gApp konfigurieren

Um gApp zu konfigurieren, sollten Sie zunächst wissen, dass gApp aus zwei verschiedenen Komponenten besteht.

Die eine Komponente ist guide4you, ein Programm, das eigentlich für die Kartendarstellung im Webbrowser entwickelt wurde.

Die andere Komponente ist die eigentliche App, die das Erfassen und Bearbeiten der Daten übernimmt, für die Übertragung von Daten auf den Server zuständig ist, Kartenmaterial für den Offline-Betrieb auf das Mobilgerät kopieren kann, usw.

Konfiguration der Komponente *guide4you*

guide4you hat zwei Konfigurationsdateien, nämlich

- **client.json** und
- **layers.json**

client.json legt das Aussehen und Verhalten der guide4you-Komponente fest, in **layers.json** wird beschrieben, welches Kartenmaterial zur Auswahl steht.

Eigentlich haben diese Dateien bei gApp exakt den gleichen Inhalt wie bei guide4you als eigenständiger Anwendung, allerdings werden in **client.json** zwei bestimmte Einträge für die korrekte Funktion benötigt:

```
{
  "baseLayers": [
    {
      "id": "OSM-local",
      "title": "lokale Kacheln",
      "type": "OSM",
      "source": {
        "url": "http://127.0.0.1:8080/tiles/{z}/{x}/{y}.png",
        "attribution": "&copy; <a href=\"#\"
onclick=\"window.parent.openExternally('http://www.openstreetmap.org/copyright');\">Ope
nStreetMap</a> Mitwirkende"
      },
      "visible": false
    }
  ],
  "fixedFeatureLayers": [
    {
      "id": "POIs",
      "title": "Erfasste Daten",
      "type": "KML",
      "source": {
        "url": "http://127.0.0.1:8080/data.kml",
        "attribution": "Selbst erhobene Daten"
      },
      "style": "#defaultStyle",
      "visible": true
    }
  ]
}
```

Der erste Eintrag wird benötigt, damit auf das Mobilgerät übertragene Kartenmaterial angezeigt werden kann, der zweite, damit die mit dem Mobilgerät erfassten Daten angezeigt werden. Sieht man von diesen Besonderheiten ab, können Sie die Beschreibung dieser Konfigurationsdateien dem guide4you-Benutzerhandbuch entnehmen.

Konfiguration der eigentlichen App

Während die Einstellungen der Komponente guide4you sich vorwiegend auf das Aussehen der Kartendarstellung auswirken, sind die Konfigurationsdateien der eigentlichen App nicht nur für das Aussehen des Erfassungsformulars sondern auch für die Struktur der Datenbank verantwortlich, es sind dies **form.html** und **form.json**

form.html - das Aussehen des eigentlichen Formulars

form.html legt hauptsächlich das Aussehen des Formulars fest, während **form.json** hauptsächlich die zu erfassenden Daten beschreibt und wie sie für die Datenbank und die Darstellung auf der Karte aufbereitet werden.

Wie diese beiden Dateien aufgebaut sind, lässt sich am besten an konkreten Beispielen darstellen. Um die Grundlagen zu erläutern, betrachten wir die relativ einfach aufgebaute Standardmaske. Als Beispiel für eine aufwendigere Konfiguration betrachten wir dann die Beispielmaske. Zunächst also die Standardmaske.

