Claude Certified Architect — תעודת Foundations

מדריך לימוד (מבוסס על מדריך המבחן הרשמי)

הקדמה

תעודת **Claude Certified Architect — Foundations** מאשרת שמומחה מסוגל לקבל החלטות מאוזנות (trade-off) בעת מימוש פתרונות אמיתיים מבוססי Claude. המבחן בוחן ידע יסודי ב-Claude Code, ב-Claude Agent SDK, ב-Model Context Protocol (MCP) — הטכנולוגיות המרכזיות לבניית אפליקציות פרודקשן עם Claude וב-Claude API.

שאלות המבחן מבוססות על תרחישי תעשייה ריאליסטיים: בניית מערכות סוכן (agentic) לתמיכת לקוחות, תכנון פייפליין מחקר רב-סוכני, אינטגרציה של Claude Code ב-CI/CD, יצירת כלי פרודוקטיביות למפתחים, וחילוץ נתונים מובנים ממסמכים לא-מובנים.

הפרופיל המבוקש

המועמד האידיאלי הוא **ארכיטקט פתרונות (solution architect)** שמתכנן ומשגר אפליקציות פרודקשן עם Claude. רצויה לפחות חצי שנת ניסיון מעשי ב:

- **Claude Agent SDK** — אורקסטרציה רב-סוכנית, הענקת משימות לתת-סוכנים, אינטגרציית כלים, hooks של מחזור חיים
- **Claude Code** — CLAUDE.md, שרתי MCP, Agent Skills, מצב תכנון (planning mode)
- **Model Context Protocol (MCP)** — כלים ומשאבים לאינטגרציה אל ה-backend
- **הנדסת פרומפט** — JSON schemas, דוגמאות few-shot, תבניות לחילוץ נתונים
- **חלון הקשר (context window)** — עבודה עם מסמכים ארוכים, העברת הקשר רב-סוכני
- **CI/CD פייפליין** — code review אוטומטי, יצירת בדיקות
- **הסלמה ואמינות** — טיפול בשגיאות, human-in-the-loop

פורמט המבחן

פרמטר	ערך
סוג שאלה	בחירה מרובה (1 נכון מתוך 4)
ניקוד	סולם 100–1000, ציון עובר **720**
עונש על ניחוש	אין (תענו על כל שאלה!)
תרחישים	4 (בחירה אקראית) מתוך 8 אפשריים

תוכן המבחן: 5 דומיינים

דומיין	משקל
1. ארכיטקטורת סוכנים ואורקסטרציה	**27%**
2. עיצוב כלים ואינטגרציית MCP	**18%**
3. הגדרות Claude Code וזרימות עבודה	**20%**
4. הנדסת פרומפט ופלט מובנה	**20%**
5. ניהול הקשר ואמינות	**15%**

תרחישי המבחן

תרחיש 1: סוכן תמיכת לקוחות

אתם בונים סוכן שמטפל בהחזרות, מחלוקות חיוב ובעיות חשבון תוך שימוש ב-Claude Agent SDK. הסוכן משתמש בכלי MCP (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`). היעד הוא 80%+ פתרון במגע ראשון עם הסלמה מתאימה.

תרחיש 2: יצירת קוד עם Claude Code

אתם משתמשים ב-Claude Code להאצת פיתוח: יצירת קוד, refactoring, debugging, תיעוד. צריך לשלב פקודות slash והגדרות מותאמות אישית במצב תכנון, CLAUDE.md ולהבין מתי להשתמש בכל.

תרחיש 3: מערכת מחקר רב-סוכנית

סוכן מתאם מאציל משימות לתת-סוכנים מתמחים: מחקר אינטרנטי, ניתוח מסמכים, סינתזה, ויצירת דוחות. המערכת חייבת להפיק דוחות מלאים עם ציטוטים.

תרחיש 4: כלי פרודוקטיביות למפתחים

הסוכן עוזר למהנדסים לחקור codebase, לייצר boilerplate לא-מוכר, ולאוטומט משימות שגרתיות. נעשה שימוש בכלים מובנים (Read, Write, Bash, Grep, Glob) ובשרתי MCP.

תרחיש 5: Claude Code ל-Continuous Integration

שילוב Claude Code בפייפליין CI/CD ל-code review אוטומטי, יצירת בדיקות ופידבק על PR. הפרומפטים חייבים להיות מתוכננים למזער false positives.

תרחיש 6: חילוץ נתונים מובנים

המערכת מחלצת מידע ממסמכים לא-מובנים, מאמתת פלט מול JSON schema ושומרת על דיוק גבוה. עליה לטפל נכון במקרי קצה.

שיחתי AI תרחיש 7: דפוסי ארכיטקטורת

אתם מתכננים מערכות שיחה רב-תורניות שמכסות ניהול חלון הקשר, התמדה של הוראות בין תורים, אסטרטגיות זיכרון, עיצוב כלים לביצוע בטוח, וטיפול בקלט משתמש דו-משמעי או סותר.

סוכניים AI תרחיש 8: כלי *(תוכן חסר — עזרו לנו להשלים!)*

תרחיש זה דווח על ידי נבחנים אך טרם כוסה במדריך. אם נתקלתם בשאלות מהתרחיש הזה במבחן האמיתי, שתפו כדי שנוסיף כיסוי מלא. התרומה שלכם תעזור לכל מי שמתכונן למבחן ב-GitHub Issues.

תיעוד רשמי

משאב	כתובת
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/anthropic-cookbook

---

חלק I: יסודות תיאורטיים

חלק זה מכסה את כל התיאוריה שצריך לעבור בהצלחה את המבחן. החומר מאורגן לפי טכנולוגיות ומושגים ולא לפי דומיינים — כך בונים הבנה עמוקה יותר של כל נושא.

פרק 1: יסודות אינטראקציה עם המודל — Claude API

> תיעוד: Messages API | Prompt Engineering

1.1 מבנה בקשת API

ה-Claude API פועל על פי מודל בקשה–תגובה. כל בקשה ל-Claude Messages API כוללת:

```
{
  "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"}
}
```

שדות מפתח:

- `model` — בחירת מודל (`claude-opus-4-6`, `claude-sonnet-4-6`, `claude-haiku-4-5`)
- `max_tokens` — מספר טוקנים מקסימלי בתגובה
- `system` — system prompt (מגדיר התנהגות המודל)

- `messages` — היסטוריית שיחה (**חובה לשלוח את ההיסטוריה המלאה** כדי לשמור על קוהרנטיות)
- `tools` — הגדרות הכלים הזמינים
- `tool_choice` — אסטרטגיית בחירת כלי

1.2 תפקידי הודעות (Message Roles)

מערך ה-`messages` משתמש בשני תפקידי שיח ותפקיד הנחיה נוסף אחד:

- `user` — הודעות משתמש, כולל תוצאות כלים (נשלחות כ-content block מסוג `tool_result` בתוך הודעה בתפקיד `user`, ולא כתפקיד `tool` נפרד)
- `assistant` — תגובות המודל (כלולות בשליחת היסטוריה), כולל בקשות לשימוש בכלים (content blocks מסוג `tool_use`)
- `system` — ניתן להגדיר דרך השדה העליון `system` או באופן משולב בתוך (חל מהתור הראשון) `messages` (חל מאותה נקודה ואילך, בכפוף לכללי מיקום — ראו בהמשך) כ-`{"role": "system", ...}`

content שתוכנה כולל `user` הן נשלחות כהודעה בתפקיד `"tool"`. תוצאות כלים אינן נשלחות כהודעה בתפקיד `tool_result` מסוג block:

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

`system` מטרת. לא רק דרך הפרמטר העליון, `messages`-יכול להופיע גם כתפקיד ישירות בתוך מערך ה `system` של השדה העליון (cache) השמור במטמון prefix-הדבר היא להוסיף הנחיות באמצע השיחה מבלי לבטל את ה `system`. לכך יש כללי מיקום מסוימים:

- שמסתיים `assistant` או תור (`tool_result` כולל כזה שמכיל בלוקי) `user` חייב לבוא מיד אחרי תור בשימוש בכלי מצד השרת.
- חייב לקדום תור `assistant` או להיות האחרון במערך.
- שלו — עשיית כך מחזירה שגיאת 400 `tool_result`-ל `tool_use` אינו יכול לשבת בין בלוק.
- הודעות `system` מאוחרות יותר (כולל כאלה שבאמצע השיחה) גוברות על קודמות ועל השדה העליון `system` עבור התורות הבאים.

בין בקשות — כל state חייבים לשלוח את **כל היסטוריית השיחה**. המודל לא שומר API **חשוב במיוחד:** בכל בקשת קריאה עצמאית.

1.3 השדה `stop_reason` בתגובה

תגובת Claude API כוללת `stop_reason` שמציין מדוע המודל הפסיק לייצר:

ערך	תיאור	פעולה
`"end_turn"`	המודל סיים את תגובתו	הציגו תוצאה למשתמש
`"tool_use"`	המודל רוצה לקרוא לכלי	הריצו את הכלי והחזירו תוצאה
`"max_tokens"`	הגעתם לגבול טוקנים	התגובה קטומה; ייתכן שצריך להגדיל גבול
`"stop_sequence"`	נתקלתם ב-stop sequence	טפלו לפי לוגיקת האפליקציה

הם החשובים ביותר — הם שולטים בלולאת הסוכן `"end_turn"`-ו `"tool_use"` במערכות סוכן.

1.4 System Prompt

ה-system prompt הוא הוראה מיוחדת שמגדירה הקשר וכללי התנהגות. הוא:

- מועבר בנפרד בשדה `system`; לא חלק ממערך ה-`messages`
- בעדיפות גבוהה יותר מהודעות משתמש
- נטען פעם אחת וחל לאורך כל השיחה
- משמש להגדרת תפקיד, מגבלות ופורמט פלט

יכול ליצור אסוציאציות לא-רצויות לכלים. למשל, הוראה כמו "תאמת תמיד system prompt **חשוב למבחן:** ניסוח של את הלקוח" עלולה לגרום למודל להשתמש יותר מדי ב-`get_customer`, גם כשזה לא נחוץ.

1.5 חלון הקשר (Context Window)

חלון ההקשר הוא הכמות הכוללת של טקסט (בטוקנים) שהמודל יכול לעבד בבת אחת. הוא כולל:

- system prompt
- היסטוריית ההודעות המלאה
- הגדרות כלים
- תוצאות כלים

בעיות מפתח של חלון ההקשר:

1. **אפקט lost-in-the-middle:** מודלים מעבדים מידע באמינות בהתחלה ובסוף של קלט ארוך, אבל עלולים לפספס פרטים באמצע. מיתון: מקמו מידע קריטי קרוב לתחילה או לסוף.
2. **הצטברות תוצאות כלי:** כל קריאה לכלי מוסיפה פלט להקשר. אם כלי מחזיר 40+ שדות אבל רק 5 רלוונטיים — רוב ההקשר מבוזבז.
3. **סיכום פרוגרסיבי:** בעת דחיסת היסטוריה, ערכים מספריים, אחוזים ותאריכים אובדים לעיתים והופכים מעורפלים ("בערך", "בקירוב", "כמה").

פרק 2: Tools ו-`tool_use`

> תיעוד: Tool Use

2.1 מה זה `tool_use`

`tool_use` מנגנון שמאפשר ל-Claude הוא מייצר קוד בעצמו — הוא מייצר לקרוא לפונקציות חיצוניות. המודל לא מריץ בקשת קריאה מובנית; הקוד שלכם מריץ אותה ומחזיר את התוצאה.

