	Höhere technische Bundeslehr- und Versuchsanstalt für Textilindustrie und Datenverarbeitung
	Abteilungen: **Aufbaulehrgang für Informatik – Tag (SFKZ 8167)** **Kolleg für Informatik – Tag (SFKZ 8242)**

Reife- und Diplomprüfung 2022/23

Haupttermin September 2023

Aufgabenstellung für die Klausurprüfung

Jahrgang:	6AAIF, 6BAIF, 6CAIF, 6AKIF, 6BKIF
Prüfungsgebiet:	Programmieren und Software Engineering
Prüfungstag:	22. September 2023
Arbeitszeit:	5 Std. (300 Minuten)
Kandidaten/Kandidatinnen:	
Prüfer/Prüferin:	Mag. Manfred BALLUCH, Martin SCHRUTEK, MSc, Michael SCHLETZ, BEd
Aufgabenblätter:	15 Seiten inkl. Umschlagbogen

Inhaltsübersicht der Einzelaufgaben im Umschlagbogen
(Unterschrift des Prüfers/der Prüferin auf den jeweiligen Aufgabenblättern)

Das versiegelte Kuvert mit der der Aufgabenstellung wurde geöffnet von:

Name: _____ Unterschrift: _____

Datum: _____ Uhrzeit: _____

Zwei Zeugen (Kandidaten/Kandidatinnen)

Name: _____ Unterschrift: _____

Name: _____ Unterschrift: _____

Geprüft: Wien, am

_____ RS. _____
Mag. Heidi Steinwender
Abteilungsvorständin

Dr. Gerhard Hager
Direktor

Genehmigt:

Wien, am _____ RS. _____
MinR Mag. Gabriele Winkler Rigler

Klausurprüfung (Fachtheorie) aus Programmieren und Software Engineering

im Haupttermin September 2022/23

für den Aufbaulehrgang für Informatik – Tag (SFKZ 8167)
für das Kolleg für Informatik – Tag (SFKZ 8242)

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

Für das 6. Semester des Aufbaulehrganges und das Kolleg am 22. September 2023.

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 1.5 Stunden für Aufgabe 1, 1.5 Stunden 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 6 zur Verfügung.

Mit der Software SQLite Studio können Sie sich zur generierten Datenbank verbinden und Werte für Ihre Unittests ablesen. Die Programmdatei befindet sich in *C:/Scratch/SQLiteStudio/SQLiteStudio.exe*.

Zusätzlich wird ein implementiertes Projekt aus dem Unterricht ohne Kommentare bereitgestellt, wo Sie die Parameter von benötigten Frameworkmethoden nachsehen können.

Pfade

Die Solution befindet sich im Ordner *P:/SPG_Fachtheorie*. Sie liegt direkt am Netzlaufwerk, d. h. es ist kein Sichern der Arbeit erforderlich.

Damit das Kompilieren schneller geht, ist der Ausgabepfad der Projekte auf *C:/Scratch/(Projekt)* umgestellt. Das ist zum Auffinden der generierten Datenbank wichtig.

Nullable Reference Types

Das Feature *nullable reference types* wurde in den Projekten aktiviert. Zusätzlich werden Compilerwarnungen als Fehler definiert. Sie können daher das Projekt nicht kompilieren, wenn z. B. ein nullable Warning entsteht. Achten Sie daher bei der Implementierung, dass Sie Nullprüfungen, etc. korrekt durchführen.

SQLite Studio

Zur Betrachtung der Datenbank in *C:/Scratch/Aufgabe1_Test/Debug/net6.0* bzw. *C:/Scratch/Aufgabe2_Test/Debug/net6.0* steht die Software *SQLite Studio* zur Verfügung. Die exe Datei befindet sich in *C:/Scratch/SPG_Fachtheorie/SQLiteStudio/SQLiteStudio.exe*. Mittels *Database - Add a Database* kann die erzeugte Datenbank (*appointments.db* oder *sticker.db*) geöffnet werden.

Auswählen und Starten der Webapplikation (Visual Studio)

In der Solution gibt es 2 Projekte: *Aufgabe3* (MVC Projekt) und *Aufgabe3RazorPages*. Sie können wählen, ob Sie die Applikation mit MVC oder RazorPages umsetzen wollen. Entfernen Sie das nicht benötigte Projekt aus der Solution und lege das verwendete Projekt als Startup Projekt fest (Rechtsklick auf das Projekt und *Set as Startup Project*).

