Game

1. Introduction and Goals

The ARZ wants to establish a new way of gamificated education for the members. ARZ means "Allgemeines Rechenzentrum" and it is an IT service provider with locations in Innsbruck and Vienna. Their homepage: https://www.arz.at/

The workers often have different knowledge about programming, so it is rather time consuming and costly to further educate them.

This is where the idea of "GAME" comes into play. The "GAME" is intended to be a platform on which employees can improve their programming knowledge with little games, quizzes or competitions.

A great focus lies on the terms "self-study" and "Gamification". Gamification means that the information is transmittetd through games to ease the learning process.

Furthermore there should be a possibility for trainers to bring in their own ideas, quizzes or modules. One of the goals of this diplome thesis is to motivate the people to continue their education in their free time. There should be a possibility to learn some skills without much effort.

Another important aspect of "GAME" is the scoring system, which will be implemented in the software. This will create transparency for employees and employers. This scoring system will be a great motivation for workers.

The most relevant Stakeholders

- Roland Moder
- Konrad Renner
- Michaela Würzl
- Cornelia Sammer

The expections of the Stakeholders should be fulfilled.

The expectations of the stakeholders regarding the thesis are:

- sufficient documentation regarding the arc42 template
- Fulfillment of the quality goals
- a clear and well-functioning software
- an easy-to-use user interface

2. Abstract (Deutsche Version)

Der Ziel der folgenden Diplomarbeit war, eine Lernplattform für Mitarbeiter einer Softwarefirma zu entwicklen. Anfangs haben wir die verschiedenen Arten wie Menschen lernen genauer betrachtet, und diese zu Lerntypen zusammengefasst. (Vester, Pask, Dunn,) Darüber sind wir auf das Thema "Wissensmanagement" näher eingegangen, und haben identifiziert, was Menschen mit ihrem Wissen anstellen. Diese Beiden Fachgebiete haben wir in unsere Website einfließen lassen, und diese so aufgebaut, dass sich Mitarbeiter Kurse anschauen können und mit dem daraus gewonnenen Wissen Quizze beantworten sollen. Das besondere hierbei ist, dass jeder "Spieler" Punkte und Embleme für richtig beantwortete Fragen bekommt. Dadurch können sich diese miteinander vergleichen und anspornen die meisten Punkte zu gewinnen. Somit ist ein spielerischer Faktor inkludiert, der dem Lernerfolg hilft.

3. Abstract (Englische Version)

In the following diploma thesis, we created a learning platform for employees of a software firm. In the beginning, we looked at the different ways to learn, which are summarized in learning types. Furthermore we looked into knowledge management and what people do with their knowledge. With these two factors in mind, we built a web-page, where employees are able to learn different programming-skills by taking courses and answering quizzes. The special thing about this tool is, that everyone is rewarded points and icons for correct answered questions. This way, employess can match and compare each other and so a playful factor is included.

4. Requirements and Overview

- create/modify a system to get a simple, easy to use education platform
- trainers should be able to upload their own tasks and games to the platform
- a variety of difficulty levels
- a point system which rewards the user and lets him compare his points with other users
- the points can also be used to rank up

5. Presentation of the project team

5.1. Eric Haneder

- Place of residence: Untergrub
- Born: 17.05.2001
- Education
  - Volkschule Göllersdorf
  - Hauptschule Göllersdorf
  - HTL Hollabrunn, Wirtschaftsingenieure-Betriebsinformatik

5.2. Jan Binder

- Place of residence: Sonnberg
- Born: 09.04.2001
- Education
  - Volksschule Hollabrunn
  - Bundesgymnasium Hollabrunn
  - HTL Hollabrunn, Wirtschaftsingenieure-Betriebsinformatik

6. Division of Tasks

Eric Haneder ist für die Funktion und Gestaltung der Website zuständig. Er ist dafür verantwortlich, dass die Homepage übersichtlich und leicht zu bedienen ist. Er programmiert die Website auf Basis von *Java Enterprise Edition, _PrimeFaces* und *HTML*, wie von der Firma gewünscht.

Jan Binder …

7. Qualitygoals

Table 1. Quality Goals

Name	Description
System ready and usability	The goal of this project is to create or modify a system to get a learning platform for the ARZ members. So the most important quality goal is the completion of the system because there cannot be any other features if the basic system does not exist. Also the system should be easy to use/ understand and should not have flaws in it.
The point system	One of the most important goals of the education platform is to implement a point system for the users. The point system rewards the ARZ members and the users should be able to compare their points with each other. This will boost the motivation of the learners to educate themselves even more, by for example trying to reach certain points.
Trainers can upload games and tasks	Another feature should be the possibility for selected trainers to upload their own tasks and games for the users of the education platform. This should be the foundation to never run out of new games and tasks.
Up to date documentation is released with every update of the system	The documentation of the whole project is another goal which is very important for the sucessfull completion. It should be without

Name	Description
	grammar or writing errors and should have an overall good looking format.

8. Stakeholders

- **Thesis supervisor** (contractor side)**:** Roland Moder roland.moder@aon.at[1]
- **Thesis supervisor** (client side)**:** Konrad Renner konrad.renner@arz.at[2]
- **Human resource development:** Michaela Würzl michaela.wuerzl@arz.at[3], Cornelia Sammer cornelia.sammer@arz.at[4]

Expectations:

Roland Moder:
He expects the diploma thesis to run smoothly and be completed within a reasonable time, this means it has to be completed by the end of April. Additionally, the diploma students should fulfill the requirements for a diploma thesis. Furthermore, there should be a sufficient documentation of the work done.

Konrad Renner:
He expects the software to contain a scoring system that records the learning progress of every member, which can be evaluated and compared. Furthermore, he expects to be able to mantain the tasks independently, this means if all tasks have been fulfilled on the platform, there should be an opportunity for coaches to be able to put in new tasks.

Michaela Würzl and Cornelia Sammer:
They expect a timely completion of the project, so that the concept is completed by 31.12.2019, and the implementation should be completed by the end of April. Moreover, the software should be utilizable and integrable, this means the user interface should be easy to use for employees, trainers and moderators.

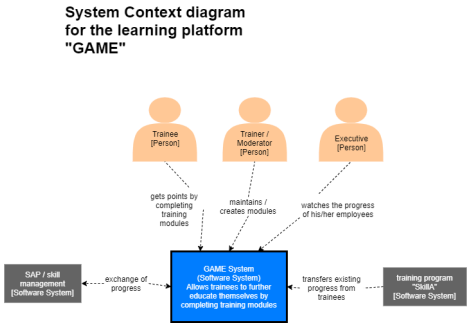
[1] mailto:roland.moder@aon.at
[2] mailto:konrad.renner@arz.at
[3] mailto:michaela.wuerzl@arz.at
[4] mailto:cornelia.sammer@arz.at

9. System, Scope and Context

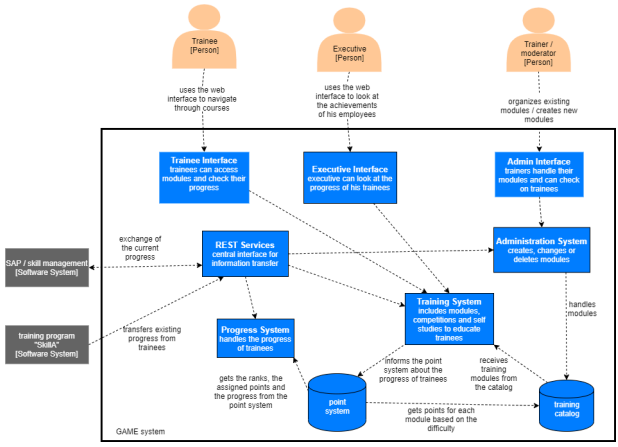
9.1. Technical Context

The technical context describes interfaces linking the system to its environment.

The abstract technical context is shown by the following C4-Context Model:



The GAME system is precisely described through this C4-Container Model:



9.2. Description

The game system communicates with the following systems:

- Skill / SAP employee management system
    - This system transmits the current progress of the employees to the GAME System and vice versa.

- training system "SkillA"
    - This system transfers progress from the "SkillA" training system to the GAME system.

9.3. Considerations

In the point system several difficulty levels are defined which reward the user with previously determined points.

The trainer has to give every module he created a certain difficulty level.

In the beginning it should be possible to manually enter the progresses of trainees from "SkillA" in the progress system.

10. Solution Strategy

10.1. Building from Scratch

The key decision of this project is the settlement to program the system from scratch, since there is no suitable, open-source platform which can be used for our purpose. Which systems were examined are written in Building from Scratch[5].

10.2. Programming Language

The programming language we will use throughout this project is the highly acknowledged Java programming language. For in depth view of technology decisions please have a look in Design Decisions.

[5] ../decisions/ADR001-BuildingFromScratch.adoc

11. Design Decisions

This chapter includes all decisions, the diploma team had to make. The following decisions are written in the Lightweight Architecture Decision Records[6] -format.

11.1. ADR001-BuildingFromScratch

Status

accpeted

Context

The ARZ wants to create a learning platform for their members. The project team evaluated possibly alternatives to adapt from. An alternative needs to be able to run on-premise, be up to date with todays standards and preferably be opensource. The evaluated systems are the following:

http://www.javaranch.com

- is based on "JForum" (a java forum)
- OpenSource
- Code is on SourceForge

```
-> doesn't meet the expectations
```

http://www.skill-guru.com

- no On-premise possible

```
-> therefore  excluded
```

http://www.betterprogrammer.com

- no further development, outdated (java version 1.5/1.6)

[6] https://adr.github.io/

```
-> therefore not effective
```

http://www.javapassion.com

- no possibility for extensions
- developed by an individual person

```
-> therefore no viable option
```

Decision

None of the evaluated systems meet the requirements, so the system will be build up from scratch by the project team.

Consequences

Development of a web application based on Java/Java Enterprise.

11.2. ADR002-Technology stack

Status

accpeted

Context

The GAME application consists of different subsystems. To make it possible, that these subsystems are maintainable in the context of the whole application, it is important that these subsystems are based on the same base technology stack.

Decision

The GAME application and its subsystems are developed with the help of the Java EE[7] technology stack. The Java EE technology is a proven technology which is standardised and offers therefore highest future-proofness.

Consequences

The GAME application uses Java EE in version 8 and is developed as application server neutral as possible. Version 8 is the latest production ready version of the plattform and therefore it is the version of choice for developing Java EE applications.

[7] https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

11.3. ADR003-Build and dependency managament

Status

accpeted

Context

The GAME application consists of different subsystems. To make it possible, that these subsystems can be automatically built and bundled a build and dependency management tool ist needed.

Decision

Apache Maven[8] will be used for build and dependency management. This tool is well-proven and can be extended with many plugins. Its convention-over-configuration paradigma lets a developer focus on creating the software.

[8] https://maven.apache.org/

11.4. ADR004-UI technology stack

Status

accpeted

Context

The primary user interface of the GAME application will be web based. Therefore a web UI Toolkit is needed for an efficient development of the UI and seamless integration with the backend systems.

Decision

As described in ADR002-Technology stack[9] Java EE is the base technology stack for the GAME application. The Java EE standard defines the JSF substandard as its UI Framework/Toolkit for web based applications. Therefore the web based UI of the GAME application will be developed with the JSF framework.

Consequences

For some parts of a modern web based UI, the JSF framework doesn't have components to develop the UI efficient. For these parts it is allowed to use components of the Primefaces[10] UI library.

[9] ADR002-Technologystack.adoc
[10] https://www.primefaces.org/

11.5. ADR005-Communication with distributed systems

Status

accpeted

Context

The GAME application must be able to communicate with other distributed systems in both directions (inbound and outbound).

Decision

The GAME application will provide resources for other distributed systems via RESTful webservices[11] .

By using a stateless protocol (HTTP[12]) and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow by reusing components that can be managed and updated without affecting the system as a whole, even while it is running. It is also evolving as the defacto standard for communication between distributed systems in the digital age.

Consequences

JAX-RS is the standard for creating RESTful Webservices in Java EE based applications and so it will be used in the GAME application.

[11] https://en.wikipedia.org/wiki/Representational_state_transfer
[12] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

11.6. ADR006-Database

Status

accpeted

Context

For persistent storage of data in the GAME application a database system is needed.

Decision

The PostgreSQL[13] database will be used as the default database system in the GAME application.

