse de una empresa que comercializa múltiples productos, resulta conveniente tanto para el usuario como para la compañía que se realicen compras combinadas. Desde el lado del usuario, se le ofrece un paquete más completo, con precios competitivos. Desde el lado de la empresa, se prioriza una venta mayor, incluso si eso implica reducir el margen de ganancia individual por producto, ya que el valor total de la operación aumenta. Esta lógica también apunta a evitar que el usuario reserve algunos servicios en Despegar y otros en plataformas de la competencia.

Después de esta etapa, se accede a la **página de checkout**, donde el usuario ingresa sus datos personales, la información de pago y puede revisar que los detalles de la reserva sean correctos antes de finalizar la compra. La Figura 2.5 muestra un ejemplo de esta etapa.

[« Volver a la página anterior](#)

¡Ya casi estás! Completá tus datos y finalizá tu compra

¡Felicitaciones! Elegiste la habitación más económica del Efe Hotel & Cowork. ¡No te la pierdas! Confirmá ahora tu reserva

¿Quién es titular de la reserva?

Titular habitación

Será responsable de hacer el check-in y el check-out en el alojamiento

NOMBRES

APELLIDOS

TIPO DE DOCUMENTO

NÚMERO



Cubrite ante imprevistos y gastos extras

Mejor precio ahora

Comprá ahora tu asistencia y viajá con tranquilidad.

Detalle del pago

Alojamiento para 2 personas	\$ 1.265.145
Impuestos, tasas y cargos	\$ 128.161
Descuento	-\$ 654.853
TOTAL	\$ 738.453

Cargos adicionales

Tarifas del destino	\$ 23.897,74
Impuesto Municipal	\$ 23.897,74

Cuando llegues a destino, el alojamiento te cobrará estos cargos adicionales por el total de la estadía.

Detalle de la compra



Efe Hotel & Cowork

★★★★

Paraguay 474

[Ver mapa](#)

Figura 2.5. Captura de la página de checkout.

Luego de **checkout**, si el usuario completa correctamente la reserva, llega a la **página de thanks**. Este es el último paso del *funnel*, donde se le agradece por su compra y se le muestra un resumen de la operación. Además, se le vuelven a ofrecer otros productos que podrían ser útiles para el mismo viaje. Esta pantalla cumple una doble función: cerrar la experiencia de compra de forma clara y brindar una última oportunidad dentro de la misma sesión para complementar la reserva antes de que el usuario abandone el sitio.

Con todo lo descrito en esta sección, se puede entender de forma general cómo está estructurado el sitio web de Despegar y cuál es el recorrido típico que realiza un usuario al momento de planificar y concretar una compra. Desde la *home* hasta la página de *thanks*, cada pantalla cumple un rol específico dentro del *funnel* y responde a objetivos tanto de experiencia de usuario como de negocio.

Habiendo detallado este flujo, en la siguiente sección se explicará cómo se integraron los distintos componentes del programa de fidelidad dentro de estas etapas, y qué desafíos surgieron en su diseño e implementación.

2.4 Aspectos generales de la implementación del programa

El desarrollo de Pasaporte Despegar presentó varios desafíos importantes. **Fue necesario integrar nuevos componentes** en una infraestructura que ya contaba con una gran cantidad de microservicios, sin afectar la experiencia de usuario. Además, la solución tecnológica tenía que ser escalable, segura, fácil de mantener y capaz de adaptarse a futuros cambios.

Para lograrlo, se trabajó en varios frentes al mismo tiempo:

- **Crear un ecosistema de aplicaciones** que permitiera acumular y canjear puntos de manera sencilla.
- **Integrar el ecosistema nuevo con las aplicaciones existentes** de Despegar, asegurando la compatibilidad entre sistemas.
- **Fomentar la adhesión de usuarios** a Pasaporte Despegar, ya que era necesario que iniciaran sesión y se registraran de forma activa.
- **Implementar herramientas de tracking** para medir el rendimiento del programa tanto al momento del lanzamiento como a largo plazo.
- **Desarrollar componentes visuales** que mostraran la cantidad de puntos acumulados en las compras y los descuentos disponibles al canjearlos. Estos componentes debían funcionar en aplicaciones implementadas con tecnologías distintas.

El tiempo disponible para lanzar la primera versión era de seis meses, y se eligió Brasil como primer mercado, por ser uno de los más grandes y estratégicos de la región.

Un punto clave del proyecto fue que el programa de fidelidad debía habilitarse de forma segmentada. Despegar opera en toda Latinoamérica, y también ofrece servicios de plataforma a otras marcas a través de su modelo de *white labels*.

Como en esa etapa el programa era exclusivo de Despegar, fue necesario desarrollar una solución que permitiera activarlo solo en ciertos sitios y países. Incluso dentro de Despegar, la implementación dependía de acuerdos bancarios y validaciones legales en cada mercado. Por eso, desde IT se buscó que la habilitación en nuevos países fuera lo más simple posible, idealmente requiriendo solo integrar sistemas bancarios locales y adaptar las comunicaciones, sin grandes cambios en la plataforma.

La implementación del programa también llevó a hacer algunos cambios en los equipos:

- Se armó un equipo de **UX** dedicado a diseñar las experiencias asociadas al programa.
- Se creó un equipo de **IT** encargado de desarrollar y mantener el nuevo ecosistema de aplicaciones, al que se suele referir como **el core de Loyalty**.
- Algunos equipos ya existentes, como el de **front-end de Alojamiento** (del que quien escribe formó y sigue formando parte), sumaron nuevas responsabilidades. En este caso, la responsabilidad asignada fue desarrollar y posteriormente mantener los componentes visuales que se usarían en todo el sitio de Despegar.

Adicionalmente, es importante mencionar que muchos otros equipos de IT, UX y Producto participaron del desarrollo. Por ejemplo, todos los equipos de IT de los distintos productos debieron integrarse con el core de Loyalty, ya sea de forma directa o indirectamente, mapeando la información que llegaba a través de otros servicios ya integrados con el *core*. A su vez, todos los equipos de front-end tuvieron que integrar los componentes

desarrollados por el equipo de Alojamientos, tanto en la primera iteración como en las versiones posteriores.

En resumen, la implementación del programa de fidelidad fue un esfuerzo que involucró a múltiples equipos, requirió adaptar sistemas existentes y construir una base tecnológica flexible para poder seguir evolucionando en el futuro.

En el próximo capítulo se profundizará en las tecnologías, metodologías de trabajo y prácticas que formaban parte del ecosistema de Despegar al momento del desarrollo, ya que fueron clave para hacer posible el lanzamiento y la evolución del programa.

Capítulo 3. Estado del arte: tecnologías y metodología en Despegar

En este capítulo se presentan los aspectos técnicos y metodológicos que caracterizan al desarrollo de software en Despegar. A lo largo de los años, la empresa ha consolidado un ecosistema tecnológico robusto, compuesto por una arquitectura de microservicios, un stack moderno y prácticas ágiles que permiten escalar sus productos y servicios.

Se describen las tecnologías predominantes tanto en el back-end como en el front-end, acompañadas de fundamentos que justifican su adopción y continuidad. También se detallan los procesos de desarrollo y prueba que acompañan la evolución del sistema, así como mecanismos clave como la normalización de datos, el uso de snapshots en la arquitectura, y la estrategia de experimentación basada en A/B testing, fundamental para validar hipótesis de negocio.

3.1. Arquitectura de microservicios

La arquitectura detrás del sitio web de Despegar está compuesta por un gran número de componentes con distintos niveles de responsabilidad.

Para dar una visión general, es clave tener presente una característica fundamental del negocio: Despegar no cuenta con aviones, alojamientos ni actividades propias. Su propuesta de valor radica en conectar a potenciales compradores con proveedores que sí ofrecen esos servicios. Bajo esta premisa, se entiende fácilmente por qué la arquitectura de microservicios de Despegar está, en muchos puntos, integrada con sistemas externos.

De manera general, puede decirse que cada producto (vuelos, alojamientos, actividades, etc.) cuenta con su propio stack de aplicaciones, en el que cada componente asume una responsabilidad bien definida. Por ejemplo, el front-end se encarga de mostrar las pantallas de cara al usuario; otras aplicaciones administran la información estática del producto (como las fichas de cada alojamiento); también existen servicios encargados de aplicar reglas de negocio, reglas de pricing, y otras reglas específicas por tipo de mercado; y, por último, componentes responsables de establecer la conexión con los proveedores externos.

A grandes rasgos, todos los productos de Despegar tienen una arquitectura de microservicios similar a la mostrada en la Figura 3.1:

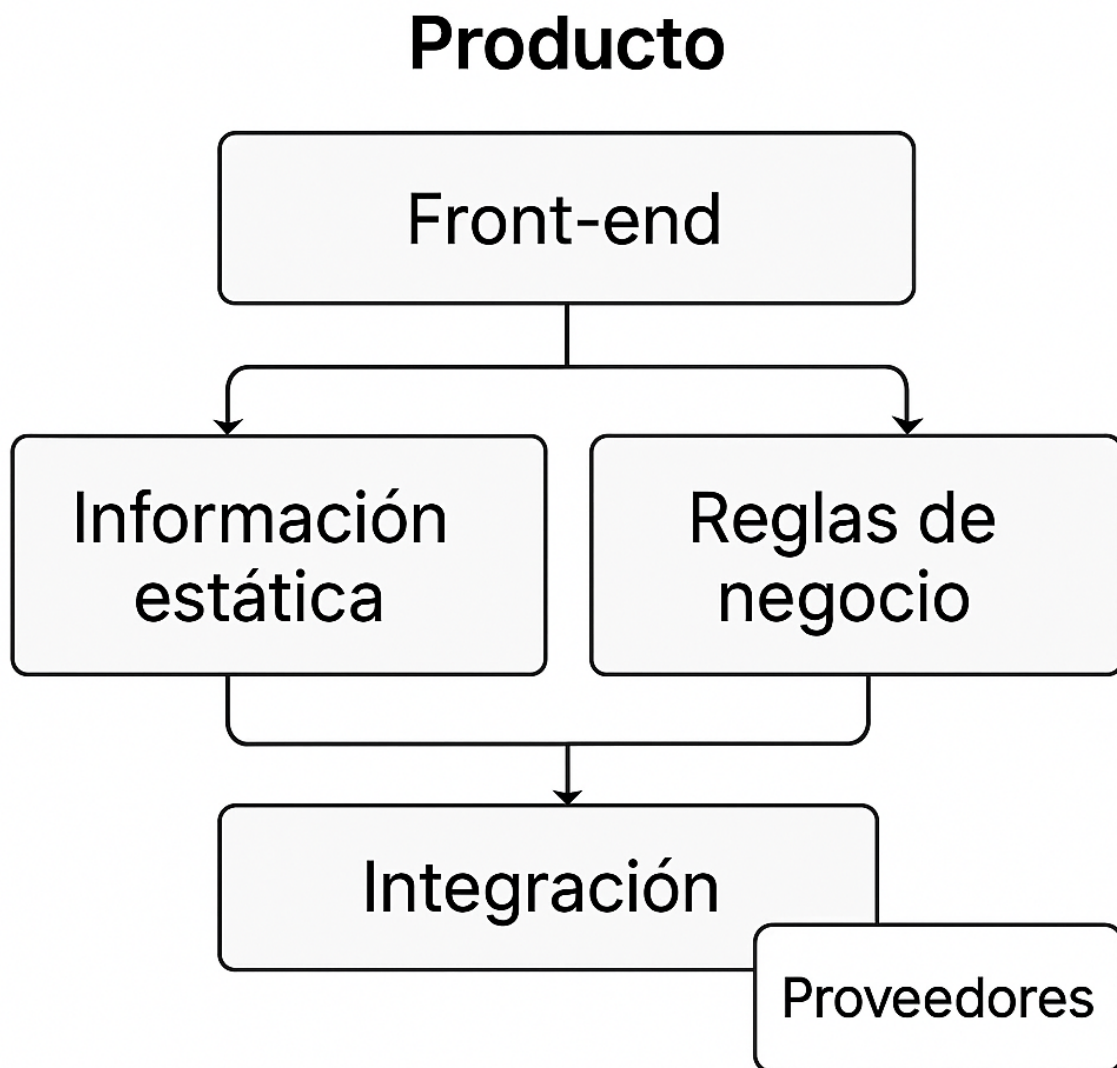


Figura 3.1. Arquitectura general de microservicios de un producto en Despegar.

Es importante mencionar también que cada ítem mencionado en el gráfico —como “Información estática”, “Reglas de negocio” o “Integración”— puede estar compuesto por varias aplicaciones con propósito finito.

Tener una arquitectura basada en microservicios es clave para poder trabajar de forma más ordenada en sistemas grandes como el de Despegar. Este tipo de arquitectura permite dividir la aplicación en partes más pequeñas e independientes, que pueden desarrollarse, probarse y desplegarse por separado. Esto ayuda a que los equipos puedan trabajar en paralelo sin pisarse, que los errores queden aislados en un solo lugar, y que cada parte del sistema pueda escalar según su necesidad. Además, si un producto necesita incorporar nuevas tecnologías o probar cambios sin afectar al resto, esta arquitectura lo permite con mucha más flexibilidad. Estas ideas están ampliamente desarrolladas por autores como Newman (2015) y Richards & Ford (2020), que destacan cómo los microservicios ayudan a construir sistemas más mantenibles, escalables y preparados para cambios constantes.

A partir de esta arquitectura surgen algunos desafíos que deben resolverse para garantizar una operación eficiente y coherente, especialmente cuando se trata de integrar múltiples fuentes de datos externas. En este contexto, cobran especial relevancia prácticas como la normalización de modelos y el uso de snapshots.

Respecto a la **normalización de modelos**, lo importante a destacar es que cada producto vendido en Despegar puede requerir la integración con múltiples proveedores externos, los cuales naturalmente manejan sus propios modelos de datos. De cara a los microservicios internos de Despegar, sería completamente inviable operar con tantos formatos distintos, y es por eso que cada producto cuenta con una aplicación encargada de transformar esa información externa a un modelo interno unificado. Esta normalización permite que el resto del ecosistema de aplicaciones del producto trabaje de forma homogénea y desacoplada del proveedor específico.

Además, si bien es esperable que los modelos varíen entre productos (por ejemplo, entre vuelos y alojamientos), también existen ciertas estructuras comunes que se encuentran normalizadas de forma transversal. Esto facilita la reutilización de componentes, reduce la duplicación de lógica y permite mantener una mayor coherencia dentro del sitio.

En cuanto al consumo de servicios y la **utilización de snapshots**, es importante entender que no toda la información presenta la misma volatilidad a lo largo del tiempo. Existen datos que tienden a cambiar con frecuencia —como la disponibilidad de habitaciones en un hotel—, mientras que otros se mantienen relativamente estables —como la ficha de un alojamiento, que incluye el nombre, la descripción, las imágenes, entre otros atributos—. En estos últimos casos, donde la información es menos propensa a cambios, resulta conveniente evitar el consumo en vivo y considerar, en su lugar, el uso de *snapshots*.

