**Катедра „ Информационни технологии ”**

**МАГИСТЪРСКА ТЕЗА**

***за придобиване на образователно-квалификационна степен „магистър“***

Тема: **Уеб-базиран каталог на дипломни работи и дисертации**

|  |  |
| --- | --- |
| **Разработил:** Ненад Костов | **Научен ръководител:** доц. д-р инж. Павлинка Радойска |
| Име на дипломанта | проф. / доц. д-р Име на ръководителя |
| Фак. № 010081 |  |
| Специалност „Информационни технологии ” |  |

**СОФИЯ**

**2020 г.**

**УТВЪРЖДАВАМ:  
ДЕКАН**

**/доц. д-р инж. Пламен Павлов/**

**ЗАДАНИЕ**

**ЗА ДИПЛОМНА РАБОТА**

**за придобиване на ОКС „магистър”**

**на студента: Ненад Костов , фак.№ 010081**

**специалност „Информационни технологии“**

**Тема:** Уеб-базиран каталог на дипломни работи и дисертации

**Цел на разработката:**

Да се създаде Уеб-базирано приложение за съхраняване на информация за дипломни работи и дисертации.

Приложението да дава следните възможности:

1. Съхраняване на пълната информация, описваща една дипломна работа или дисертация:
   1. автор, ръководител, рецензенти;
   2. дата на защита;
   3. резюме/автореферат, пълен текст, ключови думи;
   4. рецензии;
   5. оценка/диплома.
2. Да позволява търсене по различни критерии.
3. Да позволява регистриране на потребители и достъп на базата на роли.

**Изходни данни:**

Да се работи с релационна база от данни и езикът Java.

**Обхват на съдържанието:**

Анализ на съществуващи системи за съхраняване на научна информация.

Анализ и избор на технологии за решаване на проблема.

Описание на разработваната система.

Изводи и заключение.

**Дата на задаване:** 20.03.2020 **Срок на предаване:** 29.05.2020 г.

|  |  |
| --- | --- |
| **ДИПЛОМАНТ:** | **НАУЧЕН РЪКОВОДИТЕЛ:** |
| **/ Ненад Костов /** | **/ доц. д-р инж. Павлинка Радойска/** |
|  |  |
|  | **РЪКОВОДИТЕЛ КАТЕДРА:** |
|  | / **доц. д-р инж. Павлинка Радойска** / |

**СЪДЪРЖАНИЕ**

[Въведение 1](#_Toc42162429)

[Глава 1. Уеб приложения 2](#_Toc42162430)

[**1.1 Какво са уеб приложения 2**](#_Toc42162431)

[**1.2 Структура на уеб приложенията 2**](#_Toc42162432)

[1.2.1 Монолитни (еднослойни) приложения 3](#_Toc42162433)

[1.2.2 Двуслойни приложения 3](#_Toc42162434)

[1.2.3 Трислойни приложения 4](#_Toc42162435)

[**1.3 Принцип на работа на уеб приложенията 6**](#_Toc42162436)

[**1.4 Предимства и недостатъци на уеб приложенията 7**](#_Toc42162437)

[**1.5 Примери на вече разработени уеб приложения 7**](#_Toc42162438)

[Глава 2. Използвани технологги за създаване на уеб приложения 10](#_Toc42162439)

[**2.1 Бази данни 10**](#_Toc42162440)

[2.1.2 Релационни бази данни 10](#_Toc42162441)

[2.1.3 Системи за управлени на бази данни 11](#_Toc42162442)

[**2.2 Java 13**](#_Toc42162443)

[2.2.1 Java Virtual Machine 14](#_Toc42162444)

[2.2.2 Предимства и недостатъци на Java 14](#_Toc42162445)

[2.2.3 Интегрирани среди за програмиране на Java 15](#_Toc42162446)

[2.2.4 Клиент-сървър комуникация 16](#_Toc42162447)

[Глава 3. Проектиране на база данни и създаване на уеб приложение 17](#_Toc42162448)

[**3.1 Проектиране и създване на база данни 17**](#_Toc42162449)

[**3.2 Проектиране и създване на уеб приложение 22**](#_Toc42162450)

[**Заключение 65**](#_Toc42162451)

[Използвани източници 67](#_Toc42162452)

# Въведение

С напредването на технологиите използването на Интернет се увеличава и той се превръща в нов вид средство за обмен на информация между хората, което води до информационна революция в образованието, търговията и други области. Най-популярна услуга в Интернет е WWW (World Wide Web), която се състои от множество Интернет страници т.е. електронни документи, които съдържат информация за различни неща. Милиони предприятия използват Интернет като рентабилен комуникационен канал. Позволява им да обменят информация с целевата си група потребители и да извършват бързи и сигурни транзакции. За целта се използват електронни каталози за презентация на данни. Даните трябва да бъдат групирани по определени признаци и добре описани. Добрата организация осигурява както по-лесното управление, така и по-голяма гъвкавост на клиентите при работата им с каталога. Ефективната работа обаче е възможна само когато системата е в състояние да събере и съхранява всички необходими данни и да разполага със средства за обработка на тази информация и представяне на резултатите на потребителите. За тази цел се използват уеб приложения. Самите уеб приложения са произлезли от статични уеб страници. Интернет страниците в самото начало са били статични документи. Когато се изпрати заявка от браузъра към тях, те са обработвали въведените данни и изпращали нова страница. През време изискванията на потребителите са се увеличвали, както и нуждите за интерактивност. Така са се появили първите уеб приложения. Терминът „уеб приложение“ може да означава софтуерно приложение, написано на поддържан от уеб браузъра програмен език и което браузърите да успеят да рендерират. По също време животът без мобилни усторйства стана невъобразим, така че днеска рядко се среща човек, без да има поне едно мобилно устройство или таблет. Поради тези причини, уеб приложенията се адаптираха към устройствата с течение на времето, така че вече имаме достъп до почти всички уеб приложения на нашите телефони. Следователно достъп до електроните каталози на уеб приложенията имат всички лица, които притежават поне част от устройствата чрез които могат да се свързват с Интернет. В основата на уеб приложенията се намират добре организирани бази данни.

# Глава 1. Уеб приложения

## **Какво са уеб приложения**

Уеб приложенията са били създадени чрез развитието на уеб сайтове, които представляват набор от уеб страници, запазени на конкретен сървър. Уеб приложение е софтуерен продукт, който се изпълнява от уеб сървър и се показва на потребителя чрез уеб браузър. За да работи уеб приложението трябва да бъде написано на език за програмиране, който се поддържа от уеб сървъра, така че той да може да преведе и изпълни приложението. Десктоп приложенията трябва да бъдат инсталирани на клиентския компютър, за да могат да се използват. Това не е случай при уеб приложенията, тъй като те се намират на едно място, сървър или група сървъри. Това минимизира много проблеми, най-вече проблемите с разпространението и ъпгрейдите на приложенията. При уеб приложенията потребителят никога не получава самото приложение, а само интерфейса към него, което е основно всичко от което се нуждае. Уеб приложенията могат да се ползват по всяко време, навсякъде и на всякакви устройства (компютър, телефон или таблет), които имат връзка с Интернет. След като бъде разработена нова версия на уеб приложението, всички потребители могат да получат достъп до нея веднага, без нужда от обновяване. Ако приложението изисква повече ресурси, това засяга единствено хардуера на сървъра. Уеб проложенията предлагат по-голяма сигурност, тъй като нямат нужда от поддръжка на многобройни клиентски компютри, а единствено от мониторинг на уеб сървъра. Уеб базираният софтуер е по-лесен за използване, разработка, поддръжка и надграждане [1].

## **Структура на уеб приложенията**

Приложенията обикновено са разделени на логически части наречени „слоеве“, където всеки слой изпълнява определена роля. Предимството на използването на слоеве при създаването на приложение е възможността приложението да се изгражда част по част, както и по-лесно да се променят и използват повторно частите от кода. Спрямо броя на слоевете, на които е разделено приложението, различаваме:

- монолитни приложения,

- двуслойни (клиент-сървър) и

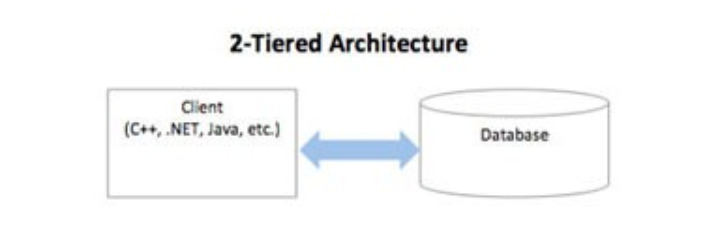
- трислойни приложения.

