شهادة Claude Certified Architect — Foundations

دليل الدراسة (بناءً على دليل الاختبار الرسمي)

المقدمة

تؤكد شهادة **Claude Certified Architect — Foundations** أن المتخصص قادر على اتخاذ قرارات موازنة سليمة عند تنفيذ حلول واقعية مبنية على Claude. يقيّم الاختبار المعرفة الأساسية بـ Claude Code وClaude Agent SDK وClaude API وModel Context Protocol (MCP)، وهي التقنيات المحورية لبناء تطبيقات إنتاجية باستخدام Claude.

تعتمد أسئلة الاختبار على سيناريوهات صناعية واقعية: بناء أنظمة وكيلة لخدمة العملاء، وتصميم خطوط بحث متعددة الوكلاء، ودمج Claude Code في CI/CD، واستخراج بيانات منظمة من مستندات غير منظمة.

المرشح المستهدف

المرشح المثالي هو **معماري حلول** يصمم ويطلق تطبيقات إنتاجية باستخدام Claude. ينبغي أن تكون لديك خبرة عملية لا تقل عن 6 أشهر في:

- **Claude Agent SDK** — تنسيق عدة وكلاء، والتفويض إلى وكلاء فرعيين، وتكامل الأدوات، وخطافات دورة الحياة
- **Claude Code** — ملفات CLAUDE.md، ومهارات الوكيل، ووضع التخطيط، وخوادم MCP
- **Model Context Protocol (MCP)** — الأدوات والموارد لتكامل الأنظمة الخلفية
- **هندسة المطالبات** — مخططات JSON، وأمثلة few-shot، وقوالب استخراج البيانات
- **نوافذ السياق** — العمل مع المستندات الطويلة، وتمرير السياق بين عدة وكلاء
- **خطوط CI/CD** — مراجعة الكود الآلية، وتوليد الاختبارات
- **التصعيد والموثوقية** — معالجة الأخطاء، وإدخال الإنسان في الحلقة

صيغة الاختبار

البند	القيمة
نوع السؤال	اختيار من متعدد (إجابة صحيحة واحدة من 4)
نظام الدرجات	من 100 إلى 1000، ودرجة النجاح **720**
عقوبة التخمين	لا توجد (أجب عن كل الأسئلة!)
السيناريوهات	4 من أصل 8 سيناريوهات محتملة (يتم اختيارها عشوائيًا)

محتوى الاختبار: 5 مجالات

المجال	الوزن
1. بنية الوكلاء والتنسيق	**27%**
2. تصميم الأدوات وتكامل MCP	**18%**
3. إعداد Claude Code وسير العمل	**20%**
4. هندسة المطالبات والمخرجات المنظمة	**20%**
5. إدارة السياق والموثوقية	**15%**

سيناريوهات الاختبار

السيناريو 1: وكيل خدمة العملاء

تبني وكيلًا للتعامل مع عمليات الإرجاع، ونزاعات الفوترة، ومشكلات الحساب باستخدام Claude Agent SDK. يستخدم الوكيل أدوات MCP مثل (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`). الهدف هو تحقيق أكثر من 80% من حل المشكلات من أول تواصل، مع التصعيد المناسب.

السيناريو 2: توليد الكود باستخدام Claude Code

لتسريع التطوير: توليد الكود، وإعادة الهيكلة، والتصحيح، والتوثيق. تحتاج إلى دمجه مع أوامر Claude Code تستخدم slash وإعدادات CLAUDE.md مخصصة، وفهم متى تستخدم وضع التخطيط.

السيناريو 3: نظام بحث متعدد الوكلاء

يفوض وكيل منسّق المهام إلى وكلاء فرعيين متخصصين: بحث الويب، وتحليل المستندات، والتركيب، وتوليد التقارير. يجب أن ينتج النظام تقارير كاملة مع الاستشهادات.

السيناريو 4: أدوات إنتاجية المطورين

يساعد الوكيل المهندسين على استكشاف قواعد كود غير مألوفة، وتوليد كود نمطي، وأتمتة المهام الروتينية. تُستخدم الأدوات المدمجة (Read, Write, Bash, Grep, Glob) وخوادم MCP.

السيناريو 5: Claude Code للتكامل المستمر

دمج Claude Code في خط CI/CD. لمراجعات الكود الآلية، وتوليد الاختبارات، وتقديم ملاحظات على طلبات السحب. يجب تصميم المطالبات لتقليل الإيجابيات الكاذبة.

السيناريو 6: استخراج البيانات المنظمة

يستخرج النظام معلومات من مستندات غير منظمة، ويتحقق من المخرجات باستخدام مخططات JSON، ويحافظ على دقة عالية. يجب أن يتعامل بشكل صحيح مع الحالات الحدية.

السيناريو 7: أنماط معمارية الذكاء الاصطناعي الحواري

تصمم أنظمة حوار متعددة الأدوار تشمل إدارة نافذة السياق، واستمرار التعليمات عبر الأدوار، واستراتيجيات الذاكرة، وتصميم أدوات للتنفيذ الآمن، والتعامل مع مدخلات المستخدم الغامضة أو المتعارضة.

السيناريو 8: أدوات الذكاء الاصطناعي الوكيل *(المحتوى ناقص — ساعدونا على إكماله!)*

تم الإبلاغ عن هذا السيناريو من بعض المتقدمين للاختبار، لكنه غير مغطى بعد في هذا الدليل. إذا واجهت أسئلة من هذا السيناريو في الاختبار الحقيقي، يرجى مشاركتها في GitHub Issues حتى نضيف تغطية كاملة. ستساعد مساهمتك كل من يستعد للاختبار.

التوثيق الرسمي

المورد	الرابط
Claude API — Messages	https://platform.claude.com/docs/en/api/messages
Claude API — Tool Use	https://platform.claude.com/docs/en/build-with-claude/tool-use
Claude API — Message Batches	https://platform.claude.com/docs/en/build-with-claude/message-batches
Claude Agent SDK — Overview	https://platform.claude.com/docs/en/agent-sdk/overview
Claude Agent SDK — Hooks	https://platform.claude.com/docs/en/agent-sdk/hooks
Claude Agent SDK — Subagents	https://platform.claude.com/docs/en/agent-sdk/subagents
Claude Agent SDK — Sessions	https://platform.claude.com/docs/en/agent-sdk/sessions
Model Context Protocol (MCP)	https://modelcontextprotocol.io/
MCP — Tools	https://modelcontextprotocol.io/docs/concepts/tools
MCP — Resources	https://modelcontextprotocol.io/docs/concepts/resources
MCP — Servers	https://modelcontextprotocol.io/docs/concepts/servers
Claude Code — Documentation	https://code.claude.com/docs/en/overview
Claude Code — CLAUDE.md and Memory	https://code.claude.com/docs/en/memory
Claude Code — Skills (incl. slash commands)	https://code.claude.com/docs/en/skills
Claude Code — Hooks	https://code.claude.com/docs/en/hooks
Claude Code — Sub-agents	https://code.claude.com/docs/en/sub-agents
Claude Code — MCP Integration	https://code.claude.com/docs/en/mcp
Claude Code — GitHub Actions CI/CD	https://code.claude.com/docs/en/github-actions

Claude Code — GitLab CI/CD	https://code.claude.com/docs/en/gitlab-ci-cd
Claude Code — Headless (non-interactive mode)	https://code.claude.com/docs/en/headless
Prompt Engineering Guide	https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview
Extended Thinking	https://platform.claude.com/docs/en/build-with-claude/extended-thinking
Anthropic Cookbook (code examples)	https://github.com/anthropics/antropic-cookbook

---

الجزء الأول: الأسس النظرية

يغطي هذا الجزء كل النظرية التي تحتاجها لاجتياز الاختبار بنجاح. تم تنظيم المادة حسب التقنيات والمفاهيم بدلًا من مجالات الاختبار، لأن ذلك يساعدك على بناء فهم أعمق لكل موضوع.

الفصل 1: Claude API — أساسيات التفاعل مع النموذج

> التوثيق: Messages API | Prompt Engineering

1.1 بنية طلب API

يتبع Claude API نموذج طلب–استجابة. يتضمن كل طلب إلى Claude Messages API ما يلي:

```
{
  "model": "claude-sonnet-4-6",
  "max_tokens": 1024,
  "system": "You are a helpful assistant.",
  "messages": [
    {"role": "user", "content": "Hi!"},
    {"role": "assistant", "content": "Hello!"},
    {"role": "user", "content": "How are you?"}
  ],
  "tools": [...],
  "tool_choice": {"type": "auto"}
}
```

الحقول الأساسية:

- `model` — اختيار النموذج (`claude-opus-4-6`, `claude-sonnet-4-6`, `claude-haiku-4-5`)
- `max_tokens` — الحد الأقصى لعدد الرموز في الاستجابة
- `system` — مطالبة النظام (تحدد سلوك النموذج)

- `messages` — سجل المحادثة (**يجب إرسال السجل الكامل** للحفاظ على الاتساق)
- `tools` — تعريفات الأدوات المتاحة
- `tool_choice` — استراتيجية اختيار الأداة

1.2 أدوار الرسائل

تستخدم مصفوفة `messages` دورين حواريين بالإضافة إلى دور توجيهي واحد:

- `user` — رسائل المستخدم، بما في ذلك نتائج الأدوات (تُرسل ككتلة محتوى `tool_result` ضمن رسالة بدور `user`، وليست دورًا منفصلاً باسم `tool`)
- `assistant` — ردود النموذج (تُدرج عند إرسال السجل)، بما في ذلك طلبات استخدام الأدوات (كتل محتوى `tool_use`)
- `system` — يمكن تعيينه عبر الحقل العلوي `system` أو ضمن `messages` (يسري من الدور الأول) كـ `{"role": "system", ...}` (يسري من تلك النقطة فصاعدًا، وفق قواعد وضع محددة — انظر أدناه)

نتائج الأدوات لا تُرسل كرسالة بدور `"tool"`. بل تُرسل كرسالة بدور `user` يتضمن محتواها كتلة محتوى `tool_result`:

```
{
  "role": "user",
  "content": [
    {
      "type": "tool_result",
      "tool_use_id": "toolu_01...",
      "content": "..."
    }
  ]
}
```

يمكن لـ `system` العلوي. وليس فقط عبر معامل `system`، أن يظهر أيضًا كدور مباشرة ضمن مصفوفة `messages`. الهدف من ذلك هو إضافة تعليمات في منتصف المحادثة دون إبطال البادئة المخزنة مؤقتًا (cached prefix) من الحقل `system` العلوي. ولهذا الأسلوب قواعد وضع محددة:

- يجب أن يأتي مباشرة بعد دور `user` (بما في ذلك دور يحتوي على كتل `tool_result`) أو دور `assistant` ينتهي باستخدام أداة من جهة الخادم (server tool use).
- يجب أن يسبق دور `assistant` أو أن يكون آخر عنصر في المصفوفة.
- لا يمكن أن يقع بين كتلة `tool_use` ونتيجتها `tool_result` — فعل ذلك يُرجع خطأ 400.
- رسائل `system` اللاحقة (بما في ذلك تلك التي تظهر في منتصف المحادثة) لها الأولوية على الرسائل السابقة وعلى الحقل العلوي `system` بالنسبة للأدوار التالية.

مهم جدًا: في كل طلب API يجب إرسال **سجل المحادثة الكامل**. لا يحتفظ النموذج بالحالة بين الطلبات، فكل استدعاء مستقل.

1.3 حقل `stop_reason` في الاستجابة

تتضمن استجابة Claude API الحقل `stop_reason`، وهو يوضح سبب توقف النموذج عن التوليد:

القيمة	الوصف	الإجراء
`"end_turn"`	أنهى النموذج استجابته	اعرض النتيجة للمستخدم
`"tool_use"`	يريد النموذج استدعاء أداة	نفذ الأداة وأعد النتيجة
`"max_tokens"`	تم الوصول إلى حد الرموز	الاستجابة مبتورة؛ قد تحتاج إلى زيادة الحد
`"stop_sequence"`	تمت مصادفة تسلسل إيقاف	عالجه حسب منطق تطبيقك

لأنهما تتحكمان في حلقة الوكيل. `"end_turn"` و `"tool_use"` في الأنظمة الوكيلة، أهم قيمتين هما

1.4 مطالبة النظام

مطالبة النظام هي تعليمات خاصة تحدد السياق والقواعد السلوكية. وهي:

- `system` ؛ تُمرر منفصلة في الحقل `messages` ليست جزءًا من مصفوفة
- لها أولوية على رسائل المستخدم
- تُحمّل مرة واحدة وتُطبق طوال المحادثة
- تُستخدم لتعريف الدور والقيود وصيغة الإخراج

مهم للاختبار: صياغة مطالبة النظام قد تنشئ ارتباطات غير مقصودة بالأدوات. مثلًا، تعليمة مثل "تحقق دائمًا من العميل" قد تجعل النموذج يفرط في استخدام `get_customer` حتى عندما لا يكون ذلك ضروريًا.

1.5 نافذة السياق

نافذة السياق هي إجمالي النص (بالرموز) الذي يستطيع النموذج معالجته دفعة واحدة. وتشمل:

- مطالبة النظام
- سجل الرسائل الكامل
- تعريفات الأدوات
- نتائج الأدوات

مشكلات نافذة السياق الأساسية:

1. **تأثير ضياع المعلومات في الوسط:** تعالج النماذج المعلومات في بداية المدخلات ونهايتها بموثوقية أكبر، وقد تفوت تفاصيل في الوسط. المعالجة: ضع المعلومات المهمة قرب البداية أو النهاية.
2. **تراكم نتائج الأدوات:** يضيف كل استدعاء أداة مخرجات إلى السياق. إذا كانت الأداة ترجع أكثر من 40 حقلًا ولا يهم منها إلا 5، فمعظم السياق يُهدر.
3. **التلخيص التدريجي:** عند ضغط السجل، غالبًا ما تُفقد القيم الرقمية والنسب والتواريخ وتتحول إلى تعبيرات مبهمة مثل "تقريبًا" و"نحو" و"بعض".

الفصل 2: الأدوات و `tool_use`

> التوثيق: Tool Use

2.1 ما هو `tool_use`

`tool_use` آلية تسمح لـ Claude باستدعاء وظائف خارجية. النموذج لا يشغل الكود مباشرة، بل يولد طلب استدعاء منظمًا؛ ثم ينفذ كودك ذلك الطلب ويرجع النتيجة أداة.

2.2 تعريف الأداة

تُعرّف كل أداة باستخدام مخطط JSON:

```
{
  "name": "get_customer",
  "description": "Finds a customer by email or ID. Returns the customer profile,
including name, email, order history, and account status. Use this tool BEFORE
lookup_order to verify the customer's identity. Accepts an email (format:
user@domain.com) or a numeric customer_id.",
  "input_schema": {
    "type": "object",
    "properties": {
      "email": {"type": "string", "description": "Customer email"},
      "customer_id": {"type": "integer", "description": "Numeric customer ID"}
    },
    "required": []
  }
}
```

نقاط مهمة جدًا في وصف الأداة:

1. **الوصف هو آلية الاختيار الأساسية.** يختار نموذج اللغة الأدوات بناءً على أوصافها. الأوصاف المختصرة مثل "يجلب معلومات العميل" تؤدي إلى أخطاء عند تداخل الأدوات.
2. **ضمّن في الوصف:**
   - ما الذي تفعله الأداة وما الذي ترجعه
   - صيغ الإدخال وقيم أمثلة
   - الحالات الحدية والقيود
   - متى تستخدم هذه الأداة مقارنة بالبدائل المشابهة
3. يملكان `analyze_document` و `analyze_content` أوصافًا شبه متطابقة، فسوف يخلط النموذج بينهما. إذا كان **تجنب** الأوصاف المتطابقة أو المتداخلة بين الأدوات.
4. **الأدوات المدمجة مقابل أدوات MCP:** قد يفضل الوكلاء الأدوات المدمجة (Read, Grep) على أدوات MCP ذات الوظائف المشابهة. لمنع ذلك، قوِّ أوصاف أدوات MCP، وابرز المزايا المحددة أو البيانات الفريدة أو السياق الذي لا تستطيع الأدوات المدمجة توفيره.

2.3 معامل `tool_choice`

يتحكم `tool_choice` في كيفية اختيار النموذج للأدوات:

متى تستخدمه	السلوك	القيمة

`{"type": "auto"}`	يقرر النموذج هل يستدعي أداة أم يجيب نصيًا	الافتراضي في معظم الحالات
`{"type": "any"}`	يجب على النموذج استدعاء أداة ما	عندما تحتاج إلى إخراج منظم مضمون
`{"type": "tool", "name": "extract_metadata"}`	يجب على النموذج استدعاء أداة محددة	عندما تحتاج إلى خطوة أولى/ترتيب تنفيذ مضمون

سيناريوهات مهمة:

- أدوات استخراج متعددة → يختار النموذج أفضل أداة، لكنك تحصل على إخراج منظم + `tool_choice: "any"` على أي حال
- عندما يجب ضمان إجراء أول محدد → الاختيار الإجباري (مثل `extract_metadata` قبل الإثراء)

2.4 مخططات JSON للمخرجات المنظمة

استخدام `tool_use` مع مخططات JSON للحصول على إخراج منظم من Claude. فهو **أوثق طريقة** هو:

- يضمن JSON صحيحًا نحويًا (لا أقواس ناقصة ولا فواصل زائدة)
- يفرض البنية المطلوبة (الحقول المطلوبة موجودة)
- لا يضمن الصحة الدلالية (قد تكون القيم خاطئة)

تصميم المخطط — مبادئ أساسية:

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["bug", "feature", "docs", "unclear", "other"]
    },
    "category_detail": {
      "type": ["string", "null"],
      "description": "Details if category = 'other' or 'unclear'"
    },
    "severity": {
      "type": "string",
      "enum": ["critical", "high", "medium", "low"]
    },
    "confidence": {
      "type": "number",
      "minimum": 0,
      "maximum": 1
    },
    "optional_field": {
      "type": ["string", "null"],
      "description": "Null if the information was not found in the source"
    }
  },
  "required": ["category", "severity"]
}
```

قواعد تصميم المخطط:

1. **الحقول المطلوبة مقابل الاختيارية:** اجعل الحقل مطلوبًا فقط إذا كانت المعلومة متاحة دائمًا. الحقول المطلوبة تدفع النموذج إلى اختلاق قيم عندما تكون البيانات مفقودة.
2. **الحقول القابلة للقيمة null:** استخدم `"type": ["string", "null"]` للمعلومات التي قد تكون غائبة. يستطيع النموذج إرجاع `null` بدلًا من الهلوسة.
3. **Enums مع `"other"`:** أضف `"other"` مع حقل تفاصيل لتجنب فقدان البيانات خارج الفئات المحددة مسبقًا.
4. **Enum `"unclear"`:** `"unclear"` للحالات التي لا يستطيع فيها النموذج اختيار فئة بثقة؛ الصادقة أفضل من فئة خاطئة.

2.5 الأخطاء النحوية مقابل الدلالية

نوع الخطأ	مثال	التخفيف
نحوي	JSON غير صالح، نوع حقل خاطئ	JSON (يزيله) مع مخطط `tool_use`
دلالي	الإجماليات لا تتطابق، قيمة في حقل خاطئ، هلوسة	فحوصات تحقق، إعادة المحاولة مع تغذية راجعة، التصحيح الذاتي

الفصل 3: Claude Agent SDK — بناء الأنظمة الوكيلة

> التوثيق: Agent SDK | Hooks | Subagents | Sessions

3.1 ما هي حلقة الوكيل

حلقة الوكيل هي النمط الأساسي لتنفيذ المهام ذاتيًا. النموذج لا يجيب فقط، بل ينفذ سلسلة من الإجراءات:

```
1. Send a request to Claude with tools
2. Receive a response
3. Check stop_reason:
   - "tool_use" -> execute the tool, append the result to history, go back to step 1
   - "end_turn" -> the task is complete, show the result to the user
4. Repeat until completion
```

هذا نهج يقوده النموذج: يقرر Claude الأداة التالية بناءً على السياق ونتائج الأدوات السابقة. وهذا يختلف عن أشجار القرار المبرمجة مسبقًا التي يكون فيها تسلسل الإجراءات ثابتًا.

أنماط مضادة (تجنبها):

- تحليل نص المساعد لاكتشاف الاكتمال ("Task completed")
- كإشارة إيقاف أساسية (مثل استخدام حد تكرار اعتباطي `max_iterations=5`)
- التحقق مما إذا كان المساعد أنتج نصًا كإشارة اكتمال

النهج الصحيح: إشارة الاكتمال الموثوقة الوحيدة هي `stop_reason == "end_turn"`.

3.2 إعداد `AgentDefinition`

`AgentDefinition` هو كائن إعداد الوكيل في Claude Agent SDK:

```
agent = AgentDefinition(
    name="customer_support",
    description="Handles customer requests for returns and order issues",
    system_prompt="You are a customer support agent...",
    allowed_tools=["get_customer", "lookup_order", "process_refund",
"escalate_to_human"],
    # For a coordinator:
    # allowed_tools=["Task", "get_customer", ...]
)
```

المعاملات الأساسية:

- `name` / `description` — تعريف الوكيل ووصفه
- `system_prompt` — مطالبة النظام مع التعليمات
- `allowed_tools` — قائمة الأدوات المسموحة (مبدأ أقل صلاحية)

3.3 نموذج المحور والأذرع: المنسق والوكلاء الفرعيون

عادة تُبنى البنية متعددة الوكلاء كطوبولوجيا محور وأذرع:

```
        Coordinator
       /     |      \
  Subagent1  Subagent2  Subagent3
   (search)   (analysis)   (synthesis)
```

مسؤوليات المنسق:

- تفكيك المهمة إلى مهام فرعية
- تحديد الوكلاء الفرعيين المطلوبين (اختيار ديناميكي)
- تفويض العمل للوكلاء الفرعيين
- تجميع النتائج والتحقق منها
- معالجة الأخطاء وإعادة المحاولة
- إيصال النتائج إلى المستخدم

مبدأ حرج: الوكلاء الفرعيون لديهم سياق معزول.

- لا يرث الوكلاء الفرعيون تلقائيًا سجل محادثة المنسق
- يجب تمرير كل السياق المطلوب صراحةً في مطالبة الوكيل الفرعي
- لا يتشارك الوكلاء الفرعيون الذاكرة بين الاستدعاءات
- تمر كل الاتصالات عبر المنسق (للملاحظة والتحكم في الأخطاء)

3.4 أداة `Task` لإنشاء الوكلاء الفرعيين

يتم إنشاء الوكلاء الفرعيين عبر أداة `Task`:

```python
# The coordinator's allowedTools must include "Task"
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

تمرير السياق الصريح إلزامي:

```
# Bad: the subagent has no context
Task: "Analyze the document"

# Good: full context in the prompt
Task: "Analyze the following document.
Document: [full document text]
Prior search results: [web search results]
Output format requirements: [schema]"
```

الإنشاء المتوازي: يستطيع المنسق استدعاء عدة `Task` بالتوازي في استجابة واحدة، فتعمل الوكلاء الفرعيون بالتوازي:

```
# One coordinator response contains:
Task 1: "Search for articles about X"
Task 2: "Analyze document Y"
Task 3: "Search for articles about Z"
# All three run concurrently
```

3.5 الخطافات في Agent SDK

تسمح الخطافات بالاعتراض والتحويل عند نقاط محددة في دورة حياة الوكيل.

PostToolUse يعترض نتيجة الأداة قبل تزويد النموذج بها:

```python
# Example: normalize date formats from different MCP tools
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Convert Unix timestamp -> ISO 8601
    # Convert "Mar 5, 2025" -> "2025-03-05"
    return normalized_result
```

خطاف اعتراض الاستدعاء الصادر يمنع الإجراءات التي تنتهك السياسة:

```python
# Example: block refunds above $500
@hook("PreToolUse")
def enforce_refund_limit(tool_call):
    if tool_call.name == "process_refund" and tool_call.args.amount > 500:
        return redirect_to_escalation(tool_call)
```

الفرق الأساسي: الخطافات مقابل تعليمات المطالبة

تعليمات المطالبة	الخطافات	الخاصية
احتمالي (<90%، وليس 100%)	**حتمي** (100%)	الضمان
التفضيلات العامة، التوصيات، التنسيق	قواعد الأعمال الحرجة، العمليات المالية، الامتثال	متى تُستخدم
“Try to solve before escalating”	منع المبالغ المستردة فوق 500 دولار	مثال

القاعدة: عندما يؤدي الفشل إلى عواقب مالية أو قانونية أو تتعلق بالسلامة، استخدم الخطافات لا المطالبات.

