للمطوّرين: من الفوضى إلى النتيجة، كتابة أوامر Claude

رامي الخطيب

محتوى هذا الكتيب

- لماذا يكتب معظم المطوّرين Prompts للذكاء الاصطناعي بشكل سيّئ
- تشريح prompt جيّد
- كتابة prompts لمهام تطوير محددة
- العمل مع Claude Code

محتوى هذا الكتيب

- متى لا تستخدم Claude
- إدارة استخدامك دون الاصطدام بالحدود
- أخطاء شائعة وإصلاحات سريعة

وقت القراءة المتوقع : 23 دقيقة

من المستهدف بهذا الكتيب

هذا الكتيب لك إذا كنت:

- مطوّرًا يستخدم Claude أو أي أداة ذكاء اصطناعي يومياً ويريد إجابات أدق من أول رسالة بدل جولات متعددة من الأخذ والرد
- مطوّر مبتدئ أو متوسط يريد أن يفهم كيف يتعامل مع AI coding agents مثل Claude Code بشكل منهجي لا عشوائي

من المستهدف بهذا الكتيب

- قائد فريق أو مدير تقني يريد أن يفهم لماذا يستنزف الفريق ميزانية الذكاء الاصطناعي بسرعة — وكيف يوقف ذلك
- أي مطوّر يشعر أن النتائج التي يحصل عليها من الذكاء الاصطناعي متذبذبة وغير موثوقة

المقدمة

ربما تستخدم Claude أو Gemini يومياً بالفعل — وربما تستخدم أيضًا Claude Code. السؤال ليس هل تستخدم الذكاء الاصطناعي، بل هل تستخدمه بشكل جيّد: هل تحصل على الإجابة الصحيحة من المرة الأولى، هل تختار الأداة المناسبة للمهمة، وهل تتجنب استنزاف حدّ الاستخدام لديك على أشياء لم تكن تستحق ذلك.

المقدمة

يغطي هذا الكتيب الصورة الكاملة — من عادة كتابة الـ prompt الأساسية، مرورًا بالعمل مع AI coding agents، وصولًا إلى معرفة متى لا ينبغي اللجوء إلى الخيار الأقوى (والأغلى). كُتب هذا من واقع عمل المطوّرين اليومي، مع أمثلة يمكنك تطبيقها مباشرة في جلستك القادمة.

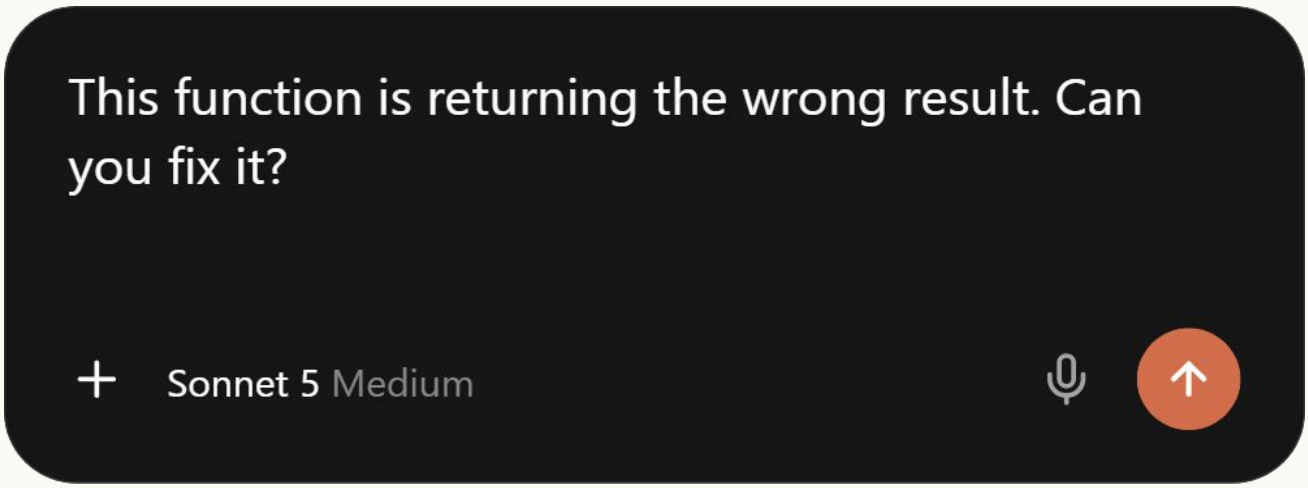
لماذا يكتب معظم المطوّرين أوامر للذكاء الاصطناعي بشكل سيّئ

تفتح Claude تكتب سؤالًا سريعًا، تحصل على إجابة متوسطة، تكتب متابعة، تحصل على إجابة أفضل قليلًا، ثم متابعة أخرى... هل يبدو ذلك مألوفًا؟

هذا هو السبب الرئيسي الذي يجعل النتائج تبدو متذبذبة. ليس لأن النموذج غير موثوق، بل لأن الـ prompt الأول لم يقدّم له ما يكفي ليعمل عليه.

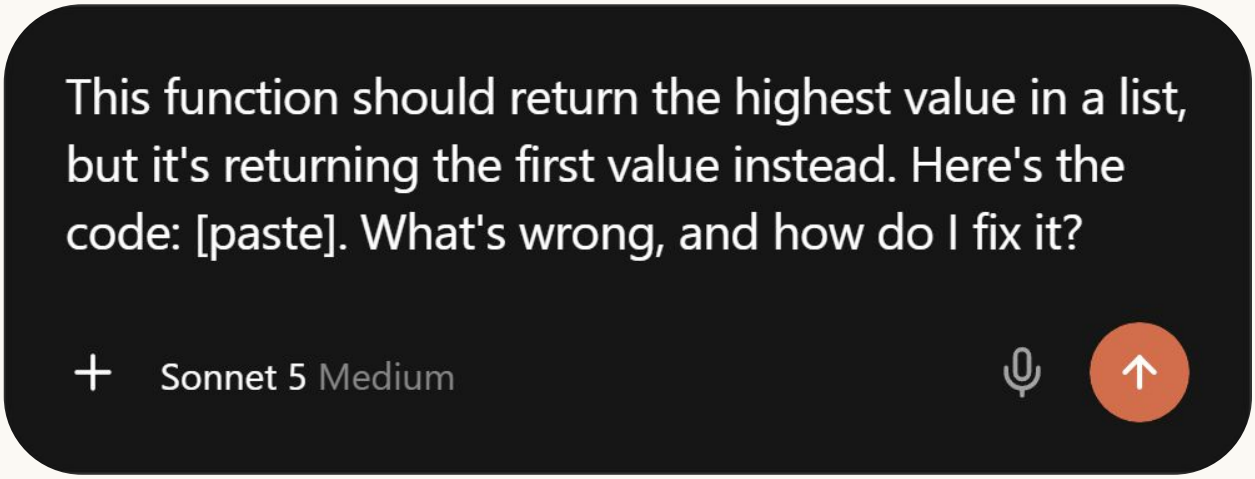
طلب إصلاح كود بدون سياق يشبه الاتصال بميكانيكي والقول: "سيارتي تصدر صوتًا، أصلحها"، ثم إغلاق الهاتف قبل أن يسأل: أي سيارة؟ أي صوت؟ ومتى يحدث؟ قد يخمّن بشكل صحيح. غالبًا لن يفعل.

أمر سيئ:



ماذا يحدث بعدها؟ يسأل الذكاء الاصطناعي ماذا تفعل الدالة. تشرح. يسأل عن الكود. تلصقه. يسأل ماذا يعني "wrong". بعد خمس رسائل، أخيرًا تحصل على إجابة مفيدة — وقد صرفت خمسة أضعاف الجهد الذي كان يجب أن يستغرقه الأمر.

أمر أفضل:



النمط:

الأمر السيئ يجعل الذكاء الاصطناعي يخمّن ما تريد. الأمر الجيد يخبره بالضبط بما لديك وما تريده.

