

Universität Stuttgart

BACHELORFORSCHUNGSPROJEKT INFORMATIK

Entwicklung eines Online-Self-Assessment-Tests für Studieninteressenten der Universität Stuttgart

Jonas Allali

Julian Blumenröther

Tim-Julian Ehret

Sokol Makolli

Jena Satkunarajan

Prüfer:

PROF. DR.-ING. STEFAN FUNKE

9. November 2018

Inhaltsverzeichnis

1	Einleitung und Motivation	4
2	Related Work	4
2.1	Test der Universität Frankfurt	4
2.2	Test der FU Berlin	5
2.3	Test der RWT Aachen	5
3	Architektur	6
3.1	Anforderungen	6
3.2	Programmaufbau	6
4	Implementierung	6
4.1	Creator	6
4.2	Parser	8
4.2.1	Interne Struktur	8
4.2.2	Speichern von Fragen	8
4.2.3	Einlesen von Fragen	9
4.3	Generator	9
4.3.1	Velocity	9
4.3.2	Erstellung der Web- seite	10
4.4	Website	10
4.4.1	Überblick über die statischen Dateien . . .	10
4.4.2	Anzeige der Fragen . . .	10
4.4.3	Zustandsverwaltung . . .	11
4.4.4	Evaluierung	12
5	Anwendungsszenario	12
5.1	Programm ausführen	12
5.2	Fragen hinzufügen	12
5.3	Webseite generieren	12
5.4	Selfasessment-Test durch- führen	13
6	Zusammenfassung und Ausblick	13
	Literaturverzeichnis	14

Kurzfassung

In dieser Arbeit wurde eine Java Applikation entwickelt, mit der ein webbasierter Self-Assessment-Test erstellt werden kann. Ersteller sind damit in der Lage auch ohne Programmierkenntnisse, eine lauffähige Webseite zu entwerfen. Das Programm generiert dann einen funktionsfähigen Online-Test, der mit einem HTTP-Server angeboten werden kann.

1 Einleitung und Motivation

In jüngster Zeit haben Universitäten mit immer höher werdenden Abbruchraten zu kämpfen. Der Grund für den Abbruch des Studiums ist dabei häufig, dass die Studierenden nicht ausreichend über die Inhalte und die Anforderungen ihres Studiengangs informiert wurden. Um diesem Problem vorzubeugen werden von vielen Universitäten sogenannte Self-Assessment-Tests angeboten.

Der Aufwand der Studieninteressenten wird dabei **miniert**, indem solche Tests online verfügbar sind. Angehende Studierende sollen durch das Absolvieren des Tests eine Einschätzung ihrer eigenen Fähigkeiten bekommen. Ferner soll ihnen verdeutlicht werden, ob der angestrebte Studiengang ihren Vorstellungen und Interessen entspricht. Hierbei ist eine eindeutige, einfache Bedienung und vor allem ein persönliches Feedback von **hoher Wichtigkeit**.


Das Ziel dieser Arbeit ist es die Erstellung eines Online-Selfassessment-Tests zu vereinfachen oder **es für einen Nichtprogrammierer überhaupt möglich zu machen**. Zu diesem Zweck wurde **unser** Testgenerator mit einer einfach zu bedienenden Benutzeroberfläche entwickelt.

2 Related Work

Um sich einen Überblick über die Thematik verschaffen zu können, wurden

verschiedene Self-Assessment-Tests anderer Universitäten analysiert. Hierbei ist wichtig zu wissen, dass folgende Beschreibungen sich nur auf die jeweiligen *Graphical User Interfaces* der Tests beziehen und es generell nicht möglich ist, auf die Implementierung, die dahinter steckt, genauer einzugehen.

2.1 Test der Universität Frankfurt

Der **Test** beginnt mit einem Motivations-text, der sowohl Sinn und Zweck des Tests erklärt, als auch den User in die Benutzung einführt. Dies wurde bei unserer Arbeit weggelassen, da eine Einführung auch auf der jeweiligen Webseite, die unseren Test integriert, hinzugefügt werden kann. Desweiteren wird angeboten den Test der Universität Frankfurt **runterzuladen**, was eine Offline-Ausarbeitung **realisiert**. Bei unserem Test wird der aktuelle Zustand in der URL der Webseite kodiert. Dies ermöglicht zwar keine Bearbeitung ohne Internet, bietet aber an, den Test jederzeit zu pausieren, indem man sich die URL rauskopiert. Der zu vergleichende Test erfordert außerdem eine persönliche Registrierung, welche sehr ausführlich ist. s dazu führt, dass die **User Experience** sinkt. Daher wurde eine Registrierung in unserem Test weggelassen, um die Flexibilität und Einfachheit zu gewährleisten. Eine weitere Einschränkung ist die Notwendigkeit des Adobe Flash Players, auf den wir ganz verzichten. In Abbildung 1 ist das Grundlayout des Tests von der Universität Frankfurt zu sehen. Das Layout ist sehr ähnlich zu unserem und weist die Frage mit ihren Antworten, einen Fortschrittsbalken, einen Next-Button und einer Zeitanzeige auf. Auch ist es möglich Grafiken anzeigen zu lassen. Der Test wird in verschiedene Kategorien eingegliedert, die wiederholbar sind. Bei unserem Test ist die Erstellung von Kategorien ebenfalls möglich, aber auf jegliche Wiederholungen wurden verzich-

