CENTRO DE E-LEARNING UTN BA
Medrano 951 CABA, Buenos Aires Argentina
(1179) // tel +54 11 7078 – 8073 / fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Documento Clase 6: IA para generar código sin conocimiento previo I

Material de lectura — Diplomatura IA para No Programadores

---

1. Introducción a la generación de código con IA sin experiencia previa

La inteligencia artificial está transformando la forma de desarrollar software. **¿Es posible crear aplicaciones sin saber programar?** Hoy en día, gracias a herramientas *No-Code/Low-Code* y asistentes de IA, **cualquier persona** puede convertir una idea en una aplicación funcional sin escribir código de forma tradicional[1][2]. Esto democratiza la innovación, permitiendo que profesionales de cualquier área experimenten con soluciones tecnológicas.

En esta guía de estudio ampliaremos los contenidos de la Clase 6 de la *Diplomatura en IA para No Programadores*, titulada **"IA para generar código sin conocimiento previo I"**. Mantendremos la estructura temática original y profundizaremos en cada concepto clave, con lenguaje claro y didáctico. Encontrarás explicaciones accesibles, ejemplos concretos, analogías simples, esquemas visuales y **ejercicios prácticos** para aplicar lo aprendido. Además, incluimos casos de estudio reales (argentinos e internacionales) y recursos recomendados en español para seguir explorando.

¿A quién está dirigida esta guía? A profesionales sin experiencia técnica en programación que desean aprovechar la IA y las herramientas No-Code/Low-Code. Verás cómo redactar requerimientos en lenguaje natural, cómo la IA puede generar código (desde sugerir fragmentos hasta construir aplicaciones completas, concepto que llamaremos "*Vibe Coding*"), el rol de plataformas como Firebase en el prototipado rápido, y cómo la IA acompaña el desarrollo en entornos No-Code como *Make*. También discutiremos las diferencias, riesgos y buenas prácticas de estas aproximaciones.

> 💡 **Nota:** Todo está redactado en lenguaje coloquial pero **riguroso**, evitando tecnicismos innecesarios. Al final encontrarás recursos útiles (tutoriales, documentación oficial, etc.) para profundizar por tu cuenta. El objetivo es fomentar una comprensión profunda y brindarte autonomía para explorar estas herramientas de IA y No-Code en tus propios proyectos.

Comencemos por la base de todo desarrollo: definir claramente qué queremos que haga nuestra aplicación, usando **requerimientos en lenguaje natural**.

2. Requerimientos en lenguaje natural: Expresando ideas claramente

Antes de crear cualquier aplicación –ya sea con código tradicional, con herramientas No-Code o con ayuda de IA– es fundamental **saber exactamente qué necesitamos**. Aquí es donde entran los *requerimientos en lenguaje natural*: descripciones claras de lo que esperamos que un sistema haga, escritas de forma comprensible para cualquiera. En esencia, es **especificar nuestra idea con palabras comunes**, sin tecnicismos, de modo que luego una IA o un equipo técnico puedan entenderla y hacerla realidad.

¿Por qué son importantes? Porque traducen nuestras ideas de negocio o proyectos en instrucciones claras. Un buen requerimiento en lenguaje natural mejora la comunicación entre las partes involucradas, reduce ambigüedades y sirve como base para generar documentación e incluso **para la generación de código asistida por IA***[3]*. En otras palabras, si describes bien lo que necesitas, **una IA podrá ayudarte mejor** a construirlo.

2.1 ¿Qué es un requerimiento en lenguaje natural? 🎯

Un requerimiento en lenguaje natural es simplemente la **descripción detallada de una funcionalidad o característica**, redactada como si se la explicaras a otra persona sin conocimiento técnico. Imagina que tienes una idea para una aplicación; ¿cómo se la describirías a un amigo que no programa? Esa explicación, refinada y estructurada, es tu requerimiento.

Objetivo principal: convertir ideas abstractas en instrucciones claras y estructuradas*[4]*. Esto sirve para que todos entiendan la visión del proyecto y se pongan de acuerdo. Si luego usamos una IA para generar código, este requerimiento es básicamente el "*prompt*" o entrada que le daremos a la IA.

Valor que aportan: escribir requerimientos precisos tiene un valor enorme (💎). Por ejemplo:
- Evitan malentendidos entre quien pide una aplicación y quien la desarrolla.
- Sirven como documento base que guía el desarrollo y las pruebas.
- Ayudan a detectar detalles importantes antes de construir nada (si algo es confuso al escribirlo, probablemente lo sería al programarlo).
- Y en nuestro contexto, son **la materia prima para que la IA genere código** o prototipos.

> **Analogía:** Piensa en **encargar un traje a medida**. Si le dices al sastre "quiero algo lindo para una fiesta", el resultado puede no ser lo esperado. En cambio, si especificas "quiero un traje negro, corte clásico, de lana fría, con dos botones y bolsillos inclinados", hay mucha más chance de obtener exactamente lo que imaginas. Lo mismo pasa con una aplicación: mientras más claro y detallado el requerimiento, más se parecerá el producto final a tu idea.

2.2 Componentes de un buen requerimiento

Para redactar requerimientos completos, podemos seguir una pequeña **estructura**. Esto nos asegura cubrir todos los ángulos importantes de la funcionalidad. Los componentes principales de un requerimiento en lenguaje natural son*[5]*:

- **Contexto**: explica *para qué sirve* la aplicación o característica y *quién la usará*. Ejemplo: "Una aplicación móvil **para gestionar tareas diarias**, pensada para personas que necesitan organizar listas de pendientes en equipo". Este contexto da una idea del objetivo y la audiencia.

- **Acción principal**: suele describirse con verbos claros que indiquen la funcionalidad esencial. Por ejemplo: *"Crear tareas, asignarlas a usuarios y marcar cuando estén completadas"*. Palabras como "Generar", "Validar", "Listar", "Enviar", "Calcular", etc., ayudan a definir qué hará el sistema.

- **Datos de entrada**: detalla qué información proporcionará el usuario o de dónde provienen los datos. Por ejemplo: "El usuario ingresará el **título de la tarea**, una **descripción**, y la **fecha límite**. También puede adjuntar un archivo opcionalmente." Aquí especificamos campos, formatos (¿texto, número, fecha?) e incluso ejemplos si hace falta.

- **Salida esperada**: describe cómo se presentarán los resultados o la respuesta del sistema. Ejemplo: "La aplicación mostrará una **lista de tareas** con el título, responsable y estado (pendiente/completada). Puede verse en formato de lista simple o calendario."

Acá podemos mencionar si la salida es visual (una lista, un gráfico), un archivo, una notificación, etc.

- **Criterios de calidad**: son condiciones o restricciones especiales. Por ejemplo: "La aplicación debe responder en menos de 2 segundos al agregar una tarea (límite de tiempo). Debe validar que el título de la tarea no esté vacío (validación), y si falla mostrar un mensaje de error claro. Debe manejar casos excepcionales, por ejemplo, si el servidor no responde, informar al usuario." Estos criterios aseguran un nivel mínimo de calidad y manejan escenarios especiales.

Con estos elementos, un requerimiento queda bien definido. Notemos que **no hay nada de código** aquí, solo lenguaje cotidiano. Sin embargo, si se lo entregamos a un desarrollador, podrá empezar a trabajar. Y si se lo entregamos a una IA (por ejemplo, a ChatGPT con capacidades de programación), tendrá información suficiente para intentar generar una solución.

🎓 *Ejemplo práctico de requerimiento*

Imaginemos que queremos una app sencilla de **lista de tareas colaborativa** (to-do list compartida). Un requerimiento en lenguaje natural podría ser:

- **Contexto:** Aplicación web para que equipos pequeños gestionen tareas diarias de forma colaborativa.
- **Acción principal:** Permite crear tareas, asignarlas a miembros del equipo y marcar las tareas como completadas cuando estén terminadas.
- **Datos de entrada:** Cada tarea tendrá un título (texto corto), una descripción (texto largo opcional), una fecha límite (fecha) y el usuario asignado (seleccionado de una lista de miembros predefinidos). Los usuarios ingresan estas datos mediante un formulario.
- **Salida esperada:** La aplicación mostrará las tareas en una lista, indicando título, responsable, fecha límite y estado (pendiente o completada). Se podrá filtrar por estado o por usuario. Además, enviará una notificación por email al usuario asignado cuando se cree una nueva tarea para él.
- **Criterios de calidad:** No se pueden crear tareas sin título ni asignación. Debe evitar tareas duplicadas con el mismo nombre. La interfaz debe ser sencilla y entendible a primera vista. El tiempo de carga inicial de la aplicación no debe exceder 3 segundos.

Observa cómo abarcamos *qué hace, para quién, qué datos maneja y cómo debe comportarse*. Con un requerimiento así, podríamos directamente preguntarle a una IA: *"Genérame el código de una aplicación web que cumpla con estos requerimientos"*. De

hecho, más adelante veremos que existen agentes de IA (como en **Firebase Studio**) capaces de generar una app completa a partir de descripciones como esta.

