Dokumentation von REST-Schnittstellen Projektarbeit SS17

von Boris Bopp (53690)

Inhalt

Bedeutung

Sowohl öffentliche als auch interne Web-Schnittstellen benötigen umfangreiche Dokumentation, damit Entwickler sie einsetzen können. Somit steht und fällt die Usability, und damit die Wettbewerbsfähigkeit eines Webdienstes, mit der angebotenen Dokumentation.

Die Möglichkeiten diese Dokumentation zu erstellen und aktuell zu halten hängen stark von den verwendeten Technologien und Frameworks ab. Wobei sich die Werkzeuge rund um die „OpenAPI Specification“ (ehemals „Swagger Specification“ [1]) immer größerer Beliebtheit erfreuen. Lediglich „HATEOAS“[2], der höchste Reifegrad einer REST-API, kann derzeit noch nicht abbildet werden[3].

Dokumentation unter ASP.NET

ASP.NET Webapi Helppage[4]

Beim Anlegen einer neuen ASP.NET Webanwendung wird standardmäßig das NuGet-Paket „Microsoft.AspNet.WebApi.HelpPage“[5] installiert. Damit wird direkt eine Auflistung der Endpunkte und deren Operationen während des Servicebetriebs zur Verfügung gestellt. Um jedoch eine ausführliche Beschreibung zu erhalten, muss neben der Generierung der XML-Dokumentation in den Projekteigenschaften, die Methode `SetDocumentationProvider` in der *HelpPageConfig.cs* aufgerufen werden. Damit werden zusätzlich die Dokumentations-Kommentare in der Hilfeseite dargestellt.

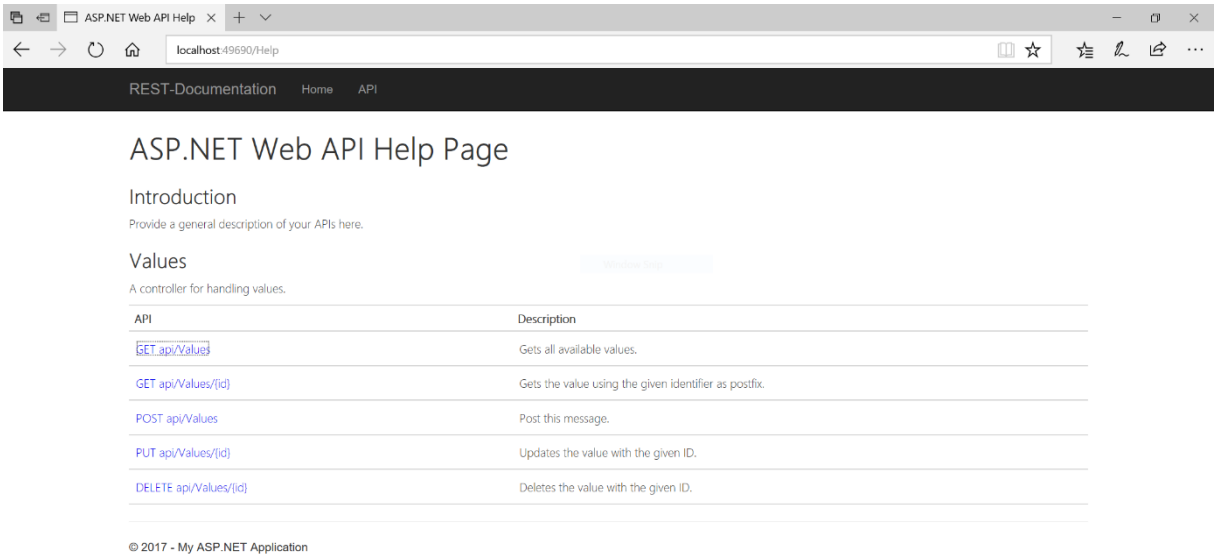

Abbildung 1 Die integrierte API-Dokumentation von ASP.NET Web API

[1] https://swagger.io/introducing-the-open-api-initiative/
[2] https://martinfowler.com/articles/richardsonMaturityModel.html
[3] https://jaxenter.de/restful-apis-dokumentieren-52052
[4] https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/creating-api-help-pages
[5] https://www.nuget.org/packages/Microsoft.AspNet.WebApi.HelpPage/

Anpassung der ASP.NET Webapi Helppage

Sobald Anpassungen an der Hilfeseite durchgeführt werden sollen, wird das Verwalten der Hilfeseite in den einzelnen Projektverzeichnissen oft umständlich. Gerade bei Projekten die in einzelne Micro Services aufgeteilt wurden soll die Dokumentation untereinander Konsistent sein.