PostgreSQL comes with many features aimed to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help you manage your data no matter how big or small the dataset. In addition to being free and open source, PostgreSQL is highly extensible. It’s worth mentioning that PostgreSQL also supports many NoSQL features as well.

Consequences

PostgreSQL is defined as the default database of the application, but database accesses must be as SQL standard compliant as possible, so that the portability of the application is still warranted.

[13] https://www.postgresql.org/

11.7. ADR007-Java EE application server

Status

accpeted

Context

Applications which are implemented on top of the Java EE specification, which is defined as the base technology of the GAME application in the link:ADR002-Technology stack.adoc[ADR002-Technology stack], need an application server as runtime environment.

Decision

The Wildfly application server[14] will be used as the default application server in the GAME application.

The Wildfly application is open source and has full support for the Java EE 8 specification. It is also widely used in the ARZ and therefore the medium of choice.

Consequences

Wildfly is defined as the default application server of the application, but the application must be as Java EE standard compliant as possible, so that the portability of the application is still warranted. Further it must be possible to run the application also on application servers which are just Java EE Web Proile compliant.

[14] https://wildfly.org/

11.8. ADR008-Standard code formatter

Status

accpeted

Context

For seamless working together on a shared code base it is advantageous to use the same code formatter. So a standard code formatter must be defined.

Decision

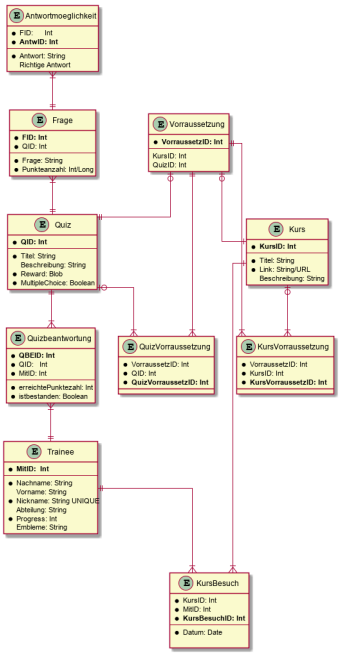
The standard code format is defined in the file formatter.xml. This XML is created as an eclipse[15]-Formatter file, but can also be importet in other IDEs, such as Apache Netbeans or IntelliJ Idea.

Consequences

Everyone who creates or changes source code for the GAME application in this repository, must use this formatter setup to format the source code.

[15] https://www.eclipse.org/

11.9. ER-Model



Declaration

The ER-Model in general is a diagram, which shows the relationships between the entities of a database.

12. Theoretische Aufgabenstellung: Eric Haneder

13. (Lernen,) Lerntypen, Lernstile, Lernpräferenzen und selbstreguliertes Lernen

13.1. Lernen

Everyone has to learn, both actively and passively. Whether at school or in professional life, further education is part of the essence of man. The term "learning" usually means the acquisition of knowledge, the development of skills or the practice of motor processes. However, "learning" also refers to activities that are carried out with the aim of changing internal conditions.

(vgl. Selbstreguliertes Lernen in der dualen Ausbildung - Lerntypen und Bedingungen, Johannes Rosendahl, 2010)

13.2. Physiological structures of learning

Regardless of the theoretical approach to learning, it is true that learning is linked to the ability to modify the central nervous system (CNS) (Birbaumer & Schmidt 1996, 2006). The central nervous system consists of the brain and spinal cord, whose impulse-transmitting cells (neurons) receive and process signals from the peripheral nervous system and generate and transmit signals for the regulation of organs such as muscles.

Despite the large amount of signals that are constantly received by the sense organs from the environment and transmitted to the CNS, the number of connections between the peripheral nervous system and the CNS is relatively small. Most of the estimated 100 billion neurons (Goldstein 1997, S. 11) serve to process information (pattern recognition, linking), they receive their input from cortical neurons and deliver their output to other cortical neurons. *(vgl. Selbstreguliertes Lernen in der dualen Ausbildung - Lerntypen und Bedingungen, Johannes Rosendahl, 2010, S.22)*

There are many people who have tried to classify different learning preferences and learning types. I will discuss these classifications in more detail below.

13.3. Learning type classification according to Vester

In the book "Denken, Lernen, Vergessen. Was geht in unserem Kopf vor, wie lernt das Gehirn und wann lässt es uns im Stich?" (1975, Stuttgart), Frederick Vester defined four types of learning. However, his book was interpreted differently, so that different versions of his classification can be found. At least on the fact that there is an auditory, a visual and a haptic learning type, there is agreement. Though there are some people who think that the fourth learning type after Frederik Vester is the communicative learning type and others think that it is an intellectual learning type.

One also comes across terms like "cognitive learning type" # "intellectual learning type", or the "kinesthetic learning type" # "haptic learning type.

Evidence for a classification according to Vester with a "communicative learning type".

```
https://www.teko.ch/die-vier-lerntypen-und-ihre-besonderheiten;
https://www.mystipendium.de/studium/lerntypen;
https://karrierebibel.de/lerntypentest/#Die-vier-Lerntypen-angelehnt-an-
Frederic-Vester;
https://learnsolution.de/die-vier-lerntypen-nach-frederic-vester/;
```

Evidence for a classification to Vester with an "intellectual learning type"".

```
http://www.rechtschreibwerkstatt-konzept.de/wp-content/uploads/2015/02/
Looss_Lerntypen.pdf;
https://wb-web.de/wissen/lehren-lernen/lernstile-und-lerntypen.html;
https://smarter-learning.de/lerntypen/klassische-lerntypen-nach-vester/;
https://wissenschafts-thurm.de/lerntypen/;
"Erforschung der Lerntypen und -strategien am Beispiel einer
 Handelsschulklasse", Petra Hochleitner, 2016
```

I will be analyzing this classification according to Vester:

1. optical/visual → Learning through seeing and observing
2. auditory/acoustically → Learning through listening and speaking
3. haptisch → Learning through action (touching and feeling)
4. cognitive → Learning through recognition/intellect

Auditory learning type

The auditory learning type learns while using his hearing. He/She has no problem listening to someone over long periods of time. The auditive type prefers auditioned learning content and learns better if he/she reads the text aloud himself/herself.

Visual learning type

This learning type learns the best, when he takes up the learning content through his eyes. Reading texts will lead to great learning achievements. This gets a better unterstanding of facts through looking at pictures.

Haptic learning type

The haptic learning type achieves the best learning succes if he cant feel the information he has to learn. "Learning by Doing" describes this lerning method pretty good. He prefers to be actively integrated into the learning process. Practical demonstrations are helpful too.

Cognitive/Intellectual learning type

This learning type understands and saves information the best, by thinking about and critically examining the information. The perception channel not important for taking up learning content.

(vgl."https://smarter-learning.de/lerntypen/klassische-lerntypen-nach-vester/"; letzter Zugriff: 14.01.2020)

Connections

Most of the time, it is not possible to assign someone a certain learning type, as there mix-types of learning. There are some people who learn faster by reading through text and writing it down afterwards (visual & haptic), and others are better of learning by reading and reciting out lout (auditory & haptic).

So if you want to learn effectively, you should try to appeal to more than one sense, to find a combination that fits your style.

How are different types of learning formed?

It would be a lot easier, if everybody could learn the same way. In reality, there a many people with different learning types, this is due to cnahges in personal characteristics, habits and previous experiences.

So how can you find out which learning type suits you best?

On the Internet you can find many so-called "learning type tests". In these tests, u have to answer a few simple questions, that depict typical life situations. Here are two hyperlinks, which lead to easy tests: * http://arbeitsblaetter.stangl-taller.at/TEST/HALB/Test.shtml * http://www.philognosie.net/index.php/tests/testsview/150/

After completing one of these tests, you will receive a recommendation as to which type of learner is more suitable for you.

It should be noted that different tests can also lead to different recommendations. This is due to the fact that the tests have different focuses and are more or less comprehensive.

If you do not want to take such tests, you can just look up different well-approved learning methods and try them yourself. Check how good you can remember something while using all of the different learning methods either alone or combined.

(vgl. "Erforschung der Lerntypen und -strategien am Beispiel einer Handelsschulklasse", Petra Hochleitner, 2016)

13.4. Learning preferences according to Dunn

Dunn and Price (1989) defined learning style as a typical way of learning that is influenced by different elements of the environment. This regards:

- physical stimuli (light, sound, temperature, design)
- social stimuli (pairs, peers, adults, groups)
- stimuli of learning material (auditiv, visuell, taktil, kinästhetisch)
- emotional stimuli (responsibility, persistence, motivation, disciplin)

These factors are measured by the "Learning Styles Inventory". However, this model takes little account of the actual cognitive processes that play a role in learning.

(vgl. "Lernorientierungen, Lernstile, Lerntypen und kognitive Stile", Ulrike Creß, in "Handbuch Lernstrategien" von Heinz Mandl & Helmut Felix Friedrich, S.373)

13.5. Learning styles according to Pask

Around 1972, Pask and Scott identified two opposing learning strategies used in problem-solving tasks where people had to search for information independently.They described the consistend usage of these strategies as a learning style. The holistic strategy means that learners always keep the big picture in mind and only turn to detailed questions in a second step. If this strategy is applied consistently, Pask speaks of the learning style of comprehension learning. On the other hand, learners with a serial strategy work their way step by step through the learning material and primarily turn to individual questions. If this strategy is used consistently, Park speaks of operation learning. Both strategies can lead to the same success. In their extreme form, however, both have a negative effect on performance, which is why Pask assigns both learning styles to corresponding learning pathologies. *Globetrotting* refers to the learning pathology of extreme comprehension learning, in which learners make inadmissible generalizations without the corresponding individual analysis. *Improvidence* describes the extreme form of operation learning, in which people lose themselves in details without being able to connect them to a big picture. Since the differences between holistic and serial approaches affect not only learning behaviour but the entire way in which information is sought and processed, they are often interpreted as cognitive styles.

(vgl."Lernorientierungen, Lernstile, Lerntypen und kognitive Stile", Ulrike Creß, in "Handbuch Lernstrategien" von Heinz Mandl & Helmut Felix Friedrich, S.369)

13.6. Learning styles according to Kolb

In 1984 David Kolb took a completely different approach to classifying learning types. According to Kolb, the learning process is based on two orthogonal bipolar dimensions. The first dimension depicts how people perceive and collect information. Persons can perceive via the senses through practical experience

or through abstract comprehension. The second dimension represents the way information is processed. It ranges from active trying to mental observation. (orthogonal → two straight lines are called orthogonal if they enclose a 90 degree angle) The following figure shows the dimensions:

Quelle (https://selfdirectedlearning.webnode.es/learning-styles-by-kolb/; letzter Zugriff 28.01.2020)

Kolb presents four learning styles defined by the four quadrants that result from these orthogonal dimensions.

Convergers explore their environment through active probing and process information in an abstract way. They are therefore interested in testing their theories and solving problems deductively.

Divergers combine mental observation with practical experience. This often leads them to creative solutions.

Assimilators connect abstract comprehension with mental observation. They are therefore mainly interested in developing abstract theories and defining problems, less in solving concrete problems.

Accomodators combine active experimentation with concrete experience. They prefer casual learning directly from the situation. The learning style of a person is measured by Kolbs' Learning Style Inventory (KLSI).

Kolb’s approach is by far the most frequently cited of the approaches for recording learning styles.

(vgl. "Lernorientierungen, Lernstile, Lerntypen und kognitive Stile", Ulrike Creß, in "Handbuch Lernstrategien" von Heinz Mandl & Helmut Felix Friedrich, S.371-372)

13.7. Selbstreguliertes Lernen

The concept of self-regulated learning is neither a precisely scientifically defined term nor a uniformly used term in everyday language. Furthermore, the terms self-regulated learning, self-directed learning, learner control can hardly be defined clearly.

Niegemann and Hofer (1997) or Büser (2003) define that in self-directed learning, in contrast to self-directed or self-regulated learning, the learning goal is determined by the person himself. Other authors, on the other hand, see the decision on learning goals explicitly as a component of self-directed or self-regulated learning (Arnold & Gomez-Tutor 2006; Dehnbostel 2003; Lang & Pätzold 2006; Neber 1978; Schreiber 1998, S. 45).