2.3 Tipos de requerimientos: funcionales vs no funcionales

Aunque en esta clase nos enfocamos en describir funcionalidades, vale la pena mencionar que hay **dos tipos de requerimientos** en desarrollo de software:

- **Requerimientos funcionales:** Son los que hemos venido describiendo; detallan *qué hace* el sistema. Cada funcionalidad o característica es un requerimiento funcional. Ej: "El sistema permite registrar nuevos usuarios con email y contraseña."
- **Requerimientos no funcionales:** Describen *cómo debe ser* el sistema en términos de calidad, rendimiento, usabilidad, etc. Por ejemplo: "La aplicación debe soportar al menos 100 usuarios concurrentes", o "La interfaz debe seguir las pautas de accesibilidad AA". En nuestro esquema anterior incluimos algunos de estos bajo "criterios de calidad" (como límites de tiempo de respuesta, validaciones, etc.).

En requerimientos en lenguaje natural para IA, normalmente combinamos ambos en la descripción. Es útil mencionar no solo qué debe hacer la app, sino también las expectativas de rapidez, seguridad, experiencia de usuario, etc., en la medida en que impacten la percepción del producto.

> 💡 **Consejo:** Al escribir tus requerimientos, **ponte en el lugar del usuario final**. Describe la experiencia desde que el usuario abre la app hasta que logra su objetivo. Esto ayuda a no olvidar pasos o detalles. Por ejemplo, ¿el usuario debe iniciar sesión? ¿Qué pasa si olvida su contraseña? Si estos detalles son importantes para tu idea, inclúyelos en la descripción de forma clara.

2.4 Ejercicio 1 – Proyecto con IA: redactando un prompt

Ahora que comprendes la estructura de un requerimiento natural, ¡vamos a practicar! Este ejercicio es breve (unos 15-20 minutos) y muy útil para *aterrizar* tus ideas abstractas en algo concreto que luego una IA pueda usar.

Ejercicio: Redacta un **prompt (requerimiento en lenguaje natural)** para documentar una idea sencilla de aplicación. Puedes usar la idea de la app de lista de tareas colaborativa u otra que se te ocurra.

Sigue estos pasos:

1. **Piensa una idea sencilla** de aplicación que te interese. Ejemplos: una app de lista de tareas compartida (como el ejemplo), un sistema para llevar gastos personales, un chatbot para responder preguntas frecuentes de un negocio, etc. Debe ser algo acotado para hacerlo en poco tiempo.
2. **Escribe el requerimiento** de esa idea, incluyendo:
3. Breve contexto de para qué sirve y quién la usaría.
4. Lista de funcionalidades o pasos principales (acción principal y quizá funcionalidades secundarias).
5. Principales datos de entrada y salida que manejará.
6. Cualquier criterio especial de calidad o restricción importante.
7. **Revisa y mejora**: Lee tu descripción y pregúntate si alguien no técnico la entendería. ¿Quedó algo ambiguo? ¿Podría la frase "hacer X" interpretarse de dos formas? Si es así, acláralo. Asegúrate de que no falten partes clave (usa la lista de componentes para verificar).
8. (Opcional) **Prueba con una IA**: Si tienes acceso a ChatGPT u otro asistente, podrías pegarle tu requerimiento y pedirle algo como "¿Qué te parece este requerimiento? ¿Lo entiendes? ¿Crees que falta algo importante?". A veces la propia IA puede señalarte detalles que podrías añadir.

¿Por qué este ejercicio? Porque redactar claramente tus ideas es el primer paso para que la IA pueda ayudarte. Esta práctica te ayuda a **transformar ideas abstractas en requerimientos claros** y útiles para la IA*[6]*. En clases posteriores, esos prompts bien escritos se convertirán en la base para generar código automáticamente.

Cuando tengas tu requerimiento listo, guárdalo: lo usaremos en el siguiente ejercicio para *prototipar* una aplicación real con ayuda de herramientas No-Code y IA.

Hasta aquí, hemos cubierto cómo definir el **qué** queremos. A continuación veremos el **cómo** la inteligencia artificial entra en juego para generar código a partir de estos requerimientos, explorando dos modalidades: **AI Code** y **Vibe Coding**.

3. IA para generar código: de la idea al programa

La premisa de "IA para generar código" suena casi mágica: *describir lo que quieres y que la máquina escriba el programa por ti*. En la práctica, ya es posible en cierta medida. La IA puede participar en el desarrollo de software principalmente de **dos formas***[7]*:

- 🤖 **AI Code** – Asistencia puntual en la escritura de código.
- 🚀 **Vibe Coding** – Desarrollo *casi autónomo* con la IA como co-programadora.

Estas dos aproximaciones no son excluyentes, de hecho se complementan. Veamos en qué consisten, sus ventajas, limitaciones y cuándo conviene usar cada una.

3.1 AI Code: la IA como asistente de programación

En el modo **AI Code**, usamos la inteligencia artificial para **asistir en tareas específicas de codificación**. Es decir, la IA actúa como un ayudante que sugiere, corrige o genera *fragmentos* de código, pero normalmente es el usuario quien integra esas partes dentro de un proyecto más grande.

Características clave de AI Code*[8]*: - Es ideal para usuarios que tienen **nociones básicas de lógica o programación**, aunque sean mínimas. No necesitas ser un experto, pero ayuda saber qué quieres lograr en términos de lógica. - Sirve para resolver problemas puntuales: por ejemplo, pedir ayuda con una función que valide un email, generar un pequeño script para procesar datos, o corregir un error en tu código existente. - También es útil para **escribir pruebas** (tests) de código automáticamente, o generar documentación de una función. Herramientas como GitHub Copilot o ChatGPT mismo entran en esta categoría cuando les pedimos *"escribe una función en Python que haga X"*.

¿Cómo se usa en la práctica? Imagina que estás construyendo una automatización en una plataforma Low-Code que te deja insertar fragmentos de código (como *Make, Zapier* o *Power Automate*). Si necesitas transformar una cadena de texto de cierta forma y no sabes programar, puedes describir el problema a ChatGPT: *"En JavaScript, ¿cómo convierto una fecha en formato dd/mm/yyyy a mm-dd-yyyy?"*. La IA te genera ese pedacito de código que luego tú **copias y pegas** en tu herramienta. De este modo, logras tu objetivo sin tener que dominar la sintaxis ni todos los detalles técnicos.

Otro ejemplo: supongamos que tienes un archivo Excel y quieres un script para limpiarlo (quitar duplicados, corregir textos). Puedes pedirle a la IA: *"Genérame un script de Python que lea un CSV, elimine filas duplicadas y corrija que la columna 'Nombre' tenga mayúscula inicial"*. Obtendrás código Python que hace eso; quizás debas ajustar detalles (como el nombre real de la columna), pero la mayor parte del trabajo la hizo la IA.

Ventajas de AI Code: - **Ahorra tiempo** en tareas de codificación repetitiva o estándar. Muchos desarrolladores hoy lo usan para acelerar su trabajo: la IA autocompleta funciones típicas rápidamente. - **Accesibilidad:** para un *no programador*, rompe la barrera de "no sé escribir código" porque la IA te lo escribe. Tú solo debes integrar la

solución. - **Aprendizaje:** de paso, uno va aprendiendo viendo las soluciones que propone la IA. Con el ejemplo de JavaScript, quizá la próxima vez recuerdes cómo formatear fechas.

Precaución: AI Code **no garantiza perfección**. Es necesario *revisar y probar* el código sugerido. Piensa en la IA como un estudiante aplicado: escribe rápido, pero puede cometer errores tontos o suponer cosas incorrectas. Por eso, si bien no necesitas ser programador experto, sí ayuda tener ojo crítico. Por ejemplo, verificar que el código haga lo que pediste, ajustarlo si tu contexto es diferente, y probarlo con datos de ejemplo.

> 📝 **Ejemplo ilustrativo:** Un usuario con conocimientos técnicos mínimos quiere automatizar en *Make* que, al llegar un email con un adjunto, ese archivo se guarde en Google Drive. Sabe que *Make* puede ejecutar un pequeño script para renombrar el archivo según la fecha. Entonces usa AI Code: le pide a ChatGPT *"Dame un código en JavaScript para renombrar un archivo adjunto con la fecha actual más el nombre original"*. La IA le devuelve 5 líneas de código. El usuario las pega en el bloque de código de *Make*, y ¡listo! Ha incorporado funcionalidad personalizada sin escribir código él mismo, solo usando el que la IA generó.

En resumen, **AI Code = IA como asistente** que te ayuda con *partes* del código. Tú sigues armando el rompecabezas general, pero tienes un ayudante que busca las piezas por ti.

3.2 Vibe Coding: la IA como co-programadora (desarrollo guiado por IA)

Ahora subamos la apuesta: **¿Y si la IA pudiera crear la estructura completa de un proyecto por nosotros?** A esto nos referimos con **Vibe Coding**: dejar que la IA genere gran parte del código base y la estructura, actuando como una *"co-programadora"* que trabaja codo a codo con nosotros*[9]*.

