eBook

DE LA IDEA AL MERCADO

El Manual Definitivo para Fundadores y Co-Fundadores.
Cómo transformar tu visión en un producto digital sin quemar tu presupuesto en el intento.

devias.ar

devias

Índice

Parte 2: Construcción – Decisiones Clave

4. ¿Herramientas sin código o código a medida? *¿No-code o Custom Code?*
    - Tabla comparativa: costos, velocidad, flexibilidad
    - Cuándo usar Bubble/Webflow vs. cuándo contratar desarrolladores
    - Señales de alerta al elegir tecnología
5. Equipos y Presupuestos
    - Freelancers vs. Agencias vs. Equipo Interno: Pros y Contras
    - Costos reales (desglose por etapas)
    - Cómo negociar con proveedores técnicos (preguntas clave)
6. El Proceso de Desarrollo
    - Las 4 etapas del desarrollo de software: diseño → desarrollo → testing → lanzamiento
    - Cómo dar seguimiento **SIN TENER CONOCIMIENTOS TÉCNICOS**

Parte 3: Lanzamiento

7. Marketing desde el Día Cero
    - Listas de espera y pruebas cerradas (ejemplo: Dropbox)
    - Estrategias de crecimiento (*growth hacking*) con bajo presupuesto: viralidad y alianzas estratégicas
8. Métricas Post-Lanzamiento
    - Qué medir (y qué ignorar): retención, CAC, churn
    - Iteración basada en feedback real
9. Conclusión: Liderazgo sin conocimientos técnicos
    - Tu rol: visión, gestión y aprendizaje continuo

Introducción

Tenés una gran idea, una visión clara de cómo resolver un problema en el mercado y la pasión para llevarla adelante. Pero una sensación de abrumo recorre tu cuerpo y las preguntas no se hacen esperar:

" ***¿Por dónde arranco? ¿Son muchas cosas? ¿Estaré olvidando algo clave?"***

Tranquilo, esa sensación es completamente normal. Aparece cuando intuís que hay una gran distancia entre tener una idea genial y convertirla en un producto digital real. Especialmente si no venís del mundo tecnológico.

El objetivo de este libro es ayudarte a ordenar ese camino, darte algo de certidumbre y ofrecerte herramientas para que tomes decisiones lo más informado posible en un mundo —el desarrollo tecnológico— que seguramente escapa a tu zona de expertise, lo cual es absolutamente natural. Y, además, la idea es que disfrutes este hermoso proceso de construir algo desde cero.

¿Por qué esto importa tanto?

Porque tomar decisiones a ciegas en tecnología puede salir MUY CARO.

Según CB Insights, el 42% de las startups fracasa porque construyen productos que nadie quiere. No porque la idea fuera mala, sino porque no la validaron bien antes de desarrollarla.

Otro 23% se cae por errores en el equipo o por falta de conocimientos técnicos. Es decir, no es un problema de ganas ni de esfuerzo: es un problema **de enfoque**.

Pero aunque las cifras suenen alarmantes, no te preocupes, éste libro existe para que no seas parte de esas estadísticas.

Acá no vas a encontrar teoría abstracta ni consejos genéricos. Esto es un manual práctico para:

✅ Evitar los errores más costosos que cometen los founders al desarrollar tecnología.
✅ Tomar decisiones informadas sobre qué construir, con quién y cómo.
✅ Acelerar tu camino al mercado sin perder meses (o años) en direcciones equivocadas.

¿Y por qué es tan fácil equivocarse?

Porque la mayoría de los fundadores repite un patrón: **una gran idea, pero una ejecución desordenada.**

Vamos a brindarte dos ejemplos que resumen bien este problema:

🧨 **Caso 1: El MVP que nadie usó**

Situación: Un fundador invirtió $50.000 USD en una app de fitness personalizada.

Error: Nunca validó si los usuarios pagarían por ella.

Resultado: La lanzó… y tuvo solo 23 descargas en 3 meses.

😵 **Caso 2: La pesadilla del desarrollo infinito**

Situación: Un fundador contrató freelancers/equipo propio/agencia para construir su plataforma.

Error: Nadie definió bien el alcance.

Resultado: Ocho meses después, el código era un lío y tuvieron que arrancar de cero.

¿Por qué pasa esto?

- Falta de validación temprana: asumir que “si lo construís, vendrán”.
- Desconocimiento del proceso tecnológico: no saber qué preguntar ni cómo medir avances.
- Mala gestión de recursos: invertir en funcionalidades innecesarias antes de testear el corazón del producto.

La buena noticia es: Todo esto se puede evitar si contás con el mapa correcto.

¿Qué vas a aprender y cómo aplicarlo?

Este libro es un atajo para transformar tu idea en un producto viable sin caer en los errores típicos: desarrollar algo que nadie usa, gastar de más en funcionalidades innecesarias, o perder meses en direcciones equivocadas.

Acá te mostramos el camino que vas a seguir:

1. **Validación antes del desarrollo**

   Cómo probar tu idea sin escribir una sola línea de código.

   Métodos rápidos y baratos para confirmar demanda real (ej: landing pages, prototipos en Figma).

2. **El proceso de desarrollo, desglosado**

   Las 4 etapas clave (diseño, desarrollo, testing, lanzamiento) y qué esperar en cada una.

3. **Costos realistas y cómo optimizarlos**

   Rangos de inversión para Producto Mínimo Viable (MVP) básico vs. producto escalable.

   ¿Freelancers, agencia o equipo interno? Pros, contras y cómo decidir.

4. **Decisiones técnicas críticas**

   No-code vs. desarrollo personalizado: cuándo usar cada uno.

   Elegir tecnologías (frontend, backend, bases de datos) sin morir en el intento.

5. **Lanzamiento y crecimiento**

   Métricas clave para medir el éxito desde el día 1.

   Estrategias de *growth hacking* para startups con bajo presupuesto.

Este no es un libro de teoría. Es una **GUÍA DE SUPERVIVENCIA** para llevar tu idea al mercado de manera eficiente, profesional y sin perder dinero en el camino.

¿Listo para empezar? ¡Vamos!

Capítulo 1: Validación y Preparación

La diferencia entre una idea brillante y un producto viable está en la validación temprana.

1.1 ¿Realmente necesitás tecnología?

Uno de los errores más comunes entre los fundadores es asumir que toda gran idea necesita una app móvil o una plataforma digital desde el inicio. Pero muchas veces, el primer paso no es programar, sino buscar formas más simples y rápidas de validar.

Preguntas clave para desafiar la idea:

- ¿Se puede resolver el problema usando herramientas que ya existen?
- ¿Se puede testear la solución con algo manual antes de construir tecnología?
- ¿El valor está en la tecnología o en la experiencia que se brinda?

Te damos un ejemplo: Una persona quiere lanzar un marketplace. Antes de contratar desarrolladores, arma un grupo de WhatsApp y usa PayPal para gestionar pagos. Así valida si la gente realmente está dispuesta a usarlo.

Caso real: *Gumroad*, la plataforma de venta digital, comenzó como una simple página con botón de PayPal. La demanda validó el concepto antes de construir algo más complejo.

1.2 Validando una idea sin invertir en desarrollo

La validación es el puente entre *intuición* y *evidencia*. A continuación, tres métodos simples y eficaces que cualquier fundador puede aplicar, incluso sin saber programar:

Método 1: El "Fake Door Test"

Pero ¿Qué es el *fake door test*? Simple: una forma de medir el interés de posibles usuarios antes de construir nada.

¿Cómo hacerlo?

- Crear una página web muy simple (puede ser con herramientas como Carrd o Unbounce).
- Escribir un título atractivo, una breve descripción del producto y agregar un botón que diga "Registrarse" o "Quiero probarlo".
- Cuando alguien hace clic, se muestra un mensaje del tipo: “Gracias por tu interés, estamos trabajando para lanzarlo pronto”.
- Lo importante es contar cuántas personas hacen clic.

¿Por qué funciona? Porque permite medir interés real sin desarrollar el producto.

Caso real: *Buffer*, una herramienta para gestionar redes sociales, usó este método. Descubrieron que la mayoría de las personas abandonaba al ver los precios. Ajustaron su estrategia antes de invertir en desarrollo.

Método 2: Prototipos interactivos

¿Qué son los prototipos interactivos? Son simulaciones visuales del producto que parecen una app real, pero no tienen funcionalidad. Sirven para mostrar cómo se usaría la solución.

¿Cómo hacerlos?

- Usar herramientas como Figma, Marvel o incluso PowerPoint.
- Crear pantallas básicas: inicio, registro, flujo de uso.
- Conectar las pantallas con enlaces para simular navegación.

¿Qué observar al testear?

- ¿La persona entiende cómo usarlo sin explicaciones?
- ¿Dónde se traba o se pierde?
- ¿Qué preguntas surgen?

Herramientas útiles: Maze o useberry, para hacer tests remotos con usuarios y obtener métricas.

Caso real: Cuando aún no existía el producto, *Dropbox* creó un video demo (una especie de prototipo simulado) para mostrar cómo funcionaría el sistema de sincronización de archivos. Ese video generó decenas de miles de inscripciones a la lista de espera y validó la idea antes de desarrollar la tecnología real, que era compleja.

