


SPENGERGASSE  <small>ausbildung mit zukunft</small>	HTBLVA, Wien V, Spengergasse Aufbaulehrgang für Informatik (Tag) Kolleg für Informatik (Tag)	Reife- und Diplomprüfung
--	---	-------------------------------------

Aufgabenstellung für die nicht standardisierte Klausurprüfung

Herbsttermin 2024/25

Jahrgang:	6AAIF, 6BAIF, 6CAIF, 6AKIF, 6BKIF
Prüfungsgebiet:	Fachtheorie
Zugeteilter Pflichtgegenstand (Zugeteilte Pflichtgegenstände)	Programmieren und Software Engineering
Prüfungstag:	22.09.2025
Arbeitszeit:	300 Minuten
Kandidaten/Kandidatinnen:	65
Prüfer/Prüferin:	Michael Schletz, BEd, René Wenz, BSc
Aufgabenblätter:	10 Blätter inkl. Umschlagbogen

(Unterschrift des Prüfers/der Prüferin auf den jeweiligen Aufgabenblättern)

Das versiegelte Kuvert mit der Aufgabenstellung wurde geöffnet von:

Name: _____ Unterschrift: _____

Datum: _____ Uhrzeit: _____

Zwei Zeugen (Kandidaten/Kandidatinnen)

Name: _____ Unterschrift: _____

Name: _____ Unterschrift: _____

Geprüft: Wien, am _____

 AV Mag. Heidi Steinwender
 Abteilungsvorständin

RS.

 Dir. Dr. Gerhard Hager
 Direktor

Genehmigt:

Wien, am _____

RS.

 Mag.^a MinRⁱⁿ Gabriele Winkler-Rigler
 Schulaufsicht

Klausurprüfung in Programmieren und Software Engineering

Klassen: 6AAIF, 6BAIF, 6CAIF, 6AKIF, 6BKIF

Datum: MO, 22. September 2025

Arbeitszeit: 300 Minuten

Generelle Hinweise zur Bearbeitung

Die Arbeitszeit für die Bearbeitung der gestellten Aufgaben beträgt 5 Stunden (300 Minuten). Die 3 Teilaufgaben sind unabhängig voneinander zu bearbeiten, Sie können sich die Zeit frei einteilen. Wir empfehlen jedoch eine maximale Bearbeitungszeit von 2 Stunden für Aufgabe 1, 1 Stunde für Aufgabe 2 und 2 Stunden für Aufgabe 3. Bei den jeweiligen Aufgaben sehen Sie den Punkteschlüssel. Für eine Einrechnung der Jahresnote sind mindestens 30% der Gesamtpunkte zu erreichen.

Hilfsmittel

In der Datei *P:/SPG_Fachtheorie/SPG_Fachtheorie.sln* befindet sich das Musterprojekt, in dem Sie Ihren Programmcode hineinschreiben. Im Labor steht Visual Studio 2022 mit der .NET Core Version 8 zur Verfügung. Da das Projekt auf dem Netzlaufwerk liegt, muss am Ende nicht extra auf ein Abgabelaufwerk kopiert werden. Beenden Sie am Ende Ihrer Arbeit alle Programme und lassen Sie den PC eingeschaltet.

Als erlaubte Hilfsmittel befinden sich im Ordner **R:\exams** bereitgestellte Unterlagen. Dies ist ein implementiertes Projekt ohne Kommentare aus dem Unterricht, wo Sie die Parameter von benötigten Frameworkmethoden nachsehen können.

