کلود برای مهندسان نرم افزار

ساخت، آزمون و خودکارسازی

کلود کد را به کار بگیرید، نیازمندی ها را به پیاده سازی های مرحله بندی شده تبدیل کنید، از توسعه مبتنی بر رفتار برای هدایت کار بهره ببرید، نتایج را با آزمون های کمک گرفته از هوش مصنوعی راستی آزمایی کنید و هوش مصنوعی را در یک گردش کار تکرارپذیر برای تحویل نرم افزار ادغام کنید.

التون استونمن، مارک وینترینگهام، توماش لِلِک و آرتور اسکوورونسکی

فصل نمونه رایگان

ترجمه فارسی

شناسنامه و حقوق نشر

کلود برای مهندسان نرم افزار

ساخت، آزمون و خودکارسازی

حق نشر © ۲۰۲۷، انتشارات منینگ. تمامی حقوق محفوظ است.

فهرست مطالب

- مقدمه
- شروع به کار با کلود کد — فصل ۵ از کتاب کلود در محیط کار، نوشته التون استونمن
- برنامه ریزی و ساخت مرحله ای با توسعه مبتنی بر رفتار (BDD) — فصل ۶ از کتاب کلود در محیط کار، نوشته التون استونمن
- آزمون نرم افزار با کمک هوش مصنوعی برای توسعه دهندگان — فصل ۴ از کتاب آزمون نرم افزار با هوش مصنوعی مولد، نوشته مارک وینترینگهام
- از تکمیل خودکار تا ناوگان عامل ها: مسیر رسیدن به هوش مصنوعی پیوسته — فصل ۵ از کتاب مهندسی وایب، نوشته توماش لِلِک و آرتور اسکوورونسکی

مقدمه

کدی که کار می کند اهمیت دارد، اما تنها بخشی از یک پروژه است. شما باید نیازمندی ها را نیز درک کنید، تصمیم های طراحی سنجیده بگیرید و مطمئن شوید چیزی که ساخته اید همان طور رفتار می کند که انتظار دارید. هوش مصنوعی می تواند در تمام این مراحل کمک کند؛ اما پیش از هر چیز باید بدانید چگونه کار آن را هدایت کنید و نتایجش را ارزیابی کنید.

این مجموعه فصل ها نشان می دهد چگونه می توان هوش مصنوعی را بدون کنار گذاشتن اصول مهندسی نرم افزار، وارد فعالیت های روزمره توسعه نرم افزار کرد. این مجموعه با تکیه بر سه کتاب از انتشارات منینگ، شما را از نخستین نشست کار با کلود کد به سوی یک گردش کار تکرارپذیر برای برنامه ریزی، پیاده سازی، آزمون و تحویل نرم افزار می برد.

کار را با یک پروژه و وظیفه ای آغاز کنید که ارزش واگذاری دارد. در فصل «شروع به کار با کلود کد» از کتاب کلود در محیط کار نوشته التون استونمن، یاد می گیرید زمینه و اطلاعات مورد نیاز را در اختیار کلود بگذارید، مجوزهای مناسبی تعیین کنید و تغییراتی را که ایجاد می کند بازبینی کنید. این گام های عملی نخست، عادت هایی را شکل می دهند که هنگام واگذاری کارهای بزرگ تر به آن ها نیاز خواهید داشت: اینکه دستیار شما چه کارهایی می تواند انجام دهد، چه چیزهایی از شما نیاز دارد و پیش از پذیرفتن کد آن باید چه مواردی را بررسی کنید.

البته حتی یک دستیار برنامه نویسی توانمند هم ممکن است کد نادرستی تولید کند. مشکل اغلب از نیازمندی هایی آغاز می شود که تا پیش از تلاش برای پیاده سازی، روشن و بی ابهام به نظر می رسند. فصل «برنامه ریزی و ساخت مرحله ای با توسعه مبتنی بر رفتار (BDD)» که آن هم از کتاب کلود در محیط کار گرفته شده، به همین موضوع می پردازد. توصیف رفتار مورد انتظار با استفاده از سناریوهای گرکین به شما اجازه می دهد ابهام ها را پیش از تبدیل شدن به کد برطرف کنید. تقسیم پیاده سازی به مرحله های کوچک تر نیز این امکان را فراهم می کند که در طول پیشرفت کار، خروجی را بررسی کنید، اصلاحات لازم را انجام دهید و هر مرحله را تایید کنید.

آزمون نیز به همین اندازه نیازمند دقت است. مجموعه ای از آزمون های تولیدشده ممکن است اطمینان بخش به نظر برسد، اما در عمل چه چیزی را به شما نشان می دهد؟ فصل «آزمون نرم افزار با کمک هوش مصنوعی برای توسعه دهندگان» از کتاب آزمون نرم افزار با هوش مصنوعی مولد نوشته مارک وینترینگهام، این پرسش را با

مثال های عملی آزمون بررسی می کند. در این فصل می بینید هوش مصنوعی چگونه می تواند به بررسی کد و طراحی آزمون های واحد کمک کند و سپس چگونه می توان آن را وارد چرخه قرمز/سبز/بازآرایی کرد. با روش های بیشتری برای بررسی ریسک ها و بهبود طراحی آشنا می شوید و مبنای بهتری برای قضاوت درباره مفید بودن آزمون های خود به دست می آورید.

گزیده پایانی، نگاه را از ویرایشگر شما به فرایند تحویل نرم افزار در سطح تیم گسترش می دهد. دستورالعمل های مشترک، خط لوله های یکپارچه سازی پیوسته (CI) و واگذاری کار میان عامل ها، همگی تعیین می کنند که آیا می توان کار مبتنی بر هوش مصنوعی را به شکلی قابل اتکا تکرار کرد یا نه. فصل «توسعه پیوسته هوش مصنوعی با چرخه عمر توسعه نرم افزار بومی هوش مصنوعی» از کتاب مهندسی وایب نوشته توماش لِلِک و آرتور اسکوورونسکی بررسی می کند که این اجزا چگونه در کنار هم قرار می گیرند و به شما کمک می کند تصمیم بگیرید خودکارسازی در کجا مناسب است و در کجا بازبینی انسانی ضرورت دارد. این گام بعدی برای آن است که روش هایی را که در یک وظیفه آزموده اید، در پروژه های متعدد به کار بگیرید.

هدف این کتاب الکترونیکی گزیده، ایجاد فرایندی برای توسعه نرم افزار است که بتوان به آن اعتماد کرد و آن را بارها تکرار کرد. چه در حال آزمایش یک دستیار برنامه نویسی باشید و چه بخواهید هوش مصنوعی را وارد گردش کار تیم خود کنید، این فصل ها نقطه شروع مناسبی هستند. اگر می خواهید این مهارت ها و مفاهیم را عمیق تر بررسی کنید، مطالعه نسخه کامل کتاب های معرفی شده را به شدت پیشنهاد می کنیم؛ آن کتاب ها مثال های بیشتری، توضیحات عمیق تر و راهنمایی هایی برای پروژه های خودتان ارائه می دهند.

با سپاس،

تیم انتشارات منینگ

کلود کد را در ویرایشگر خود و در پروژه ای به کار بگیرید که بتواند آن را درک کند؛ سپس فقط مجوزهایی را به آن بدهید که با اعطای آن ها آسوده خاطر هستید. هم زمان با ساخت نرم افزار توسط کلود، بازبینی تغییرات آن و تصمیم گیری درباره حفظ یا رد هر تغییر را تمرین خواهید کرد.

فصل اول: شروع به کار با کلود کد

برگرفته از فصل ۵ کتاب «کلود در محیط کار»، نوشته التون استونمن

در این فصل می آموزید

- کدنویسی عاملی چیست و نقش شما در مقام بازبین چگونه تغییر می کند؛
- چگونه کلود کد را در ویرایشگر خود نصب و اجرا کنید؛
- چگونه مخزن پروژه را آماده کنید تا کلود بتواند در آن پیمایش کند؛
- چگونه مجوزها را طوری تنظیم کنید که کلود بتواند به صورت خودکار کار کند؛
- چگونه نخستین سرویس کامل را بسازید و آزمایش کنید.

شما به هدایت گفت وگو با کلود عادت دارید. در یک نشست گفت وگومحور، نوبت شما و مدل به صورت یکی در میان پیش می رود و اگر پس از پاسخ کلود چیزی برای افزودن نداشته باشید، گفت وگو تمام می شود. همچنین دیده اید که در Cowork نوبت های کلود می توانند بسیار طولانی تر باشند؛ ممکن است یک درخواست شما چندین فراخوانی ابزار را راه بیندازد و پیش از آنکه پاسخ را ببینید، فایل های خروجی تولید کند. کلود کد جایی است که می توانید ببینید کلود به تنهایی تا چه اندازه قادر به انجام کار است. این ابزار کدنویسی عاملی شرکت آنتروپیک است؛ یعنی کلود درون مخزن کد شما اجرا می شود و به تمام کدها و ابزارهای موجود روی دستگاهتان دسترسی دارد. یک درخواست شما ممکن است نوبتی را آغاز کند که ساعت ها ادامه داشته باشد و هزاران خط کد و آزمون تولید کند.

