Hochschule für Wirtschaft und Gesellschaft Ludwigshafen

Fachbereich III Dienstleistung & Consulting

Praktikum Anwendungssysteme

Java Online Course

Betreuer:

Halo Röckle

Vorgelegt von:

Zhanna Belloni

Dmytro Poliskyi

# Inhalt

[Inhalt 2](#_Toc115020697)

[1 Überblick über die Webanwendung 3](#_Toc115020698)

[1.1 Zweck und den Nutzen 3](#_Toc115020699)

[1.2 Setup und Starten der Anwendung 3](#_Toc115020700)

[2 Architektur des Systems 4](#_Toc115020701)

[3 Die Bestandteile der Anwendung 4](#_Toc115020702)

[3.1 Administrative Sektion und Databank 4](#_Toc115020703)

[3.1.1 Datenbank 5](#_Toc115020704)

[3.1.2 Übungen 6](#_Toc115020705)

[3.2 Bean Klassen 6](#_Toc115020706)

[3.3 Database Connection Utility 6](#_Toc115020707)

[3.4 Externe Service Helper: Prüfung der Übung 7](#_Toc115020708)

[3.5 Webpage 8](#_Toc115020709)

[3.5.1 Striktur eine Seite. 8](#_Toc115020710)

[3.5.2 Home Page 9](#_Toc115020711)

[3.5.3 Lektionen 9](#_Toc115020712)

[3.5.4 Übungen 10](#_Toc115020713)

[3.5.5 Comments 11](#_Toc115020714)

[3.5.6 Contacts 11](#_Toc115020715)

[3.5.7 Login und Registrierung 11](#_Toc115020716)

[4 Abgrenzungserklärung 12](#_Toc115020717)

[4.1 Die Webanwendung: 12](#_Toc115020718)

[4.2 Die technische Dokumentation: 12](#_Toc115020719)

[4.3 Die Präsentation: 12](#_Toc115020720)

# Überblick über die Webanwendung

## Zweck und den Nutzen

Es ist eine Ressource, mit der man das Programmieren von Anfang an lernen kann. Die Theorie hilft bei den praktischen Aufgaben. Es gibt auch ein Benotungssystem für Aufgaben, so dass man sich jederzeit selbst testen kann. Wenn sie Fragen zu einer Aufgabe haben, können sie die richtige Antwort sehen. Was sind die Vorteile unseres Systems? Sie ist recht einfach: eine einfache Schnittstelle für den Benutzer und ein einfaches Testkonstruktionssystem für Lehrer (vorausgesetzt, jemand nutzt diese Website als Lernhilfe). Der Benutzer hat Zugang zu fast allem auf der Website, so dass man sich leicht verirren und verwirren kann. Aber wegen der leichten Verständlichkeit und der fehlenden Schwierigkeit, etwas auf der Website zu finden, ist der Benutzer nicht verloren, sondern genießt alle Vorteile der Website. Und die Lehrer müssen nur die Aufgaben durcharbeiten und die Theorie vermitteln. All dies zusammen ergibt ein hervorragendes System.

Der Nutzer, der auf Startseite Landet kann login, falls er ist schon registriert, ein neue Account anlegen oder navigieren durch die ‘frei’ Seiten.

Die Webanwengung bestehet von 3 Größe Teile:

1. die Course: eine Reihenfolge von Seiten, wo man Java sowohl in Theorie als auch in Praktikum lernt
2. die Übungen: wo man sich trainieren kann und seine Fähigkeiten prüfen
3. eine Sektion für Kommentare

Einige Teile von der Webanwendung sind nur für angemeldete Benutzer erreichbar, so ein Login- und eine Registrierungsseite ist vorhanden.

## Setup und Starten der Anwendung

Die Anwendung benötig folgende externe Software:

1. ein Webserver: *Apache TomCat (apache-tomcat-10.0.23)*
2. eine Databank: *postgreSQL*
3. *JDBC driver für postgreSQL (postgresql-42.2.26.jre7.jar)*

Man kann den Projekt in Eclipse öffnen und von dort bequem alle starten.