Wichtiger Hinweis vor Arbeitsbeginn




Füllen Sie die Datei *README.md* in *SPG_Fachtheorie/README.md* mit Ihren Daten (Klasse, Name und Accountname) aus. Sie sehen die Datei in Visual Studio unter *Solution Items* nach dem Öffnen der Solution. **Falls Sie dies nicht machen, kann Ihre Arbeit nicht zugeordnet und daher nicht bewertet werden!**

Teilaufgabe 1: Object Relation Mapping


Der Betreiber unserer Schulkantine möchte ein System entwickeln, mit dessen Hilfe Schüler:innen Menüs vorbestellen können. Er beliefert mehrere Schulen, weswegen das Entity *School* die Daten der Schule erfassen muss. Der Menüplan ist in *Meal* hinterlegt. Er speichert, an welchem Tag in welcher Schule es welche Menüs zur Auswahl gibt. Es gibt 2 Arten von Menüs: vegetarisch (*Vegetarian*) und ein Fleischgericht (*MeatDish*). Damit die Allergene ausgewiesen werden können, wird im Entity *Allergen* eine Liste der möglichen Allergene gespeichert. Das Entity *MealAllergen* weist die Allergene den einzelnen angebotenen Menüs zu. Die Bestellungen der Schüler:innen werden im Entity *MealOrder* erfasst.

Aus den Daten lässt sich z. B. die Übersicht der angebotenen Menüs generieren:




Unser aktueller Menüplan

Dienstag, 10.06.2025




Menü 1
Asiatische Hühnerpfanne mit Reis
F 4,40 €




Menü 2
Überbackene Gemüsefleckerl mit Salat
ACG 4,40 €

Mittwoch, 11.06.2025




Menü 1
Hascheehörnchen mit Salat
ACS 4,40 €




Menü 2
Kartoffelpuffer mit Knoblauchsauce
AG 4,40 €

Donnerstag, 12.06.2025




Menü 1
Holzknechtknödel mit Salat
ACS 4,40 €



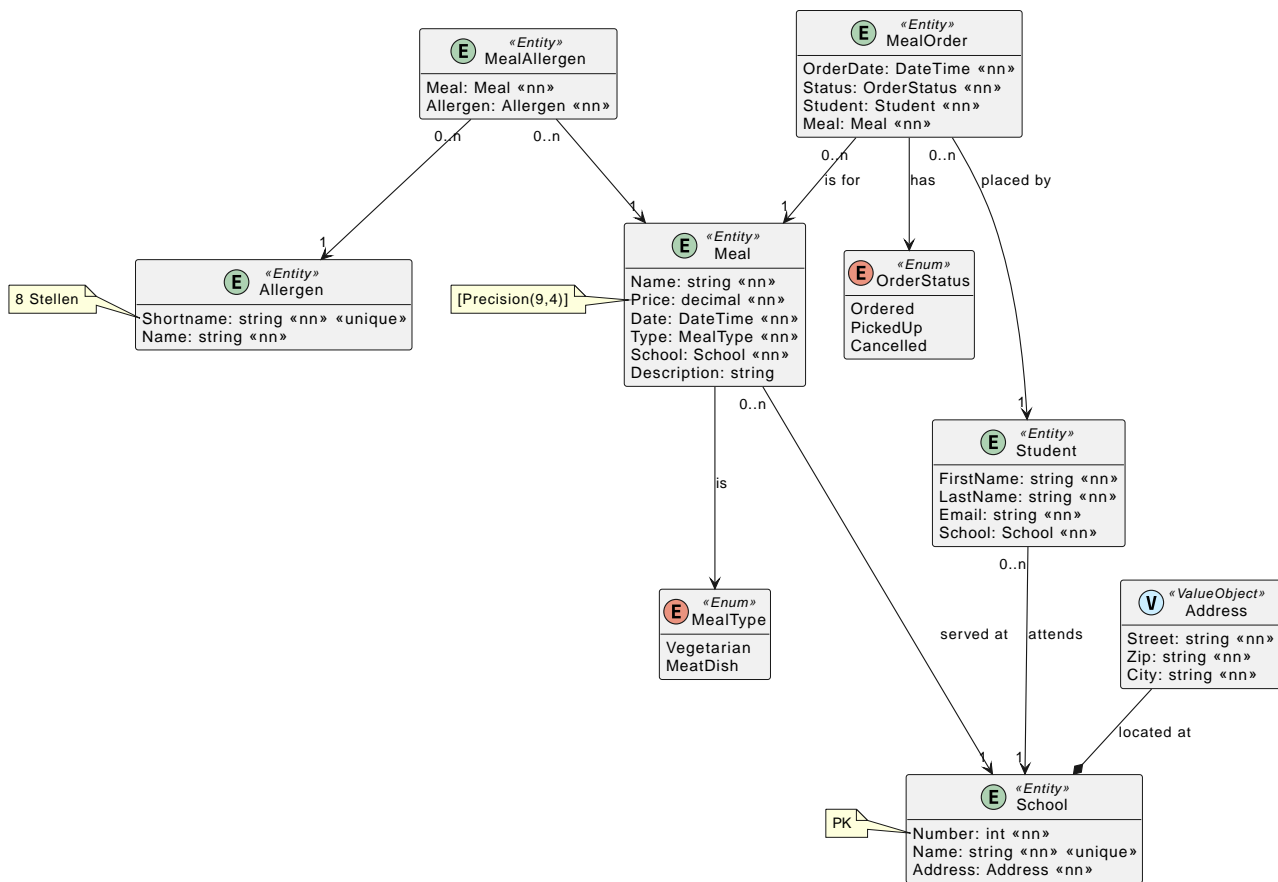
Menü 2
Gnocchi in Gorgonzolasauce
AGL 4,40 €

