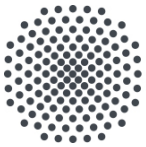


Universität Stuttgart

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

11. November 2018

Inhaltsverzeichnis

Kurzfassung

In dieser Arbeit wird 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 einen funktionsfähigen Online-Test, der mit einem HTTP-Server angeboten werden kann. Die Anzeige des Tests wird dann ohne serverseitige Zustandsverwaltung realisiert.

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 Studieninteressiernten wird dabei minimiert, 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 großer Bedeutung.

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

2 Verwandte Arbeiten

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 Benutzeroberflächen der Tests beziehen.

2.1 Test der Universität Frankfurt

Der Test [1] der Universität Frankfurt beginnt mit einem Motivationstext, der sowohl Sinn und Zweck des Tests erklärt, als auch den User in die Benutzung einführt.

Desweiteren wird angeboten den Test der Universität Frankfurt herunterzuladen, was eine Offline-Bearbeitung 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 abspeichert.

Der zu vergleichende Test erfordert außerdem eine persönliche Registrierung, welche sehr ausführlich ist, was dazu führt, dass die Benutzererfahrer sinkt. Daher wurde eine Registrierung in unserem Test weggelassen, um die Flexibilität und Einfachheit zu gewährleisten. In Abbildung 1 ist das Grundlayout des Tests von der Universität Frankfurt zu sehen. Das Layout ist sehr ähnlich zu unserem und beinhaltet die Frage mit ihren Antworten, einen Fortschrittsbalken, einen Next-Button und eine Zeitanzeige.

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, ein Mehrfachbeantwortung ist jedoch ausgeschlossen.