الفصل 4: Model Context Protocol (MCP)

> التوثيق: MCP | Tools | Resources | Servers

4.1 ما هو MCP

Model Context Protocol (MCP) هو بروتوكول مفتوح لربط الأنظمة الخارجية بـ Claude. يعرّف MCP ثلاثة أنواع أساسية من الموارد:

1. **Tools** — وظائف يستطيع الوكيل استدعاءها لتنفيذ إجراءات (تنفيذ أوامر، استدعاءات API، عمليات CRUD)
2. **Resources** — بيانات يستطيع الوكيل قراءتها كسياق (توثيق، مخططات قواعد بيانات، فهارس محتوى)
3. **Prompts** — قوالب مطالبات معدة مسبقًا للمهام الشائعة

4.2 خوادم MCP

خادم MCP هو عملية تنفذ بروتوكول MCP وتوفر أدوات/موارد. عند الاتصال بخادم MCP:

- يتم اكتشاف كل الأدوات تلقائيًا
- تكون أدوات كل الخوادم المتصلة متاحة في الوقت نفسه
- تحدد أوصاف الأدوات كيفية استخدام النموذج لها

4.3 إعداد خوادم MCP

إعداد المشروع (`.mcp.json`) — لاستخدام الفريق:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "jira": {
      "command": "npx",
      "args": ["-y", "mcp-server-jira"],
      "env": {
        "JIRA_TOKEN": "${JIRA_TOKEN}"
      }
    }
  }
}
```

نقاط أساسية:

- يُخزن `.mcp.json` في جذر المشروع ويُدار ضمن نظام التحكم بالإصدارات
- تُستخدم متغيرات البيئة (`${GITHUB_TOKEN}`) للأسرار؛ لا تُحفظ الرموز نفسها في المستودع
- يكون متاحًا لكل المساهمين في المشروع

إعداد المستخدم (`~/.claude.json`) — للخوادم الشخصية/التجريبية:

- يُخزن في مجلد المنزل للمستخدم

- لا يُشارك عبر نظام التحكم بالإصدارات
- مناسب للتجارب الشخصية والاختبار

اختيار الخوادم:

- فضّل خوادم MCP المجتمعية الحالية للتكاملات القياسية (Jira، GitHub، Slack)
- ابنِ خوادمك الخاصة فقط لسير العمل الفريدة الخاصة بالفريق

4.4 راية `isError` في MCP

عندما تواجه أداة MCP خطأً، تستخدم `isError: true` في الاستجابة. هذا يشير للوكيل إلى أن الاستدعاء فشل.

خطأ منظم (جيد):

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "The service is temporarily unavailable. Timeout while calling the
orders API.",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

خطأ عام (نمط مضاد):

```
{
  "isError": true,
  "content": "Operation failed"
}
```

الخطأ العام لا يعطي الوكيل أي معلومات لاتخاذ القرار: هل يعيد المحاولة، أم يغير الاستعلام، أم يصعّد؟

4.5 موارد MCP

الموارد هي بيانات يستطيع الوكيل طلبها للحصول على سياق دون تنفيذ إجراءات:

- فهارس المحتوى (مثل قائمة بكل مهام المشروع أو تنقل هرمي)
- مخططات قواعد البيانات (لفهم بنية البيانات)
- التوثيق (مراجع API، أدلة داخلية)
- ملخصات القضايا/المهام

ميزة الموارد: لا يحتاج الوكيل إلى استدعاءات أدوات استكشافية لفهم البيانات الموجودة. يقدّم المورد "خريطة" فورية.

الفصل 5: Claude Code — الإعداد وسير العمل

> التوثيق: Claude Code | Memory / CLAUDE.md | Skills | MCP | Hooks | Sub-agents | GitHub Actions | Headless

5.1 هرمية CLAUDE.md

توجد هرمية من ثلاثة مستويات. Claude Code هو ملف أو ملفات التعليمات الخاصة بـ CLAUDE.md:

```
1. User-level: ~/.claude/CLAUDE.md
   - Applies only to that user
   - NOT shared via VCS
   - Personal preferences and working style

2. Project-level: .claude/CLAUDE.md or a root CLAUDE.md
   - Applies to all project contributors
   - Managed via VCS
   - Coding standards, testing standards, architectural decisions

3. Directory-level: CLAUDE.md in subdirectories
   - Applies when working with files in that directory
   - Conventions specific to that part of the codebase
```

خطأ شائع: لا يحصل عضو فريق جديد على تعليمات المشروع لأنها وُضعت في `~/.claude/CLAUDE.md` (مستوى المستخدم) بدلًا من `.claude/CLAUDE.md` (مستوى المشروع).

5.2 صيغة `@path` (استيراد الملفات)

يمكن أن يشير CLAUDE.md إلى ملفات خارجية باستخدام `@path`، مما يجعل الإعداد معياريًا:

```
# Project CLAUDE.md

Coding standards are described in @./standards/coding-style.md
Test requirements are in @./standards/testing-requirements.md
Project overview is in @README.md and dependencies are in @package.json
```

قواعد `@path`:

- استخدم `@` مباشرة قبل مسار الملف (دون مسافة)
- المسارات النسبية والمطلقة مدعومة
- تُحل المسارات النسبية بالنسبة للملف الذي يحتوي على الاستيراد
- أقصى عمق لتداخل الاستيراد هو 5

يساعد ذلك على تجنب التكرار ويتيح لكل حزمة تضمين المعايير ذات الصلة فقط.

5.3 مجلد `.claude/rules/`

`.claude/rules/` بديل عن ملف CLAUDE.md ضخم، ويستخدم لتنظيم القواعد حسب الموضوع:

```
.claude/rules/
  testing.md          -- testing conventions
  api-conventions.md  -- API conventions
  deployment.md       -- deployment rules
  react-patterns.md   -- React patterns
```

الميزة الأساسية: YAML frontmatter مع `paths` للتحميل الشرطي:

```
---
paths: ["src/api/**/*"]
---

For API files, use async/await with explicit error handling.
Each endpoint must return a standard response wrapper.
```

```
---
paths: ["**/*.test.tsx", "**/*.test.ts"]
---

Tests must use describe/it blocks.
Use data factories instead of hardcoding.
Do not mock the database—use a test database.
```

كيف يعمل:

- تُحمّل القاعدة **فقط** عندما يحرر Claude Code ملفًا يطابق نمط `paths`
- هذا يوفر السياق والرموز لأن القواعد غير ذات الصلة لا تُحمّل
- تطبيق الاصطلاحات حسب نوع الملف بغض النظر عن الموقع (مثالي للاختبارات المنتشرة في قاعدة الكود) تتيح أنماط Glob

متى تستخدم `.claude/rules/` مع `paths` مقابل CLAUDE.md على مستوى المجلد:

- عندما تنطبق الاصطلاحات على ملفات منتشرة عبر مجلدات كثيرة — `.claude/rules/` مع `paths` (الاختبارات، migrations)
- على مستوى المجلد — عندما تكون الاصطلاحات مرتبطة بمجلد محدد ولا تُحتاج في غيره CLAUDE.md

5.4 أوامر Slash والمهارات المخصصة

> **ملاحظة:** في الإصدار الحالي من Claude Code، تم توحيد الأوامر المخصصة (`.claude/commands/`) مع المهارات (`.claude/skills/`). كلا التنسيقين ينشئ أوامر `/name`. يشير دليل الاختبار إلى `.claude/commands/`، وهذا التنسيق لا يزال مدعومًا.

أوامر Slash هي قوالب مطالبات قابلة لإعادة الاستخدام وتُستدعى عبر `/name`:

تنسيق `.claude/commands/` (قديم لكنه مدعوم):

```
.claude/commands/
  review.md        -- /review -- standard code review
  test-gen.md      -- /test-gen -- test generation
```

تنسيق `.claude/skills/` (الحالي):

```
.claude/skills/
  review/SKILL.md  -- /review -- with frontmatter configuration
  test-gen/SKILL.md
```

أوامر المشروع (`.claude/commands/` أو `.claude/skills/`):

- تُخزن في VCS وتتاح للجميع عند استنساخ الريبو
- تضمن سير عمل متسقًا عبر الفريق

أوامر المستخدم (`~/.claude/commands/` أو `~/.claude/skills/`):

- أوامر شخصية لا تُشارك عبر VCS
- لسير العمل الفردي

5.5 المهارات — `.claude/skills/`

المهارات أوامر متقدمة تُعد عبر frontmatter في SKILL.md:

```yaml
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "Path to the directory to analyze"
---

Analyze the code structure in the specified directory.
Output a report on dependencies and architectural patterns.
```

معاملات frontmatter:

المعامل	الوصف
`context: fork`	يشغّل المهارة في وكيل فرعي معزول. لا يلوث الإخراج المطول الجلسة الرئيسية
`allowed-tools`	يقيّد الأدوات المتاحة (أمان؛ مثلًا لا تستطيع المهارة حذف ملفات إذا لم يُسمح لها)
`argument-hint`	تلميح يطلب معاملًا عند الاستدعاء دون معاملات

متى تستخدم مهارة مقابل CLAUDE.md:

- **المهارة** — استدعاء عند الحاجة لمهمة محددة (مراجعة، تحليل، توليد)
- **CLAUDE.md** — معايير واصطلاحات عامة تُحمّل دائمًا

المهارات الشخصية (`~/.claude/skills/`):

- أنشئ نسخًا شخصية بأسماء مختلفة حتى لا تؤثر على زملائك

5.6 وضع التخطيط مقابل التنفيذ المباشر

وضع التخطيط:

- يحقق النموذج ويخطط فقط؛ لا يجري تغييرات
- يستخدم Read وGrep وGlob لاستكشاف قاعدة الكود
- ينتج خطة تنفيذ يوافق عليها المستخدم
- استكشاف آمن دون آثار جانبية

متى تستخدم وضع التخطيط:

- تغييرات كبيرة (عشرات الملفات)
- وجود عدة مقاربات محتملة (microservices: كيف نحدد الحدود؟)
- قرارات معمارية (أي إطار؟ أي بنية؟)
- قاعدة كود غير مألوفة (يجب الفهم قبل التغيير)
- ترحيلات مكتبات تؤثر على أكثر من 45 ملفًا

متى تستخدم التنفيذ المباشر:

- إصلاحات في ملف واحد مع stack trace واضح
- إضافة تحقق واحد
- تغييرات مفهومة وواضحة وغير ملتبسة

نهج مدمج:

1. وضع التخطيط للتحقيق والتصميم
2. يوافق المستخدم على الخطة
3. التنفيذ المباشر لتطبيق الخطة المعتمدة

Explore subagent — وكيل فرعي متخصص لاستكشاف قاعدة الكود:

- يعزل الإخراج المطول عن السياق الرئيسي
- يرجع ملخصًا فقط
- يمنع استنزاف نافذة السياق في المهام متعددة المراحل

5.7 الأمر `/compact`

`/compact` أمر مدمج لضغط السياق:

- يلخص السجل السابق لتحرير نافذة السياق
- يُستخدم في جلسات التحقيق الطويلة عندما يمتلئ السياق بإخراج أدوات مطول
- الخطر: قد تُفقد القيم الرقمية والتواريخ والتفاصيل الدقيقة أثناء التلخيص

5.8 الأمر `/memory`

`/memory` أمر مدمج لإدارة الذاكرة بين الجلسات:

- يفتح ملف `CLAUDE.md` للتحرير، مما يسمح بحفظ الملاحظات والتفضيلات والسياق
- تستمر المعلومات بين الجلسات وتُحمّل تلقائيًا عند البدء
- مفيد لتخزين اصطلاحات المشروع، وتفضيلات المستخدم، والأوامر المتكررة، وسياق العمل الحالي
- بديل عن إعادة شرح التعليمات نفسها في كل جلسة

5.9 واجهة Claude Code CLI لـ CI/CD

العلم `-p` (أو `--print`):

```
claude -p "Analyze this pull request for security issues"
```

- وضع غير تفاعلي: يعالج المطالبة، يطبع إلى stdout، ثم يخرج
- لا ينتظر إدخال المستخدم
- الطريقة الصحيحة الوحيدة لتشغيل Claude في خطوط CI/CD

إخراج منظم لـ CI:

```
claude -p "Review this PR" --output-format json --json-schema
'{"type":"object",...}'
```

- `--output-format json` — الإخراج بصيغة JSON
- `--json-schema` — التحقق من الإخراج مقابل مخطط
- يمكن تحليل النتيجة لنشر تعليقات PR داخلية تلقائيًا

عزل سياق الجلسة: الجلسة نفسها التي ولدت الكود غالبًا تكون أقل فعالية في مراجعته، لأن النموذج يحتفظ بسياق تفكيره ويكون أقل ميلًا لتحدي قراراته. استخدم مثيلًا مستقلًا للمراجعة.

منع التعليقات المكررة: عند إعادة المراجعة بعد commits جديدة، ضمّن نتائج المراجعة السابقة في السياق واطلب من Claude الإبلاغ فقط عن المشكلات الجديدة أو غير المحلولة.

5.10 `fork_session` وإدارة الجلسات

`--resume <session-name>` يستأنف جلسة مسماة:

```
claude --resume investigation-auth-bug
```

- يواصل محادثة سابقة مع سياق محفوظ
- مفيد للتحقيقات الطويلة عبر عدة جلسات
- الخطر: إذا تغيرت الملفات منذ الجلسة السابقة، قد تكون نتائج الأدوات قديمة

`fork_session` ينشئ فرعًا مستقلًا من سياق مشترك:

```
Codebase investigation
         |
    fork_session
    /          \
Approach A:      Approach B:
Redux            Context API
```

- يرث الفرعان السياق حتى نقطة التفرع
- بعد ذلك يتباعدان بشكل مستقل
- مفيد لمقارنة المقاربات أو اختبار الاستراتيجيات

متى تبدأ جلسة جديدة بدلًا من الاستئناف:

- نتائج الأدوات قديمة (تغيرت الملفات)
- مر وقت طويل وتدهور السياق
- بدلًا من الاستئناف بسياق أدوات قديم "Here is a short summary of what we found: ..." الأفضل أن تبدأ بـ

الفصل 6: هندسة المطالبات — تقنيات متقدمة

> التوثيق: Prompt Engineering | Anthropic Cookbook

6.1 المطالبة بأمثلة قليلة Few-shot Prompting

Few-shot prompting هو تضمين 2–4 أمثلة إدخال/إخراج في المطالبة لتوضيح السلوك المتوقع.

أكثر فعالية من الوصف النصي few-shot لماذا تكون أمثلة:

- تعليمة مبهمة مثل "كن أكثر دقة" يمكن تفسيرها بطرق كثيرة
- المثال يوضح بلا لبس الصيغة ومنطق القرار المتوقعين
- يعمم النموذج النمط على حالات جديدة (لا يكرر الأمثلة فقط)

أنواع أمثلة few-shot ومتى تستخدمها:

1. **أمثلة للسيناريوهات الغامضة:**

```
Request: "My order is broken"
Action: Call get_customer -> lookup_order -> check status.
Rationale: "broken" may mean a damaged item; you need order details.

Request: "Get me a manager"
Action: Immediately call escalate_to_human.
Rationale: The customer explicitly requests a human. Do not attempt to solve
autonomously.
```

2. **أمثلة لتنسيق الإخراج:**

```
Finding example:
{
  "location": "src/auth/login.ts:42",
  "issue": "SQL injection in the username parameter",
  "severity": "critical",
  "suggested_fix": "Use a parameterized query"
}
```

3. **أمثلة للتمييز بين الكود المقبول والمشكلة:**

```
// Acceptable (do not flag):
const items = data.filter(x => x.active);

// Problem (flag):
const items = data.filter(x => x.active == true); // Use strict equality ===
```

4. **أمثلة للاستخراج من صيغ مستندات مختلفة:**

```
Document with inline citations:
"As shown in the study (Smith, 2023), the rate is 42%."
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

Document with bibliography references:
"The rate is 42%. [1]"
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

5. **أمثلة للقياسات غير الرسمية:**

```
Text: "about two handfuls of rice"
-> {"amount": "~100g", "original_text": "two handfuls", "precision": "approximate"}

Text: "a pinch of salt"
-> {"amount": "~1g", "original_text": "a pinch", "precision": "approximate"}
```

تكون few-shot فعالة خصوصًا لاستخراج وحدات القياس غير الرسمية وغير القياسية التي يصعب تغطيتها بتعليمات قاعدية فقط.

صارمة للمخرجات المنظمة، أضف قواعد تطبيع JSON **قواعد تطبيع الصيغة في المطالبات:** عند استخدام مخططات في المطالبة:

```
Normalization:
- Dates: always ISO 8601 (YYYY-MM-DD); "yesterday" -> compute an absolute date
- Currency: numeric amount + currency code; "five bucks" -> {"amount": 5, "currency":
"USD"}
- Percentages: decimal fraction; "half" -> 0.5
```

هذا يمنع الأخطاء الدلالية حيث يكون JSON صحيحًا نحويًا لكن القيم غير متسقة.

6.2 معايير صريحة مقابل تعليمات مبهمة

سيئ (مبهم):

```
Check code comments for accuracy.
Be conservative—report only high-confidence findings.
```

جيد (معايير صريحة):

```
Flag a comment as problematic ONLY if:
1. The comment describes behavior that CONTRADICTS the actual code behavior
2. The comment references a non-existent function or variable
3. A TODO/FIXME comment refers to a bug that has already been fixed in code

Do NOT flag:
- Comments that are merely stylistically outdated
- Comments with minor wording inaccuracies
- Missing comments (that is a separate category)
```

عرّف معايير الشدة مع أمثلة:

```
CRITICAL: Runtime failure for users
  Example: NullPointerException while processing a payment

HIGH: Security vulnerability
  Example: SQL injection, XSS, missing authorization checks

MEDIUM: Logic bug without immediate impact
  Example: Wrong sorting, off-by-one error

LOW: Code quality
  Example: Duplication, suboptimal algorithm for small data
```

6.3 Prompt Chaining سلاسل المطالبات

تكسر سلاسل المطالبات المهمة المعقدة إلى خطوات مركزة متتابعة:

```
Step 1: Analyze auth.ts (local issues only)
       -> Output: list of issues in auth.ts

Step 2: Analyze database.ts (local issues only)
       -> Output: list of issues in database.ts

Step 3: Integration pass (cross-file dependencies)
       -> Output: issues at module boundaries
```

لماذا هذا مهم:

- يتجنب **تشتت الانتباه**؛ عندما يستقبل النموذج ملفات كثيرة دفعة واحدة، قد يفوت أخطاء في بعضها ويقدم تعليقات سطحية على أخرى
- يضمن جودة تحليل متسقة لكل ملف

- يسمح بتحليل منفصل للتفاعلات بين الملفات

متى تستخدم prompt chaining مقابل التفكيك الديناميكي:

- **Prompt chaining** — مهام متوقعة ومتكررة (مراجعة كود، ترحيلات ملفات)
- **التفكيك الديناميكي** — تحقيقات مفتوحة تتضح مهامها الفرعية أثناء التنفيذ

6.4 نمط "المقابلة"

قبل تنفيذ حل، يطرح Claude أسئلة توضيحية:

```
Claude: "Before implementing caching for the API, a few questions:
1. Which cache invalidation strategy do you prefer—TTL or event-based?
2. Is stale data acceptable when the cache is unavailable?
3. Should caching be per-user or global?
4. What is the expected data volume to cache?"
```

متى يكون مفيدًا:

- مجال غير مألوف (fintech، الرعاية الصحية، الأنظمة القانونية)
- مهام ذات تبعات غير واضحة (استراتيجيات التخزين المؤقت، أنماط الفشل)
- عدة مقاربات قابلة للتطبيق يعتمد اختيار الأفضل منها على السياق

6.5 التحقق وإعادة المحاولة مع التغذية الراجعة

عندما تفشل البيانات المستخرجة في التحقق:

```
Step 1: Extract data from the document
Step 2: Validate (Pydantic, JSON Schema, business rules)
Step 3: If there's an error—retry with context:
  - The original document
  - The previous (incorrect) extraction
  - The specific error: "Field 'total' = 150, but sum(line_items) = 145. Re-check
values."
```

متى تكون إعادة المحاولة فعالة:

- أخطاء الصيغة (تاريخ بصيغة خاطئة)
- أخطاء بنيوية (حقل في مكان خاطئ)
- عدم اتساق حسابي (يستطيع النموذج إعادة الفحص)

متى لا تساعد إعادة المحاولة:

- المعلومة غير موجودة في المستند المصدر
- السياق المطلوب خارجي (البيانات في مستند آخر غير مقدم)

أداة تحقق Pydantic: Pydantic مكتبة Python للتحقق من البيانات بناءً على المخططات. لأغراض الاختبار، النقاط الأساسية هي:

- **التحقق البنيوي:** الأنواع، الإلزام، وقيود enum تُفحص في الكود بعد استقبال JSON من Claude

- **التحقق الدلالي:** متحققات مخصصة تفرض منطق الأعمال (مجموع العناصر يساوي الإجمالي؛ `start_date < end_date`)
- **حلقات تحقق–إعادة محاولة:** عند فشل تحقق Pydantic، ابنِ رسالة خطأ وأعد مطالبة Claude مع سياق الخطأ
- **توليد JSON Schema:** تستطيع نماذج Pydantic توليد JSON Schema لـ `tool_use`، فتكون مصدر حقيقة واحدًا

6.6 التصحيح الذاتي

نمط لاكتشاف التناقضات الداخلية:

```
{
  "stated_total": "$150.00",
  "calculated_total": "$145.00",
  "conflict_detected": true,
  "line_items": [
    {"name": "Widget A", "price": 75.00},
    {"name": "Widget B", "price": 70.00}
  ]
}
```

بالتعامل مع `conflict_detected` يستخرج النموذج القيمة المصرّح بها والقيمة المحسوبة؛ إذا اختلفتا، يسمح التعارض.

الفصل 7: Message Batches API

> التوثيق: Message Batches

7.1 نظرة عامة

إرسال دفعات من الطلبات للمعالجة غير المتزامنة تتيح Message Batches API:

القيمة	الخاصية
50% مقارنة بالاستدعاءات المتزامنة	التوفير
24 ساعة حتى (لا يوجد ضمان SLA للكمون)	نافذة المعالجة
غير مدعوم (طلب واحد = استجابة واحدة)	استدعاء أدوات متعدد الأدوار
حقل `custom_id` لربط الطلب بالاستجابة	الربط

7.2 Synchronous API مقابل Batch API متى تستخدم

السبب	API	المهمة

فحص PR قبل الدمج	**Synchronous**	المطور ينتظر؛ 24 ساعة غير مقبولة
تقرير ديون تقنية ليلي	**Batch**	النتيجة مطلوبة صباحًا؛ توفير 50%
تدقيق أمني أسبوعي	**Batch**	غير عاجل؛ توفير 50%
مراجعة كود تفاعلية	**Synchronous**	استجابة فورية مطلوبة
معالجة 10,000 مستند	**Batch**	معالجة ضخمة؛ التوفير كبير

7.3 استخدام `custom_id`

```
{
  "custom_id": "doc-invoice-2024-001",
  "params": {
    "model": "claude-sonnet-4-6",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "Extract data from: ..."}]
  }
}
```

يسمح `custom_id` بـ:

- ربط النتيجة بالمستند الأصلي
- عند الفشل، إعادة إرسال المستندات الفاشلة فقط
- تجنب إعادة معالجة المستندات الناجحة

7.4 التعامل مع الإخفاقات في الدفعات

1. أرسل دفعة من 100 مستند
2. نجح 95 وفشل 5 (تجاوز حد السياق)
3. حدد الإخفاقات عبر `custom_id`
4. عدّل الاستراتيجية (مثل تقسيم المستندات الطويلة إلى أجزاء)
5. أعد إرسال المستندات الخمسة الفاشلة فقط

7.5 التخطيط حسب SLA

إذا كنت تحتاج النتيجة خلال 30 ساعة وكان Batch API قد يستغرق حتى 24 ساعة:

- نافذة الإرسال: 30 - 24 = **6 ساعات**
- يجب إرسال الدفعات قبل الموعد النهائي بـ 24 ساعة على الأقل
- للإرسال المتكرر، قسّمها إلى نوافذ من 4 ساعات

الفصل 8: استراتيجيات تفكيك المهام

8.1 خطوط ثابتة (Prompt Chaining)

تُعرّف كل خطوة مسبقًا:

```
Document -> Metadata extraction -> Data extraction -> Validation -> Enrichment -> Final
output
```

متى تستخدمها:

- بنية المهمة متوقعة (المراجعات تتبع القالب نفسه دائمًا)
- كل الخطوات معروفة مسبقًا
- تحتاج إلى الاستقرار وإمكانية إعادة الإنتاج

8.2 التفكيك الديناميكي التكيفي

تُولّد المهام الفرعية بناءً على النتائج الوسيطة:

```
1. "Add tests for a legacy codebase"
2. -> First: map the structure (Glob, Grep)
3. -> Found: 3 modules with no tests, 2 with partial coverage
4. -> Prioritize: start with the payments module (high risk)
5. -> During work: discovered a dependency on an external API
6. -> Adapt: add a mock for the external API before writing tests
```

متى تستخدمه:

- المهام الاستقصائية المفتوحة
- عندما لا يكون النطاق الكامل معروفًا مسبقًا
- عندما تعتمد كل خطوة على نتائج الخطوة السابقة

8.3 مراجعة كود متعددة المرور

لطلبات السحب التي تحتوي على أكثر من 10 ملفات:

```
Pass 1 (per-file): Analyze auth.ts -> list local issues
Pass 1 (per-file): Analyze database.ts -> list local issues
Pass 1 (per-file): Analyze routes.ts -> list local issues
...
Pass 2 (integration): Analyze relationships between files
  -> Cross-file issues: inconsistent types, circular dependencies
```

لماذا المرور الواحد على 14 ملفًا سيئ:

- تشتت الانتباه: تحليل عميق لبعض الملفات وسطحي لأخرى
- تعليقات غير متسقة: يُرفع نمط في ملف ويُقبل في آخر

- أخطاء فائتة: تُتجاوز أخطاء واضحة بسبب الحمل المعرفي

الفصل 9: التصعيد وإدخال الإنسان في الحلقة

9.1 متى تصعّد إلى إنسان

محفزات التصعيد (قواعد واضحة):

الحالة	الإجراء
يطلب العميل صراحة “get me a manager”	صعّد فورًا؛ لا تحاول الحل
السياسة لا تغطي الطلب	صعّد (مثل مطابقة سعر منافس عندما تكون السياسة صامتة)
لا يستطيع الوكيل إحراز تقدم	صعّد بعد عدد معقول من المحاولات
عملية مالية فوق حد معين	صعّد (يفضل فرض ذلك بخطاف لا بمطالبة)
وجود عدة مطابقات عند البحث عن عميل	اطلب معرفات إضافية؛ لا تخمن

ما لا يُعد محفزًا موثوقًا:

طريقة غير موثوقة	لماذا تفشل
تحليل المشاعر	مزاج العميل لا يرتبط بتعقيد القضية
تقييم النموذج لثقته بنفسه (1–10)	قد يكون النموذج واثقًا وهو مخطئ؛ المعايرة ضعيفة
مصنف تلقائي	تعقيد زائد؛ قد يحتاج بيانات تدريب غير متاحة

9.2 أنماط التصعيد

تصعيد فوري:

```
Customer: "I want to speak to a manager"
Agent: [immediately calls escalate_to_human]
NOT: "I can help with your issue, let me..."
```

تصعيد بعد محاولة حل:

```
Customer: "My refrigerator broke two days after purchase"
Agent: [checks the order, offers a warranty replacement]
If the customer is not satisfied -> escalate
```

تصعيد دقيق (اعتراف → حل → تصعيد عند التكرار):

```
Customer: "This is outrageous, I'm very unhappy with the quality!"
Agent: [acknowledges frustration] "I understand your frustration."
       [offers resolution] "I can offer a replacement or a refund."
Customer: "No, I want to talk to someone!"
Agent: [customer insists again -> immediate escalation]
```

المبدأ الأساسي: اعترف بالمشاعر أولًا، ثم اقترح حلًا ملموسًا، ولا تصعّد إلا إذا كرر العميل رغبته في إنسان. لا تصعّد عند أول تعبير عن عدم الرضا، فهذا ليس مثل طلب المدير.