Pero, ¿cuál es la ventaja de utilizar *snapshots*? Su uso permite desacoplar el sistema de los proveedores externos, mejorando la resiliencia y reduciendo la latencia de las operaciones. Al trabajar con copias locales de datos que no requieren actualización constante, se disminuye la cantidad de llamadas en tiempo real a servicios externos, lo que mejora la performance del sistema y reduce el riesgo de fallos derivados de la indisponibilidad o lentitud del proveedor.

En el contexto de arquitecturas distribuidas, esta estrategia se relaciona con patrones de diseño orientados a la tolerancia a fallos y al principio de *eventual consistency*, ampliamente desarrollados por Newman (2015) y tratados en términos de diseño desacoplado en entornos distribuidos por Fowler (2002). Por otra parte, el uso de *snapshots* —término que se utiliza en este trabajo para referirse a mecanismos de *background synchronization*— resulta adecuado en escenarios donde se prioriza la escalabilidad horizontal y se busca minimizar la carga operativa de servicios dependientes, según lo desarrollado por Ford et al. (2021).

En definitiva, la adopción de una arquitectura de microservicios le permitió a Despegar construir un ecosistema flexible, escalable y alineado con la complejidad de su operación. Esta estructura no solo facilita el desarrollo y mantenimiento de los distintos productos, sino que también permite incorporar nuevas funcionalidades, integrar tecnologías de manera progresiva y sostener una alta disponibilidad incluso ante volúmenes de tráfico

significativos. A partir de este diseño arquitectónico, se consolidó un stack tecnológico capaz de acompañar ese crecimiento, tema que se desarrolla en la siguiente sección.

3.2. Stack tecnológico predominante

Adentrándonos en las tecnologías utilizadas comúnmente en Despegar, podemos enfocarnos en dos áreas principales: las aplicaciones back-end y las aplicaciones front-end. Si bien también se emplean tecnologías asociadas a infraestructura, observabilidad o nube, estas suelen estar gestionadas por equipos especializados. Por lo tanto, en esta sección se presentarán las herramientas más relevantes para comprender el contexto tecnológico en el que se desarrolló el programa de fidelidad.

En el back-end, los lenguajes más utilizados son Java y Scala, con una presencia menor de Node.js. Los fundamentos para elegir Node.js en ciertas aplicaciones se detallarán más adelante, al abordar la implementación específica del programa. En cuanto a Java y Scala, su elección se basa en diversas razones técnicas y organizativas.

Java es ampliamente adoptado en entornos empresariales debido a su robustez, escalabilidad y rendimiento. Su máquina virtual (JVM) permite ejecutar aplicaciones en múltiples plataformas, facilitando el mantenimiento y la portabilidad del código. Además, cuenta con un ecosistema maduro de librerías y frameworks que agilizan el desarrollo de aplicaciones complejas. Según la documentación de Oracle, Java está diseñado para permitir a los desarrolladores “escribir una vez, ejecutar en cualquier lugar”, favoreciendo la creación de software confiable, distribuido y portable (Oracle, s. f.).

Scala, por su parte, combina paradigmas de programación funcional y orientada a objetos, ofreciendo una sintaxis concisa y expresiva. Esta característica permite escribir código más limpio y mantenible, lo cual es especialmente útil en sistemas distribuidos y de alta concurrencia. Además, al ejecutarse sobre la JVM, Scala es compatible con las librerías de Java, lo que facilita su integración en proyectos existentes (Scala Documentation, s. f.).

En el front-end, las tecnologías predominantes son Angular y React. La mayoría de las aplicaciones relacionadas con el flujo de venta están desarrolladas en Angular, mientras que las del flujo de postventa utilizan React. Esta división no responde a diferencias técnicas inherentes a los productos, sino a acuerdos entre equipos que comparten componentes y buscan facilitar la rotación de desarrolladores y la reutilización de código.

Angular es un framework completo que proporciona una estructura sólida para el desarrollo de aplicaciones web. Su uso de TypeScript mejora la escalabilidad y la seguridad del código, y su conjunto de herramientas integradas facilita la creación y mantenimiento de aplicaciones empresariales complejas. Según la documentación oficial, el framework está diseñado para escalar desde proyectos de un solo desarrollador hasta aplicaciones de nivel empresarial (Angular, s.f.).

React, en cambio, es una librería enfocada en la construcción de interfaces de usuario mediante componentes reutilizables. Su flexibilidad y rendimiento la hacen adecuada para aplicaciones que requieren interfaces dinámicas y una experiencia de usuario fluida. Además, su amplia comunidad y ecosistema de herramientas contribuyen a su adopción en

proyectos de diversa índole. La documentación oficial destaca la facilidad de integración de React en proyectos existentes, permitiendo renderizar componentes interactivos en cualquier lugar (React, s.f.).

En resumen, la elección de estas tecnologías en Despegar responde a criterios de escalabilidad, mantenibilidad y eficiencia, alineados con las necesidades específicas de cada equipo, producto y sector.

3.3. Componentes compartidos en el Front-End

Así como el back-end de Despegar está compuesto por una gran cantidad de microservicios, el front-end sigue una filosofía similar. En primer lugar, es importante destacar que el sitio no está compuesto por una única aplicación, sino por múltiples aplicaciones front-end independientes. Por ejemplo, la aplicación que desarrolla el equipo de quien escribe es responsable de los pasos del *funnel* de resultados y detalle de Alojamientos. La *home* (paso previo a resultados) y *Página Intermedia* (paso siguiente al detalle) también son aplicaciones distintas. Esta división de responsabilidades no es exclusiva de Alojamientos, sino que se replica en los demás productos ofrecidos en el sitio.

Ahora bien, más allá de esta fragmentación técnica en aplicaciones separadas, existe un objetivo compartido: brindar una experiencia de usuario coherente entre productos. Aunque cada uno tiene sus particularidades de negocio —lo que justifica ciertas diferencias visuales—, también es fundamental que un usuario que ya se familiarizó con un producto pueda navegar por otros y encontrar patrones reconocibles. Lograr esto requiere coordinación entre los distintos equipos de UX y de IT. Para eso, existen lineamientos transversales que todos los equipos deben seguir, tanto en cuestiones visuales como en otros aspectos clave de la experiencia.

En este contexto surge una solución técnica concreta: compartir componentes visuales. Este enfoque trae múltiples beneficios. Mejora la coherencia visual, reduce los tiempos de desarrollo, disminuye errores y permite que los equipos se enfoquen en la lógica específica de su negocio sin tener que reinventar soluciones ya resueltas. Además, simplifica el mantenimiento y favorece la escalabilidad, algo especialmente valioso en organizaciones donde muchos equipos trabajan sobre una misma plataforma.

Ahora bien, ¿cómo se define qué componente se comparte, cómo se lo implementa y quién se hace cargo? Existen tres enfoques principales. Por un lado, hay un equipo específico —*frontend-components*— que se encarga de desarrollar ciertos componentes compartidos. Esta modalidad permite tener un equipo claramente responsable, lo que facilita los ciclos de cambio. Estos componentes suelen tener repositorios propios, y se integran en las aplicaciones de producto a través de servicios que los exponen. Esto tiene una gran ventaja: como los componentes están desacoplados del código principal de producto, los cambios no dependen de otros equipos y su impacto en producción es inmediato una vez que se sube la nueva versión del servicio.

Por otro lado, existe una segunda modalidad, más colaborativa: un grupo de desarrolladores con representantes de distintos equipos front-end que busca detectar oportunidades para compartir soluciones. Los componentes que surgen de este grupo se escriben en un

repositorio común y se distribuyen como librerías que luego cada equipo consume como dependencia. Si bien esta opción puede ser menos ágil (ya que no hay un equipo dedicado que gestione cambios, y cada actualización requiere un nuevo deploy), también es una estrategia efectiva para reducir costos de desarrollo y asegurar consistencia visual.

Por último, y más relevante para esta tesina, hay casos donde ciertos componentes compartidos son directamente asignados a un equipo específico, tanto para su desarrollo inicial como para su evolución. Esta asignación no siempre tiene una justificación rígida, pero responde a una dinámica que, en la práctica, funciona. En particular, los componentes asociados al programa de fidelidad de Despegar fueron asignados al equipo de quien escribe: el equipo front-end de Alojamiento. ¿Por qué? En principio, se entendía —y la experiencia lo confirmó— que estos componentes no tienen una frecuencia alta de cambios, por lo que no justificaba sumarlos al ya exigido scope del equipo *frontend-components*. Tampoco se alineaban con el espíritu del grupo colaborativo de desarrolladores, que no busca asumir responsabilidades fijas, sino detectar y compartir buenas prácticas.

Esto habilita una tercera vía: delegar la responsabilidad a un equipo con foco en un determinado producto que pueda asumirla como parte de su alcance. En Despegar es común ver equipos front-end con un foco principal bien definido que, a su vez, gestionan otras responsabilidades más satelitales. En el caso del equipo a cargo del front-end de Alojamiento, además de su responsabilidad principal, también se ocupa del stack de *Reviews & NPS*⁷ y de los componentes —y su back-end asociado— del programa de puntos. De este modo, si un equipo de negocio desea realizar cambios en los componentes de puntos, debe coordinar directamente con dicho equipo, que luego se encarga de llevar la solución tanto a Alojamiento como al resto del sitio. Esta responsabilidad adicional, lejos de representar una carga, suele ser vista como una oportunidad: permite a los desarrolladores involucrarse en historias de impacto transversal, con todo lo que eso implica en términos de análisis, debate técnico y coordinación entre áreas.

En la siguiente sección se detalla la metodología de desarrollo, lo cual permitirá entender mejor cómo se organizan estos tres enfoques de asignación de componentes front-end compartidos.

3.4. Metodología de desarrollo

En Despegar, el proceso de desarrollo de software sigue una lógica secuencial inspirada en el modelo conocido como Waterfall, presentado originalmente por Winston W. Royce (1970) como un enfoque lineal en el que cada etapa del ciclo —análisis, diseño, implementación, pruebas y mantenimiento— comienza una vez finalizada la anterior. Aunque el modelo clásico propone una separación estricta entre fases, en la práctica se aplican ciertas flexibilidades que permiten avanzar con tareas de diseño o desarrollo antes de tener todas las definiciones cerradas.

⁷ Abarca los formularios para que los usuarios califiquen productos comprados (alojamiento, vuelos, autos, etc.), así como los servicios que procesan y exponen puntuaciones y comentarios en el sitio.

A nivel organizacional, la planificación de proyectos se realiza por trimestres (Q). Cada equipo de Producto —en rol de sponsor— presenta propuestas respaldadas por un *business case*, que justifica el valor de negocio de cada iniciativa. A partir de estas propuestas, se define la prioridad de las tareas, y los equipos técnicos se encargan de estimar su complejidad y esfuerzo, comprometiéndose el trabajo de acuerdo con su capacidad disponible para ese trimestre.

Una vez priorizadas las tareas, comienza el trabajo de definición, diseño y desarrollo. Si bien el enfoque general es secuencial —se procura tener los requerimientos lo más definidos posible antes de comenzar la implementación—, no es inusual que las etapas de diseño y desarrollo se inicien sin contar con todas las definiciones completamente cerradas. Esto se debe a que IT y UX están involucrados desde el inicio, lo que permite analizar en conjunto con los equipos de negocio la viabilidad de cada iniciativa, priorizando aquellas con mejor relación costo-beneficio. Esta colaboración temprana también permite resolver dudas funcionales de forma ágil y ajustar el alcance en función del impacto esperado.

Durante las primeras fases del ciclo, se llevan a cabo reuniones con stakeholders de las áreas involucradas, con el objetivo de alinear expectativas, entender dependencias técnicas o de negocio, y planificar el proyecto de forma integral. Si bien en algunos casos se trata de iniciativas de gran alcance que requieren la participación de múltiples equipos de IT, con foco en distintos productos, en muchos otros las tareas están acotadas a un solo producto y un solo equipo de IT, lo que permite un proceso de coordinación más simple y ágil.

Otro aspecto distintivo del proceso de desarrollo en Despegar es la ausencia de un equipo de QA dedicado. El testing es responsabilidad de los propios desarrolladores, quienes deben validar manualmente cada feature antes de su entrega. Esto refuerza la importancia de una definición funcional clara, una buena comunicación con Producto y UX, y la existencia de prácticas internas que aseguren la calidad del software.

Este modelo ha demostrado ser eficaz en un contexto donde los desarrollos suelen impactar a múltiples sistemas y equipos. La combinación de planificación estructurada, involucramiento temprano de todas las áreas, y ownership técnico por parte de los equipos ha permitido implementar proyectos complejos con un nivel alto de coordinación y autonomía.

3.5. Estrategia de experimentación y pruebas A/B

En Despegar, la evolución de los productos se apoya en una estrategia de experimentación constante. Una de las principales herramientas utilizadas con este fin es el **testeo A/B**, que permite validar hipótesis sobre posibles mejoras, ya sean visuales (por ejemplo, cambios en el diseño o el contenido de una página) o funcionales (como modificaciones en la lógica de negocio o en el comportamiento de ciertas funcionalidades).

El procedimiento consiste en dividir el tráfico de usuarios de forma aleatoria entre dos o más variantes de una misma funcionalidad. A cada usuario se le asigna una variante y permanece en ella durante toda la duración del experimento, lo que permite medir con precisión el impacto de los cambios en distintas métricas, como por ejemplo la tasa de conversión.

Los experimentos se configuran a través de herramientas internas que permiten definir variantes, asignar porcentajes de tráfico, y también se establecen métricas objetivo. Los resultados son procesados principalmente por el equipo de Producto, que luego los analiza en conjunto con los equipos de Diseño y Tecnología. Este trabajo colaborativo permite entender no solo si hubo un cambio estadísticamente significativo, sino también si ese cambio tiene sentido desde la perspectiva del negocio y la experiencia del usuario.

En general, cuando una variante demuestra mejoras claras en las métricas definidas, se avanza con su adopción. Sin embargo, no todos los experimentos tienen resultados concluyentes. En algunos casos, incluso si no hay una diferencia estadísticamente significativa, se decide mantener la rama test si se considera que representa una mejora cualitativa o si forma parte de una dirección de producto que se quiere seguir explorando en futuras iteraciones.

Este enfoque no solo permite tomar decisiones basadas en evidencia, sino que también promueve una cultura de validación continua, en la que se prioriza el aprendizaje y la mejora progresiva por sobre las soluciones cerradas o definitivas.

Según Optimizely (s.f.), las pruebas A/B son fundamentales para “tomar decisiones basadas en datos” al evaluar cambios en productos digitales con usuarios reales. Esta visión coincide con la práctica habitual en Despegar, donde la experimentación forma parte del día a día de los equipos.

