

# Software-Projekt 2 2014

VAK 03-BA-901.02

## Architekturbeschreibung



Patrick Hollatz	phollatz@tzi.de	2596537
Tobias Dellert	tode@tzi.de	2936941
Tim Ellhoff	tellhoff@tzi.de	2520913
Daniel Pupat	dpupat@tzi.de	2703053
Olga Miloevich	halfelv@tzi.de	2586817
Tim Wiechers	tim3@tzi.de	2925222

## Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>3</b>
1.1	Zweck . . . . .	3
1.2	Status . . . . .	4
1.3	Definitionen, Akronyme und Abkürzungen . . . . .	4
1.4	Referenzen . . . . .	4
1.5	Übersicht über das Dokument . . . . .	4
<b>2</b>	<b>Globale Analyse</b>	<b>5</b>
2.1	Einflussfaktoren . . . . .	5
2.1.1	Organisatorische Faktoren . . . . .	6
2.1.2	Technische Faktoren . . . . .	8
2.1.3	Produktfaktoren . . . . .	11
2.2	Probleme und Strategien . . . . .	14
<b>3</b>	<b>Konzeptionelle Sicht</b>	<b>22</b>
3.1	Überblick . . . . .	23
3.2	Serverkomponente . . . . .	25
3.3	Clientkomponente . . . . .	26
<b>4</b>	<b>Modulsicht</b>	<b>27</b>
<b>5</b>	<b>Datensicht</b>	<b>28</b>
<b>6</b>	<b>Ausführungssicht</b>	<b>28</b>
<b>7</b>	<b>Zusammenhänge zwischen Anwendungsfällen und Architektur</b>	<b>29</b>
<b>8</b>	<b>Evolution</b>	<b>29</b>

**Wichtiger Hinweis:** Diese Architekturbeschreibung wurde zu einem Teil aus verschiedenen Dokumententeilen der Architekturbeschreibung unserer Gruppenmitglieder aus dem Wintersemester 2013/14 erstellt (Gruppe *IT\_REVOLUTION*). Einige Teile wurden komplett übernommen, andere überarbeitet bzw. angepasst.

Diese Vereinbarung haben wir in der Kick-Off-Veranstaltung für RE SWP 2014 mit dem Veranstalter Dr. Karsten Hölscher getroffen.

## Version und Änderungsgeschichte

Version	Datum	Änderungen
1.0	28.06.2014	Einleitung
1.1	02.07.2014	Globale Analyse
1.2	02.07.2014	Konzeptionelle Sicht
1.3	03.07.2014	Modulsicht
1.4	03.07.2014	Datensicht
1.5	04.07.2014	Ausführungssicht
1.6	04.07.2014	Zusammenhänge zwischen Anwendungsfällen und Architektur
1.7	05.07.2014	Evolution

## 1 Einführung

*bearbeitet von: Tim Ellhoff*

### 1.1 Zweck

Dieses Dokument ist die Architekturbeschreibung der von uns zu entwickelnden Software. Sie dient der Kommunikation zwischen allen Interessenten. Dies ist unerlässlich für die Entwicklung des Systems, da die Entwickler der Architekturbeschreibung die Funktionalität einzelner Komponenten entnehmen. Sie dient der Aufteilung der Arbeit in unabhängig bearbeitbare Teile, besitzt anfangs einen hohen Abstraktionsgrad, der von vielen verstanden werden kann und wird in den Schichten weiter unten in diesem Dokument präziser ausgearbeitet. Die präzise Ausarbeitung der Architektur ist wichtig, um Möglichkeiten und Probleme der Entwicklung auszuloten und präventive Strategien und Maßnahmen zu entwickeln.

Die Architektur des Systems ist daher das Fundament unserer Implementierung, die direkt aus der Architektur resultiert.

## 1.2 Status

Dies ist der erste Architekturentwurf vom 06.07.2014.

## 1.3 Definitionen, Akronyme und Abkürzungen

## 1.4 Referenzen

- [https://elearning.uni-bremen.de/scm.php?cid=2b323f34b16a84e8dce31dcdfc0be6ad&show\\_scm=4c88951a202b2543c96de2c8a476d471](https://elearning.uni-bremen.de/scm.php?cid=2b323f34b16a84e8dce31dcdfc0be6ad&show_scm=4c88951a202b2543c96de2c8a476d471)  
Die Mindestanforderungen für die Quiz-App
- <http://www.elearning.uni-bremen.de> Plattform der Universität Bremen. Zugriff auf Folien der Veranstaltung Software Projekt 1 des Sommersemesters 2013 und Übungen des Software Projekts 2 des Wintersemesters 13/14 nur eingeschränkt möglich.
- Vorlage dieses Dokuments - Stud.IP - 3-Architekturbeschreibung-Vorlage.tex
- Hinweise zu diesem Dokument - Stud.IP 3-Hinweise-Abgabe-Architektur.pdf

## 1.5 Übersicht über das Dokument

Dieses Dokument basiert auf der Vorlage des IEEE P1471 2002 Standards. Der Inhalt dieses Dokuments ist wie folgt aufgegliedert:

**1. Einführung** Die Einführung beschreibt den Nutzen dieses Dokuments. Sie erläutert Definitionen, Akronyme und Abkürzungen und listet die benutzten Referenzen auf, sowie eine Übersicht über dieses Dokument.

**2. Globale Analyse** In diesem Abschnitt werden die relevanten Einflussfaktoren aufgezeigt und bewertet, sowie Strategien entwickelt, um Probleme bzw. interferierende Einflussfaktoren zu behandeln und auf diese entsprechend zu reagieren.

**3. Konzeptionelle Sicht** Die konzeptionelle Sicht zeigt grob die einzelnen Komponenten und deren Zusammenspiel des zu entwickelnden Systems auf. Dies geschieht auf einer hohen Abstraktionsebene und wird im weiteren Verlauf des Dokuments und den folgenden Sichten konkretisiert und verfeinert.

**4. Modulsicht** Im Abschnitt Modulsicht dieser Architekturbeschreibung geht es um eine tiefere Ebene der Abstraktion. Hier werden die Komponenten in einzelne Pakete

zerlegt und diese wiederum in Module, welche eine Einheit bilden, die ein Entwickler in einer Arbeitswoche implementieren kann.

**5. Datensicht** Die Datensicht beschreibt das zugrundeliegende Datenmodell und das Zusammenspiel der einzelnen Daten der Datenbank. Dies wird in Form eines erklärenden Textes und UML-Diagrammen realisiert.

