online Übungssystem

Im- & Export von Aufgaben(heft)-Einstellungen

Aufbau der JSON-Konfigurationsdateien

Autor:
Immo Schulz-Gerlach, ZMI,
FeU-Softwaretechnik

Version:
1.1 (09. Februar 2021)

FernUniversität in Hagen

Inhaltsverzeichnis

Einleitung

Normalerweise konfigurieren Sie die Inhalte Ihrer Kursumgebungen im Online-Übungssystem direkt online über die Webschnittstelle, d.h. Sie loggen sich im Betreuerzugang Ihres Übungssystem-Kurses ein, können dort bestehende Aufgabenhefte einsehen, neue Hefte hinzufügen, Einstellungen zu Heften ändern (z.B. Bearbeitungszeiträume festlegen, Benotung aktivieren, dazu ggf. ein Notenschema festlegen u.v.m.), sowie natürlich Aufgaben erzeugen und bearbeiten.

Aufgabeneinrichtungen bestehen zum Teil aus HTML-Dateien, sog. *Standard-Kursressourcen*, wie z.B. `aufgabe1.1.html` für das Aufgabenformular zur ersten Aufgabe des ersten Hefts. Wenn Sie die *Aufgabenerstellungsassistenten* verwenden, werden diese Standard-Kursressourcen für Sie automatisch erzeugt, bei der *fortgeschrittenen Aufgabenerstellung* erstellen Sie diese selbst (oder passen von Assistenten erzeugten Ressourcen nachträglich an). Letzteres kann offline passieren:

- Im Betreuer-Zugang unter »Kursressourcen« können Sie insbesondere ein ZIP-File mit allen existierenden Kursressourcen zum Kurs (einschließlich der Standard-Kursressourcen) herunterladen.
- Dort finden Sie auch eine Upload-Möglichkeit, die zwar in erster Linie zum Hochladen von *Nicht-Standard-Kursressourcen* (also z.B. Bildern, die Sie in Ihren Aufgabenseiten verwenden möchten oder PDFs mit Aufgabenstellungen etc.) dient, über die Sie aber auch offline bearbeitete (oder neu erstellte) Standard-Kursressourcen hochladen können.
- Falls Sie neue Standard-Kursressourcen (Aufgabenformulare etc.) zu noch nicht existierenden Aufgaben (in ggf. noch nicht existierenden Aufgabenheften) hochladen, werden die zugehörigen Aufgaben (und ggf. Aufgabenhefte) neu erzeugt, müssen aber im Anschluss noch konfiguriert werden.

Nun bestehen Aufgabeneinrichtungen nicht *nur* aus diesen Standard-Kursressourcen: In der fortgeschrittenen Aufgabenerstellung finden Sie auch noch einige Eingabefelder wie den Aufgabennamen, erreichbare Punktzahlen pro Teilaufgabe sowie Möglichkeiten zum Einbinden und Konfigurieren von (Vor-)Korrekturmodulen. (Wenn Sie die Aufgabenerstellungsassistenten verwenden, werden auch diese Einstellungen vom Assistenten erzeugt und müssen nicht direkt von Ihnen bearbeitet werden.)

Um auch solche Aufgaben-Einstellungen ebenso wie die Aufgabenheft-Einstellungen (s.o.) offline bearbeiten und später ins Übungssystem importieren zu können, unterstützt das Online-Übungssystem nun das Hochladen einer entsprechenden Konfigurationsdatei. Durch den Upload einer einzigen Konfigurationsdatei können Sie also alle Aufgabenhefte (samt Notenschemata etc.) und Aufgaben in einem Schritt konfigurieren. Das Hochladen erfolgt ebenfalls über die Kursressourcen-Upload-Funktion: Wird dort im Upload eine solche Konfigurationsdatei (an ihrem Dateinamen) erkannt, so wird diese nicht als Kursressource gespeichert, sondern ihre Einstellungen werden in die bestehenden Aufgabenhefte und Aufgaben importiert.

Und auch der Download des ZIP-Files mit allen Kursressourcen enthält (seit Ende Juli 2019) eine solche Konfigurationsdatei, die den aktuellen Stand aller Aufgabenheft- und Aufgabeneinstellungen zum Downloadzeitpunkt umfasst.

Auf diese Weise ist z.B. eine *Migration* von Aufgaben von einem Kurs in einen anderen, neuen Kurs möglich: Laden Sie vom Quellkurs das Kursressourcen-ZIP herunter und laden dies in einem neuen (leeren) Kurs wieder hoch, so werden daraus zunächst alle Kursressourcen importiert, zu allen gefundenen Standardkursressourcen die entsprechend benötigten Aufgabenhefte und Aufgaben abgeleitet und erzeugt und diese anschließend mit den Einstellungen aus der Konfigurationsdatei konfiguriert.

Es kann auch sinnvoll sein, vor Änderungen an Ihren Aufgaben ein solches Kursressourcen-ZIP *als Backup* herunterzuladen.

