

---

# FISORO: BROWSERSEITIGE SUCHE

Dr. Viktor Sirotin,

März 2015-August 2016.

## WAS IST FISORO UND WIE KANN ICH FISORO NUTZEN?

FiSoro ist eine freie und quellenoffene Java-Script Bibliothek, die intern AngularJS 1 nutzt.

FiSoro ermöglicht eine browserseitige Suche, praktisch ohne eine zusätzliche Belastung des Servers. FiSoro macht das mit Hilfe von vorbereiteten JSON-Dateien, die ein Site-Betreiber mit einem regelmäßigen Zeitabstand oder nach jeder relevanten Databestandesänderung erneut generieren muss.

In der Realität stellen die meistens Backend-Systeme eine oder mehrere Möglichkeiten zur Verfügung, um das zu machen. Zum Beispiel bieten Content Management Systems (CMS), die oft gar keine oder nur eine sehr schwer zugängliche Datenbank haben, eingebettete Skriptsprachen für die Generierung verschiedener Dateien an.

Wie FiSoro funktioniert, können Sie sich am Beispiel des günstigsten Shops der Welt auf der Adresse [www.sirotin.eu/FiSoro/NumberShop/NumberShop.html](http://www.sirotin.eu/FiSoro/NumberShop/NumberShop.html) oder <http://bit.ly/2c5MCt1> anschauen.

Die zwei nächsten Paragraphen erklären die Motivation für die Entwicklung von FiSoro und die Hauptidee. Wenn Sie sich nur für die praktischen Aspekte interessieren, können Sie diese Paragraphen überspringen.

## ARCHITEKTURMUSTER "STARKER SERVER/SCHWACHER CLIENT" UND SEINE PROBLEME

Die meisten, in den letzten Jahren entwickelten Web-Anwendungen bestehen aus einer sehr starken Server-Seite und einer ziemlich primitiven Client-Seite. Dabei ist die Geschäftslogik in der Regel fast vollständig auf dem Server implementiert. Man kann also von einem Architekturmuster "Starker Server/Schwacher Client" (engl. Strong Server/Weak Client - SS/WC) sprechen. Die Beliebtheit dieses Musters unter Softwarespezialisten hat unter anderem historische Gründe. Am Anfang unseres Jahrhunderts wurde die Angst vor der Anwendung von JavaScripts stark (und begründet!) verbreitet. Seitdem ist JavaScript deutlich sicherer und schneller geworden. Die großen Hersteller pushten aber mit ihren neuen Server-Technologien weiter, dann kamen Cloud-Lösungen und in den letzten Jahren die Micro-Services-Euphorie. Nichtsdestotrotz gibt es viele Anwendungsbereiche, bei denen die Server-Seite erheblich entlastet werden kann, ohne die Client-Seite zu kompliziert und zu langsam zu machen.

Ein solches Anwendungsgebiet ist die intelligente Suche in Katalogen. Im Konsum-Bereich könnte es eine Suche einer Ware in einem Shop, bei Finanzen könnte das die Auswahl eines Finanzproduktes, etwa eines Optionsscheins sein. In der Tat braucht fast jede Geschäftsanwendung eine solche Funktionalität.

Betrachten wir den Suchablauf etwas detaillierter am Beispiel eines Online-Shops für Kleidung. Der Shop bietet Tausende von Artikeln, diese unterscheiden sich nach Typ, Farbe, Hersteller, Material, Preis und anderen Aspekten. Ein guter Shop sollte nicht nur eine Suche nach Stichworten anbieten, sondern dem Benutzer eine

Möglichkeit geben, eine Kombination von Suchparametern (z.B. Farbe, Hersteller, Intervall für Größe und für Preis) zu setzen.

Es wäre natürlich unzumutbar, von dem Benutzer zu verlangen, alle Parameter zuerst zu setzen und dann den Suchvorgang zu starten. Oft könnte es nämlich sein, dass bei dem Hersteller X kein Hemd mit der Farbe Y existiert. Es macht also keinen Sinn, für diese Kombination trotzdem die Größe und den Preis festzulegen. Das bedeutet, wenn der Benutzer einen Suchparameter festlegt, sollte die in Frage kommende Auswahl von Katalog-Elementen sofort berechnet werden und wenigstens ihre Anzahl muss angezeigt werden.