Capítulo 4. Primera implementación del programa de fidelidad

Luego de los capítulos anteriores, que ofrecieron una introducción general al proyecto y al contexto en el que se desarrolló, a partir de este capítulo comienza el repaso por los distintos desarrollos relacionados al programa de fidelidad que se realizaron a lo largo de los últimos años. Puntualmente, en este capítulo se va a describir la primera implementación del programa de fidelidad, un momento clave en el que se establecieron muchas de las bases técnicas que luego permitirían construir las siguientes iteraciones del programa.

A lo largo del capítulo se describirán los requerimientos iniciales, cómo se organizaron los equipos para encarar este proyecto, y se mencionarán distintas decisiones técnicas relevantes que permitieron avanzar con la solución. También se detallarán las tecnologías utilizadas, los mecanismos de integración y distribución de componentes, el soporte a otros equipos, las pruebas realizadas, el encendido en producción y los aprendizajes que dejó esta primera etapa.

Si bien se comentarán aspectos generales del desarrollo, el foco estará puesto en el trabajo llevado adelante por quien escribe, especialmente en lo referido al desarrollo del componente visual reutilizable que permitió escalar la solución a lo largo de los distintos productos del sitio, su back-end asociado (el cual contiene reglas relacionadas a la experiencia del usuario), y la integración que debió realizarse con el modelo provisto por el *core* de Loyalty.

4.1. Requerimientos, objetivos y organización del proyecto

El desarrollo del programa de fidelidad surgió a partir de una necesidad concreta de negocio: ofrecer a los usuarios la posibilidad de acumular y canjear puntos con cada compra.

Desde el punto de vista funcional, el programa contempla tres modos de operación:

- **Modo adquisición:** el usuario todavía no forma parte del programa (es decir, no está *enrolado*). En este caso, se muestran incentivos visuales con el objetivo de motivar su incorporación.
- **Modo acumulación:** el usuario ya está *enrolado* y comienza a acumular puntos por cada compra realizada.
- **Modo canje** (también llamado *modo redención*): el usuario no solo está *enrolado*, sino que además tiene puntos disponibles para aplicar descuentos. Al canjearlos, puede reducir el precio de su compra. Si queda un saldo a pagar, ese monto genera nuevos puntos acumulables.

Como parte del desarrollo, fue necesario incluir información del programa en distintas partes del sitio. Cada ítem debía mostrar cuántos puntos se acumularían si se concretaba la compra, y si el usuario optaba por canjear puntos previamente acumulados, debía reflejarse también el descuento correspondiente. Como cada producto del sitio puede tener un modelo de datos distinto, se necesitó aplicar un cierto grado de normalización en las interfaces utilizadas.

Otro aspecto importante a tener en cuenta fue que el programa no se iba a habilitar en todos los países desde el inicio, sino de forma gradual. Esto se debía a negociaciones con bancos y otras consideraciones que variaban según el mercado. Por eso, era necesario contar con un mecanismo centralizado que permitiera determinar en qué contextos estaba habilitado el programa.

Para poder resolver de forma centralizada la cotización de cada ítem en puntos, el descuento obtenido por cada usuario luego de aplicar el canje, y la definición de los contextos en donde el programa estuviera prendido, se creó un nuevo stack de aplicaciones al que en distintos momentos quien escribe se ha referido como “el core de Loyalty”. Este stack de aplicaciones fue asignado a un nuevo equipo IT que se creó en aquel momento, y que hoy en día sigue existiendo para poder seguir incorporando distintas features al programa y por supuesto, para el mantenimiento de dicho stack de aplicaciones.

Desde lo visual, se buscaba una solución que diera visibilidad al programa sin afectar negativamente la experiencia de compra. Para eso se definieron algunos componentes:

- Un *switch*, que permitiera cambiar entre los modos de acumulación y canje.
- Un componente visual para mostrar la cantidad de puntos acumulables junto a cada ítem.
- *Banners*⁸ para promocionar el programa en distintas secciones del sitio.
- Y modificaciones en los componentes encargados de mostrar el precio de cada ítem, que ahora debían reflejar también el descuento en puntos cuando el usuario activara el modo canje.

Inicialmente, al equipo responsable del front-end de Alojamientos se le asignó la responsabilidad de desarrollar el componente *switch* y su back-end asociado. Con el tiempo, el mismo equipo fue asumiendo también la responsabilidad sobre otros componentes del programa, como el que muestra los puntos acumulables asociados a cada ítem y las adaptaciones necesarias para reflejar el modo *canje* en la visualización del precio. Sobre esto último se profundizará en el siguiente capítulo.

La definición de los componentes visuales y la aparición del nuevo stack de aplicaciones del core de Loyalty marcaron el punto de partida para el trabajo técnico de los distintos equipos. A partir de ese momento, cada producto debía comenzar a integrarse con el modelo

⁸ Elemento visual utilizado en sitios web o aplicaciones que busca atraer acciones del usuario o comunicar información.

expuesto por el *core*, y también debía integrar los componentes visuales anteriormente mencionados, incorporando en sus flujos la lógica del programa.

4.2. Integración del *core* de Loyalty

Como se mencionó anteriormente, para poder reflejar correctamente la información del programa de fidelidad en el sitio, cada producto debía integrarse con el modelo expuesto por el *core* de Loyalty.

En este punto, resulta útil recordar la Figura 3.1, en donde se graficó a nivel general la arquitectura de microservicios de cada producto en Despegar. Dentro de la capa de *reglas de negocio* mencionada en dicha imagen, es importante destacar que parte de esas reglas son las reglas de *pricing*, es decir, las reglas que se aplican para determinar el precio final de cada ítem. Con esto en mente, se entiende claramente en qué parte del stack se insertó el componente de cotización desarrollado por el *core* de Loyalty.

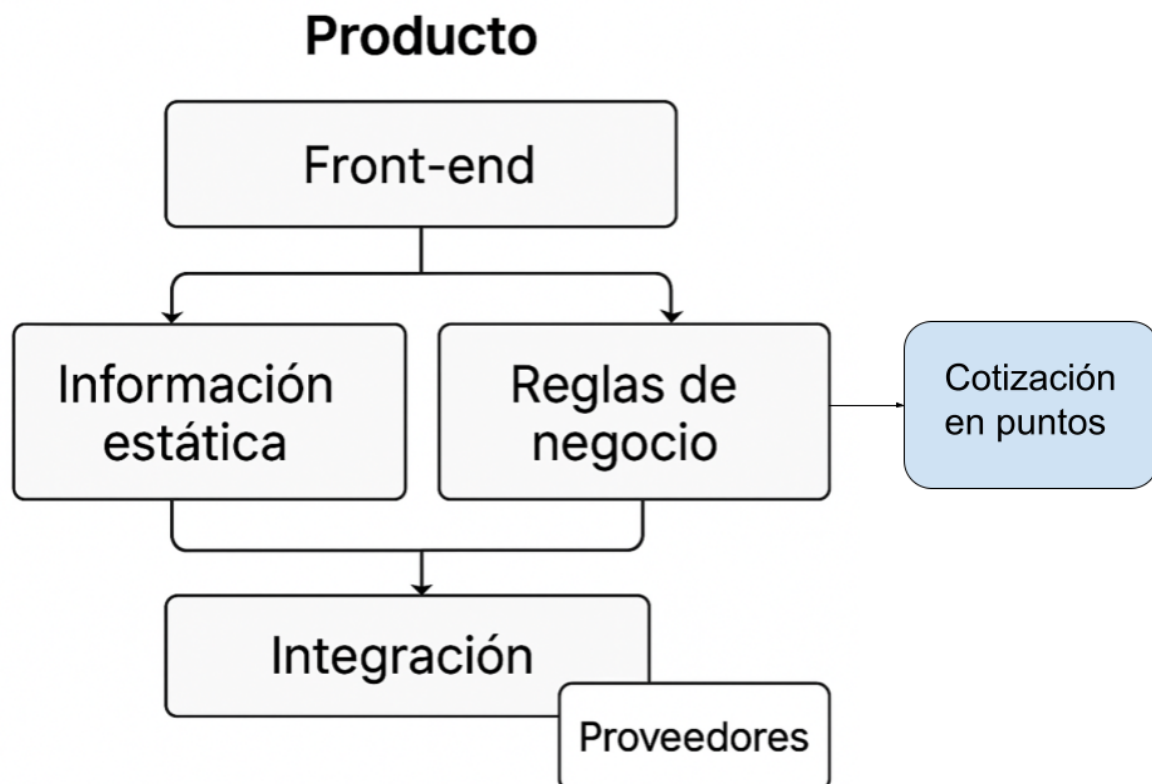


Figura 4.1. Punto de inserción del componente de cotización en puntos dentro de la arquitectura de microservicios de Despegar para un producto.

Como se observa en la Figura 4.1, el componente de cotización en puntos se integró en los distintos productos justo en el punto en el cual se construye el precio. Es decir: una vez aplicadas todas las reglas que componen el precio final, se realiza un request al componente de cotización, informando el precio de cada ítem, el identificador del usuario

para el cual se desea cotizar, y otros datos de contexto como el país y el producto (ya que pueden existir reglas diferenciales según estas variables).

Este request puede ejemplificarse de la siguiente manera:

```
curl -X POST "http://loyalty/points/quote?country=BR&product=hotel" \
-H "Content-Type: application/json" \
-d '{
  "user_id": "12345678",
  "items": [
    {
      "item_id": "HTL123",
      "final_price": 1200.50
    },
    {
      "item_id": "HTL456",
      "final_price": 875.00
    }
  ]
}'
```

Luego de aplicar algunas validaciones —como verificar si el programa está activo para el país solicitado, si el usuario está registrado, y si posee puntos disponibles para canjear— el componente responde con información como la siguiente:

```
{
  "items": [
    {
      "item_id": "HTL123",
      "redemption_price": 1000.50,
      "accrual_points": 1200,
      "accrual_points_in_redemption": 1000
    },
    {
      "item_id": "HTL456",
      "redemption_price": 725.00,
      "accrual_points": 875,
      "accrual_points_in_redemption": 725
    }
  ]
}
```

Como se puede ver, para cada ítem se devuelve:

- El **precio en dinero** que el usuario debe pagar si decide canjear sus puntos.
- La **cantidad de puntos acumulables** si no canjea.

- La **cantidad de puntos acumulables** si sí canjea.

Es importante remarcar que los puntos se acumulan siempre sobre el dinero efectivamente gastado, y no sobre la parte pagada con puntos. Llevado a un caso extremo, si el usuario cuenta con los puntos suficientes como para pagar la totalidad del ítem, no acumulará ningún punto por esa operación.

Como se mencionó anteriormente, el equipo de quien escribe no fue responsable de la integración con el componente de cotización, sino que esa tarea fue realizada por otro equipo del mismo producto, especializado en las reglas de negocio vinculadas al armado del precio. Naturalmente, dicha información termina llegando al front-end, por lo que fue necesario realizar el correspondiente mapeo.

Por último, si bien en esta sección se hizo foco en la cotización de puntos, vale aclarar que la integración con el core de Loyalty no se limitó únicamente a ese componente. También fue necesario integrar la aplicación encargada de definir si un cierto contexto estaba habilitado o no para participar del programa. Este aspecto será abordado más adelante, cuando se detalle el funcionamiento del back-end del componente *switch*.

4.4. Componente Front-End compartido

Uno de los primeros desafíos técnicos al momento de implementar el programa de fidelidad (fuera del mencionado *core* de Loyalty) fue desarrollar el componente visual que permita alternar entre los modos de acumulación y canje de puntos. Internamente, este componente fue llamado *loyalty-redemption-switch*. Desde su primera versión, su objetivo es ofrecer a los usuarios una forma clara de entender en qué modo están operando y, si corresponde, permitirles cambiar de modo para aprovechar los beneficios del programa.

En una primera instancia, se había propuesto que cada equipo de producto implementara este componente visual por su cuenta. Sin embargo, esa estrategia presentaba grandes desventajas: por un lado, implicaba duplicar la lógica en múltiples lugares, con el riesgo de generar inconsistencias entre verticales. Por otro, aumentaba los tiempos de desarrollo, dificultaba el mantenimiento a largo plazo y complicaba la introducción de mejoras o ajustes globales.

Para resolver esta situación, se tomó la decisión de **centralizar la implementación en un único componente compartido**, que pudiera insertarse en las distintas aplicaciones front-end del sitio. La responsabilidad recayó en el equipo de front-end de Alojamientos, y puntualmente en quien escribe, dado que dicho equipo contaba con la disponibilidad de tiempo necesaria.

4.4.1. Lógica funcional del componente

Desde el punto de vista funcional, el componente debe adaptarse a distintos escenarios según el estado del usuario:

- Ante un usuario **no enrolado**, el componente se presentaba con mensajes promocionales del tipo “Pasaporte Despegar” y “Empezá a sumar puntos” (Figura

4.2), funcionando como una herramienta de enrolamiento. Al hacer clic, redireccionaba a una sección del sitio donde el usuario podía inscribirse al programa. Este caso de uso luego fue discontinuado tras una iteración.



Figura 4.2. Componente *loyalty-redemption-switch* en *modo adquisición*.

- Ante un usuario ya **enrolado**, el comportamiento depende de su saldo de puntos. Si no alcanza el mínimo necesario para redimir (por ejemplo, 100 puntos), el componente simplemente se muestra en *modo acumulación* (con el *switch* deshabilitado), como se puede observar en la Figura 4.3. En cambio, si cumple con los requisitos, el usuario puede usar el *switch* para alternar entre **modo acumulación** y **modo canje**, como muestra la Figura 4.4.

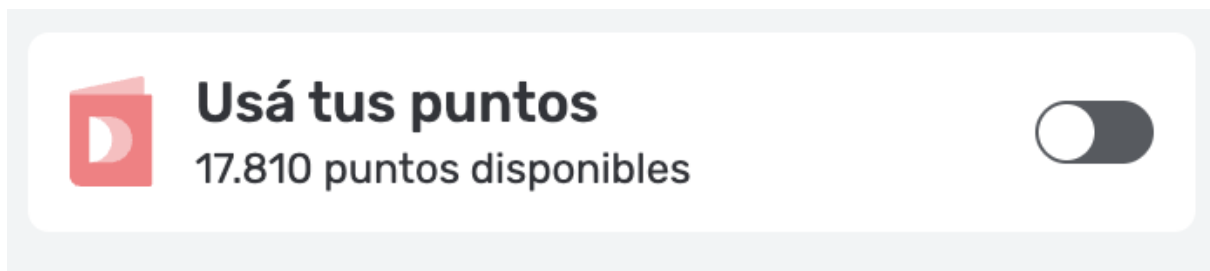


Figura 4.3. Componente *loyalty-redemption-switch* en *modo acumulación*.