**Erstens**, sollte man den Apach TomCat einsetzen, postgres jdbc driver in die build path hinzufügen und, falls notwendig, die postgres jdbc driver in das **lib** Verzeichnis von TomCat kopieren.  
Um den Anwendung zu benutzen, sind es folgende 4 Schritte notwendig:

1. Konfiguration der Verbindung zur Datenbank: man muss den File PostgreSQLAccess.java in package **de.hwg\_lu.java\_star.jdbc** anpassen.
2. Initialisierung der Applikationen:
   * Run als Java Applikation den File

**/JavaStarWebsite/src/main/java/de/hwg\_lu/java\_star/admin/AdminDB.java**  
mit Kommandozeilenargumente den vollständigen Pfad aus dem Verzeichnis **data/,** der sich im root von diesem Projekt befindet.

* + Sehen [Administrative Sektion und Databank](#_Administrative_Sektion_und)  für mehr tools für administrative Arbeit.

1. Start den TomCat
2. Run on Server den File  
   JavaStarWebsite/jsp/HomePageView.jsp  
   oder einfach rufen <http://localhost:8180/JavaStarWebsite/jsp/HomePageView.jsp> in ein Browser.

Enjoy!

# Architektur des Systems

Das Architektursystem ist über einen Webserver gebaut, der Java Servlets und JavaServer Pages unterstützt: in unserem Fall ist das Apache TomCat.

Dieser Webserver generiert dynamische Webseiten mit Hilfe in einer Databank gespeicherten Informationen: PostgreSQL.

Die Anwendung benutzt eine externe Service - compiler explorer (<https://godbolt.org/>).

Diese Ressource muss auch erreichbar sein.

WebBrowser

Web Server

Database

<https://godbolt.org/>

Request

Response

Servlet

Die Anwendung basiert sich auf Theorie und Übungen: da es in einen Kurs immer möglich neuen Lernstoff hinzufügt werden kann, war es besonders auf einfache Implementierung aufgepasst.

# Die Bestandteile der Anwendung

Die Anwendung besteht aus 5 Teilen:

1. administrative Sektion
2. Bean Klassen
3. Database Connection Utility
4. External Service Helper
5. Webpages

Nebenbei gibt es eine Reihenfolge von Helper Klassen, die man in die package **de.hwg\_lu.java\_star.utils** finden kann.

Das Projekt verwendet javadoc Annotationen: es ist möglich die Codedokumentation von Java-Quellcode im HTML-Format zu generieren.

## Administrative Sektion und Databank

Die administrative Sektion stellt die Klasse in package **de.hwg\_lu.java\_star.admin** bereit, um das System zu initialisieren oder reinigen :

1. Initialisierung der Datenbank mit Schema und alle benötigte Tabellen (sehen [Datenbank](#_Databank)).
2. Anlegung von einem Admin User, mit besonderer Privilegien (user: admin, psswrd: admin)
3. Herunterladung aller notwendigen Daten in die Datenbank (sehen [Übungen](#_Übungen)).
4. Es gibt 3 Utility die als Java Program laufen lassen können:
   1. *AdminDB.java* <pfard zu data Ordner> Initialisiert das System
   2. *AdminViewer.java* gibt die Möglichkeit, zu schauen und den Inhalt aller Tabellen

zu prüfen.

* 1. *AdminDBCleaner.java* löscht alle Daten aus System, i.e. löschen alle Accounts und Statistiker, die meisten während der Entwicklung der Applikation benutzt werden).

### Datenbank

Die Anwendung benötigt einige Tabellen, um korrekt zu funktionieren:

Die Klasse **AdminDB** legt ein Schema mit dem gegebene Name (definiert in die Klasse PostgreSQLAccess in package **de.hwg\_lu.java\_star.jdbc**) an und die folgende Tabelle:

**TABLE: account**

Wenn ein Benuzter sich registriert, werden alle dazu notwendige Daten hier gespeichert.

