LimeSurvey-Tutorial

Autocomplete

Februar 2026

Inhaltsverzeichnis

Vorbemerkung

Diese kleinen Tutorials sollen Möglichkeiten aufzeigen, durch Einsatz von javascript und css Effekte bei der Darstellung von Fragen zu zeigen, die den Teilnehmern die Beantwortung erleichtert.

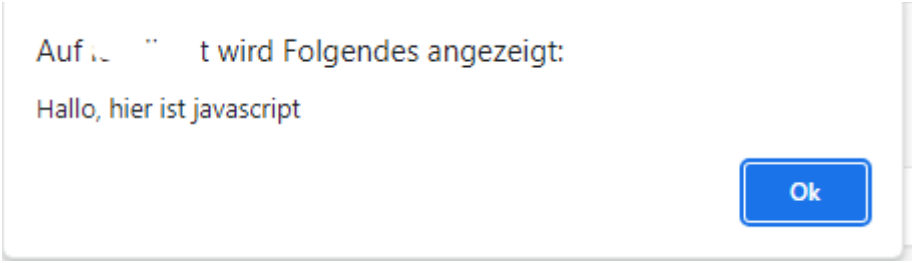
Die dargestellten Codes sollten nicht aus diesem Text in LimeSurvey kopiert werden.
Es könnten sich noch Reste der Formatierung im Text befinden, der den Code dann unbrauchbar macht.
Daher bitte diese Codestücke immer aus der Beispielstudie übernehmen.

Da nahezu alle gezeigten Beispiele mit javascript realisiert werden, ist natürlich die Grundvoraussetzung, dass dieses eingesetzt werden kann.

Ein einfacher Test ist dies in den Quellcode der ersten Frage einzufügen

```
<script type="text/javascript" charset="utf-8">
$(document).on('ready pjax:scriptcomplete',function(){
    alert("Hallo, hier ist javascript");
});
</script>
```

Wenn dann dieses Fenster aufploppt, ist alle in Ordnung; wenn nicht, ...



Und dann noch dieses:

Dies ist der Anfang – es wird mit anderen Bibliotheken weitergehen

Allgemeines

Zunächst einmal zwei Definitionen von „autocomplete“

Im Web Design bezeichnet »Autocomplete« das selbsttätige Anbieten der Vervollständigung der Suchanfrage schon während der Eingabe. Die Vorschläge werden *im Eingabefeld* angezeigt und sollen vor allem die Eingabe beschleunigen und erleichtern. Das Angebot der Vervollständigung richtet sich damit an Nutzer, die eher schon wissen, was sie suchen.

Autovervollständigung oder **Autocomplete** ist eine Funktion, die eine Benutzereingabe sinnvoll ergänzt. Grundlage für die Vervollständigung ist in jedem Fall eine begrenzte Anzahl von Möglichkeiten, die sich äquivalent zum Fortschritt und ggf. Semantik der Benutzereingabe verkleinert.

In diesem Tutorial möchte ich nun einige Möglichkeiten zeigen, solche „Autocompletes“ einzusetzen.

Außerdem sollen diese Beispiele mit vier verschiedenen „Bibliotheken“ realisiert werden, damit jeder das für ihn geeignetste auswählen kann.

- zwei erfordern den Einsatz von javascript
- eins erfordert die Installation eines plugins.
- eins erfordert die Installation einer Fragevorlage

Die dargestellten Codes sollten nicht aus diesem Text in LimeSurvey kopiert werden.
Es könnten sich noch Reste der Formatierung im Text befinden, der den Code dann unbrauchbar macht.
Daher bitte diese Codestücke immer aus der Beispielstudie übernehmen.

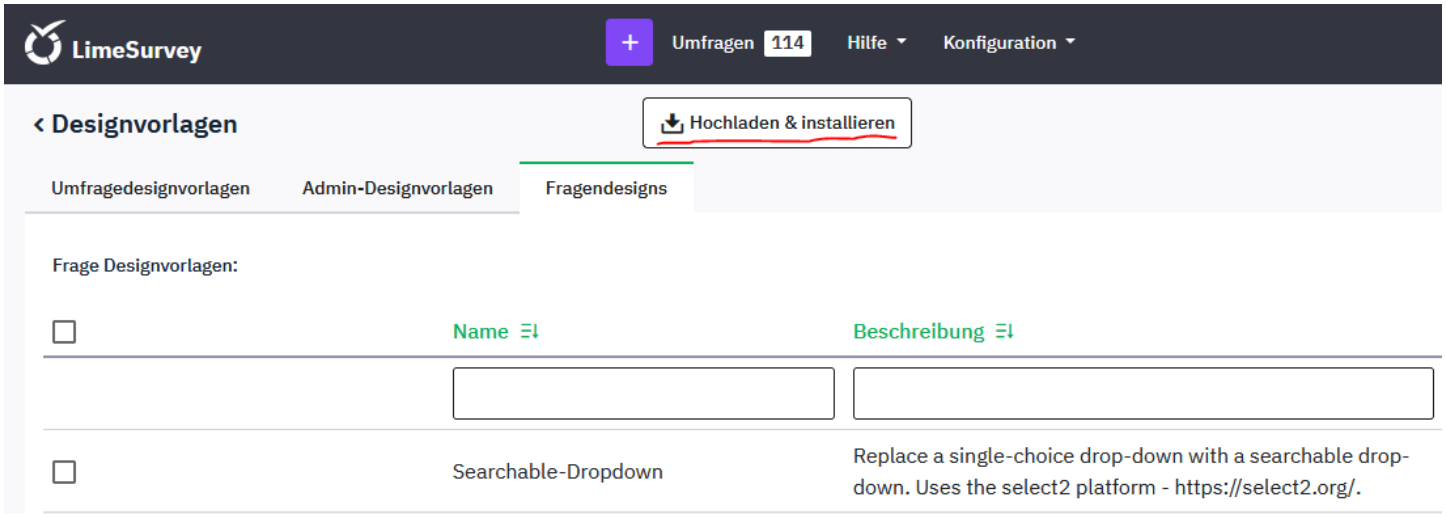
1. Searchable Drop-Down

Dies ist eine Fragevorlage, entwickelt von Tony Partner (tpartner). Man findet sie hier:

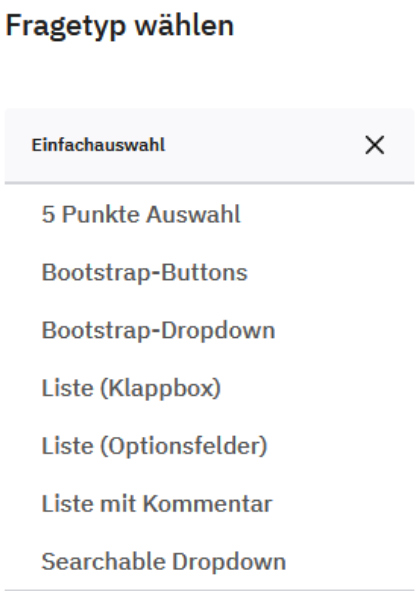
https://github.com/tpartner/LimeSurvey-Searchable-Dropdown-6x

1.1. Installation

Wichtig: Heruntergeladen wird eine Datei „LimeSurvey-Searchable-Dropdown-6x-main.zip“.
Diese muss entzippt werden, und das Verzeichnis „ Searchable-Dropdown“ wieder gezippt werden, damit man es hier „Hochladen & Installieren“ kann. (wie es auch auf der o.a. Seite steht).
Extract the download, compress (zip) the Searchable-Dropdown folder and import via the theme manager



Und danach ist es als Fragetyp hier auswählbar.