En Vibe Coding, la idea es que el usuario no necesita prácticamente experiencia en programación. *Tú aportas la idea y las especificaciones (como las del ejercicio anterior), y la IA construye un prototipo funcional*. Es como tener un desarrollador muy rápido que entiende lo que le pides en lenguaje natural.

Características de Vibe Coding*[10]*: - Permite **crear la estructura completa de un proyecto**: es decir, no solo funciones sueltas, sino todos los archivos, carpetas y configuraciones necesarios. Por ejemplo, si es una aplicación web, generaría HTML/CSS/JS o incluso una estructura de servidores. Si es móvil, podría generar el

código fuente completo para Android. - Está pensado para usuarios **sin experiencia en programación** que necesitan rapidez en el prototipado. Imagina emprendedores o analistas de negocio que quieren testear una idea de app en días, no meses. - Se apoya en plataformas o herramientas especializadas que hacen esto posible. Un ejemplo actual es **Firebase Studio** de Google, que permite ingresar una descripción en lenguaje natural y genera una aplicación funcional en base a ella*[11][12]*. (Más adelante profundizaremos en Firebase.) - También se ha popularizado con interfaces como Replit's Ghostwriter or servicios integrados en IDEs que, con un prompt, generan proyectos enteros.

¿Por qué "Vibe" Coding? Podríamos traducirlo libremente como "codificación por ambiente/intención". La idea es que trasladas la *vibra* o esencia de tu idea a la IA y ella escribe el código. Tú defines la intención y la IA se encarga de los detalles técnicos.

> **Ejemplo real:** En mayo de 2025, un editor de Applesfera (medio tecnológico español) probó crear una app **sin saber nada de programación, en 10 minutos, usando solo ChatGPT y su iPhone***[13][14]*. ¿Qué hizo? Le describió a ChatGPT la aplicación que quería (una lista de tareas, curiosamente el mismo ejemplo que venimos usando), solicitando que el código fuera en HTML sencillo. En segundos, la IA le devolvió todo el código necesario. Luego, con mínimos pasos (copiar ese código en una app de notas de código y alojarlo en un servicio web gratuito), tuvo una aplicación funcional en su teléfono*[15][16]*. *¡Eso es Vibe Coding en acción!* El usuario solo proporcionó la idea y unos detalles; la IA generó el código completo de la aplicación.

Ventajas de Vibe Coding: - **Velocidad inicial impresionante:** Puedes pasar de idea a prototipo en minutos u horas. Como dijo el editor de Applesfera: *"Es como magia, escribí unas ideas y terminé con cuatro prototipos funcionando antes del almuerzo"**[17][18]*. Herramientas como Firebase Studio permiten generar **apps completas con frontend, backend y lógica de IA integrada** a partir de un prompt*[19]*. - **Baja barrera de entrada:** Realmente acerca el desarrollo a gente sin conocimientos técnicos. Solo necesitas saber describir lo que quieres y la herramienta se encarga de lo demás. - **Prototipado iterativo:** Si la primera versión no es exactamente lo que quieres, puedes refinar la descripción o hacer ajustes menores y volver a generar. Esto permite experimentar con distintas ideas rápidamente.

Limitaciones y riesgos de Vibe Coding: - La aplicación generada es un **punto de partida**. No esperes que sea perfecta o lista para producción sin intervención. Muchas veces, el código producido por IA es funcional pero no óptimo: puede ser difícil de mantener, o contener atajos que en una escala mayor serían problemáticos. - **Conocimiento técnico para mantenerlo:** Si el proyecto sigue creciendo, alguien con

conocimientos deberá entrar a revisar o mejorar ese código. Vibe Coding te lleva del 0 al 1 muy rápido, pero del 1 en adelante (mejoras, escalabilidad, optimizaciones) puede volverse complicado si no tienes bases de programación. Por eso se dice que es "rápido al inicio, pero puede encontrar limitaciones según la plataforma y el perfil del usuario"*[20]*. - **Pérdida de gobernanza:** Este término se refiere a perder un poco el control o entendimiento de lo que se ha construido. Si dejas todo en manos de la IA, es posible que luego no sepas explicar cómo está hecho tu sistema, qué decisiones arquitectónicas tomó, etc. A pequeña escala no importa, pero en proyectos grandes es un riesgo: *¿quién mantiene esto? ¿qué pasa si algo falla?*.

AI Code vs Vibe Coding – una comparación rápida: Ambas son formas de usar IA en desarrollo, pero: - *AI Code* es **más escalable** en la medida que el usuario puede ajustar y entender el código generado*[21]*. Es decir, la IA te da un empujón, pero tú retienes el control y el conocimiento del sistema. - *Vibe Coding* es **más autónomo al inicio**, te entrega mucho de golpe. Genial para un prototipo o MVP (Producto Mínimo Viable), pero a la larga puede requerir intervención técnica para sostenerlo. Existe el *riesgo de pérdida de gobernanza al escalar*[22], porque si todo lo hizo la IA y no hay documentación clara, se vuelve difícil de manejar cuando crece.

> **Analogía:** Supongamos que quieres una casa. *AI Code* sería como **tener herramientas eléctricas y asistentes**: tú diriges la construcción y haces partes (aunque no seas experto, las herramientas facilitan), pero puedes llamar a un ayudante para tareas concretas (poner azulejos, instalar una puerta). *Vibe Coding* sería como **encargarle la casa a una impresora 3D gigante**: le das el diseño y ella levanta la casa por ti. Obtienes la casa rápido, pero quizás desconozcas detalles de la estructura, y si luego quieres hacer una modificación grande, necesitarás consultar a un arquitecto. En la primera, tú tuviste más control durante el proceso; en la segunda, delegaste casi todo y solo supervisaste el resultado final.

En conclusión, **Vibe Coding** nos muestra un futuro emocionante: aplicaciones generadas al vuelo por IA a partir de solo ideas. Pero es importante entender que sigue siendo necesario criterio humano para asegurar calidad, seguridad y mantenimiento. Por eso, en contextos profesionales, suele usarse Vibe Coding para prototipos que luego un equipo técnico refinará. En cambio, para pequeños proyectos personales o soluciones puntuales, podría ser suficiente tal cual.

En la siguiente sección exploraremos una herramienta concreta que *combina* lo mejor de ambos mundos: **Firebase**, una plataforma de Google que acelera el desarrollo (*No-Code*/*Low-Code*) y que ya integra funciones de IA para ayudarnos a crear aplicaciones rápidamente.

4. Firebase y el prototipado rápido: tu laboratorio de experimentación

Desarrollar una aplicación completa involucra muchas piezas: interfaz, base de datos, autenticación de usuarios, servidores, etc. Para alguien sin experiencia técnica, esto suena abrumador. Aquí es donde plataformas como **Firebase** brillan: ofrecen un conjunto de servicios ya listos (*backend* en la nube, base de datos, autenticación, hosting, funciones en la nube, etc.) que puedes usar fácilmente, muchas veces con clicks en una consola web o unas pocas configuraciones. Firebase es parte del ecosistema de Google y se ha vuelto muy popular para crear **prototipos rápidos** e incluso productos finales.

¿Cómo ayuda Firebase al enfoque No-Code/Low-Code con IA? Según lo visto en clase, Firebase permite combinar el enfoque visual/simplificado con generación automática de código base (**scaffolding**)*[23]*. Veamos esto en detalle.

4.1 ¿Qué ofrece Firebase?

Firebase se podría describir como una **plataforma de desarrollo de aplicaciones** todo-en-uno. Algunas de sus capacidades relevantes son*[24]*:

- **Generador de código inicial (scaffolding):** Con herramientas como la interfaz de Firebase Studio o el CLI (Command Line Interface) de Firebase, puedes crear la estructura base de un proyecto con unos pocos comandos o clics. Por ejemplo, indicar “quiero una app web con hosting y Firestore (base de datos)” y Firebase te genera el esqueleto del proyecto, los archivos de configuración y ejemplos mínimos de código para empezar.

- **Plantillas predefinidas:** Firebase (especialmente con Firebase Studio) proporciona *templates* para distintos tipos de aplicaciones: web, móviles o backend (API). En Firebase Studio en su galería de plantillas encontrarás proyectos de inicio en React, Angular, Flutter, Node.js, etc., ya configurados*[25]*. Esto ahorra mucho tiempo porque en lugar de comenzar de cero, partes de algo funcional.

- **Servicios listos para usar:** Necesitas una base de datos en tiempo real o un sistema de autenticación de usuarios? Firebase te ofrece **Firestore** (BD NoSQL) y **Firebase Authentication** con integración muy sencilla, casi sin código. ¿Quieres ejecutar lógica en la nube sin montar servidores? Ahí están las **Cloud Functions** (funciones serverless) que se disparan con eventos. ¿Quieres almacenar archivos (imágenes, pdf)? Tienes **Firebase Storage**. Todo eso **con mínimo esfuerzo de programación**: muchas configuraciones se hacen desde la consola web y la integración con tu app suele ser mediante SDKs simples.