التصعيد بسبب فجوة في السياسة:

```
Customer: "Competitor X has this item 30% cheaper—give me a discount"
Policy: covers price adjustments only on your own site
Agent: [escalates — policy does not cover competitor price matching]
```

9.3 بروتوكولات تسليم منظمة

عند التصعيد، ينبغي أن يمرر الوكيل ملخصًا منظمًا إلى الإنسان:

```json
{
  "customer_id": "CUST-12345",
  "customer_name": "Ivan Petrov",
  "issue_summary": "Refund request for a damaged item",
  "order_id": "ORD-67890",
  "root_cause": "Item arrived damaged; photos attached",
  "actions_taken": [
    "Verified customer via get_customer",
    "Confirmed order via lookup_order",
    "Offered a standard replacement — customer insists on a refund"
  ],
  "refund_amount": "$89.99",
  "recommended_action": "Approve a full refund",
  "escalation_reason": "Customer requested to speak with a manager"
}
```

لا يرى المشغل البشري النص الكامل للمحادثة، بل هذا الملخص فقط. لذلك يجب أن يكون كاملًا ومكتفيًا بذاته.

9.4 معايرة الثقة والرقابة البشرية

لأنظمة استخراج البيانات:

1. **درجات ثقة على مستوى الحقول:** يخرج النموذج درجة ثقة لكل حقل مستخرج
2. **المعايرة:** استخدم مجموعات تحقق موسومة لضبط العتبات
3. **التوجيه:**
   - ثقة عالية + دقة مستقرة -> معالجة آلية
   - ثقة منخفضة أو مصادر غامضة -> مراجعة بشرية

أخذ عينات عشوائية طبقية:

- حتى للاستخراجات عالية الثقة، دقق عينة بانتظام
- قد تخفي دقة إجمالية 97% أخطاء بنسبة 40% لنوع مستند معين
- حلل الدقة حسب نوع المستند والحقل، لا الإجمالي فقط

الفصل 10: معالجة الأخطاء في الأنظمة متعددة الوكلاء

10.1 فئات الأخطاء

الفئة	أمثلة	قابلة للإعادة؟	إجراء الوكيل
عابر	Timeout، 503، فشل شبكة	نعم	إعادة المحاولة مع backoff أسي
تحقق	صيغة إدخال غير صالحة، حقل مطلوب ناقص	لا (أصلح الإدخال)	عدّل الطلب وأعد المحاولة
أعمال	انتهاك سياسة، تجاوز حد	لا	اشرح للمستخدم واقترح بديلًا
إذن	رفض الوصول	لا	صعّد

10.2 أنماط مضادة في معالجة الأخطاء

النهج الصحيح	المشكلة	النمط المضاد
أرجع نوع الخطأ، الاستعلام، النتائج الجزئية، البدائل	لا يستطيع المنسق تقرير كيفية التعافي	حالة عامة “search unavailable”
”ميّز بين “لا نتائج” و“فشل البحث	يظن المنسق أنه لا توجد مطابقات بينما كان هناك فشل	الكبت الصامت (نتيجة فارغة = نجاح)
تابع بالنتائج الجزئية وعلّق الفجوات	تخسر كل النتائج الجزئية	إيقاف سير العمل كله بسبب فشل واحد
تعافٍ محلي (1–2 محاولة)، ثم مرّر للمنسق	كمون وموارد مهدرة	إعادة محاولات لا نهائية داخل وكيل فرعي

10.3 خطأ وكيل فرعي منظم

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "AI impact on music industry 2024",
  "partial_results": [
    {"title": "AI Music Generation Report", "url": "...", "relevance": 0.8}
  ],
  "alternative_approaches": [
    "Try a narrower query: 'AI music composition tools'",
    "Use an alternative data source"
  ],
  "coverage_impact": "Not covered: AI impact on music production"
}
```

يوفر هذا للمنسق المعلومات اللازمة لاتخاذ القرار:

- هل يعيد المحاولة باستعلام معدل؟
- هل يستخدم النتائج الجزئية؟
- هل يفوض لوكيل فرعي مختلف؟
- هل يتابع دون هذا القسم مع توضيح الفجوة؟

10.4 تعليقات التغطية في التركيب النهائي

```
## Report: AI Impact on Creative Industries

### Visual Art (FULL COVERAGE)
[research results]

### Music (PARTIAL COVERAGE — search agent timeout)
[partial results]
⚠️ Note: coverage for this section is limited due to a timeout in the search agent.

### Literature (FULL COVERAGE)
[research results]
```

الفصل 11: إدارة السياق في أنظمة الإنتاج

11.1 استخراج الحقائق في كتلة منفصلة

بدلًا من الاعتماد على سجل المحادثة (الذي يتدهور أثناء التلخيص)، استخرج الحقائق الأساسية في كتلة منظمة:

```
=== CASE FACTS (updated whenever a new fact appears) ===
Customer ID: CUST-12345
Order ID: ORD-67890
Order Date: 2025-01-15
Order Amount: $89.99
Issue: Damaged item on delivery
Customer Request: Full refund
Status: Pending manager approval
===
```

أدرج هذه الكتلة في كل مطالبة بغض النظر عن كيفية تلخيص السجل.

11.2 تقليم نتائج الأدوات

إذا كان `lookup_order` يرجع أكثر من 40 حقلًا ولا تحتاج إلا 5 للمهمة الحالية:

```python
# PostToolUse hook: keep only relevant fields
@hook("PostToolUse", tool="lookup_order")
def trim_order_fields(result):
    return {
        "order_id": result["order_id"],
        "status": result["status"],
        "total": result["total"],
        "items": result["items"],
        "return_eligible": result["return_eligible"]
    }
```

هذا يحافظ على السياق ويقلل الضجيج.

11.3 إدخال واعٍ بالموقع

ضع المعلومات الحرجة مع مراعاة تأثير الضياع في الوسط:

```
[KEY FINDINGS — at the top]
Found 3 critical vulnerabilities...

[DETAILED RESULTS — middle]
=== File auth.ts ===
...
=== File database.ts ===
...

[ACTION ITEMS — at the end]
Priority: fix auth.ts vulnerabilities before merge.
```

11.4 ملفات Scratchpad

في التحقيقات الطويلة، يستطيع الوكيل كتابة النتائج الأساسية إلى ملف scratchpad:

```
# investigation-scratchpad.md
## Key findings
- PaymentProcessor in src/payments/processor.ts inherits from BaseProcessor
- refund() is called from 3 places: OrderController, AdminPanel, CronJob
- External PaymentGateway API has a rate limit of 100 req/min
- Migration #47 added refund_reason (NOT NULL) — 2024-12-01
```

بدلًا من إعادة الاستكشاف scratchpad عندما يتدهور السياق (أو في جلسة جديدة)، يستطيع الوكيل الرجوع إلى.

11.5 التفويض للوكلاء الفرعيين لحماية السياق

```
Main agent: "Investigate dependencies of the payments module"
  -> Subagent (Explore): reads 15 files, traces imports
  -> Returns: "Payments depends on AuthService, OrderModel, and the external
PaymentGateway API"

Main agent: keeps one line in context instead of 15 files
```

طبقة سياق منفصلة: في الأنظمة متعددة الوكلاء، يعمل كل وكيل فرعي ضمن ميزانية سياق محدودة، ولا يستقبل إلا المعلومات المطلوبة لمهمته. يعمل المنسق كطبقة سياق منفصلة: يجمع مخرجات الوكلاء الفرعيين، ويخزن الحالة العامة، ويوزع السياق. يمنع هذا "تسرب السياق"، حيث يستهلك وكيل واحد النافذة بمعلومات غير ذات صلة للآخرين.

ميزانيات سياق مقيدة للوكلاء الفرعيين:

- أرسل الحد الأدنى من السياق: مهمة محددة + البيانات اللازمة
- اطلب من الوكيل الفرعي إرجاع نتائج منظمة لا تفريغات بيانات خام
- استخدم `allowedTools` لتقييد مجموعة أدوات الوكيل الفرعي؛ أدوات أقل تعني مشتتات أقل وتكلفة سياق أقل

11.6 استمرار الحالة المنظمة (للتعافي من الانهيار)

يصدّر كل وكيل حالته إلى موقع معروف:

```
// agent-state/web-search-agent.json
{
  "status": "completed",
  "queries_executed": ["AI music 2024", "AI music composition"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["music composition", "music production"],
  "gaps": ["music distribution", "music licensing"]
}
```

عند الاستئناف manifest يحمّل المنسق:

```
// agent-state/manifest.json
{
  "web-search": "completed",
  "doc-analysis": "in_progress",
  "synthesis": "not_started"
}
```

الفصل 12: الحفاظ على المصدرية

12.1 مشكلة فقدان الإسناد

عند تلخيص نتائج من مصادر متعددة، قد تضيع صلة "الادعاء ← المصدر":

```
Bad: "The AI music market is estimated at $3.2B." (No source, no year.)

Good:
{
  "claim": "The AI music market is estimated at $3.2B.",
  "source_url": "https://example.com/report",
  "source_name": "Global AI Music Report 2024",
  "publication_date": "2024-06-15",
  "confidence": 0.9
}
```

12.2 التعامل مع البيانات المتعارضة

عندما يقدم مصدران قيمًا مختلفة:

```
{
  "claim": "Share of AI-generated music on streaming platforms",
  "values": [
    {
      "value": "12%",
      "source": "Spotify Annual Report 2024",
      "date": "2024-03",
      "methodology": "Automated classification"
    },
    {
      "value": "8%",
      "source": "Music Industry Association Survey",
      "date": "2024-07",
      "methodology": "Survey of 500 labels"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "Difference in methodology and time period"
}
```

لا تختر قيمة عشوائيًا. احتفظ بالقيمتين مع الإسناد ودع المنسق يقرر.

12.3 تضمين التواريخ للتفسير الصحيح

بدون التواريخ، قد تُفسر الفروق الزمنية كتعارضات:

```
Bad: "Source A says 10%, source B says 15%. Contradiction."
Good: "Source A (2023) says 10%, source B (2024) says 15%. Likely +5% growth over a year."
```

12.4 العرض حسب نوع المحتوى

لا تجبر كل شيء على صيغة واحدة:

- البيانات المالية -> جداول
- الأخبار والتحليل -> نثر
- النتائج التقنية -> قوائم منظمة
- السلاسل الزمنية -> ترتيب زمني

الفصل 13: أدوات Claude Code المدمجة

13.1 مرجع اختيار الأدوات

مثال	الأداة	المهمة

العثور على الملفات بالاسم/النمط	**Glob**	`**/*.test.tsx`، `src/components/**/*.ts`
البحث داخل الملفات	**Grep**	اسم دالة، رسالة خطأ، import
قراءة ملف كامل	**Read**	تحميل ملف للتحليل
كتابة ملف جديد	**Write**	إنشاء ملف من الصفر
تحرير ملف موجود بدقة	**Edit**	استبدال مقطع محدد عبر تطابق نص فريد
تشغيل أمر shell	**Bash**	git، npm، تشغيل الاختبارات، build

13.2 استراتيجية التحقيق التدريجي

لا تقرأ كل الملفات دفعة واحدة. ابنِ الفهم تدريجيًا:

```
1. Grep: find entry points (function definition, export)
2. Read: read the found files
3. Grep: find usages (import, calls)
4. Read: read consumer files
5. Repeat until you have a complete picture
```

13.3 الرجوع إلى Read + Write بدلًا من Edit

عندما يفشل Edit بسبب تطابق نص غير فريد:

1. Read — حمّل محتوى الملف كاملًا
2. عدّل المحتوى برمجيًا
3. Write — اكتب النسخة المحدثة

الجزء الثاني: ملاحظات مجالات الاختبار

المجال 1: بنية الوكلاء والتنسيق (27%)

1.1 تصميم حلقات وكيلة لتنفيذ المهام ذاتيًا

المعرفة الأساسية:

- دورة حياة حلقة الوكيل: أرسل طلب Claude، افحص `stop_reason` (`"tool_use"` مقابل `"end_turn"`)، نفذ الأدوات، أعد النتائج للتكرار التالي
- تُضاف نتائج الأدوات إلى سجل المحادثة حتى يستطيع النموذج تقرير الإجراء التالي
- مقابل أشجار القرار المبرمجة مسبقًا (يختار الأداة التالية Claude) اتخاذ القرار بقيادة النموذج

المهارات الأساسية:

- التحكم في التدفق: تابع الحلقة عند `stop_reason = "tool_use"` وتوقف عند `"end_turn"`
- إضافة نتائج الأدوات إلى السياق بين التكرارات
- تجنب الأنماط المضادة: تحليل نص المساعد لاكتشاف الاكتمال، واستخدام حد تكرار اعتباطي كآلية توقف أساسية

1.2 تنسيق الأنظمة متعددة الوكلاء (منسق–وكيل فرعي)

المعرفة الأساسية:

- بنية المحور والأذرع: يملك المنسق كل اتصالات الوكلاء، ومعالجة الأخطاء، والتوجيه
- تعمل الوكلاء الفرعيون بسياق معزول؛ لا يرثون سجل المنسق تلقائيًا
- مسؤوليات المنسق: تفكيك المهمة، التفويض، تجميع النتائج، الاختيار الديناميكي للوكلاء الفرعيين
- خطر التفكيك الضيق جدًا من المنسق

المهارات الأساسية:

- تقسيم تغطية البحث بين الوكلاء الفرعيين لتقليل التكرار
- تنفيذ حلقات تحسين تكرارية (يقيم المنسق التركيب ويعيد توجيه المهام)
- توجيه كل الاتصالات عبر المنسق للملاحظة

1.3 إعداد استدعاءات الوكلاء الفرعيين، وتمرير السياق، والإنشاء

المعرفة الأساسية:

- أداة `Task` تنشئ الوكلاء الفرعيين؛ يجب أن تتضمن `allowedTools` للمنسق `"Task"`
- يجب إدراج سياق الوكيل الفرعي صراحة في المطالبة؛ لا يرث الوكلاء الفرعيون سياق الأب
- إعداد `AgentDefinition`: الأوصاف، مطالبات النظام، قيود الأدوات
- إدارة الجلسات عبر `fork_session` لاستكشاف البدائل

المهارات الأساسية:

- تضمين المخرجات الكاملة من الوكلاء السابقين في مطالبة الوكيل الفرعي
- استخدام صيغ منظمة لفصل البيانات عن البيانات الوصفية عند تمرير السياق
- إنشاء وكلاء فرعيين متوازيين عبر عدة استدعاءات `Task` في دور منسق واحد
- كتابة مطالبات المنسق من حيث الأهداف ومعايير الجودة لا التعليمات خطوة بخطوة

1.4 تنفيذ workflows متعددة الخطوات مع أنماط الإنفاذ والتسليم

المعرفة الأساسية:

- الفرق بين **الإنفاذ البرمجي** (خطافات، شروط مسبقة) و**إرشاد المطالبة** لترتيب سير العمل
- عندما تحتاج إلى ضمانات حتمية (مثل التحقق من الهوية قبل العمليات المالية)، لا تكفي المطالبات وحدها
- بروتوكولات تسليم منظمة أثناء التصعيد (معرف العميل، السبب، الإجراء الموصى به)

المهارات الأساسية:

- شروط مسبقة برمجية تمنع الاستدعاءات اللاحقة حتى تكتمل الخطوات السابقة (مثل منع `process_refund` حتى يرجع `get_customer` معرفًا محققًا)
- تفكيك طلبات العملاء متعددة الجوانب إلى عناصر منفصلة
- إنتاج ملخصات منظمة عند التصعيد إلى إنسان

1.5 خطافات Agent SDK لاعتراض استدعاءات الأدوات وتطبيع البيانات

المعرفة الأساسية:

- لاعتراض نتائج الأدوات قبل استهلاك النموذج لها (مثل `PostToolUse`) أنماط الخطافات
- خطافات تعترض الاستدعاءات الصادرة لفرض قواعد الامتثال (مثل منع refunds فوق حد معين)
- تقدم الخطافات **ضمانات حتمية** مقابل تعليمات المطالبات التي تقدم **امتثالًا احتماليًا**

المهارات الأساسية:

- خطافات `PostToolUse` لتطبيع صيغ البيانات (طوابع Unix، ISO 8601، رموز حالة رقمية)
- خطافات اعتراض لمنع الإجراءات المخالفة للسياسة مع إعادة التوجيه إلى التصعيد
- اختيار الخطافات بدل المطالبات عندما تتطلب قواعد الأعمال امتثالًا مضمونًا

1.6 استراتيجيات تفكيك المهام لسير العمل المعقد

المعرفة الأساسية:

- مقابل **تفكيك ديناميكي تكيفي** بناءً على النتائج الوسيطة (prompt chaining) **خطوط ثابتة**
- Prompt chaining: خطوات متتابعة (حلل كل ملف منفصلًا، ثم نفذ تمرير تكامل)
- خطط تحقيق تكيفية تولد مهام فرعية بناءً على الاكتشافات

المهارات الأساسية:

- استخدم prompt chaining للمراجعات متعددة الجوانب المتوقعة؛ واستخدم التفكيك الديناميكي للتحقيقات المفتوحة
- قسّم مراجعات الكود الكبيرة إلى تحليل لكل ملف ثم تمرير تكامل منفصل بين الملفات
- فكك المهام المفتوحة: ارسم البنية أولًا، ثم ابنِ خطة ذات أولوية

1.7 حالة الجلسة، والاستئناف، والتفريع

المعرفة الأساسية:

- `--resume <session-name>` لمتابعة الجلسات المسماة
- `fork_session` لإنشاء فروع تحقيق مستقلة من سياق مشترك
- أهمية إبلاغ الوكيل بتغييرات الملفات عند استئناف الجلسات
- قد تكون الجلسة الجديدة مع ملخص منظم أكثر موثوقية من الاستئناف بنتائج قديمة

المهارات الأساسية:

- استخدم `--resume` لمتابعة جلسات التحقيق المسماة
- استخدم `fork_session` لمقارنة المقاربات بالتوازي
- اختر بين الاستئناف (السياق ما يزال حديثًا) والبدء من جديد (النتائج قديمة)

المجال 2: تصميم الأدوات وتكامل MCP (18%)

2.1 تصميم واجهات الأدوات بأوصاف واضحة

المعرفة الأساسية:

- أوصاف الأدوات هي **الآلية الأساسية** التي يستخدمها LLM لاختيار الأدوات؛ الأوصاف المختصرة تؤدي إلى اختيار غير موثوق
- أهمية تضمين صيغ الإدخال، أمثلة الاستعلامات، الحالات الحدية، وحدود التطبيق
- الأوصاف الغامضة أو المتداخلة تسبب التوجيه الخاطئ
- صياغة مطالبة النظام قد تنشئ ارتباطات غير مقصودة بالأدوات

المهارات الأساسية:

- كتابة أوصاف تميز كل أداة بوضوح عن البدائل المشابهة
- إعادة تسمية الأدوات لإزالة التداخل الوظيفي (مثل `analyze_content` -> `extract_web_results`)
- تقسيم الأدوات العامة إلى أدوات متخصصة بعقود إدخال/إخراج واضحة

2.2 تنفيذ استجابات أخطاء منظمة لأدوات MCP

المعرفة الأساسية:

- راية `isError` في استجابات أدوات MCP
- الفرق بين **الأخطاء العابرة** (timeouts)، وأخطاء **أخطاء الأعمال** (انتهاكات سياسة)، **أخطاء التحقق** (إدخال سيئ)، الوصول/الأذونات
- الأخطاء العامة ("Operation failed") تمنع قرارات التعافي الصحيحة
- الفرق بين الأخطاء القابلة وغير القابلة لإعادة المحاولة

المهارات الأساسية:

- إرجاع بيانات وصفية منظمة مثل `errorCategory` (transient/validation/permission)، و`isRetryable`، ورسالة مفهومة للإنسان
- استخدام `retryable: false` لانتهاكات قواعد الأعمال مع شروحات واضحة للمستخدم
- تنفيذ التعافي المحلي داخل الوكلاء الفرعيين للأخطاء العابرة؛ وتمرير فقط ما لا يستطيعون حله
- التمييز بين فشل الوصول والنتائج الفارغة الصالحة

2.3 توزيع الأدوات بين الوكلاء وإعداد `tool_choice`

المعرفة الأساسية:

- الأدوات الكثيرة جدًا لكل وكيل (مثل 18 بدل 4–5) **تقلل** موثوقية اختيار الأداة
- الوكلاء الذين يملكون أدوات خارج تخصصهم يميلون إلى إساءة استخدامها
- نطاق وصول الأدوات: الأدوات ذات الصلة بالدور فقط، مع مجموعة محدودة من أدوات عابرة للأدوار
- `tool_choice`: `"auto"` و`"any"` والاختيار الإجباري (`{"type": "tool", "name": "..."}`)

المهارات الأساسية:

- تقييد أدوات كل وكيل فرعي بما يناسب دوره
- استبدال الأدوات العامة ببدائل مقيدة (مثل `fetch_url` -> `load_document`)
- استخدام `tool_choice: "any"` لضمان استدعاء أداة بدل إجابة نصية
- إجبار أداة محددة لضمان ترتيب التنفيذ

2.4 دمج خوادم MCP في Claude Code وسير عمل الوكلاء

المعرفة الأساسية:

- نطاق خادم MCP: المشروع (`.mcp.json`) للفِرق مقابل المستخدم (`~/.claude.json`) للتجارب
- استبدال متغيرات البيئة في `.mcp.json` (مثل `${GITHUB_TOKEN}`) لإدارة الأسرار
- تُكتشف أدوات كل خوادم MCP المتصلة عند الاتصال وتتاح في الوقت نفسه
- موارد MCP كـ "فهارس محتوى" (ملخصات مهام، مخططات قواعد بيانات) لتقليل استدعاءات الأدوات الاستكشافية

المهارات الأساسية:

- إعداد خوادم MCP مشتركة في `.mcp.json` للمشروع مع رموز عبر متغيرات البيئة
- إبقاء الخوادم الشخصية/التجريبية في `~/.claude.json`
- تفضيل خوادم MCP المجتمعية على الخوادم المخصصة للتكاملات القياسية

2.5 اختيار وتطبيق الأدوات المدمجة (Read, Write, Edit, Bash, Grep, Glob)

المعرفة الأساسية:

- **Grep**: البحث داخل محتوى الملفات (أسماء الدوال، رسائل الخطأ، imports)
- **Glob**: العثور على الملفات حسب أنماط الاسم/الامتداد
- **Read/Write**: عمليات على الملف كاملًا؛ **Edit**: تغييرات دقيقة عبر تطابق نص فريد
- إذا فشل Edit بسبب تطابقات غير فريدة، ارجع إلى Read + Write

المهارات الأساسية:

- استخدم Grep للبحث في المحتوى و Globلاكتشاف الملفات بالأنماط
- ابنِ الفهم تدريجيًا: Grep لنقاط الدخول، ثم Read لتتبع التدفقات
- تتبع استخدام الدالة عبر وحدات wrapper

المجال 3: إعداد Claude Code (20%) وسير العمل

3.1 إعداد CLAUDE.md بالهرمية والنطاق والتنظيم المعياري

المعرفة الأساسية:

- هرمية CLAUDE.md: المستخدم (`~/.claude/CLAUDE.md`)، المشروع (`.claude/CLAUDE.md` أو CLAUDE.md في الجذر)، ومستوى المجلد (CLAUDE.md في المجلدات الفرعية)
- إعدادات مستوى المستخدم تنطبق على مستخدم واحد فقط ولا تُشارك عبر VCS
- صيغة `@path` للإشارة إلى ملفات خارجية (مثل `@./standards/coding-style.md`) لجعل CLAUDE.md معياريًا
- مجلد `.claude/rules/` لملفات قواعد مركزة حسب الموضوع بدل CLAUDE.md ضخم

المهارات الأساسية:

- تشخيص مشكلات الهرمية (عضو فريق جديد لا يرى التعليمات لأنها على مستوى المستخدم بدل المشروع)
- استخدام `@path` (مثل `@./standards/testing.md`) لتضمين المعايير انتقائيًا في CLAUDE.md لكل حزمة
- تقسيم CLAUDE.md كبير إلى ملفات `.claude/rules/` متعددة (testing.md، api-conventions.md، deployment.md)

3.2 إنشاء وإعداد أوامر Slash والمهارات المخصصة

المعرفة الأساسية:

- مقابل **أوامر المستخدم** في `~/.claude/commands/` **أوامر المشروع** في `.claude/commands/` (تُشارك عبر VCS)
- المهارات في `.claude/skills/` مع frontmatter في `SKILL.md`: `context: fork`، `allowed-tools`، `argument-hint`
- `context: fork` يشغل المهارة في سياق وكيل فرعي معزول حتى لا يلوث الجلسة الرئيسية
- يمكن وضع نسخ مهارات شخصية في `~/.claude/skills/` بأسماء مختلفة

المهارات الأساسية:

- تخزين أوامر slash الخاصة بالمشروع في `.claude/commands/` حتى يحصل عليها الفريق كله
- استخدام `context: fork` لعزل المهارات ذات الإخراج المطول
- استخدام `allowed-tools` لتقييد الأدوات التي تستطيع المهارة استخدامها
- استخدام `argument-hint` لمطالبة المطورين بالمعاملات المطلوبة

3.3 استخدام قواعد خاصة بالمسارات للتحميل الشرطي للاصطلاحات

المعرفة الأساسية:

- تستطيع ملفات `.claude/rules/` تضمين YAML frontmatter مع `paths` لتفعيل القواعد حسب أنماط glob

- لا تُحمّل قواعد المسارات إلا عند تحرير ملفات مطابقة، مما يوفر السياق والرموز
- على مستوى المجلد عندما تنطبق الاصطلاحات عبر CLAUDE.md أفضل من glob قد تكون القواعد المبنية على عدة مجلدات (مثل الاختبارات)

المهارات الأساسية:

- إنشاء ملفات `.claude/rules/` مع `paths: ["terraform/**/*"]` للتحميل فقط عند العمل على ملفات مطابقة
- استخدام أنماط glob (`**/*.test.tsx`) لتطبيق الاصطلاحات حسب نوع الملف بغض النظر عن الموقع
- تفضيل القواعد الخاصة بالمسار على CLAUDE.md مستوى المجلد عندما تمتد الاصطلاحات عبر قاعدة الكود

3.4 تحديد متى تستخدم وضع التخطيط مقابل التنفيذ المباشر

المعرفة الأساسية:

- **وضع التخطيط**: للمهام المعقدة ذات التغييرات الكبيرة والمقاربات المتعددة والقرارات المعمارية
- **التنفيذ المباشر**: للتغييرات البسيطة المفهومة جيدًا (مثل إضافة تحقق واحد)
- يتيح وضع التخطيط استكشافًا آمنًا لقاعدة الكود قبل إجراء تغييرات
- يعزل Explore subagent إخراج الاكتشاف المطول

المهارات الأساسية:

- استخدم وضع التخطيط للمهام ذات التبعات المعمارية (microservices، migrations ملفًا +45 تمس)
- استخدم التنفيذ المباشر لإصلاحات ذات stack trace واضح وملف واحد
- استخدم Explore subagent لمنع استنزاف نافذة السياق في المهام متعددة المراحل
- اجمع بين النهجين: خطط للاكتشاف، ثم نفذ للتطبيق

3.5 التحسين التكراري للتطوير التدريجي

المعرفة الأساسية:

- أمثلة الإدخال/الإخراج الملموسة هي أكثر الطرق فعالية لتوصيل التوقعات
- **التكرار الموجه بالاختبارات**: اكتب الاختبارات أولًا، ثم كرر بناءً على الإخفاقات
- نمط "المقابلة": يطرح Claude أسئلة لكشف اعتبارات تصميم غير واضحة
- متى تقدم كل المشكلات في رسالة واحدة (مترابطة) مقابل متتابعة (مستقلة)

المهارات الأساسية:

- تقديم 2–3 أمثلة إدخال/إخراج ملموسة لتوضيح متطلبات التحويل
- بناء مجموعات اختبار بالسلوك المتوقع، الحالات الحدية، ومتطلبات الأداء قبل التنفيذ
- استخدام نمط المقابلة لكشف جوانب التصميم (أنماط الفشل، cache إبطال)
- تقديم حالات اختبار ملموسة مع مدخلات عينة ومخرجات متوقعة للحالات الحدية

3.6 دمج Claude Code في خطوط CI/CD

المعرفة الأساسية:

- العلم `-p` (أو `--print`) للوضع غير التفاعلي في خطوط الأتمتة
- `--output-format json` و `--json-schema` للمخرجات المنظمة في CI
- يوفر CLAUDE.md سياق المشروع (معايير الاختبار، معايير المراجعة) لـ Claude Code عند تشغيله من CI
- **عزل سياق الجلسة**: الجلسة نفسها التي ولدت الكود أقل فعالية في مراجعته من مثيل مستقل

المهارات الأساسية:

- تشغيل Claude Code في CI باستخدام `-p` لتجنب التعليق بانتظار إدخال تفاعلي
- استخدام `--output-format json` + `--json-schema` للنتائج المنظمة (مثل تعليقات PR داخلية)
- تضمين نتائج المراجعة السابقة عند إعادة التشغيل بعد commits جديدة (الإبلاغ فقط عن الجديد/غير المُصلح)
- توثيق معايير الاختبار والfixtures المتاحة في CLAUDE.md لتحسين جودة توليد الاختبارات
- تضمين ملفات الاختبار الحالية في السياق عند توليد اختبارات جديدة لتجنب التكرار والحفاظ على الأسلوب

المجال 4: هندسة المطالبات والمخرجات المنظمة (20%)

4.1 تصميم مطالبات بمعايير صريحة لتحسين الدقة

المعرفة الأساسية:

- المعايير الصريحة أكثر فعالية من التعليمات المبهمة (مثل "ارفع التعليقات فقط عندما تناقض الكود" بدل "افحص دقة التعليقات")
- الإرشاد العام مثل "كن أكثر تحفظًا" يعمل أسوأ من معايير فئوية ملموسة
- أثر الإيجابيات الكاذبة على ثقة المطور: ارتفاع الإيجابيات الكاذبة في فئات معينة يقوض الثقة حتى في الفئات الدقيقة

المهارات الأساسية:

- تعريف معايير المراجعة: ما يجب الإبلاغ عنه (أخطاء، أمن) وما يجب تجاهله (أسلوب بسيط)
- تعطيل الفئات ذات الإيجابيات الكاذبة العالية مؤقتًا
- تعريف معايير شدة صريحة مع أمثلة كود لكل مستوى

4.2 استخدام Few-shot لتحسين اتساق المخرجات

المعرفة الأساسية:

- هي أكثر الطرق فعالية لإنتاج إخراج منسق وقابل للتنفيذ باستمرار few-shot أمثلة
- تستطيع few-shot توضيح التعامل مع الحالات الغامضة (اختيار الأداة، فجوات تغطية الاختبارات)
- تساعد few-shot النموذج على التعميم إلى أنماط جديدة بدل تكرار الافتراضات
- يمكن أن تقلل few-shot الهلوسة في مهام الاستخراج

المهارات الأساسية:

- تقديم 2–4 أمثلة موجهة للسيناريوهات الغامضة مع مبررات
- تضمين أمثلة few-shot توضح صيغة الإخراج (الموقع، المشكلة، الشدة، الإصلاح المقترح)
- تقديم أمثلة تميز أنماط الكود المقبولة عن المشكلات الحقيقية
- تقديم أمثلة للاستخراج الصحيح من مستندات ذات بنى مختلفة

4.3 فرض المخرجات المنظمة باستخدام `tool_use` ومخططات JSON

المعرفة الأساسية:

- `tool_use` مع JSON Schemas هو أوثق طريقة لضمان إخراج مطابق للمخطط وإزالة أخطاء النحوية JSON
- مع `tool_choice: "auto"` يستطيع النموذج إرجاع نص؛ ومع `"any"` يجب أن يستدعي أداة؛ والاختيار الإجباري يختار أداة محددة
- الصارمة أخطاء الصيغة لكنها لا تمنع الأخطاء الدلالية (الإجماليات لا تتطابق، قيم غير صحيحة) JSON تزيل مخططات
- تمنع الاختلاق عندما تكون البيانات مفقودة nullable/الحقول الاختيارية

المهارات الأساسية:

- تصميم مخططات مع required فقط عندما تكون المعلومات موجودة دائمًا
- استخدام `enum` مع `other` / `unclear` لتجنب التصنيف الخاطئ أو فقدان البيانات
- استخدام `tool_choice: "any"` لضمان إخراج منظم
- إضافة تحقق دلالي بعد الإخراج عند الحاجة

4.4 تنفيذ التحقق وإعادة المحاولة وحلقات التغذية الراجعة لجودة الاستخراج

المعرفة الأساسية:

- التحقق البنيوي (الأنواع والحقول) لا يكفي للتحقق الدلالي (الإجماليات، العلاقات الزمنية)
- يمكن لـ JSON Schema أو Pydantic أو قواعد الأعمال التحقق من الإخراج
- إعادة المحاولة مع رسالة خطأ محددة أكثر فعالية من إعادة المحاولة العامة
- لا تساعد إعادة المحاولة إذا كانت المعلومة غير موجودة في المصدر

المهارات الأساسية:

- تنفيذ حلقة extract -> validate -> retry with feedback
- تضمين الاستخراج السابق والخطأ المحدد في إعادة المحاولة
- بناء فحوصات حسابية ومنطقية (sum(line_items) == total، start < end)
- استخدام null/unclear بدل إجبار النموذج على التخمين

4.5 تصميم استراتيجيات معالجة دفعات فعالة

المعرفة الأساسية:

- Message Batches API توفر 50% لكنها قد تستغرق حتى 24 ساعة

- لا يدعم استدعاء الأدوات متعدد الأدوار Batch API
- يربط الطلبات بالنتائج ويسهل إعادة إرسال الفاشل فقط `custom_id`
- مناسب للمهام غير العاجلة ذات الحجم الكبير

المهارات الأساسية:

- اختيار Batch للمعالجة الليلية/الأسبوعية أو المستندات الضخمة غير العاجلة
- اختيار synchronous للفحوصات قبل الدمج أو الاستخدام التفاعلي
- استخدام `custom_id` لتتبع الإخفاقات وإعادة معالجة الفاشل فقط
- التخطيط لنوافذ الإرسال حسب SLA

4.6 تصميم بنى مراجعة متعددة المثيلات ومتعددة المرور

المعرفة الأساسية:

- مراجعة الكود في الجلسة نفسها التي ولدت الكود أقل موثوقية
- المرور الواحد على ملفات كثيرة يعاني من تشتت الانتباه
- تحليل كل ملف ثم تمرير تكامل بين الملفات يزيد جودة المراجعة
- يمنع التعليقات المكررة prior review context

المهارات الأساسية:

- استخدام مثيل مستقل لمراجعة الكود المولد
- تقسيم مراجعة PR كبير إلى pass و pass تكاملي لكل ملف
- تضمين نتائج المراجعة السابقة عند إعادة المراجعة
- تصميم مطالبات تقلل false positives وتحدد الفئات بوضوح

المجال 5: إدارة السياق والموثوقية (15%)

5.1 إدارة سياق المحادثة للحفاظ على المعلومات الحرجة

المعرفة الأساسية:

- كل طلب API مستقل ويتطلب إرسال التاريخ الكامل
- التلخيص قد يفقد أرقامًا وتواريخ وتفاصيل دقيقة
- يجعل المعلومات في منتصف السياق أقل موثوقية lost-in-the-middle تأثير
- كتل الحقائق المنظمة تساعد على الحفاظ على الحالة الحرجة

المهارات الأساسية:

- استخراج facts إلى كتلة حالة منفصلة وإدراجها دائمًا في المطالبة
- وضع المعلومات الحرجة في بداية أو نهاية السياق
- تقليم نتائج الأدوات للاحتفاظ بالحقول ذات الصلة فقط
- استخدام scratchpad للنتائج المهمة في التحقيقات الطويلة

5.2 تصميم أنماط تصعيد فعالة وحل الغموض

المعرفة الأساسية:

- طلب الإنسان الصريح يؤدي إلى تصعيد فوري
- تعبير العميل عن الإحباط ليس سببًا كافيًا للتصعيد بحد ذاته
- عندما توجد عدة مطابقات، يجب طلب معرفات إضافية بدل التخمين
- التصعيد يجب أن يمرر ملخصًا منظمًا مكتفيًا بذاته

المهارات الأساسية:

- التمييز بين طلب المدير الصريح والتذمر العام
- الاعتراف بالمشاعر ثم تقديم حل ملموس قبل التصعيد عند الحاجة
- التصعيد عند فجوات السياسة أو العمليات المالية فوق الحد
- إنشاء handoff structured summary كامل

5.3 تنفيذ استراتيجيات انتشار الأخطاء في الأنظمة متعددة الوكلاء

المعرفة الأساسية:

- يجب أن تتضمن الأخطاء معلومات مثل النوع، القابلية لإعادة المحاولة، الاستعلام، والنتائج الجزئية
- الفشل الجزئي لا يجب أن يوقف سير العمل كله
- يجب التمييز بين "لا نتائج" و"فشل"
- ينبغي أن يتعافى الوكيل الفرعي محليًا من الأخطاء العابرة ثم يمرر ما يفشل

المهارات الأساسية:

- تصميم استجابات أخطاء منظمة للوكلاء الفرعيين
- الاستمرار بالنتائج الجزئية مع تعليقات تغطية
- استخدام backoff للأخطاء العابرة وعدد محدود من المحاولات
- تمرير alternative approaches للمنسق

5.4 إدارة السياق بكفاءة عند التحقيق في قواعد كود كبيرة

المعرفة الأساسية:

- قراءة كل الملفات دفعة واحدة تهدر السياق
- Read تساعد على بناء خريطة أولية قبل Grep/Glob
- Explore subagent يعزل الإخراج المطول
- CLAUDE.md والقواعد الخاصة بالمسار تقلل تحميل التعليمات غير ذات الصلة

المهارات الأساسية:

- استخدام Grep و Globلاكتشاف الملفات لنقاط الدخول
- قراءة الملفات تدريجيًا حسب الحاجة
- تفويض الاستكشاف الواسع لوكيل فرعي يعيد ملخصًا

- استخدام path-scoped rules لتقليل السياق المحمّل

5.5 تصميم workflows ثقة بشرية ومعايرة برقابة

المعرفة الأساسية:

- الثقة الذاتية للنموذج ليست موثوقة وحدها
- يجب معايرة العتبات باستخدام بيانات موسومة
- يمكن أن تخفي الدقة الإجمالية أخطاء كبيرة في فئات فرعية
- العشوائي الطبقي sampling يكشف مشكلات التغطية

المهارات الأساسية:

- استخدام field-level confidence مع عتبات مراجعة بشرية
- إجراء تدقيق عشوائي حتى على النتائج عالية الثقة
- تحليل الدقة حسب نوع المستند والحقل
- توجيه الحالات الغامضة أو منخفضة الثقة إلى الإنسان

5.6 الحفاظ على المصدرية والتعامل مع عدم اليقين في التركيب متعدد المصادر

المعرفة الأساسية:

- يجب الحفاظ على رابط الادعاء بالمصدر أثناء التلخيص
- البيانات المتعارضة يجب أن تُعرض مع المصدر والمنهجية والتاريخ
- التواريخ تساعد على تمييز النمو من التعارض
- تختلف صيغة العرض المناسبة حسب نوع المحتوى

المهارات الأساسية:

- تخزين كل claim مع source_urlو source_nameو dateوconfidence
- عرض القيم المتعارضة بدل اختيار واحدة بلا أساس
- شرح الاختلافات المحتملة في المنهجية أو الفترة الزمنية
- استخدام الجداول للبيانات، والنثر للتحليل، والقوائم للنتائج التقنية

أمثلة على أسئلة الاختبار مع الشرح

السؤال 1 (السيناريو: وكيل خدمة العملاء)

الموقف: يقول عميل: "أريد التحدث إلى المدير الآن." ما الإجراء الصحيح؟

ما الإجراء الصحيح؟