(vgl. Selbstreguliertes Lernen in der dualen Ausbildung - Lerntypen und Bedingungen, Johannes Rosendahl, 2010)

Beim selbstregulierten Lernen spielt die eigene Motivation eine große Rolle. Hierbei kann die Bildung von Zielintentionen(Lernzielen) helfen. (Die Wahl des Lernziels muss aber nicht zwingend selbstbestimmt erfolgen.) In diesen Zielintentionen kommen persönliche Standards und Referenzleistungen zum Ausdruck.

Ausgehend von diesem Ziel werden Handlungen durchgeführt, ihre Ausführung wird überwacht und ihr Ergebnis mit dem Ziel verglichen. Als Reaktion auf das Vergleichsergebnis wird das Handeln entweder angepasst oder eingestellt. Die eigentliche Selbstregulation umfasst die interagierenden Subprozesse Selbstbeobachtung, Bewertungsprozesse und Selbstreaktion (Bandura 1986, S. 337), die in Ansätzen zum selbstregulierten Lernen häufig gemeinsam mit Planungsaktivitäten unter dem Begriff der Metakognition zusammengefasst werden (Weinstein & Mayer 1986, Pintrich & De Groot 1990; Pintrich, Smith, Garcia & McKeachie 1993). Während der Selbstbeobachtung registriert der Lernende seine Leistung im Hinblick auf verschiedene Leisungsdimensionen wie Qualität, Originalität, Quantität und Moralität sowie Richtigkeit und Genauigkeit.

(S. 20) Der Begriff Selbstregulation betont, dass eine Person ihr Handeln in wesentlichem Ausmaß beeinflusst (Weinert 1982). Dies schließt allerding nicht aus, dass eine zwischenzeitliche Lernkontrolle (Vergleich Soll/Ist) durch den Asubilder/Lehrer erolgen kann.

14. Praktische Aufgabenstellung: Eric Haneder

15. Front End: Design und Graphic User Interface

15.1. Beginning

Nachdem die Technologie festgelegt war, mit der die Website programmiert werden sollte, konnten wir anfangen. Zuerst muss aber mit dem Auftraggeber ein Design abgeklärt werden, welches ihm zusagt, und für uns umsetzbar ist.

Hier ist ein abstraktes Diagramm der User-Interfaces: <Bild einfügen>

Nachdem auch das Design abgeklärt war, konnte nun die Erstellung der Interfaces beginnen.

15.2. Grundlagen

Um eine Website zu programmieren, benötigt man ein Menge an umfangreichen Wissen. Man muss html, javascript und css beherschen. Darüber hinaus sollte man auch wissen, wie aus dem Code eine fertige Seite erzeugt wird, und wie Requests vom User ausgewertet und darauf geantwortet werden kann.

JSF

"JSF Theorie einbinden"

HTML

Hypertext Markup Language (HTML) is the common used language for building web pages. A HTML page is a text document (with a .html or .htm extension) used by browsers to present text and graphics. A web page is made of content, tags to change some aspects of the content, and external objects such as images, videos, JavaScript, or CSS files.

Beispielcode für eine html page.

```
<html>
    <head>
        <title>My Webpage</title>
    </head>
```

```
    <body>
        <h1>This is my webpage</h1>
        <p>
            I hope you like it.
        </p>
        <a href="www.google.at">Link to Google</a>
    </body>
</html>
```

U can notice several tags in this code (such as <body> or <p>). Every tag has its on purpose, attributes and must be closed. Href is an attribute of the a-tag.

XHTML is just a validated version of html. This means, that there are certain rules, a html-page has to follow, to be valid. XHMTL pages have a .xhtml extension.

Some of the rules are the following:

- All tags must be closed. (so no
, <hr>, …)
- All tags are lowercase.
- Attributes appear appear between single or double quotes (<table border="0"> instead of <table border = 0>)
- There must be a strict structure with <html>, <head> and <body> tags.

CSS

Cascading Style Sheets (CSS) is a styling language used to describe the presentation of a document written in html or xhtml. CSS is used to define colors, fonts layouts, and other aspects of document presentation.It allows separation of a document's content (written in XHTML) from its presentation (written in CSS). To embedd a .css file in your html or xhtml page, use the <link> tag. (e.g. <link rel="stylesheet" type="text/css" ?

Your css file could look like this:

```
p {
    font-size: 10px;
}
h1 {
    color: red;
```

```
    font-style: italic;
}
```

15.3. Getting Started

Projektstruktur erstellen

To start with a project like this, you need a clean project structure. The project structure has been automatically created with Maven. Our project is "Open Source", that means it is available online for free. That includes our whole project structure and all of the files used to create the website. The complete GAME project can be viewed under: https://github.com/game-admin/game

Creating a layout

To simplify the creation of the UIs, I prepared a layout which servers every page as a template. This layou is realised in "mainlyaout.xhtml".

mainlyout.xhtml.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" ❶
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <link rel="shortcut icon" type="image/x-icon" href="faces/
favicon.ico" />
    </h:head>
    <h:body>
        <div id="header">

  <h:graphicImage library="img" name="header.png" style="width: 100%">
            </h:graphicImage>
        </div>
        <ui:insert name="menubar">
            <ui:include src="./faces/menubar.xhtml"></ui:include> ❷
        </ui:insert>
        <ui:insert name="header">
```

```
            <h:outputStylesheet library="resources" name="css/
stylesheet.css">
            </h:outputStylesheet>
        </ui:insert>
        <br />
        <ui:insert name="content">
            Content
        </ui:insert>
        <br />

  <h:graphicImage library="img" name="footer1.jpg" style="top:100%;
 margin-bottom: 0px; padding-bottom: 0px;"></h:graphicImage>
    </h:body>
</html>
```

❶ Implementation of taglibrarys

❷ Here, information from another page is built into the layou-page. Specifically, the menubar is implemented.

This page is not displayed directly to the user, but rather represents a template for all user interfaces. Other pages can define this page as a template and then adopt its content. By using <ui:insert> in the template file, you can give the template clients the possibility to define the content of this tag themselves with <ui:define>. This is shown in any of the actual user interfaces.

This is how the top of every page looks like:

The included menubar-file looks like the following:

menubar.xhtml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui">

    <body>
        <h:form id="menubar">
            <div id="menu">
```

```
                <p:tabMenu style="width:133.2%">
                    <p:menuitem value="Home" outcome="index.xhtml"
                    style="width:5em" icon="fa fa-home">
                    </p:menuitem>
                    <p:menuitem value="Courses" outcome="courses.xhtml"
                    style="width:7em" icon="fa fa-book">
                    </p:menuitem>
                    <p:menuitem value="Quizzes" outcome="quizzes.xhtml"
                    style="width:7em" icon="fa fa-question">
                    </p:menuitem>

  <p:menuitem value="Trainers" outcome="trainers.xhtml"
                    style="width:6em" icon="fa fa-users">
                    </p:menuitem>

  <p:menuitem value="Emblemtafel" outcome="leaderboard.xhtml"
                    style="width:8em" icon="fa fa-eye">
                    </p:menuitem>
                </p:tabMenu>
            </div>
            <br />
            <div id="logout">

  <p:splitButton id="basic" value="Account" action="index.html">
                    <p:menuitem value="Quizzes" action="quizzes.xhtml"/>
                    <p:menuitem value="Courses" action="courses.xhtml"/>
                    <p:separator/>
                    <p:menuitem value="Logout" url="http://
www.google.com"/>
                </p:splitButton>
            </div>

        </h:form>
    </body>
</html>
```

This page is unique, because its only purpose is to build the menubar, which is displayed on every Page. This is done by usind the Primefaces-tags: <p:tabMenu> and <p:splitButton>. I had to separate the menubar from the mainlyout, because of interferences with the formulars.

15.4. User-Interfaces

Die UIs liegen hier unter src\main\webapp\faces. "ev. noch was zu UIs sagen"

The index-page is the standard page the browser will run, if you enter a website. On this page, the user should get an overwiev about his statistics, and he should be able to navigate to other pages.

index.xhtml.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition template="./../mainlayout.xhtml" ❶
                  xmlns="http://www.w3.org/1999/xhtml"
                  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
                  xmlns:f="http://java.sun.com/jsf/core"
                  xmlns:h="http://java.sun.com/jsf/html"
                  xmlns:p="http://primefaces.org/ui">
❷
    <ui:define name="header"> ❸
        <title>Home</title>
        <h:outputStylesheet library="css" name="stylesheet.css" />
    </ui:define>

    <ui:define name="content"> ❹
        <h1>Activities</h1>
        <p class="description">
            Hallo
 #{traineeController.getTraineesByID("1").get(0).vorname},
            auf dieser Plattform ...
        </p>
        <hr/>
        <p id="data">
            <strong>Benutzer:</strong>
            #{traineeController.getTraineesByID("1").get(0).vorname}
            #{traineeController.getTraineesByID("1").get(0).nachname}
            <br />
            <strong>Nickname:</strong>
            #{traineeController.getTraineesByID("1").get(0).nickname}
            <br/>
            <strong>Fortschritt:</strong> 50% von 100%
            <br />
            <strong>Embleme:</strong>
            <h:graphicImage library="img"

  name="#{traineeController.getTraineesByID(&amp;quot;1&amp;quot;).get(0).embleme.get(0)}">
            </h:graphicImage>
```

```
            <br/><br/>
            Dein gesamter Fortschritt:

  <h:outputText value="#{traineeController.getTraineesByID(&quot;1&quot;).get(0).pr
 />
            <br/><br/>
            Hier sollte spezifisch zum User, dessen Aktivitäten und
 Fortschritt dargestellt werden.
        </p>
        <br />
    </ui:define>
</ui:composition>
```

❶ As you can see, the index-page is using the mainlayout with the "template" attribute.

❷ Everything above this callout will be same for every template client. I will therefore only include it here, to shorten the other code-snippets.

❹ I defined an individual title with <ui:define>. The content of this page is enclosed in <ui:define> too.

We dont need to set a footer here, because in the template page, there is already an universal footer defined.

Courses-page

The Courses-page should display a list of courses the trainee can go through. These courses can be mandatory to complete Quizzes.

courses.xhtml.

```
    <ui:define name="header">
        <title>Kurse</title>
        <h:outputStylesheet library="css" name="stylesheet.css" />
    </ui:define>

    <ui:define name="content">
        <h1 align="center">Kurse</h1>
        <p class="description">
            Hier kannst du Kurse nehmen, die dich auf die Quizzes
 vorbereiten. <br/>
            Nachdem du dir einen Kurs angeschaut hast,
            wird das jeweilige Quiz freigeschalten!
        </p>
        <p:dataTable var="kurs" value="#{kursController.kurse}"> ❶
```

```
            <p:column headerText="Titel">
                <h:outputText value="#{kurs.titel}"></h:outputText>
            </p:column>
            <p:column headerText="Beschreibung">
                <h:outputText value="#{kurs.beschreibung}"></
h:outputText>
            </p:column>
            <p:column>
                <h:form>
                    <p:commandButton
                    action="#{kursController.takeKurs(kurs.kursID)}"
                    value="Take Course!" >  ❷
                    </p:commandButton>
                </h:form>
            </p:column>
        </p:dataTable>
    </ui:define>
```

❶ The Primefaces tag <p:dataTable> take a List and knows how to display its content throught the columns. Here, I put in a List of courses and every course in the list has a titel, description and a link which is displayed in separated columns.

❷ The <p:commandButton invokes the takeKurs-method when pressed. In this method, a link of the selected course is returned.

This is how the courses interface looks like:

Quiz pages

The quiz pages include a interface, where every takeable quiz is displayed, two pages for taking a quiz and a page where the results are shown. Every quiz has its own emblem, which can be won if they quiz is taken succesfully. This means the user has to has at least half of the questions right.

quizzes.xhtml.

