Web-Services 2016

Einzelbericht

Team: Sparkyl Unicorn

Name: Manuel Huber

https://github.com/nelo112/UnicornMaker

**Glossar:**

*Optionen:*

Hiermit sind die verschiedenen Anpassungsmöglichkeiten für Einhörner gemeint. In der Version 1.0 gibt es die Optionen „Body“, „Hat“, „Shoes“ und „Wings“. Der Benutzer kann sich eine (oder keine) der zur Verfügung gestellten Auswahlmöglichkeiten pro Option auswählen. Zum Beispiel „Ein Rosa Körper, ein Roter Hot, Gummistiefel, keine Flügel“.

# Use Cases

*User – Unicorns erstellen*

Dies ist der Haupt Use-Case und eigentlich der komplette Kern der Anwendung.

Es gab die Überlegung den Vorgang in kleinere Use-Cases zu zerbrechen (z.B. „Einen Hut auswählen“), doch wäre dies eine sehr künstliche Trennung gewesen, da das Erstellen eines Einhorns ein zusammengehöriger Prozess ist. Somit hätten kleinere Use-Cases den bürokratischen Aufwand erhöht hätte ohne tatsächlichen Mehrwert zu liefern.

Dieser Use-Case beginnt mit dem Start-Bildschirm auf dem der Benutzer begrüßt wird. Er fährt dann fort zur eigentlichen Hauptseite des Use-Cases.

Dort kann er verschiedene Einhorn Körper, Flügel, Hüte und Schuhe aus einem seitlich angebrachten Menü auswählen. In diesem Menü sollen sowohl die Namen der einzelnen Auswahlmöglichkeiten, als auch eine kleine grafische Vorschau zu sehen sein. Neben dem Menü ist eine Fläche auf der das aktuell zusammengestellte Einhorn zu sehen ist. Alle Änderungen um Menu werden dort sofort angezeigt. Des Weiteren kann der Benutzer hier einen Namen für sein Unicorn eingeben. Mit einem Button wird das Einhorn gespeichert und der Benutzer wird zum nächsten Screen geführt (siehe Use-Case „Unicorn teilen“)

*User – Unicorn teilen*

Um ein Einhorn zu teilen, muss es zunächst erstellt werden (siehe Use-Case „Unicorns erstellen“). Am Ende des Erstellungsprozesses wird dem Benutzer automatisch ein Link generiert, den er an andere Benutzer schicken kann.

Andere Benutzer klicken auf den Link, sehen das erstellte Einhorn grafisch dargestellt und den Namen des Einhorns. Mit einem Button können sie selbst beginnen ein eigenes Einhorn zu erstellen.

# Schnittstelle

Die Schnittstelle die das Backend dem Frontend zur Verfügung stellt sind http Endpunkte nach dem REST Prinzip.

Im ersten Entwurf war die Schnittstell in einen öffentlichen und einen nichtöffentlichen Controller (= Endpunkt) unterteilt. Hier wurde aber sehr schnell ersichtlich, dass diese zu groß werden und, falls weitere Optionen außer Körper, Hüte, Schuhe und Flügel hinzukommen, sehr schlecht skaliert. Man hätte zu große Controller Klassen mit zu vielen fachlich unabhängigen Endpunkten. Der Code wäre unübersichtlich und schlecht wartbar gewesen.

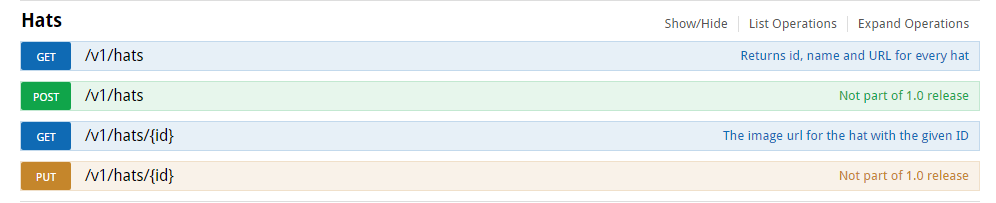
An diesem Zeitpunkt wurde auch aufgrund der Verkleinerung des Teams beschlossen, dass es für die Version 1.0 keine Admin Funktionalität geben wird.

Somit entschied ich mich dafür für jede Option (Körper, Hüte, Schuhe & Flügel) einen eigenen Controller zu erstellen. Diese enthalten die öffentlichen Endpunkte und bereits die Schnittstelle für nichtöffentliche (= Admin) Endpunkte, die jedoch nicht implementiert sind. Ob dies für v2.0 so bleibt ist ungewiss. Es besteht die Überlegung einen separaten Admin Controller zu erstellen, jedoch besteht auch hier die Gefahr, dass dies nicht gut skaliert, falls mehr Optionen hinzukommen. Das Ziel ist es möglichst kleine separate Schnittstellen und übersichtliche Controller Klassen zu erhalten.