**6. Ausführungssicht** Die Ausführungssicht zeigt im Prinzip das System in "Aktion", d.h. es zeigt auf, welche Prozesse laufen, welche Module hierfür gebraucht werden und wie diese zusammenspielen.

**7. Zusammenhänge zwischen Anwendungsfällen und Architektur** Hier werden die Zusammenhänge zwischen Architektur und den Anwendungsfällen der Anforderungsspezifikation beschrieben.

**8. Evolution** In diesem Teil der Architekturbeschreibung wird beschrieben, welche Änderungen vorgenommen werden müssen, wenn sich Anforderungen und oder Rahmenbedingungen ändern. Ein besonderes Augenmerk liegt hierbei auf die in der Anforderungsspezifikation unter "Ausblick" genannten Punkte.

## 2 Globale Analyse

*bearbeitet von: Olga Miloevich*

### 2.1 Einflussfaktoren

Die Einflussfaktoren werden im Folgenden unterteilt in:

- Organisatorische Faktoren
- Technische Faktoren
- Produktfaktoren

### 2.1.1 Organisatorische Faktoren

Tabelle 1: Organisatorische Faktoren

<b>O1</b>	<b>Time-To-Market</b>
<b>O2</b>	<b>Auslieferung von Produktfunktionen</b>
<b>O3</b>	<b>Budget</b>
<b>O4</b>	<b>Kenntnisse in Java, SQL und Android</b>
<b>O5</b>	<b>Kenntnisse in J-Unit</b>
<b>O6</b>	<b>Anzahl der Entwickler</b>

Tabelle 2: O1

<b>O1</b>	<b>Time-To-Market</b>
Faktor	Auslieferungsdatum 10.08.2014
Flexibilität und Veränderlichkeit	Die Deadline kann nicht verändert werden.
Auswirkungen	Die Software muss zum Abgabedatum lauffähig sein.

Tabelle 3: O2

<b>O2</b>	<b>Auslieferung von Produktfunktionen</b>
Faktor	Alle Mindestanforderungen
Flexibilität und Veränderlichkeit	Es müssen alle Mindestanforderungen erfüllt sein; sie sind jedoch vom Kunden oder beim Verlassen eines Gruppenmitglieds veränderbar.
Auswirkungen	Architektur muss alle Mindestanforderungen abdecken; es muss darauf geachtet werden, dass diese sich im Verlauf noch ändern.

Tabelle 4: O3

<b>O3</b>	<b>Budget</b>
Faktor	Kein finanzielles Budget
Flexibilität und Veränderlichkeit	Es werden keine finanziellen Unterstützungen für das Produkt gegeben.
Auswirkungen	Es können keine kostenpflichtigen Dienste in Anspruch genommen werden.

Tabelle 5: O4

O4	Kenntnisse in Java, SQL und Android
Faktor	Kenntnisse der Entwickler in Java, SQL und Android
Flexibilität und Veränderlichkeit	Kenntnisse sind nicht flexibel, es muss in Java programmiert werden und über Smartphone laufen. Die Server-Software muss mit der SQL-Datenbank kommunizieren. Die Kenntnisse können sich im Laufe ändern, z.B. durch neue Erfahrungen und neu erworbene Kenntnisse.
Auswirkungen	Bei wenig Kenntnissen muss mehr Zeit eingeplant werden, um sich diese anzueignen.

Tabelle 6: O5

O5	Kenntnisse in J-Unit
Faktor	Kenntnisse in J-Unit Tests
Flexibilität und Veränderlichkeit	Da Tests mit J-Unit gefordert werden, sind diese nicht verhandelbar oder flexibel.
Auswirkungen	Bei unzureichenden Tests kann es später beim Programm zu Problemen kommen, da Fehler spät oder gar nicht erkannt werden.

Tabelle 7: O6

O6	Anzahl der Entwickler
Faktor	Die Anzahl der Entwickler
Flexibilität und Veränderlichkeit	Es können keine neuen Gruppenmitglieder dazukommen, es können aber jederzeit Gruppenmitglieder wegfallen.
Auswirkungen	Wenn Gruppenmitglieder wegfallen, müssen die restlichen Mitglieder mehr Arbeit und mehr Zeit einplanen. Auch müssen Projektplan und Architektur neu angepasst werden.

### 2.1.2 Technische Faktoren

Tabelle 8: Technische Faktoren

<b>T0</b>	<b>Hardware des Kunden</b>
<b>T1</b>	<b>Software funktioniert unter Windows und Linux</b>
<b>T2</b>	<b>Software funktioniert als App(Android 2.3 oder höher)</b>
<b>T3</b>	<b>SQL-Datenbank</b>
<b>T4</b>	<b>Mehrere parallele Nutzer</b>
<b>T5</b>	<b>Client-Server System</b>
<b>T6</b>	<b>Benutzerschnittstelle</b>
<b>T7</b>	<b>Implementierungssprache Java</b>
<b>T8</b>	<b>Beschränkungsfreiheit für Fremdbibliotheken</b>
<b>T9</b>	<b>Testbarkeit</b>

Tabelle 9: T0

<b>T0</b>	<b>Hardware des Kunden</b>
Faktor	Die Hardware des Kunden stellt eine Beschränkungen dar.
Flexibilität und Veränderlichkeit	nicht flexibel. In dem entscheidenden Zeitraum werden keine Veränderungen stattfinden.
Auswirkungen	Es muss darauf geachtet werden, daß unsere Software die Hardware nicht zu sehr belastet.

Tabelle 10: T1

<b>T1</b>	<b>Software funktioniert unter Windows und Linux</b>
Faktor	Die Software muss auf den Betriebssystemen Windows und Linux laufen.
Flexibilität und Veränderlichkeit	nicht flexibel, da dies zu den Mindestanforderungen gehört. Veränderungen können jederzeit vom Kunden vorgenommen werden.
Auswirkungen	Die Entwickler müssen sich mit beiden Betriebsprogrammen befassen und sichergehen, dass es auf beiden funktioniert.



Tabelle 11: T2

<b>T2</b>	<b>Software funktioniert als App (Andriod 2.3 oder höher).</b>
Faktor	Die Software muss als Android App auf einem Smartphone laufen.
Flexibilität und Veränderlichkeit	nicht flexibel, da dies zu den Mindestanforderungen gehört. Veränderungen können jederzeit vom Kunden vorgenommen werden.
Auswirkungen	Die Software muss wie gefordert als App auf einem Android-Smartphone laufen.