[1] `https://www.gdv.informatik.uni-frankfurt.de/selfassessment/Informatik/`

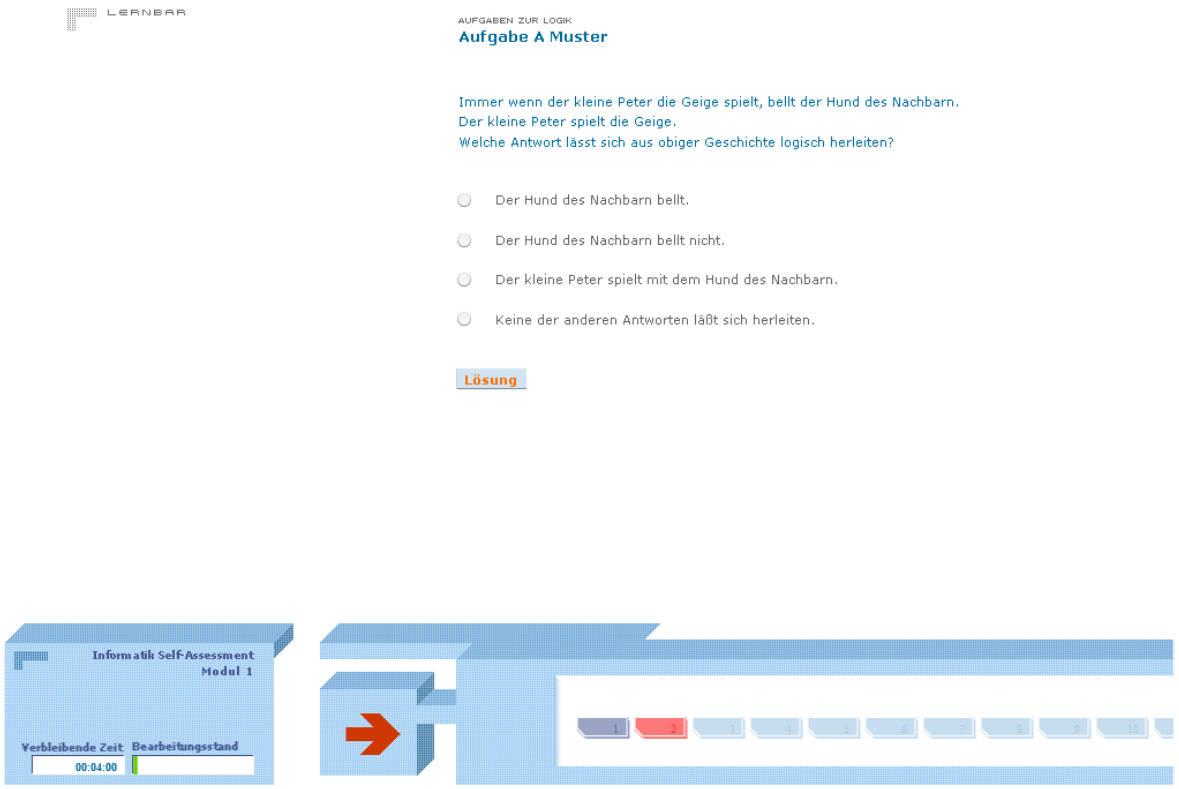


Abbildung 1:

Abschließend bietet der Test der Universität Frankfurt eine Bewertung der beantworteten Fragen. Anders als die Universität Frankfurt enthält unser Test insbesondere eine persönliche Beurteilung des Erstellers. Diese diehnt als Abschließendes Feedback für die erbrachte Leistung des Nutzers im Test.

2.2 Test der RWTH Aachen

Der Test der RWTH Aachen [2] ähnelt unserem Ansatz, wie man in Abbildung 2 sehen kann. Das Layout ist einfach gehalten und überschaubar. Es gibt einen Next-Button und einen Fortschrittsbalken. In beiden Ansätzen besteht keine Möglichkeit, eine Frage zu überspringen. Außerdem bedarf es einer Anmeldung, um am Test der Hochschule Aachen teilnehmen zu können. Damit die Hemmschwelle zur Teilnahme möglichst niedrig gehalten wird, haben wir dafür entschieden, auf jede Form der Anmeldung zu verzichten.

[2] `https://www.global-assess.rwth-aachen.de/rwth/tm_alt/`

3 Architektur

Im Fogenden soll ein Überblick über das Projekt 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 einen Online-Self-Assessment-Test 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.

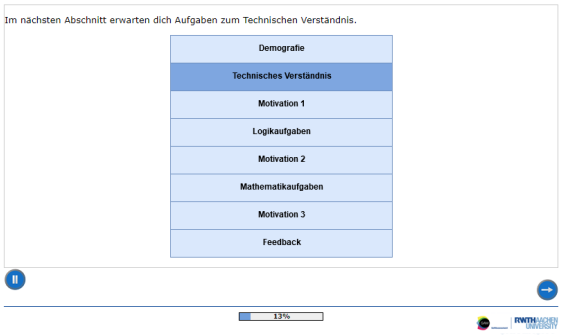


Abbildung 2:

- Die Bearbeitungszeit jeder Frage soll optional beschränkt sein.
- Die Fragen sollen in Kategorien gegliedert werden können.
- Es soll einen Forschrittsbalken geben, welcher anzeigt wie viele Fragen bereits beantwortet wurden und wie viele noch zu beantworten sind.

Damit die Studieninteressierten 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 beziehungsweise falsch beantworteten Fragen angezeigt werden.
- Basierend auf der erreichten Gesamtpunktzahl soll ein vom Ersteller gewähltes Feedback gegeben werden.

3.2 Programmaufbau

Abbildung 3 zeigt die implementierte Klassenstruktur.

Mit Hilfe der 'Creator'-Benutzeroberfläche, beschrieben in Abschnitt 4.1, kann man Objekte der im Klassendiagramm dargestellten Klassen instanziieren. 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.

Der Aufbau der generierten Webseite wird in Abschnitt 4.4 beschrieben.

