Coding mit KI

Das Praxisbuch für die Softwareentwicklung

DIE LESEPROBE

Kapitel 4
Debugging

Sobald ein Software-Projekt eine gewisse Größe überschreitet, schleichen sich Fehler in den Quellcode ein. Diese Fehler zu finden und zu beheben, ist das Ziel beim Debugging. Meistens treten diese Fehler im Laufe der Entwicklung auf, oft wird man auch von Anwendern erst später darauf hingewiesen. Wie auch immer, wir müssen das Problem lösen. Und wenn wir es mithilfe der KI schneller lösen können, dann umso besser.

In der Software-Entwicklung haben wir unterschiedliche Werkzeuge, um Problemen auf die Spur zu kommen. Temporäre `Print`-Anweisungen (oder `console.log` für Webapplikationen) im Code sind der einfachste und oft schnellste Weg zum Erfolg. Strukturiertes Logging mit unterschiedlichen Kategorien und weiteren Metadaten sind in Anwendungen hilfreich, die produktiv von Kunden eingesetzt werden. Ein Debugger, der den Code zur Laufzeit analysieren und anhalten kann, ist ein unverzichtbares Werkzeug für komplexe Anwendungen.

Diese Tools helfen Ihnen, Fehler zu erkennen und zu verstehen. Zaubermittel sind sie aber nicht: Es liegt immer noch an Ihnen, das Problem zu durchdenken, Fehlerquellen auszuschließen und nach der Ursache zu forschen. Oftmals sieht man dabei den Wald vor lauter Bäumen nicht mehr: Man ist so in den eigenen Code vertieft, dass man Dummheiten gar nicht mehr wahrnimmt. Wenn man sich erst mal in einer Sackgasse richtig verrannt hat, ist es schwer, alleine den Ausweg zu finden.

In der modernen Software-Entwicklung wird deswegen oft im Team gearbeitet: Beim *Pair Programming* arbeitet man gemeinsam am Code, denn vier Augen (und zwei Hirne) erkennen Probleme schlicht mehr als doppelt so schnell. Wenn kein Kollege verfügbar ist, hilft oft das *Rubber Duck Debugging*: Man schnappt sich eine Gummiente und erklärt ihr das Problem. Und da die Ente kein Programmierprofi ist, müssen diese Erklärungen langsam und kleinschrittig sein. Meistens fällt einem die Lösung dann auf, wenn man den Fehler so heruntergebrochen hat.

Aber was, wenn die Gummiente nicht nur eine passive Zuhörerin wäre, sondern selbst mit Analysen, Lösungsvorschlägen und Ideen glänzen könnte? Damit hätten Sie einen Partner im Pair Programming, der niemals mürrisch oder gelangweilt ist,

immer Zeit hat und zudem ordentliches Grundlagenwissen in fast jeder Sprache und jedem Framework mitbringt.

In diesem Kapitel werden wir daher versuchen, Fehler in unserem eigenen Quellcode mit KI-Unterstützung zu eliminieren. Dank unserer Erfahrung beim Beheben der Fehler können wir abschätzen, wie groß der Zeitunterschied durch die KI-Hilfe ist.

Der Nachteil dabei ist, dass sich unsere Kenntnis des Lösungswegs auf die Fragestellung auswirken kann. Genau das ist nämlich der zentrale Punkt bei der Formulierung des Prompts: Wenn Sie Ihrem KI-Helfer bereits eine Richtung mitgeben und ihn durch Keywords auf eine bestimmte Fährte schicken, wird er sehr wahrscheinlich Ihre Gedankengänge aufgreifen und replizieren. Falls sich das Problem damit nicht lösen lässt, kann es sinnvoll sein, eine neue Session zu starten und die Anfrage offener und allgemeiner zu formulieren. Das widerspricht den Ratschlägen, die wir sonst im Buch geben, nämlich dass die Prompts möglichst konkret und mit viel Kontext formuliert sein sollten, damit die KI weiß, was sie tun muss.

Wenn Sie sich aber selbst nicht sicher sind, wo sich das Problem versteckt und was der richtige Lösungsweg ist, geben Sie der KI besser etwas mehr Leine. Bitten Sie die KI um unterschiedliche Ansätze und Blickwinkel auf das Problem. Aber dann ist es natürlich Ihre Aufgabe, zwischen den verschiedenen Ideen den Weg zu wählen, der das Problem löst. Die KI kann Ihnen sicherlich beim Debugging einfacher Syntaxprobleme helfen, aber wenn es komplexer wird, gilt hier umso mehr das Fazit: Überprüfen Sie stets alle Antworten, und denken Sie immer mit. Nur weil Ihr KI-Helfer stolz sagt, dass er ein Problem gelöst hat, muss noch lange nicht alles so laufen, wie es soll.

Trotzdem können wir schon verraten, dass unsere Erfahrungen beim KI-gestützten Debugging sehr positiv waren. Die Sprachmodelle sind gut darin, Abweichungen von der Norm zu finden und zu korrigieren. Wenn sich der Fehler in Ihrem Programm damit beschreiben lässt, ist die Wahrscheinlichkeit hoch, dass Ihr KI-Helfer ihn finden wird.

4.1 Webapplikationen

Der Marktanteil von Webapplikationen wächst ungebrochen. Mit der COVID-19-Pandemie wurde der Ruf nach *mehr Online* noch lauter, und wer heutzutage eine Applikation online bringt, tut gut daran, den Webbrowser als ein mögliches Frontend nicht zu vergessen.

Der Browser als betriebssystemunabhängige Plattform entwickelt sich rasant weiter und bietet immer ausgefeiltere Werkzeuge zum Debuggen von Webapplikationen (Stichwort *Developer-Tools*). Wir wollen sehen, ob und wie uns KI-Unterstützung zu noch schnelleren Lösungen führen kann.

CSS Fixes

Wer sich schon einmal an einem vermeintlich einfachen Layout in HTML versucht hat, wird das Problem kennen: Ein Element will einfach nicht an die Position auf der Seite, an der man es gerne haben möchte. Wir haben hier als Beispiel einen sehr einfachen Seitenabschluss als `footer`-Element verwendet. Die Inhalte sollen in drei Bereichen links, mittig und rechts aufgeteilt werden. Unser erster Entwurf hat den Nachteil, dass sich die Bereiche mit der CSS-Eigenschaft `float` nicht an die Höhe des umgebenden Elements halten, sondern auch darüber hinaus fließen können.

