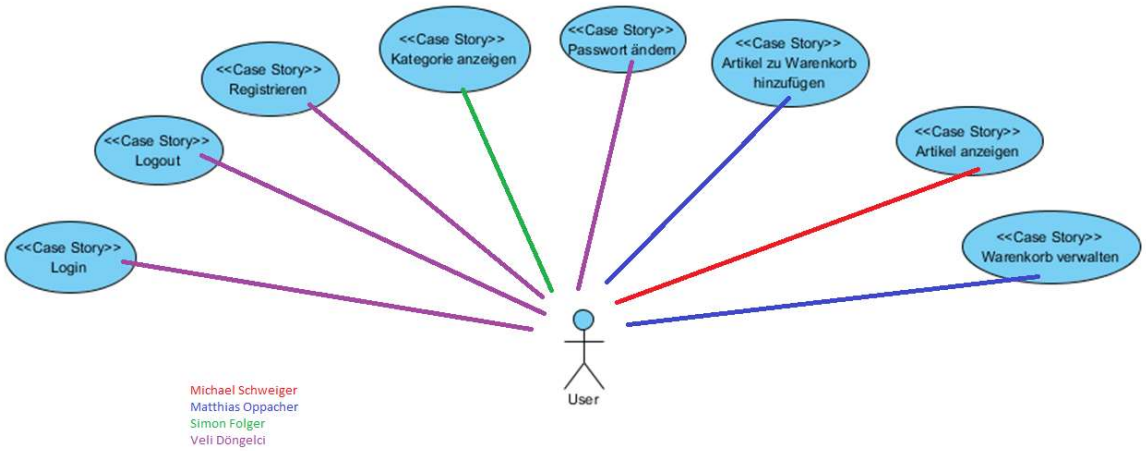
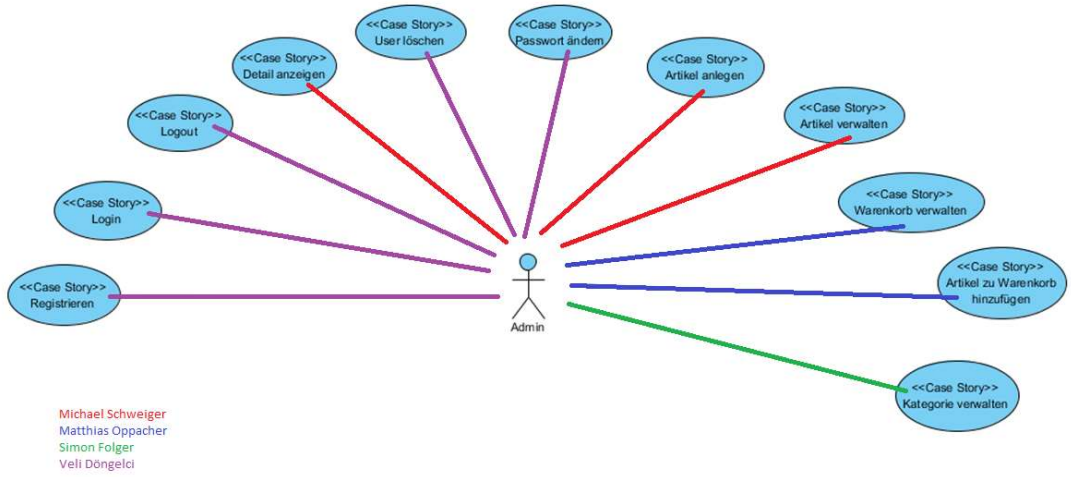
Web-Services 2016

Einzelbericht

Team: Artikelverwaltung
Name: Matthias Oppacher

1 Use Cases





Die Use Cases werden je nach Berechtigung, in einen Admin- und Userbereich aufgeteilt. Bei der Bearbeitung war ich für sämtliche Aktionen, die im Zusammenhang mit einem Warenkorb stehen, verantwortlich.

Dabei sollte es möglich sein, Warenkörbe zu verwalten, das heißt alle Warenkörbe des eingeloggten Benutzers anzuzeigen, neue Warenkörbe anzulegen, bestehende Warenkörbe umzubenennen oder auch zu löschen.

Außerdem war meine Aufgabe in der Detailansicht eines Artikels, die Möglichkeit zu schaffen, den angezeigten Artikel zu einem oder mehreren Warenkörben hinzuzufügen.

