Table of Contents

Praktische Aufgabenstellung: Eric Haneder

Front End: Design und Graphic User Interface

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.

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 <br>, <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;
}
```

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

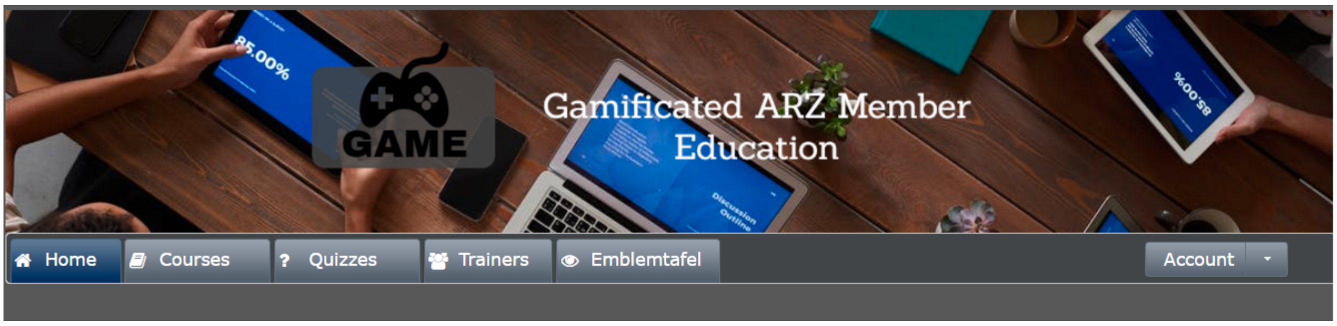
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" ①
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      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> ②
        </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>
```

① Implementation of taglibrarys

② 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.

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" ①
                    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">
②
    <ui:define name="header"> ③
        <title>Home</title>
        <h:outputStylesheet library="css" name="stylesheet.css" />
    </ui:define>

    <ui:define name="content"> ③
        <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(&quot;1
&quot;).get(0).embleme.get(0)}">
            </h:graphicImage>
            <br/><br/>
            Dein gesamter Fortschritt:
            <h:outputText value="#{traineeController.getTraineesByID(&quot;1
&quot;).get(0).progress}" />
            <br/><br/>
            Hier sollte spezifisch zum User, dessen Aktivitäten und Fortschritt
dargestellt werden.
        </p>
        <br />
    </ui:define>
</ui:composition>
```

① As you can see, the index-page is using the mainlayout with the "template" attribute.

② Everything above this callout will be same for every template client. I will therefore only include it here, to shorten the other code-snippets.

③ 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.

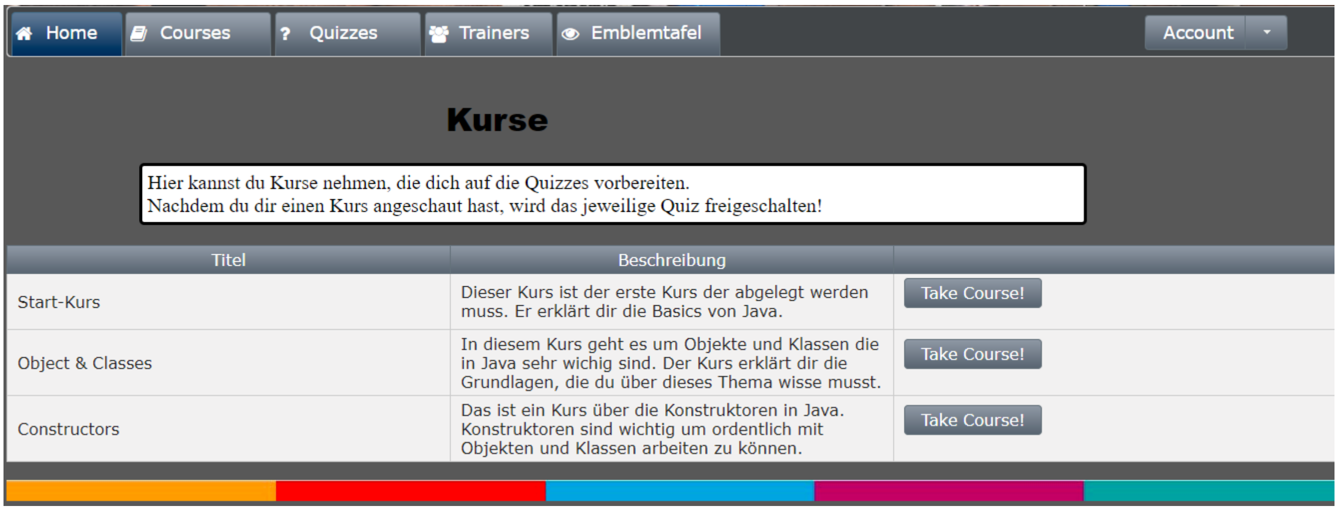
courses.xhtml