```
    <ui:define name="header">
        <title>Quizzes</title>
        <h:outputStylesheet library="css" name="stylesheet.css" />
    </ui:define>
```

```
    <ui:define name="content">
        <h1 align="center">Quizzes</h1>
        <p class="description">
            Hier kannst du ein Quiz deiner Wahl absolvieren. <br/>
            Anhand dem Titel und der Beschreibung kannst du dir
ausmalen, in welche
            Richtung das Quiz gehen wird. <br/>
            Falls du ein Quiz nicht nehmen kannst, musst du noch eine
Vorraussetzung
            dafür erfüllen.<br/>
            Dir wird in diesem Fall mitgeteilt, was du noch erfüllen
musst.
        </p>
        <p:dataTable var="quiz" value="#{quizController.quizzes}"> (1)
            <p:column headerText="Titel"
            rendered="#{quizController.isTakeable(quiz.QID,
&amp;quot;1&amp;quot;)}">
                <h:outputText value="#{quiz.titel}" />
            </p:column>
            <p:column headerText="Beschreibung"
            rendered="#{quizController.isTakeable(quiz.QID,
&amp;quot;1&amp;quot;)}">
                <h:outputText value="#{quiz.beschreibung}" />
            </p:column>
            <p:column
            rendered="#{quizController.isTakeable(quiz.QID,
&amp;quot;1&amp;quot;)}">
                <h:form>
                    <p:commandButton
                    action="#{quizController.quizUebergabe(quiz.QID)}"
                    value="Take Quiz!">
                    </p:commandButton>
                </h:form>
            </p:column>
        </p:dataTable>
        <hr/>
        <br/><br/>
    </ui:define>
```

(1) Here, <p:dataTable> is used again, this time to show all available quizzes. It gets a list of quizzes and displays a quiz whether or not it is takeable. This is evaluated in the isTakeable-method in the QuizController Bean.

This is how it looks like, when the user has not fulfilled the requirements to take a quiz:

In this picture below, the trainee has met all the requirements needed for the second quiz. To take the second quiz, the user must succesfully take the first quiz.

By clicking on the <p:commandButton>, the user can take the quiz.

takequiz.xhtml.

```
    <ui:define name="header">
        <title>Test-JavaQuiz</title>
        <h:outputStylesheet library="css" name="stylesheet.css"></
h:outputStylesheet>
    </ui:define>

    <ui:define name="content">
        <h1 align="center">Java Learning Quiz!</h1>
        <p style="margin-left: 20%; font-family: 'Arial', sans-serif;">
            Für jede richitg beantwortete Frage bekommst du 10
 Punkte! <br/>
            Es ist immer nur eine Antwort richtig! <br/>
            Wenn du alle Fragen richtig beantwortest, bekommst du dieses
 Emblem:

  <h:graphicImage library="img" name="#{quizController.emblem}">
            </h:graphicImage>
        </p>
        <div id="questions">
        <h:form>

  <ui:repeat var="frage" value="#{quizController.fragemodell}">
                <div class="question">
                    <h:outputLabel for="radio" value="#{frage.frage}" />

  <p:selectOneRadio id="radio" value="#{frage.selectedAnswer}" layout="grid" required="


  <f:selectItem itemValue="#{frage.antworten.get(0)}"
                        itemLabel="#{frage.antworten.get(0)}"></
f:selectItem>

  <f:selectItem itemValue="#{frage.antworten.get(1)}"
                        itemLabel="#{frage.antworten.get(1)}"></
f:selectItem>
```

```
  <f:selectItem itemValue="#{frage.antworten.get(2)}"
                        itemLabel="#{frage.antworten.get(2)}"></
f:selectItem>

  <f:selectItem itemValue="#{frage.antworten.get(3)}"
                        itemLabel="#{frage.antworten.get(3)}"></
f:selectItem>

                    </p:selectOneRadio>
                    <br/>
                </div>
                <!--  -->
                <br/><br/>
            </ui:repeat>
            <ui:param name="varqid" value="#{quizController.qid}"></
ui:param>
            <p:commandButton value="Check Answers" style="margin-left:
 20%"
            action="#{quizController.checkAnswersSingleChoice()}">
                <f:param name="qid" value="#{varqid}"></f:param>
            </p:commandButton>
        </h:form>
        </div>
    </ui:define>
```

This is the page for taking singlechoice-quizzes. The questions are repeatadly displayed by the <ui:repeat> tag. This tag runs through a given list, and displays the wanted data.

By clicking on the "Check Answers" button, the trainee is redirected to the results page.

takeQuizMultipleChoice.xhtml.

```
    <ui:define name="header">
        <title>Test - JavaQuiz</title>
        <h:outputStylesheet library="css" name="stylesheet.css"></
h:outputStylesheet>
    </ui:define>
```

```
    <ui:define name="content">
        <h1 align="center">Java Learning Quiz!</h1>
        <p style="margin-left: 20%; font-family: 'Arial', sans-serif;">
            Für jede richitg beantwortete Frage bekommst du 10
 Punkte! <br/>
            Es können mehrere Antworten richtig sein! <br/>
            Wenn du alle Fragen richtig beantwortest, bekommst du dieses
 Emblem:

  <h:graphicImage library="img" name="#{quizController.emblem}"></
h:graphicImage>
        </p>
        <div id="questions">
        <h:form>

  <ui:repeat var="frage" value="#{quizController.fragemodell}">
                <div class="question">
                    <h:outputText value="#{frage.frage}" />
                    <br />
                    <table>
                        <tr>
                            <td>

  <h:outputText value="#{frage.antworten.get(0)}" />
                            </td>
                            <td>

  <p:selectBooleanCheckbox value="#{quiz.buttons[0]}" />
                            </td>
                        </tr>
                        <tr>
                            <td>

  <h:outputText value="#{frage.antworten.get(1)}" />
                            </td>
                            <td>

  <p:selectBooleanCheckbox value="#{quiz.buttons[1]}" />
                            </td>
                        </tr>
                        <tr>
                            <td>

  <h:outputText value="#{frage.antworten.get(2)}" />
                            </td>
                            <td>
```

```
  <p:selectBooleanCheckbox value="#{quiz.buttons[2]}" />
                            </td>
                        </tr>
                        <tr>
                            <td>

  <h:outputText value="#{frage.antworten.get(3)}" />
                            </td>
                            <td>

  <p:selectBooleanCheckbox value="#{quiz.buttons[3]}" />
                            </td>
                        </tr>
                    </table>
                    <br/><br/>
                </div>
            </ui:repeat>
            <ui:param name="varqid" value="#{quizController.qid}"></
ui:param>
            <p:commandButton value="Check Answers" style="margin-left:
 20%"
            action="#{quizController.checkAnswersMultipleChoice}">
                <f:param name="qid" value="#{varqid}"></f:param>
            </p:commandButton>
        </h:form>
        </div>
    </ui:define>
```

On this page, multiplechoice quizzes are displayed. The questions are dislayed in the same way, as mentioned above. The difference here is, that I had to create every button seperately (<p:selectBooleanCheckBox>). Each of these buttons hast to be bound to a *Boolean*-variable.

"Bilder für takeQuizMultipleChoice"

results.xhtml.

```
    <ui:define name="header">
        <title>Results</title>
        <h:outputStylesheet library="css" name="stylesheet.css"></
h:outputStylesheet>
    </ui:define>

    <ui:define name="content">
```

```
        <h1>Results</h1>
        <p style="font-size: 1.2em; margin-left:25%;">
            Du hast #{quizController.ricounter}/
#{quizController.fragemodell.size()}
            Fragen richtig beantwortet! <br/>
            Damit bekommst du #{quizController.ricounter*10}
 Punkte! <br/>
            Die von dir richtig/falsch beantworteten Fragen siehst du
 hier mit den
            richtigen Antworten:
        </p>
        <p>
            <ui:repeat var="result" value="#{quizController.results}">
                <div id="results">
                    <h:outputText value="#{result.frage}"
                    style="#{result.istfalsch?
                    'color:red; font-weight: bold;' :
                    'color:green; font-weight: bold;'}"></h:outputText>
                    <i class="#{result.istfalsch?
                    'fa fa-fw fa-close' : 'fa fa-fw fa-check'}"></i><br/
>

                    <h:outputText value="#{result.antworten.get(0)}"
                    style="#{result.richtigeAntworten.get(0) == 1?
                    'color:green' : 'color:red'}"></h:outputText>
                    <br/>
                    <h:outputText value="#{result.antworten.get(1)}"
                    style="#{result.richtigeAntworten.get(1) == 1?
                    'color:green' : 'color:red'}"></h:outputText>
                    <br/>
                    <h:outputText value="#{result.antworten.get(2)}"
                    style="#{result.richtigeAntworten.get(2) == 1?
                    'color:green' : 'color:red'}"></h:outputText>
                    <br/>
                    <h:outputText value="#{result.antworten.get(3)}"
                    style="#{result.richtigeAntworten.get(3) == 1?
                    'color:green' : 'color:red'}"></h:outputText>
                    <br/>
                </div>
            </ui:repeat>
        </p>
    </ui:define>
```

Trainer page

The Trainers page should display all the trainers associated with the GAME platform. The trainees can contact these trainers if they need help.

trainers.xhtml.

```
    <ui:define name="header">
        <title>Trainers</title>
        <h:outputStylesheet library="css" name="stylesheet.css" />
    </ui:define>>

    <ui:define name="content">
        <h1 align="center">Trainers</h1>
        <p class="description">
            Hier siehst du alle Trainer, die dir im Falle von Problemen
 helfen können,
            aufgelistet.
        </p>
        <h:form id="trainerform">

  <p:carousel value="#{trainerController.trainers}" headerText="Trainers"
            var="trainer" itemStyle="text-
align:center" responsive="true">
                <p:panelGrid columns="2" style="width:100%;margin:10px
 0px"
                columnClasses="label,value" layout="grid"
                styleClass="ui-panelgrid-blank">
                    <f:facet name="header">

  <p:graphicImage library="img" name="trainer.jpg"/>
                    </f:facet>
                    <h:outputText value="Name:"  />
                    <h:outputText value="#{trainer.name}" />
                    <h:outputText value="Rolle:" />
                    <h:outputText value="#{trainer.role}" />
                    <h:outputText value="Abteilung:" />
                    <h:outputText value="#{trainer.branch}" />
                </p:panelGrid>
            </p:carousel>
        </h:form>
    </ui:define>
```

Here we used a primefaces tag called <p:carousel>. This tag is used to create a carousel. The <panelGrid> tag is used to display data in a grid. The

<p:graphicImage> is just like the JSF tag <h:graphicImage>. <h:outputText> is used to display text, with the function to call a Backing Bean. For easier explanation here is a picture:

Leaderboard page

The leaderboard page is used to diplay all the trainees with their names, nicknames, branches and Icons they got. You should be able to sort them by their names.

leaderboard.xhtml.

```
    <ui:define name="header">
        <h:outputStylesheet library="css" name="stylesheet.css" />
        <title>Emblemtafel</title>
    </ui:define>

    <ui:define name="content">
        <h1 align="center">Emblemtafel</h1>
        <p class="description">
            Hier siehst du eine Auflistung aller Trainees, mit ihren
Nicknames
            und ihren Emblemen.
        </p>

 <p:dataTable var="trainee" value="#{traineeController.trainees}">
            <p:column headerText="Name">
                <h:outputText value="#{trainee.vorname} " />
                <h:outputText value="#{trainee.nachname}" />
            </p:column>

            <p:column headerText="Nickname">
                <h:outputText value="#{trainee.nickname}" />
            </p:column>

            <p:column headerText="Abteilung">
                <h:outputText value="#{trainee.abteilung}" />
            </p:column>

            <p:column headerText="Embleme">
                <h:graphicImage
                value="#{trainee.embleme.get(0)}">
                </h:graphicImage>
```

```
            </p:column>
        </p:dataTable>
    </ui:define>
```

Here, the <p:dataTable> tag is used, which is rendered as a <table> tag, with some style-modifications. If you give it a list, it will know to diplay all elements of the list.

15.5. Java Classes

Trainer Classes

TrainerController.java.

```
@Named
@ViewScoped
public class TrainerController implements Serializable {

    private List<Trainer> trainers;

    private Trainer selectedTrainer;

    @Inject
    private TrainerService service;

    @PostConstruct
    public void init() {
        trainers = service.createTrainers(6);
    }

    //Getters & Setters
}
```

The TrainerController-Class is used to diplay all the trainers on the trainers.xhtml page.

TrainerService.java.

