TU.MTCH

Tu GitHub como motor de empleo

Guia práctica para perfiles tech en 2026

Como construir un perfil de GitHub que trabaje por ti: desde como te encuentran los reclutadores hasta como demostrar criterio técnico real en la era de la inteligencia artificial.

CONTENIDO

INTRODUCCION

Por que GitHub importa más que nunca (y más que tu CV)

Hay una pregunta que se repite en comunidades tech, en bootcamps, en eventos de FemCoders Club: ¿Necesito realmente tener un GitHub cuidado para encontrar trabajo? La respuesta corta es si. La respuesta larga es este ebook.

Pero antes de entrar en tacticas, conviene entender por que el contexto cambio, porque no se trata solo de tener proyectos subidos. Se trata de entender que GitHub se convirtio en algo que no era hace cinco años: tu presentación profesional más viva, más verificable y más consultada que cualquier PDF que envies por email.

El mercado que nadie te explico bien

En 2025, más de 36 millones de personas nuevas se unieron a GitHub. Uno por segundo, durante todo el año. La plataforma supero los 180 millones de desarrolladores activos, y ese número sigue creciendo.

Lo que eso significa en la práctica no es solo que hay mucha gente programando. Significa que hay mucho ruido. Muchos perfiles. Muchos repositorios. Y las personas que contratan, tanto recruiters como hiring managers y tech leads, desarrollaron formas muy concretas de filtrar rápido.

> *En ese contexto, un perfil de GitHub bien construido no es un plus. Es el filtro que decide si alguien sigue leyendo tu candidatura o pasa a la siguiente.*

Lo que el CV no puede hacer

Un CV dice lo que tu afirmas de ti misma. GitHub muestra lo que realmente produces.

Esa diferencia importa porque la verificación es inmediata. Un hiring manager puede abrir tu perfil en dos minutos y hacerse una idea bastante precisa de como trabajas: si tus proyectos tienen README claros, si hay tests, si el código esta organizado, si los commits explican decisiones o solo dicen fix y update. Todo eso comunica antes de que abras la boca en una entrevista.

Hay un dato que resume bien esta dinámica: el 81,5% de las contribuciones en GitHub ocurren en repositorios privados. La mayor parte del trabajo real es invisible. Eso convierte lo que si es público en algo con más peso del que parece, porque es lo poco que cualquier persona externa puede ver de ti.

La IA lo cambio todo, incluido el liston

Si en 2023 el estándar era tener proyectos funcionando, en 2026 eso ya no diferencia a nadie.

Las herramientas de IA permiten generar código funcional mucho más rápido. El 84% de los desarrolladores ya las usa o planea usarlas. Y eso tiene una consecuencia directa: el volumen de software aceptablemente funcional subio, pero también subio el volumen de proyectos que parecen bien pero no resisten una pregunta técnica.

Los hiring managers lo saben. Los tech leads lo saben. Y han desarrollado criterios muy específicos para distinguir un portfolio que demuestra criterio real de uno que parece generado sin comprension.

No se trata de no usar IA, esa ya no es la conversación. Se trata de demostrar que entiendes lo que produces, que puedes defenderlo, y que si algo sale mal, sabes por donde empezar a buscar. Eso es exactamente lo que este ebook te enseña a comunicar.

Para quien es esta guia

Esta guia es para cualquier persona en tech que quiera usar GitHub como una herramienta activa de captacion de oportunidades, no como un cajon donde guardar proyectos olvidados.

No importa si eres desarrolladora frontend, backend, fullstack, data engineer, DevOps o estas en transición hacia cualquiera de esos roles. No importa si llevas dos años en el sector o diez. Lo que importa es que quieres que tu trabajo sea visible, comprensible y convincente para las personas que deciden si mereces una entrevista.

> *Si eso te suena, sigue leyendo.*

Como esta organizado este ebook

Lo hemos estructurado siguiendo la lógica real del proceso de selección, no la lógica de tips de GitHub:

Parte 1	Como te encuentran: SEO interno y externo, que indexa GitHub y que no, como buscan los recruiters en la práctica.
Parte 2	Como te evaluan: el perfil, los pins, los repos, y que mira cada persona segun su rol en el proceso.
Parte 3	El factor IA: como demostrar criterio técnico en 2026, que red flags detectan los tech leads, y como documentar tu trabajo con IA sin perder credibilidad.
Parte 4	Open source y networking técnico: como contribuir con intención y construir visibilidad dentro de la comunidad.
Parte 5	Por perfiles: que señales específicas buscan segun si eres frontend, backend, data, DevOps o mobile.

Parte 6	Plan de acción: métricas que importan, automatizaciones que suman, y un checklist para dejar tu GitHub listo en 90 minutos.

Al final encontraras plantillas listas para usar: bio, README de perfil, README de repo, y una tabla de recursos curada.

PARTE 1 · COMO TE ENCUENTRAN

La lógica del reclutador: como buscan talento en GitHub

Antes de optimizar nada, conviene entender quien va a llegar a tu perfil y como. Porque no es una sola persona, ni buscan todas de la misma manera.

En un proceso de selección típico en el sector tech intervienen al menos tres perfiles distintos: el recruiter, el hiring manager y el tech lead. Los tres pueden acabar mirando tu GitHub, pero lo que buscan, el tiempo que invierten y las señales que leen son completamente diferentes. Confundir a quien te diriges es uno de los errores más comunes al construir un portfolio.

Tres personas, tres formas de mirar

El recruiter 30-60 seg	Confirma que existes y que tu perfil tiene sentido con el CV. Mira foto, bio, proyectos anclados y actividad general. Raramente entra en los repos salvo que algo le llame la atención.
El hiring manager 5-10 min	Abre el README de uno o dos proyectos, mira si hay demos o instrucciones claras, y usa lo que ve para preparar preguntas técnicas. Le interesa como piensas, no solo que tecnologías usas.
El tech lead Variable	Puede mirar el historial de commits, como estan escritos los PRs, si hay tests, que decisiones arquitectónicas tomaste y si las justificas en algun lugar. Distingue un portfolio con criterio de uno generado sin comprension.

Entender esta distincion cambia completamente como organizas tu perfil. El recruiter necesita señales rápidas y claras en la superficie. El hiring manager necesita un README que le permita entender el proyecto sin ejecutarlo. El tech lead necesita evidencia de que sabes lo que haces y por que.

Como buscan dentro de GitHub

GitHub tiene su propio buscador, y los reclutadores técnicos saben usarlo. La búsqueda de usuarios permite filtrar por campos muy concretos que vienen directamente de lo que tu escribes en tu perfil.

El qualifier location: busca por la ubicación que has indicado en tu perfil. No es un campo normalizado ni verificado, así que si escribes Bcn en lugar de Barcelona, Spain, puedes quedar fuera de búsquedas que usan la forma completa. El qualifier language: filtra por el lenguaje predominante en tus

repositorios públicos. Y existen filtros adicionales como followers: y repos: que algunos reclutadores usan para acotar por nivel de actividad o influencia.

Una búsqueda realista que podria estar usando alguien buscando perfiles en España en este momento tiene este aspecto:

```
location:barcelona language:typescript

location:spain language:python followers:>50

location:"Barcelona" repos:>10 language:java
```

Nada de esto es magia negra. Es simplemente conocer las reglas del sistema para asegurarte de que apareces cuando alguien busca exactamente lo que tú eres.

Como buscan fuera de GitHub

Más allá del buscador interno, los reclutadores técnicos con experiencia utilizan Google para lo que se conoce como X-Ray search: buscar dentro de GitHub usando operadores booleanos externos. Esto les permite encontrar perfiles que el buscador interno no devuelve fácilmente, o combinar señales de distintas fuentes.

Una búsqueda típica en Google tiene este formato:

```
site:github.com "location * Barcelona" "java"

site:github.io (cv OR resume) "frontend developer" "Barcelona"
```

La segunda variante es especialmente interesante porque rastrea páginas desplegadas en GitHub Pages, que suelen ser portfolios o CVs online donde aparece información de contacto que no esta visible en el perfil principal.

Lo que esto implica para ti es concreto: el texto visible en tu perfil, tus descripciones de repositorio y tu bio no solo importan dentro de GitHub. También son lo que indexan los motores de búsqueda externos. Cada palabra que eliges es una señal que puede o no aparecer en los resultados de alguien que busca exactamente tu perfil.

El dato que cambia la perspectiva

Hay un número que resume bien por que todo esto importa más de lo que parece: en 2025, más de 36 millones de personas nuevas se unieron a GitHub. La plataforma supera los 180 millones de desarrolladores activos. En ese contexto, no aparecer en una búsqueda no es mala suerte. Es consecuencia de no haber configurado las señales correctas.

> *La buena noticia es que la mayoria de esas señales estan completamente bajo tu control. Y eso es exactamente lo que cubre el capitulo siguiente.*

PARTE 1 · COMO TE ENCUENTRAN

SEO técnico: ser encontrable sin engañar al algoritmo

Existe una diferencia importante entre parecer relevante y ser relevante. El SEO interno de GitHub no premia el relleno de palabras clave ni los perfiles artificialmente inflados. Premia la coherencia entre lo que declaras y lo que muestras. Entender como funciona ese sistema es lo que te permite aparecer donde tienes que aparecer, sin trucos que se rompen solos.

Lo que indexa GitHub y lo que no

Antes de optimizar nada, conviene saber exactamente que campos entran en las búsquedas y cuales no. Aqui es donde la mayoria de guias se equivocan o simplifican demasiado.

En la búsqueda de repositorios, GitHub indexa por defecto tres campos: el nombre del repositorio, la descripción y los topics. Solo esos tres. El README, que es donde mucha gente pone toda su energia, no entra en las búsquedas por defecto. Para que el contenido de un README aparezca en resultados, quien busca tiene que añadir explícitamente el qualifier in:readme a su query. Esto no es un detalle menor: significa que tu estrategia de keywords no puede depender solo de lo que escribes en el README.