Método 3: Venta manual ("Wizard of Oz")

¿Qué es? Una simulación en la que el producto está funcionando automáticamente, pero en realidad todo se hace de forma manual. Quien lo usa no lo sabe y recibe una experiencia igual a la real.

¿Cómo aplicarlo?

- Ofrecer el servicio como si ya estuviera listo.
- Cuando alguien hace un pedido o interacción, se resuelve a mano (enviar correos, buscar datos, etc.).

Caso real: *Zappos*, el e-commerce de zapatos, comenzó así. El fundador sacaba fotos a los zapatos en tiendas físicas. Cuando alguien hacía un pedido online, él mismo iba, compraba y enviaba el producto. Así validaron la demanda antes de invertir en stock o tecnología.

¿Por qué sirve? Porque permite validar si la gente realmente lo usaría

1.3 Definición del MVP

El 80% del valor viene del 20% de las funcionalidades.

Muchos fundadores confunden MVP con una versión "chica" o "barata" del producto final. Pero no es así. Un MVP (Producto Mínimo Viable) es la forma más simple y efectiva de probar una idea en el mundo real, con la menor inversión posible. Su objetivo no es escalar ni impresionar, sino aprender rápido si la solución interesa y funciona para los usuarios.

Un MVP:

- Resuelve el problema central del usuario
- Tiene solo lo necesario para funcionar
- Se puede construir en poco tiempo
- Sirve para testear hipótesis concretas

No incluye todo lo que se imagina. No tiene animaciones, ni filtros de color, ni pagos integrados si no son indispensables. **La clave es enfocar**. Repitamos, pero esta vez con un poco más de énfasis: **LA CLAVE ES ENFOCAR.**

¿Cómo definir un MVP paso a paso?

1. Hacer una lista completa de funcionalidades. Anotar todo lo que se imagina que debería tener el producto ideal.
2. Clasificar en dos columnas:
    - Core : Sin esto, el producto no resuelve el problema. Core se refiere a lo esencial del producto, aquello estrictamente necesario para resolver el problema.
    - Complementarias: Agregan valor o mejoran la experiencia, pero no son críticas al inicio.
3. Dejar solo lo esencial (Core). Lo demás, se guarda para una próxima versión.
4. Definir qué se quiere testear. ¿Se busca saber si las personas usan el sistema? ¿Si completan un flujo? ¿Si lo recomiendan?

Ejemplos de MVP:

Producto Final	MVP Inicial
Uber (app global)	App para iPhone en San Francisco con 3 conductores
Instagram completo	App para subir fotos con 1 solo filtro
Mercado Libre	Sitio con publicaciones manuales y sin carrito

Guía práctica: cómo definir un MVP (con ejemplo explicado)

Para desarrollar el MVP, se puede trabajar con estas tres preguntas clave. A continuación, un ejemplo paso a paso:

Contexto: Una startup quiere conectar personas con técnicos de confianza para reparaciones del hogar.

1. **¿Cuál es el problema real que se quiere resolver?**

   Ejemplo: “Muchas personas no saben dónde encontrar técnicos

confiables para reparar electrodomésticos y pierden tiempo buscando recomendaciones en redes o sitios poco claros."

2. **¿Cómo se puede resolver ese problema de forma simple y rápida (sin todas las funciones soñadas)?**

   Ejemplo: "Una app muy simple donde se ingresa una ciudad y se muestra una lista de 3 técnicos verificados. Se puede pedir una cita a través de un botón de WhatsApp."

3. **¿Qué funcionalidades NO se van a incluir en esta primera versión?**

   Ejemplo:

   - No habrá pagos integrados (se coordina por fuera)
   - No habrá reseñas ni calificaciones visibles
   - No habrá chat dentro de la app
   - No se mostrará disponibilidad en tiempo real

Este ejercicio ayuda a evitar la trampa del "ya que estamos, sumemos esto también". Cuanto más chico y enfocado sea el MVP, más rápido se podrá testear si la idea realmente funciona para las personas.

Consejo: Un buen MVP no se construye para convencer inversores, sino para aprender del usuario real. Si lo que se construye no da feedback claro, entonces no es un MVP: **es una maqueta CARA.**

Test Interactivo: ¿Estás Realmente Listo para Construir tu MVP?

Antes de invertir tiempo, dinero y energía en desarrollar tu producto... paremos un segundo. Este test te va a ayudar a saber si tenés las bases sólidas o si estás por lanzarte al vacío sin paracaídas.

¿Cómo funciona?

Responde **Sí**, **No** o **A veces** a cada pregunta. Al final, sumá tus respuestas y descubrí en qué etapa estás.

1. Validación del Problema

- ¿Tuviste al menos 10 conversaciones reales con personas que viven el problema que querés resolver?
- ¿Tenés algún dato (encuestas, entrevistas, feedback) que confirme que tu solución hace falta?

2. Definición del MVP

- ¿Sabés cuál es la funcionalidad **mínima e indispensable** para resolver el problema?
- ¿Eliminaste todo lo “lindo de tener” y te quedaste con el corazón del producto?

3. Equipo y Presupuesto

- ¿Tenés claro qué tipo de equipo necesitás (diseñador, programadores, PM)?
- ¿Tenés un presupuesto realista o una estrategia clara para conseguir financiamiento?

4. Proceso de Desarrollo

- ¿Tenés un prototipo aunque sea en papel, Figma o PowerPoint que muestre cómo funcionaría tu producto?
- ¿Ya pensaste una hoja de ruta con hitos claros (diseño, desarrollo, testeo)?

5. Lanzamiento y Métricas

- ¿Tenés definido cómo vas a medir si tu MVP funciona (por ejemplo: retención, conversión, feedback de usuarios)?
- ¿Tenés un plan concreto para mejorar el producto después de que los primeros usuarios lo usen?

Resultados

Mayoría "No" → Estás por construir a ciegas

Podés tener una gran idea, pero todavía no tenés lo necesario para ejecutarla bien.

Volvé al **Capítulo 1 (Validación y Preparación)**.

Hablá con más clientes y creá un prototipo rápido antes de gastar en desarrollo.

Mayoría "A veces" → Estás cerca, pero hay puntos ciegos

Tu idea ya tiene forma, pero hay partes clave que no están resueltas.

Releé el **Capítulo 3 (Costos, Financiamiento y Toma de Decisiones)**.

Buscá ayuda para definir prioridades y asegurate de tener a alguien que cuide la ejecución (*tip: un buen PM*).

Mayoría "Sí" → ¡Tenés el mapa, es hora de moverse!

Validaste bien el problema, sabés qué querés construir y tenés los recursos o al menos un plan.

Es momento de lanzar una **beta privada** y empezar a testear con usuarios reales.

Bonus: Checklist de Supervivencia para Founders

📌 *"Si no podés explicar tu MVP en una frase, no está claro enough."*

📌 *"El código es caro. La validación es barata."*

📌 *"Contratá lento. Despedí rápido (especialmente en equipos técnicos)."*

Checklist de Validación

Antes de pasar al Capítulo 2 (Desarrollo), conviene verificar:

✅ Se habló con al menos 10 potenciales usuarios o clientes.

✅ Se probó al menos un método de validación (landing, prototipo, venta manual).

✅ Se cuenta con un documento de alcance del MVP con funcionalidades priorizadas.

¿Todo marcado? → Avanzar al Capítulo 2.

¿Falta algo? → Volver, ajustar y recién después seguir.

Capítulo 2: El Proceso de Desarrollo – De la Idea al Código

Ya vimos la importancia de hacerse buenas preguntas, validar lo que queremos construir y tener, al menos sobre papel, una idea clara de lo que debería incluir nuestro MVP.

Como dice Eric Ries: “Un MVP no es una versión barata de tu producto, es el experimento más rápido para aprender del mercado."

Y justamente, en un mundo donde todo parece resolverse con inmediatez —desde herramientas no-code hasta inteligencia artificial que promete construir sistemas en minutos— es fácil caer en la tentación de **saltar directamente al hacer**, sin detenernos a entender **qué** y **por qué** lo estamos construyendo.
Pero construir algo real —algo sólido— sigue requiriendo un proceso. Un camino que, mientras más consciente lo recorras, mejores serán los resultados. Conocer ese proceso (y sus tiempos) no es solo para saber qué viene después, sino también para entender qué estás eligiendo saltearte y qué riesgo asumís al hacerlo.

2.1 Las 4 Etapas Clave (y los Riesgos de Saltarse Pasos)

Desarrollar un software no se trata solo de programar. Hay al menos cuatro etapas fundamentales que, si estás apuntando a un MVP funcional y usable, vas a tener que recorrer. Para un MVP estándar, calculá entre 6 a 8 meses desde la idea hasta que esté lanzado y funcionando.

🔵 Etapa 1: Descubrimiento y Diseño (aprox. 2 meses)

Objetivo: Definir qué vas a construir y cómo debe funcionar.

Acá ya validaste tu idea, sabés qué problema querés resolver y qué necesita tener tu MVP. Pero ahora hay que traducir eso en algo que sea entendible para todos: para vos, para el equipo técnico y para el usuario final.