2.2 הגדרת כלי

כל כלי מוגדר באמצעות JSON schema:

```
{
  "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": []
  }
}
```

היבטים קריטיים של תיאור הכלי:

1. **התיאור הוא מנגנון הבחירה העיקרי.** LLM ("מאחזר מידע לקוח") מובילים לטעויות כשכלים חופפים. תיאורים מינימליים בוחר כלים על בסיס התיאור שלהם.
2. **כללו בתיאור:**
   - מה הכלי עושה ומחזיר
   - פורמטי קלט וערכים לדוגמה
   - מקרי קצה ומגבלות
   - מתי להשתמש בכלי הזה לעומת חלופות דומות
3. יש תיאורים `analyze_content` ו-`analyze_document` מתיאורים זהים או חופפים בין כלים. אם ל-**הימנעו** כמעט זהים — המודל יבלבל ביניהם.
4. **כלים מובנים מול כלי MCP:** סוכנים עשויים להעדיף כלים מובנים (Read, Grep) על פני כלי MCP עם פונקציונליות דומה. כדי למנוע — חזקו את תיאורי כלי MCP: הדגישו יתרונות קונקרטיים, נתונים ייחודיים או הקשר שכלים מובנים לא יכולים לספק.

2.3 הפרמטר `tool_choice`

`tool_choice` שולט באופן שבו המודל בוחר כלים:

ערך	התנהגות	מתי להשתמש

ברירת מחדל לרוב המקרים	המודל מחליט אם לקרוא לכלי או לענות בטקסט	`{"type": "auto"}`
כשצריך פלט מובנה מובטח	המודל **חייב** לקרוא לאיזשהו כלי	`{"type": "any"}`
כשצריך לכפות צעד ראשון / סדר הפעלה	המודל **חייב** לקרוא לכלי ספציפי	`{"type": "tool", "name": "extract_metadata"}`

תרחישים חשובים:

- `tool_choice: "any"` + מספר כלי חילוץ → המודל בוחר את המתאים ביותר אבל אתם עדיין מקבלים פלט מובנה
- (לפני העשרה `extract_metadata` למשל) בחירה כפויה → כשחייבים להבטיח פעולה ראשונה מסוימת

2.4 JSON Schemas לפלט מובנה

שימוש ב-`tool_use` עם JSON schemas מ-Claude. הוא הדרך **האמינה ביותר** לקבל פלט מובנה מ:

- תקין מבחינה תחבירית (אין סוגריים חסרים, פסיקים מיותרים) JSON מבטיח
- שדות (נוכחים required) אוכף את המבנה הנדרש
- **לא** מבטיח נכונות סמנטית (ערכים עדיין יכולים להיות שגויים)

עיצוב סכמה — עקרונות מפתח:

```
{
  "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"]
}
```

כללי עיצוב סכמה:

1. **Required מול optional:** דוחפים את המודל required רק אם המידע תמיד זמין. שדות שאינם כ-required סומנו שדות לבדות ערכים כשהמידע חסר.
2. **שדות nullable:** למידע שעלול להיות חסר. המודל יכול להחזיר ב-`"type": ["string", "null"]` השתמשו במקום להזות `null`.
3. **Enums עם `"other"`:** הוסיפו `"other"` + שדה פירוט כדי לא לאבד נתונים מחוץ לקטגוריות שהגדרתם מראש.
4. **Enum `"unclear"`:** למקרים שבהם המודל לא יכול לבחור קטגוריה בביטחון — `"unclear"` כן הוא טוב יותר מקטגוריה שגויה.

2.5 שגיאות תחביריות מול סמנטיות

סוג שגיאה	דוגמה	מיתון
תחבירית	JSON לא תקין, סוג שדה שגוי	JSON schema (מבטל) עם `tool_use`
סמנטית	סכומים לא מסתכמים, ערך בשדה לא נכון, הזיה	בדיקות אימות, retry-with-feedback, self-correction

---

פרק 3: Claude Agent SDK — בניית מערכות סוכן

> תיעוד: Agent SDK | Hooks | Subagents | Sessions

3.1 מהי לולאת סוכן (Agentic Loop)

לולאת הסוכן היא הדפוס הליבתי לביצוע משימות אוטונומיות. המודל לא רק עונה — הוא מבצע רצף פעולות:

```
1. שלחו בקשה ל-Claude עם כלים
2. קבלו תגובה
3. בדקו stop_reason:
   - "tool_use" -> 1 חזרו לצעד ,הריצו את הכלי, צרפו תוצאה להיסטוריה
   - "end_turn" -> המשימה הושלמה, הציגו תוצאה למשתמש
4. חזרו עד להשלמה
```

זוהי גישה model-driven: מחליט באיזה כלי לקרוא בא הלאה על בסיס הקשר ותוצאות קודמות. שונה מעצי החלטה מקודדים-קשיחים שבהם הרצף קבוע מראש.

אנטי-פטרנים (להימנע):

- ניתוח טקסט של ה-assistant כדי לזהות סיום ("Task completed")
- כתנאי עצירה ראשי (`max_iterations=5`) שימוש בגבול איטרציות שרירותי
- בדיקה אם ה-assistant ייצר תוכן טקסטואלי כאות סיום

גישה נכונה: הסימן האמין היחיד לסיום הוא `stop_reason == "end_turn"`.

3.2 הגדרת `AgentDefinition`

`AgentDefinition` הוא אובייקט הגדרת הסוכן ב-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", ...]
)
```

פרמטרים מרכזיים:

- `name` / `description` — זיהוי ותיאור הסוכן
- `system_prompt` — system prompt עם הוראות
- `allowed_tools` — רשימת הכלים המותרים (עקרון הרשאה מינימלית)

3.3 Hub-and-Spoke: מתאם ותת-סוכנים

ארכיטקטורה רב-סוכנית בדרך כלל בנויה כטופולוגיית hub-and-spoke:

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

המתאם אחראי על:

- פירוק המשימה לתת-משימות
- החלטה אילו תת-סוכנים נדרשים (בחירה דינמית)
- האצלת עבודה לתת-סוכנים
- צבירה ואימות תוצאות
- טיפול בשגיאות ו-retries
- העברת תוצאות למשתמש

עקרון קריטי: לתת-סוכנים יש הקשר מבודד.

- תת-סוכנים **אינם** יורשים אוטומטית את היסטוריית השיחה של המתאם
- כל ההקשר הנדרש חייב להיות **מועבר במפורש** בפרומפט של תת-הסוכן
- תת-סוכנים לא חולקים זיכרון בין קריאות
- כל התקשורת זורמת דרך המתאם (לצורך תצפיתיות ובקרת שגיאות)

3.4 הכלי `Task` ליצירת תת-סוכנים

תת-סוכנים נוצרים דרך הכלי `Task`:

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

העברת הקשר מפורשת היא חובה:

```
# 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]"
```

הפעלה מקבילית: מתאם יכול לקרוא למספר `Task`-תת-הסוכנים רצים במקביל ים בתגובה אחת — תת-הסוכנים רצים במקביל:

```
# 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 Hooks ב-Agent SDK

Hooks מאפשרים יירוט וטרנספורמציה בנקודות ספציפיות במחזור החיים של הסוכן.

PostToolUse מיירט תוצאת כלי לפני שהיא מועברת למודל:

```
# 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
```

Hook חוסם פעולות שמפרות מדיניות **ליירוט קריאות יוצאות**:

```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)
```

מול הוראות בפרומפט hooks: הבדל מרכזי

תכונה	Hooks	הוראות בפרומפט
הבטחה	**דטרמיניסטית** (100%)	**הסתברותית** (<90%, לא 100%)
מתי להשתמש	compliance, כללי עסק קריטיים, פעולות פיננסיות	העדפות כלליות, המלצות, פורמט
דוגמה	חסום החזרים > 500$	"נסה לפתור לפני הסלמה"

כלל: כששגיאה יוצרת תוצאות פיננסיות, משפטיות או בטיחותיות — השתמש ב-hooks, לא בפרומפטים.

פרק 4: Model Context Protocol (MCP)

> תיעוד: MCP | Tools | Resources | Servers

4.1 מה זה MCP

Model Context Protocol (MCP) הוא פרוטוקול פתוח לחיבור מערכות חיצוניות ל-Claude. MCP מגדיר שלושה סוגי משאבים עיקריים:

1. **Tools** — פונקציות שהסוכן יכול לקרוא להן כדי לבצע פעולות (CRUD, קריאות API, הרצת פקודות)
2. **Resources** — נתונים שהסוכן יכול לקרוא להקשר (קטלוגי תוכן, DB, תיעוד, סכמות)
3. **Prompts** — תבניות פרומפט מוגדרות מראש למשימות נפוצות

4.2 שרתי MCP

שרת MCP הוא תהליך שממשש את פרוטוקול MCP ומספק tools/resources. כשמתחברים לשרת MCP:

- כל הכלים מתגלים אוטומטית
- כלים מכל השרתים המחוברים זמינים בו-זמנית
- תיאורי הכלים קובעים איך המודל ישתמש בהם

4.3 הגדרת שרתי MCP

הגדרת פרויקט (`.mcp.json`) — לשימוש צוותי:

```
{
  "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}"
      }
    }
  }
}
```

נקודות מפתח:

- `.mcp.json` נשמר בשורש הפרויקט ומנוהל ב-VCS
- משתמשים לסודות — הטוקנים עצמם לא מוקמטים (`${GITHUB_TOKEN}` משתני סביבה)
- זמין לכל תורמי הפרויקט

הגדרת משתמש (`~/.claude.json`) — לשרתים אישיים/ניסיוניים:

- נשמר ב-home directory של המשתמש
- לא משותף דרך VCS
- מתאים לניסיונות אישיים ובדיקות

בחירת שרתים:

- קהילתיים קיימים MCP העדיפו שרתי — (Jira, GitHub, Slack) לאינטגרציות סטנדרטיות
- בנו שרתים משלכם רק לזרימות עבודה ייחודיות וצוותיות

4.4 הדגל `isError` ב-MCP

כשכלי MCP משתמש ב, הוא נתקל בשגיאה- `isError: true` נכשלה הקריאה בתגובה. זה מסמן לסוכן שהקריאה.

שגיאה מובנית (טוב):

```
{
  "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
  }
}
```

שגיאה גנרית (אנטי-פטרן):

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

שגיאה גנרית לא נותנת לסוכן שום מידע להחלטה — האם לנסות שוב, לשנות שאילתה או להסלים?

4.5 MCP Resources

Resources הם נתונים שסוכן יכול לבקש כדי לקבל הקשר בלי לבצע פעולות:

- קטלוגי תוכן (למשל רשימת כל משימות הפרויקט, ניווט היררכי)
- סכמות מסד נתונים (הבנת מבנה הנתונים)
- תיעוד (API references, מדריכים פנימיים)
- סיכומי issue/task

יתרון Resources: הסוכן לא צריך קריאות כלי חקרניות כדי להבין אילו נתונים קיימים. Resource "מפה" מספק מיידית.

פרק 5: Claude Code — הגדרות וזרימות עבודה

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

5.1 ההיררכיה של CLAUDE.md

Claude Code. יש היררכיה תלת-שכבתית-הוא קובץ/קבצי ההוראות ל CLAUDE.md:

```
1. רמת משתמש: ~/.claude/CLAUDE.md
   - חל רק על המשתמש הזה
   - לא משותף דרך VCS
   - העדפות אישיות וסגנון עבודה

2. רמת פרויקט: .claude/CLAUDE.md או CLAUDE.md בשורש
   - חל על כל תורמי הפרויקט
   - מנוהל דרך VCS
   - סטנדרטי קוד, סטנדרטי בדיקה, החלטות ארכיטקטוניות

3. רמת תיקייה: CLAUDE.md בתת-תיקיות
   - חל בעת עבודה עם קבצים בתיקייה
   - מוסכמות ספציפיות לחלק זה של ה-codebase
```

טעות נפוצה: חבר צוות חדש לא מקבל הוראות פרויקט כי הן הושמו ב-`~/.claude/CLAUDE.md` (רמת משתמש) במקום ב-`.claude/CLAUDE.md` (רמת פרויקט).

