Klausurprüfung (Fachtheorie)
aus Programmieren und Software Engineering

im Nebentermin Jänner 2023
für das Schuljahr 2021/22

für den Aufbaulehrgang für Informatik – Tag/Abend (SFKZ 8167/8168)
für das Kolleg für Informatik – Tag/Abend (SFKZ 8242/8244)

Liebe Prüfungskandidatin,
liebe Prüfungskandidat!

Bitte füllen Sie zuerst die nachfolgenden Felder in Blockschrift aus, bevor Sie mit der Arbeit beginnen.

Maturaaccount (im Startmenü sichtbar):	
Vorname	
Zuname	
Klasse	

Klausurprüfung aus Fachtheorie

Generelle Hinweise zur Bearbeitung

Die Arbeitszeit für die Bearbeitung der gestellten Aufgaben beträgt 6 Stunden (360 Minuten). Die 3 Teilaufgaben sind unabhängig voneinander zu bearbeiten, Sie können sich die Zeit frei einteilen. Wir empfehlen jedoch eine maximale Bearbeitungszeit von 1.5 Stunden für Aufgabe 1, 1.5 Stunden für Aufgabe 2 und 3 Stunden für Aufgabe 3.

Die zu vergebenen Punkte sind bei jeder Aufgabenstellung angeführt. Sie müssen für eine Berücksichtigung der Jahresnote mindestens 30% der Gesamtpunkte erreichen. In Summe sind 46 Punkte zu erreichen.

Hilfsmittel

In der Datei *SPG_Fachtheorie.sln* befindet sich das Musterprojekt, in dem Sie Ihren Programmcode hineinschreiben. Im Labor steht Visual Studio 2022 mit der .NET Core Version 6 zur Verfügung.

Mit der Software DBeaver oder SQLite Studio können Sie sich zur generierten Datenbank verbinden und Werte für Ihre Unittests ablesen.

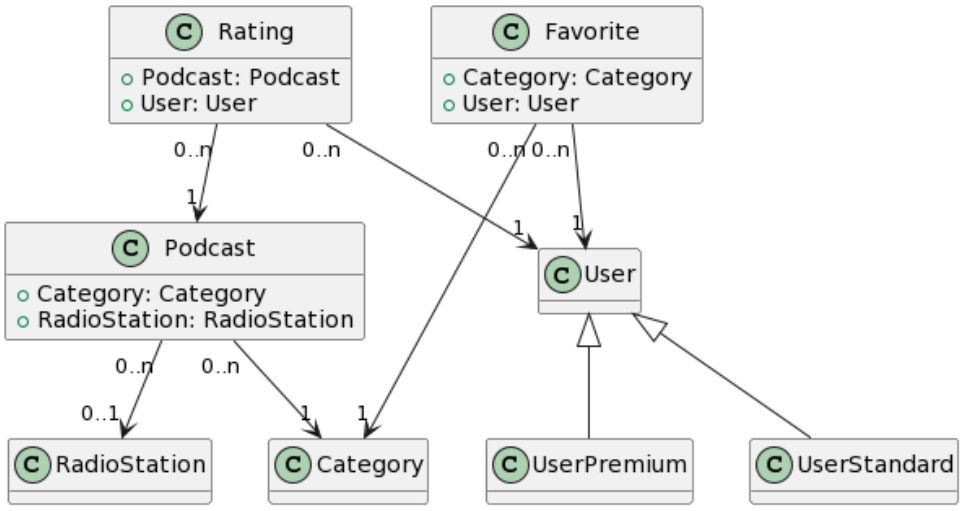
Teilaufgabe 1: Erstellen von EF Core Modelklassen

Verwaltung von Podcasts

Eine Website bietet Podcasts (eine Serie von Audiodateien) von unterschiedlichen Anbietern an. Für die Verwaltung der Podcasts und der Benutzer sollen Sie ein Domain Model, welches folgende Anforderungen abdeckt, konzipieren:

- Jeder Podcast hat eine eindeutige ID und ist genau einer Kategorie zugeordnet.
- Die Benutzer (User) müssen sich vorab auf der Website registrieren.
- Jeder Benutzer kann mehrere Lieblingskategorien (Favorites) für sich speichern.
- Für die Podcasts können durch die Benutzer Bewertungen (Ratings) erstellt werden.

Als Unterstützung wurde bereits die Umsetzung in ein Klassenmodell vorgenommen. Dieses Modell zeigt allerdings nur die Beziehungen zwischen den Klassen, zusätzliche Properties sind von Ihnen zu implementieren.