Esta etapa es ideal cuando podés reunir en una misma mesa:

- La mirada de negocio (vos y otros decisores).
- El enfoque de diseño (cómo se va a ver y sentir).
- El punto de vista técnico (cómo se va a construir).

Este sería el “mundo ideal”, y todos sabemos que a veces no tenés todo eso por temas de presupuesto o tiempos, pero es bueno saberlo para compensar o anticipar qué te puede faltar y en qué momento del proyecto.

¿Qué pasa durante esta etapa?

Durante estas semanas, se trabaja en equipo —idealmente con alguien que coordine, como un *Product Owner* o *Product Manager*.
No te asustes con los términos: simplemente es la persona que hace de puente entre vos, diseño y desarrollo, para que todo fluya.

En esas reuniones:

- El diseñador empieza a armar flujos de navegación: cómo se mueve el usuario dentro de tu app.
- Se crean bocetos en baja definición, que luego se transforman en diseños más detallados.
- El equipo técnico aporta su mirada para validar qué es viable y qué no.

¿Por qué esto es clave?
Porque uno de los errores más comunes es cuando se diseña sin consultar al equipo técnico, y después, al momento de programar, hay que rehacer todo. Esa brecha entre “lo que se imaginó” y “lo que se puede hacer” es una de las principales causas de retrabajo.

¿Qué tenés al final de esta etapa?

Una maqueta interactiva (*mockup*) de tu MVP. O sea, tu sistema ya está "dibujado", validado, y lo podés navegar como si existiera. No hay nada de código todavía, pero vas a ver tu producto en acción.

Esto te permite:

- Confirmar si lo que se está construyendo es lo que realmente querías.
- Evitar sorpresas costosas más adelante.
- Hacer cambios en esta etapa (que son mucho más baratos y rápidos que después).

❌ **Error común:**

"Empecemos a programar ya, el diseño lo ajustamos después."

→ ***Esto suele terminar con un 40% (como mínimo) de presupuesto extra en cambios tardíos.***

🔵 Etapa 2: Desarrollo Técnico (aprox. 3 a 4 meses)

Objetivo: Construir el producto funcional, basándose en los diseños validados.

Una vez que tenés los diseños aprobados, el equipo de desarrollo empieza a transformar esos mockups en software real que funciona.

Acá es donde entra el famoso "código", pero lo ideal es que como founder no te metas en el cómo técnico, sino que mantengas el foco en:

- Que el equipo avance según lo diseñado.
- Que haya entregas parciales (versiones navegables).
- Que se mantenga la comunicación clara y regular.

¿Qué deberías esperar en esta etapa?

1. Desarrollos parciales o por módulos, no todo junto al final.
    - Por ejemplo: primero el login, después la sección de productos, después los pagos.
    - Esto te permite ir viendo avances, dar devoluciones (feedback) y evitar sorpresas.

2. **Demos internas cada 2 o 3 semanas.**
    - No tenés que saber de código, pero sí podés probar la app, navegarla, y ver si lo que se está construyendo es fiel a lo que validaron juntos en diseño.

3. **Cambios menores, no estructurales.**
    - Es normal ajustar textos, imágenes o pequeñas cosas.
    - Pero si querés cambiar una lógica entera que ya se validó en diseño, eso puede implicar retrabajo y más tiempo/costo. Por eso es tan clave hacer bien la Etapa 1.

🧠 **Tip como fundador:** Pedí que te compartan el producto en un entorno de pruebas (*staging*) donde puedas probarlo sin miedo. No te preocupes por entender el código, preocúpate por entender la experiencia del usuario.

❌ **Error común:**

"Ya que estamos, ¿por qué no le agregamos también esto…?"

→ ***Cambiar el alcance en mitad del desarrollo (lo que se llama scope creep) puede hacer que el proyecto se descontrole en tiempos y costos.***

Siempre se puede mejorar después. El objetivo ahora es sacar una primera versión usable y validable.

🔄 **Acordate:**

Un MVP no es un producto final. Es una primera versión funcional para aprender del usuario real.

🔵 **Etapa 3: Pruebas, Ajustes y Preparación para Lanzamiento (1 mes)**

Objetivo: Asegurarse de que todo funcione como se espera antes de lanzar.

Una vez que el producto está desarrollado, no se lanza directo. Antes, se pasa por una etapa clave de validación técnica y funcional:

¿Qué pasa en esta etapa?

1. Testing técnico (QA)

   El equipo técnico prueba que todo funcione bien: que no haya errores graves, que el sistema no se rompa ante ciertos usos, que los flujos hagan lo que tienen que hacer.

2. Testing de usuario (UAT)

   Acá entrás vos y/o alguien de tu equipo. Se usa la aplicación como si fueras un usuario real.

   El objetivo es detectar cosas que “en teoría funcionaban” pero en la práctica generan confusión o fricción.

3. Ajustes menores

   Se corrigen errores (bugs), se afinan detalles visuales, se mejora lo que haga falta antes del lanzamiento.

   Es normal encontrar cosas para ajustar. De hecho, es preferible encontrarlas ahora que después de salir a producción.

🧠 **Como fundador, ¿qué deberías hacer acá?**

- Probar todo, no solo “mirar si se ve lindo”. Hacé clic, “rompé”, navegá como un usuario curioso.
- Pedir a alguien externo (si podés) que lo pruebe sin guías. Las reacciones de alguien que no estuvo en el proyecto son oro puro.
- Registrar observaciones claras, no vagas. No digas “no me gusta”, sino “este botón no se entiende” o “no sé qué hacer en esta pantalla”.

❌ **Error común:**

"Ya está casi listo, lancémoslo así y después lo arreglamos."

→ ***Está bien no volverse perfeccionista y no quedarse eternamente esperando el momento ideal para lanzar. Pero lanzar algo con parches evidentes suele dejar una mala primera impresión, y recuperarse después cuesta más que haber esperado una semana más para hacerlo bien.***

Por eso, hay que ser cuidadosos con este tipo de decisiones: a veces, un poco más de paciencia puede evitar mucho retrabajo (y frustración) después.

🔵 **Etapa 4: Lanzamiento, Feedback Real y Iteración**

Objetivo: Salir al mercado, obtener datos reales y seguir mejorando.

¡Llegó el gran día! Tu MVP está en producción, online, y los primeros usuarios lo empiezan a usar en el mundo real.
Después de meses de trabajo, validaciones, reuniones y ajustes, finalmente tenés algo tangible funcionando.

¡Pero ojo!: esto no es el final, es el verdadero comienzo.

¿Qué cambia en esta etapa?

Hasta ahora venías trabajando sobre suposiciones. Suposiciones bien fundamentadas, sí —basadas en validación, diseño y lógica de negocio— pero suposiciones al fin.

Ahora lo que importa es lo que realmente hacen los usuarios.
Qué usan, qué ignoran, qué entienden sin que se lo expliques, y qué los frustra sin que vos lo veas venir.

No te enamores de tu producto**.** No te enojes con el usuario si no hace lo que pensabas que haría. **Aprendé** de él. Eso es lo que convierte un MVP en un producto real: el aprendizaje.

¿Qué deberías hacer como founder?

Medir comportamiento:

Tiempo en pantalla, tasas de conversión, abandonos, flujos interrumpidos, errores.

Observar patrones:

¿Dónde se traban? ¿Qué partes ni siquiera exploran? ¿Qué funciones usan diferente de lo que imaginabas?

Hablar con usuarios reales:

No esperes que te escriban un mail detallado. Salí a buscarlos: hacé una llamada, mandá un mensaje, pedí una videollamada corta. Lo que te digan —aunque sea una frase— vale más que cualquier documento de análisis interno.

Mentalidad para esta etapa

- No salgas a defender el producto. Salí a entenderlo mejor.
- No pienses en características (features) nuevas todavía. Pensá en estabilidad, comprensión y mejora de lo que ya está.
- Este es el momento de iterar con inteligencia: ajustar sin romper, mejorar sin reinventar todo, y sobre todo, **escuchar sin justificarte**.

❌ **Error común:**

"Ya está lanzado, pasemos a otra cosa."

→ ***Si ignoras el feedback real, te estás perdiendo el mayor valor del MVP, que no es lanzar sino aprender para construir la siguiente versión con mucho más impacto.***

✅ **En resumen**:

Llevar una idea a un producto digital no es magia, ni solo programar.
Es un camino con etapas claras, donde cada paso tiene su valor, su tiempo y sus riesgos si se saltea.

Saber esto como founder te da una gran ventaja:

- Vas a tomar mejores decisiones.
- Vas a evitar frustraciones innecesarias.
- Y vas a construir productos más sólidos, más rápido y con menos costo emocional y económico.

Capítulo 3: Costos, Financiamiento y Toma de Decisiones

Ya validaste tu idea. Sabés que vas a pasar por etapas como diseño, desarrollo y lanzamiento. Ahora toca hablar de números.

Porque en tecnología, ***un presupuesto realista es tu primer filtro de supervivencia.***

Y como no existen fórmulas mágicas, este capítulo te va a ayudar a entender qué opciones tenés para construir tu MVP, cuánto cuesta cada una, qué riesgos trae, y cómo elegir según tu momento.

