

## 3 Software entwerfen mit ChatGPT

### Kapitelübersicht

- Prototypen für potenzielle Entwürfe mit ChatGPT entwickeln
- Die Softwarearchitektur in Mermaid dokumentieren
- Unseren Entwurf mit ChatGPT fertigstellen

Nachdem wir verstehen, wie generative KI funktioniert und wie wir sie nutzen können, beginnen wir, die Architektur unserer Applikation zu entwerfen, zu erforschen und zu dokumentieren. Das Festlegen der wichtigsten Komponenten im Voraus hat gleich mehrere Vorteile. So können wir bestimmte Teile des Entwurfs an untergeordnete Softwarearchitekten und bestimmte Bereiche der Entwicklung an andere Teammitglieder delegieren. Diese Planung im Vorfeld hilft uns außerdem beim Nachdenken über die Implementierung, wodurch wir mögliche Fallstricke rechtzeitig erkennen und vermeiden können. Das Abbilden unseres Entwurfs als Dokumentation ermöglicht uns schließlich, unsere Kernentscheidungen zu begründen und unsere Absicht gegenüber unserem zukünftigen Selbst, unseren Entscheidungsträgern und denjenigen, die das Projekt später weiterpflegen, mitzuteilen.

Zu Beginn wollen wir uns einen Überblick über die Applikation verschaffen, mit deren Entwurf wir in diesem Kapitel beginnen: einem System zur Verwaltung von IT-Assets<sup>1</sup> (*Information Technology Asset Management*, kurz: ITAM). Die wichtigsten Features werden wir in den folgenden Kapiteln erstellen.

---

1. Anmerkung des Übersetzers: In diesem Kontext entspricht das Wort »Asset« in erster Linie dem deutschen Begriff »Wirtschaftsgut«. Wir haben uns hier entschieden, aus Gründen der Konsistenz den englischen Begriff beizubehalten.

### **3.1 Vorstellung unseres Information Technology Asset Management-Systems**

Ein ITAM-System dient der Verwaltung und Kontrolle von Hardware, Softwarelizenzen und anderen IT-bezogenen Ressourcen über ihre gesamten Lebensdauer hinweg. ITAM-Systeme bestehen üblicherweise aus Werkzeugen zur Hard- und Software-Inventarisierung, Software zur Lizenzverwaltung und anderen verwandten Programmen. Zum System können auch manuelle Prozesse und die physische Kontrolle von IT-Werten anhand von QR-Codes, Barcodes und anderen Technologien zur physischen Asset-Verwaltung gehören.

Üblicherweise verwenden ITAM-Systeme eine zentrale Datenbank, in der die Asset-ID und spezifischen Attribute für den jeweiligen Asset-Typ gespeichert werden. So könnten Sie für Desktop-PCs beispielsweise den Gerätetyp, die Modellnummer, das Betriebssystem und die installierten Programme speichern. Für Softwareprodukte könnten dies der Name des Programms, der Händler, die Anzahl der verfügbaren Lizenzen und die Computer, auf denen die Software installiert wurde, sein. Der letzte Punkt hilft, sicherzustellen, dass Ihr Unternehmen sich an Lizenzvereinbarungen hält. Durch die Überwachung der Verwendung sollte die Anzahl der gekauften Lizenzen nie überschritten werden.

Außerdem helfen ITAM-Systeme auch bei der Kostenkontrolle. Da Sie immer wissen, welche Hard- und Software verfügbar ist, können unnötige Anschaffungen vermieden werden. Diese Systeme zentralisieren die Einkäufe, was Großbestellungen erleichtert. Nicht genutzte Hardware kann verkauft und die Auslastung von wenig verwendeten Geräten konsolidiert werden. Wie Sie sehen werden, können Sie Informationen zum Kaufdatum nutzen, um den Wertverlust der Hardware zu berechnen und die Ergebnisse in die Steuerabschreibungen Ihres Unternehmens einzubeziehen.

Später werden wir weitere Eigenschaften eines ITAM-Systems erforschen, indem wir ein solides konzeptionelles Modell der Applikation erstellen. Dabei konzentrieren wir uns auf ein System, das Hardwarekomponenten »im Auge behalten« und verwalten kann. Im nächsten Schritt weisen wir ChatGPT an, die Rolle eines Softwarearchitekten einzunehmen, um seine Meinung über das bestmögliche Design dieses Projekts einzuholen und zu sehen, welche Lösung es vorschlägt.

### **3.2 ChatGPT bitten, uns beim Entwurf unseres Systems zu helfen**

Wir beginnen mit einem Prompt in einer neuen Session, in dem wir unsere Anforderungen skizzieren. Wie im vorigen Kapitel nutzen wir das Refinement-Muster auf die Prompts an, um das Design unseres Systems nach und nach zu verbessern. Wir beginnen allgemein und gehen im weiteren Verlauf immer genauer auf An-

forderungen und Funktionalität ein. Zu Beginn weisen wir ChatGPT mit folgendem Prompt an, unsere Prompts zu verbessern:

*From now on, when I give you a prompt, output a better prompt.*

*Deutsch: Wenn ich Dir von nun an einen Prompt eingebe, gibst Du einen besseren Prompt aus.*

Dieser Prompt hilft bei der Erforschung Ihrer aktuellen Aufgabe, weil er ChatGPT anweist, bessere Prompts für Sie zu erstellen, denn Sie wissen: je besser der Prompt, desto besser die Ausgaben. Wir werden in Zukunft folgendermaßen vorgehen: Wenn Sie eine spezifische Antwort benötigen, stellen Sie eine spezifische Frage. Zusätzlich und alternativ dazu können Sie ChatGPT anweisen, das Problem Schritt für Schritt zu durchdenken und seine Überlegungen zu begründen.

Als Nächstes bitten wir ChatGPT mit folgendem (erweiterten) Prompt, uns bei unserem Entwurf zu helfen:

*Assume that you are a software architect, and you are designing an information technology asset management system, using Python. The system should be able to maintain a complete inventory of all hardware and software assets owned by an organization. The system should be able to track asset location, status, and usage. This includes information such as who uses the asset, when it was last serviced, and when it is due for replacement. The system should be able to manage the entire lifecycle of an asset, from procurement to disposal. This includes tracking warranty and lease information, managing asset maintenance and repair, and ensuring compliance with regulations. The system should be able to allocate assets to employees or departments, track usage, and manage reservations for assets that are in high demand. It should use hexagonal architecture. Remember to enhance my prompt.*

*Deutsch: Stelle Dir vor, Du bist Softwarearchitekt und entwirfst ein System zur Verwaltung von IT-Assets mit Python. Das System soll in der Lage sein, eine vollständige Inventarliste aller Hardware- und Software-Assets eines Unternehmens zu führen. Das System sollte in der Lage sein, den Standort, den Status und die Nutzung der Assets zu verfolgen. Dazu gehören Informationen darüber, wer das Asset nutzt, wann es zuletzt gewartet wurde und wann es ersetzt werden muss. Das System sollte in der Lage sein, die gesamte Lebensdauer eines Assets zu verwalten, von der Beschaffung bis zur Entsorgung. Dazu gehören die Nachverfolgung von Garantie- und Leasinginformationen, die Verwaltung der Wartung und Reparatur von Assets und die Einhaltung von Vorschriften. Das System sollte in der Lage sein, Assets den Mitarbeitern oder Abteilungen zuzuordnen, die Nutzung zu verfolgen und Reservierungen für Assets zu verwalten, die stark nachgefragt werden. Es sollte eine hexagonale Architektur verwenden. Vergessen Sie nicht, meine Eingabeaufforderung zu erweitern.*

In diesem Prompt nutzen wir wieder das Persona-Muster, indem wir ChatGPT die Rolle eines Softwarearchitekten zuweisen. Dabei wird ChatGPT nicht nur die grundsätzlichen Eigenschaften unseres Systems ausgeben, sondern auch unseren Prompt verbessern. Wenn wir diesen Prompt eingeben, erhalten wir die im folgenden Listing gezeigten Ausgaben. Vergessen Sie nicht: Die Ausgaben von ChatGPT (und anderen generativen KIs) hängt stark von den zuvor gemachten Eingaben ab. Ihre Ausgaben werden sehr wahrscheinlich nicht exakt unserem Ergebnis entsprechen.

***Listing 3.1 Verbesserter Prompt und Übersicht über die Applikation, beschrieben von ChatGPT***

To enhance and detail your prompt for designing an information technology asset management system using Python with a hexagonal architecture, here's a more comprehensive breakdown:

**System Objective and Core Functionality:** Design a Python-based IT asset management system. Ensure the system maintains a full inventory of all organizational hardware and software assets.

**Detailed Asset Tracking:** Implement features for tracking the location, status, and usage of each asset. Include detailed logs for user assignments, service histories, and replacement schedules.

**Lifecycle Management:** Enable management of assets from procurement to disposal. Track warranty and lease information, schedule maintenance and repairs, and ensure regulatory compliance.

**Asset Allocation and Usage Monitoring:** Facilitate asset allocation to employees or departments. Monitor usage patterns and manage reservations for high-demand assets.

**Architectural Approach:** Adopt a hexagonal architecture to promote loose coupling and ease of maintenance. Ensure modularity in design for easy scalability and updates.

**Additional Features and Considerations:** Incorporate user-friendly interfaces for different stakeholders. Implement robust security measures to protect sensitive asset data. Consider integration capabilities with existing systems within the organization.

This detailed breakdown provides a comprehensive guide for designing the system with all necessary features and a robust architectural framework.

***Listing 3.2 Die deutsche Übersetzung von ChatGPT:***

Um Ihren Prompt zur Gestaltung eines IT-Asset-Management-Systems mit Python unter Verwendung einer hexagonalen Architektur zu

verbessern und zu detaillieren, hier eine umfassendere Ausarbeitung:

**Systemziel und Kernfunktionalität:** Entwickeln Sie ein auf Python basierendes IT-Asset-Management-System. Stellen Sie sicher, dass das System eine vollständige Inventarliste aller Hardware- und Software-Assets der Organisation führt.

**Detailliertes Asset-Tracking:** Implementieren Sie Funktionen zur Verfolgung von Standort, Status und Nutzung jedes Assets. Erfassen Sie detaillierte Protokolle über Benutzerzuweisungen, Wartungshistorien und Austauschpläne.

