
FiSORO: BROWSER-SIDE SEARCH

Dr. Viktor Sirotin,

2015-2016.

WHAT IS FISORO AND ITS BENEFITS?

FiSoro is a free and open source Java script library. It uses internally AngularJS 1.2.

FiSoro allows a browser-side search of business data practically without load on the server.

FiSoro does it with the help of prepared JSON files. The site operator must regenerate these files after each relevant data changing.

In the real world, mostly backend systems provide one or more options to do that. For example, the content management system (CMS) often have no own database but have embedded scripting languages for generating the data files.

How FiSoro works, you can see by my special number store at <https://www.sirotin.eu/FiSoro/>.

Two next paragraphs explain the motivation for development of FiSoro and its main idea. If you are looking only to practical aspects, you can skip these paragraphs.

ARCHITECTURE PATTERN "POWER SERVER / WEAK CLIENT" AND ITS PROBLEMS

Most web applications developed in recent years consist of very power server side and rather primitive client side. The business logic is usually implemented almost entirely on the server. It corresponds an architectural pattern that I call here as "Power Server / Weak Client" - PS/WC). The popularity of this pattern among software specialists has historical reasons. At the beginning of our annual hundreds the fear about widespread use of JavaScript was strong (and justified!). Since that time JavaScript has become much safer and faster. However, the major manufacturers were pushing further with their new server technologies, after that come cloud solutions and last year micro-services euphoria. Nevertheless there are many application areas in which the server side can be greatly relieved without client side to make complicated and slow.

One of such application areas is the search in the catalogs. In consumer area, it could be a search of goods in a shop. At the finance, it can be the selection of a financial product as warrant. In fact, almost every business application needs such functionality.

Consider the search process detailed on the example of an online store for clothes. The shop offers thousands articles, which differ with type, color, manufacturer, material, price and other aspects. A good store allows not only a search by keywords. The users give a way to use a combination of search parameters (for example, color, manufacturer, interval for Large and prices).

It would be unreasonable to require the user first to set all parameters and then start the searching. Often the manufacturer X has no shirt of color Y. Therefore, it is senseless to set more parameters as size and price. This means, when the user specifies a search parameter, candidate among catalogue elements must be calculated immediately and at least the number of candidates should be displayed.

The implementation of this idea in SS/WC architecture pattern often leads to a huge waste of resources. For example, an conventional sever-side implementation means that in the moment of entering a new parameter, the browser sent to the server some request. The server converts it into some database inquiry and processes in the database. The result, for example, number of matching elements in the catalogue, sent the server back to the browser. Annoying for shop owners is that only a minor portion of searches will be completed with a purchase. Among other things, because the customer had have no intention to buy anything and just wanted "to watch something".

Processing each database query takes not only hardware resources. Regrettably, inquiries are highly differentiated in time. Often in huge number of queries come in a short time interval. In the online stores at hours when mostly customers like to shop. In the business applications to specific working hours when many employees start similar activities. Cloud solutions can reduce hardware bottlenecks somewhat, but not eliminate.

This raises the question: Is it possible to move the searching logic from the server to the client (browser)? The good news is - Yes. How it goes, I explain in the next sections.

THE IDEA

Consider some realistic example. Suppose each object from a catalog has 20 properties. Suppose that fifteen among it are binary data, two are strings with average length 40 symbols (name, short description) and three are numbers. Real numbers as prices or sizes need 3-5 symbols for their representation.

Overall, the property vector of the object will have $15 * 1 + 2 * 40 + 3 * 5 = 110$ bytes. Including additional symbols for separation and formatting (as in JSON), we arrive at about 150 bytes per vector (a row in the matrix).

A real catalog or a category of a catalog can describe dozens, hundreds, thousands or tens of thousands of objects. The size of the matrix is thus:

At 100 objects - 15 KB

At 1000 objects - 150 KB

At 10,000 objects 1.5 MB.

Therefore, these are reasonable sizes to try the matrices of browser to transfer at the beginning of the search and process the search in browser.

That is the main idea of FiSoro.

To explain the benefits of the rules FiSoro, I opened a virtual online store recently;-). In this store, the numbers from 0 to 999 are available. The customer can select the numbers according different criteria and to gain its virtual possession.

THR FiSoro COMMUNIST NUMBER STORE

The figure below shows the main part of home page of my Number Store.

