ipso! Bildung seit 1876

Dokumentation Skybooker

Atici Beren, David Mesic

ipso! Bildung seit 1876

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

Rahmen dieser Projektarbeit haben wir den Auftrag erhalten, eine verteilte Webanwendung mit dem Namen SkyBooker zu entwickeln. Die Idee dahinter ist, eine Plattform zu bauen, mit der man weltweit Flüge buchen und verwalten kann – ähnlich wie bekannte Anbieter, aber als eigene Lösung. Dabei soll SkyBooker in der Lage sein, Flüge von verschiedenen Fluggesellschaften zentral zu verwalten und den Passagieren eine einfache Möglichkeit zu bieten, ihre Flugreisen online zu buchen.
Damit das Ganze auch in der Praxis skalierbar und zuverlässig funktioniert, setzen wir auf eine Microservice-Architektur. So bleibt die Anwendung flexibel, lässt sich bei Bedarf einfach erweitern und ist auch bei wachsender Nutzerzahl stabil betreibbar. Die einzelnen Services – z. B. für Fluginformationen, Buchungen oder die Benutzerverwaltung – laufen unabhängig voneinander und nutzen verschiedene Datenbanken, die auf ihren Einsatzzweck abgestimmt sind.

Im Zentrum der Plattform stehen zwei Hauptfunktionen:

- Die Flugplanverwaltung: Hier werden alle Flüge mit Daten wie Flugnummer, Airline, Route, Abflug- und Ankunftszeit sowie verfügbaren Sitzplätzen gespeichert.
- Die Flugbuchung: Hier können Passagiere ihre Buchungen tätigen, ihre Daten eingeben und festlegen, wie viele Tickets sie benötigen.

Diese beiden Kernbereiche bilden die Grundlage für die weitere Entwicklung und Umsetzung des Projekts. Sie stellen sicher, dass die Anwendung sowohl für die Anbieter (Fluggesellschaften) als auch für die Endnutzer (Passagiere) einen echten Mehrwert bietet.

2 Projektplanung

2.1 Projektauftrag

Im Modul 321 «Verteilte Systeme programmieren» war es unsere Aufgabe, eine komplette Webanwendung zu entwickeln, die aus mehreren Microservices besteht. Wir haben uns für das Projekt **SkyBooker** entschieden – eine Plattform, mit der man Flüge von verschiedenen Airlines buchen und verwalten kann.

Die Anwendung besteht aus drei wichtigen Bereichen:
-Ein FlightService, der alle Fluginformationen wie Flugnummer, Abflugzeit oder verfügbare Sitzplätze verwaltet.

-Ein BookingService, mit dem Passagiere ihre Flüge buchen können.

-Ein AuthService, über den sich Nutzer registrieren und einloggen – abgesichert mit einem JWT-Token.

Damit alles sauber und flexibel funktioniert, haben wir die Lösung als Microservices aufgebaut und die einzelnen Dienste in Docker-Container verpackt. Zusätzlich mussten wir für jeden Service Tests schreiben, eine API-Dokumentation mit Swagger erstellen und alle Endpunkte mit Postman testen. Am Ende präsentieren wir das Projekt in einer kurzen Live-Demo.

2.2 Zeitplanung

Damit wir den Überblick behalten und nicht alles auf den letzten Tag machen, haben wir das Projekt auf vier Tage à je ca. 13.5h Stunden aufgeteilt. Wir haben zuerst überlegt, was alles gemacht werden muss, und die Arbeitsschritte danach in logischer Reihenfolge geplant.

Arbeitspakete	verant-wortlich	geplanter Zeitaufwand	effektiver Zeitaufwand	05.05.2025	06.05.2025	07.05.2025	08.05.2025
Projektsetup (Git, Solution, Services anlegen)	**BE, DA**	1	1.5				
FlightService Implementierung	DA	1	1				
BookingService Implementierung	BE	1	2				
AuthService Implementierung + JWT	DA	2	1				
Datenbankmodellierung (3 DBs)	**BE, DA**	1	1				
Testing / Unit Tests	BE	**2**	1.5				
Dockerisierung aller Services	**BE, DA**	1.5	0.5				
Swagger / OpenAPI Doku	**BE, DA**	0.5	0.5				
Postman Collection erstellen + testen	DA	0.5	0.5				
Optionale Erweiterungen (2 Stück) Gateway + Rabbitmq	BE	3	3				
Dokumentation schreiben	BE	4	4				
Projekt beenden und Fazit erstellen	**BE, DA**	1	1				
		14.5	13.5				