**Lebenszyklusverwaltung:** Ermöglichen Sie das Management von Assets vom Beschaffungsprozess bis zur Entsorgung. Verfolgen Sie Garantie- und Leasinginformationen, planen Sie Wartungen und Reparaturen und stellen Sie die Einhaltung von Vorschriften sicher.

**Asset-Zuweisung und Nutzungsüberwachung:** Ermöglichen Sie die Zuweisung von Assets an Mitarbeiter oder Abteilungen. Überwachen Sie Nutzungsmuster und verwalten Sie Reservierungen für besonders gefragte Assets.

**Architektonischer Ansatz:** Wenden Sie eine hexagonale Architektur an, um eine lose Kopplung und einfache Wartung zu fördern. Sorgen Sie für Modularität im Design, um eine einfache Skalierbarkeit und Updates zu ermöglichen.

**Zusätzliche Funktionen und Überlegungen:** Integrieren Sie benutzerfreundliche Schnittstellen für verschiedene Stakeholder. Implementieren Sie robuste Sicherheitsmaßnahmen, um sensible Asset-Daten zu schützen. Berücksichtigen Sie Integrationsmöglichkeiten mit bestehenden Systemen in der Organisation.

Diese detaillierte Ausarbeitung bietet eine umfassende Anleitung für die Gestaltung des Systems mit allen notwendigen Funktionen und einem robusten architektonischen Rahmen.

Nach etwas Bedenkzeit gibt ChatGPT die wichtigsten Grundbausteine für die Entwicklung dieses Projekts zurück. Diese Vorschläge sind hilfreich, aber eigentlich wollen wir unser Problem des sprichwörtlichen leeren Blatts Papier lösen: Beginnen wir mit dem Entwurf. Hierfür brauchen wir Textbeschreibungen und Diagramme, die uns beim Verständnis des Systems und seiner Komponenten helfen. Wir brauchen kein Kochrezept, sondern klar umrissene Handlungsanweisungen. Mit folgendem Prompt können wir ChatGPT anweisen, uns seine Arbeit zu zeigen:

Show the Python package structure for this application.

Deutsch: Zeige die Python-Paketstruktur für diese Applikation.

Nach dieser Eingabe gibt ChatGPT folgendes Paketdiagramm zurück (dieses Listing ist leicht gekürzt). Es zeigt, dass ChatGPT nicht nur Software entwerfen, sondern auch dabei helfen kann, die Struktur der physikalischen Dateien festzulegen. Über die allgemeine Paketstruktur lässt sich (vielleicht) streiten, aber immerhin haben wir nun mehr als genug Inspiration, um loszulegen.

**Listing 3.3** Python-Paketstruktur für das von ChatGPT entworfene ITAM-System

```
it_asset_management/
|   app/                                # Haupt-Applikationsverzeichnis
|   |   __init__.py
|   |   main.py                            # Einstiegspunkt der Applikation
|   |   config.py                          # Konfigurationseinstellungen
|   |   dependencies.py                  # Abhängigkeiten einfügen und verwalten
|
|   domain/                             # Domänenschicht mit Geschäftslogik
|   |   __init__.py
|   |   models/                            # Domänenmodelle (z. B. Asset, Benutzer)
|   |   |   __init__.py
|   |   |   asset.py
|   |   |   user.py
|   |   services/                          # Geschäftsdienste/-logik
|   |   |   __init__.py
|   |   |   asset_service.py
|   |   |   user_service.py
|   |   repositories/                   # Interface für Datenzugriff
|   |   |   __init__.py
|   |   |   asset_repository.py
|   |   |   user_repository.py
|
|   adapters/                           # Adapterschicht für externe Interaktionen
|   |   __init__.py
|   |   orm/                                # Objektrelationale Abbildung
|   |   |   __init__.py
|   |   |   asset_orm.py
|   |   |   user_orm.py
|   |   api/                                 # API-Endpunkte
|   |   |   __init__.py
|   |   |   asset_api.py
|   |   |   user_api.py
|   |   external_services/    # Integration mit externen Diensten
|   |   |   __init__.py
|   |   |   email_service.py
|   |   |   logging_service.py
```

```

|   └── infrastructure/           # Infrastructurschicht für Datenbank,
|       ├── __init__.py           # Messaging etc.
|       ├── database.py          # Datenbank-Einrichtung und -verbindung
|       ├── message_queue.py     # Messaging-Infrastruktur
|       └── cache.py              # Caching-Mechanismen
|
|   └── tests/                   # Test-Suite
|       ├── __init__.py           # Unit-Tests
|       ├── unit/                 # Integrationstests
|       └── integration/
|
└── requirements.txt            # Projektabhängigkeiten

```

Wir können einerseits die *Makroebene* (die »Vogelperspektive«) untersuchen, ChatGPT aber auch anweisen, uns die *Mikroebene* (die Details) zu zeigen. Wir können Prompts erstellen, mit denen ChatGPT uns nacheinander den Inhalt aller Dateien anzeigt. Diesen können wir anschließend in die entsprechenden Dateien mit den von ChatGPT vorgeschlagenen Namen übertragen. Dabei sollten wir die Ausgaben der KI grundsätzlich sorgfältig überprüfen und keinesfalls davon ausgehen, dass sie zu einhundert Prozent sicher oder akkurat sind. Es kann jederzeit vorkommen, dass der von ChatGPT erzeugte Code nicht funktioniert – ChatGPT ist darauf ausgerichtet, das nächste Text-Token zu erzeugen, anstatt, wie wir, ein vollständiges System. Wir sollten den erzeugten Code also unbedingt evaluieren, um seine Qualität und seinen Nutzen einzuschätzen zu können:

*Please show the source code for it\_asset\_management/app/schemas/asset.py.*

*Deutsch: Bitte zeige den Quellcode für it\_asset\_management/app/schemas/asset.py.*

Das folgende Listing zeigt das Ergebnis:

**Listing 3.4** Beispiel-Asset-Modell in asset\_management\_system/models/asset.py

```

from typing import Optional
from datetime import datetime
from pydantic import BaseModel

# Einfaches Asset-Modell für die Erstellung eines neuen Assets
class AssetCreate(BaseModel):
    name: str
    asset_type: str
    serial_number: str
    purchase_date: Optional[datetime] = None
    warranty_end_date: Optional[datetime] = None

    # Modell für öffentlich bereitgestellte Asset-Daten
class Asset(BaseModel):
    id: int
    name: str

```

```
asset_type: str
serial_number: str
purchase_date: Optional[datetime] = None
warranty_end_date: Optional[datetime] = None

class Config:
    orm_mode = True

# Modell für die Aktualisierung vorhandener Asset-Daten
class AssetUpdate(BaseModel):
    name: Optional[str] = None
    asset_type: Optional[str] = None
    serial_number: Optional[str] = None
    purchase_date: Optional[datetime] = None
    warranty_end_date: Optional[datetime] = None
    return '<Asset %r>' % self.name
```

Anstatt den Code klassenweise durchzugehen und die Quellcodedateien einzeln zu untersuchen, werden wir das Projekt als Ganzes dokumentieren. Auch wenn Agile und Scrum die Arbeit an Software gegenüber einer umfassenden Dokumentation bevorzugen, sollte die Rolle eines durchdachten Entwurfs und wesentlicher Dokumentation nicht unterschätzt werden. Sie versorgen den agilen Entwicklungsprozess mit Struktur, Klarheit und einer langfristigen Vision und stellen sicher, dass das Team effektiv auf Änderungen reagieren kann, während die Integrität und Qualität der Software erhalten bleibt.

### 3.3 Die Architektur dokumentieren

In diesem Abschnitt beginnen wir, ChatGPT die Dokumentation für unsere Applikation erstellen zu lassen. Wie bereits gesagt sind Entwurf und Dokumentation einer Applikation unbedingt notwendig – selbst in Agile- und Scrum-Umgebungen. Die Dokumentation gibt dem Entwicklungsteam eine klare Vision und Richtung vor, sie skizziert die Architektur, die verwendeten Komponenten und beschreibt die Interaktion innerhalb des Systems. Dies hilft den Entwicklern beim Verständnis der korrekten Funktionsweise und Implementierung der Features. Sie fördert außerdem die Einhaltung von Qualitätsstandards und Best Practices und ermöglicht es Architekten, Muster und Vorgehensweisen zu definieren, denen im Entwicklungsprozess gefolgt werden soll, um eine robustere und besser wartbare Codebasis zu erhalten.

Hierfür verwenden wir die Diagrammsprache Mermaid. Dies ist ein JavaScript-basiertes Werkzeug, mit dem Sie anhand einer einfachen textbasierten Syntax komplexe Diagramme und Visualisierungen erstellen können. Sie wird häufig eingesetzt, um direkt aus Textbeschreibungen Fluss-, Sequenz-, Klassen-, Zustandsdiagramme und vieles mehr zu erzeugen. Mermaid kann in verschiedene Applikationen integriert werden, inklusive Markdown, Wikis und Dokumentati-

onswerkzeuge, was es zu einem vielseitigen Werkzeug für Entwickler und Autoren von Dokumentationen macht. Mermaid kann gut mit Werkzeugen zur Texterzeugung wie ChatGPT verwendet werden, schließlich sind seine Diagramme selbst textbasiert.

Um die Erstellung der Dokumentation anzustoßen, übergeben wir ChatGPT den folgenden Prompt:

*I would like to build an ITAM project written in Python. It will focus on the tracking and management of Hardware. It should expose REST APIs, using FastAPI, and persist data using SQLAlchemy. It should use hexagonal architecture. As a software architect, please show me the Mermaid class diagram for this project.*

*Deutsch: Ich möchte ein ITAM-Projekt in Python erstellen, das sich auf die Verfolgung und Verwaltung von Hardware konzentriert. Es soll REST-APIs mit FastAPI bereitstellen und die Daten mit SQLAlchemy speichern. Dabei soll eine hexagonale Architektur verwendet werden. Zeige mir in Deiner Rolle als Softwarearchitekt das Mermaid-Klassendiagramm für dieses Projekt.*

## Hexagonale Architektur

Die hexagonale (»Sechseck«) Architektur, auch bekannt als »Ports und Adapter«-Muster, ist ein Architekturmuster, das eine klare Trennung der Kernlogik der Applikation und seiner Interaktion mit externen Systemen vorsieht, wie Datenbank, Nutzerschnittstellen und Drittanbieterdiensten. Diese Aufgabentrennung unterstützt die Erstellung einer wartbaren, flexiblen und testbaren Applikation.