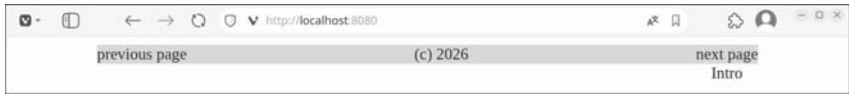

Abbildung 4.1 Fehlerhaftes Layout für den Fußbereich einer HTML-Seite

Wir wollen sehen, ob *Claude 4.5 Sonnet* hier eine bessere Lösung bieten kann:

Prompt: How can I make my footer elements all the same height:

```
<footer style="margin:auto;text-align:center;width:80%;">
  <span style="float:left;">previous page</span>
  <span>(c) 2026</span>
  <span style="float:right;">next page<br />Intro</span>
</footer>
```

> The issue is that the right-floated span has two lines (next page + Intro) while the others have one, making them different heights.
>
> Here are a few solutions:
>
> **1. Quick fix - Add line breaks to match:**
>
> ```
> <footer style="margin:auto;text-align:center;width:80%;">
>   previous page<br /> 
>   (c) 2026
>   next page<br />Intro
> </footer>
> ```
>
> **2. Better approach - Use flexbox:**
>
> ```
> <footer style="margin:auto;text-align:center;width:80%;
>     display:flex;justify-content:space-between;
>     align-items:flex-start;">
>   previous page
>   (c) 2026
> ```

```
  <span style="text-align:right;">next page<br />Intro</span>
</footer>
```

3. Grid layout (most flexible):

```
<footer style="margin:auto;width:80%;display:grid;
    grid-template-columns:1fr auto 1fr;align-items:start;">
  <span>previous page</span>
  <span style="text-align:center;">(c) 2026</span>
  <span style="text-align:right;">next page<br />Intro</span>
</footer>
```

The flexbox or grid solutions are more modern and maintainable than using floats. If you need all spans to actually be the same height (not just aligned), add `align-items:stretch` instead of `align-items:flex-start`.

Als wir uns mit dieser Fragestellung in der ersten Auflage dieses Buchs an *Claude 3.5 Sonnet* wandten, erhielten wir nur die Flexbox-Variante. Damals fragten wir explizit nach, ob es noch andere Lösungen für das Problem gibt, und wir bekamen auch die Grid-Layout-Lösung präsentiert.

Die aktuelle Antwort besticht dadurch, dass wir gleich drei mögliche Lösungen bekommen und diese auch noch korrekt kategorisiert werden: den uneleganten *Quick fix* für den Fall, dass es schnell gehen muss, das komplexe Flexbox-Modell und das, wie wir finden, für den Footer gut geeignete Grid-Layout (siehe Abbildung 4.2). Was uns bei dieser Antwort außerdem gut gefällt, ist, wie präzise alles auf den Punkt gebracht wird: im ersten Satz eine Analyse des Problems, dann die drei Lösungen und abschließend noch zwei Sätze zur Einordnung der beiden ernst zu nehmenden Alternativen.

Abbildung 4.2 Grid-Layout für den Fußbereich einer HTML-Seite mit drei Spalten (mit aktiven Developer-Tools, um die drei Bereiche hervorzuheben)

Unser Fazit für das erste Debugging: Die KI lieferte eine sehr gute Lösung für unser Problem. Dass gleich mehrere Lösungsvorschläge angeboten werden, sehen wir nicht als Nachteil, sondern durch die korrekte Kategorisierung als nützliche Information.

Ical-Download im Browser

Unser nächstes Problem beschäftigt sich mit der Programmiersprache PHP und den Möglichkeiten, externe Programme vom Browser aus zu starten. In unserer Appli-

kation haben wir einen Export für Termine im standardisierten *Ical*-Format erstellt. Beim Anklicken des Links wird aber nicht wie von uns gewünscht die Kalender-Applikation gestartet, sondern der Text wird schwer lesbar im Browser angezeigt. Wir fragen *Claude 4.5 Sonnet*, ob es dafür eine schnelle Lösung gibt:

Prompt: I have this simple PHP class which sends calendar entries over HTTPS. Unfortunatelly it opens in my browser. How can I make it open in my calendar app?

To make calendar entries open in a calendar app instead of the browser, you need to serve the iCalendar data with the correct **Content-Type header** and **Content-Disposition**. Here's what you need to change in your PHP class:

```
// Set the correct MIME type for calendar files
header('Content-Type: text/calendar; charset=utf-8');
// Force download/open in calendar app instead of browser
// display
header('Content-Disposition: attachment;
        filename="event.ics"');
// Prevent caching
header('Cache-Control: no-cache, must-revalidate');
header('Expires: Sat, 26 Jul 1997 05:00:00 GMT');
// Then output your iCalendar data
echo $icalendarContent;
```

Key points:

1. **Content-Type: text/calendar** - This tells the browser it's an iCalendar file
2. **Content-Disposition: attachment** - This triggers a download/open action instead of inline display
3. **filename="event.ics"** - The `.ics` extension helps the OS recognize it as a calendar file

If you want the calendar app to directly import it without prompting for download, you can use inline instead of attachment:

```
header('Content-Disposition: inline; filename="event.ics"');
```

The browser should then pass the file to the system's default calendar application handler (like Apple Calendar, Google Calendar, Outlook, etc.).

Would you like me to look at your actual PHP class code to provide more specific guidance?

Die Lösung ist korrekt, und die drei Änderungen, die als *Key points* erklärt werden, führen in der Tat zum gewünschten Verhalten. Ganz praktisch ist auch noch der Hinweis, dass der Browser den Kalender öffnen *sollte*, wenn `Content-Disposition: inline` ange-

geben wird. Aus der Praxis weiß man natürlich, dass unterschiedliche Browser auf verschiedenen Betriebssystemen ihre Eigenheiten beim Aufruf von externen Anwendungen haben.

Bei diesem Versuch ist uns allerdings ein Missgeschick passiert: Eigentlich wollten wir die entsprechende Codepassage zum Prompt hinzufügen. Umso größer war die Überraschung, als sich der von der KI generierte Code perfekt in unsere Codebasis integrieren ließ. Bei komplexeren Änderungen kann das Mitsenden des eigenen Codes aber sicher weitere Arbeit ersparen.

Backend-Frontend ohne Proxy-Server

Wenn Sie in der Webentwicklung mit getrennten Frontend- und Backend-Anwendungen arbeiten, werden Sie dieses Problem vermutlich kennen: Der Entwicklungsserver für die Backend-Applikation läuft auf Port 8000 und der Bundler mit Live-Reload im Browser (wie vite oder webpack) auf Port 8080. Asynchrone Zugriffe vom Frontend (AJAX) werden vom Browser blockiert.

In solchen Fällen sind die Fehlermeldungen in der Browser-Konsole zwar ausführlich, aber, wenn man sich mit dem Thema noch nicht beschäftigt hat, auch schwer verständlich. Dieses Mal wollen wir den europäischen Modellen eine Chance geben und wählen *Codestral* von der französischen Firma *Mistral*. Wir haben die Meldung im folgenden Prompt etwas gekürzt:

Prompt: My backend application is a golang json api with gin-gonic running on port 8000. Frontend is running with vite.js on port 8080. When I call backend routes I get the following error:

```
Access to fetch at 'http://localhost:8000/users/me'
from origin 'http://localhost:8080' has been blocked by CORS
policy: Response to preflight request ...
```

What can I do?

> **Fixing CORS Issues Between Gin-Gonic Backend and Vite Frontend**
>
> The error you're seeing is a CORS (Cross-Origin Resource Sharing) issue. Your frontend running on port 8080 is trying to access your backend on port 8000, which is considered a different origin by browsers.
>
> **Solutions**
>
> **1. Enable CORS in your Gin backend** The simplest solution is to enable CORS middleware in your Gin application:
>
> ...