- A) محاولة حل المشكلة أولًا.

- B) إجراء تحليل مشاعر.
- C) استدعاء `escalate_to_human` فورًا. **[صحيح]**
- D) طلب تقييم رضا العميل.

لماذا C: طلب الإنسان صراحة هو محفز تصعيد فوري. لا ينبغي للوكيل محاولة الحل الذاتي عندما يطلب العميل مديرًا أو إنسانًا بشكل واضح.

السؤال 2 (السيناريو: وكيل خدمة العملاء)

الموقف: تريد ضمان عدم تنفيذ `process_refund` قبل التحقق من هوية العميل. ما أفضل حل؟

ما أفضل حل؟

- A) أضف تعليمة في `system prompt`.
- B) استخدم hook أو شرطًا برمجيًا يمنع الاستدعاء. **[صحيح]**
- C) اطلب من النموذج تقييم ثقته.
- D) زد `max_tokens`.

لماذا B: العمليات المالية تحتاج ضمانًا حتميًا. المطالبات احتمالية، أما الخطافات والشروط البرمجية فتمنع الاستدعاء فعليًا.

السؤال 3 (السيناريو: وكيل خدمة العملاء)

الموقف: أداة MCP ترجع `isError: true` مع النص "Operation failed". ما المشكلة؟

ما المشكلة؟

- A) الخطأ ليس JSON.
- B) لا يعطي الوكيل معلومات للتعافي. **[صحيح]**
- C) يجب أن يكون `isError` هو `false`.
- D) يجب حذف الأداة.

لماذا B: الخطأ العام لا يوضح هل هو عابر، أو متعلق بالتحقق، أو بالإذن، أو بسياسة الأعمال. لذلك لا يستطيع الوكيل اختيار إعادة المحاولة أو التصعيد أو تعديل الطلب.

السؤال 4 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: تريد تغييرًا معماريًا يمس عشرات الملفات. ما النهج الأنسب؟

ما النهج الأنسب؟

- A) تنفيذ مباشر فورًا.
- B) وضع التخطيط أولًا ثم التنفيذ بعد الموافقة. **[صحيح]**
- C) استخدام `/compact` فقط.
- D) حذف CLAUDE.md.

لماذا B: التغييرات الكبيرة أو المعمارية تحتاج استكشافًا آمنًا وخطة معتمدة قبل التنفيذ.

السؤال 5 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: أين تضع معايير المشروع التي يجب أن يستفيد منها كل الفريق؟

أين تضعها؟

- A) `~/.claude/CLAUDE.md`
- B) `.claude/CLAUDE.md` أو `CLAUDE.md` **[صحيح]** في جذر المشروع.
- C) في متغير بيئة محلي فقط.
- D) في ملف لا يدخل VCS.

لماذا B: تعليمات المشروع يجب أن تكون ضمن المستودع حتى تصل لكل من يستنسخ المشروع.

السؤال 6 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: قاعدة اختبار تنطبق على كل ملفات `*.test.ts` المنتشرة في مجلدات كثيرة. ما أفضل مكان لها؟

ما أفضل مكان؟

- A) CLAUDE.md داخل مجلد واحد.
- B) `.claude/rules/` مع `paths: ["**/*.test.ts"]`. **[صحيح]**
- C) مطالبة يدوية في كل مرة.
- D) `~/.claude.json`

لماذا B: قواعد المسارات مناسبة عندما تنطبق الاصطلاحات على نوع ملف منتشر عبر قاعدة الكود.

السؤال 7 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: وكيل فرعي لا يرى نتائج البحث السابقة التي جمعها المنسق. ما السبب المرجح؟

ما السبب المرجح؟

- A) الوكلاء الفرعيون لا يرثون سياق المنسق تلقائيًا. **[صحيح]**
- B) يجب زيادة درجة الحرارة.
- C) يجب استخدام Batch API.
- D) يجب حذف `Task`.

لماذا A: سياق الوكيل الفرعي معزول. يجب تمرير كل السياق المطلوب صراحة في مطالبة `Task`.

السؤال 8 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: تريد أن يعمل ثلاثة وكلاء فرعيين بالتوازي. كيف تفعل ذلك؟

كيف تفعل ذلك؟

- A) استدعاء عدة `Task` في استجابة واحدة من المنسق. **[صحيح]**
- B) استخدام `/compact`.
- C) فرض `tool_choice: auto` فقط.
- D) إرسال كل شيء لوكيل واحد.

لماذا A: يمكن للمنسق إنشاء عدة مهام في الدور نفسه، فتعمل الوكلاء الفرعيون بالتوازي.

السؤال 9 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: فشل وكيل البحث جزئيًا لكنه وجد بعض النتائج. ما أفضل إخراج؟

ما أفضل إخراج؟

- A) إرجاع "failed" فقط.
- B) إرجاع نتائج جزئية، نوع الفشل، الاستعلام، وتأثير التغطية. **[صحيح]**
- C) إسقاط النتائج الجزئية.
- D) إعادة المحاولة إلى ما لا نهاية.

لماذا B: الإخراج المنظم يمكّن المنسق من استخدام النتائج الجزئية أو إعادة التفويض أو وضع تعليق تغطية.

السؤال 10 (السيناريو: Claude Code for CI)

الموقف: ما العلم الصحيح لتشغيل Claude Code في خط CI دون تفاعل؟

ما العلم الصحيح؟

- A) `--resume`
- B) `-p` أو `--print`. **[صحيح]**
- C) `/memory`
- D) `fork_session`

لماذا B: `-p` يشغل Claude في وضع غير تفاعلي، يطبع النتيجة ثم يخرج.

السؤال 11 (السيناريو: Claude Code for CI)

الموقف: كيف تقلل التعليقات المكررة عند إعادة مراجعة PR بعد commits جديدة؟

كيف تقلل التعليقات المكررة؟

- A) احذف كل نتائج المراجعة السابقة.

- B) ضمّن نتائج المراجعة السابقة واطلب فقط الجديد أو غير المحلول. **[صحيح]**
- C) استخدم الجلسة نفسها دائمًا.
- D) عطّل JSON output.

لماذا B: إدراج نتائج المراجعة السابقة يساعد Claude على عدم تكرار الملاحظات القديمة والتركيز على المشكلات الجديدة أو غير المعالجة.

السؤال 12 (السيناريو: مراجعة كود متعددة الملفات)

الموقف: يراجع Claude طلب سحب يحتوي على 14 ملفًا دفعة واحدة، وتلاحظ أنه يقدم تحليلًا عميقًا لأول بضعة ملفات ثم تعليقات سطحية للبقية. ما أفضل إعادة هيكلة للمراجعة؟

ما أفضل إعادة هيكلة؟

- A) زيادة `max_tokens` فقط.
- B) تحليل كل ملف على حدة أولًا، ثم تشغيل تمرير تكاملي للعلاقات بين الملفات. **[صحيح]**
- C) استخدام نموذج أصغر للمراجعة.
- D) حذف الملفات الأقل أهمية من PR.

لماذا B: المراجعة متعددة المرور تقلل تشتت الانتباه. التحليل المحلي لكل ملف يضمن تغطية أفضل، والتمرير التكاملي يلتقط مشكلات تدفق البيانات والاعتمادات بين الملفات.

اختبار تدريبي

> ملاحظة: هذه الأسئلة مصممة لتدريبك على نمط التفكير المطلوب في الاختبار. ركز على سبب صحة الإجابة، لا على حفظ الحروف.

السيناريو: نظام بحث متعدد الوكلاء

السؤال 1 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: لديك منسق يرسل نفس سؤال البحث إلى وكيل بحث ويب ووكيل تحليل مستندات. كلا الوكيلين يرجعان نتائج متداخلة، والتقرير النهائي يكرر النقاط نفسها. ما أفضل تعديل؟

ما أفضل تعديل؟

- A) اجعل كل الوكلاء يملكون كل الأدوات حتى يقرروا بأنفسهم.
- B) اجعل المنسق يحدد نطاقًا واضحًا لكل وكيل، مثل "بحث ويب حديث" مقابل "تحليل المستندات المرفقة". **[صحيح]**
- C) احذف المنسق ودع الوكلاء يتواصلون مباشرة.
- D) أضف المزيد من الرموز إلى `max_tokens`.

لماذا B: المشكلة هي تداخل النطاق. المنسق يجب أن يوزع العمل بحدود واضحة حتى يقل التكرار وتزيد التغطية.

السؤال 2 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: وكيل synthesis يحتاج إلى مصادر دقيقة، لكن الوكلاء الفرعيين يرجعون ملخصات بلا روابط ولا تواريخ. ما أفضل حل؟

ما أفضل حل؟

- A) اطلب من synthesis تخمين المصادر.
- B) اجعل كل وكيل فرعي يرجع claims منظمة مع `source_url` و`publication_date` ودرجة ثقة. **[صحيح]**
- C) زد عدد الوكلاء.
- D) أزل citations من التقرير النهائي.

لماذا B: الحفاظ على المصدرية يبدأ من الوكلاء الفرعيين. يجب تمرير الادعاءات مع مصادرها وتواريخها حتى يستطيع synthesis إنتاج تقرير قابل للتحقق.

السؤال 3 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: المنسق يطلب من وكيل فرعي "Analyze the document"، لكن الوكيل الفرعي يعطي نتيجة عامة وغير مفيدة. ما المشكلة الأساسية؟

ما المشكلة الأساسية؟

- A) المطالبة لا تمرر المستند ولا متطلبات الإخراج صراحة. **[صحيح]**
- B) يجب استخدام Batch API.
- C) يجب تقليل عدد الأدوات إلى صفر.
- D) المشكلة في Markdown.

لماذا A: الوكلاء الفرعيون لا يرثون سياق المنسق تلقائيًا. يجب تمرير المستند، الهدف، والمعايير المطلوبة داخل مطالبة `Task`.

السؤال 4 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: بعد فشل وكيل بحث بسبب timeout، أرجع الوكيل "search unavailable". لماذا هذا سيئ؟

لماذا هذا سيئ؟

- A) لأنه لا يحتوي على تفاصيل تساعد المنسق على التعافي. **[صحيح]**
- B) لأنه طويل جدًا.
- C) لأنه ليس باللغة الإنجليزية.
- D) لأنه لا يستخدم `max_tokens`.

لماذا A: يجب أن تتضمن الأخطاء نوع الفشل، هل هو قابل لإعادة المحاولة، الاستعلام الذي فشل، النتائج الجزئية، والبدائل.

السؤال 5 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: تريد تقليل زمن التنفيذ في بحث يتضمن 4 محاور مستقلة. ما أفضل تصميم؟

ما أفضل تصميم؟

- A) اجعل المنسق يرسل كل محور كـ `Task` **[صحيح]** منفصلة في الدور نفسه لتعمل بالتوازي.
- B) نفذ كل المحاور تسلسليًا دائمًا.
- C) اجعل synthesis يبحث بنفسه في كل شيء.
- D) استخدم `/memory`.

لماذا A: عندما تكون المحاور مستقلة، تشغيل الوكلاء الفرعيين بالتوازي يقلل الزمن ويحافظ على فصل السياقات.

السؤال 6 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: synthesis يفقد معلومات مهمة لأنها تقع في منتصف مدخلات طويلة جدًا. ما أفضل معالجة؟

ما أفضل معالجة؟

- A) ضع ملخص key findings **[صحيح]** في البداية، ونظم التفاصيل بعناوين واضحة.
- B) ضع كل شيء في الوسط.
- C) احذف المصادر.
- D) استخدم ترتيبًا عشوائيًا لكل مهمة.

لماذا A: وضع النتائج الأساسية في البداية يستفيد من موثوقية معالجة بداية السياق، والعناوين تساعد النموذج على التنقل داخل المحتوى الطويل.

السؤال 7 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: الوكلاء الفرعيون يرجعون 155 يعمل جيدًا تحت synthesis من محتوى خام وسلاسل تفكير، لكن 50K tokens. ما الحل الأكثر فعالية؟

ما الحل الأكثر فعالية؟

- A) اجعل الوكلاء upstream **[صحيح]** يرجعون بيانات منظمة ومختصرة: حقائق، اقتباسات، درجات صلة، ومصادر.
- B) زد حجم السياق فقط.
- C) اجعل synthesis يقرأ كل شيء كما هو.
- D) ألغِ الوكلاء الفرعيين.

لماذا A: تقليل الحجم من المصدر أفضل من تلخيص لاحق قد يفقد التفاصيل. الوكلاء يجب أن يرجعوا البيانات المهمة لا تفريغات ضخمة.

السؤال 8 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: يحتاج أحيانًا للتحقق من تاريخ أو رقم بسيط، والعودة إلى المنسق تضيف حلقات كثيرة. 85% من synthesis التحققات بسيطة و15% تتطلب بحثًا عميقًا. ما أفضل تصميم؟

ما أفضل تصميم؟

- A) أعطِ synthesis كل أدوات البحث.
- B) أعطِ synthesis أداة محدودة `verify_fact` للتحققات البسيطة، ووجه التحققات المعقدة عبر المنسق. **[صحيح]**
- C) امنع كل التحقق.
- D) اجعل البحث يحدث مرة واحدة فقط ولا يُراجع.

لماذا B: هذا يقلل الحلقات للتأكدات البسيطة، مع الحفاظ على التحكم المركزي في الأبحاث المعقدة.

السؤال 9 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: المنسق يفوض لوكيل فرعي دون تحديد صيغة الإخراج، فتأتي النتائج كنص طويل يصعب دمجه. ما الأفضل؟

ما الأفضل؟

- A) اطلب مخرجات منظمة تحتوي على claims، مصادر، ثقة، وفجوات تغطية. **[صحيح]**
- B) اطلب "be concise" فقط.
- C) امنع الوكيل من استخدام الأدوات.
- D) اجعل الوكيل يرسل raw HTML.

لماذا A: الصيغة المنظمة تجعل الدمج والتحقق والحفاظ على المصدرية أسهل وأكثر موثوقية.

السؤال 10 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: لديك مصدران يقدمان أرقامًا مختلفة عن نفس الظاهرة. ما الإجراء الصحيح؟

ما الإجراء الصحيح؟

- A) اختر الرقم الأكبر.
- B) اختر الرقم الأحدث فقط دون تفسير.
- C) احتفظ بالقيمتين مع المصدر والتاريخ والمنهجية، ووضح سبب الاختلاف المحتمل. **[صحيح]**
- D) احذف الادعاء كله.

لماذا C: البيانات المتعارضة يجب أن تُعرض مع الإسناد والسياق. قد يكون الاختلاف بسبب المنهجية أو الفترة الزمنية.

السؤال 11 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: تريد أن يستطيع النظام استئناف العمل بعد crash. ما أفضل نمط؟

ما أفضل نمط؟

- A) اجعل كل وكيل يصدر حالته المنظمة إلى ملفات مع manifest **[صحيح]** يقرأه المنسق عند الاستئناف.
- B) اعتمد على ذاكرة النموذج فقط.
- C) أعد تنفيذ كل شيء من الصفر دائمًا.
- D) احذف النتائج الجزئية.

لماذا A: استمرار الحالة المنظمة يسمح باستئناف موثوق، ومعرفة ما اكتمل وما لا يزال قيد التنفيذ.

السؤال 12 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: المنسق يختار وكلاء فرعيين كثيرين جدًا لمهمة بسيطة، مما يزيد التعقيد والوقت. ما السبب المرجح؟

ما السبب المرجح؟

- A) التفكيك مفرط الضيق أو غير مناسب لحجم المهمة. **[صحيح]**
- B) يجب زيادة عدد الأدوات.
- C) يجب حذف `Task`.
- D) يجب إلغاء كل القيود.

لماذا A: التفكيك يجب أن يناسب المهمة. استخدام وكلاء أكثر من اللازم يزيد التنسيق والتكلفة دون فائدة.

السؤال 13 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: تُظهر مراقبة الإنتاج أن جودة التركيب غير متسقة. عندما تكون النتائج المجمعة نحو 75 K tokens، يستشهد وكيل التركيب reliably بالمعلومات الموجودة في أول 15 K tokens (عناوين/مقتطفات بحث الويب) وآخر 10 K tokens، لكنه غالبًا يفوّت نتائج حرجة في وسط الـ 50 K tokens، حتى عندما تجيب مباشرة عن سؤال البحث. كيف ينبغي إعادة هيكلة المدخلات المجمعة؟

كيف ينبغي إعادة هيكلة المدخلات المجمعة؟

- A) تلخيص كل مخرجات الوكلاء الفرعيين إلى أقل من 20 K tokens قبل التجميع حتى يبقى المحتوى ضمن نطاق المعالجة الموثوقة للنموذج.
- B) بث نتائج الوكلاء الفرعيين إلى وكيل التركيب تدريجيًا، مع معالجة نتائج بحث الويب أولًا حتى الاكتمال، ثم إضافة نتائج تحليل المستندات.
- C) وضع ملخص للنتائج الأساسية في بداية المدخلات المجمعة، وتنظيم النتائج التفصيلية بعناوين أقسام واضحة لتسهيل التنقل. **[CORRECT]**
- D) تنفيذ تدوير يبدّل أي نتائج وكيل فرعي تظهر أولًا عبر مهام البحث، لضمان حصول كلا المصدرين على تموضع علوي متساوٍ بمرور الوقت.

لماذا C: وضع ملخص للنتائج الأساسية في البداية يستفيد من أثر الأولوية، بحيث تكون المعلومات الحرجة في الموضع الأكثر موثوقية للمعالجة. كما أن إضافة عناوين أقسام واضحة تساعد النموذج على التنقل والانتباه إلى محتوى منتصف المدخلات، مما يخفف مباشرة من ظاهرة "الضياع في الوسط".

السؤال 14 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: في الاختبار، يبلغ مجموع مخرجات وكيل بحث الويب (85K tokens تشمل محتوى الصفحات) ووكيل تحليل المستندات (70K tokens تشمل سلاسل التفكير) نحو 155K tokens، لكن وكيل التركيب يعمل بأفضل أداء عندما تكون المدخلات أقل من 50K tokens. أي حل هو الأكثر فعالية؟

أي حل هو الأكثر فعالية؟

- A) تعديل الوكلاء السابقين upstream agents بدل (حقائق أساسية، اقتباسات، درجات صلة) لإرجاع بيانات منظمة المحتوى المطوّل والتفكير المطوّل. **[CORRECT]**
- B) إضافة وكيل تلخيص وسيط يختصر النتائج قبل تمريرها إلى التركيب.
- C) جعل وكيل التركيب يعالج النتائج على دفعات متتابعة مع الحفاظ على الحالة بين الاستدعاءات.
- D) تخزين النتائج في قاعدة بيانات vector database وإعطاء وكيل التركيب أدوات بحث للاستعلام أثناء عمله.

لماذا A: تعديل الوكلاء السابقين لإرجاع بيانات منظمة يعالج السبب الجذري بتقليل حجم الرموز من المصدر مع الحفاظ على المعلومات الأساسية. هذا يتجنب تمرير محتوى صفحات ضخم وسلاسل تفكير تزيد الرموز دون تحسين خطوة التركيب.

السؤال 15 (السيناريو: نظام بحث متعدد الوكلاء)

الموقف: في الاختبار، تلاحظ أن وكيل التركيب يحتاج غالبًا إلى التحقق من ادعاءات محددة أثناء دمج النتائج. حاليًا، عند الحاجة إلى التحقق، يعيد وكيل التركيب التحكم إلى المنسق، فيستدعي المنسق وكيل بحث الويب ثم يعيد استدعاء التركيب مع النتائج. هذا يضيف 2–3 حلقات إضافية لكل مهمة ويزيد زمن الاستجابة بنسبة 40%. يُظهر تقييمك أن 85% من هذه التحققات هي فحوصات حقائق بسيطة (تواريخ، أسماء، إحصاءات)، وأن 15% تتطلب بحثًا أعمق. أي نهج يقلل العبء بأكبر فعالية مع الحفاظ على موثوقية النظام؟

أي نهج هو الأكثر فعالية؟

- A) إعطاء وكيل التركيب وصولًا إلى كل أدوات بحث الويب حتى يتعامل مباشرة مع أي حاجة للتحقق دون حلقات المنسق.
- B) جعل وكيل التركيب يجمع كل احتياجات التحقق ويعيدها كدفعة إلى المنسق في النهاية، ثم يرسلها المنسق كلها إلى وكيل بحث الويب مرة واحدة.
- C) جعل وكيل بحث الويب يخزّن سياقًا إضافيًا حول كل مصدر أثناء البحث الأولي تحسبًا لاحتياج التركيب إلى التحقق.
- D) إعطاء وكيل التركيب أداة محدودة النطاق `verify_fact` للفحوصات البسيطة، مع توجيه التحققات المعقدة عبر المنسق إلى وكيل بحث الويب. **[CORRECT]**

لماذا D: أداة `verify_fact` محدودة النطاق تقلل الحلقات لمعظم التحققات البسيطة، مع الحفاظ على تحكم المنسق في الحالات المعقدة أو المفتوحة. هذا يوازن بين تقليل زمن الاستجابة والحفاظ على بنية موثوقة قابلة للملاحظة.

السؤال 16 (السيناريو: Claude Code المستمر للتكامل)

الموقف: يشغّل خط CI لديك واجهة Claude Code CLI في وضع `--print` باستخدام CLAUDE.md لتوفير سياق المشروع لمراجعة الكود، ويجد المطورون عمومًا أن المراجعات ذات قيمة. لكنهم يذكرون أن دمج النتائج في PR

صعب لأن المخرجات نص حر: أحيانًا تكون المشكلات في فقرات، وأحيانًا في قوائم، وأحيانًا بلا أرقام أسطر. تريد نشر تعليقات inline تلقائيًا في GitHub. ما التغيير الأكثر فعالية؟

ما التغيير الأكثر فعالية؟

- A) اطلب من Claude "تنسيق النتائج باستمرار" داخل المطالبة، مع أمثلة لتنسيق جيد.
- B) استخدم `--output-format json` مع `--json-schema` يفرض حقولًا مثل file وline وseverity وmessage. **[CORRECT]**
- C) شغّل خطوة post-processing تستخدم regex لاستخراج أسماء الملفات وأرقام الأسطر من النص الحر.
- D) اطلب من المطورين قراءة تقرير Claude كاملًا بدل التعليقات inline.

لماذا B: الإخراج المنظم مع JSON Schema يجعل النتائج قابلة للمعالجة آليًا وموثوقة للنشر كتعليقات inline. الاعتماد على نص حر أو regex هش وغير كافٍ لتكامل CI مستقر.

السؤال 17 (السيناريو: Claude Code للتكامل المستمر)

الموقف: يستخدم فريقك Claude Code مثل لتوليد اقتراحات كود، لكنك تلاحظ نمطًا: المشكلات غير الواضحة — تحسينات أداء تكسر حالات حدية أو cleanups لا تُكتشف إلا عندما يراجعها Claude نفسه تغيّر السلوك دون قصد — في الجلسة نفسها التي اقترحت التغيير. تريد تحسين اكتشاف هذه المشكلات. ما أفضل تصميم للمراجعة؟

ما أفضل تصميم للمراجعة؟

- A) اجعل Claude يراجع الكود في الجلسة نفسها التي ولّدت الاقتراح حتى يعرف النية الأصلية.
- B) استخدم Claude المولّد مثيلًا مستقلًا أو جلسة منفصلة من Claude لمراجعة الكود المولّد. **[CORRECT]**
- C) اطلب من Claude إضافة تعليقات أكثر في الكود تشرح قراراته.
- D) شغّل الاختبارات فقط، لأن المراجعة المستقلة مكررة.

لماذا B: الجلسة نفسها قد تكون منحازة إلى قراراتها وسياقها السابق. مثيل مستقل يراجع الكود بزاوية جديدة ويكون أكثر قدرة على تحدي الافتراضات واكتشاف الأخطاء غير الواضحة.

السؤال 18 (السيناريو: Claude Code للتكامل المستمر)

الموقف: مكوّن مراجعة الكود لديك تكراري: يحلل Claude الملف المتغير، ثم قد يطلب ملفات ذات صلة (imports، base classes، tests) عبر استدعاءات أدوات لفهم السياق قبل تقديم الملاحظات النهائية. تطبيقك يحاول نقل هذه العملية إلى Message Batches API لتقليل التكلفة. ما المشكلة الأساسية؟

ما المشكلة الأساسية؟

- A) Message Batches API لا تدعم استدعاء الأدوات متعدد الأدوار داخل الطلب الواحد. **[CORRECT]**
- B) Message Batches API لا تدعم JSON output.
- C) Message Batches API لا يمكنها معالجة ملفات الكود.
- D) Message Batches API لا تسمح باستخدام `custom_id`.

لماذا A: Batch API مناسب لطلبات مستقلة ذات استجابة واحدة، لكنه لا يدعم حلقات تفاعلية متعددة الأدوار حيث يطلب النموذج أدوات ثم يتلقى نتائجها ثم يتابع التفكير. هذا النمط يحتاج API متزامنة أو حلقة وكيل خارجية.

السؤال 19 (السيناريو: Claude Code المستمر للتكامل)

الموقف: يشغّل نظام CI/CD لديك ثلاث تحليلات مبنية على Claude: (1) تمنع PR فحوصات أسلوب سريعة على كل الدمج حتى تكتمل، (2) تدقيقات أمنية أسبوعية شاملة لقاعدة الكود كلها، و(3) توليد حالات اختبار ليلية للملفات التي تغيرت خلال اليوم. تريد تقليل التكلفة دون التأثير على سير المطورين. أي استراتيجية API هي الأنسب؟

أي استراتيجية API هي الأنسب؟

- A) استخدم Message Batches API لكل التحليلات الثلاثة لتقليل التكلفة.
- B) استخدم Message Batches API للتدقيقات الأسبوعية والتوليد الليلي للاختبارات. **[CORRECT]**
- C) استخدم الاستدعاءات المتزامنة لكل شيء لتجنب اختلافات زمن المعالجة.
- D) استخدم Batch API للفحوصات التي تمنع الدمج، والاستدعاءات المتزامنة للوظائف الليلية.

لماذا B: الفحوصات التي تمنع الدمج تحتاج زمن استجابة متوقعًا وسريعًا. أما التدقيقات الأسبوعية والتوليد الليلي فليست تفاعلية ويمكنها الاستفادة من توفير Batch API رغم زمن معالجة قد يصل إلى 24 ساعة.

السؤال 20 (السيناريو: Claude Code المستمر للتكامل)

الموقف: تجد مراجعاتك الآلية مشكلات حقيقية، لكن المطورين يقولون إن الملاحظات غير قابلة للتنفيذ. تتضمن النتائج عبارات مثل "منطق توجيه التذاكر معقد" أو "احتمال null pointer". ما أفضل تحسين؟

ما أفضل تحسين؟

- A) اطلب من Claude إنتاج نتائج منظمة تشمل الموقع، الوصف، سبب المشكلة، الأثر، واقتراح إصلاح محدد. **[CORRECT]**
- B) اطلب من Claude تقليل عدد النتائج إلى أعلى 3 فقط.
- C) أضف sentiment analysis لتعليقات المطورين.
- D) اجعل Claude يستخدم لغة ألطف حتى تكون الملاحظات مقبولة أكثر.

لماذا A: المشكلة ليست النبرة أو العدد فقط، بل نقص القابلية للتنفيذ. يجب أن تتضمن كل نتيجة موقعًا واضحًا، وسببًا، وأثرًا، وإصلاحًا مقترحًا حتى يستطيع المطور التعامل معها بسرعة.

السؤال 21 (السيناريو: Claude Code المستمر للتكامل)

الموقف: يتضمن خط CI لديك وضعين لمراجعة الكود باستخدام Claude: hook يمنع دمج PR قبل الدمج حتى يكتمل، و"تحليل عميق" يعمل ليلًا، يستعلم عن اكتمال الدفعة، وينشر اقتراحات مفصلة في الصباح. أي وصف صحيح؟

أي وصف صحيح؟

- A) يجب تشغيل الوضعين عبر Message Batches API لتقليل التكلفة.
- B) يجب تشغيل الوضعين باستدعاءات متزامنة لأن كلاهما جزء من CI.
- C) استخدم الاستدعاءات المتزامنة للـ pre-merge hook، وMessage Batches API للتحليل الليلي العميق. **[CORRECT]**

- D) استخدم Message Batches API للـ pre-merge hook لأنه أقصر، والاستدعاءات المتزامنة للتحليل العميق لأنه أطول.

لماذا C: أي شيء يمنع الدمج يحتاج استجابة فورية تقريبًا ولا يناسب Batch API. أما التحليل الليلي فيناسب المعالجة غير المتزامنة لأن النتيجة مطلوبة لاحقًا.

السؤال 22 (السيناريو: Claude Code المستمر للتكامل)