```
    <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}"> ①
            <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!" >  ②
                    </p:commandButton>
                </h:form>
            </p:column>
        </p:dataTable>
    </ui:define>
```

① 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.

② 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:



Kurse

Hier kannst du Kurse nehmen, die dich auf die Quizzes vorbereiten.
Nachdem du dir einen Kurs angeschaut hast, wird das jeweilige Quiz freigeschalten!

Titel	Beschreibung	
Start-Kurs	Dieser Kurs ist der erste Kurs der abgelegt werden muss. Er erklärt dir die Basics von Java.	Take Course!
Object & Classes	In diesem Kurs geht es um Objekte und Klassen die in Java sehr wichig sind. Der Kurs erklärt dir die Grundlagen, die du über dieses Thema wisse musst.	Take Course!
Constructors	Das ist ein Kurs über die Konstruktoren in Java. Konstruktoren sind wichtig um ordentlich mit Objekten und Klassen arbeiten zu können.	Take Course!

Quiz pages

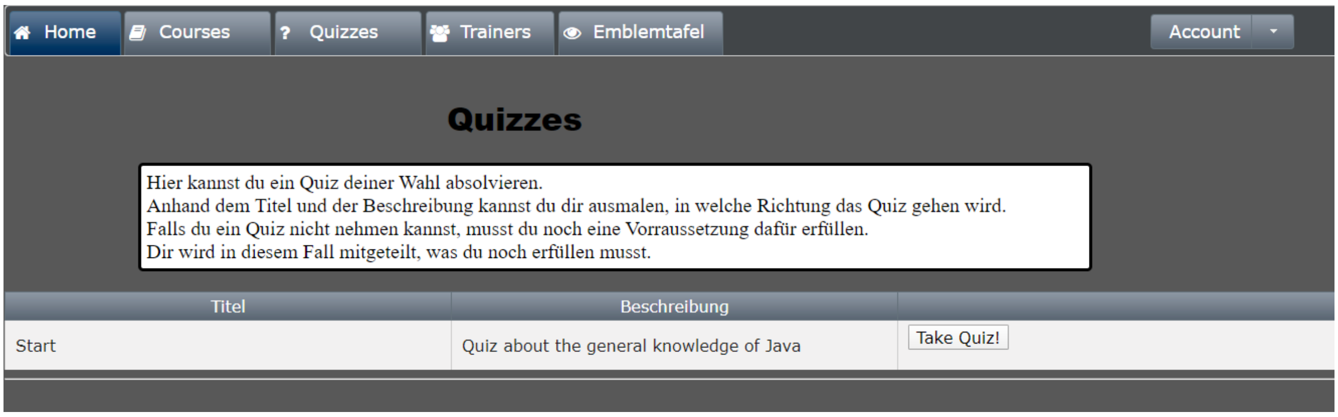
The quiz pages include a interface, where every takeable quiz is displayed, two pages for taking a quiz and a page where the results are shown. Every quiz has its own emblem, which can be won if they quiz is taken succesfully. This means the user has to has at least half of the questions right.

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, &quot;1&quot;)}">
                <h:outputText value="#{quiz.titel}" />
            </p:column>
            <p:column headerText="Beschreibung"
            rendered="#{quizController.isTakeable(quiz.QID, &quot;1&quot;)}">
                <h:outputText value="#{quiz.beschreibung}" />
            </p:column>
            <p:column
            rendered="#{quizController.isTakeable(quiz.QID, &quot;1&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:

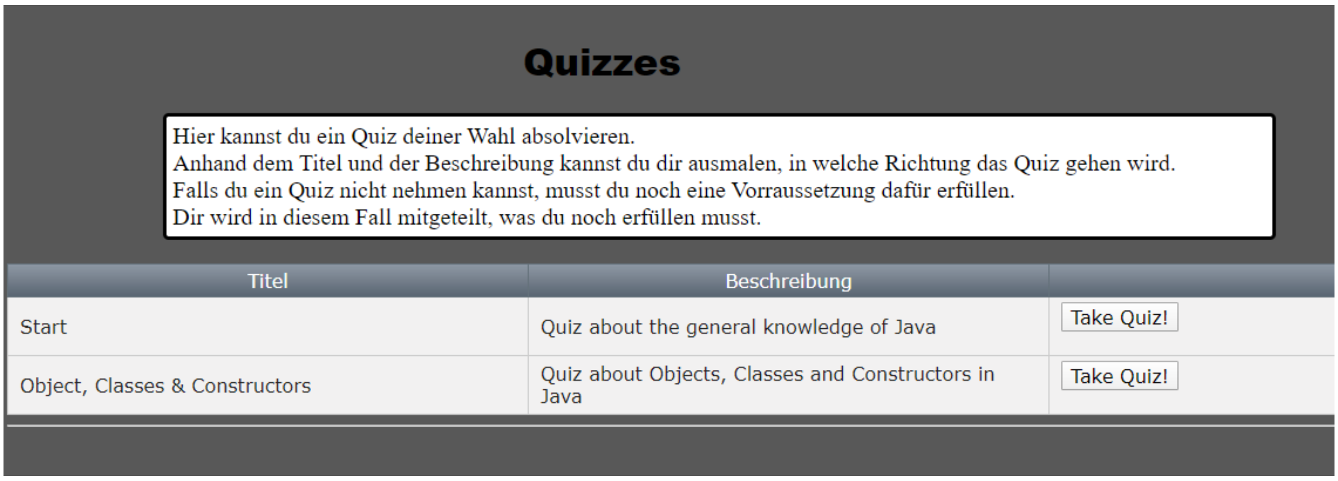


Quizzes

Hier kannst du ein Quiz deiner Wahl absolvieren.
Anhand dem Titel und der Beschreibung kannst du dir ausmalen, in welche Richtung das Quiz gehen wird.
Falls du ein Quiz nicht nehmen kannst, musst du noch eine Vorraussetzung dafür erfüllen.
Dir wird in diesem Fall mitgeteilt, was du noch erfüllen musst.

Titel	Beschreibung	
Start	Quiz about the general knowledge of Java	Take 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.



Quizzes

Hier kannst du ein Quiz deiner Wahl absolvieren.
Anhand dem Titel und der Beschreibung kannst du dir ausmalen, in welche Richtung das Quiz gehen wird.
Falls du ein Quiz nicht nehmen kannst, musst du noch eine Vorraussetzung dafür erfüllen.
Dir wird in diesem Fall mitgeteilt, was du noch erfüllen musst.

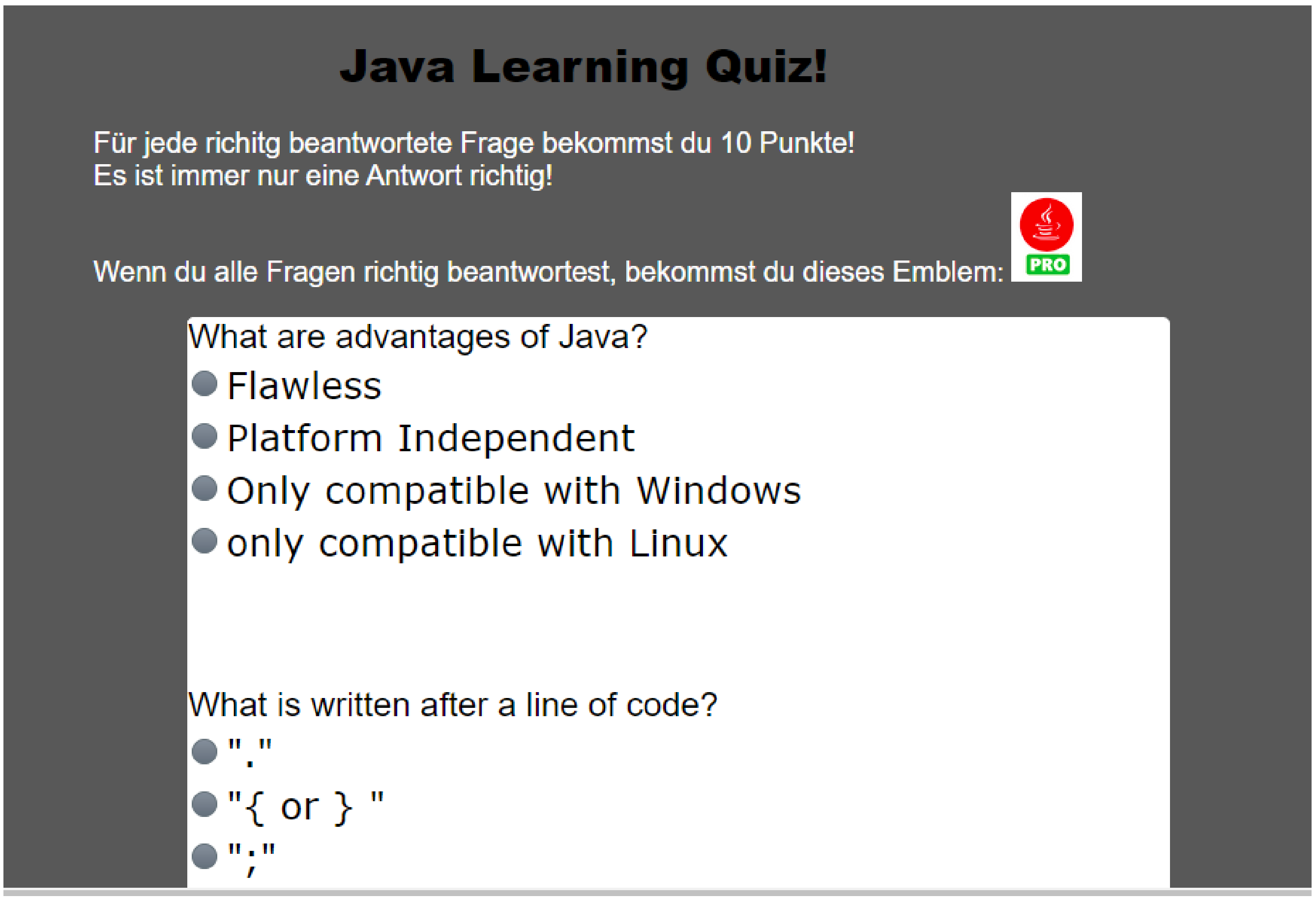
Titel	Beschreibung	
Start	Quiz about the general knowledge of Java	Take Quiz!