قبل أن تضغط إرسال، تحقّق:

- ▢ هل أدرجت الكود/الخطأ الحقيقي، وليس مجرد وصف له؟
- ▢ هل حدّدتَ ما الذي يعنيه "النجاح"؟
- ▢ هل يستطيع الذكاء الاصطناعي الإجابة دون سؤال متابعة؟

تشريح Prompt الجيد

كل أمر فعّال مبني من أربعة أجزاء.

1. السياق Context: ما هو الوضع؟ الذكاء الاصطناعي لا يعرف مشروعك إلا إذا أخبرته.

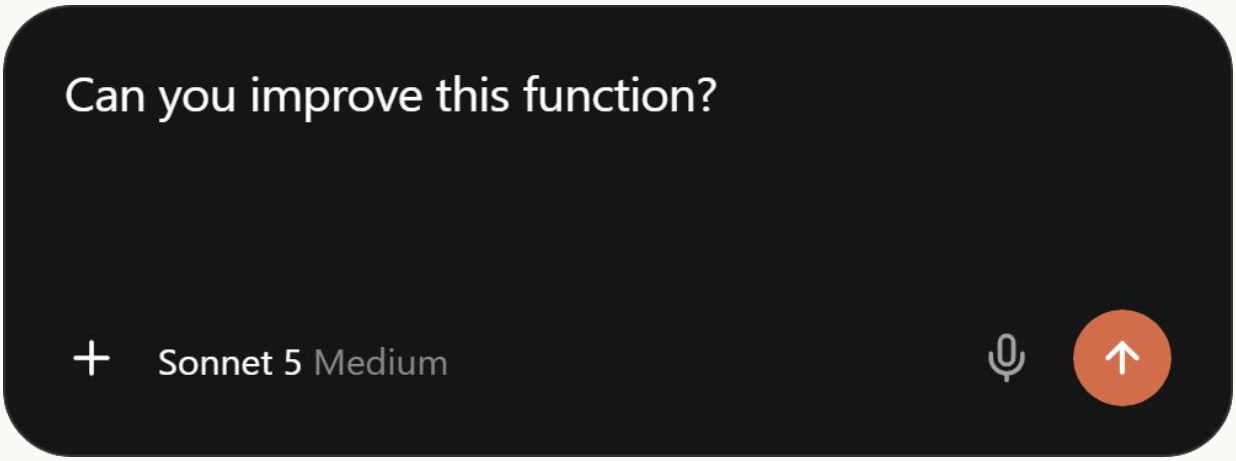
2. المادة Material: الكود أو الخطأ الحقيقي، منسوخًا كما هو. لا تَصِفه.

3. الهدف Goal: ما هي النتيجة المرجوة؟ إن أمر مثل "أصلح هذا" يحمل هدف غامض.

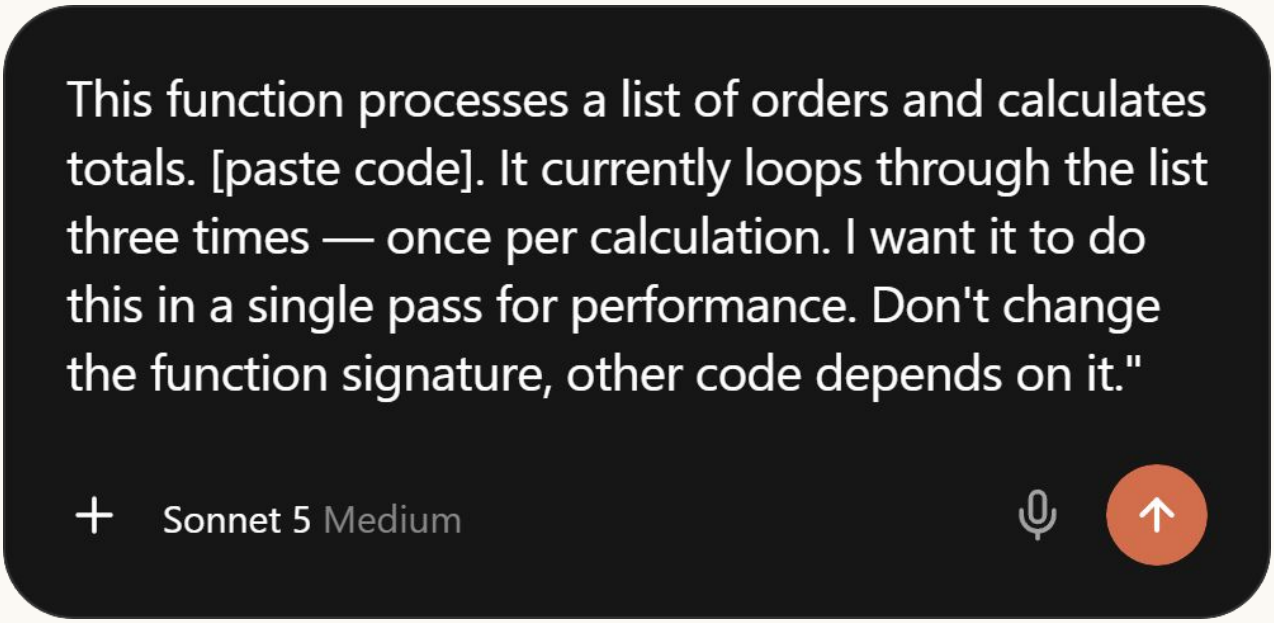
4. القيود Constraints: ما الذي يجب أن يبقى كما هو؟ conventions، libraries، أشياء لا يجب لمسها.

فكّر فيها كأنك تطلب طعامًا بإعطاء شخص كيس بقالة والقول "اطبخ شيئًا لذيذًا" — مقابل أن تقول: "عندي أرز وخضار (Material)، وأريد شيئًا بطابع آسيوي (Context)، خفيفًا لكن مُشبِعًا (Goal)، ومن دون فستق — لدي حساسية (Constraint)." نفس المطبخ، نفس المكوّنات — نتيجة مختلفة تمامًا.

مثال أمر ضعيف:



أمر قوي:



ما الذي تغيّر:

السياق Context: ماذا تفعل الدالة

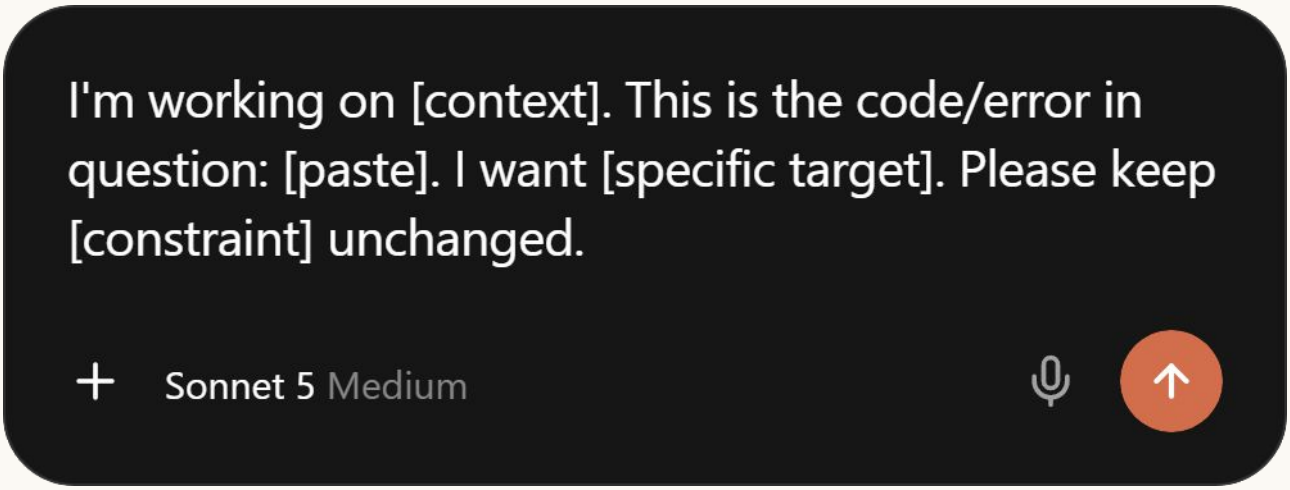
المادة Material: الكود الفعلي

الهدف Goal: مرور واحد بدل ثلاثة loops

القيود Constraints: إبقاء function signature كما هو

هذه أربع جمل — ربما 60 ثانية إضافية — لكنها الفرق بين إصلاح صحيح وقابل للاستخدام فورًا، وبين ثلاث جولات من الأخذ والرد.