- **Extensiones listas para usar:** Firebase Marketplace ofrece **extensiones** que agregan funcionalidades comunes con un clic. Por ejemplo, hay extensiones para integrar pagos con Stripe, para generar *thumbnails* de imágenes automáticamente al subir una imagen, para traducir texto usando Google Translate, etc. En lugar de programar esas cosas, instalas la extensión y listo*[26]*.

- **Hosting y despliegue sencillo:** Cuando tienes una app web, Firebase Hosting te permite desplegarla globalmente con un comando, y te da un dominio gratuito. Olvídate de configurar servidores web, certificados SSL, etc. En minutos tu aplicación está accesible en Internet.

Todo esto hace de Firebase un **aliado para el prototipado**. Puedes enfocarte en las funcionalidades de tu idea sin pelearte con infraestructura. Por eso se recalcó en clase que Firebase acelera el desarrollo y se complementa muy bien con IA: la IA puede ayudarte a diseñar o ajustar partes de la app, mientras Firebase se encarga de proveer los cimientos sólidos*[27]*.

4.2 Firebase Studio: IA integrada en el desarrollo

Merece mención especial **Firebase Studio**, la nueva interfaz de Firebase que lleva el concepto de Vibe Coding a la realidad. Firebase Studio es un entorno de desarrollo en el navegador potenciado con IA (modelo *Gemini* de Google) que te permite **prototipar aplicaciones enteras con prompts en lenguaje natural***[12]*. Está actualmente en fase *Preview*, pero ya es funcional y muy prometedor.

¿Qué puedes hacer con Firebase Studio?
- Usar el **App Prototyping Agent**: un agente de IA donde le describes tu aplicación (como en el ejercicio que hiciste de requerimientos) y él genera todo el proyecto. Acepta incluso **prompts multimodales***[28]*, o sea, podrías darle no solo texto sino dibujos o esquemas de interfaz, y lo toma en cuenta. Este agente puede configurar por ti la base de datos, autenticación, etc., si tu app lo necesita*[29]*. Literalmente, *genera apps enteras con prompts*. - Alternar a **modo código**: Firebase Studio también tiene un IDE completo (basado en VS Code) en la nube*[30][31]*. Esto significa que puedes empezar sin código (solo con prompts) y luego, si quieres afinar detalles, pasar al editor y escribir o modificar código tú mismo. Todo en el navegador, sin instalar nada. - **Asistencia constante de IA**: Mientras codificas en Firebase Studio, tienes a *Gemini* ayudando: te completa código, te sugiere correcciones, te ayuda a debuggear, escribir pruebas, etc*[32]*. Es como tener Copilot pero integrado profundamente con los servicios de Firebase. Incluso puedes hacer preguntas en lenguaje natural del tipo "¿Cómo agrego login de Google a mi app?" y el asistente te guía o escribe el código necesario. - **Despliegue integrado**: Cuando la app está lista, puedes publicarla

directamente a Firebase Hosting desde el mismo entorno[33]. Todo el ciclo de vida (prototipo -> ajuste -> prueba -> deploy) se maneja ahí.

Firebase Studio realmente materializa el concepto de desarrollo acompañado de IA y **sin configuración compleja**. Para usarlo necesitas una cuenta de Google, activar Firebase en un proyecto, y actualmente un key de acceso a Gemini (mientras está en preview). Varios tutoriales ya muestran cómo en pocos pasos la gente crea aplicaciones web funcionales simplemente describiendo lo que quieren[17][34].

> 📹 *Por ejemplo*, existe un **tutorial en español** en YouTube titulado "Cómo crear una APP con IA – Tutorial fácil de Google Firebase Studio" donde se demuestra la creación de una app móvil en minutos usando esta herramienta. La IA generó la interfaz, la lógica y conectó todo con Firebase sin que el usuario programara una sola línea. *(Puedes buscar ese video para ver la herramienta en acción.)*

En resumen, Firebase nos da el terreno fértil y las semillas para plantar nuestra aplicación rápidamente. Y con Firebase Studio, incluso nos presta el jardinero robot con IA 😊. Pero aunque suene todo automático, es importante que **tú definas bien qué quieres** (de nuevo, los requerimientos claros) y que aproveches estos recursos de manera inteligente.

A continuación, pondremos esto en práctica con un ejercicio donde crearás tu primera **Vibe App**: un prototipo simple de tu idea usando Firebase. ¡Manos a la obra!

5. Ejercicio 2 – Tu primera Vibe App con Firebase

En este ejercicio construiremos un prototipo sencillo en Firebase a partir de los requerimientos que redactaste en el ejercicio 1. El objetivo es **vivir la experiencia de crear una aplicación funcional sin programar**, aprovechando las herramientas de Firebase (y su IA si está disponible). No te preocupes, iremos paso a paso. Dedica unos 20 minutos aproximadamente.

Ejercicio: Crear una versión básica (un MVP – *Minimum Viable Product*) de tu aplicación usando Firebase. Tomemos como ejemplo la app de lista de tareas colaborativa; si tu idea fue otra, puedes seguir los mismos pasos adaptándolos.

Pasos a seguir:

1. **Accede a Firebase Studio (o Firebase Console):** Abre tu navegador y entra a *Firebase Studio* con tu cuenta de Google. Si Firebase Studio aún no está disponible

para ti, puedes usar la *Firebase Console tradicional* – en cuyo caso haremos algunos pasos manualmente.

2. **Crea un proyecto en Firebase:** Una vez dentro, haz clic en "Crear proyecto". Ponle un nombre (por ej. *"mi-lista-tareas"*). Firebase tardará unos segundos en aprovisionar el proyecto. (En Firebase Studio, esto crea un *workspace* nuevo).

3. **(Si usas Firebase Studio) Genera la app con IA:** Busca la opción de *App Prototyping* o algo similar. Allí verás un campo para ingresar una descripción en lenguaje natural. **Ingresa el prompt describiendo tu aplicación**. Puedes usar casi textualmente el requerimiento que escribiste. Ejemplo: *"Quiero una aplicación web para gestionar tareas en equipo. Debe permitir crear tareas con título, descripción, fecha y responsable; marcar tareas como completadas; y mostrar una lista de tareas filtrable. Usar Firestore para almacenar las tareas y autenticación de usuario por email."* Luego, inicia la generación. La IA de Firebase comenzará a crear el proyecto: verás cómo arma la estructura de archivos, configura la base de datos, etc. Ten un poco de paciencia hasta que indique que ha terminado o que la app está lista para probar.

4. **(Si usas Firebase Console normal) Configura servicios básicos:**

5. Ve a la sección **Firestore** y crea una base de datos (modo de prueba está bien por ahora). Esta BD almacenará tus tareas.
6. Ve a **Authentication** y habilita el método de correo electrónico/contraseña para permitir usuarios (opcional, si quieres login).
7. Ve a **Hosting** y activa el hosting (te dará instrucciones para desplegar, las usamos en un momento).
8. Descarga o prepara un **template de la app**: Puedes usar un ejemplo muy básico de app web. En la documentación oficial hay un *tutorial de inicio rápido de Firebase con hosting y funciones* que incluye un index.html de muestra*[35]*. Alternativamente, puedes pedir a ChatGPT que te genere un *index.html* y un pequeño script JS que permita agregar tareas a Firestore y mostrarlas en una lista. *(Esta segunda opción ya es aplicar AI Code dentro de Firebase.)*
9. En tu computadora, instala la herramienta de Firebase CLI (si no la tienes) y ejecuta *firebase login* (para vincular con tu cuenta) y luego *firebase init* dentro de una carpeta vacía. Selecciona configurar Hosting y Firestore. Esto creará archivos de configuración locales.
10. Copia el archivo *index.html* de tu app (modifícalo según tu requerimiento, por ejemplo usando el código que te generó ChatGPT para lista de tareas).
11. Ejecuta *firebase deploy* para subir tu app a Firebase Hosting.

12. **Prueba la aplicación:**

13. Si usaste Firebase Studio, probablemente tengas un botón de *Preview* o una URL temporal donde ver tu app corriendo. Utilízala y prueba crear, ver, completar tareas (o la funcionalidad de tu idea). Comprueba que la base de datos y otras funciones funcionen según lo esperado.
14. Si usaste la vía manual, tras el deploy Firebase te dará una URL (algo como *https://mi-lista-tareas.web.app*). Ábrela en el navegador y prueba la aplicación.
15. **Itera si es necesario:** ¿Algo no funciona del todo bien? Por ejemplo, quizá la lista no se actualiza automáticamente cuando agregas una tarea. Puedes iterar: en Firebase Studio, podrías describir ese problema al agente para que lo corrija. En la vía manual, podrías pedirle a ChatGPT que mejore el código (AI Code) o habilitar *Firebase Cloud Functions* para notificaciones en tiempo real. *Mantén la calma:* la idea no es lograr un producto perfecto, sino explorar qué tan lejos llegas con las herramientas disponibles sin programar tradicionalmente.
16. **Comparte o documenta tu resultado:** Aunque sea un prototipo, es valioso anotar qué lograste. Puedes hacer capturas de pantalla de la app funcionando o anotar cuántos pasos hiciste manualmente vs cuántos hizo la IA. Esto te servirá para reflexionar sobre la experiencia.