Arbeitsauftrag

Sie finden in der Mustersolution ein Projekt *SPG_Fachtheorie.Aufgabe1*. Dort ist in der Klasse *UserContext* bereits ein leerer Datenbankcontext für EF Core vorhanden. Leere Klassendefinitionen wurden im Ordner *Model* bereits erstellt.

Implementieren Sie das obige Klassenmodell, sodass diese Klassen mit EF Core in eine Datenbank persistiert werden können. Beachten Sie dabei die nachfolgenden Anweisungen. Am Ende führen Sie den Test *CreateDatabaseTest* aus. Er versucht, eine Datenbank anzulegen. Dieser Test muss erfolgreich durchlaufen werden.

Erforderliche Daten für jede Klasse

Überlegen Sie sich für jede Klasse einen geeigneten Primärschlüssel. Der einfachste Zugang ist sicher ein auto increment Wert. Primärschlüssel, die nicht Id heißen, müssen entsprechend mit *Key* annotiert, oder mittels Fluent API in der Context-Klasse in der entsprechenden Methode *OnModelCreating(…)* angegeben werden. Sehen Sie bei den Navigations immer ein Feld für die Speicherung des Fremdschlüsselwertes vor. Berechnete Werte dürfen natürlich nicht als Felder definiert werden.

Klasse User [3 Punkte]

Die Basisklasse umfasst für jeden Benutzer Vorname, Zuname, Email Adresse, ein Startdatum und ein Endedatum für die Mitgliedschaft. Bei aktiven Benutzern ist keine Angabe für das Endedatum vorhanden.

Von dieser Klasse erben 2 Subklassen: *UserStandard* hat als weiteres Feld ein Flag *informed*, welches die Information speichert, ob dem Benutzer bereits eine Premiummitgliedschaft angeboten wurde. *UserPremium* hat erweiterte Informationen: Ein Feld für die Speicherung des monatlichen Mitgliedsbeitrags und ein Feld um die Anzahl der Monate für die Mindestvertragsdauer zu speichern.

Klasse Podcast [1 Punkt]

Zu einem Podcast wird der Titel und die Länge in Sekunden gespeichert. Weiters wird jeder Podcast genau einer Kategorie zugeordnet. Außerdem kann ein Podcast von einem Radiosender stammen, sodass gegebenenfalls der Radiosender zu speichern ist.

Klasse RadioStation [1 Punkt]

Der Name und Adresse des Radiosenders sowie die Information, ob der Radiosender nur online, nur offline oder über beide Wege empfangbar ist, muss erfasst werden können. Bei Radiosendern, die online aufgerufen werden können, ist der entsprechende Link zu speichern. Bei Radiosendern, die offline verfügbar sind, ist die Empfangsfrequenz zu speichern.

Klasse Category [1 Punkt]

Bei der Kategorie ist der Name der Kategorie vorhanden und ein Flag *top* soll angeben, ob die Kategorie zu den wichtigsten Kategorien zählt. Ein weiteres Flag *onlyPremium* soll angeben, ob diese Kategorie nur für Premium Benutzer zugänglich ist.

Klasse Rating [1 Punkt]

Jeder Benutzer kann zu den Podcasts Bewertungen abgeben. Die Bewertungszahl ist eine Zahl zwischen 1 und 5. Optional kann auch ein Bewertungstext hinzugefügt werden.

Klasse Favorite [1 Punkt]

Die Benutzer können eine oder mehrere Kategorien als Lieblignskategorien auszeichnen. Für jede Lieblingskategorie ist ein Beginndatum und ein Endedatum, welches nur bei Lieblingskategorien aus der Vergangenheit gesetzt ist, zu speichern.

Unittest CreateDatabaseTest [1 Punkt]

Der vorgegebene Test *CreateDatabaseTest* wird erfolgreich ausgeführt und kann die Datenbank mit den oben angegebenen Klassen erstellen.

In Summe sind für diese Aufgabe 9 Punkte zu erreichen.

