Universität Bremen
FB 3 – Informatik
Dr. Karsten Hölscher
TutorIn: Karsten Hölscher

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

Abgabe: 06.Juli 2014 — Version 1.7

Inhaltsverzeichnis

Wichtiger Hinweis: Diese Architekturbeschreibung wurde zu einem Teil aus verschiedenen Dokumententeilen der Architekturbeschreibung unserer Gruppenmitglieder aus dem Wintersemester 2013/14 erstellt (Gruppe *IT_R3V0LUTION*). 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 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

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

Tabelle 2: O1

O1	**Time-To-Market**
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

O2	**Auslieferung von Produktfunktionen**
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

O3	**Budget**
Faktor	Kein finanzielles Budget
Flexibilität und Veränderlichkeit	Es werden keine finanziellen Unterstützungen für das Produkt geben.
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

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

Tabelle 9: T0

T0	**Hardware des Kunden**
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

T1	**Software funktioniert unter Windows und Linux**
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

T2	**Software funktioniert als App (Andriod 2.3 oder höher).**
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

T3	**SQL-Datenbank**
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

T4	**Mehrere parallele Nutzer**
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

T5	**Client-Server System**
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 3). Übers Internet werden die Daten zwischen Server und Client ausgetauscht.

Tabelle 15: T6

T6	**Benutzerschnittstelle**
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

T7	**Implementierungssprache Java**
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

T8	**Beschränkungsfreiheit für Fremdbibliotheken**
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

T9	**Testbarkeit**
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

P1	**Funktionalität**
P2	**Performanz**
P3	**Benutzerfreundlichkeit**
P4	**Benutzerrechte**
P5	**Fehlererkennung**
P6	**Sichercheit**
P7	**Erweiterbarkeit und Bedienbarkeit**

Tabelle 20: P1

P1	**Funktionalität**
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

P2	**Performanz**
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

P3	**Benutzerfreundlichkeit**
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

P4	**Benutzerrechte**
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

P5	**Fehlererkennung**
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

P6	**Sicherheit**
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 Kentnisse in SQL und Sicherheitsfragen.

Tabelle 26: P7

P7	**Erweiterbarkeit und Bedienbarkeit**
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.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O4 Kenntnisse in Java, SQL und Android • O5 Kenntnisse in J-Unit • O6 Anzahl der Entwickler • T9 Testbarkeit • P1 Funktionalität • P6 Sicherheit • P7 Erweiterbarkeit und Bedienbarkeit
Lösung • 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. • Strategie 2: Bibliotheken Benutzen Es werden bereits vorhandene Java Bibliotheken verwendet; dies spart Zeit, da man dann nicht alles neu schreiben muss. • Strategie 3: Fertige Lösungen verwenden Es können fertige Lösungen zum Identifizieren und Abfangen von SQL-Injektions eingesetzt werden, damit man Zeit sparen kann. Es werden alle Strategien zusammen verwendet.

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.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O4 Kenntnisse in Java und Android • O6 Anzahl der Entwickler • T1 Software funktioniert unter Windows und Linux • T5 Client-Server System • T7 Implementierungssprache Java • P1 Funktionalität
Lösung • 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. • 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. Es werden beide Strategien verwendet.

Tabelle 29: Probleme und Strategien 3

3 Mangelnde Kenntnisse in Android
Es werden Kenntnisse in Android vorausgesetzt, da eine App entwickelt werden muss.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O4 Kenntnisse in Java, SQL und Android • O6 Anzahl der Entwickler • T2 Software funktioniert als App(Andriod 2.3 oder höher) • T5 Client-Server System • T6 Benutzerschnittstelle • T7 Implementierungssprache Java • P1 Funktionalität
Lösung • 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. • 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. Es werden beide Strategien verfolgt; sollte das Gruppenmitglied mit Android zeitlich oder fachlich nicht klarkommen, wird sich ein weiteres Gruppenmitglied mit Android beschäftigen.

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.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O4 Kentnisse in Java, SQL und Android • O6 Anzahl der Entwickler • T3 SQL-Datenbank • T5 Client-Server System • T7 Implementierungssprache Java
Lösung • 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.

Tabelle 31: Probleme und Strategien 5

5 Unzureichende Softwaretests
Es werden genügend Tests benötigt, welche Module und Komponenten testen, ob diese funktionieren.
Einflussfaktoren • O1 Time-To-Market • O5 Kenntnisse in J-Unit • T7 Implementierungssprache Java • T9 Testbarkeit • P1 Funktionalität • P2 Performanz • P3 Benutzerrechte • P4 Fehlererkennung • P6 Sicherheit
Lösung • 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. • 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.
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.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O6 Anzahl der Entwickler • P1 Funktionalität • P7 Erweiterbarkeit und Bedienbarkeit
Lösung • 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. • 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.
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.
Einflussfaktoren • O1 Time-To-Market • O3 Budget • T0 Hardware des Kunden • T3 SQL-Datenbank • T4 Mehrere parallele Nutzer • P1 Funktionalität • P2 Performanz • P3 Benutzerrechte
Lösung • Strategie 1: Threads Die Clients bekommen jeweils einen Thread. Somit können sie zeitgleich auf den Server zugreifen und bekommen nur ihre Daten zurück.