قالب قابل لإعادة الاستخدام:

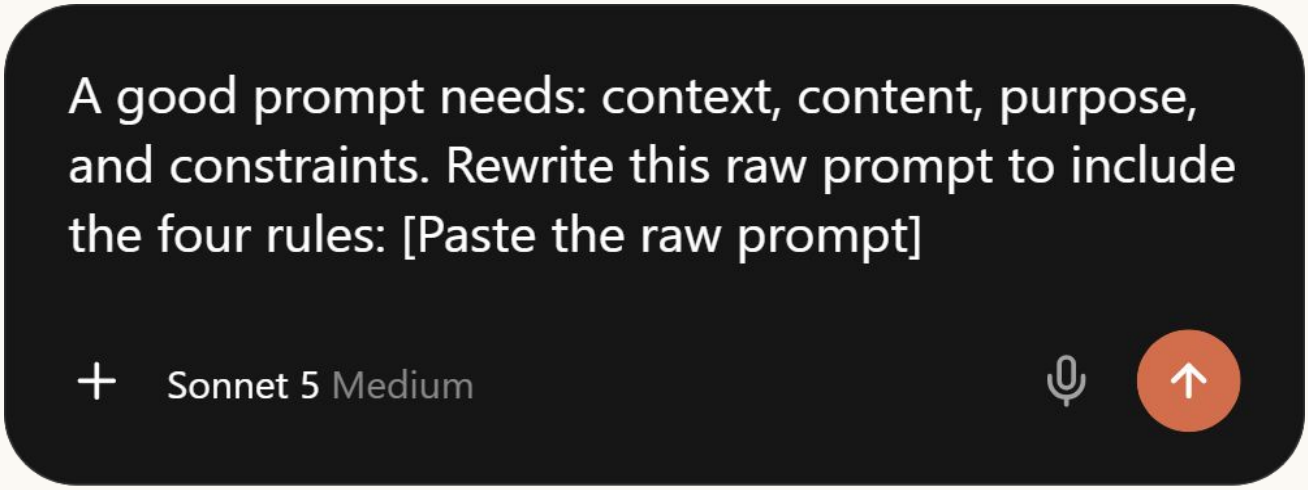


هذا ليس قالبًا جامدًا — بل قائمة تحقق.
طالما أن الأجزاء الأربعة موجودة،
فالصياغة لا تهم كثيرًا.

شيء إضافي: الذكاء الاصطناعي لا يهتم بالنبرة، بل يهتم بالمعلومات. "Please fix this, thanks so much!" بدون سياق أسوأ من طلب مباشر ومكتمل بلا أي مجاملة. اصرف جهدك على الوضوح، لا على المجاملة.

نصيحة: دع أداة مجانية تُحسّن أوامرك أولًا، إذا لم تكن متأكدًا أن الـ prompt لديك يغطي الأجزاء الأربعة — أو كنت على وشك استخدام Claude Code لمهمة تحتاج تخطيطًا جيدًا — استخدم أداة مجانية مثل Gemini كمحرّر سريع أولًا.

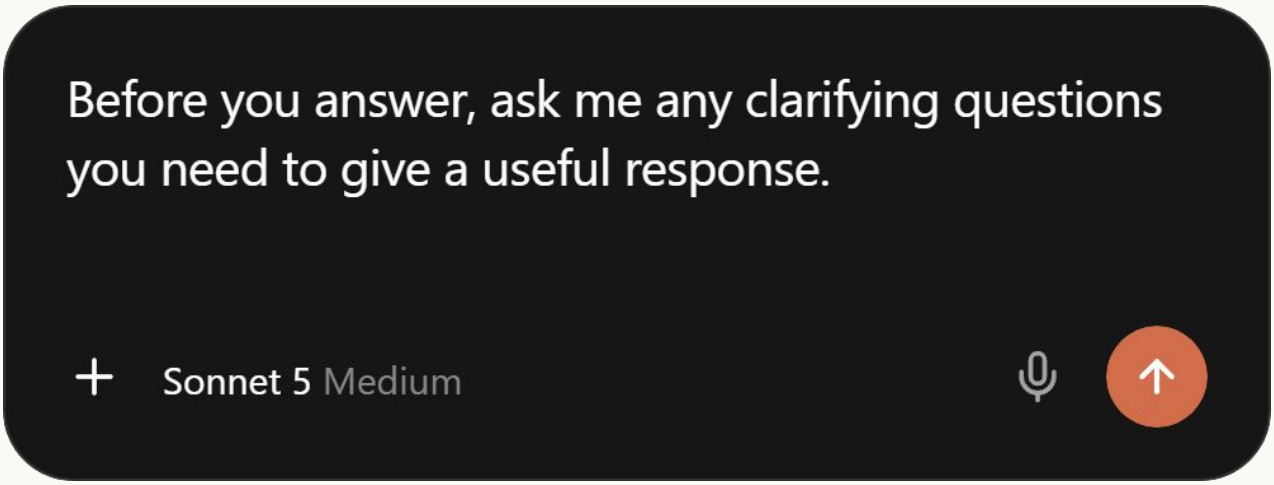
الصق الـ prompt الخام مع قواعد هذا الكتيب (معادلة الأجزاء الأربعة، أو عادة "خطّط قبل التنفيذ" من الفصل 4) واطلب منها إعادة كتابته وفقًا لذلك. مثلًا:



النتيجة: prompt لا يكلّف شيئًا، يأخذ ثوانٍ، ويعني أن الـ prompt الذي سترسله إلى Claude صار حادًا — وبالتالي رسائل أقل مهدورة.

تقنية إضافية: دع الذكاء الاصطناعي يجري مقابلة معك

للمهام الغامضة أو المفتوحة أو التخطيط، أضف هذا إلى الطلب:



هذا يعكس المسؤولية: بدل أن تضع كل شيء من البداية، يخبرك الذكاء الاصطناعي ما الذي ينقص.

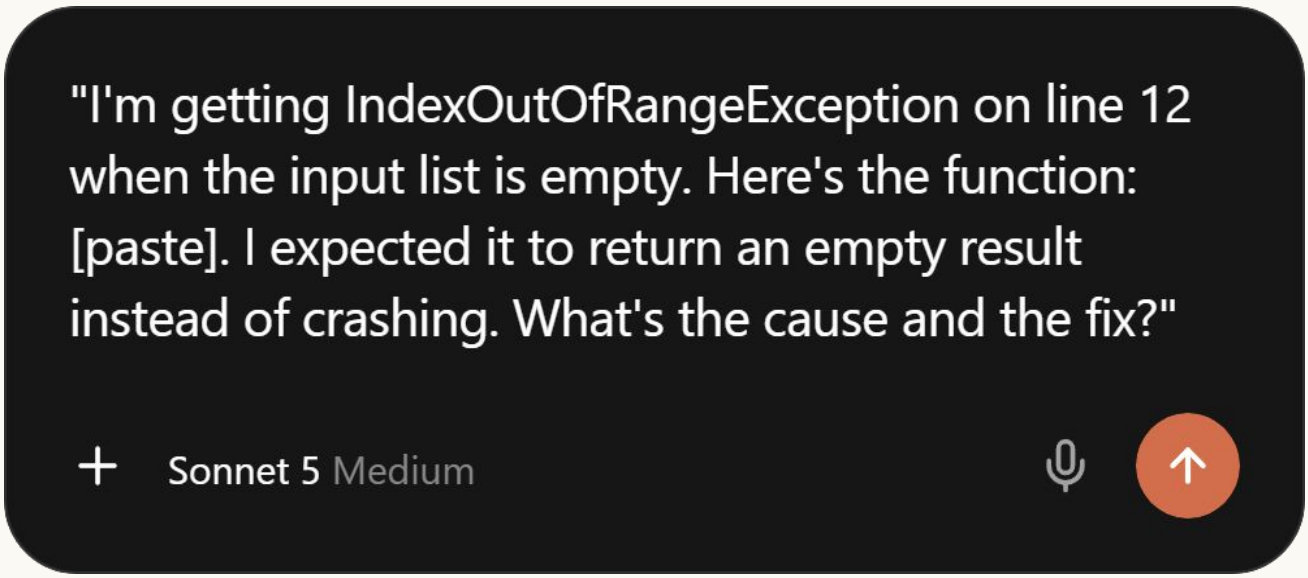
طلبات لمهام تطوير محددة

المهام المختلفة تحتاج أشكال أوامر مختلفة.
هنا ثلاثة من الأكثر شيوعًا.