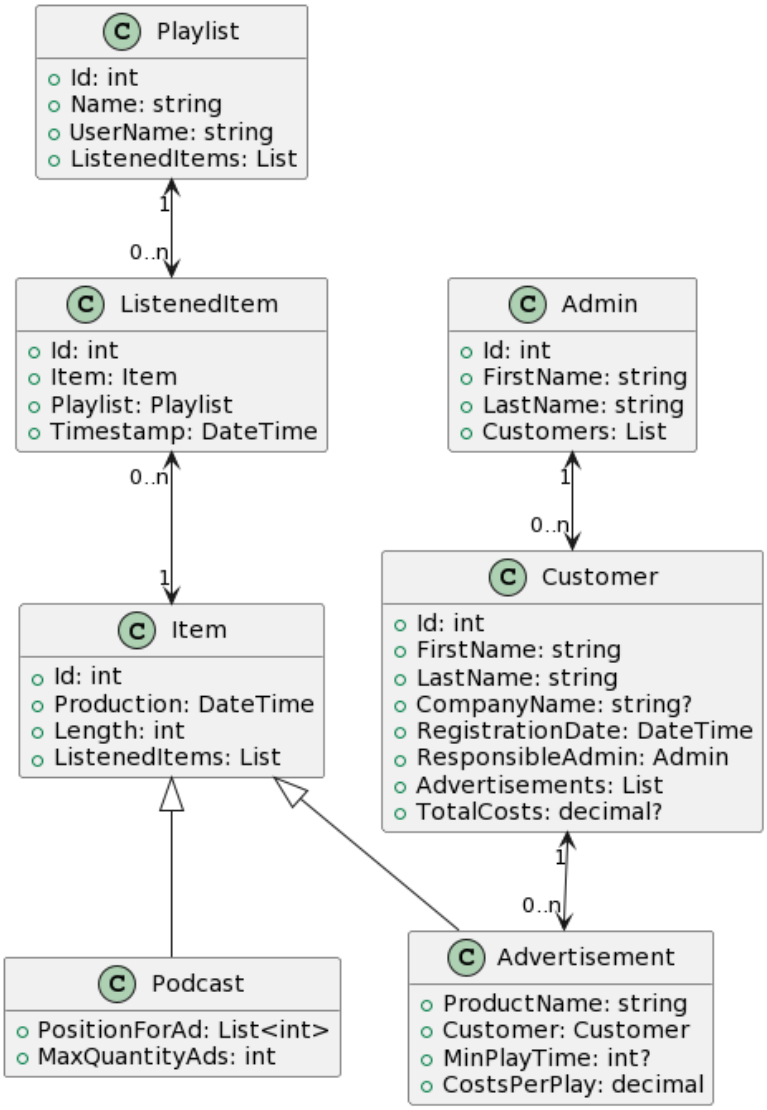
Teilaufgabe 2: Services und Unittests

Die Verwaltung der Podcasts und der während des Abspielens von Podcasts eingespielten Werbespots steht nun im Vordergrund. Die Podcasts können von den Benutzern über Wiedergabelisten (*Playlist*) angehört werden, wobei alle abgespielten Podcasts und Werbespots gespeichert werden (*ListenedItem*). Somit kann eine abgespielte Audiodatei (*Item*) ein Podcast (*Podcast*) oder ein Werbespot (*Advertisement*) sein.

Die Werbespots werden von den Kunden (*Customer*) beauftragt und jeder eingespielte Werbespot muss vom Kunden bezahlt werden. Jeder Kunde wird von einem zuständigen Mitarbeiter (*Admin*) betreut.

Die Klasse *Podcast* ist hier anders als in Aufgabe 1 implementiert. Das Datenmodell ist für Aufgabe 2 und 3 ident. Dennoch können die beiden Aufgaben völlig unabhängig voneinander bearbeitet werden.

Klassenmodell



Arbeitsauftrag

Im Projekt *SPG_Fachtheorie.Aufgabe2* ist im Ordner Model eine Service-Klasse *PodcastService* vorgegeben. Darin sind folgende 3 Methoden zu implementieren.

bool CalcTotalCosts(int customerId, DateTime begin, DateTime end)

Die Methode berechnet die Gesamtkosten für den übergebenen Kunden innerhalb des übergebenen Zeitraums. Für alle *ListenedItems*, die Referenzen zu Instanzen der Klasse *Advertisement* aufweisen und einen *Timestamp* innerhalb des übergebenen Zeitraums aufweisen, ist die Summe aus dem Property *CostsPerPlay* zu bilden und in *TotalCosts* des übergebenen Kunden zu speichern. Dabei sind verschiedene Überprüfungen durchzuführen:

1. Kann der Kunde mit der übergebenen ID nicht gefunden werden, soll *false* zurückgegeben werden.
2. Sind die Gesamtkosten für den Kunden bereits gespeichert (*TotalCosts* in *Customer* beinhaltet keinen *Null* Wert), soll *false* zurückgegeben werden.
3. Liegt der Endezeitpunkt vor dem Beginnzeitpunkt, soll *false* zurückgegeben werden.
4. Sind für den übergebenen Zeitraum keine angehörten Werbungen und somit keine *Advertisements* in den *ListenedItems* vorhanden, soll *false* zurückgegeben werden.
5. Die Methode liefert *true*, wenn die Gesamtkosten erfolgreich gespeichert werden konnten.