Aber auch, falls Sie manuell *offline Konfigurationsänderungen* vornehmen möchten, können Sie dazu das Kursressourcen-ZIP herunterladen, entpacken, die Konfigurationsdatei daraus entnehmen und diese als Vorlage für Ihre Änderungen verwenden, sie also bearbeiten und nach der Bearbeitung wieder hochladen.

Dateiname und Kurs-/Semesterbezug

Damit beim Kursressourcen-Upload eine hochgeladene Datei als Konfigurationsdatei erkannt und ausgewertet (statt als Kursressource gespeichert) wird, muss sie einen bestimmten Namen haben, der wie folgt aufgebaut ist:
`config-12345-WS20.json`

Dabei ist `12345` durch die (fünfstellige) Kursnummer zu ersetzen, und `WS20` durch die jeweilige Semesterkennung, also `SS` oder `WS` gefolgt von zweistelliger Jahreszahl. (Die Groß-/Kleinschreibung von hochgeladenen Dateien wird dabei ignoriert.)

Beachten Sie insbesondere beim Upload, dass in der Regel der im Dateinamen genannte Kurs ebenso wie die Semesterkennung „zum Zielkurs passen" müssen. Das verhindert insbesondere, dass Sie z.B. eine Konfigurationsdatei mit Bearbeitungsterminen für ein geplantes Semester versehentlich in die Kursumgebung fürs Vorsemester hochladen und dort rückwirkend alle Termine verstellen. Lediglich wenn der Zielkurs, in den Sie hochladen, noch „leer" ist (noch keine Aufgaben enthält), wird eine hochgeladene Konfigurationsdatei auch dann verarbeitet, wenn ihr Name nicht zum Kurs passt. Dabei kann erstens kein Schaden entstehen (weil ja im leeren Kurs noch keine Einstellungen existieren, die dadurch überschrieben würden), und zweitens ermöglicht das den oben beschriebenen einfachen Weg des Kopierens von Aufgaben eines Kurses in eine neue leere Kursumgebung (ohne, dass erst das ZIP entpackt und die Konfigurationsdatei darin passend zum Zielkurs umbenannt werden muss)[1].

Dateiformat und -Umfang

Als *Dateiformat* für die Konfigurationsdatei kommt das JSON-Format <https://www.json.org> (JavaScript Object Notation) zum Einsatz, da es relativ gut manuell in Texteditoren les- und editierbar ist. Als Datei-Encoding ist (wie bei JSON üblich) `UTF-8` zu verwenden. Der genaue Aufbau (des Inhalts) der JSON-Dateien wird im nachfolgenden Kapitel dokumentiert.

Der *Umfang* einer *hochzuladenden* Datei ist variabel. Das soll heißen: Eine Datei gilt nicht als unvollständig oder defekt, wenn mögliche Angaben darin fehlen. Vielmehr werden beim Upload eben alle in der Datei vorhandenen Einstellungen importiert, und alle Einstellungen im Kurs, zu denen sich keine Einträge in einer hochgeladenen JSON-Datei finden, bleiben einfach unverändert (da werden also auch keine Einstellungen gelöscht).
So können Sie z.B. eine Konfigurationsdatei ausschließlich mit den Bearbeitungsterminen aller Aufgabenhefte für ein Zielsemester füllen und hochladen. Damit werden dann eben genau diese Termine im Kurs eingetragen und alle anderen Einstellungen unangetastet gelassen. Oder wenn Ihr Kurs z.B. zwar vier Aufgabenhefte enthält, Ihre JSON-Datei aber nur Einstellungen zu den Heften Nr. 1 und 3 enthält, dann werden eben diese Hefte dadurch bearbeitet und die Hefte 2 und 4 bleiben unverändert.

Die im *Download* aller Kursressourcen (als ZIP) enthaltene JSON-Datei ist dagegen i.d.R. „vollständig", enthält also alle im JSON-Format abbildbaren Einstellungen der im Kurs vorliegenden Hefte und Auf-

gaben. Ungenutzte Einstellungen werden dabei allerdings nicht exportiert; für ein Aufgabenheft, zu dem keine Benotung eingestellt ist, wird im JSON z.B. nur notiert, dass die Benotung deaktiviert ist, es wird aber kein Notenschema mit exportiert – auch kein leeres (die exportierte Datei wäre in diesem Zustand also nicht als Vorlage für die Offline-Erstellung eines Notenschemas geeignet).

Aufbau der JSON-Datei

Root-Object mit `aufgabenhefte`-Property

Das JSON-Dokument besteht zunächst aus einem JSON-Objekt, beginnt also mit `{`, endet mit `}`, und dazwischen können Einstellungen stehen. Derzeit unterstützt dieses Format ausschließlich den Im- und Export von Aufgabenheften (mit Aufgaben, Notenschema etc. als Kindelementen). Dazu ist in dem Root-Objekt eine Property mit Key `"aufgabenhefte"` (als derzeit einzige Property[2]) einzufügen. Der Value (Wert) dieser Property ist eine Aufzählung von Aufgabenheften, für die Sie Einstellungen in der Datei sammeln möchten. Diese kann wahlweise als JSON-Array angegeben werden (dann bezieht sich das erste Array-Element auf Aufgabenheft Nr. 1, das zweite auf Heft 2 etc.) oder als Objekt mit Aufgabenheft-Nummern als Keys. Der Value dazu ist in jedem Fall wieder ein JSON-Object (also eine mit Mengenklammern eingefasste Aufzählung von Properties der Form `"key": value`) mit Einstellungen zum Aufgabenheft.