Die Implementierung dieser Idee bei der SS/WC-Architektur führt oft zu einer riesigen Ressourcen-Verschwendung. Zum Beispiel sollte in dem Moment, in dem der Benutzer einen neuen Parameter festgelegt hat, der Browser an den Server eine Anfrage senden. Diese Anfrage wird in eine Datenbank-Anfrage konvertiert und in der Datenbank prozessiert. Das Ergebnis, z.B. die Anzahl von passenden Elementen im Katalog, wird an den Browser zurückgesendet. Ärgerlich für den Shop-Betreiber ist aber, dass leider nur ein sehr geringer Anteil an Suchanfragen-Ketten mit einem Kauf endet. Das liegt unter anderem daran, dass der Kunde nicht die Absicht hat, etwas zu kaufen, sondern sich die Produkte nur "mal anschauen" möchte.

Ein Prozessieren jeder Datenbankanfrage benötigt nicht zu unterschätzende Hardware-Ressourcen. Bedauerlicherweise kommen Anfragen zeitlich sehr stark frequentiert, oft in riesigen Mengen in einem kurzen Zeitenintervall. Bei den Online-Shops passiert das oft abends, wenn die Kunden Lust haben zu shoppen. Bei den Geschäftsanwendungen passiert das zu bestimmten Arbeitszeiten, wenn viele Mitarbeiter anfangen, ähnliche Aktivitäten durchzuführen. Cloud-Lösungen können die Hardware-Engpässe etwas mindern, aber nicht vollständig beseitigen.

Es stellt sich also die Frage: Kann die Such-Logik vom Server an den Client (Browser) verschoben werden? Die gute Nachricht lautet: Ja, das geht. Wie es geht, wird im nächsten Paragraphen erklärt.

## FORMALISIERUNG DES PROBLEMS

Versuchen wir jetzt, die oben erwähnten und viele andere Anwendungsbereiche auf einem abstrakten Niveau zu formalisieren.

Unabhängig von Ihrer Natur können die sich in einem Katalog befindenden Elemente als eine Menge von Objekten mit ähnlichen Metadaten (Eigenschaften-Struktur) betrachtet werden. Die Eigenschaften jedes Elements können als ein Eigenschaftsvektor

$V = (p_1, p_2, \dots, p_n)$  dargestellt werden.

Eigenschaften können Werte aus einer bestimmten Liste (z.B. Liste von Farben) oder aus einem bestimmten numerischen Intervall (z.B. Preise) sein. Sie können auch aus Zeichenketten (String) oder binären Werten bestehen.

Alle Eigenschaften-Werte von allen in Frage kommenden Objekttypen können als eine Matrix  $A(m, n)$  dargestellt werden, wobei  $m$  die Anzahl an Elementtypen ist und  $n$  die Länge ihres Eigenschaftsvektors.

Interessante Operationen, die der Benutzer mit den Matrizen/Vektoren machen kann, sind:

1. Filterung von Elementen:  $F: A_1(m_1, n) \rightarrow A_2(m_2, n)$ . Damit  $m_2 \leq m_1$  (Anzahl von Elementen wird normalerweise verkleinert).
2. Sortierung von Elementen:  $S: A_1(m, n) \rightarrow A_2(m, n)$ . Damit ändert sich Anzahl von Elementen nicht. Die neue Matrix  $A_2$  hat aber eine andere Reihenfolge von Vektoren als  $A_1$ .

Diese zwei Operationen der Filterung und der Sortierung habe ich in FiSoro implementiert.

## DIE IDEE

Betrachten wir ein realistisches Beispiel. Nehmen wir an, dass jedes Objekt aus einem Katalog 20 Eigenschaften hat. 15 von ihnen sind binäre Daten, 2 von ihnen sind Strings mit einer durchschnittlichen Länge von 40 Symbolen (Name, Kurzbeschreibung) und 3 von ihnen sind Zahlen. Reale Zahlen, etwa Preise oder Größen, benötigen im Schnitt 3-5 Symbole für ihre Repräsentation.

Insgesamt wird der Eigenschaftenvektor des Objektes  $15 \cdot 1 + 2 \cdot 40 + 3 \cdot 5 = 110$  Bytes umfassen. Inklusive zusätzlicher Symbole für das Trennen der Eigenschaften voneinander (z.B. in JSON) kommen wir auf etwa 150 Bytes pro Vektor (eine Zeile in der Matrix).