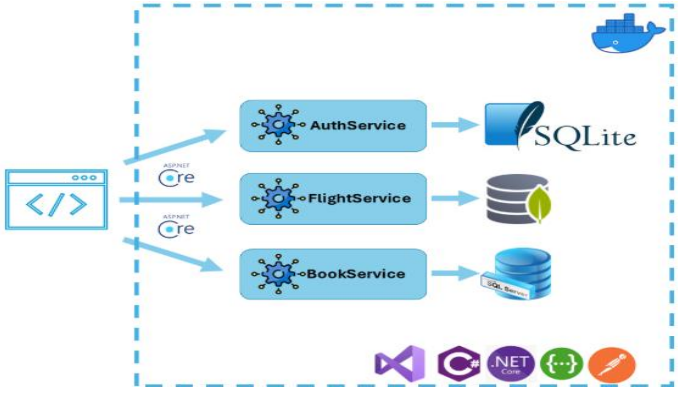
Diese Aufteilung hat uns geholfen, Schritt für Schritt vorzugehen, ohne den Fokus zu verlieren.

3 Architektur und Technologien

3.1 Übersicht der Microservices

Die Anwendung SkyBooker ist modular aufgebaut und basiert auf einer Microservices-Architektur. Jeder Dienst übernimmt eine klar definierte Aufgabe und kommuniziert über HTTP-Requests mit den anderen Services. Ziel dieser Struktur ist eine hohe Flexibilität, Skalierbarkeit und eine gute Wartbarkeit der Anwendung.

3.1.1 Abbildung 1: Grundarchitektur ohne Erweiterungen

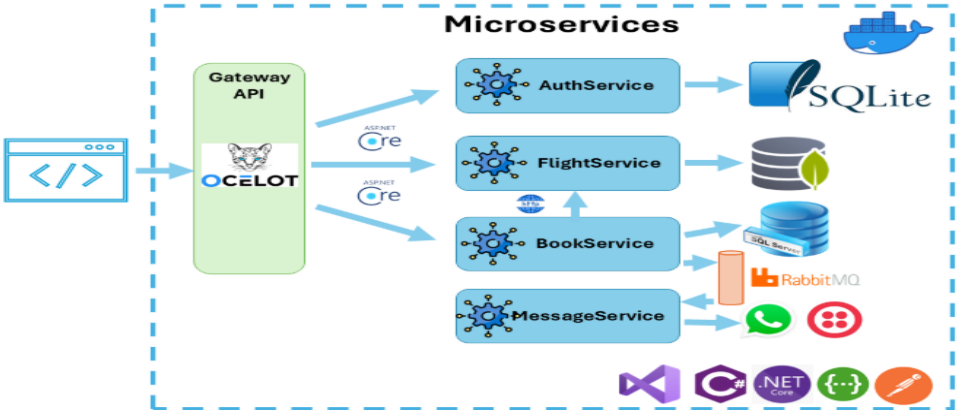


In dieser Basisversion der Architektur existieren drei Microservices:

-AuthService: Zuständig für Registrierung, Login und Verwaltung von Benutzern. Die Benutzerdaten werden in einer SQLite-Datenbank gespeichert.
-FlightService: Verwaltet Fluginformationen wie Flugnummern, Routen, Zeiten und Verfügbarkeiten. Die Datenhaltung erfolgt über eine MongoDB.
-BookingService: Erfasst und speichert Buchungen der Passagiere. Hier kommt eine MS SQL Server-Datenbank zum Einsatz.

Alle Services wurden mit ASP.NET Core in C# entwickelt und in Docker-Containern bereitgestellt.

3.1.2 Abbildung 2: Erweiterte Architektur mit Zusatzanforderungen



In der erweiterten Version wurden zusätzliche Funktionen und Technologien eingebunden, um die Plattform robuster und praxisnäher zu gestalten:

-Gateway API (Ocelot): Ein zentrales API-Gateway dient als einziger Einstiegspunkt für den Client und verteilt die Anfragen an die jeweiligen Microservices.
-MessageService: Ein separater Dienst zur Verarbeitung von Nachrichten und Benachrichtigungen, z. B. über WhatsApp oder Twilio.
-RabbitMQ: Ermöglicht die asynchrone Kommunikation zwischen BookService und FlightService, etwa zur Validierung von Flugnummern oder zur Vermeidung von Überbuchungen.