Das Ausliefern der Bilder erfolgt aktuell über den Backend Server. Effizienter wäre hierfür ein dedizierter Host, der sich auf das zur Verfügung stellen von Bildern spezialisiert hat. Doch da die Version 1.0 des Unicorn Makers nur ein Minimal Viable Produkt ist um Interesse auszumessen und uns die Unabhängigkeit von Drittanbietern wichtig war haben wir uns dafür entschieden dies für Version 1.0 selber zu übernehmen. Die in der Schnittstelle als „image url“ beschriebenen URLs können direkt in die Browser Adressleiste oder als src (source) in einem HTML-Image-Tag verwendet werden. Der Server streamt das Bild als Byte Array mit „image“ als Media-Type im Header und alle Standard Browser können dies somit ohne Probleme anzeigen.

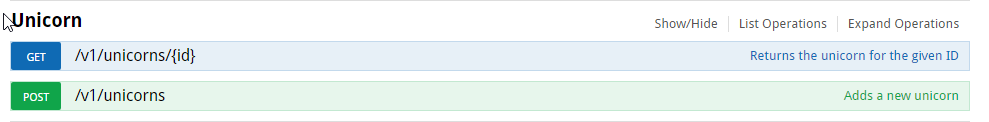
Die Schnittstelle der Optionen lieferte ursprünglich nur die ID und den Namen einer jeden Option zurück. Die Idee war, dass mit der öffentlich bekannten Schnittstelle und allen IDs das Frontend die Bild URLs selbst erstellen könne. Obwohl dies zwar wahr ist, haben wir uns dazu entschlossen dies zu ändern und die Bild URLs direkt zum Teil der Schnittstelle zu machen. Grund hierfür ist die Entlastung des Frontends. Das Erstellen der Bild URLs im Frontend benötig sowohl die Daten aus der Schnittstelle, als auch ein hart codiertes Wissen über den Zusammenbau der Bild URLs. Wenn sich also in Zukunft die URLs für Bilder ändern (z.B. durch den Wechsel zu einem Drittanbieter für Media Hosting) hätte man das Frontend ändern müssen. Außerdem liegt die Verantwortung über Verwalten & zur Verfügung stellen der Informationen / Bilder ganz klar beim Backend und das Frontend sollte möglichst ohne große eigene Arbeit alle benötigten Daten aus der entsprechenden Schnittstelle erhalten.

Hier ist die Schnittstelle einer Option:

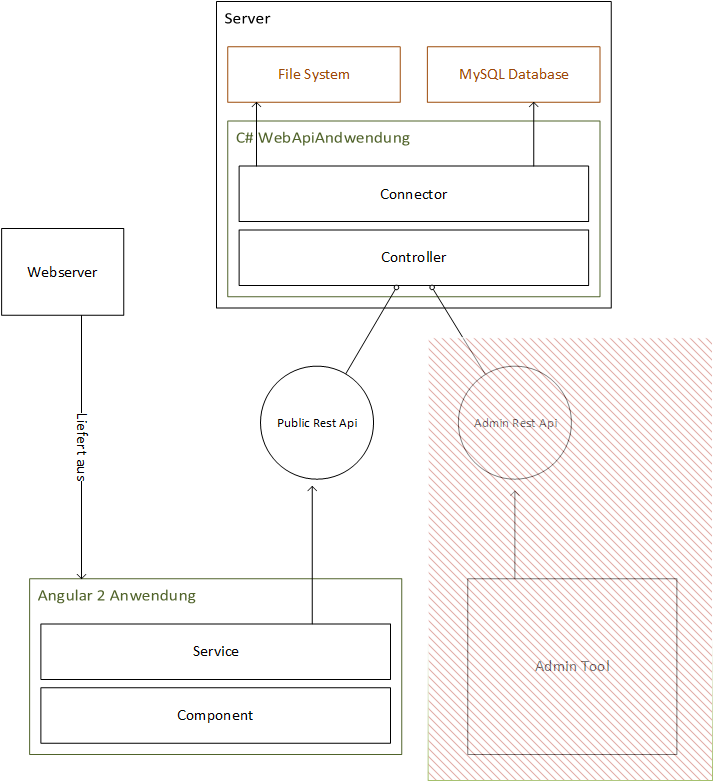


Die Schnittstellen für Körper, Schuhe und Flügel sehen äquivalent aus.

Die Unicorn Schnittstelle erlaubt das Anlegen von Einhörnern und das Auslesen von Einhörnern anhand seiner ID. Hierbei werden für alle Optionen jeweils eine ID (0 falls nichts gewählt wurde) angegeben.