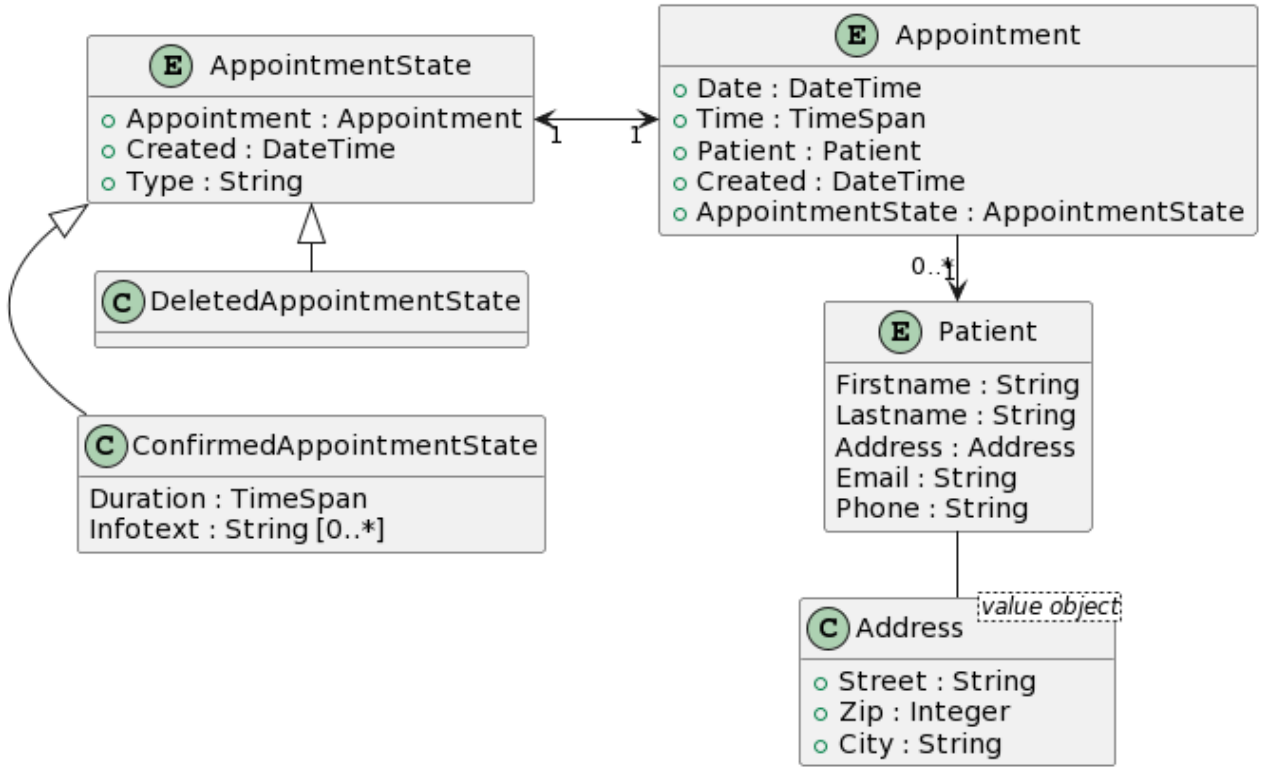
Beim ersten Start erscheint die Frage *Would you like to trust the ASP.NET Core SSL Certificate?* Wähle *Yes* und *Don't ask me again*. Bestätigen Sie den nachfolgenden Dialog zur Zertifikatsinstallation.

Teilaufgabe 1: Erstellen von EF Core Modelklassen

Terminreservierung für Patientinnen und Patienten

Für eine Arztpraxis soll ein Reservierungssystem entwickelt werden, mit dessen Hilfe sich Patientinnen und Patienten zu Behandlungen anmelden können. Es ist vorgesehen, dass sich die Patientinnen und Patienten mit den Grunddaten (Name, Adresse und Kontaktdaten) registrieren. Danach können sie ein Datum wählen und einen Termin (*Appointment*) buchen. Der Termin (*Appointment*) hat einen Status (*AppointmentState*). Der Arzt bzw. die Ärztin kann den Termin bestätigen oder ablehnen. Ein bestätigter Termin hat den Status *ConfirmedAppointmentState*. Bei einer Bestätigung wird eine Dauer eingetragen. Wird der Termin storniert, so wird der Status *DeletedAppointmentState* zugewiesen.

Das UML Klassendiagramm für so ein System kann so aussehen:



Arbeitsauftrag

Erstellung der Modelklassen

Im Projekt *SPG_Fachtheorie.Aufgabe1* befinden sich im Ordner *Model* leere Klassendefinitionen. Bilden Sie jede Klasse gemäß dem UML Diagramm ab, sodass EF Core diese persistieren kann. Beachten Sie folgendes:

- Wählen Sie selbst notwendige Primary keys.
- Definieren Sie Stringfelder mit vernünftigen Maximallängen (z. B. 255 Zeichen für Namen, etc.).
- Durch das nullable Feature werden alle Felder als *NOT NULL* angelegt. Verwenden Sie daher nullable Typen für optionale Felder. Sie sind mit *[0..*]* im Diagramm gekennzeichnet.
- Address ist ein *value object*. Stellen Sie durch Ihre Definition sicher, dass kein Mapping diese Klasse in eine eigene Datenbanktabelle durchgeführt wird.
- Legen Sie Konstruktoren mit allen Feldern an. Erstellen Sie die für EF Core notwendigen default Konstruktoren als *protected*.
- *AppointmentState* ist mit einer 1:1 Beziehung mit *Appointment* verbunden und das *child entity*. Daher hat es kein Appointment im Konstruktor.
- Das Feld *Type* in *AppointmentState* ist als Discrimiator Feld vorgesehen. Es wird von EF Core initialisiert, diese sind natürlich nicht im Konstruktor aufzunehmen. Mappen Sie in der Konfiguration das Discriminator Feld in *AppointmentState* in das Feld *Type*
- Implementieren Sie die Vererbung korrekt, sodass eine (1) Tabelle *AppointmentState* entsteht.
- Legen Sie die erforderlichen DB Sets im Datenbankcontext an.

Verfassen von Tests

Im Projekt *SPG_Fachtheorie.Aufgabe1.Test* ist in *Aufgabe1Test.cs* der Test *CreateDatabaseTest* vorgegeben. Er muss erfolgreich durchlaufen und die Datenbank erzeugen. Sie können die erzeugte Datenbank in *C:/Scratch/Aufgabe1_Test/Debug/net6.0/appointments.db* in SQLite Studio öffnen.

Implementieren Sie folgende Tests selbst, indem Sie die minimalen Daten in die (leere) Datenbank schreiben. Leeren Sie immer vor dem *Assert* die nachverfolgten Objekte mittels *db.ChangeTracker.Clear()*.

- Der Test *AddPatientSuccessTest* beweist, dass Sie einen Patienten mit Adresse in die Datenbank einfügen können.
- Der Test *AddAppointmentSuccessTest* beweist, dass Sie einen Termin (Appointment) eines Patienten speichern können. Legen Sie dafür eine Instanz von *AppointmentState* an.

- Der Test *ChangeAppointmentStateToConfirmedSuccessTest* beweist, dass Sie den Status eines bestehenden Appointment Eintrages auf *Confirmed* ändern können. Gehen Sie dabei so vor:
  - Fügen Sie ein neues Appointment mit einer Instanz von *AppointmentState* ein.
  - Speichern Sie diesen Datensatz und leeren Sie mit *db.ChangeTracker.Clear()* die verfolgten Objekte.
  - Laden Sie nun das gespeicherte Appointment **mit inkludierter Navigation *AppointmentState*** aus der Datenbank und weisen als *AppointmentState* eine Instanz von *ConfirmedAppointmentState* zu.
  - Verwenden Sie als Assert Bedingung `Assert.True(db.AppointmentStates.Count() == 1);`. Es darf nach der Änderung *nur ein Datensatz* in *AppointmentState* stehen, da es sich um eine 1:1 Beziehung zu Appointment handelt.