4 Implementierung

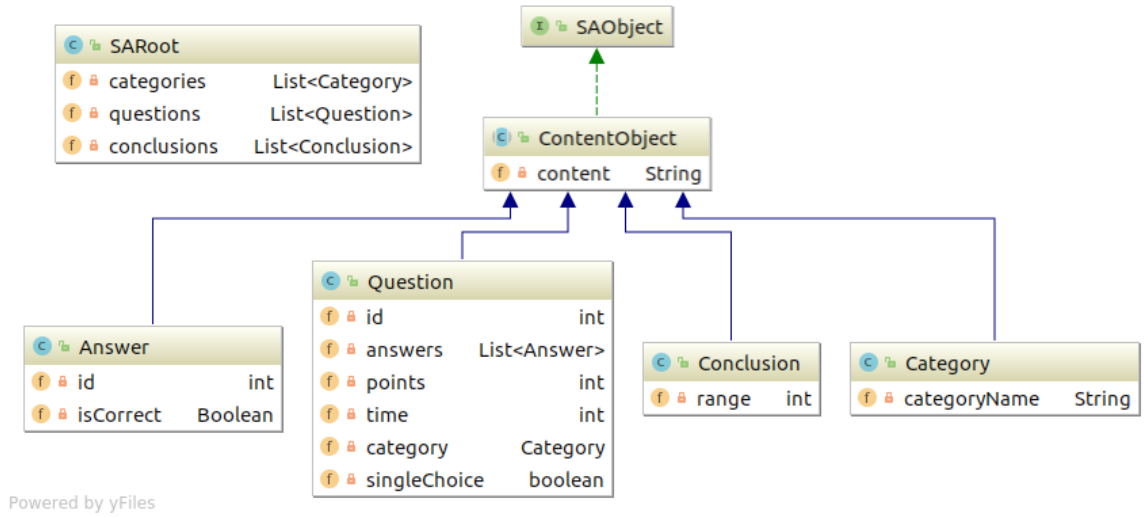
4.1 Creator

Der 'Creator' wurde dazu entwickelt, um dem Benutzer die Möglichkeit zu geben, einen Self-Assesment-Test zu erstellen ohne die dafür anderweitig nötigen Programmierkenntnisse zu besitzen. Das Programm verfügt über weitreichende Funktionen, damit ein vollständiger Test erstellt werden kann. In diesem Abschnitt werden diese Funktionen erläutert und eine Gesamtübersicht über diese Komponente gegeben.

4.1.1 Übersicht

Das Hauptfenster des 'Creators', zu sehen in Abbildung 4, beinhaltet fünf wichtige Elemente die zur Übersicht und Erstellung des Tests dienen. Auf der linken Seite ist ein 'TreeView'(A), der die Struktur des Testes darstellt. In ihm werden die Kategorien, deren zugehörigen Fragen, Antworten und die Feedbacks angezeigt. In jedem

Abbildung 3: UML Klassendiagramm des Testgenerators



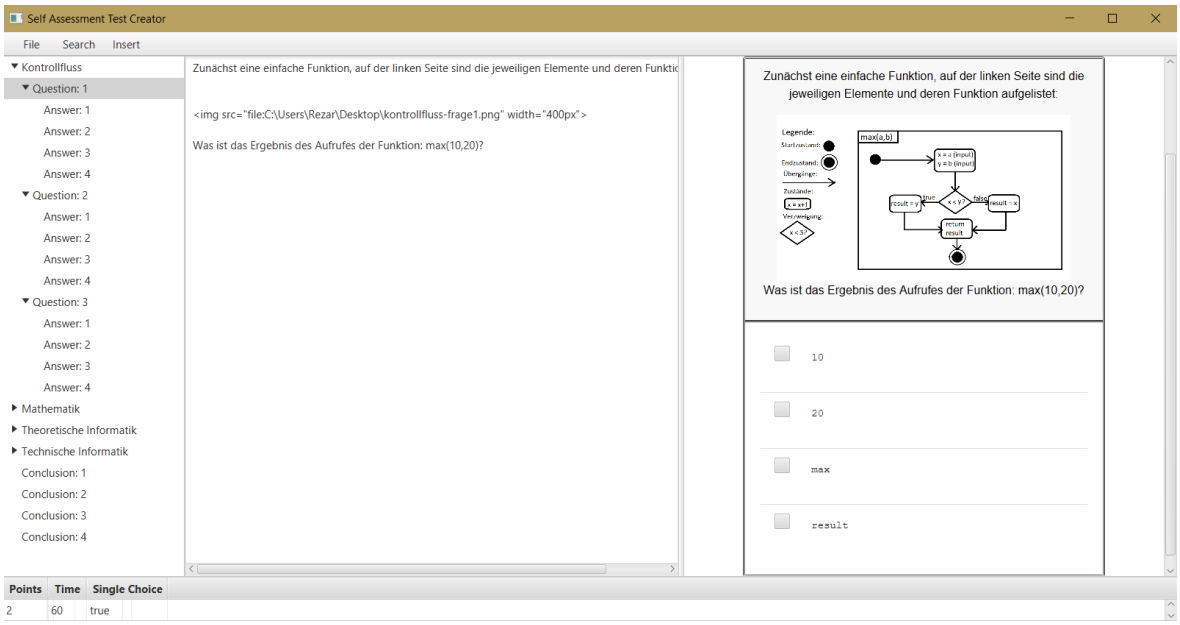


Abbildung 4:

'TreeItem' wird der Bezeichner des jeweiligen Objekts angezeigt.