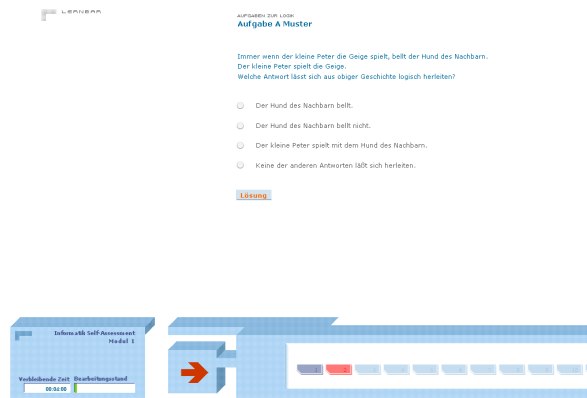


Abbildung 1:

tet. Abschließend bietet der Test der Uni-

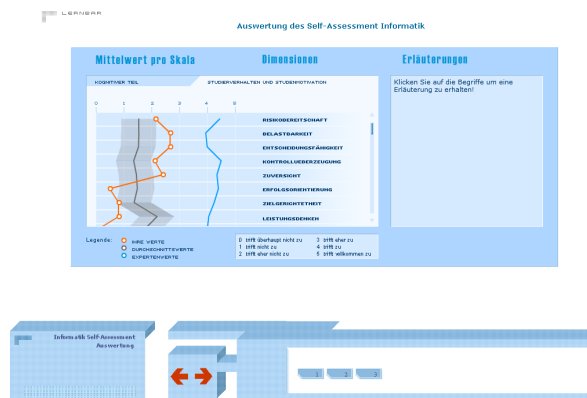


Abbildung 2:

versität Frankfurt eine Bewertung der beantworteten Fragen, die in Abbildung 2 zu sehen ist. Die Grafik ist zwar schön anzusehen, jedoch fehlt hier eine klare Interpretation der Ergebnisse: Das Endresultat muss man quasi selbst entscheiden. Daher haben wir bei unserem Test darauf geachtet, dass eine persönliche Beurteilung hinzugefügt werden kann, die zum Beispiel dem User Tipps gibt, falls er in einer Kategorie schlecht abschneidet oder setzt generell die Ergebnisse des Users in Bezug auf den Studiengang.

2.2 Test der FU Berlin

Abbildung 3 zeigt einen Ausschnitt aus dem Test der FU Berlin. Dieser veranschaulicht eine schöne Gliederung und auch die Möglichkeit das Ergebnis der Frage sofort zu sehen. Hier wird viel mit Mul-

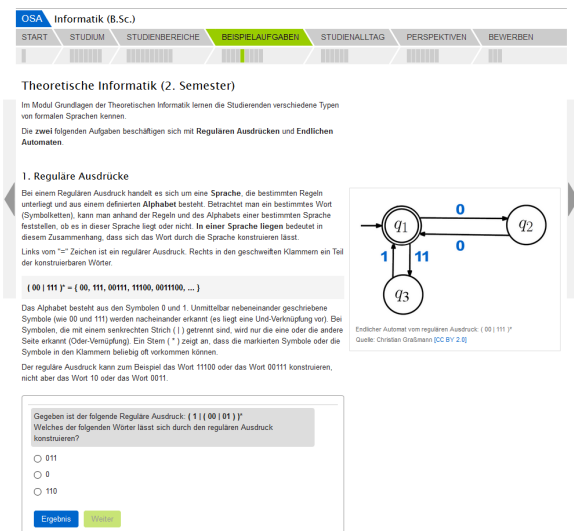


Abbildung 3:

timedia gearbeitet, was das Interesse der User fangen soll, um die User Experience zu steigern. Auch in unserem Test ist das einbetten von Multimedia-Dateien möglich.