### Монолитни (еднослойни) приложения

Еднослойните приложения представят най-простия тип архитектура. При тях няма логическо (или физическо) разделяне на функционалността в отделни слоеве – цялото приложение е едно цяло. Тези приложения обикновено се използват само от един потребител и затова са сравнително прости за реализация. Като предимство на еднослойните приложения може да се посочи това, че всички техни компоненти се намират на едно място, което улеснява проектирането и изграждането на приложението. От друга стране именно това е причина при промяна във функционалността на системата да се налага пълна преинсталация, но това не е най-големият проблем на тази архитектура. Еднослойните приложения много трудно могат да се използват едновременно от много потребители и това силно ограничава приложението им в практика. Като пример за еднослойно приложение може да посочим просто настолно приложение, използващо вградена в него локална база от данни и предназначено за работа на един компютър. При него не се очква нито голямо натоварване, нито необходимост от обработка на големи обеми от данни, нито достъп от много потребители едновременно [2].

### Двуслойни приложения

Клиент-сървър (Фигура 1) е широко разпространена софтуерна архитектура. При този модел приложението логически се разделя на клиент и сървър, като клиентът отправя заявки към сървъра, а сървъра ги изпълнява и връща резултат. Този модел често се използва при по-прости разпределени и мрежови приложения. Клиентът и сървърът се разглеждат като два отделни слоя на приложение. В клиента се разполагат потребителския интерфейс и бизнес локата на системата. Този слой е тоговорен за представяне на информацията и взаимодействие с потребителя, както и за извършване на необходимата обработка според логиката на приложението. В сървърния слой, се съхраняват данните. Той отговаря за тяхното управление и съответно за операциите с базата от данни. Не е задължително двата слоя да са физически разделени, т.е. да са на различни машини, но често е такъв случай. Основно предимство на клиент-сървър архитектурата е разделянето на функционалността между двата слоя. Така се избягва прекоменото натоварване на сървъра при обслужване на множество заявки, тъй като основната част от обработката не се извършва от него, а от клиентите. Недостатък на тази архитектура е че поддръжката на приложенията се усложнява значително. При промяна в логиката на приложението се налага преинсталация на клиентската част, а при голям брой потребители това би означавало да се прави преинсталация на приложението на голям брой компютри, което може да доведе до значителни трудности и големи разходи. Примери за двуслойни приложения са повечето системи за управление на склад. При тях складовите наличности се съхраняват на сървъра в база от данни, а обсужващият персонал към склада осъществява достъп до данните чрез специален клиентски софтуер [2].



Фигура 1 – Двуслойна архитектура на приложение[3]

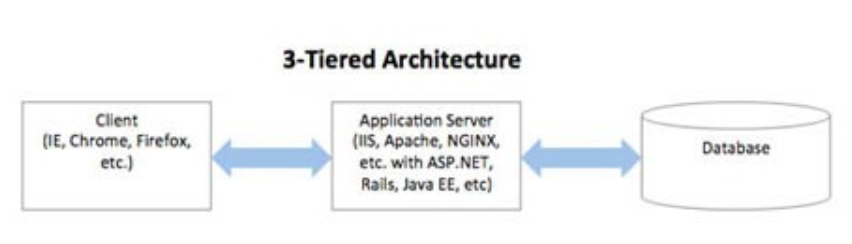
### Трислойни приложения

В едно приложение могат да се обособят три типа функционалност:

* представяне на информация на потребителя и взаимодействие с него,
* обработка, която реализира логиката на приложението и
* управление на данните.

При двуслойните приложения тези три типа функционалност се поставят в два слоя, от което проозтичат различни проблеми. Затова алтернатива на двуслойната архитектура е трислойната (Фигура 2), при която трите типа функционалност са разделени съответно в три логическа слоя:

* презентационен слой (front-end),
* бизнес слой (бизнес логика) и
* съхранение на данни (back-end).



Фигура 2 – Трислойна архитектура на приложение[4]

Презентационният слой е потребителски интерфейс на приложението. Той е отговорен за взаимодействие с потребителя. В него не се реализира обработка на данни (бизнес логика). Единствената логика, която се използва, е презентационната, т.е. подготовка на данни за вузуализация, валидация на данни, взаимодействие с потребителя и т.н. Презентационният слой не комуникира директно с базата данни. Вместо това той използва услугите на бизнес слоя.

В бизнес слоя се обособява бизнес логиката на системата, т.е. в него се реализират работните процеси и се извършва цялата обработка на данните на приложението. Бизнес слоя се грижи за правилната работа с много потребители едновременно. Той комуникира с базата данни за да съхранява и обработва данните в нея.

Слоя за съхранение на данни е отговорен за достъпа и съхранението на данните, като той извършва операциите с базата от данни. Най-често това е самият RDBMS сървър, например MS SQL Server. Слоя за данни не имплементира бизнес логика (работни процеси от системата), но може да имплементира логика, свързана с обработка на данните в самата база данни. Достъп до слоя за данни става единствено от бизнес слоя.

Основно предимство на този меодел е разделянето на трите типа функционалност. Това обособяване позволява промяна в някой от трите слоя да не окаже влияние върху останалите. Например, ако се промени бизнес логиката, изменение ще се наложи само във втория слой, като това не трябва да се отрази на работата с данните или представянето на информацията на потребителя. Друго предимство е по-голямата скалируемост на този модел в сравнение с двуслойния и по-малко натоварване на клиента, тъй като той само визуализира данните, а логиката е изнесена в бизнес слоя и не е при него. Изнасянето на логиката дава възможност за лесно отстраняване на проблеми и обновяване само на бизнес слоя без да се променя клиента. Клиентът може да е обикновен уеб браузър. Всеки от трите слоя може да се разположи физически върху отделна машина и така да се намали натоварването на всяка от машините, с което да се увеличи производителността при голям брой клиенти.

Обикновено функционалността на отделните слоеве (без потребителския интерфейс) се изпълнява от различни сървъри. Например бизнес логиката може да е на сървър за приложения (application server), а съхранението и достъпа до данните да се извършва от сървъра на базата от данни. Резултат от това е повишаването на риска за сигурността на системата, тъй като трябва да се подсигурят повече сървъри срещу евентуални атаки. Това води до по-трудна поддръжка на този тип приложения. Друг проблем на трислойната архитектура е намаляването на производителността заради нуждата от комуникация между слоевете. Проблем е също и сложността на системата – трябва да се осигурят надеждни механизми за комуникация между слоевете, трябва да се предвидят проблемите при изчзване на свързаността между тях, да се отдели повече време за проектиране и имплементация и т.н.

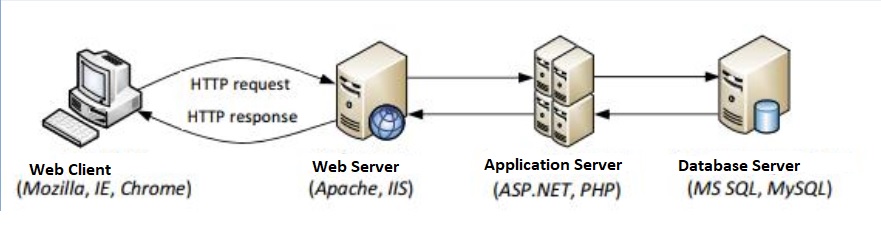
Добър пример за трислойно приложение е една банкова система. В нея данните са сметките и се съхраняват в база от данни (back-end). Логиката по управление на финансиите се разполага в бизнес слоя. Като презентационен слой (front-end) системата може да предоставя няколко приложения, например настолна система, от която работят служителите на банката, уеб система за Интернет банкиране и WAP система за банкиране от мобилни телефони [2].

## **Принцип на работа на уеб приложенията**

Основен принцип на действие на едно уеб приложение е следния:

* Потребителят изпраща заявка към уеб сървъра през Интернет, чрез уеб браузър или потребителски интерфейс на приложението.
* Уеб сървърът препраща тази заявка към съответния сървър на уеб приложението.
* Сървърът на уеб приложението изпълнява исканата задача, като изпрати заявка към база от данни, обработи данните и след това генерира резултати.
* Сървърът на уеб приложението изпраща резултати до уеб сървъра с исканата информация и обработени данни.
* Уеб сървърът отговаря на клиента с исканата информация, която след това се появява на дисплея на потребителя.

На Фигура 3 графично са представени тези стъпки.



