Wein Backend Dokumentation

Bunea, Buchinger, Bauer, Lagler

8. April 2025

Inhaltsverzeichnis

1 Einführung

Dieses Protokoll beschäftigt sich mit den Änderungen der `WineDB` während des dritten und vierten Sprints. Das Feedback der Auftraggeber wurde eingebaut.

2 Einführung von GUIDs

In früheren Versionen der `WineDB` wurden einfache `Integer`-Werte zur Identifikation von Entitäten verwendet. Diese Herangehensweise führte jedoch zu Problemen bei der Entwicklung von Offline-fähigen Applikationen, da diese ohne Zugriff auf die zentrale Datenbank keine eindeutigen IDs erzeugen konnten.

Durch die Einführung von **GUIDs** (Globally Unique Identifiers) kann nun jede Applikation, unabhängig von einer Serververbindung, eindeutige Identifikatoren erzeugen.

Ein typisches GUID-Feld in einer Entität sieht wie folgt aus:

```
1 public Guid Id { get; set; }
```

Diese Umstellung stellt sicher, dass neue Einträge bereits beim Erstellen innerhalb der Applikation eine weltweit eindeutige ID besitzen – selbst in völliger Abwesenheit des Mutterschiffs, äh, der Datenbank.

3 WineTypes

Neu hinzugekommen sind `WineTypes`. `WineTypes` repräsentieren einen Wein Typen, wie *Chardonnay* oder *Merlot*.

```
1 public class WineType
2 {
3     public Guid Id { get; set; }
4     public string? Name { get; set; }
5 }
```

4 AdditiveTypes

`AdditiveTypes` sind ähnlich wie `WineTypes`. Sie repräsentieren mögliche Additives.

```
1 public class WineType
2     public class AdditiveType
3     {
```

```
        public Guid Id { get; set; }
        public string Type { get; set; }
    }
```

5 WineBarrels

`WineBarrels` repräsentieren Weinfässer. Diese Fässer besitzen einen `WineType`, der angibt, welcher Wein derzeit im Fass gelagert wird, und können mehrere `Additives` enthalten.

Der Endpoint, um einen Wein in ein Fass einzufügen, lautet:

```
/api/WineBarrels/{id}/InsertWine/{wineTypeId}/{startDate}
```

Beim Hinzufügen wird automatisch ein neuer Eintrag in der Historie des Weines erstellt. Die `WineBarrelHistory` wird im Kapitel 5.1 weiterführend erläutert. Um einem Fass einen neuen Wein zuzuweisen, muss zunächst – sofern bereits Wein vorhanden ist – dieser mittels des folgenden Endpoints entfernt werden:

```
/api/WineBarrels/{id}/RemoveCurrentWine/{endDate}
```

Dieser Vorgang schließt auch den zugehörigen Eintrag in der `WineBarrelHistory` ab.

```
public class WineBarrel
{
    public Guid Id { get; set; }
    public Guid UserId { get; set; }
    public Guid? CurrentWineTypeId { get; set; }
    public Guid? CurrentWineBarrelHistoryId { get; set; }

    public string Name { get; set; }

    public float MostWeight { get; set; }

    public DateTime HarvestDate { get; set; }

    public float VolumeInLitre { get; set; }

    public string ProductionType { get; set; }

    public Guid? MostTreatmentId { get; set; }

    [JsonIgnore]
    [XmlIgnore]
```

```
22      public List<WineBarrelHistory> History { get; set; }
23
24      [JsonIgnore]
25      [XmlIgnore]
26      public User User { get; set; }
27
28      [JsonIgnore]
29      [XmlIgnore]
30      public MostTreatment? MostTreatment { get; set; }
31
32      [JsonIgnore]
33      [XmlIgnore]
34      public List<FermentationEntry> FermentationEntries { get;
        set; }
35
36      [JsonIgnore]
37      [XmlIgnore]
38      public List<Additive> Additives { get; set; }
39  }
```