2.3 Test der RWT Aachen

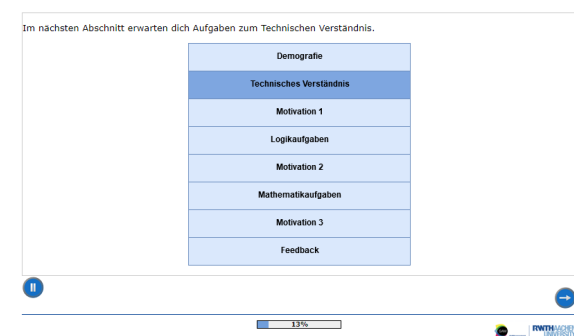


Abbildung 4:

Der Test der RWT Aachen ähnelt unserem Modell wie man in Abbildung 4 sehen kann. Das Layout ist einfach gehalten und überschaubar. Es gibt einen Next-Button und einen Fortschrittsbalken. Allerdings bietet der Test keine Möglichkeit an, die Fragen zu überspringen, was bei unserem Test durchaus machbar ist. Auch bietet der Test der RWT Aachen keinen mobilen Support. Dies ist bei uns automatisch integriert, da wir mit dem Webbrowser arbeiten.

3 Architektur

Im folgenden soll ein Überblick des Projekts vermittelt werden. Hierzu werden die Anforderungen und schließlich die daraus resultierende Softwarearchitektur vorgestellt.


3.1 Anforderungen

Das Endresultat soll ein Programm sein, mit dem der Ersteller mühelos eine Webseite generieren kann. Besonderes Augenmerk liegt dabei auf der leichten Bedienbarkeit.

Folgende Anforderungen werden an den Aufbau der Fragen gestellt:

- Der Test soll aus Multiple-Choice-Fragen bestehen.
- Die Fragen und Antworten sollen Text, Bilder und Videos enthalten können.
- Die Bearbeitungszeit jeder Frage soll optional beschränkt sein.
- Die Fragen sollen in Kategorien gegliedert werden können.
- Es soll einen Fortschrittsbalken geben, welcher anzeigt wie viele Fragen bereits beantwortet wurden und wie viele noch zu beantworten sind.

Damit die Studieninteressenten Feedback erhalten können, muss am Ende eine Bewertung der eingegebenen Antworten durchgeführt werden. Für diese Bewertung gelten folgende Anforderungen:

- Die Gewichtung einer Frage soll vom Ersteller durch Angabe von Punkten festgelegt werden können.
- Für jede Kategorie soll die Anzahl der richtig bzw. falsch beantworteten Fragen angezeigt werden.
- Basierend auf der erreichten Gesamtpunktzahl soll n Ersteller gewähltes Feedback gegeben werden.

3.2 Programmaufbau

Abbildung 3.2 zeigt die implementierte Klassenstruktur.

Mit Hilfe des **Creator GUI**, beschrieben in Abschnitt 4.1, kann man die Objekte der im Klassendiagramm dargestellten Klassen instanziiieren. So kann der Ersteller einen Test modellieren. Aus diesen Objekten kann schließlich direkt eine Webseite generiert werden. Die Beschreibung des Generators kann Abschnitt 4.3 entnommen werden.

Der Parser, dokumentiert in Abschnitt 4.2, bietet die Möglichkeit, die erstellten Objekte in einer XML-Datei zu speichern und sie wieder einzulesen.

4 Implementierung

4.1 Creator

Der 'Creator' wurde dazu entwickelt, um dem Benutzer die Möglichkeit zu geben, einen Self-Assessment-Test zu erstellen ohne die dafür anderweitig nötigen Kenntnisse in XML zu besitzen bzw. es ihm zu erleichtern, auch wenn er sie besitzt. Das Programm verfügt über weitreichende Funktionen, damit der Test die benötigten Eigenschaften bieten kann. In diesem Abschnitt werden diese Funktionen erläutert und eine Gesamtübersicht über das Programm gegeben.

Übersicht

Das **Haupt-Fenster** des Creators beinhaltet fünf wichtige Elemente die zur Übersicht und Erstellung des Testes dienen. Auf der linken Seite ist ein 'TreeView', der die Struktur des Testes darstellt. In ihm werden die Kategorien, deren zugehörigen Fragen, Antworten und die Folgerungen angezeigt. Jedes 'TreeItem', das in ihm existiert, repräsentiert ein Java-Objekt der zugehörigen Klasse, dessen Name in den 'TreeItems' angezeigt wird.