Die KI hat das Problem sofort erkannt und korrekt gelöst. Wir müssen das `cors`-Modul von der von uns verwendeten Bibliothek `gin-gonic` einbinden und entsprechend konfigurieren. Wir bekommen noch weitere Konfigurationsvarianten, die sich explizit auf eine Entwicklungsumgebung (`--host` für vite) oder eine Produktionsumgebung beziehen. Ein nützlicher Hinweis dabei ist, dass für ein produktives System ein vorgeschalteter Proxy sinnvoll sein kann.

Perfekt, das Problem scheint gelöst. Nur das Login funktioniert immer noch nicht. Die Anfrage mit Benutzernamen und Passwort wird zwar mit dem HTTP-Statuscode `200` als gültig bestätigt, wir werden aber immer wieder zur Login-Seite zurückgeleitet. Vielleicht kann die KI auch hier weiterhelfen?

Prompt: Great! My requests are working now, but my login functionality is broken. Session variables in go are not present. What is the problem?

> I'm glad the CORS issue is resolved! Now, regarding your login functionality and session variables not being present, there could be a few reasons for this. Let's explore some common issues and solutions:
>
> ...

Die KI ist offenbar froh, dass wir das erste Problem lösen konnten. Und froh sind auch wir über die folgende Antwort: Wir werden darauf hingewiesen, dass man beim Übertragen von Zugangsdaten eine weitere Einstellung am Frontend und am Backend vornehmen muss. Das konnte die KI im Vorhinein nicht wissen, wir haben bei unserer ersten Frage ja nichts von einem Login-Problem gesagt. Die ausführliche Antwort enthält auch Anweisungen, um die Cookies, die für die Session verwendet werden, korrekt zu speichern (sodass sie eben auch für andere Ports gelten).

Einzig hier hätten wir erwartet, dass die KI gleich einen Lösungsvorschlag für die von uns verwendete Bibliothek (`gin-gonic`) und das Session-Management dort liefert. Die Antwort erklärte uns die Einstellung für Cookies allgemein in der Programmiersprache Go. Wir müssen also noch einmal nachfragen:

Prompt: I am using gin-gonic sessions, how do I set relaxed restrictions for samesite here?

Jetzt bekommen wir die korrekte Antwort, und die Applikation funktioniert wie gewünscht. Im Entwicklungsprozess wird für diesen Zweck auch gerne ein Proxy-Server verwendet, der Frontend und Backend vom gleichen Port und vom gleichen Host an den Browser sendet. Wenn Sie die Applikation produktiv auch verteilt (also nicht am gleichen Host) betreiben wollen, müssen Sie sich mit CORS ohnehin beschäftigen.

Informationen zum Thema *CORS* gibt es im Internet zur Genüge, und sie sind auch leicht zu finden. Eine Websuche nach den Begriffen *CORS* und *MDN* (für *Mozilla developer network*) führt uns zu den ausgezeichneten Webseiten, die das Problem im Detail beschreiben. Wir bekommen aber nicht die fertige Lösung für unser Programm in der Programmiersprache *Go*, wie es die KI geliefert hat. Natürlich kann man hier einwenden, dass es sich um ein sicherheitsrelevantes Thema handelt und man nicht Code ohne tiefere Kenntnis der Materie übernehmen sollte. In diesem Fall hat die KI aber gute Erklärungen geliefert, und wir konnten die Anwendung nach kürzester Zeit entwickeln, ohne einen halben Tag nach Lösungen bei Stack Overflow zu suchen (wie wir es ohne KI-Hilfe machten).

JavaScript-fetch mit falschem Content-Type

Unser nächstes Problem stammt von einer Umstellung, bei der wir eine alte JavaScript-Bibliothek zum Ausführen von asynchronen Webanfragen im Browser (AJAX) durch die moderne `fetch`-Schnittstelle ersetzt haben. Die meisten Aufrufe liefen problemlos, aber an ein paar Stellen hakte die Umstellung ein wenig. Ob uns die KI auch für dieses Problem eine schnelle Lösung bieten kann?

Für den Versuch kopieren wir die gesamte Funktion, die das Problem auslöst, und die Fehlermeldung aus der Browser-Konsole in den Prompt (die Antwort stammt von *Codestral*):

Prompt: javascript: When I call this function:

```
const _newComment = function (topic, meetingId, event) {
  event.target.disabled = true;
  fetch(`api/comments`, {
    body: JSON.stringify({
      id: topic,
      comment: this.newCommentContent.trim(),
      meetingId,
    }),
  })
    .then((json) => {
      this.comments.push({
        id: json.id,
        content: this.newCommentContent.trim(),
        date: moment().format("YYYY-MM-DD"),
        meeting: meetingId,
        user: this.$store("user").fullname,
      });
      this.newCommentContent = "";
```

```
    })
    .finally(() => {
      autosize.update(event.target);
      event.target.disabled = false;
      this.$focus.focus(event.target);
    });
};
```

I get this error message in my browser console window:

```
comments.js:8 Uncaught (in promise)
TypeError: Failed to execute 'fetch' on 'Window': Request with
  GET/HEAD method cannot have body.
    at Proxy._newComment (comments.js:8:3)
    ...
```

> The error occurs because you're trying to send a request body with a `GET` request, which is not allowed in HTTP. By default, `fetch()` uses the `GET` method when no method is specified.
>
> To fix this, you need to explicitly set the method to `POST` (or another appropriate HTTP method) in your fetch request options. Here's the corrected version of your function:
>
> ... Code ...
>
> Key changes made:
>
> 1. Added `method: 'POST'` to the fetch options
> 2. Added `headers` with `'Content-Type': 'application/json'` (recommended for JSON payloads)
> 3. Added `response.json()` to properly parse the response (assuming your API returns JSON)
>
> If your API expects a different HTTP method (like PUT), you should use that instead of POST. Also, make sure your server-side endpoint is configured to handle POST requests to `/api/comments`.

Hier hat uns die KI wieder einmal zum Staunen gebracht: Der Fehler, dass beim Senden der Daten mit `fetch` die falsche HTTP-Methode verwendet wird (`GET` statt `POST`), ergibt sich ja fast schon aus der Meldung in der Browser-Konsole. Aber dass zwei weitere Fehler (der fehlende `Content-Type` und das fehlende Konvertieren der Antwort von Text in ein JavaScript-Objekt mit `response.json()`) auch noch gleich aufgelöst wurden, hatten wir nicht erwartet.

Jam

Bei unserer Suche nach Werkzeugen mit KI-Unterstützung zum Debugging sind wir auf die Browser-Erweiterung *Jam* gestoßen. Mithilfe dieser Erweiterung können Sie Fehler, die Ihnen beim Verwenden einer Webanwendung auffallen, sehr einfach dokumentieren. Sie können damit Screenshots oder Screencasts (also ein Video der Bildschirmaufzeichnung) erstellen und teilen. Außerdem, und darin liegt die große Stärke der Erweiterung, werden viele Zusatzinformationen (wie die Ausgabe der Browser-Konsole oder die ausgeführten Netzwerkanfragen) aufgezeichnet und ebenfalls gespeichert.

Alle Informationen werden auf die Plattform *https://jam.dev* geladen, wo jeder Fall, oder *Jam*, angesehen und kommentiert werden kann. Der Service ist nach einer Registrierung frei verfügbar, natürlich gibt es Bezahlzugänge für Business- und Enterprise-Kunden. Diese bieten unter anderem Anschluss an ein bestehendes Ticket-System und eine Anmeldung über das Firmennetzwerk.

