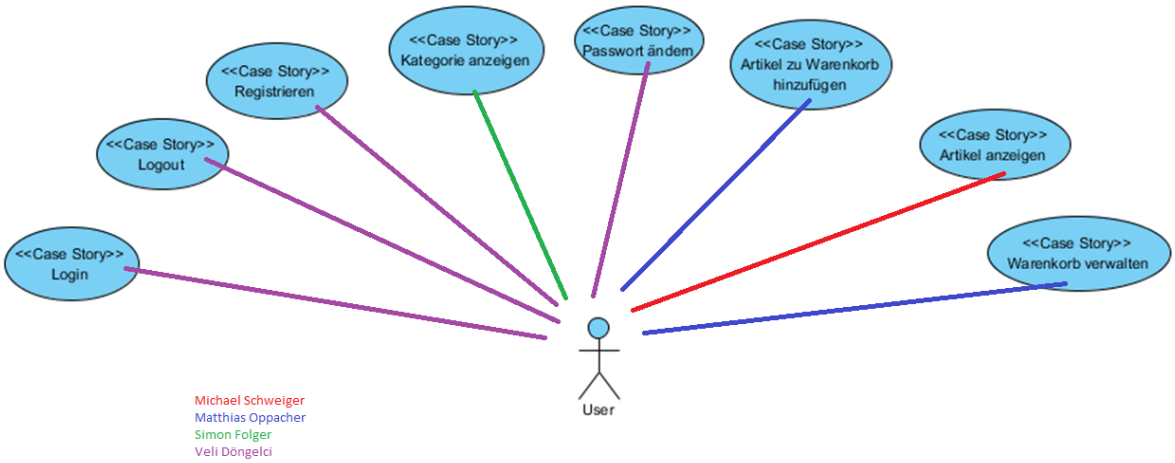
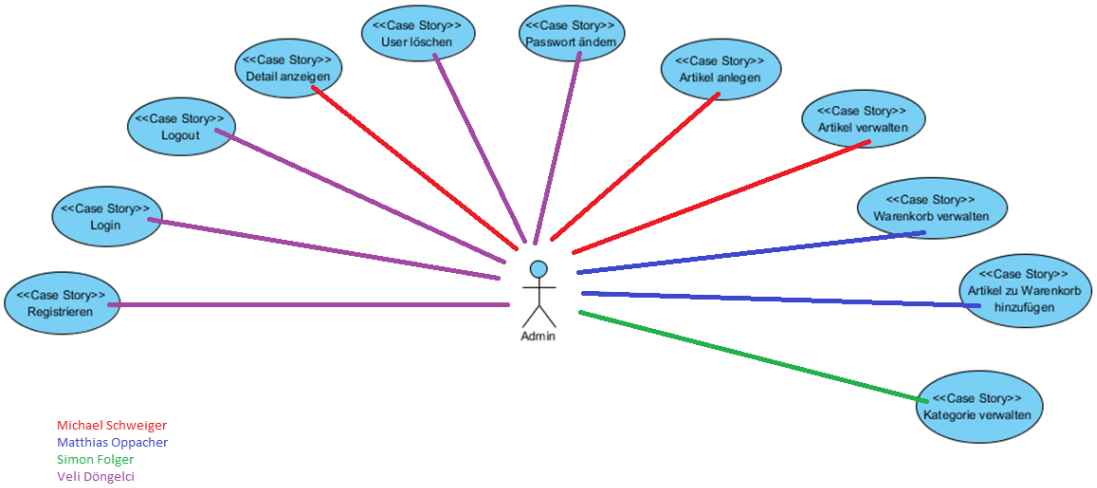
Web-Services 2016

Einzelbericht

Team: Artikelverwaltung
Name: Michael Schweiger

1 Use Cases





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

Bei der Useransicht musste zu jeder Kategorie eine Liste mit sämtlichen Artikeln dieser Kategorie angezeigt werden. Bei der Auswahl eines Artikels, aus dieser Liste, sollte eine Detailansicht des entsprechenden Artikels erscheinen.