Campo	Indexa por defecto	Como forzarlo
Nombre del repo	**Si**	in:name
Descripción	**Si**	in:description
Topics	**Si**	in:topics o topic:
README	No	in:readme (explícito)

> *Nombre del repo, descripción y topics son tus tres palancas de descubribilidad. El README es tu palanca de conversion: lo que convence a quien ya llego.*

El nombre del repositorio como señal de búsqueda

El nombre de un repositorio es el primer campo que indexa GitHub y también el que más peso tiene en los resultados. Sin embargo, es el campo que más se descuida.

Hay una diferencia enorme entre proyecto-final, app, mi-portfolio e invoice-api-spring-boot-postgres. El primero dice nada. El segundo dice menos. El tercero dice quien lo hizo. El cuarto dice que hace, con

que tecnología y que problema resuelve, todo en cuatro palabras.

Esto no significa que todos tus repositorios tengan que tener nombres tecnicamente descriptivos. Si estas construyendo un proyecto con nombre propio de producto, usalo. Pero para proyectos de portfolio orientados a empleo, el nombre es una oportunidad de aparecer en búsquedas que no deberias desperdiciar.

La descripción como anuncio de una línea

La descripción de un repositorio tiene que hacer el trabajo de un titular: en una frase, explicar que problema resuelve y con que stack. No es un lugar para ser creativa con el lenguaje. Es un lugar para ser precisa.

Una descripción que funciona sigue este patrón: problema que resuelve + tecnología principal + resultado o contexto. Por ejemplo:

```
API REST de gestión de facturas con Spring Boot y PostgreSQL. Auth JWT y docs OpenAPI.
```

```
Plataforma de matching de empleo tech. Next.js, TypeScript, PostgreSQL. Deploy en Railway.
```

```
Dashboard de análisis de ventas en tiempo real. Python, FastAPI, React y Redis.
```

Cada una de esas descripciones aparece en búsquedas de repositorios sin necesidad de que nadie añada qualifiers adicionales. Y cada una comunica, en menos de dos líneas, exactamente lo que alguien que busca ese perfil quiere ver.

Topics: la etiqueta que trabaja por ti

Los topics son el campo más subestimado del SEO interno de GitHub. Fueron diseñados explícitamente para que otros encuentren y contribuyan a proyectos, y funcionan como etiquetas indexables que agrupan repositorios por tecnología, dominio y propósito.

Las reglas técnicas son simples: solo minusculas, números y guiones; máximo 50 caracteres por topic; máximo 20 topics por repositorio. Dentro de esas reglas, la estrategia es elegir topics que coincidan con lo que buscan las personas que quieres que te encuentren.

Un repositorio de portfolio bien etiquetado puede tener entre 8 y 12 topics que cubran tres capas: la tecnología del stack, el dominio o sector de aplicación, y el tipo de proyecto. Por ejemplo, un proyecto de backend con Java podria tener: java, spring-boot, postgresql, rest-api, jwt, docker, testing, ci-cd, backend, portfolio. Diez topics que cubren las búsquedas más habituales para ese perfil.

Lo que conviene evitar son topics genericos que no te diferencian de nada, como code, programming o project. Y también los topics que describen el estado del proyecto en lugar de su contenido, como work-in-progress o learning. Esos ocupan espacio sin aportar señal útil.

Ubicación: el campo más ignorado y más importante

La ubicación es el campo que decide si apareces o no en búsquedas geograficas, que son una de las formas más habituales de sourcing dentro de GitHub. Y es un campo de texto libre: GitHub no normaliza ni sugiere formatos, simplemente muestra lo que escribiste.

Eso significa que Bcn, Barcelona, Barcelona Spain, Barcelona ES y Spain son valores diferentes que no devuelven los mismos resultados. Si alguien busca location:barcelona y tu tienes escrito Bcn, no apareces.

La recomendacion es simple: usa la forma más completa y más buscable. Barcelona, Spain cubre tanto búsquedas en ingles como en español, y es el formato que más se repite en fuentes de sourcing documentadas. Si trabajas en remoto y estas abierta a oportunidades fuera de tu ciudad, puedes combinar ubicación y disponibilidad en la bio, pero en el campo de ubicación mantén algo concreto y buscable.

La diferencia entre descubribilidad y conversion

Todo lo que hemos visto en este capitulo sirve para una sola cosa: que alguien llegue a tu perfil. Pero llegar no es suficiente. Lo que ocurre después depende de señales distintas que cubriremos en la Parte 2.

La forma más clara de separar los dos objetivos es esta: el nombre, la descripción, los topics y la ubicación son para ser encontrada. El README, los pins, los commits y los tests son para convencer a quien ya te encontro.

> *Intentar hacer las dos cosas con los mismos elementos es el error más común. Cada campo tiene su función. Usarlos bien es lo que separa un perfil que aparece de un perfil que convierte.*

PARTE 2 · COMO TE EVALUAN

El embudo de conversion de un perfil

Cuando alguien llega a tu perfil de GitHub, no lo lee. Lo escanea.

Eso no es un insulto ni una queja sobre la atención humana. Es simplemente como funciona la evaluación en contexto de selección: hay muchos perfiles, poco tiempo, y el objetivo no es disfrutar del recorrido sino tomar una decisión rápida sobre si vale la pena invertir más tiempo. Entender ese proceso te permite diseñar tu perfil para que trabaje a tu favor en cada etapa.

El embudo en cuatro fases

1. Descubrimiento	Tu perfil aparece en una búsqueda, en una recomendacion, en un hilo de GitHub, o porque alguien vio tu nombre en un proyecto de open source. Lo que puedes controlar es la primera impresion que da tu perfil cuando alguien hace clic.
2. Evaluación rápida	Dura entre 30 y 120 segundos. Los elementos que entran en juego son exactamente cuatro: la foto, la bio, el README de perfil si lo tienes, y los repositorios anclados. Todo lo demas existe pero no se procesa en esta fase.
3. Prueba técnica	Si la evaluación rápida genero interes, viene la profundizacion. Alguien entra en uno o dos repos, lee el README del proyecto, mira si hay demo, revisa los commits o los tests. Esta fase puede durar cinco minutos o veinte.
4. Conversion	La persona hace algo: te sigue, te manda un mensaje, te contacta por LinkedIn o email, o añade tu nombre a una lista de candidatos. Ocurre solo si las tres fases anteriores funcionaron.

> *El error más común es optimizar solo para una de estas fases. El embudo completo tiene que funcionar: si una fase falla, las siguientes no ocurren.*

Lo que ocurre en los primeros 30 segundos

Cuando alguien abre tu perfil, lo primero que ve es la columna izquierda: foto, nombre, bio, ubicación, enlaces. Luego el README de perfil si existe. Luego los repositorios anclados.

La foto importa más de lo que parece, no por razones esteticas sino por razones de confianza. Un perfil con foto de persona real transmite que hay alguien detrás. Un perfil sin foto o con avatar generico

genera una pequeña duda que, en contexto de evaluación rápida, puede ser suficiente para que alguien pase al siguiente resultado.

La bio es tu única oportunidad de decir quien eres en 160 caracteres. No es un lugar para ser vaga. Developer no dice nada. Passionate about technology dice menos. Lo que funciona es ser específica: que haces, con que tecnología principal, y que buscas o que ofreces.

Los repositorios anclados son el elemento que más peso tiene en la evaluación rápida porque son la única señal directa de que tipo de trabajo produces. GitHub permite anclar hasta seis, pero la recomendacion es quedarse en tres o cinco. Seis pins sin criterio generan ruido. Tres pins bien elegidos generan una narrativa clara sobre tu perfil.

Por que el README de perfil cambia todo

El README de perfil es el elemento que más diferencia hay entre un perfil básico y uno trabajado. Es el archivo que aparece directamente en tu página de perfil cuando creas un repositorio público con el mismo nombre que tu usuario.

No todos los perfiles lo tienen. Y los que lo tienen no siempre lo usan bien. Un README de perfil que funciona no es una lista de tecnologías con iconos de colores. Es una presentación estructurada que responde en menos de un minuto las preguntas que tiene quien te evalua: quien eres, que produces, que buscas y como contactarte.

La estructura que mejor funciona combina una presentación breve, dos o tres resultados concretos que demuestran capacidad, los proyectos clave con enlace directo, las habilidades agrupadas por area, y la información de contacto. Sin florituras. Sin widgets de estadisticas que no dicen nada. Sin animaciones que ralentizan la carga.

La señal que más se ignora: el grafo de contribuciones

El grafo de contribuciones, los famosos cuadritos verdes, es el elemento más malinterpretado de GitHub tanto por quienes construyen perfiles como por quienes los evaluan.

La realidad técnica es esta: el 81,5% de las contribuciones en GitHub ocurren en repositorios privados. Un perfil con pocos cuadritos verdes puede pertenecer a alguien que trabaja intensamente en proyectos privados de empresa. Un perfil con muchos cuadritos verdes puede ser el resultado de commits automáticos o contribuciones triviales.

Los reclutadores con poca experiencia técnica tienden a darle más peso del que merece. Los hiring managers y tech leads con experiencia lo miran como una señal secundaria, no primaria. Lo que si leen es la calidad de lo que hay dentro de los repositorios públicos.

Dicho esto, mantener cierta actividad pública consistente ayuda a transmitir que GitHub es parte de tu práctica habitual, no un perfil que creaste para una búsqueda de empleo. No se trata de forzar commits diarios. Se trata de que cuando alguien mire tu perfil, vea que hay vida ahi.

Diseñar para el escaneo, no para la lectura

La conclusión práctica de entender el embudo es que tu perfil no se diseña para ser leido de arriba a abajo. Se diseña para ser escaneado en distintas profundidades por distintas personas con distintos objetivos.

Quien	Que escanea
Recruiter	Foto, bio, pins. Superficie del perfil.
Hiring manager	README de proyecto, demo, instrucciones de uso.
Tech lead	Commits, tests, decisiones documentadas, coherencia interna.