2 Schnittstelle

Bei der Erstellung der Service Schnittstelle, habe ich den *BaseAPIController* angelegt, der die Vaterklasse von allen Controllern des Service ist. Im *BaseAPIController* werden die verschiedenen Repositories (*UserRepository*, *CartRepository*, …) und die *ModelFactory*, die die Entity-Klassen auf ein entsprechendes Data-Transfer-Object und umgekehrt mappen, für die verschiedenen Service Schnittstellen zur Verfügung gestellt.

Hauptsächlich war ich für die Warenkorb-Schnittstelle verantwortlich. Hier sollte es, wie vorhin erwähnt, möglich sein, die Warenkörbe eines Benutzers anzuzeigen, neue Warenkörbe hinzuzufügen, bestehende Warenkörbe umzubenennen (bearbeiten) und zu löschen. Hierfür wurden folgende HTTP-Methoden im Controller für die Schnittstelle ausprogrammiert.

GET	/api/v1/users/{userId:int}/carts
POST	/api/v1/users/{userId:int}/carts
PUT	/api/v1/users/{userId:int}/carts/{cartId:int}
DELETE	/api/v1/users/{userId:int}/carts/{cartId:int}

Außerdem war meine Aufgabe, Artikel, die einem Warenkorb zugeordnet sind, anzuzeigen, Artikel in Warenkörben zu speichern und gespeicherte Artikel wieder aus dem Warenkorb zu entfernen. Dafür wurden die folgenden HTTP-Methoden im Controller implementiert:

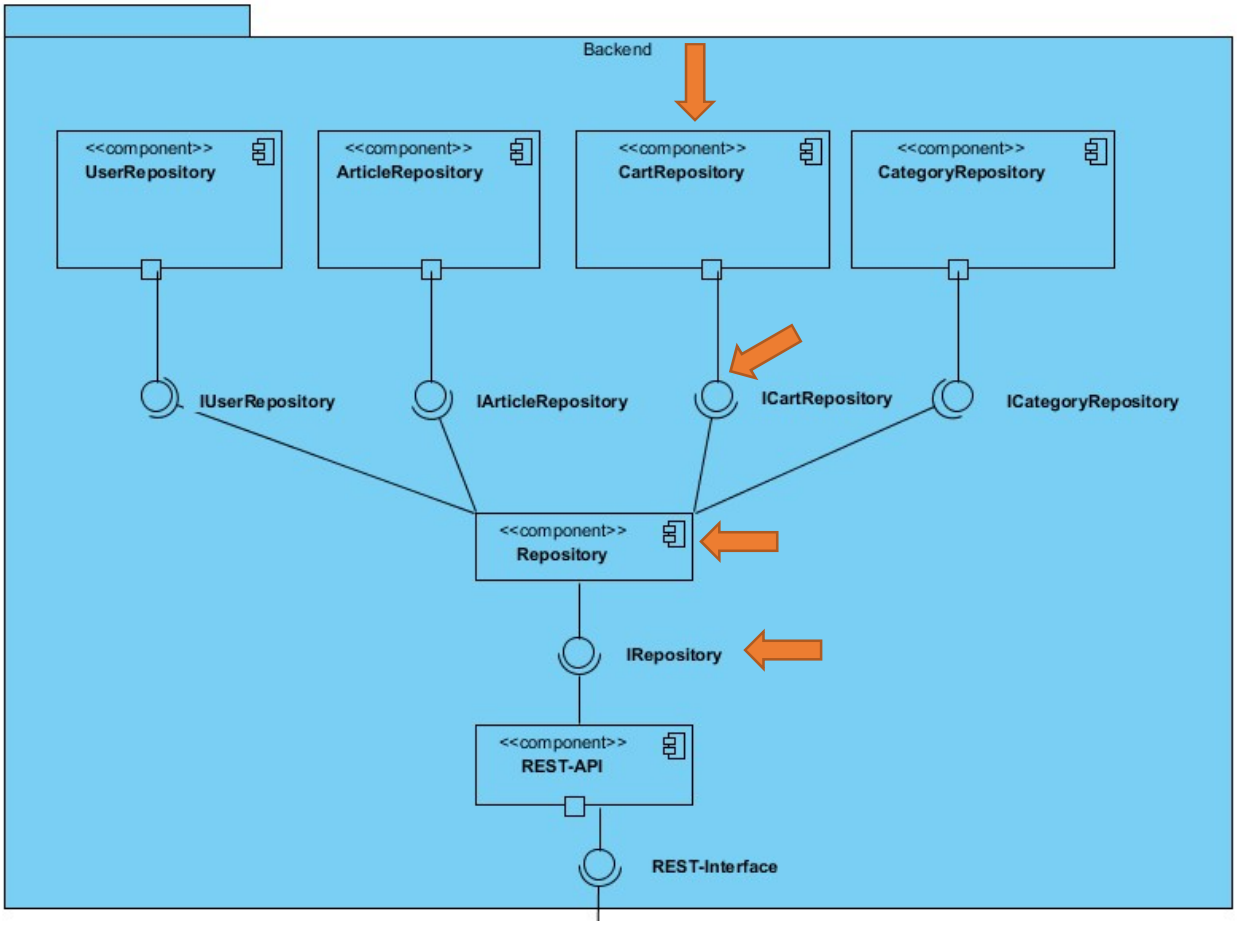
GET	/api/v1/users/{userId:int}/carts/{cartId:int}/articles
POST	/api/v1/users/{userId:int}/carts/{cartId:int}/articles
DELETE	/api/v1/users/{userId:int}/carts/{cartId:int}/articles/{articleId:int}