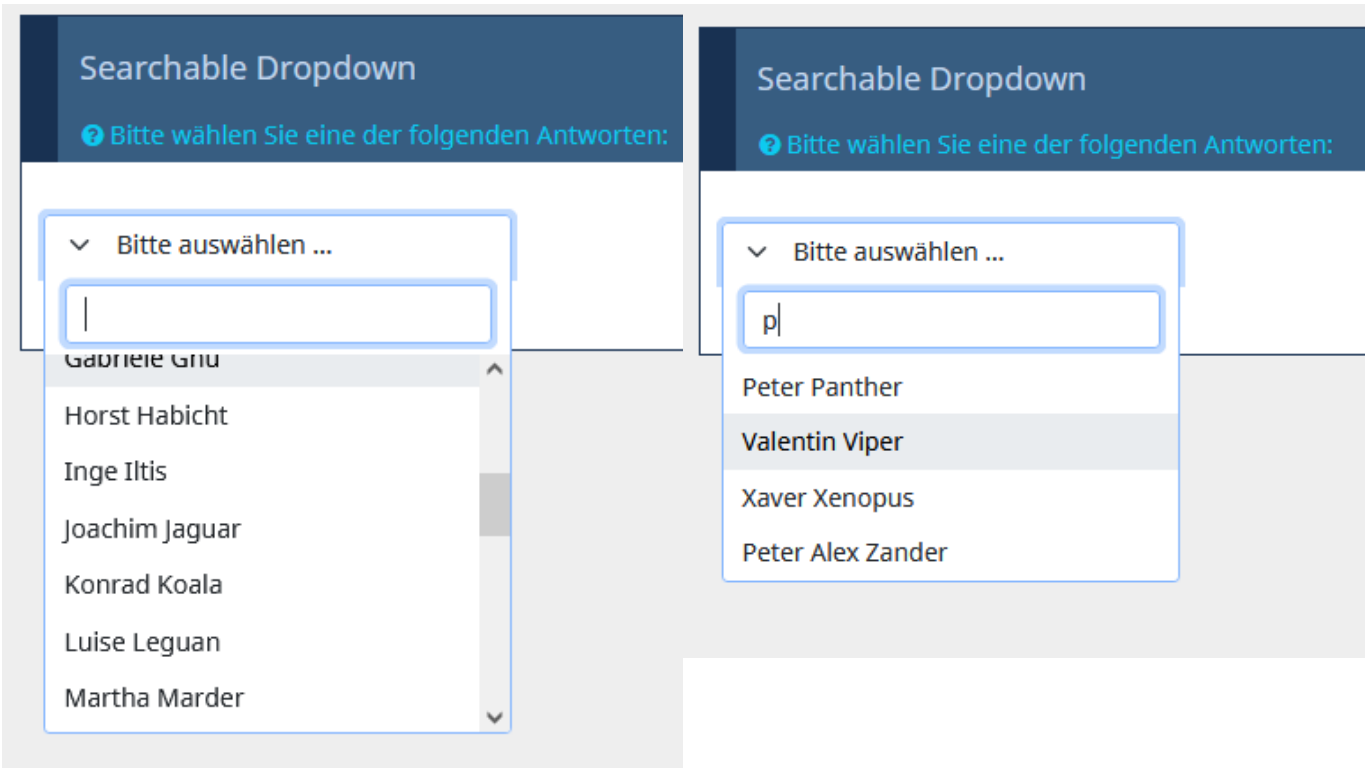
1.2. Anwendungsbeispiel

Häufig kommt es vor, dass ein Element aus einer sehr großen Menge auszuwählen ist..

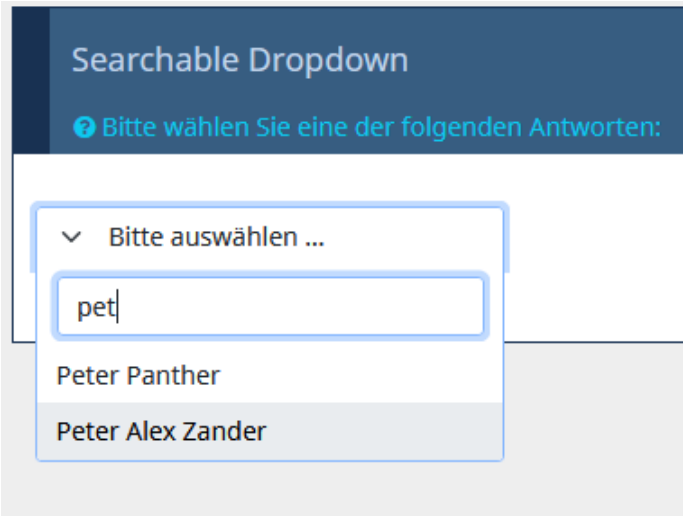
Dazu wird häufig ein „Drop-Down“ benutzt. Diese hat allerdings den Nachteil, dass der Teilnehmer durch die vorgegebenen Antwortmöglichkeiten durchscrollen muss, um seine gewünschte Antwort zu finden. Dies ist recht mühsam, insbesondere, wenn es sich um recht viele Optionen handelt.

Nehmen wir einmal als Beispiel die Mitschüler in der Klasse.

Beim „üblichen“ Start des Drop-Downs“ sieht man alle Optionen – allerdings mit einem Scrollbalken.
Nach Eingabe von „p“ werden nur noch die Namen angezeigt, die irgendwo ein „p“ enthalten.



Und bei weiterer Eingabe „pet“ werden nur noch die beiden „Peter“ angezeigt.



1.3. Implementierung

Das es sich um eine ganz normale Einfachnennung handelt, ist hier nichts zu schreiben. Die Antwortoptionen werden wie gewohnt in LimeSurvey eingetragen.

Und hier kommt auch ein gewisser Nachteil. Sofern es sich um sehr viele Optionen handelt – und dafür ist ein „Autocomplete“ ja gedacht, können Einstellungen in der zugrundeliegenden Programmiersprache php dafür sorgen, dass es ein Limit gibt. Diese sind sicherlich bei jedem verschieden.

Zum Beispiel konnte ich in meiner Installation 1000 Antwortoptionen speichern, 1500 dagegen nicht mehr.
Eine Liste „Städte mit Postleitzahlen in Deutschland“ ist also nicht möglich.
Aber für Dinge wie „Länder der Erde“ ist es allemal geeignet

Für jede Antwortoption wird ja ein neuer Datensatz in der Datei „lime_answers“ gespeichert.