Freitag, 13.06.2025



Menü 1
Tagliatelle mit Lachsrahmsauce
ACDG 4,40 €

Quelle: <https://spengergasse.easymensa.at>



Arbeitsauftrag

Erstellung der Modelklassen

Implementieren Sie das dargestellte Diagramm als EF Core Modelklassen. Im Projekt *SPG_Fachtheorie_Aufgabe1* befinden sich leere Klassen sowie die Klasse *MealContext*, die Sie nutzen sollen. Beachten Sie bei der Umsetzung folgende Punkte:

- Legen Sie nötige Konstruktoren an. Ein *public* Konstruktor soll alle im Modell enthaltenen Properties initialisieren. Ergänzen Sie die für EF Core nötigen *protected* Konstruktoren.
- Beachten Sie Attribute Constraints wie *not null* ([nn]).
- Strings sollen - wenn nicht anders angegeben - mit einer Maximallänge von 255 Zeichen definiert werden.
- *Allergen.Shortname* soll maximal 8 Stellen lang sein.
- Der Preis in *Meal* soll mit 9 Stellen (5 Vorkomma- und 4 Nachkommastellen) gespeichert werden. Sie können das Attribut *[Precision(9, 4)]* verwenden.
- Das Attribut *Address* soll in *School* als *value object* definiert werden.
- Verwenden sie eigene primary keys mit dem Namen *Id* (autoincrement), außer im Modell ist mit *PK* explizit ein Schlüssel angegeben.
- Speichern Sie die enum *OrderStatus* und *MealType* in *MealOrder* bzw. *Meal* als String in der

Datenbank ab.

- Beachten Sie die *unique* constraints für *Allergen.Shortname* und *School.Name*.

Verfassen von Tests

In der Klasse *Aufgabe1Test* im Projekt *test/SPG_Fachtheorie.Aufgabe1.Test* sollen Testmethoden verfasst werden, die die Richtigkeit der Konfiguration des OR Mappers beweisen sollen.

- *PersistEnumInMealTest* beweist, dass die Enum *MealType* in *Meal* korrekt (als String) gespeichert wird.
- *PersistEnumInMealOrderTest* beweist, dass die Enum *OrderStatus* in *MealOrder* (als String) korrekt gespeichert wird.
- *PersistValueObjectInSchoolTest* beweist, dass Sie ein Entity vom Typ *School* mit einer Adresse als value object speichern können.
- *EnsureNameInSchoolIsUniqueTest* beweist, dass Sie nur ein Entity vom Typ *School* mit gleichem Namen speichern können.
- *EnsureShortnameInAllergenIsUniqueTest* beweist, dass Sie nur ein Entity vom Typ *Allergen* mit gleichem Shortname speichern können.