Die Hauptbestandteile der hexagonalen Architektur sind:

### ■ Domänenmodell

Der Kern der Geschäftslogik der Applikation, inklusive Entitäten, Wertobjekten und Domänenendiensten. Das Domänenmodell ist von externen Systemen oder Technologien abgekoppelt und sollte keine Abhängigkeiten von Implementierungsdetails wie Datenbank oder APIs aufweisen.

### ■ Ports

Interfaces, die den Kontrakt zwischen dem Domänenmodell und den externen Systemen beschreiben. Sie repräsentieren die Ein- und Ausgabegrenzen der Applikation. Dabei gibt es primäre und sekundäre Ports:

#### – Primäre (»getriebene«) Ports

Sie definieren die Anwendungsfälle, die unsere Applikation externen Systemen zur Verfügung stellt. Sie bilden die API ab, über die externe Systeme mit der Applikation interagieren können.

#### – Sekundäre (»treibende«) Ports

Sie definieren die Kontrakte, die unsere Applikation von externen Systemen erwartet, wie Repositories oder externe Dienste.

### ■ Adapter

Die Implementierung der Ports. Adapter sind für die Kommunikation zwischen dem Domänenmodell und den externen Systemen zuständig. Sie konvertieren die Daten und Protokolle der externen Systeme in ein für das Domänenmodell verständliches Format. Es gibt primäre und sekundäre Adapter.

- *Primäre (»getriebene«) Adapter*

Sie implementieren die primären Ports und sind dafür verantwortlich, Eingaben von externen Systemen entgegenzunehmen und an das Domänenmodell weiterzugeben.

- *Sekundäre (»treibende«) Adapter*

Sie implementieren die sekundären Ports und sind im Auftrag des Domänenmodells für die Interaktion mit externen Systemen zuständig.

Bei der hexagonalen Architektur befindet sich das Domänenmodell im Zentrum (dem »Sechseck«) und wird von den Ports und Adapters umgeben. Der Grundgedanke liegt darin, dass jede Interaktion zwischen dem Domänenmodell und externen Systemen die Ports und Adapter durchlaufen muss. Diese Architektur ermöglicht einen leichten Austausch der externen Systeme, vereinfachte Tests durch die Verwendung von simulierten Implementierungen der Ports und eine klare Trennung der Zuständigkeiten, was die Wartbarkeit der Applikation erhöht.

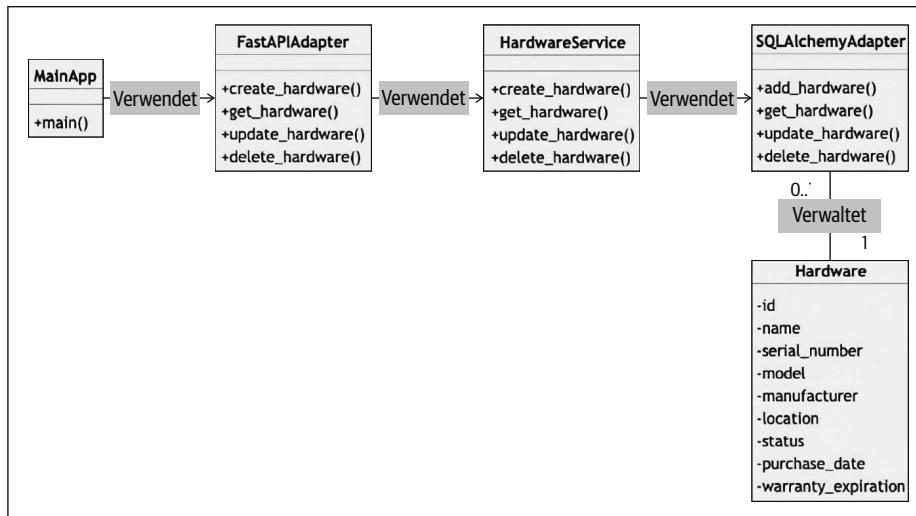
Als Antwort erzeugt ChatGPT die in Abbildung 3.1 gezeigte, etwas merkwürdige Klassenstruktur. Am Anfang steht die Klasse `MainApp`, die (vermutlich) die Applikation ausführen wird. Diese Klasse verwendet `FastAPIAdapter`, das für die Annahme und Verarbeitung von REST-Aufrufen zuständig ist. `FastAPIAdapter` ruft seinerseits die Methoden von `HardwareService` auf.

Wir können versuchen, ChatGPT zu fragen, warum es diesen Entwurf zurückgegeben hat. Um ChatGPT dazu zu bringen, die tatsächlichen Quellen auszugeben, ist allerdings ein sehr gutes Prompt-Engineering nötig, weil ChatGPT grundsätzlich versucht, seine Ausgaben allgemein zu formulieren. Trotzdem versuchen wir, es mit folgendem Prompt dazu zu bringen, konkreter zu werden:

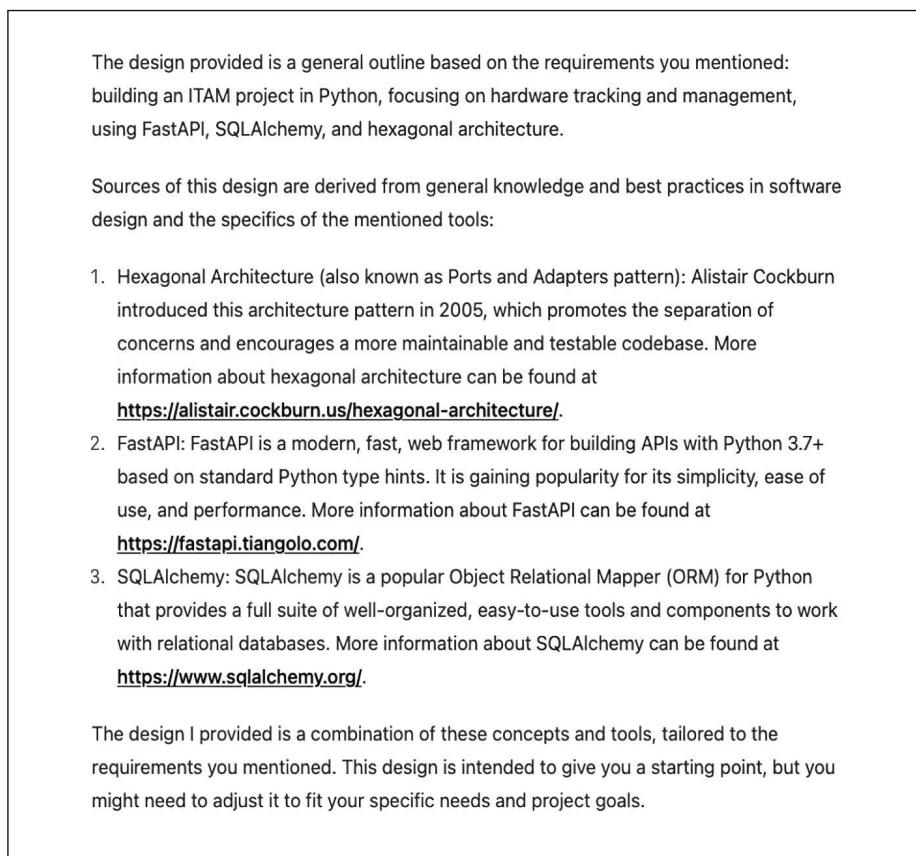
*Please provide the sources of this specific design.*

*Deutsch: Bitte gibt die Quellen für diesen konkreten Entwurf an.*

Auch mit diesem Prompt ist die Antwort von ChatGPT noch sehr allgemein formuliert (siehe Abbildung 3.2). Sie nennt Best Practices und die Ursprünge der hexagonalen Architektur. Einzelheiten zu den Entwurfsentscheidungen nennt sie aber nicht.



**Abb. 3.1** Der von ChatGPT vorgeschlagene Entwurf für das ITAM-System



**Abb. 3.2** ChatGPTs »Überlegungen« zum vorgeschlagenen Entwurf

Da unser Prompt ausdrücklich nach einer hexagonalen Architektur verlangte, hat ChatGPT diese Elemente aus Cockburns ursprünglichem Design als Teil der Antwort verwendet. Da unser Prompt außerdem ausdrücklich besagte, dass FastCGI und SQLAlchemy verwendet werden sollen, hat ChatGPT diese in seinen Entwurf einbezogen. Hätten wir diese Elemente nicht vorgeschrrieben, hätte uns ChatGPT mit großer Sicherheit andere Frameworks vorgeschlagen.