3.2 Warum Microservices?

Wir haben uns bewusst für eine Microservice-Architektur entschieden, weil sie viele praktische Vorteile bietet. Jeder Service – zum Beispiel für Benutzer, Flüge oder Buchungen – ist ein eigenständiger Teil der Anwendung. Das bedeutet: Man kann einzelne Teile unabhängig voneinander entwickeln, testen oder aktualisieren, ohne dass gleich das ganze System betroffen ist.

Ausserdem können die Services bei Bedarf einzeln skaliert werden – zum Beispiel, wenn besonders viele Buchungen gemacht werden. Jeder Service kann auch mit der jeweils passenden Technologie und Datenbank arbeiten. Für ein System wie SkyBooker, das aus mehreren klar abgegrenzten Bereichen besteht, ist dieser modulare Aufbau ideal.

3.3 Verwendete Technologien

Technologie	Zweck / Einsatzbereich
ASP.NET Core	Framework für die Entwicklung der Web-APIs
C#	Programmiersprache
MongoDB	NoSQL-Datenbank für Fluginformationen
MS SQL Server	Relationale DB für Flugbuchungen
SQLite	Leichte Datenbanklösung für Authentifizierung
Docker	Containerisierung und Deployment der Services
Docker Compose	Verwaltung mehrerer Container
Swagger (OpenAPI)	Dokumentation der Web-APIs
Postman	Testen der HTTP-Endpunkte
JWT (Json Web Token)	Authentifizierung und Zugriffskontrolle
RabbitMQ (optional)	Nachrichtenaustausch zwischen Services
Ocelot (optional)	API-Gateway zur Steuerung der Kommunikation
Serilog (optional)	Logging & Fehlerprotokollierung