🔍 **Consejo:** Si enfrentas dificultades técnicas (por ejemplo, errores de configuración), no dudes en buscar en Google o en la documentación oficial de Firebase. La comunidad Firebase es grande y muchos problemas comunes ya están resueltos en foros. Parte de ser autónomo con estas herramientas es aprender a *encontrar respuestas* cuando algo no sale a la primera.

¿Qué se espera de este ejercicio? Que obtengas una **mini-aplicación funcional** basada en tu idea inicial. Puede que no tenga todos los detalles ni un diseño pulido, ¡pero fue creada en muy poco tiempo! Piensa que, tradicionalmente, esto requeriría saber programar en frontend y backend, configurar un servidor, etc. Tú lo lograste básicamente con una buena descripción y aprovechando Firebase (y posiblemente ChatGPT).

Esta experiencia consolida varios conceptos: - La importancia de un requerimiento claro: fue la *receta* que siguió la herramienta. - El poder de las herramientas No-Code/Low-Code como Firebase para cubrir la complejidad técnica. - Cómo la IA (en Firebase Studio o ChatGPT) puede suplir carencias de conocimiento, permitiéndote avanzar donde te trabarías. - Te habrá dado una idea de las **limitaciones** también: por ejemplo, si la IA malinterpretó algo de tu prompt o si firebase generó algo diferente a tu expectativa, viste que hay que ajustar. Es normal y parte del proceso.

¡Felicitaciones! Has creado tu primera **Vibe App** 😀. Aunque sea sencilla, piénsalo: sin escribir código tradicional, ya tienes un prototipo que podrías mostrar a colegas,

clientes o inversores para comunicar mejor tu idea. En futuras clases aprenderás a interpretar y ajustar los resultados automáticos (como mejor tarea de la Clase 7) y a añadir funcionalidades más avanzadas, pero por ahora has dado un gran paso.

6. IA acompañando el desarrollo *No-Code*: el caso de Make

Hasta ahora hablamos de generar código o apps completas. Sin embargo, la IA también puede ayudarnos en entornos donde *no se escribe código*, es decir, en plataformas **No-Code** puras. Un buen ejemplo es **Make** (antes conocida como Integromat), una herramienta visual para automatizar flujos de trabajo entre distintas aplicaciones. En Make creas integraciones dibujando diagramas: conectas módulos (ej: un disparador de Gmail, seguido de una acción en Google Drive, etc.) para definir un flujo.

En plataformas como Make (o Zapier, n8n, Microsoft Power Automate, etc.), la IA no va a generar una aplicación desde cero (porque *no hay código que escribir* directamente). Pero sí puede desempeñar dos roles muy útiles*[36][37]*: - 🤝 **IA como acompañante** del usuario durante el diseño del flujo. - ⚡ **IA como generadora de "nodos"** o configuraciones que se importan en la herramienta.

Analicemos cada rol.

6.1 IA como acompañante en flujos No-Code

Aquí la IA funciona como **coach o asesor** mientras tú construyes un flujo en Make. Piensa que al ser principiante, podrías no saber por dónde empezar o cómo seguir. Puedes entonces recurrir a un asistente de IA (como ChatGPT) para que te guíe paso a paso.

¿Cómo se ve esto en la práctica? Supongamos que quieres automatizar un proceso: *Cada vez que recibo un email con archivo adjunto en Gmail, guardar ese archivo en Google Drive y luego enviar un aviso por Slack.* No estás seguro cómo implementarlo en Make. Puedes preguntarle a la IA en lenguaje natural, por ejemplo:

> **Prompt de ejemplo:** "Explícame paso a paso cómo armar un flujo en Make que guarde archivos adjuntos de Gmail en Google Drive y luego avise por Slack."*[38]*

Un buen asistente de IA te responderá algo como: "*Primero, en Make, añade un módulo de Gmail con el trigger 'Watch emails' configurado para tu cuenta. Ajusta el*

filtro para que solo capte emails con adjuntos. Luego añade un módulo de Google Drive 'Upload a file' y mapea el adjunto del correo al archivo a subir. Después, añade un módulo de Slack 'Post a message' configurando el texto del mensaje (por ejemplo: 'Archivo ${filename} guardado en Drive') y la canal donde avisar. Finalmente, ejecuta el escenario para probarlo."

Básicamente, la IA te **describe la solución** en términos de la plataforma Make. Esto **fomenta tu aprendizaje y autonomía***[39]*: sigues las indicaciones, construyes el flujo tú mismo en la interfaz de Make, y de paso entiendes qué hace cada paso. La IA compañera puede además: - Sugerir *buenas prácticas* (por ejemplo, "añade una condición para saltar Slack si no hay adjunto, así evitas mensajes vacíos"). - Resolver *dudas puntuales*: Si no sabes qué hace exactamente cierta opción en un módulo, puedes preguntarle. Es como tener un tutor disponible 24/7.

Ventaja: No te da la solución hecha (como en Vibe Coding), sino que te **acompaña didácticamente**. Para un perfil no técnico, esto es oro porque aprendes haciendo, pero con una red de seguridad. Ya no estás solo frente a la interfaz visual; tienes un *copiloto* que te guía.

Consideración: La calidad de la guía depende de qué tan bien formules la pregunta y del conocimiento del asistente sobre la herramienta. ChatGPT, por ejemplo, ha "leído" documentación pública hasta 2021, pero plataformas como Make evolucionan; puede que alguna instrucción quede obsoleta. Siempre valida con la documentación oficial o probando en la plataforma. Afortunadamente, herramientas como Make tienen academias y guías (por cierto, existe *Make Academy* con cursos gratuitos en español y otros idiomas*[40]*, recurso muy útil).

6.2 IA como generadora de configuraciones (nodos JSON)

Make (y otras plataformas similares) suelen permitir exportar e importar flujos o partes de flujos en formato **JSON**. Es decir, toda la configuración de módulos y sus conexiones se puede representar en un texto estructurado (JSON) que la herramienta entiende. Esto abre la puerta a que la IA **genere directamente ese JSON**, para luego importarlo en la plataforma. Es un atajo más técnico, pero potente.

Uso práctico: Retomemos el ejemplo del flujo Gmail -> Drive -> Slack. Si ya sabemos exactamente qué módulos queremos y sus configuraciones, podríamos pedir a la IA: *“Crea el JSON para un nodo HTTP en Make que consulte la API de OpenWeather con mi clave y devuelva la temperatura de Buenos Aires.”**[41]*. Este es el ejemplo dado en clase. Aquí se asume que conoces que Make tiene un tipo de módulo HTTP y cómo más o menos es su JSON. La IA, idealmente, produce algo como:

```
{
  "module": "HTTP",
  "action": "Make a request",
  "parameters": {
    "method": "GET",
    "url":
"https://api.openweathermap.org/data/2.5/weather?q=Buenos%20Aires&appid=TU_API_KEY&units=metric",
    "response_body": true
  }
}
```


Esto es un JSON hipotético de configuración. En Make, tú podrías copiar ese JSON e importarlo como nuevo módulo configurado (Make tiene una opción de *Import Blueprint* donde pegas JSON). Automáticamente tendrías un módulo que llama a la API del clima. Sin escribir ni hacer clic en la configuración manual, la IA lo configuró por ti.

Ventajas y riesgos:
- *Velocidad:* si sabes lo que necesitas, es más rápido que tú mismo rellenar los campos en la interfaz. - *Complejidad:* requiere entendimiento de qué esperar. Puede ser algo avanzado para un usuario totalmente no técnico, porque básicamente es generar *código de configuración*. Si nunca has usado Make, es mejor el enfoque anterior (IA compañera). Si ya lo manejas un poco y quieres ahorrar tiempo, este enfoque brilla. - *Validación:* Al final, tendrás que probar ese módulo importado. Muchas veces el JSON generado necesitará ajustes (por ejemplo, poner tu API key real donde dice TU_API_KEY). Así que nuevamente, la IA no es infalible: hay que revisar que la configuración tenga sentido. Pero te ahorró escribir la estructura.

Ejemplo de prompt y utilidad: Imagina que ya tienes un flujo en Make con 5 módulos y quieres agregar un sexto que convierta moneda usando un API. Podrías describir ese módulo a ChatGPT, obtener el JSON y pegarlo. Te ahorraste buscar en la documentación de la API y la sintaxis del módulo. No obstante, debes entender cómo encajarlo en tu flujo.

Para perfiles semitécnicos, esta capacidad es impresionante: es un nivel de automatización meta, *la IA programando dentro del ambiente No-Code*. Y cabe destacar, **Make anunció integraciones de IA nativas** en 2023, donde incorporarán asistentes dentro de su plataforma para, por ejemplo, generar escenarios a partir de descripciones. Así que estas técnicas manuales quizás se vuelvan funcionalidades integradas pronto.