Werfen wir zunächst einen Blick auf **form.html**. Grundsätzlich handelt es sich dabei - wie der Dateiname schon suggeriert - um eine ganz normale HTML-Datei, bei der jedoch einige Besonderheiten zu beachten sind:

```
<table class="lightcyan">
    <tr>
        <td colspan="2" style="padding-top: 10px;">
            <textarea class="textarea" rows="7" id="details"
placeholder="Bemerkungen"></textarea>
        </td>
    </tr>
    <tr>
        <td colspan="2">
            <div class="label">Geometrietyp</div>
            <select id="data_geometry_type">
                <option selected="selected" value="Point">Punkt</option>
                <option value="LineString">Linienzug</option>
                <option value="Polygon">Polygon</option>
            </select>
        </td>
    </tr>
    <tr>
        <td colspan="2">
            <div class="label"><label for="data_position">Koordinaten (Längengrad,
Breitengrad)</label></div>
            <textarea class="textarea" rows="3" id="data_position"
placeholder="'ÜBERNEHMEN' fügt die aktuelle Position ein."></textarea>
        </td>
    </tr>
</table>
<input type="hidden" id="data_accuracy"></input>
<table style="margin-top: 10px;" class="initial-input-only">
    <tr>
        <td class="button-inside" style="text-align: left;">
```

```
            <ons-button onclick="fn.formHandler.clearForm()">Zurücksetzen</ons-button>
        </td>
        <td class="button-inside" style="text-align: center;">
            <ons-button onclick="fn.formHandler.takePhoto()">Foto</ons-button>
        </td>
        <td class="button-inside" style="text-align: right;">
            <ons-button onclick="fn.formHandler.possiblySave()">Speichern</ons-button>
        </td>
    </tr>
</table>
<table style="margin-top: 10px;" class="modify-data-only">
    <tr>
        <td class="button-inside" style="text-align: left;">
            <ons-button onclick="fn.formHandler.deleteById()">Löschen</ons-button>
        </td>
        <td class="button-inside" style="text-align: center;">
            <ons-button onclick="fn.formHandler.takePhoto()">Foto</ons-button>
        </td>
        <td class="button-inside" style="text-align: right;">
            <ons-button onclick="fn.formHandler.updateById()">Aktualisieren</ons-
button>
        </td>
    </tr>
    <input type="hidden" id="data_id" value="-1"></input>
</table>
<table>
    <tr><td id="photos"></td></tr>
</table>
```

Wichtig sind zunächst die IDs, die in zwei unterschiedliche Kategorien fallen: Solche, die fest vorgegeben sind und solche, die weitgehend frei gewählt werden können. Die fest vorgegebenen sind:

ID	Bedeutung
`data_geometry_type`	Hiermit wird festgelegt, welcher Geometrietyp erfasst werden soll. In diesem Fall ist auch vorgegeben, welche Werte verwendet werden können: • `value="Point"` Punkt • `value="LineString"` Linienzug • `value="Polygon"` Polygon (Vieleck)
`data_position`	Hierin befindet sich eine Liste von Punkten, die beim Speichern zu einem Punkt, einem Linienzug oder einem Polygon werden sollen.
`data_accuracy`	Hierin befindet sich die Information, wie genau die Positionsdaten laut GPS sind
`data_id`	Beim Bearbeiten bereits vorhandener Daten ist hierin die ID des in Bearbeitung befindlichen Datensatzes enthalten. Beim Erfassen eines neuen Datensatzes muss der Wert -1 sein.
`photos`	Hier werden neu aufgenommene Fotos zu einem Datensatz als Miniaturbild dargestellt.

Die IDs spielen noch eine weitere Rolle, zu der wir aber erst im Zusammenhang mit **form.json** kommen werden.

Kommen wir zunächst zu `ons-button`. Es handelt sich hierbei um eine Schaltfläche wie

`button` - Unterschiede gibt es hauptsächlich im Aussehen, nicht in der Funktion. `button` passt vom Aussehen her nicht zu den übrigen Schaltflächen, ansonsten könnte genauso gut auch diese Art Schaltfläche verwendet werden.

Betrachten wir jetzt **`class="initial-input-only"`** und **`class="modify-data-only"`**. Diese beiden CSS-Klassen geben an, ob der betreffende Teil des Formulars nur beim Neuerfassen eines Elements angezeigt werden soll bzw. nur beim Bearbeiten eines bereits vorhandenen Elements.

Beim Neuerfassen werden die Schaltflächen **Zurücksetzen**, **Foto** und **Speichern** angezeigt, beim Bearbeiten eines schon vorhandenen Datensatzes die Schaltflächen **Löschen**, **Foto** und **Aktualisieren**.