> *La coherencia entre lo que dices y lo que muestras es la señal más difícil de falsificar y la más fácil de verificar. Y en 2026, con el volumen de perfiles que existe, es también la que más diferencia.*

PARTE 2 · COMO TE EVALUAN

El perfil como landing page: bio, README de perfil y pins

Si tuvieras que elegir tres elementos de tu perfil para trabajar primero, serian exactamente estos. No porque los demas no importen, sino porque son los que cualquier persona ve antes de decidir si sigue mirando. Son tu primera conversación con quien te evalua, y ocurre sin que estes presente.

La bio: 160 caracteres que hacen el trabajo de una presentación

GitHub limita la bio a 160 caracteres. Eso no es una restriccion, es una disciplina. Te obliga a elegir que es lo más importante que alguien tiene que saber de ti en el momento en que llega a tu perfil.

El error más común es usar ese espacio para describir una actitud en lugar de una capacidad. Frases como apasionada por la tecnología, siempre aprendiendo o amante del código limpio ocupan caracteres sin decir nada verificable. No son mentira, pero tampoco son señal.

Lo que funciona sigue una estructura simple: que haces + con que + que buscas o que ofreces. No tiene que sonar como un anuncio. Tiene que sonar como una persona real que sabe exactamente lo que hace.

Perfil	Ejemplo de bio
Fullstack	`Fullstack developer · React, Spring Boot, PostgreSQL · Buscando rol en producto · Proyectos con demo y tests abajo`
Backend	`Backend Java · APIs REST, arquitectura limpia, CI/CD · Construyo sistemas que escalan · Open to work`
Data	`Data Engineer · Python, dbt, Airflow · Pipelines reproducibles y documentados · Buscando equipo con cultura de datos real`
Junior	`Dev Jr · React y Node · Tres proyectos completos con demo, tests y CI · Busco primera oportunidad`

Nota lo que tienen en común: rol claro, stack principal, prueba o señal de calidad, y disponibilidad o intención. En menos de 160 caracteres. Sin adjetivos que no se pueden verificar.

El README de perfil: tu presentación extendida

El README de perfil es el espacio donde puedes desarrollar lo que la bio solo puede insinuar. Aparece directamente en tu página principal de GitHub cuando creas un repositorio público con exactamente el mismo nombre que tu usuario y añades un archivo README.md en la raíz.

No todos los perfiles lo tienen. Tenerlo ya es una señal de que conoces la plataforma y te has tomado el tiempo de trabajar tu presencia. Pero más importante que tenerlo es que funcione.

Un README de perfil que funciona responde, en este orden, las preguntas que tiene quien llega:

Quien eres y que haces	Dos o tres líneas que situen tu rol, tu stack principal y el tipo de problemas que resuelves. Tan clara que alguien que no sabe nada de ti entienda tu perfil en 15 segundos.
Que has producido	Dos o tres resultados concretos, no responsabilidades. No desarrolle una API sino construi una API de gestión de pedidos con Spring Boot que procesa más de 10.000 transacciones diarias. Si tus proyectos son personales, habla de decisiones técnicas: arquitectura, autenticación, CI.
Tus proyectos clave	Con enlace directo al repositorio y una línea de descripción. Los mismos tres o cinco que tienes anclados. La coherencia entre README y pins refuerza la narrativa de tu perfil.
Tus habilidades agrupadas	No una lista interminable. Agrupa por area: Frontend, Backend, Testing, DevOps, Datos. Pon las que realmente usas con soltura. Una lista de 40 tecnologías que incluye cosas tocadas en un tutorial no convence a nadie.
Como contactarte	Email, LinkedIn, portfolio si tienes. Claro y visible. Parece obvio pero hay perfiles completos y bien trabajados donde no hay ninguna forma de contactar a la persona.

> *Lo que no funciona: widgets de estadisticas de GitHub, animaciones de texto y GIFs decorativos. Generan la sensacion de que priorizas la apariencia sobre el contenido, y no aportan señal útil sobre tu calidad como profesional.*

Los pins: tu selección editorial

Los repositorios anclados son la decisión más importante que tomas en tu perfil. Son los proyectos que eliges mostrar cuando podrias mostrar cualquier otro. Esa eleccion dice mucho sobre como entiendes tu propio valor.

GitHub permite anclar hasta seis repositorios, pero la recomendacion es trabajar con tres a cinco. Con seis pins sin criterio editorial claro, la persona que te evalua no sabe por donde empezar. Con tres o cuatro bien elegidos, hay una narrativa inmediata.

El criterio de selección tiene que responder a una pregunta: que proyecto demuestra mejor lo que puedo hacer para el rol que busco. No el más antiguo porque te costo mucho. No el que tiene más estrellas. El que mejor muestra tu capacidad actual en el stack y el tipo de trabajo que quieres hacer.

Una combinación que funciona bien es mezclar proyectos propios con una o dos contribuciones a proyectos de open source. Los proyectos propios demuestran que puedes llevar algo de principio a fin. Las contribuciones demuestran que puedes integrarte en código de otros, seguir convenciones y comunicarte dentro de un equipo.

Dentro de cada proyecto anclado, hay tres elementos visibles antes de entrar al repositorio: el nombre, la descripción y el lenguaje principal. Esos tres elementos tienen que ser suficientes para que alguien entienda de que va el proyecto sin hacer clic. Si el nombre es críptico, la descripción esta vacia y el lenguaje es el único dato visible, has perdido la oportunidad.

La coherencia como estrategia

Bio, README y pins no son tres elementos independientes. Son tres capas de una misma historia.

Si tu bio dice que eres backend con Java y tus pins son todos proyectos de JavaScript sin tests, hay una incoherencia que alguien va a notar. Si tu README promete proyectos con CI y ninguno de tus repos anclados tiene un badge de Actions, hay una promesa incumplida. Si describes resultados en el README pero los proyectos que enlazas no tienen README propio ni demo, hay un vacio que genera dudas.

> *La coherencia entre estas tres capas es lo que convierte un perfil de GitHub en una presentación profesional real. Y es también lo que diferencia a alguien que construyo su perfil con intención de alguien que simplemente tiene cosas subidas.*

PARTE 2 · COMO TE EVALUAN

Los repos que convierten: anatomia de un proyecto de portfolio

Un repositorio de portfolio tiene un objetivo muy concreto: convencer a alguien que no te conoce de que sabes lo que haces, en el menor tiempo posible, sin que tenga que ejecutar el código para verificarlo.

Eso cambia completamente como se construye un proyecto orientado a empleo versus un proyecto personal o de aprendizaje. No es cuestión de hacerlo más bonito. Es cuestión de hacerlo más legible para alguien que tiene cinco minutos y cero contexto previo.

El README del repositorio: diseñado para la lectura rápida

El README es lo primero que ve cualquier persona que entra a un repositorio. Y quien evalua revisara tus proyectos durante un par de minutos. Eso no es mucho tiempo para convencer a nadie.

Un README de repositorio orientado a empleo tiene que responder cinco preguntas en este orden, sin que haya que buscarlas:

Que es y que problema resuelve	Una o dos frases. No la historia del proyecto ni las motivaciones personales. El problema concreto que resuelve y a quien le resulta útil.
Como se ejecuta	Comandos copiables, paso a paso, que funcionen en un entorno limpio. El objetivo es que cualquier persona pueda clonar el repositorio y tenerlo funcionando en menos de cinco minutos.
Que tecnologías usa y por que	No una lista. Una explicacion breve de las decisiones principales: por que elegiste ese stack, que problema resuelve cada pieza, que alternativas consideraste. Esto diferencia a quien sabe lo que usa de quien siguio un tutorial.
Como esta probado	Que tipo de tests hay, como ejecutarlos, que cobertura tienen. Si hay CI, un badge visible en la cabecera del README que muestre el estado actual del pipeline.
Una demo o capturas	Algo visual que permita entender el resultado sin ejecutar nada. Puede ser un GIF, capturas de pantalla con texto alternativo, o un enlace a una versión desplegada.

La prueba de los cinco minutos

Hay una forma muy directa de saber si el README de tu repositorio funciona: dáselo a alguien que no conoce el proyecto y mide cuánto tarda en tener el entorno levantado y entender que hace la aplicación. Si tarda más de cinco minutos, el README necesita trabajo.

Esta prueba revela problemas que son invisibles para quien construyo el proyecto: dependencias que se asumen instaladas, variables de entorno que no estan documentadas, pasos que se dan por conocidos. Todo eso es friccion que, en contexto de evaluación, se traduce en perdida de interes.

> *La reproducibilidad no es un detalle técnico. Es una señal directa de que piensas en quien va a usar tu código, no solo en que funcione en tu maquina.*

Tests y CI: la señal de madurez más verificable

En 2026, un repositorio de portfolio sin tests es una declaracion involuntaria. No dice que el código no funciona. Dice que no tienes hábito de verificar que funciona de forma sistemática. Y eso, para cualquier persona que haya trabajado en un equipo técnico real, es una señal de riesgo.

Los tests no tienen que ser exhaustivos para comunicar lo que tienen que comunicar. Un conjunto de tests unitarios sobre la lógica de negocio principal, algunos tests de integración sobre los endpoints críticos, y un pipeline de CI que los ejecuta en cada push es suficiente para transmitir que trabajas con criterio de calidad.

Lo que el badge de CI añade es una capa de verificación inmediata. Quien llega al repositorio ve en dos segundos si el código pasa los tests. No tiene que ejecutar nada. No tiene que confiar en tu palabra. El sistema lo verifica de forma objetiva y visible.

Dependabot activado en repositorios con dependencias es otra señal que se nota: dice que el repositorio esta mantenido, que alguien se preocupa por la seguridad y la actualización de las dependencias. No requiere trabajo activo: se configura una vez y funciona solo.

La arquitectura documentada: mostrar como piensas

Hay una diferencia fundamental entre un proyecto que funciona y un proyecto que se entiende. El código puede funcionar perfectamente y seguir siendo opaco para quien no lo escribio.

Documentar la arquitectura no significa escribir un manual de 50 páginas. Significa incluir en el README, o en un archivo separado dentro de /docs, una descripción breve de como esta organizado el proyecto, que responsabilidad tiene cada capa o módulo, y por que se tomaron las decisiones principales.

