Preiswürfel Backend - Code Review Zusammenfassung

Gesamtbewertung: 6.5/10 (Gut mit kritischen Punkten)

Insgesamt ist das ein solides Projekt mit modernen .NET-Praktiken, aber es gibt einige wichtige Bereiche, die Aufmerksamkeit benötigen.

🏗️ Architektur: 8/10

- **Stärken**: Saubere Schichtarchitektur, ordentliche Dependency Injection, interface-basiertes Design
- **Verbesserungspotenzial**: Einige enge Kopplungen in der Geschäftslogik

Das ist wirklich gut durchdacht! Die Struktur mit den verschiedenen Layern (Service, Business, Data) ist professionell umgesetzt.

🔒 Sicherheit: 4/10 ⚠️ **KRITISCH**

- **Hauptrisiken**: SQL-Injection-Schwachstellen, exponierte Verbindungsstrings
- **Dringend nötig**: Parametrisierte Abfragen, sichere Konfigurationsverwaltung
- **Positiv**: JWT + Keycloak Authentifizierung ist korrekt implementiert

Das ist der Bereich, der wirklich sofortige Aufmerksamkeit braucht! Die Sicherheitslücken sind ernst und sollten vor jedem Produktionseinsatz behoben werden.

📊 Code-Qualität: 7/10

- **Stärken**: Modernes .NET 8.0, konsistente Patterns, gute Dokumentation
- **Verbesserungspunkte**: Statische Felder in Controllern, gemischte deutsche/englische Begriffe
- **Standards**: Folgt generell den C#-Konventionen

Der Code ist größtenteils sehr sauber und gut strukturiert - toll gemacht!

🧪 Testing: 5/10

- **Framework**: xUnit + FluentAssertions + Moq sind korrekt eingerichtet
- **Probleme**: Unvollständige Testimplementierungen, fehlende Coverage-Metriken
- **Struktur**: Gute Organisation der Test-Projekte

Das Testing-Framework ist da, aber es braucht noch etwas Liebe um vollständig zu werden.

🚀 Performance: 6/10

- **Stärken**: Async/await Patterns, modernes Framework
- **Fehlt**: Caching-Layer, Connection Pooling, Query-Optimierung

Solide Basis, aber mit ein paar Performance-Verbesserungen könnte es noch besser werden.

👨‍💻 Entwicklererfahrung: 6/10

- **Positives**: Swagger-Dokumentation, Docker Compose, strukturiertes Logging

- **Fehlt**: README, Build-Skripte, Entwicklungsumgebung-Setup

Das Setup ist schon ganz gut, aber neue Entwickler würden sich über mehr Dokumentation freuen.

🔧 Wartbarkeit: 7/10

- **Stärken**: Modulares Design, Dependency Injection, klare Trennung
- **Bedenken**: Komplexe Geschäftslogik, hartcodierte Regeln

Die Architektur macht es einfach, das System zu erweitern und zu warten.

Empfehlungen für die nächsten Schritte

🔴 Sofort (Kritisch)

1. SQL-Injection-Schwachstellen in DatabaseManager.cs beheben
2. Sensitive Konfiguration in sicheres Key-Management verschieben
3. Unvollständige Testimplementierungen abschließen
4. Statische Felder aus Controllern entfernen
5. Feste Werte für eine Sachmerkmalsgruppe im Mapping über SQL flexibel machen
6. Die Anzeige des Ergebnisses in i:qu durch Austausch mit Michael Loell ermöglichen

🟡 Mittelfristig

1. Caching-Layer und Connection Pooling hinzufügen
2. Umfassendes Error-Handling-Middleware implementieren
3. Ordentliche Dokumentation und README erstellen
4. Health Checks und Monitoring hinzufügen

🟢 Langfristig

1. Code-Sprache standardisieren (Deutsch vs. Englisch)
2. Richtige Datenbank-Migrationen implementieren
3. API-Versionierungsstrategie hinzufügen
4. Microservices-Architektur in Betracht ziehen

Fazit

Das Preiswürfel Backend ist ein solides Projekt mit einer **sehr guten architektonischen Grundlage** und modernen .NET-Praktiken. Die Entwickler haben wirklich gute Arbeit bei der Strukturierung geleistet!

Aber: Die **kritischen Sicherheitslücken müssen unbedingt vor dem Produktionseinsatz** behoben werden. Sobald diese Punkte erledigt sind, habt ihr ein wirklich starkes System.

Die Kombination aus sauberer Architektur, modernem Tech-Stack und durchdachter Struktur zeigt, dass hier mit Kompetenz und Sorgfalt entwickelt wurde. Mit den empfohlenen Verbesserungen wird das zu einem erstklassigen Backend-System!

Einschätzung der Arbeit des neuen Entwicklers

Es ist zu berücksichtigen, dass dieses Backend mit bestehenden Lücken und Mängeln an den neuen Entwickler übergeben wurde. Seit der Übernahme hat Thomas Bernecker das Testing deutlich erweitert und das Mapping in eine bevorzugte objektorientierte Programmierung überführt. Bei der Erstellung neuer Klassen hat er die Architektur- und Codekonventionen sehr gut eingehalten, wodurch die Struktur des Codes sowohl innerhalb als auch außerhalb der Klassen weiterhin übersichtlich bleibt. Das Logging einzelner Berechnungen, das vor der Übergabe nur teilweise umgesetzt war, wurde von ihm wesentlich ausgebaut.

An dem Übergang von Oxaion zu Oxaion infinite wird weiterhin in Zusammenarbeit von Thomas Bernecker und Michael Loell umgesetzt. Die hierfür erforderlichen Anpassungen am Preiswürfel wurden erfolgreich durchgeführt. An der Anzeige des Rechnungsergebnisses wird aktuell aktiv gearbeitet.

Insgesamt verlief die Projektübergabe sehr gut. Thomas hat sich zügig und gründlich in das Thema Preiswürfel eingearbeitet und verfügt mittlerweile über ein gutes Verständnis des Projekts. Die ersten Aufgaben konnte er nach einer kurzen Einarbeitungszeit zunehmend schneller umsetzen. Es ist jedoch zu erwarten, dass größere Änderungen und Erweiterungen künftig auch mehr Zeit für Planung und Abstimmung erfordern werden. Auftretende Fehler kann er inzwischen auf unterschiedlichen Wegen analysieren und beheben. Auch die Navigation durch die internen Systeme und Server der Lindner Group wird kontinuierlich sicherer und effizienter.

Diese Analyse wurde am 15. Juli 2025 erstellt. Bei Fragen oder Unklarheiten gerne nachfragen! 😊