آیا واقعا چنین چیزی می خواهید؟ این کار به معنای اعتماد زیاد به مدل برای نوشتن کد خوب و آزمون های درست است و در پایان، وظیفه بزرگی برای بازبینی بر دوش شما می گذارد. این رویکرد برای کاری معتبر است که مشخصات روشنی دارد و آزمون پذیرش آن ساده است؛ اما برای همه وظایف مناسب نیست. کارآمدی در استفاده از کلود کد تا اندازه ای به درست نوشتن درخواست ها و به کارگیری فنونی بستگی دارد که تاکنون آموخته ایم. بخش دیگر آن، یادگیری این موضوع است که چه زمانی باید کلود را در سطحی جزئی تر هدایت کنید، از نزدیک با آن همکاری کنید و در تصمیم ها سهیم شوید، و چه زمانی می توانید اختیار کامل را به کلود بسپارید و اجازه دهید خودش کار را بسازد.

یک پروژه بزرگ در سطح تولید که قرار نیست هرگز واقعا در محیط عملیاتی اجرا شود، بستر آزمایشی بسیار خوبی است. چنین پروژه ای به شما اجازه می دهد با کلود کد آزمایش کنید و ببینید چه کارهایی می تواند انجام دهد و چگونه آن ها را انجام می دهد؛ بدون آنکه نگران نیازمندی های مشتری، فرایندهای پشتیبانی یا مقیاس پذیری باشید. ما تمام مولفه های برنامه خود را طراحی و مشخص کرده ایم. اکنون یکی از سرویس های اصلی را انتخاب می کنیم و می بینیم چگونه می توانیم با کمترین راهنمایی، کلود را به ساخت آن هدایت کنیم.

۵.۱ کدنویسی عاملی چیست؟

نوشتن کد کاری ایده آل برای هوش مصنوعی است. زبان های برنامه نویسی مجموعه قواعد روشنی دارند، الگوهای الگوریتمی به خوبی مستند شده اند و میلیاردها خط کد متن باز وجود دارد که مدل ها با آن ها آموزش دیده اند. بنابراین بخش تولید کد نسبتا ساده است؛ اما نقطه قوت واقعی کدنویسی عاملی در این است که از همان ابزارهایی استفاده می کند که شما به کار می برید. این ابزار فقط کد نمی نویسد؛ بلکه برای بررسی خطاهای کامپایل، دستور `dotnet build` را اجرا می کند، برای اجرای برنامه از `npm start` استفاده می کند و آزمون ها را با `pytest` اجرا می کند.

لحظه روشن شدن چراغ

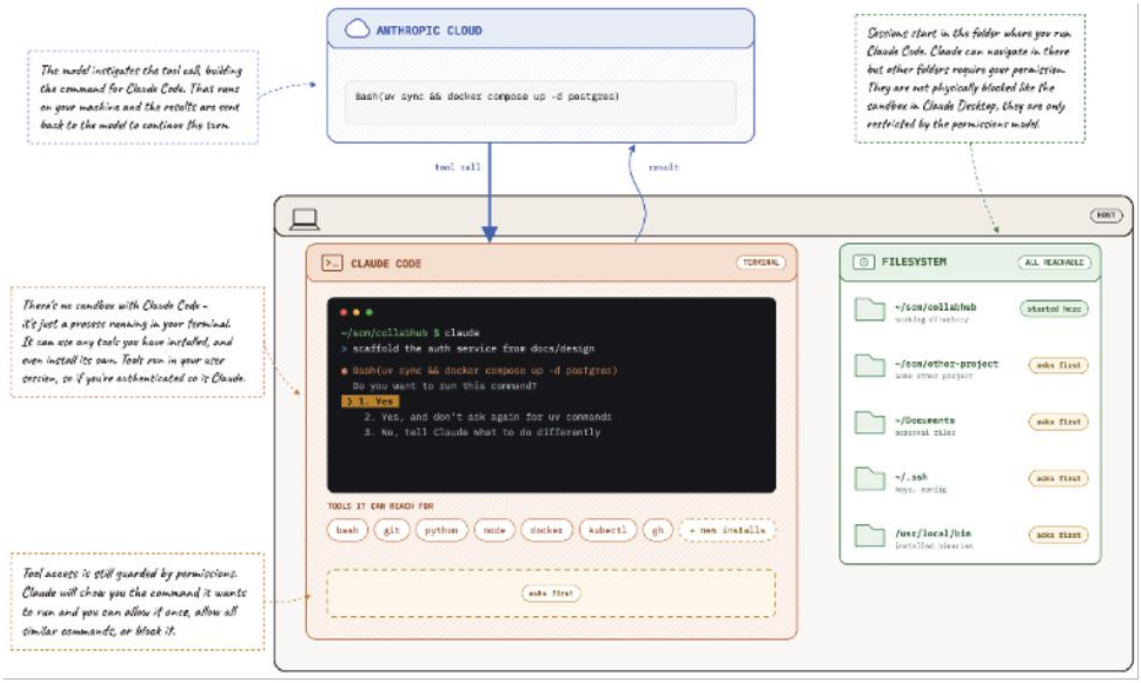
این همان لحظه ای است که معمولا موضوع برایتان روشن می شود. وقتی می بینید کلود همان ابزارهایی را اجرا می کند که خودتان به کار می برید، متوجه می شوید مدل بسیار فراتر از تولید چند خط کد ثابت توانایی دارد. من در اوایل کار، زمانی که می خواستم مشکلی را در یک استقرار Kubernetes پیدا کنم، چنین تجربه ای داشتم. فایل YAML را در ویرایشگر بررسی می کردم و سوالی درباره آن از کلود پرسیدم. کلود در پاسخ گفت می خواهد یک دستور `kubectl` اجرا کند. ابتدا با خودم گفتم: «عجب! اصلا نمی دانستم کلود می تواند چنین کاری انجام دهد.» بعد فکر کردم: «هوشمندانه است که می خواهد استقرار زنده را ببیند و آن را با فایل YAML مقایسه کند.» و در نهایت با خودم گفتم: «این دقیقا همان دستوری است که خودم قصد داشتم در گام بعدی اجرا کنم.»

کدنویسی عاملی یعنی استفاده از هوش مصنوعی برای نوشتن کد؛ اما این کار می تواند تقریبا کاملا بدون دخالت مستقیم شما انجام شود: مسئله را توضیح می دهید و مدل به صورت خودکار برای ساخت راه حل اقدام می کند.

کلود کد یک برنامه ترمینالی است که آن را در یک نشست پوسته یا در پنجره ترمینال ویرایشگر خود اجرا می کنید. این برنامه همان مجموعه مدل هایی را در اختیار دارد که در رابط گفت وگو و Cowork ارائه می شوند؛ با

این تفاوت که آن را داخل پوشه مخزن پروژه اجرا می کنید تا به تمام فایل های منبع، آزمون ها، اسکریپت های ساخت و مستندات دسترسی داشته باشد. کلود می تواند در سراسر پایگاه کد پیمایش کند. بنابراین اگر در درخواست خود پیام خطایی بیاورید، می تواند محل تولید آن پیام را جست وجو کند و پشته فراخوانی و داده هایی را که به بروز خطا منجر شده اند، بررسی کند.

گردش کار بنیادی همچنان همان است: درخواست شما به مدلی فرستاده می شود که در فضای ابری آنتروپیک اجرا می شود و مدل پاسخی تولید می کند. اما در کلود کد، پاسخ می تواند شامل چندین فراخوانی ابزار باشد: جست وجوی یک متن، ویرایش چند فایل، اجرای اسکریپت ساخت و اجرای آزمون ها. همه این کارها در قالب یک نوبت واحد انجام می شوند و ابزارها روی دستگاه شما اجرا شده و پاسخ را به مدل بازمی گردانند. تفاوت بزرگ کلود کد با رابط های دیگر در این است که هیچ محیط ایزوله فیزیکی وجود ندارد که دسترسی مدل به ابزارها یا فایل ها را محدود کند. شکل ۵.۱ این ساختار را نشان می دهد: کلود کد می تواند به هر چیزی دسترسی داشته باشد که اجازه استفاده از آن را به او داده اید.



شکل ۵.۱ کلود کد درون مخزن پروژه شما و بدون محیط ایزوله اجرا می شود. سازوکار مجوزها برای دسترسی به فایل ها و اجرای ابزارها از شما اجازه می گیرد؛ اما هیچ مانع فیزیکی مستقلی برای جلوگیری از رفتار ناخواسته وجود ندارد.

همین ویژگی، کلود کد را به ابزاری بسیار قدرتمند و بهره ور تبدیل می کند. ممکن است در یک نشست، در چند خط درخواستی بنویسید که کلود را به انجام کاری یک ساعته وادار کند؛ در این مدت شما می توانید در نشست های دیگر به کارهای دیگری بپردازید. اکنون وقت آن است که کلود کد را نصب کنیم و ببینیم چگونه کار می کند.

