README und Anleitung

Businesskontext

CoffeeClub ist ein digitales Backend-System für ein abonnementbasiertes Kaffeemodell, das gezielt auf kleine Cafés ausgerichtet ist. Es soll die Kundenbindung stärken, die Servicequalität verbessern und gleichzeitig den Verwaltungsaufwand für Cafébetreiber:innen reduzieren. Die zentrale Idee: Kund\:innen schließen ein wöchentliches Kaffeeabo ab und erhalten damit ein bestimmtes Kontingent an Kaffees zu einem festen Preis. Dabei können die Abomodelle von jedem Café individuell angepasst werden – z. B. in Bezug auf die Anzahl der Kaffees, den Preis oder die Art der Getränke.
Technisch basiert das System auf einer modularen FastAPI-Architektur mit SQLite als Datenbank. Es wird containerisiert bereitgestellt (z. B. über Docker), sodass jedes Café eine eigene, voneinander unabhängige Instanz betreiben kann. Diese Instanzen laufen isoliert, was bedeutet, dass Probleme wie fehlerhafte Konfigurationen, Serverausfälle oder Datenprobleme nicht das gesamte System betreffen, sondern nur die jeweilige Café-Instanz. So wird die Ausfallsicherheit erhöht und gleichzeitig die Skalierung vereinfacht.
Die Software wird den Cafés als eigenständig betreibbare Anwendung zur Verfügung gestellt – entweder lokal (z. B. auf einem kleinen Server im Café oder auf einem gehosteten Server in der Cloud) oder über ein zentrales Deployment-Modell mit Container-Orchestrierung (z. B. Docker Compose). Optional kann eine benutzerfreundliche Admin-Oberfläche (Web UI oder mobile App) mit dem Backend verbunden werden. Die Bedienung erfolgt dann durch autorisiertes Personal im Café: Hauptadmins können Abo-Modelle erstellen, Mitarbeiter\:innen neue Kunden registrieren oder genutzte Kaffees eintragen. Kund\:innen wiederum nutzen das System über einen digitalen Zugang (z. B. Kundenkonto oder QR-Code), um ihren Abo-Status, ihre Nutzungsstatistik oder offene Kaffees einzusehen.
Damit bietet CoffeeClub eine leichtgewichtige, skalierbare und einfach wartbare Softwarelösung, die sowohl den wirtschaftlichen als auch den betrieblichen Anforderungen kleiner Cafés gerecht wird.

Implementierung

Zusätzlich zu denen im Code vorhanendenen Kommentare, sowie Doc- und Method-Strings, folgt hier eine kleine Übersicht über den Aufbau des Projekts.
Der Aufbau des Repository ist folgendermaßen:

```
.
├── app.db
├── durchfuehrung.http
├── controllers
│   ├── __init__.py
│   ├── abo_controller.py
│   ├── abomodel_controller.py
│   ├── cafe_controller.py
│   ├── customer_controller.py
│   └── employee_controller.py
├── database.py
├── Dockerfile
├── main.py
├── models
│   ├── __init__.py
│   ├── model.py
│   └── utility_crud.py
├── README.md
├── requirements.txt
├── requirements1.txt
├── schemas
│   ├── __init__.py
│   ├── abo.py
│   ├── abomodel.py
│   ├── cafe_abo.py
│   ├── cafe.py
│   ├── customer.py
│   ├── employee.py
│   └── token.py
├── services
│   ├── __init__.py
│   ├── abo_service.py
│   ├── abomodel_service.py
│   ├── auth_service.py
│   ├── cafe_service.py
│   ├── customer_service.py
│   └── employee_service.py
├── tests
│   ├── abo_test.http
│   ├── abomodel_test.http
│   ├── cafes.http
│   ├── customer_test.http
│   └── employee_test.http
└── utils
    ├── __init__.py
    ├── security.py
    └── validation.py
```

Die Architektur des Systems basiert auf einer model-service-controller-Systematik. Konkret bedeutet das, dass im Controller-Modul alle API-Endpunkte definiert sind. Die eigentliche Businesslogik findet hier jedoch nicht statt.
Diese findet man in den services. Hier wird berechnet und Anfragen an die Datenbank gestellt. Werdenn Daten aus der Datenbank benötigt, wird dies über die Dateien in Model gemacht. Hier wurde mit sqlalchemy auch der Aufbau der Datenbank definiert.
In utils finden sich verschiedene Hilfsfunktionen, beispielsweise um JWT-Token zu generieren und verifizieren aber auch um Passwörter zu hashen, sodass diese verschlüsselt gespeichert werden können.

5 Aspekte aus der Vorlesung

1. Authentifizierung

Authentizierung bezeichnet den Vorgang, wie ein Nutzer seine Identität gegenüber dem System bestätigt, z.B. durch Einloggen mit Benutzerdaten. Beim Login wird oft ein Token (z.B. ein JSON Web Token, JWT) ausgestellt, der für die Dauer der Session den Zugang ermöglicht.
Es kann auch als Log-In oder Anmeldung bezeichnet werden.
Im Projekt werden Log Ins für Mitarbeiter, sowie für Kunden genutzt. Dadurch werden ihnen aufgrund ihrer Rolle verschiedene Rechte zugewiesen.
Bei Mitarbeitern kann zwischen Admins und Nichtadmins unterschieden werden, wobei Admins mehr Rechte als Nichtadmins haben.
Nichtadmins können neue Abonnements anlegen und die Kundendaten von Kunden einlesen, um Kaffee auszugeben. Admins haben zudem die Möglichkeit Statistiken einzusehen und Abonnements sowie Abomodelle zu verwalten.