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.
Einflussfaktoren • O1 Time-To-Market • T0 Hardware des Kunden • T1 Software funktioniert unter Windows und Linux • T2 Software funktioniert als App(Android 2.3 oder höher) • T3 SQL-Datenbank • T4 Mehrere parallele Nutzer • P1 Funktionalität • P2 Performanz • P3 Benutzerrechte
Lösung • Strategie 1: Code effizient schreiben Den Code effizient schreiben, damit die Software kurze Ausführungszeiten hat. Dabei werden wir die bekannte Algoritmen verwenden, die kleineren Zeit- und Speicheraufwand haben. Auch Hardwaremöglichkeiten des Kunden wird von uns in Anspruch genommen.

Tabelle 35: Probleme und Strategien 9

9 unterschiedliche Benutzerrechte
Die Software hat unterschiedliche Benutzer, welche unterschiedliche Rechte besitzen und diese müssen unterschieden werden.
Einflussfaktoren • O1 Time-To-Market • T3 SQL-Datenbank • T4 Mehrere parallele Nutzer • P1 Funktionalität • P3 Benutzerrechte • P6 Sicherheit
Lösung • 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.

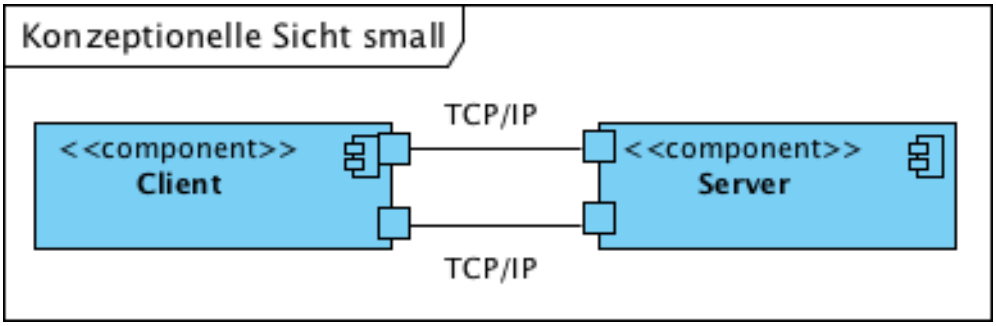
3 Konzeptionelle Sicht

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)