Aber dies ist eine sehr einfache, schnell zu realisierende Möglichkeit, für ein solches „Autocomplete“, welches nur eine einzige Liste anzeigen soll.

Im weiteren Verlauf werden Beispiele gezeigt, bei denen Listen aufeinander aufbauen,

z.B. „Bundesland - Landkreis - Ort“

2. Jquery-UI-Autocomplete

Dies ist das „autocomplete“, welches auch im Handbuch unter

https://www.limesurvey.org/manual/Workarounds:_Manipulating_a_survey_at_runtime_using_Javascript#Update_for_LimeSurvey_2.05+

beschrieben wird.

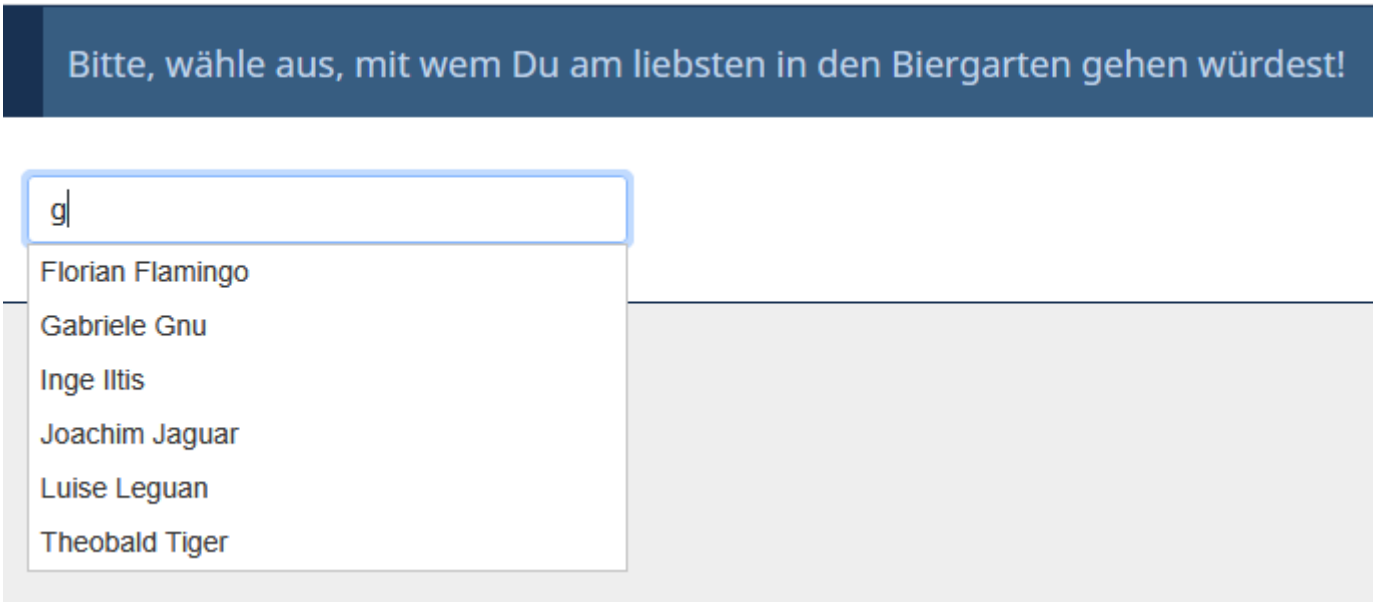
Ich werde hier trotzdem noch einmal alles zeigen, da die im Handbuch gezeigte Version manches enthält, was heutzutage nicht mehr notwendig ist.

2.1. Anwendungsbeispiel

2.1.1. Fragestellung

Hierfür nehmen wir wieder die Mitschüler und kreieren eine Frage vom Typ „kurzer Text“.

Mithilfe des scripts wird daraus dann ein Drop-Down.



Das sieht ziemlich ähnlich dem vorherigen aus, allerdings befinden sich hier die Optionen im Fragetext.

Und dieser Fragetext ist in der Datenbank als Type „mediumtext“ implementiert. D.h. er kann eine Länge von ca. 16MB speichern.

2.1.2. Implementierung 1 (innerhalb des scriptes)

Im Fragetext – im Quellcode-Modus – werden zunächst zwei Bibliotheken geladen

```
<link href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css" rel="stylesheet" />
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
```

Diese werden hier vom CDN (Content Delivery Network) geladen; man kann sie aber auch separate Dateien herunterladen und im „files“-Verzeichnis speichern und dann von dort aufrufen (Dies ist sicher im Sinne der DSGVO die sauberere Lösung)

Danach folgt der eigentliche Code

```
<script type="text/javascript" charset="utf-8">
  $(document).ready(function() {
    $('#question{QID} input[type="text"]').autocomplete({
      minLength: 1,
      source: [
        "Adalbert Ameise", "Berta Biber",  "Carlo Chamäleon",  "Doris Dachs",   "Ernst Eidechse",
        "Florian Flamingo",  "Gabriele Gnu",  "Horst Habicht",  "Inge Iltis",  "Joachim Jaguar",
        "Konrad Koala",  "Luise Leguan",  "Martha Marder",  "Norbert Natter",  "Ottilie Ozelot",
        "Peter Panther",  "Quasimodo Qualle",  "Renier Reiher",   "Sabine Storch",  "Theobald Tiger",
        "Ulrike Unke",  "Valentin Viper",  "Wanda Waran",   "Xaver Xenopus",  "Yvonne Yak",
        "Zacharias Zebra",  "Peter Alex Zander"
      ]
    });
  });
</script>
```

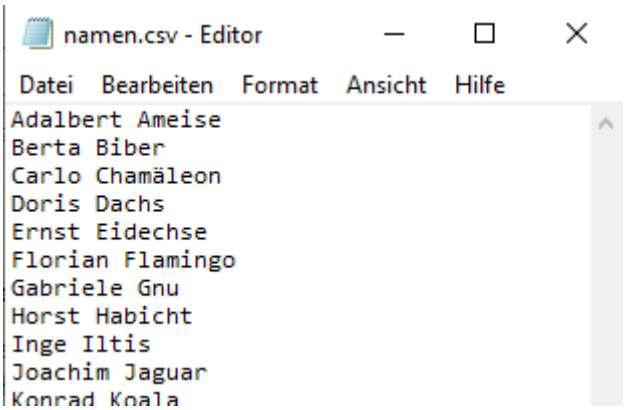
Das einzig anzumerkende ist: „minLength: 1“ bedeutet, dass nach dem ersten eingegebenen Buchstaben das System anspringt; das kann man also beliebig ändern.

2.1.3. Implementierung 2 (als Textdatei)

Die oben gezeigte Implementierung ist sicher für wenige Optionen gut geeignet – auch die „Staaten der Erde“ passen noch ganz gut – aber bei mehr wird es dann doch unhandlich.

Daher kann man die Optionen auch aus einer Textdatei lesen lassen.