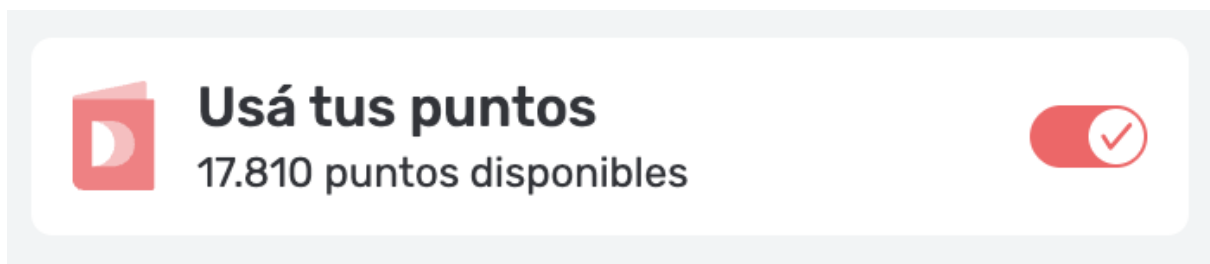


Figura 4.4. Componente *loyalty-redemption-switch* en *modo canje*.

Este comportamiento debe mantenerse consistente en todas las verticales, lo cual reforzó la necesidad de encapsular la lógica en un único lugar.

4.4.2. Arquitectura del componente y parámetros

El componente fue diseñado con un enfoque **configurable y portable**, de modo que pudiera ser utilizado en múltiples contextos sin necesidad de ajustes específicos. Para eso, recibe una serie de parámetros obligatorios, entre los cuales se destacan:

- El tipo de producto donde se muestra (por ejemplo, “hotels”, “flights”).
- El paso dentro del *funnel* de compra (por ejemplo, “results”, “detail”, “checkout”).
- Locale (información necesaria para personalizar textos).

Al inicializarse, el componente ejecuta un request HTTP al servicio *s-commons-vr*, enviando estos parámetros junto con información relacionada al usuario. A partir de la respuesta, el componente determina qué estado mostrar: si el *switch* debe estar activo, si el usuario puede o no redimir y también qué ícono y textos deben mostrarse.

Toda esta lógica se resuelve **dentro del componente mismo**, de forma que las aplicaciones host no necesitan realizar ningún tipo de integración adicional. De esta forma, alcanza con insertar el componente y pasarle los parámetros correspondientes.

Para lograr esto, se utilizó como tecnología base **vanilla JavaScript**. Esto permitió mantener una dependencia cero con frameworks o tecnologías particulares, garantizando compatibilidad total con las distintas aplicaciones front-end existentes en Despegar, muchas de las cuales estaban escritas en Angular, React o incluso tecnologías anteriores.

4.4.3. Evaluación de tecnologías posibles

Previo a definir la tecnología base, se realizó un breve análisis de viabilidad sobre otras alternativas:

- **Web Components (Stencil):** Stencil es un compilador que permite generar Web Components reutilizables, compatibles con cualquier framework o librería. Su objetivo principal es facilitar la creación de componentes portables y performantes, con soporte para lazy loading y prerendering (Stencil, s. f.). Sin embargo, en 2019 su adopción aún era incipiente dentro de la empresa, y su integración con ciertas aplicaciones no estaba del todo probada. Esto hizo que se descartara para este desarrollo puntual.
- **Componentes embebidos con React o Angular:** tanto React como Angular ofrecen estructuras sólidas para el desarrollo de interfaces modulares. En el caso de React, incluso es posible renderizar componentes aislados dentro de sitios existentes. Angular, por su parte, proporciona herramientas para encapsular módulos reutilizables. Sin embargo, en este caso se optó por realizar un componente extremadamente liviano y sin dependencias externas, capaz de insertarse fácilmente en aplicaciones heterogéneas. Incorporar estos frameworks solo para renderizar el *switch* habría implicado aumentar el tamaño del *bundle*⁹ y acoplar el componente a tecnologías que no siempre coincidirían con las del host.
- **Vanilla JS (JavaScript puro, sin usar librerías ni frameworks externos):** fue finalmente la opción elegida. Su principal ventaja es la compatibilidad absoluta con cualquier entorno web, sin necesidad de ajustes especiales. JavaScript permite crear

⁹ Conjunto de archivos de código (como JavaScript, CSS o HTML) agrupados y optimizados para ser servidos al navegador de forma eficiente.

funcionalidades reutilizables e integrables directamente sobre el DOM, sin necesidad de frameworks, utilizando estructuras como funciones, clases y módulos, y aprovechando APIs nativas basadas en atributos y eventos del navegador (MDN Web Docs, s.f.).

La implementación de este componente fue realizada por quien escribe, incluyendo también la definición de la tecnología a utilizar. Esta elección fue validada con miembros más experimentados del equipo, quienes aportaron su visión para asegurar una base sólida y escalable.

4.5. Back-End del componente Front-End

Para que el componente *loyalty-redemption-switch* pudiera operar de forma verdaderamente autónoma y reutilizable en múltiples verticales, fue necesario construir un servicio back-end que resolviera toda la lógica contextual y de negocio. La nueva aplicación creada para este fin fue llamada *s-commons-vr*. Su principal responsabilidad es brindar al componente *loyalty-redemption-switch* los datos necesarios para funcionar correctamente sin depender de la lógica local de cada aplicación. Luego, como veremos más adelante, también empezó a manejar lógica asociada a otros componentes visuales del programa.

Entre sus responsabilidades principales se incluyen:

- Determinar si el usuario está enrolado en el programa de fidelidad.
- Consultar si el usuario tiene la cantidad mínima de puntos para poder redimir.
- Verificar si el país donde se encuentra tiene activa la funcionalidad.
- Verificar si la funcionalidad debe estar habilitada en el paso actual del funnel.

Este diseño permitió mantener la lógica del componente **centralizada y desacoplada**, intentando asegurar una experiencia consistente en todas las verticales, independientemente de cómo estuviera construida cada aplicación.

4.5.1. Elección de tecnología: ¿por qué Node.js?

Dentro del ecosistema de Despegar, lo habitual es desarrollar servicios en Java o Scala, tecnologías ampliamente utilizadas por su robustez, herramientas de desarrollo maduras y ecosistemas consolidados. Sin embargo, para el caso particular del servicio *s-commons-vr*, se optó por una solución más liviana: **Node.js combinado con el framework Express**.

Desde el punto de vista técnico, esta decisión tiene sentido por la naturaleza misma del servicio. *s-commons-vr* no requiere procesamiento intensivo ni lógica compleja. Su responsabilidad es actuar como un **orquestador**: recibe solicitudes desde múltiples aplicaciones de distintos productos (siempre desde *loyalty-redemption-switch*), consulta otros servicios internos (como el estado del usuario en el programa o las configuraciones activas para un país), aplica reglas simples y devuelve una respuesta uniforme. En este tipo de escenarios, Node.js resulta especialmente eficiente gracias a su modelo asíncrono,

basado en eventos, que permite manejar múltiples requests concurrentes sin bloquear el hilo principal (Node.js, s.f.). Esta característica lo convierte en una alternativa ideal para servicios de back-end cuyo principal objetivo es enrutar y transformar información.

A esto se suma la ventaja del ecosistema. Node.js ofrece un entorno ágil respaldado por una comunidad muy activa, lo que simplifica la incorporación de librerías y herramientas mediante npm. Esta herramienta permite instalar dependencias, reutilizar código y mantener entornos consistentes gracias a archivos como package.json y package-lock.json (npm, s. f.). Todo esto permite comenzar a desarrollar rápidamente, sin una etapa extensa de configuración, y mantener el servicio liviano y fácil de ajustar.

Sobre esa base se construyó el servicio utilizando **Express**, un framework minimalista pensado específicamente para crear APIs de forma rápida y flexible. Express simplifica la definición de rutas, middlewares, validaciones y respuestas de error, manteniendo el control total del flujo de ejecución sin imponer una estructura rígida (Express, s. f.). Su adopción generalizada también garantiza soporte, documentación y estabilidad.

Además de los fundamentos técnicos sobre Node.js y Express, también se tuvieron en cuenta las habilidades del equipo. Dentro del equipo encargado de desarrollar *s-commons-vr* ya existía experiencia previa trabajando con Node.js en otros proyectos, lo que permitió avanzar con una curva de aprendizaje baja.

Por otro lado, este enfoque también responde a una estrategia de simplicidad. Se buscó una solución fácil de mantener, que cumpla con su propósito sin agregar fricción ni acoplamientos fuertes. Si en el futuro fuera necesario migrar esta lógica a un entorno más robusto, hacerlo sería simple. Mientras tanto, Node.js ofrece el balance justo entre rendimiento, velocidad de entrega y flexibilidad para evolucionar.

4.5.2. Alternativas consideradas

Antes de decidir por Node.js, se evaluó la posibilidad de implementar el servicio en Scala, aprovechando los recursos y la infraestructura ya existente. Esta opción garantizaba mayor alineación con el resto de servicios internos mantenidos por el equipo, pero no pareció ser un argumento fuerte para decidir esta opción por sobre los beneficios de Node.js anteriormente mencionados.

También se descartó dejar toda la lógica embebida dentro del componente front-end. Aunque técnicamente era posible, presentaba varios problemas:

- Dificultaba el mantenimiento de reglas de negocio, especialmente ante cambios frecuentes.
- Exponía información sensible al cliente, como el mínimo de puntos o el estado del usuario.
- Hacía más costosa la evolución del producto, ya que obligaba a actualizar la versión del componente en las distintas aplicaciones que lo integran con mayor frecuencia.

En definitiva, la creación de *s-commons-vr* y la elección de Node.js respondieron a una combinación de eficiencia técnica y visión de producto: era una solución liviana, flexible y coherente con los tiempos y objetivos del equipo.

Esta aplicación también fue desarrollada por quien escribe. Nuevamente, la elección de la tecnología y el diseño general de la solución se llevaron a cabo con el acompañamiento de desarrolladores de mayor experiencia, lo que permitió fortalecer la calidad técnica del entregable.

4.6. Integración del programa en Alojamientos

Como se mencionó anteriormente, en el caso del equipo a cargo del front-end de Alojamientos, la integración con el *core* de Loyalty no implicó una integración directa contra el cotizador, ya que esa conexión se realizó a través de otro back-end intermedio (anteriormente referenciada como *la capa de reglas de negocio*). El equipo a cargo de dicho servicio, que ya formaba parte del ecosistema, se encargó de consumir la información del cotizador y exponer los datos necesarios para implementar la lógica del programa del lado del front-end.

A partir de esa respuesta, el equipo de quien escribe debió adaptar su back-end y front-end para poder consumir y mapear correctamente la información relacionada al programa. Esto incluye, especialmente, el precio de cada ítem en *modo canje* y en *modo acumulación*, así como la cantidad de puntos acumulables en cada caso. Luego, dicha información comenzó a exponerse en pantalla, en la sección que muestra el precio.

Además de estos cambios, también se comenzó a trabajar en los componentes visuales vinculados al programa. El foco inicial respecto a esto fue integrar el componente compartido *loyalty-redemption-switch*. Es importante destacar que quien escribe no realizó la integración del componente en Alojamientos, sino que esa tarea se delegó en un compañero del equipo, con el objetivo de recopilar información (respecto a la experiencia de integración) para armar una guía que resulte útil para la integración en aplicaciones de otros productos.

En paralelo, quien escribe participó activamente en la implementación de otros componentes visuales asociados al programa, como los que indican la cantidad de puntos acumulables o el ahorro por canje. En esta primera etapa, esos componentes no se desarrollaron como componentes reutilizables a nivel transversal, sino que se implementaron de forma puntual dentro de las distintas aplicaciones front-end, dado que su complejidad era muy baja. Esa experiencia, junto con las oportunidades de mejora detectadas, fue uno de los disparadores que llevó a quien escribe a proponer más adelante una estandarización y componentización transversal de estos elementos, como se verá en el capítulo siguiente.

4.7. Distribución e integración del componente compartido

Una vez que el desarrollo de *loyalty-redemption-switch* alcanzó un primer estado funcional, comenzó la etapa de distribución e integración en las distintas aplicaciones front-end de

cada producto del sitio. Para facilitar su adopción, se buscó una solución que permitiera distribuir el componente como una librería reutilizable, desacoplada y de fácil integración, incluso en aplicaciones con tecnologías distintas.

Para lograrlo, se utilizó *Webpack* como herramienta de empaquetado. *Webpack* es un *module bundler* ampliamente utilizado en el ecosistema JavaScript. Su función principal es tomar los distintos archivos del proyecto (JavaScript, estilos, imágenes, etc.) y generar un *bundle* final optimizado, apto para ser importado desde otras aplicaciones (*Webpack*, s.f.).

El componente se publicó en un repositorio interno de paquetes de la empresa, junto con otras librerías compartidas. Las aplicaciones que deseaban integrarlo lo instalaron utilizando *npm*. Para facilitar su adopción, se estableció una política de versionado semántico (*semver*), permitiendo a cada equipo definir cuándo aplicar actualizaciones sin temor a generar regresiones.

Junto con la publicación del paquete se preparó una guía de integración detallada (gracias a la experiencia obtenida de la integración del componente dentro del front-end de Alojamientos), que incluía ejemplos concretos de implementación en distintos contextos. La guía explicaba cómo inicializar el componente correctamente, cómo propagar el modo activo (adquisición, acumulación o canje) hacia otros componentes, y también cómo utilizar ciertos headers para forzar comportamientos específicos en flujos particulares. Esta documentación fue enviada por correo a los equipos involucrados y sirvió como material base para una reunión técnica donde se presentó el componente formalmente.

La reunión de difusión fue coordinada por quien escribe, en un momento donde todavía no había coordinado demasiadas reuniones internas, y ninguna externa al equipo. Más allá del contenido técnico, esta instancia representó un desafío importante desde lo organizativo: había que alinear a múltiples equipos, responder preguntas en el momento y definir expectativas claras para la integración. La coordinación de ese espacio, junto con el soporte brindado durante todo el proceso, fue una experiencia clave en esa etapa de la carrera de quien escribe, y también una oportunidad para ganar exposición por fuera del equipo.

Cada equipo designó a una persona responsable de implementar el componente en su front-end. Aunque el componente estaba diseñado para minimizar fricciones, surgieron dudas relacionadas con la configuración de parámetros o formas de interacción con otros elementos visuales. A lo largo de esta etapa, quien escribe mantuvo espacios 1:1 con desarrolladores de distintos equipos, brindando soporte técnico y validando casos puntuales. En varios de esos encuentros, se recibió feedback valioso que permitió identificar mejoras tanto en el componente como en su documentación.

El proceso de distribución e integración de *loyalty-redemption-switch* representó un paso importante no solo desde lo técnico, sino también desde lo organizativo y colaborativo. Asegurar una experiencia unificada en todo el sitio implicó coordinar esfuerzos entre equipos diversos, documentar de forma clara y acompañar activamente cada implementación.