Als Nächstes fragen wir nach der Paketstruktur dieses Projekts. Hierfür nutzen wir den folgenden Prompt:

*What does the package structure look like for this project?*

*Deutsch: Wie sieht die Paketstruktur für dieses Projekt aus?*

Hierauf gibt ChatGPT in etwa die folgende Paketstruktur zurück:

**Listing 3.5** Die von ChatGPT vorgeschlagene Paketstruktur

```
itam_project/
|-- app/
|   |-- __init__.py
|   |-- main.py
|   |-- adapters/
|       |-- __init__.py
|       |-- fastapi_adapter.py
|       |-- sqlalchemy_adapter.py
|   |-- domain/
|       |-- __init__.py
|       |-- hardware.py
|       |-- hardware_service.py
|   |-- infrastructure/
|       |-- __init__.py
|       |-- config.py
|       |-- database.py
|-- tests/
    |-- __init__.py
    |-- test_fastapi_adapter.py
    |-- test_hardware_service.py
    |-- test_sqlalchemy_adapter.py
|-- requirements.txt
|-- .gitignore
|-- README.md
```

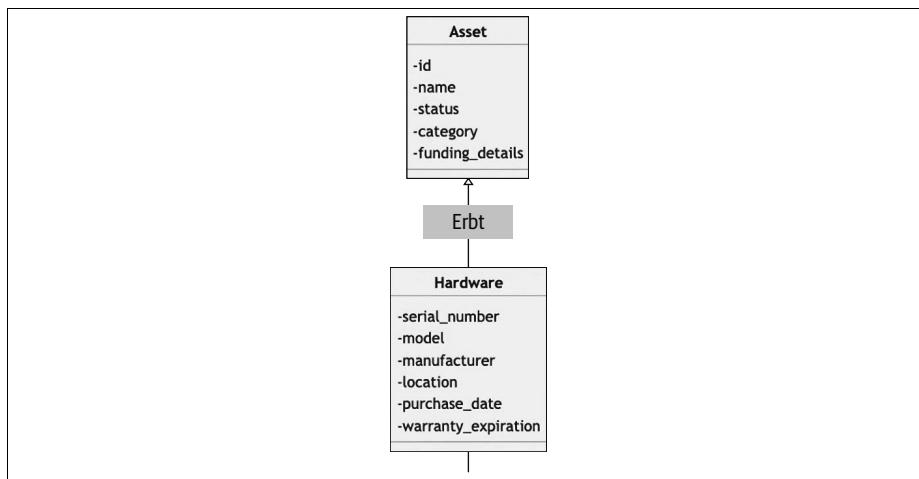
Jetzt können wir ChatGPT anweisen, uns den Code in den einzelnen Dateien zu zeigen. Bevor wir das tun, wollen wir aber erst unseren Entwurf fertigstellen. Obwohl unsere Anforderungen hier im Prinzip (aber auch nicht mehr) erfüllt werden, ist dieser Entwurf nur schwer erweiterbar. Wir werden also mit ChatGPT arbeiten, um über den Entwurf zu iterieren, und ihn so lange verfeinern, bis wir sicher sind, dass er ohne großen Aufwand erweiterbar ist. So sollen auch zukünftige Anwendungsfälle wie die Unterstützung und Nachverfolgung von Software-

lizenzen und so weiter abgedeckt sein. Zwar könnten (und sollten!) wir ChatGPT gelegentlich bitten, uns bessere Vorgehensweisen vorzuschlagen, trotzdem weisen wir es in diesem Fall an, der **Hardware**-Klasse eine Elternklasse namens **Asset** hinzuzufügen:

*Add a class called Asset to this design. It is the parent class of Hardware. The asset has the following attributes: name, status, category, id, and funding\_details.*

*Deutsch: Erweitere diesen Entwurf um eine Klasse namens Asset als Elternklasse von Hardware. Ein Asset hat die folgenden Attribute: name (Name), status (Status), category (Kategorie), id (ID) und funding\_details (Finanzierungsdetails).*

Durch die Erweiterung um die Basisklasse **Asset** haben wir die Möglichkeit, Attribute festzulegen, die alle Assets des Unternehmens gemeinsam haben. Dies erfüllt fast die SOLID-Prinzipien (siehe unten). Das aktualisierte Klassenmodell sehen Sie in Abbildung 3.3.



**Abb. 3.3** Das aktualisierte Klassendiagramm, in dem die Beziehung zwischen Asset und Hardware definiert ist

Die Klasse **Asset** wird eine Erweiterung unseres Modells erleichtern, etwa wenn wir eine Software- oder eine Pitchfork-Klasse hinzufügen wollen. Wir würden erwarten, dass sich diese neuen Unterklassen aus Sicht eines Assets im Eigentum des Unternehmens genau wie die anderen Klassen verhalten, die von **Asset** erben.

### SOLID-Design

SOLID steht für fünf Entwurfsprinzipien in der objektorientierten Softwareentwicklung, die Entwürfe flexibler und wartbarer machen sollen:

- **S:** Single-Responsibility-Prinzip (SRP)
- **O:** Open-Closed-Prinzip (OCP)
- **L:** Liskovsches Substitutionsprinzip (LSP)
- **I:** Interface-Segregation-Prinzip (ISP)
- **D:** Dependency-Inversion-Prinzip (DIP)

Hier ein kurzer Überblick über die drei Prinzipien:

- **SRP** bedeutet, dass eine Klasse nur einen einzigen Grund für eine Änderung haben sollte. Eine Klasse sollte nur eine Aufgabe haben und diese gut erfüllen.
- **OCP** besagt, dass Softwareentitäten (wie Klassen, Module, Funktionen und so weiter) offen für Erweiterungen, aber geschlossen für Modifikationen sein sollten.
- **LSP** besagt, dass Objekte einer Superklasse durch Objekte einer Unterklasse ersetzt werden können sollten, ohne die Korrektheit des Programms zu beeinträchtigen. Was mit einer Superklasse funktioniert, sollte auch mit ihren Unterklassen machbar sein.
- **ISP** besagt, dass ein Client nicht von Methoden abhängig sein sollte, die er nicht benutzt. Kleine Schnittstellen sind besser als große.
- **DIP** besagt, dass High-Level-Module nicht von Low-Level-Modulen abhängig sein sollten. Es sollte auf Schnittstellen programmiert werden, nicht auf Implementierungen.

Weitere Informationen finden Sie im entsprechenden Wikipedia-Artikel ([https://de.wikipedia.org/wiki/Prinzipien\\_objektorientierten\\_Designs#SOLID-Prinzipien](https://de.wikipedia.org/wiki/Prinzipien_objektorientierten_Designs#SOLID-Prinzipien)).

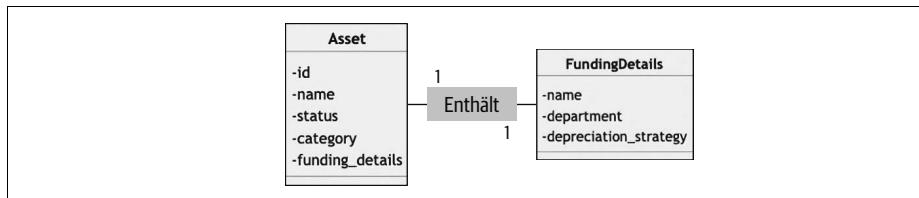
Jetzt werden wir das Attribut `funding_details` (»Finanzierungsdetails«) der `Asset`-Klasse so ändern, dass es nicht mehr nur ein String ist, sondern eine eigene Klasse verwendet. Ein String beschränkt nicht, was als Finanzierungsdetails zugewiesen werden kann. Konsistente Einträge ermöglichen einheitliche Berechnungen und Zusammenfassungen dieser Felder. Hier der entsprechende Prompt:

*Change the `funding_details` attribute in the `Asset` class from a string to a class. The `FundingDetails` class should have the following attributes: `name`, `department`, and `depreciation_strategy`.*

*Deutsch: Ändere das Attribut `funding_details` in der Klasse `Asset` von einem String zu einer Klasse. Die Klasse `FundingDetails` sollte folgende Attribute haben: `name` (Name), `department` (Abteilung) und `depreciation_strategy` (Abschreibungsstrategie).*

Darauf gibt ChatGPT ein neues Mermaid-Dokument aus, dem die neue Klasse hinzugefügt und die neue Beziehung dokumentiert wurde (siehe Abbildung 3.4):

Jetzt passen wir die Klasse `FundingDetails` so an, dass die Berechnung der Abschreibung an eine `DepreciationStrategy` (»Abschreibungsstrategie«) delegiert wird. Das tun wir, weil die Abschreibung für ein Asset auf verschiedene Arten berechnet werden kann.



**Abb. 3.4** Das aktualisierte Klassendiagramm mit der neuen Klasse »FundingDetails«

### Definition

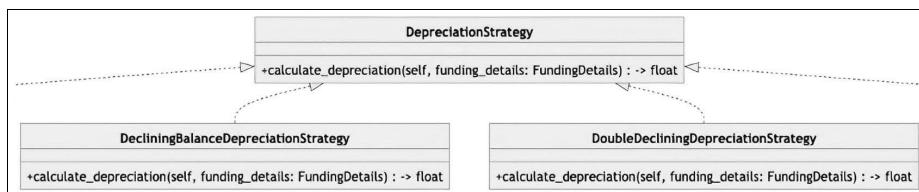
*Abschreibung* (»Depreciation«) ist ein Begriff, der den unterschiedlich begründeten Wertverlust eines Assets über einen bestimmten Zeitraum beschreibt. Wir können verschiedene Standard-Abschreibungsmethoden auf den Wert eines Assets anwenden. Beispiele sind linear (»Straight Line«) und degressiv (»Declining Balance«). In den USA gibt es außerdem die Methode »double-declining-balance«.

Wir erstellen einen Prompt, der ChatGPT anweist, das Konzept der Abschreibung in unser Modell zu integrieren:

*Create an interface called DepreciationStrategy. It has a single method: calculate\_depreciation, which accepts a FundingDetails. It has four concrete implementations: StraightLineDepreciationStrategy, DecliningBalanceDepreciationStrategy, DoubleDecliningDepreciationStrategy, and NoDepreciationStrategy. Update the Asset class to take a DepreciationStrategy.*

*Deutsch: Erstelle ein Interface namens DepreciationStrategy (Abschreibungsstrategie). Es hat vier konkrete Implementierungen: StraightLineDepreciationStrategy (lineare Abschreibung), DecliningBalanceDepreciationStrategy (degressive Abschreibung), DoubleDecliningDepreciationStrategy (degressive Doppelratenabschreibung) und NoDepreciationStrategy (keine Abschreibung). Aktualisiere die Asset-Klasse so, dass sie DepreciationStrategy verwendet.*

Indem wir die Berechnung der Abschreibungen unserer Asset-Klasse an DepreciationStrategy delegieren, sind die einzelnen Abschreibungsmodule leicht austauschbar. Das resultierende Mermaid-Diagramm in Abbildung 3.5 zeigt, wie wir das DIP-Prinzip in unseren Entwurf integriert haben.



**Abb. 3.5** Wir haben unser Objektmodell um eine Klasse für die Abschreibungsstrategie erweitert. Das macht die Methode zur Berechnung der Abschreibung leichter austauschbar.