Beispiel 1.1 mit JSON-Array:

```
{"aufgabenhefte": [
    {
        "name": "Mein erstes Heft",
        "bearbeitungsbeginn": "01.04.2018, 00:00",
        "bearbeitungsende": "14.04.2018, 24:00"
    },
    {   "name": "Mein zweites Heft",
        "bearbeitungsbeginn": "07.04.2018, 00:00",
        "bearbeitungsende": "21.04.2018, 24:00"
    }
]}
```

Beispiel 1.2 mit JSON-Object:

```
{"aufgabenhefte": {
    "1": {
        "name": "Mein erstes Heft",
        "bearbeitungsbeginn": "01.04.2018, 00:00",
        "bearbeitungsende": "14.04.2018, 24:00"
    },
    "2": {
        "name": "Mein zweites Heft",
        "bearbeitungsbeginn": "07.04.2018, 00:00",
        "bearbeitungsende": "21.04.2018, 24:00"
    }
}}
```

Im Zweifel empfehlen wir die zweite Variante (Objects mit vorangestellter Aufgabenheftnummer), da sie folgende Vorteile aufweist:

1. Falls Sie nicht zu *allen* Heften Einstellungen per JSON importieren möchten, können Sie auch *nur bestimmte* Hefte aufzählen, z.B. nur Hefte Nr. 1 und 3, aber ohne Einstellungen zu Heft 2. Beim Array können Sie dagegen keine Lücken lassen.
2. Bei umfangreicheren Dateien ist es übersichtlicher, die Heftnummer direkt vor dem Objekt stehen zu sehen und nicht die Array-Elemente von oben abzählen zu müssen.
3. Die Reihenfolge der Nennung der Aufgabenhefte ist beim Objekt übrigens ebenfalls egal, Sie können – wenn Sie das im konkreten Fall bevorzugen sollten – z.B. die Aufzählung mit Heft Nr. 4 beginnen, gefolgt von Heft Nr. 1 etc.

Die drei Properties `name`, `bearbeitungsbeginn` und `bearbeitungsende` aus obigen Beispiellistungs sind nur eine beispielhafte Auswahl möglicher Aufgabenheft-Properties. Der nächste Abschnitt erläutert sämtliche möglichen Aufgabenheft-Properties.

Aufgabenheft-Properties

Für jedes Aufgabenheft-Objekt (siehe vorigen Abschnitt) können die Aufgabenheft-Einstellungen über die folgenden Properties angegeben werden. Zur Bedeutung der einzelnen Einstellungen lesen Sie bitte die Online-Hilfe zu den entsprechenden Eingabefeldern in der Benutzeroberfläche (Aufgabenheftverwaltung).

Aufgabenheft-Einstellungen für alle Kurse

Key	Value-Type	Inhalt / Format / Werte
`name`	String	Optionaler Name fürs Aufgabenheft. Leerstring (`""`) für unbenanntes Heft: Anders als wenn Sie diese Property ganz weglassen, wird bei Angabe von `"name":""` dafür gesorgt, dass das Heft anschließend unbenannt ist, ein ggf. im System noch vorhandener Name also gelöscht wird.
`bearbeitungsbeginn`	String	Datum und Uhrzeit des Bearbeitungsbeginns in der Form `tt.mm.jjjj, hh:mm`
`bearbeitungsende`	String	Datum und Uhrzeit des Bearbeitungsendes in der Form `tt.mm.jjjj, hh:mm`
`korrekturfreigabe`	String	entweder `"VOR_BEARBEITUNGSENDE_ERLAUBT"` oder `"ERST_NACH_BEARBEITUNGSENDE"`
`autokorrektur-autofreigabe`	String	entweder `"KEINE"`, `"NUR_AUTO"` oder `"AUCH_AUTOHAND"`
`aufgaben`	Object oder Array	siehe `aufgaben`-Property von Aufgabenheft

Aufgabenheft-Einstellungen für spezielle Kurse

Die folgenden Einstellungen stehen nicht für alle Kurse im Online-Übungssystem zur Verfügung, sondern nur in speziellen Sonderfällen. (Prüfen Sie im Zweifel in Ihrem Kurs, ob die entsprechenden Einstellungen in der Benutzeroberfläche, also der Aufgabenheftverwaltung, von Ihnen gesetzt bzw. eingesehen werden können. Einstellungen, die Sie online nicht einstellen können, können Sie auch nicht per JSON konfigurieren, die JSON-Properties werden dann ignoriert.)