4.8. Trackeo y medición

Como en toda funcionalidad importante, la posibilidad de realizar un seguimiento claro y consistente del comportamiento de los usuarios fue un aspecto importante desde el inicio. Aunque en una primera instancia se habían definido algunos eventos que se debían trackear en los distintos productos, la forma en que estos trackeos debían implementarse cambió de manera significativa cuando se tomó la decisión de componentizar *loyalty-redemption-switch* y centralizar su lógica en un único lugar.

En el planteo original, se asumía que cada aplicación que implementara el *switch* se encargaría de realizar los trackeos internamente, según las necesidades del equipo y respetando ciertos lineamientos generales. Sin embargo, al unificar el desarrollo del componente, también se unificó la responsabilidad de instrumentar los trackeos. Esto no solo amplió el alcance del trabajo del equipo de quien escribe, sino que además introdujo nuevos desafíos técnicos, ya que ahora era el componente el que debía decidir cuándo, cómo y con qué datos realizar estos trackeos.

Para resolver esto, quien escribe se encargó de la integración de *s-commons-vr* con un servicio interno de trackeo de eventos, encargado de almacenar los datos recibidos desde distintas fuentes. Cada vez que el componente debe trackear un evento, se envía un request POST a este servicio, con los parámetros necesarios para registrar la acción.

En total, se definieron tres eventos principales:

- **Click en modo adquisición:** trackeo asociado a un usuario no enrolado en el programa que hace clic en el *switch*. Representaba una señal de intención de enrolamiento, pero como se mencionó anteriormente, más adelante este caso de uso fue discontinuado.
- **Click en modo acumulación o canje:** trackeo asociado a un usuario que ya está enrolado y alterna el modo activo.
- **Inicialización del componente:** se trackea cada vez que se construye el componente, registrando el estado inicial. Lo más relevante de este evento es el *modo* en que se encuentra el componente al inicializarse.

Para que los eventos pudieran ser emitidos correctamente desde el componente, es necesario contar con determinada información contextual que no siempre resulta trivial de obtener. Como parte del proceso de diseño, se trabajó para que el componente tuviera acceso a todos los datos necesarios desde su propio contexto de ejecución, sin depender de llamadas adicionales ni lógica duplicada.

Esto permitió que los trackeos pudieran ejecutarse de forma inmediata, manteniendo la eficiencia del componente y asegurando una integración más simple con los sistemas de analítica internos. La decisión también ayudó a reducir la complejidad técnica, minimizando puntos de falla y optimizando la performance general de la solución.

Por último, se construyeron tableros de seguimiento de métricas, utilizados principalmente por el equipo de Producto para monitorear la adopción y comportamiento del programa en producción. Si bien el armado y mantenimiento de estos tableros fue gestionado desde otras áreas, el equipo de quien escribe brindó soporte durante su construcción, facilitando conocimiento sobre las estructuras de datos y validación de resultados.

La unificación del componente no solo permitió garantizar una experiencia visual unificada, sino también una estrategia de medición más sólida y escalable.

4.9. Pruebas y puesta en producción

Como en la mayoría de los desarrollos en Despegar, la etapa de pruebas estuvo a cargo de los propios desarrolladores. A medida que cada equipo integraba el componente *loyalty-redemption-switch* y desarrollaba la integración con el modelo de precio en puntos proveniente del *core*, comenzaba a realizar validaciones individuales sobre su funcionamiento, asegurándose de que los parámetros fueran correctos, que la visibilidad de los componentes fuera coherente y que no se rompiera el comportamiento esperado del flujo de venta.

Una vez que todos los equipos completaron sus primeras implementaciones, se organizó una reunión presencial de pruebas conjuntas, con representantes técnicos de varios equipos de IT. La coordinación de esa instancia estuvo a cargo del Senior Manager del área —responsable de un conjunto de equipos— y de quien escribe. Fue una instancia importante para identificar comportamientos inconsistentes o mejoras necesarias antes de la prenda del programa en producción.

Durante esa jornada de validación, se realizaron pruebas manuales en distintos entornos y configuraciones. Los bugs y oportunidades de mejora detectados fueron documentados en una planilla compartida, donde cada ítem incluía su descripción, el equipo impactado y su nivel de prioridad. Esto permitió tener un panorama general del estado del proyecto, y facilitó la asignación de tareas correctivas para que cada equipo pudiera enfocarse en resolver los temas que le correspondían.

Una vez corregidos los principales puntos observados, se comenzó a subir la funcionalidad a producción. El programa fue desplegado con su comportamiento por defecto apagado, de modo que su inclusión en el sitio no modificara la experiencia de los usuarios finales. Esta estrategia permitió realizar un monitoreo exhaustivo en producción, validando que no se vieran afectadas otras funcionalidades ni el rendimiento general del sitio.

Aunque este aspecto será retomado más adelante en el capítulo, vale mencionar que la activación del programa se gestionó mediante un **experimento A/B**. La habilitación se realizó de forma incremental: durante los primeros días se fue ampliando gradualmente el porcentaje de usuarios expuestos al nuevo comportamiento, hasta alcanzar un escenario 50/50 dentro del contexto elegido para la prenda inicial.

Esta estrategia permitió monitorear el impacto del programa de manera controlada, comparando variantes del sitio con una muestra significativa. La activación progresiva fue

posible gracias a la combinación del experimento A/B y de un servicio del *core* de Loyalty, encargado de exponer el estado del programa y definir en qué países debe estar habilitado.

4.10. Alarmas y monitoreo en la nueva aplicación

Desde las primeras etapas de despliegue del programa de fidelidad se implementaron mecanismos de monitoreo y alertas para asegurar la estabilidad de la nueva funcionalidad y detectar posibles fallas a tiempo. La estrategia de observabilidad se apoyó en las herramientas de monitoreo utilizadas habitualmente en la empresa.

La participación de quien escribe respecto a esto estuvo relacionada con la aplicación *s-commons-vr*, el back-end asociado a los componentes visuales. Se establecieron umbrales para el porcentaje de transacciones erróneas: si durante un determinado período de tiempo el sistema supera cierto porcentaje de errores, se activa un estado de *warning*; y si en ese mismo período se supera un umbral mayor, la aplicación pasa a estado de *alert*. Estas condiciones están orientadas a detectar comportamientos anómalos de forma proactiva y asegurar una rápida respuesta ante incidentes.

Cuando se alcanzan estos estados, el sistema de monitoreo genera notificaciones automáticas tanto en un canal general del equipo como de forma individual a determinadas personas responsables de la operación, entre las cuales siempre estuvo quien escribe.

Con el paso del tiempo y el encendido progresivo del programa en distintos contextos (como países o productos específicos), se configuraron nuevas alertas, incluyendo por ejemplo errores segmentados por país, detección de *timeouts* en servicios consumidos por el back-end, entre otros casos relevantes. Esta evolución permitió ajustar el monitoreo a medida que se ampliaba el alcance del programa.

En paralelo, también se emplearon otras herramientas para el seguimiento y la resolución de problemas, como el acceso a logs de las aplicaciones, que permite investigar el detalle de cada transacción y facilitar el diagnóstico de errores.

El conjunto de estas herramientas y prácticas es clave para garantizar una operación estable, minimizar el impacto ante posibles fallos, y dar soporte a un despliegue controlado y seguro del programa.

4.11. A/B Testing y análisis de resultados

Como se explicó en el capítulo anterior, un A/B test es una técnica que permite validar hipótesis sobre posibles mejoras mediante la exposición controlada a usuarios reales. Para ello, se divide la audiencia de forma aleatoria en dos grupos (o más): uno ve la versión actual del sitio (grupo de control) y otro una versión modificada con algún cambio específico (grupo de test). Al comparar el comportamiento de ambos grupos, es posible determinar si el cambio tuvo un impacto real, incluso si fue sutil pero estadísticamente significativo.

4.11.1. Cómo se hacen los A/B tests en Despegar

En Despegar existe una aplicación interna encargada de administrar todos los A/B tests del sitio, la cual es responsable de *sortear* al grupo de usuarios alcanzado por el experimento de forma pareja entre las ramas *test* y *control*, y también de mantener esa asignación estable a lo largo del experimento.

Esto también permite que los experimentos sean consistentes entre distintos productos. En concreto, para este experimento se definieron dos ramas:

- **Control:** sin ningún componente del programa.
- **Test:** con el programa habilitado y visible para el usuario.

Dicho esto, es importante reforzar que si un usuario navega el flujo de Vuelos y entraba en el grupo de test (es decir, veía los componentes del programa de fidelidad), luego al entrar a Alojamientos también seguiría en la misma rama y por ende iba a seguir viendo la funcionalidad, sin que cambie de un producto a otro.

4.11.2. A/B tests sincronizados

Aunque el programa de fidelidad era una iniciativa transversal, cada producto ejecutó su propio A/B test. Esto se hizo así porque cada producto maneja diferentes volúmenes de tráfico, lo que afecta directamente cuánto tiempo se necesita para analizar los resultados.

Por ejemplo, productos con más visitas (como Vuelos, Alojamientos o Paquetes) pudieron cerrar su test antes que otros como Autos o Actividades. Pero más allá de esas diferencias, todos los tests estuvieron sincronizados en términos de asignación: cada usuario estaba siempre en la misma rama sin importar en qué parte del sitio navegara.

4.11.3. Hipótesis y criterios de evaluación

A diferencia de otros A/B tests más orientados a buscar un incremento de ventas o clics en determinados componentes, en este caso el objetivo era más conservador. No se buscaba necesariamente que el programa incrementara la conversión, sino confirmar que **no la afectara negativamente**.

Esto se debe a que el programa de fidelidad fue pensado como una herramienta de largo plazo, para fomentar la recompra y la retención de usuarios. Por eso, en esta etapa inicial el foco estuvo puesto en validar que no hubiera una caída de performance al incorporar los nuevos componentes visuales y funcionales.

Como se mencionó anteriormente, el experimento se ejecutó inicialmente en Brasil, donde se repartió el tráfico 50/50 entre las dos ramas.

4.11.4. Trackeos y análisis de resultados

Durante el desarrollo de esta iniciativa se generaron trackeos de eventos específicos para poder analizar el comportamiento de los usuarios frente al programa. Algunos de los eventos más relevantes fueron:

- Uso de *loyalty-redemption-switch*, detallado en la sección 4.8.
- Visualización de los otros componentes visuales del programa.
- Enrolamiento de los usuarios en el programa.

Estos trackeos se mantuvieron de forma permanente e incluso hoy siguen siendo útiles para hacer seguimiento del uso del programa o entender posibles mejoras.

El análisis posterior a los A/B tests confirmó que no se registraron caídas significativas en la conversión. Además, en los casos donde el usuario comenzó a formar parte del programa, se observó una buena interacción con los componentes, lo cual era un indicio positivo en cuanto a entendimiento y aceptación.

4.11.5. Qué se decidió a partir de esto

Con los resultados sobre la mesa, y una vez validado que la conversión no se veía afectada, se decidió avanzar con la activación general del programa en Brasil. Más adelante se empezó a planificar el mismo esquema para otros países.

También quedó instalada una base de análisis para iteraciones futuras. Al tener todos los trackeos ya incorporados, cualquier nuevo cambio relacionado al programa puede ser testeado o medido con mucha más facilidad.

4.12. Conclusiones de la primera implementación

Desde la perspectiva de quien escribe, la experiencia de participar en esta primera salida a producción del programa fue sumamente positiva. Se trató de una iniciativa de gran impacto dentro de la compañía, lo que permitió ganar exposición frente a otros equipos y áreas de negocio, tanto a nivel técnico como organizativo. Además, implicó desafíos concretos que ayudaron a fortalecer habilidades técnicas (especialmente en lo relacionado al diseño de componentes compartidos, definición de tecnologías a utilizar y distribución vía librerías), así como habilidades blandas como la coordinación, la comunicación entre equipos y la gestión de pruebas en conjunto.

En cuanto al rendimiento del programa en producción, se consideró que el lanzamiento fue exitoso. No se registraron caídas de conversión, el sistema se comportó de manera estable, y se obtuvo información valiosa sobre el uso de la funcionalidad por parte de los usuarios reales. Aun así, el análisis de los primeros resultados dejó un punto de mejora claro: el porcentaje de usuarios que se enrolaban en el programa fue menor al esperado.

El diagnóstico fue que, al momento del lanzamiento, el flujo completo para participar del programa requería cumplir con dos condiciones: estar logueado en el sitio (registro o login) y luego enrolarse al programa de puntos. Esa segunda acción agregaba una fricción adicional que terminaba dejando afuera a parte de los usuarios. A partir de ese hallazgo, se planificó una nueva iteración del programa, que se mencionará más adelante, orientada a reducir esa barrera.

La estrategia de encender primero en Brasil se consideró acertada. Permitió validar en un mercado muy importante antes de escalar a otros mercados también relevantes.

Esta primera etapa dejó al equipo en una buena posición para encarar lo que vendría después. Por un lado, se confirmaron muchas de las decisiones técnicas y de producto. Por otro lado, se aprendió de lo que no funcionó tan bien y se lograron mejoras concretas en las siguientes versiones. También quedó armado un ecosistema técnico robusto —con trackeos, componentes reutilizables y procesos establecidos— que sirvió como base para las iteraciones futuras.

4.13. Iteraciones posteriores y evolución del programa

Luego de validar el comportamiento del programa de fidelidad en Brasil, se comenzó a planificar su expansión a otros países. Esto no fue inmediato, ya que en cada mercado era necesario cerrar acuerdos con bancos u otros partners, principalmente para definir la mecánica de acumulación y las tarjetas de crédito asociadas al programa. El siguiente país fue Argentina, donde se concretó un acuerdo con el banco ICBC, que actualmente distribuye la tarjeta de Pasaporte Despegar. Aunque se discutió internamente si valía la pena repetir un A/B test, se concluyó que sí, dada la relevancia del mercado. Más adelante vino México, otro mercado estratégico para la compañía, también con su propio test, y progresivamente se fueron sumando otros países, aunque ya sin experimentos formales.

La incorporación del programa en cada nuevo país requirió trabajo coordinado entre distintos equipos. Por un lado, el equipo a cargo del *core* de Loyalty debía adaptar su lógica para el nuevo mercado. Por otro lado, desde el lado del front-end, también hubo pequeñas tareas de adecuación, sobre todo relacionadas con comunicaciones localizadas, algunas de las cuales estuvieron a cargo de quien escribe.

Una de las iteraciones más importantes que surgió a partir de los primeros aprendizajes fue la relacionada con el enrolamiento de los usuarios. Como se mencionó anteriormente, al principio para formar parte del programa era necesario no solo estar logueado en el sitio, sino también completar un segundo paso de enrolamiento. Esto agregaba fricción, y se notó en los primeros resultados. A partir de eso se diseñó una mejora que consistió en enrolar automáticamente a todos los usuarios que iniciaran sesión. Si bien esto podía implicar que algunos usuarios se convirtieran en miembros del programa sin notarlo al principio, la hipótesis fue que una base más grande permitiría ofrecer mayor valor.