Im Zentrum gibt es ein großes Textfeld (B), das den Inhalt des momentan ausgewählten 'TreeItems' wiedergibt. Im Textfeld kann dieser Inhalt mithilfe von HTML und Markdown editiert werden.

Im unteren Bereich (C) wird eine Tabelle erstellt, die alle Attribute des im 'TreeView' ausgewählten Elements anzeigt. Kategorien besitzen als Attribut ihren Namen. Fragen hingegen haben die Attribute Punkte und Zeit. Verschiedenen Feebacks können hier verschiedene Punktebereiche zugeordnet werden.

Die obere Leiste (D) enthält 'MenuItems' mit mehreren Funktionen, die im nächsten Abschnitt näher betrachtet werden.

Auf der rechten Seite gibt es ein großes Vorschaufenster (E). Dieses zeigt eine Vorschau der gerade ausgewählten Frage an.

4.1.2 Funktionen

Unter der Option 'File' gibt es neun Buttons, die nach Funktionen sortiert sind. Die ersten vier, also 'New Category', 'New Question', 'New Answer' und 'New Conclusion', erstellen die jeweils zugehörigen Objekte. 'Delete Item' entfernt ein ausgewähltes Objekt. Generate Website' erstellt einen Zip-Ordner, der die fertige Webseite enthält. Mit 'Import XML' und 'Export XML' kann der Ersteller den bisherigen Forschritt in einer XML-Datei speichern, beziehungsweise einen bereits erstellten Test laden. Das letzte 'MenuItem' 'Exit' beendet das Programm.

Die zweite Option 'Insert' ist dazu da, Medien in Form von Bildern oder Videos in das Textfeld einzufügen. Dabei wird ein Medium eingefügt, indem der Ersteller den zugehörigen Dateinamen angibt.

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 werden 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, Feedback) 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 Feedback 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 anderen Objekte beinhaltet.

Intern verwendet die Parser Komponente die 'Java Architecture for XML Binding (JAXB)' [3]. 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 Speichermedium, 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. Dabei findet 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. Diese 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 wieder 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[4] 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 Tamplate und das Template an sich übergeben. Velocity liest daraufhin dieses Template, sucht sich die Stellen heraus, die Variablen enthalten, und ersetzt diese mit den Inhalten des Java-Objekts.

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 '$' in dem Template signalisiert Velocity, dass der danach kommende Text für Velocity vorgesehen ist. So erkennt Velocity, dass '$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

[3] `https://javaee.github.io/jaxb-v2/`

[4] `http://velocity.apache.org/`

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>$question.getContent()</h1>
<ul>
#foreach($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 ein Template erstellt. Daraufhin werden die ausgewerteten Templates mit den anderen für die Webseite benötigten Dateien in ein ZIP-Archiv gepackt und an 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 Webseite

Im folgenden soll skizziert werden, wie die Webseite 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 Webseite zusammengestellt werden. Ein besonderer Schwerpunkt liegt dabei auf der Anzeige der Fragen 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 Elementen gefüllt ist
- den CSS-Dateien für das Styling der Seiten
- den JavaScript-Dateien zur Manipulation der HTML-Seite
- den Fragen in 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. 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 Fragenseite, zu sehen in Abbildung 5, von oben nach unten betrachtet und erläutert werden.