Un diagrama simple, incluso en texto plano o con Mermaid dentro del README, que muestre como fluye la información desde la peticion hasta la respuesta dice más sobre como piensas que cien líneas de código bien formateadas.

Las decisiones técnicas documentadas son especialmente valiosas porque son exactamente lo que un tech lead va a preguntar en una entrevista. Si ya estan en el repositorio, demuestras que piensas en la

justificacion de tus elecciones, no solo en que el código funcione.

La demo desplegada: evidencia que no se puede falsificar

Una aplicación funcionando en un entorno real fuerza decisiones que un proyecto local nunca enfrenta: como gestionar las variables de entorno, que pasa con los errores en producción, como se comporta el sistema bajo condiciones reales.

Un enlace a una demo desplegada en el README es una de las señales más potentes que puede tener un repositorio de portfolio, precisamente porque es verificable de forma inmediata. No es una promesa. Es una evidencia.

Plataforma	Uso recomendado
Railway / Render	Backends con base de datos. Plan gratuito suficiente para portfolio.
Vercel / Netlify	Frontends y aplicaciones Next.js o React. Deploy automático desde GitHub.
Fly.io	Aplicaciones con Docker. Buena opción para proyectos con dependencias complejas.
Hugging Face Spaces	Proyectos de machine learning y demos de modelos. Gratuito y visible en la comunidad.

Si por alguna razón el proyecto no puede estar desplegado de forma permanente, un video corto de demostracion, dos o tres minutos mostrando las funcionalidades principales, cumple la misma función. Es visual, es inmediato, y no requiere que nadie clone ni ejecute nada.

Diseño para quien no tiene contexto

La diferencia entre un repositorio que no convierte y uno que si lo hace no es de calidad de código. Es de presentación y accesibilidad.

Un repositorio con README de una línea, sin tests, sin instrucciones de instalación y sin demo puede contener código excelente. Pero nadie lo va a saber porque nadie va a invertir el tiempo necesario para descubrirlo.

Un repositorio con README claro, tests visibles, CI funcionando, arquitectura explicada y demo accesible puede contener código imperfecto. Pero transmite que quien lo construyo sabe como trabaja un equipo profesional, y eso es lo que la mayoria de procesos de selección buscan verificar.

> *El código mejora con el tiempo. Los hábitos de trabajo son más difíciles de cambiar. Un portfolio bien presentado dice exactamente eso: que ya tienes los hábitos correctos.*

PARTE 3 · EL FACTOR IA

El nuevo estándar: criterio, no cantidad

Durante años, el consejo para mejorar un perfil de GitHub fue siempre el mismo: sube más proyectos, haz más commits, contribuye más. La cantidad era la señal. Ese consejo ya no funciona. Y entender por que cambia completamente lo que tienes que priorizar.

Por que el volumen dejo de ser diferenciador

En 2025, GitHub registro casi mil millones de commits, un 25% más que el año anterior. Mas de 36 millones de personas nuevas se unieron a la plataforma. Las herramientas de IA permiten generar código funcional en minutos, README completos en segundos, y proyectos con estructura aparentemente profesional sin haber tomado una sola decisión técnica consciente.

El resultado es que el umbral de tener cosas subidas bajo drasticamente. Ya no diferencia. Lo que diferencia ahora es la evidencia de que detrás de lo que esta subido hay una persona que entiende lo que produce, que tomo decisiones con criterio, y que puede defenderlas.

> *Eso tiene un nombre en la comunidad técnica: criterio de ingenieria. Y es exactamente lo contrario del vibe coding.*

Que es el vibe coding y por que importa entenderlo

El termino vibe coding describe una forma de construir software con herramientas de IA donde no necesariamente entiendes el código que se esta produciendo. Describes lo que quieres, la IA lo genera, tu lo copias, funciona más o menos, y sigues adelante.

No es una práctica nueva en esencia: programar sin entender del todo lo que se copia de Stack Overflow es una versión anterior del mismo problema. Pero la escala y la velocidad con que la IA puede generar código plausible lo convierte en algo cualitativamente diferente. En cuestión de horas puedes tener un repositorio con estructura de proyecto profesional, tests generados, README detallado y commits con mensajes bien redactados, sin haber tomado una sola decisión técnica propia.

El problema no es etico en primera instancia. El problema es práctico: ese tipo de portfolio no resiste una conversación técnica. Y las personas que contratan lo saben.

Como se detecta en la práctica

Los hiring managers y tech leads con experiencia no necesitan herramientas especiales para detectar un portfolio construido sin criterio. Hay patrones que se repiten y que son reconocibles incluso en una revisión rápida.

Exceso inutil	Código que esta ahi pero no hace nada relevante, funciones duplicadas, imports sin usar, estructuras de datos sobredimensionadas para el problema que resuelven. El código generado por IA tiende a ser verboso porque optimiza para parecer completo, no para ser preciso.
Inconsistencia de estilo	Dentro del mismo archivo, convenciones de nombrado que cambian, patrones de error handling que se aplican en un sitio y no en otro, niveles de abstracción que saltan sin razón aparente.
Decisiones sin justificacion	Un índice sobre un campo booleano, un try/catch que captura Exception sin hacer nada con el error, una abstracción de tres capas para un problema que no la necesita. Decisiones que no tienen sentido para alguien que conoce el dominio.
Incapacidad de explicar	No visible en el repositorio, pero si en la entrevista. Pedir que expliquen una decisión específica, que justifiquen por que esta así y no de otra manera, que hablen sobre que fallo durante el desarrollo. El código generado no tiene historia. El código construido con criterio si.

El nuevo liston: que se espera en 2026

Lo que ha cambiado no es que ahora se penalice usar IA. El 84% de los desarrolladores la usa. Penalizarla seria absurdo e imposible de aplicar.

Lo que ha cambiado es que usar IA ya no es suficiente para impresionar a nadie, y que usarla mal, sin criterio, sin verificación, sin responsabilidad sobre el output, es una señal negativa activa.

El nuevo estándar se puede resumir en tres expectativas concretas:

Responsabilidad sobre el output	Que lo que esta en el repositorio, hayas generado con IA o sin ella, lo puedas entender, explicar y defender. Si no puedes sostener una conversación técnica sobre tu propio código, el origen de ese código se vuelve irrelevante.
Verificación sistemática	Que haya tests que demuestren que el código hace lo que se supone que hace, que el CI ejecute esos tests en cada cambio, y que los resultados sean visibles. Lo que cambio es que ahora es el filtro mínimo para que un repositorio sea tomado en serio.
Coherencia interna	Que las decisiones del proyecto tengan sentido entre si, que el código coincida con lo que dice el README, que los tests cubran la lógica que se describe como crítica. La coherencia es difícil de fabricar sin comprension real, y por eso se convirtio en la señal más fiable.

Lo que esto significa para tu portfolio

No significa que tengas que demostrar que no usaste IA. Esa señalizacion defensiva, añadir una nota en el README diciendo este proyecto fue escrito sin IA, genera más preguntas de las que responde y suena a inseguridad más que a confianza.

Lo que significa es que el portfolio que funciona en 2026 es el que evidencia comprension en cada capa: en el README que explica decisiones, en los tests que verifican lógica de negocio, en los commits que cuentan la historia del desarrollo, en la demo que prueba que el sistema funciona en condiciones reales.

> *No se trata de cuánto tienes subido. Se trata de que lo que tienes subido demuestre que sabes lo que haces.*

PARTE 3 · EL FACTOR IA

Red flags que detectan hiring managers y tech leads

Hay una diferencia importante entre un portfolio que no impresiona y uno que genera desconfianza activa. El primero simplemente no avanza. El segundo cierra puertas que podrian haberse abierto.

Conocer las señales que generan desconfianza no es para obsesionarse con evitar errores. Es para entender el criterio con el que te evaluan y poder tomar decisiones informadas sobre como presentas tu trabajo.

Lo que mira un tech lead en los primeros cinco minutos

Cuando un tech lead abre un repositorio de portfolio, no empieza por el código. Empieza por la estructura. Y lo que busca en esa primera revisión no es perfeccion: busca coherencia y señales de que hay criterio detrás.

El README	No para leerlo entero, sino para comprobar si existe, si tiene sentido, y si lo que dice corresponde con lo que hay en el repositorio. Un README detallado que describe funcionalidades no implementadas, o que usa un stack diferente al que aparece en los archivos, es una señal inmediata de que algo no cuadra.
El historial de commits	No el número, la calidad. Commits con mensajes como fix, update, changes o wip seguidos de un commit que sube todo el proyecto de golpe cuentan una historia: que el desarrollo no ocurrio de forma iterativa y que los commits son una formalidad, no una herramienta de comunicación.
La presencia de tests	No hace falta abrir los archivos de test para saber si los hay. La estructura del proyecto lo dice. Si no hay directorio de tests, si no hay dependencias de testing, si no hay workflow de CI, la ausencia es visible antes de leer una sola línea de código.

Red flags específicas en el código

Más allá de la estructura, cuando un tech lead entra en el código hay patrones concretos que generan alerta. Estos no son errores de sintaxis: son decisiones cuestionables que sugieren falta de comprension del dominio o del lenguaje.

Manejo indiscriminado de excepciones	Un bloque try/catch que captura Exception sin hacer nada con el error, sin log, sin mensaje, sin relanzar. En producción, ese patrón convierte errores silenciosos en problemas imposibles de diagnosticar.

Decisiones sin sentido técnico	Un índice sobre un campo booleano, abstracciones de tres capas para un problema simple, estructuras de datos sobredimensionadas. Decisiones que no tienen sentido para alguien que conoce el dominio.
Código duplicado sin razón	La misma lógica repetida en tres sitios distintos sin abstracción. Sugiere que el proyecto crecio sin diseño y que el código fue generado sin estructura pensada.
Comentarios que no añaden contexto	Un comentario que dice 'This function handles user authentication' encima de una función llamada authenticateUser no añade valor. Los comentarios útiles explican el por que, no el que.