Bewertung (25 P, 37.3% der Gesamtpunkte)

Jedes der folgenden Kriterien wird mit 1 Punkt bewertet.

- Die Klasse *Patient* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
- Die Stringfelder der Klasse *Patient* verwenden sinnvolle Längenbegrenzungen.
- Die Klasse *Patient* wurde korrekt im DbContext registriert.
- Die Klasse *Patient* besitzt ein korrekt konfiguriertes value object *Address*.
- Die Klasse *Address* beinhaltet die im UML Diagramm abgebildeten Felder und einen korrekten Konstruktor.
- Die Stringfelder der Klasse *Address* verwenden sinnvolle Längenbegrenzungen.
- Die Klasse *Address* ist ein value object, d. h. sie besitzt keine Schlüsselfelder.
- Die Klasse *AppointmentState* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
- Die Klasse *AppointmentState* besitzt eine korrekt konfigurierte 1:1 Beziehung zu *Appointment*.
- Die Klasse *AppointmentState* wurde korrekt im DbContext registriert.
- Die Klasse *ConfirmedAppointmentState* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
- Die Klasse *ConfirmedAppointmentState* erbt von *AppointmentState*.
- Die Klasse *AppointmentState* wurde korrekt im DbContext registriert.
- Die Klasse *DeletedAppointmentState* erbt von *AppointmentState*.
- Die Klasse *DeletedAppointmentState* besitzt einen korrekten public bzw. protected Konstruktor.
- Die Klasse *AppointmentState* wurde korrekt im DbContext registriert.
- Die Klasse *Appointment* beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.

- Die Klasse *Appointment* besitzt eine korrekt konfigurierte 1:1 Beziehung zu *Appointment*.
- Die Klasse *Appointment* wurde korrekt im DbContext registriert.
- Der Test *AddPatientSuccessTest* ist korrekt aufgebaut.
- Der Test *AddPatientSuccessTest* läuft erfolgreich durch.
- Der Test *AddAppointmentSuccessTest* ist korrekt aufgebaut.
- Der Test *AddAppointmentSuccessTest* läuft erfolgreich durch.
- Der Test *ChangeAppointmentStateToConfirmedSuccessTest* ist korrekt aufgebaut.
- Der Test *ChangeAppointmentStateToConfirmedSuccessTest* läuft erfolgreich durch.

Teilaufgabe 2: Services und Unittests

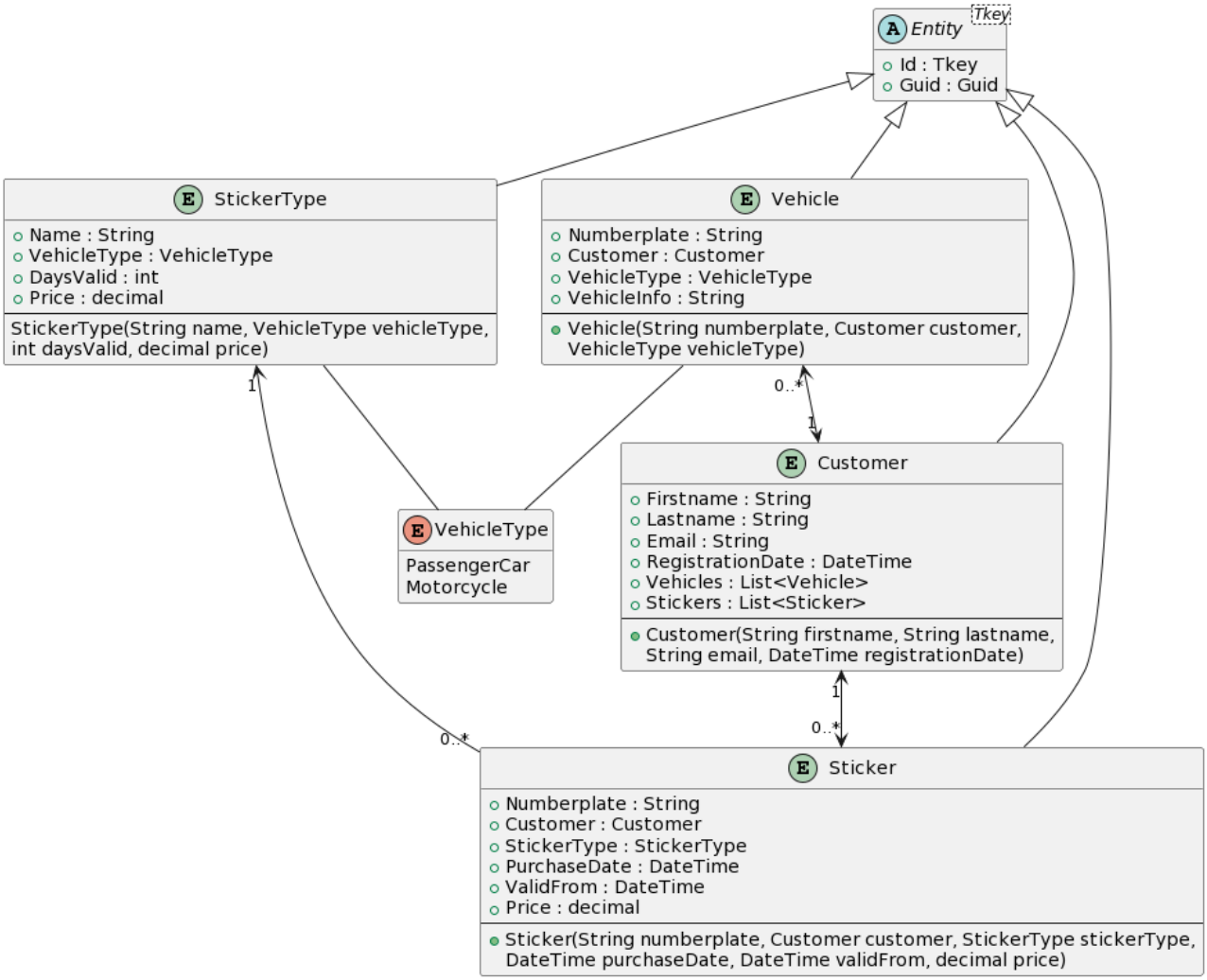
In Österreich muss für die Benützung der Autobahnen und Schnellstraßen eine Gebühr bezahlt werden. Durch den Kauf einer "Vignette" (im Modell englisch *Sticker* genannt) wird diese Gebühr bezahlt. Es gibt Vignetten, die 10 Tage, 2 Monate oder 1 Jahr gültig sind. Es gibt für Autos (PKW) und Motorräder verschiedene Tarife.