4 Microservices im Detail

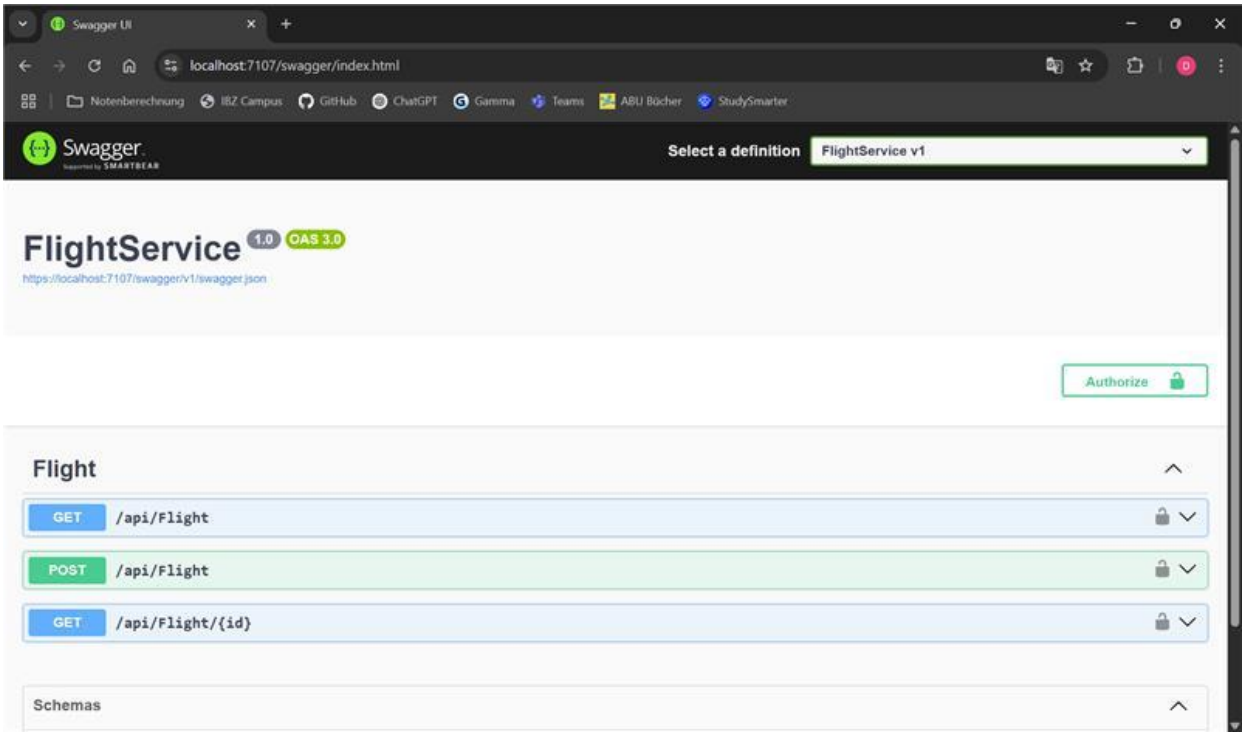
4.1 FlightService

Der FlightService ist dafür zuständig, alle Fluginformationen zu verwalten. Dazu gehören z. B. Flugnummern, Abflug- und Zielorte, Zeiten und wie viele Plätze noch frei sind. Die Daten werden in einer MongoDB gespeichert, weil das gut funktioniert für flexible und dynamische Datenstrukturen.

Was der Service kann:

- Neue Flüge erfassen
- Alle verfügbaren Flüge anzeigen
- Einzelne Flüge nach ID abrufen
- Flüge bearbeiten oder löschen

Endpunkte:

- POST /api/flight – Einen neuen Flug anlegen
- GET /api/flight – Liste aller Flüge
- GET /api/flight/{id} – Flug mit bestimmter ID anzeigen
- 

4.2 BookingService

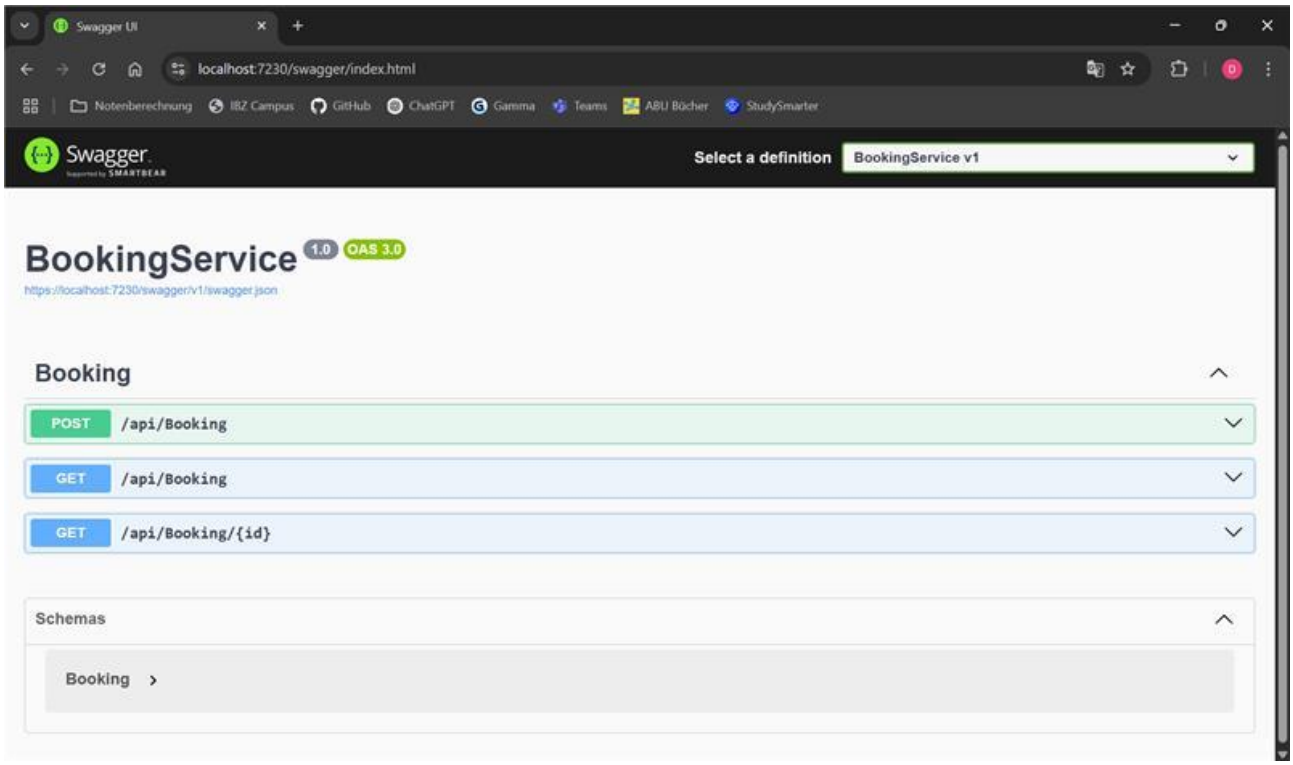
Der BookingService übernimmt alles rund ums Buchen von Flügen. Wenn ein Nutzer ein Ticket reservieren möchte, passiert das über diesen Service. Die Buchungsdaten werden in einer SQL Server-Datenbank gespeichert, weil sie sich gut für strukturierte Daten eignet.