Bei der Administratoransicht wird ein extra Bereich in der Navigation angezeigt, wo die Artikel bearbeitet werden können. So können zum einen, wie bei der Useransicht, beim Klicken auf eine Kategorie, die jeweiligen Artikel als Liste angezeigt werden. Hier ist es möglich, bei einer bestimmten Kategorie einen Artikel anzulegen. Es öffnet sich dann ein Eingabefeld, um die Attribute, wie Name, Beschreibung und Preis einzugeben, sowie den Artikel zu speichern. Weiter ist es möglich, bereits angelegte Artikel zu bearbeiten, oder diese zu löschen.

2 Schnittstelle

Im Folgenden werden die Schnittstellen behandelt, aufgeteilt in die Bereiche Backend und Frontend.

2.1 Backend

Der Artikelservice kann grob in zwei Bereiche unterteilt werden. Auf der einen steht das Frontend, welches mit Angular funktioniert. Im Backend kommt DotNet zum Einsatz. Die Schnittstellte zwischen Backend und Frontend für sämtliche Artikelfunktionen, wie z.B. Artikel anlegen, befindet sich auf Seite des Backends im CategoryController.

```
namespace ArtikelVerwaltung.API.Controllers
{
    public class CategoryController : BaseAPIController
    {
```

Dies hat den Grund, dass jeder Artikel einer Kategorie angehören muss. Zur Besserung Vorstellung kann hier als Beispiel die CRUD-Operation betrachtet werden, mit der alle Artikel einer entsprechenden Kategorie angezeigt werden können:

```
[Route("~/api/v1/categories/{id:int}/articles")]
[HttpGet]
public IHttpActionResult getAllArticle(int id)
{
    List<ArticleDTO> articles = ModelFactory.Create(CategoryRepository.GetArticle(id));

    return Ok(articles);
}
```

2.2 Frontend

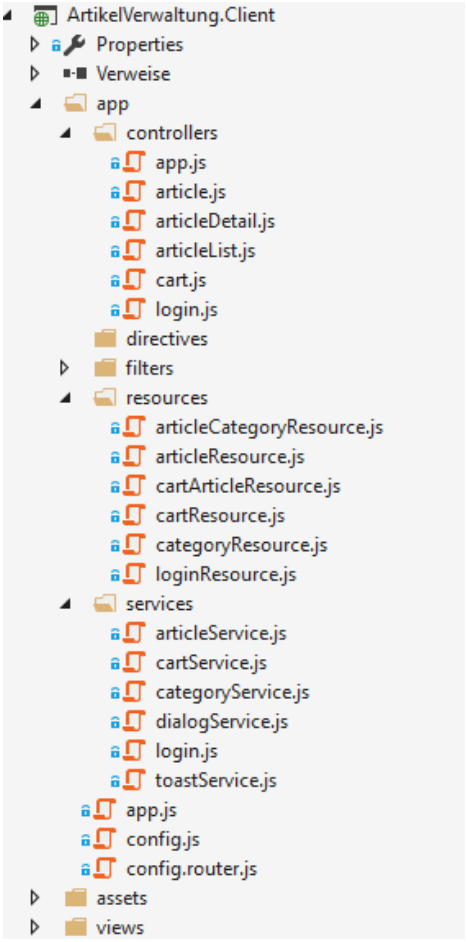
Im Frontend befindet sich ebenfalls eine Schnittstelle, mit der Daten bei der Eingabe an das Backend übertragen werden. Diese bestehen in Angular als Resource.