Tarife 2023

Fahrzeugart	Jahres-Vignette	2-Monats-Vignette	10-Tages-Vignette
Auto und Kfz bis 3,5 t hzG	€ 96,40 online kaufen →	€ 29,00 online kaufen →	€ 9,90 online kaufen →
Motorrad	€ 38,20 online kaufen →	€ 14,50 online kaufen →	€ 5,80 online kaufen →

Quelle: https://www.asfinag.at/maut-vignette/vignette, abgerufen am 20.5.2023

Im letzten Jahr wurde auch dieser Bereich digitalisiert. So können die Verkehrskameras das Kennzeichen erkennen und abfragen, ob eine gültige Vignette existiert. Für die Implementierung des Services werden Ihnen eine Datenbank und die Modelklassen vorgegeben. Sie haben folgenden Aufbau:



Customer: *Customer* speichert die Daten eines Kunden, der sich auf der Verkaufsseite registriert hat.
Vehicle: Der Kunde kann seine Fahrzeuge für den schnelleren Kauf mit Kennzeichen und Fahrzeugart erfassen. Diese Klasse speichert die erfassten Fahrzeuge des Kunden.
Sticker: Entspricht der Vignette für ein konkretes Fahrzeug. Sie ist einem Kunden fix zugeordnet. Da ein Kunde auch für nicht registrierte Fahrzeuge Vignetten kaufen kann, wird das Kennzeichen in dieser Klasse gespeichert und nicht auf *Vehicle* verwiesen. Der Preis, zu dem der Kunde die Vignette gekauft hat, wird hier ebenfalls gespeichert.
StickerType: Hier werden die zur Auswahl stehenden Vignettenarten (10 Tage PKW, ...) mit dem aktuellen Verkaufspreis gespeichert.

Die SQLite Datenbank wird durch den Unittest *CreateDatabaseTest* im Projekt *SPG_Fachtheorie.Aufgabe2.Test* erzeugt und mit Musterdaten befüllt. Sie wird in **C:/Scratch/Aufgabe2_Test/Debug/net6.0/sticker.db** geschrieben und kann mit SQLite Studio geöffnet werden.

Arbeitsauftrag

Implementierung von Servicemethoden

Im Projekt *SPG_Fachtheorie.Aufgabe2* befindet sich die Klasse *Services/StickerService.cs*. Es sind 2 Methoden zu implementieren:

bool HasPermission(string numberplate, DateTime dateTime, VehicleType carType)

Diese Methode soll bestimmen, ob ein durch die Kamera erkanntes Verkehrszeichen (*Numberplate*) eine gültige Vignette besitzt. Zusätzlich erkennen die Kameras noch den Fahrzeugtyp (*PassengerCar* für PKW oder *Motorcycle* for Motorrad). Damit eine Vignette gültig ist, müssen 3 Bedingungen zutreffen:

- Das gespeicherte Kennzeichen (*Numberplate* in *Sticker*) muss dem übergebenen Wert von *numberplate* entsprechen.
- Der Typ der Vignette (*StickerType.VehicleType* in *Sticker*) muss dem übergebenen Wert von *carType* entsprechen.
- Das übergebene Datum muss im Gültigkeitszeitraum (*ValidFrom* in *Sticker*) der Vignette entsprechen.

In *StickerType.DaysValid* ist die Gültigkeitsdauer in Tagen gespeichert. Verwenden Sie *AddDays()*, um das Ende der Gültigkeit zu ermitteln.

Die Methode liefert *true*, wenn ein Datensatz mit den oben beschriebenen Kriterien existiert. Die Methode liefert *false*, wenn kein Datensatz mit den oben beschriebenen Kriterien existiert.

Hinweis: Verwende eine (1) *Any* Abfrage mit allen Kriterien.

List<SaleStatistics> CalcSaleStatistics(int year)

In der Klasse *StickerService* ist ein record für die Statistik einer Vignettenart (10 Tage PKW, 2 Monate Motorrad, ...) definiert:

```
public record SaleStatistics(string StickerTypeName, decimal
TotalRevenue);
```

Die Methode *CalcSaleStatistics* soll den Umsatz einer Vignettenart für das übergebene Jahr aufsummieren. Der Umsatz berechnet sich aus dem Preis, den der Kunde für die Vignette bezahlt hat (Feld *Price* in *Sticker*). Für die Filterung ist das Jahr des Verkaufsdatums (*PurchaseDate* in *Sticker*) heranzuziehen.

Hinweis: Verwenden Sie *GroupBy* in LINQ. SQLite unterstützt kein Aufsummieren von *decimal* Feldern mit EF Core. Führen Sie daher vor und nach der Summierung mit `(decimal)g.Sum(s => (double)s.Price)` einen Typcast durch.
In der Testklasse *StickerServiceTests* im Projekt *SPG_Fachtheorie.Aufgabe2.Test* ist der Test *CalcSaleStatisticsTest* bereits vorgegeben. Nutzen Sie diesen Test, um Ihre Implementierung zu prüfen.

Testen der Methode *HasPermission*

Schreiben Sie im Projekt *SPG_Fachtheorie.Aufgabe2.Test* in die Klasse *StickerServiceTests* Unittests, die die Korrektheit von *HasPermission* prüfen. Sie können mit der Methode *GetSeededDbContext* die Datenbank mit Musterdaten generieren.

