Qualitätsbericht zum Projekt SE_Platzverwaltung

1. Refactoring-Zusammenfassung

Im Verlauf der Entwicklung wurden mehrere Refactorings vorgenommen, um die Codequalität zu steigern und die Wartbarkeit zu verbessern:

- Trennung der Verantwortlichkeiten (SRP): Controller wie BookingController, BuildingController usw. sind thematisch sauber getrennt.
- Entfernung redundanter Logik: Getter/Setter wurden sauber gekapselt, Validierungslogik ist konsistent gehalten.
- Zentralisierung der Konfiguration: Nutzung von appsettings.json und appsettings.Development.json für Umgebungsvariablen.
- Dependency Injection: Controller und Dienste wie DatabaseHandler werden sauber über den DI-Container verwaltet.
- REST-konforme API-Designs: Klare, sprechende Endpunkte wie api/bookings, api/desks, api/buildings.

Konkretes Beispiel (aus dem Review):

- Im BuildingController wurden Refactorings durchgeführt, um Methodennamen wie Index() in klarere Bezeichnungen wie GetAllBuildings() umzuwandeln und die Fehlerbehandlung (z. B. bei ungültigen IDs) zu verbessern.

———

2. Technical Review – Meetingbericht

Datum: 26. Mai 2025
Thema: Review des BuildingController
Teilnehmende: Team SE_Platzverwaltung + Philip (externe Perspektive aus dem Team Online-Shop)

Ablauf:

Ein gemeinsamer Walkthrough des Codes, bei dem der BuildingController detailliert analysiert wurde. Ziel war die Verbesserung der Wartbarkeit, Codequalität und Fehlerbehandlung.

Erkenntnisse:

Stärken:

- Lesbare, strukturierte Umsetzung
- Funktionaler Zugriff auf Gebäude- und Etagendaten

Verbesserungspotenzial:

- Fehlerbehandlung unvollständig: bei ungültigen IDs soll NotFound() zurückgegeben werden
- Methodennamen wie Index() sind nicht REST-konform

- Kommentare und Dokumentation fehlen größtenteils
- Validierung von Eingaben ist bislang rudimentär

Beschlossene Maßnahmen:
- Umbenennung von Methoden zur besseren Lesbarkeit
- Einführung systematischer Fehlerbehandlung
- Nachpflegen technischer Kommentare
- Überprüfung weiterer Controller auf ähnliche Probleme
- Optional: zentrale Validierung über Middleware oder FluentValidation

Fazit:

Das Review war ein wichtiger Meilenstein zur Sicherstellung der Softwarequalität. Besonders hilfreich war die externe Perspektive, durch die neue Impulse gesetzt wurden.

———

3. Test Report

Automatisierte Tests:
Im Repository sind derzeit keine automatisierten Tests vorhanden (z. B. mit xUnit, NUnit oder ähnlichen Frameworks).

Manuelle Tests:
- API-Endpoints wurden manuell mit Tools wie Postman getestet.
- CRUD-Operationen für Buchungen, Gebäude, Etagen, Tische und Benutzer wurden manuell evaluiert.

Empfehlung:
- Einführung eines Unit-Test-Frameworks für Models & Business-Logik
- Integrationstests für API-Endpunkte (z. B. über WebApplicationFactory)
- CI-Anbindung zur automatisierten Testausführung

———

4. Softwaremetriken

Allgemeine Struktur:
- Backend: ASP.NET Core Projekt mit typischer MVC-Struktur
- Frontend: Vue.js + TailwindCSS (in diesem ZIP nicht vollständig enthalten)
- Datenbank: SQLite (Datei database.db)

Quantitative Metriken (geschätzt):

Kategorie	Anzahl / Umfang
LOC (C# Backend)	ca. 1500
Controller-Klassen	6 (Auth, Booking, Desk, …)

Model-Klassen	5 (Building, Floor, Bookings, …)
Config-Dateien	2 (appsettings.json, .Development)

Qualitative Einschätzung:

- Namenskonventionen konsistent
- SRP & REST-Prinzipien überwiegend eingehalten
- Kein Einsatz von Linter, Analysetools oder CI-Checks sichtbar
- Keine dedizierte Fehler-Logging-Infrastruktur