Und wenn ein Benutzer sich einmeldet, werden alle Daten hier geprüft.[[1]](#footnote-1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| userid CHAR(32) | email CHAR(32) | Password CHAR(16) | active CHAR(1) | admin CHAR(1) |
| PRIMARY KEY |  |  |  |  |

**TABLE: comments**

Ein angemeldeter User kann Kommentare hinterlassen: diese Tabelle spechert es mitte in serial id, ein timestamp, der username und den Kommentar selbt. Der Admin kann ein Kommentar löschen, in dem das hinterlassene Kommentar wird mit ‘comment deleted by the admin’ überschrieben.

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Time TIMESTAMP | userid CHAR(32) | Comment CHAR(512) |
| SERIAL PRIMARY KEY |  |  |  |

**TABLE: exercise**

Alle Informationen für die Übungen (Übungsaufgaben, Lösungen usw. werden in dieser Tabelle gespeichert (sehen [Übungen](#_Übungen))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id  INTEGER | exercise\_text VARCHAR(1024) | exercise\_out  VARCHAR(1024 | exercise\_solution  VARCHAR(1024 | exercise\_test VARCHAR(1024 |
| PRIMARY KEY |  |  |  |  |

**TABLE: statistics**

Ein angemeldeter Benutzer kann Übungen lösen: in folgender Tabelle wird die Statistik für die Users Achievements gespeichert, und zwar welche Übungen hat der User schon gelöst und mit welcher Leistung.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| userid CHAR(32) | exerciseid INTEGER | tried\_to\_solved BOOL | compile\_error BOOL | test\_error BOOL |
| FOREIGN KEY (userid) REFERENCES account(userid) |  |  |  |  |

### Übungen

Die Übungen ergeben den komplexen des Systems:

Der Benutzer muss ein Java Program schreiben, das in der Aufgabe beschrieben ist:

die Anwendung akzeptiert einen freien Text und sorgt für Compilierung und überprüft ob es korrekt ist.

Bei Initialisierung werden alle benötigte Informationen aus txt-Datei ausgelesen und in die Databank, in die Tabelle *exercises*, geladen:

Jedes Teil ist in einem eigenen File gespeichert, die hat folgende Strukture:

**exercise\_<id-übung>\_<Spezifikation>.txt**

1. Text der Ausgabe: Spezifikation=instructions -> in spalte exercise\_text
2. Erwartetes Output: Spezifikation=out -> in spalte exercise\_out
3. Lösung der Aufgabe: Spezifikation=solution -> in spalte exercise\_solution
4. Class zu testen von einer Lösung: Spezifikation: test -> in spalte exercise\_test

Neue Übungen können hinzugefügt werden, solang die entsprechenden Strukturen haben und müssen in dem Ordner **data** gespeichert werden.

## Bean Klassen

In package **de.hwg\_lu.java\_star.beans** gibt es folgende Klassen:

* **AccountBean**: speichert Username, Password, Email und die Flag admin und active Benutzer.
* **LoginBean**: falls der User angemeldet ist, speichert die Informationen über einen angemeldeten Benutzer.
* **MessageBean**: speichert Error Messages, die die Anwendung einem Benutzer entsprechend zeigt.
* **GiuBean**: es ist ein Utility, um bestimmte Teile der Webpages zu generieren d.h. Navigation und Lektion Pages.
* **ExerciseResultBean**: speichert alle Informationen über eine gelöste Übung d.h. die Übung ID, die Input Source Code, die Compilierung Fehler oder einen

Fehler in Test, falls es gibt, und das Ergebnis des Tests.