Tabelle 12: T3

<b>T3</b>	<b>SQL-Datenbank</b>
Faktor	Software läuft über eine relationale Datenbank.
Flexibilität und Veränderlichkeit	Flexibel, jedoch muss eine Datenbank mit SQL oder SQL-ähnlichen Abfragen verwendet werden.
Auswirkungen	Es muss eine relationale Datenbank für die serverseitige Persistenz benutzt werden. Es muss eine Datenbank mit SQL oder SQL-ähnlichen abfragen verwendet werden.

Tabelle 13: T4

<b>T4</b>	<b>Mehrere parallele Nutzer</b>
Faktor	Es greifen mehrere Nutzer zur gleichen Zeit auf die Software zu.
Flexibilität und Veränderlichkeit	Es ist uns überlassen, wie viele Nutzer zur gleichen Zeit auf das System zugreifen dürfen.
Auswirkungen	Die Software muss darauf ausgelegt sein, mehrere Nutzer zur gleichen Zeit zu verwalten.

Tabelle 14: T5

<b>T5</b>	<b>Client-Server System</b>
Faktor	Die Software arbeitet über ein Client-Server System.
Flexibilität und Veränderlichkeit	Da wir übers Internet auf den Server zugreifen müssen, ist es notwendig, ein Server-Client System zu verwenden.
Auswirkungen	Die Implementierung wird in Server und Client aufgeteilt (siehe <a href="#">3</a> ). Übers Internet werden die Daten zwischen Server und Client ausgetauscht.

Tabelle 15: T6

<b>T6</b>	<b>Benutzerschnittstelle</b>
Faktor	Es sollte eine übersichtliche und ansprechende GUI geben.
Flexibilität und Veränderlichkeit	Die Gestaltung der GUI ist uns überlassen.
Auswirkungen	Für eine benutzerfreundliche Gestaltung sind Kenntnisse in LibGDX und in HTML5 notwendig.

Tabelle 16: T7

<b>T7</b>	<b>Implementierungssprache Java</b>
Faktor	Die Software muss in Java 6 oder höher geschrieben werden.
Flexibilität und Veränderlichkeit	Nicht flexibel, da dies zu den Mindestanforderungen gehört.
Auswirkungen	Die Software muss in Java geschrieben werden, daher müssen alle Entwickler diese Sprache beherrschen.

Tabelle 17: T8

<b>T8</b>	<b>Beschränkungsfreiheit für Fremdbibliotheken</b>
Faktor	Fremdbibliotheken dürfen für den Einsatz in Forschung und Lehre keine Beschränkungen aufweisen.
Flexibilität und Veränderlichkeit	Nicht flexibel, da dies zu den Mindestanforderungen gehört.
Auswirkungen	Es darf keine Software oder Bibliothek verwendet werden, die kostenpflichtig ist.

Tabelle 18: T9

<b>T9</b>	<b>Testbarkeit</b>
Faktor	Die Software muss auf Richtigkeit getestet werden.
Flexibilität und Veränderlichkeit	Geringer Einfluss auf Testumfang. Es sind keine Veränderungen zu erwarten.
Auswirkungen	Es muss darauf geachtet werden, daß die Blackbox-Tests implementierbar sind.

### 2.1.3 Produktfaktoren

Tabelle 19: Produktfaktoren

<b>P1</b>	<b>Funktionalität</b>
<b>P2</b>	<b>Performanz</b>
<b>P3</b>	<b>Benutzerfreundlichkeit</b>
<b>P4</b>	<b>Benutzerrechte</b>
<b>P5</b>	<b>Fehlererkennung</b>
<b>P6</b>	<b>Sicherheit</b>
<b>P7</b>	<b>Erweiterbarkeit und Bedienbarkeit</b>

Tabelle 20: P1

<b>P1</b>	<b>Funktionalität</b>
Faktor	Das Produkt muss alle Mindestanforderungen enthalten.
Flexibilität und Veränderlichkeit	Alle Anforderungen müssen zum Bestehen erfüllt werden. Die Anforderungen können vom Kunden oder Dozenten verändert werden oder die Anforderungen werden bei einem Austritt eines Mitglieds verringert.
Auswirkungen	Es müssen alle Mindestanforderungen implementiert werden. Hat allgemein großen Einfluss auf die Architektur.

Tabelle 21: P2

<b>P2</b>	<b>Performanz</b>
Faktor	Möglichst schnelle Ausführungszeiten
Flexibilität und Veränderlichkeit	Flexibel, da nichts davon in den Mindestanforderungen steht.
Auswirkungen	Es sollte bei der Implementierung auf einen schnellen Datenaustausch zwischen Server und Client geachtet werden.

Tabelle 22: P3

<b>P3</b>	<b>Benutzerfreundlichkeit</b>
Faktor	Das Produkt soll so einfach wie möglich und so schwer wie nötig zu bedienen sein.
Flexibilität und Veränderlichkeit	Es gibt keine Vorschriften, sodass wir eigenständig entscheiden können.
Auswirkungen	Hat Auswirkung auf den Bereich der Architektur, der direkt vom Benutzer verwendet wird.

Tabelle 23: P4

<b>P4</b>	<b>Benutzerrechte</b>
Faktor	Es gibt verschiedene Benutzer mit unterschiedlichen Rechten.
Flexibilität und Veränderlichkeit	Nicht flexibel, da dies vom Kunden gefordert wird.
Auswirkungen	Es müssen unterschiedliche Benutzer implementiert werden, die unterschiedliche Rechte haben und diese auch nicht überschreiten dürfen.

Tabelle 24: P5

<b>P5</b>	<b>Fehlererkennung</b>
Faktor	Fehler sollten von der Software erkannt werden und entsprechend behandelt werden.
Flexibilität und Veränderlichkeit	Flexibel, da dies nicht ausdrücklich vom Kunden gefordert wird.
Auswirkungen	Fehler müssen erkannt und durch entsprechende Exceptions korrigiert werden. Die Software sollte weiter laufen.

Tabelle 25: P6

<b>P6</b>	<b>Sicherheit</b>
Faktor	Stabiler Datenaustausch, mögliche SQL-Injektions sollen möglichst abgefangen werden.
Flexibilität und Veränderlichkeit	Flexibel, da dies nicht ausdrücklich vom Kunden gefordert wird.
Auswirkungen	Die Software soll sorgfältig auf Schwachstellen untersucht werden, dies ist zeitaufwendig. Erfordert erweiterte Kenntnisse in SQL und Sicherheitsfragen.