Red flags en la documentación

Un README perfecto en un repositorio con código caotico es una incoherencia que cualquier revisor nota. La documentación tiene que ser proporcional al código: ni inexistente, ni tan elaborada que parezca generada para impresionar en lugar de para informar.

Las descripciones de funcionalidades que no estan implementadas son especialmente dañinas. Si el README describe un sistema de autenticación con OAuth y el código solo tiene un formulario de login sin validación, alguien que lo revise va a preguntarse si el resto del proyecto también esta inflado.

Los commits de documentación masivos, un único commit que añade cien líneas de README seguido de commits de código menores, sugieren que la documentación fue escrita después del código para presentar el proyecto, no durante el desarrollo para ayudar a entenderlo.

Red flags en la comunicación

GitHub no es solo código. Las issues, los PRs y los comentarios son parte de lo que un tech lead puede leer, y la calidad de esa comunicación dice mucho sobre como trabaja una persona en un equipo.

Un PR con descripción vacia o con una descripción generada que no explica que cambia ni por que transmite que quien lo abrio trata los PRs como un tramite. En un equipo real, un PR es una conversación: explica el problema, la solución elegida, las alternativas consideradas, como se puede verificar que funciona.

Las issues abiertas y nunca resueltas, o las issues cerradas sin explicacion de como se resolvieron, dan una imagen de un proyecto donde el seguimiento no existe.

Lo que no es una red flag

Vale la pena decirlo explícitamente porque hay ansiedad innecesaria en torno a algunos aspectos del portfolio.

Un proyecto inacabado	No es una red flag si esta presentado honestamente. Un README que dice en desarrollo y describe lo que hay implementado y lo que esta pendiente es perfectamente válido. Lo que genera desconfianza es un proyecto que pretende estar terminado y no lo esta.
Un proyecto simple	Un CRUD bien construido, con tests, con CI, con README claro y con decisiones documentadas dice más sobre como trabajas que una aplicación compleja sin ninguna de esas cosas.
Tener pocos proyectos	Tres repositorios de portfolio solidos son más convincentes que diez repositorios a medias. La cantidad nunca fue el criterio real.

> *Lo que genera desconfianza no es la imperfeccion. Es la incoherencia entre lo que se promete y lo que se entrega, y la ausencia de señales de que hay comprension real detrás del código.*

PARTE 3 · EL FACTOR IA

Como demostrar juicio real trabajando con IA

Este capitulo es el más práctico de la Parte 3: que hacer exactamente para que tu portfolio demuestre criterio técnico real en un contexto donde el 84% de los desarrolladores usa IA.

La premisa es simple: no se trata de esconder que usas IA ni de proclamar que no la usas. Se trata de que lo que produces, con IA o sin ella, demuestre que entiendes lo que haces y que puedes responder por ello.

Transparencia útil, no performativa

Hay una forma de hablar sobre el uso de IA en un portfolio que ayuda, y otra que no ayuda.

La que no ayuda es la señalizacion defensiva: añadir una nota que dice este proyecto fue construido sin IA o, en el extremo opuesto, un badge que dice powered by AI sin más contexto. La primera suena a inseguridad. La segunda no dice nada sobre si entiendes lo que hay dentro.

La que ayuda es la transparencia con contexto: explicar brevemente que partes del proyecto fueron aceleradas con herramientas de IA, que validación hiciste, y que decisiones tomaste tu. No en forma de confesion, sino en forma de información útil para quien revisa.

Un formato que funciona en el README o en una sección de decisiones técnicas:

```
"La estructura inicial del proyecto fue generada con ayuda de GitHub Copilot. Los tests fueron escritos manualmente para verificar la lógica de negocio. Las decisiones de arquitectura, separación en capas, eleccion de PostgreSQL sobre MongoDB, estrategia de autenticación, fueron tomadas y documentadas antes de generar código."
```

Ese parrafo no pide disculpas ni justifica nada. Informa. Y dice algo muy concreto sobre como trabajas: que usas herramientas con criterio, que verificas lo que produces, y que las decisiones importantes las tomas tu.

Los commits como diario de decisiones

El historial de commits es el elemento más honesto de un repositorio porque es muy difícil de fabricar de forma convincente después del hecho.

Un historial que demuestra criterio no tiene que ser perfecto. Tiene que ser legible. Los commits tienen que contar la historia del desarrollo: que se construyo primero, donde hubo problemas, que se refactorizo y por que. Esa narrativa es imposible de generar con IA porque requiere que el desarrollo haya ocurrido de forma real e iterativa.

Sin criterio	Con criterio
`fix bug update changes wip final`	`Add JWT middleware — previous approach exposed endpoints without auth Refactor user service — split validation from persistence logic Fix race condition in order processing — add optimistic locking`

Una práctica concreta que marca diferencia: cuando tomas una decisión técnica significativa, escribe el razonamiento en el mensaje de commit o en una nota de la PR. Eso es exactamente lo que un tech lead busca cuando quiere entender si hay criterio detrás del código.

Tests como prueba de comprension

Hay una razón por la que los tests son la señal de criterio más difícil de falsificar: para escribir un test útil, tienes que entender que hace el código y que condiciones pueden hacer que falle.

La diferencia visible en un repositorio entre tests superficiales y tests con criterio no requiere leer el código: esta en los nombres de los tests. testLogin() versus testLoginFailsWhenPasswordExceedsMaxLength() son dos señales completamente distintas sobre la profundidad de comprension.

> *Para proyectos de portfolio, una cobertura modesta pero bien dirigida es más convincente que una cobertura alta con tests triviales. Tres tests que cubren los casos de negocio principales con nombres descriptivos dicen más que veinte tests que comprueban que las funciones existen.*

Las PRs como demostracion de como trabajas en equipo

Incluso en proyectos individuales, trabajar con pull requests en lugar de hacer commits directamente a la rama principal tiene un valor de señal que muchos subestiman.

Una PR bien descrita muestra como comunicarias un cambio a un equipo: que problema resuelve, que decisión se tomo, como se puede verificar que funciona, que casos limite se consideraron. Es exactamente el formato que se espera en cualquier equipo técnico profesional.

— Una línea de resumen del cambio
— El problema que resuelve
— La solución implementada y por que se eligio
— Como probarlo o verificarlo
— Limitaciones conocidas o mejoras pendientes

Esa última parte, reconocer limitaciones conscientemente, es una señal fuerte de criterio técnico. Es mucho más convincente que una PR que pretende ser perfecta.

Documentar las decisiones que no se ven en el código

Hay decisiones técnicas que son invisibles en el código final porque son las que descartaste. Elegiste PostgreSQL en lugar de MongoDB, por que. Decidiste no usar un ORM completo, que ganaste y que perdiste. Optaste por una arquitectura monolitica en lugar de microservicios, que consideraste.

Esas decisiones descartadas son algunas de las señales más potentes de criterio técnico, precisamente porque demuestran que consideraste alternativas y tomaste una decisión informada, no que simplemente usaste lo primero que se te ocurrio o lo que genero la IA por defecto.

Una sección breve en el README o en un archivo de decisiones arquitectónicas, no más de cinco o seis decisiones, cada una con dos o tres frases de justificacion, transforma un proyecto de portfolio en una conversación técnica que ya empezo antes de la entrevista.

El criterio como hábito, no como presentación

El criterio técnico no se puede añadir como una capa sobre un proyecto terminado. Se construye durante el desarrollo, en cada decisión, en cada commit, en cada test, en cada línea de documentación.

Lo que si puedes hacer con un proyecto existente es auditarlo con honestidad: los commits cuentan una historia, los tests cubren lo que importa, el README corresponde con lo que hay, hay decisiones técnicas que no estan documentadas en ningún sitio.

> *Esa auditoria, y el trabajo de mejorar lo que encuentres, es exactamente la demostracion de criterio que estas buscando.*

PARTE 4 · OPEN SOURCE Y NETWORKING

Contribuir con intención: como elegir y que aporta

Contribuir a proyectos de open source aparece en casi todos los consejos sobre como mejorar un perfil de GitHub. El problema es que ese consejo suele quedarse en el nivel de contribuye a open source sin explicar que contribuir, a que proyectos, con que criterio, y por que eso importa más que añadir un repositorio propio más.

Por que las contribuciones a terceros pesan diferente

Un proyecto propio demuestra que puedes llevar algo de principio a fin. Una contribución a un proyecto externo demuestra algo distinto y complementario: que puedes entender código que no escribiste, integrarte en las convenciones de un equipo que no elegiste, comunicarte de forma efectiva con personas que no conoces, y entregar algo que supera el criterio de revisión de otra persona.

Esas son exactamente las habilidades que se ejercen en cualquier trabajo técnico real. Y son muy difíciles de demostrar con proyectos propios, por muy bien construidos que esten.

> *Un Pull Request aceptado en un proyecto activo con mantenedores reales es una validación externa de tu trabajo. No la diste tu: la dio alguien que no tiene ningún incentivo para ser amable contigo.*

Como elegir a que proyectos contribuir

No todos los proyectos son igual de útiles para un portfolio. El criterio de selección tiene tres dimensiones:

Alineacion con tu stack objetivo	Si buscas trabajo como backend con Java, contribuir a un proyecto de Python puede ser interesante para aprender, pero no construye la señal específica que necesitas. Los proyectos que uses en tu stack diario, frameworks, librerias, herramientas, son el punto de partida natural.
Salud del proyecto	Un proyecto activo, con commits recientes, con mantenedores que responden, con una guia de contribución clara, con issues etiquetadas. Eso es un proyecto donde tu contribución va a ser revisada, va a recibir feedback real, y va a quedar registrada como parte de un historial credible. Un proyecto abandonado no construye nada.

Barrera de entrada	Las issues etiquetadas como good first issue existen exactamente para esto: tareas que los mantenedores han identificado como abordables por alguien nuevo, sin necesidad de conocer toda la base de código. No son issues menores en terminos de valor: estan diseñadas para que nuevos contribuidores puedan hacer un aporte real.