۵.۲ نصب و اجرای کلود کد

برای اجرای کلود کد روی macOS، ویندوز و لینوکس نصب کننده های بومی وجود دارد. به نشانی https://code.claude.com/ بروید و نصب کننده را دریافت و اجرا کنید؛ یا آن را از ترمینال نصب کنید:

```
# macOS یا Linux:
curl -fsSL https://claude.ai/install.sh | bash

# Windows در PowerShell یا #
irm https://claude.ai/install.ps1 | iex
```

اکنون دستور claude در اختیار شماست. پیش از آنکه بتوانید کار مفیدی انجام دهید، باید وارد حساب خود شوید:

```
claude auth login
```

با اجرای این دستور، مرورگر باز می شود تا با حساب آنتروپیک خود وارد شوید. مراحل ورود را دنبال کنید؛ سپس در ترمینال پیام Login successful را مشاهده خواهید کرد.

کلود کد از همان الگوی مصرف توکن سایر بخش های پلتفرم استفاده می کند و حساب شما یک سهمیه مشترک دارد که برای تمام ابزارها اعمال می شود. بنابراین اگر در یک نشست کلود کد تمام سهمیه خود را مصرف کنید، تا زمان بازنشانی سهمیه نمی توانید در برنامه های موبایل یا دسکتاپ کاری انجام دهید و برعکس.

وقتی کلود شروع به نوشتن کد می کند، بخش بزرگی از نقش شما بازبینی خروجی آن خواهد بود. اجرای کلود کد در ویرایشگر همیشگی تان این کار را بسیار آسان تر می کند؛ زیرا فهرست تمام فایل هایی را که کلود ایجاد یا ویرایش کرده است می بینید و می توانید تفاوت نسخه های تغییر یافته را بررسی کنید. هر ویرایشگری مناسب

است، اما بهتر است از یکپارچگی با Git و امکان باز کردن ترمینال در کنار ویرایشگر فایل برخوردار باشد. اگر ترجیح خاصی ندارید، VS Code انتخاب خوبی است.

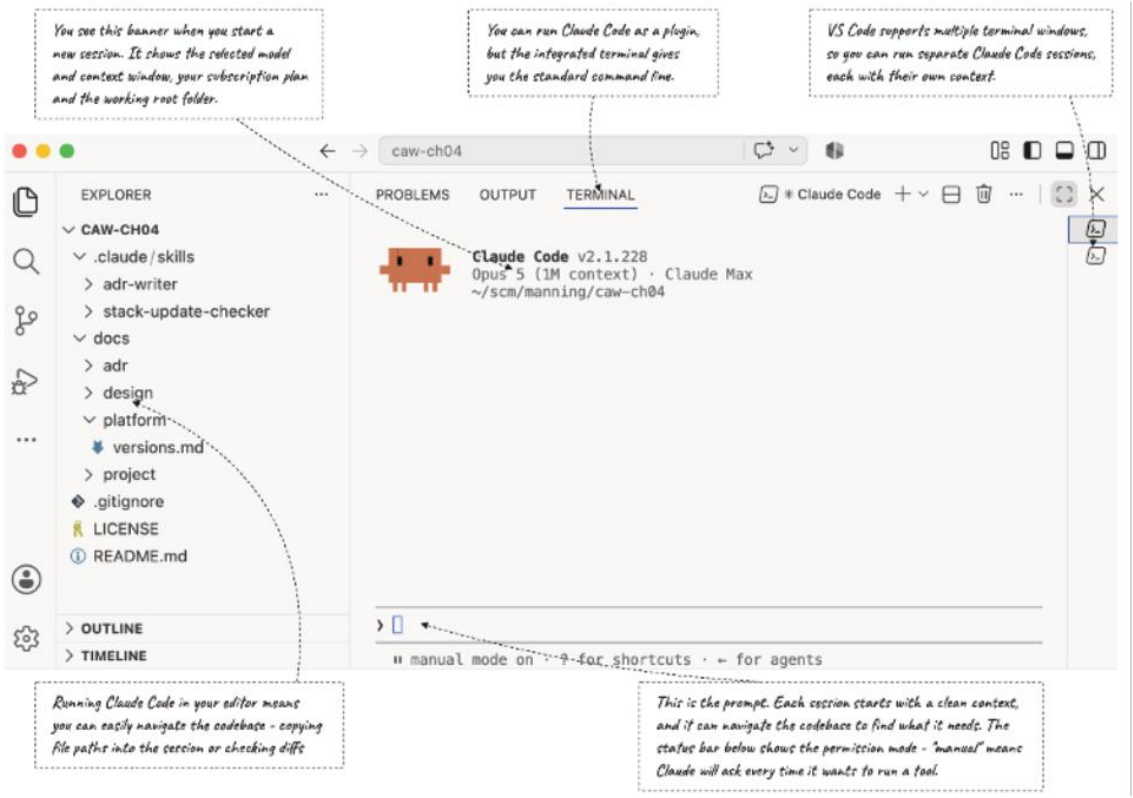
> نکته: افزونه های کلود کد برای Cursor ،VS Code و محیط های توسعه یکپارچه JetBrains وجود دارند. این افزونه ها برای نمایش درون خطی تفاوت فایل ها و تاریخچه گفت وگو با ویرایشگر یکپارچه می شوند، اما همگی از موتور یکسان کلود کد استفاده می کنند. من گاهی برای بررسی قابلیت های جدید از افزونه VS Code استفاده می کنم؛ ولی معمولا دستور claude را از پنجره ترمینال داخل VS Code اجرا می کنم. ترجیح می دهم با برنامه بومی کار کنم تا هنگام کار روی دستگاهی که افزونه یا حتی رابط گرافیکی ندارد نیز همان تجربه را داشته باشم.

۵.۳ همین حالا امتحان کنید: اجرای کلود کد در ویرایشگر

ویرایشگر خود را اجرا کنید و پوشه ای را باز کنید که مخزن Git برنامه شما در آن قرار دارد. باید پوشه docs را ببینید که شامل تمام اسناد طراحی تهیه شده برای شماست. یک پنجره ترمینال باز کنید؛ در VS Code می توانید از میانبر Ctrl-]` استفاده کنید. سپس کلود کد را در حالت مجوزدهی دستی اجرا کنید تا پیش از ویرایش فایل ها یا اجرای ابزارها، تایید شما را بگیرد:

```
claude --permission-mode manual
```

شکل ۵.۲ ویرایشگر من را نشان می دهد که کلود کد در آن با حالت دستی اجرا شده است.



شکل ۵.۲ کلود کد در پنجره ترمینال VS Code اجرا شده است. شما و کلود به یک مخزن واحد نگاه می کنید و به همین دلیل همکاری با یکدیگر آسان است.

از همین نشست برای شروع ساخت برنامه استفاده خواهیم کرد.

۵.۴ نخستین نشست کلود کد شما

نشست های کلود کد با یک درخواست خالی و بدون هیچ زمینه قبلی آغاز می شوند؛ درست مانند گفت وگو در محیط چت و Cowork که هر بار از نو شروع می کنید. خوشایند نیست که کلود در هر نشست تازه مجبور باشد تمام پایگاه کد شما را زیر و رو کند تا بفهمد باید از چه زبان و ابزارهای ساختی استفاده کند. بنابراین می توانید توضیح کوتاهی از مخزن پروژه را در یک فایل Markdown به نام CLAUDE.md ثبت کنید. اگر هنگام آغاز نشست کلود کد، این فایل در مخزن وجود داشته باشد، به طور خودکار به زمینه مدل افزوده می شود و اطلاعات کلیدی مورد نیاز را از همان ابتدا در اختیار کلود می گذارد.

می توانید فایل CLAUDE.md را خودتان بنویسید؛ اما کلود نیز می تواند با دستور /init این کار را انجام دهد. اگر این دستور را در یک پایگاه کد تثبیت شده اجرا کنید، کلود فایلی معادل README پروژه می نویسد که جزئیات فناوری ها و ابزارهای مورد استفاده را شرح می دهد. این فایل همان هدف README را

دنبال می کند: آشنا کردن افراد تازه وارد با مخزن پروژه؛ با این تفاوت که مخاطب اصلی آن، تازه واردانی هستند که در آینده نمونه های دیگری از کلود خواهند بود.

هنوز هیچ کدی نداریم، اما اسناد طراحی فراوانی وجود دارد که کلود می تواند برای پایه گذاری دستورالعمل ها از آن ها استفاده کند.

۵.۵ همین حالا امتحان کنید: اجازه دهید کلود مستندات عامل را بسازد

احتمالا یک درخواست ساده /init کافی بود؛ اما من جزئیات بیشتری اضافه کردم تا کلود مستند کند توسعه دهندگان چگونه برنامه را به صورت محلی اجرا خواهند کرد و ما چگونه آن را در محیط های دیگر مستقر می کنیم.

درخواست نمونه:

```
/init - this repo will build out the whole app specced in the
docs/design folder. we
will use docker to run the stack locally, and package with
helm for deployment to
kubernetes
```

در نشست من، کلود پرسید که آیا می خواهم فایل CLAUDE.md در سطح پروژه و داخل مخزن قرار بگیرد یا یک فایل شخصی در پوشه اسناد خودم داشته باشم. می خواستیم همه افرادی که روی پروژه کار می کنند از این فایل دستورالعمل بهره ببرند و بتوانند هم زمان با تکامل برنامه آن را گسترش دهند؛ بنابراین گزینه سطح پروژه را انتخاب کردم.