Properties of a	Properties of b	Properties of c	Properties of $x = a*100+b*10+c$	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 <input type="text" value="200"/> Max value <input type="text" value="799"/> Min digit sum <input type="text" value="5"/> Max digit sum <input type="text" value="18"/>	Substring in text representation (e.g. Five) <input type="text" value="Hundred"/> Substring in description (e.g. 15) <input type="text"/>

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.

FIGURE 1: MAIN PAGE OF NUMBERS-STORE

Each offered number consists of up to three digits. The customer can set properties of each digit separately (first three columns). The whole number also has the properties as single digits, but a few special. For example, the integer can be round (divisible on ten) or "horizontal" - (all digits are the same).

Penultimate column allows considering very fine wishes of customers: for example the interval for the value of desired numbers or sum of its three digits.

Last filters allows selection in the description and textual representation of desired number.

In the beginning, all check boxes are empty and the list offered thousand numbers. If the guest of page clicks a box first time, it reduces the list of numbers. Selecting a second, third, etc. checkboxes in same column usually leads to enlargement of the selection. This happens because the operations are linked together within logical operands "AND". Thus, certain combinations may be meaningless. For example, in the displayed example, the user has checked the checkboxes for odd and even numbers in the third column. (When both checkbox are not checked, the customer would have the same result).

For numeric properties, FiSoro can set a search interval. For example, when shown on figure situation the customer is looking for the numbers with have the value between 200 and 799 and the sum of the numbers between 5 and 18.

The fields in the last column can restrict result through simple text filters yet.

Number of satisfied criteria currently sorts the selection in current version. One can see that the number 222 senses 8 criteria, and the number 444 just six.

HOW DO YOU PREPARE YOUR DATA FOR FISORO?

In next paragraphs I will explain, what steps you need to follow in order to apply FiSoro at your shop, your content management system (CMS) etc.

STEP 1: DISTRIBUTING YOUR PRODUCTS TO THE GROUPS

From experience, we can say that mostly products in some shop or system are belong to the few specific groups or categories. Try to understand your groups. Members of a group should be similar from the user's

point of view. Technically, the members of the group should have the same set of properties. I recommend managing in one group no more than a couple of thousands of products.

At next steps, you will set FiSoro separately for each group. Technically this means that you manually have to prepare two JSON files and one HTML page for each group.

First JSON file (in our case - model.json) defines the meta-model for the product group. Second file (in our case - data.json) contains the property-vectors of the group members.

In my Number Store, I have only one group of products - the whole numbers between 0 and 999.

STEP 2: DEFINE THE META-MODEL FOR YOUR PRODUCTS

At this step, you must define the relevant for user selecting properties. When it comes to the products of a shop, remember that relevant for the selection properties not necessarily are the same as properties proposed by the manufacturer. For example, the colors can be not only "Red" or "Blue, but also "Cool", "Hot" or "Hipster Style".

FiSoro supports the features the following types:

- Logical. The product has this property or not. For example, "Has red color", "Is Waterproof" etc. In the Number Store these are the properties of numbers as "The number is factorial" etc.
- Numerical properties, such as large or weight. In our shop, these are the value of the number and the sum of its digits.
- Text properties, such as a brief description, a manufacturer name, etc. In our case, this is the text representation of the number, and its brief description.

From experience, we can say that a good selection of the logical properties is the most important contribution to the success of the search.

STEP 3: GROUP PROPERTIES TOGETHER

It is often useful to structure the characteristics of the product to the groups. For example, in an audio system, it could be the Group for the sound quality, for appearance and for control. In my shop number there are groups for single digits and for the number itself.

CONSTRUCTION OF PROPERTIES VECTORS

Some JSON-line for a product from my shop (the number 221) you can see below. To better explanation of the structure of the line, its similar items are divided on four typographic lines.

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

LISTING 1: PRESENTATION OF THE NUMBER 221 IN DATA.JSON

First row contains numeric properties of numbers - their value and the sum of its digits.

The second line contains four sub-groups of logical properties. Three first groups for digits of number and last for the number itself. For example, the third group "01011010" lists the properties the digit "1". First element (0) says, that the number do not have the property "Is an even number".

The file "model.js" defines what property in which position is presented in the property vector in the data file. Structure of this file will be explained below.