En este capítulo vas a ver:

- Qué camino tecnológico te conviene (No-code o Custom)
- Qué tipo de proveedor contratar (freelancer, agencia, equipo interno)
- Cuánto cuesta realmente un MVP
- Qué errores financieros evitar
- Qué preguntar a un proveedor para no caer en promesas vacías

Todas estas decisiones parecen aburridas y poco importantes pero en realidad definen si vas a avanzar con claridad o no. Para arrancar, mirá dos caminos distintos que tomaron otros founders.

Lo que sigue es más habitual de lo que imaginás:

Feli tenía una idea que le quemaba la cabeza: digitalizar clases de cocina para mujeres mayores. No sabía programar, pero armó su MVP con Glide y Notion en 2 meses. Vendió 50 accesos a $20 a través de Instagram. Con eso validó, reunió algo de capital propio y contrató una agencia con servicio tipo Stack Team, y en 6 meses tenía una app propia, funcionando en 3 países.

No empezó con inversión externa, ni con equipo técnico. **Empezó con claridad.**

Y esto es más habitual aún:

José quería lanzar una app para organizar turnos médicos. No tenía presupuesto, así que contrató a un desarrollador full stack que no validó bien. Le prometieron todo listo en 8 semanas. Al principio todo parecía ir bien, pero en la semana 6 el dev dejó de responder. El código estaba incompleto y no había backups. José tuvo que empezar de cero, esta vez con una agencia, y

gastar mucho más de lo que tenía previsto. Perdió tres meses de trabajo, la confianza en sí mismo y una gran oportunidad de salir antes que un competidor.

Lección: No es solo lo que cuesta. Es lo que puede costarte si sale mal.

Visualizar dónde no querés estar ya es un buen punto de partida

Antes de meternos de lleno en el capítulo, hagamos una pausa y pongamos sobre la mesa algunas definiciones.

Como fundador habrá decisiones técnicas que quizás no tomes vos directamente, pero que van a aparecer en muchas propuestas o reuniones. Entender al menos lo básico de esas definiciones te va a dar más claridad, más poder de negociación y menos sorpresas.

No hace falta que te conviertas en programador, pero sí que el "oído" te suene cuando alguien hable de *no-code*, *full stack* o *stack team*.

Así que arranquemos por lo primero:

¿No-code o Custom Code?

Elegir entre *no-code* o *custom code* es como decidir si vas a armar tu negocio con piezas de Lego o si vas a construir todo desde cero con tus propias herramientas y materiales.

🔵 **No-code: el camino rápido y sin código**

No-code es una forma de crear productos digitales —como apps, sitios web o herramientas internas— usando plataformas que no requieren programar. Estas herramientas tienen interfaces visuales, donde podés "armar" tu producto con bloques, módulos o componentes preconstruidos.

En vez de escribir código, trabajás con elementos que ya están listos y solo hay que combinarlos y configurarlos para que hagan lo que necesitás. Esto acelera mucho el desarrollo y hace posible que personas sin experiencia técnica puedan crear soluciones digitales por sí mismas.

¿Cuándo sirve el no-code?

- Tenés poco presupuesto y querés lanzar rápido.
- Estás validando una idea y no necesitás nada súper personalizado.
- Querés tener más control directo sin depender siempre de alguien técnico.

Ejemplos de herramientas no-code: Bubble, Glide, Webflow.

¿Cuál es la contra?

Hay límites. A veces tu idea crece y la plataforma no-code ya no alcanza. Es como querer correr una maratón con un monopatín: no fue hecho para eso.

🔴 **Custom Code: la solución a medida**

Cuando necesitás algo muy específico, complejo o único para tu negocio, entra en juego el custom code. Acá los programadores escriben el software desde cero, usando lenguajes como JavaScript, Python, o frameworks como React.

Esto significa que cada línea de código está diseñada especialmente para vos y tu producto, sin las limitaciones de las plataformas no-code. Podés tener total control sobre cómo funciona, cómo se ve y qué puede hacer.

Pero este camino es más lento y requiere más inversión, porque construir algo a medida es como crear tu propio motor en vez de usar uno ya hecho. Además, necesitás un equipo técnico para mantener y actualizar el producto con el tiempo.

¿Cuándo conviene el custom code?

- Tu producto necesita lógica compleja, automatizaciones o integraciones que no están en las plataformas no-code.
- Querés escalar en serio y tener total flexibilidad.
- Buscás un diseño único, con rendimiento optimizado.

¿Cuál es la contra?

Es más caro, lleva más tiempo y vas a depender de un equipo técnico (freelancers, una agencia o contratar gente).

En resumen:

	No-code	Custom Code
Velocidad	Muy rápido	Más lento
Costo inicial	Bajo	Más alto
Flexibilidad	Limitada por la plataforma	Máxima
Escalabilidad	Buena para etapa validación	Ideal para crecimiento largo plazo
Dependencia	Podés avanzar solo o con mínima ayuda	Necesitás equipo técnico

👨‍💻 Desarrollador Full Stack

Pero ¿Qué es un *desarrollador Full Stack*?

Un **Full Stack** es un programador que puede hacer **todo lo necesario para que tu producto digital funcione**, desde lo que ve el usuario hasta lo que pasa detrás.

Es como un "hombre orquesta" del software: puede diseñar las pantallas, conectar la base de datos, programar la lógica, y subirlo todo a internet.

🔧 ¿Qué hace un Full Stack?

- Diseña y programa la interfaz del usuario (frontend)
- Crea la lógica que corre detrás del sistema (backend)
- Maneja la base de datos donde se guarda la información
- A veces también se encarga de subirlo todo a un servidor (hosting)

👥 *Stack Team*

Pero ¿Qué es un *Stack Team*?

Un **Stack Team** es un equipo de especialistas que trabaja para vos durante una cierta cantidad de horas mensuales, organizados por una agencia o software factory. Vos no contratás personas, contratás un **servicio flexible de equipo completo**.

Es como tener tu propio mini-departamento de tecnología alquilado por mes: PM, diseñadores, programadores, DevOps, QA… todos disponibles según la etapa de tu proyecto.

🧩 ¿Cómo funciona?

- Comprás un “paquete de horas” por mes (ej. 160 hs)
- La agencia distribuye esas horas según lo que tu proyecto necesite: puede ser más diseño al principio, más programación después, más testing al final
- Incluye gestión, planificación y coordinación interna. Vos solo tenés que participar como líder de producto.

3.1 ¿Cuánto cuesta realmente un MVP?

Depende del camino que elijas:

Tipo de MVP	Costo Estimado	Tiempo	¿Para quién es?
No-code	$5K–$15K USD	1–2 meses	Fundadores que están validando ideas básicas
Custom Code	$15K–$35K USD	3–4 meses	Startups que quieren base sólida para escalar
Custom Code + Escalable	$35K–$75K USD	5–6 meses	Empresas con visión de largo plazo y funding

3.2 Los 3 Costos Ocultos que arruinan Startups

Cuando se habla de "costos", la mayoría de los founders piensa en programadores, herramientas o infraestructura. Pero hay otros costos, más silenciosos, que aparecen cuando el proyecto ya está en marcha y, si no los ves venir, pueden arruinar todo antes de despegar.

No importa si estás trabajando con freelancers, una agencia o un equipo propio: estos tres errores se repiten una y otra vez, y terminan costando tiempo, plata y mucha frustración.

1. Cambios de Alcance ("Scope Creep")
    - *Caso:* Feature no planeada que aumenta el costo en 40%.
    - *Solución:* Usar un documento de alcance validado entre vos y el equipo tecnico.
2. Mala Gestión de Equipos
    - *Estadística:* 62% de los freelancers requieren supervisión intensiva (Upwork, 2023).
    - *Solución:* Contrata un Product Manager aunque sea part-time.
3. Infraestructura Subestimada
    - *Ejemplo:* App que colapsó al llegar a 10K usuarios por no usar autoescalado (AWS/GCP).
    - *Costo extra:* $20K+ en reparaciones de emergencia.

3.3 ¿Freelancers Full Stack, Agencia o Equipo Interno?

Criterio / Opción	Freelancers Full Stack	Agencia / Software Factory	Equipo Interno (In-house)
Costo Inicial	💲 Bajo	💲 💲 Medio	💲 💲 💲 Alto
Velocidad de Arranque	⚡ Rápido (si encontrás el indicado)	⚡⚡ Rápido con procesos armados	🐢 Lento (buscar, entrevistar, contratar)
Calidad Técnica Esperada	🎲 Variable (depende de cada freelancer)	✅ Alta y más consistente	🎯 Alta, pero depende del talento que consigas
Gestión del Proyecto	🧩 La hacés vos (o alguien de tu equipo)	🧠 Incluye gestión y seguimiento	🕹️ La hacés vos (o alguien de tu equipo)

Escalabilidad del equipo	🔄 Limitada (difícil de escalar rápido)	📈 Alta (pueden sumar recursos rápido)	🧱 Lenta y costosa
Dependencia a largo plazo	🔒 Alta (dependés de 1 o 2 personas clave)	⚖️ Media (más institucionalizado)	🧬 Baja (vos construís el know-how)
Visión del producto	📦 Puede ser limitada al entregable puntual	🔍 Aportan experiencia de producto y UX/UI	🧭 Alta alineación si el equipo entiende bien el negocio
Flexibilidad	🤸 Alta (poca burocracia)	🔐 Media (procesos preestablecidos)	🧰 Alta (si tenés equipo ágil)
Riesgo	🚨 Alto	✅ Bajo	✅ Bajo

¿Por qué ese riesgo?	Dependés 100% de una persona: si falla, se va o no rinde, tu proyecto se frena. También puede no cubrir bien todas las áreas (ej. diseño, QA).	El equipo se adapta, reemplazan perfiles si es necesario y tienen procesos para mitigar errores. Menos riesgo	El equipo se adapta, reemplazan perfiles si es necesario y tienen procesos para mitigar errores. Menos riesgo
Ideal si..	💡 Tenés poco presupuesto y sabés qué querés	🧩 Querés despreocuparte de lo técnico y enfocarte en el negocio	🏗️ Querés construir a largo plazo y tener control completo

Entonces, ¿cuál es el mejor camino para vos?