Además del cambio técnico, también se ajustaron algunos textos en la experiencia para no logueados, con el objetivo de reforzar el mensaje de que registrarse o loguearse brinda beneficios asociados al programa. Algunos de estos ajustes también fueron implementados por quien escribe. El impacto fue notorio: a partir de esa iteración, el crecimiento de la base de miembros fue sostenido, y actualmente cerca del 75% de los usuarios que compran en Despegar ya forman parte del programa de fidelidad.

Otra de las evoluciones que tuvo el programa fue en relación al uso del *modo canje*. Si bien el componente *loyalty-redemption-switch* permitía cambiar entre ver precios normales o con descuento por canje de puntos, su nivel de uso no era tan alto como se esperaba. Sin embargo, los datos mostraban que los usuarios que efectivamente usaban el *modo canje*

solían (y suelen) tener un ticket promedio más alto. Esto llevó a una serie de decisiones orientadas a darle más visibilidad al *switch* en el flujo.

Inicialmente, se probó incluir el componente en el header de la página, dentro de un menú asociado al programa, para unificar la experiencia. Luego, se lo incorporó directamente en el paso de búsqueda de algunos productos, con el objetivo de aumentar la cantidad de clics y mejorar la percepción del beneficio. Estas decisiones se tomaron en base a los trackeos ya implementados desde el inicio, lo cual permitió analizar el comportamiento real y ajustar la estrategia.

En resumen, la primera implementación del programa de fidelidad marcó un antes y un después tanto a nivel técnico como organizacional. Se logró incorporar una nueva capa de valor al sitio, construir una solución robusta y modular, y establecer una estrategia de despliegue gradual que permitió obtener aprendizajes valiosos.

Gracias a las decisiones tomadas en esta etapa —desde los A/B tests hasta las mejoras de enrolamiento y visibilidad— el programa logró consolidarse, crecer y sentar las bases para lo que vendría después. Hoy, con una base amplia de usuarios fidelizados y un ecosistema técnico maduro, se encuentra en una posición sólida para seguir evolucionando.

Capítulo 5. Unificación técnica y liderazgo en la implementación de componentes Angular

Este capítulo describe una mejora técnica a la implementación inicial que tuvo un gran valor personal para quien escribe, tanto por su impacto técnico como por el rol asumido durante su implementación.

La propuesta surgió como una idea propia y logró incorporarse en la planificación de todos los equipos front-end encargados del desarrollo de los pasos de resultados y detalle, entre otros, en los distintos productos del sitio, alineando esfuerzos en torno a una historia técnica común. La iniciativa no solo trajo múltiples beneficios en términos de unificación y eficiencia, sino que también representó una oportunidad concreta para ejercer liderazgo técnico desde un rol de desarrollador por aquel entonces semi senior.

A continuación, se detalla el contexto en el que surgió la propuesta, su diseño técnico, el proceso de coordinación y los aprendizajes obtenidos.

5.1. Contexto y motivación

Luego de un tiempo desde la primera implementación del programa de fidelidad, todas las aplicaciones front-end de la gerencia ya habían adoptado Angular como framework principal. Esta homogeneización tecnológica facilitó la colaboración entre equipos y la reutilización de componentes. Sin embargo, cada equipo había resuelto la integración del programa de forma independiente, lo que dio lugar a múltiples variantes en la implementación.

Un punto común era el uso del componente *loyalty-redemption-switch*, adoptado por todos los productos como parte central del flujo. No obstante, la forma en que se construían y conectaban los demás elementos —como el componente que indicaba la cantidad de puntos acumulados, el que mostraba los mensajes asociados al canje de puntos, o incluso la forma en que se compartía el modo en el que estaba el usuario (adquisición, acumulación o canje)— variaba considerablemente entre equipos. Esta falta de homogeneidad dificultaba la evolución del programa, ya que cada nuevo cambio requería un análisis y una implementación separados en cada equipo.

A partir de conversaciones técnicas con distintos equipos, quien escribe detectó esta situación y comenzó a preguntarse si era posible sentar una base técnica más sólida que simplificara el trabajo futuro. En particular, algunas ideas ya en evaluación para nuevas iteraciones del programa resultaban notoriamente más costosas bajo el esquema fragmentado vigente. Resolver esta deuda técnica no solo permitiría ordenar la implementación actual, sino también facilitar y abaratar los desarrollos futuros.

5.2. La propuesta: una historia técnica transversal

Con ese diagnóstico en mente, quien escribe propuso una historia técnica orientada a unificar la implementación de los componentes vinculados al programa. Lo más relevante de esta iniciativa no fue solamente la propuesta en sí, sino también el hecho de que quien la impulsó ocupaba, en ese momento, un rol de desarrollador semi senior. Pese a ello, logró convencer a los distintos equipos front-end necesarios de reservar un espacio de su capacidad para ejecutar esta mejora, articulando prioridades y consensos entre múltiples interlocutores.

La idea era construir una base común que permitiera a todos los equipos adoptar componentes reutilizables y una solución centralizada. Para lograrlo, se propuso:

- Implementar un componente encargado de mostrar la cantidad de puntos acumulados por el usuario (para reemplazar las implementaciones propias de cada equipo).
- Crear un componente encargado de mostrar el mensaje asociado al canje de puntos (nuevamente, para reemplazar las implementaciones propias de cada equipo).
- Desarrollar un wrapper de *loyalty-redemption-switch*, en lugar de reescribirlo, para adaptarlo al ecosistema Angular sin perder compatibilidad con otras aplicaciones (de otras gerencias) que no usaban dicho framework.
- Construir un servicio en Angular que unificara la manera en que se compartía, entre los distintos componentes, el modo de visualización del usuario (con o sin el modo canje activo).

A nivel de arquitectura, fue clave rediseñar el rol de la aplicación *s-commons-vr*. Hasta ese momento, dicha aplicación funcionaba únicamente como back-end de *loyalty-redemption-switch*. Sin embargo, a partir de esta mejora, *s-commons-vr* pasó a actuar también como fuente de todos los textos necesarios para los nuevos componentes. El flujo siguió iniciándose con un único request desde el *switch*, pero los datos obtenidos son distribuidos al resto de los componentes a través del servicio Angular previamente mencionado.

Este enfoque permitió lograr una centralización que redujo la dependencia entre equipos, facilitó el mantenimiento y allanó el camino para futuras extensiones del programa.

5.3. Arquitectura de la solución

La solución técnica propuesta se apoya en una arquitectura modular, diseñada para maximizar la reutilización de código y garantizar un comportamiento consistente en todas las aplicaciones Angular que integran el programa de fidelidad. A continuación, se detallan los principales componentes y servicios que forman parte de esta mejora, así como el nuevo rol asumido por *s-commons-vr* dentro del ecosistema.

5.3.1. Componentes Angular unificados

El primero de los elementos desarrollados fue un componente visual para mostrar los puntos acumulados por el usuario, que reemplazó a las múltiples implementaciones que existían en las distintas aplicaciones. Este componente debía ser lo suficientemente flexible como para integrarse en distintas secciones del sitio (por ejemplo, en los componentes *price-box*¹⁰ o *price-breakdown*¹¹), adaptándose a diferentes estilos y contextos sin comprometer la lógica principal.

En segundo lugar, se construyó un componente asociado al modo canje, cuyo objetivo es informar al usuario cuántos puntos está utilizando al activar dicho modo.

Un aspecto importante del diseño fue que ambos componentes debían ser consumibles de manera simple por los distintos equipos, sin requerir configuraciones complejas ni conocimiento profundo de la lógica interna. Esto requirió una arquitectura clara, con buenas interfaces y una adecuada separación de responsabilidades.

5.3.2. Wrapper del *loyalty-redemption-switch*

Dado que el *loyalty-redemption-switch* ya existía como componente compartido y era utilizado por aplicaciones construidas en otros frameworks además de Angular, se optó por no modificar su implementación original. En su lugar, se creó un *wrapper* Angular que encapsula el comportamiento del *switch*, permitiendo su inclusión en los nuevos flujos sin alterar su funcionamiento ni romper compatibilidades previas.

Este *wrapper* funciona como puente entre el ecosistema Angular y el componente original, exponiendo los eventos relevantes y facilitando la comunicación con el resto del sistema.

5.3.3. Servicio Angular de estado compartido

Una de las principales diferencias entre las implementaciones previas era la forma en que cada equipo gestionaba el modo del usuario (adquisición, acumulación o canje). Para unificar este comportamiento, se desarrolló un servicio en Angular que centraliza y expone el modo activo del usuario.

Este servicio funciona como única fuente de verdad para los componentes, permitiéndoles reaccionar automáticamente ante cambios en el estado y evitando la duplicación de lógica o la propagación incorrecta de valores.

5.3.4. Nuevo rol de *s-commons-vr* como back-end unificado

Un aspecto central de esta mejora fue redefinir la función de la aplicación *s-commons-vr*, que hasta ese momento se limitaba a servir como back-end del *loyalty-redemption-switch*. A partir de esta unificación, se amplió su responsabilidad: pasó a encargarse de devolver todos los textos relevantes para los distintos componentes del programa.

¹⁰ Componente visual encargado de presentar el precio de un ítem de manera simplificada.

¹¹ Componente visual encargado de presentar los distintos conceptos que conforman el precio final, junto con su valor asociado.

El flujo técnico se mantuvo simple: al instanciarse el componente *loyalty-redemption-switch*, se realiza un único request a *s-commons-vr*. La respuesta contiene, además de la información habitual, todos los textos necesarios para ser consumidos por los otros componentes. A través del servicio de estado, esos datos son distribuidos de forma controlada y eficiente al ecosistema de componentes.

Este diseño otorgó mayor control y autonomía al equipo responsable del funcionamiento del programa del lado front-end, eliminando la necesidad de coordinar con terceros para actualizar mensajes, manejar condiciones especiales o adaptar textos a nuevas reglas de negocio.

Antes de la implementación de esta mejora técnica, la respuesta de *s-commons-vr* era similar a la siguiente:

```
{
  ...
  "mode": "redemption",
  "title": "Pasaporte Despegar",
  "subtitle": "Usá tus 5.000 puntos",
  "tracking_info": { ... },
  ...
}
```

Luego del desarrollo, la respuesta pasó a ser similar a:

```
{
  ...
  "mode": "redemption",
  "loyalty_switch": {
    "title": "Pasaporte Despegar",
    "subtitle": "Usá tus 5.000 puntos",
    "tracking_info": { ... },
  },
  "loyalty_info": {
    "main_message": "Sumá {0} puntos",
    "description": "Con esta compra sumás {0} puntos que te
permitirán obtener descuentos en tu próximo viaje",
  },
  "redeeming_points": {
    "message": "usando {0} puntos",
  },
  ...
}
```

Al observar el modelo de respuesta de *s-commons-vr*, resulta evidente que los textos para los distintos componentes visuales del programa comenzaron a gestionarse desde el back-end. Es importante aclarar que, dado que estos textos se resuelven en un flujo distinto al de la cotización en puntos de cada ítem, en *s-commons-vr* deben contemplarse casos en

los que ciertos textos —como los correspondientes a los componentes *loyalty-info* y *redeeming-points*— incluyan *placeholders*¹². Estos *placeholders* son luego completados en el front-end, una vez que se dispone de la información necesaria.

A modo de cierre, se presenta la Figura 5.1, en la que se visualizan los distintos componentes que forman parte de la solución propuesta y cómo interactúan entre sí dentro del flujo.

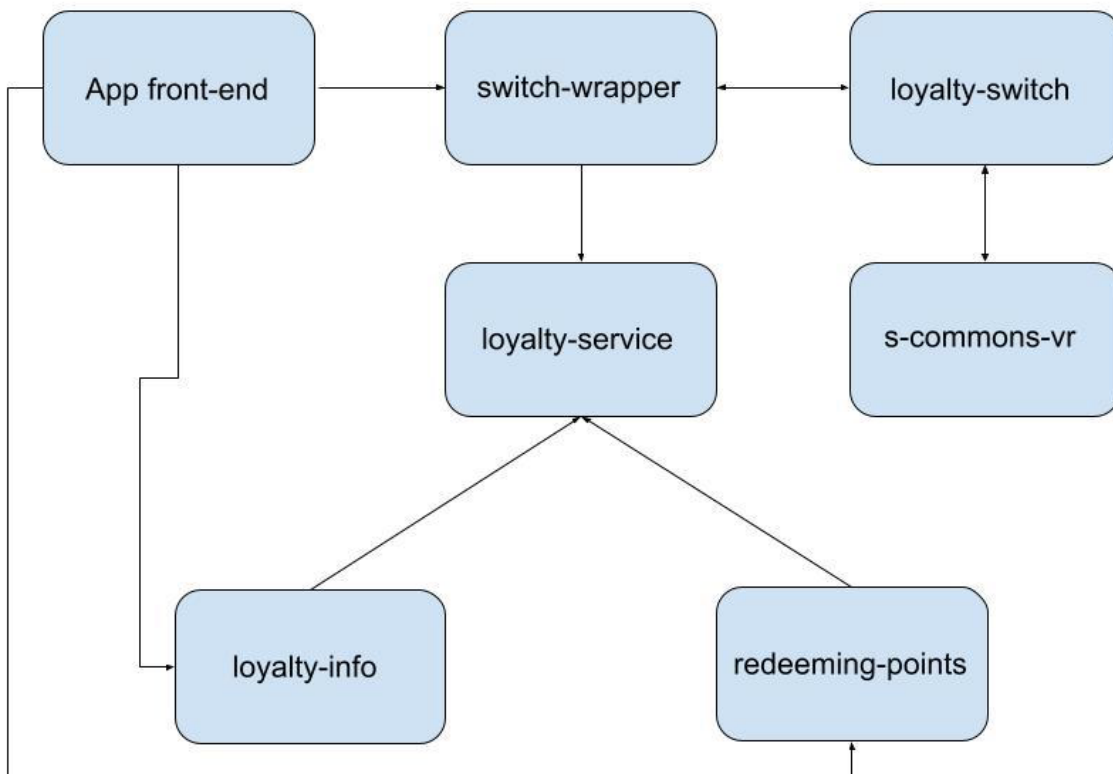


Figura 5.1. Interacción entre los distintos componentes que forman parte de la solución

5.4. Coordinación y ejecución

Más allá de los aspectos técnicos, uno de los mayores desafíos de esta iniciativa fue la **coordinación entre equipos**. La propuesta requería no solo el desarrollo de nuevos componentes, sino también la validación de su adopción por parte de los distintos equipos considerados dentro del alcance, cada uno con sus propios objetivos, tiempos y prioridades de negocio. Lograr ese alineamiento, especialmente desde un rol de desarrollador semi senior, implicó asumir un liderazgo informal que resultó clave para el éxito de la mejora.

