www.IT-Visions.de

Dr. Holger Schwichtenberg

Aus .NET Data Annotations mach Angular UI

Democode unter
https://github.com/drjofu/WebAPI-Dynamic-Angular-Forms

Dr. Joachim Fuchs

j.fuchs@it-visions.de

Ausgangssituation

- Gewünscht: Single Page Application für administrative Zwecke
  - Tabellarische Ansichten, CRUD-Operationen
- Server
  - ASP.NET Core 2 Web API
  - Datenklassen mit Attributen für die Validierung
  - Datenzugriffe z. B. über Entity Framework
  - Controller für Zugriff auf die Datenklassen
- Client
  - Angular x (hier Angular 5)

- Gesucht:
  - Möglichst einfache Entwicklung des UI

Schritte für jede einzelne Datenklasse

- Server
  - Erstellen der Datenklasse mit ihren Eigenschaften und Attributen
  - Einrichten der Datenzugriffe, DbContext
  - Erstellen einer Controller-Klasse mit den HTTP-Zugriffsmethoden
- Client
  - Kapselung der HTTP-Zugriffe auf der Clientseite
  - Eventuell Erstellen einer Proxiklasse für die serverseitige Datenklasse
  - UI für tabellarische Darstellung der Daten
  - Editor-Form für einen Datensatz
    - Validierungen
    - DropDown o. ä. für 1:n-Beziehungen (+ Daten hierfür)
  - Bedienlogik der Oberfläche (Schaltflächen für Edit, Save, Cancel, Delete etc.)

Inspiration

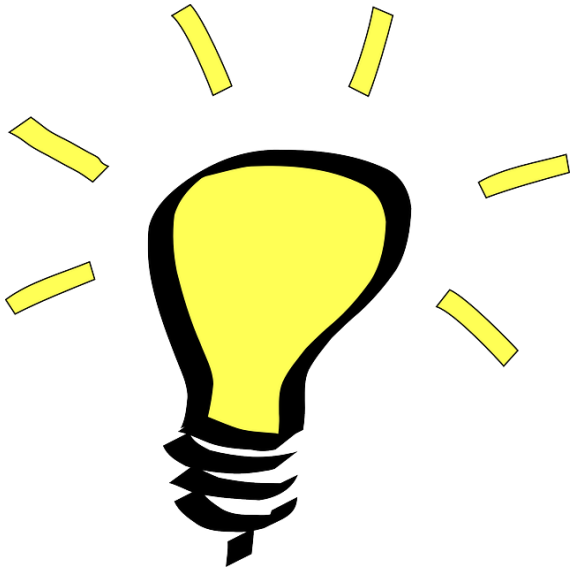
- ASP.NET Dynamic Data (vor 10 Jahren)
  https://msdn.microsoft.com/en-us/library/cc488546(v=vs.100).aspx

- Angular Reactive Forms (Model Driven Forms)
  https://angular.io/guide/reactive-forms

- Angular Dynamic Forms
  https://angular.io/guide/dynamic-form

Idee

- Clientseite automatisch erstellen
  - keine Code-Generierung via Swagger o. ä. sondern
  - dynamischer Aufbau des UI basierend auf serverseitigen Metadaten
  - Beispielanwendung mit
    - Liste der verfügbaren Datentypen
    - Tabellarische Anzeige der Daten eines ausgewählten Typs
    - Editor für ausgewählte oder neue Entität



Herausforderungen

- Bereitstellen aller erforderlichen Metadaten auf der Serverseite
  - Liste der verfügbaren Datentypen
  - Aufbau der Datentypen (Eigenschaftsnamen, -typen)
  - Die zusätzlichen Metadaten, die in der Oberfläche relevant sind
    - Anzeigetexte, Reihenfolge
    - Validierungsinformationen
    - Look-Up-Definitionen für 1:n-Beziehungen
- Umsetzen des UI in Angular
  - Laden und interpretieren der Metadaten
  - Tabellenanzeige mit Texten aus Metadaten und aus Look-Up-Definitionen
  - Editoranzeige mit Berücksichtigung der Validierungsinformationen

Datenmodell

- Model-Klassen
  - Artikel, Artikelkategorien, …
  - WarenhausContext, DatenInit
- Controller (mit Visual Studio generiert)
  - ArtikelController, ArtikelkategorienController, …
- Startup
  - DbContext via Dependency Injection

In der Demo unterstützte Attribute

- Validierung
  - Required, StringLenth, MinLength, MaxLength,
  - Range
- Visualisierung, Editor
  - Display, DataType, ScaffoldColumn, ReadOnly
- Infrastruktur
  - JsconIgnore
- Eigene
  - LookupTable
  - (EvenNumber, DividableBy)