1. تصحيح الأخطاء Debugging

خطأ شائع: لصق الخطأ وحده ثم سؤال "?why"

الحل: الخطأ + الكود المرتبط + ما الذي كنت تتوقعه.



جميع النماذج موجودة هنا:

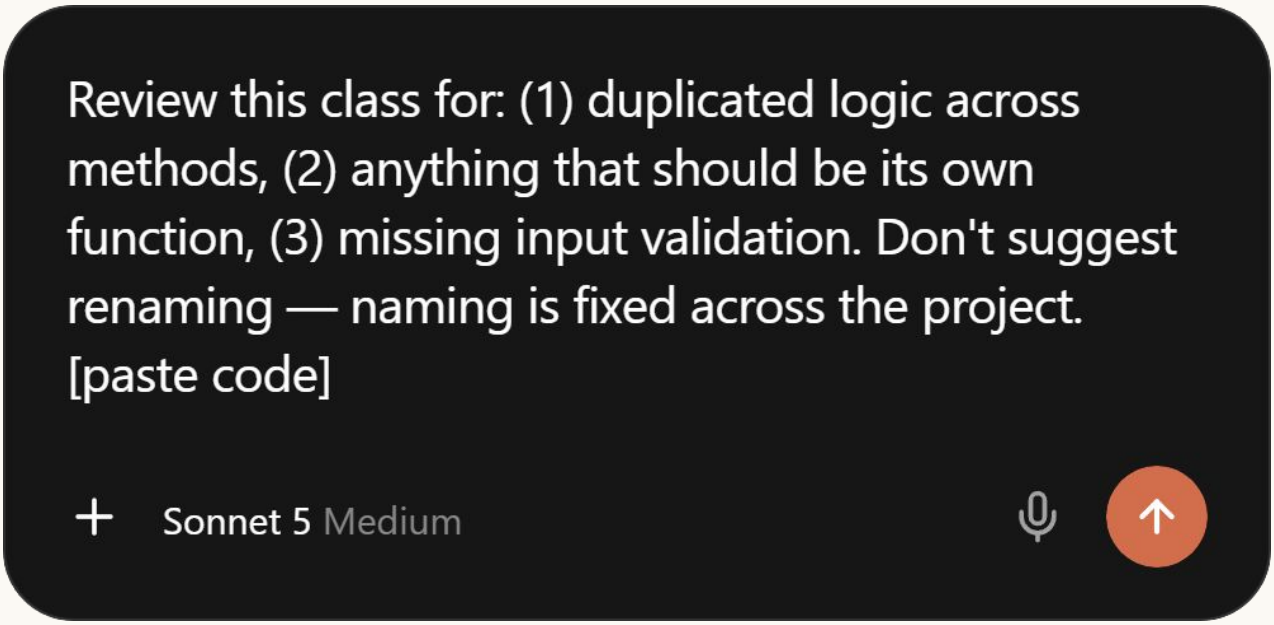
Cheatsheet

مراجعة الكود / إعادة هيكلته

Code review / refactoring

خطأ شائع: "review this code" — واسع جدًا، وستحصل على ملاحظات عامة.

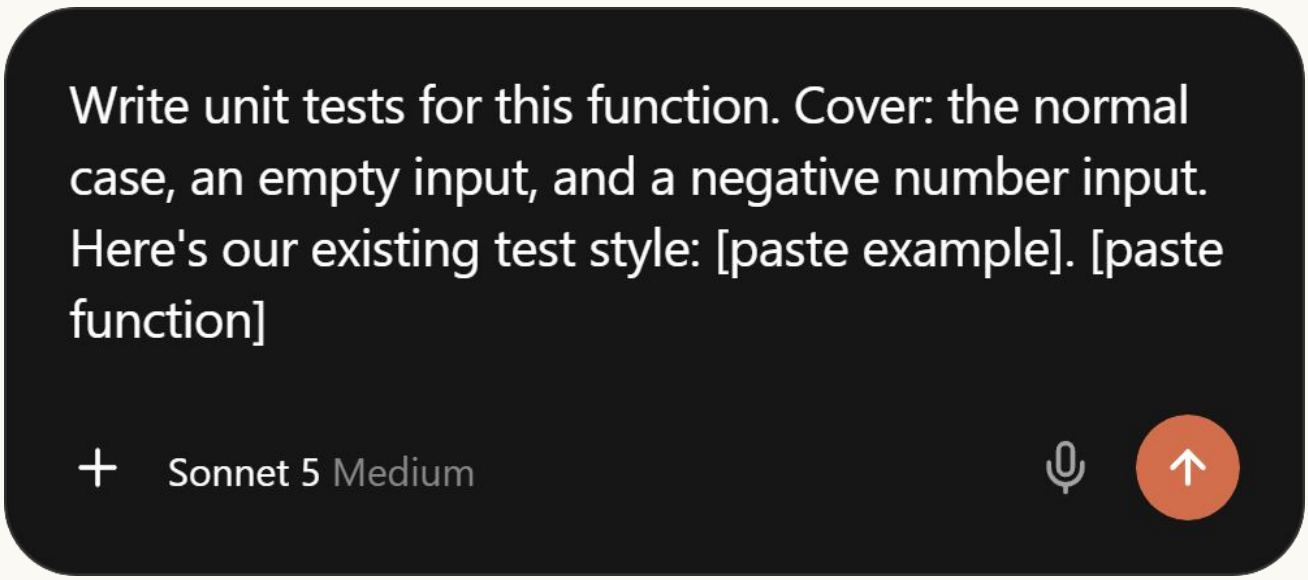
الحل: قل ما الذي تُحسّنه بالضبط.



توليد الاختبارات (Generating tests)

الخطأ: "اكتب اختبارات لهذا" غالبًا ينتج اختبارات سطحية ومسارًا سعيدًا فقط.

الإصلاح: حدّد التغطية التي تحتاجها فعليًا.