کلود همچنین پوشه مهارت های من را پیدا کرد و پرسید آیا می خواهم مهارت های دیگری را نیز راه اندازی کند یا نه. پاسخ من منفی بود. سپس سراغ جزئیات ابزارهای Python مورد نظرم رفت. بعد از آن، تمام مستندات را بررسی کرد و چند ناسازگاری میان اسناد طراحی اولیه و فهرست نسخه مولفه ها پیدا کرد. از آنجا که می دانست سند طراحی زودتر نوشته شده است، پرسید آیا فایل نسخه ها مرجع معتبر و نهایی محسوب می شود یا نه.

در طول فرایند راه اندازی، پرسش های دیگری نیز مطرح می شوند. اینجا باید تصمیم هایی را که از قلم انداخته اید یا فقط به شکلی مبهم پاسخ داده اید، نهایی کنید. وقتی به کلود می گویید آماده آغاز ساخت هستید، پرسش های بیشتری مطرح می شوند؛ زیرا فایل CLAUDE.md باید آن قدر کامل باشد که عامل بعدی بتواند کار

را شروع کند. اجازه دهید نشست شما را راهنمایی کند: کلود شکاف ها را مشخص می کند و برای پر کردن آن ها پیشنهاد می دهد.

فایل CLAUDE.md من در این نشانی قرار دارد:

https://github.com/sixeyed/claude-at-work-project/blob/main/CLAUDE.md

می توانید تاریخچه فایل را بررسی کنید و ببینید در گذر زمان چگونه تکامل یافته است. فهرست ۵.۱ بخشی از نسخه اولیه ای را نشان می دهد که کلود تولید کرده بود.

فهرست ۵.۱ – فایل CLAUDE.md؛ راهنمای عامل های هوش مصنوعی که به پروژه می پیوندند

```
# CLAUDE.md

This file provides guidance to Claude Code
(claude.ai/code) when working
with code in this repository.

## What this is

CollabHub — team chat + collaborative canvas. Five
Python backend
services and a React SPA, specified in `docs/design/` and
being built
out from that spec. Run locally with Docker Compose;
deploy to
Kubernetes with Helm.

**Read `docs/design/00-platform-conventions.md`
before writing or
changing any service code.** It is the authoritative cross-
service
contract (errors, pagination, auth, IDs, jobs, config,
observability).
Where a per-service doc disagrees with it, it wins. Per-
```

```
service specs
are `docs/design/01`–`06`.
```

این فایل Markdown و برای انسان خواناست؛ درست مانند یک فایل SKILL.md. در آن از ساختار متداول Markdown استفاده شده است: محتوا به بخش هایی با عنوان های جداگانه تقسیم شده و با حروف ضخیم، دستورالعمل های مهمی برجسته شده اند که عامل ها نباید از آن ها غافل شوند. سند من شامل بخش هایی درباره ساختار مخزن، تصمیم های باز و «قراردادهایی که به سادگی ممکن است اشتباه فهمیده شوند» بود. بخش آخر نتیجه نوعی خودارزیابی از سوی کلود است و نشان می دهد چرا بهتر است تولید و نگهداری فایل CLAUDE.md را به خود کلود بسپاریم. این فایل حوزه هایی را فهرست می کند که عامل ها ممکن است در آن ها مسیر اشتباهی انتخاب کنند و از پیش، با دستورالعمل هایی که شاید هرگز به ذهن من نمی رسید، آن ها را در مسیر درست قرار می دهد.

چند نوبت دیگر طول کشید تا کلود مطمئن شود تمام تصمیم های اصلی مشخص شده اند. در این فرایند، کلود مهارت ثبت سوابق تصمیم های معماری (Architecture Decision Record) را در پوشه پروژه دید و پیشنهاد کرد نکات کلیدی مورد توافقمان را با استفاده از آن ثبت کند. این نشست هیچ کدنویسی ای نداشت و صرفا کار آماده سازی برای عامل های آینده بود تا بتوانند سریع تر با پروژه آشنا شوند.

وقتی خودتان این فرایند را طی کنید، احتمالا از میزان دقت و جامعیت کلود شگفت زده می شوید؛ اما در عین حال ممکن است از اینکه برای انجام کارهای مختلف بارها از شما اجازه می گیرد، کلافه شوید. اگر می خواهید کلود را برای انجام وظیفه ای طولانی بدون نظارت مستقیم رها کنید، باید حالت مجوزدهی را تغییر دهید.

> پیگیری پیشرفت من: من برنامه خود را در یک مخزن عمومی GitHub می سازم و برای ثبت نقاط پیشرفت از برچسب ها استفاده می کنم. این مخزن با مخزن sixeyed/claude-at-work متفاوت است؛ مخزن دوم شامل رونوشت تمام نشست ها و منابع مرجع فصل هاست. مخزن پروژه یک ساخت تمیز در نشانی sixeyed/claude-at-work-project است. اگر می خواهید روند تکامل کار را در فصل های مختلف دنبال کنید، می توانید مخزن را کلون کنید و میان برچسب ها جابه جا شوید:

```
git clone https://github.com/sixeyed/claude-at-work-
project.git
git checkout ch04-setup
```

۵.۶ تغییر حالت مجوزدهی برای آزاد گذاشتن کلود

وقتی کلود کد را اجرا کردیم، حالت مجوزدهی را صراحتا روی manual گذاشتیم؛ در این حالت تقریبا همه چیز بررسی می شود. درخواست خود را ارسال می کنید و پاسخ کلود پیوسته قطع می شود، زیرا برای اجرای هر ابزار به اجازه نیاز دارد: از دستور cat برای خواندن محتوای فایل ها گرفته تا حلقه های for برای پیمایش پوشه ها و دستورهای git برای بررسی تاریخچه. در حالت دستی، چنین درخواست های تاییدی را بارها مشاهده خواهید کرد:

```
This command requires approval

Do you want to proceed?
> 1. Yes
  2. Yes, and allow access to caw-project/ and git ls-tree commands
  3. No
```

گزینه دوم استفاده از ابزار را تا پایان نشست تایید می کند و در نگاه اول به نظر می رسد زمان زیادی صرفه جویی خواهد شد؛ اما در عمل، هر فراخوانی ابزار اندکی با مورد قبلی تفاوت دارد و همچنان با درخواست های تایید فراوانی روبه رو می شوید. کلود کد حالت های مجوزدهی دیگری هم دارد تا بار تاییدها را کاهش دهد. گزینه های اصلی عبارت اند از؛ البته حالت پیش فرض ممکن است در گذر زمان تغییر کند، پس بهتر است آن ها را آزمایش کنید و در درخواست خود صریح باشید:

- `manual`: خواندن فایل ها و اجرای دستورهای فقط خواندنی از پیش مجاز هستند؛ برای هر کار دیگری باید مجوز گرفته شود.
- `acceptEdits`: خواندن و ویرایش فایل ها و دستورهایی که در محدوده پوشه کاری ایمن تشخیص داده شوند، از پیش مجاز هستند.
- `plan`: خواندن فایل ها و پیمایش مجاز است، اما ویرایشی انجام نمی شود؛ کلود برنامه ای می سازد که باید پیش از شروع کار آن را تایید کنید.
- `auto`: از همان سازوکار حالت دستی استفاده می کند، اما به جای شما به عنوان تاییدکننده انسانی، یک طبقه بند هوش مصنوعی درباره مجوزها تصمیم می گیرد.

- `bypassPermissions`: بررسی مجوزها را برای همه چیز کنار می گذارد و فقط دستورهایی را مسدود می کند که دیسک سخت شما را پاک کنند.

همان طور که از نامش پیداست، استفاده از bypassPermissions به همان اندازه که به نظر می رسد ایده بدی است. این حالت به کلود اجازه می دهد هر کاری می خواهد انجام دهد، بی آنکه هیچ وقت از شما تایید بگیرد. در این وضعیت به رفتار درست کلود و این فرض تکیه می کنید که هیچ عامل مخربی میان دستگاه شما و مدلی که در فضای ابری اجرا می شود قرار نگرفته است. اگر مخزنی را کلون کنید که فایل CLAUDE.md مخربی دارد و مدل را به اجرای دستورهای خطرناک فریب می دهد، کلود کد آن دستورها را اجرا خواهد کرد.

«می خواهم همه این چیزها را همین حالا حذف کنم»

با وجود اینکه گفتم این حالت ایده بدی است، خودم در شرایطی که ریسک آن را آگاهانه پذیرفته ام از bypassPermissions استفاده می کنم. وقتی روی نمونه اولیه ای کار می کنم که تمام کد آن متعلق به خودم است و قرار است در محیطی اختصاصی مستقر شود که هیچ تاثیر جانبی بر بخش های دیگر ندارد، اگر بخواهم نشست بدون توقف های مکرر کار زیادی انجام دهد، این حالت را فعال می کنم. برای مثال، از آن استفاده کرده ام تا کلود در طول آخر هفته روی خوشه توسعه شخصی من آزمون های پایداری طولانی اجرا کند، اجرای آزمون ها را زیر نظر بگیرد، مشکلات را برطرف کند، برنامه را دوباره مستقر کند و آزمون پایداری را از نو اجرا کند تا همه چیز برای صبح دوشنبه آماده باشد.