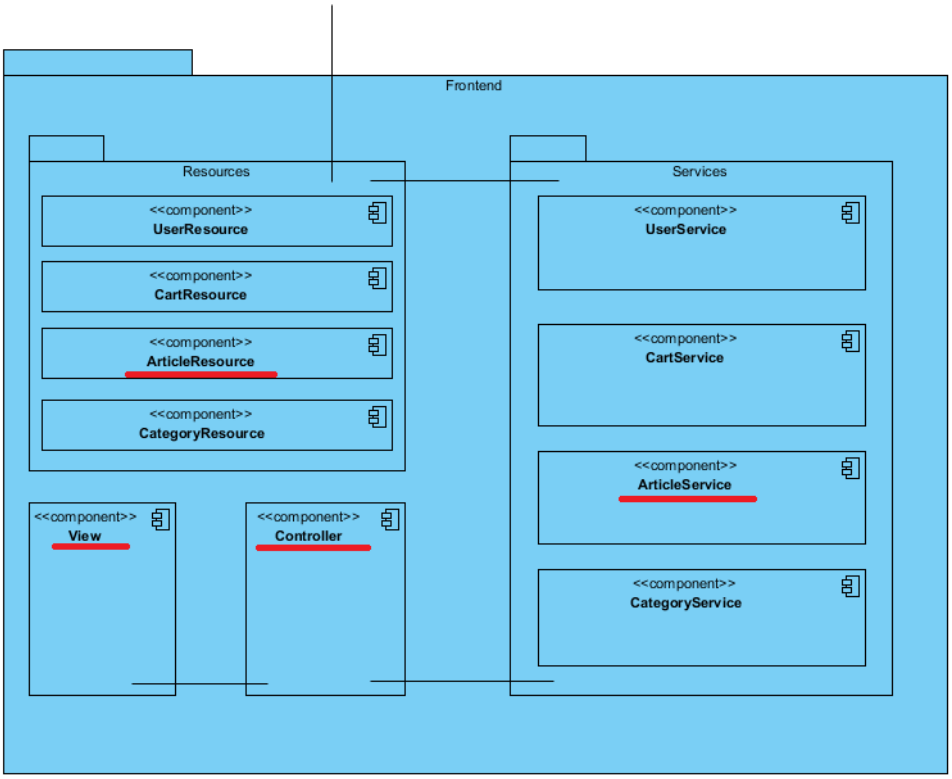
```
1  app.factory('ArticleCategoryResource', ['urls', '$resource', function (urls, $resource) {
2      return $resource(urls.BASE_API + 'categories/:catId/articles/:id', { catId: '@catId', id: '@id' }, {
3          query: { method: 'GET', isArray: true, params: {} },
4          save: { method: 'POST', params: { catId: '@catId' } },
5          update: { method: 'PUT', params: { catId: '@catId', id: '@id' } },
6          delete: { method: 'DELETE', params: { catId: '@catId', id: '@id' } }
7      });
8  }]);
```

Hier werden die entsprechenden CRUD-Operationen definiert und angegeben, welche Parameter, bei einem entsprechenden CRUD-Befehl, an das Backend mitgesendet werden. Weiter ist hier die Route des jeweiligen Backend-Controllers angegeben, an welchen die Operation gesendet werden muss.

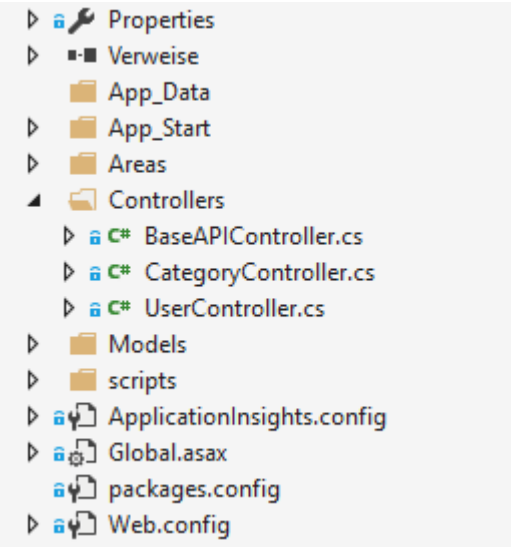
3 Technische Architektur – Blockschaubild

Zum einen musste Clientseite die Angular Struktur entworfen werden. Diese besteht in Betracht der Artikel, aus einer articleList-, sowie articleDetailController. Weiter wurde ein articleService integriert, der sämtliche clientseitige Funktion eines Artikels implementiert. In der oben genannten articleCategoryResource werden dann mithilfe der Controller die Daten an das Backend übertragen.