5.2 תחביר `@path` (יבוא קבצים)

CLAUDE.md יכול להתייחס לקבצים חיצוניים באמצעות `@path`, מה שהופך את ההגדרה למודולרית:

```
# 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
```

כללי `@path`:

- השתמשו ב-`@` מיד לפני נתיב הקובץ (בלי רווח)
- נתמכים נתיבים יחסיים ומוחלטים
- נתיבים יחסיים נפתרים יחסית לקובץ שמכיל את היבוא
- עומק קינון מקסימלי של יבוא הוא 5

זה מונע כפילויות ומאפשר לכל חבילה לכלול רק סטנדרטי רלוונטיים.

5.3 התיקייה `.claude/rules/`

`.claude/rules/` היא חלופה ל-CLAUDE.md אחד מונוליטי, לארגון כללים לפי נושא:

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

תכונה מרכזית: YAML frontmatter עם `paths` לטעינה מותנית:

```
---
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.
```

איך זה עובד:

- כלל נטען **רק** כש-Claude Code עורך קובץ שמתאים לתבנית `paths`
- זה חוסך הקשר וטוקנים — כללים לא-רלוונטיים לא נטענים
- תבניות glob מאפשרות להחיל מוסכמות לפי סוג קובץ ללא תלות במיקום (אידיאלי לבדיקות פזורות)

ברמת תיקייה CLAUDE.md לעומת `paths` עם `.claude/rules/` -מתי להשתמש ב:

- `.claude/rules/` עם `paths` — כשמוסכמות חלות על קבצים פזורים בהרבה תיקיות (tests, migrations)
- ברמת תיקייה — כשמוסכמות קשורות לתיקייה ספציפית ולא נחוצות במקום אחר CLAUDE.md

5.4 Custom Slash Commands ו-Skills

> **הערה:** בגרסה הנוכחית של Claude Code, פקודות מותאמות אישית (`.claude/commands/`) מאוחדות עם skills (`.claude/skills/`). שני הפורמטים יוצרים פקודות `/name` ומדריך המבחן מתייחס ל-`.claude/commands/` — הפורמט הזה עדיין נתמך.

Slash commands הן תבניות פרומפט הניתנות לשימוש חוזר שמופעלות דרך `/name`:

פורמט `.claude/commands/` (legacy, נתמך):

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

פורמט `.claude/skills/` (נוכחי):

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

פקודות פרויקט (`.claude/commands/` או `.claude/skills/`):

- של הריפו clone וזמינות לכולם בעת נשמרות ב-VCS
- מבטיחות זרימות עבודה עקביות בכל הצוות

פקודות משתמש (`~/.claude/commands/` או `~/.claude/skills/`):

- פקודות אישיות שלא משותפות דרך VCS
- לזרימות עבודה אינדיבידואליות

5.5 Skills — `.claude/skills/`

הן פקודות מתקדמות שמוגדרות דרך Skills של SKILL.md frontmatter:

```
---
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.
```

פרמטרי Frontmatter:

פרמטר	תיאור
`context: fork`	מריץ את ה-skill בתת-סוכן מבודד. פלט verbose הראשי לא מזהם את ההשן
`allowed-tools`	מגביל אילו כלים זמינים (למשל ה-skill אבטחה — לא יכול למחוק קבצים אם לא הותר)
`argument-hint`	רמז שמבקש ארגומנט בעת הפעלה ללא פרמטרים

מתי skill לעומת CLAUDE.md:

- **Skill** — הפעלה on-demand למשימה ספציפית (review, יצירה, ניתוח)
- **CLAUDE.md** — סטנדרטי ומוסכמות כלליים שטעונים תמיד

Skills אישיים (`~/.claude/skills/`):

- צרו ואריאנטים אישיים בשמות שונים כדי לא להשפיע על חברי הצוות

5.6 לעומת ביצוע ישיר (Planning Mode) מצב תכנון

מצב תכנון:

- המודל רק חוקר ומתכנן; לא מבצע שינויים
- משתמש ב-Read, Grep, Glob כדי לחקור את ה-codebase
- מייצר תוכנית מימוש שהמשתמש מאשר
- חקירה בטוחה בלי תופעות לוואי

מתי להשתמש במצב תכנון:

- שינויים גדולים (עשרות קבצים)
- מספר גישות סבירות (microservices: איך להגדיר גבולות?)
- החלטות ארכיטקטוניות (איזה framework? איזה מבנה?)
- codebase (חייבים להבין לפני שמשנים לא מוכר)
- מיגרציות ספרייה שמשפיעות על 45+ קבצים

מתי להשתמש בביצוע ישיר:

- ברור stack trace תיקון בקובץ יחיד עם
- בודדת validation הוספת בדיקת
- שינויים מובנים היטב ולא דו-משמעיים

גישה משולבת:

1. מצב תכנון לחקירה ועיצוב
2. המשתמש מאשר את התוכנית
3. ביצוע ישיר ליישום התוכנית המאושרת

Explore subagent — מתמחה לחקירת ה-codebase-תת-סוכן:

- מבודד פלט verbose מההקשר הראשי
- מחזיר רק סיכום
- מונע אזילת חלון הקשר במשימות רב-שלביות

5.7 הפקודה `/compact`

`/compact` היא פקודה מובנית לדחיסת הקשר:

- מסכמת היסטוריה קודמת כדי לפנות חלון הקשר
- משמשת בסשני חקירה ארוכים שבהם ההקשר מתמלא בפלט כלי verbose
- סיכון: ערכים מספריים מדויקים, תאריכים ופרטים ספציפיים עלולים ללכת לאיבוד בסיכום

5.8 הפקודה `/memory`

`/memory` היא פקודה מובנית לניהול זיכרון בין סשנים:

- לעריכה, מאפשרת לשמור הערות, העדפות והקשר `CLAUDE.md` פותחת את הקובץ
- מידע נמשך בין סשנים ונטען אוטומטית בהפעלה
- שימושית לשמירת מוסכמות פרויקט, העדפות משתמש, פקודות בשימוש תכוף, והקשר עבודה נוכחי
- חלופה להסבר חוזר של אותן הוראות בכל סשן

5.9 Claude Code CLI ל-CI/CD

הדגל `-p` (או `--print`):

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

- יוצא stdout-מצב לא-אינטראקטיבי: מעבד את הפרומפט, מדפיס ל
- לא מחכה לקלט משתמש
- הדרך הנכונה היחידה להריץ Claude בפייפליין CI/CD

פלט מובנה ל-CI:

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

- `--output-format json` — פלט ב-JSON
- `--json-schema` — אימות פלט מול סכמה
- אוטומטית PR-על ה inline ניתן לפרסר את התוצאה כדי לפרסם הערות

שייצר קוד הוא לרוב פחות אפקטיבי בסקירתו (המודל שומר את הקשר Claude **בידוד הקשר של סשן:** אותו סשן עצמאי ל-instance השיקול שלו ופחות סביר שיאתגר את ההחלטות של עצמו). השתמשו ב-review.

הקודמות בהקשר והנחו review-אחרי קומיטים חדשים, כללו את תוצאות ה re-review **מניעת הערות כפולות:** בעת לדווח רק על בעיות חדשות או לא-פתורות את Claude.

5.10 `fork_session` וניהול סשנים

`--resume <session-name>` ממשיך סשן בעל שם:

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

- ממשיך שיחה קודמת עם הקשר שמור
- שימושי לחקירות ארוכות שמתפרסות על מספר סשנים
- סיכון: אם קבצים השתנו מאז הסשן הקודם, תוצאות הכלים עלולות להיות מיושנות

`fork_session` יוצר סניף עצמאי מהקשר משותף:

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

- יורשים הקשר עד נקודת ההסתעפות forks-שני ה
- אחרי כן הם מתפצלים עצמאית
- שימושי להשוואת גישות או בדיקת אסטרטגיות

מתי להתחיל סשן חדש במקום להמשיך:

- תוצאות הכלים מיושנות (קבצים השתנו)
- עבר הרבה זמן וההקשר ירד באיכותו
- עדיף להתחיל מחדש עם "הנה סיכום קצר של מה שמצאנו: ..." במקום להמשיך עם נתוני כלי ישנים

פרק 6: הנדסת פרומפט — טכניקות מתקדמות

> תיעוד: Prompt Engineering | Anthropic Cookbook

6.1 Few-shot Prompting

Few-shot prompting הוא הכללה של 2–4 דוגמאות קלט/פלט בפרומפט כדי להדגים את ההתנהגות הצפויה.

למה few-shot אפקטיבי יותר מתיאורים טקסטואליים:

- הוראה מעורפלת כמו "תהיה מדויק יותר" יכולה להתפרש בהרבה דרכים
- דוגמה מראה באופן חד-משמעי את הפורמט הצפוי ולוגיקת ההחלטה
- המודל מכליל את הדפוס למקרים חדשים (הוא לא רק חוזר על הדוגמאות)

סוגי דוגמאות few-shot ומתי להשתמש בהן:

1. **דוגמאות לתרחישים דו-משמעיים:**

```
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. **דוגמאות לפורמט פלט:**

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

3. **דוגמאות להפרדת קוד מקובל מקוד בעייתי:**

```
// 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. **דוגמאות לחילוץ מפורמטי מסמכים שונים:**

```
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. **דוגמאות למדידות לא-פורמליות:**

```
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"}
```

Few-shot אפקטיבי במיוחד לחילוץ יחידות מדידה לא-פורמליות ולא-סטנדרטיות, שמגוונות מדי לכללים בלבד.

מחמירים לפלט מובנה, הוסיפו כללי JSON schemas-**כללי נורמליזציית פורמט בפרומפט:** בעת שימוש ב נורמליזציה בפרומפט:

```
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
```

תקין תחבירית אבל הערכים לא עקביים JSON-זה מונע שגיאות סמנטיות שבהן ה.

6.2 קריטריונים מפורשים לעומת הוראות מעורפלות

רע (מעורפל):

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

טוב (קריטריונים מפורשים):

```
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)
```

הגדירו קריטריוני חומרה עם דוגמאות:

```
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

Prompt chaining מפרק משימה מורכבת לרצף שלבים ממוקדים:

```
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
```

למה זה חשוב:

- מונע **דילול תשומת לב** — כשהמודל מקבל יותר מדי קבצים בבת אחת, הוא עלול לפספס באגים בחלקם ולתת הערות שטחיות על אחרים
- מבטיח איכות ניתוח עקבית לכל קובץ
- מאפשר ניתוח נפרד של אינטראקציות בין-קבצים

מתי prompt chaining לעומת dynamic decomposition:

- **Prompt chaining** — משימות צפויות וחזרתיות (code review, מיגרציות קבצים)
- **Dynamic decomposition** — חקירות פתוחות שבהן תת-המשימות מתבהרות רק במהלך הביצוע

6.4 דפוס ה-"Interview"

לפני יישום פתרון, Claude שואל שאלות הבהרה:

```
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?"
```

מתי זה שימושי:

- דומיין לא מוכר (fintech, healthcare, מערכות משפטיות)
- משימות עם השלכות לא-מובנות (מצבי כשל, cache, אסטרטגיות)
- מספר גישות סבירות שבחירת הטובה תלויה בהקשר

6.5 Validation ו-Retry-with-Feedback

כשנתונים מחולצים נכשלים באימות:

```
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."
```

מתי retry יהיה אפקטיבי:

- שגיאות פורמט (תאריך בפורמט שגוי)
- שגיאות מבניות (שדה ממוקם במקום הלא נכון)
- אי-עקביות אריתמטיות (המודל יכול לבדוק מחדש)

מתי retry לא יעזור:

- המידע לא קיים במסמך המקור
- ההקשר הנדרש חיצוני (הנתונים במסמך אחר שלא סופק)

ככלי אימות Pydantic: Pydantic היא ספריית Python החשובות לאימות נתונים מבוסס-סכמה. למבחן, הנקודות החשובות:

- **אימות מבני:** סוגים, requiredness, מגבלות enum נבדקות בקוד אחרי קבלת JSON מ-Claude
- **אימות סמנטי:** validators מותאמים אכופים לוגיקה עסקית (סכום פריטים שווה לסך הכל; start_date < end_date)
- **validate–retry לולאות:** עם כשל אימות Pydantic, בנו הודעת שגיאה וקראו שוב ל-Claude עם הקשר השגיאה
- **יצירת JSON Schema:** מודלים של Pydantic יכולים לייצר JSON Schema ל-`tool_use`, מספקים מקור אמת יחיד

6.6 Self-correction

דפוס לזיהוי סתירות פנימיות:

```
{
  "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}
  ]
}
```

מאפשר לטפל באי- `conflict_detected`, המודל מחלץ גם את הערך המוצהר וגם ערך מחושב — אם הם שונים התאמה.