Sie können die vorgegebenen Tests in *test/SPG_Fachtheorie.Aufgabe1.Test/Aufgabe1MasterTests* verwenden, um Ihre Konfiguration bei der Bearbeitung der Aufgabenstellung zu prüfen.

Teilaufgabe 2: Abfragen und Servicemethoden

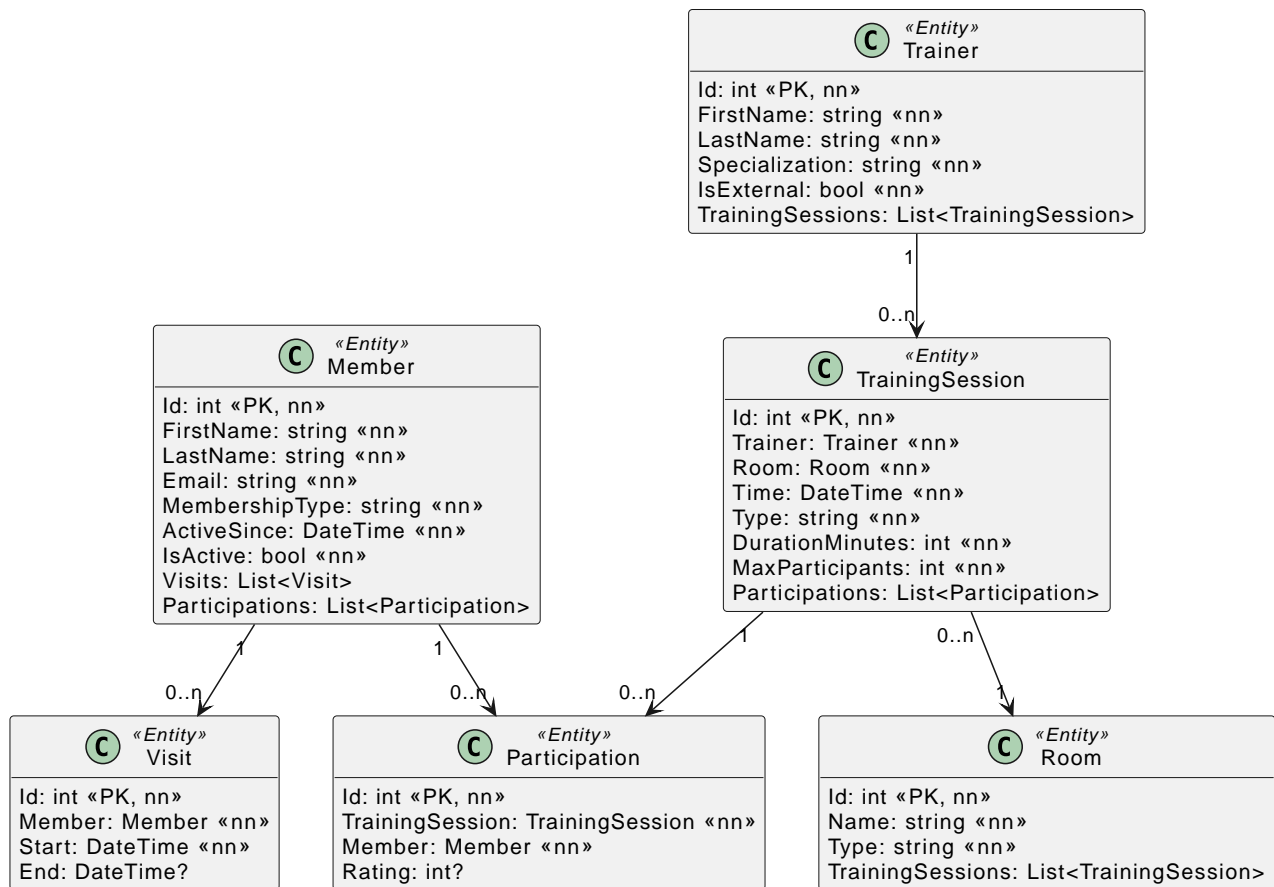
Das nachfolgende Modell zeigt eine Umsetzung eines Fitnesscenters. Zum Besuchen des Studios müssen sich Kunden anmelden. Die Kundendaten sind im Entity *Member* gespeichert. Es gibt 2 Arten von Memberships: *Basic* und *Premium*. Bei jedem Besuch meldet sich der Kunde an der Tür mit einem QR Code an. Bei der Anmeldung wird ein Datensatz in *Visit* geschrieben. Verlässt der Kunde wieder das Fitnesscenter, wird *Visit.End* auf den Zeitstempel gesetzt.