Ein realer Katalog oder eine Kategorie eines Kataloges können Dutzende, Hunderte, Tausende oder Zigtausende Objekte beschreiben. Die Größe der Matrix wird damit:

Bei 100 Objekten - 15 KB

Bei 1000 Objekten - 150 KB

Bei 10000 Objekten 1.5 MB.

Das sind also vernünftige Größen um zu versuchen, die Matrizen an den Browser am Anfang der Auswahl zu übertragen und dort (im Browser) eine Auswahl durchzuführen.

Und genau das ist die Hauptidee von FiSoro.

Um die Nutzungsregeln von FiSoro zu erklären, habe ich seit kurzem einen virtuellen Online-Shop geöffnet ;-). In diesem Shop werden die Zahlen von 0 bis 999 angeboten. Der Kunde kann die Zahlen nach verschiedenen Kriterien aussuchen und in seinen virtuellen Besitz übernehmen.

## DER GÜNSTIGSTE SHOP DER WELT

Auf der Abbildung unten sehen Sie die Hauptseite meines Zahlen-Shops.

Properties of a	Properties of b	Properties of c	Properties of x $x = a \cdot 100 + b \cdot 10 + c$	Interval for value and digits sum $y = a + b + c$	Substrings
<input checked="" type="checkbox"/> Even number <input type="checkbox"/> Uneven number <input checked="" type="checkbox"/> Prime number <input type="checkbox"/> Not prime number <input type="checkbox"/> Fibonacci number <input type="checkbox"/> Not Fibonacci number <input type="checkbox"/> Factorial number <input type="checkbox"/> Not factorial number	<input type="checkbox"/> Even number <input type="checkbox"/> Uneven number <input type="checkbox"/> Prime number <input checked="" type="checkbox"/> Not prime number <input type="checkbox"/> Fibonacci number <input type="checkbox"/> Not Fibonacci number <input checked="" type="checkbox"/> Factorial number <input type="checkbox"/> Not factorial number	<input checked="" type="checkbox"/> Even number <input checked="" type="checkbox"/> Uneven number <input checked="" type="checkbox"/> Prime number <input type="checkbox"/> Not prime number <input type="checkbox"/> Fibonacci number <input type="checkbox"/> Not Fibonacci number <input type="checkbox"/> Factorial number <input type="checkbox"/> Not factorial number	<input type="checkbox"/> Even number <input type="checkbox"/> Uneven number <input type="checkbox"/> Prime number <input type="checkbox"/> Not prime number <input type="checkbox"/> Fibonacci number <input type="checkbox"/> Not Fibonacci number <input type="checkbox"/> Factorial number <input type="checkbox"/> Not factorial number <input type="checkbox"/> Round <input checked="" type="checkbox"/> Horizontal	Min value 200 Max value 799 Min digit sum 5 Max digit sum 18	Substring in text representation (e.g. Five) Hundred Substring in description (e.g. 15)

3 numbers available. [Show Selection](#)

**222 (8) matches.**  
666 (7) matches.  
444 (6) matches.

[Click to be virtual owner of number 222!](#)  
Representation: Two Hundred Twenty Two  
Description: This is the number 222.

ABBILDUNG 1: HAUPTSEITE DES ZAHLEN-SHOPS.

Zugegeben, schön ist diese Sete nicht. Sie erlaubt es aber den Kunden, schnell und bequem ihre passenden Zahlen auszusuchen.

Jede angebotene Zahl besteht aus maximal drei Ziffern. Die Kunde kann die Eigenschaften jeder Ziffer separat setzen (erste drei Spalten). Die Zahl als Ganzes besitzt auch die Eigenschaften der einzelnen Ziffern und hat noch ein paar spezielle Zusatz-Eigenschaften. Zum Beispiel kann sie „round“ sein (durch 10 teilbar) oder „horizontal“ (alle Ziffern sind gleich).

Die vorletzte Spalte erlaubt es, ganz feine Wünschen des Kunden zu berücksichtigen – das Intervall für den Wert der gefilterten Zahlen und für die Summe ihrer drei Ziffern.

Die letzte Spalte ermöglicht es, weitere Filtermöglichkeiten für die textuelle Repräsentation der Zahl und für den Text ihrer Beschreibung zu setzen.

