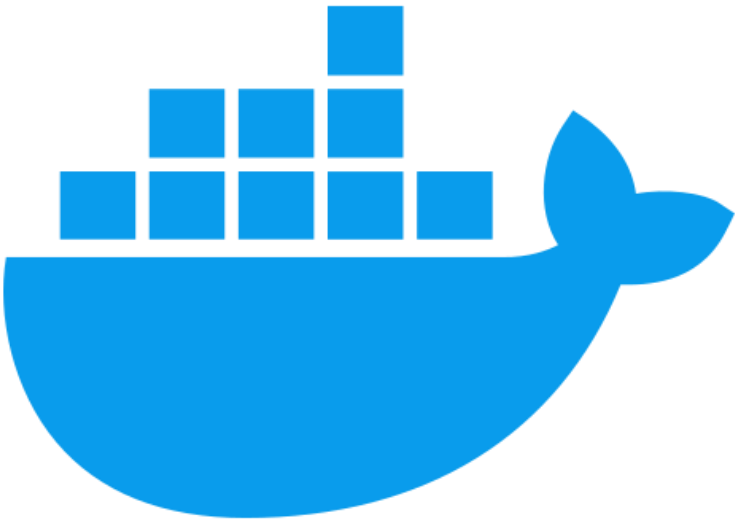
Modul 321 Skybooker Projektarbeit



Von Tunahan Keser und Yannick Frei

Abgabedatum 08.05.2025

Inhalt

1. Informieren

1.1 Ausgangslage

Die Plattform *SkyBooker* soll eine globale Flugbuchungs- und Ticketreservierungslösung darstellen. Ziel ist es, Passagieren zu ermöglichen, Flüge von unterschiedlichen Fluggesellschaften zu buchen und diese Buchungen selbst zu verwalten. Um eine hohe Skalierbarkeit und Flexibilität zu erreichen, basiert die Architektur auf Microservices. Jeder Microservice ist zuständig für einen spezifischen Anwendungsbereich und kann unabhängig betrieben und entwickelt werden.

1.2 Zielsetzung

Ziel der Projektarbeit ist die vollständige Umsetzung der Anwendung *SkyBooker* unter Einhaltung der vorgegebenen Anforderungen. Dazu gehören insbesondere:

- Erstellung von drei separaten Microservices für Fluginformationen, Buchungen und Benutzerverwaltung
- Anbindung an jeweils geeignete Datenbanksysteme (MongoDB, MS SQL Server, SQLite)
- Sicherung der APIs mittels JWT-Authentifizierung
- Bereitstellung in Docker-Containern
- Ausführliche API-Dokumentation mit Swagger
- Testung mit Postman und automatisierten Unit Tests
- Versionierung und Nachverfolgbarkeit über Git

1.3 Hilfsmittel und Rahmenbedingungen

- Entwicklungsumgebung: Visual Studio 2022, VS Code
- Programmiersprache: C# (.NET 8)
- Tools: Docker, Git, Swagger, Postman
- Datenbanken: MongoDB, SQL Server, SQLite
- Zusätzliche Libraries: Serilog, FluentValidation, Entity Framework Core, MongoDB.Driver

2. Planen

2.1 Projektplanung (Gantt-Diagramm)

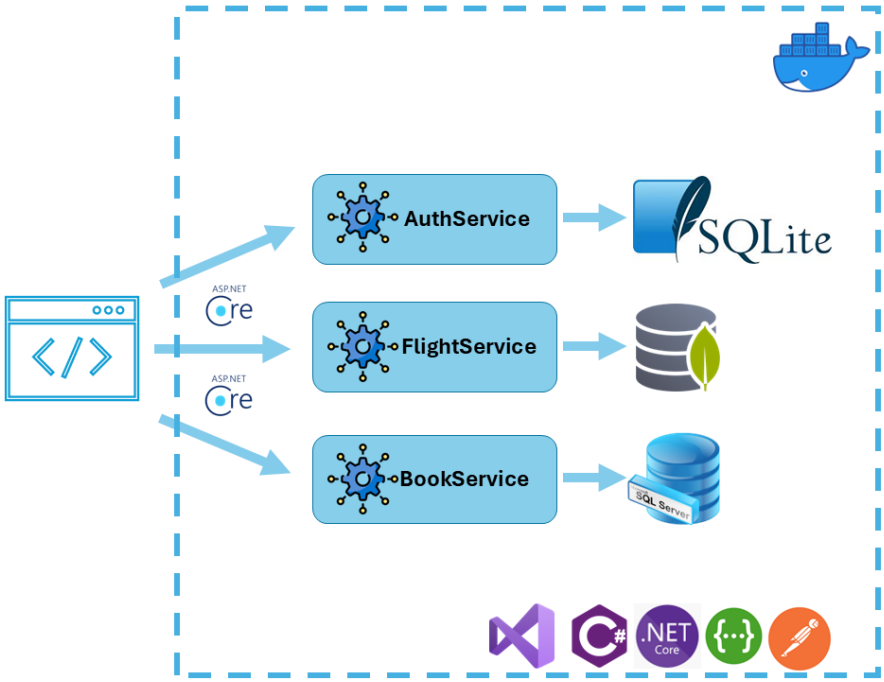
Arbeitspaket	Verantwortlich	Stunden		2.05.2025				3.05.2025				4,52025				5.05.2025				6.05.2025				7.05.2025				8.05.2025			
Informieren		Soll	Ist	17:00	18:00	19:00	20:00	17:00	18:00	19:00	20:00	17:00	18:00	19:00	20:00	17:00	18:00	19:00	20:00	17:00	18:00	19:00	20:00	17:00	18:00	19:00	20:00	17:00	18:00	19:00	20:00
Zieldefinition	beide	0,5	1																												
Anforderungsanalyse	beide	1	2																												
Technologie- und Toolwahl	beide	0,5	1																												
Planen																															
Datenbankmodellierung	Tunahan	1	2																												
Architekturentwurf Microservices	Yannick	1	2																												
Entscheiden																															
API-Endpunkte definieren	beide	1	1																												
Sicherheitskonzept JWT	beide	1	1																												
Realisieren																															
FlightService implementieren	Tunahan	2	3																												
BookingService implementieren	Yannick	2	3																												
AuthService implementieren	Beide	2	2																												
Dockerisierung und Compose	Beide	3	3																												
Swagger-Doku erstellen	Beide	1,5	2																												
Kontrollieren																															
Testing und Debugging	Beide	2	3																												
Unit Tests schreiben	Beide	2	2,5																												
Auswerten																															
Präsentation und Live Demo	Beide	0,5	0,5																												
Dokumentation	Beide	3	4																												
Total		23,5	33																												