El primer paso fue presentar la propuesta técnica, explicando en distintos espacios los beneficios concretos de la unificación: reducción de duplicación de código, menor

¹² Referencia dentro de un texto que se reemplaza por un valor dinámico en tiempo de ejecución (por ejemplo, {0}).

mantenimiento a futuro, mayor control sobre los textos mostrados de forma centralizada y una integración más fluida entre los componentes del programa. Esta etapa fue fundamental para obtener el compromiso de los equipos involucrados, ya que permitió explicar no solo el *qué*, sino sobre todo el *por qué* detrás del esfuerzo.

Una vez obtenida la aprobación general, se avanzó con la planificación conjunta de la implementación. Cada equipo se comprometió a reservar parte de su capacidad dentro de los siguientes meses para colaborar con esta iniciativa.

La ejecución se organizó en etapas:

1. **Desarrollo del ecosistema técnico** por parte del equipo de quien escribe, incluyendo los nuevos componentes, el *wrapper* del *switch*, el servicio de estado y los ajustes en *s-commons-vr*.
2. **Publicación de documentación técnica y ejemplos de integración**, para facilitar la adopción por parte de otros equipos.
3. **Soporte durante la implementación a los distintos desarrolladores**, resolviendo dudas y proponiendo mejoras.
4. **Validación funcional y visual**, asegurando que los componentes se comportaran de manera consistente en las distintas aplicaciones que los incorporaban.

La estrategia adoptada permitió avanzar con un enfoque progresivo pero coordinado: cada equipo fue incorporando los componentes en su propio ritmo, pero sobre una base técnica común y con soporte activo. Esto evitó bloqueos y facilitó la transición sin necesidad de aplicar una migración forzada o disruptiva.

A lo largo del proceso, quien escribe se mantuvo como punto de contacto central, resolviendo cuestiones técnicas, recogiendo feedback y ajustando detalles cuando fue necesario. Esta participación activa no solo permitió mantener el alineamiento, sino que también sirvió para construir confianza entre equipos.

5.5. Impacto y aprendizajes

La unificación de componentes Angular vinculados al programa de fidelidad tiene un impacto tangible en distintos niveles. Desde el punto de vista técnico, el ecosistema resultante es **más robusto, mantenible y coherente**, permitiendo a los equipos trabajar sobre una base común, reducir errores y acelerar nuevas implementaciones. La posibilidad de reutilizar componentes y centralizar parte de la lógica y configuración evita desarrollos paralelos y contribuye a mantener una experiencia de usuario consistente en todo el sitio.

En términos de eficiencia, esta mejora tiene un impacto directo en el esfuerzo requerido para implementar nuevas funcionalidades o adaptar reglas de negocio. Ya no es necesario coordinar múltiples implementaciones independientes ni alinear cambios en paralelo entre distintos equipos, salvo —por supuesto— que el cambio sea lo suficientemente grande como para replantear la arquitectura, o que requiera algún parámetro adicional. En cambio,

en la mayoría de los casos, alcanza con actualizar los componentes compartidos o modificar la respuesta de *s-commons-vr* para que los cambios se propaguen correctamente.

Desde una perspectiva personal, esta iniciativa marcó un punto de inflexión en el recorrido profesional de quien escribe. Si bien ya había participado en historias técnicas complejas (por ejemplo, la implementación original de este programa), fue la primera vez que asumió la coordinación transversal de una mejora con impacto directo en varios equipos. Haber impulsado esta propuesta desde un rol de desarrollador semi senior y lograr su implementación efectiva permitió desarrollar nuevas habilidades de liderazgo técnico y de articulación con otras áreas.

En resumen, la unificación de componentes Angular no solo resuelve una deuda técnica importante, sino que mejora la capacidad de evolución del programa de fidelidad y reduce el esfuerzo necesario para sostenerlo y ampliarlo en el tiempo.

Capítulo 6. Adaptación del programa de puntos al modelo de white labels

La implementación en Brasil del programa de fidelidad de Despegar representó el primer gran hito del proyecto. Las posteriores expansiones del programa a otros países también fueron hitos muy importantes a nivel negocio. Sin embargo, para lograr un impacto mayor, era necesario extender esa lógica a otras marcas que operan bajo el modelo de *white label*. Estas marcas —muchas de ellas bancos que ofrecen sistemas de fidelidad propios a sus clientes— no solo representan un volumen importante de tráfico y ventas, sino también un conjunto de requerimientos específicos a nivel visual, comunicacional y funcional.

En este capítulo se describe cómo se reutilizó la base técnica desarrollada previamente para integrar el programa de puntos en *white labels*, manteniendo la robustez del modelo original, pero adaptándolo a las particularidades de cada marca. Se detallan los distintos modos de funcionamiento soportados, la configuración centralizada por marca, los ajustes en la experiencia de usuario, y el rol de los componentes visuales en este nuevo contexto. También se destaca la importancia que tuvo el trabajo de estandarización comentado en el capítulo anterior para habilitar este tipo de integraciones de forma ágil y escalable.

6.1. Particularidades del modelo white label

El modelo de *white label* permite a marcas externas operar un sitio bajo su identidad visual y comunicacional, utilizando la infraestructura técnica de Despegar. En este esquema, cada marca puede definir parte del comportamiento del sitio, adaptando elementos visuales, textos y reglas de negocio.

Este modelo es especialmente relevante en el caso de los bancos, quienes integran sus programas de fidelidad con el sitio para que sus clientes puedan canjear puntos acumulados (por ejemplo, por uso de tarjetas de crédito) por viajes. Este tipo de alianzas demanda un cierto nivel de personalización y control, sin resignar la eficiencia operativa que brinda una solución técnica centralizada.

6.2. Reutilización de la implementación base

La integración del programa de puntos en *white labels* se apoyó en los mismos componentes funcionales, servicios back-end y estructuras de front-end diseñadas para el programa de fidelidad de Despegar. Gracias al trabajo previo de estandarización, fue posible extender el alcance del programa sin necesidad de construir variantes específicas para cada marca.

Esta reutilización incluyó tanto la lógica de cotización en puntos, acumulación y canje como también los elementos visuales. El objetivo fue mantener una única base técnica que pudiera adaptarse a múltiples entornos a través de configuración, reduciendo así la complejidad y los tiempos de implementación.

6.3. Configuración centralizada de comportamiento

A diferencia de la página de Despegar, donde el modo de visualización del programa se delega al usuario (ya que, por ejemplo, puede decidir habilitar el *modo canje* o permanecer en *modo acumulación*), en las *white labels* se utilizan **modos de funcionamiento predefinidos**. Algunas marcas permiten solo acumulación, otras solo canje, y otras admiten ambos.

Estas definiciones se gestionan a través de un **servicio centralizado de configuración de white labels**, utilizado por diversos equipos dentro de la compañía. Este servicio no solo indica el modo operativo de cada marca, sino también otras variables que afectan directamente el comportamiento de los componentes, como la visibilidad de ciertos elementos o los textos a mostrar.

Esta capa de configuración fue clave para mantener un sistema genérico, donde la incorporación de una nueva marca no implicara esfuerzos de desarrollo, sino simplemente un ajuste de parámetros.

6.4. El componente “switch” en white labels

El componente *switch*, que en el sitio de Despegar cumple la función de ofrecer al usuario la posibilidad de canjear puntos (o no canjearlos), **no forma parte de la experiencia visual en las white labels**. En estos sitios, el componente *loyalty-redemption-switch* es inicializado por las aplicaciones front-end al igual que en el sitio principal —realizando la consulta correspondiente a *s-commons-vr* y enviando la respuesta al servicio compartido—, pero **no se muestra de forma visible al usuario** en ninguna de las marcas asociadas.

La decisión de ocultar este componente responde tanto a motivos de simplicidad como a lineamientos definidos por los socios comerciales, que suelen preferir una experiencia más directa y basada en reglas fijas. Desde el punto de vista técnico, se utiliza exactamente el mismo componente, simplemente adaptado para operar de manera invisible como inicializador del aspecto visual del programa.

6.5. Personalización por marca

Uno de los aspectos más importantes en la adaptación del programa de puntos a *white labels* fue la **personalización del contenido textual**. Cada marca puede definir su propia terminología para el programa, así como los mensajes asociados a los distintos modos en los que puede estar el usuario (*acumulación* o *canje*).

Para soportar esta flexibilidad, el servicio *s-commons-vr* obtiene los textos desde el **mismo servicio de configuración centralizado** mencionado anteriormente. Este servicio actúa como fuente de verdad para determinar qué mensajes deben mostrarse en cada *white label*, permitiendo ajustar la experiencia comunicacional sin necesidad de modificar los componentes.

Los textos pueden abarcar desde el nombre del programa hasta el uso de terminologías específicas (por ejemplo, algunas marcas llaman “puntos” al concepto que acumula el usuario, y otras lo llaman “millas”), adaptándose a las condiciones y tono de cada marca.

También es importante destacar que, de la misma forma en que se personalizan los textos, se puede personalizar la **iconografía**: se define en el servicio de configuración centralizado un link a una imagen con el ícono deseado por la marca, y esto termina impactando en los distintos componentes del programa. Esta capacidad agrega un mayor grado de personalización visual.

6.6. Beneficios de la mejora técnica previa

La mejora técnica descrita en el capítulo anterior fue clave para hacer viable esta personalización sin demandar un esfuerzo considerable de muchos equipos. Gracias a la estandarización de los componentes Angular vinculados al programa de puntos y al diseño de un esquema claro de comunicación entre ellos, fue posible **centralizar la lógica de obtención de textos** en el servicio *s-commons-vr*.

Tal como se explicó previamente, este servicio responde al request del componente *loyalty-redemption-switch* con todos los textos necesarios para los distintos componentes visuales. A su vez, cada componente extrae de esa respuesta únicamente los textos que le corresponden, sin necesidad de hacer llamadas independientes.

Esta estrategia evitó la necesidad de mantener múltiples versiones de cada componente para cada marca, redujo errores, y permitió escalar el programa a nuevas *white labels* de forma rápida y controlada. Una nueva marca puede sumarse simplemente agregando entradas específicas en el servicio de configuración, sin alterar el código de los componentes ni romper la experiencia del resto de las marcas ya integradas.

6.7. Comparativa entre el sitio de Despegar y las white labels

Para visualizar las diferencias más relevantes entre la implementación del programa de puntos en el dominio principal de Despegar y su adaptación al modelo de *white labels*, se presenta a continuación un cuadro comparativo con los principales aspectos técnicos y funcionales:

Aspecto	Despegar	White Labels
Visibilidad del switch	Visible.	Oculto.
Modos de funcionamiento	Dinámico. El usuario elige si quiere canjear sus puntos o no.	Estáticos. Se configuran por marca.
Textos del programa	Centralizados en <i>s-commons-vr</i> .	Personalizados por marca en el servicio centralizado de <i>white labels</i> . <i>s-commons-vr</i> los obtiene desde ahí.
Iconografía	Unificada en todo el sitio.	Unificada en todo el sitio y personalizable por marca.

6.8. Ejemplo visual de personalización en white labels

Se incluyen a continuación ejemplos visuales que muestran cómo se adapta la implementación del programa de puntos en *white labels*, junto con una comparación directa con el sitio de Despegar. Esto permite visualizar de forma concreta los conceptos desarrollados en este capítulo.

The screenshot displays the Despegar website's search results for accommodations in Miami, United States. The search criteria are: Miami, Estados Unidos, from Saturday, July 5th to Sunday, July 8th, for 3 nights and 2 people. The results are sorted by 'Más elegidos'. The featured hotel is Tru by Hilton Miami West Brickell, located in Miami, Little Havana, 3.95 km from the center. The hotel has a 9.2 rating, 4 stars, and offers amenities like Wi-Fi and breakfast. The price for 3 nights for 2 people is \$353,509, which can be reduced to \$353,509 using 17,810 points. The card also indicates that there is no need to pay a perception fee (RG5617) and that the price includes breakfast. A 'Ver detalle' button is visible at the bottom right of the card.

Figura 6.1. Programa de fidelidad en el sitio de Despegar

Como se observa en la Figura 6.1, el sitio de Despegar presenta al usuario el componente *loyalty-redemption-switch* (identificado con el texto “Usá tus puntos”). Al activarlo, se habilita el modo canje, lo que hace que cada ítem del listado muestre su precio con el canje aplicado. En la misma imagen también puede verse el componente *redeeming-points* (correspondiente al texto “usando 17.810 puntos”), que indica cuántos puntos se están utilizando en la reserva. Cabe destacar que, en este sitio, no se muestra la cantidad de puntos a acumular en la página de *resultados* de Alojamientos.

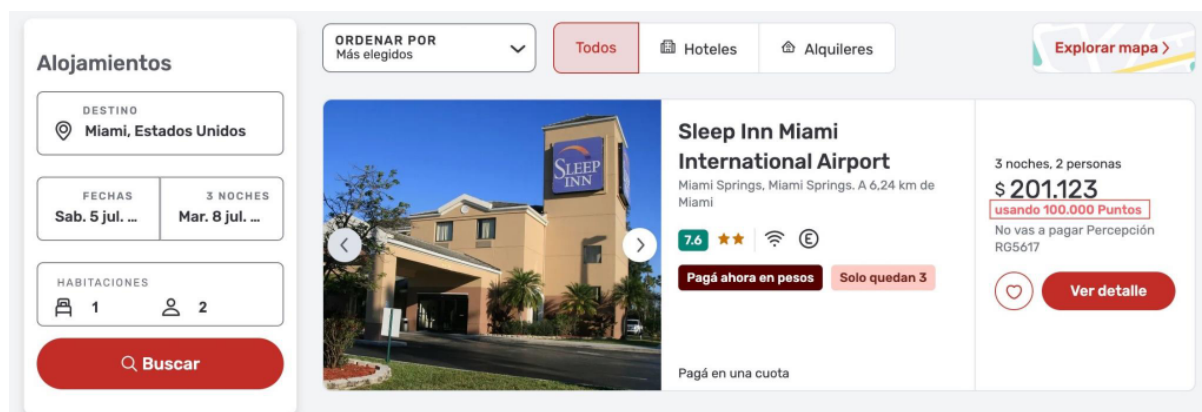


Figura 6.2. Programa de fidelidad en una white label de canje

En la Figura 6.2 también se muestra la página de *resultados* de Alojamientos, pero en este caso correspondiente a una *white label* configurada para el **modo canje de puntos**. Como se mencionó anteriormente, el componente *loyalty-redemption-switch* no está disponible, ya que el modo se define por configuración. En este escenario, el usuario está siempre en *modo canje*, motivo por el cual los precios se muestran con el descuento aplicado y vuelve a aparecer en pantalla el componente *redeeming-points*.