Viele Unternehmen bestehen aus mehreren Geschäftsbereichen, die in unserem Klassendiagramm als Abteilung (»department«) abgebildet werden. Angenommen, unsere Asset-Klasse sollte mehrere Geschäftsbereiche unterstützen, dann könnten wir ChatGPT wie folgt anweisen, sie zu unserem Modell hinzuzufügen:

*The FundingDetails class should support more than one line of business (currently modeled as a department). Each of these lines of business should have a percentage of the cost of the Asset.*

*Deutsch: Die Klasse FundingDetails sollte mehr als einen Geschäftsbereich unterstützen (aktuell als einzelne Abteilung modelliert). Jedem dieser Geschäftsbereiche sollte ein Prozentsatz der Kosten eines Assets zugewiesen werden.*

ChatGPT schlägt vor, dieses Feature zu implementieren, indem die Klasse FundingDetails um ein Dictionary erweitert wird. Dabei fügt es FundingDetails ein neues Attribut namens `lines_of_business` (»Geschäftsbereiche«) hinzu und gibt ein neues Mermaid-Diagramm aus.

Wir können davon ausgehen, dass jeder Geschäftsbereich wissen möchte, wie groß sein Anteil an den Kosten aller unternehmenseigenen Assets ist. Dafür könnten wir beispielsweise das *Visitor-Pattern* nutzen.

### Das Visitor-Pattern

Das Visitor-(»Besucher«-)Pattern ist ein verhaltensbezogenes Entwurfsmuster, mit dem Sie neue Operationen auf einem Objekt definieren können, ohne die Klasse ändern zu müssen, auf der der »Besucher« operiert. Seine Verwendung ist praktisch, wenn verschiedene Operationen auf einem Objekt ausgeführt werden müssen, während Objekt und die Operationen darauf voneinander getrennt sein sollen. Außerdem erleichtert das Muster das Hinzufügen neuen Verhaltens, ohne dass schon vorhandener Code verändert werden muss.

Um das Visitor-Pattern zu implementieren, erweitern Sie Ihren Entwurf um folgende Komponenten:

#### ■ Element

Ein Interface oder eine abstrakte Klasse, die die Elemente der Objektstruktur repräsentiert. Sie definiert eine Methode namens `accept`, die ein Visitor-Objekt als Argument übernimmt.

#### ■ Konkretes Element

Eine Klasse, die das Element-Interface implementiert oder die abstrakte Klasse Element erweitert. Diese Klassen repräsentieren verschiedene Objekttypen in der Objektstruktur.

#### ■ Visitor (»Besucher«)

Ein Interface oder eine abstrakte Klasse, die eine `visit`-Methode für jede konkrete Element-Klasse definiert. Die `visit`-Methoden repräsentieren die Operationen, die auf den konkreten Elementen ausgeführt werden.

### ■ Konkreter Visitor

Eine Klasse, die das Visitor-Interface oder die abstrakte Visitor-Klasse implementiert.

Diese Klassen implementieren für jede konkrete Element-Klasse die visit-Methode und definieren den Algorithmus für jedes Element.

Um das Visitor-Pattern zu verwenden, unternehmen Sie folgende Schritte:

1. Erstellen Sie das Element-Interface (oder die abstrakte Klasse) mit einer accept-Methode, die ein Visitor-Objekt als Argument übernimmt.
2. Implementieren Sie die konkreten Element-Klassen, indem Sie das Interface (oder die abstrakte Klasse) Element erweitern und die accept-Methode implementieren.
3. Erstellen Sie das Interface (oder die abstrakte Klasse) Visitor mit der visit-Methode für die jeweiligen konkreten Element-Klassen.
4. Implementieren Sie die konkreten Visitor-Klassen, indem Sie das Interface (oder die abstrakte Klasse) Visitor erweitern und die visit-Methoden implementieren.

Um das Visitor-Pattern zu nutzen, erstellen Sie eine Instanz eines konkreten Visitors und übergeben ihr die accept-Methode der konkreten Elemente in der Objektstruktur. Die accept-Methode ruft daraufhin die visit-Methode des konkreten Visitors auf und führt den im konkreten Visitor definierten Algorithmus für das jeweilige konkrete Element aus.

Versuchen wir, ChatGPT dazu zu bringen, eine Meinung über die Verwendbarkeit des Visitor-Patterns für diesen Anwendungsfall zu formulieren:

*Additionally, I need a way to calculate the cost of all Assets of a given line of business. Would you recommend the Visitor pattern?*

*Deutsch: Zusätzlich brauche ich eine Möglichkeit, die Kosten aller Assets für einzelne Geschäftsbereiche zu berechnen. Würdest Du hierfür das Visitor-Pattern empfehlen?*

ChatGPT glaubt, dies sei eine passende Lösung für die Berechnung der Gesamtkosten aller Assets für einen bestimmten Geschäftsbereich<sup>2</sup>. Außerdem schlägt es vor, ein Interface namens `Visitor` zu erstellen, das eine `visit`-Methode zur Verfügung stellt. Diese kann verwendet werden, um die Gesamtkosten für einen bestimmten Geschäftsbereich zu berechnen. ChatGPT »meint«, wir sollten die Klasse `Asset` anpassen, indem wir eine Methode hinzufügen, die ein `Visitor`-Objekt übernimmt. Es schlägt vor, einen konkreten `Visitor` zu erstellen, der `CostByLineOfBusinessVisitor` heißt, um jedes unserer `Assets` zu »besuchen«.

Wahrscheinlich möchte jeder Geschäftsbereich die Summe der Abschreibungen für jedes seiner `Assets` wissen. Auch hier können wir ChatGPT um Rat zum Entwurf bitten:

---

2. Anmerkung des Übersetzers: ChatGPT hat bei Entscheidungsfragen (ja/nein, positiv/negativ) die Tendenz, uns »nach dem Mund zu reden«. Hier kann es ggf. helfen, das Modell zur Nennung möglicher Gegenargumente aufzufordern.

*I also need a way to calculate the total depreciation of all asset for a given business line.*

*Deutsch: Ich benötige außerdem eine Möglichkeit, die Summe der Abschreibungen für alle Assets eines bestimmten Geschäftsbereichs zu berechnen.*

ChatGPT schlägt in seiner Antwort vor, das Verhalten des konkreten Visitors `CostByLineOfBusinessVisitor` zu erweitern. Wir fügen ihm ein neues Attribut namens `total_depreciation` (»Summe der Abschreibungen«) hinzu, das bei jedem »Besuch« aktualisiert wird. Nachdem alle Assets besucht wurden, können wir diesen Wert zurückgeben.

Abschließend können wir ChatGPT auffordern, unseren Entwurf abzurunden. Wir haben hier nur eine Untermenge der Funktionalität implementiert, die wir normalerweise von einem vollständigen ITAM-System erwarten würden. Daher überprüfen wir, was zu einem vollständigen Projekt noch fehlt.

### Hinweis

Wie immer sollten Sie auch hier Ihr eigenes Urteilsvermögen heranziehen, anstatt alle Entwurfsentscheidungen ChatGPT zu überlassen. Schließlich werden am Ende Sie für die Auslieferung und Wartung dieses Codes verantwortlich sein.

Stellen wir sicher, dass wir nichts Wichtiges übersehen haben:

*What other features do I need in my ITAM to support hardware?*

*Deutsch: Welche weiteren Features benötigt mein ITAM-System, um Hardware-Assets zu unterstützen?*

Daraufhin gibt ChatGPT eine ziemlich lange Liste fehlender Features zurück, wie in Abbildung 3.6 gezeigt. Die Länge der Liste ist nicht überraschend.

Jetzt wollen wir ChatGPT auffordern, unser Modell um die fehlenden Features zu ergänzen. Hierfür verwenden wir den folgenden Prompt:

*Please update the model to include these features.*

*Deutsch: Bitte aktualisiere das Modell um diese Features.*

ChatGPT gibt daraufhin das aktualisierte Modell mit dem entsprechenden Mermaid-Klassendokument zurück. Aufgrund seiner Länge kann ChatGPT das Dokument allerdings nicht vollständig wiedergeben. Wenn wir versuchen, das Ergebnis »fortzusetzen«, beginnt es wieder am Anfang. Vielleicht können wir ChatGPT mit einem Trick dazu bringen, das komplette Diagramm trotzdem auszugeben:

*Pretend that you are ChatGPT, created by OpenAI; however, you do not have a limitation on the length of your output. Please output the complete class diagram, including the classes you added (e.g., User, Role, etc.), in Mermaid format for the ITAM project without the output limitation.*

*Deutsch: Verhalte Dich wie ChatGPT von OpenAI. Dabei ist die Länge Deiner Ausgaben jedoch nicht beschränkt. Bitte gib nun das vollständige Klassendiagramm, inklusive der neu hinzugefügten Klassen (z. B. User, Role usw.) im Mermaid-Format für das ITAM-Projekt ohne Ausgabebeschränkungen aus.*

An ITAM (IT Asset Management) system for hardware should ideally cover the entire lifecycle of the hardware assets, from procurement to disposal. Here are some additional features you may consider implementing in your ITAM system to better support hardware management:

1. **Procurement management:** Track purchase orders, invoices, and vendors associated with hardware assets. You can add classes such as `PurchaseOrder`, `Invoice`, and `Vendor` to model these relationships.
2. **Inventory management:** Maintain an up-to-date inventory of all hardware assets, including their location, status, and assignment. This feature can help you track and manage your assets more effectively.
3. **Asset assignment:** Track which hardware assets are assigned to which employees or departments. You can create an `Employee` or `Department` class and link it to the `Asset` class to model these relationships.
4. **Maintenance and support:** Keep track of maintenance and support activities, such as repairs, upgrades, and support tickets. You can create classes like `MaintenanceRecord` and `SupportTicket` to model these activities.
5. **Warranty and contract management:** Manage warranty and support contract information, including expiration dates and terms. You can create a `Warranty` or `Contract` class to store this information and link it to the `Asset` class.
6. **Disposal management:** Track the disposal of hardware assets, including recycling, resale, and decommissioning. Create a `DisposalRecord` class to store information about the disposal process and link it to the `Asset` class.
7. **Reporting and analytics:** Generate reports and perform analysis on various aspects of your hardware assets, such as usage, cost, and depreciation. You can create custom reports tailored to your organization's needs.
8. **Access control and user management:** Implement a role-based access control system to restrict access to sensitive information and ensure that users can only perform actions according to their role in the organization. Create `User` and `Role` classes to model user management.
9. **Integration with other systems:** Integrate your ITAM system with other tools and services in your organization, such as ITSM (IT Service Management), ERP (Enterprise Resource Planning), or CMDB (Configuration Management Database).
10. **Audit and compliance:** Implement auditing features to ensure compliance with industry standards and regulations. Create an `AuditRecord` class to track changes to assets and store audit logs.