Was der Service kann:

- Neue Buchungen erstellen
- Alle bisherigen Buchungen anzeigen
- Einzelne Buchung anschauen

Endpunkte:

- POST /api/booking – Neue Buchung erfassen
- GET /api/booking – Alle Buchungen abrufen
- GET /api/booking/{id} – Details einer bestimmten Buchung



4.3 AuthService

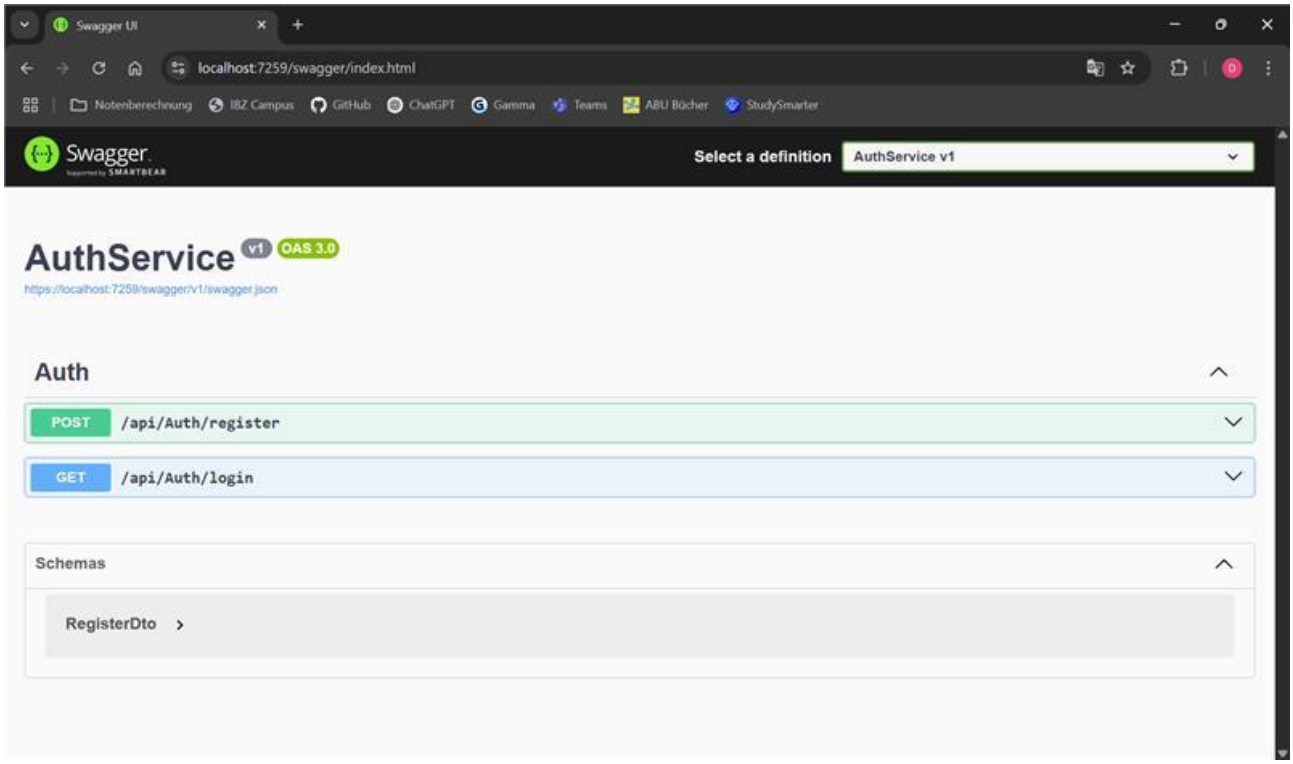
Der AuthService kümmert sich um die Benutzer – also Registrierung und Login. Sobald sich jemand einloggt, bekommt er einen JWT-Token, mit dem er dann auf geschützte Bereiche der Anwendung zugreifen kann. Die Benutzerdaten werden in einer SQLite-Datenbank gespeichert.