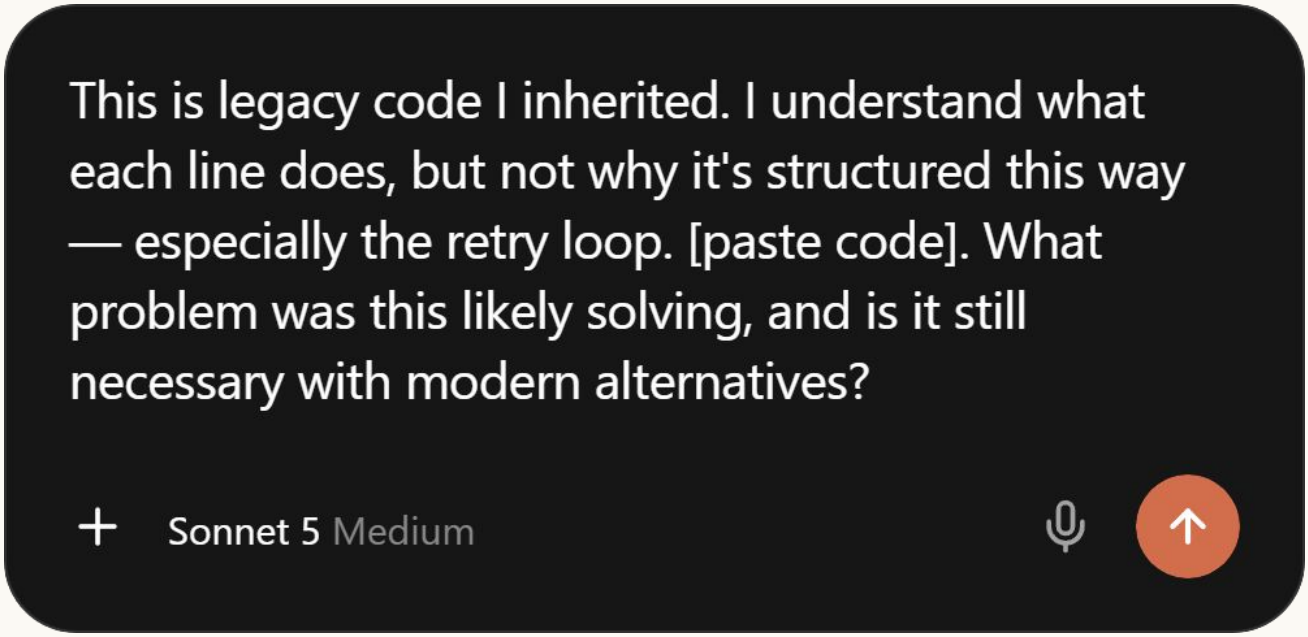


إعطاء مثال على أسلوب الاختبارات يبقي الاختبارات المولّدة متسقة مع الموجود في قاعدة الكود.

فهم كود قديم/موروث (Understanding unfamiliar/legacy code)

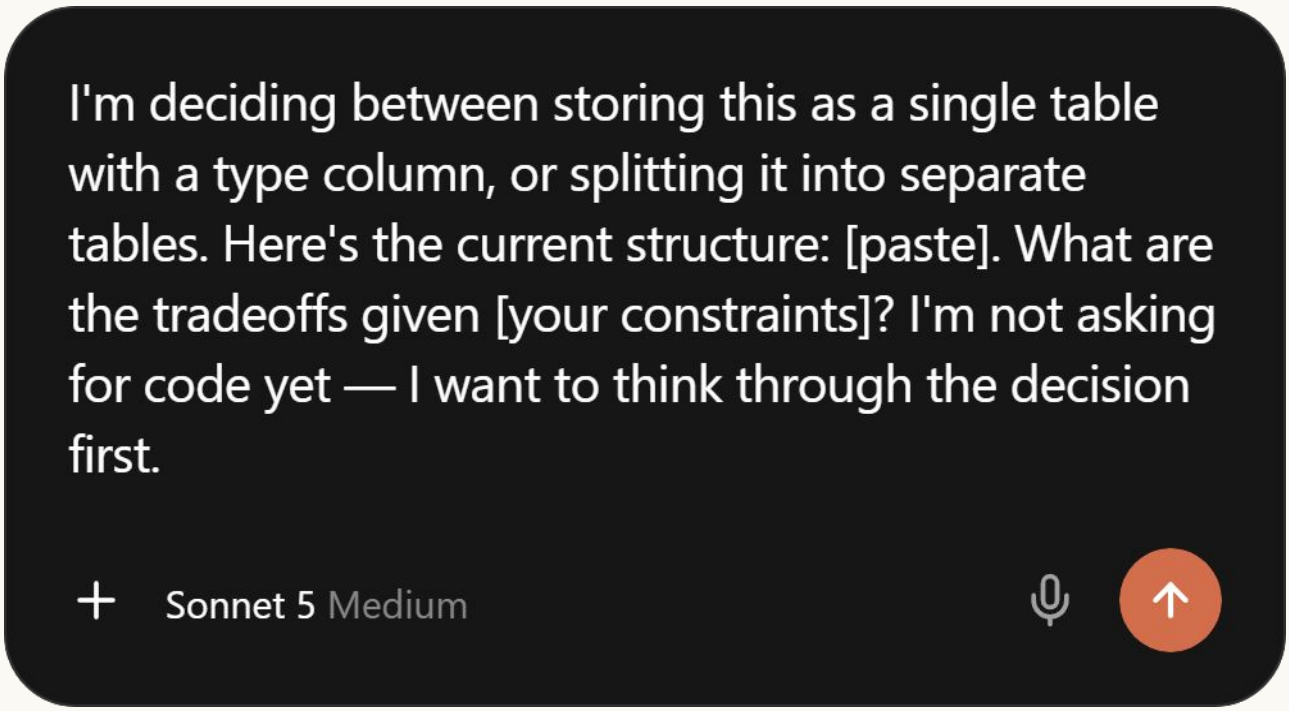
الخطأ: "اشرح هذا الكود" يعطي سردًا سطرًا بسطر يمكنك فهمه بنفسك غالبًا.

الإصلاح: اسأل عن لماذا لا عن ماذا.



نقاشات التصميم Design discussions

هنا يكون الذكاء الاصطناعي الأكثر فائدة — والأقل استخداماً. بدل طلب كود، اطلب محادثة.



هذه الجملة الأخيرة — "I'm not asking for code yet" — تنقل الذكاء الاصطناعي من وضع "generate output" إلى وضع "think with me"، وهذا ينتج نقاشات معمارية أفضل بكثير.

العمل مع Claude Code

إن Claude Code يستطيع قراءة ملفاتك، تشغيل أوامر، والتعديل عبر قاعدة كودك. هذه القوة تحتاج أسلوبًا مختلفًا عن الدردشة في المتصفح.

1. جهّز ملف سياق للمشروع

معظم أدوات AI coding تدعم ملفًا يُقرأ تلقائيًا في بداية كل جلسة — التقنية المستخدمة، تنظيم المجلدات، معايير الكتابة، الأشياء التي يجب تجنبها. اكتبه مرة واحدة.

مثال:

بدونه ستعيد شرح إعداداتك في كل جلسة. ومعه تبدأ كل جلسة وهي مُطّلعة مسبقًا.

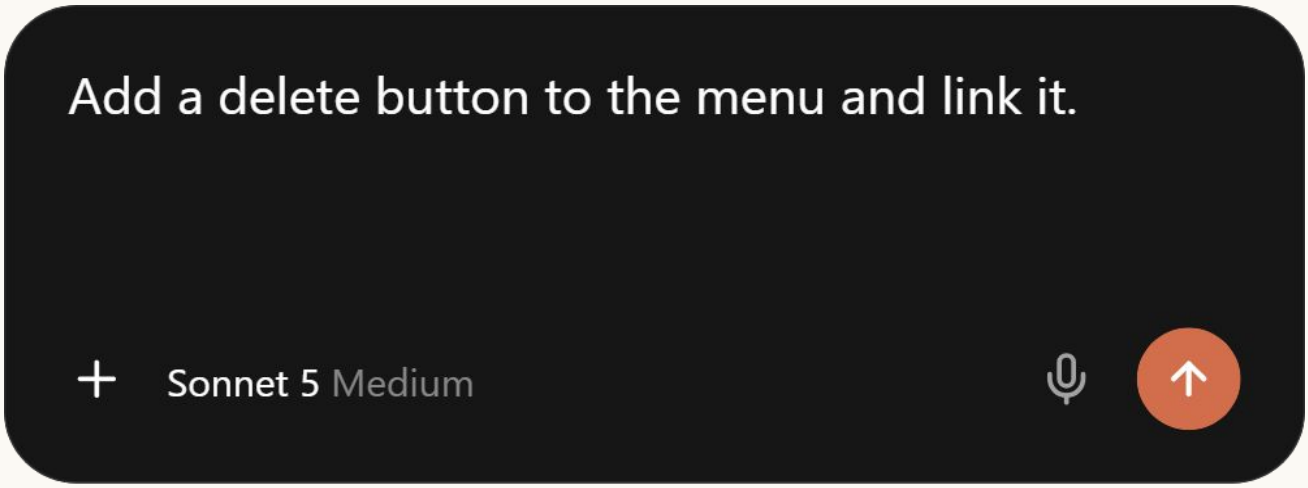
2. خطّط قبل التنفيذ

أكبر فارق بين المبتدئين والمتمرسين: المبتدئون يقفزون مباشرة إلى "افعل". المتمرسون يطلبون خطة أولًا.