Unit Tests

Diese Methode ist durch Unit Tests zu prüfen.

1. Der Test *CalcTotalCosts_Invalid_CustomerId* beweist, dass eine nicht vorhandene *customerId* den Wert *false* liefert.
2. Der Test *CalcTotalCosts_TotalCosts_Already_Calculated* beweist, dass der Wert *false* zurückgegeben wird, wenn am Kunden bereits *TotalCosts* eingetragen sind.
3. Der Test *CalcTotalCosts_Invalid_TimePeriod* beweist, dass der Wert *false* zurückgegeben wird, wenn der Endzeitpunkt *end* vor dem Beginnzeitpunkt *begin* liegt.
4. Der Test *CalcTotalCosts_No_Advertisements* beweist, dass der Wert *false* zurückgegeben wird, wenn für den übergebenen Zeitraum keine *Advertisements* in den *ListenedItems* vorhanden sind.
5. Der Test *CalcTotalCosts_Success* beweist, dass die Methode die Gesamtkosten korrekt speichert. (Es wird *true* geprüft.)

int CalcQuantityAdditionalAds(int playlistId)

Mit Hilfe dieser Methode soll berechnet werden, wie viele Werbungen in der übergebenen *Playlist* zusätzlich untergebracht werden können. Folgendes ist zu implementieren:

- Es ist für alle *ListenedItems* der übergebenen *Playlist*, die Referenzen zu Instanzen der Klasse *Podcast* aufweisen, die Summe der maximal zulässigen Anzahl von Werbungen durch Summierung des Properties *MaxQuantityAds* zu ermitteln.
- Weiters ist für alle *ListenedItems* der übergebenen *Playlist*, die Referenzen zu Instanzen der Klasse *Advertisement* aufweisen, die Anzahl und somit die Anzahl der tatsächlichen Werbungen zu ermitteln.
- Schließlich ist die Differenz zwischen der maximal zulässigen Anzahl von Werbungen und der Anzahl der tatsächlichen Werbungen zu berechnen und als Rückgabewert der Methode zu implementieren.

Es sind folgende Bedingungen zu erfüllen:

1. Kann die Playlist mit der übergebenen ID (*playlistId*) nicht gefunden werden, soll der Zahlenwert „-3“ zurückgegeben werden.
2. Sind der Playlist keine *ListenedItems*, die Referenzen zu Instanzen der Klasse *Podcast* aufweisen, zugewiesen, soll der Zahlenwert „-2“ zurückgegeben werden.
3. Ist die maximal zulässige Anzahl von Werbungen geringer als die Anzahl der tatsächlichen Werbungen, soll der Zahlenwert „-1“ zurückgegeben werden.
4. Die Methode liefert in allen anderen Fällen die Differenz zwischen der maximal zulässigen Anzahl von Werbungen und der tatsächlichen Anzahl der Werbungen.

bool AddPostionForAd(int itemId, int position)

Bei einem Podcast soll mit Hilfe dieser Methode eine Positionangabe für eine Werbung im Property *PositionForAd* hinzugefügt werden um beispielweise 50 Sekunden nach dem Start des Podcasts eine Werbung einzuspielen. Dabei sind folgende Bedingungen zu erfüllen:

1. Kann der Podcast mit der übergebenen *ItemId* nicht gefunden werden, soll *false* zurückgegeben werden.
2. Ist die übergebene Position größer als die im Property *Length* gespeicherte Zahl, soll *false* zurückgegeben werden.
3. Wäre durch den neuen Eintrag die Anzahl der Einträge im Property *PositionForAd* größer als die im Property *MaxQuantityAds* gespeicherte Zahl, soll *false* zurückgegeben werden.
4. In allen anderen Fällen soll die Methode die übergebene Position als neuen Eintrag im Property *PositionForAd* hinzufügen und *true* zurückgeben.