Jetzt können Sie zu Recht fragen, was das mit künstlicher Intelligenz zu tun hat. Als wir die erste Auflage dieses Buchs erstellten, hatte Jam eine direkte Verbindung zu einem Sprachmodell, an das Jam die Fehlermeldungen und weitere Metadaten schickte und so oft zu einer guten Fehleranalyse kam. Die Verwendung war sehr komfortabel, die Kosten für die Verwendung des LLMs blieben wohl bei Jam hängen.

Inzwischen hat sich die KI-Strategie von Jam etwas geändert: Die aktuelle Version von Jam stellt einen MCP-Server bereit (lesen Sie mehr zu MCP in Kapitel 13, »MCP und Skills«), über den sich ein Sprachmodell Informationen zu dem Vorfall von der Jam-Plattform holen kann (siehe Abbildung 4.3).

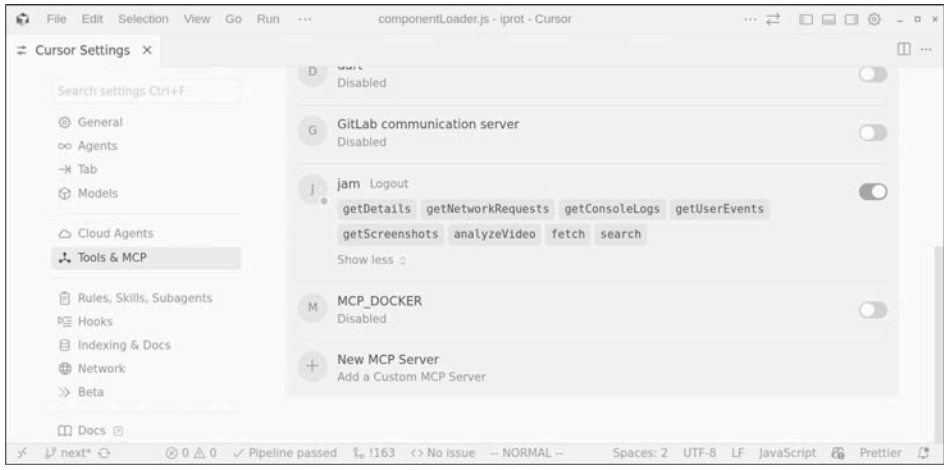

Abbildung 4.3 Der aktivierte Jam-MCP Server mit den vorhandenen Werkzeugen wie `getDetails` oder `getConsoleLogs` im KI-Editor Cursor

Wir haben den Workflow mit einem Fehler an einem unserer Software-Projekte durchgespielt. Der Bug sieht folgendermaßen aus: Bei einer Aktion im Frontend öffnet sich zwar der gewünschte Dialog, dieser wird aber nicht korrekt ausgefüllt. Eine weitere Fehlermeldung gibt es nicht. Mit der Screenrecording-Funktion von Jam zeichnen wir ein 8 Sekunden langes Video von dem Fehler auf und laden den Vorfall auf die Jam-Plattform. Beim Anlegen des neuen *Jams* bekommen wir Unterstützung von einem KI-Modell, das unser Video analysiert und in vier Schritten beschreibt. Wir ergänzen diese Beschreibung noch mit dem Fehlen der Dialoginhalte. Damit ist der Jam abgeschlossen.

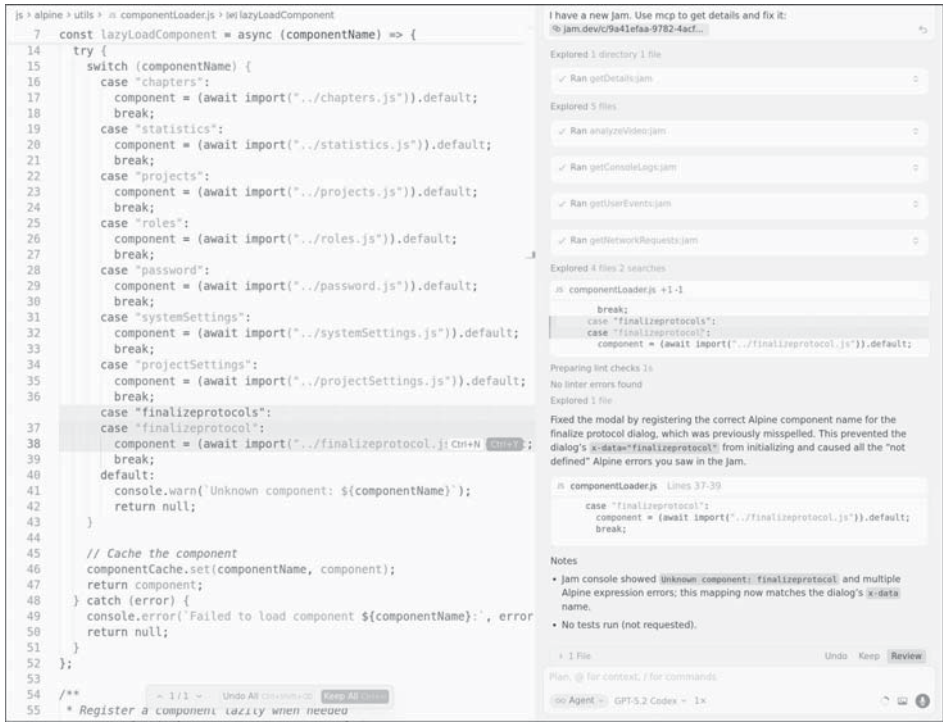

Abbildung 4.4 Die erfolgreiche Fehlerbehebung durch Cursor mithilfe der MCP-Schnittstelle zu Jam

Wir wenden uns jetzt unserer Entwicklungsumgebung Cursor-AI zu und geben folgenden Prompt ein:

> **Prompt:** I have a new jam. Use mcp to get details and fix it:
>
> *https://jam.dev/c/9a41efaa-9782-4acf-bf0e-b2c70376d98f*

Cursor beginnt nun, die vom MCP-Server bereitgestellten Funktionen aufzurufen, und holt sich die Details zu dem Vorfall. Die Analyse der Fehlermeldungen in der Browser-Konsole bringt Cursor schnell auf die richtige Spur, und nach wenigen Minuten ist der Fehler behoben (siehe Abbildung 4.4):

> Fixed the modal by registering the correct Alpine component name for the finalize protocol dialog, which was previously misspelled. This prevented the dialog's `x-data="finalizeprotocol"` from initializing and caused all the »not defined« Alpine errors you saw in the Jam.

Jam schafft es in der Tat, das Erstellen eines aussagekräftigen Bug-Reports für ein Problem bei einer Webapplikation sehr niederschwellig zu gestalten. Ob Sie Ihre Kunden davon überzeugen können, für einen einmaligen Bug-Report ein Plug-in zu installieren, das (zwangsläufig) weitreichenden Zugriff auf den Browser hat, ist fraglich. Wenn Sie ein Team von Testern haben, das Ihre Applikation regelmäßig überprüft, könnte Jam ein hilfreiches Werkzeug sein, um Bugs gut zu dokumentieren und mithilfe von KI schnell zu korrigieren.