3 Technische Architektur – Blockschaubild

Da meine Aufgabe die Verwaltung der Warenkörbe war, habe ich das *CartRepository* implementiert, das den Datenzugriff auf die Datenbank steuert. Hier sind sämtliche Methoden, die für die Implementierung der Service Schnittstelle benötigt werden, implementiert. Als Framework für die Kommunikation zur Datenbank wurde das *Entity Framework* verwendet. Darauf gehe ich im nächsten Abschnitt näher ein.

Um auf das *Repository* selbst im Controller selbst zugreifen zu können, wird das Repository per Dependency-Injection in den Controller injiziert. Hierfür wird das .Net Package „Ninject“ verwendet, da die Implementierung und Konfiguration von Ninject relativ einfach war.

Im folgenden Diagramm sind die Bereiche markiert, die ich entworfen habe:



4 Technische Architektur – Datenhaltung

In unserem Projekt kam für die Datenhaltung der SQL-Server von Microsoft zum Einsatz. Als Datenbank-Framework haben wir uns für das Entity-Framework von Microsoft entschieden, um uns die Arbeit mit der Datenbank zu vereinfachen. Da wir von Anfang an wussten, wie die Datenbank aufgebaut werden muss, haben wir zuerst die Datenbank mit all ihren Tabellen mit dem SQL Server Management Studio angelegt, um im zweiten Schritt die Entitäten durch das Entity Framework generieren zu lassen (Model First Ansatz).

Da meine Aufgaben die Warenkorb Funktionen waren, waren für mich die *Cart* und die *ArticleCart* Tabellen wichtig. Die *Cart* Tabelle hat folgende Spalten:

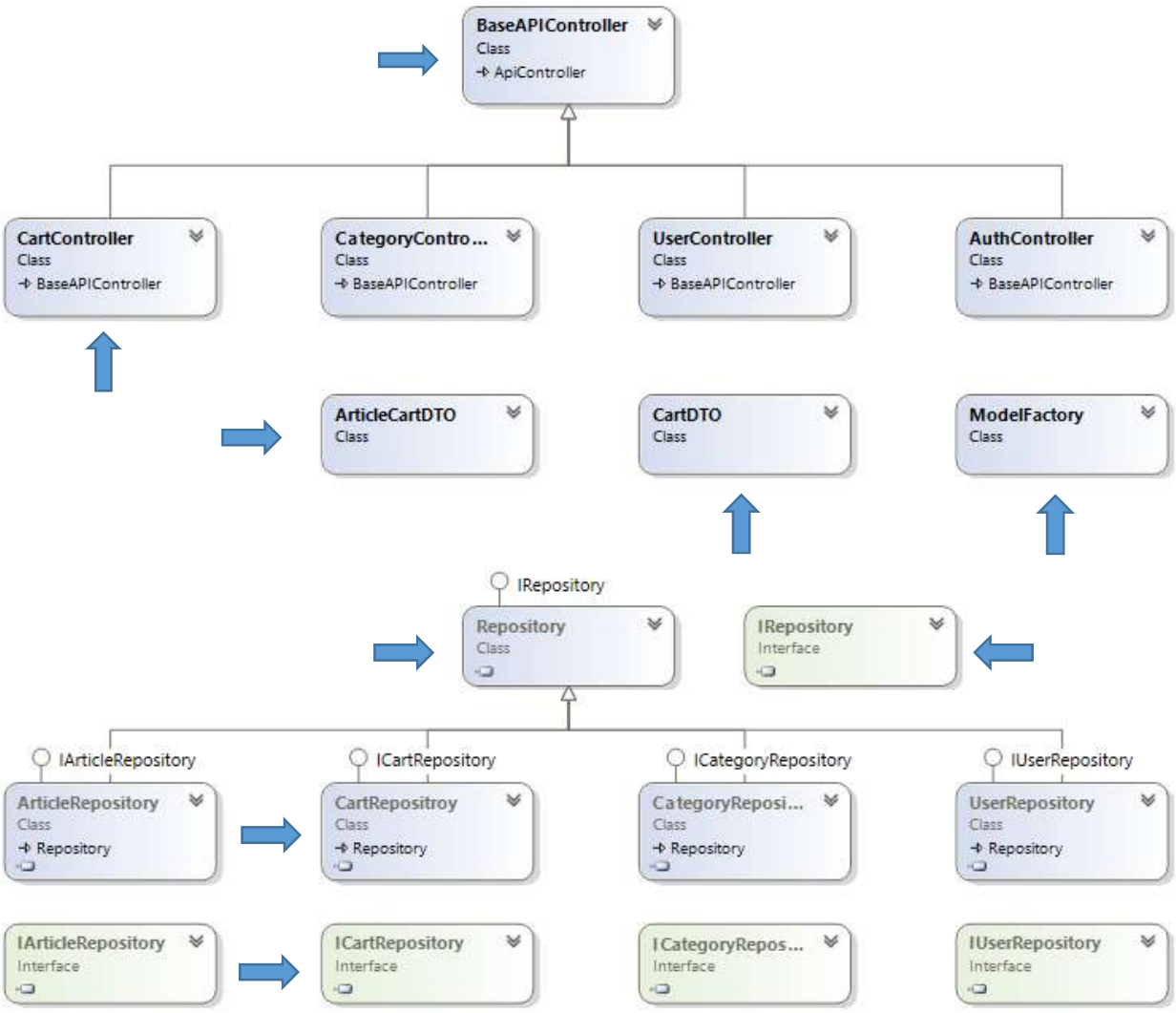
ID	int	Die ID des Warenkorbs
Name	Nvarchar(255)	Der Name des Warenkorbs
UserID	Int	Die ID des Users der den Warenkorb angelegt hat

Die *ArticleCart* Tabelle ist eine einfache Kreuztabelle, in der die m:n Beziehungen zwischen den Artikeln und Warenkörben gespeichert sind.

5 Technische Architektur – Klassendiagramm

Im folgenden Diagramm sind die Klassen gekennzeichnet, die ich implementiert habe.

BaseAPIController	Verschiedenen Repositories und ModelFactory für Kinder-Klassen
CartController	Service Schnittstelle für Warenkörbe
ArticleCartDTO	DTO für ArticleCart Entity
CartDTO	DTO für Cart Entity
ModelFactory	Mapping zwischen Entity Objekt ->DTO und umgekehrt
Repository	Bündelt die verschiedenen Repositories, um nicht alle Repositories einzeln in die Controller zu injezieren.
CartRepository	Stellt die verschiedenen CRUD Methoden für die Verwaltung von Warenkörben zur Verfügung



Um den Datenzugriff zu kapseln, haben wir uns für das Repository Entwurfsmuster entschieden, damit der Zugriff auf die Daten problemlos ausgetauscht werden kann, falls zum Beispiel das Entity Framework durch ein anderes Framework ausgetauscht werden soll.

Um auf das Repository in den verschiedenen Controllern zugreifen zu können, wird dieses per Dependency-Injection in den Controller injiziert.