```
@Named
@ApplicationScoped
public class TrainerService {
```

```
    private final static String[] roles;
    private final static String[] branches;
    private final static String[] names;

    static {
        roles = new String[4];
        roles[0] = "Abteilungsleiter";
        roles[1] = "Gruppenleiter";
        roles[2] = "Projektleiter";
        roles[3] = "Projektmitglied";

        branches = new String[3];
        branches[0] = "Development";
        branches[1] = "Architecture";
        branches[2] = "Design";

        names = new String[10];
        names[0] = "Max";
        names[1] = "Peter";
        names[2] = "Tim";
        names[3] = "Tom";
        names[4] = "Alex";
        names[5] = "Josef";
        names[6] = "Tobias";
        names[7] = "Benji";
        names[8] = "Michael";
        names[9] = "Martin";

    }

    public List<Trainer> createTrainers(int size) {
        List<Trainer> list = new ArrayList<>();
        for(int i = 0 ; i < size ; i++) {
            list.add(new Trainer(getRandomName(), getRandomBranch(),
 getRandomRole()));
        }
        return list;
    }

    //Getters
}
```

The TrainerService class is used to create dummy data, that can be displayed on the trainers page.

Trainee Classes

TraineeController.java.

```
@Named
@ViewScoped
public class TraineeController implements Serializable {
    private List<Trainee> trainees;
    private Trainee selectedTrainee;

    @Inject
    private TraineeEJB traineebean;


    public List<Trainee> getTraineesByID(String mitid) {
        List<Trainee> list = new ArrayList<>(1);
        Trainee trainee = traineebean.find(mitid);
        list.add(trainee);
        return list;
    }

    //Getters & Setters
}
```

The TraineeController-Class is used to display all trainees on the leaderboard-page.

Quiz Classes

QuizController.

```
@Named
@SessionScoped
public class QuizController implements Serializable {

    private List<Quiz> quizzes;
    private List<Results> results;
    private int score;
    private Trainee trainee;
    private int ricounter;
    private String emblem = "javapro.png";
    private List<FrageModell> fragemodell;
    private String qid;
```

```
    @Inject
    private ModellCreator creator;
    @Inject
    private TraineeEJB traineebean;
    @Inject
    private FrageEJB fragebean;
    @Inject
    private QuizEJB quizbean; ❶
    @Inject
    private QuizbeantwortungEJB quizbeantw;
    @Inject
    private QuizVoraussetzungEJB quizvoraussetzung;
    @Inject
    private VoraussetzungEJB voraussetzungejb;

    @PostConstruct ❷
    public void init() {
        fragemodell = creator.createModell(qid);
        results = new ArrayList<>();
        ricounter = 0;
    }

    public String checkAnswersSingleChoice() {
        FacesContext fc = FacesContext.getCurrentInstance();
        Map<String,String> params =
fc.getExternalContext().getRequestParameterMap();
        String varqid = params.get("qid");
        this.qid = varqid;
        evaluateScoreRadio();
        return "result.xhtml";
    }

    public String checkAnswersMultipleChoice() {
        FacesContext fc = FacesContext.getCurrentInstance();
        Map<String,String> params =
fc.getExternalContext().getRequestParameterMap();
        String varqid = params.get("qid");
        this.qid = varqid;
        evaluateScoreMultiple();
        return "result.xhtml";
    }

    public void evaluateScoreMultiple() { ❸
        List<Integer> falsche = new ArrayList<>();
```

```
        int richtige=0;
        for(int i=0; i<fragemodell.size(); i++ ) {
            List<Integer> indexrichtig =
 umwandler(fragemodell.get(i).indexrichtig);
            for(int z=0; z<4; z++) {
                if(indexrichtig.get(z) == 1 && !
fragemodell.get(i).buttons[z] || indexrichtig.get(z) == 0 &&
 fragemodell.get(i).buttons[z]) {
                    falsche.add(i);
                    z=999;
                } else {
                    richtige++;
                }
            }
            if(richtige==4) {
                score+=10;
                ricounter++;
                falsche.add(9999);
            }
            richtige = 0;
        }
        checkResults(falsche);
    }

    public void evaluateScoreRadio() { ❹
        List<Integer> falsche = new ArrayList<>();
        for(int i=0; i<fragemodell.size(); i++) {

  if(fragemodell.get(i).selectedAnswer.equals(fragemodell.get(i).antworten.get(frag
 {
                score+=10;
                ricounter++;
                falsche.add(9999);
            } else {
                falsche.add(i);
            }
        }
        checkResults(falsche);
    }

    public String quizUebergabe(String qid) {
        this.qid = qid;
        fragemodell = creator.createModell(qid);
        ricounter = 0;
        score=0;
```

```
        if(quizbean.find(qid).getMultiplechoice()) {
            return "takeQuizMultipleChoice.xhtml";
        } else {
            return "takequiz.xhtml";
        }
    }

    public List<Integer> umwandler(int indexrichtig) {
        List<Integer> liste = new ArrayList<>();
        for(int i=0; i<4; i++)
            if(indexrichtig==i) {
                liste.add(1);
            } else {
                liste.add(0);
            }
        return liste;
    }
    public int makeListToIndexRichtig(List<Antwortmoeglichkeiten>
antworten) { //Used here?
        for (int i = 0; i < 4; i++) {
            if(antworten.get(i).isRichtigeAntwort()) {
                return i;
            }
        }
        return 0;
    }

    public void checkResults(List<Integer> falsche) {
        results = new ArrayList<>();
        for(int i=0; i<fragemodell.size(); i++) {
            List<Integer> indexrichtig =
umwandler(fragemodell.get(i).indexrichtig);
            if(falsche.get(i) == i) {
                results.add(new Results(fragemodell.get(i).frage,
fragemodell.get(i).antworten, indexrichtig, true));
            } else {
                results.add(new Results(fragemodell.get(i).frage,
fragemodell.get(i).antworten, indexrichtig, false));
            }
        }
     trainee = traineebean.find("1");
     trainee.setProgress(trainee.getProgress()+score);
     traineebean.update(trainee);
     List<Quizbeantwortung> list =
quizbeantw.findByQIDAndMITID(qid, "1");
```

```
     list.get(0).setErreichtePunkte(score);
     if(score > fragemodell.size()*10/2) {
        list.get(0).setIstbestanden(true);
     }
     quizbeantw.update(list.get(0));
    }

    public Boolean isTakeable(String qid, String mitid) {
        List<QuizVoraussetzung> quizvor =
 quizvoraussetzung.findAllQuizVoraussetzzungen(qid);
        if(quizvor.isEmpty()) {
            return TRUE;
        }
        Voraussetzung vor =
 voraussetzungejb.find(quizvor.get(0).getQuizVorraussetzID());
        String id = vor.getQuiz().getQID();
        List<Quizbeantwortung> list = quizbeantw.findByQIDAndMITID(id,
 mitid);
        if(list.get(0).isIstbestanden()) {
            return TRUE;
        }

        return FALSE;
    }

    //Getters & Setters
}
```

❶ Here, the QuizEJB is injected via the Jave EE Dependy Injection System. This way, we can use everything from the injected Class, without the need of calling a contructor.

❷ The annotation *PostContruct* functions as a note to the container, that this method must be run, before the construction of the Class.

❸ This method is used to evaluate the results of a MultipleChoice-Quiz. Each button is bound to a boolean-wert of the buttons[]. Every Question is checked, if every button matches the right answers. The user only gets points, if he answers the question correctly.

❹ Here, the singlechoice-quizzes get evaluated. This is much easier, because the radiobuttons function differently than the normal buttons. Every set of radiobuttons is bound to one value (selectedAnswer). We only need to check if the selected Answer matches the correct Answer. <5> In *checkResults*,

a *Results-List* is generated. This list is used to diplay the results on the results.xhtml page.

The QuizController is used to handle everything surrounding the action of taking a quiz. It is responsible for displaying the content on the quizzes-, takequiz- and results-page. It will forward the user from the quizzes page to the takequizpage, where he/she can take the quiz. By clicking on th Submit button, the user is forwarded to the results-site, where their results are shown.

15.6. Summary

"was ghört hir hin?"

16. Theoretische Aufgabenstellung: Jan Binder

17. Knowledge management

17.1. Knowledge

Knowledge is omnipresent whereever we go. Everyone around has more or less knowledge about every kind of topic.

The definition of knowledge is a problem since the early ages. Philosophers like Aristoteles or Platon, who lived ~400 years before christ, worried about what knowledge really is. It is connected to the search of truth. The approach of Platon says that knowledge is acquired by means of deduction, the absolut truth develops through thinking logically. Aristoteles opinion is, that the sensory perception is the only source of insight. The next assumption to this topic wanted to include both approaches and said it is a combination of both. Till now there is only an agreement of the fact, that knowledge is not static and not the absolut truth.

(vgl. Wissensmanagement in der Schulentwicklung-Theoretische Analyse und empirische Exploratio aus systematischer Sicht - Definitionsproblematik, Kaja Heitmann, 2012)

17.2. Knowledge management in context of the knowlegde socitey

17.3. Knowledge management models

17.4. Knowledge management methods and tools

Repertory-Grid-Technique

This technique is used to acquire knowledge and represent it graphically. People often use mental models to solve problems in their everyday life. These models can be collected and presented transparent with this technique.

Origin and Background

(vgl. Wissensmanagement in der Praxis-Herkunft und Hintergrund, Christian Stary, Monika Moschner, Edith Stary, 2013)

Critical-Incident-Technique (CIT)

The critical-incident technique allows the collection of observed behaviour that led to particular success or failure (this behaviour is therefore referred to as "critical" in the following) in performing a particular activity. Critical incidents are collected by means of questionnaires or interviews either with the performers of the successful or failed task themselves or with observers of the performance of tasks. The method takes particular account of the circumstances that led to the event, i.e. activities and factors that made the event successful or unsuccessful. It thus leads to specific descriptions of behaviour. The collection of critical events of an activity sharpens the awareness of the actions taken. In some cases the interviewees only become aware of the existence of these events during the survey, as they have not yet or only partially dealt with them beforehand.

(vgl. Wissensmanagement in der Praxis-Critical Incident Technik, Christian Stary, Monika Moschner, Edith Stary, 2013)

Origin

The origin of the critical incident technique goes back to Sir Francis Galton, who conducted studies in this direction as early as the end of the 19th century. Subsequently, controlled observation tests were developed, studies on recreational activities were conducted and anecdotal records were kept. The foundation of the Critical Incident Technique in its present form was not laid until the middle of the Second World War. In the summer of 1941, John Flanagan conducted studies as part of a US Air Force flight psychology program. The purpose of the program was to develop a procedure for selecting and classifying crews. The Critical Incident Technique has since been steadily developed and is now used in a variety of other areas.

Objective and possible applications

Critical Incident Technique aims to detect extraordinarily successful or unsuccessful work behaviour caused by critical events. Since the underlying

behaviour is analysed, implicit (expert) knowledge is to be recorded and collected. Critical Incident Technique aims to improve work processes and help to avoid experienced errors in the future. Its application should make it easier for employees or those carrying out work activities to perform their tasks more effectively or more easily in the future. The method can also support those responsible in their decision-making in many areas, for example when hiring new employees. Typical issues where the Critical Incident Technique can be helpful are What qualifications should new employees have? How can I increase the motivation and productivity of my employees? How can frequently made mistakes be avoided in the future? On the basis of observed facts, the technology helps to find conclusive answers to such questions and thus to develop options for action. The critical incident technique is particularly suitable for surveys in which a structured, behavioural method is required to raise knowledge or to make explicit knowledge transparent in a structured form and thus accessible to a new group of users. The technique has already been successfully applied in the following areas:

- Military: The method was introduced by John Flanagan during the Second World War in order to identify and process critical situations in aviation. Concrete events of effective and ineffective behaviour in aviation during the war should be found. To this end, he asked war veterans about events that were particularly important, helpful or inadequate for them to carry out the missions they had been assigned. One question for obtaining these descriptions of behaviour was for example: "Describe the officer’s action. What exactly did he do?"
- Police: In this context, the method was used to analyse the activities of police officers in specific work situations. The relationship between stressful circumstances and certain behavioural patterns could be investigated using this technique. Sales: Analogous to the police, work situations were investigated which were characterised by certain customer-product constellations on the one hand and certain behavioural patterns on the other hand.
- (Software) development: In the context of this assignment, the coordination of tasks between managers and employees was the focus of interest.
- customer service: In this assignment, services were analyzed in detail from the consumer’s perspective.

- Housework: In this context, the method was used to analyze conflicts that arise when couples with professional careers divide up their housework.
- Training - concept development: In this area, the method was used to develop definitions and theories of leadership and professionalism in order to impart knowledge.

Flanagan (1954) himself pointed out a number of other areas of application:

- Measurement of typical performance criteria: In this context, the critical incident technique was used to create an observation protocol list that included all the important courses of action for an activity. This list can then be used to objectively evaluate a person’s performance.
- Measurement of skills/knowledge (standard samples): Standard samples were used to assess the knowledge of people concerning important aspects of their activities. This form is often used at the end of training courses to assess whether students have retained the knowledge they have acquired or can apply it correctly.
- Teaching: Many applications of critical incident techniques to problems in training have been developed for specific situations in the military. The technique is intended to help create better conditions for teaching, for example by strengthening motivating didactic moments.
- Job design: For a long time, insufficient attention was paid to job design, although it is essential to promote the motivation of individuals. In this area, the critical-incident technique attempts to limit the number of critical job elements of employees to two or three critical elements. This is intended to maximise the effectiveness of performance in relation to each of the different types of tasks.
- Operating procedures: Another application of the method is the study of operating procedures. The method helps to efficiently collect detailed, factual data based on successes or failures that can be systematically analysed. This is an essential prerequisite for improving the effectiveness and performance of operating procedures.
- Equipment design: Here, the design of equipment or fittings is to be improved by collecting critical events in the handling of operating resources and tools. Reports "from the field" form the basis for improvements. Critical-incident

technology facilitates the collection and processing of information to improve equipment and tools.

- Motivation and leadership: Critical incident technology was used in this context to collect data on specific actions, including decisions made and options chosen. From these data, causal relationships between work actions and leadership activities could be derived.
- Psychotherapy: The method is also used in this field. It serves as an aid in the collection of professional-critical events, with particular attention being paid to the interrelation of factors.

The Critical Incident Technique can therefore be used in many economic and social areas due to its openness in terms of content.

(vgl. Wissensmanagement in der Praxis-Critical Incident Technik-Zielsetzung und Einsatzmöglichkeiten, Christian Stary, Monika Moschner, Edith Stary, 2013)

Balanced Scorecard (BSC)

The BSC is a method for the development and organisation-wide communication of an organisation's mission, vision and strategies derived from them. It can be described as a management system for the strategic management of an organisation with key figures. It is presented by means of a clearly arranged report sheet which contains not only results but also actions with which organisations prepare future activities. Furthermore, the results and actions are considered from different perspectives and in a balanced manner. Different types of BSCs are used in organisational practice. What these approaches have in common is that strategies are translated into concrete actions.

BSCs initially contain the formulation of a central strategic goal (key objective or vision) and the corresponding concretization of the key objective through sub-goals. The sub-goals are derived from several elements: Strategic orientations (topics or factors critical to success). Expectations of various stakeholders (= perspectives) regarding organizational potential. These are: Customers, business processes that primarily have an after-effect, employees (learning and development, innovation), finance and controlling, partners or competitors (suppliers, cooperation partners, associations etc.). The financial management is the focus of attention. The utilization of financial capital is definitely seen as an organization's ultimate goal. Therefore, the financial perspective represents

the top level of a hierarchically structured BSC. This perspective is followed by the customer perspective, which describes the value proposition that is made available to the market. Below this is the perspective of the internal business processes, which comprise the value chain of the organization. This chain includes all activities necessary to create the value proposition for customers and transform it into growth and profitability for the shareholder. The foundation of the three perspectives is the learning and development perspective, as it defines intangible assets that are needed to take entrepreneurial activities and customer relationships to a higher level. The other elements of the Balanced Scorecard are: defined metrics as measures for key objectives and selected sub-objectives (strategic themes, perspectives), derived actions that meet the sub-objectives, defined metrics for the actions, organization of joint work for the practical implementation of the strategy (projects, action programs), integration of the metrics into the reporting system.

(vgl. Wissensmanagement in der Praxis-Balanced Scorecard, Christian Stary, Monika Moschner, Edith Stary, 2013)

18. Praktische Aufgabenstellung: Jan Binder

19. Back End: Datenbank und Datenbankverbindung

19.1. Begin

Zu aller erst wurden die allgemeinen Grundlagen, wie die Technologie, die verwendet wird, und das Design festgelegt. Auch die Datenbank, in der die entsprechenden Tabellen und Datensätze stehen soll, wurde vereinbart.

First of all the basics, like the technology we are using and the design are set. The database, in which all the tables and data sets are, is also appointed.

19.2. Grundlagen

Man braucht, um eine Datenbank und eine Datenbankverbindung zu erzeugen, ein gewisses Vorwissen. Hier werden SQL, JPA und Java von Nöten sein. Also wie das Programm mit der Datenbank kommuniziert und wie sogenannte Queries erstellt werden.

You need some foreknowledgement to build a database and a database connectivity. Knowlegde about SQL, JPA and Java is required. You need these languages to create queriesor to understand how the program communicates with the database.

SQL

SQL is short for "Structured Query Language". It is used to insert, change and delete data sets or create or delete tables.

Beispielcode von SQL.