Señales concretas de un proyecto sano:

— README con instrucciones claras
— Archivo CONTRIBUTING con proceso documentado
— Issues con labels good first issue o help wanted
— Actividad reciente en PRs y commits
— Mantenedores que responden en plazos razonables

Por donde empezar: documentación y tests

Hay una forma de contribuir a open source que tiene una barrera de entrada muy baja, produce PRs que se aceptan con frecuencia, y deja un rastro de colaboración real en tu historial: mejorar documentación y añadir tests.

La documentación de proyectos open source suele estar incompleta, desactualizada, o escrita asumiendo un nivel de conocimiento previo que los usuarios nuevos no tienen. Encontrar esos huecos y rellenarlos con claridad es una contribución genuinamente útil que cualquier mantenedor agradece.

Los tests tienen un valor adicional: escribir un test para una funcionalidad existente obliga a entender como funciona esa funcionalidad en profundidad. Es una forma de aprendizaje activo que además produce algo útil para el proyecto, y que demuestra en tu historial que sabes escribir tests en un codebase real, no solo en proyectos propios.

El flujo de una contribución que cuenta

Una contribución que construye señal en tu perfil no es solo el código que acabas en el PR. Es todo el proceso: como encontraste la issue, como confirmaste el alcance antes de ponerte a trabajar, como comunicaste tu progreso, y como respondiste al feedback de revisión.

1. Confirma antes de empezar	Comenta en la issue que vas a trabajar en ella y describe brevemente tu enfoque. Evita hacer trabajo que no se va a aceptar y genera un registro de participación incluso antes del PR.
2. Abre un PR descriptivo	Descripción del problema, solución implementada, como verificarla. Si el proyecto tiene plantilla de PR, usala. No seguirla es una señal inmediata de que no leiste las guias de contribución.

3. Responde el feedback sin ego	Los mantenedores no tienen tiempo para ser amables por amabilidad: si señalan algo, es porque importa. Incorporar ese feedback bien y agradecer la revisión es parte de lo que construye reputación dentro de una comunidad.

Lo que queda en tu historial

Cada contribución aceptada en un proyecto externo deja tres trazas en tu historial de GitHub: el commit en el repositorio del proyecto, el PR con toda la conversación de revisión, y la referencia en tu perfil de contribuciones.

Esas trazas son verificables por cualquier persona que revise tu perfil. No son una declaracion tuya: son una evidencia externa. Y en un mercado donde el volumen de perfiles crece más rápido que la capacidad de evaluarlos, las evidencias externas pesan más que las declaraciones propias.

> *Una sola contribución bien hecha a un proyecto relevante y activo dice más sobre como trabajas que diez proyectos propios sin contexto. No porque el código sea necesariamente mejor, sino porque paso por el criterio de revisión de otra persona y sobrevivio.*

PARTE 4 · OPEN SOURCE Y NETWORKING

GitHub como grafo social

GitHub no fue diseñado como una red social. Pero funciona como una. Tiene seguidores, actividad pública, discusiones, organizaciones, trending, y un sistema de reputación basado en el trabajo visible. Ignorar esa dimensión es dejar sobre la mesa una parte del valor que la plataforma puede generar para tu carrera.

Este capitulo no es sobre acumular seguidores ni sobre construir una audiencia. Es sobre entender como la visibilidad dentro de la plataforma funciona como señal profesional, y como participar de forma que esa señal sea genuina.

Seguir con criterio, no por volumen

Seguir a personas y organizaciones en GitHub no es solo un gesto social. Tiene consecuencias prácticas en lo que aparece en tu feed, en los proyectos que descubres, y en la comunidad técnica con la que te asocias publicamente.

Seguir a los mantenedores de los proyectos que usas en tu stack tiene un valor inmediato: ves cuando hay nuevas versiones, discusiones sobre el roadmap, decisiones de diseño en proceso. Eso te mantiene actualizada sin esfuerzo adicional y te permite participar en conversaciones relevantes antes de que sean noticias en otros canales.

Seguir a organizaciones técnicas relacionadas con tu sector construye un contexto visible en tu perfil. Cuando alguien revisa a quien sigues, eso cuenta una historia sobre tus intereses y tu nivel de compromiso con el ecosistema.

> *El conteo de seguidores importa menos de lo que parece. La calidad de las conexiones, proyectos a los que contribuyes, discusiones donde participas, importa más.*

GitHub Discussions como escenario de reputación

Las GitHub Discussions son uno de los espacios menos aprovechados por personas que buscan construir visibilidad técnica. Muchos proyectos las usan como foro principal para preguntas, propuestas de features, y debates sobre el roadmap. Participar ahi tiene un valor diferente al de abrir issues o PRs.

Cuando respondes una pregunta en las Discussions de un proyecto relevante, o cuando participas en un debate técnico con un argumento bien construido, estas demostrando capacidad de comunicación técnica asíncrona en público. Eso es exactamente lo que los equipos remotos necesitan y lo que es difícil de evaluar en una entrevista de una hora.

La calidad importa más que la frecuencia. Una respuesta bien elaborada que resuelve un problema real tiene más valor que diez respuestas de una línea. El objetivo no es estar en todas partes: es que cuando aparezcas, lo que digas valga la pena.

El valor de mantener proyectos activos

Hay una diferencia que los evaluadores técnicos notan rápido entre un repositorio que fue subido y abandonado y uno que esta activo. No tiene que ser actividad diaria ni semanal. Tiene que ser actividad que responde a lo que pasa: cerrar issues cuando se resuelven, actualizar dependencias, responder a preguntas, añadir documentación cuando alguien pregunta algo que no estaba claro.

Para proyectos de portfolio, el mantenimiento activo es además una forma natural de generar actividad pública consistente sin forzar commits. Cada issue cerrada, cada PR revisado, cada etiqueta añadida contribuye al historial de forma organica.

Como el ecosistema de GitHub cambio en 2025-2026

GitHub en 2025 no es la misma plataforma que era hace tres años. El lanzamiento de GitHub Copilot Free a finales de 2024 fue seguido de un salto sin precedentes en nuevos usuarios, más de 36 millones en un año. Eso cambio la dinámica de visibilidad dentro de la plataforma de forma significativa.

GitHub MCP server	Permite a las herramientas de IA interactuar directamente con repositorios: buscar issues, crear PRs, revisar código. La estructura y la documentación de tus repositorios importa no solo para humanos sino para sistemas que los procesan de forma automatizada.
Copilot coding agent	Disponible para todos los suscriptores de pago desde 2025. Puedes asignar issues directamente a Copilot y recibir un draft PR como resultado. La calidad con la que escribes los issues se convierte en una señal adicional de criterio técnico y de capacidad para trabajar con IA productivamente.
GitHub Spark	Lanzado en Universe 2025. Permite construir micro-aplicaciones directamente desde la plataforma con lenguaje natural. Para un portfolio, abre la posibilidad de tener aplicaciones funcionales y desplegadas sin la friccion de configurar infraestructura desde cero.

La reputación como resultado, no como objetivo

La trampa en la que cae mucha gente al pensar en visibilidad en GitHub es tratar la reputación como un objetivo en si mismo: conseguir seguidores, acumular estrellas, aparecer en trending. Esas métricas son consecuencias de hacer trabajo bueno en público, no estrategias que se puedan ejecutar directamente.

Lo que si puedes ejecutar directamente es el trabajo: contribuir a proyectos con criterio, mantener tus repositorios activos, participar en discusiones con argumentos solidos, y documentar tu proceso de

forma que sea útil para otros. La reputación, si llega, llega como resultado de eso.

> *Si no llega en forma de seguidores o estrellas, sigue siendo real en forma de historial verificable, de PRs aceptados, de conversaciones técnicas registradas. Que es exactamente lo que alguien que te evalua para un trabajo va a buscar.*

PARTE 5 · POR PERFILES

Señales específicas por rol

Todo lo que hemos visto hasta aqui aplica a cualquier perfil técnico. Pero hay una capa adicional que marca diferencia: cada especialidad tiene señales propias que los evaluadores buscan de forma específica. Usa este capitulo para identificar las señales de tu especialidad y asegurarte de que estan presentes en tus repositorios de portfolio.

Frontend y Full-stack

El frontend es el rol donde la demo más importa. El código puede ser impecable, pero si no hay nada visual que mostrar, la evaluación queda incompleta. Una captura de pantalla o un GIF en el README que muestre la interfaz funcionando es el primer filtro que supera o no supera un proyecto de frontend.

Más allá de lo visual, lo que diferencia un perfil de frontend con criterio es la arquitectura de componentes. Los componentes son reutilizables o esta todo acoplado, hay separación entre lógica y presentación, se usa algun sistema de diseño o las decisiones de UI son ad hoc. Eso es lo que un tech lead revisa cuando abre el código.

TypeScript es en 2026 el estándar esperado para cualquier proyecto de frontend profesional, no un diferenciador. Si tus proyectos de portfolio usan JavaScript sin tipos, eso genera una pregunta sobre si estas al dia con las prácticas actuales del sector.

— Tests de componentes con Testing Library
— CI que ejecuta tests y linter en cada push
— Accesibilidad mínima: atributos alt, estructura semántica HTML
— TypeScript como lenguaje principal
— Demo visual en el README: captura o GIF
— Para full-stack: coherencia entre frontend y backend en el mismo proyecto

Backend

En backend, la señal más valorada es la arquitectura. No la complejidad, la claridad. Un proyecto donde la lógica de negocio esta aislada de la infraestructura, donde los modelos de datos tienen sentido, dice más sobre la madurez técnica de quien lo construyo que un proyecto con muchas funcionalidades pero sin estructura.

Las APIs son el producto visible del backend, y la forma en que estan documentadas importa. Un proyecto con especificacion OpenAPI o Swagger, aunque sea mínima, demuestra que quien lo construyo piensa en quien va a consumir la API, no solo en que funcione.

Para Java específicamente, que sigue siendo el lenguaje más demandado en España segun los datos de cierre de 2025, Spring Boot con arquitectura hexagonal o clean architecture, tests con JUnit y

Mockito, y documentación de la API con SpringDoc OpenAPI son las señales que mejor reconocen los evaluadores en el mercado local.