- *HasPermissionReturnsFalseIfNumberplateIsInvalidTest* prüft, ob die Methode *false* zurückliefert, wenn ein Kennzeichen übergeben wurde, das keine gültige Vignette hat.
- *HasPermissionReturnsFalseIfCarTypeIsInvalidTest* prüft, ob die Methode *false* zurückliefert, wenn ein gültiges Kennzeichen und ein gültiger Zeitraum, aber ein falscher Fahrzeugtyp übergeben wurde. Das tritt dann auf, wenn jemand für einen PKW eine Motorradvignette gekauft hat.
- *HasPermissionReturnsFalseIfDateTimeNotInValidTimespanTest* prüft, ob die Methode *false* zurückliefert, wenn für den übergebenen Zeitbereich keine gültige Vignette für ein existierendes Kennzeichen existiert. Das tritt dann auf, wenn jemand mit einer abgelaufenen Vignette die Autobahn benützen will.
- *HasPermissionReturnsTrueIfSuccessTest* prüft, ob die Methode *true* zurückliefert, wenn eine korrekte Benützungsberechtigung vorliegt.

Bewertung (15 P, 22.4% der Gesamtpunkte)

Jedes der folgenden Kriterien wird mit 1 Punkt bewertet.

- Die Methode *HasPermission* berücksichtigt den Parameter *numberplate* in der Abfrage korrekt.
- Die Methode *HasPermission* berücksichtigt den Parameter *dateTime* in der Abfrage korrekt.
- Die Methode *HasPermission* berücksichtigt den Parameter *carType* in der Abfrage korrekt.
- Die Methode *HasPermission* verwendet LINQ und keine imperativen Konstrukte wie Schleifen, ...
- Die Methode *CalcSaleStatistics* berücksichtigt den Parameter *year* in der Abfrage korrekt.
- Die Methode *CalcSaleStatistics* gruppiert korrekt nach dem StickerType.
- Die Methode *CalcSaleStatistics* verwendet LINQ und keine imperativen Konstrukte wie Schleifen, ...

- Der Unittest *HasPermissionReturnsFalseIfNumberplateIsInvalidTest* hat den korrekten Aufbau (arrange, act, assert).
- Der Unittest *HasPermissionReturnsFalseIfNumberplateIsInvalidTest* läuft erfolgreich durch.
- Der Unittest *HasPermissionReturnsFalseIfCarTypeIsInvalidTest* hat den korrekten Aufbau (arrange, act, assert).
- Der Unittest *HasPermissionReturnsFalseIfCarTypeIsInvalidTest* läuft erfolgreich durch.
- Der Unittest *HasPermissionReturnsFalseIfDateTimeNotInValidTimespanTest* hat den korrekten Aufbau (arrange, act, assert).
- Der Unittest *HasPermissionReturnsFalseIfDateTimeNotInValidTimespanTest* läuft erfolgreich durch.
- Der Unittest *HasPermissionReturnsTrueIfSuccessTest* hat den korrekten Aufbau (arrange, act, assert).
- Der Unittest *HasPermissionReturnsTrueIfSuccessTest* läuft erfolgreich durch.

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.

Arbeitsauftrag

Implementieren Sie die folgenden Seiten im Projekt *SPG_Fachtheorie.Aufgabe3*, falls Sie mit MVC arbeiten möchten. Verwenden Sie *SPG_Fachtheorie.Aufgabe3.RazorPages*, falls Sie mit Razor Pages arbeiten möchten. Löschen Sie das nicht benötigte Projekt aus der Solution und legen Ihr gewünschtes Webprojekt als Startprojekt fest.

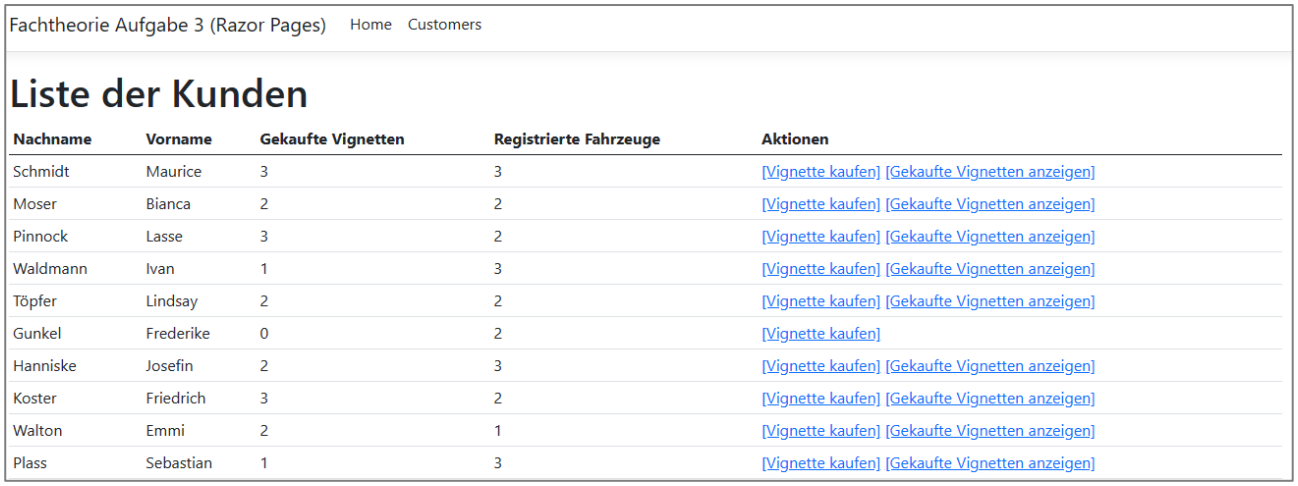
Seite /Customers/Index

Diese Seite soll alle Kunden der Tabelle *Customers* auflisten. In der Tabelle wird beim Kunden die Anzahl der gekauften Vignetten ausgegeben. Das sind die Datensätze in der Tabelle *Stickers*, die dem Kunden gehören. Die Anzahl der registrierten Fahrzeuge ermittelt sich aus der Anzahl der Datensätze in der Tabelle *Vehicles*, die dem Kunden gehören.