Als Service bietet das Studio auch begleitete Trainings an. Alle Trainer:innen sind im Entity *Trainer* registriert. Die Spezialisierung gibt den bevorzugten Bereich des Trainers (Yoga, Krafttraining, ...) an.

Die Trainer:innen bieten Trainingssessions (*TrainingSession*) an. Eine Session findet zu einer bestimmten Zeit in einem bestimmten Raum statt. Damit nicht zu viele Mitglieder:innen teilnehmen, kann die Anzahl durch das Property *TrainingSession.MaxParticipants* begrenzt werden.

Mitglieder:innen können sich über eine App zu diesen Sessions anmelden. Bei der Anmeldung entsteht ein Eintrag im Entity *Participation*. Nach der Session können die Mitglieder:innen den Trainer bewerten. Das Feld *Participation.Rating* sieht einen Zahlenwert von 1 (schlecht) bis 5 (super) vor.

Das folgende Modell zeigt die im Projekt vorhandenen Entities.



Arbeitsauftrag

In den Dateien `src/SPG_Fachtheorie.Aufgabe2/Infrastructure/FitnessContext.cs` steht ein vollständig implementierter DbContext zur Verfügung. Das Model ist bereits in `src/SPG_Fachtheorie.Aufgabe2/Model` implementiert. **Sie müssen keine Implementierung des Modelles vornehmen, sondern es in Ihren Servicemethoden verwenden.**

Implementierung der Servicemethoden

Führen Sie Ihre Implementierungen in `src/SPG_Fachtheorie.Aufgabe2/Services/FitnessService.cs` durch. Verwenden Sie den über Dependency Injection bereitgestellten *FitnessContext*.

public List<ActiveMemberDto> GetActiveMembers()

Diese Methode soll alle aktiven Mitglieder (*Member.IsActive* ist *true*) zurückgeben. Als Rückgabetyt steht Ihnen der folgende Record zur Verfügung.

```
public record ActiveMemberDto(
    int Id, string FirstName, string LastName, string Email);
```

Id: Mitglieder ID, First- und Lastname: Name des Mitgliedes, Email: Email des Mitgliedes.

public List<TrainingSessionWithCountDto> GetTrainingSessionsWithParticipantCounts()

Diese Methode soll alle Trainings-Sessions zurückliefern. Es soll auch die Anzahl der Teilnehmer (*Participations*) mitgeliefert werden. Als Rückgabetypp steht Ihnen der folgende Record zur Verfügung.

```
public record TrainingSessionWithCountDto(
    int Id, string RoomName, DateTime DateTime,
    string TrainerFirstName, string TrainerLastName, int ParticipantCount);
```

public List<MaxRatingCountPerTrainer> GetMaxRatingCountPerTrainer()

Diese Methode soll ermitteln, wie viele Bestnoten (*Participation.Rating* = 5) ein Trainer bekommen hat. Dabei sollen die Ratings über alle Training-Sessions aufsummiert werden. Pro Training-Sessions soll über alle Participations mit dem Rating = 5 gezählt werden. Als Rückgabetypp steht Ihnen der folgende Record zur Verfügung.

```
public record MaxRatingCountPerTrainer(
    int Id, string FirstName, string LastName, int MaxRatingCount);
```