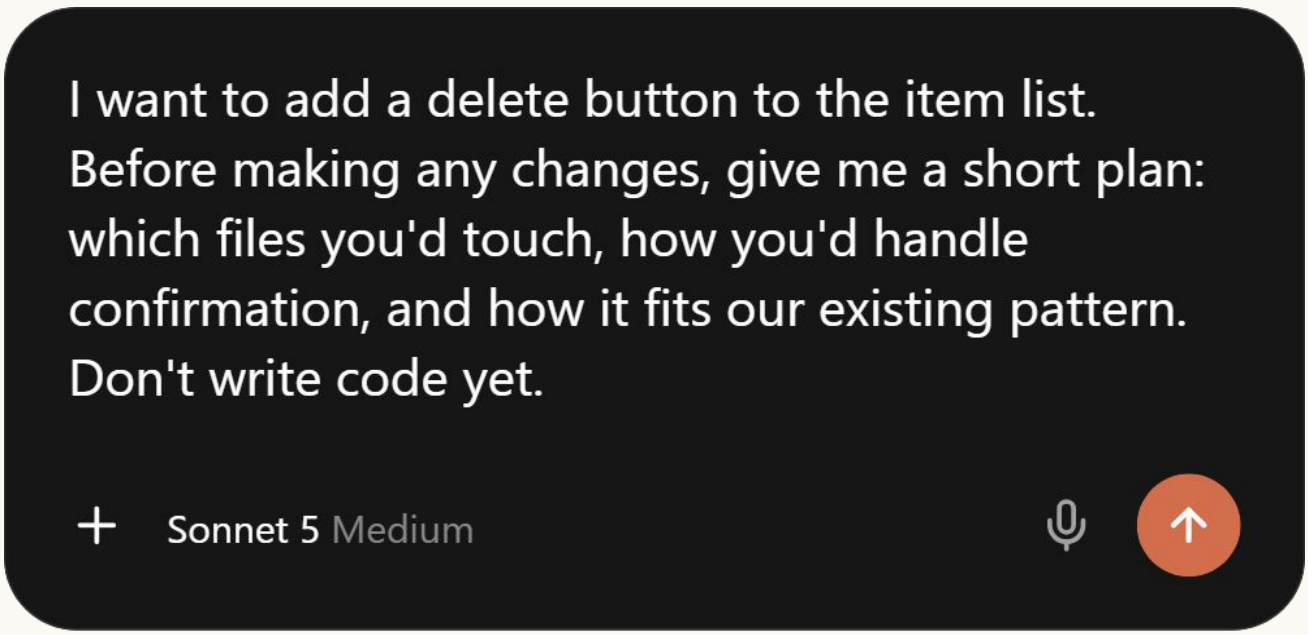
Claude Code يدعم فعليًا وضعاً مخصصاً لهذا يسمى Plan Mode — وهو وضع يجعل Claude يحلّل مشروعك ويقترح خطة تنفيذ كاملة قبل تعديل أي ملف. هذا ليس مجرد أسلوب تواصل جيد، بل ميزة مدمجة تقلل الأخطاء بشكل كبير عند التعامل مع أكثر من 3 ملفات في آن واحد.

ترك AI agent يعدّل دون خطة يشبه أن تعطي شخصًا مفاتيح بيتك وتقول: "غيّر الديكور، فاجئني." قد ينجح، أو قد يطلي غرفة النوم وأنت نائم فيها. طلب الخطة أولًا يعني فقط رؤية "عينات الطلاء" قبل أن يبدأ العمل.

ضعيف:


Add a delete button to the menu and link it.
Sonnet 5 Medium

أفضل:


I want to add a delete button to the item list. Before making any changes, give me a short plan: which files you'd touch, how you'd handle confirmation, and how it fits our existing pattern. Don't write code yet.
Sonnet 5 Medium

بعد أن توافق على الخطة، حينها اطلب التنفيذ. هذا يلتقط سوء الفهم قبل كتابة أي كود — وهو أرخص بكثير من اكتشافه بعد ذلك.

3. طابق النموذج (Model) مع المهمة

عند استخدام Claude Code، يمكنك اختيار أي نموذج تريد تشغيل المهمة عليه. النموذج هنا يعني الإصدار نفسه من الذكاء الاصطناعي — مثلاً Claude Haiku (أصغر وأسرع وأرخص) مقابل Claude Sonnet أو Opus (أكبر وأقدر وأغلى تكلفةً من الاستخدام).

القاعدة البسيطة:

نماذج أصغر/أسرع (مثل Claude Haiku) — للمهام الصغيرة والمحددة جداً:

- إعادة تسمية متغير
- إضافة null check على دالة موجودة
- تصحيح خطأ إملائي في تعليق
- تنسيق كود موجود

نماذج أكبر/أقدر (مثل Claude Sonnet أو Opus) — للمهام التي تحتاج تفكيراً عميقاً:

- التخطيط لميزة جديدة تمس عدة ملفات
- قرارات معمارية ("هل أستخدم Microservice أم Monolith؟")
- تحليل كود (Debug) معقد مع stack trace طويل
- مراجعة كود شاملة مع اقتراح إعادة هيكلة

مثال واقعي:

تخيّل أنك تريد إضافة null check بسيطة على دالة — هذا لا يحتاج Sonnet. لأن Haiku يحلّها في ثانية بتكلفة أقل بكثير في الاستخدام. لكن لو كنت تخطط لإضافة نظام permissions كامل يمس 10 ملفات مختلفة — هنا Sonnet أو Opus يستحق التكلفة لأن الخطأ في التخطيط سيكلفك أكثر من فارق الاستخدام.

استخدام Sonnet أو Opus لإصلاح typo هو من أكثر مصادر هدر الاستخدام شيوعاً — مثل استئجار طائرة خاصة لتوصيل رسالة إلى الجار.

4. استعمال Git كشبكة أمان

قبل أن تسمح لـ AI agent بإجراء تغييرات متعددة الملفات، اعمل commit للحالة الحالية أو اعمل على branch. راجع الـ diff كما لو كان Pull Request — بنفس الانضباط الذي تستخدمه مع تغييرات مطوّر مبتدئ.

متى لا تستخدم Claude

استخدام النموذج الأقوى لإصلاح خطأ إملائي يشبه الاتصال بمهندس إنشائي لتعليق إطار صورة. الخبرة ليست "مهدورة" لأنها سيئة — بل لأنها لم تكن مطلوبة.

ليس كل شيء يستحق Claude. معرفة الفرق هي ما يميّز من يستخدم الذكاء الاصطناعي بشكل جيد ممن يستخدمه فقط بكثرة.

تخطَّ الذكاء الاصطناعي تمامًا عندما:

- أسرع أن تفعلها يدويًا من أن تشرحها (إعادة تسمية متغير، إصلاح typo، تغيير إعداد سطر واحد)
- أنت تعرف الإجابة أصلًا وتريد "تأكيدًا" فقط — هذه عادة، ليست prompt

استخدم أداة أخف (Gemini أو نموذج أصغر) عندما:

- سؤال معلوماتي سريع ("ما syntax الخاص بـ X؟")
- تنسيق بسيط، ترجمة سريعة، إعادة صياغة قصيرة

احتفظ بـ Claude / Claude Code لـ:

- مهام متعددة الخطوات حيث السياق مهم
- أي شيء يلمس عدة ملفات أو يحتاج خطة
- قرارات معمارية، debugging مع stack traces حقيقية، مراجعة وفق معايير حقيقية

للمدراء وقادة الفرق:

إذا كان استهلاك ميزانية الذكاء الاصطناعي في الفريق عاليًا، فغالبًا ليس لأن الأداة "مكلفة جدًا" — بل لأن الاستخدام يذهب لمهام لم تكن تحتاجها: lookups سريعة، متابعات متكررة، تعديلات سطر واحد عبر نموذج ثقيل.