Фигура 3 – Принцип на действие на уеб приложение[5]

## **Предимства и недостатъци на уеб приложенията**

Некои от предимства на уеб проложения са:

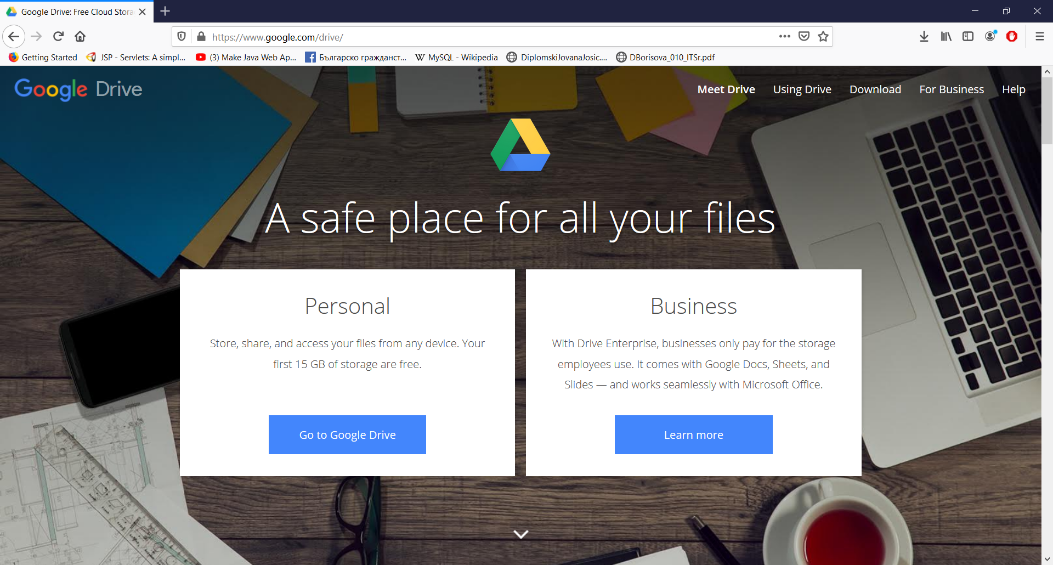
* Работят на множество платформи, независимо от операционната система или устройството, стига браузърът да е съвместим.
* Всички потребители имат достъп до една и съща версия, елиминирайки всички проблеми със съвместимостта.
* Не са инсталирани на твърдия диск, като по този начин се премахват ограниченията за пространството.
* Намаляват софтуерното пиратство чрез въвеждане на предплатени услуги.
* Намаляват разходите както за бизнеса, така и за крайния потребител, тъй като има по-малка поддръжка и по-ниски изисквания към компютъра на крайния потребител.

Уеб приложенията имат и недостатъци, некои от тях са :

* Уеб приложенията изискват съвместими уеб браузъри. Ако браузъра не имплементира некои функции или не подържа некоя версия на операционна система, потребителите ще имат проблем със зареждането на приложението.
* Приложенията обменят информация с отдалечени сървъри използвайки Интернет. Ако се прекъсне връзката, приложението става неизползваемо. Обаче ако се използва HTML5 API приложението може да се свали и използва и без връзка с Интернет.
* Уеб приложенията зависят от способност на сървъра да разпространява информация. Ако сървъра на компанията която е създала приложението прекъсне да работи, то и самото приложение ще бъде неизползваемо, докато при десктоп приложенията функционалността си остава една и съща и след това.
* Компаниите могат да преследват къкво потребителите работят, което може да доведе до проблем с личните им данни[6].

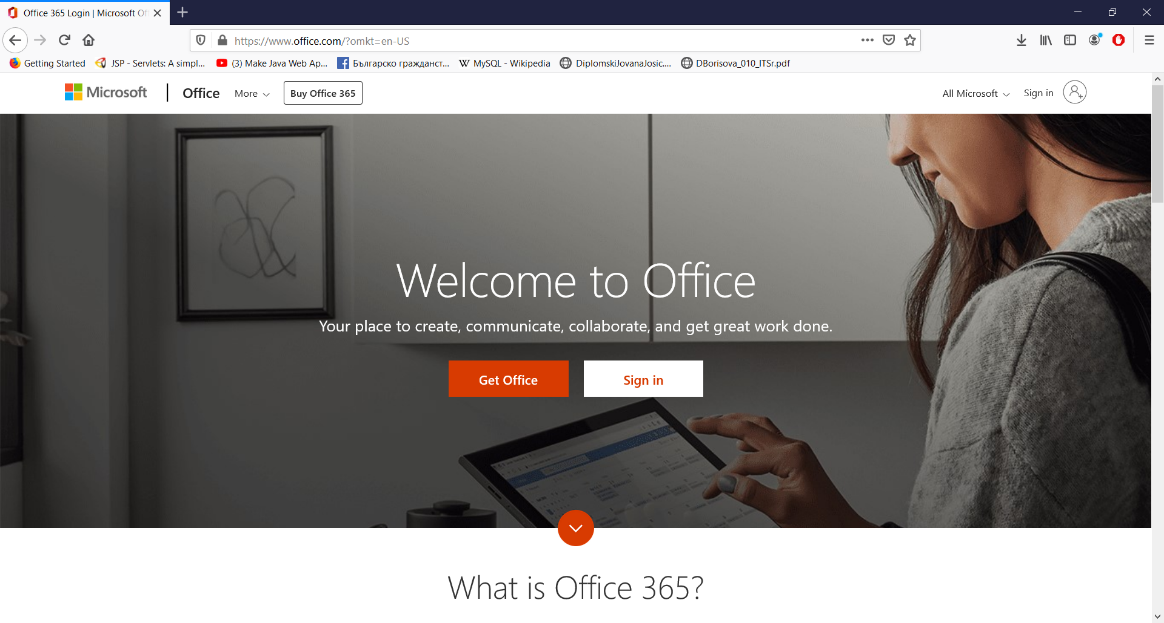
## **Примери на вече разработени уеб приложения**

Google Drive (Фигура 4) е безплатно уеб приложение от Google, което дава възможност за съхранение на различни файлове. Файловете могат да бъдат достъпени от всеки смартфон, таблет или компютър. Така потребителят, където и да се намира, има достъп до съхранената информация. Google Drive дава възможност да повече потребитела работят върху един и същ файл. Приложението се предлага в два варианта, за лична употреба и за бизнес употреба. Ако се използва за лична употреба, първоначално се предлагат 15GB безплатно пространство, но може да се закупи допълнителен капацитет за съхранение. При версията за бизнеса плаща се само хранилището, което служителите използват. Към узлугата са включени Google документи, таблици, презентации и формуляри.



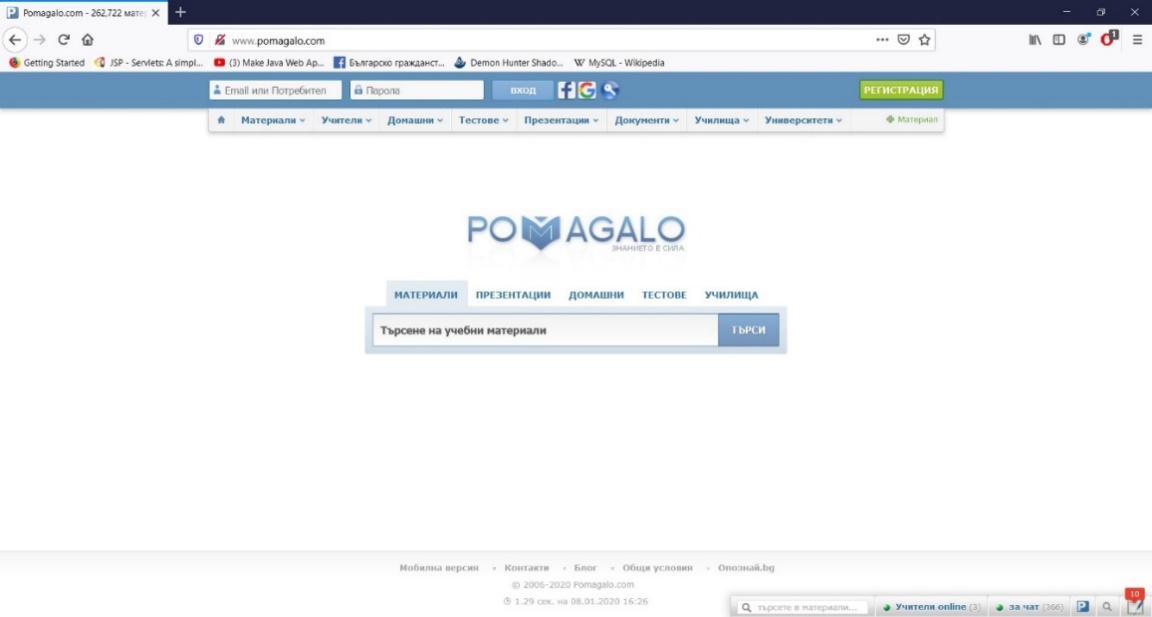
Фигура 4 – Google Drive [7]

Office 365 (Фигура 5) е уеб приложение създадено от Microsoft. Дава възможност, както и Google Drive, за съхранение на файлове, така и за създаване и редактиране на тях. На разположение са ви Outlook, OneDrive, Word, Excel, PowerPoint, OneNote, SharePoint, Microsoft Teams, Yammer и др. Microsoft дава възможност за предплащане, с която потребителите получават най-нови приложения на Office. Те могат да бъдат настолни и онлайн версии. Office приложенията могат да се достъпят от смартфон, таблет и компютър. Предлага се в неколко версии: Office for Home, Office for Bussines и Office for Education. Те се различват по приложенията която се предлагат на потребителите, функциите на приложенията и пространството за съхраняване в облака.



Фигура 5 – Office 365[8]

Pomagalo (Фигура 6) е приложение което съхранява огромен брой различни материали за курсови проекти, дипломни работи, бележки от лекции, тестове и други. Това уеб приложение дава и възможност за комуникация с други потребители. Некои от тях са преподаватели, което може да бъде доста полезно. Единствения недостатък на приложението е че потребителите имат ограничен достъп до материалите, и трябва да се предплатят за да могат да четат, свалят и качват информация.