Diese sieht ganz analog aus, pro Zeile eine Option; hier einmal im „Microsoft Editor“



Nun benötigen wir aber noch eine Bibliothek, um diese Datei zu lesen.

```
// Die zwei bekannten
<link href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css" rel="stylesheet" />
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
// Und diese hier zusätzlich
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-csv/1.0.11/jquery.csv.min.js"></script>
```

Danach folgt der eigentliche Code zum Aufruf:

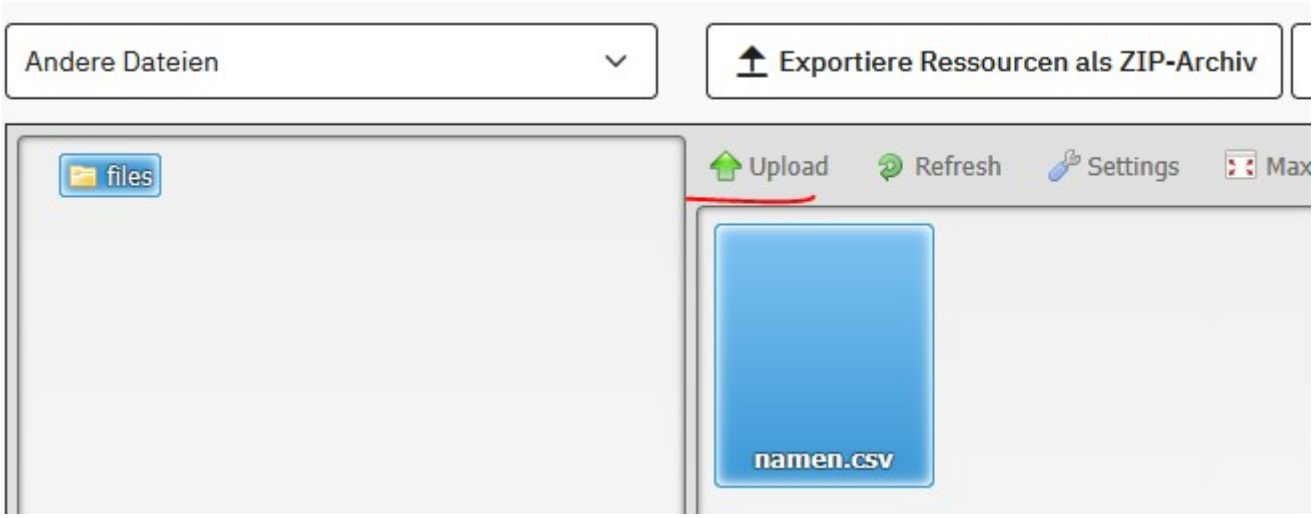
```
<script type="text/javascript" charset="utf-8">
  $(document).on('ready pjax:complete',function() {
    // Hier wird der Speicherort der Datei angegeben
    var url = "/upload/surveys/{SID}/files/namen.csv";

    var namen = new Array();

    $.get(url,function(data){
      fullArray = $.csv.toArrays(data);
      $(fullArray).each(function(i, item){
        namen.push(item[0]);
      });
      $("#question{QID} input[type=text]").autocomplete({
        minLength: 1,
        source: namen
      });
    });
  });
</script>
```

Noch eine Erläuterung zur Url:

Die Datei „namen.csv“ wurde in das „files“-Verzeichnuis der Umfrage hochgeladen.



2.1.4. Ergänzung

Dieses „autocomplete“ kann auch in andere Fragetypen eingebaut werden.

2.1.4.1. „mehrfache kurze Texte“

Dann ändert sich nur eine einzige Zeile.

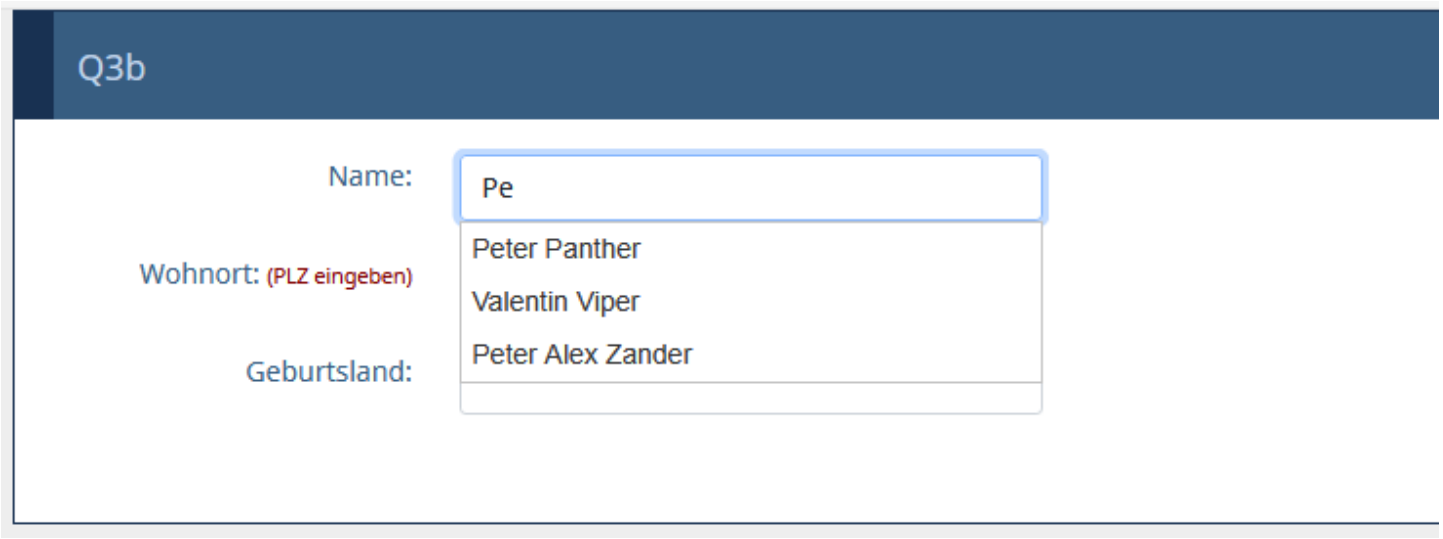
```
$("#question{QID} input[type=text]").autocomplete({
```