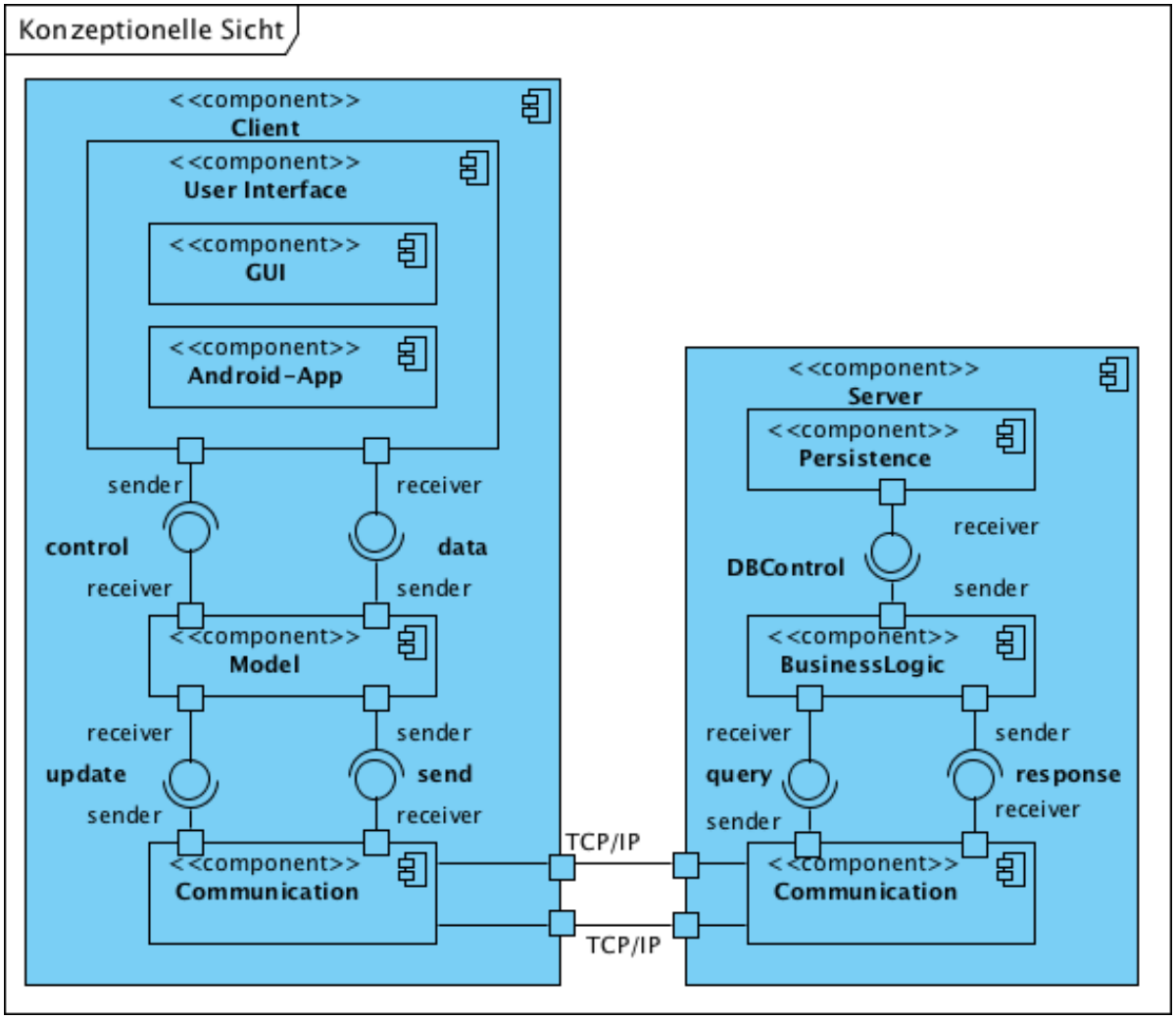


Unsere Architektur besteht aus zwei grundlegenden Komponenten, der Serverkomponente und der Clientkomponente (siehe Abb. 1 und Abb. 2 auf der nächsten Seite). Diese Komponenten beinhalten wiederum weitere Komponenten. Auf der einen Seite haben wir unsere Serverkomponente, die alle benötigten Daten für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

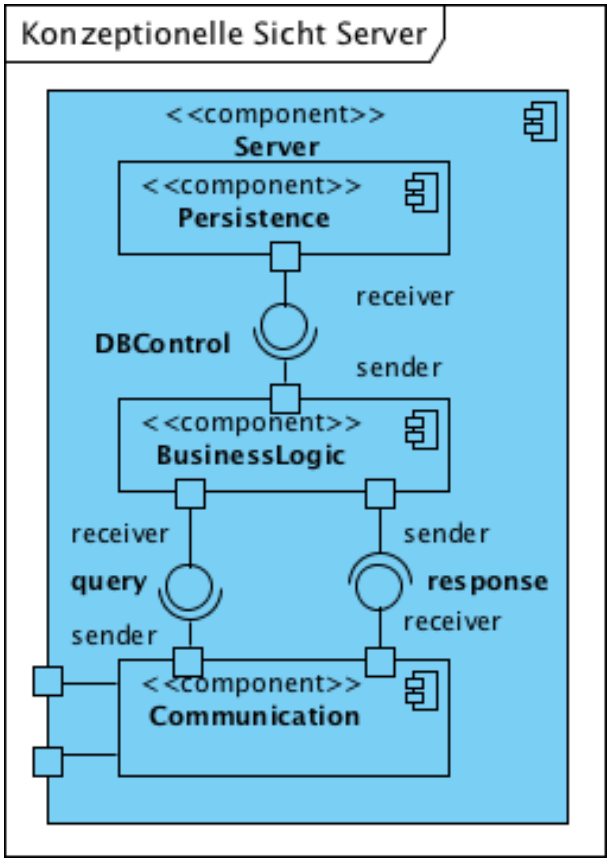
Abbildung 2: Konzeptionelle Sicht



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.