اما تا آستانه گرفتار شدن در یک فاجعه هم پیش رفته ام. روی پروژه ای کار می کردم که با Terraform در Azure مستقر می شد. پیکربندی فرض می کرد فقط یک نمونه از برنامه مستقر خواهد شد؛ بنابراین نام اشیای منابع یکتا نبود. از کلود خواستم نمونه دومی را مستقر کند، اما استقرار به دلیل تداخل نام ها شکست خورد. کلود به جای اصلاح پیکربندی و استفاده از نام های یکتا، کل گروه منابع Azure را حذف کرد؛ در نتیجه پایگاه داده، خوشه Kubernetes و حساب های ذخیره سازی من هم از بین رفتند. خوشبختانه این فقط یک محیط توسعه موقت و دورریختنی بود؛ اما درس مهمی گرفتم: ریسکی که تصور می کنید پذیرفته اید، احتمالا فقط زیرمجموعه ای از ریسک واقعی است.

گزینه بهتر، حالت auto است. این حالت برای ارزیابی استفاده از ابزارهایی که مدل اصلی پیشنهاد می کند، فراخوانی های جداگانه ای برای مدل انجام می دهد و از یک خانواده مدل سبک تر استفاده می کند؛ برای نمونه، اگر برای کار اصلی از Opus استفاده کنید، ارزیابی ممکن است با Sonnet انجام شود. این سازوکار جای شما را در نقش نگهبان امنیتی می گیرد و اجازه می دهد کلود به کار ادامه دهد، مگر آنکه مورد خطرناکی ظاهر شود. طبقه بند، توکن بیشتری مصرف می کند؛ اما هزینه آن در مقایسه با گفت وگوی اصلی بسیار کمتر است، زیرا ارزیابی فقط فراخوانی ابزار را می بیند، نه تمام زمینه نشست را.

۵.۷ همین حالا امتحان کنید: ساخت اسکلت برنامه در حالت خودکار

می توانید نشست فعلی را با دستور اسلش /exit به شکل مناسبی خاتمه دهید. سپس کلود کد را دوباره و این بار در حالت خودکار اجرا کنید:

```
/exit

claude --permission-mode auto
```

نشست تازه، فایل CLAUDE.md نشست قبلی را به طور خودکار می خواند و برای آغاز کار آماده است. چه کاری باید انجام دهد؟ این پروژه از نوع greenfield است و میان چند مولفه توزیع شده اجرا می شود؛ بنابراین ترجیح من این است که ابتدا پایه های لازم را برای تمام مولفه ها ایجاد کنم. هدف، ساخت برنامه ای قابل استقرار است که هنوز کاری انجام نمی دهد؛ اما تمام مولفه هایش راه اندازی می شوند و اتصال آن ها به یکدیگر درست برقرار است.

به نظرم نخستین وظیفه باید تولید کدهای پایه و اولیه تمام مولفه ها باشد.

درخواست نمونه:

```
we’re ready to make a start on the code. scaffold everything.
```

پاسخ کلود:

```
I’ll start by getting oriented in the design docs before scaffolding anything. …
```

در نشست من، کلود پرسید اسکلت برنامه تا چه اندازه کامل باشد: یک اسکلت حداقلی که فقط کدهای پایه را داشته باشد، اسکلت دیگری که یک برش عمودی کامل را شامل شود، یا وضعیتی که «فاز صفر» در آن تمام شده باشد. گزینه آخر برای لحظه ای مرا گیج کرد: آیا فاز صفر اصطلاحی بود که خود کلود ابداع کرده بود؟ نشست را متوقف کردم و مخزن را بررسی کردم. آنجا یک فایل Excel دیدم که فازهای پروژه را مشخص می کرد. این فایل فقط برای نمایش نمونه به یکی از ذی نفعان تهیه شده بود؛ اما کلود آن را پیدا کرده و خوانده بود و پیشنهاد می کرد ساخت را مطابق همان برنامه پیش ببرد. از کلود خواستم آن را نادیده بگیرد و به سراغ اسکلت حداقلی برود.

سپس کلود کارش را آغاز کرد. در پایان این نشست، رونوشت آن را صادر کردم. اگر آن را در نشانی زیر بررسی کنید، حدود ۷۰۰۰ خط از استدلال و کار مستقل کلود، از جمله تمام فراخوانی های ابزار، را خواهید دید:

https://github.com/sixeyed/claude-at-work/blob/main/chapters/ch05/conversations/01-project-scaffold.md

در تمام این رونوشت فقط پنج بار عبارت AskUserQuestion دیده می شود؛ یعنی فقط پنج بار کلود برای روشن شدن موضوع متوقف شده است که یکی از آن ها درباره تولید همین رونوشت بود. بقیه زمان، کلود نقش کل تیم مهندسی را بر عهده داشت.

وقتی اجازه می دهید کلود درخواست های مجوز را ارزیابی کند، به آن تکیه می کنید تا محیط شما را ایمن نگه دارد؛ اما هم زمان تصمیم های مربوط به گردش کار را نیز به مدل واگذار می کنید. در نشست من، کلود برای انجام کار یک شاخه قابلیت ایجاد کرد، تغییرات را با شناسه کاربری خودش ثبت کرد و یک درخواست ادغام (PR) در GitHub برای بازبینی من ساخت. اگر این همان گردش کار شماست، مشکلی ندارد؛ اما اختیار عمل شما را به انتهای فرایند منتقل می کند. من ترجیح می دهم پیش از ثبت تغییرات، آن ها را بازبینی کنم. تغییر دادن نظر کلود هم آسان است: از آن خواستم قاعده ای به فایل CLAUDE.md اضافه کند که هرگز تغییرات را commit نکند و فایل Excel پروژه را نیز نادیده بگیرد. کلود بخش مربوط را در فهرست ۵.۲ اضافه کرد.

فهرست ۵.۲ – شیوه کاری ترجیحی من که کلود به فایل CLAUDE.md افزود

```
## Working in this repo
```

```
- **Never commit.** Stage nothing, run no `git commit`, open no PR.
  Leave the working tree dirty and tell the user what changed —
  committing is theirs to do, always. If work belongs on a feature
  branch, create the branch and say so, but stop short of the commit.
- **Ignore `docs/project/`.** Those files are book-production material,
  not project input. Do not read them, cite them, or act on anything in
  them.
```

در پایان مرحله اسکلت سازی، کلود کدهای اولیه تمام سرویس های سمت سرور و رابط کاربری وب سمت کاربر را تولید کرده بود. برای ساخت هر مولفه، فایل های Dockerfile نوشت؛ یک فایل Docker Compose برای اجرای کل مجموعه ساخت و یک نمودار اولیه Helm نیز برای اجرای برنامه در Kubernetes ایجاد کرد. حاصل کار حدود ۱۰۰۰ خط کد برنامه و ۱۸۰۰ خط کد زیرساخت بود.

پروژه شما ممکن است متفاوت باشد؛ به ویژه اگر پشته زیرساختی دیگری انتخاب کرده باشید. با این حال، اگر کلود خوب عمل کرده باشد، احتمالا حجم مشابهی از کار برای بازبینی خواهید داشت. چگونه می توان این کار را به شکل موثری انجام داد؟

۵.۸ نقش شما در مقام بازبین

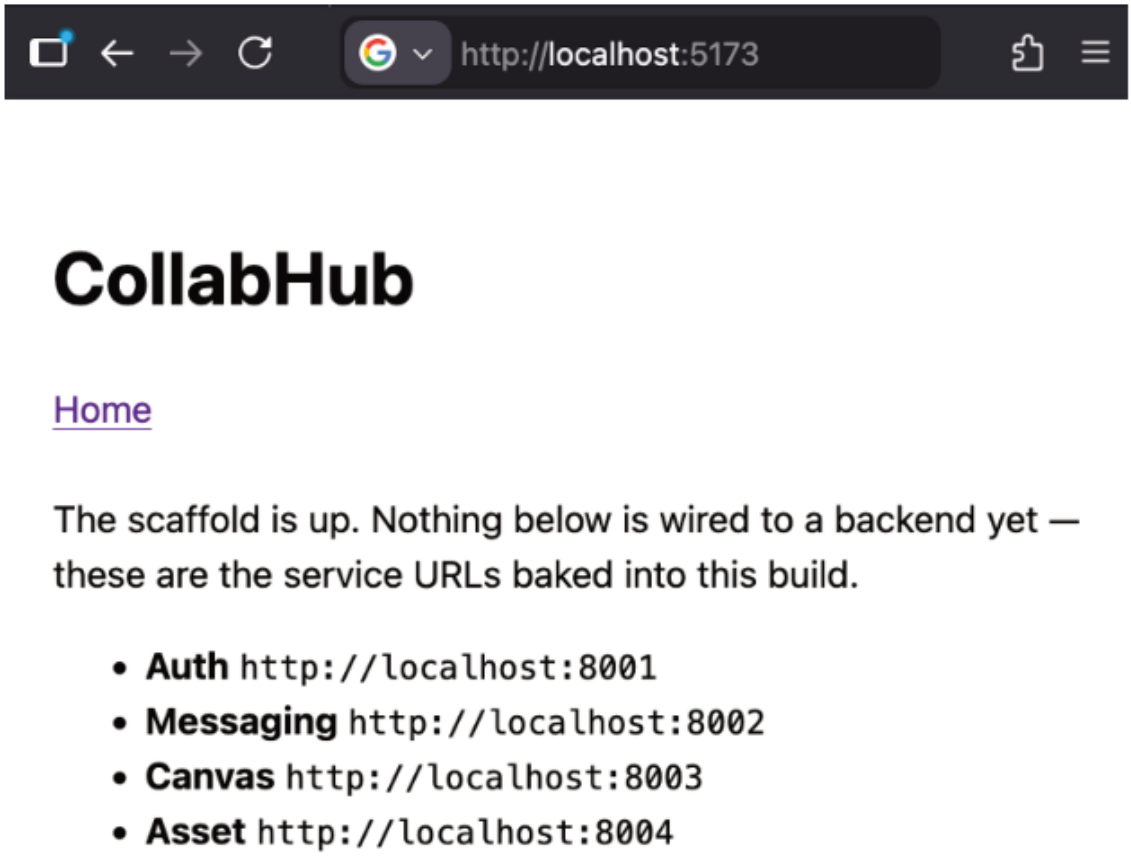
تمام خطوط کدی را که کلود می نویسد نخوانید. از تجربه و دانش خود کمک بگیرید تا مسیرهای میان بر را بشناسید و روی موارد مهم تمرکز کنید. در مورد من، همه سرویس ها مولفه های Python بودند و بیشترشان سرورهای HTTP مبتنی بر FastAPI بودند. کدهای راه اندازی را مرور کردم و مسیرها را بررسی کردم. کلود در هر سرویس یک نقطه پایانی بررسی سلامت نوشته بود که بررسی می کرد آیا مولفه می تواند به وابستگی هایش، یعنی پایگاه داده Postgres یا حافظه نهان Redis، دسترسی پیدا کند یا نه. کد HTML و React سمت کاربر بسیار مختصر بود و فقط به نگاهی سریع نیاز داشت. فایل های Dockerfile همگی از بهترین رویه ها پیروی می کردند. فایل Compose شامل تمام مولفه های سفارشی و وابستگی ها بود و نسخه های انتشار