Key	Value-Type	Inhalt / Format / Werte
`nur-individuelle-termine`	Boolean	Nur für Prüfungen: Wenn `true`, können Sie zum Aufgabenheft keine Bearbeitungsbeginn- und -Ende-Termine mehr global festlegen. Vielmehr können nur Studierende die Aufgaben bearbeiten, die zur zugeordneten Prüfung (siehe `pruefungsnr`) angemeldet sind und zu deren Anmeldung ein individueller Bearbeitungszeitraum in der Prüfungsverwaltung festgelegt wurde.
`bearbeitungsende-vz`	String	Nur für Prüfungen, für die zwei verschiedene Einsendefristen festgelegt werden können: Dies ist dann der Einsendeschluss für Vollzeitstudenten, `bearbeitungsende` (s.o.) legt dann den Termin für alle anderen Hörerstatus fest. Format wie bei `bearbeitungsende`.
`bestehensgrenze`	Integer oder String	Nur für Kurse mit Punktzahl-Export zu Einsendearbeiten oder Selbstkontrollarbeiten nach SLO: Eine Zahl gibt eine Grenze in Prozentpunkten an, ab der eine Einsendearbeit als bestanden gilt. Ein Leerstring-Literal (`""`) wählt explizit die Standardeinstellung (derzeit 50%).
`pruefungsnr`	Integer	Nur für Prüfungen: Prüfungsnummer. Zahl 0 für explizite Angabe, dass *keine* Prüfungsnummer zugeordnet werden soll (und eine ggf. vorhandene zu löschen ist).
`pruefungs-schluessel`	Integer	Nur für bestimmte Prüfungen, zu denen keine Anmeldungen erwartet werden. Zahl 0 löscht explizit bestehende Schlüssel.
`prueferid`	String	Nur für bestimmte Prüfungen (wenn dem Kurs-Veranstalter nicht bereits eine Prüfer-ID fest zugeordnet ist): Prüfer-ID / Prüferkürzel für das Aufgabenheft

Benotungseinstellungen

Für die meisten Kurse lässt sich optional eine Benotung zum Aufgabenheft einstellen:

Key	Value-Type	Inhalt / Format / Werte
`benotung`	String	Nur für Kurse, die eine Benotung unterstützen (Regelfall): Entweder `"KEINE"`, `"PRO_AUFGABE"` oder `"PER_SCHEMA"`
`notenspiegel`	Boolean	Falls unter `benotung` *nicht* ***`KEINE`*** eingestellt ist, steuern Sie hiermit, ob Studenten nach Korrekturfreigabe den Notenspiegel einsehen dürfen (`true`) oder nicht (`false`).
`normalisierung`	Object oder Array	Nur falls `"benotung":"PER_SCHEMA"` eingestellt ist: Notenschema-Normalisierungseinstellungen, s.u.
`notenschema`	Object	Nur falls `"benotung":"PER_SCHEMA"` eingestellt ist: Notenschema, s.u.

Normalisierung-Properties

Wie in der Online-Maske zur Erfassung eines Notenschemas auch, haben Sie hier folgende Normalisierungsmöglichkeiten zur Auswahl:

1. Angabe einer Geraden-Funktion per Steigung und/oder Bonuspunkten (Y-Achsverschiebung),

wobei Sie letztere als Bonus-Rohpunkte oder Bonus-Normalpunkte angeben können (Normalsierungs-Variante aus dem Lotse-System),

2. Angabe einer Normalisierungsfunktion durch einzelne Kontrollpunkte,
3. Deaktivieren der Normalisierung (so dass das Notenschema direkt Rohpunkte auf Noten abbildet).

Details zur Normalisierung werden an dieser Stelle nicht erläutert, lesen Sie im Zweifel die Online-Hilfe im Notenschema-Editor der Aufgabenheftverwaltung. (Wenn Sie die Einstellungen per JSON-Import vornehmen möchten, gehen wir davon aus, dass Sie deren Bedeutung bereits kennen.)

ad 1. (Lotse-Normalisierung)

Im ersten Fall muss der Value der `normalisierung`-Property ein JSON-Object sein, das wiederum selbst folgende Properties haben darf:

Key	Value-Type	Inhalt / Format / Werte
`bonusNormalpunkte`	Integer	Punktzahl, die nach Multiplikation der Rohpunkte mit der Steigung zu addieren ist.
`bonusRohpunkte`	Integer	Punktzahl, die zu den Rohpunkten vor Multiplikation mit der Steigung zu addieren ist.
`steigung`	Fließkommazahl oder String	Faktor, mit dem die Rohpunkte (ggf. nach Addition von `bonusRohpunkte`) zu multiplizieren sind. Oder Leerstring-Literal (`""`) für die explizite Festlegung, dass keine Steigung konfiguriert werden soll (also die Steigung der Prozentpunkte-Funktion zu verwenden ist). (Wenn Sie die Property `steigung` gar nicht angeben, bleibt beim JSON-Import die Steigungs-Einstellung unverändert, während bei Angabe von `""` eine ggf. gespeicherte Steigung gelöscht wird.)
`rohpunkte`	Boolean	`false` (Die Angabe dieser Property ist optional, aber *wenn* sie angegeben wird, ist ihr Wert auf `false` einzustellen, um die Normalisierung zu aktivieren. Der Wert `true` deaktiviert die Normalisierung, siehe *ad 3.*.))