Bewertung [18 Punkte]

Implementierung der Methode *CalcTotalCosts()* [5 Punkte]

- Bedingung 1 ist korrekt implementiert (Kunde kann nicht gefunden werden).
- Bedingung 2 ist korrekt implementiert (Gesamtkosen bereits vorhanden).
- Bedingung 3 ist korrekt implementiert (Endezeitpunkt vor dem Beginnzeitpunkt).
- Bedingung 4 ist korrekt implementiert (Werbungen nicht vorhanden).
- Bedingung 5 ist korrekt implementiert (Erfolgreicher Durchlauf).

Unit Tests zu *SubscribeCourse()* [5 Punkte]

- Unit Test *CalcTotalCosts_Invalid_CustomerId* beweist Bedingung 1.
- Unit Test *CalcTotalCosts_TotalCosts_Already_Calculated* beweist Bedingung 2.
- Unit Test *CalcTotalCosts_Invalid_TimePeriod* beweist Bedingung 3.
- Unit Test *CalcTotalCosts_No_Advertisements* beweist Bedingung 4.
- Unit Test *CalcTotalCosts_Success* beweist Bedingung 5.

Implementierung der Methode *CalcQuantityAdditionalAds()* [4 Punkte]

- Bedingung 1 ist korrekt implementiert (Playlist kann nicht gefunden werden).
- Bedingung 2 ist korrekt implementiert (Podcasts nicht vorhanden).
- Bedingung 3 ist korrekt implementiert (maximale Anzahl von Werbungen überschritten).
- Bedingung 4 ist korrekt implementiert (Erfolgreicher Durchlauf).

Implementierung der Methode *AddPostionForAd()* [4 Punkte]

- Bedingung 1 ist korrekt implementiert (Podcast kann nicht gefunden werden).
- Bedingung 2 ist korrekt implementiert (Positionsangabe ungültig).
- Bedingung 3 ist korrekt implementiert (maximale Anzahl von Einträgen bereits erreicht).
- Bedingung 4 ist korrekt implementiert (Erfolgreicher Durchlauf).

Teilaufgabe 3: Webapplikation

Das Datenmodell aus Aufgabe 2 soll nun herangezogen werden, um eine *Server Side Rendered Web Application* zu erstellen. Die Ausgaben der nachfolgenden Layouts können abweichen, müssen aber alle geforderten Features anbieten. Da es sich um Muster handelt, weichen die konkreten Daten in der zur Verfügung gestellten Musterdatenbank ab.

Das zu implementierende Front-End verfügt über Authentication. Diese ist vollständig implementiert. An dieser Stelle ist nichts zu erledigen. Lediglich der *AuthService* ist in den entsprechenden Controllern oder Pages über Dependency Injection bereitzustellen und zu verwenden.

Die Klasse *AuthService* stellt das *read only property AdminId* zur Verfügung. Dieses liefert die *Id* des angemeldeten Administrators (*Admin*). Damit Sie dieses Service in Ihren Controllern oder Pages nutzen können, müssen Sie die Klasse *AuthService* über Dependency Injection in den Konstruktor einfügen.

Arbeitsauftrag

Für das Datenmodell aus Aufgabe 2 soll nun ein Web-Portal implementiert werden. Für folgende Routen sind *Controller* und *Views* (MVC) ODER Razor-Pages vollständig zu implementieren. Informationen über die Bewertung der einzelnen Features entnehmen Sie aus dem Punkt *Bewertung*.

Seiten (Routen)

/Playlist/Index

Diese Seite stellt eine Liste aller *Playlist* Datensätze als Tabelle dar. In dieser Tabelle sind zu jedem Datensatz der *Name* der Playlist, der *UserName* und die Anzahl der enthaltenen Podcasts (Items von der Klasse *Podcast*) anzuzeigen. Die Ausgabe soll nach dem Namen der Playlist sortiert erfolgen.

/Customer/Index

Diese Seite soll die Kunden ausgeben, für die der angemeldete Admin zuständig ist (siehe Property *ResponsibleAdmin* in der Klasse *Customer*). Hinweis: Verwenden Sie das Property *AdminId* aus dem Service *AuthService*, um die *Id* des angemeldeten Administrators auszulesen.

Es sollen die Kunden (*Customer*) in Tabellenform mit *FirstName*, *LastName*, *CompanyName* und Links auf die Werbungen des Kunden (*Advertisement*) ausgegeben werden. Der Text der Links auf die Werbungen soll die *Id* und den *ProductName* der jeweiligen Werbung umfassen. Weist eine Werbung eine Länge (*Length*) von weniger als 5000 auf, dann ist der Link auf diese Werbung nicht anzuzeigen.