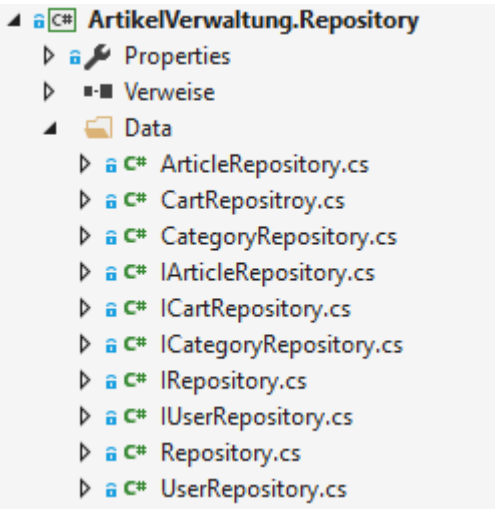


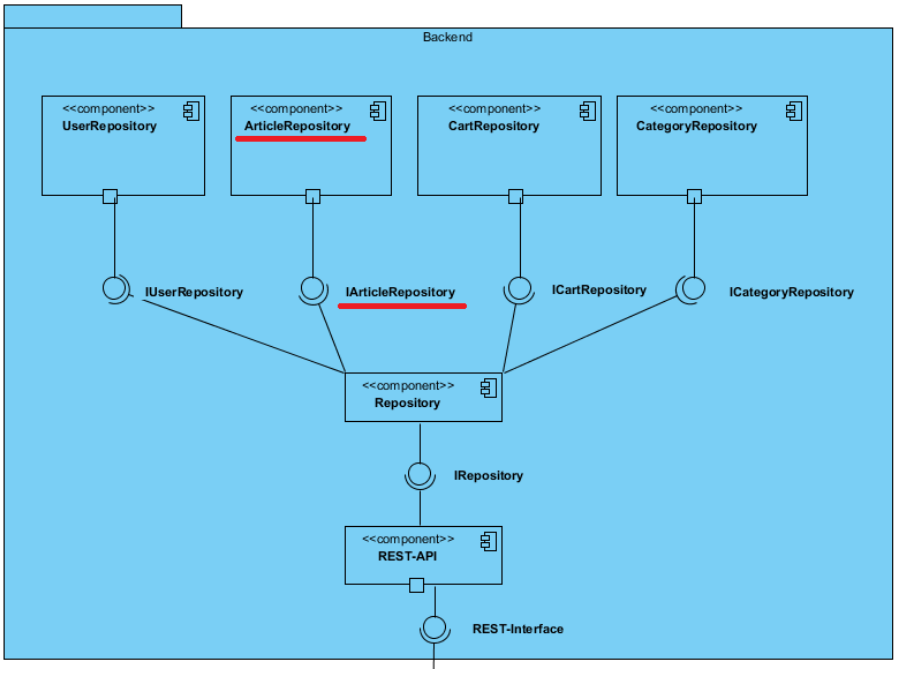


Als nächsten Schritt wurde der CategoryController hinzugefügt, welcher die Daten aus dem Frontend verarbeitet und Daten in die Datenbank schreibt, sowie liest.



Der letzte Schritt war dann die Programmierung des Repositorys, welches auf dem Repository-Pattern basiert. Hier gibt es eine IArticleRepository als Interface für die ArticleRepository Klasse. Hier finden sich Funktionen wie add, delete, usw. wieder um in die Datenbank zu schreiben, bzw. von ihr zu lesen. An dieser Stelle kam das Entity Framework zum Einsatz. Dieses erleichtert den Datenbankzugriff erheblich.



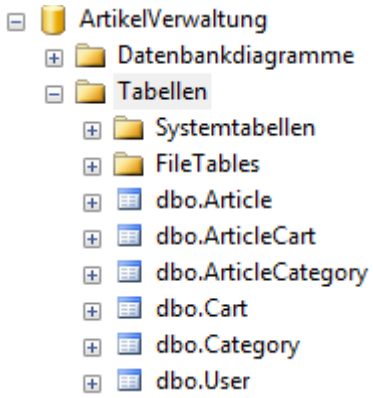


4 Technische Architektur – Datenhaltung

Als Datenbank wurde MSSQL verwendet. Vor dem Beginn der eigentlichen Programmierarbeit, haben wir hier zuerst eine Datenbank angelegt. Dabei wurde von mir die Tabelle Article angelegt, welche die Spalten ID, Name, Description, sowie Preis enthält.