Beispiel 2.1 (Lotse-Normalisierung):

```
{"aufgabenhefte": {
    …
    "2": {
        …
        "benotung": "PER_SCHEMA",
        "notenspiegel": true,
        "normalisierung": {
            "bonusNormalpunkte": -10,
            "bonusRohpunkte": 0,
            "steigung": 7.333333333333333
        },
        "notenschema": …
    }
}}
```

ad 2. (Kontrollpunkt-Normalisierung)

Für eine Kontrollpunkt-Normalisierung muss der Value der `normalisierung`-Property dagegen ein

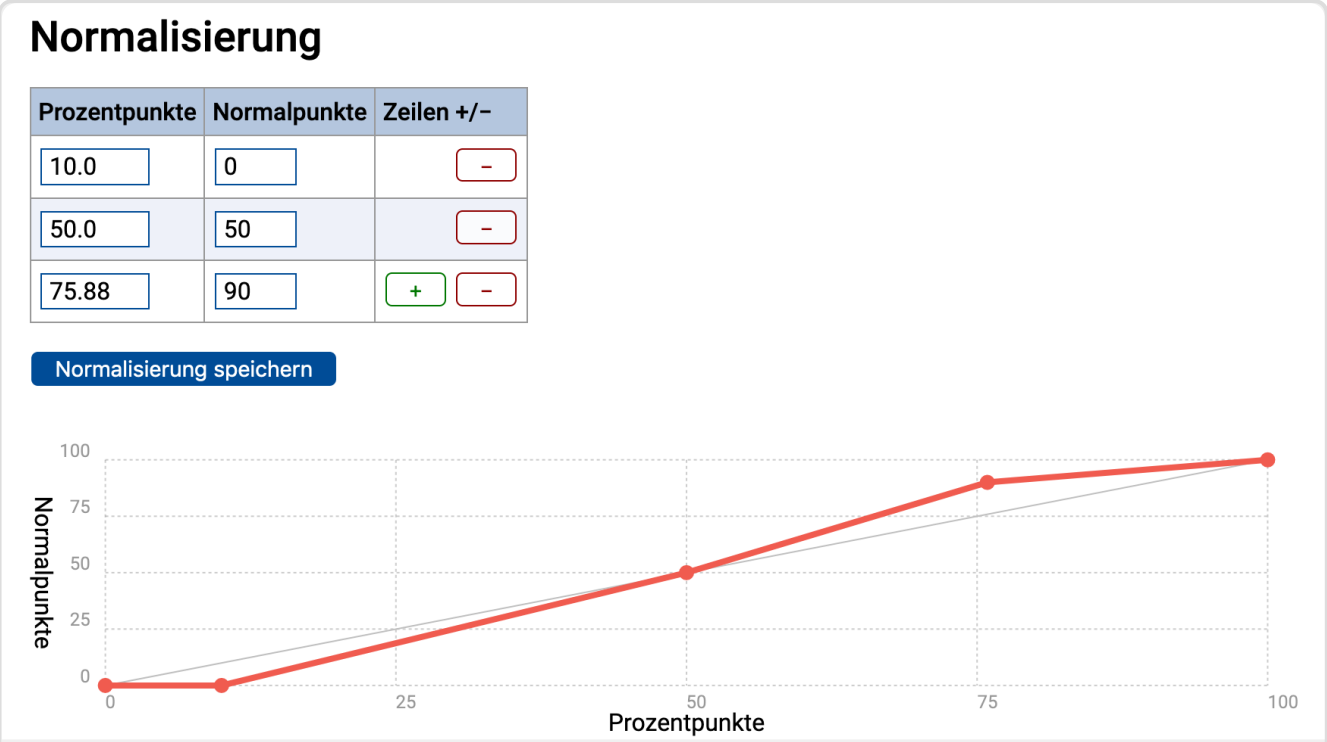
JSON-Array mit den einzelnen Kontrollpunkten sein. Jeder Eintrag dieses Arrays, also jeder Kontrollpunkt, ist wiederum als JSON-Array mit genau zwei Elementen anzugeben, von denen das erste Element die Prozentpunktzahl ist (als Fließkommazahl), der eine bestimmte Normalpunktzahl (Ganzzahl/Integer) zuzuordnen ist. Diese Normalpunktzahl wird als zweiter Wert ins Kontrollpunkt-Array geschrieben.