Tabelle 26: P7

<b>P7</b>	<b>Erweiterbarkeit und Bedienbarkeit</b>
Faktor	Es ist wünschenswert, dass unser Produkt sich leicht erweitern läßt.
Flexibilität und Veränderlichkeit	Flexibel, da dies nicht ausdrücklich vom Kunden gefordert wird. Man kann auf die Erweiterbarkeit verzichten, um an Abstraktion zu sparen.
Auswirkungen	Solange es kein zu großes Hinderniss darsteht, kann beim Entwurf darauf geachtet werden, dass die Software erweiterbar bleibt.

## 2.2 Probleme und Strategien

Tabelle 27: Probleme und Strategien 1

1 Zeitprobleme
Es gibt einen festgesetzten Abgabetermin, der eingehalten werden muss.
<b>Einflussfaktoren</b> <ul style="list-style-type: none"><li>• O1 Time-To-Market</li><li>• O2 Auslieferung von Produktfunktionen</li><li>• O4 Kenntnisse in Java, SQL und Android</li><li>• O5 Kenntnisse in J-Unit</li><li>• O6 Anzahl der Entwickler</li><li>• T9 Testbarkeit</li><li>• P1 Funktionalität</li><li>• P6 Sicherheit</li><li>• P7 Erweiterbarkeit und Bedienbarkeit</li></ul>
<b>Lösung</b> <ul style="list-style-type: none"><li>• Strategie 1: Modularisierung für paralleles Arbeiten Durch Modularisierung können mehrere Entwickler zur gleichen Zeit am Projekt arbeiten und die Module unabhängig voneinander implementieren. Diese werden dann später zusammengesetzt.</li><li>• Strategie 2: Bibliotheken Benutzen Es werden bereits vorhandene Java Bibliotheken verwendet; dies spart Zeit, da man dann nicht alles neu schreiben muss.</li><li>• Strategie 3: Fertige Lösungen verwenden Es können fertige Lösungen zum Identifizieren und Abfangen von SQL-Injektions eingesetzt werden, damit man Zeit sparen kann.</li></ul> <p>Es werden alle Strategien zusammen verwendet.</p>

Tabelle 28: Probleme und Strategien 2

2 Mangelnde Kenntnisse in Java
Es werden Vorkenntnisse in Java vorausgesetzt; ohne diese könnte es zu großen Problemen kommen, da ohne ausreichende Kenntnisse das Projekt nicht realisiert werden kann.
<b>Einflussfaktoren</b> <ul style="list-style-type: none"><li>• O1 Time-To-Market</li><li>• O2 Auslieferung von Produktfunktionen</li><li>• O4 Kenntnisse in Java und Android</li><li>• O6 Anzahl der Entwickler</li><li>• T1 Software funktioniert unter Windows und Linux</li><li>• T5 Client-Server System</li><li>• T7 Implementierungssprache Java</li><li>• P1 Funktionalität</li></ul>
<b>Lösung</b> <ul style="list-style-type: none"><li>• Strategie 1: Modularisierung Der Code wird in verschiedene Module aufgeteilt. Wenn ein Gruppenmitglied nicht genügend Kenntnisse besitzt, kann dieses Modul von einem anderen Mitglied neu erstellt werden und der inkompetente Entwickler kann keinen Schaden auf andere Module anrichten.</li><li>• Strategie 2: Aufteilen in Server und Client Die Implementierung wird unter den Entwicklern so aufgeteilt, dass ein Teil den Client und der andere Teil den Server macht; so müssen sich die Gruppenmitglieder nicht Kenntnisse in beiden Bereichen aneignen.</li></ul> <p>Es werden beide Strategien verwendet.</p>

Tabelle 29: Probleme und Strategien 3

3 Mangelnde Kenntnisse in Android
Es werden Kenntnisse in Android vorausgesetzt, da eine App entwickelt werden muss.
<b>Einflussfaktoren</b> <ul style="list-style-type: none"> <li>• O1 Time-To-Market</li> <li>• O2 Auslieferung von Produktfunktionen</li> <li>• O4 Kenntnisse in Java, SQL und Android</li> <li>• O6 Anzahl der Entwickler</li> <li>• T2 Software funktioniert als App(Android 2.3 oder höher)</li> <li>• T5 Client-Server System</li> <li>• T6 Benutzerschnittstelle</li> <li>• T7 Implementierungssprache Java</li> <li>• P1 Funktionalität</li> </ul>
<b>Lösung</b> <ul style="list-style-type: none"> <li>• Strategie 1: Modularisierung Der Code wird in verschiedene Module aufgeteilt. Wenn ein Gruppenmitglied nicht genügend Kenntnisse besitzt, kann dieses Modul von einem anderen Mitglied neu erstellt werden und der inkompetente Entwickler kann keinen Schaden auf andere Module anrichten.</li> <li>• Strategie 2: Bearbeitung von Gruppenmitgliedern mit Android-Erfahrung Wir werden die Implementierung einem Gruppenmitglied überlassen, das bereits Erfahrung mit Android hat. So müssen sich die anderen nicht in Android einarbeiten und können sich bei Fragen an eben diesen wenden.</li> </ul> <p>Es werden beide Strategien verfolgt; sollte das Gruppenmitglied mit Android zeitlich oder fachlich nicht klarkommen, wird sich ein weiteres Gruppenmitglied mit Android beschäftigen.</p>



Tabelle 30: Probleme und Strategien 4

4 Mangelnde Kenntnisse in Datenbanksystemen
Es werden Kenntnisse in Datenbanksystemen vorausgesetzt, da wir für die Bibliothek eine Datenbank verwenden. Dabei werden SQL- oder SQL-ähnliche Abfragen verwendet und entsprechende Kenntnisse verlangt.
<b>Einflussfaktoren</b> <ul style="list-style-type: none"><li>• O1 Time-To-Market</li><li>• O2 Auslieferung von Produktfunktionen</li><li>• O4 Kenntnisse in Java, SQL und Android</li><li>• O6 Anzahl der Entwickler</li><li>• T3 SQL-Datenbank</li><li>• T5 Client-Server System</li><li>• T7 Implementierungssprache Java</li></ul>
<b>Lösung</b> <ul style="list-style-type: none"><li>• Strategie 1: Bearbeitung von Gruppenmitgliedern mit Erfahrung in Datenbanksystemen Wir werden die Implementierung Gruppenmitgliedern überlassen, die bereits Erfahrung mit Datenbanksystemen haben. So müssen sich die anderen nicht in Datenbanksystemen einarbeiten und können sich bei Fragen an diese wenden.</li></ul>