```
❶
CREATE TABLE mitarbeiter (
    id integer,
    nachname text,
    vorname text,
    gehalt integer
);

❷
```

```
INSERT INTO mitarbeiter (id, nachname, vorname, gehalt)
VALUES (1, Mustermann, Max, 2000);



❸
SELECT * FROM mitarbeiter;

❹
ALTER TABLE mitarbeiter ADD CONSTRAINT "MITARBEITER_pkey" PRIMARY KEY
 ("id");
```

❶ All this code does is it creates a new table called "mitarbeiter" which contains the columns "id", "nachname", "vorname" and "gehalt". Every column has its datatype next to it. Optionally there are constraints to the columns, such as "NOT NULL". which tells the database that this column cannot be empty or "UNIQUE", which means that a data set must be unique in this column.

❷ The next command is INSERT INTO. This code tells the database to write data into the table. First you need to set the table with all its columns and then the values like shown.

❸ The "SELECT * FROM mitarbeiter;" command picks out every data set from the table "mitarbeiter". This is done by the '*' parameter. Further there is a possibility to only select a few specific data sets.

❹ This command lets you change an already existing table. In this example I updated the tables constraints, in detail I set the primary key of the table.

JPA

19.3. Getting Started

Entities

Kurs Entities

Kurs.java.

```
/**
 *
 * @author Jan
 */
@Entity
```

```
@Table(name = "kurs", schema = "game")
@NamedQuery(name = Kurs.QUERY_FINDALLKURSE, query = "SELECT kurs FROM
 Kurs kurs")
public class Kurs implements Serializable {
    public static final String QUERY_FINDALLKURSE = "Kurs.findAll";
    @Id
    @Column(name="kursid")
    private String KursID;
    @NotNull
    @Column(name="titel")
    private String titel;
    @NotNull
    @Column(name="link")
    private String link;
    @Column(name="beschreibung")
    private String beschreibung;

//Getter, Setter
```

Here is the Kurs-Entity, it has the primary key "KursID", which helps to find a specific course from the table. The table also has a column "titel" which is basically the title of this course. It also has the constraint NOT NULL so if there is a new course there must be a title as well. The next column is the "link" it contains the links of the courses so the visitors can be redirected to the page with the explanation of the topic. This cannot be empty too. The last column is "beschreibung", which is the description of the course. This column can be empty, but it is not recommended since it helps the user specify which course contains what information.

KursBesuch.java.

```
/**
 *
 * @author Jan
 */

@Entity
@Table(name = "kursbesuch", schema = "game")
public class KursBesuch implements Serializable {
    @Id
    @Column(name="KURSBESUCHID")
    private String KursBesuchID;
    @ManyToOne(fetch = FetchType.LAZY)
```

```
    @JoinColumn(name="MitID")
    @NotNull
    private Trainee trainee;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="KursID")
    @NotNull
    private Kurs kurs;
    @NotNull
    @Column(name="DATUM")
    private Date datum;
//Getter, Setter
```

The entity "KursBesuch" is a table, which splits the many-to-many relationship between "trainee" and "kurs". That means it has the PK (primary key) of both tables as a foreign key and it also provides a column named "datum" which indicates the date the user has visited the course. The function of this entity is to show which trainee has done which course and when.

Vorausetzung Entities

Vorausetzung.java.

```
import org.game.trainee.quiz.Quiz;

/**
 *
 * @author Jan
 */

@Entity
@Table(name="voraussetzung", schema = "game")
public class Voraussetzung implements Serializable {
    @Id @Column(name="VORAUSSETZID")
    private String VoraussetzID;
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name="KursID")
    private Kurs kurs;
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name="qid")
    private Quiz quiz;


//Getter, Setter
```

This table defines the requirement a course or a quiz can have. Therefor it containes a "KursId" and a "qid". The primary key of this table is then given to "KursVoraussetzung" or "QuizVoraussetzung" respectively.

KursVorausetzung.java.

```
/**
 *
 * @author Jan
 */
@Entity
@Table(name = "kursvoraussetzung", schema = "game")
public class KursVoraussetzung implements Serializable {
    @Id
    @Column(name="KURSVORAUSSETZID")
    private String KursVoraussetzID;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="KursID")
    private Kurs kurs;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="VoraussetzID")
    private Voraussetzung voraussetzung;

//Getter, Setter
```

The function of "KursVoraussetzung" is that a course can have a course or a quiz as a requirement to take this course. This table contains courses and their required quizzes or courses.

QuizVorausetzung.java.

```
/**
 *
 * @author Jan
 */
@Entity
@Table(name = "quizvoraussetzung", schema = "game")
@NamedQuery(name = QuizVoraussetzung.QUERY_FINDALLVORAUSSETZUNGEN, query
 = "SELECT quizvor FROM QuizVoraussetzung quizvor WHERE quizvor.quiz.QID
 = :QID")
public class QuizVoraussetzung implements Serializable{
    public static final String QUERY_FINDALLVORAUSSETZUNGEN
 = "QuizVoraussetung.findAllQuizVoraussetzzungen";
    @Id
```

```
    @Column(name="QUIZVORAUSSETZID")
    private String QuizVoraussetzID;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="QID")
    private Quiz quiz;

//Getter, Setter
```

This table has the same function as "KursVoraussetzung", but instead of courses with quizzes. So this table contains quizzes and their required courses or quizzes.

Quiz Entities

Quiz.java.

```
/**
 *
 * @author Jan
 */
@NamedQuery(name = Quiz.QUERY_FINDBY_BESCHREIBUNG, query = "SELECT quiz
 FROM Quiz quiz WHERE quiz.beschreibung like :beschreibung")
@NamedQuery(name = Quiz.QUERY_FINDALL, query = "SELECT quiz FROM Quiz
 quiz")
@Entity
@Table(name = "quiz", schema = "game")
public class Quiz implements Serializable {
    public static final String
 QUERY_FINDBY_BESCHREIBUNG="Quiz.findByBeschreibung";
    public static final String QUERY_FINDALL="Quiz.findAll";
    @Id @Column(name = "qid")
    private String QID;
    @NotNull @Column(name = "titel")
    private String titel;
    @Column(name = "beschreibung")
    private String beschreibung;
    @NotNull @Column(name = "reward")
    private String reward;
    @Column(name="multiplechoice")
    private Boolean multiplechoice;
    @OneToMany(fetch = FetchType.EAGER, mappedBy = "quiz")
    private List<Frage> fragen;
    @OneToMany(fetch = FetchType.EAGER, mappedBy = "quiz")
    private List<Quizbeantwortung> quizbeantwortung;
    @OneToOne(fetch = FetchType.EAGER, mappedBy = "quiz")
    private Voraussetzung voraussetzung;
```

```
//Getter, Setter
```

This is the table for all quizzes. It contains the "QID", which is the primary key, a title, a description and a reward. The title has a constraint called NOT NULL because every quiz must have a specification which topic it has. There are also so called queries to find data with specific parameters from the database.

Quizbeantwortung.java.

```
/**
 *
 * @author Jan
 */
@Entity
@Table(name = "quizbeantwortung", schema = "game")
@NamedQuery(name = Quizbeantwortung.QUERY_FINDBY_QIDANDMITID,
 query="SELECT quizbeantw FROM Quizbeantwortung quizbeantw WHERE
 quizbeantw.quiz.QID = :QID AND quizbeantw.trainee.MitID = :MitID")
public class Quizbeantwortung implements Serializable {
    public static final String
 QUERY_FINDBY_QIDANDMITID="Quizbeantwortung.findByQIDAndMITID";
    @Id @Column(name = "qbeid")
    private String QBEID;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="qid")
    private Quiz quiz;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="mitid")
    private Trainee trainee;
    @NotNull
    @Column(name="erreichtepunktezahl")
    private int erreichtePunkte;
    @Column(name="istbestanden")
    private boolean istbestanden;



//Getter, Setter
```

This entity is essential for the implementation of the "Voraussetzungs-Logik". We need this table to check if a trainee has already done a quiz. This is done with the column "istbestanden".

Frage.java.