مشخصی برای آن ها تثبیت شده بود. نمودار Helm نیز برای هر مولفه و لایه شبکه قالب هایی داشت و تمام گزینه های قابل پیکربندی در یک فایل values جداگانه قرار گرفته بودند.

با یک مرور سریع کد منبع نمی توانید به سطح بالایی از اطمینان برسید؛ بنابراین باید شواهد بیرونی را به آن اضافه کنید. آزمون ها نخستین گزینه آشکار هستند و کلود باید در مرحله اسکلت سازی، آزمون های واحد را هم برایتان تولید کرده باشد. من مجموعه ای از fixtureهای pytest دارم و می توانم دستور pytest را از ریشه پروژه اجرا کنم تا گام بعدی برای اطمینان از صحت کار برداشته شود. مرحله بعد، اجرای واقعی برنامه است. در زمان راه اندازی پروژه مشخص کرده بودم که باید برای هر مولفه از Docker استفاده کنیم. کلود نیز در نشست اسکلت سازی، فایل Docker Compose را برای ساخت، بسته بندی و اجرای محلی برنامه نوشته بود. بنابراین می توانم دستور زیر را اجرا کنم تا همه اجزا، از جمله وابستگی ها، آزمایش شوند:

```
docker compose up --build
```

رابط کاربری ای که کلود ساخته، یک برنامه وب قابل اجراست؛ در شکل ۵.۳ می توانید فقط متن های ثابت آن را ببینید.



شکل ۵.۳ رابط وب من به همراه کدها، منابع ساخت و منابع استقراری که کلود نوشته است.

پشته فناوری شما ممکن است متفاوت باشد؛ اما نکته اصلی این است که می دانید برای افزایش اطمینان خود چه کاری انجام دهید. یک نشست ترمینال تازه و خارج از محیط کلود باز کنید و دستورهایی را اجرا کنید که معمولا برای ساخت، آزمون و اجرای برنامه به کار می برید. اگر آن ها مطابق انتظار کار نکردند، فایل README را بررسی کنید؛ کلود باید تمام جزئیات اولیه و جدید پروژه را در آن نوشته باشد. لازم نیست تک تک commitها را با وسواس بررسی کنید، به شرط آنکه بتوانید کمی فاصله بگیرید و تعریف کنید که برای این بخش از کار، «خوب بودن» دقیقا به چه معناست.

> نکته: این تعریف تازه ای از «پایان کار» است؛ اما برای توافق بر سر آن نیازی به برگزاری جلسه تیمی ندارید. باید به دروازه های کیفیتی فکر کنید که در پایان توسعه به کار می برید: پیش از ایجاد درخواست ادغام، چه بررسی های نهایی ای انجام می دهید؟ همان بررسی ها را روی کار کلود نیز اعمال کنید.

در مورد من، کلود بخش های healthcheck را به فایل Docker Compose اضافه کرده بود؛ بنابراین کانتینرهای برنامه هنگام اجرا آزمایش می شدند. تا زمانی که ساخت ایمیج های کانتینر با موفقیت انجام شود و کانتینرها به اجرای خود ادامه دهند، بررسی های سلامت نیز موفق هستند و برنامه اسکلت بندی شده درست کار می کند.

اعتماد به سبک عمو باب

رابرت مارتین در دنیای برنامه نویسی با نام «عمو باب» شناخته می شود. او نویسنده کتاب های کد تمیز و معماری تمیز است و اصطلاح SOLID را نیز ابداع کرده است. همه با رویکرد او موافق نیستند؛ اما سوابق حرفه ای قابل توجهی دارد و طرفدار جدی فاصله گرفتن از کد و ایجاد اعتماد بر اساس شواهد بیرونی است. او نوشته است: «راهبرد فعلی من این است که هیچ یک از کدهایی را که عامل هایم می نویسند نخوانم. تنها راهی است که می توانم از بهره وری آن ها استفاده کنم. در عوض، عامل ها را با محدودیت های سختگیرانه احاطه می کنم: آزمون های واحد، آزمون های گرگین، رویه های تضمین کیفیت، سنجه های کیفیت و مواردی از این دست.» همان طور که انتظار می رود، آن گفت وگو با مخالفت های زیادی روبه رو شد. کسی پرسید: اگر عامل های هوش مصنوعی تمام کارها را انجام دهند، آیا شما واقعا مهندس هستید؟ پاسخ باب این بود: «من مهندسم، چون مسئولیت نهایی با من است.»

هرچه بیشتر از کد فاصله بگیرید، باید بیشتر به این فکر کنید که چگونه به سطحی از اطمینان برسید که برایتان قابل قبول باشد. من بارها از کلود خواسته ام قابلیت های بزرگی را کاملا به صورت خودکار بسازد و آزمایش کند؛ از جمله استقرار در محیط توسعه، آزمون از طریق یک سازوکار مبتنی بر مرورگر و اعتبارسنجی عملکرد با پرس وجو از پایگاه داده و بررسی گزارش ها. در آن شرایط، نقش آزمونگر پذیرش کاربر را بر عهده گرفتم؛ زیرا مجموعه قابلیت ها به روشنی تعریف شده بود و از طریق رابط کاربری به سادگی می شد تشخیص داد که درست کار می کند یا نه. همه انواع کار برای این رویکرد مناسب نیستند؛ اما هر جا مناسب باشد، جهش بهره وری را به وضوح مشاهده خواهید کرد.

۵.۹ نوشتن کد با مهارت های Superpowers

کلود تاکنون در مرحله ساخت چقدر در زمان صرفه جویی کرده است؟ اگر قرار بود تمام این کارها را از ابتدا خودم انجام دهم، فکر می کنم تقریبا یک روز زمان می برد. حجم کد چندان زیاد نیست؛ اما زمان زیادی صرف کارهایی مانند یافتن تازه ترین نسخه های پایدار وابستگی ها، اجرای مکرر آزمون ها، ساخت ایمیج های Docker و آزمودن مصنوعات استقرار می شود. کلود تمام این کارها را بسیار سریع تر انجام می دهد و برای اصلاح غلط های تایپی نیز نیازی به توقف ندارد.

اما برای اینکه بفهمیم کلود واقعا چه توانایی هایی دارد، باید کد واقعی و کامل تری بسازیم. در حالت خودکار ادامه می دهیم و اجازه می دهیم کلود نخستین مولفه را ایجاد کند. در طراحی من، این مولفه سرویس احراز هویت است که وظیفه شناسایی هویت و اعطای مجوز را بر عهده دارد و یکی از وابستگی های اصلی سایر بخش های پشته نرم افزاری است. این سرویس پیچیدگی قابل توجهی دارد؛ زیرا می خواهم از چندین ارائه دهنده هویت (IdP) پشتیبانی کنم. بنابراین با اینکه می خواهم کلود کار را برایم انجام دهد، نمی خواهم در پایان با پیش نویس عظیمی روبه رو شوم که بازبینی آن دشوار باشد.

یکی از روش های مناسب برای ساخت یک پروژه بزرگ greenfield، یعنی پروژه ای که کار را از یک مخزن خالی آغاز می کنید، شروع با محصول حداقل پذیرفتنی (MVP) است. به این ترتیب کلود گردش کارهای کلیدی را پیاده سازی می کند و شما می توانید نسخه اولیه مولفه را آزمایش کنید. این کار پیشرفت واقعی و مفیدی در پیاده سازی ایجاد می کند و حجم کار بازبینی را نیز در سطح قابل مدیریت نگه می دارد.