Beide	
Tunahan	
Yannick	

3. Entscheiden

3.1 Architekturentscheidungen

- **Microservices:** Ermöglichen unabhängige Entwicklung, Skalierung und Deployment
- **MongoDB für FlightService:** Ideal für schemalose Daten und flexible Flugpläne
- **MS SQL Server für BookingService:** Robuste Transaktionen und Integrität bei relationalen Daten
- **SQLite für AuthService:** Leichtgewichtig, ausreichend für Benutzerdaten



3.2 Technische Entscheidungen

- **JWT:** Für sichere Authentifizierung und rollenbasierten Zugriff
- **Serilog:** Zur Protokollierung wichtiger Systemereignisse
- **Swagger:** Für öffentliche API-Dokumentation
- **HTTP-Kommunikation zwischen Services:** Validierung der Flugnummern

3.3 Erweiterungen

- **AO1:** Serilog für Logging in allen Microservices integriert
- **AO4:** HTTP-Kommunikation zwischen Flight- und BookingService zur Validierung von Flugnummern

4. Realisieren

4.1 FlightService (MongoDB)

- Endpunkte:
  - POST /api/flight
  - GET /api/flight
  - GET /api/flight/{id}
- Felder: flightId, airlineName, source, destination, departure_time, arrival_time, available_seats
- Datenbank: MongoDB Collection "flights"

4.2 BookingService (MS SQL Server)

- Endpunkte:
  - POST /api/booking
  - GET /api/booking
  - GET /api/booking/{id}
- Validierung: Anzahl Tickets darf Verfügbarkeit nicht übersteigen
- Kommunikation mit FlightService zur Flugnummerprüfung

4.3 AuthService (SQLite)

- Endpunkte:
  - POST /api/register
  - POST /api/login
- JWT-Ausgabe bei erfolgreichem Login
- Passwortverschlüsselung mit BCrypt

4.4 Sicherheit

- Alle Endpunkte gesichert mit JWT (außer Login/Registrierung)
- Zugriffskontrolle in Middleware integriert

4.5 Containerisierung mit Docker

- Jeder Service in eigenem Container
- Dockerfiles erstellt, Docker Compose für Orchestrierung

4.6 Testing

- Unit Tests: Getrennt je Service implementiert (z.B. xUnit)
- Integrationstests via Postman

5. Kontrollieren

5.1 Erreichung der Projektziele

Alle gesetzten Anforderungen konnten erfolgreich umgesetzt werden. Die drei Microservices wurden vollständig entwickelt und korrekt an die jeweils passende Datenbank angebunden. Die JWT-Authentifizierung funktioniert zuverlässig und schützt die API-Endpunkte angemessen. Auch die Dockerisierung der Services verlief erfolgreich, sodass alle Komponenten containerisiert ausgeführt werden können. Die APIs wurden mit Swagger dokumentiert und durch Integrationstests mit Postman sowie Unit Tests überprüft. Die zusätzlichen Anforderungen, wie Logging mit Serilog und die HTTP-Kommunikation zwischen den Services, wurden ebenfalls realisiert. Das Projekt wurde über ein Git-Repository versioniert und dokumentiert.

5.2 Tests

- Alle API-Endpunkte getestet
- Fehlerbehandlung bei ungültigen Eingaben erfolgreich getestet
- JWT-Token wird korrekt geprüft

6. Auswerten

6.1 Fazit

Das Projekt *SkyBooker* konnte nicht vollständig abgeschlossen werden. Der Hauptgrund war, dass wir zu spät mit der Umsetzung begonnen und den Aufwand sowie die Komplexität des Projekts unterschätzt haben. Zwar wurden grundlegende Komponenten wie einzelne Microservices und API-Endpunkte umgesetzt, jedoch fehlte die Zeit für vollständige Integration,

Sicherheit und Testing. Trotz des unvollständigen Ergebnisses wurde eine solide Basis geschaffen, auf der später aufgebaut werden kann.

6.2 Lessons Learned

- Früher Projektstart ist entscheidend , Verzögerungen wirken sich stark auf die Umsetzbarkeit aus.
- Die Komplexität von Microservice-Architekturen sollte nicht unterschätzt werden.
- Planung und realistische Einschätzung des Umfangs sind zentrale Erfolgsfaktoren.
- Auch aus unvollständigen Projekten lassen sich wertvolle technische und organisatorische Erfahrungen gewinnen – besonders im Bereich .NET, Docker, API-Design und Teamarbeit.