Dazu werden drei Pakete installiert, um sowohl die zur Laufzeit kompilierten Views[6,7] als auch den statischen Inhalt[8] in ein gemeinsam genutztes Projekt auszulagern.

Anschließend wurde noch die Darstellung leicht modifiziert und ein einfacher Testclient[9] integriert.

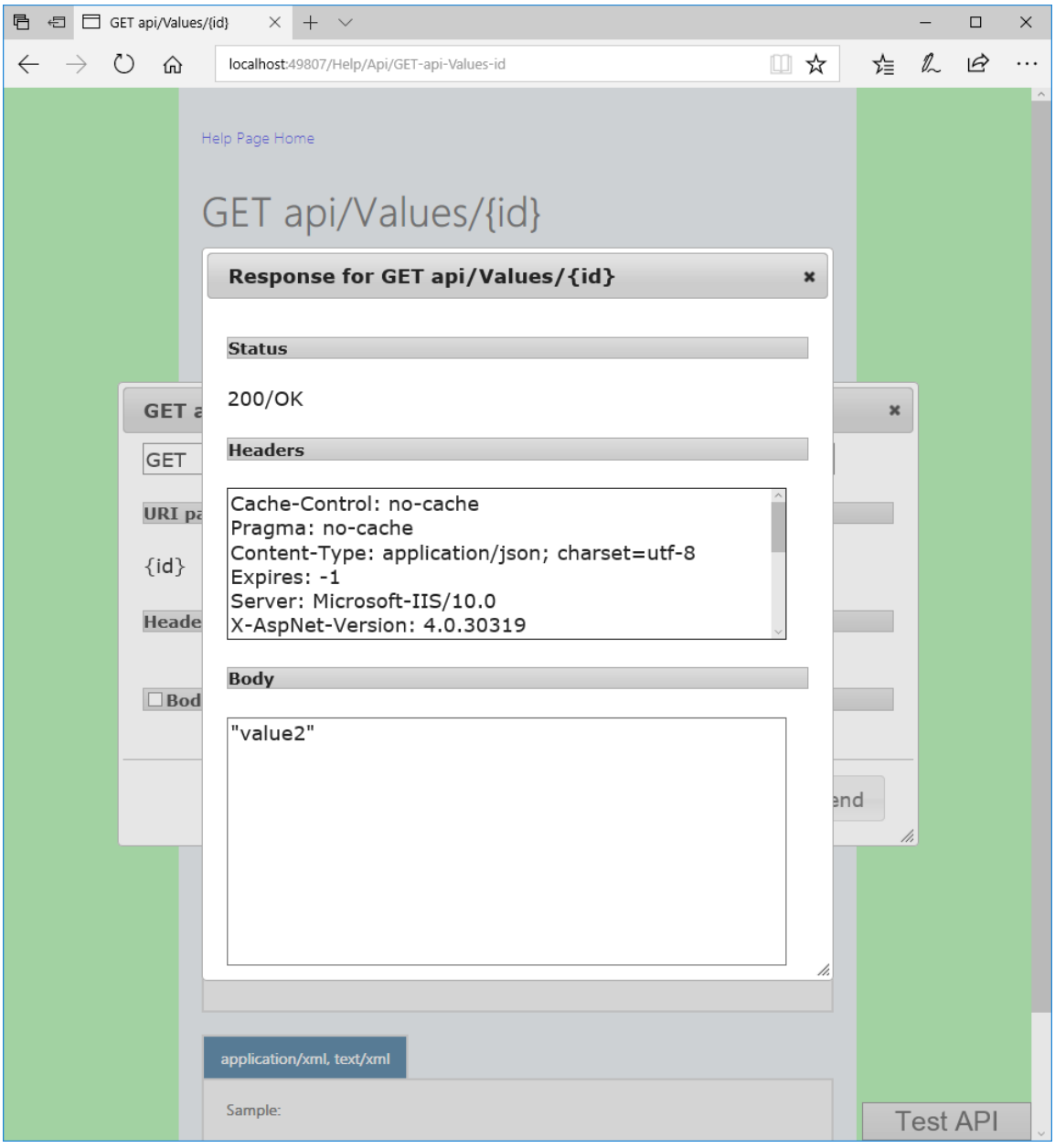

Abbildung 2 Anpassung der ASP.NET Webapi Dokumentation

[6] https://github.com/RazorGenerator/RazorGenerator/wiki/Using-RazorGenerator.MsBuild
[7] https://github.com/RazorGenerator/RazorGenerator/wiki/Using-RazorGenerator.Mvc
[8] https://github.com/mcintyre321/EmbeddedResourceVirtualPathProvider
[9] https://github.com/wuchang/WebApiTestClient

ASP.NET Swashbuckle

Das NuGet Paket „Swashbuckle“[10] kombiniert die Generierung der Swagger-Spezifikation aus den Metadaten des Webservice mit einer integrierten „Swagger UI“. Wie bei der Webapi Hilfe muss auch hier manuell die Generierung der Dokumentation im XML-Format aktiviert werden. Die Methode zum Einbinden dieser Datei heißt unter Swashbuckle `IncludeXmlComments` und befindet sich in der Datei *SwaggerConfig.cs*. Swashbuckle lässt sich problemlos parallel zur Webapi Helppage installieren und verwenden, wie im Projekt „DualDocumentation“ gezeigt. Swashbuckle zeigt seine Stärken neben dem Design auch in der Angabe der möglichen Antwortnachrichten und Http-Status-Codes.

Eine Alternative Swagger-Implementierung für ASP.NET Webapi ist „NSwag“[11].