4.2 App-Entwicklung

Die folgenden Debugging-Beispiele beziehen sich auf die Flutter-Bibliothek. Damit können Sie Apps für iOS und Android sowie Webapplikationen erstellen.

Flutter und Null-Safe-Werte

Die mobile App, die wir mit der Flutter-Bibliothek erstellt haben, sollte mehrsprachig werden. Flutter als modernes Toolkit bringt die Werkzeuge dafür natürlich mit, aber wir kamen doch zu einem kleinen Problem. Die Anwendung wollte nicht mehr kompilieren, weil die Übersetzungseinträge nicht *Null-Safe* sind. Grob gesagt kann der Compiler zur Compilezeit nicht sicherstellen, dass das Übersetzungsobjekt zur Laufzeit auch zur Verfügung steht.

Die ausführlichen Fehlermeldungen des Dart-Compilers geben eigentlich schon den deutlichen Hinweis, was zu tun ist, wir wollten uns das Problem aber auch von der KI lösen lassen:

Prompt: flutter: My app is not compiling after I inserted translations for labels.

```
ERROR: lib/widgets/team.dart:43:62: Error: Property 'warning'
  cannot be accessed on 'AppLocalizations?' because it is
  potentially null.
  - 'AppLocalizations' is from
  'package:flutter_gen/gen_l10n/app_localizations.dart'
```

```
('.dart_tool/flutter_gen/gen_l10n/app_localizations.dart').
Try accessing using ?. instead.
title: Text(AppLocalizations.of(context).warning),
                                        ^^^^^^^
```

> The error occurs because AppLocalizations.of(context) can return null if the localization data isn't properly initialized. To fix this, you need to handle the potential null case. Here's how to resolve it:
>
> **Solution 1: Use the null-aware operator (?.)**
>
> ```
> title: Text(AppLocalizations.of(context)?.warning ?? 'Warning')
> ```
>
> ...
>
> **Best Practice:**
>
> The most robust solution is to use the null-aware operator (`?.`) with a fallback value, as it handles cases where localization might not be available.

Wie schon aus der Fehlermeldung hervorgeht, soll der *null-aware operator*, also das Fragezeichen, für die `AppLocalizations` verwendet werden. Die KI geht aber noch einen Schritt weiter und hängt den Ersatz-Text `Warning` an (das doppelte Fragezeichen bietet das Fallback für den `Null`-Wert in Flutter/Dart). Ohne diese Änderung würde die App nämlich immer noch nicht kompilieren, weil die umgebende `Text()`-Funktion unbedingt einen String erwartet und eben nicht `Null`.

Wir haben die Problemstellung an unterschiedliche Sprachmodelle gesendet. Oben abgedruckt sehen Sie die Antwort von *Codestral* der französischen Firma *Mistral*. *Claude Opus 4.5*, das zum Zeitpunkt des Tests stärkste Modell von Anthropic, konnte noch eine interessante Erweiterung hinzufügen:

> **Option 3 (Recommended): Create an extension for convenience**
>
> Add this to a utility file:
>
> ```
> extension AppLocalizationsX on BuildContext {
>   AppLocalizations get l10n => AppLocalizations.of(this)!;
> }
> ```
>
> Then use it like:
>
> ```
> title: Text(context.l10n.warning),
> ```

Da wir nicht täglich mit Flutter/Dart programmieren, war uns die Möglichkeit der *Extension* gar nicht bewusst. Um den Code nicht ohne weiteres Verständnis zu kopieren, ließen wir uns die *Option 3* noch ausführlicher erklären, wodurch sich uns auch

der ungewöhnliche Name `AppLocalizationsX` erschloss und wir auf die mögliche Codevervollständigung in der IDE hingewiesen wurden.

Als kritische Software-Entwickler wollten wir noch wissen, ob diese Methode Nachteile mit sich bringt. Auch hier zeigte sich *Claude Opus* auskunftsfreudig und wies vor allem auf die erhöhte Komplexität durch zusätzliche `import`-Anweisungen hin.

Unser Fazit: Sogar bei einem so trivialen Fehler konnte uns die KI positiv überraschen, und wir haben wieder etwas über Flutter/Dart dazugelernt.

Layouts in Flutter

Bei der App-Entwicklung mit der Flutter-Bibliothek muss man in der Web-Entwicklung etwas umdenken. Werden Bereiche nicht ganz von Widgets ausgefüllt oder überschreiten sie den vorhandenen Platz, so füllt Flutter die Fläche mit einem schwarzgelb gestreiften Hintergrund, um auf das Problem hinzuweisen (siehe Abbildung 4.5). Das hilft uns zwar, das Problem zu erkennen, aber leider nicht, es zu lösen. Wir wollen sehen, ob uns die KI auch hier aus der Patsche helfen kann:

Abbildung 4.5 Die Flutter-Anzeige für ein fehlerhaftes Layout am unteren Rand

Prompt: flutter: I get a yellow-black striped area at the bottom of my app. The widget I am using looks like this:

```
return Scaffold(
  drawer: const MyDrawer(),
  body: Column(
    children: <Widget>[
```

```
      SizedBox(
        width: MediaQuery.of(context).size.width,
        height: 250,
        child:
            Image.file(File(state.currentEntry.image),
              fit: BoxFit.cover),
      ),
      const TagsWidget(),
      MarkdownPreview(
        entry: state.currentEntry,
        picturePath: state.picturePath,
      ),
    ],
  ),
  appBar: AppBar(
      title: Text(state.currentTrip.title),
      actions: action.getActions(context)),
);
```

> The yellow-black striped area indicates an overflow error in Flutter. This happens when your Column children exceed the available screen height. The issue is that your Column isn't scrollable, but contains:
>
> - A fixed 250px image
> - TagsWidget
> - MarkdownPreview
>
> These widgets together are taller than the screen, causing overflow.
>
> **Solution:** Wrap your `Column` in a `SingleChildScrollView`:
>
> ...

Claude 4.5 Sonnet erklärt das Problem korrekt und hat eine pragmatische Lösung zur Hand: Durch den `SingleChildScrollView` wird der gesamte Bildschirm verschiebbar, und man kann nach unten scrollen.

In der ersten Auflage des Buchs verwendeten wir *Claude 3.5 Sonnet*, das weniger klar in seinen Ausführungen war: Wir bekamen verschiedene Vorschläge, wie das Problem zu lösen sei, ohne einen Weg als den geeignetsten zu bezeichnen. Wir mussten damals nachfragen, was denn nun der beste Weg sei, bevor wir die `SingleChildScrollView`-Lösung empfohlen bekamen.

Wir geben Claude aber noch eine härtere Nuss zu knacken: Dieses Mal laden wir einen Screenshot der fehlerhaften App in einen neuen Chat und fragen Claude, was zu tun

ist (siehe Abbildung 4.6). Im Prompt geben wir keine weiteren Informationen zu dem Fehler:

> **Prompt:** My flutter app shows an error at the bottom. What can I do?

Aus der Antwort erkennen wir, dass der Text `BOTTOM OVERFLOWED BY...` im Bild erkannt und auch inhaltlich interpretiert wurde:

> The error message at the bottom indicates a layout overflow issue in Flutter: "BOTTOM OVERFLOWED BY 1.7976931348623157e+308 PIXELS". That extremely large number (essentially infinity) suggests your layout constraints are severely misconfigured.
>
> ...