In der Spalte *Aktionen* befinden sich 2 Links:

- Der Link *Vignette kaufen* verweist auf die Seite */Stickers/Add/(customerGuid)*.
- Der Link *Gekaufte Vignetten anzeigen* verweist auf die Seite */Stickers/Index/(customerGuid)*. Dieser Link wird nur angezeigt, falls der Kunde schon Vignetten gekauft hat.

Die Ausgabe kann so aussehen. Die Daten sind bereits aus der "echten" Datenbank abgefragt, d. h. die Werte sollen bei Ihnen ident sein.

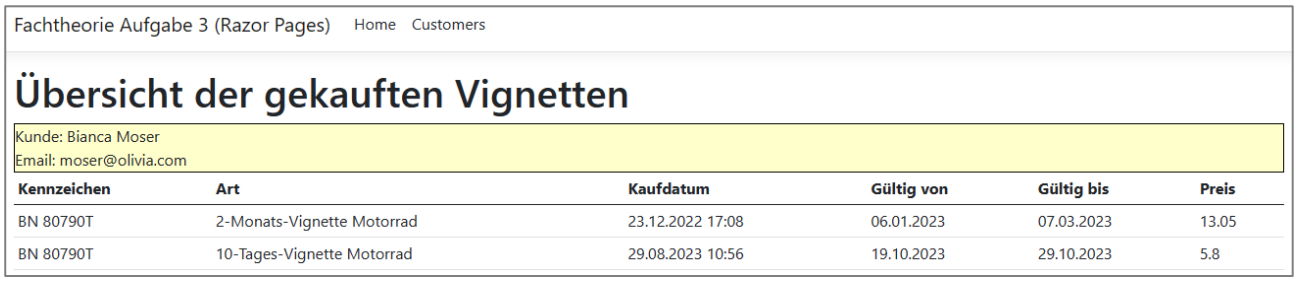


Seite /Stickers/Index/(customerGuid)

Klickt man in der Kundenübersicht auf *Gekaufte Vignetten anzeigen*, sollen die gekauften Vignetten des Kunden aufgelistet werden. Am Anfang der Seite werden die Kundendaten mit Vorname, Nachname und E-Mail nochmals ausgegeben.

Pro gekaufter Vignette wird das Kennzeichen, die Vignettenart (*StickerType.Name*), das Kaufdatum (*Sticker.PurchaseDate*) und die Informationen zur Gültigkeit (*Sticker.ValidFrom*) ausgegeben. Der Wert für *GültigBis* muss aus den Informationen von *Sticker.ValidFrom* und *StickerType.DaysValid* berechnet werden. Verwenden Sie hierfür die *AddDays()* Methode.

Existiert die übergebene Kunden GUID nicht, so wird auf die Seite */Customers/Index* zurückverwiesen.



Seite /Stickers/Add/(customerGuid)

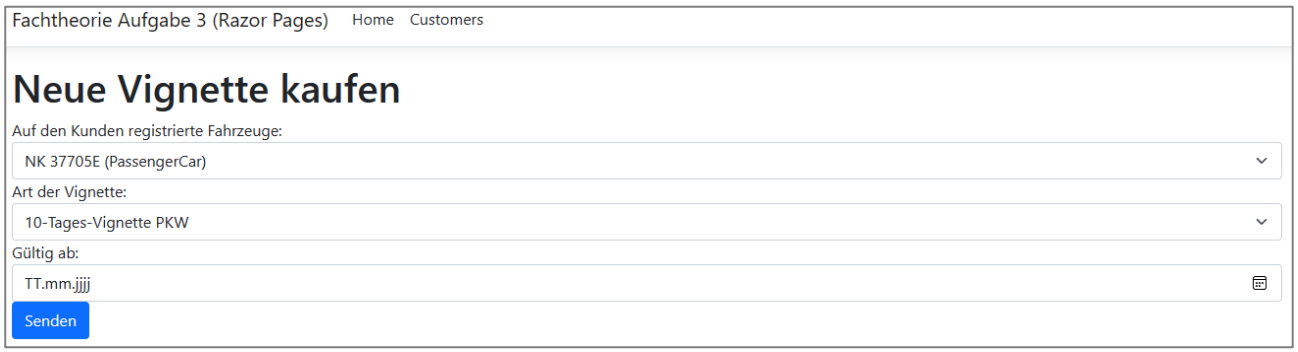
Klickt man in der Kundenübersicht auf *Vignette kaufen*, soll ein Eingabeformular für den Kauf einer neuen Vignette angezeigt werden. Die Seite besteht aus 2 Dropdownfeldern:

- Das Dropdownfeld *registrierte Fahrzeuge* listet die Kennzeichen der registrierten Fahrzeuge aus der Tabelle *Vehicles* auf. Verwenden Sie nur Fahrzeuge, die auch dem Kunden gehören. Verwenden Sie das Feld *Vehicle.VehicleInfo*, um den Text für das Dropdownfeld zu ermitteln.
- Das Dropdownfeld *Art der Vignette* zeigt alle Datensätze der Tabelle *StickerTypes* an.
- Das Gültigkeitsfeld soll als Datumsfeld ohne Zeit gerendert werden (Typ *date*).

Für die Validierung sind 2 Bedingungen zu prüfen:

- Das Gültigkeitsdatum darf nicht in der Vergangenheit liegen. Es ist aber möglich, am selben Tag eine Vignette zu kaufen, d. h. am 23.9.2023 darf das Gültigkeitsdatum 23.9.2023 verwendet werden. Verwenden Sie *DateTime.Now.Date* zur Ermittlung des aktuellen Datums (ohne Zeitkomponente).
- Der Typ der gekauften Vignette muss dem Fahrzeugtyp entsprechen. Es darf nicht möglich sein, dass ein PKW aus der Liste der Fahrzeuge gewählt und eine Vignette für ein Motorrad gekauft wird. Verwenden Sie die Felder *Vehicle.VehicleType* bzw. *StickerType.VehicleType* für die Überprüfung.
- Bei Fehlern zeigen Sie eine entsprechende Meldung auf dieser Seite an.