Фигура 6 – Pomagalo[9]

# Глава 2. Използвани технологги за създаване на уеб приложения

## **Бази данни**

В основа на всяко уеб приложение се намира база данни. База данни е структурирано множество от данни, записано на носител, който е достъпен за четене и запис чрез компютър с цел да обслужи едновременно много потребители поотделно и за разумно време. Бази данни наричаме всичко от най-обикновен списък за пазаруване, през изображения в галерия до огромните количества информация в корпоративните мрежи. В зависимост от това как базата данни описва връзките, съществуват три модела:

* йерархичен - данните са представени под формата на дърво, състоящо се от корен и възли. Те са описани чрез връзки 1:N, насочени надолу. При този модел не се позволяват връзки от типа M:N,
* мрежов - данните са представени под формата на ориентиран граф. Връзките и при този модел не могат да бъдат от тип M:N. Ако такива съществуват, те се разбиват на две връзки от вида 1:N,
* релационен - в този модел данните се съхраняват в таблици, наречени още релации, между които има зададени връзки. Всяка таблица се състои от колони и редове.

При създаване на уеб приложение в тази дипломна работа най-подходящ за използване ще бъде релационен модел и поради тази причина той ще бъде по-детайлно разгледан.

### Релационни бази данни

Релационните бази данни са съставени от таблици. Всяка таблица съдържа редове с информация за всеки съхранен в нея обект (Фигура 7). Редът съдържа данни за един отделен обект и е съставен от колони с информация, която описва единствено този обект.

|  |  |  |
| --- | --- | --- |
| ЕГН | Име и фамилия | Место на раждане |
| 9508278888 | Иван Иванов | София |
| 8911256987 | Петар Петров | Пловдив |

Фигура 7 – Таблица в релационните бази данни

Всеки файл на базата данни може да съдържа множество таблици и всяка таблица може да съдържа множество колони. Една или множество таблици могат да бъдат достъпени и чрез изглед (view). Изгледът е логическо подмножество на таблица или комбинация от таблици. Той дефинира различен начин за достъпване, или разглеждане, на данните от таблицата, но самият той не съдържа данни. Позволява създаване на по-нататъшни логически връзки между таблиците, без да е нужно създаване на второ копие на данните. Идеята за релационен модел на базите данни появила се е през 1970г. През последвалите години, до ден днешен, релационните базите данни представляват основен начин за съхранение на данни [10].

### Системи за управлени на бази данни

За да се добави, получи достъп и работи с данни, които са съхранени в базите данни на компютъра, има нужда от система за управление на базите данни. Приложният софтуер, който осигурява възможност за работа с бази от данни, се нарича система за управление на бази от данни (СУБД). СУБД включва три основни компонента:

* средства за разработване на приложения,
* потребителски интерфейс и
* ядро, което извършва операциите по търсене, сортиране и актуализиране на данните в базата от данни.

Основни функции с които се характеризират СУБД са [11]:

* Описание на данните – СУБД трябва да предоставя на потребителя средство за описание на множеството на данните. Има няколко нива за описание: концептуално, логическо и физическо.
* Използваемост – тази функция предлага на потребителя взаимодействие с базата данни. Съществуват два основни типа потребители: специалисти, които използват и създават алгоритмични процедури и непрофесионални потребители, които използват предварително създадени функции.
* Интегритет – колкото повече данни има в една база данни, толкова е по-голям рискът тези данни да съдържат грешки. За да се намали този риск СУБД трябва да предлага на потребителите възможност да дефинират ограничения за интегритет (правила, позволяващи да се поддържа интегритета). Тези ограничения дефинират контрола, който СУБД може да упражнява върху данните.
* Секретност – Когато една база данни има много потребители, известни подмножества от данните трябва да са достъпни само за упълномощени лица. СУБД трябва да предлага средства за проверка на правата на достъп на потребителите.
* Конкурентеност (едновременен достъп) – Когато потребителите ползват едновременно една и съща информация от базата данни, СУБД трябва да разрешава конфликтите при достъпа и да ги обработва правилно.
* Сигурност на работа – При произшествие с хардуера или софтуера, базата данни не е в работещо състояние. С цел да може да се пусне на ново след като повредата е отстранена, СУБД трябва да поддържа запис на контролни точки, за да не бъде изгубен голям обем от данни.

Тъй като компютрите вече могат да работят с огромни количества от данни, системите за управление на базите данни играят ключова роля, било то като отделни приложения или като част от такива.

Microsoft SQL Server е система за управление на релационни бази данни, разработена от Microsoft. Като сървър за база данни, това е софтуерен продукт с основна функция за съхраняване и извличане на данни, чрез получаване на заявки от други софтуерни приложения, които може да са инсталирани локално или на друг компютър в мрежа. Microsoft е разбраотил много версии на тази система за различни групи потребители, със съответни функции. Една от най-новите версии е Azure. Той е базирана на облак система. Отличава се със рентабилност и изключителна сигурност. Много големи компании като Coca Cola, FedEx, Walmart, Samsung и други използват тази система за организация на техните бази данни [12].

MySQL е СУБД, която използва SQL език за въвеждане, достъп и извършване на други операции с бази данни. MySQL е известен основно заради скоростта си, сигурността и гъвкавостта си. Системата MySQL осигурява приложен програмен интерфейс (API) за C, C++, Java, Perl, PHP и Python. MySQL се разпространява като свободен софтуер (MySQL Community Server) под GNU General Public License (GPL), но също така и като комерсиална версия (MySQL Enterprise Server). MySQL Enterprise Server се различава от бесплатната версия с поредица от патентовани разширения, които се инсталират като сървърни добавки. MySQL Community Server позволява на потребителите да го използват за създаване на високоефективни и мащабируеми приложения. Написан е на C и C++ и работи на много различни операционни системи включващи различни версии на Windows, както и различни дистрибуции на Linux и MacOS. Компонент е от LAMP (Linux, Apache, MySQL, Perl/PHP/Python) платформа, която е съвкупност от софтуерни продукти предназначени за предоставяне на всичко необходимо за създаване на уеб приложения. Използва се от много уеб базирани приложения, включително Drupal, Facebook, Flickr, MediaWiki, Twitter, YouTube и WordPress [13].

## **Java**

Java е обектно-ориентиран език за програмиране. Първата версия на Java е била разработена от Sun Microsystems и пусната в употреба през 1995г., като част от Java платформата. В последствие се появяват множество други реализации включително от GNU, Microsoft, IBM, Oracle и други технологични доставчици [14]. Java платформата съдържа в себе си Java виртуалната машина и множество помощни инструменти и библиотеки. Съществуват няколко разновидности на Java платформата съобразно целевата потребителска група, като възможностите, които те предлагат на разработчиците, се застъпват.

Java Standard Edition е стандартна версия на Java платформата и е предназначена за разработка и използване на настолни приложения. Тази версия включва в себе си [15]:

* Java компилатор – стандартна версия,
* Java Virtual machine – стандартна версия,
* Графичен потребителски интерфейс,
* TCP/IP мрежов интерфейс,
* Работа с XML,
* Работа с файлова система,
* Интерфейс за работа с бази данни (JDBC),
* 2D и 3D графични библиотеки,
* Библиотека за сигурност и други.

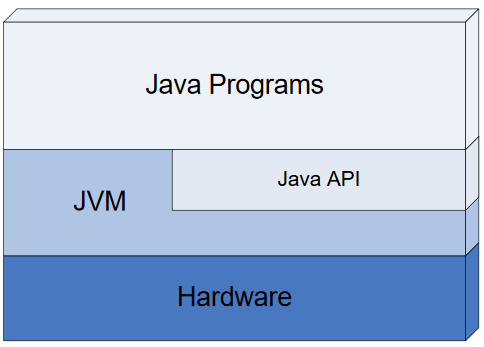
Java Enterprise Edition е предназначена за големи приложения с корпоративен характер, най-често разработени да бъдат използвани от хиляди клиенти. Java EE съдържжа всичко от стандартната версия на Java и я обогатява с още библиотеки и технологии. Главните характеристики на тази версия са:

* HTTP сървър и Servlet контейнет за разработка на уеб приложения,
* EJB контейнер за разработка на преизползваеми компоненти за отдалечено извикване,
* Предаване на съобщения (JMS),
* Повишена сигурност и
* Уеб услуги [10].

Java Micro Edition е ограничена версия на стандартната с по-малко възможности, предназначена за използване в малки и ограничени откъм ресурси хардуерни устройства. Най-често това са мобилни телефони, персонални помощници и дори домакински уреди. За да постигне задоволителна производителност при тези скромни възможности на устройствата, Java ME опростява и ограничава възможности на Java SE понякога до съвсем малък набор от базови операции. Това се отразява както на компилатора, така и на JVM [15].

### 2.2.1 Java Virtual Machine

В самия център на Java намира се нейния основен елемент – Java Virtual Machine, който осигурява независимостта от хардуера и операционна система.