Sie sehen, dass die genannten Schaltflächen beim Betätigen bestimmte Funktionen aufrufen (beim Namen der Funktion ist jeweils weggelassen:

Funktionsname	**Funktionalität**
`fn.formhandler.`**`clearForm()`**	Setzt die Elemente des Formulars auf den jeweiligen Standardwert zurück
`fn.formhandler.`**`takePhoto()`**	Nimmt ein Foto auf, das dem aktuellen Datensatz zugeordnet wird.
`fn.formhandler.`**`possiblySave()`**	Speichern - aber nur sofern die Bedingungen an die erfassten Daten erfüllt sind
`fn.formhandler.`**`deleteById()`**	Den aktuellen Datensatz löschen
`fn.formhandler.`**`updateById()`**	Den aktuellen Datensatz mit den geänderten Daten aktualisieren.

Beachten Sie bitte auch das `class="lightcyan"` - diese Klasse unterlegt die Tabelle mit einem hellen Cyan/Türkis, was einer visuellen Strukturierung des Formulars dient - der Farbton ist darauf abgestellt zum restlichen Farbschema von gApp zu passen.

form.json - Zusatzinformationen zum Formular

form.json konfiguriert verschiedene Einstellungen rund um das Formular, betrachten wir zunächst die grundsätzliche Struktur der Datei:

```
{
  "tableName": "gappstandard",
  "user_kmls": [ ... ],
  "form": {
    "name": "Standardmaske",
    "description": "Erfassung allgemeiner Daten zu Punkten, Linienzügen und Polygonen",
    "baseUrl": "https://gcloud.benndorf.de/",
    "defaultGeometry": "Point",
    "dataSources": [ ... ],
    "marker": {
      "outputFormat": [ ... ]
    }
  }
}
```

tableName ist der Name der Datenbanktabelle, in der die Daten des Formulars

gespeichert werden. Auch wenn bei gApp stets nur ein Formular aktiv ist, können die Daten zu mehreren Formularen lokal gespeichert sein. `tableName` steuert, in welcher der Tabellen die Daten zu welchem Formular gespeichert werden.

user_kmls ist eine Liste von KML-Dateien. Wenn sich eine hier aufgelistete Datei im Unterverzeichnis **user_kmls** des Formularordners (z. B. des Ordners Standardformular) befindet, wird sie zusätzlich zu den Konfigurationsdateien auf das Mobilgerät übertragen. Sie kann dann als `featureLayer` bzw. `fixedFeatureLayer` verwendet werden.

Die Datei `user_kmls/0.kml` wird in der Datei **layers.json** als http://localhost:8080/user_kmls/0.kml angesprochen, analoges gilt selbstverständlich auch für Dateien mit anderen Namen.

form beschreibt das eigentliche Formular.

form.name ist der Name, der für das Formular angezeigt wird, hier *Standardmaske*.

form.description ist eine kurze Beschreibung des Formulars.

form.baseUrl gibt die Basisadresse an, auf die sich verschiedene verwendete Adressen beziehen. Die Adresse ist in aller Regel die Hauptseite des gCloud-Servers, in diesem Fall https://gcloud.benndorf.de/.

defaultGeometry gibt an, welcher Geometrietyp als Standardtyp verwendet wird.

dateSources gibt an, aus welchen Datenquellen die Spalten der Datenbanktabelle befüllt werden, dazu gleich mehr.

marker steuert die Darstellung der erfassten Daten in gApp. Aktuell steht nur **marker.outputformat** zur Verfügung. Hiermit wird angegeben, wie die erfassten Daten im Popup-Fenster zu einem Punkt, Linienzug oder Polygon angezeigt werden.

Kommen wir nun zu den Auslassungszeichen.