Wenn alle Bedingungen erfüllt sind, wird der Datensatz in der Tabelle *Stickers* gespeichert. Verwenden Sie *DateTime.Now* für das Feld *PurchaseDate*. Der Preis wird aus dem ausgewählten *StickerType* Datensatz geladen. Wurde der Datensatz erfolgreich gespeichert, verweisen Sie auf die Seite */Stickers/Index/(customerGuid)* zurück. Hinweis:
Mit `RedirectToPage("/Stickers/Index", new { Guid = ... });` (Razor Pages)
bzw. *RedirectToAction* (MVC) können Sie einen Routingparameter beim Redirect mitgeben.



Bewertung (27 P, 40.3% der Gesamtpunkte)

- Die Seite */Customers/Index* besitzt eine korrekte Dependency Injection der Datenbank.
- Die Seite */Customers/Index* fragt die benötigten Informationen aus der Datenbank korrekt ab.
- Die Seite */Customers/Index* zeigt den Vor- und Nachnamen aller Kunden an.
- Die Seite */Customers/Index* zeigt die Anzahl der gekauften Vignetten aller Kunden an.
- Die Seite */Customers/Index* zeigt die Anzahl der registrierten Fahrzeuge aller Kunden an.
- Die Seite */Customers/Index* verlinkt korrekt auf die Seite */Stickers/Add/(customerGuid)*.
- Die Seite */Customers/Index* verlinkt korrekt auf die Seite */Stickers/Index/(customerGuid)*.
- Die Seite */Customers/Index* zeigt den Link *Gekaufte Vignetten anzeigen* nur an, wenn auch gekaufte Vignetten existieren.
- Die Seite */Stickers/Index* besitzt eine korrekte Dependency Injection der Datenbank.
- Die Seite */Stickers/Index* besitzt einen Routingparameter für die Kunden GUID.
- Die Seite */Stickers/Index* fragt die benötigten Informationen aus der Datenbank korrekt ab.
- Die Seite */Stickers/Index* verweist auf die Seite */Customer/Index*, falls der Kunde nicht existiert.
- Die Seite */Stickers/Index* ermittelt den Wert für *Gültig bis* korrekt.
- Die Seite */Stickers/Index* zeigt Vor- und Nachnamen und E-Mail des Kunden an.
- Die Seite */Stickers/Index* zeigt die Felder Kennzeichen, Vignettentyp, Kaufdatum, gültig von, gültig bis und Preis an.
- Die Seite */Stickers/Add* besitzt eine korrekte Dependency Injection der Datenbank.
- Die Seite */Stickers/Add* besitzt einen Routingparameter für die Kunden GUID.
- Die Seite */Stickers/Add* fragt die benötigten Informationen für die Liste der registrierten Fahrzeuge korrekt ab.
- Die Seite */Stickers/Add* fragt die benötigten Informationen für die Liste Vignettenarten korrekt ab.
- Die Seite */Stickers/Add* stellt die registrierten Fahrzeuge des Kunden als Dropdownfeld dar.
- Die Seite */Stickers/Add* stellt die Vignettenarten als Dropdownfeld dar.
- Die Seite */Stickers/Add* prüft, ob das Gültigkeitsdatum in der Zukunft liegt.
- Die Seite */Stickers/Add* gibt Validierungsfehler für das Gültigkeitsdatum korrekt aus.
- Die Seite */Stickers/Add* prüft, ob der Fahrzeugtyp der Vignette dem des ausgewählten Fahrzeuges entspricht.
- Die Seite */Stickers/Add* gibt Validierungsfehler für den Vignettentyp korrekt aus.
- Die Seite */Stickers/Add* speichert den Datensatz korrekt in der Datenbank.
- Die Seite */Stickers/Add* verweist nach der Speicherung auf die Seite */Stickers/Index/(customerGuid)*

Haupttermin September 2023

PROGRAMMIEREN UND SOFTWARE ENGINEERING

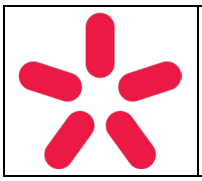
Aufbaulehrgang für Informatik – Tag (SFKZ 8167)
Kolleg für Informatik – Tag (SFKZ 8242)

Bewertungsblatt (vom Prüfer auszufüllen)

Für jede erfüllte Teilaufgabe gibt es 1 Punkt. In Summe sind also 67 Punkte zu erreichen. Für eine Berücksichtigung der Jahresnote müssen mindestens 30 % der Gesamtpunkte erreicht werden. Für eine positive Beurteilung der Klausur müssen mindestens 50 % der Gesamtpunkte erreicht werden.

Beurteilungsstufen:
67 – 59 Punkte: Sehr gut, 58 – 51 Punkte: Gut, 50 – 42 Punkte: Befriedigend, 41 – 34 Punkte: Genügend

Aufgabe 1 (jew. 1 Punkt, 25 in Summe)	Erf.	Nicht erf.
Die Klasse Patient beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.		
Die Stringfelder der Klasse Patient verwenden sinnvolle Längenbegrenzungen.		
Die Klasse Patient wurde korrekt im DbContext registriert.		
Die Klasse Patient besitzt ein korrekt konfiguriertes value object Address.		
Die Klasse Address beinhaltet die im UML Diagramm abgebildeten Felder und einen korrekten Konstruktor.		
Die Stringfelder der Klasse Address verwenden sinnvolle Längenbegrenzungen.		
Die Klasse Address ist ein value object, d. h. sie besitzt keine Schlüsselfelder.		
Die Klasse AppointmentState beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.		
Die Klasse AppointmentState besitzt eine korrekt konfigurierte 1:1 Beziehung zu Appointment.		
Die Klasse AppointmentState wurde korrekt im DbContext registriert.		
Die Klasse ConfirmedAppointmentState beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.		
Die Klasse ConfirmedAppointmentState erbt von AppointmentState.		
Die Klasse AppointmentState wurde korrekt im DbContext registriert.		
Die Klasse DeletedAppointmentState erbt von AppointmentState.		
Die Klasse DeletedAppointmentState besitzt einen korrekten public bzw. protected Konstruktor.		
Die Klasse AppointmentState wurde korrekt im DbContext registriert.		
Die Klasse Appointment beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.		
Die Klasse Appointment besitzt eine korrekt konfigurierte 1:1 Beziehung zu Appointment.		
Die Klasse Appointment wurde korrekt im DbContext registriert.		
Der Test AddPatientSuccessTest ist korrekt aufgebaut.		
Der Test AddPatientSuccessTest läuft erfolgreich durch.		
Der Test AddAppointmentSuccessTest ist korrekt aufgebaut.		
Der Test AddAppointmentSuccessTest läuft erfolgreich durch.		
Der Test ChangeAppointmentStateToConfirmedSuccessTest ist korrekt aufgebaut.		
Der Test ChangeAppointmentStateToConfirmedSuccessTest läuft erfolgreich durch.		