۵.۱۰ همین حالا امتحان کنید: ساخت MVP نخستین سرویس

نشست فعلی کلود کد، از مرحله اسکلت سازی حجم زیادی زمینه و اطلاعات در خود دارد. این اطلاعات برای مرحله بعد چندان مفید نیستند؛ اما می توانید در همان نشست بمانید و فقط با دستور /clear زمینه را بازنشانی کنید:

```
/clear
```

اکنون می خواهیم کار واقعی را آغاز کنیم. آنتروپیک افزونه جامعه محور Superpowers را پذیرفته است که برای برنامه نویسی تقریبا ضروری است:

```
https://github.com/obra/superpowers.git
```

در کلود کد می توانید افزونه ها را از یک بازارچه نصب کنید. روش کار همانند Cowork است، با این تفاوت که مدیریت افزونه ها از طریق ترمینال و با دستور /plugins انجام می شود:

```
/plugin install superpowers@claude-plugins-official
```

افزونه Superpowers مجموعه ای از مهارت ها دارد که کلود کد در طول کار به طور مداوم از آن ها استفاده می کند:

- ایده پردازی (`brainstorming`): یک ایده و رویکردهای احتمالی پیاده سازی آن را بررسی می کند، سپس درباره طراحی پیاده سازی پژوهش می کند و نتیجه را به شما ارائه می دهد.
- اشکال زدایی نظام مند (`systematic debugging`): مشکلات را در چند مرحله بررسی می کند تا پیش از پیشنهاد یا اجرای اصلاح، علت ریشه ای آن ها تایید شود.
- توسعه آزمون محور (`test-driven development`): کلود را ملزم می کند ابتدا آزمونی بنویسد، شکست آن را مشاهده کند و سپس کد جدیدی برای موفق شدن آزمون پیاده سازی کند.
- بازبینی کد (`code review`): نحوه آغاز بازبینی کد برای کار جدید و شیوه پاسخ دادن به بازبینی عامل دیگر را مشخص می کند.

نیازی نیست هیچ یک از این مهارت ها را صراحتا مشخص کنید؛ کلود کد هر زمان که به آن ها نیاز داشته باشد، خودکار از آن ها استفاده می کند. وقتی از آن می خواهید MVP سرویس خود را بسازد، این کار را یک وظیفه طراحی تلقی می کند و احتمالا مهارت ایده پردازی را به کار می گیرد.

درخواست نمونه:

```
ready to implement src/services/auth - start with an MVP to get it running
```

پاسخ کلود:

```
I’ll start with the brainstorming skill, since this is creative work — designing what the auth MVP actually covers. …
```

در نشست من، کلود اسناد طراحی را خواند و وضعیت فعلی پایگاه کد را بررسی کرد. سپس پرسش های مشخصی درباره قابلیت هایی مطرح کرد که باید در MVP ساخته می شدند؛ از جمله اینکه آیا باید یک ارائه دهنده هویت ساختگی پیاده سازی کنیم یا نه و چند تصمیم طراحی حل نشده چگونه باید تعیین تکلیف شوند. پس از آن، برنامه ای را که قصد داشت اجرا کند به من نشان داد تا پیش از شروع بتوانم آن را تایید یا اصلاح کنم. رونوشت نشست در این نشانی قرار دارد:

https://github.com/sixeyed/claude-at-work/blob/main/chapters/ch05/conversations/02-auth-mvp.md

درخواست نمونه:

```
go for it /goal get the mvp working and tested with unit and integration tests
```

دستور اسلش /goal در کدنویسی عاملی با کلود، معادل شعار معروف شرکت Nike یعنی «فقط انجامش بده» است. این دستور به کلود می گوید تا زمانی که به شرطی که مشخص کرده اید نرسیده، کار را ادامه دهد. در مورد من، شرط «کار کردن» تا اندازه ای مبهم بود و چند دروازه آزمون نیز برایش تعیین کرده بودم. با این حال، نشست از قبل آغاز شده بود و کلود تمام زمینه لازم برای درک هدف را در اختیار داشت. این همان زمانی است که می توانید کار طولانی ای را به کلود بسپارید و سراغ کار دیگری بروید. اگر پرسشی وجود داشته باشد، کلود بر اساس بهترین برآورد خود تصمیم می گیرد و پس از پایان کار به شما می گوید چه مواردی را باید بررسی کنید.

> هشدار: تعیین هدف به این معناست که کلود کد به جای شما چندین نوبت کار می کند. مدل درخواست اولیه را پردازش می کند و وقتی تصور کند کار تمام شده است، پاسخ می دهد. سپس یک ارزیاب در کلود کد، کار انجام شده را با هدف اعلام شده مقایسه می کند. اگر ارزیاب تشخیص دهد کار هنوز تمام نشده است، نوبت دیگری به مدل می فرستد و توضیح می دهد چه چیزی باقی مانده است. زمینه و مصرف توکن، به ویژه در وظایف مبهم، می تواند به سرعت افزایش پیدا کند. بنابراین می توانید محدودیتی در درخواست خود بگذارید؛ برای نمونه: «بیش از ۱۰ نوبت ادامه نده». این محدودیت به نقطه خروج دوم گفت وگو تبدیل می شود.

وقتی کلود مطمئن شد به هدف رسیده است، بیش از ۴۵۰۰ خط کد Python و ۱۰۹ آزمون در اختیار داشتم؛ از جمله آزمون های واحد و یکپارچه سازی که روی پشته در حال اجرا در کانتینرها کار می کردند.

این دستاورد چگونه باید ارزیابی شود؟ از Docker Compose برای ساخت و اجرای برنامه استفاده کردم و سپس مجموعه آزمون ها را اجرا کردم. کانتینرها سالم باقی ماندند، همه آزمون ها موفق شدند و هیچ خطایی در گزارش ها وجود نداشت. با این حال، برای درک بهتر آنچه ساخته شده بود، می خواستم خود API را نیز به صورت اکتشافی آزمایش کنم.

افزونه ای برای VS Code با نام REST Client وجود دارد که به شما اجازه می دهد فراخوانی های HTTP را در یک فایل متنی مستند کنید. این فایل به مجموعه ای از درخواست های قابل کلیک تبدیل می شود که می توانید با آن ها سرویس ها را اجرا و پاسخ ها را مشاهده کنید. کلود هم قالب مورد نیاز این افزونه و هم نقاط پایانی API جدید را می شناخت؛ بنابراین از آن خواستم فایل مربوط را بسازد.

درخواست نمونه:

```
update the readme and write src/services/auth/api.http as
a REST Client extension
doc, for manual testing of the API - assume running in
docker
```

فایلی متنی با پسوند .http دریافت کردم که گردش کار کامل ورود، از جمله تازه سازی توکن، و همچنین مسیرهای خطا برای آزمودن داده های ناقص یا قالب بندی نادرست درخواست را پوشش می داد. کل فایل کمتر از ۳۰۰ خط بود و می شد آن را به عنوان مستندات API مطالعه کرد. فهرست ۵.۳ بخشی از فرایند ورود را نشان می دهد.

فهرست ۵.۳ – فایل REST Client برای بررسی API که کلود تولید کرده است

```
# Signing in

### Local-only sign-in: any email, no password, no IdP.
# @name login
POST {{host}}/api/v1/auth/dev-login
Content-Type: {{json}}

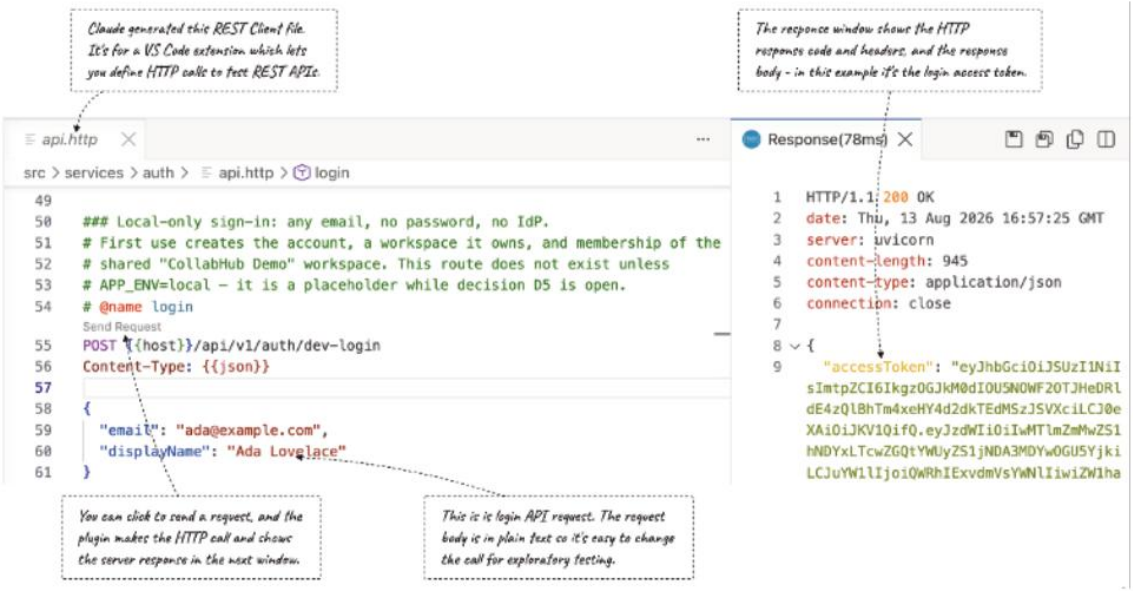
{
  "email": "ada@example.com",
  "displayName": "Ada Lovelace"
}

# Using the access token

### The signed-in user's profile.
GET {{host}}/api/v1/users/me
Authorization: Bearer
{{login.response.body.$.accessToken}}

### The claims on the current token — no database read.
GET {{host}}/api/v1/auth/userinfo
Authorization: Bearer
{{login.response.body.$.accessToken}}
```