الموقف: تحلل المراجعة الآلية التعليقات و docstrings. تطلب المطالبة الحالية من Claude “التحقق من أن التعليقات أو تعليقات وصفية بسيطة، أو تعليقات لا تغطي TODO markers، دقيقة ومحدثة”. غالبًا ترفع النتائج أنماطًا مقبولة مثل كل التفاصيل. كيف تحسن الدقة؟

كيف تحسن الدقة؟

- A) أضف معايير صريحة لما يجب اعتباره تعليقًا غير صحيح، وما لا يجب رفعه. **[CORRECT]**
- B) اطلب من Claude “أكثر تحفظًا أن يكون”.
- C) عطّل كل مراجعة التعليقات لأنها تسبب false positives.
- D) استخدم نموذجًا أكبر فقط.

لماذا A: التعليمات المبهمة تؤدي إلى تفسيرات واسعة. المعايير الصريحة، مثل رفع التعليق فقط عندما يناقض سلوك الكود أو يشير إلى دالة غير موجودة، تقلل الإيجابيات الكاذبة مباشرة.

السؤال 23 (السيناريو: Claude Code المستمر للتكامل)

الموقف: يُظهر نظام المراجعة الآلي تقييمات شدة غير متسقة؛ مشكلات متشابهة مثل مخاطر null pointer تُقيّم في بعض “medium” و PRs في أخرى “critical”. تُظهر استطلاعات المطورين انخفاض الثقة لأنهم لا يعرفون كيف يفسرون الشدة. ما أفضل تحسين؟

ما أفضل تحسين؟

- A) عرّف معايير شدة صريحة مع أمثلة لكل مستوى. **[CORRECT]**
- B) أزل كل تقييمات الشدة.
- C) استخدم رقم ثقة فقط بدل الشدة.
- D) اطلب من Claude “استخدم الشدة باستمرار”.

لماذا A: معايير الشدة الواضحة مع أمثلة تقلل التذبذب وتوفر أساسًا مشتركًا لتفسير النتائج. التعليمات العامة لا تكفي لتوحيد التصنيف.

السؤال 24 (السيناريو: Claude Code المستمر للتكامل)

الموقف: تولد المراجعة الآلية اقتراحات حالات اختبار لكل PR. عند مراجعة PR يضيف تتبع إكمال الدورات، يقترح Claude عشر حالات اختبار، لكن ملاحظات المطورين تُظهر أن 6 سيناريوهات مكررة مغطاة بالفعل في ملفات اختبار موجودة. ما أفضل تحسين؟

ما أفضل تحسين؟

- A) واطلب اقتراح الاختبارات الجديدة غير المغطاة Claude ضمّن ملفات الاختبار الحالية ذات الصلة في سياق فقط. **[CORRECT]**
- B) اطلب من Claude توليد عدد أقل من الاختبارات.
- C) احذف كل الاختبارات القديمة قبل التوليد.
- D) اجعل Claude يقترح اختبارات end-to-end فقط.

لماذا A: المشكلة هي أن Claude لا يرى التغطية الموجودة. توفير ملفات الاختبار الحالية يسمح له بتجنب التكرار واقتراح فجوات حقيقية.

السؤال 25 (السيناريو: Claude Code المستمر للتكامل)

الموقف: بعد مراجعة آلية أولية تحدد 12 نتيجة، يدفع المطور commits جديدة لمعالجة المشكلات. إعادة تشغيل المراجعة تنتج 8 نتائج، لكن المطورين يقولون إن 5 منها تعليقات مكررة سابقة على كود لم يعد موجودًا أو تم إصلاحه جزئيًا. ما أفضل طريقة لتقليل التكرار؟

ما أفضل طريقة؟

- A) ضمّن نتائج المراجعة السابقة وسياق التغييرات الجديدة، واطلب من Claude الإبلاغ فقط عن المشكلات الجديدة أو التي لا تزال غير محلولة. **[CORRECT]**
- B) امنع Claude من رؤية أي مراجعات سابقة حتى يكون مستقلًا.
- C) لا تعيد تشغيل المراجعة بعد commits جديدة.
- D) احذف كل التعليقات القديمة قبل تشغيل المراجعة.

لماذا A: إدراج النتائج السابقة يساعد Claude على تمييز ما تم الإبلاغ عنه سابقًا وما تم إصلاحه وما لا يزال قائمًا. هذا يقلل التعليقات المكررة ويركز على المشكلات الجديدة أو غير المعالجة.

السؤال 26 (السيناريو: Claude Code المستمر للتكامل)

الموقف: يشغّل سكربت pipeline الأمر `claude "Analyze this pull request for security issues"`، ينتظر إدخالًا تفاعليًا. ما النهج الصحيح Claude Code يظل معلقًا إلى أجل غير مسمى. تُظهر السجلات أن job لكن لتشغيل Claude Code في CI؟

ما النهج الصحيح؟

- A) استخدم `claude -p "Analyze this pull request for security issues"` أو `--print`. **[CORRECT]**
- B) استخدم `/compact` قبل الأمر.
- C) استخدم `--resume` دائمًا داخل CI.
- D) أضف timeout قصير فقط واتركه يفشل.

لماذا A: العلم `-p` أو `--print` يشغّل Claude Code في وضع غير تفاعلي، فيعالج المطالبة ويطبع النتيجة ويخرج. هذا هو النمط المناسب لـ CI.

السؤال 27 (السيناريو: Claude Code المستمر للتكامل)

الموقف: يغيّر PR غير متسقة: ملاحظات مفصلة لبعض الملفات، وتعليقات سطحية لأخرى، وأخطاء واضحة فائتة، وتغذية راجعة متناقضة. أربعة عشر ملفًا في وحدة تتبع المخزون. مراجعة واحدة تمر على كل الملفات معًا تنتج نتائج غير كيف ينبغي إعادة هيكلة المراجعة؟

كيف ينبغي إعادة هيكلة المراجعة؟

- A) تقسيمها إلى تمريرات مركزة: تحليل كل ملف منفردًا للمشكلات المحلية، ثم تشغيل تمريرة تكامل منفصلة لتدفقات البيانات بين الملفات. **[CORRECT]**
- B) إجبار المطورين على تقسيم PRs الكبيرة إلى 3–4 ملفات.
- C) التبديل إلى نموذج بسياق أكبر ومراجعة كل الملفات في تمريرة واحدة.
- D) تشغيل ثلاث مراجعات كاملة مستقلة والإبلاغ فقط عن المشكلات التي تظهر في اثنتين منها على الأقل.

لماذا A: التمريرات المركزة تعالج السبب الجذري، وهو تشتت الانتباه عند معالجة ملفات كثيرة دفعة واحدة. التحليل لكل ملف يضمن عمقًا متسقًا، وتمرير التكامل يلتقط المشكلات بين الملفات.

السؤال 28 (السيناريو: Claude Code المستمر للتكامل)

الموقف: متوسط المراجعة الآلية لديك 15 نتيجة لكل PR، ويبلغ المطورون عن معدل false positive عنق قدره 40%. لمعرفة هل هي حقيقية. يحتوي Claude الزجاجة هو وقت التحقيق: يجب على المطورين فتح كل نتيجة وقراءة منطق بالفعل على قواعد شاملة للأنماط المقبولة، ورفض أصحاب المصلحة أي نهج يرشح النتائج قبل أن يراها CLAUDE.md المطورون. ما التغيير الذي يعالج وقت التحقيق بأفضل شكل؟

ما التغيير الذي يعالج وقت التحقيق بأفضل شكل؟

- A) اطلب من Claude تضمين المبرر وتقدير الثقة مباشرة داخل كل نتيجة. **[CORRECT]**
- B) أضف post-processor يحلل أنماط النتائج ويخفي تلقائيًا النتائج التي تطابق توقيعات false positives تاريخية.
- C) صنّف النتائج إلى “blocking issues” مقابل “suggestions” مع متطلبات مراجعة مختلفة حسب المستوى.
- D) اجعل Claude يعرض النتائج عالية الثقة فقط، مع ترشيح العلامات غير المؤكدة قبل أن يراها المطورون.

لماذا A: تضمين المبرر والثقة داخل كل نتيجة يقلل وقت التحقيق لأنه يسمح للمطورين بالفرز السريع دون فتح كل نتيجة. كما يلتزم بقيد “عدم الترشيح” لأن كل النتائج تبقى مرئية مع تسريع قرار المطور.

السؤال 29 (السيناريو: Claude Code المستمر للتكامل)

الموقف: يُظهر تحليل مراجعة الكود الآلية اختلافات كبيرة في معدلات false positives حسب الفئة: نتائج الأمن/الصحة فيها correctness 8% false positives، الأداء 18%، الأسلوب/التسمية 52%، والتوثيق 48%. تُظهر استطلاعات المطورين تراجعًا في الثقة؛ كثيرون يبدأون بتجاهل النتائج دون قراءتها لأن “نصفها خطأ”. الفئات ذات false positives العالية تضعف الثقة حتى في الفئات الدقيقة. أي نهج يستعيد ثقة المطورين بأفضل شكل مع تحسين النظام؟

أي نهج يستعيد ثقة المطورين بأفضل شكل؟

- A) تعطيل الفئات ذات false positives العالية مؤقتًا (الأسلوب، التسمية، التوثيق)، والإبقاء فقط على الفئات عالية الدقة أثناء تحسين المطالبات. **[CORRECT]**
- B) إبقاء كل الفئات مفعلة مع عرض درجات ثقة لكل نتيجة حتى يقرر المطورون ما يحققون فيه.

- C) إبقاء كل الفئات مفعلة وإضافة أمثلة few-shot لكل فئة خلال الأسابيع القادمة لتحسين الدقة.
- D) تطبيق تقليل موحد للصرامة عبر كل الفئات لخفض معدل false positives الإجمالي.

لماذا A: تعطيل الفئات المزعجة مؤقتًا يوقف تآكل الثقة فورًا بإزالة النتائج الصاخبة التي تجعل المطورين يتجاهلون كل شيء، مع الحفاظ على قيمة الفئات عالية الدقة مثل الأمن والصحة. كما يتيح تحسين مطالبات الفئات المشكلة قبل إعادة تفعيلها.

السؤال 30 (السيناريو: Claude Code للتكامل المستمر)

الموقف: يريد فريقك تقليل تكاليف workflow(1) المتزامنة Claude API للتحليل الآلي. حاليًا تدعم استدعاءات ين: (1) تقرير ديون تقنية يُولّد ليلًا للمراجعة في pre-merge فحص مانع يجب أن يكتمل قبل أن يستطيع المطورون الدمج، و(2) لتوفير 50%. كيف ينبغي تقييم هذا الاقتراح؟ Message Batches API صباح اليوم التالي. يقترح مديرك نقل كليهما إلى

كيف ينبغي تقييم هذا الاقتراح؟

- A) انقل كليهما إلى المعالجة بالدفعات مع fallback إلى الاستدعاءات المتزامنة إذا استغرقت الدفعات وقتًا طويلًا.
- B) انقل كلا الـ workflows إلى المعالجة بالدفعات مع polling للتحقق من الاكتمال.
- C) استخدم المعالجة بالدفعات فقط لتقارير الديون التقنية، واحتفظ بالاستدعاءات المتزامنة لفحوصات pre-merge. **[CORRECT]**
- D) احتفظ بالاستدعاءات المتزامنة لكليهما لتجنب مشكلات ترتيب نتائج الدفعات.

لماذا C: قد تستغرق Message Batches API حتى 24 ساعة دون SLA للكمون، وهذا مقبول لتقارير الديون التقنية الليلية، لكنه غير مقبول لفحوصات pre-merge التي ينتظر المطورون. هذا يطابق كل workflow المانعة حيث API المناسب حسب متطلبات زمن الاستجابة.

السيناريو: توليد الكود باستخدام Claude Code

السؤال 31 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: طلبت من Claude Code تنفيذ دالة تحول استجابات API إلى صيغة داخلية موحدة. بعد تكرارين، لا يزال هيكل الإخراج لا يطابق التوقعات: بعض الحقول متداخلة بشكل مختلف، والتواريخ تُنسق بطريقة خاطئة. وصفت المتطلبات نثريًا، لكن Claude يفسرها بشكل مختلف في كل مرة.

ما النهج الأكثر فعالية للتكرار التالي؟

- A) اكتب JSON Schema يصف بنية الإخراج المتوقعة وتحقق من مخرجات Claude مقابله بعد كل تكرار.
- B) قدّم 2–3 أمثلة إدخال/إخراج ملموسة توضح التحويل المتوقع لاستجابات API تمثيلية. **[CORRECT]**
- C) أعد كتابة المتطلبات بدقة تقنية أكبر، مع تحديد خرائط الحقول، وقواعد التداخل، وصيغ التواريخ بالضبط.
- D) اطلب من Claude شرح فهمه الحالي للمتطلبات لتحديد مواضع اختلاف التفسير.

لماذا B: الأمثلة الملموسة للإدخال والإخراج تزيل الغموض الموجود في الوصف النثري لأنها تعرض النتيجة المتوقعة بالضبط. هذا يعالج السبب الجذري مباشرة، وهو سوء تفسير المتطلبات النصية، عبر تقديم أنماط واضحة لبنية الحقول وتنسيق التواريخ.

السؤال 32 (سيناريو: توليد الكود باستخدام Claude Code)

الموقف: تحتاج إلى إضافة Slack كقناة إشعارات جديدة. قاعدة الكود الحالية لديها أنماط واضحة لقنوات البريد الإلكتروني و SMS و push. لكن Slack API تقدم طرق تكامل مختلفة جذريًا: incoming webhooks (بسيطة وأحادية الاتجاه)، أو Slack Apps (ثنائية الاتجاه وتتطلب موافقة)، أو bot tokens (تدعم تأكيد التسليم والتحكم البرمجي). المهمة تقول "أضف دعم Slack" دون تحديد طريقة التكامل أو طلب مزايا متقدمة مثل تتبع التسليم أو workspace.

كيف ينبغي التعامل مع هذه المهمة؟

- A) ابدأ بوضع التنفيذ المباشر باستخدام incoming webhooks لتطابق نمط الإشعارات أحادي الاتجاه الحالي.
- B) انتقل إلى وضع التخطيط لاستكشاف خيارات التكامل وتبعاتها المعمارية، ثم قدم توصية قبل التنفيذ. **[CORRECT]**
- C) ابدأ بوضع التنفيذ المباشر بإنشاء Slack channel class باستخدام الأنماط الحالية، وأجّل قرار طريقة التكامل.
- D) ابدأ بوضع التنفيذ المباشر باستخدام bot-token حتى يكون تأكيد التسليم ممكنًا.

لماذا B: يملك Slack عدة طرق صحيحة، ولكل طريقة تبعات معمارية مختلفة، والمتطلبات غامضة. وضع تكامل Slack Apps و bot tokens و webhooks التخطيط يسمح بتقييم المفاضلات بين والاتفاق على النهج قبل التنفيذ.

السؤال 33 (سيناريو: توليد الكود باستخدام Claude Code)

الموقف: أصبح ملف CLAUDE.md لديك أكثر من 400 سطر، ويحتوي على معايير كتابة الكود، واصطلاحات الاختبار، وإجراءات ترحيل قاعدة البيانات. تريد أن يتبع Claude دائمًا معايير الكود مفصلة، وتعليمات النشر، وقائمة مراجعة PR واصطلاحات الاختبار، لكن يطبق إرشادات migrations و deploy و PR review فقط عند تنفيذ تلك المهام.

ما نهج إعادة الهيكلة الأكثر فعالية؟

- A) انقل كل الإرشادات إلى ملفات Skills منفصلة منظمة حسب نوع workflow، واترك وصفًا مختصرًا للمشروع فقط في CLAUDE.md.
- B) أبقِ كل شيء في CLAUDE.md لكن استخدم صيغة `@import` لتنظيمه في ملفات منفصلة حسب الفئة.
- C) قسّم CLAUDE.md إلى ملفات تحت `.claude/rules/` مع glob patterns مرتبطة بالمسارات بحيث لا تُحمّل كل قاعدة إلا لأنواع الملفات ذات الصلة.
- D) أبقِ المعايير العامة في CLAUDE.md، وأنشئ Skills للإرشادات الخاصة بالـ workflows مثل PR review و migrations و deploy مع trigger keywords. **[CORRECT]**

لماذا D: محتوى CLAUDE.md يُحمّل في كل جلسة، لذلك يناسب المعايير العامة التي يجب أن تطبق دائمًا. أما Skills فتُستدعى عند الحاجة عندما يكتشف Claude كلمات trigger مناسبة، وهذا مثالي للإرشادات الخاصة بسير عمل معين مثل مراجعة PR أو النشر أو migrations.

السؤال 34 (سيناريو: توليد الكود باستخدام Claude Code)

الموقف: كُلّفت بإعادة هيكلة تطبيق monolithic إلى microservices. يؤثر هذا على عشرات الملفات ويتطلب قرارات حول حدود الخدمات واعتماديات الوحدات.

أي نهج ينبغي اختياره؟

- A) انتقل إلى وضع التخطيط لاستكشاف قاعدة الكود وفهم الاعتماديات وتصميم نهج التنفيذ قبل إجراء التغييرات. **[CORRECT]**
- B) ابدأ بوضع التنفيذ المباشر، وانتقل إلى التخطيط فقط بعد مواجهة تعقيد غير متوقع أثناء التنفيذ.
- C) ابدأ بوضع التنفيذ المباشر ونفذ تغييرات تدريجية، ودع التنفيذ يكشف حدود الخدمات الطبيعية.
- D) استخدم التنفيذ المباشر مع تعليمات مسبقة مفصلة تحدد بنية كل خدمة.

لماذا A: وضع التخطيط هو الاستراتيجية المناسبة لإعادة هيكلة معمارية معقدة مثل تقسيم monolith؛ فهو يسمح باستكشاف آمن واتخاذ قرارات مستنيرة حول الحدود قبل الالتزام بتغييرات مكلفة عبر ملفات كثيرة.

السؤال 35 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: أنشأ فريقك مهارة `/analyze-codebase` تنفذ تحليلًا عميقًا للكود: فحص الاعتماديات، وإحصاءات تغطية الاختبارات، ومقاييس جودة الكود. بعد تشغيل الأمر، يذكر أعضاء الفريق أن Claude يصبح أقل استجابة في الجلسة ويفقد سياق المهمة الأصلية.

كيف تصلح ذلك بأفضل طريقة مع الحفاظ على قدرات التحليل الكاملة؟

- A) أضف `context: fork` في frontmatter الخاص بالمهارة لتشغيل التحليل في سياق وكيل فرعي معزول. **[CORRECT]**
- B) أضف `model: haiku` في frontmatter لاستخدام نموذج أسرع وأرخص للتحليل.
- C) قسّم المهارة إلى ثلاث مهارات أصغر، بحيث تنتج كل واحدة مخرجات أقل.
- D) أضف تعليمات إلى المهارة لضغط كل النتائج في ملخص قصير قبل عرضها.

لماذا A: `context: fork` يشغل التحليل في سياق وكيل فرعي معزول، بحيث لا تلوث المخرجات الكبيرة نافذة سياق الجلسة الرئيسية Claude ولا يفقد المهمة الأصلية. هذا يحافظ على قدرات التحليل الكاملة مع إبقاء الجلسة الرئيسية مستجيبة.

السؤال 36 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: يستخدم فريقك مهارة `/commit` في `.claude/skills/commit/SKILL.md`. يريد أحد المطورين، دون التأثير على زملائه (وفحوصات إضافية، صيغة مختلفة لرسالة commit) تخصيصها لسير عمله الشخصي.

بماذا توصي؟

- A) إنشاء نسخة شخصية تحت `~/.claude/skills/` باسم مختلف، مثل `/my-commit`. **[CORRECT]**
- B) إضافة منطق شرطي حسب اسم المستخدم في frontmatter الخاص بمهارة المشروع.
- C) إنشاء نسخة شخصية في `~/.claude/skills/commit/SKILL.md` بالاسم نفسه.
- D) تعيين `override: true` في frontmatter الخاص بالمهارة الشخصية لجعلها أعلى أولوية من نسخة المشروع.

لماذا A: المهارات الشخصية لها أولوية على مهارات المشروع عند استخدام الاسم نفسه، لذا فإن إعادة استخدام الاسم `commit` سيؤدي إلى حجب مهارة الفريق بصمت لهذا المطور وحده — لن يتلقى التحديثات كلما حسّن الفريق مهارة `/commit`، وسيحتاج إلى تذكّر أنه يشغّل مهارة مختلفة تحت نفس الأمر. تسمية النسخة الشخصية `/my-commit` تتجنب هذا التعارض تمامًا: يستمر المطور في استخدام مهارة `/commit` التي يصونها الفريق، ويحصل على مهارة منفصلة وواضحة الاسم لسير عمله الشخصي، دون خطر الخلط بينهما أو فقدان تحديثات الفريق.

السؤال 37 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: يستخدم فريقك Claude Code منذ أشهر. مؤخرًا، يقول ثلاثة مطورين إن Claude يتبع إرشاد "always include comprehensive error handling"، لكن مطورًا رابعًا انضم حديثًا يقول إن Claude في الأربعة يعملون لا يتبعه. الريبو نفسه ولديهم أحدث نسخة من الكود.

ما السبب الأكثر احتمالًا وما الإصلاح؟

- A) الإرشاد موجود في ملفات `~/.claude/CLAUDE.md` الخاصة بالمطورين الأصليين على مستوى المستخدم، وليس في ملف المشروع `.claude/CLAUDE.md`. انقل التعليمة إلى ملف مستوى المشروع حتى يحصل عليها كل أعضاء الفريق. **[CORRECT]**
- B) ملف `~/.claude/CLAUDE.md` لدى المطور الجديد يحتوي تعليمات متعارضة تتجاوز إعدادات المشروع؛ يجب حذف القسم المتعارض.
- C) Claude Code يتعلم تفضيلات كل مستخدم بمرور الوقت؛ يجب على المطور الجديد تكرار المتطلب حتى يتذكره" Claude.
- D) Claude Code يخزن CLAUDE.md مؤقتًا بعد أول قراءة؛ المطورون الأصليون يستخدمون نسخًا مخزنة. يجب على الجميع مسح cache الخاص بـ Claude Code.

لماذا A: إذا أضيف الإرشاد فقط إلى إعدادات المستخدم الخاصة بالمطورين الأصليين ولم يُضف إلى `.claude/CLAUDE.md` على مستوى المشروع، فلن يحصل عليه أعضاء الفريق الجدد. نقله إلى إعداد المشروع يضمن أن كل الأعضاء الحاليين والمستقبليين يحصلون عليه تلقائيًا.

السؤال 38 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: تجد أن تضمين 2–3 أمثلة كاملة لتنفيذ API endpoints كسياق يحسن الاتساق كثيرًا عند توليد endpoints جديدة، وليس عند التصحيح أو مراجعة الكود أو الأعمال الأخرى في مجلد API.

ما نهج الإعداد الأكثر فعالية؟

- A) أضف أمثلة endpoints وتوثيق الأنماط إلى CLAUDE.md الخاص بالمشروع حتى تكون متاحة دائمًا.
- B) أشر يدويًا إلى أمثلة endpoints في كل طلب توليد عبر نسخ الكود إلى المطالبة.
- C) أعد قواعد خاصة بالمسارات في `.claude/rules/api/` تتضمن أمثلة endpoints وتُفعّل عند العمل في مجلد API.
- D) أنشئ مهارة تشير إلى أمثلة endpoints وتحتوي تعليمات اتباع النمط، وتُستدعى عند الحاجة عبر slash command. **[CORRECT]**

لماذا D: المهارة التي تُستدعى عند الطلب تحمل سياق الأمثلة فقط عند توليد endpoints جديدة، وليس أثناء مهام غير ذات صلة مثل التصحيح أو المراجعة. هذا يحافظ على نظافة السياق الرئيسي مع الحفاظ على جودة عالية عند التوليد.

السؤال 39 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: أنشأ فريقك مهارة `/migration` تولد ملفات database migration. تأخذ اسم migration عبر `$ARGUMENTS`. في الإنتاج لاحظت ثلاث مشكلات: (1) كثيرًا ما يشغل المطورون المهارة دون arguments، مما ينتج

من محادثات سابقة غير ذات صلة، و(3) شغّل مطور schema ملفات بأسماء سيئة، (2) تستخدم المهارة أحيانًا تفاصيل تنظيفًا اختباريًا مدمرًا بالخطأ لأن المهارة كان لديها وصول واسع للأدوات.

أي نهج إعداد يصلح المشكلات الثلاث كلها؟

- A) استخدم positional parameters بدل `2$` و `1$` مثل `$ARGUMENTS` لفرض مدخلات محددة، وأضف مراجع frontmatter صريحة عبر صيغة `@` للتحكم في السياق، وأضف وصف schema يحذر من العمليات المدمرة.
- B) أضف `argument-hint` في frontmatter لطلب المعاملات المطلوبة، واستخدم `context: fork` لعزل التنفيذ، وقيّد `allowed-tools` إلى عمليات كتابة الملفات. **[CORRECT]**
- C) قسّمها إلى مهارتين `/migration-apply` و `/migration-create`، وأضف تعليمات تحقق لطلب اسم migration عند غيابه، واستخدم نطاقات مختلفة لـ `allowed-tools`.
- D) أضف تعليمات تحقق في SKILL.md لضمان أن `$ARGUMENTS` اسم صالح، وأضف مطالبات لتجاهل سياق المحادثات السابقة، واسرد العمليات المحظورة.

لماذا B: هذا يستخدم ثلاث مزايا إعداد مختلفة لمعالجة كل مشكلة: `argument-hint` يحسن إدخال المعاملات ويقلل غيابها، و `context: fork` يمنع تسرب سياق المحادثات السابقة، و `allowed-tools` يقيّد المهارة بعمليات كتابة آمنة للملفات، مما يمنع الإجراءات المدمرة.

السؤال 40 (سيناريو: توليد الكود باستخدام Claude Code)

الموقف: تحتوي قاعدة الكود لديك على مناطق ذات اصطلاحات مختلفة: مكونات React تستخدم functional style مع hooks، وAPI handlers تستخدم async/await مع معالجة أخطاء محددة، ونماذج قاعدة البيانات تتبع repository pattern. ملفات الاختبار موزعة عبر قاعدة الكود بجوار الكود المختبر (مثل `Button.test.tsx` بجوار `Button.tsx`)، وتريد أن تتبع كل الاختبارات الاصطلاحات نفسها بغض النظر عن الموقع.

ما الطريقة الأكثر دعمًا لضمان أن Claude يطبق الاصطلاحات الصحيحة تلقائيًا عند توليد الكود؟

- A) ضع كل الاصطلاحات في CLAUDE.md الجذري تحت عناوين لكل منطقة واعتمد على Claude ليستنتج القسم المناسب.
- B) أنشئ Skills في `.claude/skills/` لكل نوع كود، وضع الاصطلاحات داخل كل SKILL.md.
- C) ضع ملف CLAUDE.md منفصلًا في كل مجلد فرعي يحتوي اصطلاحات تلك المنطقة.
- D) أنشئ ملفات قواعد تحت `.claude/rules/` مع YAML frontmatter يحدد glob patterns لتطبيق الاصطلاحات شرطيًا بناءً على مسارات الملفات. **[CORRECT]**

لماذا D: ملفات `.claude/rules/` مع YAML frontmatter وglob patterns مثل `**/*.test.tsx` تتيح تطبيقًا حتميًا للاصطلاحات بناءً على المسار، بغض النظر عن بنية المجلدات. هذا هو النهج `src/api/**/*.ts` و الأكثر دعمًا للأنماط العابرة للمجلدات مثل ملفات الاختبار الموزعة.

السؤال 41 (سيناريو: توليد الكود باستخدام Claude Code)

الموقف: تريد إنشاء slash command مخصص `/review` يشغل قائمة مراجعة الكود القياسية لفريقك. يجب أن يكون متاحًا لكل مطور عند استنساخ الريبو أو تحديثه.

أين ينبغي إنشاء ملف الأمر؟

- A) في `~/.claude/commands/` داخل مجلد home لكل مطور.

- B) في مستودع المشروع تحت `.claude/commands/`. **[CORRECT]**
- C) في `.claude/config.json` كمصفوفة commands.
- D) في CLAUDE.md الجذري للمشروع.

لماذا B: داخل مستودع المشروع يجعلها مدارة عبر `.claude/commands/` المخصصة تحت slash وضع أوامر ومتاحة تلقائيًا لكل مطور يستنسخ الريبو أو يحدثه. هذا هو الموقع المقصود لأوامر المشروع في version control Claude Code.

السؤال 42 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: أصبح CLAUDE.md الخاص بفريقك أكثر من 500 سطر، ويمزج اصطلاحات TypeScript، وإرشادات الاختبار، وإجراءات النشر. يجد المطورون صعوبة في العثور على الأقسام الصحيحة وتحديثها، وأنماط API.