Abbildung 5: UML Klassendiagramm des Testgenerators

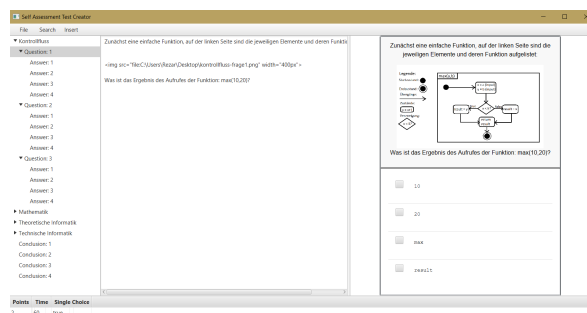
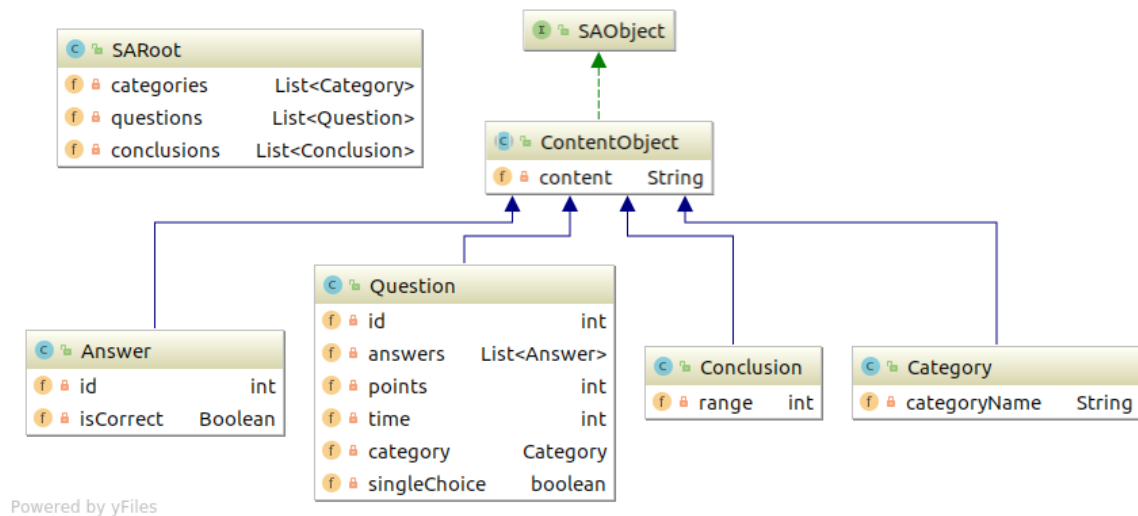


Abbildung 6:

In der Mitte gibt es ein großes Textfeld, das das Inhaltsattribut des zugehörig ausgewählten 'TreeItems' wiedergibt. In ihm kann man dieses mithilfe von HTML und Markdown editieren bzw. nach belieben verändern.

Im unteren Bereich wird eine Tabelle erstellt, die alle Attribute des im 'TreeView' ausgewählten Elements anzeigt. Diese Attribute variieren je nach dem welche Klasse das Element besitzt. Kategorien besitzen als Attribut ihren Namen, Fragen hingegen haben die Attribute Punkte und Zeit, Folgerungen enthalten das Attribut Umfang.

Die obere Leiste enthält 'MenuItems' mit mehreren Funktionen, die in einem späteren Abschnitt näher betrachtet

werden.

Auf der rechten Seite gibt es ein großes Feld, dessen Funktion darin besteht, aus den im 'TreeView' ausgewählten Fragen und deren Antworten ein HTML-Dokument zu erstellen und anzuzeigen, damit der Benutzer sehen kann, wie die Website am Ende aussehen wird. Dieses Feld wird immer aktualisiert, sobald der Benutzer in dem Textfeld etwas verändert oder ein anderes Element im 'TreeView' ausgewählt wird.

Der Balken zwischen dem Textfeld und der Seitengenerierung ist dazu, da die Größe jedes Feldes beliebig anzupassen, außerdem kann der Benutzer die Größe des gesamten Fensters ohne Probleme nach belieben verändern.

Funktionen

Unter der Kategorie 'File' gibt es neun Buttons, die nach Funktionen sortiert und mit 'SeparatorItems' voneinander getrennt sind. Die ersten vier, also 'New Category', 'New Question', 'New Answer' und 'New Conclusion', erstellen die jeweils zugehörigen 'TreeItems', die anschließend im 'TreeView' angezeigt werden. 'Delete Item' entfernt ein ausgewähltes 'TreeItem'

und dessen zugehöriges Java Objekt. Die bisher aufgezählten Funktionen sind auch in einem 'ContextMenu' des 'TreeViews' zu finden. 'Generate Website' erstellt einen Zip-Ordner, der HTML, Javascript und CSS-Dateien enthält, die später zur Erstellung der Webseite des Servers dienen. Die zwei 'MenuItems' 'Import XML' und 'Export XML' fragen den Benutzer nach einer XML-Datei, in die der Inhalt gelesen bzw. gespeichert wird. Das letzte 'MenuItem' 'Exit' beendet das Programm, dabei wird der Benutzer zu einer Bestätigung aufgefordert.

Die zweite Kategorie 'Search' enthält zwei Funktionen, die jeweils neue Fenster aufrufen, mit denen der Benutzer in dem großen Textfeld nach einem beliebigen Text suchen bzw. ihn ersetzen kann. Dabei kann er auch die Option auswählen, Groß- und Kleinschreibung nicht zu beachten.