Object, Classes & Constructors	Quiz about Objects, Classes and Constructors in Java	Take Quiz!

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="true" unselectable="true" columns="1">

                        <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.

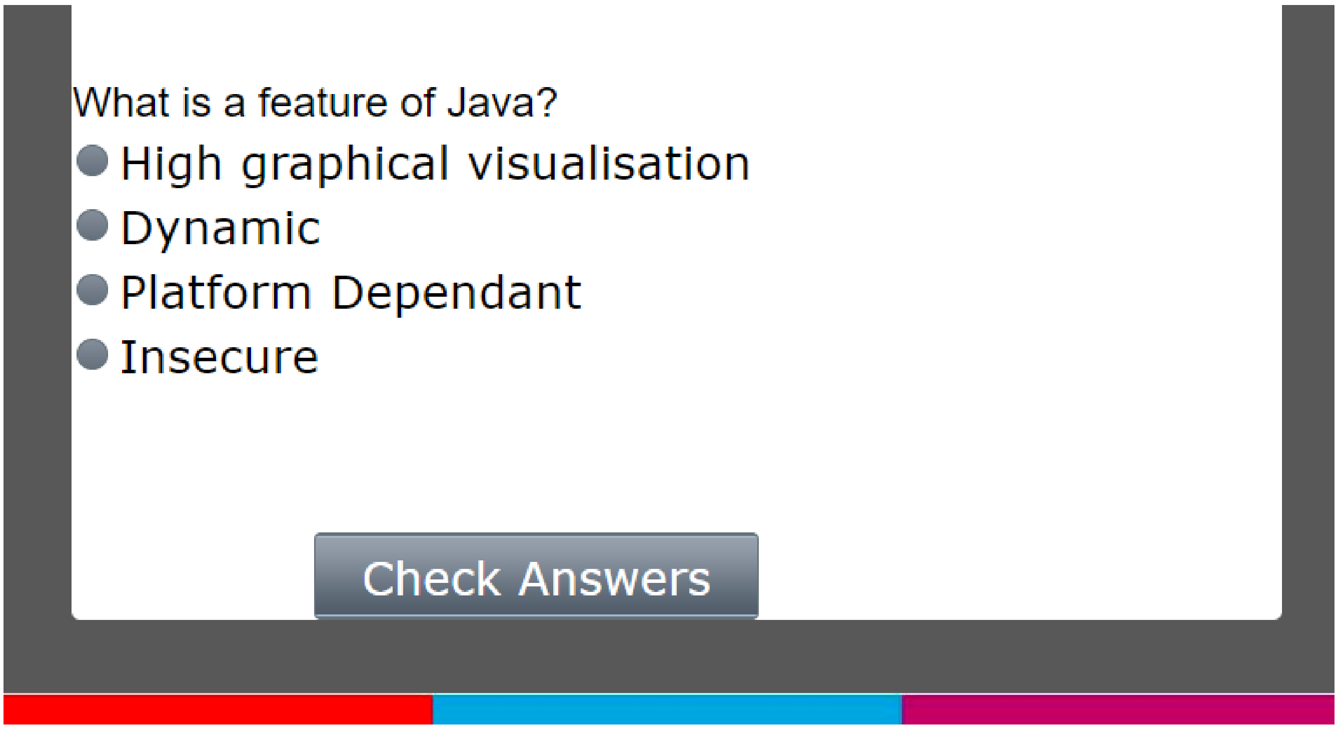

Java Learning Quiz!

Für jede richitg beantwortete Frage bekommst du 10 Punkte!
Es ist immer nur eine Antwort richtig!

Wenn du alle Fragen richtig beantwortest, bekommst du dieses Emblem:

What are advantages of Java?
- Flawless
- Platform Independent
- Only compatible with Windows
- only compatible with Linux

What is written after a line of code?
- "."
- "{ or } "
- ";"


What is a feature of Java?
- High graphical visualisation
- Dynamic
- Platform Dependant
- Insecure

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

takeQuizMultipleChoice.xhtml

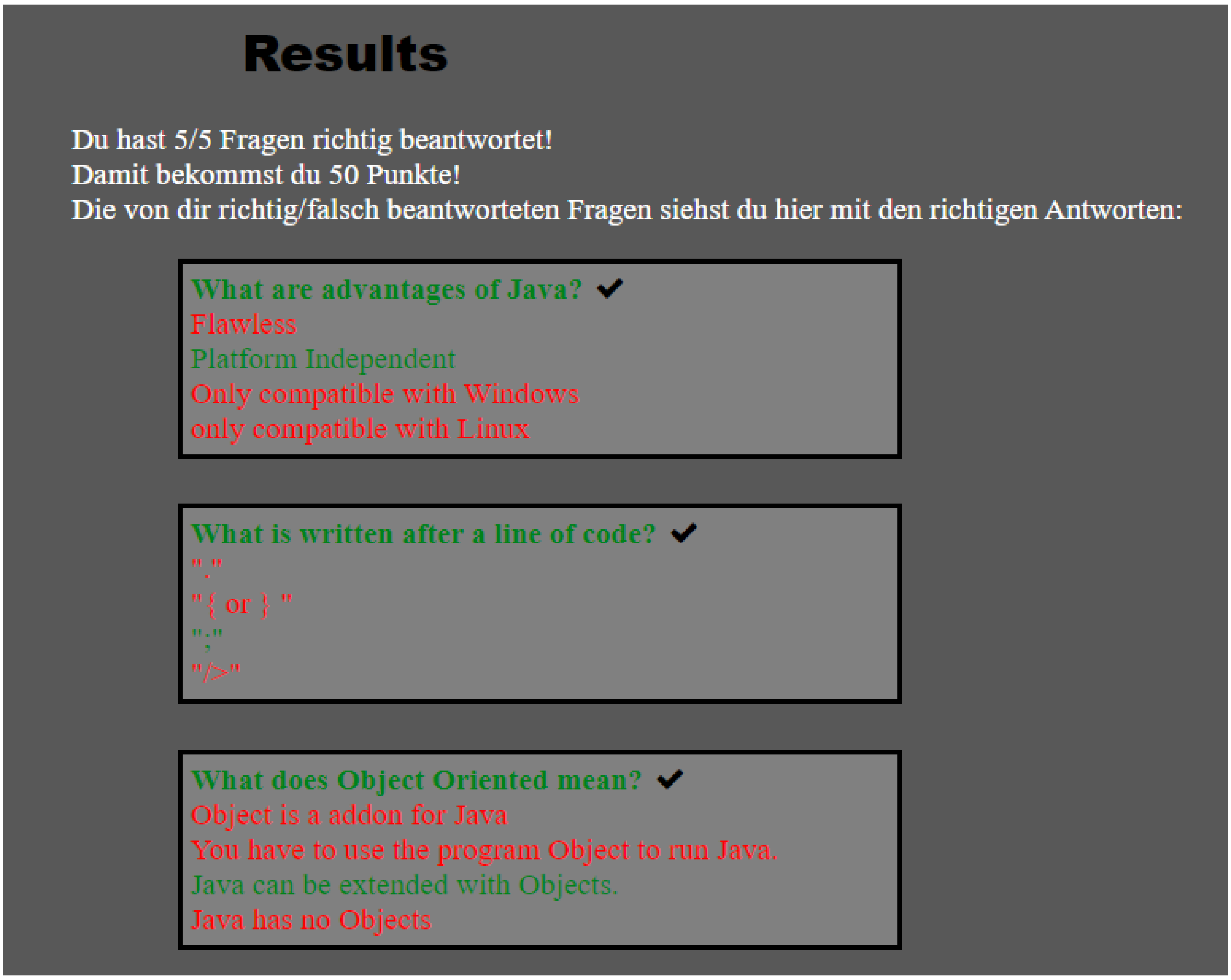
```
    <ui:define name="header">
        <title>Test - JavaQuiz</title>