Фигура 8 – Структура на Java [15]

Java Virtual Machine е абстрактна изчислителна машина. По аналогия на реалните електронноизчислителни машини съдържа набор от инструкции и манипулира области от паметта по време на изпълнение на програмите. Това не означава, че виртуалната машина по някакъв начин има зависимост от технологията за реализация, хардуера или операционна система. Виртуалната машина е реализирана програмно като интерпретатор на bytecode, но може да бъде реализирана да компилира до инструкции на истинския процесор и дори да се изгради хардуерно базирана реализация – силициев микропроцесор, способен да изпълнява bytecode. Виртуалната машина не знае за програмния езика Java, а само изпълнява инструкциите на bytecode, записани като class файлове [15].

### 2.2.2 Предимства и недостатъци на Java

Основно предимство и причина, поради която езикът и платформата Java са се наложили, е възможността кодът веднъж компилиран да работи на произволни, поддържани от платформата, операционна система или хардуерно устройство. Програмите могат да бъдат компилирани на Windows и след това да бъдат изпълнявани както на Windows така и на Linux, MacOS и всяка друга операционна система, поддържана от Java платформата [16]. Програмите могат да бъдат изпълявани дори и на мобилните телефони, поддържащи Java. Идея за независимост от среда е заложена при самото създаване на Java платформата и е реализирана с малка хитрина. Изходният код не се компилира до инструкции, предназначени за даден микропроцесор, и не използва специфични възможности на дадена операционна система, а се компилира до междинен код - така нареченият bytecode. Този bytecode не се пуска за директно изпълнение от микропроцесора, а се изпълнява от негов аналог – виртуален процесор, наречен Java Virtual Machine (JVM). Едно от най-големите предимства на Java е предлаганото от нея автоматично управление на паметта. То предпазва програмистите от сложната задача сами да заделят памет за обектите и да следят подходящия момент за нейното освобождаване. Това рязко засилва производителността на програмистите и увеличава качеството на програмите, писани на Java. За управлението на паметта се грижи специален компонент от виртуалната машина, наречен "събирач на боклука" или "система за почистване на паметта" (Garbage Collector). Основните задачи на събирача на боклука са да следи кога заделената памет за променливи и обекти вече не се използва, да освобождава тази памет и да я прави достъпна за последващи заделяния [15]. Главен недостатък е необходимостта от допълнителни ресурси (под формата на процесорно време и памет) за изпълнението на самата виртуална машина. Съществуват и компилатори, които превеждат bytecode до машинен код. Недостатъка при използването им е, че програмата тогава не може да се стартира на различни платформи [14].

### 2.2.3 Интегрирани среди за програмиране на Java

Писането с текстов редактор, компилирането и изпълнението на една малка програма от командния ред е сравнително проста работа, но за големи проекти това може да се окаже сложно и турдоемко занимание. За намаляване на сложността, улесняване на писането, компилирането, изпълнението и интегрирането им в един единствен инструмент, съществуват визуални приложения, наречени интегрирани среди за разработка (Integrated Development Environment - IDE). Средите за разработка най-често предлагат множество допълнения към основните функции, като дебъгване, пускане на unit тестове, проверка на често срещани грешки, контрол на версиите и т.н. Среди за разработка на Java приложения са: Eclipse, Netbeans и IntelliJ IDEA. IntelliJ IDEA може да се изтегли от официалния сайт https://www.jetbrains.com/idea/. Предлага се в две варианта, бесплатна – Community и платена Enterprise версия. IntelliJ IDEA анализира написания код, търсейки връзки между символи във всички файлове на проекта за зададен език. Използвайки тази информация, тя осигурява помощ при кодиране, бърза навигация и анализ на грешки [17]. С комбинация на различни клавижи от клавиятурата могат да бъдат извикани различни инструменти с цел улесняване на писане на програми. За целта на дипломната работа ще бъде използвана IntelliJ IDEA Enterprise Edition, понеже дава възможност за създаване на уеб приложения.

### 2.2.4 Клиент-сървър комуникация

Един от най-разпространените модели за комуникация в Интернет е моделът клиент-сървър. Този модел описва отношенията между две компютърни програми. Една от тях се нарича клиент, който заявява дадена услуга, информация или ресурс от друга програма, която се нарича сървър - той от своя страна обработва и изпълнява заявката, като предоставя заявения ресурс [18]. Apache Tomcat е уеб сървър за Java уеб приложения. Той е безплатен, с отворен код и имплементира технологиите JSP и Servlets. JSP са Java сървърни страници, които вижда потребителя, докато сървлетите са технологии използвани при създаване на уеб приложения. Във сървлетите са дефинирани методи за работа със сървъра на уеб приложението [19]. Apache Tomcat представлява контейнер за Java уеб приложения, който осигурява среда, в която да се изпълняват те. Спецификациите на Java Servlet и JSP (Java Server Pages) на Sun Microsystems са внедрени от Tomcat. Apache Tomcat може да бъде конфигуриран с помощта на XML конфигурационни файлове [20].

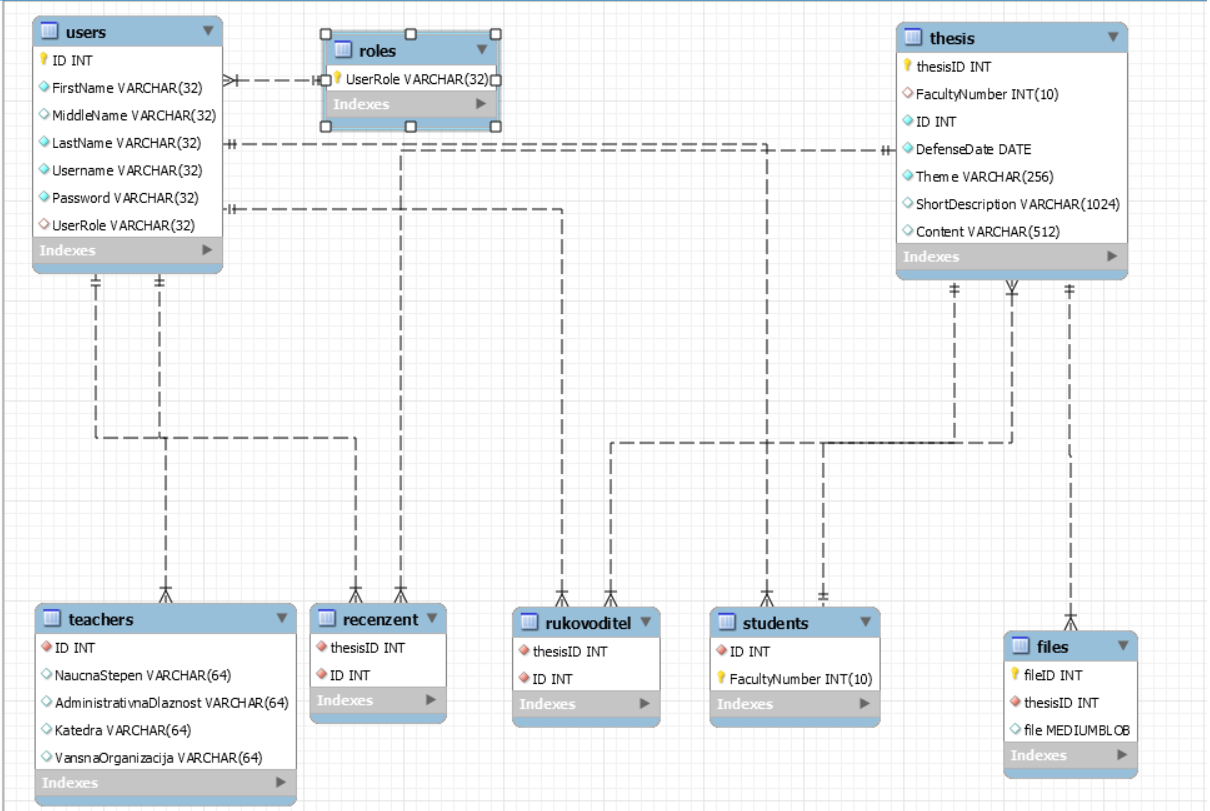
# Глава 3. Проектиране на база данни и създаване на уеб приложение

## **3.1 Проектиране и създване на база данни**

Първа стъпка преди създаването на база данни е планирането и проектирането й. Данните в релационните бази се съхраняват в таблици. Базата данни най-често съдържа една или повече таблици. Таблиците в релационните бази данни свързан са помежду си с връзки от типа 1:N (един към много). Всяка таблица се идентифицира с име и представлява съвкупност от свързани записи на данни. Таблиците се състоят от колони и редове. Колоните в таблиците описват обектите в тях. Всяка колона се характеризира с име и тип данни (datatype), които могат да се съхраняват в нея. При създаване на нашите таблици ще бъдат използвани следните типове данни:

* varchar - символен низ,
* integer - цели числа,
* date - дата,
* blob (Binary Large Objects) - различни текстови и мултимедийни файлове.