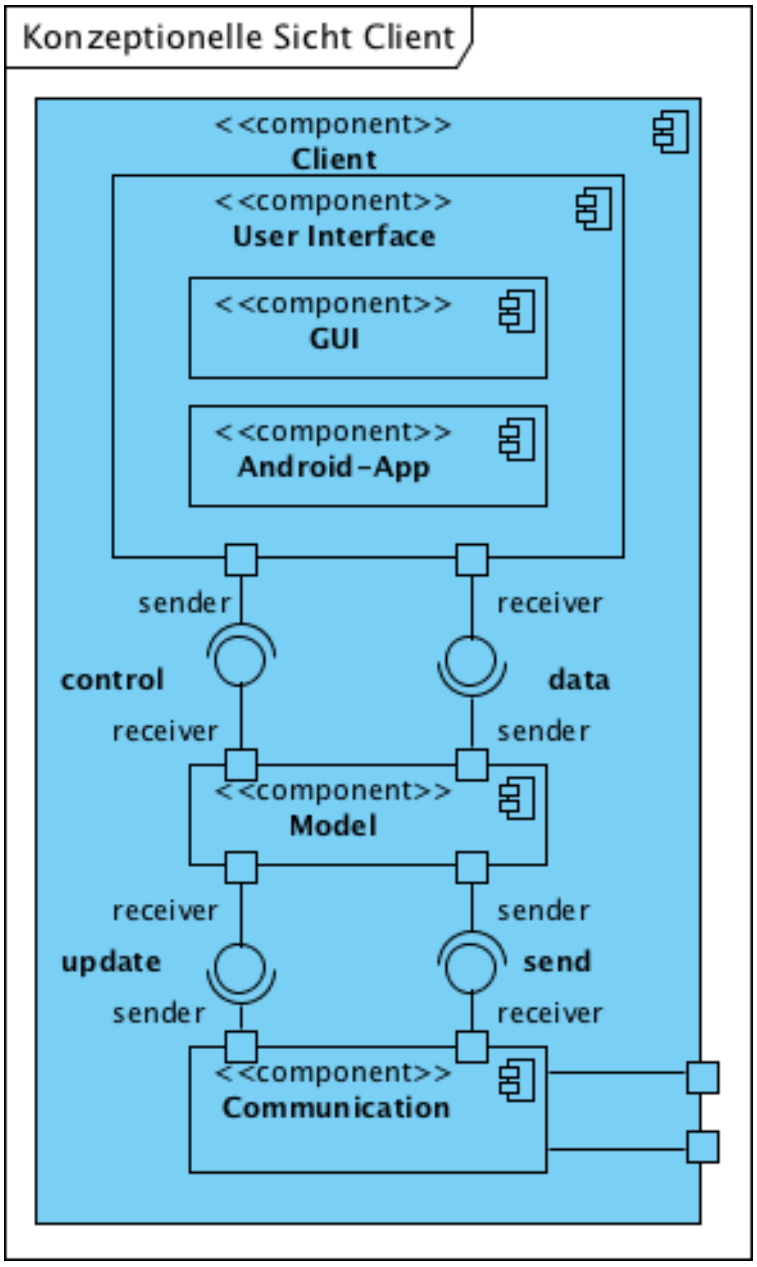
- Persistence

  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

4.1 Common

4.1.1 Entities

Die `Common`-Schicht stellt im Paket `Entities` Klassen bereit, die sowohl von Server als auch Client genutzt werden. Im Mindestfall umfasst dies alle für die Anwendung definierten Datentypen, auf die im nächsten Kapitel näher eingegangen wird.

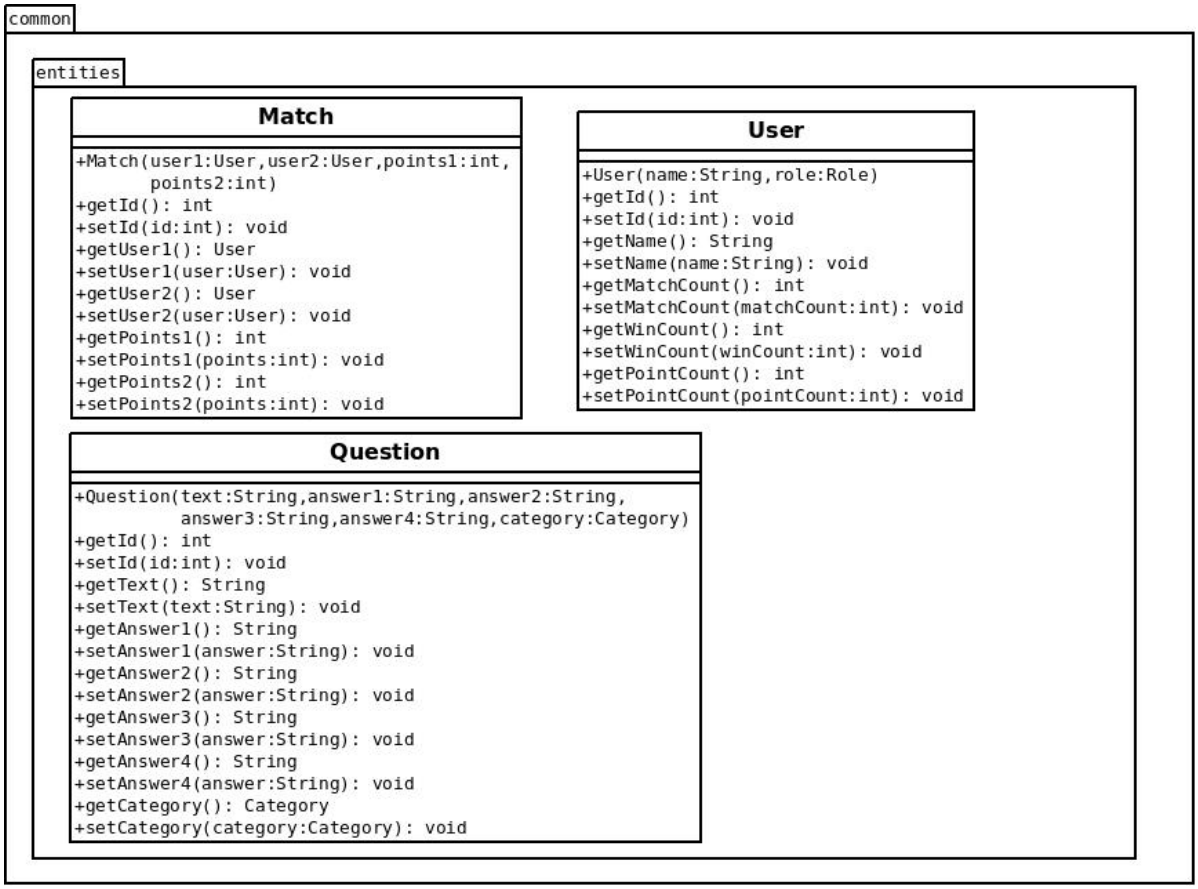


Abbildung 5: Entities

4.1.2 Net

Das `Net`-Paket enthält eine statische Helferklasse `NetUtils`, von der Zugangsdaten des Servers abgerufen werden können und in der das Lesen und Schreiben von Sockets implementiert ist. Die Nachrichten, die über die Sockets zwischen Server und Client geschickt werden, sind durch spezielle Klassen definiert, die zur Übertragung in JSON serialisiert und nach der Übertragung deserialisiert werden, um den Informationsaustausch so einfach wie möglich zu machen. Diese Klassen unterteilen sich in `Requests` und `Responses`:

Requests werden von einem Client an den Server geschickt, um Daten anzufordern oder mit einem anderen Client zu kommunizieren.