Como podés ver, no hay una respuesta única. Cada opción tiene sus ventajas y sus riesgos. Lo importante es que puedas hacer un **análisis honesto de tu momento**, tus recursos y tu nivel de involucramiento.

- ¿Tenés tiempo y conocimientos para coordinar un equipo técnico?
- ¿O preferís enfocarte en el negocio y delegar la ejecución?

Bonus: Si contás con algo de presupuesto y querés avanzar con foco en tu negocio, una agencia con servicio tipo Stack Team suele ser la opción más equilibrada: te da estructura, calidad y menos riesgo operativo.

3.4 Fuentes de Financiamiento para el MVP

Si querés desarrollar tu MVP pero todavía no levantaste inversión formal, hay caminos viables que podés explorar. No necesitas millones: necesitás claridad, enfoque y saber elegir entre estas **cuatro fuentes comunes de financiamiento temprano.**

💼 1. Bootstrapping

¿Qué es el Bootstrapping?

Es autofinanciar tu startup con recursos propios: ahorros personales,

ingresos de otro negocio, o incluso apoyo familiar o de socios. No dependés de nadie externo y conservás el 100% del control.

¿Cuándo sirve?

- Si tenés ingresos disponibles o podés recortar gastos temporalmente.
- Si venís de un negocio anterior (consultoría, ecommerce, servicios) y querés reinvertir.
- Si preferís ir paso a paso, validando sin presiones externas.

📌 **Ejemplo real:**

Basecamp, una herramienta popular para gestión de proyectos, se financió con ingresos de su propia consultora. Nunca buscó inversión.

✅ **Ventajas:**

- Avanzás rápido y tomás decisiones solo.
- No diluís tu participación.
- Te obliga a mantener foco y eficiencia.

⚠️ **Desventajas:**

- El riesgo es 100% tuyo.
- Si te quedás sin fondos, no hay red de contención.
- El crecimiento puede ser más lento.

🏛️ 2. Grants / Fondos Públicos

¿Qué son?

Son subsidios no reembolsables que otorgan gobiernos, municipios o entidades públicas para apoyar proyectos de innovación, tecnología o impacto social. En general no te piden devolución ni participación accionaria, pero sí cumplir con ciertos objetivos y presentar resultados.

¿Cuándo sirven?

- Si tu proyecto está vinculado a educación, salud, inclusión, ciencia o desarrollo económico.
- Si tenés tiempo y energía para armar una postulación formal.
- Si buscás respaldo institucional o recursos sin ceder equity.

📌 Ejemplo:

Una startup de salud digital en Argentina recibió $50.000 USD de un programa de Aportes No Reembolsables (ANR) para desarrollar su MVP enfocado en atención primaria.

✅ Ventajas:

- No cede participación ni genera deuda.
- Te posiciona mejor frente a otras fuentes de financiamiento.
- Ayuda a profesionalizar el proyecto.

⚠️ **Desventajas:**

- Requiere tiempo y preparación.
- La burocracia puede demorar desembolsos (hasta 12 meses de demora).
- No todas las industrias califican.

💰 3. Preventas o Crowdfunding

¿Qué son?

Es vender acceso anticipado a tu producto (antes de desarrollarlo) a cambio de un descuento, un beneficio exclusivo o simplemente confianza en tu visión.

Podés hacerlo por tu cuenta o en plataformas como **Kickstarter** o **Indiegogo**.

¿Cuándo sirven?

- Si tu público objetivo entiende bien el problema que estás resolviendo.
- Si tenés comunidad o presencia online (aunque sea pequeña pero activa).
- Si tu MVP puede entregarse en un plazo claro (ej. 2–3 meses).

📌 **Ejemplo:**

Un emprendedor armó una landing, ofreció su app de productividad a $30 anticipado, y vendió 200 accesos: con eso recaudó $6.000 para construir el producto.

✅ **Ventajas:**

- Validás si la gente está dispuesta a pagar.
- Financiás el desarrollo sin ceder control.
- Puede convertirse en tu primera base de usuarios.

⚠️ **Desventajas:**

- Necesitás saber comunicar bien la idea.
- Si no cumplís con lo prometido, puede dañar tu reputación.
- Dependés mucho del marketing y del momentum inicial.

🦸 **4. Capital Ángel**

¿Qué es?

Son personas con capital propio (normalmente ex-founders o ejecutivos) que invierten en startups en etapas tempranas a cambio de un porcentaje minoritario de la empresa (equity). No solo aportan dinero: también experiencia, contactos y visión.

¿Cuándo sirve?

- Si tu idea es más ambiciosa y necesitás más capital desde el inicio.
- Si querés sumar un socio estratégico que entienda de negocios.
- Si estás dispuesto a ceder un % del proyecto a cambio de impulso.

📌 **Ejemplo:**

Una startup edtech levantó $25.000 USD de un inversor ángel. Además del dinero, el inversor le abrió puertas en escuelas y la ayudó a definir pricing y modelo de negocio.

✅ **Ventajas:**

- Conseguís capital con acompañamiento.
- No tenés que devolver el dinero.
- Te abre puertas a rondas futuras o alianzas estratégicas.

⚠️ **Desventajas:**

- Cedés parte de tu startup (aunque sea minoritaria).
- Puede haber presión por resultados o plazos.

🧮 Comparativa final

Opción	¿Qué es?	¿Cuándo sirve?	Disponibilidad de dinero estimado	Cede participación
Bootstrapping	Usar tus propios recursos	Tenés ahorros, ingresos, o querés control total	Inmediato	❌ No
Grants	Fondos públicos no reembolsables	Proyectos con impacto social o tecnológico	6–12 meses	❌ No
Preventas	Cobrar antes de construir tu MVP	Tenés comunidad o	1–2 meses	❌ No

		sabés vender la idea		
Capital Ángel	Inversor individual que aporta dinero + experiencia	Buscás más capital + alguien que sume estrategia	1–3 meses	✅ Sí

Bonus: Ya tenés una idea en la cabeza de cuánto podrías gastar y qué equipo vas a necesitar para desarrollar tu proyecto. Pero ojo, solo calcular lo estimado no alcanza. Siempre aparecen costos imprevistos, como:

- Cambios de alcance (cuando se suman cosas nuevas)
- Demoras inesperadas
- Errores o ajustes técnicos
- Gastos extras en infraestructura o soporte

Por eso, lo más inteligente es sumar un 20% extra como reserva. Es una regla práctica para no quedarte corto y evitar que tu proyecto se frene por falta de plata.

Por ejemplo, si el presupuesto inicial es así:

- Desarrollo: $40K
- Gastos operativos (6 meses): $18K
- Total estimado: $58K

Lo ideal sería contar con algo cerca de $70K para cubrir esos imprevistos y estar tranquilo.

3.5 Cómo Negociar con Proveedores Tecnológicos

Cuando llegás a la etapa de negociar con un proveedor para desarrollar tu producto, no basta con tener la idea clara; necesitás hacer las preguntas correctas para evitar sorpresas y perder tiempo y dinero. Acá te dejamos una guía sencilla con preguntas clave y señales para detectar posibles problemas.

5 Preguntas Clave para Evaluar a un Desarrollador o Agencia

1. **¿Pueden mostrarme ejemplos?**

   No pedimos solo una demo bonita o screenshots. Queremos ver que efectivamente pueden construir lo que prometen, y que no van a copiar y pegar soluciones genéricas. Esto te ayuda a saber si tienen experiencia real en proyectos parecidos.

2. **¿Cómo manejan los cambios en el alcance del proyecto?**

   En cualquier proyecto, las cosas cambian. Necesitás saber cómo te van a cobrar si modificás algo o agregás funcionalidades. ¿Hay un sistema claro? ¿Se factura extra por hora o hay paquetes? Esto evita facturas inesperadas.

3. **¿Qué indicadores usan para medir la calidad del software?**

   Preguntá sobre métricas concretas, por ejemplo, cuántos bugs críticos toleran, cómo prueban el código, si hacen pruebas automáticas o manuales. La calidad es clave para no tener problemas cuando tu producto esté en manos de usuarios.