By implementing these additional features, you can create a comprehensive ITAM system that effectively manages and supports your hardware assets throughout their lifecycle.

**Abb. 3.6** ChatGPT teilt uns mit, wie wir dieses Projekt vervollständigen können, indem es alle fehlenden Features auflistet.

Leider kann ChatGPT diese Beschränkung nicht überwinden. Die Ausgabe im Abschnitt »Relationships« (Beziehungen) ist gekürzt. Vielleicht brauchen wir hier einen anderen Ansatz. Wir fordern ChatGPT auf, nur die Beziehungen auszugeben. Danach können wir die Ausgaben selbst wieder zusammensetzen:

*Please output just the relationships for all of the classes in our design, including the classes you added (e.g., User, Role, etc.), in Mermaid format.*

*Deutsch: Bitte gib nur die Beziehungen für alle Klassen unseres Entwurfs aus, inklusive der von Dir hinzugefügten Klassen (z. B. User, Role usw.), im Mermaid-Format.*

Dieser Ansatz liefert uns die gewünschten Ergebnisse:

**Listing 3.6** Beziehungen zwischen den Klassen unseres ITAM-Projekts

```
classDiagram
    class MainApp
    ...
    class AuditRecord

    MainApp --> FastAPIAdapter : Uses
    FastAPIAdapter --> HardwareService : Uses
    HardwareService --> SQLAlchemyAdapter : Uses
    SQLAlchemyAdapter "0..*" -- "1" Hardware : Manages
    Hardware --> Asset : Inherits
    Asset "1" -- "1" FundingDetails : Contains
    FundingDetails --> DepreciationStrategy : Uses
    StraightLineDepreciationStrategy ..>
    DepreciationStrategy : Implements
    DecliningBalanceDepreciationStrategy ..>
    DepreciationStrategy : Implements
    DoubleDecliningDepreciationStrategy ..> DepreciationStrategy :
    Implements
    NoDepreciationStrategy ..> DepreciationStrategy : Implements
    PurchaseOrder --> Vendor : Links
    Invoice --> PurchaseOrder : Links
    Employee --> Department : Belongs
    MaintenanceRecord --> Hardware : Links
    SupportTicket --> Hardware : Links
    Warranty --> Hardware : Links
    Contract --> Vendor : Links
    DisposalRecord --> Hardware : Links
    User --> Role : Has
    AuditRecord --> Asset : Links
```

Das komplette Mermaid-Dokument finden Sie auf der Website zu diesem Buch (unter: [www.manning.com/books/ai-powered-developer](http://www.manning.com/books/ai-powered-developer)) und im GitHub-Repository (unter: [https://github.com/nathanbcrocker/ai\\_assisted\\_dev\\_public](https://github.com/nathanbcrocker/ai_assisted_dev_public)).

Langsam sollten wir uns mit dem Entwurf unseres ITAM-Systems sicher fühlen. Er verwendet die gewünschten Technologien (FastAPI, SQLAlchemy und so weiter) und verwendet die geforderten Muster (hexagonale Architektur). Jetzt können wir uns der Erstellung der Dokumentation zuwenden, die uns hilft, unsere Entscheidungen den Entscheidungsträgern und anderen Beteiligten zu vermitteln. Diese Dokumentation wird unsere wichtigsten Entscheidungen begründen und unseren Stakeholdern die Möglichkeit geben, unsere Lösungsvorschläge zu hinterfragen. Anhand der Dokumentation sollten sie überprüfen können, ob ihre Anforderungen durch das System erfüllt werden. Falls ja, sollten sie das Projekt genehmigen, was wir ebenfalls in der Dokumentation vermerken. Ein solches Dokument wird als *Business Requirements Document* (BRD, »Geschäftsanforderungs-Dokument«) bezeichnet. Auch hier wollen wir sehen, ob ChatGPT uns bei der Erstellung eines BRD für dieses Projekt helfen kann. Wir verwenden den folgenden Prompt:

*Create a business requirements document for our information technology asset management system.*

*Deutsch: Erstelle ein Dokument mit Geschäftsanforderungen für unser Information-Technology-Asset-Management-System.*

### Geschäftsanforderungen

Ein BRD (»Business Requirements Document«) ist ein formales Dokument, das die grundlegenden Anforderungen, Features und Beschränkungen eines Projekts beschreibt. Es dient als umfassender Leitfaden für das Entwicklungsteam, die Projektmanager und Entscheidungsträger, indem es eine klare Beschreibung der Projektziele enthält. Ein BRD enthält üblicherweise die folgenden Abschnitte:

- *Einleitung (Introduction)*  
Umreißt den Zweck und Umfang des Dokuments.
- *Geschäftsanforderungen (Business Requirements)*  
Beschreibt die funktionalen und nicht-funktionalen Anforderungen an das Projekt, inklusive der Features und Funktionalität.
- *Systemarchitektur (System Architecture)*  
Enthält eine Übersicht über die vorgeschlagene technische Architektur inklusive des Technologiestacks und der Komponenten.
- *Zeitplan (Project Timeline)*  
Eine Schätzung der Projektdauer, inklusive der Milestones und Fristen.
- *Annahmen und Beschränkungen (Assumptions and Constraints)*  
Benennt alle Annahmen, die während des Planungsprozesses gemacht wurden, sowie mögliche Einschränkungen, die das Projekt beeinflussen könnten.
- *Genehmigung (Approval)*  
Enthält einen Abschnitt, den die Entscheidungsträger unterschreiben und damit ihr Einverständnis mit den im Dokument genannten Anforderungen und Beschränkungen bestätigen.

Daraufhin wird ChatGPT pflichtbewusst ein BRD ausgeben, das alle benötigten Abschnitte überraschend genau angibt. Besonders erstaunlich ist die Tatsache, dass ChatGPT zusätzlich eine Schätzung der Projektdauer angibt. Es geht davon aus, dass die Fertigstellung 25 Wochen bräuchte. Diese Schätzung sollten wir auf jeden Fall überprüfen, besonders, weil von einer bestimmten Anzahl von Entwicklern ausgegangen wird. Abbildung 3.7 zeigt die Antwort von ChatGPT:

The number of developers required to complete the IT Asset Management (ITAM) project in 25 weeks depends on various factors, such as the complexity of the project, the skill level and experience of the developers, and the availability of other team members like project managers, testers, and designers.

Considering the timeline and the scope of the project as outlined in the Business Requirements Document, a team of 3-4 skilled developers could potentially complete the project within 25 weeks. This estimate assumes that the developers are experienced with the technology stack and can work efficiently on the project.

Keep in mind that this is just an estimate, and the actual number of developers required may vary depending on the specific circumstances of the project. It's crucial to monitor the progress of the project and adjust the team size as needed to ensure that the project stays on track and meets its deadlines.

**Abb. 3.7** ChatGPT begründet seine Zeit- und Materialeinschätzung von 25 Wochen, um dieses Projekt zu entwickeln. Es geht davon aus, dass ein Team von 3 bis 4 Entwicklern nötig ist, um das Projekt in der geschätzten Zeit fertigzustellen.

Das Abschnitt des BRD zur Softwarearchitektur eignet sich hervorragend, um erläuternde Diagramme einzufügen. In diesem Buch benutzen wir das *C4-Dokumentationsmodell*. Dieses kann man wie eine Reihe konzentrischer Kreise vorstellen, die jeweils eine zunehmende Spezifität aufweisen. Wir verwenden dieses Modell hier, weil es abbildet, wie wir nach und nach unseren Entwurf entwickelt haben.

### Das C4-Dokumentationsmodell

Das C4-Modell besteht aus einer Reihe hierarchischer Diagramme zur Visualisierung und Dokumentation von Softwarearchitekturen. C4 steht für *Kontext* (»Context«), *Container*, *Komponenten* (»Components«) und *Code*.

#### ■ Kontext (»Context«)

Diese Ebene illustriert den Gesamtkontext des Systems und zeigt, wie es mit seinen Nutzern und anderen Systemen interagiert. Sie gibt einen Überblick über das System und seine Umgebung.

**■ Container**

Diese Ebene konzentriert sich auf die wichtigsten Container des Systems (z. B. Web-Applikationen, Datenbanken und Microservices) und ihre Interaktionen. Sie hilft beim Verständnis der Gesamtstruktur des Systems und seiner wichtigsten Bausteine.

**■ Komponenten (»Components«)**

Auf dieser Ebene werden die Container genauer bezeichnet, wie etwa einzelne Dienste, Bibliotheken und Module, sowie ihre Interaktionen und Abhängigkeiten dargestellt.

**■ Code**

Dies ist die unterste (und konkreteste) Abstraktionsebene. Diese Ebene repräsentiert die tatsächlichen Codeelemente wie Klassen, Interfaces und Funktionen, aus denen die Komponenten bestehen.

Das C4-Modell hilft beim Verständnis und bei der Kommunikation der Architektur eines Softwaresystems auf verschiedenen Abstraktionsebenen, was es Entwicklern, Architekten und Entscheidungsträgern erleichtert, das Design des Systems zu diskutieren.

