Praktikum Anwendungssysteme

JAVA STAR – Course to learn Java

# 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 Seite wo Java ist beigebracht
2. Die Übungen: wo man kann sich trainieren und prüfen seine Fähigkeiten
3. Eine Sektion, um Kommentare zu lassen

Teile von der Webanwendung sind erreichbar, nur rein angemeldete Benutzer, so ein login und ein Registrierung seite sind da.

# Setup und Starten der Anwendung

Die Anwendung benötig als Abhängigkeit

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

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

**Erste**, sollten man der Apach TomCat einsetzen, die postgres jdbc driver in die build path und, falls notwendig, kopieren die postgres jdbc driver in die **lib** Verzeichnis in die Installazion Ordner von die TomCat.  
Um den Anwendung zu benutzen, 4 Schritte sind notwendig:

1. Konfigurieren die Verbindung zum Datenbank: man muss den File e PostgreSQLAccess.java in package **de.hwg\_lu.java\_star.jdbc** anpassen.
2. Inizializieren die Applikationen:
   * Run als Java Applicazionen der File

**/JavaStarWebsite/src/main/java/de/hwg\_lu/java\_star/admin/AdminDB.java**  
mit command line argument die komplett Pfard von die verzeichnis **data/** im root von diese Projekt.

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

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

Enjoy!

# Architektur Systems

Das Architektursystem ist über ein Webserver eingerichtet, der Java Servlets und JavaServer Pages unterstützt: in unserem Falls es ist Apache TomCat.

Diese Webserver generiert Dynamik Webseiten mit Hilfe von Informationen gespeichert in ein Database: in unserem Fall: PostgreSQL.

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

Diese Ressource muss erreichbar sein.

WebBrowser

Web Server

Database

<https://godbolt.org/>

Request

Response

Servlet

Die Anwendung basiert sich auf Vorlesungen Skripte und Übungen: da ein in ein Kurse immer neue material kann hinzufügt werden, würde besondere aufgepasst, dass neue Skripte und Übungen können einfach hinzufügt werden.

# 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 es gibt ein Reihenfolge von Helper Klasse die man kann in die package **de.hwg\_lu.java\_star.utils** finden.

## 

## Administrative Sektion und Databank

Die Administrative Sektionen bereitstellt Klasse in die package **de.hwg\_lu.java\_star.admin**, um der System zu initialisieren oder reinigen :

1. Initialisierung der Datenbank mit Schema und alle benötige Tabellen (sehen Sektion [Datenbank](#_Databank)).
2. Anlegen ein Admin User, mit besondere Privilegien (user: admin, psswrd: admin)
3. laden in die Datenbank alle benötige Daten (sehen [Übungen](#_Übungen)).
4. Es gibt eine 3 Utility die als Java Program laufen lassen können:
   1. *AdminDB.java* <pfard zu data Ordner> Initialisieren der System
   2. *AdminViewer.java* die Möglichkeit, zu schauen und prüfen die Inhalt alle Tabelle
   3. *AdminDBCleaner.java* zu löschen alle Daten aus System, i.e. löschen alle Account und Statistiker meisten benutz wehrend Entwicklung des Applikation)

### Datenbank

Die Anwendung benötig einige Tabelle, um korrekt zu funktionieren:

Der Klasse **AdminDB** sorgt dafür:

Anlegt ein Schema mit dem gegebene Name (definiert in die Klasse PostgreSQLAccess in package **de.hwg\_lu.java\_star.jdbc**) und die Folgende Tabelle

**TABLE: account**

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

Wenn ein Benuzter sich regiestriert, alle notwendige daten warden hier gespeichert.

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

**TABLE: comments**

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

Ein angemeldete user kann comments hinzufügen: diese Tabelle spechert es mite in serial id, ein timestamp, der username und der kommentar selbt. Der Admin Kann ein Komment löschen, in dem der Komment text wird überschrieben mit ‘comment deleted by the admin’

**TABLE: exercise**

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

Alle Informationen für die übungen (übungausgabe, lösungen u.s.w werden in diese Tabelle gespeichert (sehen [Übungen](#_Übungen))

**TABLE: statistics**

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

Ein angemeldete Benutzer kann Übungen lösen: in diese Tabelle werden die Statistik für die Users Achievements gespeichert, i.e. welche Übungen hat der User probiert zu lösen und mit welche Erfolg.

### 

### Übungen

Die Übungen sind die komplexe Teil von der System:

Der Benutzer müss ein Java Program schreiben, die in die Aufgabe beschrieben ist:

der Anwendung akzeptiert eine freie Text und sorgt für Kompilierung und überprüft ob es korrekt ist.