Tabelle 31: Probleme und Strategien 5

5 Unzureichende Softwaretests
Es werden genügend Tests benötigt, welche Module und Komponenten testen, ob diese funktionieren.
<b>Einflussfaktoren</b> <ul style="list-style-type: none"> <li>• O1 Time-To-Market</li> <li>• O5 Kenntnisse in J-Unit</li> <li>• T7 Implementierungssprache Java</li> <li>• T9 Testbarkeit</li> <li>• P1 Funktionalität</li> <li>• P2 Performanz</li> <li>• P3 Benutzerrechte</li> <li>• P4 Fehlererkennung</li> <li>• P6 Sicherheit</li> </ul>
<b>Lösung</b> <ul style="list-style-type: none"> <li>• Strategie 1: Modularisierung            Es werden Tests für die jeweilig implementierten Module geschrieben. Ziel ist es, zu prüfen, ob diese ihren Zweck erfüllen und danach werden Module zusammen getestet.</li> <li>• Strategie 2: Gruppenaufteilung            Die Entwicklungsgruppe wird in die kleinere Gruppen aufgeteilt. Unterschiedliche Gruppen schreiben Tests für unterschiedliche Module. Diese Strategie läßt die Implementierung und die Tests von unterschiedlichen Menschen schreiben.</li> </ul>
Es werden beide Strategien von uns verwendet.

Tabelle 32: Probleme und Strategien 6

6 Ausfall eines Gruppenmitglieds
Es kann jederzeit ein Gruppenmitglied aus der Gruppe austreten oder durch Krankheit etc. für eine gewisse Zeit ausfallen.
<b>Einflussfaktoren</b> <ul style="list-style-type: none"><li>• O1 Time-To-Market</li><li>• O2 Auslieferung von Produktfunktionen</li><li>• O6 Anzahl der Entwickler</li><li>• P1 Funktionalität</li><li>• P7 Erweiterbarkeit und Bedienbarkeit</li></ul>
<b>Lösung</b> <ul style="list-style-type: none"><li>• Strategie 1: Modularisierung Der Code wird in verschiedene Module aufgeteilt, die von einem Entwickler bearbeitet werden. Wenn nun ein Entwickler ausfällt, kann ein Modul von einem anderen Entwickler übernommen werden.</li><li>• Strategie 2: Ziel auf Mindestanforderungen Es kann zum Not auf die Erweiterbarkeit weniger Wert gelegt werden, damit die ganze Kraft für Mindestanforderungen verwendet werden kann.</li></ul>
Es können beide Strategien zusammen verwendet werden.

Tabelle 33: Probleme und Strategien 7

7 Mehrere parallele Nutzer
Es greifen mehrere Nutzer zur gleichen Zeit auf das System zu, auf das der Server antworten muss. Dabei soll der Server die Daten nicht an alle Clients senden.
<b>Einflussfaktoren</b> <ul style="list-style-type: none"><li>• O1 Time-To-Market</li><li>• O3 Budget</li><li>• T0 Hardware des Kunden</li><li>• T3 SQL-Datenbank</li><li>• T4 Mehrere parallele Nutzer</li><li>• P1 Funktionalität</li><li>• P2 Performanz</li><li>• P3 Benutzerrechte</li></ul>
<b>Lösung</b> <ul style="list-style-type: none"><li>• Strategie 1: Threads Die Clients bekommen jeweils einen Thread. Somit können sie zeitgleich auf den Server zugreifen und bekommen nur ihre Daten zurück.</li></ul>

Tabelle 34: Probleme und Strategien 8

8 Performanz
Die Software sollte kurze Ausführungszeiten haben. Dabei ist zu beachten, dass die Software/App auch auf Geräten mit geringer Leistung schnell und problemlos läuft.
<b>Einflussfaktoren</b> <ul style="list-style-type: none"><li>• O1 Time-To-Market</li><li>• T0 Hardware des Kunden</li><li>• T1 Software funktioniert unter Windows und Linux</li><li>• T2 Software funktioniert als App(Android 2.3 oder höher)</li><li>• T3 SQL-Datenbank</li><li>• T4 Mehrere parallele Nutzer</li><li>• P1 Funktionalität</li><li>• P2 Performanz</li><li>• P3 Benutzerrechte</li></ul>
<b>Lösung</b> <ul style="list-style-type: none"><li>• Strategie 1: Code effizient schreiben Den Code effizient schreiben, damit die Software kurze Ausführungszeiten hat. Dabei werden wir die bekannte Algorithmen verwenden, die kleineren Zeit- und Speicheraufwand haben. Auch Hardwaremöglichkeiten des Kunden wird von uns in Anspruch genommen.</li></ul>

Tabelle 35: Probleme und Strategien 9

9 unterschiedliche Benutzerrechte
Die Software hat unterschiedliche Benutzer, welche unterschiedliche Rechte besitzen und diese müssen unterschieden werden.
<b>Einflussfaktoren</b> <ul style="list-style-type: none"><li>• O1 Time-To-Market</li><li>• T3 SQL-Datenbank</li><li>• T4 Mehrere parallele Nutzer</li><li>• P1 Funktionalität</li><li>• P3 Benutzerrechte</li><li>• P6 Sicherheit</li></ul>
<b>Lösung</b> <ul style="list-style-type: none"><li>• Strategie 1: Identifikation durch Group Id In der Datenbank wird eine Group Id eingefügt, die dann die verschiedenen Nutzer speichert. Über diese werden dann die verschiedenen Rechte geregelt.</li></ul>

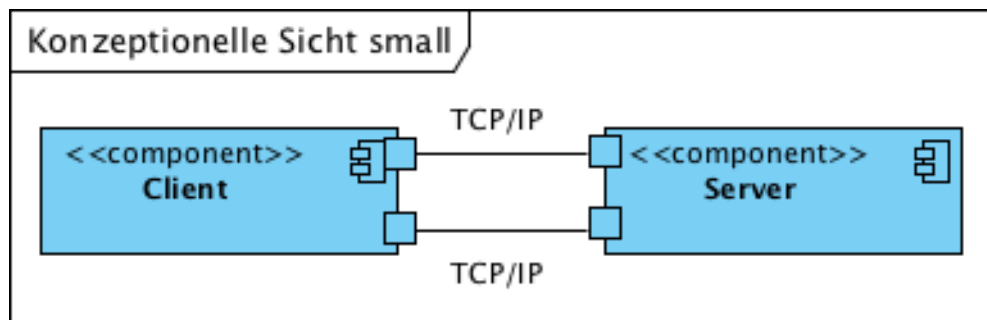
## 3 Konzeptionelle Sicht

*bearbeitet von: Tim Ellhoff*

Wir haben mithilfe von UML-Diagrammen die konzeptionelle Sicht realisiert. Im Folgenden werden die einzelnen Diagramme aufgezeigt und beschrieben und in nachfolgenden Sichten zusätzlich verfeinert und konkretisiert.