        <h:outputStylesheet library="css" name="stylesheet.css"></h:outputStylesheet>
    </ui:define>
    <ui:define name="content">
        <h1 align="center">Java Learning Quiz!</h1>
        <p style="margin-left: 20%; font-family: '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>
```


Results

Du hast 5/5 Fragen richtig beantwortet!
Damit bekommst du 50 Punkte!
Die von dir richtig/falsch beantworteten Fragen siehst du hier mit den richtigen Antworten:

What are advantages of Java? ✓
Flawless
Platform Independent
Only compatible with Windows
only compatible with Linux

What is written after a line of code? ✓
"."
" { or } "
";"
"/>"

What does Object Oriented mean? ✓
Object is a addon for Java
You have to use the program Object to run Java.
Java can be extended with Objects.
Java has no Objects

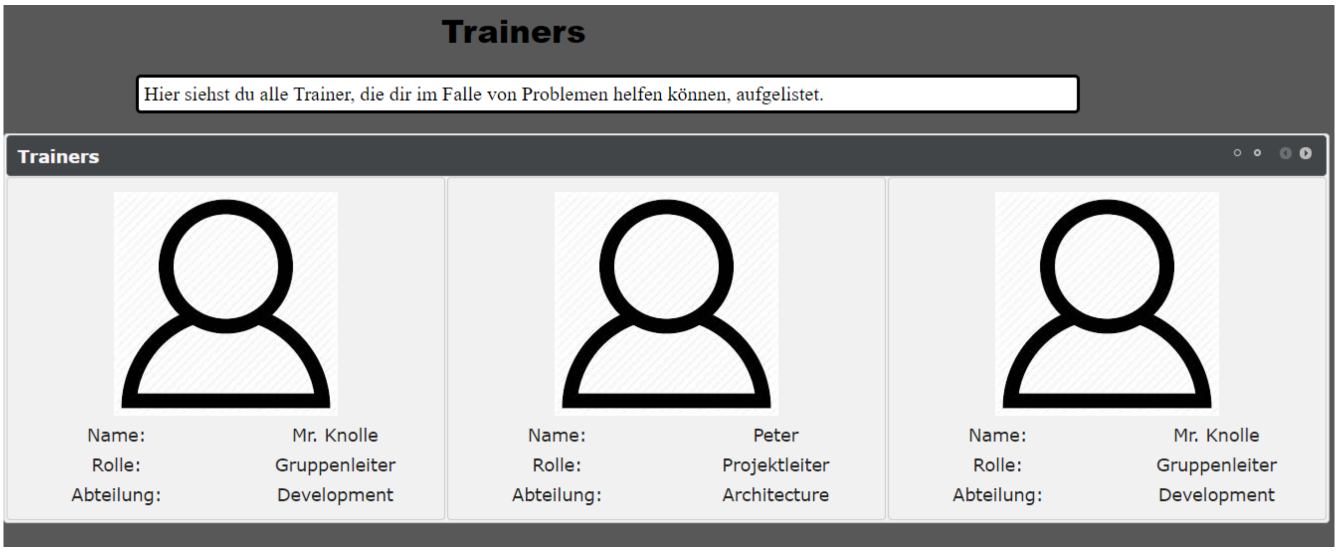
Trainer page

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>
```