## Database Connection Utility

In package **de.hwg\_lu.java\_star.jdbc** gibt es alle Klasse, die die Verbindung zur Databank vereinfachen. Da verschiedene Tabelle vorhanden sind, es gibt eine Klasse, die speichert, liest oder Werte in einer bestimmten Tabelle update:

* Klasse, die zur Verbindung mit der Datenbank dienen:
  + **JDBAccess.java:** Utility, um Connections zu erzeugen
  + **PostgreSQLAccess.java:** Connection Username und Passwort, Schema Name und End Point für die Driver postgres zu setzten.
  + **NoCollectionException.java:** Exception Klasse, falls es Probleme mit den Verbindung zu Datenbank entstehen.
* Klasse, die eine spezifische Aufgabe erledigt:
  + **CommentDB.java:** Klasse Kommentare der Benutzer zu bearbeiten: es sorgt für neue Kommentare, liest Kommentare oder markiert als deleted.
  + **ExericiseDB.java:** Klasse zu zurückholen Informazionen für die Bearbeitung ein Übung, d.h. Anzahl von Übungen, Übungausgabe, Test Code und Lösung.
  + **StatisticDB.java:** Klasse, die die Statistik von Benutzerleistungen update:die speichert das Ergebnis von einer Übung und ergibt die Ergebnisse als hex Farbe.

## Externe Service Helper: Prüfung der Übung

Die Klasse **Tester** in Package **de.hwg\_lu.java\_star.excercises** ermöglicht External Service compile-explorer (<https://godbolt.org/>) zu benutzen.

Diese Klasse bietet 2 primäre Funktionalitäten an:

1. compileExcercise(String sourceCode, **int** excerciseNumber)
2. testExcercise(String sourceCode, **int** excerciseNumber)

**compileExcercise** kompiliert die angegebene **String** sourceCode und gibt das Ergebnis des Requests zurück, **testExcercise** integriert den Input des Benutzers in die TestCode aus die spalte exercise\_test in Tabelle exercises und gibt das Ergebnis von dem Test zurück.

In der Abbildung unten kann man sehen, wie das Request gebaut und gesendet ist.

* 1. Lesen Input von User.
  2. Löschen alle newlines, tabs und komments in Quelle Code.
  3. Bauen ein JSON con die Request, wo die key „source“ enthalt die gereinigte Input.
     1. Falls mann will nur kompilieren, dann die input ist gesendet zu der extern Service
     2. Falls die Kompilierung war Erfolgreich, dann mann nimmt die test Klasse aus die exercise Tabelle, und ersetzt die TAG „%\_INSERT\_CODE\_HERE\_%” mit dem Benutzer Input.
  4. Man sendet die Request und Wartet auf ein Response.

// saved in TABLE   
// exercise, column exercise\_test

class Tester {

%\_INSERT\_CODE\_HERE\_%

void static public main(String[] args)   
 {  
 // code used for testing  
 }

}

class UserInputSourceCode {

}

Öffnet ein Http Request zu <https://godbolt.org/api/compiler/java1800/compile>  
mit die folgene data als JSON String

{

“source”: {

class Tester {

%\_INSERT\_CODE\_HERE\_%

void static public main(String[] args)   
 {  
 // code used for testing  
 }

}

class UserInputSourceCode {

}

},

“option”: …

}

Falls man will nur kompilieren Users Code, wird einfach der user code als JSON schicken, und setzen in die Optionen, das kein execution durchgeführt werden sollte.

Um den http Request zu senden wird die Klasse HttpURLConnection verwendet.

## Webpage

Die Webanwendung ist nach gewöhnliche Standard organisiert:   
Die Benutzer Oberfläche ist primär implementiert in **jsp** Files, die sorgen um Benutzer Spezifik Code zu generieren, falls Notwendig, während die Verschiedene Elemente Darstellungen sind in **css** Files geschpechert.

In der Folder **js** sind gespeichert java script Quell Code, in die **html** Verzeichnis man kann die Hinhalte von die Lektionen finden, während die Bilder sind in die Verzeichnis **images**.