Was der Service kann:

- Neue Benutzer registrieren
- Login mit E-Mail und Passwort
- Token generieren für den Zugriff auf andere Services

Endpunkte:

- POST /api/register – Benutzer registrieren
- POST /api/login – Login durchführen und Token erhalten



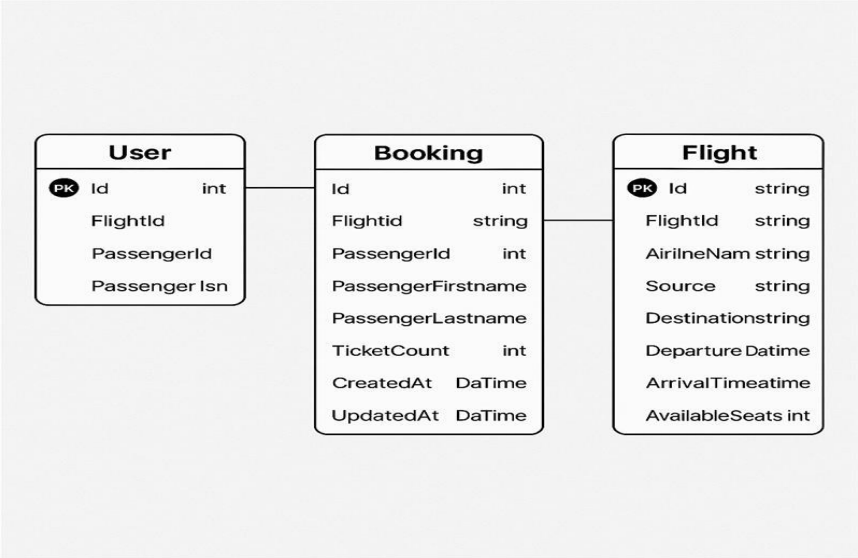
5 Datenbankdesign

5.1 ER-Diagramm

Das Unten gezeigte ER-Diagramm stellt die Datenstruktur der SkyBooker-Anwendung vereinfacht dar. Es zeigt die drei zentralen Tabellen bzw. Collections: User, Booking und Flight.

- In der User-Tabelle werden die registrierten Benutzer gespeichert.
- Die Booking-Tabelle dokumentiert alle Flugbuchungen und enthält die Verbindung zwischen Benutzern und Flügen.
- Die Flight-Tabelle enthält sämtliche Fluginformationen wie Flugnummer, Route und verfügbare Sitzplätze.

Zwischen den Tabellen bestehen logische Verknüpfungen über PassengerId und FlightId, wodurch Buchungen eindeutig einem Nutzer und einem Flug zugeordnet werden können. Das Datenmodell ist bewusst schlank gehalten und auf die wichtigsten Informationen fokussiert.



6 Testing & Qualitätssicherung

Damit unsere Anwendung zuverlässig funktioniert, haben wir uns auch mit dem Thema Testing beschäftigt. Ziel war es, die wichtigsten Funktionen in den einzelnen Microservices zu prüfen – also z. B. ob Flüge richtig gespeichert werden, ob Buchungen korrekt ablaufen oder ob sich ein Benutzer erfolgreich einloggen kann.

6.1 Was wir testen wollten

- Im FlightService: Ob man einen neuen Flug korrekt hinzufügen und abrufen kann.
- Im BookingService: Ob die Buchung nur funktioniert, wenn z. B. genug Sitzplätze verfügbar sind.
- Im AuthService: Ob der Login klappt und ein gültiger JWT-Token ausgegeben wird.