## 3.1 Überblick

Abbildung 1: Konzeptionelle Sicht (Klein)

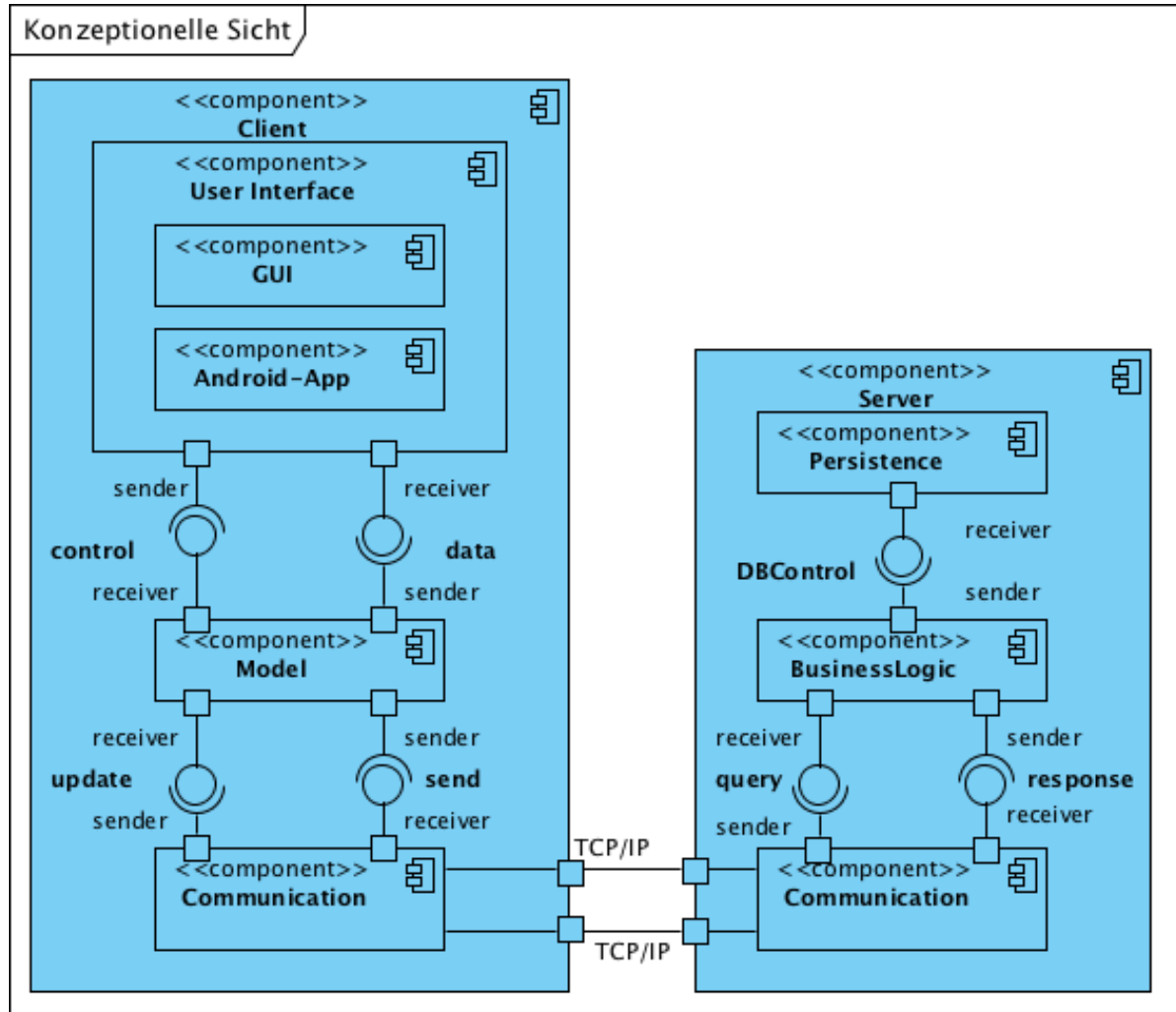


Unsere Architektur besteht aus zwei grundlegenden Komponenten, der Serverkomponente und der Clientkomponente (siehe Abb. 1 und Abb. 2 auf der nächsten Seite). Diese Komponenten beinhalten wiederum weitere Komponenten. Auf der einen Seite haben wir unsere Serverkomponente, die alle benötigten Daten für die Quizfragen, der Nutzer sowie z.B. Punkteständen usw. der App speichert.

Auf der anderen Seite, der Clientkomponente, muss zwischen zwei Komponenten unterschieden werden. Einmal der Komponente GUI-Client, welche sich in erster Linie an den Administrator, der über eine Webseite die Spiele konfigurieren und die Fragen und Antworten redaktionell bearbeiten kann, richtet und dann noch der mobile Android-Client, der sich ausschließlich an die Nutzer bzw. Spieler richtet.

Der GUI-Client stellt für die Verantwortlichen bzw. die Administratoren alle benötigten Funktionen bereit, um die App zu verwalten. Der Android-Client ermöglicht dem Spieler, die Quizapp zu spielen und Einstellungen vorzunehmen.

Abbildung 2: Konzeptionelle Sicht

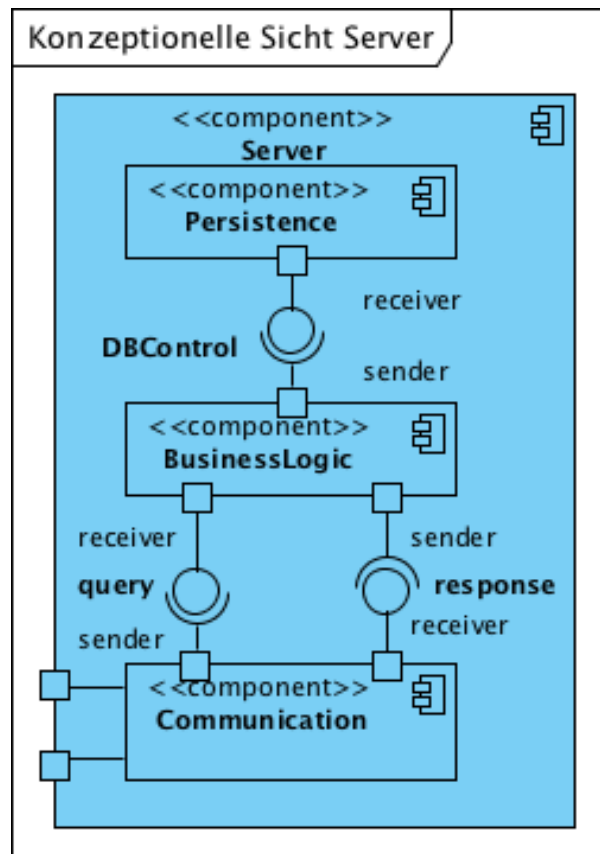


Als Architekturstil verwenden wir das Model-View-Controller-Pattern.