Die dritte und letzte Kategorie 'Insert' ist dazu da, Medien, also Bilder und/oder Videos, in das Textfeld einzufügen. Dabei wird der Benutzer danach gefragt eine Datei auszuwählen, deren Dateipfad anschließend in das Textfeld, an der Stelle, an dem sich der Cursor des Benutzers befindet, eingefügt wird.

4.2 Parser

In diesem Kapitel wird die Parser Komponente vorgestellt. Ihre primäre Aufgabe besteht darin, zwischen dem Speichermedium und dem Generierungsprozess zu vermitteln. Als Speichermedium dienen hierbei einfache XML Dateien, während die aktive Generierung mit Java Objekten arbeitet.

Im folgenden wird zunächst die Interne Struktur vorgestellt, welche der Implementierung zugrunde liegt. Anschließend wird die Funktionalitäten der Parser Komponente im Detail vorgestellt.

4.2.1 Interne Struktur

Generell sind alle hier relevanten Teile eines Self Assessment Tests (Kategorien, Fragen, Antworten, Ergebnis) Java Objekte. Nachdem eine Frage erstellt wurde ist sie zunächst ein Java Objekt der Klasse 'Question'. Als solches besitzt die Frage verschiedene Attribute wie z.b. ihre Kategorie, den Aufgabentext und insbesondere eine Liste der ihr zugehörigen Antworten. Somit ist gleichzeitig das Mapping zwischen Fragen und Kategorien, sowie zwischen Fragen und Antworten gegeben. Um den ganzen erstellten Test zu sichern genügt es also, eine Liste der Fragen speichern. Der Einfachheit halber haben wir eine Toplevel Klasse namens 'SARoot' eingeführt. Ein Test wird so letztlich durch ein Objekt der Klasse 'SARoot' repräsentiert, welches eine Liste von Fragen und Kategorien, sowie das Ergebnis beinhaltet.

4.2.2 Speichern von Fragen

Wie in Kapitel 4.2 beschrieben, kann der Ersteller den aktuellen Arbeitsstand speichern, indem er den Test als XML Datei exportiert. In der GUI findet man diese Operation unter 'Export XML'. Hierbei werden im Hintergrund alle erstellten Java Objekte an die Parser Komponente übergeben, welche diese dann in XML Elemente umwandelt und in einer Datei abspeichert. Genauer gesagt wird hier genau ein Toplevel Objekt übergeben, welches alle Objekte beinhaltet.

Intern verwendet die Parser Komponente die 'Java Architecture for XML Binding (JAXB)' ¹. Der von JAXB bereitgestellte 'marshaller' ermöglicht es Java Objekte von spezifizierten Klassen direkt in XML Elemente zu verwandeln und anschließend in eine XML Datei zu schreiben. Um vom 'marshaller' erkannt zu werden benötigen alle unsere Java Klassen die von JAXB vorgeschlagenen XML Tags.

Die Parser Komponente nutzt diese Funktion der JAXB API um unser Spei-

¹<https://javaee.github.io/jaxb-v2/>

chermedium, eine XML Datei, zu erstellen. So können die nur zur Laufzeit existierenden Objekte persistent gespeichert werden.

4.2.3 Einlesen von Fragen

Um gespeicherte Fragen oder ganze Tests wiederverwendbar zu machen, bietet die Parser Komponente die Möglichkeit XML Dateien einzulesen. Es findet als eine Überführung von XML Elementen in Java Objekte statt. Ähnlich wie beim Speichern verwendet die Parser Komponente hierzu wieder eine Funktionalität des JAXB 'marshaller's. Die Funktionalität besteht darin, aus XML Elementen Java Objekte von Klassen mit passenden XML Tags zu erzeugen.

Das Resultat des Einlesens ist ein Toplevel Objekt der Klasse 'SARoot'. Aus diesem Objekt kann der gesamte Self Assessment Test erstellt und die Webseite generiert werden.

Die GUI greift auf die Parser Komponente zurück um die 'Import XML' Operation durchzuführen. Außerdem ist es dem erfahrenen Ersteller nun möglich den gesamten Self Assessment Test 'von Hand' in einer XML Datei zu verfassen, ohne hierfür die GUI zu verwenden.

4.3 Generator

Für das einfache Verwalten der Webseite, ist es vorgesehen, dass sie statisch ist. In diesem Fall heißt das, dass es keinen Server geben soll, der dynamisch auf Anfragen des Benutzers reagiert. Alle dynamischen Funktionen, wie das Laden neuer Fragen, finden auf der Benutzerebene statt. Der Server liefert dem Benutzer bei Bedarf, also nur statische Dateien.

Der Vorteil dieses Designs ist, dass die Webseitendateien nur ein Mal aus den Java Objekten generiert werden müssen. Für diese Funktion haben wir uns für die Template Engine Velocity ² entschieden.