Bei ... (**user_kmls**) ist wenig zu sagen, es handelt sich um eine einfach Auflistung von Dateinamen.

```
  "user_kmls": [ "1.kml", "2.kml", "3.kml" ],
```

Die Formate bei **dataSources** und **outputFormat** sind verglichen hiermit deutlich komplexer, da sich andernfalls die notwendige Flexibilität nicht erreichen ließe.

dataSources

Betrachten wir die dataSources-Liste der Standardmaske:

```
"dataSources": [
  {
    "name": "data_position",
    "kind": "position",
    "type": "TEXT",
    "required": "Welche Koordinaten sollen verwendet werden?",
    "resetTo": ""
  },
```

```
  {
    "name": "data_accuracy",
    "kind": "accuracy_input",
    "type": "REAL",
    "required": true,
    "resetTo": ""
  },
  {
    "name": "date",
    "kind": "date",
    "type": "TEXT",
    "required": "Wann wurde erfasst?"
  },
  {
    "name": "details",
    "kind": "input",
    "type": "TEXT",
    "required": false,
    "resetTo": ""
  }
]
```

name gibt den Namen einer Tabellenspalte an, in der ein erfasster Wert gepeichert wird. Normalerweise handelt es sich um eine ID aus **form.html**. Ausnahme ist `date`. Wie wir gleich sehen werden, wird hierin der Zeitpunkt der Erfassung bzw. Änderung gespeichert.

kind gibt an, um was für eine Art von Datum es sich handelt.

- **input** steht für einen manuell eingegebenen Wert
- **date** steht für den Zeitpunkt der Erfassung bzw. Änderung
- **position** steht für die Positionsinformation (die nach Maßgabe von data_geometry_type in ein geometrisches Objekt im WKT-Format umgewandelt wird)
- **accuracy_input** gibt an, dass es sich um Genauigkeitsinformationen des GPS-Empfängers handelt

type ist der Datentyp der Datenbankspalte, in der der Wert gespeichert wird. Bei der Standardmaske kommen hier nur die Typen REAL (Fließkommazahl) und TEXT (Zeichenfolge) vor.

required gibt an, ob ein Datum vorhanden sein muss oder nicht. Es gibt drei mögliche Werte:

- **false** - der Wert muss nicht gesetzt sein
- **true** - der Wert muss gesetzt sein. Fehlt der Wert, wird eine Standardmeldung angezeigt.
- eine Zeichenfolge - der Wert muss gesetzt sein. Fehlt der Wert, wird die angegebene Meldung angezeigt.

resetTo - gibt an, welchen Wert das jeweilige Eingabeelement beim Betätigen der Schaltfläche **Zurücksetzen** zugewiesen bekommt. Wenn Sie diese Angabe weggelassen, wird die entsprechende Information beim Zurücksetzen nicht angetastet. Dies ist etwa **date** der Fall, da hier kein Element vorhanden ist, das zurückgesetzt werden könnte.

Die Option resetTo wegzulassen kann auch bei Eingabeelemente sinnvoll sein. Bei der Standardmaske werden die Bemerkungen beim Zurücksetzen des Formulars gelöscht. Es kann sein, dass Sie das nicht wollen. In diesem Fall lassen Sie einfach unter **"details"** das Element **resetTo** weg.

outputFormat

Kommen wir nun zum **outputFormat**, das die Anzeige der Informationen in den Popups zu Punkten, Linienzügen und Polygonen vorgibt. Zunächst einmal handelt es sich dabei um eine Liste von Elementen, die in der angegebenen Reihenfolge hintereinander für die Anzeige zusammengesetzt werden.

```
"marker": {
  "outputFormat": [
    [
      "!",
      "<p>",
      "<span style='font-weight: bold;'>Geometrie</span><br />",
      "?data_position",
      "</p>"
    ],
    [
      "!",
      "<p>",
      "<span style='font-weight: bold;'>Datum und Uhrzeit<span><br />",
      "?date",
      "</p>"
    ],
    [
      "?",
      "<p>",
      "<span style='font-weight: bold;'>Bemerkungen</span><br />",
      "?details",
      "</p>"
    ],
    [
      "!",
      "<button ontouchstart='parent.window.fn.editData(event, \"",
      "?id",
      "\");'><span style='font-weight: bold;'>",
      "Bearbeiten / Löschen",
      "</span></button>"
    ]
  ]
}
```

Im vorliegenden Fall dient das erste Element der Anzeige der erfassten Geometrie, das zweite der Anzeige des Erfassungs- bzw. Änderungsdatums, das dritte der Anzeige der erfassten Bemerkungen und das letzte der Anzeige der Schaltfläche, mit der Sie zum Bearbeiten bzw. Löschen des angezeigten Elements gelangen.