① 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:


Trainers

Hier siehst du alle Trainer, die dir im Falle von Problemen helfen können, aufgelistet.

Trainers

Name:	Mr. Knolle	Name:	Peter	Name:	Mr. Knolle
Rolle:	Gruppenleiter	Rolle:	Projektleiter	Rolle:	Gruppenleiter
Abteilung:	Development	Abteilung:	Architecture	Abteilung:	Development

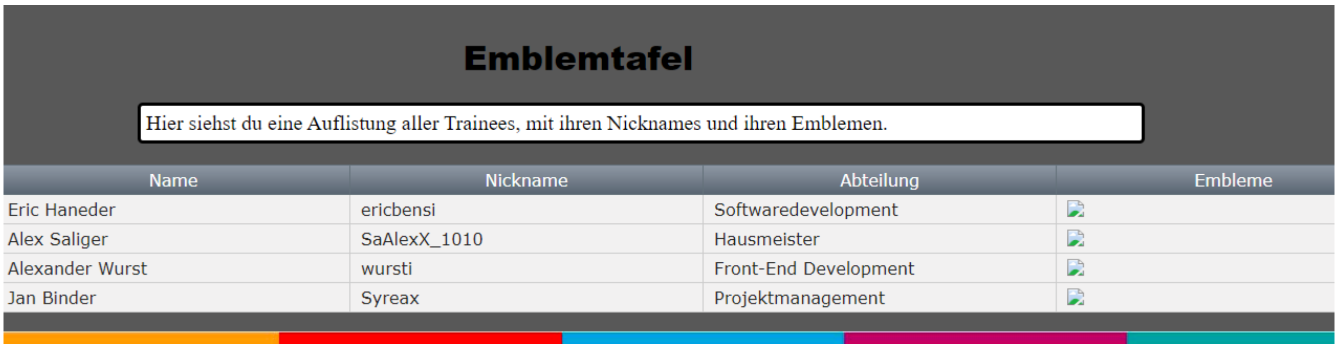
Leaderboard page

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="data:image/png;base64,#{trainee.embleme.get(0)}">
                </h:graphicImage>
            </p:column>
        </p:dataTable>
    </ui:define>
```

① 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.



Emblemtafel

Hier siehst du eine Auflistung aller Trainees, mit ihren Nicknames und ihren Emblemen.

Name	Nickname	Abteilung	Embleme
Eric Haneder	ericbensi	Softwaredevelopment	
Alex Saliger	SaAlexX_1010	Hausmeister	
Alexander Wurst	wursti	Front-End Development	
Jan Binder	Syreax	Projektmanagement	

Java Classes

Trainer Classes

TrainerController.java

```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; ①
    @Inject
    private QuizbeantwortungEJB quizbeantw;
    @Inject
    private QuizVoraussetzungEJB quizvoraussetzung;
    @Inject
    private VoraussetzungEJB voraussetzungejb;

    @PostConstruct ②
    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() { ③
        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() { ④
        List<Integer> falsche = new ArrayList<>();
        for(int i=0; i<fragemodell.size(); i++) {
            if(fragemodell.get(i).selectedAnswer.equals(fragemodell.get(i).antworten
.get(fragemodell.get(i).indexrichtig))) {
                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
}
```

① 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.

② The annotation *PostContruct* functions as a note to the container, that this method must be run, before the construction of the Class.

③ 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.

④ 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.

Summary

"was ghört hir hin?"