6 Testkonzept

Unit Tests wurden aus Zeitgründen nicht programmiert. Getestet wurde nur während der Entwicklung im Browser.

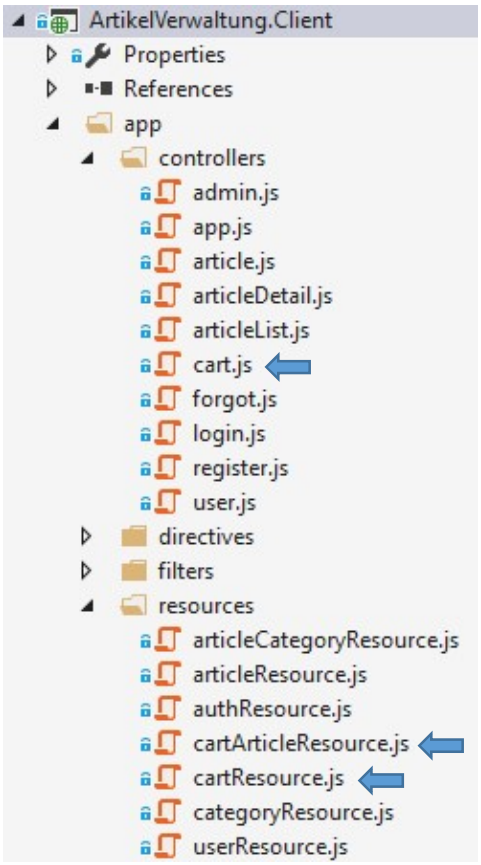
7 Implementierung der App

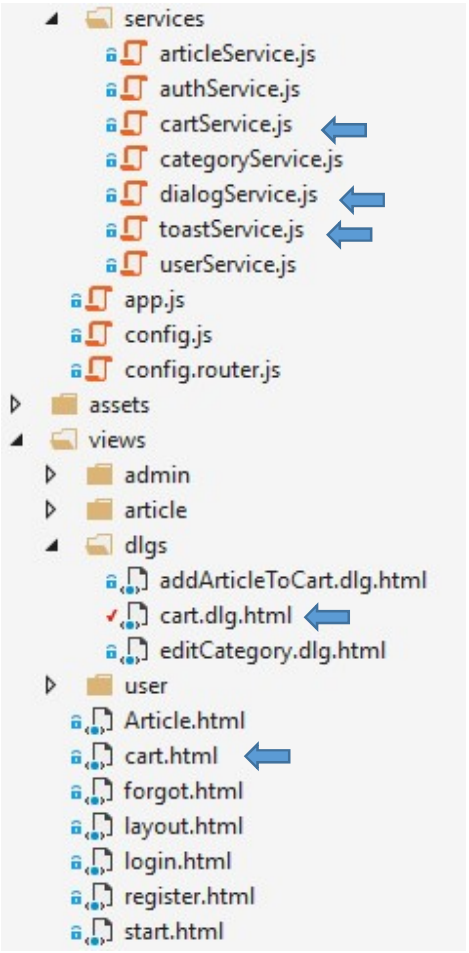
In der App selbst, habe ich das Grundgerüst, also die grobe Ordnerstruktur angelegt. Für die App haben wir uns für Angular 1 als Basis entschieden, da Angular mit seinen integrierten Funktionen wie dem Data-Binding oder dem Modul *Angular-Resource* uns einiges an Arbeit abgenommen hat. Für die Darstellung der App haben wir Angular-Material und Bootstrap verwendet.

Da ich für die Warenkorb Funktionen verantwortlich war, habe in dem Zusammenhang alle nötigen Views, Controller, Services und Angular-Resourcen implementiert.

Außerdem habe ich den Dialog und Toast-Service implementiert, die die jeweiligen Angular-Material-Services kapseln, um die Wiederverwendbarkeit in der App zu vereinfachen.

In den folgenden Bildern sind die Dateien markiert, die ich angelegt habe.




services
articleService.js
authService.js
cartService.js
categoryService.js
dialogService.js
toastService.js
userService.js
app.js
config.js
config.router.js
assets
views
admin
article
dlgs
addArticleToCart.dlg.html
cart.dlg.html
editCategory.dlg.html
user
Article.html
cart.html
forgot.html
layout.html
login.html
register.html
start.html