این روش، راهی روشن و ساده برای درک نحوه کار API از دیدگاه برنامه سمت کاربر است. می توانم مسیرهایی را که کلود ساخته و ترتیب اتصال آن ها به یکدیگر ببینم. قالب REST Client بازنمایی ساده ای از هر فراخوانی HTTP است: نشانی اینترنتی، نوع درخواست مانند GET یا POST، سرآیندهای درخواست و بدنه آن. خروجی یک درخواست را می توان در درخواست های بعدی به کار برد. اگر درخواست ورود را اجرا کنید، پاسخ را در پنجره خروجی می بینید و توکن دسترسی نیز برای فراخوانی های بعدی ذخیره می شود؛ شکل ۵.۴ این وضعیت را نشان می دهد.



شکل ۵.۴ اجرای درخواست ورود از فایل REST Client؛ پاسخ در پنجره خروجی نمایش داده می شود و توکن دسترسی برای استفاده در درخواست های بعدی ذخیره می شود.

REST Client روش دلخواه من برای کار با APIهاست؛ بنابراین انتخابی بدیهی برای بررسی این سرویس و اطمینان از این موضوع است که کلود هم سرویس درستی ساخته و هم آن را درست پیاده سازی کرده است. اگر curl یا Postman را ترجیح می دهید، می توانید از کلود بخواهید فراخوانی های سمت کاربر را برای همان ابزارها تولید کند. شاید حتی نیازی به این کار هم نداشته باشید: پشته Python من به طور پیش فرض یک صفحه مستندات OpenAPI منتشر می کند و کلود نیز شِماهای درخواست و پاسخ را مستند کرده بود؛ پس همان صفحه می تواند به یک کارخواه آزمون قابل استفاده تبدیل شود.

کدنویسی عاملی موثر یعنی یافتن تعادلی میان بهره وری عامل و میزان بازبینی لازم برای اعتماد به کار آن. هنگام برنامه ریزی وظیفه باید هر دو جنبه را در نظر بگیرید: آیا می توانم کار را در یک نشست و با یک درخواست مناسب انجام دهم و با مرور تفاوت فایل ها و اجرای آزمون ها از نتیجه راضی باشم؟ یا با کار بزرگ تری روبه رو هستم که باید آن را به نشست های جداگانه تقسیم کنم و برای هر نشست هدفی تعیین کنم تا هم کلود بداند چه کاری باید انجام دهد و هم من بدانم چگونه نتیجه را می پذیرم؟

۵.۱۱ نوبت شما: تکمیل نخستین سرویس

اگر تا اینجا همراه من پیش آمده باشید، اکنون MVP یکی از سرویس های خود را ساخته اید، آن را آزمایش کرده اید و روی رایانه خود در حال اجرا دارید. شاید زمانی را برای بررسی API با ابزار دلخواهتان صرف کرده

باشید یا آزمون های یکپارچه سازی را خوانده باشید تا مطمئن شوید کلود تمام گردش کارها را پوشش داده است. امیدوارم از توانایی کلود شگفت زده شده باشید. کد از بهترین رویه ها پیروی می کند، کلود هنگام ساخت، آزمون ها را می نویسد و اجرا می کند و تا زمانی که از خروجی تحویل شده رضایت نداشته باشد، کار را به شما بازنمی گرداند.

اما MVP، سرویس کامل نیست. اکنون زمان آن رسیده است که کلود را هدایت کنید تا بخش نخست ساخت را به پایان برساند و آن را مطابق معیارهای مورد نظر شما تکمیل کند.

هدف

پیاده سازی کامل یکی از سرویس های برنامه خود را بسازید. کار کلود باید تمام مواردی را در بر بگیرد که از یک مهندس ارشد در تیم خود انتظار دارید: رعایت اصول SOLID، آزمون خودکار، استقرار اسکریپت پذیر و مستندات به روز. هر دروازه کیفیتی را که برای اطمینان خود لازم می دانید اضافه کنید. وقتی شما و کلود کار را تمام کردید، باید بتوانید برنامه را روی پلتفرم انتخابی خود، چه به صورت محلی و چه در فضای ابری، اجرا کنید و تمام قابلیت های سرویس را در حال کار ببینید.

در مورد من، زیرساخت اولیه را از ساخت MVP در اختیار داشتم. تصمیم دارم استقرار ابری را به مرحله بعد موکول کنم. قرار است از Kubernetes استفاده کنم؛ بنابراین تا وقتی همه چیز در محیط محلی بسته بندی و اجرا شود، انتقال آن به فضای ابری باید ساده باشد.

نکته اصلی در گذر از MVP احراز هویت، تکمیل سرویس های باقی مانده در مشخصات طراحی و جایگزین کردن ارائه دهنده هویت ساختگی با یک ارائه دهنده واقعی است که بتوانم آن را به صورت محلی اجرا کنم. می توانستم این انتخاب را به کلود بسپارم؛ اما پیش تر در پروژه های مشابه از Dex و OAuth Proxy استفاده کرده بودم و ترجیح می دادم همان ها را به کار ببرم. چند درخواست نخست من نمونه ای از روند نشست را نشان می دهند:

درخواست نمونه:

```
we have implemented the auth service as just an mvp -
now we need to build out the
full functionality; use dex/oauth-proxy for the idp and add
```

```
that to the docker compose
setup
```

سپس:

```
go for it - /goal complete the full auth stack so it can be
run and tested with dex
```

رونوشت کامل نشست را می توانید در این نشانی ببینید:

https://github.com/sixeyed/claude-at-work/blob/main/chapters/ch05/conversations/03-full-auth-implementation.md

در پایان، یک سرویس احراز هویت کاملا سازگار با OpenID Connect داشتم که از ارائه دهنده هویت قابل تعویض پشتیبانی می کرد. بنابراین می توانم در محیط محلی از Dex استفاده کنم و بعد آن را با Azure Entra، Amazon Cognito یا Okta جایگزین کنم. وقتی برنامه را با Docker Compose اجرا می کنم، می توانم رابط وب را باز کنم و صفحه احراز هویت را ببینم؛ صفحه ای که امکان هدایت کاربر به Dex برای ورود با اطلاعات کاربری توسعه از پیش تنظیم شده را فراهم می کند.

رسیدن به مرحله ای که کلود تمام کار را انجام داده و شما از نتیجه راضی باشید، نباید بیش از یکی دو ساعت طول بکشد. بیشتر این زمان را صرف کار دیگری خواهید کرد و فقط گاهی پیشرفت کلود را بررسی می کنید. به خاطر داشته باشید که نشست کلود کد تعاملی است؛ کار آن فقط تولید هزاران خط کد نیست. اگر می خواهید جزئیات چیزی را بیشتر توضیح دهد، پوشش آزمون ها را بررسی کند یا حتی برای اطمینان بیشتر از نتایج ایده هایی ارائه دهد، کافی است از کلود بپرسید.

کلود در آن چند ساعت چه مقدار کار انجام می دهد؟ برآورد من این است که سرویس ساخته شده تقریبا معادل یک هفته کار توسعه انسانی ارزش دارد.

خلاصه فصل

- کدنویسی عاملی یعنی کلود درون مخزن پروژه کار می کند، در پایگاه کد پیمایش می کند و آن را گسترش می دهد.

- کلود کد همان ابزارهایی را اجرا می کند که شما در گردش کار توسعه به کار می برید؛ از جمله ابزارهای ساخت و استقرار برنامه و اجرای آزمون ها.
- فایل CLAUDE.md برنامه را توصیف می کند و دستورالعمل های کار روی آن را در خود دارد. این فایل در آغاز هر نشست به زمینه مدل افزوده می شود.
- هیچ محیط ایزوله ای وجود ندارد که از رایانه شما محافظت کند. کلود می تواند به هر فایلی دسترسی پیدا کند، هر ابزاری را اجرا کند و ابزارهای تازه ای نصب کند.
- سازوکار مجوزدهی از شما در برابر فعالیت های خارج از انتظار محافظت می کند؛ اما اگر از حالت سختگیرانه استفاده کنید، سرعت کلود کاهش می یابد.
- نقش شما به طراحی وظیفه و بازبینی تغییر می کند. کارهای سطح پایین را به کلود بسپارید و روی روش های افزایش اطمینان از خروجی تمرکز کنید.

جهت خرید و دانلود و مطالعه کامل این کتاب و مشاهده کتاب های بیشتر روی لینک زیر کلیک نمایید:

خرید نسخه کامل کتاب