6.3 Ejercicio 3 – *Desafío IA + Make* (opcional)

Para reforzar estos conceptos, te proponemos (de manera opcional, si tienes tiempo extra) un mini ejercicio práctico:

Ejercicio: Diseña un flujo simple en Make con la ayuda de la IA.

- **Objetivo del flujo:** Elige algo sencillo, por ejemplo: *Tomar mensajes de un formulario de Google Forms y enviarlos por email*; o *Cuando se añade una fila nueva en Google Sheets, mandar ese dato a un chat de Telegram*.
- **Usa la IA como acompañante:** Pregúntale a ChatGPT cómo lo haría, que módulos usar, etc. Sigue las instrucciones para armarlo en Make.
- **Prueba el flujo:** Ingresa un dato en la herramienta disparadora (un envío de form, una nueva fila, lo que hayas elegido) y verifica que la acción suceda (llegue el email, o el mensaje al chat, etc.).
- **Usa la IA generadora (si te animas):** Identifica un módulo de tu escenario que podrías haber creado por JSON. Pide a ChatGPT que te dé el JSON de ese módulo. Luego elimínalo en Make y reintégralo importando el JSON proporcionado. Comprueba que funcione igual.

Este ejercicio te permitirá experimentar ambas formas de ayuda de la IA en un contexto No-Code real. Al final, reflexiona: ¿Te resultó más útil la guía paso a paso o el JSON directo? ¿Dónde viste más riesgo de error? ¿Dónde aprendiste más? No hay respuestas correctas únicas; depende de tu estilo. Lo importante es reconocer que **la IA puede ser tu aliada incluso cuando no estás escribiendo código**, guiándote o acelerando configuraciones.

Con esto, hemos explorado cómo la IA se integra a las plataformas No-Code para **amplificar tus capacidades**. Ya sea sugiriendo cómo armar un flujo o generando configuraciones complejas en un chasquido, tener un "copiloto" de IA abre posibilidades enormes para usuarios no técnicos.

7. Diferencias, riesgos y buenas prácticas

Hemos cubierto varias modalidades: desarrollo tradicional asistido por IA (AI Code), desarrollo guiado completamente por IA (Vibe Coding), plataformas Low-Code, y plataformas No-Code con IA de apoyo. Cada una tiene sus pros y contras. Es útil hacer un **resumen comparativo** para tener un panorama claro, especialmente de cara a proyectos reales en empresas:

- **Vibe Coding:** Genera el código completo por ti a partir de una descripción. **Ventaja:** velocidad inicial y baja barrera técnica; ideal para prototipos. **Riesgos:** el código resultante puede no ser óptimo; *requiere conocimiento técnico para mantenerlo* y escalarlo*[22]*. Si confías ciegamente, puedes perder control sobre tu proyecto (pérdida de gobernanza). Buena práctica: úsalo para experimentar rápidamente, pero involucra a expertos si vas a convertir ese prototipo en un producto serio.

- **Low-Code:** Combina entornos visuales con la posibilidad de añadir fragmentos de código personalizado. Es útil para perfiles *semitécnicos*[42], por ejemplo, un analista que no programa full-time pero sabe hacer pequeñas funciones. **Ventaja:** rápido desarrollo con flexibilidad intermedia – no partes de cero (tienes componentes listos) pero puedes extender funcionalidad con código donde haga falta. **Riesgos:** Puede caer en tierra de nadie: ni tan simple como No-Code puro, ni tan potente como código full. Requiere tener a alguien capaz de escribir esas partes de código cuando se necesite. Buena práctica: definir qué partes críticas necesitan codificación y aislarlas bien, mientras el resto se hace visualmente.

- **No-Code:** Construcción 100% visual, sin escribir código. Ejemplos: crear un sitio en Wix, automatizar con Zapier, hacer una app en Glide. **Ventaja:** cualquiera con algo de entrenamiento puede hacerlo; desarrollo *muy rápido* de MVPs; enfoque en la lógica de negocio más que en detalles técnicos. **Riesgos:** límites de flexibilidad – si lo que necesitas se sale de lo previsto por la herramienta, no podrás hacerlo; dependencia del proveedor; y a veces *costos ocultos* si escalas en volumen (ej: muchos registros pueden implicar un plan pago alto). Además, la **calidad** puede verse limitada: no-code suele estar pensado para soluciones estándar; si necesitas optimización, puede no ser posible. Buena práctica: usar No-Code para validar ideas y cubrir gaps rápidos, pero si la solución prueba ser vital para el negocio, evaluar eventualmente migrar a un desarrollo más a medida o híbrido (low-code).

- **AI Code vs IA en No-Code:** En AI Code la IA produce código que igual requiere ser entendido/integrado por alguien, mientras en No-Code la IA te ayuda sin exponerte código. Para un no programador absoluto, la segunda vía suele ser más amigable. Sin embargo, notarás que las fronteras se difuminan: un usuario de Make que pide JSON a ChatGPT ya está tocando "código" en cierto modo. Siempre es bueno **conocer tus límites y poco a poco ampliarlos**. Quizá hoy te apoyes 100% en No-Code, mañana te animas a copiar un snippet de AI Code en tu flujo, y pasado mañana aprendes un poquito de JavaScript porque te fue útil.

Riesgos generales de usar IA en desarrollo (resumen): - **Alucinaciones de la IA:** a veces te dará respuestas incorrectas o inexistentes, como referencias a funciones que no existen. Por eso, hay que validar todo resultado de IA con prueba o documentación. - **Seguridad y privacidad:** no subas a ChatGPT código o datos sensibles de tu empresa (a menos que uses su versión empresarial con garantías). Tampoco reveles llaves API en los prompts. Mejor, dile "mi clave API es XYZ" y luego pega manualmente

donde corresponda en el código. - **Dependencia excesiva:** es tentador dejar que la IA haga todo. Pero entonces no desarrollas entendimiento. Si la herramienta cambia o falla, puedes quedarte en el aire. Siempre que la IA te dé algo, trata de comprender al menos superficialmente qué hace. Pregúntale si hace falta: *"Explícame el código que me diste"*. Te servirá para aprender. - **Actualizaciones de plataformas:** las herramientas No-Code/Low-Code y APIs cambian. La IA puede no estar actualizada. Combina su ayuda con fuentes confiables (documentación oficial, comunidades). Por ejemplo, si ChatGPT te sugiere un módulo que fue descontinuado en Make, ve a la documentación de Make Academy a confirmarlo*[40]*.

Buenas prácticas recomendadas: - **Documenta tus prompts y resultados:** Si con IA generaste código o configuraciones, anota en comentarios o en un documento cómo lo obtuviste. Esto puede ayudar a otros (o a ti en el futuro) a entenderlo. Ej: en un script generado, pon un comentario "// Generado con ChatGPT, revisado el 10/10/2025". - **Itera gradualmente:** No intentes que la IA te haga un sistema enorme en un solo prompt. Mejor pídele módulos o partes, pruébalas, y luego integra. O usa un enfoque incremental: primero que genere algo básico, prueba, luego ve agregando requisitos. - **Mantente actualizado:** El mundo de IA + No-Code evoluciona rápido. Suscríbete a newsletters, sigue canales de YouTube oficiales (por ej. el [YouTube de Firebase en español]*[35]*), únete a comunidades (como foros de Bubble, Glide, Make). Así te enterarás de nuevas herramientas que quizás hagan tu vida más fácil aún. - **Aprende lo esencial de lógica/programación:** Aunque no programes, conceptos como "variables", "condicionales", "bucles", "APIs", etc., son útiles de entender. Te permitirán dialogar mejor tanto con IA como con programadores humanos. Muchos cursos para no técnicos enseñan "pensamiento computacional" sin código — recomendable para potenciar tu perfil.

Con esta visión más completa, ya estás en condiciones de decidir qué enfoque usar según la situación: - ¿Necesitas algo **rapidísimo para probar una idea**? => No-Code con un toque de Vibe Coding (por ej. Firebase Studio). - ¿Debes **automatizar un proceso interno** en tu empresa sin sobrecargar IT? => Herramientas Low-Code/No-Code (Make, Power Automate) con IA ayudándote a construirlo. - ¿Tienes un **problema puntual de datos o código**? => AI Code te echa una mano generando ese snippet en Python, JavaScript, etc. - ¿Ves que tu **proyecto crece y requieres robustez**? => Quizá pases de No-Code a Low-Code, o de Vibe Coding a involucrar desarrolladores, usando la IA ahora más como asistente de productividad que como generador principal.

El conocimiento que estás adquiriendo te permitirá moverte con flexibilidad por este espectro de opciones.

8. Casos de estudio reales: IA + No-Code en acción

Para aterrizar todos estos conceptos, veamos algunos casos reales de uso, tanto en Argentina como a nivel internacional, donde la combinación de **IA, No-Code y Low-Code** ha producido resultados interesantes. Estos ejemplos muestran desde pequeñas empresas hasta grandes corporaciones aprovechando estas herramientas.