פרק 7: Message Batches API

> תיעוד: Message Batches

7.1 סקירה

ה-Message Batches API מאפשר לכם להגיש batch של בקשות לעיבוד אסינכרוני:

תכונה	ערך
חיסכון	לעומת קריאות סינכרוניות **50%**
חלון עיבוד	עד **24 שעות** (לאיחור SLA אין הבטחת)
Multi-turn tool calling	**לא נתמך** (בקשה אחת = תגובה אחת)
קישור	לקישור בין בקשה לתגובה `custom_id` שדה

7.2 מתי Batch API לעומת Synchronous API

משימה	API	למה
בדיקת PR לפני merge	**Synchronous**	המפתח מחכה; 24 שעות לא מקובל
דוח tech-debt לילה	**Batch**	התוצאה נדרשת לבוקר; חיסכון של 50%
ביקורת אבטחה שבועית	**Batch**	לא דחוף; חיסכון של 50%
Code review אינטראקטיבי	**Synchronous**	נדרשת תגובה מיידית

עיבוד 10,000 מסמכים	**Batch**	עיבוד מסיבי; החיסכון משמעותי

7.3 שימוש ב-`custom_id`

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

`custom_id` מאפשר לכם:

- לקשר את התוצאה למסמך המקורי
- בכשל, להגיש מחדש רק את המסמכים שנכשלו
- להימנע מעיבוד חוזר של מסמכים שהצליחו

7.4 טיפול בכשלים ב-Batches

1. הגישו batch של 100 מסמכים
2. 95 מצליחים; 5 נכשלים (חרגו מגבול ההקשר)
3. זהו כשלים לפי `custom_id`
4. שנו אסטרטגיה (למשל פצלו מסמכים ארוכים ל-chunks)
5. הגישו מחדש רק את 5 המסמכים שנכשלו

7.5 תכנון SLA

אם צריך תוצאה תוך 30 שעות ו-Batch API יכול לקחת עד 24 שעות:

- חלון הגשה: 30 - 24 = **6 שעות**
- Batches חייבים להיות מוגשים לא יאוחר מ-24 שעות לפני ה-deadline
- להגשות תכופות, פצלו לחלונות של 4 שעות

פרק 8: אסטרטגיות פירוק משימות

8.1 Fixed Pipelines (Prompt Chaining)

כל שלב מוגדר מראש:

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

מתי להשתמש:

- מבנה המשימה צפוי (ביקורות תמיד עוקבות אחר אותה תבנית)
- כל השלבים ידועים מראש
- צריך יציבות ושחזוריות

8.2 Dynamic Adaptive Decomposition

תת-משימות נוצרות על בסיס תוצאות ביניים:

```
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
```

מתי להשתמש:

- משימות חקרניות פתוחות
- כשהיקף מלא לא ידוע מראש
- כשכל שלב תלוי בתוצאות הקודם

8.3 Multi-pass Code Review

עם 10+ קבצים ל-PRs:

```
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
```

למה pass יחיד מעל 14 קבצים גרוע:

- דילול תשומת לב: ניתוח עמוק לחלק מהקבצים, שטחי לאחרים
- הערות לא-עקביות: דפוס מסומן בקובץ אחד ומאושר באחר
- באגים מפוספסים: שגיאות ברורות נדלגות עקב עומס קוגניטיבי

פרק 9: הסלמה ו-Human-in-the-Loop

9.1 מתי להסלים לאדם

טריגרים להסלמה (כללים ברורים):

פעולה	מצב
הסלימו מיד; אל תנסו לפתור	"הלקוח מבקש מפורשות "תביא לי מנהל
הסלימו (למשל התאמת מחיר למתחרה כשהמדיניות שותקת)	המדיניות לא מכסה את הבקשה
הסלימו אחרי מספר ניסיונות סביר	הסוכן לא מצליח להתקדם
הסלימו (לא בפרומפט, hook רצוי דרך)	פעולה פיננסית מעל סף
בקשו מזהים נוספים; אל תנחשו	מספר תוצאות בחיפוש לקוח

מה לא טריגר אמין:

למה נכשלת	שיטה לא-אמינה
מצב רוח של לקוח לא בקורלציה עם מורכבות התיק	ניתוח sentiment
המודל יכול לטעות בביטחון; קליברציה גרועה	ביטחון עצמי של המודל (1–10)
over-engineering; שאין training data עלול לדרוש	מסווג אוטומטי

9.2 דפוסי הסלמה

הסלמה מיידית:

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

הסלמה אחרי ניסיון פתרון:

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

הסלמה ניואנסית (acknowledge → resolve → escalate on reiteration):

```
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]
```

עיקרון מפתח: הכירו ברגש קודם, אז הציעו פתרון קונקרטי, והסלימו רק אם הלקוח חוזר על הבקשה לבן אדם. אל תסלימו על ביטוי ראשון של חוסר שביעות רצון (זה לא זהה לבקשת מנהל).

הסלמה על פער במדיניות:

```
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 פרוטוקולי Handoff מובנים

בהסלמה, הסוכן צריך להעביר סיכום מובנה לאדם:

```
{
  "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"
}
```

האופרטור האנושי לא רואה את כל תמלול השיחה — הוא רואה רק את הסיכום הזה. לכן הוא חייב להיות שלם ומכיל בתוך עצמו.

9.4 Confidence Calibration ופיקוח אנושי

למערכות חילוץ נתונים:

1. **ציוני ביטחון ברמת שדה:** המודל מוציא ציון ביטחון לכל שדה מחולץ
2. **קליברציה:** השתמשו בסטים מתויגים לכיוון ספים
3. **ניתוב:**
   - ביטחון גבוה + דיוק יציב -> עיבוד אוטומטי
   - ביטחון נמוך או מקורות עמומים -> סקירה אנושית

Stratified random sampling:

- שוטף של מדגם audit גם לחילוצים בביטחון גבוה, בצעו
- דיוק מצרפי של 97% יכול להסתיר 40% שגיאות לסוג מסמך מסוים
- נתחו דיוק לפי סוג מסמך ולפי שדה, לא רק כללי

פרק 10: טיפול בשגיאות במערכות רב-סוכניות

10.1 קטגוריות שגיאה

קטגוריה	דוגמאות	Retryable	פעולת סוכן
Transient	Timeout, 503, כשל רשת	כן	Retry עם exponential backoff
Validation	פורמט קלט לא תקין, שדה חובה חסר	לא (תקנו קלט)	שנו בקשה ונסו שוב
Business	הפרת מדיניות, חציית סף	לא	הסבירו למשתמש; הציעו חלופה
Permission	גישה נדחתה	לא	הסלימו

10.2 אנטי-פטרנים בטיפול שגיאות

אנטי-פטרן	בעיה	גישה נכונה
סטטוס גנרי "search unavailable"	המתאם לא יכול להחליט איך להתאושש	החזירו סוג שגיאה, query, תוצאות חלקיות, חלופות
השתקה שקטה (תוצאה ריקה = הצלחה)	המתאם חושב שלא היו תוצאות, אבל זה היה כשל	"הבדילו "אין תוצאות" מ"כשל בחיפוש
ביטול כל הזרימה על כשל אחד	אתם מאבדים את כל התוצאות החלקיות	המשיכו עם תוצאות חלקיות; סמנו פערים
Retries אינסופיים בתוך תת-סוכן	latency ובזבוז משאבים	אז הפצה, (2–1 retries) החלמה מקומית למתאם

10.3 שגיאת תת-סוכן מובנית

```
{
  "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"
}
```

זה מספק למתאם את המידע הדרוש כדי להחליט:

- Retry עם query שונה?
- להשתמש בתוצאות חלקיות?
- להאציל לתת-סוכן אחר?
- להמשיך בלי הסקציה הזו ולסמן את הפער?

10.4 הערות כיסוי בסינתזה הסופית

```
## 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]
```

פרק 11: ניהול הקשר במערכות פרודקשן

11.1 חילוץ עובדות לבלוק נפרד

במקום להסתמך על היסטוריית שיחה (שיורדת באיכות בעת סיכום), חלצו עובדות מפתח לבלוק מובנה:

```
=== 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
===
```

כללו את הבלוק הזה בכל פרומפט, ללא קשר לאיך ההיסטוריה מסוכמת.

11.2 קיצוץ תוצאות כלי

אם `lookup_order` מחזיר 40+ שדות אבל אתם צריכים רק 5 למשימה הנוכחית:

```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"]
    }
```

זה חוסך הקשר ומפחית רעש.

11.3 קלט מודע-מיקום

מקמו מידע קריטי תוך התחשבות באפקט lost-in-the-middle:

```
[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 קבצי Scratchpad

בחקירות ארוכות, הסוכן יכול לכתוב ממצאים מרכזיים לקובץ 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
```

כשההקשר יורד באיכותו (או בסשן חדש), הסוכן יכול להתייעץ ב-scratchpad במקום להריץ discovery מחדש.

11.5 האצלה לתת-סוכנים כדי להגן על ההקשר

```
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
```

שכבת הקשר נפרדת: במערכות רב-סוכניות, כל תת-סוכן פועל בתוך תקציב הקשר מוגבל — הוא מקבל רק את גלובלי, ומקצה state המידע הנדרש למשימתו. המתאם פועל כשכבת הקשר נפרדת: הוא מצרף פלטי תת-סוכן, שומר שבו סוכן אחד צורך את החלון במידע לא רלוונטי לאחרים, "context leakage". זה מונע הקשר.

תקציבי הקשר מוגבלים לתת-סוכנים:

- שלחו הקשר מינימלי: משימה ספציפית + נתונים נדרשים
- של נתונים גולמיים dumps הנחו את תת-הסוכן להחזיר תוצאות מובנות, לא
- כדי להגביל את ערכת הכלים של תת-הסוכן — פחות כלים פירושו פחות הסחות `allowedTools`-השתמשו ב דעת ועלות הקשר נמוכה יותר

11.6 מובנה (להתאוששות מקריסה) State שמירת

שלו למיקום ידוע state-כל סוכן מייצא את ה:

```
// 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"]
}
```

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

בהמשך manifest המתאם טוען.

פרק 12: שימור Provenance (מקור הציטוט)

12.1 בעיית אובדן Attribution

בעת סיכום תוצאות ממקורות מרובים, הקישור "טענה → מקור" יכול ללכת לאיבוד:

```
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 טיפול בנתונים סותרים

כששני מקורות מספקים ערכים שונים:

```
{
  "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"
}
```

ותנו למתאם להחליט attribution אל תבחרו ערך אחד שרירותית. שמרו את שניהם עם.