Die Seiten beginnen stets mit dem Logo der Universität Stuttgart. Es folgt eine Übersichtsleiste (A), 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 (B). 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

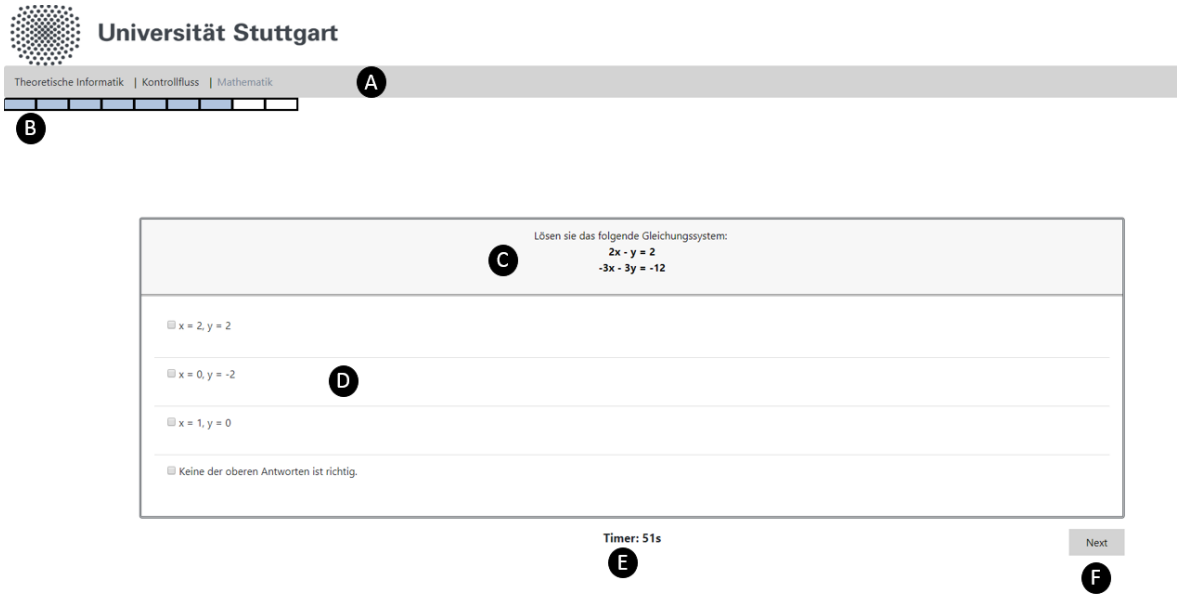


Abbildung 5: Die Anzeige einer Frage für einen Beispieltest.

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 (C), in welcher optional eine Erklärung zur Frage und ein Bild angezeigt werden können. Separiert von der Frage folgen dann ihre Antwortmöglichkeiten (D), die jeweils eigene Felder erhalten. Für alle Fragen wird links neben jeder Antwort eine Checkbox platziert. Hierbei können Antworten auch Bilder beinhalten, die innerhalb der Antwortfelder angezeigt werden. Wurde die optionale Zeitbeschränkung für die Frage aktiviert, so wird die verbleibende Zeit (E) mittig und unterhalb dem Feld der Frage und ihrer Antworten angezeigt.

Zuletzt findet sich der Next-Button (F) rechtsbündig unter dem Feld der Frage und ihrer Antworten, mit welchem die nächste Frage anzeigen lässt.

Es wird zunächst die nahezu leere HTML-Seite aufgerufen, welche als Hülle für die Seiten der Fragen dient. In der Folge wird der Inhalt der HMTL-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 das optionale Zeitlimit und die Punkte für die korrekte Beantwortung der Frage fest.

Die bisher angezeigten und beantworteten Fragen werden im Rahmen der Zustandsverwaltung mitverwaltet, die aktuelle Frage 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 aufweißt und wie die Frage durch den Nutzer beantwortet wurde. Die Anzahl der Antwortmöglichkeiten einer Frage wird 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 genau dann gesetzt ist, wenn die Antwortmöglichkeit angekreuzt wurde. 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 wird dann die Anzahl bisher beantworteter Fragen errechnet und anhand dessen die HTML und JSON-Datei der aktuellen Frage abgeleitet.

Beim initialen Aufruf der Webseite wird somit die erste Frage angezeigt, wobei 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. Dieses 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 in Abbildung 6 zu sehen ist.

Die Evaluationsseite setzt sich aus der Anzeige der erreichten Punkte für jede einzelne Kategorie (A) und einem Fazit abhängig von der Gesamtpunktzahl (B) zusammen. Für die einzelnen Kategorien wird anhand eines Fortschrittsbalkens angezeigt, wie viele der maximal erreichbaren Punkte der betroffenen Kategorie erreicht wurden.