Metadaten Tabelle

- Klasse Tabledefinition
  - Name (Anzeigename im UI)
  - Url (Url des Controllers)
  - Typename (Typ der Entity-Klasse)
  - PrimaryKey (Primärschlüssel, in Demo eingeschränkt)
  - PropertyDescriptors (Metadaten für die anzuzeigenden Eigenschaften)

Metadaten Tabelleneigenschaften (1)

- AngularPropertyDescriptorBase
  - Feldname, Anzeigename
  - Reihenfolge
  - Control-Typ (TextBox, DropDown, …)
  - Validierungen
    - Required
    - Min-, MaxLength (Zeichenketten)
    - Minimum, Maximum (Zahlenwerte)
  - Visible
  - ReadOnly

Metadaten Tabelleneigenschaften (2)

- AngularPropertyDescriptorSelektor
  - LookupUrl (Url zum Abruf der Daten)
  - LookupIdField (Schlüsselfeld)
  - LookupValueField (Feld mit den anzuzeigenden Informationen)

Metadaten erzeugen

- Tabledefinition
  - via Reflection, DbContext
- AngularPropertyDescriptionGenerator
  - via Reflection
  - Auswerten der Attribute

- Endpunkt mit allen Metadaten:
  - http://localhost:33554/api/metadata/tabledefinitions

App

MainWindow

Tabelle 1 | Tabelle 2 | … | Tabelle n

DataView

C1	C2	C3	Neu
Text	123	456	Edit
			Edit
			Edit

DynamicForm

DynamicFormProperty

DynamicFormProperty

DynamicFormProperty

Speichern | Abbrechen

ApiService

Client - Rahmenprogramm

- AppComponent (nur Router-Outlet)
- Routing (AppRoutingModule)
  - tablename als Parameter
- MainWindow
  - Lesen der Metadaten, einmalig beim Start
  - Anzeige eines Buttons für jede Tabelle
  - DataViewComponent an Tabellenauswahl gebunden

Service für Zugriff auf Web-API

- ApiService
  - getTabledefinitions() (Laden aller Tabellen-Metadaten)
  - getTabledata(tabledefinition: Tabledefinition)
  - putEntityData(tabledefinition: Tabledefinition, data: object)
  - postEntityData(tabledefinition: Tabledefinition, data: object)
  - deleteEntity(tabledefinition:Tabledefinition, id:any)
  - resolveLookups(propertyDescriptors: PropertyDescriptorBase[])
- Hilfsklassen TableDefinition, PropertyDescriptorBase etc. analog zur Serverseite

Tabellenanzeige

- DataViewComponent
  - Input: ausgewählte Tabledefinition
  - Lädt Tabellendaten vom Controller
  - Anzeige in HTML-Tabelle
  - Schaltflächen für New, Edit, Delete
  - Hilfsvariablen zur Steuerung der Oberfläche (Sperren von Buttons und Navigation)

  - Anzeige / Freischaltung des Editor-Fensters bei Klick auf New bzw. Edit

Editor-UI

- DynamicFormComponent
  - zeigt Eingabefelder für jede darzustellende Eigenschaft an
  - verwendet Reactive (Model Driven) Forms
- PropertyDescriptorConverterService
  - generiert aus den Metadaten die Datenstruktur für das Form-Model
  - zu ergänzen:
    - Werte
    - Validierungen

Eingabefelder

- DynamicFormPropertyComponent
  - generiert das benötigte HTML-Control
    - <input>
      - Vorsicht mit dem Attribut "type"
      - siehe auch DateValueAccessor für DateTime-Werte
    - <select>
      - Laden und Einbinden der Lookup-Daten
  - Einblenden der spezifischen Fehlermeldungen

Demo

- Neue Datenklasse
- Zusätzliche Validierungen

Weitere Ideen

- Serverseitige Validierungen asynchron vom Client aufrufen
  - würde Implementierung auf dem Client überflüssig machen
  - dauert eventuell was länger
- Ein einziger Controller kann unter Umständen ausreichen
  - Datenzugriffe über Reflection
  - Man spart sich das Anlegen der Controller-Klassen mit VS
  - Nachteil: keine individuelle Anpassung möglich

Fazit

Mit etwas Reflection auf der Serverseite lassen sich alle benötigten Metadaten generieren

Das UI kann vollständig generisch erzeugt werden

Somit muss das UI nur einmalig beschrieben werden und kann dann für beliebig viele Datentabellen /-klassen verwendet werden

Vielen Dank für Ihre Aufmerksamkeit!

Democode unter
https://github.com/drjofu/WebAPI-Dynamic-Angular-Forms