Die Name von die Files entsprechen die Benutzung.

### Striktur eine Seite.

body

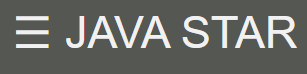
Top Navigation

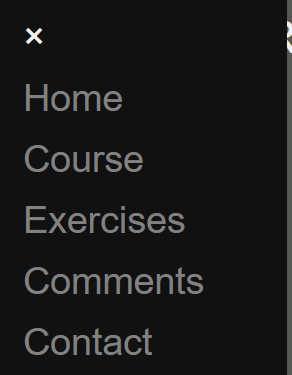
In Abbildung, man kann die Struktur eine webpage sehen:

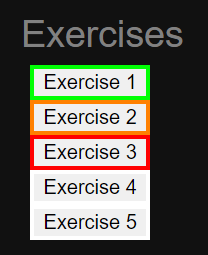
Diese ist gültig generell für die Mehrheit der Seiten.

Side Navigation

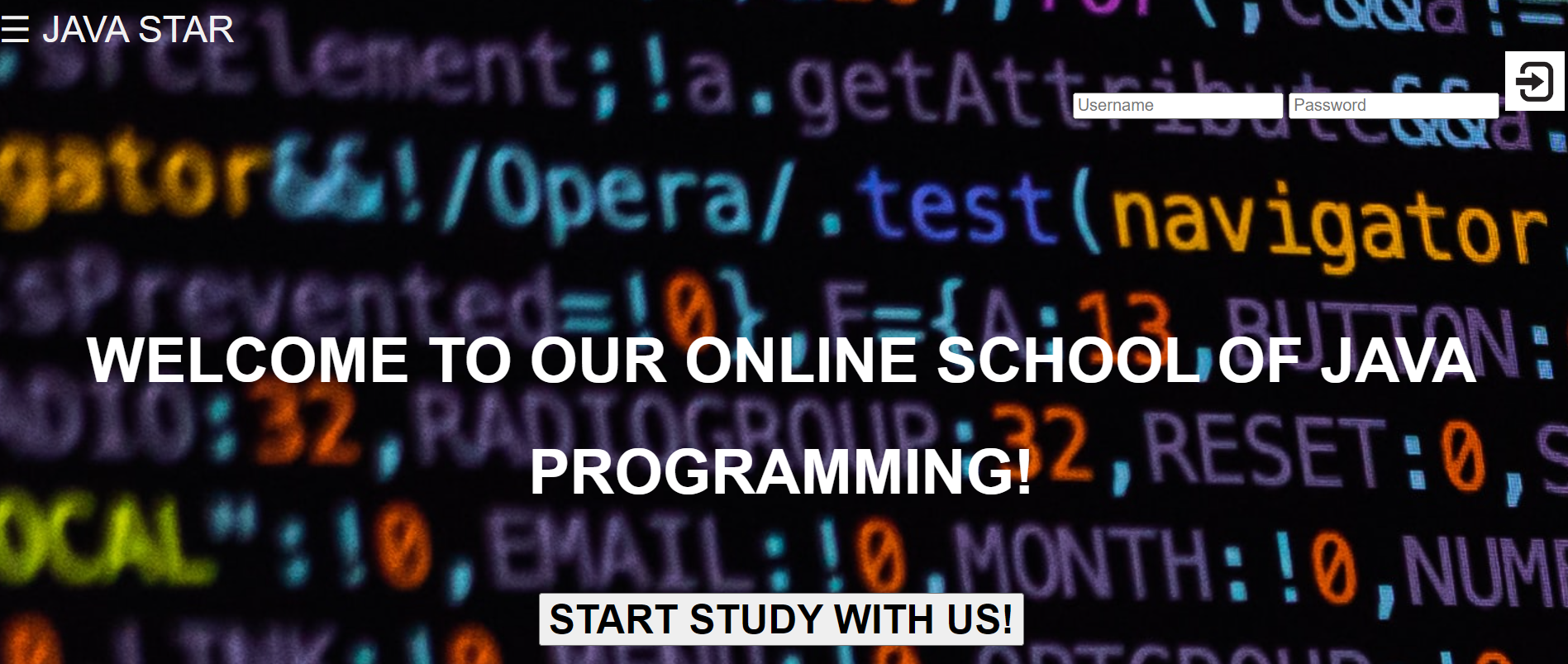
Die Klasse **GiuBean** sorgt für die Navigation bars