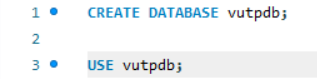
На Фигура 9 графично е представена структура на базата данни, която ще бъде създадена. Тя се състои от осем таблици, които са свързани помежду си и изпълняват различни роли.



Фигура 9 – Структура на базата данни

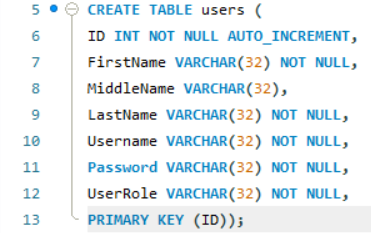
Нашето приложение ще трябва да предостави възможност за регистиране на потребители. Тези потребители ще бъдат съхранени в таблица „users“, която ще се състои от седем колони. В тези седем колони ще бъдат записани ID на потребителя, име, бащино име, фамилия, потребителско име, парола и ролята му. Ролите, които регистираните потребителите ще могат да имат ще бъдат записани в таблица „roles“. Регистрираните потребители ще бъдат преподаватели и студенти. За да бъдат правилно разпределени ще бъдат създадени две отделни таблици „teachers“ и „students“. Всеки преподавател ще се описва чрез научна степен, административна длъжност, катедра и външна организация, ако принадлежи на такава. Понеже приложението ще бъде ориентирано към дипломни работи, преподавателите ще бъдат разделени от своя страна на ръководители и рецензенти за съответна дипломна работа. Ръководителите ще бъдат добавяни в таблица „rukovoditel“, където ще бъде записан ID на дипломна работа за която те са задължени. Рецензентите по същ начин ще бъдат добавени в таблица „recenzent“ и към тяхното ID, ще бъде добавен ID на дипломни работи, където те са разпределени като рецензенти. Студентите ще бъдат записани в таблица „students“. Към техните ID-та ще бъде добавен факултетен номер, който ще ги прави уникални обекти в таблицата. Информация за дипломните работи, като ID на дипломната работа, факултетен номер на студента, ID на ръководитела, дата на защита, тема, кратко описание на дипломната и съдържание ще бъдат съхранени в таблица „thesis“. Самите дипломни работи, като текстови файлове, ще бъдат съхранени в таблица „files“. В тази таблица всеки файл ще има кореспондиращ ID на дипломна работа, към която се отнася.

Следваща стъпка, след планирането и проектирането е създаването на самата база данни, със всички нейни таблици. За създаване на база данни ще се използва SQL заявка „CREATE DATABASE“ в комбинация с името на базата данни. За да се работи върху създадената база данни използва се заявка „USE“ (Фигура 10).



Фигура 10 – Заявки за създаване на база данни и изпора й

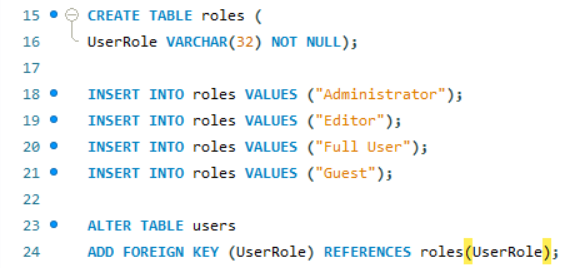
След избор на базата данни, следваща стъпка ще бъде създаване на всичките необходими таблици. Първо ще бъде създадена таблица „users“ със всичките си колони (Фигура 11). Много е важно всеки обект в таблицата да бъде еднозначно идентифициран. Затова се въвежда понятието първичен ключ (PRIMARY KEY). Първичен ключ е уникално свойство на всеки обект в базата данни. За първичен ключ в една таблица избира се такава колона, стойностите на която за всеки обект ще бъдат уникални. В таблицата „users“ това ще бъде колоната „ID“.



Фигура 11 – Таблица „users“

В релационните бази данни ключовете са в основата на осъществяване на връзките между отделните таблици. Външен ключ (FOREIGN KEY) е колона в таблица, която сочи към колона с първичен ключ в друга таблица. Таблица садържаща външен ключ се нарича „дете“, а таблица към която ключат сочи се нарича „родител“. Може да се приеме че външния ключ е референция на една към друга таблица.

Както беше казано в колоната „UserRole“ на таблицата „users“ ще бъдат записани роли на потребителите. Тези роли ще бъдат предефинирани в таблица „roles“. Таблицата „roles“ ще се състои само от една колона със също име като и колоната от таблицата „users“. С помощ на тази колона ще бъде създаден външен ключ, който ще направи връзка между двете таблици. На Фигура 12 представено е създаване на таблица „roles“, записване на стойности в нея и създаване на външен ключ.



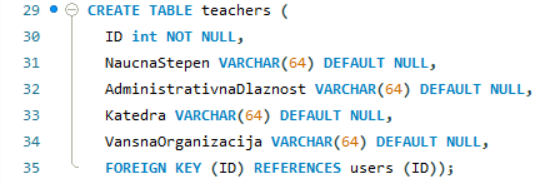
Фигура 12 – Създаване и попълване на таблица „roles“. Създаване на връзка между таблиците „users“ и „roles“

Заявка, която беше използва за записване на нови данни в таблица е „INSERT INTO“. В таблицата „roles“ бяха добавени четири стойности, които ще представляват роли на потребители. Зависимо от ролите, потребителите ще имат достъп до различни функции на приложението. При регистрация на всеки нов потребител, ролята му ще бъде „Guest“ и това ще бъде изпълнено с помощ на заявка „DEFAULT“ (Фигура 13). Външния ключ се задава с помощ на заявка „ADD FOREIGN KEY“. При случаи че таблицата, към която се добавя външния ключ вече е създадена, използва се заявка „ALTER TABLE“. „ALTER TABLE“ заявката се използва за модифициране на вече създадени таблици.



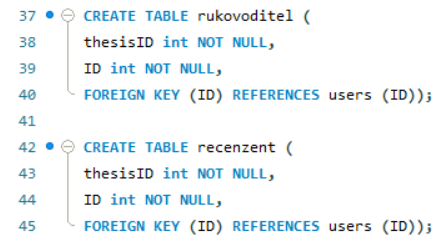
Фигура 13 – Задаване на стойност по подразбиране

Следваща стъпка ще бъде създаване на таблица „teaches“, която ще има връзка с таблицата „users“, посредством външен ключ върху колона „ID“ (Фигура 14).



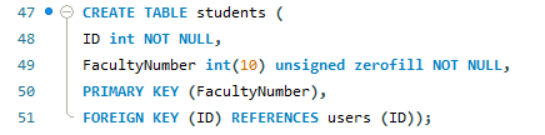
Фигура 14 – Таблица „teachers“

Както беше казано, преподавателите ще бъдат разделени в две таблици „rukovoditel“ и „recenzent“. Тези две таблици ще имат по два външни ключа, един сочещ към таблицата „users“, а другия ще бъде добавен след като бъде създадена таблицата „thesis“. На фигура 15 представени са заявки за създаване на двете таблици.



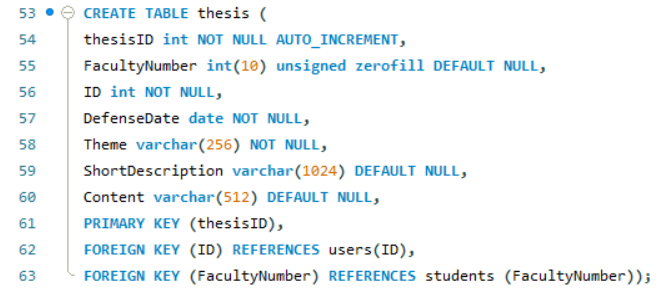
Фигура 15 – Таблици „rukovoditel“ и „recenzent“

Таблица „students“ (Фигура 16), ще съдържа един първичен и един външни ключа. Първичния ключ ще бъде сложен върху колоната „FacultyNumber“, в която ще бъдат записани факултетните номера на студентите. Външния ключ ще бъде сложен върху колона „ID“ за създаване на връзка с таблицата „users“.



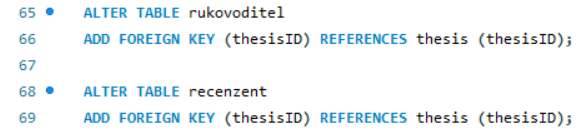
Фигура 16 – Таблица „students“

Таблица „thesis“ (Фигура 17), в която ще бъде записана информация за дипломни работи ще съдържа първичен ключ върху колоната „thesisID“. Така всички дипломни работи в тази таблица ще бъдат уникални. Освен първичен, таблицата ще съдържа и два външни ключа. Единият външен ключ ще бъде върху колоната „ID“, която ще представлява ID на ръководитела за съответната дипломна работа. Втория външен ключ ще бъде върху колоната „FacultyNumber“, която ще представлява факултетен номер на студента защитаващ съответната дипломна работа.



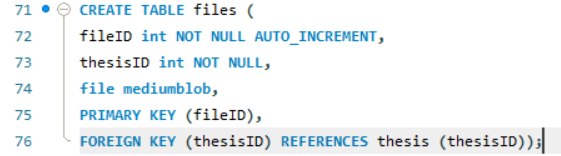
Фигура 17 – Таблица „thesis“

След като таблицата „thesis“ беше създадена, вече могат да бъдат добавени външните ключове за таблиците „rukovoditel“ и „recenzent“ (Фигура 18).