## 3.2 Serverkomponente

Abbildung 3: Konzeptionelle Sicht Server



Die Serverkomponente (siehe Abb. 3) besteht aus insgesamt drei Teilkomponenten, welche sich wie folgt aufgliedern:

- Communication

Die Komponente **Communication** nimmt Anfragen des Clients entgegen und leitet sie an die Komponente **BusinessLogic** weiter, wo die Anfragen verarbeitet werden und sendet die Ergebnisse zurück an den Client.

- BusinessLogic

Die Komponente **BusinessLogic** dient zum Verarbeiten der Anfragen und leitet diese verarbeiteten Anfragen dann an die Komponente **Persistence** weiter.

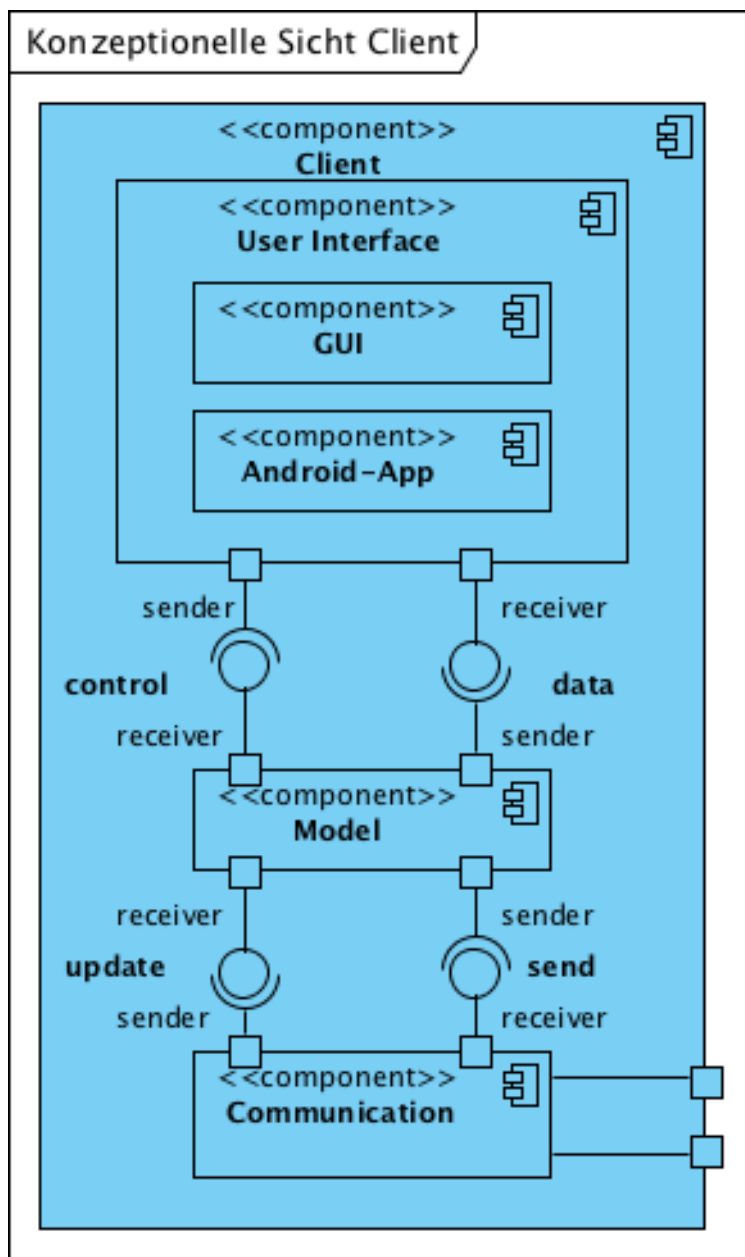
- Persistence

Die Komponente **Persistence** ist die Schnittstelle zur Datenbank. Über das Interface **DBControl** werden die verarbeiteten Anfragen von der Komponente **BusinessLogic** empfangen und in Datenbankabfragen umgewandelt, welche dann

von der Datenbank entgegen genommen werden.

### 3.3 Clientkomponente

Abbildung 4: Konzeptionelle Sicht Client



Die Clientkomponente (siehe Abb. 4) besteht so wie die Serverkomponente aus drei Teilkomponenten, welche sich wie folgt aufgliedern:

- Communication

Die Komponente **Communication** sendet Anfragen des Clients an den Server, welche dort verarbeitet werden und nimmt die Ergebnisse entgegen, um diese an die Komponente **Model** zu übergeben, wo die Ergebnisse der Anfrage weiter verarbeitet werden.

- Model

Die Komponente **Model** nimmt Ergebnisse von der Komponente **Communication** entgegen und schickt diese an die Komponente **User Interface**.

- User Interface

Die Komponente **User Interface** muss in zwei unterschiedliche Komponenten zerlegt werden:

- GUI

Die GUI richtet sich in erster Linie an den Administrator bzw. die Verantwortlichen und nimmt alle möglichen Aktionen des Administrators entgegen und schickt diese an die Komponente **Model**, um weiter verarbeitet zu werden. Sie bildet ebenfalls eine eigenständige Komponente.

- Android-App

Die Android-App richtet sich ausschließlich an die Nutzer bzw. Spieler der App und bietet ihnen die Funktionen, die in der Anforderungsspezifikation erarbeitet wurden. Zu diesen gehören z.B. das Registrieren beim Server sowie das Spielen von Fragerunden oder das Anzeigenlassen von Ranglisten oder sonstigen spielerelevanten Statistiken.