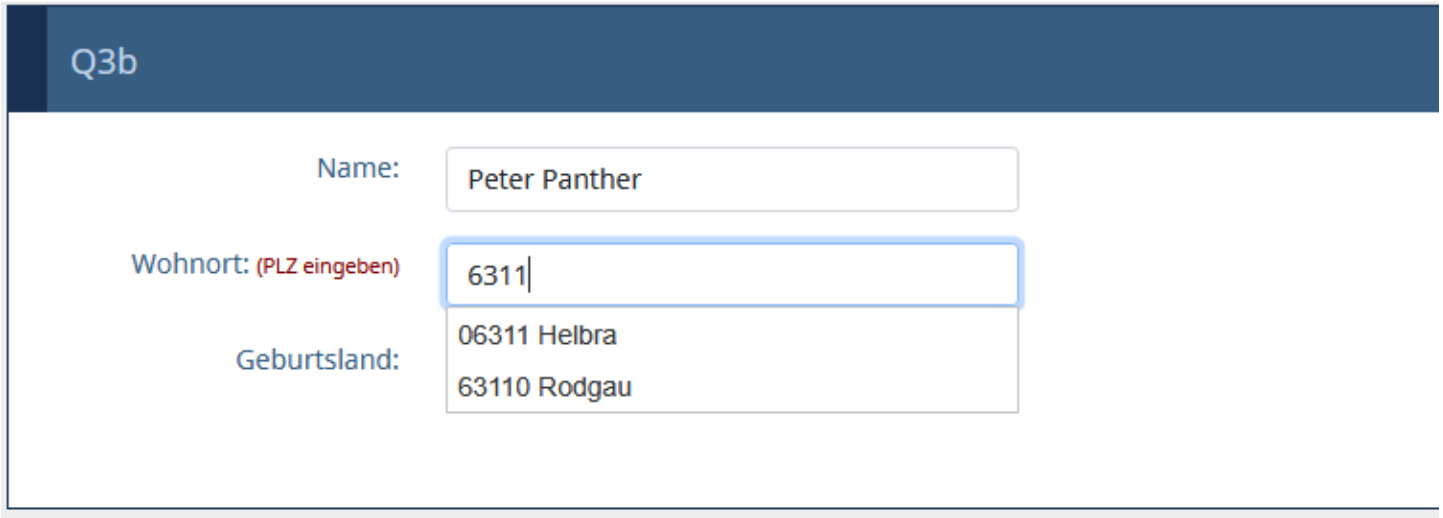
wird zu:

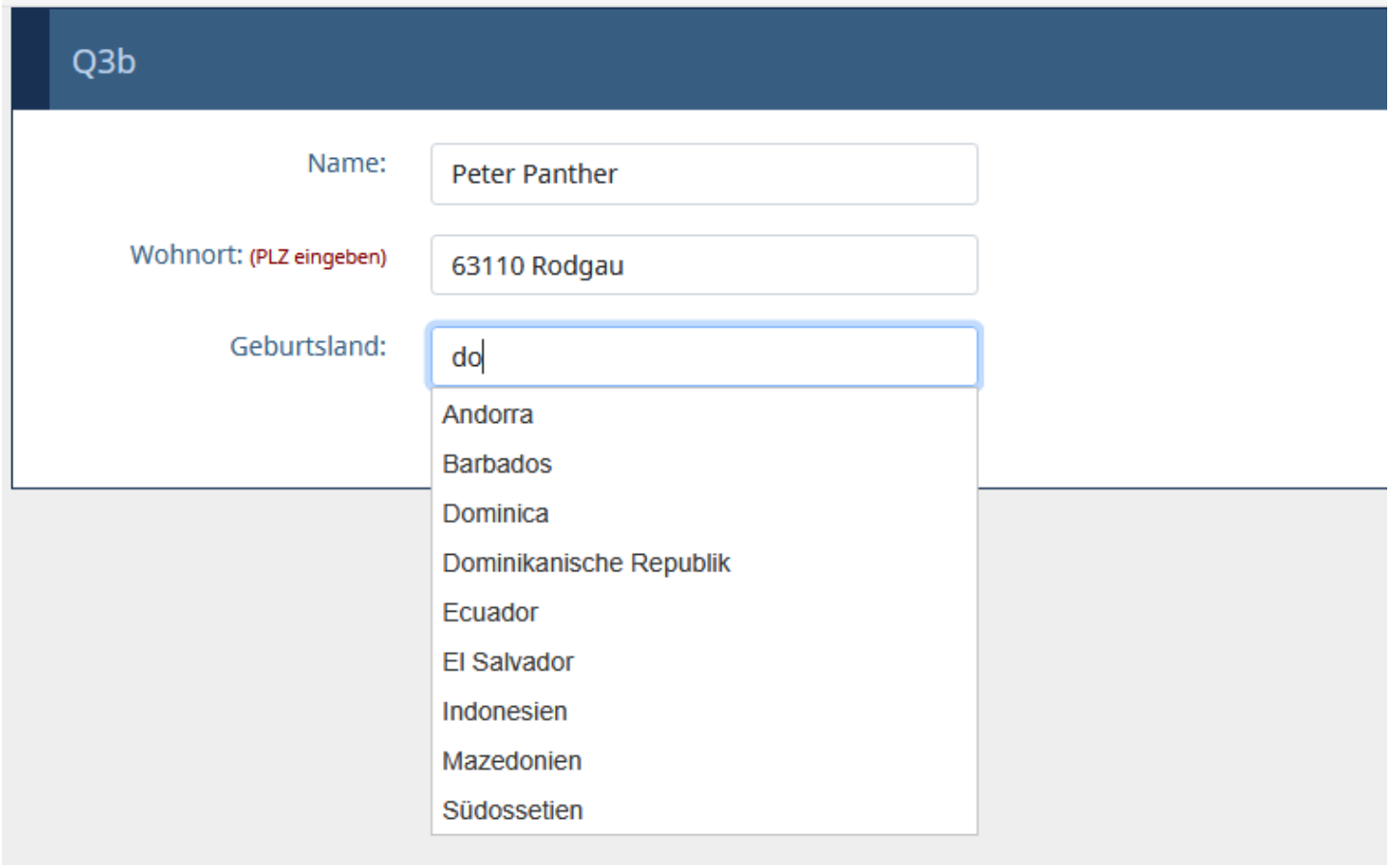
```
$("#question{QID} input[type=text]:eq(0)").autocomplete({
```

Hier wird durch „:eq(x)“ gewählt, in welcher Zeile das „autocomplete“ eingefügt wird; Zählung beginnt bei „0“ für die erste Zeile.

Auf diese Weise kann man also mehrere Angaben zur gleichen Zeit abfragen.






Q3b
Name:
Peter Panther
Wohnort: (PLZ eingeben)
63110 Rodgau
Geburtsland:
do
Andorra
Barbados
Dominica
Dominikanische Republik
Ecuador
El Salvador
Indonesien
Mazedonien
Südossetien

2.1.4.2. „Matrix(Text)“

Hier wird das

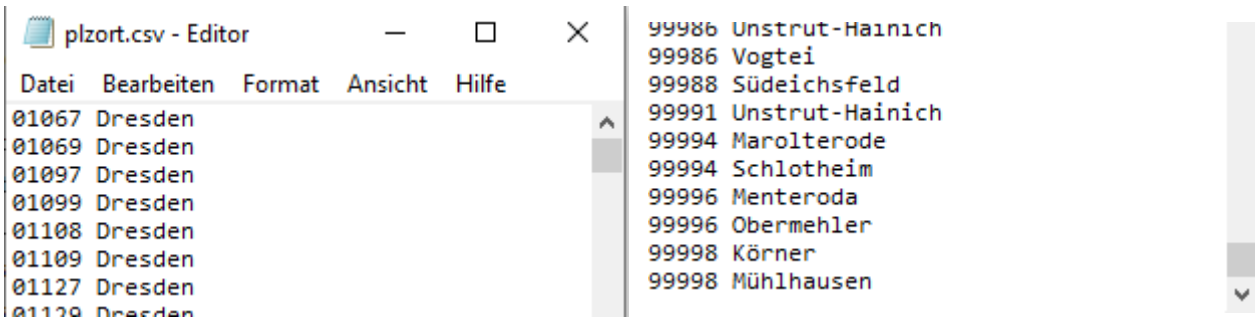
```
$("#question{QID} input[type=text]").autocomplete({
```

zu:

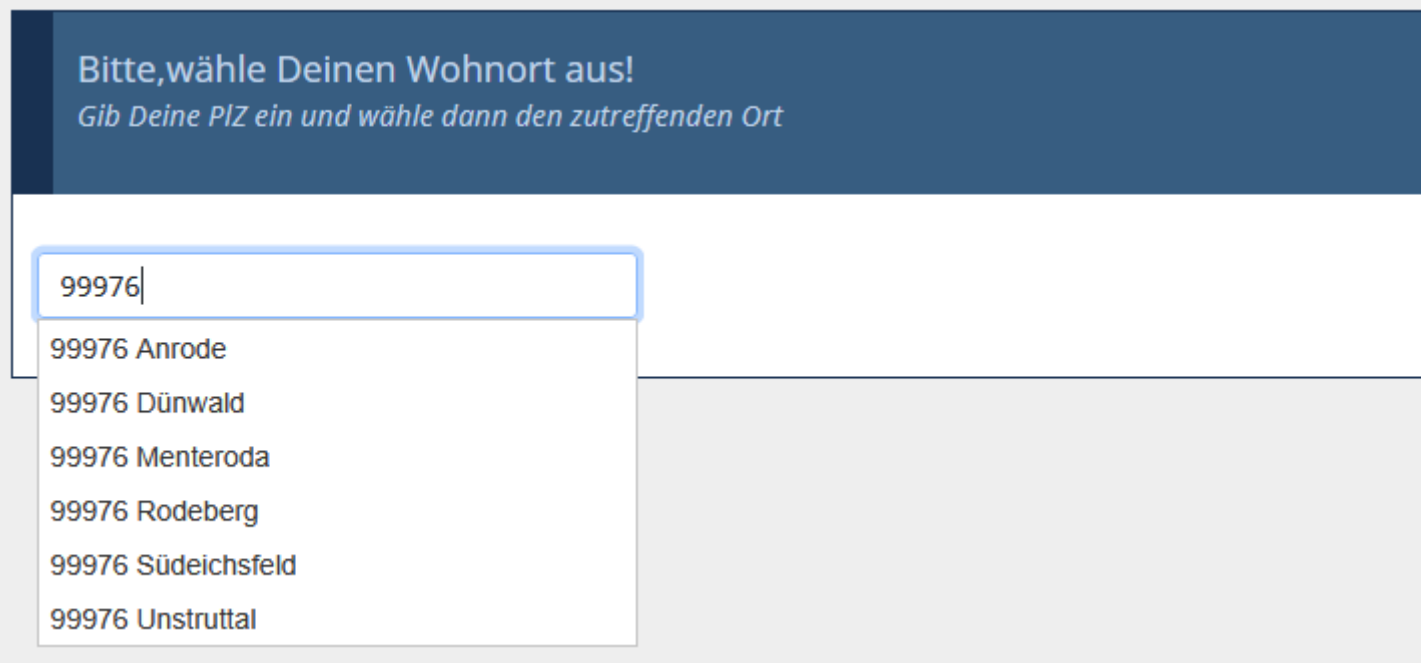
```
$("#question{QID} .answer-item:nth-child(x) input[type=text]").autocomplete({
```

Durch „:nth-child(x)“ wird die Spalte gewählt, in welcher das „autocomplete“ eingefügt wird; Zählung beginnt bei „2“ für die erste Spalte. (Die Spalte mit den Teilfragentexten hat den Index „1“)

Übrigens ist die Datei der Postleitzahlen ca. 13000 Einträge. Sie sieht so aus:



Dies ist nicht trivial, da es manchmal mehrere Orte mit derselben PLZ gibt.