Obwohl Claude den Quellcode nicht gesehen hat, wird abermals die Lösung mit einem `SingleChildScrollView` vorgeschlagen. Außerdem erhalten wir noch die Information, dass es bei der Angabe des Quellcodes eine genauere Problemlösung geben wird:

> If you can share the relevant code section where this image is displayed (particularly the widget tree around this image), I can provide a more specific fix for your situation.

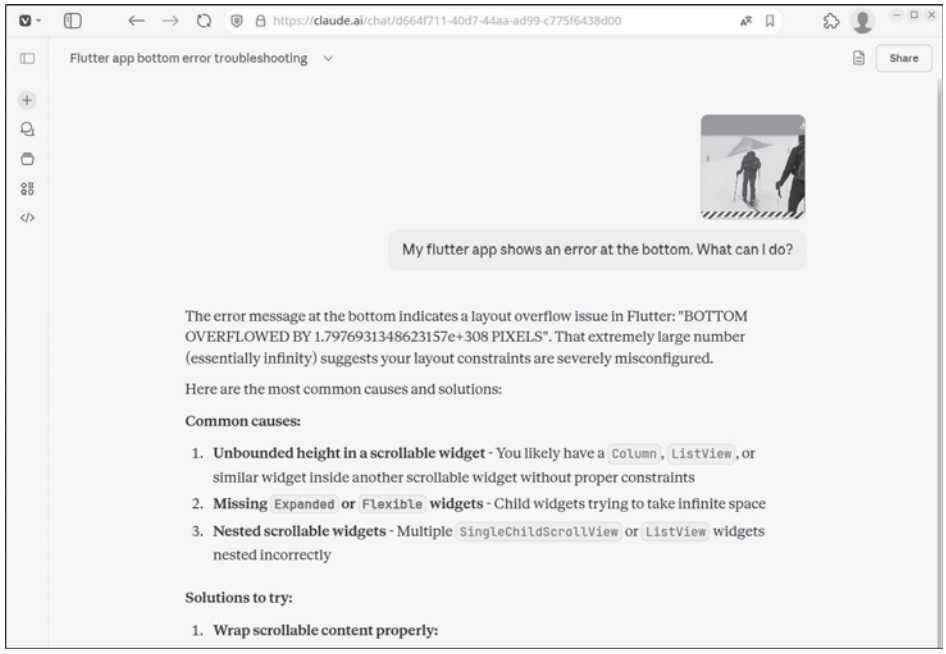

Abbildung 4.6 Die Flutter-Anzeige für ein fehlerhaftes Layout am unteren Rand

Fehlererkennung anhand von Bildern oder Videos ist noch eine recht junge Funktion, bietet aber ein großes Potenzial. Da wir das Bild nur im Web-Chat hochgeladen haben, in dem die KI keinen Zugriff auf den Quellcode hat, erhalten wir nur eine Vermutung zu der Fehlerursache. In einer IDE wie Cursor kann der Software-Agent den Fehler im Code finden und ausbessern.

4.3 Entwicklung am Raspberry Pi

Bei der Software-Entwicklung auf *Embedded Devices* wie dem Raspberry Pi läuft die Entwicklungsumgebung in der Regel nicht auf dem Gerät selbst, sondern auf dem Arbeitsplatzrechner der Entwicklerin. Dadurch ergeben sich leicht unterschiedliche Anforderungen beim Debugging. Um den Debugger mit dem Gerät zu verbinden, muss eine Netzwerkverbindung mit den entsprechenden Port-Weiterleitungen hergestellt werden. Läuft der Raspberry Pi produktiv dann ohne Internetzugang, ist ein gutes Logging der Applikation wichtig, um später Fehlern auf die Spur zu kommen.

In den folgenden beiden Abschnitten wollen wir auf diese beiden Themen eingehen.

Step-by-step-Debugging auf einem Raspberry Pi

Beim *klassischen* Debugging setzt man *Breakpoints* im Code und führt das Programm mit einem Debugger aus. Sobald die Stelle mit dem Breakpoint erreicht wird, wird die Ausführung angehalten, und man hat die Möglichkeit, den Status der Applikation zu diesem Zeitpunkt zu überprüfen. Außerdem kann man dann schrittweise, also *step-by-step*, jeden Aufruf verfolgen und den Inhalt von Variablen untersuchen.

Wir haben keine gute Anwendung für KI bei dieser Tätigkeit gefunden. Es bleibt Handarbeit, die Breakpoints zu setzen und den Applikationsstatus zu überprüfen. Wobei uns die KI aber schon unterstützen kann, ist das Setup, damit wir Step-by-step-Debugging überhaupt erst beginnen können. Wir haben eine Python-Anwendung auf einem Raspberry Pi entwickelt, die einen Schrittmotor steuert und über ein integriertes Touch-Display gesteuert wird.

Prompt: What tools do I need to step-by-step debug my Python app on a remote Raspberry Pi?

> I can help you with that! Debugging a Python application on a remote Raspberry Pi typically involves a combination of tools for remote connection, code editing, and debugging. Here's what you need:

- **Remote Connection Tools**

  SSH (Secure Shell)

  – Built-in SSH client (Linux/macOS) or PuTTY (Windows)
  – Enables command-line access to your Raspberry Pi
  – Required for most remote debugging setups

  VS Code Remote - SSH Extension

  – Lets you edit and debug code on the Pi as if it were local
  – Seamless integration with VS Code's debugger

- **IDE/Editor with Remote Debugging Support**

  Option A: VS Code (Recommended)

  ...

This setup gives you full step-by-step debugging with breakpoints, variable inspection, call stacks, and watch expressions — all running on your remote Pi!

Claude Opus 4.5 klärt uns zuerst über die notwendigen Werkzeuge auf und zeigt uns anschließend eine Schritt-für-Schritt-Anleitung für den gängigen Editor VS Code. Mithilfe des VS-Code-*Remote-SSH-Plug-ins* wird das Debugging zum Kinderspiel.

Nun ist VS Code wirklich ein großartiger Editor, einer der Autoren verwendet aber sehr intensiv den rein textbasierten Editor *vi*. Wir wollten von Claude wissen, ob es auch für vi, beziehungsweise die moderne Variante *neovim*, eine Remote-Debugging-Lösung gibt:

Prompt: Can I do remote debugging in neovim?

Yes, absolutely! You can do remote debugging in Neovim. Here's what you need:

Core Tools for Neovim Remote Debugging

1. nvim-dap (Debug Adapter Protocol client)

...

Alternative: Edit Directly on Pi via SSH

You can also SSH into the Pi and run Neovim directly there:

...

Auch hier bekommen wir eine ausführliche und funktionierende Anleitung, inklusive der Einstellung der bekannten Funktionstasten-Zuweisungen (F5, F10, F11, F12). Der Hinweis, dass es mit Neovim auch möglich ist, direkt auf dem Raspberry Pi zu arbeiten, ist gut gemeint, Neovim-Benutzern dürfte das aber ohnehin klar sein.