8.1 Caso 1 – PyME argentina: pizzería digitalizada en La Plata

Una **pizzería familiar en La Plata, Argentina**, logró transformarse digitalmente usando herramientas No-Code. Antes gestionaban los pedidos de forma tradicional (teléfono y papel). Decidieron crear una **app de pedidos online** para sus clientes, pero contratar un desarrollo a medida era costoso. Optaron por usar **Glide**, una plataforma no-code que convierte hojas de cálculo en aplicaciones móviles.

Implementación: En aproximadamente **1 semana** configuraron su app vinculada a Google Sheets, donde listan su menú y reciben pedidos*[43]*. La interfaz se personalizó con los colores del negocio y se habilitó una notificación por email para nuevos pedidos.

Resultado: La pizzería vio un incremento de **+150% en ventas online** tras lanzar la app*[44]*. Ahora reciben pedidos las 24 horas (incluso cuando el local está cerrado, para preparar al día siguiente) y gestionan todo desde un simple spreadsheet que entiende cualquier empleado. Este es un gran ejemplo de cómo una *PyME argentina* pudo crear una solución digital **sin saber programar**, aprovechando No-Code. Si bien aquí no hubo IA generando código, demuestra la base: la democratización del desarrollo. Y podemos imaginar que si los dueños quisieran añadir, por ejemplo, un chatbot de recomendación de pizzas, podrían integrarlo con alguna IA usando plataformas como ManyChat u otras mencionadas en la clase*[45]*.

8.2 Caso 2 – Corporate: Roche Argentina y el self-service BI

La empresa farmacéutica **Roche Argentina** buscaba agilizar cómo sus empleados obtenían reportes y analizaban datos internos. Tradicionalmente, para cualquier reporte nuevo dependían del equipo de IT o analistas de datos, lo que tomaba tiempo. Implementaron una solución de **Low-Code** orientada a *Business Intelligence de autoservicio*. Según comentó Diego Branca (Digital & Data Lead de Roche Arg.) a la prensa, montaron una plataforma donde los usuarios de negocio pueden **crear sus propios reportes** basándose en el data warehouse corporativo*[46]*.

Con una herramienta de low-code (no se menciona el producto específico, pero podría ser algo tipo Power BI con funciones de low-code, o Qlik, etc.), los empleados arrastran campos, filtran datos y generan visualizaciones sin escribir SQL ni código.

Impacto: Esto les permite **sacar prototipos de informes o dashboard en días**, probarlos con usuarios, iterar según feedback y eventualmente formalizar soluciones más completas*[47]*. En vez de esperar semanas a que IT desarrolle un reporte oficial, las áreas pueden experimentar con los datos de forma controlada. Es un uso diferente (no es "generar código" directamente) pero se basa en el mismo espíritu: *no depender de desarrolladores para cada necesidad digital*.

Branca destaca que una gran ventaja es que **los usuarios finales participan más en el diseño de la solución** (por ejemplo, eligiendo cómo quieren ver la información)*[48]*. Esto aumenta la adopción y satisfacción, porque el producto final se ajusta mejor a lo que necesitan (ellos mismos lo co-crearon).

Este caso muestra un uso de Low-Code en el ámbito de datos/BI dentro de una corporación en Argentina, generando ahorros de tiempo y costos significativos*[49]*. Si bien no se menciona IA generativa aquí, es fácil imaginar que podrían luego incorporar IA para análisis más avanzado (p.ej., un asistente que en lenguaje natural responda consultas sobre esos datos, apoyándose en esta plataforma). De hecho, la combinación de IA con BI es una tendencia creciente.

8.3 Caso 3 – Corporate: Randstad y desarrollo ágil con Flutter

Randstad, gigante de recursos humanos, también se volcó a herramientas modernas para agilizar ciertas soluciones internas. En Argentina, la directora de tecnología Sandra Boidi comentó que usan **Flutter** (el SDK de UI de Google) como parte de su enfoque Low-Code*[50]*. Cabe aclarar que Flutter en sí es un framework de desarrollo tradicional (requiere programar en Dart), pero al parecer lo usan junto a herramientas visuales (posiblemente FlutterFlow, una plataforma low-code basada en Flutter, o librerías de componentes pre-hechos).

¿Por qué Flutter/Low-Code? Según Boidi, estas herramientas les permiten desarrollar **rápido aplicaciones web y móviles** para necesidades específicas, cuando esas aplicaciones **no involucran lógica demasiado compleja ni reglas de negocio críticas***[51]*. Por ejemplo, podría tratarse de apps internas para coordinaciones, agendas, pequeñas automatizaciones de RH, etc., donde importan más la interfaz y la rapidez que la sofisticación técnica.

Beneficios observados: Velocidad para cubrir necesidades **puntuales** de distintos departamentos sin pasar por largos ciclos de desarrollo. Además, las usan para

automatización de procesos simples en la compañía[52] – es decir, no todo pasa por implementaciones en los grandes sistemas corporativos, sino que muchas tareas repetitivas se resolvieron con apps/flows hechos con estas herramientas en semanas.

Este caso muestra que incluso empresas grandes confían en enfoques low/no-code para cierto tipo de proyectos, encontrando un balance entre robustez y rapidez. Y sumando IA: no sería extraño que en esos desarrollos en Flutter incorporen **Augmented Coding** (herramienta de Globant para sugerir código con IA, dado que Globant es partner de Randstad) u otros copilotos para hacer más eficiente la tarea de sus desarrolladores. De hecho, la integración de Copilots está ocurriendo en muchas empresas con sus equipos de desarrollo tradicionales[53], mejorando la productividad.

8.4 Caso 4 – Internacional: Airbnb potencia su plataforma sin código

Un ejemplo a nivel internacional muy citado es **Airbnb**. Esta empresa, aunque tiene un equipo enorme de ingenieros, ha sabido aprovechar herramientas No-Code para ciertas facetas de su negocio. Según un artículo del IEBS Business School, Airbnb implementó soluciones NoCode para **gestionar la gran cantidad de datos generados por los usuarios y optimizar su plataforma**[54].

Con No-Code pudieron crear rápidamente **herramientas internas personalizadas** para el manejo de propiedades y la comunicación con anfitriones, sin depender de desarrollo tradicional para cada ajuste[55]. También les permitió experimentar con nuevas funcionalidades **sin ciclos largos de desarrollo**, probándolas internamente de forma ágil antes de decidir invertir recursos de ingeniería.

Este caso demuestra que No-Code no es solo para quienes no tienen desarrolladores, ¡también las big tech lo usan! La razón es productividad: si un *product manager* de Airbnb puede por sí mismo armar una herramienta para visualizar ciertos datos de usuarios y tomar decisiones, ¿por qué esperar a que un equipo de data se lo programe en Python? Le da autonomía y rapidez. Airbnb se mantiene competitivo en parte por esta cultura de innovación continua, y **No-Code ha sido un factor clave para acelerar experimentos e iteraciones** sin sobrecargar su pipeline de desarrollo[56].

Aunque no hay muchos detalles públicos, es plausible que utilicen combinaciones de herramientas como Retool (plataforma low-code para crear dashboards internos) o Zapier para integraciones entre sistemas, etc. La cita de IEBS sugiere justamente eso: conectaron sistemas sin escribir nuevas líneas de código, logrando cambios eficientes.

8.5 Caso 5 – Individual: "*No sé programar y hice una app en 10 minutos*" (España)

Volviendo al caso mencionado de Applesfera, es tan ilustrativo que vale la pena resumirlo como estudio de caso personal:

Quién: Guille Lomener, editor de Applesfera (no desarrollador de profesión).
Qué hizo: En 2025, con cero experiencia en desarrollo iOS, creó una **aplicación web estilo lista de tareas** en su iPhone en menos de 10 minutos usando ChatGPT*[13][14]*.
Cómo: Le pidió a ChatGPT el código HTML/JS para una lista de tareas con almacenamiento local (que guardase las tareas en el navegador). ChatGPT generó el código completo *al instante[57]*. Luego, Guille usó una app llamada Kodex en el iPhone para pegar ese código en un archivo HTML, lo subió a un servicio web (Neocities) directamente desde el móvil, y abrió la "app" en Safari, añadiéndola a su pantalla de inicio*[16][58]*. Tenía así una PWA (app web progresiva) básica funcionando. Todo esto sin tocar una PC ni abrir un editor tradicional.

Resultado: Una app de tareas funcional en su teléfono, creada más rápido de lo que tardaría en explicarla. Por supuesto, él mismo admite que el código generado era muy básico y que los *desarrolladores de verdad se llevarían las manos a la cabeza* ante ciertas cosas*[59][60]*. Pero cumplía su objetivo: le quitó el miedo al código y demostró que con las herramientas actuales cualquiera puede atreverse a crear.