* Der ***Top Bar*** enthält eine Form, um einzuloggen, falls man ist ausgeloggt, sonst es gibt ein Welcome + username Message, mit ein Bild, die jetzt kann man nicht personalisieren.
* Die ***Side Navigation[[2]](#footnote-2)*** hat alle link zu die verschiedene Seite der Webanwendung.  
  Diese Navigazionen kann dynamisch verschwinden
  + geschlossen:  
     
  + geöffnet



* + Home page
  + Course, wo mann kann alle Lektionen toggle mit ein Klick
  + **Exercises**, hier auch mann kann toggle die verschiedene Übungen.
    - Falls man ist nicht eingelogt, dann kein Übung ist verfügbar.
    - Falls man ist engemeldet, dann die verschiedene Übungen sind gefärbigt nach der Erfolg:
      * Grün – korrekt gelöst
      * Rot – Fehler beim Test
      * Orange – Fehler bei Kompilierung
      * Weiß – Kein Versuch gemacht
  + **Comments** - kann zum Kommentare Seite springen
  + **Contact** hat Entwickler Kontakte und Weitere Online Resources, um ein Java Profi zu werden

### Home Page



Einfache Home Page mit login und regiestrierung forms.

### Lektionen

#### Struktur Lektion Seite

In die Lektionen, werden die Navigation teile wie beschrieben generiert und es wird dynamisch der Content von der body auf Basis der Request Parameter currentPage gebaut.

Die Lektionen sind zu finden unten  
 JavaStarWebsite/jsp/Lesson.jsp?currentPage=<name>  
Default Name von die Parameter *currentPage* ist *JavaBasic.html*

Der Klasse **GuiBean** enthält die sortierte Liste von alle Lektionen in die Variable

**public** **static** String[] *lessonList* = { "JavaBasic.html", … , };

Aus Basis der currentPage, mann kann die previousPage und nextPage bestimmen. falls gibt es eine

In **GuiBean** der Metode **getNavigationCourse** gibt die *html* Element für die navigationen zwischen Lektionen