12.3 כללו תאריכים לפרשנות נכונה

בלי תאריכים, הבדלים זמניים יכולים להתפרש בטעות כסתירות:

```
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 רינדור לפי סוג תוכן

אל תכפו הכל לפורמט אחד:

- נתונים פיננסיים -> טבלאות
- חדשות וניתוח -> פרוזה
- ממצאים טכניים -> רשימות מובנות
- סדרות זמן -> סדר כרונולוגי

פרק 13: כלים מובנים של Claude Code

13.1 הפניית בחירת כלי

משימה	כלי	דוגמה
מציאת קבצים לפי שם/תבנית	**Glob**	`**/*.test.tsx`, `src/components/**/*.ts`
חיפוש בתוך קבצים	**Grep**	שם פונקציה, הודעת שגיאה, import
קריאת קובץ במלואו	**Read**	טעינת קובץ לניתוח
כתיבת קובץ חדש	**Write**	יצירת קובץ מאפס
עריכת קובץ קיים מדויקת	**Edit**	ספציפי דרך התאמת טקסט ייחודית החלפת snippet
הרצת פקודת shell	**Bash**	git, npm, הרצת בדיקות, build

13.2 אסטרטגיית חקירה הדרגתית

אל תקראו את כל הקבצים בבת אחת. בנו הבנה הדרגתית:

```
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 Fallback: Read + Write במקום Edit

נכשל בגלל התאמת טקסט לא-ייחודית Edit-כש:

1. Read — טענו את תוכן הקובץ המלא
2. שנו את התוכן תוכנתית
3. Write — כתבו את הגרסה המעודכנת

חלק II: הערות לפי דומיין מבחן

דומיין 1: ארכיטקטורת סוכנים ואורקסטרציה (27%)

1.1 תכנון לולאות סוכן לביצוע משימות אוטונומיות

ידע מפתח:

- בדקו Claude, מחזור חיים של לולאת סוכן: שלחו בקשת `stop_reason` (`"tool_use"` מול `"end_turn"`), הריצו כלים, החזירו תוצאות לאיטרציה הבאה
- תוצאות כלי מצורפות להיסטוריית השיחה כדי שהמודל יחליט על הפעולה הבאה
- קבלת החלטות model-driven (Claude בוחר את הכלי הבא) לעומת עצי החלטה מקודדים-קשיחים

מיומנויות מפתח:

- בקרת זרימה: המשיכו את הלולאה כש-`stop_reason = "tool_use"` ועצרו ב-`"end_turn"`
- צירוף תוצאות כלי להקשר בין איטרציות
- לסיום, שימוש בגבולות איטרציה שרירותיים כמנגנון עצירה ראשי assistant אנטי-פטרנים להימנע: ניתוח טקסט

1.2 אורקסטרציה של מערכות רב-סוכניות (מתאם–תת-סוכן)

ידע מפתח:

- ארכיטקטורת hub-and-spoke: המתאם מחזיק בכל התקשורת בין סוכנים, טיפול שגיאות וניתוב
- תת-סוכנים פועלים עם הקשר מבודד — הם לא יורשים אוטומטית את היסטוריית המתאם
- אחריות המתאם: פירוק משימה, האצלה, צבירת תוצאות, בחירה דינמית של תת-סוכנים
- סיכון של פירוק מצומצם מדי על ידי המתאם

מיומנויות מפתח:

- חלקו כיסוי מחקרי בין תת-סוכנים כדי למזער כפילות
- מימשו לולאות שיפור איטרטיביות (המתאם מעריך סינתזה ומנתב מחדש משימות)
- נתבו את כל התקשורת דרך המתאם לצורך תצפיתיות

1.3 הגדרת קריאות תת-סוכן, העברת הקשר ויצירה

ידע מפתח:

- הכלי `Task` יוצר תת-סוכנים; `allowedTools` של המתאם חייב לכלול `"Task"`
- הקשר תת-סוכן חייב להיכלל מפורשות בפרומפט; תת-סוכנים לא יורשים הקשר הורה
- הגדרת `AgentDefinition`: תיאורים, system prompts, מגבלות כלים
- ניהול סשנים דרך `fork_session` לחקירת חלופות

מיומנויות מפתח:

- כללו פלטים מלאים מסוכנים קודמים בפרומפט תת-הסוכן
- השתמשו בפורמטים מובנים להפרדת נתונים ממטא-דאטה בהעברת הקשר
- צרו תת-סוכנים מקבילים דרך מספר קריאות `Task` בתור מתאם יחיד
- כתבו פרומפטי מתאם במונחי מטרות וקריטריוני איכות במקום הוראות שלב-אחר-שלב

1.4 מימוש זרימות עבודה רב-שלביות עם אכיפה ו-handoff

ידע מפתח:

- לבין **הנחיית פרומפט** לסדר זרימת עבודה (hooks, preconditions) ההבדל בין **אכיפה תוכנתית**
- כשצריך הבטחות דטרמיניסטיות (למשל אימות זהות לפני פעולות פיננסיות), פרומפטים לבדם לא מספיקים
- פרוטוקולי handoff מובנים בהסלמה (זיהוי לקוח, סיבה, פעולה מומלצת)

מיומנויות מפתח:

- Preconditions תוכנתיים שחוסמים קריאות במורד הזרימה עד שצעדים קודמים הושלמו (למשל חסום `process_refund` עד ש-`get_customer` מחזיר ID מאומת)
- פרקו בקשות לקוח רב-היבטיות לפריטים נפרדים
- הפיקו סיכומים מובנים בהסלמה לאדם

1.5 ליירוט קריאות כלי ונורמליזציה ב-Agent SDK Hooks

ידע מפתח:

- ליירוט תוצאות כלי לפני שהמודל צורך אותן (למשל `PostToolUse` Hook דפוסי)
- שמיירטים קריאות יוצאות לאכיפת כללי compliance (למשל חסום החזרים מעל סף) Hooks
- מספקים **הבטחות דטרמיניסטיות** לעומת הוראות פרומפט שמספקות Hooks **הסתברותי compliance**

מיומנויות מפתח:

- לנורמליזציית פורמטי נתונים hooks `PostToolUse` (Unix timestamps, ISO 8601, קודי סטטוס מספריים)
- ליירוט שחוסמים פעולות שמפרות מדיניות עם ניתוב להסלמה Hooks
- מובטח compliance על פני פרומפטים כשכללי עסק דורשים hooks בחרו

1.6 אסטרטגיות פירוק משימות לזרימות מורכבות

ידע מפתח:

- לעומת **פירוק אדפטיבי דינמי** מבוסס תוצאות ביניים (prompt chaining) **Fixed pipelines**
- Prompt chaining: צעדים סדרתיים (אינטגרציה pass נתחו כל קובץ בנפרד, ואז הריצו)
- תוכניות חקירה אדפטיביות שיוצרות תת-משימות לפי מה שהתגלה

מיומנויות מפתח:

- לסקירות צפויות רב-היבטיות; השתמשו בפירוק דינמי לחקירות פתוחות prompt chaining-השתמשו ב
- אינטגרציה נפרד לבין-קבצים pass גדולים לניתוח לפי קובץ ועוד code reviews פצלו
- פרקו משימות פתוחות: מפו מבנה קודם, אז בנו תוכנית מתועדפת

1.7 Session State, Resume ו-Fork

ידע מפתח:

- `--resume <session-name>` להמשך סשנים בעלי שם
- `fork_session` ליצירת ענפי חקירה עצמאיים מהקשר משותף
- חשיבות יידוע הסוכן על שינויי קבצים בעת המשך סשנים
- סשן חדש עם סיכום מובנה יכול להיות אמין יותר מהמשך עם תוצאות מיושנות

מיומנויות מפתח:

- להמשך סשני חקירה בעלי שם `--resume`-השתמשו ב
- להשוואת גישות במקביל `fork_session`-השתמשו ב
- לעומת התחלת סשן חדש (תוצאות מיושנות) (הקשר עדיין רלוונטי) resume בחרו בין

דומיין 2: עיצוב כלים ואינטגרציית MCP (18%)

2.1 עיצוב ממשקי כלי עם תיאורים ברורים

ידע מפתח:

- משתמש בו לבחירת כלים; תיאורים מינימליים מובילים לבחירה לא- LLM-תיאורי כלי הם **המנגנון העיקרי** ש אמינה
- חשיבות הכללת פורמטי קלט, שאילתות לדוגמה, מקרי קצה וגבולות תחולה
- תיאורים דו-משמעיים או חופפים גורמים לניתוב שגוי
- יכול ליצור אסוציאציות לא-רצויות לכלים system prompt ניסוח

מיומנויות מפתח:

- כתבו תיאורים שמבדילים בבירור כל כלי מחלופות דומות

- שנו שמות כלים כדי לבטל חפיפה פונקציונלית (למשל `analyze_content` -> `extract_web_results`)
- פצלו כלים כלליים למתמחים עם חוזי קלט/פלט ברורים

2.2 מימוש תגובות שגיאה מובנות לכלי MCP

ידע מפתח:

- הדגל `isError` בתגובות כלי MCP
- ההבדל בין **שגיאות transient** (timeouts), **שגיאות validation** (קלט שגוי), **שגיאות business** (הפרות מדיניות), ו**שגיאות גישה/הרשאה**
- מונעות החלטות התאוששות נכונות ("Operation failed") שגיאות גנריות
- ההבדל בין שגיאות retryable ל-non-retryable

מיומנויות מפתח:

- החזירו מטא-דאטה מובנית כמו `errorCategory` (transient/validation/permission), `isRetryable`, והודעה קריאה לאדם
- עם הסברים ברורים למשתמש business-rule להפרות השתמשו ב-`retryable: false`
- הפיצו רק שגיאות שהם לא יכולים לפתור transient; בצעו החלמה מקומית בתוך תת-סוכנים לכשלים
- מתוצאות ריקות תקפות (אין תוצאות) הבדילו כשלי גישה (החלטת retry)

2.3 הקצאת כלים בין סוכנים והגדרת `tool_choice`

ידע מפתח:

- יותר מדי כלים לסוכן (למשל 18 במקום 4–5) **מפחית** אמינות בחירת כלים
- סוכנים עם כלים מחוץ להתמחותם נוטים להשתמש בהם לרעה
- בין-תפקידים utilities גישת כלים מצומצמת: רק כלים רלוונטיים לתפקיד ועוד ערכה מוגבלת של
- `tool_choice`: `"auto"`, `"any"`, ובחירה כפויה (`{"type": "tool", "name": "..."}`)

מיומנויות מפתח:

- הגבילו את ערכת הכלים של כל תת-סוכן למה שרלוונטי לתפקידו
- החליפו כלים כלליים בחלופות מוגבלות (למשל `fetch_url` -> `load_document`)
- להבטחת קריאת כלי במקום תשובה טקסטואלית השתמשו ב-`tool_choice: "any"`
- כפו כלי ספציפי כדי להבטיח סדר ביצוע

2.4 שילוב שרתי MCP ב-Claude Code וזרימות סוכן

ידע מפתח:

- היקף שרת MCP: פרויקט (`.mcp.json`) לצוותים לעומת משתמש (`~/.claude.json`) לניסיונות
- החלפת משתני סביבה ב-`.mcp.json` (למשל `${GITHUB_TOKEN}`) לניהול סודות
- כלים מכל שרתי MCP המחוברים מתגלים בחיבור וזמינים בו-זמנית
- להפחתת קריאות כלי חקרניות (סיכומי משימות, סכמות DB) "כ"קטלוגי תוכן MCP resources

מיומנויות מפתח:

- הגדירו שרתי MCP משותפים ב-`.mcp.json` עם טוקנים מבוססי env-var של הפרויקט
- שמרו שרתים אישיים/ניסיוניים ב-`~/.claude.json`
- קהילתיים על פני שרתים מותאמים אישית לאינטגרציות סטנדרטיות MCP העדיפו שרתי

2.5 בחירה ויישום של כלים מובנים (Read, Write, Edit, Bash, Grep, Glob)

ידע מפתח:

- **Grep**: חיפוש בתוך תוכן קבצים (imports, שמות פונקציות, הודעות שגיאה)
- **Glob**: מציאת קבצים לפי תבניות שם/סיומת
- **Read/Write**: פעולות מלאות על קבצים; **Edit**: שינויים מדויקים דרך התאמות טקסט ייחודיות
- אם Edit נכשל עקב התאמות לא-ייחודיות, fallback ל-Read + Write

מיומנויות מפתח:

- השתמשו ב-Grep לחיפוש תוכן וב-Glob לגילוי קבצים לפי תבניות
- בנו הבנה הדרגתית: Grep entry points, אז Read לעקיבת flows
- עקבו אחר שימוש בפונקציות דרך מודולי wrapper

דומיין 3: הגדרות Claude Code (20%) וזרימות עבודה

3.1 הגדרת CLAUDE.md עם היררכיה, היקף וארגון מודולרי

ידע מפתח:

- CLAUDE.md היררכיית: משתמש (`~/.claude/CLAUDE.md`), פרויקט (`.claude/CLAUDE.md` או CLAUDE.md בשורש), ורמת תיקייה (CLAUDE.md בתת-תיקיות)
- הגדרות ברמת משתמש חלות רק על משתמש אחד ולא משותפות דרך VCS
- תחביר `@path` להתייחסות לקבצים חיצוניים (למשל `@./standards/coding-style.md`) למודולריות CLAUDE.md
- מונוליטי CLAUDE.md לקבצי כללים ממוקדי-נושא במקום תיקיית `.claude/rules/`

מיומנויות מפתח:

- אבחנו בעיות היררכיה (חבר צוות חדש מפספס הוראות כי הן ברמת משתמש במקום פרויקט)
- בכל חבילה standards להכללה סלקטיבית של (למשל `@./standards/testing.md`) השתמשו ב-`@path`
- פצלו CLAUDE.md גדול לקבצי `.claude/rules/` מרובים (testing.md, api-conventions.md, deployment.md)

3.2 יצירה והגדרה של Custom Slash Commands ו-Skills

ידע מפתח:

- **פקודות משתמש** ב (VCS משותפות דרך) `.claude/commands/` -לעומת **פקודות פרויקט** ב-`~/.claude/commands/`
- Skills ב-`.claude/skills/` עם frontmatter של `SKILL.md`: `context: fork`, `allowed-tools`, `argument-hint`
- `context: fork` בהקשר תת-סוכן מבודד כדי שלא יזהם את הסשן הראשי skill-מריץ את ה
- בשמות שונים `~/.claude/skills/` -אישיים יכולים לחיות ב skills ואריאנטי

מיומנויות מפתח:

- כדי שכל הצוות יקבל אותן `.claude/commands/` -של פרויקט ב slash commands שמרו
- verbose עם פלט skills לבידוד `context: fork` -השתמשו ב
- יכול להשתמש בהם skill להגבלת אילו כלים `allowed-tools` -השתמשו ב
- כדי לדרבן מפתחים לפרמטרים נדרשים `argument-hint` -השתמשו ב

3.3 לטעינת מוסכמות מותנית Path Specific שימוש בכללי

ידע מפתח:

- קבצי `.claude/rules/` יכולים לכלול frontmatter YAML עם `paths` להפעלת כללים על בסיס תבניות glob
- נטענים **רק** בעריכת קבצים תואמים, חוסכים הקשר וטוקנים path-כללים מסומני
- ברמת תיקייה כשמוסכמות חלות על הרבה CLAUDE.md יכולים להיות עדיפים על glob-מבוססי path כללי (למשל tests) תיקיות

מיומנויות מפתח:

- צרו קבצי `.claude/rules/` עם `paths: ["terraform/**/*"]` לטעינה רק בעבודה על קבצים תואמים
- השתמשו בתבניות glob (`**/*.test.tsx`) להחלת מוסכמות לפי סוג קובץ ללא תלות במיקום
- ברמת תיקייה כשמוסכמות מתפרסות על ה CLAUDE.md על ה-path העדיפו כללים מסומני codebase-

3.4 מתי להשתמש במצב תכנון מול ביצוע ישיר

ידע מפתח:

- **מצב תכנון**: למשימות מורכבות עם שינויים גדולים, מספר גישות סבירות, והחלטות ארכיטקטוניות
- **ביצוע ישיר**: לשינויים פשוטים ומובנים היטב (בודד validation למשל הוספת)
- מצב תכנון מאפשר חקירה בטוחה של ה codebase- לפני ביצוע שינויים
- מבודד פלט Explore subagent מילולי discovery

מיומנויות מפתח:

- השתמשו במצב תכנון למשימות עם השלכות ארכיטקטוניות (מיגרציות שנוגעות ל-45+ קבצים, microservices)
- ברור וקובץ יחיד stack trace השתמשו בביצוע ישיר לתיקונים עם

- כדי למנוע אזילת חלון הקשר במשימות רב-שלביות Explore subagent-השתמשו ב
- שלבו גישות: תכנון לגילוי, אז ביצוע ליישום

3.5 שיפור איטרטיבי לשיפור פרוגרסיבי

ידע מפתח:

- דוגמאות קונקרטיות של קלט/פלט הן הדרך האפקטיבית ביותר לתקשר ציפיות
- על בסיס כשלים iterate **איטרציה מונחית-בדיקות**: כתבו בדיקות קודם, אז
- שואל שאלות כדי להעלות שיקולי עיצוב לא-מובנים Claude :"interview"דפוס ה
- מתי לספק את כל הבעיות בהודעה אחת (תלויות הדדית) לעומת ברצף (עצמאיות)

מיומנויות מפתח:

- ספקו 2–3 דוגמאות קונקרטיות של קלט/פלט להבהרת דרישות טרנספורמציה
- בנו ערכות בדיקה עם התנהגות צפויה, מקרי קצה ודרישות ביצועים לפני מימוש
- (מצבי כשל ,cache invalidation) להעלאת היבטי עיצוב interview השתמשו בדפוס
- קונקרטיים עם קלטים לדוגמה ופלטים צפויים למקרי קצה test cases ספקו

3.6 שילוב Claude Code בפייפליין CI/CD

ידע מפתח:

- למצב לא-אינטראקטיבי בפייפליין אוטומטי (`--print` או) `-p` הדגל
- `--output-format json` ו-`--json-schema` לפלט מובנה ב CI-
- CLAUDE.md שמופעל מ Claude Code-ל CI- (review סטנדרטי בדיקות, קריטריוני) מספק הקשר פרויקט
- עצמאי instance-**בידוד הקשר סשן**: אותו סשן שייצר קוד הוא פחות אפקטיבי בסקירתו מ

מיומנויות מפתח:

- כדי להימנע מתליה על קלט אינטראקטיבי `-p` עם CI-ב Claude Code הריצו
- (PR על inline למשל הערות) לתוצאות מובנות `--json-schema` + `--output-format json`-השתמשו ב
- קודמות בעת הרצה מחדש אחרי קומיטים חדשים (דווחו רק על בעיות חדשות/לא-מתוקנות) review כללו תוצאות
- לשיפור איכות יצירת בדיקות CLAUDE.md-זמינים ב fixtures-תעדו סטנדרטי בדיקות ו
- כללו קבצי בדיקה קיימים בהקשר בעת יצירת בדיקות חדשות כדי להימנע מכפילות ולשמור על עקביות סגנון

דומיין 4: הנדסת פרומפט ופלט מובנה (20%)

4.1 עיצוב פרומפטים עם קריטריונים מפורשים לשיפור דיוק

ידע מפתח:

- קריטריונים מפורשים אפקטיביים יותר מהוראות מעורפלות (למשל "סמן הערות רק כשהן סותרות את הקוד" מול "בדוק דיוק הערות")
- הנחיה גנרית כמו "תהיה שמרני יותר" עובדת פחות טוב מקריטריונים קטגוריאליים קונקרטיים
- גבוהים בקטגוריות מסוימות פוגעים באמון false-positive על אמון מפתחים: שיעורי false positives ההשפעה של בקטגוריות מדויקות

מיומנויות מפתח:

- הגדירו קריטריוני review: (סגנון מינורי) לעומת מה להתעלם (באגים, אבטחה)
- בטלו זמנית קטגוריות עם שיעורי false-positive גבוהים
- הגדירו קריטריוני חומרה מפורשים עם דוגמאות קוד לכל רמה

4.2 שימוש ב-Few-shot Prompting לשיפור עקביות פלט

ידע מפתח:

- few-shot דוגמאות הן השיטה האפקטיבית ביותר להפקת פלט עקבי בפורמט וניתן לפעולה
- Few-shot יכול להדגים טיפול במקרים דו-משמעיים (בחירת כלי, פערים בכיסוי בדיקות)
- Few-shot עוזר למודל להכליל לדפוסים חדשים במקום רק לחזור על ברירות מחדל
- Few-shot יכול להפחית הזיות במשימות חילוץ

מיומנויות מפתח:

- ספקו 2–4 דוגמאות ממוקדות לתרחישים דו-משמעיים עם הצדקה
- few-shot שמדגימות את פורמט הפלט (מיקום, בעיה, חומרה, תיקון מוצע) כללו דוגמאות
- ספקו דוגמאות שמבדילות דפוסי קוד מקובלים מבעיות אמיתיות
- ספקו דוגמאות לחילוץ נכון ממסמכים עם מבנים שונים

4.3 אכיפת פלט מובנה עם `tool_use` ו-JSON Schemas

ידע מפתח:

- `tool_use` הוא הדרך האמינה ביותר להבטיח פלט תואם-סכמה ולבטל שגיאות תחביר עם JSON Schemas JSON
- הוא חייב לקרוא לכלי; בחירה כפויה `"any"` המודל יכול להחזיר טקסט; עם `tool_choice: "auto"` בוחרת כלי ספציפי
- מחמירים מבטלים שגיאות תחביר אך לא מונעים שגיאות סמנטיות (סכומים לא מסתכמים; JSON Schemas ערכים בשדות שגויים)

- ועוד שדה פירוט להרחבה "other" עם optional; enums לעומת required עיצוב סכמה: שדות

מיומנויות מפתח:

- הגדירו כלי חילוץ עם JSON Schemas ופרסרו נתונים מתוצאות `tool_use`
- להבטחת פלט מובנה כשקיימות מספר סכמות השתמשו ב-`tool_choice: "any"`
- כפו קריאה לכלי ספציפי: `tool_choice: {"type": "tool", "name": "extract_metadata"}`
- כשהמקור עשוי לא להכיל מידע, כדי להימנע מבדיית ערכים/nullable הפכו שדות לאופציונליים
- ועוד שדות פירוט לקטגוריזציה הניתנת להרחבה `"other"`-ו `"unclear"` כמו enum השתמשו בערכי

4.4 מימוש Validation, Retries ולולאות פידבק לאיכות חילוץ

ידע מפתח:

- Retry-with-error-feedback: לכוון תיקונים קונקרטיות בפרומפט ה validation כללו שגיאות
- Retries לא אפקטיביים כשהמידע פשוט נעדר מהמקור
- עיצוב לולאת פידבק: עקבו אחר הדפוס שהפעיל ממצא (`detected_pattern`)
- מטופלות על ידי `tool_use`) שגיאות סמנטיות (סכומים לא מסתדרים) לעומת שגיאות תחביר)

מיומנויות מפתח:

- ספציפיות validation עם המסמך המקורי, חילוץ שגוי ושגיאות follow-up פרומפטי
- לא יהיה אפקטיבי (המידע הנדרש רק במסמך חיצוני) retry זהו מתי
- כללו שדות `detected_pattern` במצאים לניתוח false positives
- לזיהוי אי-התאמות `stated_total` וגם `calculated_total` על ידי חילוץ גם self-correction עצבו

4.5 עיצוב אסטרטגיות יעילות לעיבוד Batch

ידע מפתח:

- Message Batches API: 50% חיסכון, חלון עיבוד עד 24 שעות, אין הבטחות SLA על latency
- עיבוד batch מתאים למשימות חוסמות (דוחות לילה, audits) ולא מתאים למשימות לא-חוסמות (בדיקות לפני merge)
- Batch API לא תומך ב multi-turn tool calling בתוך בקשה יחידה
- שדות `custom_id` מקשרים request/response בתוך batches

מיומנויות מפתח:

- השתמשו ב API-סינכרוני לבדיקות חוסמות; השתמשו ב Batch API לעומסי לילה/שבועיים
- תכננו cadence הגשת batch על בסיס צרכי SLA (למשל חלונות של 4 שעות להבטחת 24 שעות עם עיבוד 30 שעות)
- טפלו בכשלים על ידי הגשה מחדש רק של מסמכים שנכשלו (`custom_id` מזוהים לפי)
- iterate על פרומפטים תוך שימוש במדגם לפני הרצת עיבוד בקנה מידה גדול

4.6 עיצוב ארכיטקטורות review רב-instance ורב-pass

ידע מפתח:

- מגבלות self-review: המודל שומר את הקשר השיקול שלו ופחות סביר שיאתגר את ההחלטות של עצמו
- ביקורת עצמאיים (בלי הקשר היצירה) טובים יותר במציאת בעיות עדינות Instances
- Multi-pass review: אינטגרציה בין-קבצים כדי להימנע מדילול תשומת לב pass ניתוח מקומי לכל קובץ ועוד

מיומנויות מפתח:

- שני עצמאי לסקירת שינויים בלי הקשר היצירה Claude instance-השתמשו ב
- אינטגרציה לניתוח זרימת נתונים בין-קבצים passes לפי קובץ ועוד passes-פצלו ביקורות רב-קבציות ל
- אימות עם ביטחון עצמי לניתוב ביקורות באופן מכויל passes-השתמשו ב

דומיין 5: ניהול הקשר ואמינות (15%)

5.1 ניהול הקשר שיחה לשימור מידע קריטי

ידע מפתח:

- סיכוני סיכום פרוגרסיבי: ערכים מספריים, אחוזים ותאריכים מתעבים לסיכומים מעורפלים
- מודלים מעבדים אמינות התחלה וסוף של קלטים ארוכים, אבל עלולים לפספס lost-in-the-middle: אפקט ממצאים מהאמצע
- פלטי כלי יכולים להצטבר בהקשר באופן לא-פרופורציוני לרלוונטיות (40+ שדות כש-5 נחוצים)
- עוקבות API חשיבות שליחת היסטוריית השיחה המלאה בבקשות

מיומנויות מפתח:

- קבוע מחוץ להיסטוריה המסוכמת "case facts" חלצו עובדות טרנזקציוניות לבלוק
- לשדות רלוונטיים verbose קצצו פלטי כלי
- מקמו ממצאי מפתח בתחילת נתונים מצורפים עם כותרות סקציה מפורשות
- דרשו שתת-סוכנים יכללו מטא-דאטה (תאריכים, מקורות) בפלטים מובנים

5.2 עיצוב דפוסי הסלמה אפקטיביים ופתרון דו-משמעיות

ידע מפתח:

- טריגרי הסלמה מתאימים: בקשה מפורשת לאדם, פערים/חריגים במדיניות, חוסר יכולת להתקדם
- הסלמה מיידית (בקשה מפורשת) לעומת ניסיון פתרון (בתחום הסוכן)
- וביטחון עצמי של המודל הם פרוקסי לא-אמינים למורכבות תיק sentiment ניתוח
- מספר התאמות לקוח דורש בקשת מזהים נוספים, לא ניחוש היוריסטי

מיומנויות מפתח:

- קריטריוני הסלמה מפורשים עם דוגמאות few-shot ב-system prompt
- בצעו בקשות מפורשות לאדם מיד ללא חקירה נוספת
- הסלימו כשהמדיניות דו-משמעית או שותקת לבקשה ספציפית
- בקשו מזהים נוספים כשתוצאות כלי מכילות מספר התאמות

5.3 מימוש אסטרטגיות הפצת שגיאות במערכות רב-סוכניות

ידע מפתח:

- הקשר שגיאה מובנה (סוג כשל, query, חלופות, תוצאות חלקיות) מאפשר התאוששות חכמה יותר של מתאם
- הבדילו כשלי גישה (timeouts החלטת דורשים retry) מתוצאות ריקות תקפות (אין התאמות)
- סטטוסי שגיאה גנריים ("search unavailable") מסתירים הקשר חשוב מהמתאם
- השתקה שקטה או ביטול כל הזרימה על כשל אחד הם אנטי-פטרנים

מיומנויות מפתח:

- החזירו הקשר שגיאה מובנה: סוג כשל, מה נוסה, תוצאות חלקיות, חלופות אפשריות
- הבדילו כשלי גישה מתוצאות ריקות תקפות
- הפיצו רק שגיאות לא-ניתנות-להתאוששות עם תוצאות חלקיות; בצעו החלמה מקומית בתת-סוכנים לכשלים transient
- סמנו כיסוי בסינתזה: מה מבוסס היטב לעומת היכן נשארו פערים

5.4 ניהול הקשר יעיל בחקירת Codebases גדולים

ידע מפתח:

- ירידה באיכות הקשר בסשנים ארוכים: המודל מתחיל להפיק תשובות לא-יציבות ומתייחס ל"דפוסים אופייניים" במקום למחלקות ספציפיות
- משמרים ממצאי מפתח מעבר לגבולות הקשר Scratchpad קבצי
- מילולי discovery האצלה לתת-סוכנים מבודדת פלט
- מובנה מאפשרת התאוששות מקריסה state שמירת

מיומנויות מפתח:

- צרו תת-סוכנים לשאלות ספציפיות תוך שמירה על תיאום ברמה גבוהה בסוכן הראשי
- לאחסון ממצאי מפתח והפניה אליהם בהמשך scratchpad השתמשו בקבצי
- סכמו ממצאי מפתח לפני יצירת תת-סוכני שלב הבא
- להפחתת שימוש בהקשר בחקירות ארוכות `/compact`-השתמשו ב

5.5 עיצוב זרימות עם פיקוח אנושי וקליברציית ביטחון

ידע מפתח:

- מדדים מצרפיים (למשל 97% דיוק כללי) יכולים להסוות ביצועים גרועים בסוגי מסמכים או שדות ספציפיים

- מודד שיעורי שגיאה בחילוצים בביטחון גבוה Stratified random sampling
- קליברציית ביטחון ברמת שדה תוך שימוש בסטים מתויגים
- אמתו דיוק לפי סוג מסמך ומגזר שדה לפני אוטומציה

מיומנויות מפתח:

- לזיהוי דפוסי שגיאה חדשים stratified random sampling מימשו
- נתחו דיוק לפי סוג מסמך ושדה לאימות ביצועים יציבים
- תוך שימוש בנתונים מתויגים review הוציאו ציוני ביטחון ברמת שדה וכיילו ספי
- נתבו חילוצים בביטחון נמוך או ממקורות עמומים לסקירה אנושית

5.6 שימור Provenance וטיפול באי-ודאות בסינתזה רב-מקורית

ידע מפתח:

- "אובד בסיכום ללא שימור מיפוי "טענה ← מקור Attribution
- מיפויים מובנים חייבים להישמר בעת צבירה
- במקום בחירה שרירותית של ערך אחד attribution טפלו בסטטיסטיקות סותרות על ידי הערה על קונפליקטים עם
- כללו תאריכי פרסום/איסוף כדי להימנע מקריאה שגויה של הבדלים זמניים כסתירות

מיומנויות מפתח:

- "דרשו שתת-סוכנים יוציאו מיפויי "טענה ← מקור (URL, ציטוטים ,שם מסמך)
- בנו דוחות שמפרידים ממצאים יציבים מאלה הנמצאים במחלוקת
- שמרו ערכים סותרים עם הערות והעבירו אותם למתאם להתאמה
- כללו תאריכי פרסום לפרשנות זמנית נכונה
- רנדרו תוכן לפי סוג: נתונים פיננסיים כטבלאות, חדשות כפרוזה, ממצאים טכניים כרשימות מובנות

דוגמאות שאלות מבחן עם הסברים

שאלה 1 (תרחיש: Customer Support Agent)

מצב: הנתונים מראים שב-12% מהמקרים הסוכן מדלג על `get_customer` וקורא ל-`lookup_order` רק עם שם הלקוח, מה שמוביל להחזרים שגויים.

איזה שינוי הכי אפקטיבי?

- A) הוסיפו precondition שחוסם תוכנתי `lookup_order` ו-`process_refund` עד שמתקבל ID מ-`get_customer` **[נכון]**
- B) שפרו את ה-system prompt
- C) הוסיפו דוגמאות few-shot
- D) מימשו מסווג ניתוב

למה A: כשלוגיקה עסקית קריטית דורשת רצף כלים מסוים, תוכנה מספקת **הבטחות דטרמיניסטיות** שגישות מבוססות-פרומפט (B, C) לא יכולות. D מטפל בזמינות, לא בסדר כלים.

שאלה 2 (תרחיש: Customer Support Agent)

מצב: הסוכן קורא ל-`get_customer` במקום ל-`lookup_order` לשאלות הקשורות בהזמנה. תיאורי הכלים מינימליים ודומים.

מה הצעד הראשון?

- A) דוגמאות few-shot
- B) **[נכון]** הרחיבו את תיאור כל כלי עם פורמטי קלט, דוגמאות וגבולות
- C) הוסיפו שכבת ניתוב
- D) מזגו את הכלים

למה B: תיאורי כלים הם מנגנון הבחירה העיקרי של המודל. זה התיקון בעלות הנמוכה ביותר עם ההשפעה הגבוהה ביותר. A זה over-engineering. C מוסיף טוקנים בלי לטפל בשורש הבעיה. D דורש יותר מאמץ מהמוצדק.

שאלה 3 (תרחיש: Customer Support Agent)

מצב: הסוכן פותר רק 55% מהבעיות במטרת 80%. הוא מסלים מקרים פשוטים ומנסה לטפל באופן אוטונומי בחריגי מדיניות מורכבים.

איך משפרים את הקליברציה?

- A) **[נכון]** הוסיפו קריטריוני הסלמה מפורשים עם דוגמאות few-shot
- B) ביטחון עצמי (1–10) עם הסלמה אוטומטית
- C) מסווג נפרד שאומן על נתונים היסטוריים
- D) ניתוח sentiment

למה A: מטפל ישירות בשורש הבעיה — גבולות החלטה לא-ברורים. B (המודל יכול לטעות בביטחון) לא אמין. C זה over-engineering. D (מורכבות =! מצב רוח) פותר בעיה אחרת.

שאלה 4 (תרחיש: Code Generation with Claude Code)

מצב: אתם צריכים פקודה מותאמת `/review` ל-code review סטנדרטי שזמינה לכל הצוות בעת clone של הריפו.

איפה ליצור את קובץ הפקודה?

- A) **[נכון]** הפרויקט ב-repo `.claude/commands/`
- B) `~/.claude/commands/`
- C) `CLAUDE.md` בשורש
- D) `.claude/config.json`

למה A: פקודות פרויקט הנשמרות ב-`.claude/commands/` הן version-controlled וזמינות אוטומטית לכולם. B לא קיים. C להוראות, לא להגדרות פקודה. D לפקודות אישיות.

שאלה 5 (תרחיש: Code Generation with Claude Code)

מצב: צריך לבצע ארגון מחדש של מונוליט ל-microservices (עשרות קבצים, החלטות גבולות שירות).

באיזה גישה להשתמש?