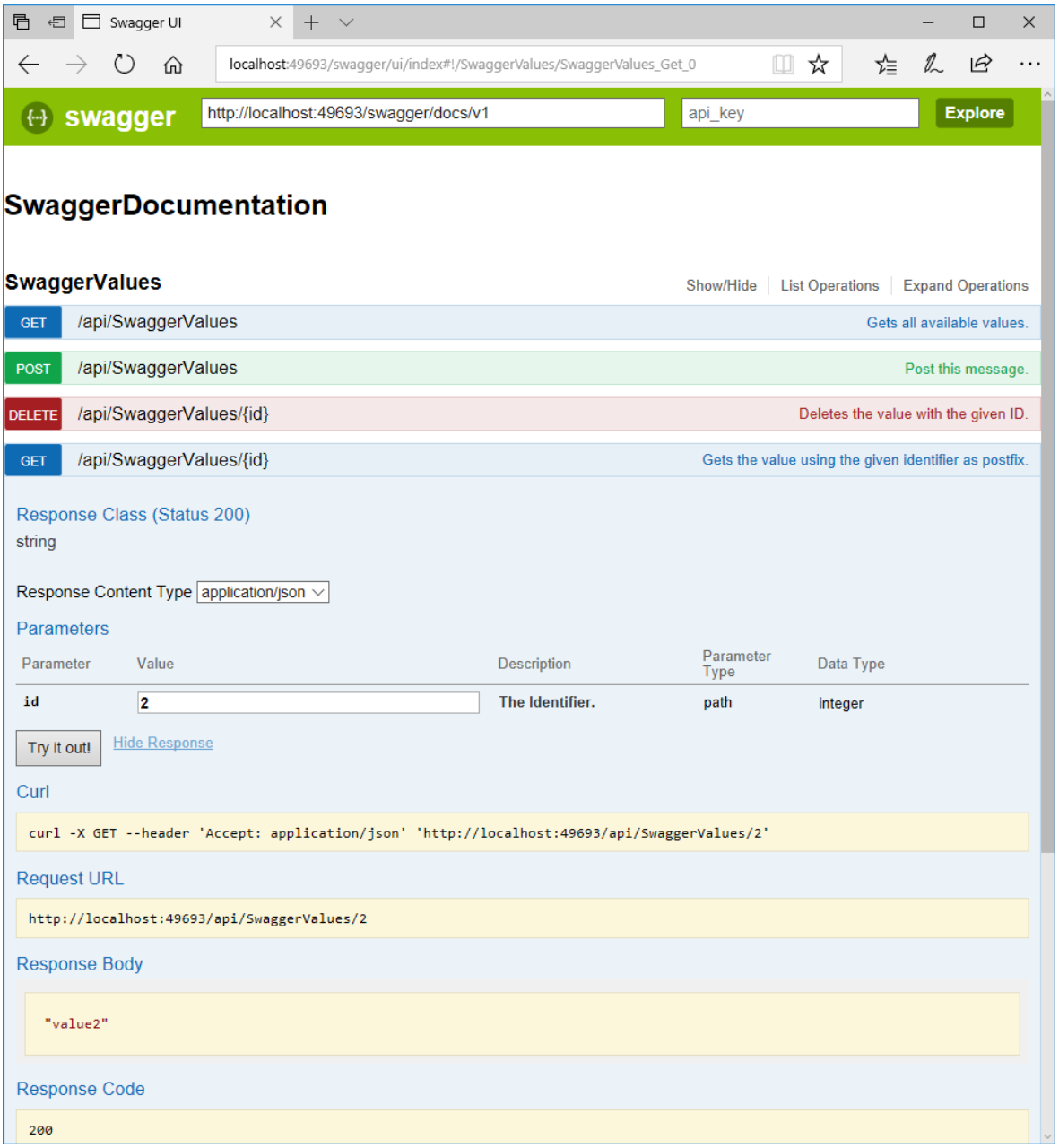

Abbildung 3 Swagger-UI unter ASP.NET Webapi bereitgestellt durch Swashbuckle

[10] https://github.com/domaindrivendev/Swashbuckle
[11] https://github.com/RSuter/NSwag

Dokumentation unter ASP.NET Core mit Swashbuckle

Mit ASP.NET Core wurde die Entwicklung der eigenen Hilfeseite eingestellt. Als Ersatz wird Swagger durch Swashbuckle empfohlen[12]. Da sich die plattformübergreifende Modernisierung des .NET-Frameworks bedeutend von den Vorgängerversionen unterscheidet wurde eine neue Edition von Swashbuckle dafür entwickelt[13]. Die gesamte Konfiguration findet dabei nun in der Datei *Startup.cs* statt.

Contract-First mit SwaggerHub

Die vorherige Lösung generiert aus dem vorhandenen Quelltext inklusive dessen Kommentaren eine menschenlesbare Dokumentation und eine Swagger-Spezifikation im JSON-Format. Diese Vorgehensweise ist auch als „Code-First“ bekannt.

Seit Version 2 wird jedoch versucht die Abhängigkeit zwischen Implementierung und Spezifikation zu trennen. Dazu soll vor der Implementierung die Spezifikation im „YAML“-Format erstellt werden. Eine komfortable Möglichkeit dazu bietet der im Onlineeditor „SwaggerHub“[14].