4.3.1 Velocity

Velocity erlaubt es Dokumente mit Variablen zu bestücken, die dann von Velocity mit dem Text aus den Java Objekten ersetzt werden. Diese Dokumente werden Templates genannt. Dafür werden Velocity das Java Objekt, der Name des Java Objektes in dem Template und das Template an sich übergeben. Velocity liest daraufhin das Template, sucht sich die Stellen heraus, die Variablen enthalten, und ersetzt diese mit den Inhalten des Java Objektes.

Ein Beispiel eines Velocity Templates ist in Listing 1 gegeben. Für das Beispiel wird angenommen, dass Velocity ein Question Objekt übergeben bekommt. Dieses Question Objekt hat eine Funktion 'getContent()', die einen String zurückgibt, und eine andere Funktion 'getAnswers()', die eine Liste mit Answer Objekten zurückgibt. Das Answer Objekt hat wiederum auch eine Funktion 'getContent()'. Außerdem wird Velocity der Variablenname 'question' und das aufgeführte Template übergeben.

Das Zeichen '\$' in dem Template signalisiert Velocity, dass der danach kommende Text für Velocity vorgesehen ist. So bemerkt Velocity, dass '\$question', das übergebene Question Objekt referenziert und ruft im ersten Fall die Funktion 'getContent()' auf. Die Funktion wird ausgewertet und der zurückgegebene String wird an der Stelle des Funktionsaufrufs gesetzt.

Im Beispiel-Template sieht man auch eine 'foreach' Schleife, die mit '#foreach' beginnt und mit '#end' endet. Diese Schleife sorgt dafür, dass der Abschnitt, der sich in der Schleife befindet, so oft geschrieben wird, wie es Answer Objekte in der von 'question.getAnswers()' zurückgegebenen Liste gibt.

Daraufhin wird '\$answer.getContent()' mit der entsprechenden Rückgabe der Answer Objektes ersetzt.

Listing 1: Beispiel eines Velocity Templates.

```
<h1>$question.getContent()</h1>
```

²<http://velocity.apache.org/>


```
<ul>
#foreach($answer in $question.getAnswers())
<li>$answer.getContent()</li>
#end
</ul>
```

4.3.2 Erstellung der Webseite

Für alle Dateien der Webseite, die Inhalte von den Java Objekten benötigen, wird eine **Template Datei** erstellt. Daraufhin werden die ausgewerteten Templates, mit den anderen für die Webseite benötigten Dateien, in ein ZIP-Archiv gepackt und in einen von dem Benutzer des Generators festgelegten Ort gespeichert.

Um die Webseite dann zu veröffentlichen, müssen die Dateien im Archiv über einen HTTP-Server angeboten werden.

4.4 Website

Im folgenden soll skizziert werden, wie die Website des Online-Assessment-Tests bereitgestellt und präsentiert wird. Dazu wird zunächst ein Überblick über die durch den Generator bereitgestellten statischen Dateien erstellt. Anschließend wird beschrieben, wie aus den statischen Daten die Seiten der Website zusammengestellt werden. Ein besonderer Schwerpunkt liegt dabei auf der Anzeige der Fragen auf den Seiten und der damit verbundenen Verwaltung des Zustandes. Zuletzt wird die Auswertung und die Anzeige der Evaluierung des Tests beleuchtet.

4.4.1 Überblick über die statischen Dateien

Die im Abschnitt zuvor erwähnten statischen Dateien im durch den Generator erzeugten Zip-Archiv bestehen aus

- einer einzelnen HTML-Seite, welche rudimentär mit zu ersetzenden *Stubs* gefüllt ist
- den CSS-Dateien für das Styling der Seiten
- den JavaScript-Dateien zur Manipulation der HTML-Seite

- den Fragen in der Form von HTML und JSON-Dateien

- und die im Text verwendeten Bilder und Videos

Die genannten Dateien werden anschließend mithilfe eines HTTP-Servers dem Client angeboten.

4.4.2 Anzeige der Fragen

Die gesamte Webseite basiert auf einer einzelnen HTML-Seite, die für jede Frage des Tests verändert wird. Die Webseite erweckt damit den Anschein, jeder einzelnen Frage eine Seite zu widmen, operiert jedoch nur auf der einzelnen HTML-Seite. Ist im folgenden die Rede von einer Seite, so wird damit entsprechend die manipulierte HTML-Seite für eine Frage referenziert.

