Datenbanken mit C#:
Portfolio Auftrag

1 Auftrag

Im Rahmen dieses Portfolio-Auftrags erweitern Sie Ihre BattleShip Implementation um eine Persistenzschicht für folgende Informationen:

- Benutzerprofile (Benutzername, Passwort, Geburtsjahr)
- Erreichte Punktestände

Verwenden Sie für die Speicherung Entity-Framework Core. Als Datenbank-Backend verwenden Sie anstelle von MySQL die Dateibasierte Datenbank SQLite[1]. Planen Sie die Implementation so, dass Sie im Minimum einen Migrationsschritt mit der Datenbank ausführen müssen. Konkret bedeutet dies: Sie lassen zunächst z.B. das Geburtsjahr sowie die Punktestände weg, erzeugen ihre erste Version der Datenbank und fügen im Anschluss die entsprechenden Erweiterungen mit entsprechenden Migrationen hinzu. BattleShip soll dabei selbstständig beim Verbinden die Datenbank aktualisieren.

1.1 Detailanforderungen

1.1.1 User Management

Beim Start von BattleShip soll eine Benutzerauthentifizierung über Benutzername/Passwort angefordert werden. Diese Identitätsinformation wird im Anschluss verwendet um die Spielergebnisse einem Spieler zuordnen zu können.

Jedem Benutzer wird ein Zugriffslevel zugeordnet (zurzeit bekannt: «Benutzer» oder «Administrator»). Diese Zugriffslevel unterscheiden sich aktuell nur in einem Punkt: Während Benutzer lediglich spielen können, können Administratoren zusätzliche Operationen ausführen. Im Fall des User Managements ist dies konkret das Erfassen, Editieren und Deaktivieren von Benutzern. Das Deaktivieren eines Benutzers führt dazu, dass sich dieser nicht mehr einloggen kann.

1.1.2 Erreichte Punktestände

Die wichtigsten Eigenschaften jedes Spiels soll auf geeignete Weise erfasst werden:

- Zeitpunkt des Spielendes
- Welche Spieler waren involviert?
- Wie viele Runden wurde gespielt?
- Punktzahl
- Sieger

Diese Daten sollen in einer High-Score-Liste nach Punkten absteigend sortiert in BattleShip abrufbar sein. Auf Knopfdruck ist das Scoreboard für einen Administrator löschbar. Dadurch werden keine Einträge mehr angezeigt. Dennoch sollen aber die Einträge in der Datenbank gespeichert bleiben (man weiss ja nie 😉).

[1] https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=visual-studio

2 Bewertungsraster

Fügen Sie Ihrem Portfolio ein Kapitel «Datenbanken mit C#» hinzu. Dokumentieren Sie in diesem Kapitel die nachfolgend verlangten Angaben. Das Bewertungsschema liefert Ihnen eine Übersicht über die erwarteten Punkte:

Thema	Kriterien	Bewertung
Konzeption	Erstellen Sie zunächst die Konzeption für Ihre Erweiterung entlang nachfolgender Kriterien: • Die Lösung ist so konzipiert, dass sie erweiterbar und wartbarbar bleibt. • Die Lösung folgt dem Prinzip der Clean Architecture. • Das Datenmodell beherbergt sämtliche nötigen Daten und entspricht der 3. Normalform • Die Speicherung der Daten ist so gestaltet, dass sensible Informationen, insbesondere Passwörter nicht rekonstruiert werden können (Stichwort: Salted Hashes[2])	☐0 ☐1 ☐2 ☐3 ☐4
Dokumentation	Die Lösung ist im Portfolio ausreichend dokumentiert. • Die Lösung ist durch geeignete Diagramme (ERM, Klassendiagramme) dokumentiert • Die Diagramme werden durch Beschreibungen ergänzt, welche die grundlegenden Konzepte der Lösung ausführen. • Die Sicherheitserwägungen hinsichtlich der Speicherung der Passwörter sind nachvollziehbar	☐0 ☐1 ☐2 ☐3 ☐4
Umsetzung	Die Umsetzung in C# entspricht den gängigen Praktiken: • Der Code ist verständlich gestaltet (Clean Code). • SOLID Prinzipien werden eingehalten. • Datenzugriff erfolgt über EF Core und ist so gestaltet, dass die Materialisierung der Abfragen so spät wie möglich erfolgt • Die Lösung wurde auf GitHub gepushed mit einem Tag namens `release/portfolio_db_code` versehen. Der Link zu diesem Tag ist im Portfolio hinterlegt, das Repository für die Lehrperson zugänglich.	☐0 ☐1 ☐2 ☐3 ☐4

[2] https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/