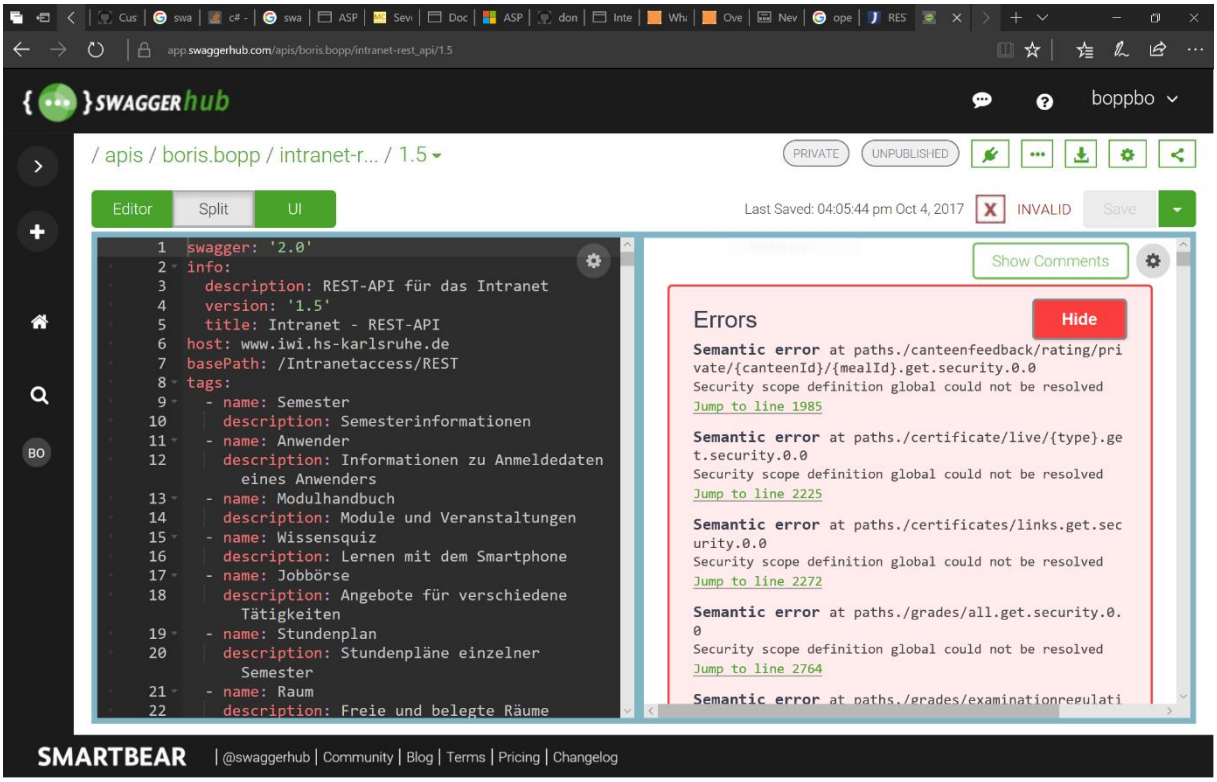

Abbildung 4 Mit dem SwaggerHub-Onlineeditor lassen sich APIs erstellen und validieren

Die automatische Service-Codegenerierung aus der Spezifikation funktioniert derzeit unter ASP.NET noch nicht[15]. Der Code-First Ansatz kann jedoch weiterhin verwendet werden.

[12] https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger
[13] https://github.com/domaindrivendev/Swashbuckle.AspNetCore
[14] https://swaggerhub.com/
[15] https://github.com/OAI/OpenAPI-Specification/blob/master/IMPLEMENTATIONS.md

Offene Probleme

Nicht behandelt wurde die Integration von Authentifizierung in die Testclients, da die Einrichtung der verschiedenen Authentifizierungsmöglichkeiten den Projektumfang übersteigt. Swashbuckle bietet jedoch Möglichkeiten eine Authentifizierung vor dem API-Aufruf durchzuführen.

Fazit

Es ist möglich eine eigene API-Dokumentation auf Basis der ASP.NET Webapi Hilfeseite zu implementieren. Die Zukunft gehört jedoch ASP.NET Core und dort setzt Microsoft auf die etablierte Swagger-Spezifikation. Diese bietet nicht nur den Vorteil der integrierten Swagger-UI, sondern auch die Möglichkeit aus der Spezifikation in verschiedenen Programmiersprachen Clients zu generieren.

Abschließend schneidet die bewährte Swagger-Lösung auf Grund des geringeren Aufwands und der einfachen Wartbarkeit meistens besser ab als eine Eigenentwicklung.