Im folgenden soll der Aufbau jeder Frage-Seite, von oben nach unten betrachtet, erläutert werden. Die Seiten beginnen stets mit dem Logo der Universität Stuttgart. Es folgt eine Übersichtsleiste, welche die verschiedenen Kategorien des Tests anzeigt. Die Kategorie der aktuell angezeigten Frage wird zur Hervorhebung grau markiert. Unterhalb der Übersichtsleiste findet sich der Fortschrittsbalken. Dieser setzt sich aus "n" kleineren Balken zusammen, wobei "n" die Gesamtanzahl aller Fragen ist. Die Anzahl der gefüllten Balken gibt dabei an, wie viele Fragen, inklusive der aktuellen Frage, bisher beantwortet wurden, die Anzahl der leeren Balken stellt dagegen die verbleibende Anzahl an Frage dar. Unterhalb des Fortschrittsbalkens und zentral im Sichtfeld werden die aktuelle Frage und ihre Antwortmöglichkeiten angezeigt. Die Frage erhält ein eigenes Feld, in welcher optional eine Erklärung zur Frage und ein Bild angezeigt werden können. Separiert von der Frage folgen dann ihre Antwortmöglichkeiten, die jeweils eigene Felder erhalten. Die Darstellung der Antworten hängt vom Fragentyp ab. Für alle Fragentypen

wird links neben der Antwort eine Checkbox platziert. Handelt es sich um eine Medienfrage, welche Bilder oder Videos beinhaltet, so können die betroffenen Medien im Raum rechts von der Antwort angezeigt werden. Abbildung X zeigt die Benutzeroberfläche der Webseite für die Seite einer Multiple-Choice-Frage ohne zusätzliche Medien, Abbildung Y zeigt wie Bilder in die Frage und die Antworten integriert werden. Zuletzt findet sich der "Next"-Button rechtsbündig unter dem Feld der Frage und ihrer Antworten, mit welchem die nächste Frage angezeigt werden lassen kann.

[Abbildung X: MC Frage]

[Abbildung Y: Image/Video Frage]

Es wird zunächst die nahezu leere HTML-Seite aufgerufen, welche als das Skelett für die Seiten der Fragen dient. In der Folge wird der Inhalt der HTML-Seite durch die aktuell anzuzeigende Frage bestimmt. Für jede Frage des Online-Assessment-Tests wird das Codefragment, welches zur Anzeige der Frage benötigt wird, aus der serverseitig gespeicherten HTML-Datei der Frage gelesen und in das HTML-Dokument geschrieben. Zusätzlich legt eine JSON-Datei pro Frage fest das optionale Zeitlimit und die Punkte für die korrekte Beantwortung der Frage fest. Die Auswahl der korrekten HTML und JSON-Dateien geschieht durch die Nummerierung der Fragen und der korrespondierenden Dateien von "0" bis "n", wobei "n" die Gesamtanzahl aller Fragen ist. Jedoch werden weder die aktuelle Nummer, noch die bereits angezeigten und beantworteten Fragen serverseitig abgespeichert. Die bisher angezeigten und beantworteten Fragen werden im Rahmen der Zustandsverwaltung mitverwaltet, die aktuelle Nummer wird dann anhand des Zustands abgeleitet.

4.4.3 Zustandsverwaltung

Der Zustand der Webseite setzt sich im wesentlichen aus den bisher beantworteten Fragen und ihren Antwortmöglichkeiten

zusammen. Für jede beantwortete Frage wird ein Bitstring der Länge "m" verwaltet. Dieser gibt an, wie viele Antwortmöglichkeiten die Frage aufwies und wie die Frage durch den Nutzer beantwortet wurde. Die Antworten werden dabei ausgewertet, indem für jede Antwort überprüft wird, ob die dazugehörige Checkbox angekreuzt wurde. Die Anzahl der Antwortmöglichkeiten einer Frage wird zunächst binär umgewandelt und wird in der Folge in den ersten 5 Bit des Bitstrings kodiert. Die restlichen $m - 5$ Bit des Bitstrings stellen die einzelnen Antwortmöglichkeiten der Frage dar. Jede einzelne Antwortmöglichkeit wird dabei durch ein einzelnes Bit repräsentiert, welches gesetzt ist, wenn die Antwortmöglichkeit angekreuzt wurde, ansonsten ist es 0. Wird beispielsweise eine Frage mit 3 Antwortmöglichkeiten gestellt, wobei nur die erste der drei Antwortmöglichkeiten angekreuzt wurde, so ist der resultierende Bitstring der Länge 8 für diese Frage 00011100.

Die einzelnen Bitstrings der jeweiligen Fragen werden dann konkateniert und der resultierende String durch die Umwandlung in *Base64* verkürzt. Anschließend wird diese Zeichenfolge als URL-Parameter an die aktuelle URL angehängt. Für die Anzeige einer Frage wird somit die aktuelle URL abgerufen, der URL-Parameter extrahiert und dekodiert, aus dem dekodierten Bitstring dann die Anzahl bisher beantworteter Fragen errechnet und anhand dessen die Nummer der zu lesenden HTML und JSON-Datei abgeleitet.