Id: ID des Trainers, First- und Lastname: Vor- und Zuname des Trainers, MaxRatingCount: Anzahl der Ratings mit Wert 5.

public Participation RegisterMemberToTrainingSession(int memberId, int trainingSessionId)

Diese Methode soll ein Mitglied zu einer Trainingssession anmelden. Dafür soll ein Datensatz in *Participation* geschrieben werden. Als Rating wird *null* eingetragen. Beachten Sie folgende Randbedingungen:

- Wurde der Parameter *memberId* nicht in *Members* gefunden, so ist eine *FitnessServiceException* mit dem Text *Member not found.* zu werfen.
- Wurde der Parameter *trainingSessionId* nicht in *TrainingSessions* gefunden, so ist eine *FitnessServiceException* mit dem Text *TrainingSession not found.* zu werfen.
- Hat sich das Mitglied bereits für diese Session angemeldet, so ist eine *FitnessServiceException* mit dem Text *Member already registered.* zu werfen.
- Ist die Session voll (es sind bereits so viele Mitglieder wie in *TrainingSession.MaxCount* definiert angemeldet), so ist eine *FitnessServiceException* mit dem Text *Training Session is already full.* zu werfen.

Geben Sie das erstellte Entity als Rückgabewert zurück. Beachten Sie den Text der Fehlermeldung genau (mit dem Punkt am Ende), er wird im vorgegebenen Test geprüft.

public void UpdateRating(int memberId, int trainingSessionId, int rating)

Diese Methode soll das Rating eines Mitgliedes für eine Trainings Session aktualisieren. Dafür soll der Wert *Participation.Rating* auf den Wert von *rating* gesetzt werden. Beachten Sie folgende Randbedingungen:

- Das Rating muss ein Wert zwischen 1 und 5 sein. Falls nicht, so ist eine *FitnessServiceException* mit dem Text *Rating must be between 1 and 5.* zu werfen.
- Hat das Mitglied die Trainingssession gar nicht besucht (d. h. es gibt keinen Eintrag in *Participations*), so ist eine *FitnessServiceException* mit dem Text *Participation not found.* zu werfen.

Beachten Sie den Text der Fehlermeldung genau (mit dem Punkt am Ende), er wird im vorgegebenen Test geprüft.

Testen Ihrer Implementierung

Verwenden Sie die vorgegebenen Tests in *test/SPG_Fachtheorie.Aufgabe2.Test/Aufgabe2Master-Tests*, um Ihre Implementierung bei der Bearbeitung zu prüfen.

Teilaufgabe 3: REST(ful) API

Für das vorige Modell des Fitness Studios soll eine RESTful API implementiert werden. Wenn Sie das Projekt in *src/SPG_Fachtheorie.Aufgabe3* starten, steht Ihnen unter der URL <http://localhost:5080/swagger/index.html> ein Endpoint Explorer zur Verfügung. Die Datenbank beinhaltet Musterdaten, sodass Sie die Funktionalität Ihrer Controller damit testen können.

Arbeitsauftrag

Implementieren Sie die folgenden REST API Routen. In *src/SPG_Fachtheorie.Aufgabe3/Controllers/FitnessController.cs* steht dafür ein leerer Controller bereit. Über Dependency Injection wird ein mit Musterdaten gefüllter *FitnessContext* bereitgestellt. Sie können die Abfragen direkt in den Controllermethoden implementieren. Führen Sie alle Abfragen nicht blockierend (mit *await* und *async*) durch. Achten Sie bei Fehlern auf eine RFC-9457 kompatible Antwort, indem Sie die Methode *Problem()* mit geeignetem Statuscode verwenden.