— Separación clara de capas: dominio, aplicación, infraestructura
— Documentación de API con OpenAPI o Swagger
— Manejo de errores consistente con códigos y mensajes útiles
— Tests de integración sobre los endpoints principales
— Logs estructurados sin exponer información sensible
— Variables de entorno documentadas en .env.example

Data Engineer y AI Engineer

Los proyectos de data tienen un reto específico: el código solo cuenta parte de la historia. El proceso, como se exploraron los datos, que decisiones se tomaron, que se descarto y por que, es igual de importante que el resultado final.

Los notebooks de Jupyter que convierten tienen celdas de texto que explican que se esta haciendo y por que, visualizaciones que iluminan los datos antes de modelar, y una estructura narrativa que lleva al lector desde el problema hasta la conclusión.

Para AI Engineers en 2026, las señales que más peso tienen son las relacionadas con sistemas de IA en producción, no con experimentos de notebooks. RAG implementado con fuentes de datos reales, evaluación sistemática de la calidad de las respuestas, gestión de prompts como código versionado, y observabilidad básica del sistema son las señales que distinguen un perfil que ha llevado IA a producción.

— Notebooks con narrativa: markdown que explica decisions, no solo código
— Reproducibilidad: requirements.txt o Dockerfile, datos documentados
— Para AI Engineer: RAG con fuentes reales y evaluación de calidad
— Visualizaciones que cuentan algo, no solo gráficos por defecto
— Pipeline reproducible de principio a fin
— Métricas de evaluación documentadas y justificadas

DevOps y Platform Engineer

En DevOps, el portfolio es diferente por naturaleza: el trabajo principal es infraestructura, y la infraestructura es difícil de mostrar en un repositorio de la misma forma que el código de aplicación.

La forma de resolverlo es tratar la infraestructura como código y documentar lo que hace. Un repositorio con scripts de Terraform o Ansible que aprovisionen una infraestructura real, con un README que explique que se despliega, por que se eligieron esos servicios, y como se puede reproducir el entorno, es una demostracion concreta de competencia.

La diferencia entre un pipeline básico y uno que demuestra criterio DevOps esta en que hace ese pipeline: si solo ejecuta tests, es básico. Si ejecuta tests, linting, análisis de seguridad, construye una

imagen Docker, la pública en un registry y despliega en staging con cada merge, eso demuestra comprension del ciclo completo.

— Infraestructura como código: Terraform, Ansible o CloudFormation
— Pipeline CI/CD completo: tests, build, security scan, deploy
— Dockerfiles optimizados con multi-stage builds
— CodeQL y Dependabot activados en repositorios públicos
— Secret scanning activado y HTTPS en demos desplegadas
— README que explica la arquitectura de infraestructura

Mobile

El portfolio de mobile tiene un problema específico: compilar y ejecutar una aplicación movil tiene una friccion altisima para quien evalua. Si alguien tiene que instalar Xcode o Android Studio, clonar el proyecto, resolver dependencias y compilar para ver que hace la aplicación, la mayoria no lo va a hacer.

La solución es multiple. El enlace a la App Store o Google Play es la forma más directa de demostrar que la aplicación existe en el mundo real y que paso el proceso de revisión de las plataformas. Si la aplicación no esta publicada, un video de demostracion de dos a tres minutos que muestre las funcionalidades principales resuelve el problema de forma efectiva.

La arquitectura es la segunda señal que buscan los evaluadores de mobile. MVVM es el patrón más extendido tanto en iOS como en Android, y su presencia en el proyecto comunica que quien lo construyo conoce las prácticas estándar del sector.

— Enlace a App Store o Google Play si la app esta publicada
— Video de demostracion como alternativa si no esta publicada
— Arquitectura MVVM documentada en el README
— Manejo de concurrencia en llamadas a API
— Persistencia local: CoreData, Room o equivalente
— Capturas de pantalla de la UI en el README

Resumen por rol

Rol	Señal más diferenciadora
Frontend / Full-stack	Demo visual + TypeScript + coherencia entre capas
Backend	Arquitectura limpia + OpenAPI + tests de integración
Data / AI	Notebooks narrativos + reproducibilidad + IA en producción
DevOps	IaC + pipeline completo + seguridad integrada

Mobile	Demo en video o App Store + MVVM documentado

> *Las señales específicas por rol no reemplazan las señales generales: README claro, tests visibles, CI funcionando y demo accesible siguen siendo el punto de partida para cualquier perfil. Las señales de rol son la capa adicional que comunica especializacion real.*

PARTE 6 · METRICAS Y PLAN DE ACCION

Medir sin engañarte

Hay una trampa en la que cae mucha gente cuando empieza a prestar atención a las métricas de GitHub: optimizar para los números que se ven fácil en lugar de para los números que dicen algo útil.

El grafo de contribuciones es el ejemplo más claro. Es visible, es colorido, y es fácil de manipular con commits automáticos o triviales. Pero no dice nada sobre la calidad del trabajo, y las personas que saben evaluar perfiles lo saben. Optimizar para tener el grafo verde todos los dias es trabajo que no construye nada real.

Las métricas que importan y lo que dicen

GitHub ofrece métricas de trafico para cada repositorio donde tienes permisos de escritura. Se accede desde la pestaña Insights de cada repo, en la sección Traffic. La ventana de datos es de 14 dias, así que si quieres seguimiento a largo plazo, tienes que exportar esos datos periodicamente.

Vistas y visitantes únicos	Las vistas son el total de veces que alguien abrio la página del repositorio. Los visitantes únicos son personas distintas. La diferencia te dice si tienes visitas repetidas de alguien que evalua en profundidad, o muchas visitas de personas que solo pasan una vez.
Clones	La métrica más interesante para un portfolio. Alguien que clona quiere ejecutarlo, leerlo en detalle, o reutilizar algo. Muchos visitantes y pocos clones sugiere que algo en la primera impresion no genera interes suficiente para profundizar.
Referrers	Los sitios de referencia te dicen de donde viene el trafico. Si el 80% viene de LinkedIn, confirma que incluir el enlace funciona. Si viene de GitHub search, válida que el SEO interno esta funcionando.
Contenido popular	Que páginas dentro del repositorio se ven más. Normalmente es el README, pero si hay documentación que genera más trafico, eso te dice que parte del proyecto despierta más interes.

Como acceder a las métricas via API

Para quienes quieren un seguimiento más sistemático, GitHub expone estas métricas a traves de su API REST. Con GitHub CLI instalado y autenticado:

```
gh api /repos/{owner}/{repo}/traffic/views
```

```
gh api /repos/{owner}/{repo}/traffic/clones
gh api /repos/{owner}/{repo}/traffic/referrers
gh api /repos/{owner}/{repo}/community/profile
```

Estos endpoints devuelven los datos de los últimos 14 dias con desglose diario. Si quieres construir un histórico, un script sencillo con cron o con GitHub Actions que los exporte periodicamente es suficiente para tener visibilidad a largo plazo.

La métrica que no existe en GitHub: entrevistas generadas

Con todo lo que GitHub ofrece para medir trafico y actividad, hay una métrica que no existe de forma nativa y que es la que más importa: cuántas entrevistas ha generado el perfil.

Esa métrica hay que construirla de forma externa. La forma más simple es una hoja de calculo donde registras cada proceso en el que participas, con la fuente de origen y si el GitHub fue mencionado o revisado durante el proceso. Sin esa métrica, estas optimizando a ciegas.

Que no medir

Número de estrellas	Las estrellas las pone cualquiera por cualquier razón. No hay correlacion demostrada entre número de estrellas y calidad de un repositorio ni con la probabilidad de conseguir empleo a traves de el.
Número de seguidores	Genera la sensacion de audiencia sin que esa audiencia necesariamente tenga valor para el objetivo concreto de encontrar trabajo en tu especialidad.
Commits en el grafo	Manipulable de demasiadas formas para ser una señal fiable. Los evaluadores técnicos con experiencia no lo usan como criterio primario.

> *Las métricas que importan son las que reflejan intención real: visitantes únicos, clones, referrers, y entrevistas generadas. Todo lo demas es ruido con apariencia de señal.*

PARTE 6 · METRICAS Y PLAN DE ACCION

Automatizaciones que suman vs las que restan credibilidad

GitHub Actions es una de las herramientas más potentes para demostrar madurez técnica en un portfolio, y también una de las más mal usadas. La diferencia entre una automatización que suma y una que resta no esta en la complejidad técnica. Esta en si la automatización sirve a un propósito real o si existe solo para generar actividad visible.

Automatizaciones que suman

CI con tests en cada push	Un workflow que se activa con cada push, ejecuta los tests, y muestra el resultado en un badge en el README. Es la automatización más básica y la más importante. El badge verde confirma que existe un mecanismo de verificación automática.
Linting y formateo automático	Un workflow que ejecuta el linter en cada PR y falla si hay violations. Comunica que el proyecto tiene estándares de código explícitos y facilita la revisión de cambios.
Dependabot para actualizaciones	Configurar Dependabot para que abra PRs automáticos cuando hay actualizaciones de dependencias. Requiere exactamente un archivo de configuración y comunica que el proyecto esta mantenido activamente.
Releases automáticas con changelog	Para proyectos con versiones, un workflow que genera release notes a partir de los mensajes de commit cuando se crea un tag. Herramientas como release-please lo hacen con configuración mínima.
README de perfil con contenido dinámico	Automatizar secciones del README de perfil con Actions y scripts: últimos posts de un blog, repos más recientes, métricas de actividad. Demuestra que sabes combinar Actions con scripts de forma práctica.

Automatizaciones que restan credibilidad

Commits automáticos vacios	Un workflow que hace un commit vacio todos los dias para mantener el grafo verde no engaña a nadie con experiencia técnica. El patrón es reconocible por la regularidad perfecta y la ausencia de contenido real.
PRs generadas por agentes sin revisión	Tener PRs que claramente fueron generadas por IA sin evidencia de revisión humana, sin comentarios, sin iteración, sin ajustes, es una señal de que el repositorio no esta realmente mantenido por una persona con criterio.