Bei initielizierung alle benotige informazionen werden aus txt-Datei ausgelesen und in die Databank, in die Tabelle exercises, geladen:

jeder Tail ist gespeichert in ein eigene File, die hat die Strukture:

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

1. Text des Ausgabe: spezifikation=instructions -> in spalte exercise\_text
2. Expected output: spezifikation=out -> in spalte exercise\_out
3. Lösung des Aufgabe: spezifikation=solution -> in spalte exercise\_solution
4. Class zu testen von die Lösung: spezifikation: test -> in spalte exercise\_test

Neue Übungen können hinzugefügt, solang die entsprechen Strukturen haben und müssen in die Folder **data** gespeichert werden.

## Bean Klassen

In der package **de.hwg\_lu.java\_star.beans** es gibt die Folgende Klasse:

* **AccountBean**: speichert username, Password, email und die Flag admin und active benutzer: Es ist benutzt, um neue Benutzer in System zu spechern.
* **LoginBean**: Falls der User ist angemeldet speichert die Informationen von die angemeldet Benutzer und ob der Admin ist.
* **MessageBean**: speichert Error Messages, die der Anwendung sollten der Benutzer zeigen.
* **GiuBean**: Es ist ein Utility, um bestimmte teil der Webpages zu generieren, d.h. navigation bar und Lektion Pages.
* **ExerciseResultBean**: Speichert Alle informazionen zu der Versuch ein bestimmte Übung zu lösen, d.h. die Übung ID, die Input Source Code, die compilierung Fehler oder der test Fehler, fall gibt, und die Ergebnis des Test.

## Database Connection Utility

In der package **de.hwg\_lu.java\_star.jdbc** Es gibt alle klasse, die der Ferbindung zu der Database vereinfachen. Da verschiedene Tabelle vorhanden sind, es gibt eine Klasse die kümmert sich zu speichere, lesen oder update Werte in eine bestimmte Tabelle:

* Klasse, um ein Verbindung mit der Datenbank zu schaffen:
  + **JDBAccess.java:** Utility, um Connections zu erzeugen
  + **PostgreSQLAccess.java:** Connection username und password, Schema Name und End Point für die Driver postgres zu setzten.
  + **NoCollectionException.java:** Exception Klasse, falls es gibt Probleme mit den Verbindung zu Datenbank
* Klasse, die eine Spezifik Aufgabe haben:
  + **CommentDB.java:** Klasse zu bearbeiten Beuzter Komments: es sorgt für neue Kommentare hinzufügen, lesen Kommentare, oder markieren als deleted.
  + **ExericiseDB.java:** Klasse zu zurückholen Informazionen für die Bearbeitung ein Übung, d.h. Anzahl von Übungen, Übungausgabe, Übungaufgabe, test code und Lösung.
  + **StatisticDB.java:** Klasse die update die Statistik von ein Benutzer Erflog: Es speichert die Ergebnis ein Versuch von ein Übung und zurückholt die Ergebnissen als hex Farbe, um in ein html element zu benutzen.

## Externe Service Helper: Prüfung von die Übung

Der package **de.hwg\_lu.java\_star.excercises** enthält die funktionalität in die kalsse **Tester**, um die External service compile-explorer (<https://godbolt.org/>) zu benutzen.

Diese Klasse anbietet primäre 2 Funktionalität:

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

**compileExcercise** kompiliert die angegebene **String** sourceCode und return die Ergebnis der Request, **testExcercise** integriert der input des Benutzer in die TestCode aus die spalte exercise\_test in Tabelle exercises und ausgibt die Ergenis von der Test.

In die Abbildung man kann sehen wie die Request ist gebaut und gesendet.

* 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

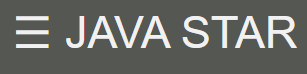
Top Navigation

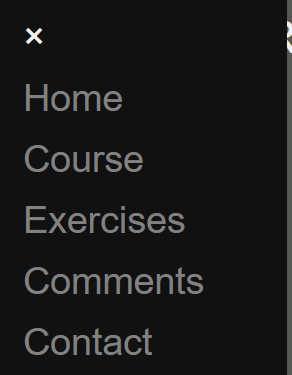
In Abbildung, man kann die Struktur eine webpage sehen:

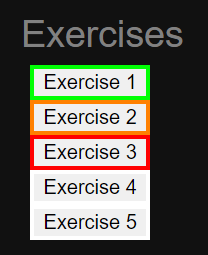
Diese ist gültig generell für die Mehrheit der Seiten.