GET /fitness/members?membershipType=(string)

Diese REST API Route soll eine Liste von Mitgliedern zurückliefern. Der Filterparameter *membershipType* filtert die Liste nach dem Property *Member.MembershipType*. Ist der Filter leer oder wurde nicht übergeben, so ist die Liste ungefiltert zurückzugeben. Für die Rückgabe steht Ihnen folgender Record bereit:


```
public record MemberDto(int Id, string FirstName, string LastName, string Email, string
MembershipType);
```

Id: ID des Mitgliedes, First- und Lastname: Name des Mitgliedes, Email: E-Mail Adresse des Mitgliedes, MembershipType: Wert des Properties *Member.MembershipType*.

Table 1. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
200	Eine Liste von <i>MemberDto</i> Objekten mit allen Mitgliedern, wenn der Parameter <i>membershipType</i> leer oder nicht gesetzt ist.
200	Eine Liste von <i>MemberDto</i> Objekten mit gefilterten, wenn der Parameter <i>membershipType</i> gesetzt ist.

Response für GET /fitness/members

```
[
  {
    "id": 1,
    "firstName": "Alice",
    "lastName": "Example",
    "email": "alice@example.com",
    "membershipType": "Premium"
  },
  {
    "id": 2,
    "firstName": "Bob",
    "lastName": "Fit",
    "email": "bob@example.com",
    "membershipType": "Basic"
  },
  {
    "id": 3,
    "firstName": "Charlie",
    "lastName": "Chill",
    "email": "charlie@example.com",
    "membershipType": "Basic"
  },
  {
    "id": 4,
    "firstName": "Dora",
    "lastName": "Power",
    "email": "dora@example.com",
    "membershipType": "Premium"
  },
  {
    "id": 5,
    "firstName": "Eli",
    "lastName": "Motion",
    "email": "eli@example.com",
    "membershipType": "Basic"
  }
]
```

Response für `_GET /fitness/members?membershipType=Premium`

```
[
  {
    "id": 1,
    "firstName": "Alice",
    "lastName": "Example",
    "email": "alice@example.com",
    "membershipType": "Premium"
  },
  {
    "id": 4,
    "firstName": "Dora",
    "lastName": "Power",
    "email": "dora@example.com",
    "membershipType": "Premium"
  }
]
```

DELETE /fitness/member/{id}

Dieser Endpunkt soll Mitglieder aus der Datenbank löschen. Die übergebene ID als Routingparameter ist der Wert in *Member.Id*. Beachten Sie, dass Sie in der Datenbank zuerst alle *Participations* und alle *Visits* löschen müssen, bevor Sie das Mitglied löschen. Der Controller liefert immer - auch wenn das Mitglied nicht gefunden wurde - HTTP 204 (no content).

Table 2. Erwartete HTTP-Antworten:

HTTP Status	Bedingung
204	No content, wenn erfolgreich gelöscht wurde oder das Mitglied gar nicht existiert.

Verfassen von Integration Tests

In `test/SPG_Fachtheorie.Aufgabe3.Test/FitnessControllerTests.cs` sollen Integration Tests verfasst werden. Es soll jede Zeile in den Tabellen bei *Erwartete HTTP-Antworten* geprüft werden. Prüfen sie bei DELETE Requests auch, ob der Inhalt in der Datenbank tatsächlich gelöscht wurde. Verwenden Sie dafür die in der *TestWebApplicationFactory* bereitgestellten Hilfsmethoden:

Methode	Beschreibung
InitializeDatabase	Erstellt eine leere Datenbank und fügt ggf. Werte ein.
QueryDatabase	Erlaubt das Abfragen der Datenbank.
GetHttpContent<T>	Führt einen GET Request durch und liefert das Ergebnis als Typ <i>T</i> .
DeleteHttpContent	Führt einen DELETE Request durch und liefert den Status Code zurück.

Wichtiger Hinweis nach Arbeitsende



Starten Sie am Ende Ihrer Arbeit - nach dem Schließen der IDE - das Skript *compile.cmd* im Ordner *SPG_Fachtheorie*. Es kontrolliert, ob Ihre Projekte kompiliert werden können. **Projekte, die nicht kompilieren, können nicht bewertet werden!**