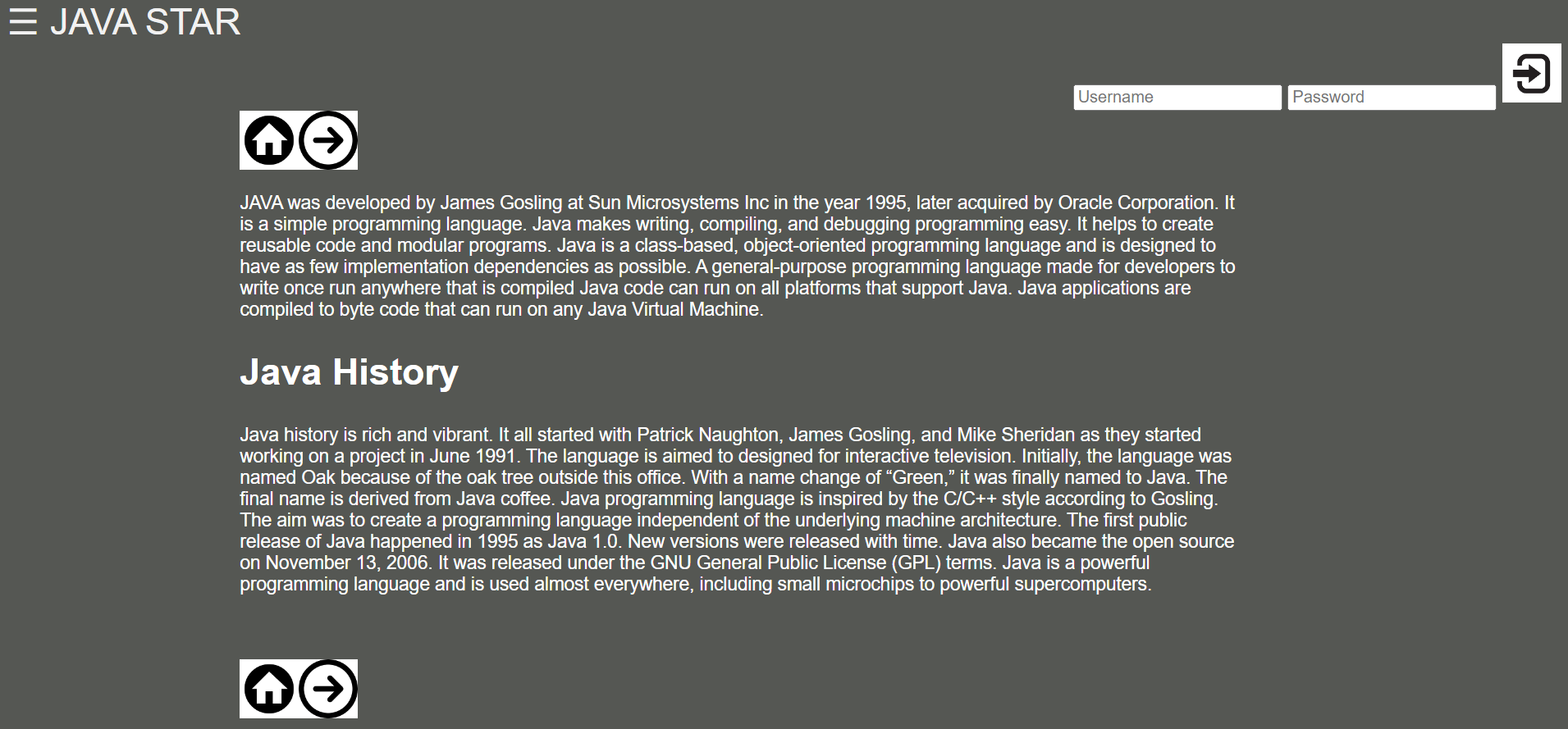
Die sind einfache <form> die die gewünscht Seite Rufen: **Home** (HomePageView.jsp) oder **Lession** mit name „title“ (Lesson.jsp?currentPage=title),

Die Inhalt des Seite ist mit der direktive

<jsp:include page=*"*<%=pagePathFull%>*"*></jsp:include>

aus der verzeichnis **html** geladet:

Top Navigation



Side Navigation



JAVA COURSE



#### Eine Neue Lektion Hinzufügen

Wie schon geschrieben, es würde besondere aufgepasst, dass neue Skripte (und Übungen) einfach hinzufügt werden können.

Es ist genug:

1. Eine neue Seite *<name>.html* hinzufügen in die Verzeichnis **html**
   * Der Seite enthält nur die body des html
   * Die entsprechen Darstellungsvorgaben findet mann in die file **css/Lesson.css**
2. Erwieter die Liste   
    **public** **static** String[] *lessonList* = { "JavaBasic.html", … , };

mit *<name>.html* in die gewünschte Positionen.

### Übungen

Die Übung Seite hat auch die Normale Navigationen und ist organiziert mit die folgende Workflow:  
ExerciseView.jsp -> ExerciseAppl.jsp -> ExerciseResultView.jsp :

ExerciseAppl.jsp

ExerciseView.jsp

ExerciseResultView.jsp

Es gibt die Möglichkeit die Lösung anzuschauen und ladet in die Input.