Beispiel 2.2 (Kontrollpunkt-Normalisierung):

Das folgende Beispiel legt eine Normalisierung über die folgenden drei Kontrollpunkte (10; 0), (50; 50) und (75,88; 90) fest:

```
{"aufgabenhefte": {
    …
    "3": {
        …
        "benotung": "PER_SCHEMA",
        "notenspiegel": false,
        "normalisierung": [
            [10, 0],
            [50, 50],
            [75.88, 90]
        ],
        "notenschema": …
    }
}}
```

Eine entsprechende Normalisierungseinstellung (z.B. nach Import des obigen Fragments) sähe in der Weboberfläche wie folgt aus:



Screenshot zu obiger Beispiel-Kontrollpunkt-Normalisierung

ad 3. (Normalisierung deaktivieren)

In diesem Fall muss der Wert der `normalisierung`-Property ein JSON-Object sein, das seinerseits

die (oben unter *ad 1.* bereits eingeführte) Boolean-Property `rohpunkte` auf `true` setzt:

```
"normalisierung": {"rohpunkte": true}
```

Standard-Normalisierung (Prozentpunkte auf Noten abbilden)

Die Standardeinstellung für Notenschema-Normalisierung liegt vor, wenn im Aufgabenheft *weder* Bonuspunkte oder Steigung *noch* Normalisierungskontrollpunkte angegeben wurden *noch* die Normalisierung explizit abgeschaltet wurde. In dem Fall berechnet die Normalisierungsfunktion einfach die Prozentpunkte, d.h. das Notenschema bildet direkt Prozentpunkte auf Noten ab.

Falls zum Aufgabenheft gar keine Property `normalisierung` in einer der drei obigen Varianten angegeben wird, heißt das prinzipiell, dass diese Standard-Normalisierung (Anwendung des Notenschemas auf Prozentpunkte) verwendet wird.

Beim *Import* einer JSON-Datei gilt jedoch auch hier – wie bei anderen Properties auch –, dass im JSON fehlende Angaben bedeuten, dass die entsprechenden Angaben im Zielkurs nicht überschrieben werden sollen. Falls Sie also in einen Kurs, der bereits ein Aufgabenheft *mit* Nicht-Standard-Normalisierung oder abgeschalteter Normalisierung enthält, ein JSON importieren, das keine Normalisierungs-Angaben zu diesem Heft enthält, werden die bestehenden Normalisierungseinstellungen dabei nicht überschrieben, also nicht auf Standard-Prozentpunkte-Normalisierung zurückgesetzt.

Um also in einer zu importierenden JSON-Datei explizit vorzugeben, dass Sie die Standard-Normalisierung einstellen möchten, selbst wenn im Ziel-Heft bereits eine abweichende Einstellung vorliegen sollte, aktivieren Sie diese auf eine der beiden Weisen, die unter „ad 1." und „ad 2." beschrieben wurden. Am einfachsten ist es, eine leere Kontrollpunkte-Funktion (s. ad 2.) festzulegen, indem Sie einfach ein leeres Array (also eine leere Kontrollpunkte-Funktion) zuweisen:

```
"normalisierung": []
```

Alternativ funktioniert natürlich auch eine Default-Angabe zur Lotse-Normalisierung (ad 1.) wie folgt:

```
"normalisierung": {
    "bonusNormalpunkte": 0,
    "bonusRohpunkte": 0,
    "steigung": ""
}
```

Beides legt explizit Standard-Normalisierung fest, die erste Variante ist aber offensichtlich die einfachere.

Notenschema

Ein Notenschema wird angegeben als Menge von Paaren aus jeweils einer Note sowie der Mindestpunktzahl (in Normalpunkten), ab welcher die Note zu vergeben ist.

In der Konfigurationsdatei sind diese Paare als Properties des `notenschema`-Objekts anzugeben, wobei die Note als Key (String der Form `3,0`) und die Mindest-Normalpunktzahl als Integer-Value angegeben werden.

Beispiel 2.3 (Notenschema):

```
{"aufgabenhefte": {
    …
    "3": {
        …
        "benotung": "PER_SCHEMA",
        "notenspiegel": false,
        "normalisierung": …,
        "notenschema": {
            "1,0": 95,
            "1,3": 90,
            "1,7": 85,
            "2,0": 80,
            "2,3": 75,
            "2,7": 70,
            "3,0": 65,
            "3,3": 60,
            "3,7": 55,
            "4,0": 45
        }
    }
}}
```

`aufgaben`-Property von Aufgabenheft

Die Einstellungen zu einer bestimmten Aufgabe geben Sie ebenfalls als JSON-Object an (mit den im folgenden Abschnitt beschriebenen Properties). Im JSON-Dokumentbaum ist ein solches Aufgaben-Objekt in die Aufgabenliste eines Aufgabenheftes einzufügen.

Dazu besitzt jedes Aufgabenheft-Objekt (Eintrag innerhalb der `aufgabenhefte`-Property des Root-Objects, s.o.) eine Property namens `aufgaben`. Analog zur `aufgabenhefte`-Property kann es sich auch dabei wieder wahlweise um ein JSON-Array oder ein JSON-Object handeln. Im Falle eines Arrays korrespondiert wieder der erste Array-Eintrag mit der ersten Aufgabe des jeweiligen Hefts, der zweite Array-Eintrag mit Aufgabe Nr. 2 etc.. Bei Notation als JSON-Object ist wieder die Aufgabennummer als Key dem Objekt mit den Aufgaben-Properties voranzustellen.