## 4 Modulsicht

*bearbeitet von: Tim Wiechers*

*Diese Sicht beschreibt den statischen Aufbau des Systems mit Hilfe von Modulen, Subsystemen, Schichten und Schnittstellen. Diese Sicht ist hierarchisch, d.h. Module werden in Teilmodule zerlegt. Die Zerlegung endet bei Modulen, die ein klar umrissenes Arbeitspaket für eine Person darstellen und in einer Kalenderwoche implementiert werden können. Die Modulbeschreibung der Blätter dieser Hierarchie muss genau genug und ausreichend sein, um das Modul implementieren zu können.*

*Die Modulsicht wird durch UML-Paket- und Klassendiagramme visualisiert.*

*Die Module werden durch ihre Schnittstellen beschrieben. Die Schnittstelle eines Moduls  $M$  ist die Menge aller Annahmen, die andere Module über  $M$  machen dürfen, bzw. jene Annahmen, die  $M$  über seine verwendeten Module macht (bzw. seine Umgebung, wozu auch Speicher, Laufzeit etc. gehören). Konkrete Implementierungen dieser Schnittstellen sind das Geheimnis des Moduls und können vom Programmierer festgelegt werden. Sie*

*sollen hier dementsprechend nicht beschrieben werden.*

*Die Diagramme der Modulsicht sollten die zur Schnittstelle gehörenden Methoden enthalten. Die Beschreibung der einzelnen Methoden (im Sinne der Schnittstellenbeschreibung) geschieht allerdings per Javadoc im zugehörigen Quelltext. Das bedeutet, dass Ihr für alle Eure Module Klassen, Interfaces und Pakete erstellt und sie mit den Methoden der Schnittstellen verseht. Natürlich noch ohne Methodenrümpfe bzw. mit minimalen Rümpfen. Dieses Vorgehen vereinfacht den Schnittstellenentwurf und stellt Konsistenz sicher.*

*Jeder Schnittstelle liegt ein Protokoll zugrunde. Das Protokoll beschreibt die Vor- und Nachbedingungen der Schnittstellenelemente. Dazu gehören die erlaubten Reihenfolgen, in denen Methoden der Schnittstelle aufgerufen werden dürfen, sowie Annahmen über Eingabeparameter und Zusicherungen über Ausgabeparameter. Das Protokoll von Modulen wird in der Modulsicht beschrieben. Dort, wo es sinnvoll ist, sollte es mit Hilfe von Zustands- oder Sequenzdiagrammen spezifiziert werden. Diese sind dann einzusetzen, wenn der Text allein kein ausreichendes Verständnis vermittelt (insbesondere bei komplexen oder nicht offensichtlichen Zusammenhängen).*

*Der Bezug zur konzeptionellen Sicht muss klar ersichtlich sein. Im Zweifel sollte er explizit erklärt werden. Auch für diese Sicht muss die Entstehung anhand der Strategien erläutert werden.*

## 5 Datensicht

*bearbeitet von: Tobias Dellert*

*Hier wird das der Anwendung zugrundeliegende Datenmodell beschrieben. Hierzu werden neben einem erläuternden Text auch ein oder mehrere UML-Klassendiagramme verwendet. Das hier beschriebene Datenmodell wird u.a. jenes der Anforderungsspezifikation enthalten, allerdings mit implementierungsspezifischen Änderungen und Erweiterungen. Siehe die gesonderten Hinweise.*

## 6 Ausführungssicht

*bearbeitet von: Daniel Papat*

*Die Ausführungssicht beschreibt das Laufzeitverhalten. Hier werden die Laufzeitelemente aufgeführt und beschrieben, welche Module sie zur Ausführung bringen. Ein Modul kann von mehreren Laufzeitelementen zur Laufzeit verwendet werden. Die Ausführungssicht beschreibt darüber hinaus, welche Laufzeitelemente spezifisch miteinander kommunizieren. Zudem wird bei verteilten Systemen (z.B. Client-Server-Systeme) dargestellt, welche Module von welchen Prozessen auf welchen Rechnern ausgeführt werden.*

## 7 Zusammenhänge zwischen Anwendungsfällen und Architektur

*bearbeitet von:*

*In diesem Abschnitt sollen Sequenzdiagramme mit Beschreibung(!) für zwei bis drei von Euch ausgewählte Anwendungsfälle erstellt werden. Ein Sequenzdiagramm beschreibt den Nachrichtenverkehr zwischen allen Modulen, die an der Realisierung des Anwendungsfalles beteiligt sind. Wählt die Anwendungsfälle so, dass nach Möglichkeit alle Module Eures entworfenen Systems in mindestens einem Sequenzdiagramm vorkommen. Falls Euch das nicht gelingt, versucht möglichst viele und die wichtigsten Module abzudecken.*

## 8 Evolution

*bearbeitet von: Tim Ellhoff*

In diesem Abschnitt geht es um mögliche Änderungen, Anpassungen bzw. Erweiterungen, die vorgenommen werden müssten, wenn sich Anforderungen des Systems ändern. Dabei ist wichtig, dass sich solche Änderungen möglichst modular realisieren lassen, ohne die bestehende Architektur komplett zu verändern, was sehr aufwändig und somit nicht wünschenswert wäre.

Im Folgenden werden einige wichtige mögliche neue Anforderungen bzw. Erweiterungen aufgelistet und deren jeweiligen zu implementierenden Änderungen an der Architektur beschrieben.

### Erweiterungsmöglichkeiten

Da sich aus der Anforderungsspezifikation im Abschnitt "Ausblick" noch keine genauen absehbaren Änderungen ergeben haben, werden im Folgenden potenzielle Änderungen aufgezeigt, die näher beschrieben werden.

#### 1. Erweiterung des GUI-Layouts

Es wäre möglicherweise wünschenswert, wenn man als Benutzer nicht nur ein GUI-Design in der Quiz-App verwenden könnte, sondern mehrere. Dafür müsste eine Funktionalität hinzugefügt werden, um zwischen verschiedenen Benutzeroberflächen wählen zu können (z.B. verschiedene Themes oder Farbwahlen).

Dazu müssten entsprechende Referenzierungen von neuen GUI-Style-Änderungen mit Bilddateien stattfinden sowie für Textanpassungen die XML-Dateien im entsprechendem Paket verändert bzw. erweitert werden.

## **2. Mehrsprachigkeit**

Auch wenn die Mindestanforderungen keine Mehrsprachigkeit für die Quizapp vorschreibt, wäre es ggf. wünschenswert, dass die Möglichkeit besteht, mehrere Sprachen für die Quiz-App zu unterstützen, um einen größeren bzw. internationaleren Spielerkreis anzusprechen. Insofern wäre es denkbar, dass neben Deutsch eine weitere Sprache eingebaut werden könnte. Englisch würde sich natürlich anbieten.