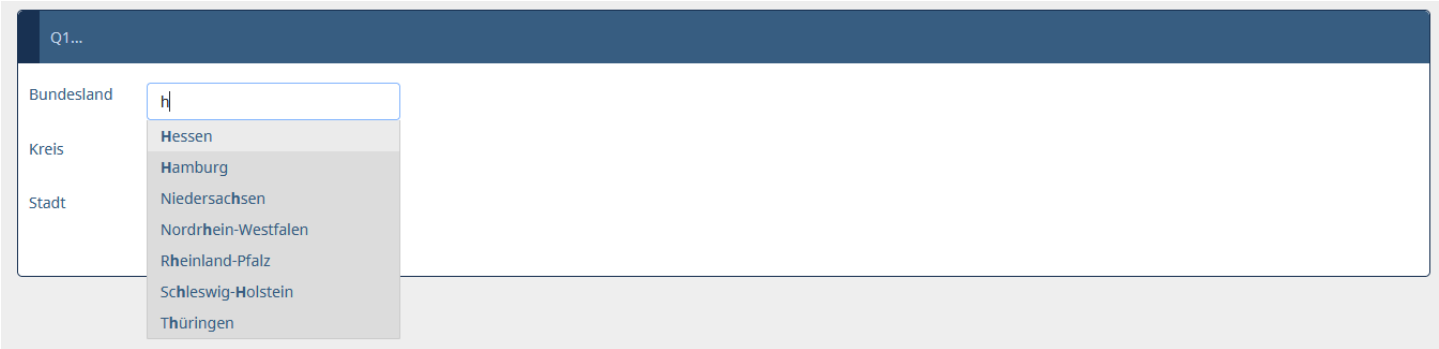
3. Easy-autocomplete

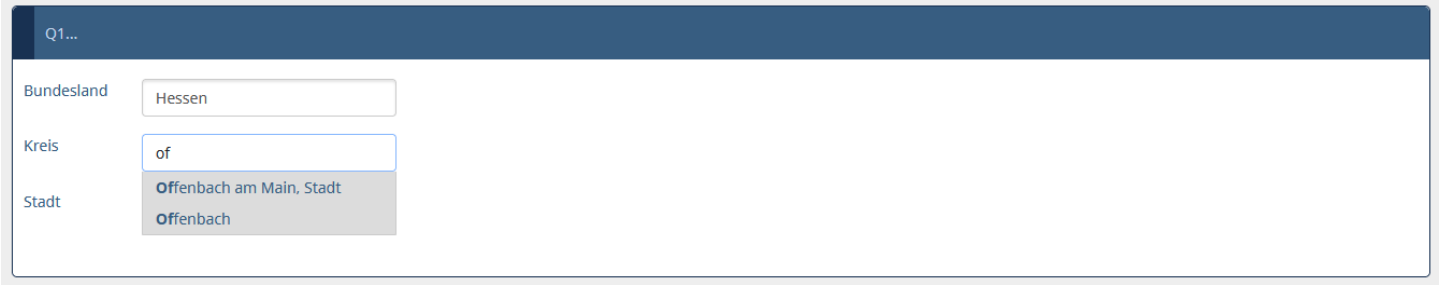
3.1. Anwendungsbeispiel

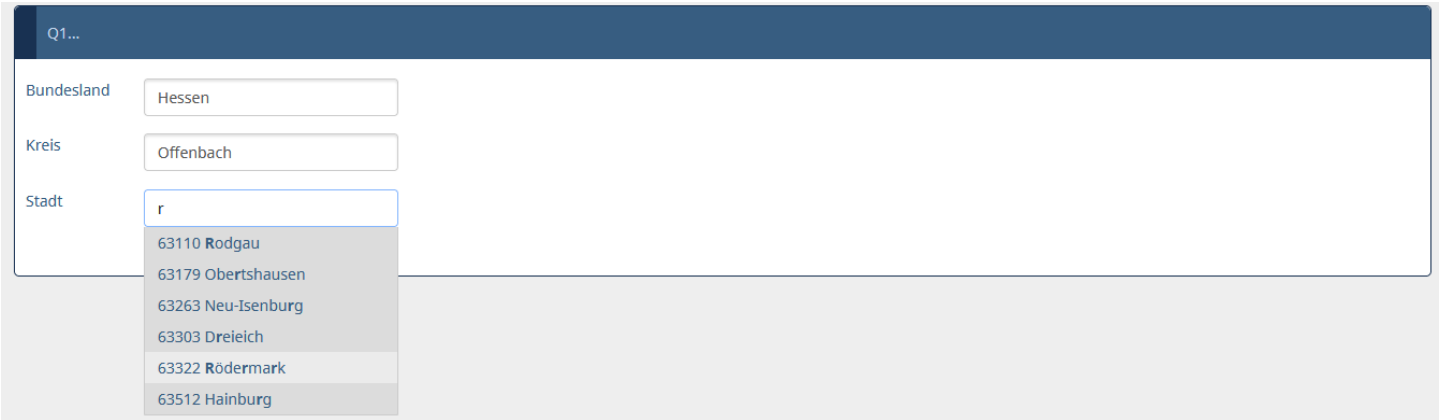
Die Lösung in 2.1.4. ließ zwar mehrere Abfragen in einer Frage vom Typ „mehrfache kurze Texte“ zu, allerdings bauten diese nicht aufeinander auf, sondern die Teilfragen waren voneinander unabhängig.

Mit dieser Implementierung von „easy-autocomplete“ lassen sich nun voneinander abhängige Teilfragen erstellen.

Hier soll es einmal die Hierarchie „Bundesland – Landkreis – Ort“ sein.







3.2. Implementierung

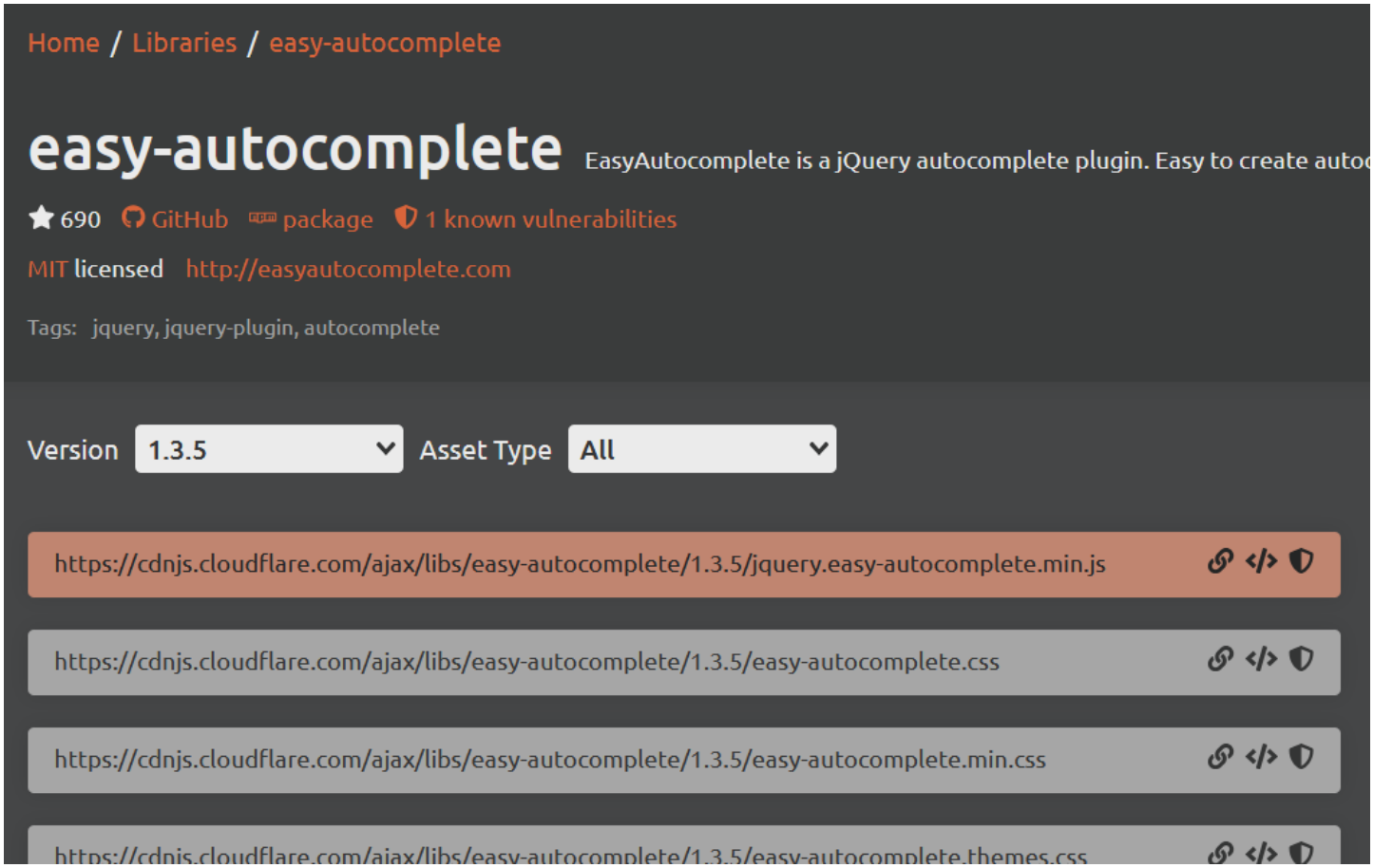
Hierzu werden zwei Dateien benötigt – wie so oft.