Hilfe beim Erstellen von Log-Einträgen

Beim Debugging von Geräten, die nicht vor Ort und womöglich auch noch offline sind, ist es eine gute Praxis, wichtige Meldungen in eine Logdatei zu schreiben. So kann man auch zu einem späteren Zeitpunkt Fehler entdecken. Wir arbeiteten an einem Projekt, bei dem man das WLAN von einer Cloud-Plattform aus auf Raspberry Pis konfigurieren kann, die über das Mobilfunknetz verbunden sind. Eine Python-Anwendung auf dem Raspberry Pi macht die entsprechenden Systemaufrufe, um die Einstellungen permanent zu speichern.

Wir versuchten, etwas Unterstützung beim Erstellen der Logging-Informationen von Mistral zu erhalten:

Prompt: Help me debug this function with some logging information.

```
def set_wifi_wpa_cli(ssid, psk):
  if ssid == '' or psk == '':
    return
  basecmd = ['sudo', 'wpa_cli', '-iwlan0']
  cmd = basecmd + ['list_networks']
  res = subprocess.run(cmd, capture_output=True)
  out_str = res.stdout.decode('utf-8')
  if re.search(rf"\t{ssid}\t", out_str, re.MULTILINE):
    # network already there
  else:
    cmd = basecmd + ['add_network']
    res = subprocess.run(cmd, capture_output=True)
    netnr = res.stdout.decode('utf-8')
    cmd = basecmd + ['set_network', netnr, 'ssid',
                     f'"{ssid}"']
    res = subprocess.run(cmd, capture_output=True)
    cmd = basecmd + ['set_network', netnr, 'psk', f'"{psk}"']
    res = subprocess.run(cmd, capture_output=True)
    ...
```

To debug this function, we can add logging statements at key points to track its execution flow, input values, and intermediate results. Below is the updated function with detailed logging:

```
# Configure logging
logging.basicConfig(
  level=logging.DEBUG,
  format='%(asctime)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)
...
```

```
def set_wifi_wpa_cli(ssid, psk):
  logger.info(
    f"Starting set_wifi_wpa_cli with SSID: '{ssid}' and "
    f"PSK: '{'*' * len(psk) if psk else ''}'"
  )
if ssid == '' or psk == '':
  logger.warning("Empty SSID or PSK provided. Exiting.")
  return
...
if re.search(rf"\t{ssid}\t", out_str, re.MULTILINE):
  logger.info(f"Network with SSID '{ssid}' already exists. S...
else:
  logger.info(f"Network with SSID '{ssid}' not found. Adding...
  cmd = basecmd + ['add_network']
  logger.debug(f"Running command: {cmd}")
  res = subprocess.run(cmd, capture_output=True)
  netnr = res.stdout.decode('utf-8').strip()
  logger.debug(f"Added network with ID: {netnr}")
...
```

Wie Sie in den Ausschnitten der Antwort sehen können (wir haben die Ausgabe hier stark gekürzt), leistet *Mistral Large 2512* ganze Arbeit. Zuerst wird das Logging für Python konfiguriert, in einem weiteren Schritt werden die Log-Ausgaben nach ihrer Wichtigkeit gewertet und entsprechend mit `info`, `warning` oder `debug` gekennzeichnet. Das macht es für uns einfacher, die Logdatei später zu filtern. Bei den Systemaufrufen (`subprocess.run`) werden sowohl das Kommando als auch das Ergebnis geloggt. Außerdem verwendet die KI die relativ moderne Python-Schreibweise für Zeichenketten mit *F-Strings*.

Was uns besonders gut gefällt, ist, dass Mistral den WLAN-Schlüssel (hier *psk* für *Pre Shared Key*) nicht ins Logfile schreibt, sondern es mit Sternchen ersetzt. Der Logging-Aufruf gleich nach dem Beginn der Funktion `set_wifi_wpa_cli` war ursprünglich in einer Zeile formuliert. Da überlange Zeilen im Buchdruck kein schönes Bild ergeben, fragen wir das LLM nach einer mehrzeiligen Version dieses Logging-Aufrufs:

Prompt: Line 13, starting with `logger.info(f"Starting set_wifi_wpa_cli...` is very long. How can I break it in two lines?

> You can break a long f-string into multiple lines in Python using parentheses or by concatenating strings. Here are a few ways to do it for your logging line:
>
> **Option 1: Parentheses (Recommended)**
>
> Wrap the f-string in parentheses and break it into multiple lines: ...

Wir bekommen gleich vier mögliche Varianten, um die Zeile aufzuteilen, werden aber darauf hingewiesen, dass die empfohlene Option 1 die sauberste und am besten lesbare ist. Bei der ebenfalls sehr lesbaren Version mit Multi-Line-F-Strings (hier nicht abgedruckt) weist die KI darauf hin, dass Python dafür mindestens in der Version 3.12 vorliegen muss.

Unser Fazit für diesen Einsatz von KI: Auch wenn es keiner allzu großen Intelligenz bedurfte, um die Log-Einträge hinzuzufügen, waren wir wieder positiv überrascht. Nicht nur, dass wir diese wenig interessante Arbeit an das LLM auslagern konnten, die KI gibt auch acht, keine Sicherheitslücken durch Passwörter in Logfiles entstehen zu lassen.

4.4 Visual Studio und VS Code

Die nächsten beiden Abschnitte beschäftigen sich mit zwei wichtigen Entwicklungsumgebungen (*Integrated Development Environment*, kurz einfach IDE) von Microsoft: Visual Studio und VS Code. Ersteres kann getrost als Urgestein in der IDE-Geschichte bezeichnet werden. Bereits 1997 kam die erste Version für Windows auf den Markt. Die proprietäre Software war bis 2014 nur gegen Bezahlung zu haben, seitdem gibt es eine *Community Edition*, die nach einer Registrierung bei Microsoft kostenlos verwendet werden kann. Die IDE ist sehr beliebt, weil sie eine kontextbezogene Hilfe und eine Syntaxprüfung für viele unterstützte Programmiersprachen bietet. Das Programm verlangt einiges an Ressourcen von Ihrem Computer, weshalb viele Entwickler nach schlankeren Alternativen Ausschau hielten.

Seit 2015 entwickelte Microsoft den Editor *VS Code* auf der Plattform *Electron*, die ursprünglich als Basis für den freien Editor *Atom* erstellt wurde. Electron stellt dabei eine Laufzeitumgebung für Webapplikationen zur Verfügung, die Desktop-Anwendungen ohne die Einschränkungen der Browser-Sandbox ermöglicht. Der Kern von VS Code ist schlank und kann mit unzähligen Extensions erweitert werden. In Umfragen auf Stack Overflow liegt VS Code seit Jahren an der Spitze der beliebten IDEs von Entwicklern.

Fehlende Python-Module in VS Code

Die Python-Anwendung im folgenden Abschnitt war schon ein paar Jahre alt, als wir uns wieder damit beschäftigen mussten. Beim Aufruf des Debuggers trat eine Exception auf, da ein eingebundenes Modul nicht gefunden werden konnte. Leider wurde die Anwendung damals ohne ein Python Virtual Environment und auch ohne eine `requirements.txt` erstellt.