Beispiel 3.1 mit JSON-Array:

```
{"aufgabenhefte": {
    "1": {
        "bearbeitungsbeginn": "01.04.2018, 00:00",
        "bearbeitungsende": "14.04.2018, 24:00",
        "aufgaben": [
            {
                "name": "Meine erste Aufgabe",
                "teilaufgaben": [
                    {"punkte": 10},
                    {"punkte": 1}
                ],
                "korrekturart": "HAND"
            },
            {
                "name": "Aufgabe 2",
                "punkte": 12,
                "korrekturart": "HAND"
            },
        ]
    }
}}
```

Beispiel 3.2 mit JSON-Object:

```
{"aufgabenhefte": {
    "1": {
        "bearbeitungsbeginn": "01.04.2018, 00:00",
        "bearbeitungsende": "14.04.2018, 24:00",
        "aufgaben": {
            "1": {
                "name": "Meine erste Aufgabe",
                "teilaufgaben": [
                    {"punkte": 10},
                    {"punkte": 1}
                ],
                "korrekturart": "HAND"
            },
            "2": {
                "name": "Aufgabe 2",
                "punkte": 12,
                "korrekturart": "HAND"
            },
        }
    }
}}
```

Aufgaben-Properties

Für jedes Aufgaben-Objekt (siehe vorigen Abschnitt) können wieder die Aufgaben-Einstellungen über die folgenden Properties angegeben werden. (Nicht im JSON enthalten sind die Standard-Kursressourcen, also die HTML-Dokumente mit Aufgabenformular, Quittungsvorlage, Korrekturvorlage, Musterlösungstext etc.; die sind neben der Konfigurationsdatei getrennt als Kursressourcen hochzuladen.)

Zur Bedeutung der einzelnen Einstellungen lesen Sie bitte die Online-Hilfe zu den entsprechenden Eingabefeldern in der Benutzeroberfläche der fortgeschrittenen Aufgabenerstellung.

Key	Value-Type	Inhalt / Format / Werte
`name`	String	Aufgabenname
`korrekturart`	String	Einer der folgenden Werte: `"AUTO"`, `"AUTOLEER"`, `"AUTOHAND"`, `"HAND"`, `"KEINE"`
`korrekturmodul`	Object	Name und Einstellungen zum Korrektur-/Bewertermodul (nur bei `"korrekturart": "AUTO…"`), siehe (Vor-)Korrekturmodul-Einstellungen
`thema`	String	Nur bei bestimmten Aufgaben zu Prüfungen, nicht für alle Kurse: Prüfungsthema, das von den Aufgabenteilnehmern gewählt sein muss oder durch Bearbeitung der Aufgabe gewählt wird.
`teilaufgaben`	Array oder Object	Teilaufgaben-Properties (im Falle einer in mehrere (technische) Teilaufgaben zerlegten Aufgabe, s.u.)

Teilaufgaben-Einstellungen (Punkte, Vorkorrekturmodule)

Einstellungen wie erreichbare Punktzahl oder Vorkorrekturmodule werden pro (technischer) Teilaufgabe vorgenommen. Dazu dient die Property `teilaufgaben`. Analog zu den Properties `aufgabenhefte` und `aufgaben` darf auch dies wieder wahlweise ein JSON-Array oder JSON-Object sein. Im ersten Fall bezieht sich wieder das erste Element des Arrays auf die erste Teilaufgabe (`a`), das zweite auf Teilaufgabe `b` etc., bei Notation als Object sind den Teilaufgaben-Objekten jeweils die Keys der Form `"a"` für die erste, `"b"` für die zweite Teilaufgabe etc. voranzustellen.

Da die Anzahl der Teilaufgaben einer Aufgabe (wenn überhaupt größer als 1) in der Regel eher klein ist und Teilaufgaben auch jeweils nur bis zu zwei Properties haben, und da außerdem typischerweise in der JSON-Datei auch *alle* Teilaufgaben zur Aufgabe (und nicht nur z.B. die zweite) konfiguriert werden sollen, ist hier i.A. die Array-Schreibweise hinreichend übersichtlich.

Falls eine Aufgabe aus *genau einer* Teilaufgabe besteht, wird bereits in der Weboberfläche der fortgeschrittenen Aufgabenerstellung eine andere Darstellung gewählt, indem Punkte und Vorkorrekturmodul einfach als Eigenschaften der „atomaren" Aufgabe präsentiert werden und nicht als Eigenschaften der (einzigen) Teilaufgabe. Auch im JSON ist es dann nicht nötig, eine `teilaufgaben`-Property anzugeben, sondern Sie können die Teilaufgaben-Properties dann auch direkt im Aufgaben-Objekt angeben.