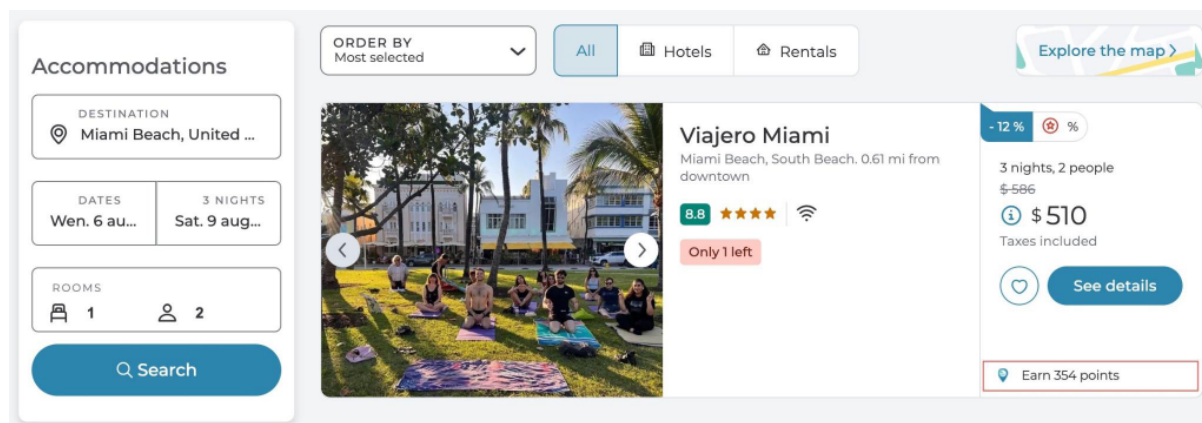


Figura 6.3. Programa de fidelidad en una white label de acumulación

Por último, en la Figura 6.3 se muestra la página de *resultados* de Alojamientos para una *white label* configurada en **modo acumulación de puntos**. Al igual que en el caso anterior, el componente *loyalty-redemption-switch* no está disponible, ya que el modo se define por configuración y permanece en acumulación. A diferencia de lo que se muestra en las Figuras 6.1 y 6.2, en este caso se aparece el componente *loyalty-info*, el cual muestra el

texto *“Earn 354 points”*, que indica la cantidad de puntos que se acumularían en caso de concretar la compra del ítem.

En conjunto, estas capturas ilustran cómo el programa de fidelidad se adapta visual y funcionalmente según la configuración de cada marca. Ya sea permitiendo la interacción del usuario o definiendo el modo por defecto, la experiencia se mantiene consistente y alineada a las necesidades de cada contexto.

Capítulo 7. Conclusiones y próximos pasos

La participación en las distintas etapas del programa de fidelidad fue, sin lugar a dudas, una de las experiencias más enriquecedoras en la carrera profesional de quien escribe. Desde las primeras implementaciones hasta las optimizaciones más recientes, este proyecto no solo permitió desarrollar habilidades técnicas, sino también crecer en términos de liderazgo, visibilidad y entendimiento del negocio.

Con el paso del tiempo, el rol de quien escribe fue evolucionando: comenzó como desarrollador junior, luego fue asumiendo más responsabilidades tanto en los desarrollos vinculados al programa como también en muchos otros desarrollos importantes, y actualmente se desempeña como Engineer Manager del equipo encargado del front-end de Alojamientos. Esta oportunidad representó un punto de inflexión en su recorrido profesional, y considera haberla aprovechado al máximo.

Participar con un rol tan activo en este proyecto lo llevó, incluso, a replantearse su camino profesional. Fue precisamente al involucrarse en las iteraciones más importantes del programa —liderando historias, coordinando esfuerzos técnicos entre verticales y acompañando la definición con UX y Producto— que surgió con claridad el deseo de convertirse en manager. El hecho de tener desde etapas tempranas un rol de coordinación, asegurar que las estimaciones fueran realistas, y guiar al equipo hacia una solución consistente, resultó tan interesante como desafiante. Con el tiempo, esta motivación se tradujo en una búsqueda consciente por tomar la iniciativa en la coordinación de historias dentro del equipo, estuviera o no involucrado directamente en el desarrollo.

También fue fundamental la progresiva cercanía con otros equipos. En las primeras etapas, quien escribe participaba de reuniones con UX o Producto acompañado por el manager del equipo, pero con el correr del tiempo fue ganando autonomía en estos espacios, contribuyendo cada vez más en las definiciones funcionales y técnicas de la solución. Este involucramiento más transversal, que va más allá del código, también marcó un cambio significativo en su perfil.

Desde el punto de vista técnico, el impacto del programa también fue considerable. El refactoring mencionado en el capítulo 5 continúa vigente y permitió no solo los desarrollos mencionados en este trabajo (como la arquitectura de *white labels*), sino también otras integraciones de menor alcance que no fueron incluidas en esta tesina. Esta base técnica común redujo significativamente los tiempos de desarrollo y facilitó la estandarización del producto. De hecho, el equipo de Producto de Loyalty valoró especialmente esta unificación, ya que permitió implementar mejoras de forma más ágil y con menor esfuerzo técnico en cada vertical.

Actualmente, el programa de fidelidad se encuentra activo para la marca Despegar en Brasil, Argentina, México, Colombia, Ecuador y Uruguay. También está integrado en muchas de las *white labels* disponibles en la plataforma, lo cual refuerza su importancia y alcance dentro del ecosistema.

En cuanto a tecnología, se continúa trabajando con Angular como framework principal del front-end y se sigue promoviendo el uso de componentes compartidos. Uno de los hitos destacables fue la actualización a Node.js 20 en el servicio *s-commons-vr*, lo que generó mejoras notables en el rendimiento y los tiempos de respuesta del servicio (Node.js, 2023).

En su rol actual como manager, quien escribe ya no programa de forma constante, pero mantiene el compromiso de que Loyalty siga siendo un espacio de oportunidad para el equipo. Se busca reducir dependencias entre equipos, mejorar la autonomía técnica y mantener la evolución del programa. Uno de los desafíos actuales está en el modelo de distribución de los componentes: al instalarse como dependencias, es necesario coordinar las versiones entre los equipos, lo cual en ocasiones ha requerido mantener *s-commons-vr* con modelos retrocompatibles.

Frente a este problema, se está considerando una estrategia diferente: comenzar a servir el *bundle* estático de algunos componentes —en particular *loyalty-redemption-switch*— directamente desde *s-commons-vr*. Esto permitiría evitar la necesidad de actualizaciones manuales por parte de los equipos integradores, a la vez que facilita la adopción de mejoras o nuevas funcionalidades. Otros componentes comunes a distintos equipos también se distribuyen de esta manera, como los desarrollados por el equipo *frontend-components*, mencionado en el capítulo 3. Solo sería necesaria la participación de otros equipos en el caso de requerir nuevos parámetros mandatorios.

Otra posible evolución es incorporar a *s-commons-vr* dentro del camino crítico de las aplicaciones. En el modelo actual, el request a *s-commons-vr* y el de *disponibilidad* (por ejemplo, para buscar alojamientos para fechas determinadas) se realizan en paralelo. Esto impide aplicar reglas de negocio basadas en el *modo* que está utilizando el usuario en los back-ends asociados a cada producto. Se está evaluando una reestructuración para que las aplicaciones consulten primero a *s-commons-vr* y luego pasen su respuesta como parámetro al componente *loyalty-redemption-switch*, de modo que este pueda evitar hacer una nueva consulta para buscar dicha información. Esta modificación permitiría aplicar reglas más complejas en las capas de back-end.

A continuación, en la Figura 7.1, se muestra la diferencia entre el modelo actual (a la izquierda) y la propuesta que incorpora a *s-commons-vr* dentro del camino crítico (a la derecha). Si bien esta modificación puede percibirse como un avance en cuanto a las posibilidades de configuración que habilita, también implica que el tiempo de espera hasta obtener los resultados del servicio de disponibilidad será ligeramente mayor. No obstante, dado que *s-commons-vr* es una aplicación liviana, con una operatoria simple y sin consumo intensivo de otros servicios, su tiempo de respuesta es muy bajo (alrededor de 25 ms); por lo tanto, incorporarlo al camino crítico no representa un inconveniente significativo.

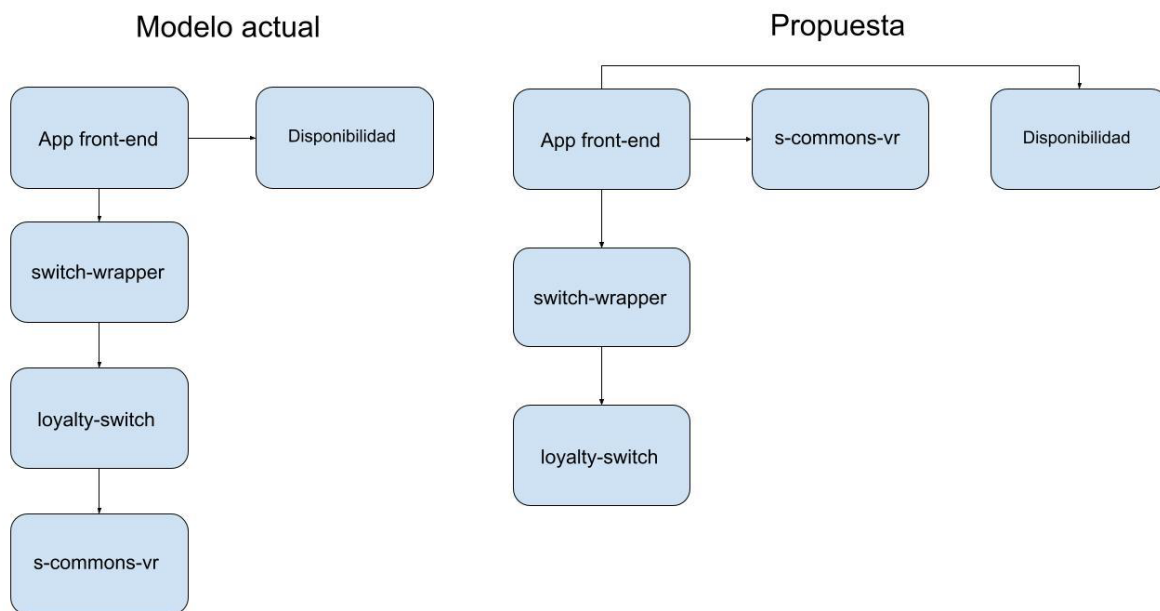


Figura 7.1. Flujo propuesto para incluir a *s-commons-vr* en el camino crítico

Esta necesidad se evidenció durante una de las pruebas realizadas recientemente, en la que se forzó el modo canje por defecto para todos los usuarios, incluso si no habían utilizado el switch. El objetivo era que vieran precios más bajos desde el inicio, con la expectativa de mejorar la conversión. Sin embargo, el A/B test no mostró un impacto positivo en las ventas y la prueba fue desactivada. Al analizar la experiencia de usuario durante ese período, se detectaron inconsistencias de estado entre componentes, lo que reforzó la necesidad de avanzar con el cambio arquitectónico mencionado anteriormente y podría explicar el resultado desfavorable del experimento.

Finalmente, si bien el recorrido fue exitoso en muchos sentidos, también dejó aprendizajes para el futuro. Una decisión que quien escribe considera que podría haberse abordado de otra forma fue la elección tecnológica para los componentes compartidos. Tal como se menciona en el capítulo 4, utilizar Angular fue razonable en el contexto original, pero hoy —con mayor madurez técnica— tal vez sería más conveniente considerar opciones como Stencil, que permiten construir componentes web reutilizables sin acoplarse a un framework específico. Esta decisión habría facilitado una mayor escalabilidad de la solución, evitando forzar dependencias entre equipos con stacks tecnológicos distintos.

En definitiva, el proyecto de Loyalty no solo consolidó una funcionalidad de negocio clave para la empresa, sino que se convirtió en un espacio de aprendizaje constante, con impacto transversal y desafíos técnicos de alto nivel. La evolución del programa continúa, y con esta evolución también se renuevan las oportunidades para seguir creciendo, tanto a nivel individual como colectivo.

Referencias bibliográficas

- Angular. (s. f.). *What is Angular*. Recuperado el 4 de mayo de 2025, de <https://angular.io/guide/what-is-angular>
- Express. (s. f.). *Express - Node.js web application framework*. Recuperado el 2 de junio de 2025, de <https://expressjs.com/>
- Ford, N., Richards, M., Sadalage, P. J., & Dehghani, Z. (2021). *Software architecture: The hard parts*. O'Reilly Media.
- Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Professional.
- HubSpot. (2024). *Qué son los programas de fidelización y 17 ejemplos*. Recuperado el 8 de marzo de 2025, de <https://blog.hubspot.es/service/que-son-los-programas-de-fidelizacion>
- MDN Web Docs. (s. f.). *JavaScript Guide*. Recuperado el 3 de junio de 2025, de <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>
- Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media.
- Node.js. (2023). *Node v20.0.0 (Current)*. Recuperado el 1 de junio de 2025, de <https://nodejs.org/es/blog/release/v20.0.0/>
- Node.js. (s. f.). *About Node.js*. Recuperado el 3 de junio de 2025, de <https://nodejs.org/en/about>
- npm. (s. f.). *About npm*. Recuperado el 2 de junio de 2025, de <https://docs.npmjs.com/about-npm>
- Optimizely. (s. f.). *What is A/B testing? With examples*. Recuperado el 8 de marzo de 2025, de <https://www.optimizely.com/optimization-glossary/ab-testing/>
- Oracle. (s. f.). *About the Java technology*. Recuperado el 4 de mayo de 2025, de <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>
- Pretii. (2024). *Estadísticas de fidelización de clientes en 2024*. Recuperado el 8 de marzo de 2025, de <https://pretii.lat/articulo/estadisticas-fidelizacion-clientes-2024>
- React. (s. f.). *Add React to an existing project*. Recuperado el 4 de mayo de 2025, de <https://react.dev/learn/add-react-to-an-existing-project>
- Reichheld, F. F., & Scheffer, P. (2000). *E-loyalty: Your secret weapon on the web*. Harvard Business Review. Recuperado el 4 de marzo de 2025, de <https://hbr.org/2000/07/e-loyalty-your-secret-weapon-on-the-web>

Richards, M., & Ford, N. (2020). *Fundamentals of software architecture: An engineering approach*. O'Reilly Media.

Royce, W. W. (1970). *Managing the development of large software systems*. Proceedings of IEEE WESCON, 1–9

Scala. (s. f.). *A tour of Scala*. Recuperado el 4 de mayo de 2025, de <https://docs.scala-lang.org/tour/tour-of-scala.html>

Semrush. (2024). *65 estadísticas de retención de clientes que debes conocer en 2024*. Recuperado el 8 de marzo de 2025, de <https://es.semrush.com/blog/estadisticas-de-retencion-de-clientes/>

Stencil. (s. f.). *Stencil*. Recuperado el 2 de junio de 2025, de <https://stenciljs.com/>

Webpack. (2024). *Concepts*. Recuperado el 4 de junio de 2025, de <https://webpack.js.org/concepts/>