Фигура 18 – Добавяне на външни ключове на таблици „rukovoditel“ и „recenzent“

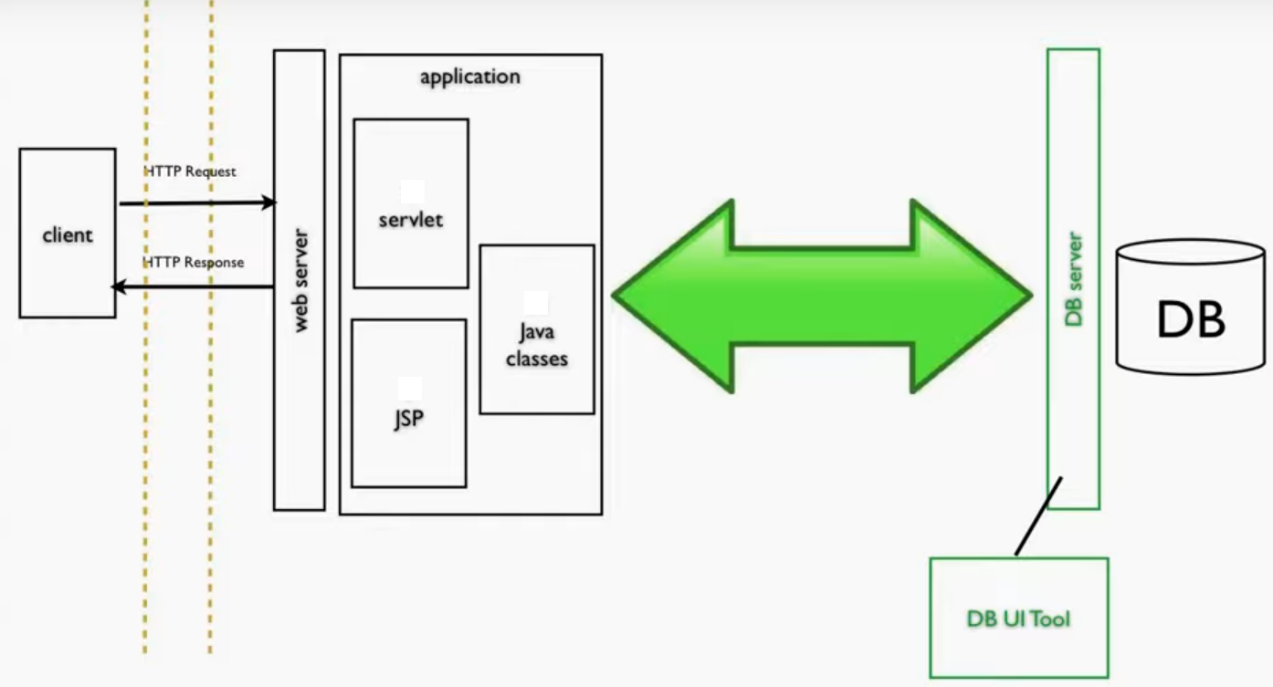
Последна таблица, която ще бъде създадена, е таблицата „files“ (Фигура 19), в която ще бъдат съхранени дипломните работи във вид на текстови файлове. Таблицата ще съдържа един първичен ключ върху колона „fileID“ и един външен ключ върху колона „thesisID“, за създаване на връзка между таблиците „thesis“ и „files“.



Фигура 19 – Таблица „files“

## **3.2 Проектиране и създване на уеб приложение**

Уеб приложенията работят чрез HTTP заявки, които клиента праща на уеб сървър, където самото уеб приложение е инсталирано. Тези заявки ще бъдат обработвани чрез сървлети (Servlets). Самите сървлети ще използват помощни Java класове за обработка на постъпилата информация. HTTP отговори ще бъдат предоставени на клиента чрез JSP-та, които представляват визуална форма. В много случаи самото уеб приложение ще трябва да се свърже с база данни за да достъпи данните за оформяне на отговор. Базите данни обикновено се сахранени на някое друго место. За тази цел ще се използва Database Server, в нашия случй MySQL Server. Този сървър ще позволи достъп до базата дании, обработка, съхранение и извличане на данни от нея. На фигура 20 графично е представен принцип на работа на едно уеб приложение.

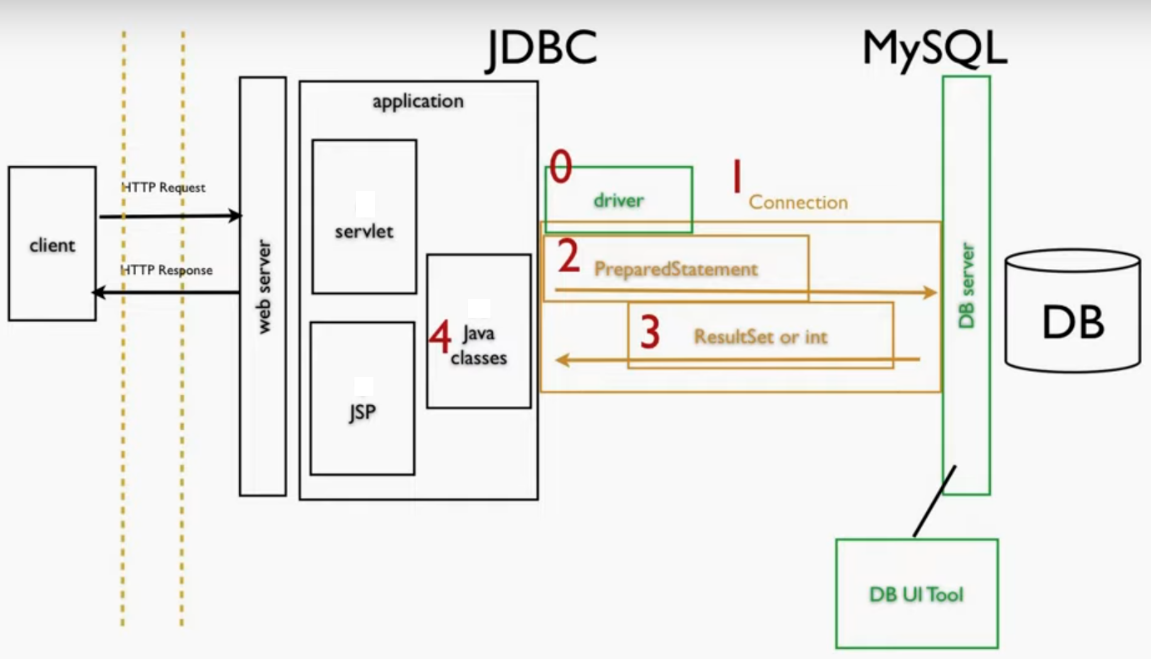


Фигура 20 – Принцип на работа на уеб приложение

За да получи информация от база данни, уеб приложението ще трябва да изпрати команди до сървъра на базата данни. След това сървъра на базата данни ще трябва да обработи получените команди и генерира резултат от тях. Този резултат представлява желаната информация за уеб приложението и трябва да бъде изпратен обратно към него. За реализация на този процес ще трябва да се създаде връзка между уеб приложението и самия сървър на базата данни. За тази цел ще се използват драйвъри. Те са малки програми, които играят роля на преводач между две технологии. Освен драйвъри ще има нужда и от JDBC (Java Database Connectivity), Java базирана технология за достъп до данни. JDBC предлага API за Java, където са дефинирани съвкупност от класове за достъп до база данни чрез самото уеб приложение. За добавяне, модифициране или изтриване на информация от базата данни, винаги ще се минава през следните стъпки:

1. Създаване на връзка между уеб приложението и базата данни.
2. Създаване на PreparedStatement, който позволява използване на SQL команди.
3. Извършване на SQL командите за да се получи резултат.
4. Обработка на получния резултат.