6.2 Aktueller Stand

Da wir gegen Ende des Projekts zeitlich unter Druck standen, konnten wir leider nur wenige Unit Tests umsetzen. Die Testprojekte sind zwar vorbereitet, aber viele Methoden fehlen noch. Einige Funktionen haben wir stattdessen manuell über Postman getestet, um sicherzustellen, dass die wichtigsten Abläufe wie Flugerfassung und Buchung funktionieren. Die Tests lassen sich später problemlos ergänzen.

7 Dockerisierung

Die Dockerisierung haben wir gegen Ende des Projekts umgesetzt, da wir zuerst die Grundfunktionen der Microservices fertigstellen wollten. Ziel war es, alle Services in eigenen Docker-Containern laufen zu lassen, um die Anwendung einfacher starten und verteilen zu können – unabhängig vom Betriebssystem oder der Umgebung.

7.1 Aufbau der Dockerfiles

Für jeden Microservice wurde ein eigenes Dockerfile erstellt. Darin wird das Projekt gebaut, alle Abhängigkeiten installiert und die Anwendung als Container ausführbar gemacht. Wir haben das offizielle .NET SDK-Image verwendet, um den Build-Prozess durchzuführen.

7.2 Docker-Compose Datei

Mit einer gemeinsamen docker-compose.yml-Datei konnten wir alle Microservices gleichzeitig starten. Die Datei enthält:

-die drei Hauptservices (FlightService, BookingService, AuthService)

-die benötigten Datenbanken (MongoDB, SQL Server, SQLite)

7.3 Containerstruktur

Hier wird unsere Docker Containerstruktur abgebildet:

Name	Container ID	Image	Port(s)	CPU (%)	Last sta	Actions
sqlserver-1	88d622cf1507	mssql/serv	1433:1433	2.73%	2 minute	
mongo-1	48a4c0f2b9fc	mongo:late	27017:27017	1.03%	2 minute	
rabbitmq-1	7c43145bcbfa	rabbitmq:3-	15672:15672 Show all ports (2)	0.6%	2 minute	
flightservice-	de035ab12ba4	skybooker-	8080:8080	0.02%	2 minute	
messageserv	67540cd83ff4	skybooker-	8100:80	1.7%	2 minute	
bookingservi	5ee3a57572b4	skybooker-	8090:8090	0%	2 minute	
authservice-1	c8594fb79cbe	skybooker-	8070:8070	0.01%	2 minute	
apigateway-1	1f7a0b73f3b2	skybooker-	5000:5000	0.02%	2 minute	

8 Herausforderungen & Lösungen

Eine der grössten Herausforderungen in diesem Projekt war definitiv die Zeitplanung. Wir haben uns zu Beginn zu wenig Zeit für einzelne Arbeitsschritte eingeplant vor allem für Testing, Dockerisierung und optionale Erweiterungen. Dadurch gerieten wir gegen Ende des Projekts etwas unter Druck.

Trotzdem haben wir unser Bestes gegeben und die wichtigsten Anforderungen erfolgreich umgesetzt. Besonders beim Zusammenspiel der Microservices und bei der Containerisierung mussten wir improvisieren und flexibel reagieren. Auch wenn nicht alles perfekt lief, haben wir viel gelernt sowohl technisch als auch in der Zusammenarbeit und Organisation.

9 Fazit & Lessons Learned

Das Projekt SkyBooker war für uns eine spannende, aber auch herausfordernde Erfahrung. Im Vergleich zu früheren Projekten war dieses deutlich umfangreicher sowohl technisch als auch organisatorisch. Wir konnten viele neue Dinge lernen, vor allem im Umgang mit Microservices, Docker, JWT-Authentifizierung und verschiedenen Datenbanken.

Was gut funktioniert hat, war die grundlegende Umsetzung der Services und die Kommunikation zwischen den Komponenten. Weniger gut lief unsere Zeitplanung – wir haben gemerkt, dass wir früher hätten starten und unsere Aufgaben besser aufteilen sollen. Gerade Testing, Dockerisierung und optionale Features kamen dadurch etwas zu kurz.

Für die Zukunft nehmen wir mit: Mehr Zeit einplanen, früher beginnen und die Arbeitsschritte klarer strukturieren. So hätten wir das volle Potenzial des Projekts noch besser ausschöpfen können. Trotzdem sind wir mit dem Resultat zufrieden – und vor allem stolz auf das, was wir technisch gelernt und umgesetzt haben.