4. **¿Tienen experiencia en mi industria?**

   No es lo mismo desarrollar una app para fintech que para educación o salud. Cada sector tiene regulaciones, necesidades y riesgos particulares. Que conozcan tu industria puede ahorrar tiempo y mejorar el resultado.

5. **¿Quién será mi punto de contacto único durante el proyecto?**

   Es fundamental que haya una persona clara con la que puedas hablar directamente, que te entienda y agilice la comunicación. Evitá proveedores que te pasen de un lado a otro sin darte respuestas concretas.

Señales de Alerta para Evitar Problemas

- **No tienen portfolio o quieren que firmes un NDA demasiado restrictivo antes de mostrar trabajos.** Esto puede esconder falta de experiencia real o poco profesionalismo
- **Cobran por hora sin ningún límite o tope claro.** Podés terminar pagando mucho más de lo esperado sin saber cuánto.
- **Prometen plazos imposibles o milagrosos.** Por ejemplo, decir que harán una app compleja como Uber en un mes suele ser señal de falta de experiencia o querer vender humo.

Bonus: *Confiá pero verificá*

Es normal que confíes en tu proveedor, pero pedí siempre referencias o charlá con clientes anteriores. El desarrollo tecnológico es un viaje con altibajos y tener una comunicación transparente con el equipo hace la diferencia.

Checklist para Negociar con Proveedores Tecnológicos

Antes de la reunión:

[___] Revisé el portfolio o ejemplos de trabajos previos.

[___] Identifiqué qué aspectos de mi proyecto son críticos para validar experiencia (industria, tecnologías, tamaño).

[___] Preparé mis dudas sobre plazos, costos y manejo de cambios.

Preguntas clave para hacer en la entrevista:

1. ¿Podés mostrarme ejemplos de proyectos similares al mío?
2. ¿Cómo gestionan los cambios de alcance? ¿Cómo me van a cobrar si agrego o cambio funcionalidades?
3. ¿Qué indicadores usan para medir la calidad del software? ¿Cómo aseguran que el producto funcione bien?
4. ¿Tienen experiencia en mi industria o con productos parecidos?
5. ¿Quién será mi contacto principal durante el proyecto? ¿Cómo será la comunicación?

Señales para tener en cuenta:

[___] ¿Tienen portfolio público y pueden mostrar referencias?

[___] ¿Las respuestas sobre costos y tiempos son claras y realistas?

[___] ¿Evitan dar plazos imposibles o promesas poco creíbles?

[___] ¿El proveedor propone una persona responsable con la que pueda comunicarme siempre?

Después de la reunión:

[___] Verifiqué referencias o comentarios de otros clientes.

[___] Evalué si la comunicación fue clara y transparente.

[___] Analicé si la propuesta cumple con mis expectativas reales y presupuesto.

📘 **Glosario rápido (porque a los técnicos nos encantan los términos raros):**

Frontend: lo que el usuario ve y toca (pantallas, botones, etc.)

Backend: lo que pasa detrás de escena (base de datos, lógica)

Stack: conjunto de tecnologías que usa un producto digital

Full Stack: programador que maneja frontend + backend

DevOps: persona que se encarga de la infraestructura (servidores, despliegue).

QA (Quality Assurance): quien prueba el software buscando errores

Scope creep: cuando se suman cambios al proyecto que no estaban previstos

Checklist para founders antes de contratar:

Antes de avanzar, verifica:

[___] Tengo un presupuesto con +20% de reserva para imprevistos

[___] Comparé al menos 3 propuestas con el mismo alcance

[___] Definí mi fuente de financiamiento inicial (bootstrapping, preventa, grants, inversores)

[___] Sé qué tipo de proveedor me conviene (freelancer, agencia, equipo propio)

[___] Tengo claridad sobre el alcance y los entregables esperados

[___] Tengo alguien que revise los contratos (idealmente, abogado tech)

[___] Sé qué métricas pedir para evaluar calidad del trabajo

Si todo esto te suena bien, estás en condiciones de avanzar con tu MVP con más confianza.

➡️ Próximo Capítulo: *"Tecnología: Elegir Stack y No Morir en el Intento"*.

Capítulo 4

Tecnología: Elegir la Tecnología (Stack) y No Morir en el Intento

Uno de los miedos más comunes entre founders no técnicos aparece cuando se empieza a hablar de tecnología. Y no por nada: seguramente escuchaste frases como "este stack ya está viejo", "eso no escala bien", o "necesitás usar lo último en inteligencia artificial".

Y ahí, lógicamente, aparece la duda:
"¿Estoy usando la tecnología correcta?"
No es una duda menor. Porque una mala elección puede costarte tiempo, plata, y dolores de cabeza. Pero también, y esto es importante: **no hay una única respuesta correcta**. Mucho menos en las primeras etapas.

Antes de entrar en detalle, dejame decirte algo fundamental: **No necesitás ser ingeniero para tomar buenas decisiones tecnológicas.**
Lo que sí necesitás es entender cómo se toman esas decisiones y por qué son importantes para tu producto.

Porque aunque no escribas una línea de código, la tecnología que elijas impacta directamente en tu tiempo, tus costos y tu capacidad de escalar.

Ahora bien, no todas las decisiones tecnológicas se toman de la misma forma. Dependen, en gran parte, del camino que elijas para construir tu producto.

- ¿Vas a usar una plataforma no-code?
- ¿Vas a contratar una agencia o un equipo externo?
- ¿Vas a armar tu propio equipo de desarrollo?

Cada uno de esos caminos condiciona qué tan involucrado vas a estar en las decisiones técnicas… y hasta dónde podés influir.

Vamos por partes:

Si vas con no-code, la tecnología ya está elegida por vos

Plataformas como Glide, Softr, Bubble o Webflow están construidas sobre stacks específicos. Y eso es parte de su propuesta de valor.
Vos no elegís si usan React, si están en la nube o si están montadas sobre AWS o Google Cloud. Y eso, en las primeras etapas, **es una ventaja**. Porque no necesitás saber programar, ni contratar a nadie. Simplemente te enfocas en crear una solución que funcione para tu cliente.

¿Significa que no hay nada técnico que considerar?

Un poco sí:

- Asegurate de que la plataforma tenga **buena reputación y uptime estable**.
- Que tenga una **comunidad activa** donde puedas consultar errores o buscar tutoriales.
- Y que te permita **exportar tus datos**. Porque el día que quieras migrar, no te conviene quedar atado.

📌 **Tip rápido:**

Buscá en Google: "nombre de la plataforma + Reddit" o "nombre de la plataforma + limitations". Te va a dar una idea real de sus puntos fuertes y sus límites.

Si trabajás con una agencia o stack team, confiá... pero con criterio

Cuando contratás un equipo externo (una agencia o software factory), lo más probable es que te propongan un stack de desarrollo.

Y está bien confiar. **Pero no a ciegas.**

Tené en cuenta:

- ¿Tienen experiencia previa con ese stack?
- ¿Cuántas personas en el equipo manejan bien esa tecnología?
- ¿Qué pasa si el dev principal se va? ¿Tienen reemplazo?

No es raro que una agencia proponga un stack "moderno" para seducirte... pero que solo una persona lo sepa usar. Si esa persona se va, tu proyecto queda vulnerable.

📌 **Tip práctico:**

Pedí ver proyectos anteriores con ese stack. Y preguntá cuántos perfiles disponibles tienen para cubrirlo.

Si armás tu propio equipo, tenés libertad... y más responsabilidad

Si decidís contratar programadores directamente, la decisión del stack va a pasar por vos (o por alguien que elijas como referente técnico).

Acá es donde aparece el miedo a elegir mal:

- "¿No es muy viejo usar Ruby on Rails?"
- "¿No conviene Node.js o algo moderno como Elixir?"
- "¿Python no es más fácil de escalar?"

La verdad: **no hay tecnologías buenas o malas en sí mismas**. Lo importante es elegir algo que te permita avanzar sin trabas, que tenga buen soporte, y que no te deje aislado.

📊 **Mini tabla comparativa:**

Camino	¿Elegís el stack?	Riesgo	Libertad
No-code	No	Bajo	Baja
Agencia	Parcial	Medio	Media
Equipo propio	Sí	Alto	Alta

Consejo: Elige Tecnologías con comunidad grande

Este consejo aplica para cualquier camino que elijas.

Elegí herramientas y lenguajes que tengan comunidades activas y maduras.

Porque cuando algo se rompe (y siempre algo se rompe), vas a necesitar:

- Tutoriales
- Foros
- Ejemplos en GitHub
- Otras personas que lo conozcan

💡 **¿Cómo saber si una comunidad es grande?**

Buscá en Google:

- "nombre del lenguaje" site:stackoverflow.com
- "nombre del lenguaje" site:github.com

Si hay miles de resultados, vas bien. Si no encontrás nada... cuidado.

🔄 Y si un día tenés que cambiar de stack...

Te lo decimos así de claro: **no pasa nada.**

Cambiar de stack es común. Muy común.

¿Ejemplo real?

🐦 **Twitter empezó con Ruby on Rails.**

Porque Rails te permite lanzar muy rápido. Pero cuando la plataforma creció, migraron partes a otros lenguajes.