Responses werden vom Server an einen Client geschickt, um angeforderte Daten zu übertragen oder ihn über etwas zu benachrichtigen.

4.2 Server

4.2.1 Net

Im `Net`-Paket ist die Funktionsweise des Servers durch die Klassen `ServerDirectory` und `ServerInbox` definiert: Wird er gestartet, wird ein Thread geöffnet, auf dem eine `ServerDirectory` auf einem Port arbeitet. Die Klasse dient zur Kontaktaufnahme mit den Clients. Für jeden Client wird zudem eine `ServerInbox` eingerichtet, die - wie der Name schon sagt - als eine Art Postfach für den Server dient und ebenfalls auf einem eigenen Thread läuft. Dort wird für die gesamte Dauer der Verbindung auf Nachrichten des Clients gewartet und anschließend verarbeitet. Ebenfalls ist in der Klasse definiert, wie der Server ggf. auf diese Nachrichten antwortet.

4.2.2 Persistence-Modul

Die Persistence-Schnittstelle dient zum Zugriff auf die Datenbank.

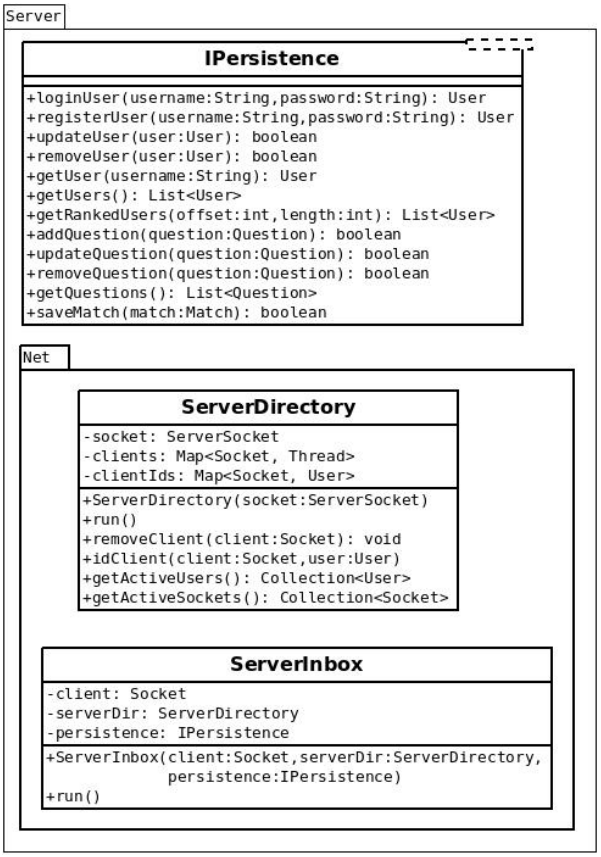


Abbildung 6: Server

4.3 App-Client

4.3.1 Client-Modul

Die Client-Schnittstelle ist durch die abstrakte Klasse `AbstractClient` gegeben und ist standardmäßig in der Lage, einen Socket zum Server zu öffnen. Analog zum Server wird auch hier ein Thread gestartet, auf dem eine `ClientInbox` läuft. Diese bearbeitet vom Server empfangene Nachrichten, indem die jeweilige Methode des Game-Moduls aufgerufen wird. letzteres wird durch das Modul implementiert.

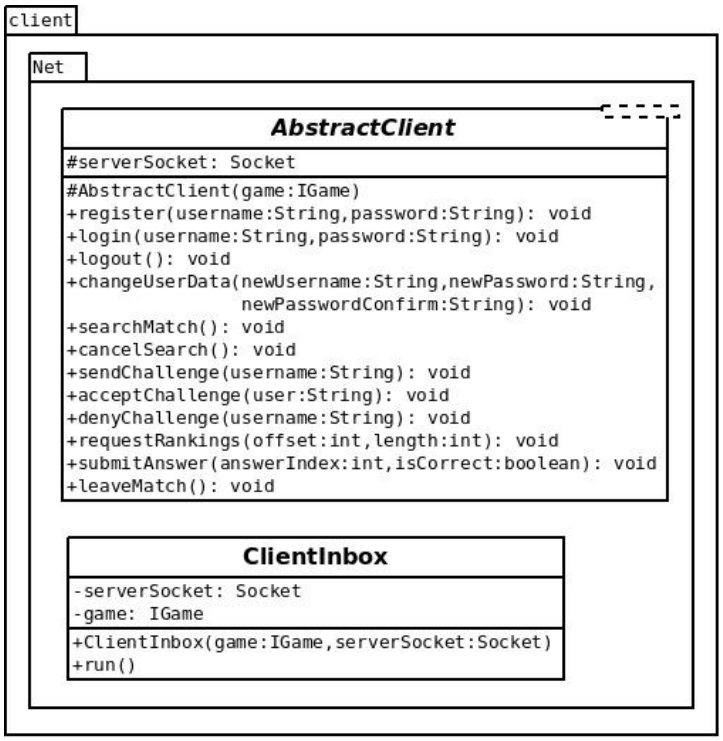


Abbildung 7: Client.Net

4.3.2 Game-Modul

Die Game-Schnittstelle definiert, welche Funktionalität die App bieten muss und beeinflusst dadurch essentiell Spielmechanik und Menüführung:

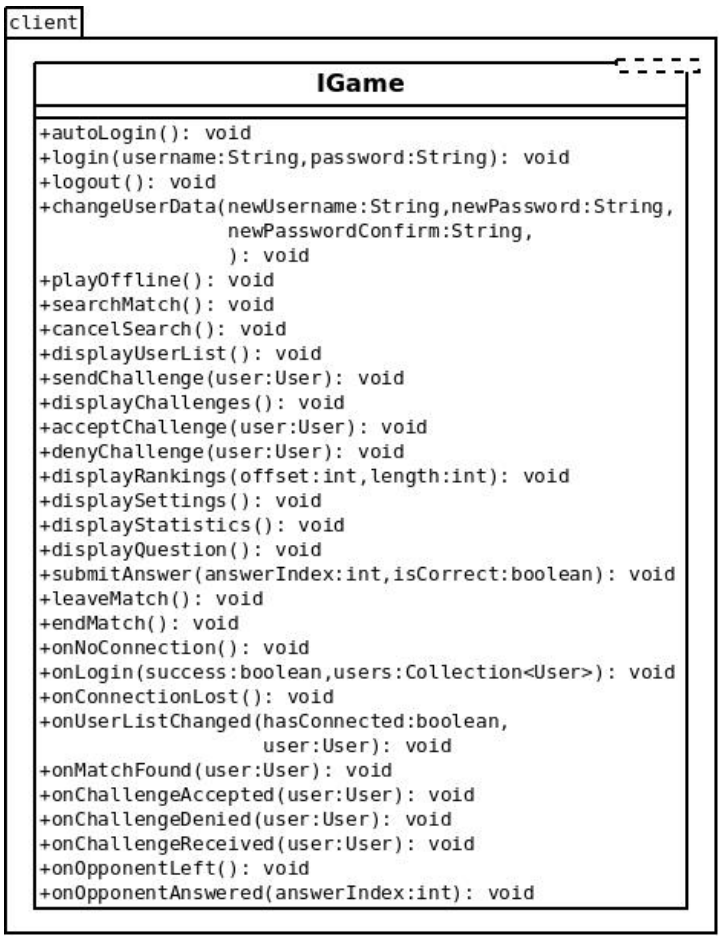


Abbildung 8: IGame

4.4 HTML-Client

Beim HTML-Client beschränkt sich die Kommunikation mit dem Server auf HTTP-Anfragen. Je nachdem, auf welche Adresse zugegriffen wird, werden intern Methoden ausgeführt, um die benötigten Daten darzustellen.

5 Datensicht

bearbeitet von: Tobias Dellert

Beschreibung der Datensicht

1. **Serversystem**

   Steht für das gesamte bereitgestellte System des Servers, zu dessen Aufgaben Kommunikation mit den Systemnutzern und das Speichern von Informationen gehört. Diese Klasse stellt keine Implementierungsklasse dar.

2. **Server**

   Stellt ServerSockets für die Clients bereit. Server ist ein Teil der Klasse Serversystem.

3. **IPersistence**

   Diese Klasse ist ein Interface und stellt Operationen für Data bereit.

4. **Data**

   Über die Klasse Data laufen die eigentlichen Prozesse der Datenspeicherung und Nutzerkommunikation. Es besitzt Methoden, um sowohl User, als auch Questions zu speichern oder zu verändern und den Ausgang abgeschlosser Spiele oder Matches zu speichern.

5. **Quizapp**

   Ähnlich wie die Klasse Serversystem, ist auch Quizapp keine implementierte Klasse, sondern steht für den ganzen Bereich der App und dessen Nutzer. In Quizapp befinden sich die Klassen Client und User und besitzt selbst eine Collection<Question> für alle Fragen, damit der Nutzer auch offline, also Serversystemunabhängig spielen kann.

6. **Client**

   Die Klasse Client stellt mittels eines ServerSockets eine Verbindung mit dem Serversystem durch die Bereitschaft der Sockets von der Klasse Server her.

7. **User**

   In der Klasse User befinden sich alle Informationen über den entsprechenden Nutzer, wie das Serversystem und vielleicht andere Nutzer es registrieren müssen oder wollen. User besitzt unter anderem das Attribut “role“, wodurch entschieden wird, ob es sich bei dem Nutzer um den Admin oder einen AppNutzer handelt.

8. **IGame**

   Diese Klasse ist ein Interface, welches Operationen für die Klasse QuizGame bereitstellt.