# Technische Architektur – Blockschaubild



Die Architektur haben wir so entworfen, dass es einen Backend-Server gibt, der eine REST http Schnittstelle zur Verfügung stellt und eine Webanwendung mit Angular 2 die diese Schnittstelle nutzt. Die Idee dahinter war eine klare Trennung zwischen Back- & Frontend und die Möglichkeit weitere Frontends (z.B. Android- & iOS-App) leicht anzuschließen.

Der Backendserver ist eine C# .NET Anwendung mit der WebApi2. Zuerst war eine typische 3 Schichten Architektur geplant, doch da der Server quasi keine Business Logik enthält und dies nur ein MVP sein sollte haben wir uns dafür entschieden die Struktur nicht unnötig zu verkomplizieren. Somit haben wir nun eine Controller Schicht, die die http Endpunkte anbietet und eine Connector Schicht, die auf die Datenbankzugriffe kapselt.

Für den Zugriff benutzen wir das EntityFramework von Microsoft, da es unsere Anforderung schnell und unkompliziert lauffähige Software zu generieren erfüllt.

Als Datenbank benutzen wir in der Entwicklungsumgebung die lokale Datenbank von Visual Studio, da dies eine sehr einfache Variante ist bereits beim Entwickeln möglichst produktionsähnliche Bedingungen zu imitieren.

Als Framework zur Dependency Injection haben wir uns auf Grund der Empfehlung von Herrn Wachs für Autofac entschieden.

Das Frontend ist eine Angular 2 Anwendung. Geschrieben im TypeScript, HTML und LESS.

TypeScript ist ein typisiertes superset von JavaScript (TypeScript wird zu JavaScript kompiliert). Wir haben uns dafür entschieden, da Typsicherheit die Fehleranfälligkeit verringert und nach unserem Ermessen leichter wartbar ist. Außerdem habe ich bereits Erfahrungen in TypeScript gesammelt.

LESS ist ein superset von CSS (LESS wird zu CSS kompiliert), welches einige praktische Features wie z.B. die Definition von Variablen für Farben bringt.

Gebaut wird die Anwendung mit Webpack. Webpack durchläuft die Anwendung von einem gegebenen Startpunkt und erstellt ein Bundle File, dass den kompletten Code enthält.

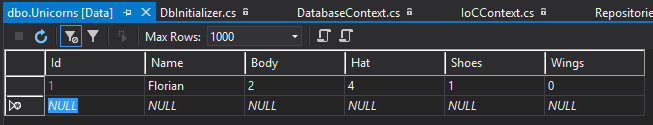
Auch wenn Angular 2 erst in der Beta ist, haben sich bisher TypeScript und Webpack als einer der Standards hervorgetan und somit gab es viele Ressourcen online, auf die wir zurückgreifen konnten.

# Technische Architektur – Datenhaltung

In unserer Anwendung gibt es 2 Systeme zur Datenhaltung. Eine Datenbank (aktuell die in Visual Studio integrierte LocalDB) und .NET Ressourcen.

Die Anwendung ist aktuell (für die Entwicklung) so konfiguriert, dass beim Start alle Datenbanktabellen gelöscht, neu erstellt und mit hart codierten Daten befüllt werden.

Optionen werden mit Namen & ID gespeichert. Die Einhörner werden mit Namen und lediglich der ID der Optionen gespeichert. Somit ist leider nicht sichergestellt, dass bei Löschen bzw. Ändern der Optionen die Einhörner wie ursprünglich erstellt angezeigt werden können. Doch da weder Löschen noch Ändern von Optionen geplant ist (neue Hinzufügen im Gegenzug schon) haben wir uns für diese Variante entschieden.



Die Bilder wurden in Visual Studio als Ressource hinzugefügt und müssen nach einer festgelegten Konvention (OPTION\_ID.png , z.B. body\_2.png) benannt sein. Dies macht das manuelle hinzufügen von neuen Grafiken etwas mühselig, doch mit dem Release 2.0 und der Admin GUI wird das Verwalten von Optionen über eine eigene Schnittstelle passieren und die Datenhaltung wird entsprechend automatisiert.

# Implementierung der App

Von mir implementiert wurden:

In der C# WebApi Anwendung:

* App\_Start/\*
* Connectors/BodiesConnector.cs
* Connectors/ImageConnector.cs
* Controllers/BodiesController.cs
* Infrastructure/\*
* Models/\*

In der Angular 2 Anwendung:

* Services:
  + Option-service.ts
  + Unicorn-service.ts
* Components (bestehend aus einem .ts, einem .html und einem .less File)
  + App
  + Main
  + View-Only
  + Unicorn Station
  + Option-Menu
    - Option-Menu-Category
    - Option-Menu-Entry
  + Unicorn-Display
* Models:
  + Option.d.ts
  + Unicorn.d.ts