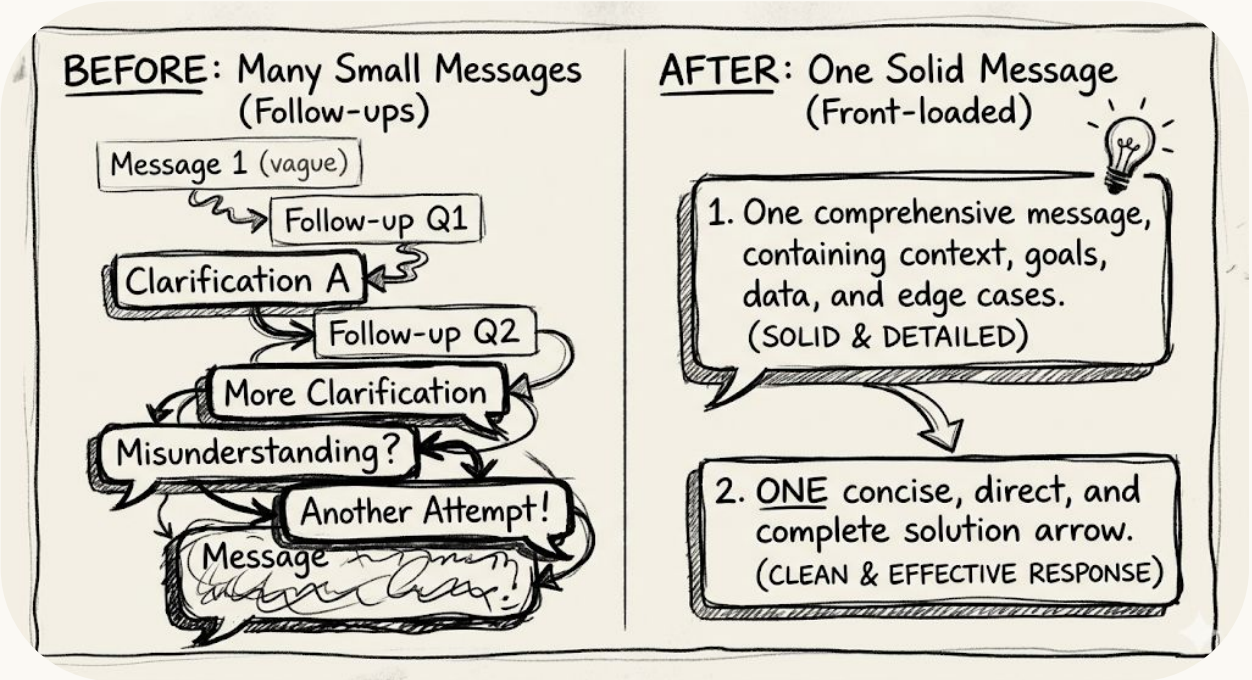
سؤال مفيد قبل فتح AI coding agent: "هل هذا يحتاج تخطيطًا أو سياقًا متعدد الملفات — أم أنا فقط أبحث عن معلومة؟" الحالة الثانية مكانها أرخص. هذه ليست محاولة لتقييد الذكاء الاصطناعي — بل لمطابقة الأداة مع المهمة، وهذا غالبًا يعني إنجاز أكثر ضمن نفس الميزانية.

إدارة استخدامك دون الاصطدام بالحدود

هذه عادات، وليست حيلًا.

1. ضع كل شيء من البداية، ولا تعتمد على المتابعات

كل متابعة تعيد إرسال كامل المحادثة — كأنك تعيد سرد يومك كاملًا من البداية كلما أضفت تفصيلًا صغيرًا. في النهاية ستقضي وقتًا في إعادة السرد أكثر من السرد نفسه. حوار من 10 رسائل ذهابًا وإيابًا يكلّف أكثر بكثير من prompt واحد مكتمل من البداية. لهذا فصل 2 (معادلة الأجزاء الأربعة) مهم للتكلفة، وليس فقط للجودة.



2. ابدأ محادثة جديدة لموضوع جديد

تغيير الموضوع داخل نفس المحادثة يجر سياقًا قديمًا معك حتى لو لم يعد ذا صلة. موضوع جديد = محادثة جديدة. أرخص وغالبًا أدق.

هذه النقطة تحتاج شرحاً تقنياً بسيطاً لتفهم السبب الحقيقي وراءها.

كيف يقرأ Claude محادثتك؟

Claude لا "يتذكر" ما قلته في المحادثة كما يتذكر الإنسان. في كل مرة ترسل رسالة جديدة، يقرأ Claude كامل تاريخ المحادثة من أولها — كل رسالة أرسلتها، وكل رد أعطاك إياه، من أول سطر حتى آخر رسالة. ثم يجيب على سؤالك الجديد.

هذا يعني أن كل رسالة جديدة تُرسل تحمل معها وزن كل ما سبقها.

تشبيه يوضح الفكرة:

تخيّل أنك تطلب من مساعد أن يقرأ ملف كامل قبل أن يجيب على كل سؤال تطرحه.

في البداية الملف صفحة واحدة — القراءة سريعة والإجابة سريعة.

بعد 20 رسالة، الملف أصبح 20 صفحة. المساعد لا يزال يقرأ كل الـ 20 صفحة قبل كل إجابة — حتى لو سؤالك الجديد لا علاقة له بالصفحات الأولى.

بعد 50 رسالة، الملف 50 صفحة. كل سؤال جديد يكلّف أكثر — في الوقت، وفي الاستخدام.

ماذا يحدث عندما تغيّر الموضوع في نفس المحادثة؟

لنقل أنك بدأت محادثة تناقش فيها مشكلة في API معين، وبعد 15 رسالة انتهيت منها وقررت تسأل عن شيء مختلف تماماً — مثلاً كيف تكتب Unit Tests.

Claude سيقرأ الـ 15 رسالة الخاصة بالـ API كاملةً قبل أن يجيب على سؤال الـ Unit Tests — رغم أنها لا علاقة لها بالموضوع الجديد إطلاقاً. أنت تدفع من رصيدك ثمن قراءة سياق لا تحتاجه.

الحل:

عندما تنتهي من موضوع وتريد البدء بموضوع جديد — افتح محادثة جديدة.

القاعدة البسيطة: موضوع جديد = محادثة جديدة. دائماً.

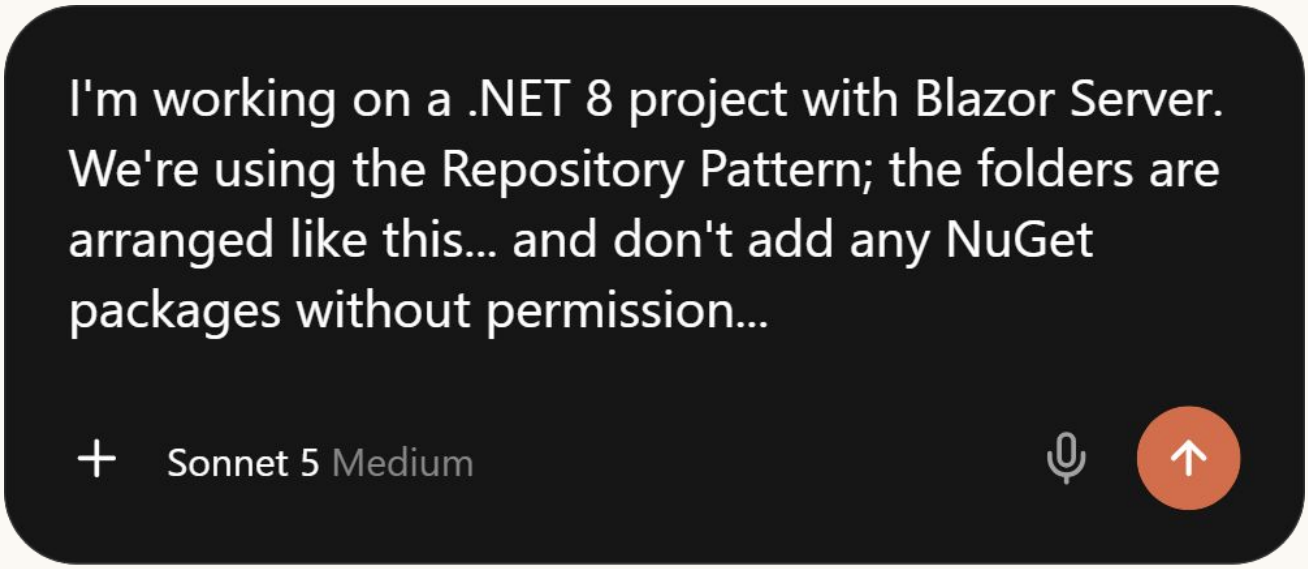
3. اجمع طلباتك (Batch your prompts)