Beispiel 4.1:

Betrachten Sie erneut obiges Listing aus Beispiel 3.2 oder 3.1: In beiden Fällen besteht Aufgabe 1.1 aus zwei Teilaufgaben, und da die Punkte eine Teilaufgaben-Property sind (s.u.) werden sie unter `teilaufgaben` notiert – hier in Array-Schreibweise. Die alternative Object-Schreibweise sähe wie folgt aus :

```
    "teilaufgaben": {
        "a": {"punkte": 10},
        "b": {"punkte": 1}
    },
```

Aufgabe 1.2 dagegen besteht im Beispiel aus nur einer einzigen Teilaufgabe, weshalb auf eine (ebenfalls korrekte, aber unnötig komplexe) Angabe wie …

```
    "teilaufgaben": [
        {"punkte": 12}
    ]
```

… verzichtet wurde und die Punkte-Property vielmehr direkt im Aufgaben-Objekt mit aufgenommen wurde.

An Teilaufgaben-Properties beschränken sich obige Beispiele auf den typischen Fall reiner Punktzahlangaben, d.h. es wird kein Vorkorrekturmodul konfiguriert. Eine `vorkorrekturmodul`-Property (vgl. folgenden Abschnitt) stünde ansonsten jeweils im selben Objekt wie die `punkte`-Property.

Teilaufgaben-Properties

Key	Value-Type	Inhalt / Format / Werte
`punkte`	Integer	Erreichbare Punktzahl zur Teilaufgabe
`vorkorrekturmodul`	Object	Name und Einstellungen zum Vorkorrekturmodul, das aus den Eingaben zur Teilaufgabe ein Sofortfeedback erzeugen soll, siehe (Vor-)Korrekturmodul-Einstellungen

Die zu einer Aufgabe erreichbare Punktzahl errechnet sich als Summe der in den einzelnen Teilaufgaben erreichbaren Punktzahlen. Die getrennte Erfassung von Teilpunktzahlen pro Teilaufgabe ist nicht unbedingt nötig, das ist eher eine Frage der Übersichtlichkeit. Es ist genauso möglich, die gesamte erreichbare Punktzahl der ersten Teilaufgabe zuzuordnen und zu allen weiteren Teilaufgaben jeweils 0 erreichbare Punkte einzutragen.

(Vor-)Korrekturmodul-Einstellungen

Die JSON-Objekte für Korrekturmodule und Vorkorrekturmodule sind jeweils gleich aufgebaut und können folgende zwei Properties umfassen:

Key	Value-Type	Inhalt / Format / Werte
`name`	String	Name des (Vor-)Korrekturmoduls, unter dem es sich beim Online-Übungssystem registriert.
`eigenschaften`	String	Properties-String mit den Einstellungen zum (Vor-)Korrekturmodul. White-Spaces wie Zeilenumbrüche und Tabulatoren im Properties-String sind durch Escape-Characters darstellbar.

Beispiel: Korrekturmodul (interner Aufgabenbewerter)

```
{"aufgabenhefte": {
    "4": {
        "aufgaben": {
            "2": {
                "name": "AB",
                "punkte": 100,
                "korrekturart": "AUTO",
                "korrekturmodul": {
                    "name": "AufgabenBewerter",
                    "eigenschaften":
"Init:\r\n\tAufgabentyp:\t1AUSN\r\n\tName:\t\tEinfachauswahl\r\n\tKommentar:\t\
r\n\tPunkte:\t\t100\r\n\tGewichtung:\t1\r\n\tN:\t\t4\r\n\tVorgabe:\tA\r\n"
                }
            }
            …
        }
    }
}}
```

Fußnoten

1. Was genau beim Upload einer ZIP-Datei mit Konfigurationsdatei und Standard-Kursressourcen (Aufgaben-Ressourcen) in einen leeren Kurs passiert: Zuerst wird überprüft, ob der Zielkurs schon Aufgaben enthält. Ist das der Fall, muss die Config-JSON-Datei passend zum Kurs benannt sein, sonst wird sie ignoriert. Enthält der Kurs dagegen noch keine Aufgaben, ist ihr Name egal. Ausgewertet wird die Datei noch nicht (das hätte ja auch noch keinen Effekt, so lange noch keine Aufgaben existieren). Vielmehr werden dann zunächst alle (Standard- und Nicht-Standard-)Kursressourcen aus dem ZIP-File importiert. Falls Standard-Kursressourcen (wie `aufgabe1.1.html`) dabei waren, werden im Anschluss die zugehörigen Aufgaben (z.B. Aufgabe 1 zu Heft 1) erzeugt, und die Hefte dazu ebenfalls, sollten sie nicht bereits existieren. Die neu erzeugten Hefte und Aufgaben bekommen zunächst Standard-Einstellungen. Abschließend wird dann die im Upload gefundene Config-JSON (sofern vorhanden und nicht ignoriert, s.o.) verarbeitet, also alle darin vermerkten Einstellungen auf die (ggf. im vorigen Schritt erst erzeugten) Aufgabenhefte und Aufgaben übertragen.
2. Zukünftig könnte das Format eventuell um weitere Einstellungen wie Kursparameter erweitert werden.