5.1 Wine Barrel History

Die `WineBarrelHistory` speichert den Zeitraum, in dem ein spezifischer Wein in einem Fass gelagert wurde. Diese Einträge werden automatisch beim Einfügen und Entfernen von Weinen erzeugt.

```
1   public class WineBarrelHistory
2   {
3       public Guid Id { get; set; }
4
5       public Guid WineBarrelId { get; set; }
6       public Guid WineTypeId { get; set; }
7
8       public DateTime? StartDate { get; set; }
9       public DateTime? EndDate { get; set; }
10
11      [JsonIgnore]
12      [XmlIgnore]
13      public WineBarrel WineBarrel { get; set; }
14
15      [JsonIgnore]
16      [XmlIgnore]
17      public WineType WineType { get; set; }
18
19  }
```

6 Constraints

`Constraints` definieren Mindest- und Höchstwerte sowie Validierungsregeln für verschiedene Entitäten. Sie sorgen dafür, dass fehlerhafte oder unvollständige Daten frühzeitig erkannt werden.

Beim Aufruf der API wird eine zentrale Check-Funktion ausgeführt, welche alle relevanten Validierungen durchführt. Alle Fehlermeldungen werden gesammelt und dem Endnutzer über die API als Antwort zurückgegeben:

```
var (isValid, Error) = WineBarrelConstraints.CheckBarrel(wine);

if (!isValid)
{
    return BadRequest(Error);
}
```

```
public class WineBarrelConstraints
{
    public static float MinMostWeight = 1;
    public static float MaxMostWeight = 100000;

    public static float MinVolume = 0;
    public static float MaxVolume = 100000;

    public static (bool IsValid, string ErrorMessage) CheckBarrel(WineBarrel barrel)
    {
        List<string> errors = new();

        if (barrel == null)
        {
            return (false, "Wine barrel cannot be null.");
        }

        if (barrel.MostWeight < MinMostWeight || barrel.MostWeight > MaxMostWeight)
            errors.Add($"MostWeight must be between {MinMostWeight} and {MaxMostWeight}.");

        if (barrel.VolumeInLitre < MinVolume || barrel.VolumeInLitre > MaxVolume)
            errors.Add($"VolumeInLitre must be between {MinVolume} and {MaxVolume}.");

        if (string.IsNullOrWhiteSpace(barrel.Name))
            errors.Add("Name cannot be empty or whitespace.");
```

```
26
27          if (errors.Count == 0)
28              return (true, string.Empty);
29
30          return (false, string.Join("\n", errors));
31      }
32  }
```

Listing 1: Beispielhafte Constraints-Validierung für `WineBarrels`

7 Authorisation

Die Authorisation Funktionalität wurde etwas verbessert.

7.1 AuthHelper

Die `AuthHelper` Klasse erleichtert die Authenzierung innerhalb der API Endpoints. Mithilfe der *GetAuthenticatedUser* Methode wird ein Request authentiziert. Das Resultat wird als AuthResults Klasse zurückgeliefert.

```
1
2  public AuthResults GetAuthenticatedUser(ControllerBase
       controller)
3  {
4      var userIdClaim = controller.User.Claims.FirstOrDefault(c
       => c.Type == "userId");
5      if (userIdClaim == null)
6      {
7          return new AuthResults(false, null, controller.
       Unauthorized());
8      }
9
10     if (!Guid.TryParse(userIdClaim.Value, out var userId))
11     {
12         return new AuthResults(false, null, controller.
       BadRequest("Invalid user ID in token."));
13     }
14
15     return new AuthResults(true, userId, null);
16 }
17
18 public async Task<bool> UserHasAdminRights(WineDbContext
       context, Guid? userId)
19 {
20     if (userId == null)
21         return false;
22
```

```
23      var user = await context.Users.FindAsync(userId);
24
25      if (user == null)
26          return false;
27
28      return user.AdminRights;
29  }
```

Authentication erfolgt wie folgt:

```
var results = authHelper.GetAuthenticatedUser(this);

if (!results.IsAuthenticated)
{
    return results.ErrorResult;
}
```

7.2 Admin User

Die `User` Klasse hat das Feld *AdminRights* bekommen. Dieses Kennzeichnet einen `Admin User`. `Admins` haben das Recht neue `WineTypes` und `AdditiveTypes` anzulegen.

```
    public class User
    {
        public Guid Id { get; set; }

        public string Username { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
        public bool AdminRights { get; set; } = false;
    }
```