بدلًا من "أصلح هذا" ثم "والآن أيضًا ذاك" وبعدها "وهذا أيضًا" عبر ثلاث رسائل، اجمعها في واحدة: "أصلح هذه الثلاثة: [1] [2] [3]." prompt واحد، تحميل سياق واحد.

4. ضع السياق المتكرر في ملف بدل إعادة كتابته

ما المشكلة أصلاً؟

كثير من المطوّرين يبدأون كل جلسة مع Claude بنفس الشرح:



هذا مضيعة للوقت والاستخدام — خصوصاً إذا كنت تكرره يومياً.

ما هو ملف السياق؟

هو ملف نصي بسيط تضعه في مشروعك مرة واحدة، يقرأه Claude تلقائياً في بداية كل جلسة بدل أن تشرح أنت كل مرة.

في Claude Code، هذا الملف اسمه CLAUDE.md — تضعه في الـ root folder للمشروع.

كيف تُعِدّه؟

ببساطة أنشئ ملف نصي اسمه CLAUDE.md في مجلد المشروع الرئيسي، واكتب فيه المعلومات التي تجد نفسك تكررها دائماً. مثال:

```
# Project Information

## Technology Used

- .NET 8 + Blazor Server
- SQL Server with Entity Framework Core
- Repository Pattern for all data access

## Folder Structure

- /Domains/ — Business Logic
- /Infrastructure/ — Database Connection
- /Components/ — Blazor UI components

## Important Rules

- Do not add new NuGet packages without permission.
- Always use the Repository Pattern; do not write
  SQL directly.
- Always use English variable names.
- Do not modify CSS files — the design is final.
```

كيف يعمل بشكل عملي؟

عندما تفتح Claude Code في مشروعك، يقرأ CLAUDE.md تلقائياً — بدون أي خطوة إضافية منك. كل جلسة تبدأ وClaude يعرف مسبقاً تقنيتك وقواعدك وهيكل مشروعك.

بدل أن تكتب:

"أنا أعمل على NET 8. مع Repository Pattern ولا تغيّر CSS..."

تكتب مباشرة:

"أضف validation على دالة الحفظ في UserService" وClaude يعرف تلقائياً ما الـ pattern المطلوب وما الذي لا يجب لمسه.

ماذا تضع فيه؟

أي شيء تجد نفسك تكرره — لا أكثر ولا أقل. نقطة البداية الجيدة: افتح محادثاتك الأخيرة مع Claude وابحث عن الجمل التي كررتها أكثر من مرتين. تلك الجمل هي محتوى ملفك.

5. طابق حجم النموذج مع حجم المهمة

نفس فكرة الفصل 5 — لا تستخدم النموذج الأقوى لسؤال سطر واحد.

6. انتبه لنظام حدود الاستخدام (Usage Limits) وكيف يُعاد حسابها

كيف يعمل حد الاستخدام في Claude؟

الفكرة الأساسية أولاً:

إن Claude لا يمنحك "رصيداً يومياً" تستهلكه حتى منتصف الليل ثم يُعاد. بدلاً من ذلك، هو يراقب باستمرار: "كم رسالة أرسلتَ في آخر 5 ساعات؟" — هذا السؤال يُعاد حسابه كل لحظة، لا مرة يومياً.

الحد ليس "يومياً تُعاد"، بل هو "نافذة متحركة من 5 ساعات" — كلما مضت ساعات على رسائلك القديمة، فتحت مساحة جديدة تلقائياً.

كيف تستفيد من هذا عملياً؟

إذا ظهرت لك رسالة تقول إنك وصلت للحد — لا تنتظر يوماً كاملاً. انتظر ساعة أو ساعتين فقط، وستجد أن المساحة عادت جزئياً وتستطيع المتابعة. لهذا السبب: المهام غير العاجلة التي تستهلك كثيراً من الاستخدام، أجّلها لوقت لاحق بدل أن تحشرها كلها في جلسة واحدة.

أخطاء شائعة وإصلاحات سريعة

❌ "Why doesn't this work" بدون كود أو خطأ مرفق

✅ ألصق دائماً الكود والخطأ الحقيقي — لا تكتفِ بوصفهما

❌ المتابعة رسالة بعد رسالة

✅ اجمع كل شيء في أمر واحد قبل الإرسال

❌ طلب كود عندما تحتاج قراراً

✅ قل: "let's — Don't write code yet
think this through"

❌ أهداف غامضة ("make this better")

✅ حدد بوضوح كيف يبدو "done"

❌ نفس المحادثة لمواضيع غير مرتبطة

✅ موضوع جديد = محادثة جديدة

العادة الواحدة التي تصلح معظم هذه الأخطاء:

قبل إرسال أي أمر، توقّف 10 ثوانٍ واسأل: "إذا انضم مطور جديد إلى هذا المشروع دون أي معلومات مسبقة، فهل ستكون هذه الرسالة وحدها كافية لمساعدته؟" إذا لا، فأنت تفتقد Context — وكذلك الذكاء الاصطناعي.

الخاتمة

إذا كانت هناك فكرة واحدة تخرج بها من هذا الكتيب فهي: الذكاء الاصطناعي لا يكافئ الذكاء في الصياغة — بل يكافئ الوضوح. المطوّرون الذين يحصلون على أفضل النتائج من Claude وGemini لا يكتبون تعاويذ سحرية. إنهم يصفون حالتهم بشكل كامل من المرة الأولى، ويختارون الأداة المناسبة للمهمة.

السياق Context, المادة Material, الهدف Goal, القيود Constraints. قدّمها من البداية بدل تقطيرها. خطّط قبل التنفيذ. طابق الأداة مع المهمة — أحيانًا Claude، أحيانًا Gemini، وأحيانًا لا ذكاء اصطناعي أصلًا.

كل عادة في هذا الكتيب تعود إلى مبدأ واحد: prompt كامل يعني جودة أعلى وتكلفة أقل، لأن جودة الـ prompting وكفاءة الاستخدام هما نفس المهارة.

لا شيء من هذا يتطلب أن تكون "خبيرًا في الذكاء الاصطناعي". يتطلب نفس الحدس الذي تستخدمه كمطوّر: قبل أن تلمس لوحة المفاتيح، اعرف ما الذي تبنيه وما الذي لديك بالفعل.

ابدأ صغيرًا. اختر عادة واحدة — كتابة prompts مكتملة من البداية، أو كتابة ملف سياق للمشروع، أو ببساطة التوقف قبل "لماذا لا يعمل هذا؟" لتلصق الخطأ الفعلي.

طبّقها لمدة أسبوع. ستلاحظ غالبًا رسائل متابعة أقل، إجابات أولى أفضل، واستخدامًا يدوم أطول.

الأدوات ستستمر بالتغيّر. عادة التواصل الواضح معها لن تصبح قديمة.

إذا ساعدك هذا الكتيب، فسأكون سعيدًا فعلًا بسماع ما الذي نجح وما الذي لم ينجح — ملاحظاتك تشكّل ما يأتي لاحقًا.

رامي الخطيب