Unabhängig von dieser Konfiguration wird in der ersten Zeile stets *Erfasstes Datum **id*** ausgegeben, dabei ist **id** die nummerische Kennung, unter der das Element in der Datenbanktabelle aufgeführt ist.

Schauen wir uns nun die einzelnen Elemente nun genauer an.

"!" / "?" - Sie sehen, dass in der ersten Zeile entweder `"!"` oder `"?"` steht. Diese Angabe steuert, ob das jeweilige Element unbedingt (`"!"`) oder nur dann angezeigt werden soll, wenn ein Wert angegeben ist. Im konkreten Fall wird das dritte Element nur bei Bedarf ausgegeben, alle anderen Elemente werden in jedem Fall angezeigt.

"?*name*" - In den hervorgehobenen Zeilen treffen wir auf Bekannte aus **dataSources** (`data_position`, `date`, `details`) und einen Neuling (`id`), denen jeweils ein Fragezeichen vorangestellt ist.

Das Fragezeichen deutet an, dass wir an dieser Stelle wissen wollen, wie der jeweilige Wert lautet. Und tatsächlich werden diese Zeichenfolgen durch den jeweiligen Wert ersetzt. Um das Format nicht unnötig zu komplizieren, darf nichts außer dem Fragezeichen und dem Namen des interessierenden Datums in der Zeichenfolge stehen.

Wie Sie vielleicht schon richtig vermutet haben, werden die einzelnen angegebenen Zeichenfolgen zu einer einzigen zusammengesetzt. Daher stellt die gerade dargelegte Vorgabe keine Einschränkung dar und hat zudem den Vorteil, dass die Auswahl des Werts in der Konfigurationsdatei leichter aufzufinden ist, als wenn sie sich in einer langen Zeichenfolge versteckt.

Wie schon gesagt: Neben den alten Bekannten gibt es einen Neuling: `id`. Ein gänzlich Unbekannter ist er allerdings nicht, denn der zugehörige Wert wird zwingend angezeigt. Es ist nämlich die schon bei *Erfasstes Datum* ***id*** erwähnte nummerische Kennung, unter der das angezeigte Element in der Datenbanktabelle aufgeführt ist.

Abgesehen von den hervorgehobenen Zeilen enthalten die einzelnen Elemente nur HTML-Code, der zusammengesetzt wird. Nehmen wir an, dass die folgenden Werte vorhanden sind:

Spalte	**Wert**
`data_position`	POINT(8.4777 49.7652)
`date`	2026-08-13T05:45:08.015
`details`	Hier wurde angeblich ein Schatz versenkt
`id`	4

Dann wird die Vorgabe zusammengesetzt zu:

```
<p><span style='font-weight: bold;'>Geometrie</span><br />POINT(8.4777 49.7652)</p>
<p><span style='font-weight: bold;'>Datum und Uhrzeit<span><br />2026-08-
13T05:45:08.015</p>
<p><span style='font-weight: bold;'>Bemerkungen</span><br />Hier wurde angeblich ein
Schatz versenkt</p>
<button ontouchstart='parent.window.fn.editData(event, "4");'><span style='font-weight:
bold;'>Bearbeiten / Löschen</span></button>
```

Bitte beachten Sie, dass sie doppelte Anführungszeichen, die im HTML vorkommen sollen, als **\"** eingeben müssen.

Wie schon bei **form.html** wird auch hier eine Funktion verwendet. Diese wird aufgerufen, um das Bearbeiten bzw. Löschen das angezeigten Datums zu auszulösen. Wichtig ist,

dass **`parent.window.fn.editData(event, id)`** mit zwei Parametern aufgerufen werden muss.

Damit diese Funktion das gewünschte Verhalten zeigt, muss als erster Parameter `event` (**ohne Anführungszeichen!**) angegeben werden, als zweiter die nummerische Kennung, die Sie mit Hilfe von **`?id`** erhalten.