```
import javax.persistence.Table;
import javax.validation.constraints.NotNull;

/**
 *
 * @author Jan
 */
@Entity
@Table(name = "frage", schema = "game")
@NamedQuery(name = Frage.QUERY_FINDALLFRAGEN, query = "SELECT frage FROM
 Frage frage")
@NamedQuery(name = Frage.QUERY_FINDFRAGENZUQID, query = "SELECT frage
 FROM Frage frage WHERE frage.quiz.QID = :QID")
//@NamedQuery(name = "QUERY_FINDALLANTWORTEN", query = "SELECT frage,
 antwort FROM Antwortmoeglichkeiten antwort INNER JOIN antwort.frage
 frage");
public class Frage implements Serializable {
    //public static final NAMED_QUERY = "SELECT antwort FROM Frage
 frage";
    public static final String QUERY_FINDALLFRAGEN = "Frage.findAll";
    public static final String QUERY_FINDFRAGENZUQID
 = "Frage.findFrageByQID";
    //public static final String QUERY_FINDALLANTWORTEN =
 "Frage.findAllAntworten";
    @Id
    private String FID;
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name="qid")
    private Quiz quiz;
    @NotNull
    @Column(name="frage")
    private String Frage;
    @NotNull
    @Column(name="punktezahl")
    private int punktezahl;
    @OneToMany(fetch = FetchType.EAGER)
    @JoinColumn(name="fid")
    private List<Antwortmoeglichkeiten> antworten;


//Getter, Setter
```

Every Quiz contains of many questions. These questions are stored in the table "Frage". Each question has a "FID", a "QID", so it can be connected to a quiz. A question also has the question itself and points you get for each question.

Antwortmoeglichkeiten.java.

```
import javax.validation.constraints.NotNull;

/**
 *
 * @author Jan
 */

@Entity
@Table(name = "antwortmoeglichkeiten", schema = "game")
@NamedQuery(name =
 Antwortmoeglichkeiten.QUERY_FINDANTWORTEN_BYFIDANDQID, query = "SELECT
 antwort FROM Antwortmoeglichkeiten antwort WHERE antwort.frage.FID
 = :FID AND antwort.frage.quiz.QID = :QID")
@NamedQuery(name = Antwortmoeglichkeiten.QUERY_FINDANTWORTEN_BYFID,
 query = "SELECT antwort FROM Antwortmoeglichkeiten antwort WHERE
 antwort.frage.FID = :FID")
public class Antwortmoeglichkeiten implements Serializable {
    public static final String
 QUERY_FINDANTWORTEN_BYFID="Antwortmoeglichkeiten.findAntwortenByFID";
    public static final String
 QUERY_FINDANTWORTEN_BYFIDANDQID="Antwortmoeglichkeiten.findAntwortenByFIDAndQID";
    @Id
    @Column(name="ANTWID")
    private String AntwID;
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name="FID")
    private Frage frage;
    @NotNull
    @Column(name="ANTWORT")
    private String antwort;
    @Column(name="RICHTIGEANTWORT")
    private boolean richtigeAntwort;


//Getter, Setter
```

Every question has 4 answers. These answers are stored in this table. Each answer is connected to its question. Also the answertext itself is stored here, as well as a boolean value if the answer is correct or not.

Trainee Entity

Trainee.java.

```
/**
 *
 * @author Jan
 */

@Entity
@Table(name = "trainee", schema = "game")
@NamedQuery(name = Trainee.QUERY_FINDBY_PROGRESS, query = "SELECT
 trainee.progress FROM Trainee trainee")
@NamedQuery(name = Trainee.QUERY_FINDALLTRAINEES, query = "SELECT
 trainee FROM Trainee trainee ORDER BY trainee.MitID")
public class Trainee implements Serializable {
    public static final String QUERY_FINDBY_PROGRESS
 = "Trainee.findProgress";
    public static final String QUERY_FINDALLTRAINEES
 = "Trainee.findAll";
    @Id
    @Column(name="mitid")
    private String MitID;
    @NotNull @Column(name="vorname")
    private String vorname;
    @NotNull @Column(name="nachname")
    private String nachname;
    @NotNull @Column(unique = true, name="nickname")
    private String nickname;
    @Column(name="abteilung")
    private String abteilung;
    @Column(name="progress")
    private int progress;
    @Column(name="embleme")
    private String embleme;
    @OneToMany(fetch = FetchType.EAGER, mappedBy = "trainee")
    private List<Quizbeantwortung> quizbeantwortung;

//Getter, Setter
```

This table contains all trainees. It contains their first name, last name, the nickname they use on the website, their department in the company and their progress they have by doing quizzes.

EJBs

EJBs are components that summarize the business logic and take care of transactions and security. It is basically a connection to the database.

KursEJB.java.

```
/**
 *
 * @author Jan
 */
@Stateless
public class KursEJB {
    @PersistenceContext(unitName = "Diplomarbeit")
    private EntityManager em;

    public Kurs find(String KursID) {
        return em.find(Kurs.class, KursID);
    }

    public void update(Kurs kurs) {
        em.merge(kurs);
    }

    public void delete(int KursID) {
        em.getTransaction().begin();
        Kurs k = em.getReference(Kurs.class, KursID);
        em.remove(k);
        em.getTransaction().commit();
    }

    public List<Kurs> findAll() {
        return em.createNamedQuery(Kurs.QUERY_FINDALLKURSE,
 Kurs.class).getResultList();
    }
}
```

The main use of the EJB is to provide the courses page with all the courses within the database. This is done with the "findAll" methode.

FrageEJB.java.

```
/**
 *
 * @author Jan
 */
@Stateless
public class FrageEJB {
    @PersistenceContext(unitName = "Diplomarbeit")
    private EntityManager em;
    private Frage frage;

    public Frage find(String FID) {
        return em.find(Frage.class, FID);
    }

    public List<Frage> findAll() {
        return em.createNamedQuery(Frage.QUERY_FINDALLFRAGEN,
 Frage.class)
                .getResultList();
    }

    public void update(Frage f) {
        em.merge(f);
    }


    public void delete(int FID) {
        em.getTransaction().begin();
        Frage f = em.getReference(Frage.class, FID);
        em.remove(f);
        em.getTransaction().commit();
    }

    public List<Frage> findFrageByQID(String qid) {
        return em.createNamedQuery(Frage.QUERY_FINDFRAGENZUQID,
 Frage.class)
                .setParameter("QID", qid).getResultList();
    }

}
```

The use of the "FrageEJB" is for example to find all questions connected to a qid. This is used to show the questions when you take a quiz. Also it is available to only find one question or all questions in form of a list.

QuizEJB.java.

```
/**
 *
 * @author Jan
 */

@Stateless
public class QuizEJB {
    @PersistenceContext(unitName = "Diplomarbeit")
    private EntityManager em;

    public Quiz find(String QID) {
        return em.find(Quiz.class, QID);
    }

    public List<Quiz> findAll() {
        return em.createNamedQuery(Quiz.QUERY_FINDALL,
 Quiz.class).getResultList();
    }

    public void update(Quiz q) {
        em.merge(q);
    }

    public void delete(int QID) {
        em.getTransaction().begin();
        Quiz q = em.getReference(Quiz.class, QID);
        em.remove(q);
        em.getTransaction().commit();
    }

    public List<Quiz> findByBeschreibung(String beschreibung) {
        return em.createNamedQuery(Quiz.QUERY_FINDBY_BESCHREIBUNG,
 Quiz.class)
                .setParameter("beschreibung", "%"+beschreibung+"%")
                .getResultList();
    }
}
```

The function of this EJB is to find all quizzes from the database and return them in a list to the website. Also you can find single quizzes with the methode "find", you can delete and update quizzes.

TraineeEJB.java.

```
/**
 *
 * @author Jan
 */
@Stateless
public class TraineeEJB {
    @PersistenceContext(unitName = "Diplomarbeit")
    private EntityManager em;

    public Trainee find(String MitID) {
        return em.find(Trainee.class, MitID);
    }

    public List<Trainee> findAll() {
        return em.createNamedQuery(Trainee.QUERY_FINDALLTRAINEES,
 Trainee.class).getResultList();
    }

    public void update(Trainee t) {
        em.merge(t);
    }

    public void delete(int MitID) {
        em.getTransaction().begin();
        Trainee t = em.getReference(Trainee.class, MitID);
        em.remove(t);
        em.getTransaction().commit();
    }
}
```

The use of this class is to provide all the trainees deposited on the database to the associated web page. As in every other EJB there is an opportunity to find, update or delete a trainee.

AntwortmoeglichkeitenEJB.java.

```
/**
 *
```

```
 * @author Jan
 */
@Stateless
public class AntwortmoeglichkeitenEJB {
    @PersistenceContext(unitName = "Diplomarbeit")
    private EntityManager em;

    public Antwortmoeglichkeiten find(int AntwID) {
        return em.find(Antwortmoeglichkeiten.class, AntwID);
    }

    public void update(Antwortmoeglichkeiten antw) {
        em.merge(antw);
    }

    public void delete(int AntwID) {
        em.getTransaction().begin();
        Antwortmoeglichkeiten antw =
 em.getReference(Antwortmoeglichkeiten.class, AntwID);
        em.remove(antw);
        em.getTransaction().commit();
    }

    public List<Antwortmoeglichkeiten> findAntwortenByFID(String fid) {
        return
 em.createNamedQuery(Antwortmoeglichkeiten.QUERY_FINDANTWORTEN_BYFID,
 Antwortmoeglichkeiten.class)
                .setParameter("FID", fid).getResultList();
    }

}
```

This EJB provides the page where you take the quiz with all the corresponding answers to the questions. This is done by the methode "findAntwortenByFID" which starts a query when invoked. This query is defined in the "Antwortmoeglichkeiten" entity.

QuizVoraussetzungEJB.java.

```
/**
 *
 * @author Jan
 */
@Stateless
```

```
public class QuizVoraussetzungEJB {
    @PersistenceContext(unitName = "Diplomarbeit")
    private EntityManager em;

    public QuizVoraussetzung find(String QuizVoraussetzID) {
        return em.find(QuizVoraussetzung.class, QuizVoraussetzID);
    }



    public void update(QuizVoraussetzung QuizVoraussetzID) {
        em.merge(QuizVoraussetzID);
    }

    public void delete(String QuizVoraussetzID) {
        em.getTransaction().begin();
        QuizVoraussetzung quizvoraussetzung =
 em.getReference(QuizVoraussetzung.class, QuizVoraussetzID);
        em.remove(quizvoraussetzung);
        em.getTransaction().commit();
    }

    public List<QuizVoraussetzung> findAllQuizVoraussetzzungen(String
 qid) {
        return
 em.createNamedQuery(QuizVoraussetzung.QUERY_FINDALLVORAUSSETZUNGEN,
 QuizVoraussetzung.class)
                .setParameter("QID", qid).getResultList();
    }
}
```

The "QuizVoraussetzungsEJB" allocates all "QuizVoraussetzungen" with the "findAllQuizVoraussetzungen" methode. It is used in the "Voraussetzungs-Logic".

VoraussetzungEJB.java.

```
/**
 *
 * @author Jan
 */
@Stateless
public class VoraussetzungEJB {
    @PersistenceContext(unitName = "Diplomarbeit")
    private EntityManager em;
```

```
    public Voraussetzung find(String VoraussetzID) {
        return em.find(Voraussetzung.class, VoraussetzID);
    }

    public void update(Voraussetzung v) {
        em.merge(v);
    }

    public void delete(int VoraussetzID) {
        em.getTransaction().begin();
        Voraussetzung v = em.getReference(Voraussetzung.class,
 VoraussetzID);
        em.remove(v);
        em.getTransaction().commit();
    }
}
```

This EJB is used for the requirements logic. It contains a find methode where you can search for a requirement via a "VoraussetzID", which then returns a "Voraussetzung"-Entity. Also it has a update and delete methode.

QuizbeantwortungEJB.java.