Klickt der angemeldete Admin auf den Link zu einer Werbung, wird die Detailseite der Werbung angezeigt (siehe übernächster Punkt). Der Link *Add Advertisement* führt zur Seite zum Hinzufügen einer Werbung (siehe nächster Punkt).

/Advertisement/Create

Klickt der Administrator auf der Kundenübersichtsseite auf *Add Advertisement*, so wird die Seite für das Hinzufügen einer Werbung geöffnet. Auf dieser Seite kann der Kunde als Dropdownliste ausgewählt werden. Es werden alle Kunden, für die der angemeldete Administrator verantwortlich ist, angeboten.

Der *ProductName*, der Produktionszeitpunkt (*Production*), die *Length*, die *MinPlayTime* und die *CostsPerPlay* können in den entsprechenden Eingabefeldern angegeben werden. Der *ProductName* muss zumindest 3 Zeichen umfassen, der Produktionszeitpunkt (*Production*) muss nach dem

RegistrationDate vom Kunden (*Customer*) liegen, die *Length* und die *CostsPerPlay* müssen angegeben werden. Falls einer dieser Bedingungen nicht erfüllt ist, ist eine entsprechende Fehlermeldung nach dem Klick auf *Speichern* auszugeben. Nach dem erfolgreichen Speichern ist die Seite unter der Route */Customer/Index* aufzurufen.

/Advertisement/Details/(id)

Zu jeder Werbung sollen die *Id*, der *ProductName*, der Kunde (*Customer*) im Format „*LastName FirstName*“, der Produktionszeitpunkt (*Production*), die *Length*, die *MinPlayTime*, die *CostsPerPlay* und die Anzahl der Aufrufe (Anzahl der *ListenedItems*) angezeigt werden.

Wird die angegebene ID nicht gefunden, so wird *404 not found* geliefert. Falls die ID zwar existiert, die Werbung aber nicht von einem Kunden des angemeldeten Administrators stammt, wird ebenfalls *404 not found* geliefert.

Bewertung [19 Punkte]

Route /Playlist/Index [3 Punkte]

- Es wird die Liste der *Playlist* Datensätze aus der Datenbank wie spezifiziert angezeigt.
- Die Anzahl der enthaltenen Podcasts (Items von der Klasse *Podcast*) wird für jedes Listenelement korrekt angezeigt.
- Die Darstellung ist nach dem Namen der Playlist sortiert.

Route /Customer/Index [5 Punkte]

- Das AuthService wird über Dependency Injection korrekt genutzt.
- Es wird die Liste der Kunden des angemeldeten Administrators aus der Datenbank angezeigt.
- Die spezifizierten Werbungen (*Advertisement*) erscheinen in der Liste.
- Jede Werbung besitzt einen Link auf die entsprechende Detailseite der Werbung.
- Der Link *Add Advertisement* verweist auf die Create Seite einer Werbung.

Route /Advertisement/Create [8 Punkte]

- Das AuthService wird über Dependency Injection korrekt genutzt.
- Die Dropdownliste wird mit allen Kunden des angemeldeten Administrators befüllt.
- Der Produktionszeitpunkt (*Production*) wird korrekt als Eingabefeld für Datumswerte dargestellt.
- Der *ProductName*, die *Length*, die *MinPlayTime* und die *CostsPerPlay* werden korrekt als Eingabefelder für Texte bzw. Zahlen dargestellt.
- Bei der Eingabe eines ungültigen Produktionszeitpunktes (der Zeitpunkt liegt nicht nach dem *RegistrationDate* vom *Customer*) wird nach dem Klicken auf *Speichern* eine entsprechende Fehlermeldung auf dieser Seite dargestellt.
- Bei der Eingabe von einem ProductName mit weniger als 3 Zeichen oder bei fehlender *Length* oder *CostsPerPlay* wird nach dem Klicken auf *Speichern* eine entsprechende Fehlermeldung auf dieser Seite dargestellt.
- Der Button *Speichern* speichert den Datensatz korrekt in der Datenbank.
- Der Button *Speichern* verweist nach dem Speichern auf die Route */Customer/Index* zurück.

Route /Advertisement/Details/(id) [3 Punkte]

- Die Daten zu einer Werbung werden wie spezifiziert angezeigt.
- Der Kundenname wird im Format „*LastName FirstName*“ angezeigt.
- Wird die ID nicht gefunden (ist also keine gültige Advertisement-ID) oder ist die ID nicht von einer Werbung eines Kunden des angemeldeten Administrators, wird 404 not found geliefert.