Beispielmaske

Den größten Teil der Konfiguration von gApp kennen Sie nun bereits. Was **client.json** und **layers.json** angeht gibt es auch nichts mehr zu sagen. Bei **form.html** und **form.json** jedoch gibt es jedoch noch einige bislang nicht erwähnte zusätzliche Möglichkeiten.

form.html

Betrachten wir zunächst den folgenden Auszug aus **from.html**:

```
<table class="lightcyan">
  <tr>
    <td>
      <div class="label">Zustand des Baums</div>
      <select id="tree_state">
        <option value="" selected="selected" disabled="disabled">- bitte wählen -
</option>
        <option value="0">umgestürzt</option>
        <option value="1">gefällt</option>
        <option value="2">Schädlingsbefall</option>
        <option value="3">schlecht</option>
        <option value="4">mittel</option>
        <option value="5">gut</option>
        <option value="6">sehr gut</option>
        <option value="7">exzellent</option>
      </select>
    </td>
    <td>
      <div class="label">Umsturzgefahr</div>
      <input type="checkbox" id="may_fall" /> besteht
    </td>
  </tr>

  <tr>
    <td colspan="2">
      <div class="label">Fällen?</div>
      <input type="radio" name="log_tree" value="ja" />ja
      <input type="radio" id="log_tree_default" name="log_tree" value="nein"
checked="checked" />nein
      <input type="radio" name="log_tree" value="unklar">prüfen
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <div class="label"><label for="data_accuracy">Durchschnittliche Genauigkeit (in
m)</label></div>
      <div>
        <input class="textarea" type="text" id="data_accuracy" placeholder="Wird
automatisch berechnet."></input>
      </div>
    </td>
  </tr>
</table>
```

Zunächst einmal sehen Sie, dass Ihnen als Eingabeelement eine checkbox zur Verfügung steht, also ein Kästchen, das mit einem Häkchen für „trifft zu“ versehen werden kann.

Schauen wir uns nun zuerst `data_accuracy` an. In diesem Element wird die Genauigkeit der Positionsmessungen laut GPS-Empfänger angezeigt, genauer gesagt deren Mittelwert. Der Zugriff auf diesen Wert kann nützlich sein, aber beachten Sie bitte, dass der vom GPS-Empfänger gelieferte Wert nur zu einer qualitiativen Einschätzung der Genauigkeit der Messungen geeignet ist und nicht als quantitatives Maß herangezogen werden sollte.

Sie erinnern sich vermutlich, dass Sie in **form.json** angeben können, auf welchen Standardwert ein Eingabeelement zurückgesetzt werden soll, ohne dass hierfür etwas in **form.html** stehen soll. Nur bei Radiobuttons ist dies technisch unmöglich.

Wirklich benötigt wird dieses Eingabeelement nicht, da es immer durch ein Auswahlmenü oder ein zu setzendes Häkchen ersetzt werden kann. Sollten Sie es dennoch verwenden wollen, ist dies möglich aber mit ein wenig Aufwand verbunden.

Gehen wir von einer Gruppe von Radiobuttons mit `name="name_der_gruppe"` aus. Bei einem (und nur einem) dieser Buttons müssen sie zusätzlich eine id angeben, und zwar `id="name_der_gruppe_default"` Damit ist die entsprechende Auswahl als Standard markiert. Sinnvollerweise ist dies die Auswahlmöglichkeit, die auch als „checked“ markiert ist. Sie müssen dann nur noch in forms.json angeben, **ob** das Element auf diesen Standardwert zurückgesetzt werden soll oder nicht.

form.json

Der folgende Auszug aus form.json illustriert zweierlei:

```
{
  "form": {
    "dataSources": [
      {
        "name": "tree_state",
        "kind": "input",
        "type": "INTEGER",
        "required": "Wie ist der Zustand des Baumes?",
        "resetTo": ""
      },
      {
        "name": "log_tree",
        "kind": "input",
        "type": "TEXT",
        "required": "Soll gefällt werden?",
        "resetToDefault": true
      },
      {
        "name": "may_fall",
        "kind": "input",
        "type": "BOOLEAN",
        "required": true,
        "resetTo": false
      }
    ],
    "marker": {}
  }
}
```

Zunächst einmal illustriert er, wie Sie steuern, ob der Wert bei Radiobuttons auf den Standardwert zurückgesetzt werden soll oder nicht: Sie verwenden **resetToDefault** anstelle von **resetTo** und geben **true** an, falls der Wert zurückgesetzt werden soll oder **false**, wenn nicht. Sie können resetToDefault auch weglassen, dann wird der Wert ebenfalls nicht zurückgesetzt.