Die KI-Unterstützung in VS Code auf Basis von GitHub Copilot und der entsprechenden Extension konnte uns aber auch hier etwas Arbeit abnehmen. Durch die

Exception wird die Ausführung angehalten und der Cursor an der entsprechenden Stelle im Code positioniert. Die Sprachunterstützung für Python in VS Code (abermals eine Extension) kennzeichnet bereits den nicht vorhandenen Import. Mit Copilot können wir nun auf Knopfdruck eine Lösung vorschlagen lassen (`/fix`) und, sofern der *Agent-Mode* aktiv ist, das entsprechende Paket gleich installieren (siehe Abbildung 4.7). In unserem Fall werden wir von Copilot zuvor darauf hingewiesen, dass diese Python-Anwendung noch keine virtuelle Umgebung enthält. Da diese virtuellen Umgebungen viel Ärger mit Paketen ersparen, lassen wir Copilot gerne ein `venv` erstellen.

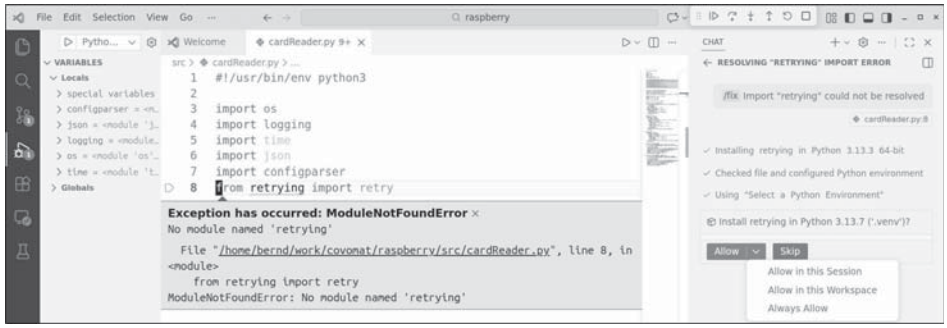

Abbildung 4.7 GitHub Copilot kann im *Agent-Mode* Kommandos auf Ihrem Computer ausführen, in diesem Fall ist das `pip install`.

Der eben angesprochene *Agent-Mode* ist ein wesentliches Element für ein effizientes Debugging. Wie wir in Kapitel 13, »MCP und Skills« noch zeigen werden, können moderne KI-Tools auf ausgewählte Werkzeuge zugreifen und *wissen* auch, wann und wie diese einzusetzen sind. Dass hier das Python-spezifische `pip install` verwendet wird, ist nur ein Beispiel dafür.

Visual Studio Exception Debugging

Für Anwender von Microsofts Entwicklungsumgebung Visual Studio (nicht das freie Visual Studio Code), die zudem auch Kunden von GitHub Copilot sind, gibt es eine komfortable KI-Erweiterung: Wenn Sie Ihre Anwendung im Debugging-Modus starten und ein unerwarteter Fehler (eine *Exception*) auftritt, dann bietet Visual Studio auf Knopfdruck die Möglichkeit, das Problem von Copilot untersuchen zu lassen (*Analyze with Copilot*).

Unsere Experimente mit dieser Funktion hinterließen einen positiven Eindruck. Im Prompt wird offensichtlich nicht nur der Text der Exception gesendet, sondern auch der umliegende Code. Nur dadurch konnten wir uns erklären, dass die (absichtlich herbeigeführte) `System.ArgumentException` so exakt erklärt werden konnte. Der C#-Aufruf `items.GetRange(0, 3)` extrahiert die ersten drei Elemente aus der Liste `items`.

Bei unserem Aufruf war die Liste aber leer, weshalb die Exception entstand (siehe Abbildung 4.8).

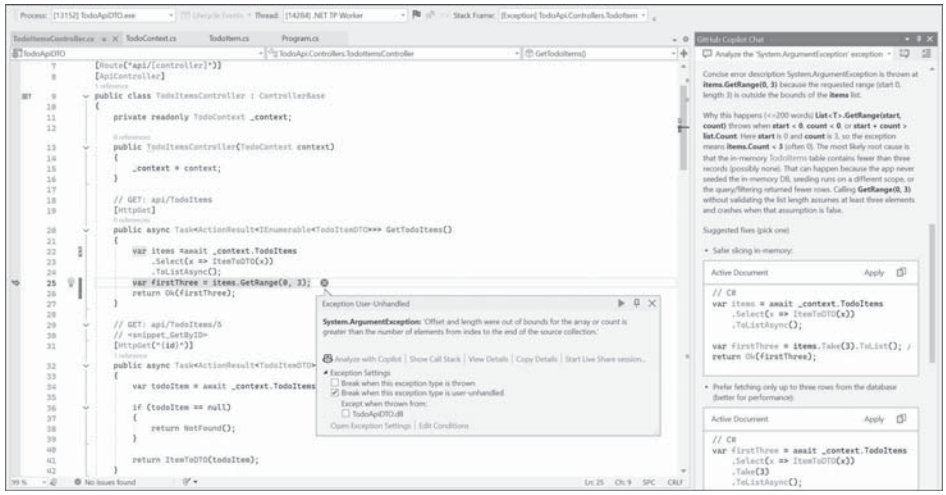

Abbildung 4.8 Die Unterstützung von GitHub Copilot beim Debugging in Visual Studio

GitHub Copilot konnte das Problem sehr ausführlich beschreiben und schlug gleich drei mögliche Verbesserungen vor, wodurch die Exception verhindert werden soll. Bei unseren Versuchen zur ersten Auflage dieses Buchs waren wir noch begeistert von der Funktionsweise und den Codevorschlägen. Aber die Konkurrenz schläft nicht: Durch die Arbeit mit der Cursor-IDE oder verschiedenen Visual Studio Code KI-Plug-ins ist man eine derartige Funktion heute gewohnt beziehungsweise erwartet man, dass die IDE die Codeänderung selbstständig durchführt. Visual Studio wirkt hier fast schon nicht mehr ganz *up to date*.

4.5 Fazit

Wenn wir uns abschließend zu diesem Kapitel die Frage stellen, ob KI-Unterstützung hilfreich beim Debugging sein kann, so können wir das für uns zweifellos mit *ja* beantworten. Wir sehen hier zwar nicht einen so großen Mehrwert wie bei der Codevervollständigung, aber schnelle Hilfe für weitverbreitete Probleme, wie zum Beispiel die CORS-Problematik in Abschnitt 4.1, »Web-Applikationen«, ist ein Gewinn.

Der Bereich, in dem KI-Unterstützung wieder einmal ihre Stärke ausspielen kann, ist bei Programmiersprachen, die man selbst nicht täglich verwendet oder gerade erst lernt. Hier können kleine Fehler schon zu mehrstündigen Internet-Recherchen bei Stack Overflow und anderen Foren führen, während die KI oft nicht nur das Problem

prägnant beschreiben kann, sondern auch gleich noch den korrigierten Quellcode erzeugt.

Nicht zu unterschätzen sind die erweiterten Möglichkeiten durch *Agentic AI*: Während die KI die Schleife aus *Code Ändern - Kompilieren/Ausführen - Ausgabe/Fehler analysieren* in hoher Geschwindigkeit wiederholt, kann sich der Mensch um andere Dinge kümmern. Die Verwendung von Werkzeugen und die Kombination von Sprachmodellen mit einem großen Kontextfenster machen diesen Workflow so wertvoll.