#### Eine Neue Übung Hinzufügen

Wie Für die Lektionen, auch neue Übungen können schmerzlos hinzugefügt werden:

1. Generiert die Files:
   * *exercise\_<N>\_instruction.txt* mit html Formattierte text von die aufgabe
   * *exercise\_<N>\_out.txt* mit erwartet stdout auf der test
   * *exercise\_<N>\_solution.txt* mit der Lösung von der Übung
   * *exercise\_<N>\_test.txt* mit eine Klasse die inhält den tag %\_INSERT\_CODE\_HERE\_%, wo mann sollte der Beutzer Input hinzufügen und der main, wo mann kann ein Klein Test für die Klasse schreiben – Sehen die Übungen in die Ordner **data/**
2. Hinzufüge die Aufgabe in die Tabelle **exercises**.

### Comments

Standard Seite mit Navigation, mit eine <form>, um Kommentar (max 512 charachter) zu schicken.

Hier sind Relevant die zwei Files ForumView.jsp und ForumAppl.jsp.

Wen der admin ist angemeldet, er kann sehen ein Special <input>  , um ein Kommentar zu löschen.

Zu diese Zweck dient die Metode:  
 **public** **void** deleteComment(**int** commentId)in der Klasse **class** CommentsDB.

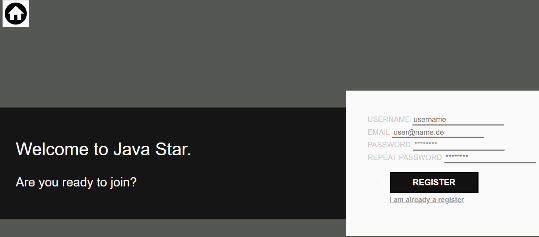
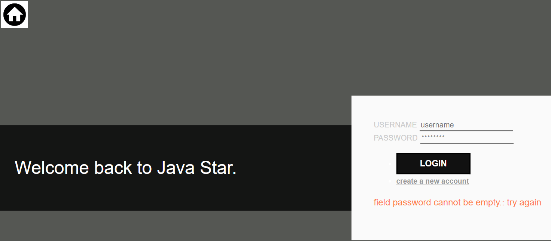
Die eigetlich überschreibt die alte Komment mit ein

### Contacts

Standard Seite mit Navigazionen – Relevant ist die File Contacts.jsp

### Login und Registrierung

Diese zwei Seiten teilen viele Komponente der Oberfläche, aber wurden separat gehalten.

1. **Registrierung**
   * username
   * Email
   * Password
   * Wiederholen password
2. **Login**:
   * username
   * password

Die inputs werden überprüft in die jeweilige Appl.jsp Files mit Standard check:

Falls ein Fehler auftritt, ein entsprechen Message wird visulaiziert.

# Abgrenzungserklärung

## Die Webanwendung:

1. HTML-Oberfläche: Dmytro Poliskyi
2. Anwendung/Plattform: Zhanna Belloni
3. Datenspeicherung: Zhanna Belloni, Dmytro Poliskyi

## Die technische Dokumentation:

1. Überblick über Webanwendung, Architektursystem: Dmytro Poliskyi
2. Bestandteile der Webanwendung: Zhanna Belloni

## Die Präsentation:

1. Einleitung, HTML-Oberfläche, technische Dokumentation: Dmytro Poliskyi
2. Webanwendung (JSP) und Datenspeicherung: Zhanna Belloni

1. **NOTE** 1: Der Zeit die Anwendung Speicher der Password in Klaar text, es solte aber besser nu der hash des Password.

   **NOTE** 2: Die Active spalte ist der Zeit nicht benutzt.  
   Die Idee war zu sorge für eine Form von Autentifizierung, wie z.B. ein code gesendet zu der gegebene email, aber ist ist leider nicht implementiert. [↑](#footnote-ref-1)
2. Die Navigazionen würden in Rahmen der Kurs <https://www.w3schools.com/> erstellt. [↑](#footnote-ref-2)