9. **Quizgame**

Diese Klasse implementiert IGame und enthält alle Operationen die für den Spiel-, Ein-/Auslog, Registrier- und Updatevorgang von seiten der App nötig sind. Daher ist diese Klasse ein Teil von sowohl Spielen, als auch Einloggen/..., beides Assoziationsklassen, welche für die eben genannten Vorgänge stehen.

10. **Match**

    Die Klasse Match beinhaltet alle Daten, die für den Ablauf eines "1gegen1-Spielsnnötig sind, wie unter anderem die beiden Namen der Spieler, sowie deren momentante Punktzahl usw.

11. **Question**

    Diese Klasse beinhalten alle Daten, die für jede Frage benötigt wird, wie ihren Text, ihre Kategorie und zur Auswahl stehenden Antowrten auf die Frage an sich.

12. **Spielen**

    Diese Assoziationsklasse steht für den Spielvorgang und benötigt daher die Klassen Question und Match.

13. **Einloggen/Ausloggen/Registrieren/Update**

    Diese Assoziationsklassen steht für die in dessen Namen stehenden Vorgänge und beschreibt damit eine der Relationen von Quizapp und Serversystem.

14. **Bearbeiten/Hinzufügen/Löschen**

    Diese Assoziationsklasse beschreibt die Relation der Klasse User in der Rolle des Admin und Serversystem. Der Admin soll Fragen und Nutzer berarbeiten, löschen oder hinzufügen können.