Side Navigation

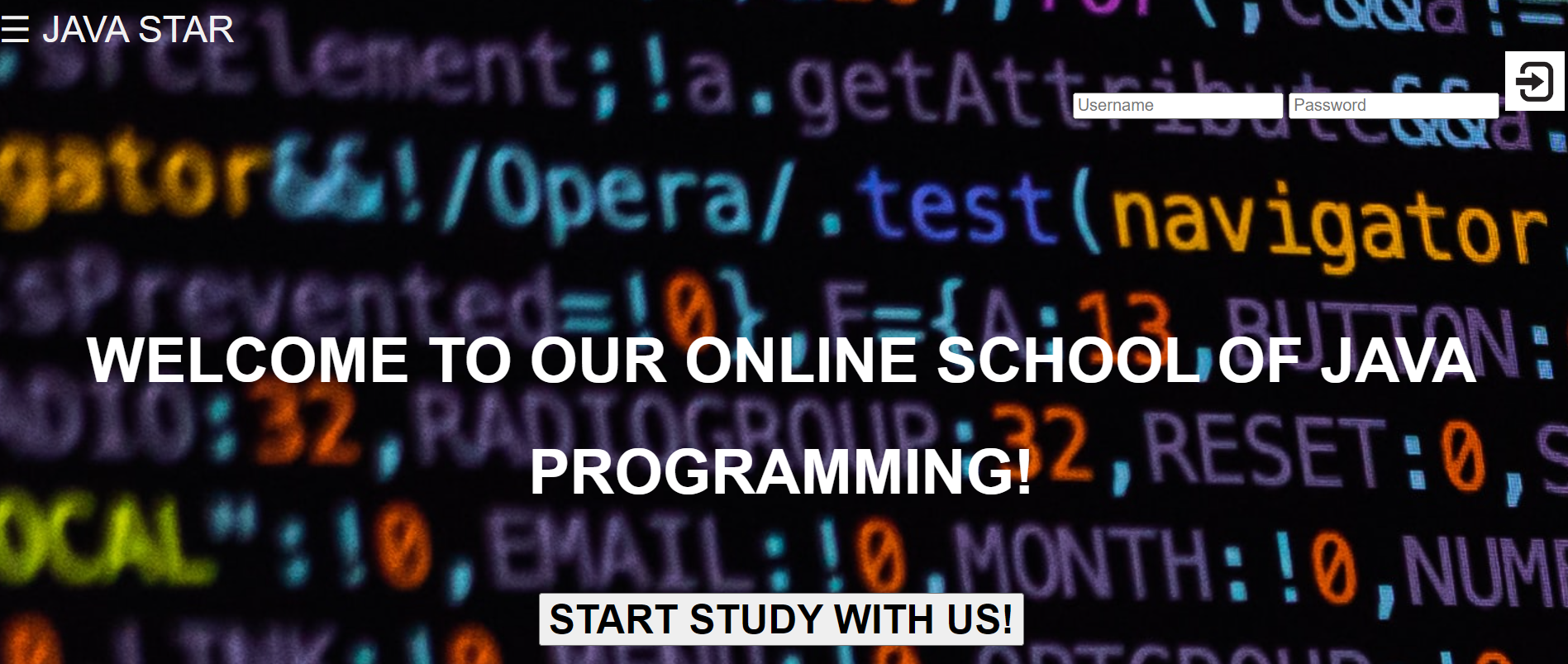
Die Klasse **GiuBean** sorgt für die Navigation bars