Haupttermin September 2023

PROGRAMMIEREN UND SOFTWARE ENGINEERING

Aufbaulehrgang für Informatik – Tag (SFKZ 8167)

Kolleg für Informatik – Tag (SFKZ 8242)

Aufgabe 2 (jew. 1 Punkt, 15 in Summe)	Erf.	Nicht erf.
Die Methode HasPermission berücksichtigt den Parameter numberplate in der Abfrage korrekt.		
Die Methode HasPermission berücksichtigt den Parameter dateTime in der Abfrage korrekt.		
Die Methode HasPermission berücksichtigt den Parameter carType in der Abfrage korrekt.		
Die Methode HasPermission verwendet LINQ und keine imperativen Konstrukte wie Schleifen, ...		
Die Methode CalcSaleStatistics berücksichtigt den Parameter year in der Abfrage korrekt.		
Die Methode CalcSaleStatistics gruppiert korrekt nach dem StickerType.		
Die Methode CalcSaleStatistics verwendet LINQ und keine imperativen Konstrukte wie Schleifen, ...		
Der Unittest HasPermissionReturnsFalseIfNumberplateIsInvalidTest hat den korrekten Aufbau (arrange, act, assert).		
Der Unittest HasPermissionReturnsFalseIfNumberplateIsInvalidTest läuft erfolgreich durch.		
Der Unittest HasPermissionReturnsFalseIfCarTypeIsInvalidTest hat den korrekten Aufbau (arrange, act, assert).		
Der Unittest HasPermissionReturnsFalseIfCarTypeIsInvalidTest läuft erfolgreich durch.		
Der Unittest HasPermissionReturnsFalseIfDateTimeNotInValidTimespanTest hat den korrekten Aufbau (arrange, act, assert).		
Der Unittest HasPermissionReturnsFalseIfDateTimeNotInValidTimespanTest läuft erfolgreich durch.		
Der Unittest HasPermissionReturnsTrueIfSuccessTest hat den korrekten Aufbau (arrange, act, assert).		
Der Unittest HasPermissionReturnsTrueIfSuccessTest läuft erfolgreich durch.		

Aufgabe 3 (jew. 1 Punkt, 27 in Summe)	Erf.	Nicht erf.
Die Seite /Customers/Index besitzt eine korrekte Dependency Injection der Datenbank.		
Die Seite /Customers/Index fragt die benötigten Informationen aus der Datenbank korrekt ab.		
Die Seite /Customers/Index zeigt den Vor- und Nachnamen aller Kunden an.		
Die Seite /Customers/Index zeigt die Anzahl der gekauften Vignetten aller Kunden an.		
Die Seite /Customers/Index zeigt die Anzahl der registrierten Fahrzeuge aller Kunden an.		
Die Seite /Customers/Index verlinkt korrekt auf die Seite /Stickers/Add/(customerGuid).		
Die Seite /Customers/Index verlinkt korrekt auf die Seite /Stickers/Index/(customerGuid).		
Die Seite /Customers/Index zeigt den Link Gekaufte Vignetten anzeigen nur an, wenn auch gekaufte Vignetten existieren.		
Die Seite /Stickers/Index besitzt eine korrekte Dependency Injection der Datenbank.		
Die Seite /Stickers/Index besitzt einen Routingparameter für die Kunden GUID.		
Die Seite /Stickers/Index fragt die benötigten Informationen aus der Datenbank korrekt ab.		
Die Seite /Stickers/Index verweist auf die Seite /Customer/Index, falls der Kunde nicht existiert.		
Die Seite /Stickers/Index ermittelt den Wert für Gültig bis korrekt.		
Die Seite /Stickers/Index zeigt Vor- und Nachnamen und E-Mail des Kunden an.		
Die Seite /Stickers/Index zeigt die Felder Kennzeichen, Vignettentyp, Kaufdatum, gültig von, gültig bis und Preis an.		
Die Seite /Stickers/Add besitzt eine korrekte Dependency Injection der Datenbank.		
Die Seite /Stickers/Add besitzt einen Routingparameter für die Kunden GUID.		
Die Seite /Stickers/Add fragt die benötigten Informationen für die Liste der registrierten Fahrzeuge korrekt ab.		
Die Seite /Stickers/Add fragt die benötigten Informationen für die Liste Vignettenarten korrekt ab.		

Die Seite /Stickers/Add stellt die registrierten Fahrzeuge des Kunden als Dropdownfeld dar.		
Die Seite /Stickers/Add stellt die Vignettenarten als Dropdownfeld dar.		
Die Seite /Stickers/Add prüft, ob das Gültigkeitsdatum in der Zukunft liegt.		
Die Seite /Stickers/Add gibt Validierungsfehler für das Gültigkeitsdatum korrekt aus.		
Die Seite /Stickers/Add prüft, ob der Fahrzeugtyp der Vignette dem des ausgewählten Fahrzeuges entspricht.		
Die Seite /Stickers/Add gibt Validierungsfehler für den Vignettentyp korrekt aus.		
Die Seite /Stickers/Add speichert den Datensatz korrekt in der Datenbank.		
Die Seite /Stickers/Add verweist nach der Speicherung auf die Seite /Stickers/Index/(customerGuid)		