Am Anfang sind alle Checkboxes leer und die Liste enthält alle angebotenen Tausend Zahlen. Wenn der Gast der Seite eine Box in einer der vier linken Spalten zum ersten Mal anklickt, verringert sich die Liste der Zahlen. Die Auswahl einer zweiten, dritten usw. Checkbox in der gleichen Spalte führt in der Regel zur Vergrößerung der Auswahl. Das passiert, weil die Bedingungen innerhalb den Spalten von 1 bis 4 mit dem logischen Operator „und“ miteinander verknüpft sind. Dabei können bestimmte Kombinationen sinnlos sein. Im angezeigten Beispiel hat der Benutzer in der dritten Spalte gleichzeitig die Checkboxes für gerade und ungerade Ziffern eingecheckt. Das bedeutet, dass damit in dieser Spalte überhaupt alle Ziffern in Frage kommen. Das ist aber sinnlos, denn wenn beide Checkboxes nicht gecheckt wären, hätte der Kunde das gleiche Ergebnis.

Bei numerischen Eigenschaften lässt FiSoro ein Suchintervall setzen. In der in der Abbildung dargestellten Situation sucht der Kunde zum Beispiel die Zahlen mit dem Wert zwischen 200 und 799 und solche, bei denen die Summe der Ziffern zwischen 5 und 18 liegt.

Die Eingabefelder in der letzten Spalte ermöglichen es, die Auswahl durch simple Text-Filter einzuschränken.

Die gefilterten Zahlen sind im Moment nach der Anzahl der erfüllten Kriterien sortiert. Man kann sehen, dass die Zahl 222 acht Kriterien erfüllt, und die Zahl 444 nur sechs.

## WIE BEREITET MAN DATEN FÜR FISORO AUF?

In diesem Paragraph erkläre ich, welche Schritte Sie durchführen müssen, um FiSoro bei einem Shop, einem Content Management System (CMS) etc. anzuwenden.

### SCHRITT 1: VERTEILEN SIE IHRE PRODUKTE AUF GRUPPEN

Ihr Shop oder ein anderes System kann mehrere Produkte verwalten, für die auch die Suche implementiert sein muss. Aus der Erfahrung kann man sagen, dass diese Produkte sich in Gruppen bzw. Kategorien unterteilen lassen. Die Mitglieder einer Gruppe sollen vom Blickwinkel des Benutzers ähnlich sein. Technisch gesehen sollen die Mitglieder der Gruppe möglichst gleiche Eigenschaften haben. Ich empfehle, in einer Gruppe nicht mehr als ein paar Tausend Produkte zu verwalten. Bei größeren Anzahlen probieren Sie zuerst aus, ob die Performance ausreicht (insbesondere an mobilen Geräten).

Im nächsten Schritt werden Sie FiSoro für jede einzelne Gruppe separat einrichten. Technisch bedeutet das, dass Sie mit der Hilfe eines Scripts oder manuell zwei JSON-Dateien und eine HTML-Suchmaske vorbereiten müssen.

Die erste JSON-Datei (in unserem Fall - model.json) definiert das Metamodell für die Produktgruppe. Die zweite Datei (in unserem Fall – data.json) enthält die Eigenschaftsvektoren der Gruppenmitglieder.

In unserem Zahlenshop habe ich nur eine Gruppe von Produkten – die ganzen Zahlen zwischen 0 und 999.

### SCHRITT 2 : DEFINIEREN SIE DAS METAMODELL FÜR IHRE PRODUKTE

In diesem Schritt müssen Sie die für die Benutzersuche relevanten Eigenschaften definieren. Wenn es um die Produkte aus einem Shop geht, denken Sie daran, dass für die Suche relevante Eigenschaften nicht unbedingt den vom Hersteller benannten Eigenschaften gleich sind. Zum Beispiel können die Farben nicht nur „Rot“ oder „Blau“ sein, sondern auch „Hell“, „Warm“ oder „Hipster-Style“.

FiSoro unterstützt Eigenschaften folgender Art:

- a) logische Eigenschaften, die ein Produkt besitzt oder nicht besitzt, z.B. „hat rote Farbe“, „Ist Wasserdicht“ usw. Im Zahlenshop sind das die Eigenschaften von Ziffern und Zahlen etwa „Die Zahl ist gerade“ usw.
- b) numerische Eigenschaften, wie z.B. Größe oder Gewicht. In unserem Shop sind das die Werte der Zahlen und die Summe ihrer Ziffern.
- c) Text-Eigenschaften, wie z.B. eine Kurzbeschreibung, Hersteller-Name usw. In unserem Fall sind das die Text-Repräsentation der Zahl und ihre Kurzbeschreibung.