Weiter musste eine Tabelle ArticleCategory angelegt werden, mit den Spalten ID, ArticleID, CategoryID. Diese Tabelle war dringend notwendig um festzulegen, welcher eindeutige Article zu

welcher eindeutigen Category gehört.



Bei der Erstellung der Datenbank musste ich mich daher ausführlich mit Herrn Oppacher (Warenkorb) und Herrn Folger (Kategorie) absprechen, wie wir die Tabellen ArticleCategory und ArticleCart gestalten, da diese als Schnittstelle unserer Arbeiten dienen.

5 Technische Architektur – Klassendiagramm

Bei der Gestaltung der Klasse CategoryController arbeite ich mit Herrn Folger zusammen, da jeder Artikel von einer Kategorie abhängig war. In dieser Klassen befinden sich alle CRUD Methoden, welche im Zusammenhang mit einer Kategorie, sowie eines Artikels stehen. Zur Veranschaulichung kann die updateArticle Methode betrachtet werden, um einen bestimmten Article, einer bestimmten Category zu bearbeiten.

```
[Route("~/api/v1/categories/{cId:int}/articles/{aId:int}")]
[HttpPut]
public IHttpActionResult updateArticle(int cId, int aId, [FromBody] ArticleDTO articleDTO)
{
```

6 Implementierung der App

Die Implementierung der App kann grob in die Bereiche Backend, Frontend, sowie Repository eingeteilt werden.

6.1 Backend

Im Backend wurde von mir und Herrn Folger die Klasse CategoryController implementiert.

Diese enthält sämtliche Methoden der CRUD-Operationen. Als Beispiel ist hier die Delete Methode eines bestimmten Artikels, aus einer bestimmen Category, zu sehen. Diese ruft dann mit die entsprechende Delete-Methode aus der ArticleRepository Klasse auf.

```
[Route("~/api/v1/categories/{cId:int}/articles/{aId:int}")]
[HttpDelete]
public IHttpActionResult deleteArticle(int cId, int aId)
{
     try
     {
         Category category = CategoryRepository.GetCategoryById(cId);

         if (category == null)
             throw new ArgumentException("Kategorie exisitert nicht!");

         Article article = ArticleRepository.GetArticleByID(aId);

         if (article == null)
             throw new ArgumentException("Artikel existiert nicht!");

         ArticleRepository.Delete(article);

         if (ArticleRepository.SaveAll())
         {
             return Ok(ModelFactory.Create(article));
         }
         else
         {
             return BadRequest();
         }
     }
     catch (ArgumentException)
     {
         return NotFound();
     }
 }
```

6.2 Frontend

Hier wurden im Service des Articles die Angular Methoden ausprogrammiert. Als Beispiel ist hier das Gegenstück zur Delete-Methode aus dem vorherigen Beispiel zu sehen,

```
var deleteArticle = function (article, handler) {
    var original = article;

    if ((article instanceof ArticleResource) == false) {
        var article = new ArticleResource();
        article.id = original.id;
        article.name = original.name;
    }

    Dialog.confirm({
        title: 'Artikel löschen',
        content: '"' + article.name + '" wirklich löschen?',
        ok: 'Löschen'
    }, function () {
        article.$delete(function () {
            articleInCategory.splice(articleInCategory.indexOf(original), 1);
            articleInCart.splice(articleInCart.indexOf(original), 1);
            handler(articleInCategory);
        });
    });
};
```

6.3 Repository

Im Repository musste das Interface IArticleRepository, sowie die Klasse ArticleRepository ausprogrammiert werden. ArticleRepository enthält sämtliche Funktionen für einen Datenbankzugriff. Als Beispiel wird hier wieder die Delete-Methode gezeigt, um einen bestimmten Artikel, der einer bestimmten Kategorie angehört, aus der Datenbank zu löschen.

```
public class ArticleRepository : Repository, IArticleRepository
{
    public ArticleRepository(ArtikelVerwaltungEntities ctx) : base(ctx)
    {
    }

    public Article Add(Article article)
    {
        return ctx.Article.Add(article);
    }

    public Article Delete(Article article)
    {
        ctx.ArticleCategory.RemoveRange(ctx.Article.Find(article.ID).ArticleCategory);
        return ctx.Article.Remove(article);
    }
```