Lecciones: Este caso encarna perfectamente la filosofía *"IA + No-Code para No Programadores"*. Guille actuó como *prompt engineer* dando instrucciones claras a la IA (volvemos a la importancia de los requerimientos bien formulados) y combinó varios servicios accesibles (app Kodex, hosting gratuito) para lograr un fin. Aprendió en el proceso sobre los pasos para llevar una idea a algo tangible, y seguramente inspiró a otros a probar. Además, resalta que aunque la IA haga fácil crear algo, *no significa que ya seas un programador profesional* – hay muchas capas de conocimiento para crear software de calidad – pero sí abre la puerta para que más gente entre a este mundo y entienda sus posibilidades*[61]*.

Estos casos, variados entre sí, comparten una conclusión: **la unión de herramientas No-Code/Low-Code con la IA está generando innovaciones rápidas en múltiples sectores.** Desde un negocio familiar hasta un gigante tecnológico, desde un empleado experimentando con datos hasta un entusiasta tecnológico a nivel personal, todos pueden beneficiarse de esta tendencia. Y en Argentina, en particular, vemos adopción tanto en PyMEs como en corporaciones, lo que augura un ecosistema creciente de soluciones creadas sin código.

9. Recursos recomendados para profundizar

Para cerrar esta guía, te dejamos una recopilación de **recursos útiles** (en español siempre que sea posible) que complementan lo visto en clase. Estos incluyen tutoriales, documentación oficial y comunidades de las herramientas mencionadas, para que puedas seguir aprendiendo y practicando por tu cuenta:

- **Firebase – Canal oficial de YouTube:** Tutoriales y demos sobre Firebase (muchos en inglés, pero con opción de subtítulos en español). Especial atención a listas de reproducción sobre *Firebase Basics* y las nuevas funciones de IA. Puedes encontrarlo como **Firebase en YouTube***[35]*. Ideal para ver casos prácticos y proyectos ejemplo.

- **Primeros pasos con Cloud Functions (Firebase) – Tutorial oficial:** Si te interesa extender Firebase con lógica personalizada, este tutorial te guía para escribir, probar y desplegar tu primera Function paso a paso*[40]*. Disponible en español en la documentación de Google: *"Primeras funciones: escribir, probar y desplegar"*.

- **Make Academy – Cursos gratuitos:** La plataforma Make ofrece su propia academia en línea con cursos que van desde nivel básico hasta avanzado*[40]*. Los cursos están en inglés, pero incluyen materiales visuales fáciles de seguir. Aprenderás desde crear tu primer escenario hasta integraciones avanzadas. Muy recomendable si planeas usar Make en serio.

- **Documentación oficial de Firebase Studio y GenAI:** Dado que Firebase Studio es reciente, Google publicó documentación y guías. Por ahora, en inglés, pero con conceptos valiosos. Busca en Google *"Firebase Studio Overview"* y *"App Prototyping agent Firebase"* para ver cómo describen su uso*[62][12]*. También el blog de Firebase tiene artículos con tips de uso de la nueva herramienta (algunos ya traducidos).

- **Comunidades y foros:** Participar en comunidades puede darte apoyo. En español puedes unirte a foros de **Indie Makers** o grupos de Facebook sobre No-Code. También el subreddit */r/nocode* (en inglés) y */r/AskProgramming* son lugares para plantear dudas. Para IA generativa, el grupo de Discord de OpenAI suele tener canales útiles.

- **Herramientas No-Code de IA destacadas:** Para seguir explorando, aquí algunas plataformas interesantes (mencionadas en artículos como Eseid y Inesdi):

- *RunwayML:* creación de videos e imágenes con IA sin código (genial para proyectos creativos)*[2]*.
- *Pictory:* genera vídeos a partir de texto, útil para marketing*[63]*.
- *Bubble:* líder en No-Code web, con muchos plugins (algunos integran IA).
- *Zapier/Make:* ya cubiertos, automatización con posibles integraciones de asistentes.

- *Power Apps (Microsoft):* plataforma low-code corporativa; si trabajas en entorno Microsoft, vale conocerla.
- **Cursos y diplomaturas:** Si te interesa formalizar el aprendizaje, la misma Diplomatura en IA para No Programadores de la que forma parte este material es un gran paso. Otras opciones: cursos de **Platzi, Udemy o Coursera** sobre No-Code y AI (busca "No-Code AI" o "Citizen Developer"). Por ejemplo, hay cursos de creación de chatbots con IA sin programar, etc.
- **Lecturas recomendadas:**
- Artículo *"IA generativa sin saber programar"* de ESEID*[64][65]* – para inspirarte con casos de diferentes sectores donde se aplica IA sin código.
- Informe de Gartner sobre Low-Code (citado en InnovaciónDigital360) – da contexto de por qué las empresas invierten en estas tecnologías*[66]*.
- Blog de Globant sobre Augmented Coding – para ver cómo una empresa argentina implementó IA para ayudar a sus programadores*[53]*.

Recuerda: el aprendizaje es continuo. La tecnología no se detiene, y menos en este campo. Pero **ya cuentas con una base sólida** de conceptos y prácticas para seguir explorando. Usa estos recursos para profundizar en lo que más te interese, ya sea perfeccionar cómo escribes prompts, dominar una herramienta en particular, o simplemente mantenerte al día de novedades.

10. Conclusión

En esta guía hemos recorrido un amplio panorama: desde escribir requerimientos en lenguaje natural hasta hacer que una IA genere una aplicación completa; desde casos de uso en Argentina hasta ejemplos internacionales; desde tips prácticos hasta precauciones importantes. Todo con el objetivo de que **pierdas el miedo a la tecnología** y veas a la inteligencia artificial y las plataformas No-Code/Low-Code como **aliados para innovar**, no como cajas negras incomprensibles.

Al ser un profesional sin formación técnica, quizás antes veías la creación de software como algo fuera de tu alcance. Esperamos que ahora esa percepción haya cambiado. Tienes a tu disposición un **arsenal de herramientas** accesibles y poderosas. Con ellas, y aplicando las buenas prácticas que discutimos, podrás prototipar ideas, automatizar tareas y hasta liderar proyectos de inteligencia artificial aplicada, todo *sin escribir código* o escribiendo muy poco.

¿Qué sigue? La autonomía que adquieras con estas habilidades te permitirá colaborar mejor con equipos técnicos (hablarás su idioma un poco más) y también sacar

adelante proyectos propios sin depender completamente de otros. Quizá descubras que te gusta tanto este mundo que termines aprendiendo programación formal más adelante, ¿por qué no? Pero aun si no, ya formas parte de la nueva generación de *"constructores digitales"* que impulsan la transformación en sus organizaciones aprovechando IA y No-Code.

Te animamos a que sigas **experimentando**. Equivócate, prueba, rompe cosas (en entornos de prueba 😅) y vuelve a intentar. La ventaja de estos enfoques es el *costo bajo de la experimentación*: puedes iterar rápido hasta dar con soluciones valiosas.

Como dice un eslogan famoso en tecnología: *"El mejor modo de predecir el futuro es crearlo."* Ahora cuentas con conocimientos y herramientas para crear ese futuro, al menos en el ámbito de tus proyectos. ¡Mucho éxito en tu camino de aprendizaje y creación con IA para generar código sin conocimientos previos! 🚀

---

[1] [2] [63] [64] [65] IA generativa sin saber programar - ESEID AI Business School

https://eseid.com/ia-generativa-sin-saber-programar/

[3] [4] [5] [6] [7] [8] [9] [10] [20] [21] [22] [23] [24] [26] [27] [35] [36] [37] [38] [39] [40] [41] [42] NP.CLASE 6.Documento de clase.docx

file://file_00000000f18c622f834114f1cb91b0a8

[11] [12] [19] [25] [28] [29] [30] [31] [32] [33] [62] Firebase Studio

https://firebase.google.com/docs/studio

[13] [14] [15] [16] [57] [58] [59] No sé nada de programación y he creado esta app en menos de 10 minutos. Solo con ChatGPT y el iPhone

https://www.applesfera.com/tutoriales/no-se-programar-he-creado-app-10-minutos-usando-chatgpt-iphone-paso-a-paso-para-principiantes

[17] [18] [34] [60] [61] Firebase Studio: Build & Prototype Apps Before Your Noodles Are Done | by Angga Agia Wardhana | Google Cloud - Community | Medium

https://medium.com/google-cloud/lets-explore-building-ai-apps-in-minutes-with-firebase-studio-b5044c0afd0c

[43] [44] [45] Desarrollo de Apps para PyMEs: La Revolución No-Code que Está Democratizando la Tecnología | Blog | Grupo La Red | Grupo La Red

https://www.grupolared.com.ar/blog/desarrollo-apps-no-code-nuevas-tecnologias-pymes

[46] [47] [48] [49] [50] [51] [52] [66] Qué es el low code, ventajas y casos de éxito | InnovaciónDigital360

https://www.innovaciondigital360.com/software/que-es-el-low-code-ventajas-y-casos-de-exito/

[53] Using AI to build powerful digital experiences - Globant

https://www.globant.com/stay-relevant/white-paper/using-ai-build-powerful-digital-experiences

[54] [55] [56] Casos de éxito de empresas con Nocode | IEBS Business School

https://www.iebschool.com/hub/casos-de-exito-de-empresas-con-nocode-agile-scrum/