Aus der Erfahrung kann man sagen, dass in der Praxis eine gute Auswahl der logischen Eigenschaften der wichtigste Beitrag zum Erfolg der Suche ist.

### SCHRITT 3: GRUPPIEREN SIE VERWANDTE EIGENSCHAFTEN ZUSAMMEN

Oft ist es auch sinnvoll, die Eigenschaften des Produkts auf die Gruppen zu strukturieren. Zum Beispiel könnten das bei einer Audio-Anlage die Gruppen für die Klangqualität, Optik und Steuerung sein. In meinem Zahlenshop sind das Gruppen für einzelne Ziffern und die Zahl selber.

### AUFBAU DER EIGENSCHAFTENVEKTOREN

Unten ist die JSON-Zeile für ein Produkt aus meinem Shop dargestellt (die Zahl 221). Um die Struktur der Zeile besser zu erklären, sind alle Elemente auf vier typographische Zeilen unterteilt.

```
[ "221", "5",  
  "10101010", "10101010", "01011010", "0101010100",  
  "Two Hundred Twenty One", "This is the number 221.",  
  "www.magic-numbers.examples/221.html" ],
```

#### LISTING 1: DARSTELLUNG DER ZAHL 221 IN DATA.JSON

Die erste Zeile enthält numerische Eigenschaften der Zahl – ihren Wert und die Summe ihrer Ziffern.

Die zweite Zeile enthält vier Sub-Gruppen logischer Eigenschaften: drei für die Ziffern und eine vierte für die Zahl selbst. Zum Beispiel listet die dritte Gruppe **"01011010"** die Eigenschaften der Ziffern „1“ auf. Das erste Element (0) gibt an, dass die Zahl die Eigenschaft „ist eine gerade Zahl“ nicht besitzt. Welche Eigenschaft auf welcher Position im Eigenschaftsvektor abgebildet sein muss, definiert die Datei „model.js“, die unten erklärt wird.

Die dritte Zeile in der Liste oben enthält die Werte von zwei Text-Eigenschaften des Produkts – die textuelle Präsentation der Zahl und ihre Kurzbeschreibung.

Die letzte Zeile sollte der Link zur Präsentation dieses Produkts in einem Katalog sein. Normalerweise enthält diese Seite Fotos und eine ausführliche Beschreibung des Produktes durch den Hersteller, Bewertungen und Meinungen von Kunden usw. Leider fehlen im Moment diese Dinge bei meinem Shop;-)

### AUFBAU DER METAMODELL-DATEI

Das Metamodell FiSoro ist hierarchisch aufgebaut und definiert auf der obersten Ebene die Gruppen der Eigenschaften (sog. Filters-Gruppen).

In der Abbildung unten ist die Gruppe für die erste Ziffer (die Anzahl von Hundert in Zahlen) dargestellt. Diese Gruppe hat den Namen „Nx100“. Der Type „Bit Vector“ verweist darauf, dass die Gruppe einen Vektor mit Elementen aus 0 und 1 darstellt. Der Eintrag „columnInData“ zeigt an, an welcher Stelle in der JSON- Zeile diese Gruppe steht (angefangen mit 0, siehe das Zeilen-Beispiel im vorigen Paragraphen).

Danach folgen die Filter. Die Reihenfolge der Filter im Metamodell muss genau in jeder JSON-Zeile abgebildet werden. Der Pfeil in der Abbildung zeigt, dass die Eigenschaft „ist Primzahl“ auf die dritte Position abgebildet sein muss.

```
{ "filterGroups": [
  {
    "name": "Nx100",
    "type": "Bit Vector",
    "columnInData": "2",
    "filters": [
      { "name": "Even number" },
      { "name": "Uneven number" },
      { "name": "Prime number" },
      { "name": "Not prime number" },
      { "name": "Fibonacci number" },
      { "name": "Not Fibonacci number" },
      { "name": "Factorial number" },
      { "name": "Not factorial number" }
    ]
  },
  ...
],
"10101010", "10101010", "01011010", "0101010100",
```

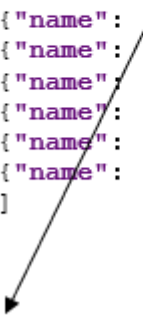


ABBILDUNG 2: MAPPING ZWISCHEN DATA-MODELL UND ZAILE-ELEMENTEN IN DATA-DARSTELLUNG

## AUFBAU DER HTML-SEITE

Wir wollen in diesem Artikel nicht auf alle Implementierungsdetails von FiSoro eingehen. Unser Ziel ist es zu zeigen, wie man FiSoro anwenden kann.