Unschwer zu erkennen ist auch, dass Sie Datenbankspalten vom Typ **INTEGER** oder **BOOLEAN** definieren können.

Erinnern Sie sich noch, dass Sie Werte mit Hilfe von ?wert auslesen können und dass außer einem Fragezeichen und dem Namen des Werts nichts angegeben werden kann? Das stimmt auch - aber nicht immer. Es stimmt, sofern der Wert unverändert ausgegeben werden soll.

Nun gibt es aber Fälle, in denen dies sehr unpraktisch sein kann. Schauen Sie sich noch einmal unter **form.html** an, wie dort **tree_state** aussieht. Sie erkenne, dass im Formular zwar Beschreibungen der Auswahlmöglichkeiten stehen, diese aber als Zahlenwerte kodiert werden.

Natürlich ist es möglich, diese Zahlen auswendig zu lernen, wünschenswert ist aber eine Möglichkeit, einen entsprechenden Text auszugeben. Auch wenn ein Häkchen gesetzt werden kann ist es sinnvoll, statt 0 (kein Häkchen gesetzt) und 1 (Häkchen gesetzt) einen sinnvollen Text auszugeben. Wie das funktioniert zeigt folgender Auszug aus der **form.json**:

```
{
  "form": {
    "marker": {
      "outputFormat": [
        [
          "!",
          "<p style='color: #900;'>",
          "<span style='font-weight: bold;'>Zustand des Baums</span><br />",
          "?[0:umgestürzt|1:gefällt|2:Schädlingsbefall|3:schlecht|4:mittel|5:gut|6:sehr
gut|7:exzellent]tree_state",
          "</p>"
        ],
        [
          "!",
          "<p>",
          "<span style='font-weight: bold;'>Umsturzgefahr?</span><br />",
          "?[0:ja|1:nein]may_fall",
          "</p>"
        ],
        [
          "!",
          "<p>",
          "<span style='font-weight: bold'>Alternative Darstellung!</span><br />",
          "Umsturzgefahr ",
          "?[0:besteht|1:besteht nicht]may_fall",
          "</p>"
        ]
      ]
    }
  }
}
```

Zwischen dem Fragezeichen und dem Namen des Werts wird eine Liste in eckigen Klammern eingefügt. Die einzelnen Elemente der Liste werden durch einen senkrechten Strich getrennt (auf einer DIN-konformen Tastatur finden Sie das Zeichen unten links; um es zu erhalten muss - genauso wie bei **@** und **€** - gleichzeitig die AltGr-Taste betätigt werden).

Die einzelnen Elemente der Liste enthalten den Wert, der umgewandelt werden soll, einen Doppelpunkt und dann den anzuzeigenden Text. Der Text kann alle Zeichen mit Ausnahme des senkrechten Strichs **|** und der schließenden eckigen Klammer **]** enthalten. Doppelpunkte sind möglich.

Die folgende Tabelle erläutern, wie die Werte dargestellt werden:

Wert	**Darstellung**
0	umgestürzt
1	gefällt
2	Schädlingsbefall
3	schlecht
4	mittel
5	gut
6	sehr gut
7	exzellent

Zum Schluss muss noch erwähnt werden, dass die Speicherung der Daten in einer sqLite-Datenbank erfolgt. Daher werden Daten vom Typ **BOOLEAN** nicht als True und False sondern als 0 und 1 kodiert. Das erklärt, wieso für die Darstellung der Daten der Checkbox **may_fall** ebenfalls Zahlenwerte angegeben sind.