Third line in the listing above shows the values of two text properties of the product: the textual presentation of numbers and their brief description.

The last line is a link to the presentation of this product in a catalog. Normally this page contains photos and a detailed description of the product, customer reviews etc. Unfortunately, these things not existing now in my shop ;-)

STRUCTURE OF META-MODEL FILE

Meta-model FiSoro is hierarchical. On top level, it contains so-called filter groups.

In the figure below, the group for the first left digit is presented. This digit is the amount of hundreds in some number. This group has the name "NX100". The Type "Bit Vector" indicates that the group represents a vector with bit elements (they can be either zero or one). The entry "column data" defines the position of entry in the JSON row for this group (starting with 0).

After this data you can see the element with FiSoro filters. The order of filters in the meta-model must be the same in all JSON-line. The arrow in the figure shows that the property "is prime" must be displayed on third position.

```
{ "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" }
    ]
  },
  { "name": "10101010", "10101010", "01011010", "0101010100",
```

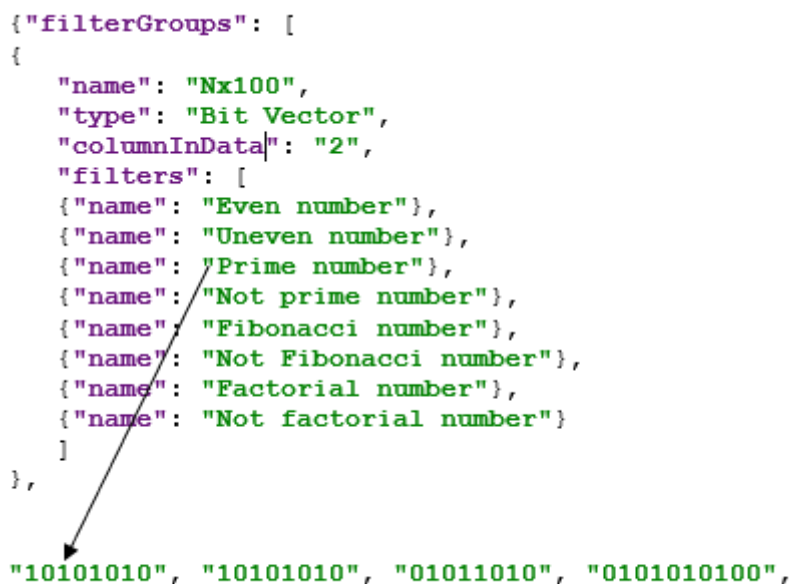


FIGURE 2: MAPPING BETWEEN ELEMENTS IN MODEL.JSON AND DATA.JSON.

STRUCTURE OF THE HTML PAGE

We will not discuss in this article implementation details of FiSoro. Our goal is to show how to apply FiSoro.

By processing the JSON data and the display of data on the HTML page helps us AngularJS 1.

That is why our HTML-page ends with the instructions for loading an Angular library and FiSoro Library:

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

LISTING 2: END OF HTML-PAGE

Interesting for us part of page starts with the div element with embedded controllers. Inside this div-element the table and its headers are defined.

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

LISTING 3: DEFINITION OF CONTROLLER ON HTML-PAGE

The next row of the table contains the cells with the Angular's elements. These elements enable to switch-on and switch-off corresponding filters.

```
<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>
  </tr>
...
```

```
<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>
  </td>
</tr>
```

LISTING 4: ELEMENTS FOR SWITCH-ON AND SWITCH-OFF OF FILTERS.

Let us to compare the Figure 2 and Listing 4. Each group of properties from Data Model is mapped to the element of HTML using an Angular's element of type "data-ng-repeat". On the other hand, each such group is represented in data.json with some "column". Under "column" is means here the elements at the same positions in different lines of the file.

You find more details when you analyze the source code of the HTML page and JSON files. I recommend to download the files from the demo subdirectory and to adjust them step-by-step to your task.

CONCLUSION AND OUTLOOK

Experience from the first applications of FiSoro show sufficient its functionality for many practically interesting tasks. It was amazing, how easily and quickly this tool can be set up in production.

Many modifications and improvements are thinkable. If you have interesting ideas, please use tickets in GitHub.

Unfortunately, I cannot promise you that these or other feature will be quickly implemented. Nevertheless, if you himself implement some new feature or fix some bug - please make corresponding pull request in GitHub.