No fue un error arrancar con Rails. Fue lo que les permitió escalar rápido.

¿Entonces?

Elegí lo que te permita avanzar hoy. Si el producto crece, vas a tener tiempo y recursos para mejorar la tecnología.

Tip para evitar complicaciones innecesarias

Si vas por custom code (agencia o equipo propio), evitá dividir frontend y backend desde el primer día.

Muchos founders quieren usar React para el frontend y algo como Node o Django para el backend. Pero eso duplica la complejidad técnica y las chances de error.

👨‍🔧 Mejor elegí un stack que te permita hacer todo con un mismo lenguaje. Como Rails, Laravel, o incluso Next.js.

No es obligatorio. Pero es una buena forma de empezar con menos fricción.

En resumen:

- No hay un stack "perfecto", pero sí podés tomar decisiones informadas.
- No te obsesiones con estar "a la moda": elegí lo que te permita avanzar.
- Hacé preguntas. Confiá, pero entendé lo que te proponen.
- Apuntá a tecnologías con comunidades activas.
- Cambiar de stack no es fracasar. Es parte del crecimiento.

Un último empujón antes de cerrar

Si llegaste hasta acá, ya tenés más herramientas para no entrar en pánico cuando aparezca la palabra "stack" en una propuesta.

Y aunque no seas técnico, es casi seguro que en algún momento te sientes a una mesa donde digan palabras raras. O que te manden una propuesta llena de nombres que no sabés si son lenguajes, frameworks o personajes de videojuegos.

No necesitás hablar el idioma perfecto. Pero sí reconocerlo. Como cuando viajás a otro país y sabés que "menu" y "wifi" te van a salvar.

Así que, para cerrar, te dejamos esto:

Mini-glosario (versión antipánico)

- **Stack**: conjunto de tecnologías que se usan para construir tu producto.
- **Frontend**: lo que ve y usa el usuario (pantallas, botones, etc).
- **Backend**: lo que pasa "detrás": la lógica, los datos, los servidores.
- **API**: forma en la que dos sistemas se comunican entre sí.
- **Framework**: una estructura o "marco de trabajo" para desarrollar más rápido.
- **Uptime**: porcentaje de tiempo en que una plataforma está funcionando sin caerse.

Próximo capítulo:

"Cómo liderar un proyecto tech sin ser técnico"

Tips para coordinar, dar seguimiento y tomar decisiones sin saber programar

Capítulo 5

Cómo liderar un proyecto tech sin ser técnico en software

Si sos fundador y no tenés experiencia técnica, no estás solo ni estás en desventaja. Al contrario: muchos líderes de startups exitosas arrancaron sin saber escribir una línea de código. Lo que sí tenés que tener claro es que tu rol no es “hacer el código”, sino **liderar el proyecto**, tomar decisiones clave, motivar al equipo y garantizar que todo avance en la dirección correcta.

Este capítulo es para vos, que necesitás herramientas prácticas para manejar un equipo técnico sin volverte loco ni sentir que el tema “te supera”.

1. Entendé tu rol real en el proyecto tecnológico

Tu valor no está en programar, sino en:

- Definir **qué problema estás resolviendo** y cuál es la visión del producto.
- Comunicar claramente esa visión y las prioridades.
- Tomar decisiones basadas en información clara y oportuna.
- Gestionar recursos (tiempo, dinero, gente).
- Ser el puente entre el equipo técnico y otros stakeholders (inversores, usuarios, socios).

No esperes saber todo de tecnología, pero sí aprende a hacer las preguntas correctas y a interpretar lo que te dicen.

2. Comunicate claro y simple

El lenguaje técnico puede ser un laberinto. Por eso:

- Pedí siempre explicaciones en palabras simples, con ejemplos.
- No tengas miedo de pedir que te repitan o aclaren algo.
- Usa metáforas o analogías para entender conceptos complejos (por ejemplo, “el backend es como la cocina de un restaurante, y el frontend es el salón donde atienden a los clientes”).

La comunicación clara evita malentendidos y hace que el equipo trabaje alineado.

3. Pide actualizaciones regulares pero breves

La clave para mantener el control sin ahogarte en detalles es la **disciplina en las reuniones de seguimiento**.

¿Cómo?

- Agenda reuniones cortas, de máximo 15 minutos.
- Que cada miembro del equipo responda a tres preguntas:

- ¿Qué logré desde la última vez?
- ¿Qué me está bloqueando o complicando?
- ¿Qué voy a hacer antes del próximo encuentro?

Estas reuniones pueden ser semanales si el proyecto es muy activo o quincenales para proyectos con ritmo más pausado.

Lo importante es que sean puntuales, directas y sin entrar en tecnicismos innecesarios. Esto te permite detectar problemas a tiempo y mantener el proyecto en marcha.

4. Usa herramientas simples para el seguimiento

No hace falta usar software complejo ni gastar en herramientas caras. Algunas opciones muy útiles y sencillas:

- **Trello o Asana**: para organizar tareas en tableros y listas. Podés ver en qué está cada persona, qué está pendiente y qué ya se hizo.
- **Google Docs o Notion o ClickUp**: para documentación compartida, donde se dejan las decisiones, especificaciones y avances.
- **Slack o WhatsApp**: para comunicación rápida y ágil.

El secreto está en que todo quede registrado y accesible para vos y el equipo, evitando que algo se pierda en mails o chats dispersos.

5. Aprendé a priorizar

No todo puede hacerse a la vez ni es necesario desde el primer día.

Tu rol es ayudar a definir **qué es realmente urgente e importante**, y qué puede esperar.

Para eso:

- Enfocate en entregar valor rápido: ¿qué funcionalidad es la mínima necesaria para que un usuario use el producto y te dé feedback real?
- Evitá el "síndrome del feature infinito" donde se agregan funciones que complican y demoran el lanzamiento.
- Preguntale al equipo si alguna tarea puede simplificarse o demorarse sin afectar el objetivo principal.

6. Tomá decisiones informadas, no inmediatas

Cuando te planteen una opción técnica, evitá decidir en caliente. Pedí que te expliquen pros y contras de forma sencilla, y tomá un tiempo para pensarlo.

Si no entendés algo, decílo sin miedo. Lo peor que podés hacer es fingir y tomar una decisión a ciegas.

Recordá: un buen equipo técnico valora que preguntes y te involucres, no que "desconozcas todo" ni que te impacientes.

7. Delegá lo que puedas, pero mantené el control

No podés (y no debés) controlar cada detalle. Por eso:

- Definí un **punto de contacto** claro en el equipo técnico, puede ser un project manager o un desarrollador senior.
- Dejá que esa persona te reporte avances y problemas.
- Confiá en el equipo para que ejecuten, pero pedí resultados medibles.
- Si algo no te queda claro, pedí explicaciones o datos concretos, no solo promesas.

8. Prepárate para imprevistos

Los proyectos tecnológicos tienen sorpresas: errores, demoras o cambios. No te frustres, es parte del camino.

Lo que sí podés hacer es:

- Tener siempre un **colchón de tiempo y presupuesto** para imprevistos.
- Documentar las causas cuando aparecen problemas, para aprender y mejorar la próxima vez.
- Mantener una actitud flexible y de colaboración.

9. Aprendé a leer números y métricas básicas

No necesitás ser un experto, pero sí saber interpretar indicadores clave para tu proyecto, como:

- % de avance de funcionalidades
- Errores (bugs) detectados y corregidos
- Tiempo estimado vs. tiempo real
- Costos acumulados vs. presupuesto

Estas métricas te ayudan a tomar decisiones y negociar mejor con tu equipo o proveedores.

10. No temas pedir ayuda externa

Si te sentís perdido, podés recurrir a:

- Mentores con experiencia tecnológica.
- Consultores externos para auditorías o revisiones puntuales.
- Cursos básicos de tecnología para founders (hay muchos gratuitos).

Lo importante es que nunca estés solo frente a las decisiones que afectan el corazón de tu producto.

En resumen

Liderar un proyecto tecnológico sin saber programar es posible y puede ser exitoso.

Solo necesitás:

- Entender tu rol y responsabilidades.
- Comunicarte claro y pedir explicaciones.
- Mantener seguimiento puntual y ordenado.
- Priorizar y delegar con confianza.
- Aprender lo básico para tomar decisiones informadas.
- Ser flexible y preparar para imprevistos.

Recordá que tu fuerza está en la visión, la gestión y la capacidad de liderazgo.

La tecnología es solo una herramienta para lograr tus objetivos.

Capítulo 6

Lanzamiento: si nadie lo usa, no existe

Hay una verdad incómoda ”**Muchos productos geniales mueren en silencio”.**

Esto es cierto, si, pero ¿cuál es la razón?

Porque se pensó cada pantalla, cada característica, cada detalle pero nadie pensó cómo hacer que llegue a las manos correctas.

Y esto es super normal. Cuando estás metido en construir el MVP, todo lo demás parece secundario. El marketing suena a "después vemos", o algo que se resuelve contratando a alguien que sepa de redes. Pero **ese enfoque es peligroso**. Porque si nadie se entera de lo que hiciste, da igual cuán bueno sea.

Lanzar no es el final. Es el principio de lo verdaderamente difícil: lograr que alguien lo use, lo entienda, lo recomiende y vuelva.