```
/**
 *
 * @author Jan
 */
@Stateless
public class QuizbeantwortungEJB {
    @PersistenceContext(unitName = "Diplomarbeit")
    private EntityManager em;

    public Quizbeantwortung find(String qbeid) {
        return em.find(Quizbeantwortung.class, qbeid);
    }



    public void update(Quizbeantwortung quizbeantw) {
        em.merge(quizbeantw);
    }

    public void delete(String qbeid) {
        em.getTransaction().begin();
```

```
        Quizbeantwortung quizbeantw =
 em.getReference(Quizbeantwortung.class, qbeid);
        em.remove(quizbeantw);
        em.getTransaction().commit();
    }

    public List<Quizbeantwortung> findByQIDAndMITID(String qid, String
 mitid) {
        return
 em.createNamedQuery(Quizbeantwortung.QUERY_FINDBY_QIDANDMITID,
 Quizbeantwortung.class)
                .setParameter("QID", qid).setParameter("MitID",
 mitid).getResultList();
    }
}
```

The use of this bean is to provide information to all the quizzes a specific trainee has taken and when. This is done by a query, which is defined in the entity of this bean and it is required in the requirement logic.

Persistence.xml

persistence.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://java.sun.com/xml/ns/
persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://
xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
  <!-- Define Persistence Unit -->
  <persistence-unit name="Diplomarbeit" transaction-type="JTA">
    <jta-data-source>jdbc/postgrespool</jta-data-source>
    <class>org.game.trainee.kurs.Kurs</class>
    <class>org.game.trainee.kurs.KursBesuch</class>
    <class>org.game.trainee.testquiz.Antwort</class>
    <class>org.game.trainee.kurs.Voraussetzung</class>
    <class>org.game.trainee.kurs.KursVoraussetzung</class>
    <class>org.game.trainee.quiz.Quiz</class>
    <class>org.game.trainee.quiz.Frage</class>
    <class>org.game.trainee.quiz.Quizbeantwortung</class>
    <class>org.game.trainee.quiz.Antwortmoeglichkeiten</class>
    <class>org.game.trainee.quiz.QuizVoraussetzung</class>
    <class>org.game.trainee.trainee.Trainee</class>
    <properties>
```

```
      <!--<property name="javax.persistence.schema-
generation.database.action" value="drop-and-create"/>
      <property name="javax.persistence.schema-generation.create-source"
 value="script"/>
      <property name="javax.persistence.schema-generation.drop-source"
 value="script"/>
      <property name="javax.persistence.schema-generation.drop-script-
source" value="META-INF/dropSQL.sql"/>
      <property name="javax.persistence.schema-generation.create-script-
source" value="META-INF/createSQL.sql"/>
      <property name="javax.persistence.sql-load-script-source"
 value="META-INF/insertSQL.sql"/>  -->
    </properties>
  </persistence-unit>
</persistence>
```

The "persistence.xml" is used to configure many things, such as the source of the scipt used for dropping, if the database should always drop and then create all tables, and much more.

SQL-Files

In this chapter I will introduce all SQL-files used in this project int he order they are when the program is started.

dropSQL.sql.

```
DROP TABLE game.antwortmoeglichkeiten;

DROP TABLE game.frage;

DROP TABLE game.kurs CASCADE;

DROP TABLE game.kursbesuch;

DROP TABLE game.kursvoraussetzung;

DROP TABLE game.quiz CASCADE;

DROP TABLE game.quizbeantwortung;

DROP TABLE game.quizvoraussetzung;

DROP TABLE game.trainee;
```

```
DROP TABLE game.voraussetzung;
```

This script has the function of dropping every table before creating so there are no errors.

createSQL.sql.

```
CREATE SCHEMA IF NOT EXISTS game AUTHORIZATION darbeit;

CREATE TABLE game.antwortmoeglichkeiten(
    "antwid" text NOT NULL,
    "fid" text NOT NULL,
    "antwort" text COLLATE pg_catalog."default" NOT NULL,
    "richtigeantwort" boolean
);

CREATE TABLE game.frage(
    "fid" text NOT NULL,
    "qid" text NOT NULL,
    "frage" text COLLATE pg_catalog."default" NOT NULL,
    "punktezahl" bigint NOT NULL
);

CREATE TABLE game.kurs(
    "kursid" text NOT NULL,
    "titel" text COLLATE pg_catalog."default" NOT NULL,
    "link" text COLLATE pg_catalog."default" NOT NULL,
    "beschreibung" text COLLATE pg_catalog."default"
);

CREATE TABLE game.kursbesuch(
    "kursbesuchid" text NOT NULL,
    "kursid" text NOT NULL,
    "mitid" text NOT NULL,
    "datum" date NOT NULL
);

CREATE TABLE game.kursvoraussetzung(
    "kursvoraussetzid" text NOT NULL,
    "voraussetzid" text NOT NULL,
    "kursid" text NOT NULL
);

CREATE TABLE game.quiz(
```

```
    "qid" text NOT NULL,
    "titel" text COLLATE pg_catalog."default" NOT NULL,
    "beschreibung" text COLLATE pg_catalog."default",
    "reward" text COLLATE pg_catalog."default" NOT NULL,
    "multiplechoice" boolean
);

CREATE TABLE game.quizbeantwortung(
    "qbeid" text NOT NULL,
    "qid" text NOT NULL,
    "mitid" text NOT NULL,
    "erreichtepunktezahl" bigint NOT NULL,
    "istbestanden" boolean
);

CREATE TABLE game.quizvoraussetzung(
    "quizvoraussetzid" text NOT NULL,
    "voraussetzid" text NOT NULL,
    "qid" text NOT NULL
);

CREATE TABLE game.trainee (
    "mitid" text NOT NULL,
    "nachname" text COLLATE pg_catalog."default" NOT NULL,
    "vorname" text COLLATE pg_catalog."default",
    "nickname" text COLLATE pg_catalog."default" NOT NULL,
    "abteilung" text COLLATE pg_catalog."default",
    "progress" bigint NOT NULL,
    "embleme" text COLLATE pg_catalog."default"
);

CREATE TABLE game.voraussetzung(
    "voraussetzid" text NOT NULL,
    "kursid" text,
    "qid" text
);

ALTER TABLE game.antwortmoeglichkeiten ADD CONSTRAINT
 "ANTWORTMOEGLICHKEITEN_pkey" PRIMARY KEY ("antwid");

ALTER TABLE game.frage ADD CONSTRAINT "FRAGE_pkey" PRIMARY KEY ("fid");

ALTER TABLE game.kurs ADD CONSTRAINT "KURS_pkey" PRIMARY KEY ("kursid");

ALTER TABLE game.kursbesuch ADD CONSTRAINT "KURSBESUCH_pkey" PRIMARY KEY
 ("kursbesuchid");
```

```
ALTER TABLE game.kursvoraussetzung ADD CONSTRAINT
 "KURSVORAUSSETZUNG_pkey" PRIMARY KEY ("kursvoraussetzid");

ALTER TABLE game.quiz ADD CONSTRAINT "QUIZ_pkey" PRIMARY KEY (qid);

ALTER TABLE game.quizbeantwortung ADD CONSTRAINT "QUIZBEANTWORTUNG_pkey"
 PRIMARY KEY ("qbeid");

ALTER TABLE game.quizbeantwortung ADD CONSTRAINT "QIDMITID" UNIQUE (qid,
 mitid);

ALTER TABLE game.quizvoraussetzung ADD CONSTRAINT
 "QUIZVORAUSSETZUNG_pkey" PRIMARY KEY ("quizvoraussetzid");

ALTER TABLE game.trainee ADD CONSTRAINT "TRAINEE_pkey" PRIMARY KEY
 ("mitid");

ALTER TABLE game.trainee ADD CONSTRAINT "NICKNAME" UNIQUE ("nickname");

ALTER TABLE game.voraussetzung ADD CONSTRAINT "VORAUSSETZUNG_pkey"
 PRIMARY KEY ("voraussetzid");

ALTER TABLE game.antwortmoeglichkeiten ADD CONSTRAINT "Frage" FOREIGN
 KEY ("fid")
        REFERENCES game.frage ("fid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;

ALTER TABLE game.frage ADD CONSTRAINT "Quiz" FOREIGN KEY ("qid")
        REFERENCES game.quiz ("qid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;

ALTER TABLE game.kursbesuch ADD CONSTRAINT "Kurs" FOREIGN KEY ("kursid")
        REFERENCES game.kurs ("kursid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;

ALTER TABLE game.kursbesuch ADD CONSTRAINT "Trainee" FOREIGN KEY
 ("mitid")
        REFERENCES game.trainee ("mitid") MATCH SIMPLE
        ON UPDATE CASCADE
```

```
        ON DELETE CASCADE
        NOT VALID;

ALTER TABLE game.kursvoraussetzung ADD CONSTRAINT "Kurs" FOREIGN KEY
 ("kursid")
        REFERENCES game.kurs ("kursid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;

ALTER TABLE game.kursvoraussetzung ADD CONSTRAINT "Voraussetzung"
 FOREIGN KEY ("voraussetzid")
        REFERENCES game.voraussetzung ("voraussetzid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;

ALTER TABLE game.quizbeantwortung ADD CONSTRAINT "Quiz" FOREIGN KEY
 ("qid")
        REFERENCES game.quiz ("qid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;


ALTER TABLE game.quizbeantwortung ADD CONSTRAINT "Trainee" FOREIGN KEY
 ("mitid")
        REFERENCES game.trainee ("mitid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;

ALTER TABLE game.quizvoraussetzung ADD CONSTRAINT "Quiz" FOREIGN KEY
 ("qid")
        REFERENCES game.quiz ("qid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;

ALTER TABLE game.quizvoraussetzung ADD CONSTRAINT "Voraussetzung"
 FOREIGN KEY ("voraussetzid")
        REFERENCES game.voraussetzung ("voraussetzid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;
```

```
ALTER TABLE game.voraussetzung ADD CONSTRAINT "Kurs" FOREIGN KEY
 ("kursid")
        REFERENCES game.kurs ("kursid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;

ALTER TABLE game.voraussetzung ADD CONSTRAINT "Quiz" FOREIGN KEY ("qid")
        REFERENCES game.quiz ("qid") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID;
```

This file is used to create all the tables we are using. It produces every table with its associated columns. This file is also used to give all the tables its associated primary keys and foreign keys.

insertSQL.sql.

```
INSERT INTO game.trainee ("mitid", "nachname", "vorname", "nickname",
 "abteilung", "progress", "embleme") VALUES ('1', 'Binder', 'Jan',
 'Syreax', 'Projektmanagement', 500, NULL);

INSERT INTO game.quiz ("qid", "titel", "beschreibung", "reward",
 "multiplechoice") VALUES ('1', 'Start', 'Quiz about the general
 knowledge of Java', 'test', false);

INSERT INTO game.frage ("fid", "qid", "frage", "punktezahl") VALUES
 ('1', '1', 'What are advantages of Java?', 10);

INSERT INTO game.antwortmoeglichkeiten ("antwid", "fid", "antwort",
 "richtigeantwort") VALUES ('1', '1', 'Flawless', false);

INSERT INTO game.quizbeantwortung ("qbeid", "qid", "mitid",
 "erreichtepunktezahl", "istbestanden") VALUES ('1', '1', '1', 0,
 false);

INSERT INTO game.kurs ("kursid", "titel", "link", "beschreibung") VALUES
 ('1', 'Start-Kurs', 'https://www.tutorialspoint.com/java/index.htm',
 'Dieser Kurs ist der erste Kurs der abgelegt werden muss. Er erklärt
 dir die Basics von Java.');

INSERT INTO game.voraussetzung("voraussetzid", "kursid", "qid") VALUES
 ('1', null, '1');
```

```
INSERT INTO game.quizvoraussetzung("quizvoraussetzid", "voraussetzid",
 "qid") VALUES ('1', '1', '2');
```

Last there is the "insertSQL" script, which is used to insert all the data we want into the belonging table. The order of the commands is very important because, if you insert "antwortmoeglichkeiten" at first there would be no existing matching "FNR" or with "frage " no fitting "QID". Here are only the first insert of every table because otherwise there would be too much code, which would make it too complex.