- A) מצב תכנון: חקרו את ה-codebase, **[נכון]** הבינו תלויות, עצבו גישה
- B) ביצוע ישיר הדרגתי
- C) ביצוע ישיר עם הוראות מפורטות מראש
- D) ביצוע ישיר ומעבר לתכנון כשנהיה קשה

למה A: מצב תכנון מתוכנן לשינויים גדולים, מספר גישות אפשריות והחלטות ארכיטקטוניות. B מסכן בעבודה חוזרת. C יקרה. D מניח שאתם כבר יודעים את המבנה. ריאקטיבי.

שאלה 6 (תרחיש: Code Generation with Claude Code)

מצב: ל-codebase יש מוסכמות שונות באזורים שונים (React, API, database). בדיקות ממוקמות יחד עם הקוד. אתם רוצים שמוסכמות יוחלו אוטומטית.

באיזה גישה להשתמש?

- A) קבצי `.claude/rules/` עם YAML frontmatter ותבניות glob **[נכון]**
- B) שמו הכל ב-CLAUDE.md בשורש
- C) Skills ב-`.claude/skills/`
- D) CLAUDE.md בכל תיקייה

למה A: מאפשר החלת מוסכמות אוטומטית על (למשל `**/*.test.tsx`) עם תבניות glob `.claude/rules/`. B מסתמך על הסקת המודל. C בסיס נתיבי קבצים — אידיאלי לבדיקות פזורות ב-codebase. D ידני/לפי דרישה. לא עובד טוב כשקבצים רלוונטיים בהרבה תיקיות.

שאלה 7 (תרחיש: Multi-agent Research System)

מצב: המערכת חוקרת "השפעת AI על תעשיות יצירה", אבל הדוחות מכסים רק אמנות חזותית. המתאם פירק את הנושא ל: "AI באמנות דיגיטלית", "AI בעיצוב גרפי", "AI בצילום".

מה הסיבה?

- A) סוכן הסינתזה לא מזהה פערים
- B) **[נכון]** המתאם פירק את המשימה בצורה צרה מדי
- C) סוכן חיפוש האינטרנט לא מחפש מספיק יסודית
- D) סוכן ניתוח המסמכים מסנן מקורות לא-חזותיים

למה B: הלוגים מראים שהמתאם פירק את "תעשיות יצירה" רק לתת-נושאים חזותיים, מפספס לגמרי מוזיקה, ספרות וקולנוע. תת-הסוכנים ביצעו נכון — הבעיה היא במה שהוקצה להם.

שאלות נוספות

הקובץ המקורי כולל למעלה מ-70 שאלות. המבנה והעקרונות זהים — מקרי שימוש מתרחישי המבחן, ארבע לעיל. II-ו I אפשרויות, והסבר למה התשובה הנכונה היא הנכונה. כל אלה מבוססים על העקרונות שכוסו בחלקים לתרגול מלא, ראו את practical_test_en.html.

מומלץ להתאמן בשאלות באנגלית כיוון שכך תקבלו את ניסוח השאלות הזהה למבחן האמיתי.

תרגילים מעשיים

תרגיל 1: סוכן רב-כלים עם לוגיקת הסלמה

מטרה: לתכנן לולאת סוכן עם אינטגרציית כלים, טיפול שגיאות מובנה והסלמה.

שלבים:

1. הגדירו 3–4 כלי MCP (כללו שני כלים דומים לבחינת בחירת כלי) עם תיאורים מפורטים
2. מימשו לולאת סוכן הבודקת `stop_reason` (`"tool_use"` / `"end_turn"`)
3. הוסיפו תגובות שגיאה מובנות: `errorCategory`, `isRetryable`, תיאור
4. מימשו hook ליירוט שחוסם פעולות מעל סף ומנתב להסלמה
5. בדקו עם בקשות רב-היבטיות

דומיינים: 1 (ארכיטקטורת סוכן), 2 (כלים ו-MCP), 5 (הקשר ואמינות)

תרגיל 2: הגדרת Claude Code לפיתוח צוותי

מטרה: להגדיר CLAUDE.md, פקודות מותאמות אישית, כללי path ושרתי MCP ספציפיים.

שלבים:

1. צרו CLAUDE.md ברמת פרויקט עם סטנדרטי universal
2. צרו קבצי `.claude/rules/` עם YAML frontmatter לאזורי קוד שונים (`paths: ["src/api/**/*"]`, `paths: ["**/*.test.*"]`)
3. צרו skill פרויקט תחת `.claude/skills/` עם `context: fork` ו-`allowed-tools`
4. הגדירו שרת MCP ב-`.mcp.json` עם משתני סביבה + override אישי ב-`~/.claude.json`
5. בדקו מצב תכנון לעומת ביצוע ישיר במשימות בעלות מורכבות שונה

דומיינים: 3 (הגדרות Claude Code), 2 (כלים ו-MCP)

תרגיל 3: Pipeline לחילוץ נתונים מובנים

מטרה: JSON schemas, `tool_use` לפלט מובנה, לולאות validation/retry, עיבוד batch.

שלבים:

1. הגדירו כלי חילוץ עם JSON schema (שדות required/optional, enums עם "other", שדות nullable)
2. בנו לולאת validation: בשגיאה, retry ספציפית עם validation המסמך, חילוץ שגוי ושגיאת
3. למסמכים עם מבנים שונים few-shot הוסיפו דוגמאות
4. מסמכים, טיפול בכשלים דרך Message Batches API: 100 `custom_id` השתמשו בעיבוד דרך batch
5. ניתוב לאדם: ציוני ביטחון ברמת שדה, ניתוח לפי סוג מסמך

דומיינים: 4 (הנדסת פרומפט), 5 (הקשר ואמינות)

מחקרי רב-סוכני pipeline של debugging-תרגיל 4: עיצוב ו

מטרה: אורקסטרציית תת-סוכנים, העברת הקשר, הפצת שגיאות, סינתזה עם מעקב מקורות.

שלבים:

1. (הקשר מועבר מפורשות בפרומפטים, `"Task"` כולל `allowedTools`) מתאם עם 2+ תת-סוכנים
2. בתגובה אחת `Task` הריצו תת-סוכנים במקביל דרך מספר קריאות
3. מקור, תאריך פרסום URL, דרשו פלט תת-סוכן מובנה: טענה, ציטוט
4. של תת-סוכן: החזירו הקשר שגיאה מובנה למתאם והמשיכו עם תוצאות חלקיות timeout דמו
5. הפרידו ממצאים מאומתים ממוכרזים ;attribution בדקו עם נתונים סותרים: שמרו את שני הערכים עם

דומיינים: 1 (ארכיטקטורת סוכן), 2 (הקשר ואמינות) 5 ,(MCP-כלים ו)

נספח: טכנולוגיות ומושגים

טכנולוגיה	היבטי מפתח
Claude Agent SDK	יצירת תת-סוכנים דרך AgentDefinition, agent loops, `stop_reason`, hooks (PostToolUse), Task, `allowedTools`
Model Context Protocol (MCP)	שרתי MCP, tools, resources, `isError`, תיאורי כלים, `.mcp.json`, משתני סביבה
Claude Code	היררכיית CLAUDE.md, `.claude/rules/` עם תבניות glob, `.claude/commands/`, `.claude/skills/` עם SKILL.md, מצב תכנון, `/compact`, `--resume`, `fork_session`
Claude Code CLI	`-p` / `--print` למצב לא-אינטראקטיבי, `--output-format json`, `--json-schema`
Claude API	`tool_use` עם JSON schemas, `tool_choice` ("auto"/"any"/forced), `stop_reason`, `max_tokens`, system prompts
Message Batches API	50% חיסכון, חלון של עד 24 שעות, `custom_id`, אין multi-turn tool calling
JSON Schema	Required לעומת optional, שדות nullable, סוגי enum, "other" + מצב, פירוט strict
Pydantic	אימות סכמה, שגיאות סמנטיות, לולאות validation/retry

כלים מובנים	Read, Write, Edit, Bash, Grep, Glob — מטרה וקריטריוני בחירה
Few-shot prompting	דוגמאות ממוקדות למצבים דו-משמעיים, הכללה לדפוסים חדשים
Prompt chaining	פירוק סדרתי ל-passes ממוקדים
חלון הקשר	scratchpad קבצי, "lost in the middle", תקציבי טוקנים, סיכום פרוגרסיבי
ניהול סשנים	Resume, `fork_session`, סשנים בעלי שם, בידוד הקשר
קליברציית ביטחון	stratified sampling, ניקוד ברמת שדה, קליברציה על סטים מתויגים

נושאים מחוץ להיקף

הנושאים הסמוכים הבאים **לא** יהיו במבחן:

- Fine-tuning של מודלי Claude או אימון מודלים מותאמים
- חיוב או ניהול חשבון, אימות Claude API
- מימוש מפורט בשפות תכנות או frameworks ספציפיים (מעבר למה שדרוש להגדרת tool/schema)
- פריסה או אירוח של שרתי MCP (תשתית, רשת, container orchestration)
- תהליך אימון או משקלי מודל, ארכיטקטורה פנימית של Claude
- Constitutional AI, RLHF או מתודולוגיות אימון בטיחות
- פרטי מימוש של מודלי embedding או vector database
- Computer use (אוטומציית דפדפן, אינטראקציה דסקטופ)
- יכולות ניתוח תמונה (Vision)
- Streaming API או server-sent events
- Rate limiting, מפורטים API מכסות, או חישובי עלות
- OAuth, סבב API keys, או פרטי פרוטוקול אימות
- הגדרות ספציפיות לספק ענן (AWS, GCP, Azure)
- בנצ'מרקים של ביצועים או מדדי השוואת מודלים
- פרטי מימוש של prompt caching (מעבר לידיעה שזה קיים)
- אלגוריתמי ספירת טוקנים או פרטי tokenization

המלצות הכנה

1. **בנו סוכן עם Claude Agent SDK** — מימשו לולאת סוכן מלאה עם קריאה לכלים, טיפול שגיאות וניהול סשנים. תתאמנו בתת-סוכנים והעברת הקשר מפורשת.
2. **הגדירו Claude Code לפרויקט אמיתי** — השתמשו בהיררכיית CLAUDE.md, כללי path ספציפיים ב-`.claude/rules/`, skills עם `context: fork` ו-`allowed-tools`, ושילוב שרת MCP.
3. **תכננו ובדקו כלי MCP** — כתבו תיאורים שמבדילים כלים דומים, החזירו שגיאות מובנות עם קטגוריות ודגלי retry, ובדקו מול בקשות משתמש דו-משמעיות.

4. **בנו pipeline** — השתמשו ב-**לחילוץ נתונים** `tool_use` עם JSON schemas, לולאות validation/retry, שדות optional/nullable, ועיבוד batch דרך Message Batches API.
5. הוסיפו דוגמאות — **תתאמנו בהנדסת פרומפט** few-shot לתרחישים דו-משמעיים, קריטריוני review מפורשים, multi-pass ל-code reviews גדולים וארכיטקטורות.
6. **למדו דפוסי ניהול הקשר** — חלצו עובדות מפלטים מילוליים, השתמשו בקבצי scratchpad, והצילו discovery לתת-סוכנים כדי לטפל בגבולות הקשר.
7. **הבינו הסלמה ו-human-in-the-loop** — מתי להסלים (פערי מדיניות, בקשה מפורשת של משתמש, חוסר יכולת להתקדם) וזרימות ניתוב מבוסס-ביטחון.
8. **קחו מבחן תרגול** לפני האמיתי. הוא משתמש באותם תרחישים ופורמט.

> 📝 **על התרגום:** המדריך הזה הוא תרגום עברי של המדריך האנגלי `guide_en.md`, מותאם להישראלי-נשמרו באנגלית לפי המוסכמה הישראלית. (CLAUDE.md, MCP, tool_use, וכו') טכני. מונחים טכניים נשמרו באנגלית כיוון שהם תחביר enum דוגמאות קוד, סכמות וערכי API.
>
> תרומות לשיפור התרגום מתקבלות בברכה דרך GitHub Issues.