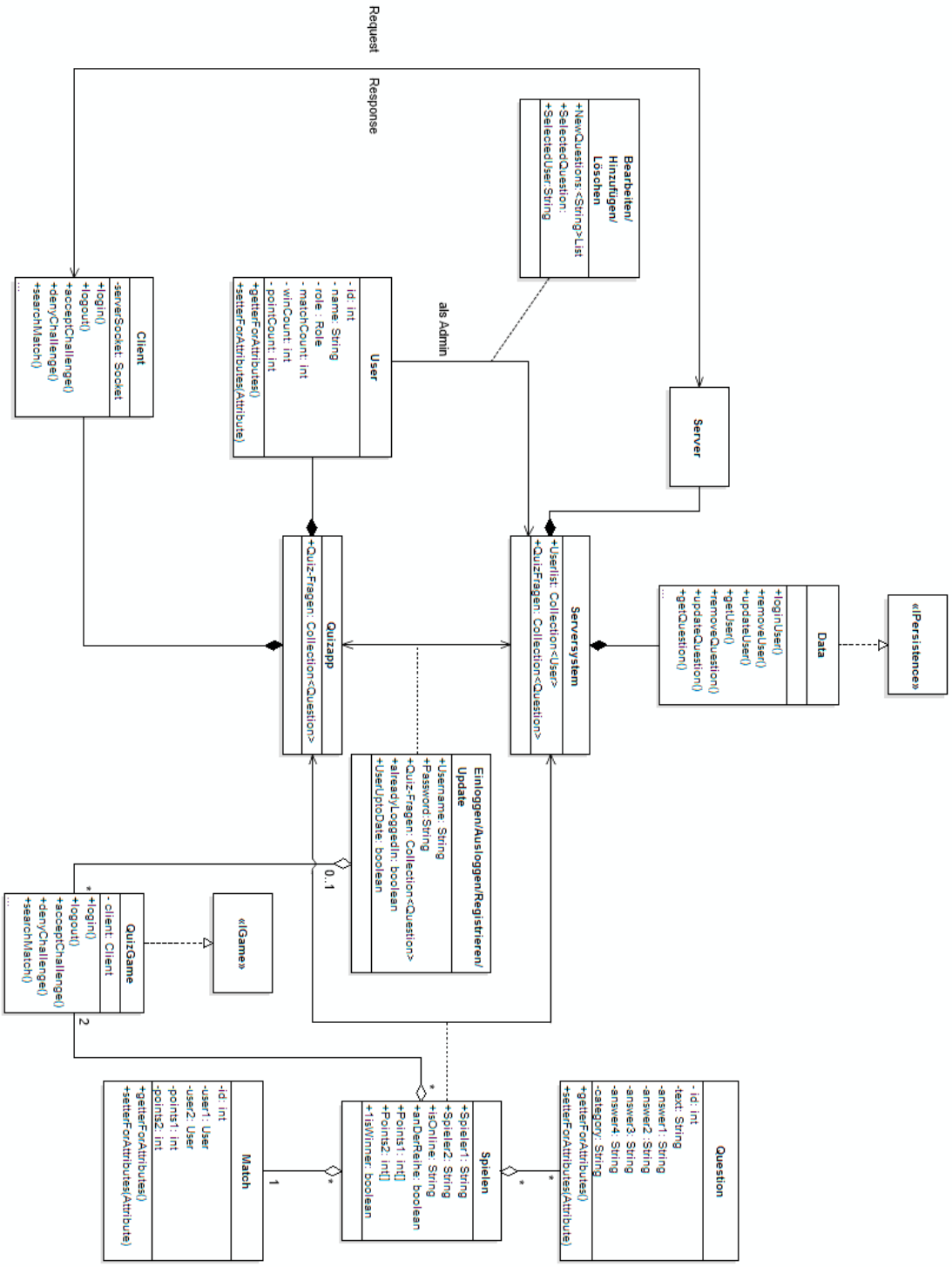


Abbildung 9: Datensicht

6 Ausführungssicht

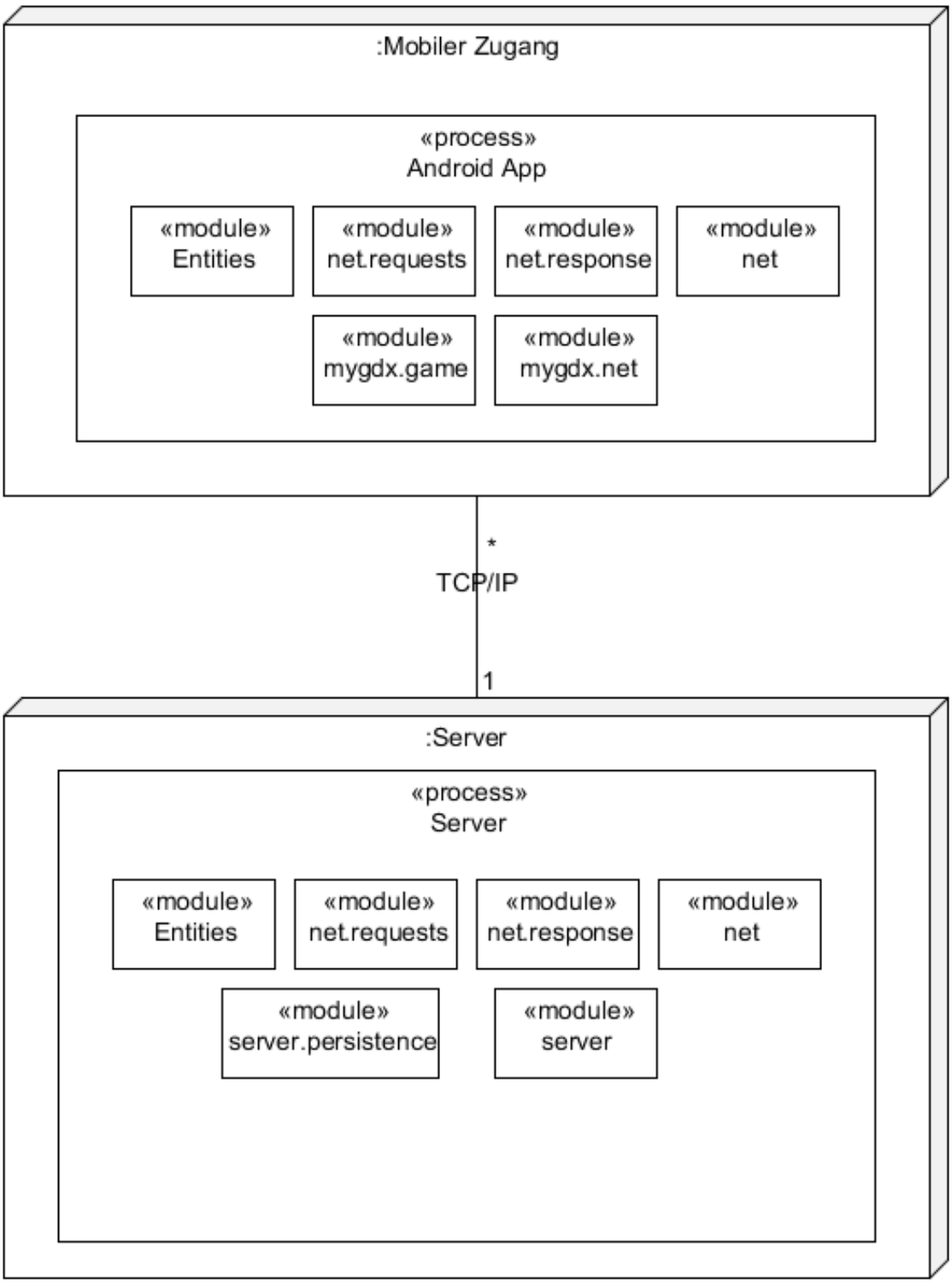
bearbeitet von: Daniel Pupat

Das folgende Diagramm 10 auf der nächsten Seite zeigt das Laufzeitverhalten der Software. Auf der einen Seite haben wir den mobilen Zugang, auf dem die Android App als Prozess läuft. Die App selbst verwendet die Module `net.requests`, `net.responses`, `net`, `mygdx.game` und `mygdx.net`. Von diesem mobilem Zugang kann eine TCP/IP-Verbindung zu dem Server aufgebaut werden. Dabei gibt es mehrere Verbindungen zu immer nur einem Server, daher die Multiplizitäten `*` und `1`.

Auf der anderen Seite nimmt nun der Server die TCP/IP-Verbindungen an. Er ist gleichzeitig Server und Datenbankserver. Der Server verwendet die Module `net.requests`, `net.responses`, `net`, `server.persistence` und `server`.

Das komplette System läuft mit zwei Prozessen: Einmal mit der Android App und dem anderen Prozess Server. Prinzipiell gibt es unendlich viele mobile Zugänge bzw. App-Prozesse, die auf einen Server zugreifen.

Abbildung 10: Ausführungssicht



7 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.