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 überpürft in welchem Intervall die erreichte Gesamtpunktzahl liegt und das entsprechende Fazit im Dropdown-Feld unter den Fortschrittsbalken angezeigt (C).

5 Zusammenfassung und Ausblick

Im Folgenden soll der in dieser Arbeit entwickelte Ansatz und seine Komponenten zusammenfassend betrachtet werden. Abschließend werden einige mögiche Verbesserungs- und Erweiterungsmöglichkeiten präsentiert.

5.1 Zusammenfassung

Das Ziel dieser Arbeit war die Entwicklung einer Java-Applikation zur vereinfachten Erstellung von webbasierten Self-Assessment-Tests. Dazu wurden zunächst einige vewandte Arbeiten betrachtet und auf ihre Stärken und Schwächen bei der Erstellung von Self-Assessment-Tests untersucht. In der Folge wurden die Architektur des Systems und die Anforderungen an dieses bestimmt. Ausgehend von der resultierenden Klassenstruktur wurden dann die einzelnen Komponenten des Systems

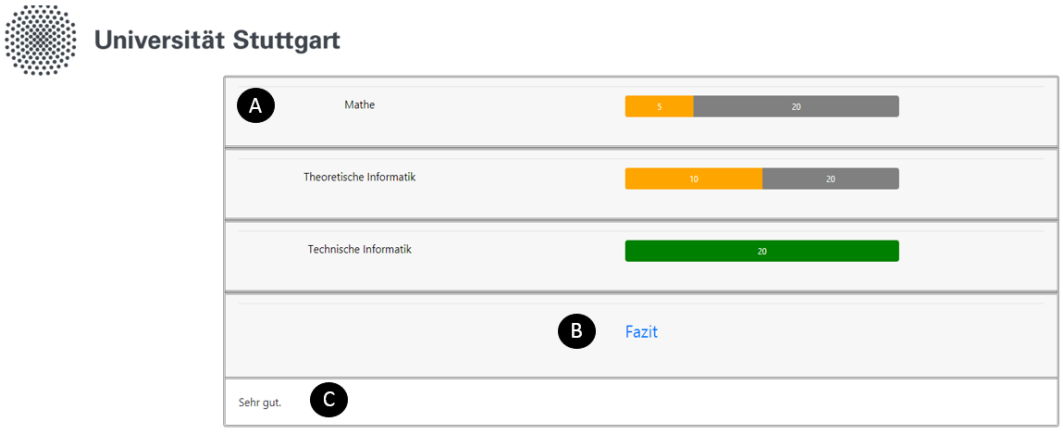


Abbildung 6: Die Evaluationsseite für einen Beispieltest mit drei Kategorien und maximal erreichbaren 60 Punkten.

implementiert. Der 'Creator' dient dabei als Benutzeroberfläche zur Erstellung des Tests. Mithilfe des des Parsers lassen sich bereits erstellte Tests effizient abspeichern und wiederverwenden. Zur Erstellung und Verwaltung der Webseite generiert der Generator dann die notwendigen statischen Ressourcen. Mittels eines Webservers werden diese Dateien schließlich veröffentlicht und clientseitig zur Anzeige der Fragen verwendet.

5.2 Ausblick

Zur Verbesserung der Benutzererfahrung kann der Ansatz um einige Funktionen ergänzt werden bestehende Funktionen weiter optimiert werden.

Künftig können das Verhalten und die Anzeige der Website weiter optimiert und individualisiert werden. Eine mögliche Erweiterung wäre die Hinzunahme der Option, beantwortete Fragen erneut besuchen zu können. Der Ersteller des Tests kann dann entscheiden, ob er diese Funktion aktivieren will oder nicht.

Die Anzeige der Fragen und ihrer Antworten könnte künftig interaktiv gestaltet werden, sodass diese beispielsweise als Dropdown-Feld realisiert werden. Denkbar wäre zudem das Einbinden von eigenen Stylesheets, um das Aussehen der Webseite nach belieben durch den Ersteller des Tests zu verändern.

Vor jeder Kategorie könnte eine Einstiegsseite angezeigt werden, sodass der Benutzer weiß, welche Art von Fragen ihn erwarten.