Hier beide vom CDN geladen.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/easy-autocomplete/1.3.5/jquery.easy-autocomplete.min.js"></
script>

<link href="https://cdnjs.cloudflare.com/ajax/libs/easy-autocomplete/1.3.5/easy-autocomplete.css"
rel="stylesheet" />
```

Im Sinne der DSGVO ist es sauberer, die beiden Dateien hier https://cdnjs.com/libraries/easy-autocomplete herunterzuladen und im „files“-Ordner der Umfrage zu speichern.



In einer Frage vom Typ „mehrfache kurze Texte“ werden also diese drei Teilfragen angelegt und im Fragetext (im Quellcode-Modus) wird nun dieses sehr, sehr lange script eingefügt.

Zunächst die Anfangszeilen (mit dem Hinweis, dass das Original von Tony Partner stammt – wie die meisten der scripte, die ich hier zeige)

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/easy-autocomplete/1.3.5/jquery.easy-autocomplete.min.js"></
script>
<link href="https://cdnjs.cloudflare.com/ajax/libs/easy-autocomplete/1.3.5/easy-autocomplete.css"
rel="stylesheet" />


<script type="text/javascript" data-author="Tony Partner">
  $(document).on('ready pjax:scriptcomplete',function(){


    // Identify this question
    var thisQuestion = $('#question{QID}');
```

Und nun werden die hierarchischen Daten eingetragen.

Man sieht auch, dass die Frage nach dem Ort eigentlich die Frage nach der Postleitzahl ist.

```
    // Define the data
    var fullData = [
...
{ 'Bland': 'Bremen', 'Lkreis': 'Bremerhaven, Stadt', 'Stadt': '27576 Bremerhaven' },
{ 'Bland': 'Bremen', 'Lkreis': 'Bremerhaven, Stadt', 'Stadt': '27578 Bremerhaven' },
{ 'Bland': 'Bremen', 'Lkreis': 'Bremerhaven, Stadt', 'Stadt': '27580 Bremerhaven' },
{ 'Bland': 'Bremen', 'Lkreis': 'Bremen', 'Stadt': '28195 Bremen' },
{ 'Bland': 'Bremen', 'Lkreis': 'Bremen', 'Stadt': '28197 Bremen' },
...
{ 'Bland': 'Hessen', 'Lkreis': 'Lahn-Dill-Kreis', 'Stadt': '35585 Wetzlar' },
{ 'Bland': 'Hessen', 'Lkreis': 'Lahn-Dill-Kreis', 'Stadt': '35586 Wetzlar' },
{ 'Bland': 'Hessen', 'Lkreis': 'Lahn-Dill-Kreis', 'Stadt': '35606 Solms' },
{ 'Bland': 'Hessen', 'Lkreis': 'Lahn-Dill-Kreis', 'Stadt': '35614 Aßlar' },
{ 'Bland': 'Hessen', 'Lkreis': 'Lahn-Dill-Kreis', 'Stadt': '35619 Braunfels Tiefenbach' },
...
{ 'Bland': 'Thüringen', 'Lkreis': 'Unstrut-Hainich-Kreis', 'Stadt': '99998 Körner' }
];
```

Das heißt also, dass für alle Postleitzahlen diese Zeile angelegt wird.

Man sieht die Struktur; die Kennungen sind ‚Bland‘, ‚Lkreis‘ und ‚Stadt‘, die nun als key benutzt werden.

```
    // Define some elements and vars
    var keys = ['Bland', 'Lkreis', 'Stadt'];
    var input1 = $('.answer-item:eq(0) input:text.form-control', thisQuestion);
    var inputs = {
      0: $('.answer-item:eq(0) input:text.form-control', thisQuestion),
      1: $('.answer-item:eq(1) input:text.form-control', thisQuestion),
      2: $('.answer-item:eq(2) input:text.form-control', thisQuestion)
    }
    var answers = {
      0: $.trim($(inputs[0]).val()),
      1: $.trim($(inputs[1]).val()),
      2: $.trim($(inputs[2]).val())
    }
```

```
    // Create an array of "Bland" values
    var Blands = [];
    $.each(fullData, function(i, val) {
      if(!Blands.includes(val.Bland)) {
        Blands.push(val.Bland);
      }
    });

    // Initiate autocomplete on the first input
    var aOptions1 = {
      data: Blands,
      list: {
        maxNumberOfElements: 100,
        match: {
          enabled: true
        },
        onChooseEvent: function() {

          //If new selection...
          if($.trim(inputs[0].val()) != answers[0]) {

            answers[0] = $.trim(inputs[0].val());

            // Reset following items
            $('li.answer-item:gt(0) input:text', thisQuestion).val('').trigger('keyup').easyAutocomplete({ 'data':
'' });

            //Initiate autocomplete on next input
            childAutocomplete(0);
          }
        }
      }
    };
    inputs[0].easyAutocomplete(aOptions1);

    function childAutocomplete(index) {

      if((index+1) < $('li.answer-item', thisQuestion).length) {
        var parentInput = inputs[index];
        var nextInput = inputs[(index+1)];

        var key1 = keys[index];
        var key2 = keys[(index+1)];

        // Define the new data
        var thisFullData = fullData.filter(function(obj) {
          return (obj[key1] == $.trim(parentInput.val()));
        });
        var thisData = [];
        $.each(thisFullData, function(i, val) {
```

```
          if(!thisData.includes(val[key2])) {
            thisData.push(val[key2]);
          }
        });

        //Initiate autocomplete on next input
        var aOptions = {
          data: thisData,
          list: {
            maxNumberOfElements: 100,
            match: {
              enabled: true
            },
            onChooseEvent: function() {

              //If new selection...
              if($.trim($(nextInput).val()) != answers[(index+1)]) {

                answers[(index+1)] = $.trim($(nextInput).val());

                // Reset following items
                $('li.answer-item:gt('+index+1+') input:text',
thisQuestion).val('').trigger('keyup').easyAutocomplete({ 'data': '' });

                //Initiate autocomplete on next input
                childAutocomplete(index+1);
              }
            }
          }
        };
        nextInput.easyAutocomplete(aOptions);
      }
    }

    // Returning to page
    if($.trim(inputs[0].val()) != '') {
      childAutocomplete(0);
    }
    if($.trim(inputs[1].val()) != '') {
      childAutocomplete(1);
    }
    });
</script>
```

Zum Schluss noch ein bisschen css, wo man natürlich die Werte nach Wunsch anpassen kann; z.B. die „max-height“, die maximale Höhe des Drop-Downs, und auch dessen Hintergrundfarbe, die ich hier auf ‚Gainsboro‘ gesetzt habe.

```
<style type="text/css">
.easy-autocomplete-container {
  left: 0;
  position: absolute;
  width: 100%;
  z-index: 2;
  max-height: 240px;
  overflow: auto;
}
  .easy-autocomplete-container ul {
  background: none repeat scroll 0 0 Gainsboro;
</style>
```

Wer sich näher mit den umfangreichen Features von ‚easy-autocomplete‘ beschäftigen möchte, hier zwei Links

https://shift.digital/insights/jquery-easyautocomplete-documentation-and-usage-guide

https://pub.dev/documentation/easy_autocomplete/latest/

Zugegeben, beide nicht so „das Gelbe vom Ei“

4. Anhang

4.1.1. lss-Export einer Beispiel-Datei

Hier ist noch kein Link zu einer gezippten lss-Datei, die die hier beschriebenen Beispiele enthält.