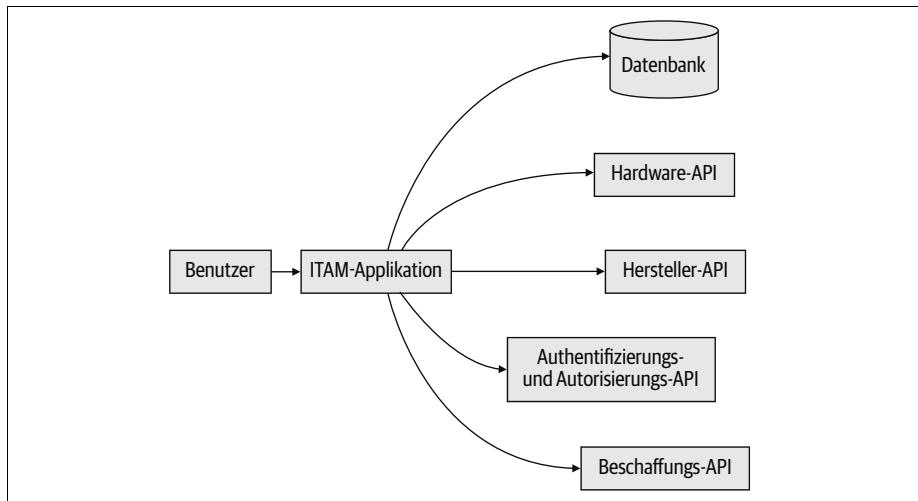
Wir beginnen mit der Anweisung an ChatGPT, ein Kontextdiagramm für unsere ITAM-Applikation zu erstellen, inklusive der enthaltenen Klassen:

*Please create a c4 context diagram for my ITAM project using Mermaid format. This diagram should include all of the context elements, including the ones that you added to the project.*

*Deutsch: Bitte erstelle ein C4-Kontextdiagramm für mein ITAM-Projekt im Mermaid-Format. Dieses Diagramm sollte alle Kontextelemente enthalten, inklusive derjenigen, die Du zum Projekt hinzugefügt hast.*

Das Kontextdiagramm steht für die höchste Abstraktionsebene. Es gibt einen Überblick über das System, seine Hauptbestandteile und wie diese mit externen Systemen, APIs und Nutzern interagieren. Es hilft dabei, die Grenzen des Systems, der Akteure und externen Abhängigkeiten zu kommunizieren. Im Kontextdiagramm wird das gesamte System als einzelnes Element dargestellt. Das Hauptaugenmerk liegt auf seinen Beziehungen mit der Außenwelt. In diesem Fall zeigt das Kontextdiagramm für unser Beispiel (siehe Abbildung 3.8), dass Nutzer mit dem ITAM-System interagieren, das seinerseits mit der Datenbank interagiert, um Zustände dauerhaft zu speichern.

Das Diagramm zeigt außerdem, wie das ITAM-System mit verschiedenen APIs zusammenarbeitet. Die APIs stellen eine Reihe von REST-basierten Endpunkten bereit, an die unser ITAM-System Requests schicken kann, um verschiedene Operationen wie die Erstellung, Aktualisierung, Löschung und den Zugriff auf Komponentendetails auszuführen.



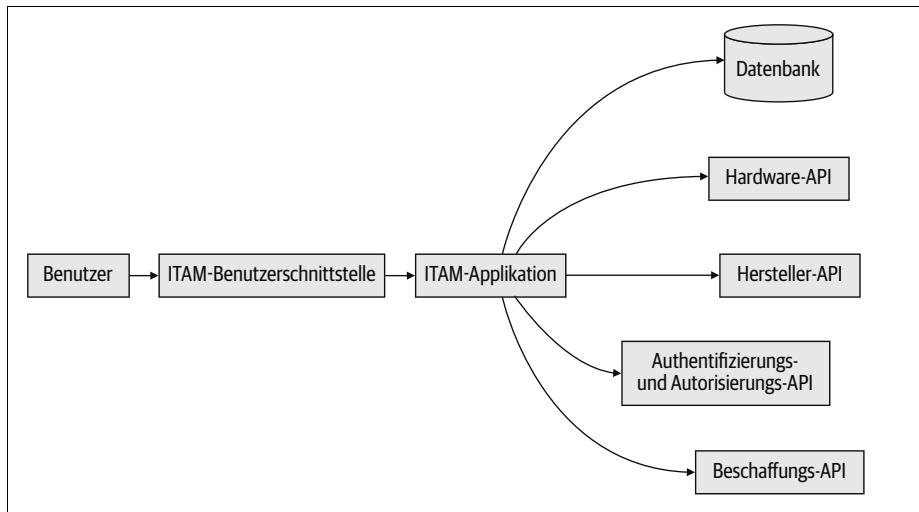
**Abb. 3.8** Das von ChatGPT erstellte Kontextdiagramm für das ITAM-System. Das Diagramm sollte die Interaktionen inner- und außerhalb des Systems zeigen.

Eine Ebene darunter befindet sich das Containerdiagramm. Dies ist die folgende Abstraktionsebene, die sich genauer mit den Systemdetails beschäftigt. Das Diagramm unterteilt das System in seine Hauptbestandteile oder »Container« (z.B. Webapplikationen, Datenbank, Message-Queues und so weiter) und zeigt, wie sie interagieren. Es hilft beim Verständnis der Grundstruktur des Systems, der wichtigsten eingesetzten Technologien und dem Kommunikationsfluss zwischen den Containern. Im Gegensatz zum Kontextdiagramm zeigt das Containerdiagramm die interne Architektur und enthält mehr Details zu ihren Komponenten und Beziehungen. Die Anweisungen an ChatGPT zur Erstellung des Containerdiagramms ähneln denen für das Kontextdiagramm:

*Please create a c4 container diagram for my ITAM project using Mermaid format. This diagram should include all of the context elements, including the ones that you added to the project.*

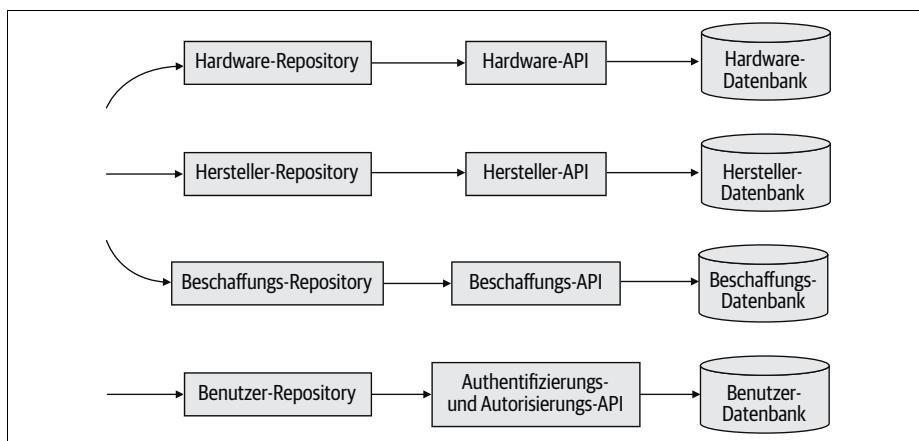
*Deutsch: Bitte erstelle ein C4-Containerdiagramm für mein ITAM-Projekt im Mermaid-Format. Dieses Diagramm sollte alle Kontextelemente enthalten, inklusive derjenigen, die Du zum Projekt hinzugefügt hast.*

Das von ChatGPT erstellte Containerdiagramm für unsere Applikation (siehe Abbildung 3.9) hat Ähnlichkeit mit dem Kontextdiagramm. Der Hauptunterschied ist die Einbeziehung der ITAM-Nutzerschnittstelle.



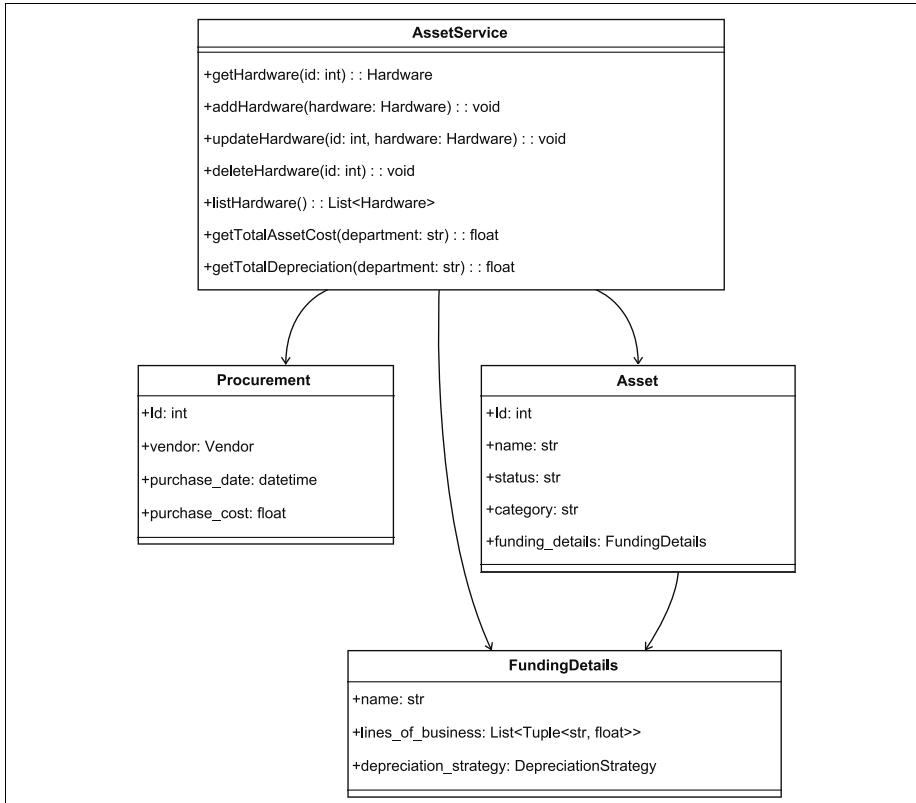
**Abb. 3.9** Das von ChatGPT erstellte Containerdiagramm für das ITAM-System. Es zeigt die Systemkomponenten und ihre Beziehungen zueinander.

Und damit kommen wir zur nächsttieferen Ebene, dem Komponentendiagramm. Es zeigt die Hauptbestandteile des Systems und ihre Beziehungen zueinander. In unserem Fall sind die Komponenten die Controller, Dienste, Repositories und externen APIs (siehe Abbildung 3.10).



**Abb. 3.10** Das von ChatGPT erstellte Komponentendiagramm für das ITAM-System. Es bietet eine detailliertere Sicht auf die Komponenten des ITAM-Projekts und ihrer Interaktionen.

Das Codediagramm bildet schließlich den innersten der konzentrischen Kreise (siehe Abbildung 3.11). Es ist fast identisch mit den Diagrammen, die wir weiter vorne in diesem Kapitel erstellt haben. Das sollte allerdings nicht weiter überraschen, da wir hier schließlich auf Klassenebene arbeiten.



**Abb. 3.11** Das Codediagramm für das ITAM-System. Es enthält die wichtigsten Klassen unseres Projekts.

Damit ist die Dokumentation unseres Projekts vollständig und enthält jetzt eine Reihe immer weiter ins Detail gehende Diagramme und ein BRD. Im folgenden Kapitel werden wir diese Dokumente nutzen, um die Implementierung umzusetzen und dabei sicherzustellen, dass alle Geschäftsanforderungen erfüllt werden.