2. Inputvalidierung

Inputvalidierung meint den Vorgang Nutzereingaben systematisch auf deren Korrektheit und Vollständigkeit zu überprüfen, sodass die Datenbankkonsistenz bewahrt wird.
Außerdem können so Sicherheitslücken geschlossen werden, da durch die Möglichkeit von Nutzereingaben auch Angriffsmöglichkeiten entstehen.
Hieraus lässt sich auch die Relevanz für Inputvalidierung im Backend ableiten. Während eine Inputvalidierung im Frontend eine schnelle Rückmeldung ermöglicht und potenziell die Anzahl der Requests an den Server reduziert, ermöglicht die Inputvalidierung im Backend, dass auch Requests, die nicht durch die Benutzeroberfläche gesendet werden auf ihre Richtigkeit überprüft werden.
Im Projekt wurde Inputvalidierung durch Pydantic-Schemas und vorhandenen Datentypen, die durch Pydantic bereitgestellt werden, wie ein EMail-String, oder durch selbst implementierte field validators, realisiert.

3. Containerisierung

Um die Portabilität eines Systems kann es sinnvoll sein, dieses in Containern zu verpacken.Container sind im Grunde genommen kleine voll funktionsfähige Rechenumgebungen, Sodass das System unabhängig vom Betriebssystem und der Umgebung ist, in der es erstellt wurde.
Durch Containerisierung kann so sehr viel Platz gespart und die Effektivität erhöht werden, im Gegensatz zu einem herkömmlichen Server oder einer virtuellen Maschine.
Im Projekt wurde Docker, ein Open-Source-Tool, dass häufig zur Containerisierung eingesetzt wird, genutzt.

Ein Dockerfile ermöglicht es in diesem Projekt, das System schnell und unkompliziert zum Laufen zu bringen. Um ein Dockerimage zu bauen und auszuführen, ist es wichtig Docker geöffnet zu haben.

4. Wiederkehrende Aufgaben

Viele Systeme haben Aufgaben die nach regelmäßigen Zeiträumen ausgeführt werden sollten und zeitlich planbar sind. Diese kann man so implementieren, dass sie beispielsweise zu Tageszeiten durchgeführt werden, wenn sonst nicht viel Rechenleistung benötigt wird. Beispiele für wiederkehrende Aufgaben, die in diesem Projekt anfallen könnten sind:

- das Erstellen von wöchentlichen oder monatlichen Reports, die das Konsumverhalten der Aboinhaber analysieren, sowohl für die Cafés als auch für die Aboinhaber.
- Das Versenden von Erinnerungsmails
- Überprüfung von Abos und Zahlungen, sodass Abos auf inaktiv gesetzt werden können.

Im Projekt wurde bisher kein Mechanismus für wiederkehrende Aufgaben implementiert, es existiert jedoch ein Dummy mit Endpunkt, der einen Test weekly report zurück gibt.

5. Sessionhandling

Sessionhandling sorgt für eine sichere und nutzerfreundliche Verwaltung von Benutzerzuständen. Hier wird mit Token gearbeitet, welche Informationen über die Rolle und Berechtigungen erhält. Dieses wird bei jeder Anfrage überprüft.
Dies bietet eine höhere Sicherheit, da Tokens digital signiert sind und so Manipulationen erkennen und so gefährdende Abfragen abzufangen.
Zudem können Tokens zeitliche begrenzt sein. So ist es für Angreifer nicht unbegrenzt möglich mit einem abgefangenen Token auf sensible Daten zuzugreifen.
Im Projekt wurde die Vergabe von Tokens im API-Endpunkt für den Mitarbeiter-Login implementiert. Bei erfolgreicher Authentifizierung wird ein Token zurückgegeben.
Aus Einfachheitsgründen sind viele Endpunkte nicht protected, sie erfordern also kein Token, das verifiziert wird.
Die Route, die beispielhaft die Überprüfung des Tokens beinhaltet, ist leider nicht funktionsfähig.

Weitere Aspekte

- **ETL:** Extract, Transform, Load bezeichnet den Vorgang in dem Daten von einer Darstellungsform ausgelesen, verarbeitet und anderweitig weiterverarbeitet werden. Im Projekt werden beispielsweise Datensätze aus der Datenbank an verschiedenen Stellen als JSON-Dateien ausgegeben. Daher ist dieser Aspekt relevant für dieses Projekt.
- **Verbindungstechnologien:** Um die Kommunikation zwischen Frontend und Backend wird eine RESTAPI genutzt. Dies ist besonders relevant für dieses Projekts.
- **Schnittstellen:** Schnittstellen zu anderen Systemenn können genutzt werden um weitere Funktionen anzubinden, die nicht selbst implementiert wurden. Im Projekt kann dies genutzt werden, um beispielsweise Zahlungsdienstleister wie PayPal einzubinden. So können die Zahlungen der Abonnements darüber abgewickelt werden.
- **Skalierbarkeit:** Da das System eigens für jedes Kaffee oder Kette gehostet wird, ist die Skalierbarkeit nicht allzu relevant, wenn es um die Hardware geht. Die einzelenen Komponenten des Backends lassen sich beliebig erweitern, da es keine Beschränkungen für die Anzahl der Mitarbeiter, Abomodelle und Cafes gibt. Dies könnte man jedoch im Verkauf beschränlen und so verschiedene Lizenzmodelle nutzen.

Durchführungsanleitung

Voraussetzung für die Durchführung der Tests:
Docker, REST Client Extension von Huachao Mao in Visual Studio Code

1. Starte den Container mit `docker-compose up --build`
2. Führe die Tests in der Datei `` `Durchfuehrung.http` `` aus.