Beim initialen Aufruf der Webseite wird somit die erste Frage angezeigt, während der URL-Parameter leer ist. Beim Drücken des "Next"-Buttons werden dann die Antworten ausgewertet, ehe das Skript zur Verwaltung des Zustandes mit dem Ergebnis aufgerufen wird. Die URL wird dann entsprechend angepasst und die nächste Frage geladen.

4.4.4 Evaluierung

Wurde die letzte Frage erreicht und der "Next"-Button gedrückt, wird der finale Zustand aus der URL gelesen, anschließend wird das Skript zur Auswertung der Evaluation aufgerufen. Das Skript vergleicht die Antworten des Nutzers mit einem durch den Generator bereitgestellten Lösungsstring. Für jede korrekt beantwortete Frage wird die durch die Frage vergebene Punktzahl zur erreichten Kategoriepunktzahl und Gesamtpunktzahl addiert. Die Kategoriepunktzahl gibt die erreichte Punktzahl für sämtliche Fragen einer Kategorie an, die Gesamtpunktzahl die erreichte Punktzahl für die Gesamtheit aller Fragen. Die verschiedenen Kategoriepunktzahlen und die Gesamtpunktzahl werden dann in einer spezifischen Evaluationsseite dargestellt, die durch die Manipulation der HTML-Seite erzeugt wird. Abbildung Z zeigt die Evaluation nach einem Durchlauf eines Beispieltests.

[Abbildung Z: Evaluation für 3 Kategorien mit 1x grün, 1x gelb, 1x rot + Fazit]

Die Evaluationsseite setzt sich aus einer kurzen Beschreibung der Seite, der Anzeige der erreichten Punkte für jede einzelne Kategorie und einem Fazit abhängig von der Gesamtpunktzahl zusammen. Für die einzelnen Kategorien wird anhand eines Fortschrittsbalkens angezeigt, wie viele der maximal erreichbaren Punkte der betroffenen Kategorie erreicht wurden. Der Fortschrittsbalken wird dabei grün gefärbt, wenn mindestens 75% der Fragen der Kategorien korrekt beantwortet wurden, gelb gefärbt wenn zwischen 25% und 74% der Punkte erreicht wurden und rot gefärbt sonst.

Für das Fazit wird eine spezifische JSON-Datei anhand der erreichten Gesamtpunktzahl ausgewertet. Der Ersteller des Tests legt in dieser Datei fest, welcher Fazittext für welches Punkteintervall angezeigt werden soll. Es wird dann überprüft in welchem Intervall die erreichte Gesamtpunktzahl liegt und das entsprechende Fazit im Dropdown-Feld unter den

Fortschrittsbalken angezeigt.

5 Anwendungsszenario

5.1 Programm ausführen

Zuerst führt man die Datei *TextEditor.java* aus, die wie folgt im Projekt-Ordner zu finden ist:

```
src»main»java»creator»TextEditor.java
```

Jetzt öffnet sich das Haupt-Fenster des Creators (siehe Abbildung 6).

5.2 Fragen hinzufügen

Nun fügen wir unter File in der **Toolbar** neue Kategorien mit Fragen und **deren** Antworten hinzu. Dabei geben wir an, wie viele Punkte eine Antwort gibt, wie viel Zeit sie maximal **beötigt**, und **ob sie Single Choice zulässt an** (Abbildung 6). Nachdem wir angegeben haben welche Antworten korrekt sind, fügen wir noch entsprechende **Conclusions** hinzu, deren Range angibt, bis zu wie vielen Punkten jene Conclusion am Ende in der Bewertung angezeigt wird.

5.3 Webseite generieren

Falls **wir** noch nicht fertig sind und eine Pause machen möchten, kann unser Fortschritt als XML-Datei **export** und zu einem späteren Zeitpunkt wieder importiert werden.

Um die letztendliche HTML-Datei zu erstellen **clickt** man als Nächstes auf *Generate Website* unter *File* und speichert das Projekt mit der inkludierten HTML-Datei in einem beliebigen Ordner.

optional: lokalen Server erstellen

Da der Selfassessment-Test einen Server benötigt, kann man, falls man keinen Server besitzt, auch das Ganze auf einem Lo-

kalen Server testen. Dies gelingt zum Beispiel mit *Node.js*.

5.4 Selfassessment-Test durchführen

Sobald die Verbindung zu einem Server besteht, kann der Test benutzt werden. Der Test an sich ist sehr intuitiv zu bedienen. Man wählt seine Antworten aus und **clickt** auf *Next*. Sollte der Timer ablaufen wird die **eingeloggte** Antwort genommen und zur nächsten Frage gesprungen. Am Ende kann man noch seine Bewertung **einsehen**.

6 Zusammenfassung und Ausblick

Literaturverzeichnis