6.1 Marketing desde el día cero (no desde el día del lanzamiento)

El gran error es pensar que el marketing arranca cuando el producto está listo. Pero **el marketing empieza el mismo día en que decidís construir algo**.

Desde ese instante, ya podés empezar a generar expectativa, atraer a los primeros curiosos, y aprender de ellos.

No necesitás tener miles de dólares ni una agencia. Solo mentalidad y **pensar en distribución desde el principio.**

6.2 Empezá con una lista de espera (aunque no tengas nada aún)

Crear una landing page y ofrecer acceso anticipado a cambio de un mail puede parecer **básico pero funciona**.

Empresas como **Clubhouse** y **Superhuman** lo hicieron, y lo usaron como termómetro de interés. Dropbox, en 2008, pasó de 5.000 a 75.000 usuarios en su lista en solo días gracias a un simple video explicativo.

Podés armar una landing con Carrd o Webflow, y usar Waitwhile o Mailchimp para organizar la lista.
No subestimes esto: tener 300 personas esperando tu producto te obliga a lanzarlo, a mejorar tu mensaje y a sentir que no estás solo.

🧪 **Una beta cerrada vale más que 1.000 likes**

Antes de abrir las puertas, **hacé una prueba controlada con usuarios reales**, no con tus amigos ni familia. Invitá a 50 o 100 personas que se parezcan a tu usuario ideal, y pediles que usen tu producto como lo harían en la vida real.

Pero no alcanza con que lo usen. Necesitás **feedback estructurado**:
3 preguntas clave pueden cambiar tu rumbo:

- ¿Qué fue lo más confuso?
- ¿Qué te gustaría que hiciera el producto y no hace?
- ¿Lo recomendarías? ¿Por qué sí o por qué no?

Podés incentivar con acceso extendido, meses gratis, o simplemente con la promesa de participar en algo que están ayudando a crear.
Notion estuvo **dos años en beta cerrada** con equipos de startups, antes de abrirse al público.

6.3 Conseguir visibilidad sin pagar una agencia de prensa

Muchos founders creen que salir en medios es cosa de contactos o agencias caras. Pero la prensa de nicho está más cerca de lo que pensás.

Sitios como **Product Hunt**, comunidades como **Indie Hackers**, y subreddits específicos de tu industria son excelentes lugares para lanzar. Si tu producto resuelve un problema concreto, **hay una comunidad que quiere saberlo.**

¿Un truco que funciona?

Buscá personas que hayan escrito sobre temas parecidos y escribiles un mail directo. Deciles:

"Vi que cubriste X, estoy lanzando algo relacionado que resuelve Y. ¿Te interesa verlo?"

Lo importante es que se note que hiciste la tarea. Nadie quiere un mail genérico.

6.4 Después del lanzamiento: no solo midas, interpretá

Lanzaste. Llegaron los primeros usuarios. ¿Y ahora?

Ahora empieza el aprendizaje más valioso. No te ciegues con métricas de vanidad (“tuvimos 10.000 descargas”).

Mejor preguntate:

¿Cuántos siguen usando el producto después de una semana?

¿Qué parte abandonan?

¿Cuánto me cuesta conseguir cada nuevo usuario?

Algunas métricas básicas te dan un termómetro real:

- **Retención D7:** cuántos siguen usando el producto 7 días después.

- **CAC (Costo de adquisición):** cuánto te cuesta traer un nuevo usuario.
- **LTV (Lifetime Value):** cuánto valor genera cada usuario en promedio.
- **Tasa de abandono (churn):** cuántos se van cada mes.

No necesitás un equipo de analítica. Google Analytics, Mixpanel, Hotjar y hasta una buena hoja de cálculo te pueden mostrar lo que importa.

Growth: el arte de hacer mucho con poco

Cuando tenés poco presupuesto, necesitás ser creativo.

💡 **Diseñá loops virales:**

Que tus usuarios inviten a otros. Calendly creció así. “Compartí tu link para agendar una reunión” se convirtió en promoción gratuita.

📝 **Creá contenido útil:**

Responde preguntas comunes de tu industria con blogs, posts en LinkedIn o newsletters. Si alguien busca en Google "cómo hacer X" y encuentra tu producto, ganaste.

🤝 **Aliate con otros:**

Buscá marcas no competidoras con tu misma audiencia y ofrecé algo conjunto: un webinar, una guía, un pack de bienvenida.

Slack creció integrándose con Trello, Google Drive y más.

🔁 El lanzamiento no termina con el botón "Publicar"

Después del MVP, viene el momento más delicado: decidir qué mejorar, qué eliminar y cómo crecer.

No te enamores de tu producto.
Escuchá a los usuarios, analizá qué features piden más, dónde se traban, y cómo podés monetizar sin romper la experiencia.

Un ejemplo de primeros 6 meses post-lanzamiento:

- **Mes 1-2:** corregí errores críticos y mejorá el onboarding.
- **Mes 3-4:** sumá la funcionalidad más pedida o integraciones clave.
- **Mes 5-6:** empezá a pensar en escalabilidad o monetización real.

Lo importante es seguir avanzando con foco y escucha activa.

En resumen

Construir un buen producto es solo la mitad del camino.
La otra mitad es lograr que ese producto llegue, se entienda y se use.

No dejes el marketing para el final.

No lo tercerices ciegamente.

No creas que va a pasar solo porque "es tan bueno que se vende solo".

Tu producto necesita una historia, una audiencia y una estrategia para crecer.

El mejor producto del mundo, sin marketing, **es un secreto bien guardado**.

Capítulo 7

Tu startup no necesita que sepas programar. Necesita que sepas liderar.

Llegaste hasta el final. Y éso, en este mundo de distracciones, ya dice algo de vos.

Este libro no buscó convertirte en desarrollador. Tampoco enseñarte a armar apps en 10 días.
Lo que buscó es más profundo:

👉 **Que te animes a liderar en un terreno desconocido sin sentirte impostor.**

👉 **Que te lleves un mapa mental, una brújula, una estructura para no quedarte paralizado.**

🧭 **Mapa Mental**

- **Capítulo 1** – *Tu rol no es programar*

  Es definir la visión, el foco y la cultura del producto.

  Aprendiste a dejar de lado la ansiedad por “no ser técnico”.

- **Capítulo 2** – *Validar primero, construir después*

  El MVP no es una versión chiquita del producto, es un experimento para aprender.

  Las ideas no se validan en tu cabeza: se validan en la calle.

- **Capítulo 3** – *Presupuesto, decisiones y caminos reales*

Viste cómo ajustar los recursos a tus etapas.
Aprendiste que gastar menos no siempre es ahorrar, y que el bootstrapping es una opción válida.

- **Capítulo 4** – *¿Qué stack uso? ¿Y si me equivoco?*

  Entendiste que no hay una única tecnología correcta.
  Que la comunidad, la madurez del stack y tu equipo pesan más que la moda.

- **Capítulo 5** – *Liderar sin código, pero con claridad*

  Viste cómo dar seguimiento, cómo comunicarte con un equipo técnico, cómo priorizar y cuándo pedir ayuda externa.

- **Capítulo 6** – *Lanzar es el inicio, no el final*

  El marketing no es "publicar en redes". Es pensar desde el día cero cómo vas a llegar al usuario correcto.
  Aprendiste a testear, recolectar feedback, y crecer con creatividad (no con plata).

Recordá esto:

Liderar no es saberlo todo, sino animarse a avanzar incluso cuando hay incertidumbre. Es rodearse bien, tomar decisiones con criterio y nunca perder de vista para quién estás construyendo.Tu startup no necesita que seas un

experto en código. Necesita que seas un experto en escuchar, priorizar y seguir adelante. Y ahí, vos tenés más ventaja de la que creés.

🚀 ¿Y ahora qué?

Todo esto está buenísimo, pero no alcanza con leer. El juego real empieza cuando aplicas lo que entendiste.

Probá esto ahora mismo:

✅ 1. Escribí una decisión que venís postergando por “no saber de tech”.

> ¿Elegir equipo? ¿Lanzar? ¿Buscar ayuda?
> Esta semana, avanzá **un solo paso**. Lo demás se va a acomodar.

✅ 2. Releé tu parte favorita y compartila con otro founder.

> Este libro nació para ser compartido. Para que más founders se animen a meterse en proyectos tech sin miedo.

✅ 3. Sumate a la comunidad

Estamos armando un espacio para founders. Un lugar donde compartimos recursos, aprendizajes y experiencias sin filtro.

📲 Seguinos en redes para no perderte nada:

- LinkedIn

- Instagram
- Newsletter

🤝 Acá no se trata de saber más. Se trata de **acompañarse mejor**.

🧡 Gracias por leer

Gracias por confiar en que había otra forma de entender lo técnico. Gracias por apostar a construir, incluso cuando el camino no es claro.

Si este libro logró que tomes una decisión, que salgas del limbo, o que entiendas algo que parecía chino… Entonces ya valió la pena.

Nos vemos construyendo cosas que el mundo necesita. Con más propósito que miedo. Con más claridad que dudas. Y con la certeza de que no necesitás saber todo para empezar.