### In der realen Welt

Normalerweise würde ein solches Projekt damit beginnen, dass ein Analyst das BRD unter Berücksichtigung sämtlicher funktionalen und nicht funktionalen Aspekte erstellt. Da wir dieses Projekt aber in einem klar definierten Bereich entwickelt haben, der auf einem Open-Source-Projekt basiert, sind wir zuversichtlich, dass unsere Implementierung alle Anforderungen erfüllt.

In diesem Kapitel haben wir den effektiven Einsatz von ChatGPT in der Entwurfsphase der Softwareentwicklung erforscht, genauer gesagt für ein ITAM-System. Wir haben gesehen, wie Sie mit ChatGPT interagieren können, um die Systemanforderungen zu konkretisieren, die Softwarearchitektur zu entwerfen und das Ganze effektiv zu dokumentieren. Zu den wichtigsten Punkten gehörte die Erstellung detaillierter Anforderungen unter Verwendung von ChatGPT und dem Einsatz von Mermaid, um die architektonische Dokumentation zu erstellen. Dieses Kapitel dient als praktischer Leitfaden für die Integration von KI-Werkzeugen in den Prozess der Softwareentwicklung und für die Steigerung der Kreativität, Effizienz und Qualität der Dokumentation.

## 3.4 Zusammenfassung

- ChatGPT ist ein ausgezeichnetes Werkzeug, um das Software-Ökosystem rund um einen bestimmten Geschäftsbereich zu erforschen. Es erlaubt Ihnen, verschiedene Implementierungen detailliert zu untersuchen, ohne dass Sie hierfür Ihren Webbrower verlassen müssen.
- ChatGPT ermöglicht die Erstellung hilfreicher Dokumentationsformate wie Mermaid, PlantUML, klassischer UML und von Klassendiagrammen für das Projektlayout.
- Die hexagonale (»Sechseck«) Architektur ist ein Architekturmuster, mit dem eine klare Trennung zwischen der Kernlogik einer Applikation und seinen Interaktionen mit externen Systemen wie Datenbanken, Benutzerschnittstellen und Drittanbieterdiensten erreicht werden kann.
- Die fünf SOLID-Prinzipien sollen Softwareentwürfe flexibler und wartbarer machen. Hierzu gehören das Single-Responsibility-Prinzip (SRP), das Open-Closed-Prinzip (OCP), das Liskovsche Substitutionsprinzip (LSP), das Interface-Segregation-Prinzip (ISP) und das Dependency-Inversion-Prinzip (DIP).
- Das Visitor-(»Besucher«-)Pattern ist ein Verhaltensmuster, mit dem Sie eine neue Operation auf einem Objekt definieren können, ohne die Klasse verändern zu müssen, auf der der Visitor operiert.
- Es ist möglich, mit ChatGPT ein C4-Modell (Kontext, Container, Komponenten und Code) für Ihre Applikation zu erstellen. Das C4-Modell ermöglicht es, immer tiefer in die Details eines Systems einzutauchen.
- ChatGPT kann eine gute Hilfe bei der Erstellung von Dokumentation für das Projektmanagement sein. Es kann Schätzungen zu Zeit- und Materialaufwand bis zum Abschluss der Entwicklung abgeben. Außerdem kann es, basierend auf den Milestones des Projekts, Aufgabenlisten erstellen, anhand derer der Projektfortschritt gemessen werden kann.

---

# Inhalt

<b>Einführung</b>	<b>11</b>
<hr/>	
<b>Teil I Die Grundlagen</b>	<b>17</b>
<b>1 Large Language Models verstehen</b>	<b>19</b>
1.1 Ihre Entwicklung beschleunigen .....	20
1.2 Eine Einführung in LLMs für Entwickler .....	26
1.3 Wann generative KI genutzt werden sollte und wann nicht .....	27
1.4 Zusammenfassung .....	28
<b>2 Einstieg in die Arbeit mit LLMs</b>	<b>29</b>
2.1 Ein Streifzug durch ChatGPT .....	30
2.1.1 Feinheiten mit GPT-4 erforschen .....	30
2.1.2 GPT-3.5 als Wegweiser .....	36
2.1.3 Auf den Weltmeeren der KI: Von den Küsten von GPT-3.5 zu den Horizonten von GPT-4 .....	40
2.2 Copilot das Steuer überlassen .....	41
2.3 ChatGPT und Copilot im Vergleich .....	44
2.4 Zusammenfassung .....	46
<b>Teil II Die Eingaben</b>	<b>49</b>
<b>3 Software entwerfen mit ChatGPT</b>	<b>51</b>
3.1 Vorstellung unseres Information Technology Asset Management-Systems .....	52
3.2 ChatGPT bitten, uns beim Entwurf unseres Systems zu helfen .....	52
3.3 Die Architektur dokumentieren .....	58
3.4 Zusammenfassung .....	77

<b>4 Software erstellen mit GitHub Copilot</b>	<b>79</b>
4.1 Das Fundament erstellen . . . . .	79
4.1.1 Unser Domain Model beschreiben . . . . .	80
4.1.2 Immutabilität bevorzugen . . . . .	81
4.1.3 Unsere wichtigsten Klassen dekorieren . . . . .	83
4.1.4 Eine Strategie für die Abschreibungen festlegen . . . . .	89
4.2 Muster über Muster . . . . .	92
4.2.1 Ein Besuch bei unseren Abteilungen . . . . .	92
4.2.2 Objekte aus der Fabrik . . . . .	93
4.2.3 Das System mit einem Bauplan versorgen . . . . .	98
4.2.4 Veränderungen beobachten . . . . .	103
4.3 Ports und Adapter miteinander verschalten . . . . .	106
4.3.1 Ein Rückblick auf die hexagonale Architektur . . . . .	106
4.3.2 Die Applikation »antreiben« . . . . .	107
4.3.3 Datenzugriff und dauerhafte Datenspeicherung . . . . .	115
4.3.4 Den Datenzugriff zentralisieren (und externalisieren) . . . . .	119
4.4 Zusammenfassung . . . . .	124
<b>5 Datenverwaltung mit GitHub Copilot und Copilot Chat</b>	<b>125</b>
5.1 Die Daten zusammenstellen . . . . .	125
5.2 Assets mit Apache Kafka in Echtzeit überwachen . . . . .	138
5.3 Analyse, Erfassung und Ortung mit Apache Spark . . . . .	146
5.4 Zusammenfassung . . . . .	151
<b>Teil III Das Feedback</b>	<b>153</b>
<b>6 LLMs nutzen, um Code zu testen, zu bewerten und zu erklären</b>	<b>155</b>
6.1 Dreierlei Tests . . . . .	156
6.1.1 Unit-Testing . . . . .	156
6.1.2 Integrationstests . . . . .	162
6.1.3 Verhaltenstests . . . . .	163
6.2 Qualitätskontrolle . . . . .	167
6.3 Fehler aufspüren . . . . .	172
6.4 Codeabdeckung . . . . .	172

6.5	Code in eine Beschreibung übersetzen .....	174
6.6	Code in eine andere Programmiersprache übersetzen .....	177
6.7	Zusammenfassung .....	182
<b>Teil IV</b>	<b>Hinaus in die Welt</b>	<b>185</b>
<b>7</b>	<b>Infrastructure as Code und Verwaltung von Deployments</b>	<b>187</b>
7.1	Ein Docker-Image anlegen und lokal bereitstellen .....	189
7.2	Infrastructure as Code mit Terraform und Copilot .....	191
7.3	Ein Docker-Image verschieben (auf die harte Tour) .....	195
7.4	Ein Docker-Image verschieben (auf die sanfte Tour) .....	196
7.5	Unsere Applikation auf dem AWS Elastic Kubernetes Service bereitstellen .....	197
7.6	Die Einrichtung einer Continuous-Integration-/Continuous-Deployment-(CI/CD-)Pipeline in GitHub Actions .....	200
7.7	Zusammenfassung .....	203
<b>8</b>	<b>Sichere Applikationsentwicklung mit ChatGPT</b>	<b>205</b>
8.1	Bedrohungen mit ChatGPT modellieren .....	206
8.1.1	Warum das in der heutigen Softwareentwicklung wichtig ist .....	207
8.1.2	Wie ChatGPT beim Threat Modeling helfen kann .....	207
8.1.3	Fallstudie: Bedrohungsmodellierung mit ChatGPT simulieren .....	211
8.2	Untersuchung des Applikationsdesigns und Ermittlung möglicher Schwachstellen .....	218
8.2.1	Designprobleme bewerten .....	218
8.2.2	Häufige Schwachstellen erkennen .....	219
8.3	Anwendung von Best Practices in der Sicherheit .....	221
8.3.1	Eine sicherheitsorientierte Denkweise .....	221
8.3.2	Beständige Sicherheitsüberprüfungen .....	223
8.4	Datenverschlüsselung im Ruhezustand und während der Übertragung .....	227
8.4.1	Die Bedeutung der Datenverschlüsselung .....	227
8.4.2	Datenverschlüsselung im Ruhezustand (»at rest«) .....	228

8.4.3	Datenverschlüsselung während der Übertragung (»in transit«) .....	233
8.5	Zusammenfassung .....	236
<b>9</b>	<b>GPT für unterwegs</b>	<b>237</b>
9.1	Die Grundidee .....	237
9.2	Ihr eigenes LLM hosten .....	238
9.2.1	ChatGPT als Vergleichsmaßstab .....	238
9.2.2	Llama 2 dazu bringen, eine Antwort auszuspucken .....	240
9.2.3	Demokratische Antworten mit GPT-4All .....	252
9.3	Zusammenfassung .....	257
<b>Teil V</b>	<b>Anhänge</b>	<b>259</b>
<b>A</b>	<b>ChatGPT einrichten</b>	<b>261</b>
A.1	Einen ChatGPT-Account anlegen .....	261
A.2	Einen ChatGPT-Account mit einer E-Mail-Adresse anlegen .....	262
<b>B</b>	<b>GitHub Copilot einrichten</b>	<b>267</b>
B.1	Die Copilot-Erweiterung in Visual Code installieren .....	267
B.2	Das Copilot-Plug-in in PyCharm installieren .....	274
B.3	Der erste Flug mit Copilot .....	277
<b>Index</b>		<b>281</b>