Bei der Bearbeitung der JSON Daten und der Anzeige der Daten auf der HTML-Seite hilft uns AngularJS 1.2.

Deswegen endet unsere Seite mit den Anweisungen für das Laden einer Angular-Bibliothek und einer FiSoro-Bibliothek:

```
...
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.min.js"></script>
<script src="js/fisoro.js"></script>
</body>
</html>
```

ABBILDUNG 3: ENDE DER HTML-SCRIPT DER HAUPTSEITE

Am Anfang der HTML-Datei wird ein div-element mit eingebettetem Controller definiert. Darunter fängt die Definition der Tabelle an. Das Codefragment unten zeigt die Zellen mit den Namen der Tabellen-Spalten.

```

<body >
<div data-ng-app="" data-ng-controller="FiSoroController">
  <table border="1" >
    <tr>
      <td>
        Properties of a
      </td>
      <td>
        Properties of b
      </td>

```

ABBILDUNG 4: DEFINITION DES CONTROLLER N HTML-SEITE

In der nächsten Tabellenzeile kommen die Zellen mit den AngularJS' Elementen, die es ermöglichen, entsprechende Filter an- und ausschalten zu lassen.

```

<tr>
  <td>
    <p data-ng-repeat="y1 in filterGroups.Nx100.filters">
      <input type="checkbox" data-ng-model="y1.value" id="{{y1.id}}"
        data-ng-true-value="true"
        data-ng-false-value="false" />{{y1.name}}</p>
    </td>
    <td>
      <p data-ng-repeat="y2 in filterGroups.Nx10.filters">
        <input type="checkbox" data-ng-model="y2.value" id="{{y2.id}}"
          data-ng-true-value="true"
          data-ng-false-value="false" />{{y2.name}}</p>
      </td>

```

...

```

<td>
  <div data-ng-repeat="y5 in filterGroups.Values.filters">
    <p>{{y5.name}}</p><p>
    <input type="text" size="4" data-ng-model="y5.value"
      id="{{y5.id}}"></p></div>
  </td>
  <td>
    <div data-ng-repeat="y6 in filterGroups.Texts.filters">
      <p>{{y6.name}}</p><p>
      <input type="text" size="10" data-ng-model="y6.value"
        id="{{y6.id}}"></p></div>
    </td>

```

ABBILDUNG 5: DEFINITION VON ELEMENTEN FÜR DARSTELLUNG IN TABELLE-SPALTEN

Vergleichen Sie doch einmal Abbildung 2 und Abbildung 5. Jede Gruppe von Eigenschaften aus dem Datamodel wird auf der HTML-Sete mit Hilfe eines AngularJS-Element vom Typ „data-ng-repeat“ abgebildet. Andererseits wird eine jede solche Gruppe in data.json mit einer „Spalte“ repräsentiert. Unter einer Spalte verstehen wir an dieser Stelle die Elemente an gleichen Positionen in verschiedenen Zeilen der Datei.

Weitere Details finden Sie, wenn Sie den Quellcode der HTML-Seite und die JSON-Dateien analysieren. Ich empfehle die Dateien aus dem Zahlung-Shop als Vorlage zu nehmen und Schritt für Schritt auf Ihre Rahmenbedingungen anzupassen.

## AUSBLICK

Die Erfahrung aus den ersten Anwendungen von FiSoro zeigt, dass der Funktionsumfang für viele Such-Aufgaben ausreicht. Außerdem habe ich gesehen, wie erstaunlich leicht und schnell diese Bibliothek eingesetzt und benutzt werden kann.

Viele Änderungen und Verbesserungen sind denkbar. Wenn Sie interessante Erweiterungsideen haben, schauen Sie in die existierenden Tickets mit geplanten Features in GitHub. Leider kann ich Ihnen nicht versprechen, dass diese oder andere Features schnell implementiert werden können. Wenn Sie aber selbst das eine oder andere Feature implementieren oder einen Bug fixen möchten – bitte machen Sie einen entsprechenden Pull Request in GitHub.