ما النهج الذي يدعمه Claude Code لتنظيم تعليمات المشروع إلى وحدات موضوعية مركزة؟

- A) عرّف ملف `.claude/config.yaml` يربط أنماط الملفات بأقسام محددة داخل CLAUDE.md.
- B) أنشئ ملفات Markdown منفصلة في `.claude/rules/`، بحيث يغطي كل ملف موضوعًا واحدًا مثل `testing.md` و`api-conventions.md`. **[CORRECT]**
- C) قسّم التعليمات إلى ملفات README.md في المجلدات ذات الصلة بحيث يحمّلها Claude تلقائيًا كتعليمات.
- D) أنشئ عدة ملفات باسم CLAUDE.md في مستويات مختلفة من شجرة المجلدات، بحيث يتجاوز كل ملف تعليمات الأب.

لماذا B: يدعم Claude Code مجلد `.claude/rules/` حيث يمكنك إنشاء ملفات Markdown منفصلة للتوجيهات الموضوعية مثل `testing.md` و`api-conventions.md`، مما يسمح للفريق بتنظيم مجموعات التعليمات الكبيرة في وحدات مركزة وأسهل صيانة.

السؤال 43 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: تنشئ مهارة مخصصة `/explore-alternatives` يستخدمها الفريق للعصف الذهني وتقييم مقاربات اللاحقة بنقاش البدائل، وأحيانًا Claude التنفيذ قبل اختيار واحدة. يذكر المطورون أنه بعد تشغيل المهارة، تتأثر ردود تشير إلى مقاربات مرفوضة أو تحتفظ بسياق الاستكشاف مما يتداخل مع التنفيذ الفعلي.

كيف ينبغي إعداد هذه المهارة بأكثر طريقة فعالية؟

- A) استخدم بادئة `!` في المهارة لتشغيل منطق الاستكشاف كعملية bash subprocess.
- B) أضف `context: fork` في frontmatter الخاص بالمهارة. **[CORRECT]**
- C) قسّمها إلى مهارتين: `/explore-start` و`/explore-end` لتحديد حدود سياق الاستكشاف الذي يجب تجاهله.
- D) أنشئ المهارة في `~/.claude/skills/` بدل `.claude/skills/`.

لماذا B: `context: fork` يشغل المهارة في سياق وكيل فرعي معزول، بحيث لا تلوث نقاشات الاستكشاف سجل المحادثة الرئيسي. هذا يمنع المقاربات المرفوضة وسياق العصف الذهني من التأثير على أعمال التنفيذ اللاحقة.

السؤال 44 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: يريد فريقك إضافة GitHub MCP server للبحث في PRs والتحقق من حالة CI عبر Claude Code. لكل شخصي. تريد أدوات متسقة عبر الفريق دون حفظ بيانات الاعتماد في GitHub واحد من ستة مطورين رمز وصول version control.

ما نهج الإعداد الأكثر فعالية؟

- A) اجعل كل مطور يضيف الخادم في نطاق المستخدم عبر `claude mcp add --scope user`.
- B) أنشئ MCP server wrapper يقرأ الرموز من ملف `.env` ويعمل كوسيط لاستدعاءات GitHub API، ثم أضف wrapper إلى `.mcp.json` الخاص بالمشروع.
- C) أضف الخادم إلى `.mcp.json` الخاص بالمشروع باستخدام استبدال متغير البيئة (`${GITHUB_TOKEN}`) للمصادقة، ووثق متغير البيئة المطلوب في README الخاص بالمشروع. **[CORRECT]**
- D) أعد الخادم في نطاق المشروع مع token placeholder، ثم أخبر المطورين بتجاوزه في إعداداتهم المحلية.

لماذا C: استخدام `.mcp.json` على مستوى المشروع مع استبدال متغيرات البيئة هو النهج الطبيعي: يوفر مصدرًا مع السماح لكل مطور بتوفير بيانات اعتماده عبر متغيرات البيئة. توثيق MCP، واحدًا مضبوطًا بالإصدارات لإعداد المتغير يجعل onboarding أسهل دون حفظ الأسرار في المستودع.

السؤال 45 (السيناريو: توليد الكود باستخدام Claude Code)

الموقف: تضيف wrappers استدعاءات API لمعالجة الأخطاء حول 120 ملفًا خارجية عبر قاعدة كود تحتوي على العمل له ثلاث مراحل: (1) اكتشاف كل مواضع الاستدعاء والأنماط، (2) تصميم نهج معالجة الأخطاء بشكل تعاوني، و(3) تنفيذ wrappers مخرجات كبيرة تسرد مئات مواضع الاستدعاء مع السياق، يولد Claude باتساق. في المرحلة الأولى، مما يملأ نافذة السياق بسرعة قبل انتهاء الاكتشاف.

ما النهج الأكثر فعالية لإكمال المهمة مع الحفاظ على اتساق التنفيذ؟

- A) استخدم Explore subagent للمرحلة الأولى لعزل مخرجات الاكتشاف المطولة وإرجاع ملخص، ثم تابع المرحلتين 2 و3 في المحادثة الرئيسية. **[CORRECT]**
- B) نفذ كل المراحل في المحادثة الرئيسية، مع استخدام `/compact` دوريًا لتقليل استخدام السياق أثناء التنقل بين الملفات.
- C) انتقل إلى headless mode باستخدام `--continue`، مع تمرير ملخصات سياق صريحة بين استدعاءات batch للحفاظ على الاستمرارية.
- D) عرّف نمط معالجة الأخطاء في CLAUDE.md، ثم عالج الملفات على دفعات عبر جلسات متعددة مع الاعتماد على ملف الذاكرة المشترك للاتساق.

لماذا A: يعزل Explore subagent إلا مخرجات الاكتشاف المطولة في سياق منفصل، ولا يعيد إلى المحادثة الرئيسية ملخصًا موجزًا. هذا يحافظ على نافذة السياق الرئيسية لمراحل التصميم التعاوني والتنفيذ المتسق، وهي المراحل التي يكون فيها الاحتفاظ بالسياق أكثر قيمة.

السيناريو: وكيل خدمة العملاء

السؤال 46 (السيناريو: وكيل خدمة العملاء)

عندما يسأل المستخدمون عن حالة الطلب، `get_customer` **الموقف:** أثناء الاختبار، تلاحظ أن الوكيل غالبًا يستدعي ستكون أنسب. ما أول شيء ينبغي فحصه لمعالجة المشكلة؟ `lookup_order` رغم أن

ما أول شيء ينبغي فحصه؟

- A) تنفيذ مصنف preprocessing لاكتشاف طلبات الطلبات وتوجيهها مباشرة إلى `lookup_order`.
- B) تقليل عدد الأدوات المتاحة للوكيل لتبسيط الاختيار.
- C) إضافة أمثلة few-shot إلى system prompt لتحسين اختيار الأداة تغطي كل أنماط طلبات الطلبات الممكنة.
- D) فحص أوصاف الأدوات للتأكد من أنها تميز بوضوح وظيفة كل أداة. **[CORRECT]**

لماذا D: أوصاف الأدوات هي المدخل الأساسي الذي يستخدمه النموذج لتحديد الأداة التي يستدعيها. عندما يختار الوكيل أداة خاطئة باستمرار، فأول خطوة تشخيصية هي التأكد من أن أوصاف الأدوات تفصل بوضوح بين وظيفة كل أداة وحدود استخدامها.

السؤال 47 (السيناريو: وكيل خدمة العملاء)

الموقف: يتعامل وكيلك مع الطلبات ذات المشكلة الواحدة بدقة 94%، مثل: "أحتاج refund للطلب #1234". لكن عندما يضمّن العملاء عدة مشكلات في رسالة واحدة، مثل: "أحتاج refund للطلب #1234 وأريد أيضًا تحديث عنوان الشحن للطلب #5678"، تنخفض دقة اختيار الأدوات إلى 58%. عادةً يحل الوكيل مشكلة واحدة فقط أو يخلط المعاملات بين الطلبات. ما النهج الأكثر فعالية لتحسين موثوقية الطلبات متعددة المشكلات؟

ما النهج الأكثر فعالية؟

- A) تنفيذ طبقة preprocessing تستخدم استدعاء نموذج منفصل لتفكيك الرسائل متعددة المشكلات إلى طلبات منفصلة، ومعالجة كل طلب باستقلال، ثم دمج النتائج.
- B) دمج الأدوات ذات الصلة في أدوات عامة أقل عددًا.
- C) إضافة أمثلة few-shot إلى المطالبة توضح الاستدلال الصحيح وتسلسل الأدوات للطلبات متعددة المشكلات. **[CORRECT]**
- D) تنفيذ تحقق من الاستجابة يكتشف الإجابات غير المكتملة ويعيد مطالبة الوكيل تلقائيًا لحل المشكلات الفائتة.

لماذا C: أمثلة few-shot التي توضح الاستدلال الصحيح وتسلسل الأدوات للطلبات متعددة المشكلات هي الأكثر فعالية لأن الوكيل يعمل جيدًا أصلًا مع المشكلات المنفردة. ما يحتاجه هو إرشاد حول نمط تفكيك عدة مشكلات وتوجيه كل واحدة والحفاظ على فصل المعاملات.

السؤال 48 (السيناريو: وكيل خدمة العملاء)

الموقف: تُظهر سجلات الإنتاج أنه في الطلبات البسيطة مثل "refund للطلب #1234"، يحل الوكيل المشكلة خلال 3–4 استدعاءات أدوات بنسبة نجاح 91%. لكن في الطلبات المعقدة مثل "تمت فوترتي مرتين، والخصم لم يُطبق، وأريد الإلغاء"، يتجاوز المتوسط 12 استدعاء أداة مع نجاح 54% فقط. غالبًا يحقق في المشكلات بشكل تسلسلي ويجلب بيانات العميل نفسها مرارًا لكل مشكلة. ما التغيير الأكثر فعالية لتحسين التعامل مع الطلبات المعقدة؟

ما التغيير الأكثر فعالية؟

- A) إضافة نقاط تحقق صريحة بين المراحل، بحيث يُطلب من الوكيل تسجيل التقدم بعد حل كل مشكلة قبل الانتقال إلى التالية.
- B) تقليل عدد الأدوات بدمج `get_customer` و `lookup_order` وأدوات الفوترة في أداة واحدة `investigate_issue`.
- C) تفكيك الطلب إلى مشكلات منفصلة، ثم التحقيق في كل واحدة بالتوازي باستخدام سياق العميل المشترك قبل تركيب حل نهائي. **[CORRECT]**
- D) إضافة أمثلة few-shot إلى system prompt توضح تسلسلات tool-call المثالية لسيناريوهات فوترة متعددة الجوانب.

لماذا C: تفكيك الطلب إلى مشكلات منفصلة والتحقيق فيها بالتوازي مع سياق عميل مشترك يعالج المشكلتين الأساسيتين: يقلل جلب البيانات المكرر بإعادة استخدام السياق المشترك، ويقلل حلقات استدعاء الأدوات عبر تنفيذ التحقيقات بالتوازي قبل تركيب حل واحد.

السؤال 49 (السيناريو: وكيل خدمة العملاء)

الموقف: يحقق وكيلك 55% فقط من الحل من أول تواصل، وهو أقل بكثير من هدف 80%. تُظهر مراجعة السجلات أنه يصعّد حالات بسيطة مثل طلبات refund القياسية، لكنه يحاول حل استثناءات سياسية معقدة بنفسه. ما أفضل تدخل أولي لتحسين المعايرة؟

ما أفضل تدخل أولي؟

- A) اطلب من الوكيل تقييم ثقته من 1 إلى 10 قبل كل رد، والتوجيه تلقائيًا للبشر عندما تنخفض الثقة عن حد معين.
- B) انشر مصنفًا منفصلًا مدربًا على tickets الرئيسي تاريخية للتنبؤ بالطلبات التي تحتاج تصعيدًا قبل بدء الوكيل.
- C) أضف معايير تصعيد صريحة إلى system prompt مع أمثلة few-shot توضح متى يصعّد ومتى يحل ذاتيًا. **[CORRECT]**
- D) نفذ sentiment analysis لتحديد مستوى إحباط العميل والتصعيد تلقائيًا عند تجاوز عتبة سلبية.

لماذا C: تعالج السبب الجذري مباشرة: حدود قرار غير واضحة بين الحالات few-shot معايير التصعيد الصريحة مع أمثلة البسيطة والمعقدة. هذا هو التدخل الأكثر تناسبًا وفعالية أولًا لأنه يعلّم الوكيل متى يصعّد ومتى يحل ذاتيًا دون إضافة بنية تحتية جديدة.

السؤال 50 (السيناريو: وكيل خدمة العملاء)

الموقف: بعد استدعاء `get_customer` و `lookup_order`، أصبح لدى الوكيل كل بيانات النظام المتاحة لكنه ما زال يواجه عدم يقين. أي موقف هو الأكثر تبريرًا لاستدعاء `escalate_to_human`؟

أي موقف يبرر التصعيد أكثر؟

- A) عميل يريد إلغاء طلب شُحن أمس وسيصل غدًا. ينبغي للوكيل التصعيد لأن العميل قد يغيّر رأيه بعد استلام الشحنة.
- B) عميل يدعي أنه لم يستلم طلبًا، لكن التتبع يظهر أنه تم التسليم والتوقيع في عنوانه قبل ثلاثة أيام. ينبغي للوكيل التصعيد لأن عرض دليل يناقض كلام العميل قد يضر العلاقة.
- C) عميل يطلب مطابقة سعر منافس. سياساتك تسمح بتعديل السعر عند انخفاض السعر في موقعك خلال 14 يومًا، لكنها لا تذكر أسعار المنافسين. ينبغي للوكيل التصعيد لتفسير السياسة. **[CORRECT]**

- D) رسالة العميل تحتوي سؤالًا عن الفوترة وطلب إرجاع منتج. ينبغي للوكيل التصعيد حتى ينسق الإنسان المشكلتين في تفاعل واحد.

لماذا C: هذه فجوة سياسة حقيقية: قواعد الشركة تغطي انخفاض الأسعار في موقعك لكنها لا تغطي مطابقة أسعار المنافسين. لا ينبغي للوكيل اختراع سياسة، بل يصعّد للحكم البشري حول تفسير أو توسيع القواعد الحالية.

السؤال 51 (السيناريو: وكيل خدمة العملاء)

الموقف: تُظهر سجلات الإنتاج أنه في 12% من الحالات، يتخطى الوكيل `get_customer` ويستدعي `lookup_order` مباشرة باستخدام الاسم الذي قدمه العميل فقط، مما يؤدي أحيانًا إلى تحديد حسابات خاطئة و refunds غير صحيحة. ما التغيير الأكثر فعالية لإصلاح مشكلة الموثوقية هذه؟

ما التغيير الأكثر فعالية؟

- A) إضافة أمثلة few-shot توضح أن الوكيل يستدعي `get_customer` دائمًا أولًا، حتى عندما يقدم العملاء تفاصيل الطلب طوعًا.
- B) تنفيذ مصنف routing يحلل كل طلب ويفعّل فقط مجموعة فرعية من الأدوات المناسبة لذلك النوع من الطلبات.
- C) إضافة شرط برمجي مسبق يمنع `lookup_order` و `process_refund` حتى يرجع `get_customer` معرف عميل محققًا. **[CORRECT]**
- D) تقوية system prompt بعبارة أن التحقق من العميل عبر `get_customer` إلزامي قبل أي عمليات طلبات.

لماذا C: الشرط البرمجي المسبق يوفر ضمانًا حتميًا بأن التسلسل المطلوب سيُتبع. هذا هو النهج الأكثر فعالية لأنه يزيل احتمال تخطي التحقق بغض النظر عن سلوك LLM.

السؤال 52 (السيناريو: وكيل خدمة العملاء)

الموقف: تُظهر مقاييس الإنتاج أن رضا العملاء في نزاعات الفوترة المعقدة أو الإرجاعات متعددة الطلبات أقل بـ 15% من الحالات البسيطة، حتى عندما يكون الحل صحيحًا تقنيًا. يوضح تحليل السبب الجذري أن الوكيل يقدم حلولًا دقيقة لكنه يشرح المبرر بشكل غير متسق: أحيانًا يحذف تفاصيل سياسة ذات صلة، وأحيانًا يفوّت معلومات الجدول الزمني أو الخطوات التالية. فجوات السياق المحددة تختلف من حالة لأخرى. تريد تحسين جودة الحل دون إضافة رقابة بشرية.

ما النهج الأكثر فعالية؟

- A) إضافة قالب استجابة منظم يتطلب من الوكيل تضمين المبرر، والسياسة ذات الصلة، والجدول الزمني، والخطوات التالية قبل إنهاء الحالات المعقدة. **[CORRECT]**
- B) تصعيد كل الحالات المعقدة إلى إنسان حتى لا يترك الوكيل تفاصيل تفسيرية ناقصة.
- C) مطالبة الوكيل بأن يكون "أكثر تفصيلًا" في الحالات المعقدة.
- D) إضافة حد أدنى لطول الاستجابة في الحالات المعقدة لضمان شرح كافٍ.

لماذا A: المشكلة ليست دقة الحل بل اكتمال الشرح واتساقه. قالب استجابة منظم يجعل العناصر المهمة مطلوبة صراحة، مثل السياسة والجدول الزمني والخطوات التالية، دون الحاجة إلى تصعيد بشري أو الاعتماد على تعليمات عامة.

السؤال 53 (السيناريو: وكيل خدمة العملاء)

الموقف: في نزاعات الفوترة، يتبع الوكيل نمطًا تسلسليًا: يستدعي `get_customer`، ثم `lookup_order`، ثم `check_billing`، ثم مرة أخرى `lookup_order`، ثم مرة أخرى `check_billing`. ينتج عن ذلك زمن استجابة مرتفع وتكرار في الأدوات، مع أن معظم البيانات الأساسية يمكن جلبها مرة واحدة وإعادة استخدامها.

ما التغيير الأكثر فعالية؟

- A) توجيه الوكيل إلى جمع سياق العميل والطلب والفوترة مرة واحدة في بداية الحالة، ثم استخدام هذا السياق المشترك أثناء حل المشكلات. **[CORRECT]**
- B) زيادة `max_tokens` حتى يستطيع الوكيل الاحتفاظ بمزيد من مخرجات الأدوات.
- C) دمج كل أدوات العميل والطلب والفوترة في أداة واحدة كبيرة.
- D) إضافة تعليمات تطلب من الوكيل تقليل عدد استدعاءات الأدوات.

لماذا A: المشكلة هي نمط استدعاءات متكرر وتسلسلي. جمع السياق المشترك مرة واحدة ثم إعادة استخدامه يعالج التكرار مباشرة، ويقلل زمن الاستجابة دون توسيع الأدوات أو الاعتماد على تعليمات عامة.

السؤال 54 (السيناريو: وكيل خدمة العملاء)

الموقف: تُظهر سجلات الإنتاج نمطًا: يشير العملاء إلى مبالغ محددة، مثل "الخصم 15% الذي ذكرته"، لكن الوكيل يرد بقيم خاطئة. يوضح التحقيق أن هذه التفاصيل ذُكرت قبل أكثر من 20 دورًا ثم ضُغطت في ملخصات مبهمة مثل "تمت مناقشة تسعير ترويجي".

ما الإصلاح الأكثر فعالية؟

- A) زيادة عتبة التلخيص من 70% إلى 85% حتى يكون للمحادثات مساحة أكبر قبل تشغيل التلخيص.
- B) عندما يكتشف الوكيل إشارات مثل "كما ذكرت" تخزين سجل المحادثة الكامل في تخزين خارجي وتنفيذ retrieval.
- C) استخراج الحقائق التبادلية، مثل المبالغ والتواريخ وأرقام الطلبات، إلى كتلة "case facts" مستمرة تُدرج في كل مطالبة خارج السجل الملخص. **[CORRECT]**
- D) تعديل مطالبة التلخيص لتطلب صراحة الحفاظ على كل الأرقام والنسب والتواريخ وتوقعات العميل حرفيًا.

لماذا C: التلخيص يفقد التفاصيل الدقيقة بطبيعته. استخراج الحقائق التبادلية إلى كتلة منظمة خارج السجل الملخص يحفظ المعلومات الحرجة بحيث تكون متاحة بثبات في كل مطالبة مهما طال عدد الأدوار التي تم تلخيصها.

السؤال 55 (السيناريو: وكيل خدمة العملاء)

الموقف: أداة `get_customer` لديك ترجع كل النتائج عند البحث بالاسم. حاليًا، عند وجود عدة نتائج، يختار Claude العميل صاحب أحدث طلب، لكن بيانات الإنتاج تُظهر أن هذا يختار الحساب الخطأ في 15% من الحالات الغامضة. كيف تعالج ذلك؟

كيف تعالج ذلك؟

- A) تنفيذ نظام درجات ثقة يتصرف ذاتيًا فوق 85% ثقة ويطلب توضيحًا تحت العتبة.

- B) توجيه Claude إلى طلب معرف إضافي، مثل البريد الإلكتروني أو الهاتف أو رقم الطلب، عندما يرجع `get_customer` عدة نتائج قبل اتخاذ أي إجراء خاص بالعميل. **[CORRECT]**
- C) تعديل `get_customer` لترجع تطابقًا واحدًا "الأكثر احتمالًا" بناءً على خوارزمية ترتيب، مما يزيل الغموض.
- D) إضافة أمثلة few-shot إلى المطالبة توضح الاستدلال الصحيح وتسلسل الأدوات للتطابقات الغامضة.

لماذا B: طلب معرف إضافي من المستخدم هو الطريقة الأكثر موثوقية لحل الغموض لأن المستخدم يملك معرفة قطعية بهويته. دور محادثة إضافي واحد ثمن بسيط لإزالة معدل خطأ 15% بسبب اختيار الحساب الخطأ.

السؤال 56 (السيناريو: وكيل خدمة العملاء)

الموقف: تُظهر سجلات الإنتاج نمطًا ثابتًا: عندما يضمّن العملاء كلمة "account" في رسالتهم، مثل "I want to check my account for an order I made yesterday"، يستدعي الوكيل `get_customer` أولًا في 78% من الحالات. وعندما يصيغ العملاء طلبات مشابهة دون "account"، مثل "I want to check an order I made yesterday"، يستدعي `lookup_order` أولًا في 93% من الحالات. أوصاف الأدوات واضحة وغير غامضة. ما السبب الجذري الأكثر احتمالًا لهذا الفرق؟

ما السبب الجذري الأكثر احتمالًا؟

- A) يحتوي system prompt على تعليمات حساسة للكلمات المفتاحية توجه السلوك بناءً على مصطلحات مثل "account"، مما ينشئ أنماط اختيار أدوات غير مقصودة. **[CORRECT]**
- B) التدريب الأساسي للنموذج ينشئ ارتباطات بين مصطلحات "account" وعمليات العملاء تتجاوز أوصاف الأدوات.
- C) يحتاج النموذج إلى بيانات تدريب أكثر عن الرسائل متعددة المفاهيم وينبغي fine-tuning على أمثلة تحتوي مصطلحات الحساب والطلب معًا.
- D) تحتاج أوصاف الأدوات إلى أمثلة سلبية إضافية تحدد متى لا تستخدم كل أداة لمنع هذا الالتباس الناتج عن الكلمة المفتاحية.

لماذا A: النمط المنهجي المدفوع بالكلمة المفتاحية (78% مقابل 93%) يشير بقوة إلى وجود منطق توجيه صريح في system prompt يتفاعل مع كلمة "account"، ويدفع الوكيل نحو أدوات العميل. وبما أن أوصاف الأدوات واضحة أصلًا، فالاختلاف يشير إلى تعليمات على مستوى المطالبة تخلق توجيهًا سلوكيًا غير مقصود.

السؤال 57 (السيناريو: وكيل خدمة العملاء)

الموقف: تُظهر سجلات الإنتاج أن الوكيل غالبًا يستدعي `get_customer` عندما يسأل المستخدمون عن الطلبات، مثل "check my order #12345"، بدلًا من `lookup_order`. كلتا الأداتين لهما أوصاف مختصرة جدًا ("Gets customer information" / "Gets order details") وتقبلان صيغ معرفات تبدو متشابهة. ما أول خطوة أكثر فعالية لتحسين موثوقية اختيار الأداة؟

ما أول خطوة أكثر فعالية؟

- A) تنفيذ طبقة routing تحلل إدخال المستخدم قبل كل دور وتختار مسبقًا الأداة الصحيحة بناءً على الكلمات المفتاحية وأنماط المعرفات.
- B) دمج الأداتين في أداة واحدة `lookup_entity` تقبل أي معرف وتقرر داخليًا أي backend تستعلم عنه.
- C) إضافة أمثلة few-shot إلى system prompt توجه 5–8 أمثلة توضح أنماط اختيار الأدوات الصحيحة، مع استعلامات الطلبات إلى `lookup_order`.

- D) توسيع وصف كل أداة ليشمل صيغ الإدخال، وأمثلة الاستعلامات، والحالات الحدية، والحدود التي تشرح متى تستخدمها مقابل الأدوات المشابهة. **[CORRECT]**

لماذا D: توسيع أوصاف الأدوات بإضافة صيغ الإدخال، وأمثلة الاستعلامات، والحالات الحدية، والحدود الواضحة يعالج السبب الجذري مباشرة: الأوصاف المختصرة لا تمنح LLM معلومات كافية للتمييز بين أدوات متشابهة. هذه خطوة أولى منخفضة الجهد وعالية الأثر لأنها تحسن آلية الاختيار الأساسية التي يعتمد عليها النموذج.

السؤال 58 (السيناريو: وكيل خدمة العملاء)

الموقف: تنفذ حلقة الوكيل لوكيل الدعم. بعد كل استدعاء Claude API، تشغيل) يجب أن تقرر هل تواصل الحلقة أم تتوقف (عرض الإجابة النهائية للعميل). ما الذي يحدد هذا القرار؟ (مجددًا الأدوات المطلوبة ثم استدعاء Claude)

ما الذي يحدد هذا القرار؟

- A) افحص حقل `stop_reason` في استجابة Claude؛ واصل إذا كان `tool_use` وتوقف إذا كان `end_turn`. **[CORRECT]**
- B) حلل نص Claude للبحث عن عبارات مثل “I’m done” أو “Can I help with anything else?”؛ فالإشارات اللغوية الطبيعية تدل على اكتمال المهمة.
- C) يشير Claude عيّن حدًا أقصى للتكرارات، مثل 10 استدعاءات، وتوقف عند الوصول إليه بغض النظر عما إذا كان إلى الحاجة لمزيد من العمل.
- D) افحص هل تحتوي الاستجابة على نص assistant؛ إذا أنشأ Claude نصًا تفسيريًا فينبغي إنهاء الحلقة.

لماذا A: `stop_reason` هو إشارة Claude المنظمة والصريحة للتحكم في الحلقة: `tool_use` يعني أن Claude يريد تشغيل أداة واستلام نتائجها، بينما `end_turn` يعني أن Claude أكمل استجابته وينبغي إنهاء الحلقة.

السؤال 59 (السيناريو: وكيل خدمة العملاء)

الموقف: تُظهر سجلات الإنتاج أن الوكيل يسيء تفسير مخرجات أدوات MCP طوابع Unix الزمنية من لديك: `lookup_order`، ورموز حالة رقمية (1=pending، 2=shipped). بعض `get_customer`، وتواريخ ISO 8601 من الأدوات من خوادم MCP خارجية لا يمكنك تعديلها. أي نهج لتطبيع صيغ البيانات هو الأكثر قابلية للصيانة؟

أي نهج هو الأكثر قابلية للصيانة؟

- A) استخدم PostToolUse hook لاعتراض مخرجات الأدوات وتطبيق تحويلات التنسيق قبل أن يعالجها الوكيل. **[CORRECT]**
- B) أنشئ wrappers لأدوات الطرف الثالث عدّل الأدوات التي تتحكم بها لترجع صيغًا مفهومة للبشر، وأنشئ.
- C) أنشئ أداة `normalize_data` يستدعيها الوكيل بعد كل جلب بيانات لتحويل القيم.
- D) أضف توثيقًا تفصيليًا للصيغ إلى system prompt يشرح اصطلاحات بيانات كل أداة.

لماذا A: يوفر PostToolUse hook نقطة مركزية وحتمية لاعتراض كل مخرجات الأدوات وتطبيعها، بما في ذلك بيانات الخارجية، قبل أن يعالجها الوكيل. هذا أكثر قابلية للصيانة لأن التحويلات تعيش في الكود وتطبق بشكل MCP خوادم موحد، بدل الاعتماد على تفسير LLM.

السؤال 60 (السيناريو: وكيل خدمة العملاء)

الموقف: تُظهر سجلات الإنتاج أن الوكيل يختار أحيانًا `get_customer` عندما تكون `lookup_order` أنسب، خصوصًا في الاستعلامات الغامضة مثل “I need help with my recent purchase”. قررت إضافة أمثلة few-shot إلى system prompt لتحسين اختيار الأدوات. أي نهج يعالج المشكلة بأكبر فعالية؟

أي نهج هو الأكثر فعالية؟

- A) إضافة إرشادات صريحة “use when” و“don’t use when” في وصف كل أداة.
- B) إضافة أمثلة مجمعة حسب الأداة: كل سيناريوهات `lookup_order` معًا، ثم كل سيناريوهات `get_customer` معًا.
- C) إضافة 4–6 أمثلة موجهة للحالات الغامضة، مع مبرر لكل مثال يوضح لماذا اختيرت أداة معينة بدل البدائل الممكنة. **[CORRECT]**
- D) إضافة 10–15 مثالًا لطلبات واضحة وغير غامضة توضح اختيار الأداة الصحيح للسيناريوهات النموذجية لكل أداة.