Badges que no corresponden a nada real	Añadir badges de cobertura o build status que apuntan a servicios no configurados o que muestran valores ficticios destruye la credibilidad del README de forma inmediata.
Contenido generado sin valor añadido	Un README que se actualiza solo para añadir última actualización: hoy todos los dias, o que genera estadisticas irrelevantes, no demuestra habilidad técnica. Demuestra que priorizas la apariencia sobre el contenido.

Antes de añadir cualquier automatización: esta automatización resuelve un problema real o genera actividad que parece productiva? Si resuelve un problema real, suma. Si principalmente genera actividad visible, es mejor no añadirla.

PARTE 6 · METRICAS Y PLAN DE ACCION

Checklist: tu GitHub listo en 90 minutos

Todo lo que hemos cubierto en este ebook puede parecer mucho trabajo si se mira como un todo. Pero la mayoria de los cambios que más impacto tienen son rápidos de implementar una vez que sabes que hacer y por que.

Este checklist esta diseñado para que puedas dejar tu perfil en un estado funcional y profesional en una sesion de trabajo concentrada de 90 minutos. No es el estado final optimo: es el punto de partida solido desde el que iterar.

Bloque 1 — Perfil base 15 minutos

- ☐ Foto real y clara, fondo neutro, que se vea la cara
- ☐ Nombre completo o nombre profesional buscable
- ☐ Ubicación en formato estándar: Barcelona, Spain o Madrid, Spain
- ☐ Bio de 160 caracteres: rol + stack + señal de disponibilidad
- ☐ Activar Disponible para contratar en configuración si aplica
- ☐ Email o enlace de contacto visible en el perfil
- ☐ Enlace a LinkedIn o portfolio si tienes

Bloque 2 — README de perfil 25 minutos

- ☐ Crear repositorio público con el mismo nombre que tu usuario si no existe
- ☐ Sección de presentación: rol, stack, tipo de problemas que resuelves
- ☐ Dos o tres resultados concretos o proyectos destacados con enlace
- ☐ Habilidades agrupadas por area sin listar tecnologías que no usas
- ☐ Información de contacto clara al final
- ☐ Sin widgets de estadisticas, sin animaciones, sin GIFs decorativos

Bloque 3 — Repositorios anclados 25 minutos

- ☐ Elegir tres a cinco proyectos para anclar con criterio editorial
- ☐ Nombre del repositorio descriptivo y buscable
- ☐ Descripción de una línea: problema + stack + resultado
- ☐ Topics entre 8 y 12: stack, dominio, tipo de proyecto
- ☐ README con TL;DR, instalación rápida y demo o capturas
- ☐ Badge de CI si hay workflow configurado
- ☐ Demo desplegada o enlace a video de demostracion si es posible

Bloque 4 — Señales de calidad y IA 25 minutos

- ☐ Workflow básico de CI en al menos uno de los repositorios anclados
- ☐ Dependabot activado en repositorios con dependencias
- ☐ Sección de decisiones técnicas en el README del proyecto principal
- ☐ Si usaste IA, una línea de transparencia explicando como y que verificaste
- ☐ Issues o PRs con descripciones completas si hay alguna abierta

Errores comunes que anulan el trabajo anterior

Email del commit no vinculado	Las contribuciones no aparecen en el historial si el email del commit no esta conectado a tu cuenta. Verificalo en la configuración de contribuciones de GitHub.
README de perfil no aparece	Si el repositorio fue creado antes de julio de 2020 puede no mostrarse automáticamente. Activalo manualmente con la opción Share to profile.
Ubicación no estándar	Bcn, Barcelona o Spain sin formato consistente pueden excluirte de búsquedas por location. Usa Barcelona, Spain o Madrid, Spain.
Topics con espacios o mayusculas	Los topics solo admiten minusculas, números y guiones. Los que tienen espacios o mayusculas no funcionan como etiquetas.
Forks sin contribuciones propias anclados	Los forks sin commits propios cuentan menos que proyectos propios o contribuciones reales a proyectos externos.

Lo que viene después

El checklist de 90 minutos es el punto de partida. Lo que construye un perfil realmente solido a lo largo del tiempo es más simple de describir que de ejecutar: seguir haciendo trabajo bueno, documentarlo bien, y mantener los proyectos activos.

No hay atajo para eso. Pero con el perfil base en orden, cada proyecto nuevo, cada contribución a open source, y cada conversación técnica en Discussions añade señal real que cualquier evaluador puede verificar.

> *Tu GitHub no es un CV estatico. Es un sistema vivo que refleja como trabajas. Cuánto más lo trates como tal, más trabajo hara por ti.*

ANEXO

Plantillas listas para usar y recursos curados

Plantillas de bio por perfil

Listas para copiar y adaptar. Criterio: rol claro, stack principal, señal de calidad o disponibilidad, en 160 caracteres o menos.

Perfil	Bio lista para copiar
Frontend	Frontend dev · React, TypeScript, CSS · Componentes accesibles y bien testeados · Buscando rol en producto · Proyectos con demo abajo
Backend	Backend dev · Java, Spring Boot, PostgreSQL · APIs REST con tests y CI · Arquitectura limpia · Open to work · Ver repos anclados
Fullstack	Fullstack · React + Spring Boot + PostgreSQL · Deploy en Railway · Tests, CI y docs en todos mis proyectos · Buscando rol en producto
Data Engineer	Data Engineer · Python, dbt, Airflow, SQL · Pipelines reproducibles y documentados · Buscando equipo con cultura de datos real
AI Engineer	AI Engineer · Python, LangChain, PostgreSQL · RAG en producción, evaluación sistemática · Busco equipo que lleve IA a producción real
DevOps	DevOps · Terraform, Docker, Kubernetes, GitHub Actions · IaC y CI/CD como práctica habitual · Open to work · Ver repos anclados
Mobile iOS	iOS dev · Swift, SwiftUI, MVVM · Apps publicadas en App Store · Tests y arquitectura limpia · Buscando rol en equipo de producto
Mobile Android	Android dev · Kotlin, Jetpack Compose, MVVM · Apps en Google Play · Coroutines, Room, tests · Busco primer rol o cambio de empresa
Junior	Dev Jr · React y Node · Tres proyectos completos con demo, tests y CI · Busco primera oportunidad · Feedback bienvenido

Plantilla de README de perfil

```
# Hola, soy [Tu Nombre]

**Rol objetivo:** [Frontend / Fullstack / Backend / Data / DevOps]

**Stack principal:** [Tus tecnologías principales]

**Ubicación:** [Ciudad, País / Remote]

## En 30 segundos
```

```
- Construyo [tipo de apps] con foco en [calidad / producto / escala].
- Trabajo con tests y CI en todos mis proyectos.
- Público demos y READMEs para ejecutar en menos de 5 minutos.

## Proyectos destacados

- **[Proyecto 1]** — [Problema que resuelve] · [Stack] · [Link demo]
- **[Proyecto 2]** — [Que implementaste] · [Link repo]
- **[Proyecto 3]** — [Contribución OSS] · [Link]

## Habilidades

**Frontend:** [...] **Backend:** [...] **Testing:** [...]
**DevOps / CI:** [...] **Datos / IA:** [...]

## Contacto

Email: [...] | LinkedIn: [...] | Portfolio: [...]
```

Topics estrategicos por rol

Rol	Topics recomendados
Frontend	`react, typescript, nextjs, tailwindcss, testing-library, jest, ci-cd, frontend, portfolio`
Backend Java	`java, spring-boot, postgresql, rest-api, jwt, docker, junit, clean-architecture, backend`
Backend Python	`python, fastapi, postgresql, rest-api, pytest, docker, ci-cd, backend, portfolio`
Fullstack	`fullstack, react, typescript, spring-boot, postgresql, docker, ci-cd, portfolio`
Data	`python, pandas, sql, dbt, airflow, data-engineering, etl, pipeline, portfolio`
AI Engineer	`python, langchain, rag, llm, fastapi, postgresql, ai-engineering, portfolio`
DevOps	`terraform, docker, kubernetes, github-actions, ci-cd, infrastructure-as-code, devops`
Mobile iOS	`swift, swiftui, ios, mvvm, coredata, xcode, mobile, portfolio`
Mobile Android	`kotlin, android, jetpack-compose, mvvm, room, coroutines, mobile, portfolio`

Queries de búsqueda que usan los reclutadores

Queries reales dentro de GitHub para sourcing de perfiles en España:

```
location:barcelona language:typescript
location:barcelona language:java
location:madrid language:python followers:>20
location:spain language:javascript repos:>5
location:"Barcelona" language:kotlin
location:spain language:go
```

Queries de X-Ray search en Google para encontrar perfiles y CVs:

```
site:github.com "location * Barcelona" "java"
site:github.com "location * Madrid" "python"
site:github.io (cv OR resume) "frontend developer" "Spain"
site:github.com "location * Spain" "typescript" followers
```

Recursos curados

Necesidad	Recurso	URL
Badges estaticos	**Shields.io**	`shields.io/badges`
Badges dinámicos	**Shields.io endpoint**	`shields.io/badges/endpoint-badge`
Logos de tecnologías	**Simple Icons**	`simpleicons.org`
Iconos de stacks	**Devicons**	`github.com/devicons/devicon`
Skill icons	**Skill Icons**	`skillicons.dev`
Generador README perfil	**Rahul Dkjain**	`github.com/rahuldkjain/github-profile-readme-generator`
Generador README rápido	**GPRM**	`gprm.itsvg.in`
Stats cards	**GitHub Readme Stats**	`github.com/anuraghazra/github-readme-stats`
Typing SVG	**Readme Typing SVG**	`github.com/DenverCoder1/readme-typing-svg`
Banner capsule	**Capsule Render**	`github.com/kyechan99/capsule-render`
Snake contributions	**Platane/snk**	`github.com/Platane/snk`
Trofeos GitHub	**GitHub Profile Trophy**	`github.com/ryo-ma/github-profile-trophy`
Diagramas en Markdown	**Mermaid**	`mermaid.js.org`

Tu GitHub como motor de empleo · tu-mtch.com · 2026