* 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

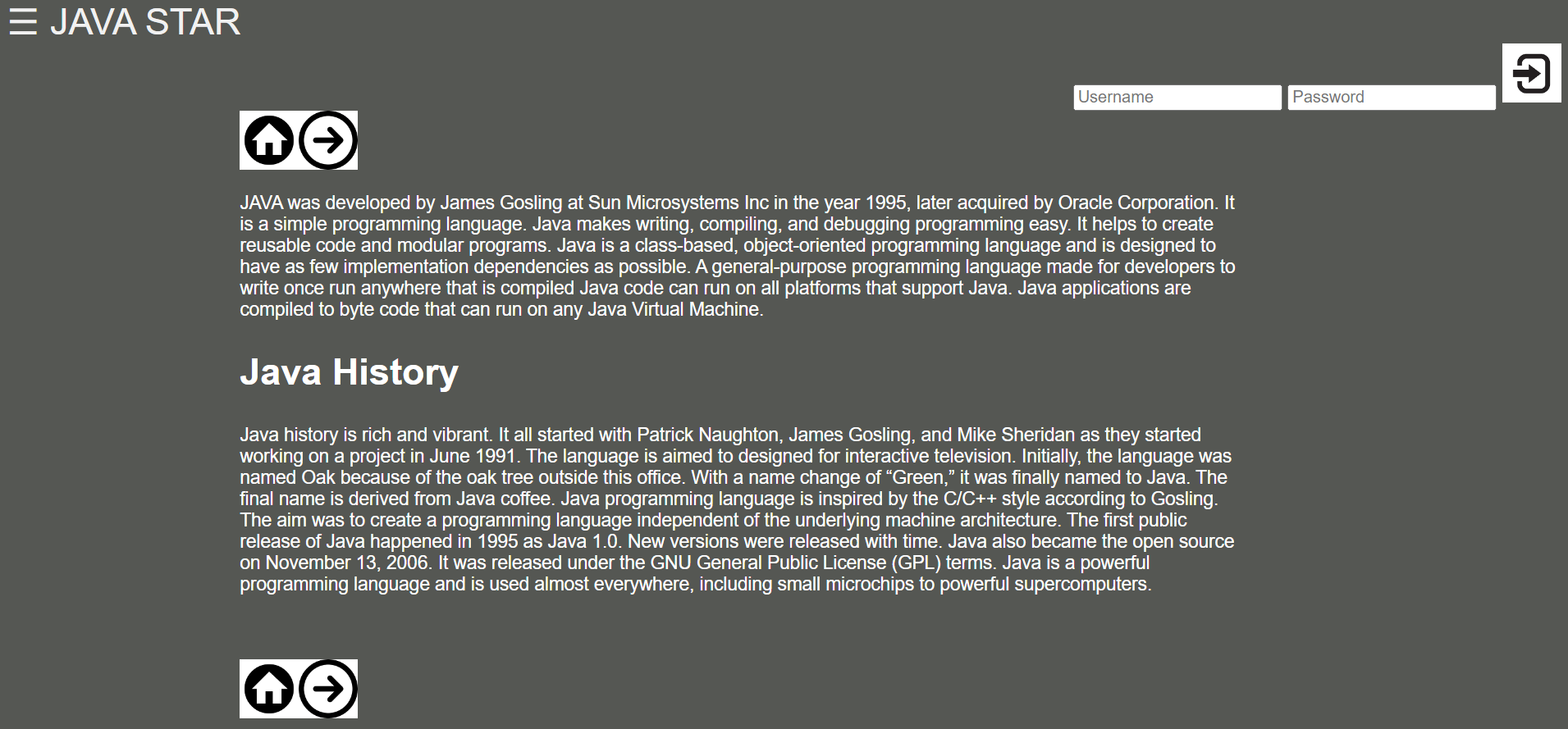
Die sind einfache <form> die die gewünscht Seite Rufen: **Home** (HomePageView.jsp) oder **Lession** mit name „title“ (Lesson.jsp?currentPage=title),

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 und schmerzlos 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

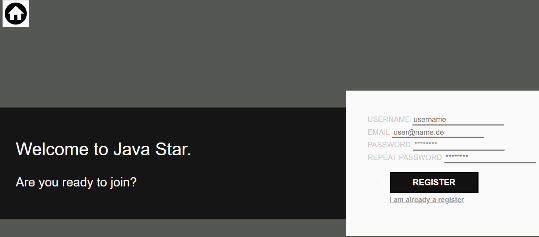
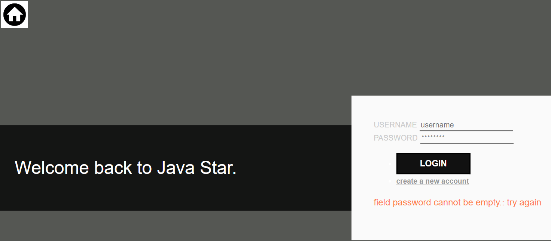
### **Contacts**

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

### **Login und Regiestrirung**

Diese Zwei Seite teilen viele Oberfläche komponente, aber würden separat gehalten, weil den Code ein bissen sauberer ist.

Sie bestehen auf ein form, mit die input die man eingeben muss:

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

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

Falls ein Fehler auftritt, ein entsprechen Message wird visulaiziert.

### **TODOs**

Die anwendung kann verbessert in viele stelle:

Wir listen hier die wichtigsten:

1. Mehr Lektionen und Übungen hinzufügen
2. Speicher nicht die Passwords als Klar text.
3. Implementieren ein Mechanismus zu validieren die email, s.h.

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)