لماذا C: استهداف أمثلة few-shot للحالات الغامضة المحددة التي تحدث فيها الأخطاء، مع مبرر صريح لسبب تفضيل أداة على بدائلها، يعلّم النموذج عملية القرار المقارن المطلوبة للحالات الحدية. هذا أكثر فعالية من الأمثلة العامة أو القواعد التصريحية.

السؤال 61 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: تستخدم أداة `remove_team_member` معامل `dry_run: boolean` لمعاينة التأثيرات قبل التنفيذ. تُظهر مراقبة الإنتاج أن الوكيل يتجاوز خطوة المعاينة باستدعاء الأداة مباشرة مع `dry_run=false`. تحتاج إلى ضمان أن كل عملية إزالة يسبقها عرض معاينة يؤكدها المستخدم صراحة.

ما النهج الأكثر موثوقية؟

- A) إضافة تحقق من جهة الخادم يسمح بـ `dry_run=false` فقط إذا حدث استدعاء `dry_run=true` بالمعاملات نفسها خلال آخر 60 ثانية.
- B) وسم الأداة بأنها تتطلب تأكيدًا، وإعداد طبقة orchestration لتطلب موافقة المستخدم قبل تمرير أي استدعاءات للأدوات الموسومة.
- C) إضافة تعليمات مفصلة وأمثلة few-shot باستدعاء `dry_run=true` أولًا وانتظار تأكيد المستخدم قبل الاستدعاء مرة أخرى.
- D) استبدالها بأداتين: `preview_remove_member` ترجع تفاصيل التأثير و confirmation token صالحًا للاستخدام مرة واحدة، و `execute_remove_member` تتطلب ذلك الرمز، بحيث يرتبط التنفيذ بالمعاينة. **[CORRECT]**

لماذا D: نهج الأداتين مع ربط التنفيذ بالرمز يجعل التنفيذ دون معاينة سابقة مستحيلًا معماريًا؛ أداة التنفيذ تتطلب حرفيًا رمزًا لا تولده إلا أداة المعاينة. هذا يفرض القيد على مستوى الكود بدل الاعتماد على امتثال LLM للتعليمات، أو على heuristics زمنية، أو على بنية orchestration خارجية.

السؤال 62 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: تُظهر مراقبة الإنتاج أن أداة `search_catalog` تفشل بنسبة 12%: منها 8% أخطاء network timeouts تنجح عند إعادة المحاولة، و4% أخطاء query syntax لا تنجح مهما أُعيدت المحاولة. حاليًا يُرجع النوعان بالطريقة نفسها، مما يسبب إعادة محاولات مهدرة.

كيف ينبغي تعديل معالجة الأخطاء في الأداة؟

- A) إضافة أمثلة few-shot إلى system prompt توضح كيفية التمييز بين أخطاء الشبكة وأخطاء الصياغة.
- B) تطبيق exponential backoff retry logic على كل الأخطاء بشكل موحد.
- C) تنفيذ إعادة محاولة تلقائية مع backoff لأخطاء network timeout داخل الأداة، وإرجاع أخطاء الصياغة فورًا مع تفاصيل التحقق من المعاملات. **[CORRECT]**
- D) إرجاع كل الأخطاء مع راية `retryable` وتفاصيل نوع الخطأ.

لماذا C: التعامل مع إعادة المحاولة داخل الأداة للأخطاء العابرة هو الحد التجريدي الصحيح؛ الأداة تعرف نوع الخطأ بشكل حاسم ويمكنها تنفيذ retry logic حتمي دون الاعتماد على الوكيل لتفسير flag أو اتباع تعليمات prompt-level. أما backoff الموحد فيهدر الوقت على أخطاء syntax التي لن تنجح.

السؤال 63 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: خلال عدة أدوار عن استراتيجية استثمار، قال المستخدم: "I have a very low risk tolerance"، ثم قال لاحقًا: "I want to maximize my returns." والآن يسأل: "What should I invest in?"

أي نهج يضمن بأفضل شكل أن التوصية تتوافق مع أولوية المستخدم الفعلية؟

- A) إبراز التناقض وسؤال المستخدم توضيح أي الأمرين أهم. **[CORRECT]**
- B) تقديم توصيات منفصلة لكلا السيناريوهين.
- C) المتابعة بناءً على آخر تفضيل ذكره المستخدم.
- D) توصية بمحفظة متوازنة دون معالجة التعارض.

لماذا A: عندما تتعارض تفضيلات المستخدم مباشرة، فإن إبراز التعارض وطلب التوضيح هو الطريقة الوحيدة لضمان أن التوصية تتوافق مع نيته الحقيقية. أي نهج آخر يتضمن افتراضًا قد يكون خاطئًا؛ تعظيم العوائد وتحمل المخاطر المنخفض هدفان متعارضان جوهريًا ويتطلبان قرارًا من المستخدم.

السؤال 64 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: يضبط المستخدمون تفضيلات playlist عبر عدة أدوار. بعد رسالتين من قول المستخدم "I love jazz"، يسأل Claude: "What genres do you enjoy?"

ما السبب الأكثر احتمالًا؟

- A) يحتاج Claude إلى اتصال vector database للحفاظ على ذاكرة المحادثة.
- B) تم تجاوز نافذة السياق الخاصة بالنموذج.
- C) يتطلب Claude API معامل `session_id`.
- D) تطبيقك لا يضمّن الرسائل السابقة في مصفوفة `messages`. **[CORRECT]**

لماذا D: إذا لم تُضمّن سجل المحادثة الكامل في API stateless. ذاكرة من جهة الخادم؛ كل استدعاء لا يملك Claude الأدوار السابقة لكل طلب، فلن يعرف Claude. Vector databases و `session_id` ليست مصفوفة `messages` وتجاوز نافذة السياق مستحيل في تبادل من رسالتين، جزءًا من معمارية.

السؤال 65 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: بعد جلسة طبخ مدتها 40 دقيقة، وصلت المحادثة إلى 78,000 tokens. يتضمن السجل حساسية غذائية، وتعديل كميات الوصفة، وتوضيح مصطلحات طبخ، ونقاشًا عامًا. يجب تقليل الرموز مع الحفاظ على المعلومات المهمة.

أي نهج يوازن بأفضل شكل بين الحفظ وتقليل الرموز؟

- A) تلخيص سجل المحادثة كله.
- B) الاحتفاظ بآخر 20,000 tokens فقط.
- C) استخراج البيانات الحرجة المنظمة (الحساسية، الكميات، التفضيلات)، وتلخيص النقاش العام، والاحتفاظ بالتبادلات الأخيرة حرفيًا. **[CORRECT]**
- D) تخزين المحادثة كاملة خارجيًا واسترجاع الأجزاء ذات الصلة عبر semantic search.

لماذا C: النهج الهجين يحفظ أعلى المعلومات قيمة بأقل تكلفة. الحقائق الحرجة مثل الحساسية والكميات تُستخرج في كتلة منظمة مضغوطة، مما يمنع فقدان الدقة الذي يحدث أثناء التلخيص. أما النقاش العام فيُلخص، والتبادلات الأخيرة تُحفظ حرفيًا للحفاظ على اتساق الحوار. الخياران A وB يخاطران بفقدان معلومات غذائية حرجة، و D تعقيد معماري زائد لجلسة طبخ واحدة.

السؤال 66 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: يذكر المستخدمون أنه أثناء المحادثات الطويلة يفقد المساعد تتبع المواضيع والتفضيلات السابقة. تنفيذك الحالي يحتفظ فقط بآخر 25 زوجًا من الرسائل.

ما الحل الأكثر فعالية؟

- A) نهج هجين: تلخيص الرسائل القديمة مع الاحتفاظ بالرسائل الحديثة حرفيًا. **[CORRECT]**
- B) استخدام vector similarity search على سجل المحادثة الكامل.
- C) زيادة النافذة إلى 50 زوجًا من الرسائل.
- D) تلخيص الرسائل التي تُحذف في كل دور وإضافة running summary في البداية.

لماذا A: النهج الهجين يعالج بُعدين للمشكلة: الاحتفاظ بالسياق الحديث بشكل دقيق، وهو مهم لاتساق الحوار، مع الحفاظ على تمثيل مضغوط للتفضيلات القديمة حتى لا تضيع كليًا عند حذف الأزواج القديمة. زيادة النافذة تؤجل المشكلة فقط. وقد يفوّت vector search سياقًا مهمًا غير مشابه دلاليًا للاستعلام الحالي. أما التلخيص الكامل في كل دور فيضيف تكلفة ويُراكم أخطاء التلخيص.

السؤال 67 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: يذكر المستخدمون أن زمن الاستجابة والتكلفة يرتفعان عندما تتجاوز المحادثات 50 دورًا.

ما السبب الأساسي؟

- A) يتم تضمين سجل المحادثة الكامل مع كل طلب API. **[CORRECT]**
- B) يولد النموذج ردودًا أطول تدريجيًا.
- C) تبطؤ عمليات قاعدة البيانات كلما نما السجل.
- D) يبني النموذج ملفًا داخليًا للمستخدم يتطلب معالجة أكثر.

لماذا A: Claude API stateless بالكامل، لذلك يجب تضمين سجل المحادثة الكامل في مصفوفة `messages` مع كل طلب. ومع نمو المحادثات، يحمل كل طلب رموزًا أكثر، مما يزيد زمن المعالجة والتكلفة مباشرة. لا يحتفظ النموذج بأي حالة داخلية بين الاستدعاءات، وطول الرد ليس مرتبطًا بالضرورة بطول المحادثة.

السؤال 68 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: بعد ثلاثة أشهر من جلسات أسبوعية، نما سجل المحادثة إلى 85,000 tokens. عندما يسأل المستخدم: “What did we conclude about the theme of isolation?” يعطي المساعد إجابات عامة بدل الرجوع إلى نقاشات سابقة.

ما النهج الأكثر فعالية؟

- A) rolling window truncation.
- B) progressive summarization يلتقط الاستنتاجات الأساسية.
- C) semantic embeddings مع استرجاع التبادلات ذات الصلة. **[CORRECT]**
- D) إضافة structured XML tags لتمييز استنتاجات النقاش.

لماذا C: semantic search على سجل المحادثة هو النهج الوحيد الذي يتوسع لثلاثة أشهر من النقاشات ويستطيع إظهار تبادلات محددة ذات صلة عند الطلب. rolling window سيحذف معظم السجل، و progressive summarization يضغط النقاشات إلى تجريدات تفقد الاستنتاجات المحددة التي يسأل عنها المستخدم، و XML tags تتطلب إعادة هيكلة كل المحتوى السابق ولا تحل مشكلة الاسترجاع بهذا الحجم.

السؤال 69 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: أثناء اختبار QA، يتبع Claude إرشادات system prompt بدقة في أول 10–15 دورًا، لكن الردود اللاحقة تنحرف. ما زالت المحادثة ضمن حدود الرموز.

ما أفضل حل؟

- A) نقل الإرشادات السلوكية إلى أول رسالة user.
- B) بدء محادثة جديدة بعد 20 دورًا.
- C) تعزز الإرشادات عند نقاط فاصلة في المحادثة إدراج رسائل user-role. **[CORRECT]**
- D) استخدام post-response validation لإعادة توليد الردود غير المتوافقة.

لماذا C: إدراج تذكيرات سلوكية دورية يعالج instruction drift مباشرة بإعادة تثبيت القيود في نقاط منتظمة مع تراكم سجل المحادثة. نقل الإرشادات إلى أول رسالة user يقلل سلطتها، وبدء محادثة جديدة يدمر السياق، أما post-response validation فهو تصحيحي لا وقائي ويضيف زمن استجابة كبيرًا.

السؤال 70 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: لدى AI tutor الخاص بك system prompt بطول 2,800 tokens. بعد 12 يحدد منهجية التعليم وقواعد التكيّف. دورًا، يبدأ المساعد بتجاهل مستويات الإتقان.

ما الإصلاح الأكثر فعالية؟

- A) إدراج تذكيرات كل 4–5 أدوار.

- B) توضح التكيّف مع مستوى الإتقان few-shot استبدال القواعد المطولة بأمثلة. **[CORRECT]**
- C) وضع القواعد الحرجة في نهاية system prompt.
- D) تقييم الردود وإعادة توليدها إذا لم يطابق مستوى الصعوبة.

لماذا B: system prompt طويل مليء بقواعد تصريحية يكون عرضة للانحراف؛ لأن القواعد المجردة تتطلب من النموذج التفكير فيها في كل دور. few-shot ملموسة توضح التكيّف الصحيح مع استبدال القواعد المطولة بأمثلة مستوى الإتقان يعطي النموذج أنماطًا سلوكية واضحة يطابقها، وهذا أكثر موثوقية عبر الأدوار من التعليمات المجردة. التذكيرات تساعد لكنها تعالج العرض، ووضع القواعد في النهاية يساعد في البداية فقط، وإعادة التوليد مكلفة وتصحيحية.

السؤال 71 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: يجب أن يحافظ المساعد على نبرة حماسية، ويشرح منطقه، ويسأل أسئلة توضيحية. أين ينبغي تعريف هذه الإرشادات السلوكية؟

أين ينبغي تعريف هذه الإرشادات السلوكية؟

- A) إضافتها قبل كل رسالة user.
- B) في system prompt. **[CORRECT]**
- C) في أول رسالة assistant.
- D) في environment variables.

لماذا B: system prompt مصمم خصيصًا للقيود والإرشادات السلوكية المستمرة التي تطبق طوال المحادثة. إضافتها قبل كل رسالة user غير موثوقة لأن النموذج قد ينحرف عن تصريحاته السابقة، وأول رسالة assistant تكرار زائد، وenvironment variables لا تؤثر في سلوك النموذج.

السؤال 72 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: يذكر المستخدمون بدايات ردود متكررة مثل “I’d be happy to help!” و“Certainly!”

ما النهج الأكثر فعالية؟

- A) إلحاق partial assistant message ببداية رد مباشرة. **[CORRECT]**
- B) خفض إعداد temperature.
- C) post-process للردود لإزالة التحيات.
- D) إضافة تعليمات في system prompt لتجنب هذه العبارات.

لماذا A: prefill لبداية رد assistant مباشرة يمنع أنماط التحية على مستوى التوليد؛ النموذج يكمل من prefill بدل توليد عبارات افتتاحية جديدة. تعليمات system prompt قد تساعد لكنها أقل موثوقية لأن النموذج قد ينتج تنويعات post-processing. temperature يتحكم بالعشوائية لا بعبارات محددة، و حل هش

السؤال 73 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: يخبر نظامك بأن طرد المستخدم قد شُحن بينما المستخدم ما يزال في محادثة نشطة. تريد أن webhook يدمج المساعد هذه المعلومة طبيعيًا في الرد التالي.

ما أفضل نهج؟

- A) إضافة حالة الشحن إلى system prompt.
- B) إرسال synthetic user message فورية.
- C) إجبار المساعد على استدعاء status tool في كل دور.
- D) إلحاق تحديث الحالة كبادئة برسالة المستخدم التالية. **[CORRECT]**

لماذا D: إضافة تحديث الحالة كبادئة لرسالة المستخدم التالية يحقن سياقًا آنيًا عند حد طبيعي في المحادثة دون تعطيل التدفق. تعديل system prompt يتطلب إعادة بناء الجلسة أو يكون مرهقًا معماريًا. synthetic user message قد يكسر التدفق الطبيعي ويلبس attribution. وإجبار tool call في كل دور هدر عندما تكون الأحداث نادرة.

السؤال 74 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: كثيرًا ما يرسل المستخدمون طلبات مثل “Book a venue for the party.” يطرح المساعد أكثر من 4 أسئلة توضيحية، مما يسبب 35% abandonment.

أي نهج يحسن المفاضلة بأفضل شكل؟

- A) المتابعة باستخدام defaults مخفية.
- B) طرح كل الأسئلة التوضيحية في رسالة مركبة واحدة.
- C) ذكر الافتراضات صراحة والمتابعة مع دعوة المستخدم لتصحيحها. **[CORRECT]**
- D) استخدام structured intake form.

لماذا C: ذكر الافتراضات صراحة والمتابعة يعطي المستخدم ردًا مفيدًا فوريًا مع الحفاظ على قدرته على تصحيح الافتراضات الخاطئة. defaults المخفية لا تخبر المستخدم بما تم افتراضه، وقائمة الأسئلة المركبة ما تزال تطلب جهدًا upfront، والنموذج structured form يزيد الاحتكاك بدل تقليل abandonment.

السؤال 75 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: يستخدم المساعد system prompt بشخصية contractor. تتبع الأدوار الأولى القواعد، لكن عند الدور 7 يقدم المساعد نصائح عامة. طول المحادثة فقط 2,500 tokens.

ما السبب الأكثر احتمالًا؟

- A) system prompts لا تؤسس إلا السلوك الأولي.
- B) يضعف انتباه النموذج مع تراكم الأدوار.
- C) تراكم ردود assistant يخفف تأثير system prompt. **[CORRECT]**
- D) يتم إرسال system prompt مرة واحدة فقط.

لماذا C: system prompt داخل سجل المحادثة، تنخفض نسبة النص الذي يعكس قيود assistant مع تراكم ردود assistant. يبدأ النموذج أكثر فأكثر بمطابقة أنماط ردوده السلوكية مقارنة بجسم متزايد من المحتوى الذي ولده

يُضمّن في كل استدعاء system prompt، مما يراكم الانحراف حتى عند عدد رموز صغير. system prompt السابقة بدل tokens. ليس هو العامل في attention degradation 2,500ليست تفسيرًا قائمًا بذاته، و D لذلك، API

السؤال 76 (السيناريو: أنماط معمارية الذكاء الاصطناعي الحواري)

الموقف: يرسل المستخدمون طلبات غامضة مثل “Can you help with the report?” فيرد المساعد بطرح عدة أسئلة: أي تقرير؟ أي نوع من المساعدة؟ ما الموعد النهائي؟ وهذا يسبب 40% abandonment.

ما أفضل حل؟

- A) وضع افتراضات معقولة، وذكرها صراحة، وعرض تعديلها. **[CORRECT]**
- B) تصنيف الغموض باستخدام نموذج أصغر قبل الرد.
- C) استخدام تفسيرات مسبقة دون ذكر الافتراضات.
- D) قصر المساعد على سؤال توضيحي واحد في كل دور.

لماذا A: المتابعة بافتراضات معقولة ومعلنة تلغي تبادل الأسئلة الطويل مع إبقاء المستخدم مطلعًا ومتحكمًا. التفسيرات الصامتة تربك المستخدم إذا لم تطابق نيته. حد السؤال الواحد ما يزال يتطلب أدوارًا من الأخذ والرد. ونموذج التصنيف الأصغر يضيف زمنًا وتعقيد بنية دون حل مشكلة تجربة المستخدم الأساسية.

تمارين عملية

التمرين 1: وكيل متعدد الأدوات مع منطق تصعيد

الهدف: تصميم حلقة وكيل مع تكامل الأدوات، ومعالجة أخطاء منظمة، وتصعيد.

الخطوات:

1. عرّف 3–4 أدوات MCP بأوصاف مفصلة، وتضمين أداتين متشابهتين لاختبار اختيار الأداة.
2. نفذ حلقة وكيل تفحص `stop_reason` (`"tool_use"` / `"end_turn"`).
3. أضف استجابات أخطاء منظمة: `errorCategory` , `isRetryable` , description.
4. نفذ interceptor hook يمنع العمليات فوق حد معين ويوجهها إلى التصعيد.
5. اختبر الطلبات متعددة الجوانب.

المجالات: 1 (بنية الوكيل)، 2 (الأدوات وMCP)، 5 (السياق والموثوقية)

التمرين 2: إعداد Claude Code لتطوير الفريق

الهدف: إعداد CLAUDE.md، والأوامر المخصصة، والقواعد الخاصة بالمسارات، وخوادم MCP.

الخطوات:

1. أنشئ CLAUDE.md على مستوى المشروع يحتوي على المعايير العامة.
2. أنشئ ملفات `.claude/rules/` مع YAML frontmatter لمناطق كود مختلفة (`paths: ["src/api/**/*"]` , `paths: ["**/*.test.*"]`).

3. أنشئ skill تحت `.claude/skills/` على مستوى المشروع مع `context: fork` و`allowed-tools`.
4. أعد خادم MCP في `.mcp.json` باستخدام متغيرات البيئة + override شخصي في `~/.claude.json`.
5. اختبر وضع التخطيط مقابل التنفيذ المباشر على مهام ذات تعقيد مختلف.

المجالات: 3 (إعداد Claude Code)، 2 (الأدواتMCP و)

التمرين 3: خط استخراج بيانات منظمة

الهدف: مخططات JSON، و`tool_use` للمخرجات المنظمة، وحلقات التحقق/إعادة المحاولة، ومعالجة الدفعات.

الخطوات:

1. عرّف أداة استخراج مع JSON schema (حقول required/optional، وenums مع `"other"`، وحقول nullable).
2. ابنِ حلقة تحقق: عند الخطأ، أعد المحاولة مع المستند، والاستخراج غير الصحيح، وخطأ التحقق المحدد.
3. أضف أمثلة few-shot لمستندات ببنى مختلفة.
4. استخدم batch processing عبر Message Batches API: 100 مستند، وتعامل مع الإخفاقات عبر `custom_id`.
5. وجّه الحالات إلى البشر: درجات ثقة على مستوى الحقول، وتحليل حسب نوع المستند.

المجالات: 4 (هندسة المطالبات)، 5 (السياق والموثوقية)

التمرين 4: تصميم وتصحيح خط بحث متعدد الوكلاء

الهدف: تنسيق الوكلاء الفرعيين، وتمرير السياق، وانتشار الأخطاء، والتركيب مع تتبع المصادر.

الخطوات:

1. منسق مع أكثر من وكيل فرعي (`allowedTools` يتضمن `"Task"`، ويتم تمرير السياق صراحة داخل المطالبات).
2. شغّل الوكلاء الفرعيين بالتوازي عبر عدة استدعاءات `Task` في استجابة واحدة.
3. اطلب إخراجًا منظمًا من الوكيل الفرعي: claim، quote، source URL، publication date.
4. حاكِ timeout لوكيل فرعي: أعد سياق خطأ منظم إلى المنسق وتابع بالنتائج الجزئية.
5. اختبر البيانات المتعارضة: احتفظ بالقيمتين مع الإسناد، وافصل النتائج المؤكدة عن المتنازع عليها.

المجالات: 1 (بنية الوكيل)، 2 (الأدواتMCP و)، 5 (السياق والموثوقية)

الملحق: التقنيات والمفاهيم

التقنية	الجوانب الأساسية
Claude Agent SDK	إنشاء الوكلاء (PostToolUse)، الخطافات، `stop_reason`، حلقات الوكيل، AgentDefinition، و Task، الفرعيين عبر `allowedTools`
Model Context Protocol (MCP)	متغيرات البيئة، `.mcp.json`، أوصاف الأدوات، `isError`، الأدوات، الموارد، خوادم MCP

Claude Code	هرمية CLAUDE.md، و `.claude/rules/` مع glob patterns، و `.claude/commands/`، و `.claude/skills/` مع SKILL.md، و `/compact` ووضع التخطيط، و `--resume`، و `fork_session`
Claude Code CLI	`-p` / `--print` للوضع غير التفاعلي، و `--output-format json` و `--json-schema`
Claude API	`tool_use` مع JSON schemas، و `tool_choice` (`"auto"` / `"any"`/forced)، وsystem prompts، و `max_tokens` و `stop_reason`
Message Batches API	متعدد الأدوار tool calling لا يدعم `custom_id`، توفير 50%، نافذة تصل إلى 24 ساعة
JSON Schema	الوضع الصارم، detail + `"other"`، أنواع enum، الحقول nullable، optional مقابل required
Pydantic	التحقق من المخطط، الأخطاء الدلالية، حلقات التحقق/إعادة المحاولة
الأدوات المدمجة	Read, Write, Edit, Bash, Grep, Glob — الغرض ومعايير الاختيار
Few-shot prompting	أمثلة موجهة للحالات الغامضة، والتعميم إلى أنماط جديدة
Prompt chaining	تفكيك متتابع إلى تمريرات مركزة
نافذة السياق	ملفات scratchpad، “lost in the middle”، ميزانيات الرموز، التلخيص التدريجي
إدارة الجلسات	Resume، و `fork_session`، والجلسات المسماة، وعزل السياق
معايرة الثقة	درجات على مستوى الحقول، والمعايرة على بيانات موسومة، والعينات الطبقية

مواضيع خارج النطاق

لن تكون ضمن الاختبار المواضيع المجاورة التالية:

- Fine-tuning نماذج Claude أو تدريب نماذج مخصصة
- مصادقة Claude API أو الفوترة أو إدارة الحساب
- تفاصيل التنفيذ في لغات برمجة أو أطر عمل محددة، باستثناء ما يلزم لإعداد الأدوات/المخططات
- نشر أو استضافة خوادم MCP، و container orchestrationمثل البنية التحتية، والشبكات،
- البنية الداخلية لـ Claude، أو عملية التدريب، أو model weights
- Constitutional AI أو RLHF أو منهجيات تدريب السلامة
- نماذج embeddings أو تفاصيل تنفيذ vector database
- Computer use، مثل أتمتة المتصفح أو التفاعل مع سطح المكتب
- قدرات تحليل الصور (Vision)
- Streaming API أو server-sent events
- Rate limiting أو quotas أو حسابات تكلفة API التفصيلية
- OAuth أو تدوير API keys أو تفاصيل بروتوكولات المصادقة
- إعدادات خاصة بمزودي السحابة مثل AWS أو GCP أو Azure

- مقاييس الأداء أو مقارنة النماذج
- تفاصيل تنفيذ prompt caching، باستثناء معرفة أنه موجود
- خوارزميات عدّ الرموز أو تفاصيل tokenization

توصيات الاستعداد

1. **ابنِ وكيلًا باستخدام Claude Agent SDK** — نفذ حلقة وكيل كاملة مع tool calling، وإدارة، ومعالجة الأخطاء. تدرب على الوكلاء الفرعيين وتمرير السياق الصريح الجلسات.
2. **أعد Claude Code لمشروع حقيقي** — استخدم هرمية CLAUDE.md، والقواعد الخاصة بالمسارات في `.claude/rules/`، والمهارات مع `context: fork` و`allowed-tools`، وتكامل خوادم MCP.
3. **صمم واختبر أدوات MCP** — اكتب أوصافًا تميز الأدوات المتشابهة، وأرجع أخطاء منظمة مع فئات وretry flags، واختبرها ضد طلبات مستخدم غامضة.
4. **ابنِ خط استخراج بيانات** — استخدم `tool_use` مع JSON schemas، وحلقات validation/retry، وحقول optional/nullable، ومعالجة الدفعات عبر Message Batches API.
5. **تدرب على هندسة المطالبات** — أضف أمثلة few-shot للحالات الغامضة، ومعايير مراجعة صريحة، وبنى متعددة المرور لمراجعات الكود الكبيرة.
6. **ادرس أنماط إدارة السياق** — استخرج الحقائق من المخرجات المطولة، واستخدم ملفات scratchpad، وفوض الاكتشاف إلى الوكلاء الفرعيين للتعامل مع حدود السياق.
7. **افهم التصعيد وإدخال الإنسان في الحلقة** — متى تصعّد: فجوات السياسة، طلب المستخدم الصريح، عدم القدرة على إحراز تقدم، و workflows التوجيه حسب الثقة.
8. **خذ اختبارًا تدريبيًا قبل الاختبار الحقيقي.** فهو يستخدم السيناريوهات والصيغة نفسها.