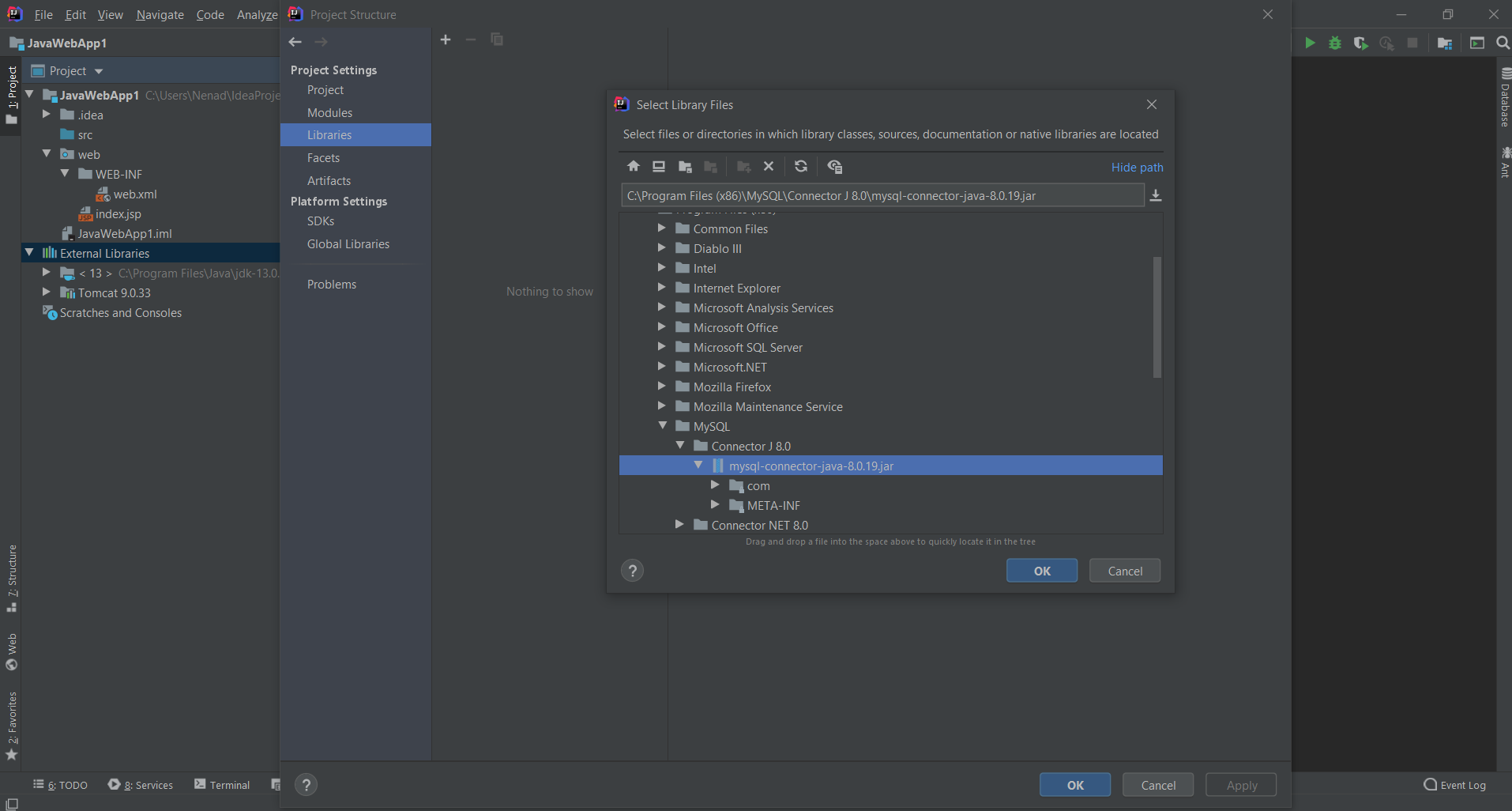
На фигура 21 представена е диаграма, която представлява принцип на работа на горе споменатите стъпки.



Фигура 21 - Принцип на работа на уеб приложение използващо Driver и JDBC

Вързката между приложението и сървъра на базата данни може да се представи като тунел, изобразен като оранжев правоъгълник от фигура 21. Тази връзка може да се осъществи с помощ на Connector J (Driver от фигура 21), който може да бъде изтеглен от MySQL страницата безплатно. След като драйвъра бъде прикачен към приложението, използването на JDBC класовете става възможно. След това се създава връзка с базата данни посредством Connection клас (стъпка 1 от фигура 21). За пращане на SQL заявки от приложението към сървъра на базата данни ще се използва PreparedStatement клас (стъпка 2 от фигура 21). Резултатите получени от изпълнение на SQL заявката ще се записват като обект на ResultSet класа и ще се връщат обратно към уеб приложението (стъпка 3 от фигура 21), където ще се обработват в Java класове (стъпка 4 от фигура 21).

Тези стъпки ще бъдат имплментирани върху уеб приложението, което ще бъде разработено в тази дипломна работа. Първо в IntelliJ IDEA ще бъде създаден нов проект тип „Web Application“ и ще му бъде зададено наименование. След това от падащото меню „File“ ще бъде отворен „Project Structure“ и избрана картичка „Libraries“. Тук ще бъде добавен „Connector J“ файла (Фигура 22), който представлява драйвър за връзка между приложението и сървъра на базата данни.



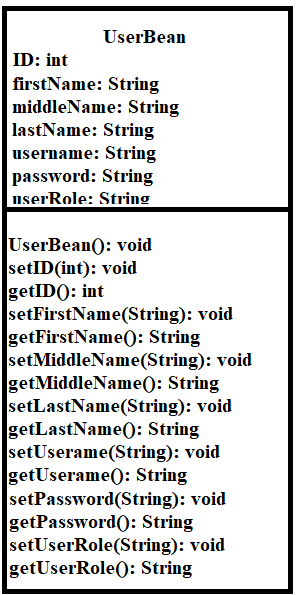
Фигура 22 – Добавяне на Connector J

След като драйвъра бъде прикачен, преминава се към създаване на връзка между приложението и сървъра на базата данни. В класа „DBConnection“ (Фигура 23) ще бъде създаден метод „createConnection“, който ще върща обект тип „Connection“. Ще бъдат дефинирани полета съдържащи необходима информация за създаване на връзка с базата данни: името на базата (dbName), адреса (url) й, потребителско име (user) и парола (pass) за достъп до нея. С помощ на метода „getConnection“ от класа „DriverManager“, който има нужда от дефинираните полета създава се връзка с базата данни. Връзката между базата данни и приложеобект ще бъде запазена в обект тип „Connection“.

package bean;  
public class DBConnection {  
 public static Connection createConnection(){  
 Connection conn = null;  
 String dbName = "vutpdb"; //Set your DB name  
 String url = "jdbc:mysql://localhost:3306/"+dbName; //Set URL and PORT to your DB  
 String user = "root"; //Set username for your DB  
 String pass = "ennoiretblanc"; //Set password for your DB  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver"); //Loading MySQL drivers  
 conn = DriverManager.*getConnection*(url,user,pass); //Attempting to connect  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 return conn;  
 }  
}

Фигура 23 – Java клас за осъществяване на връзка с база данни

Уеб приложенията, както вече беше казано работят с много потребители, затова приложението, което ще бъде разработено в дипломната работа ще дава възможност за регистриране на потребители. Успешно регистрираните потребители ще имат достъп до улугите на системата. На фигура 24 представена е UML клас диаграма описваща полета и методи на класа „UserBean“, чийто обекти ще съхраняват потребителите на системата. „UserBean“ е Java клас, който ще съдържа седем променливи от различни типове данни (int и String) и ще съдържа методи за задаване и получване на стойности на тези променливи, както и „default“ конструктор за създаване на обекти от този клас.



Фигура 24 – UML клас диаграма за UserBean

Базиран на UML клас диаграма, ще бъде създаден и Java клас в проекта. Java класовете ще бъдат сортирани в три пакета: „bean“ (основни елементи), „dao“ (помощни класове) и „controller“ (сървлети). В пакета „bean“ ще бъде създаден нов Java клас с име „UserBean“ и в него ще бъдат дефинирани променливи и „Getters & Setters“, т.е. методи за задаване и получаване на стойности на променливите (Фигура 25).

package bean;  
public class UserBean {  
 //Variables  
 private int ID;  
 private String firstName;  
 private String middleName;  
 private String lastName;  
 private String username;  
 private String password;  
 private String userRole;  
 //Getters and Setters  
 public int getID() {return ID;}  
 public void setID(int ID) {this.ID = ID;}  
 public String getFirstName() {return firstName;}  
 public void setFirstName(String firstName) {this.firstName = firstName;}  
 public String getMiddleName() {return middleName;}  
 public void setMiddleName(String middleName) {this.middleName = middleName;}  
 public String getLastName() {return lastName;}  
 public void setLastName(String lastName) {this.lastName = lastName;}  
 public String getUsername() {return username;}  
 public void setUsername(String username) {this.username = username;}  
 public String getPassword() {return password;}  
 public void setPassword(String password) {this.password = password;}  
 public String getUserRole() {return userRole;}  
 public void setUserRole(String userRole) {this.userRole = userRole;}  
}

Фигура 25 – Клас „UserBean“

След създаване на класа „UserBean“ ще трябва да се създаде помощен клас, който взима информация запазена в „UserBean“ обект за да добави потребителя в таблица „users“. Освен това този помощен клас ще трябва да проверява дали въведеното потребителско име не се използва от некои вече регистриран потребител. За тази цел в пакеата „dao“ ще бъде създаден нов Java клас „RegisterDao“. Той ще има два метода „registerUser“ и „doExist“. И двата метода за параметри ще взимат обект тип „UserBean“. На фигури 26 и 27 са представени методите.

public class RegisterDao {  
 //Register User (Insert entered data to users table)  
 public String registerUser (UserBean userBean){  
 String fName = userBean.getFirstName();  
 String mName = userBean.getMiddleName();  
 String lName = userBean.getLastName();  
 String uName = userBean.getUsername();  
 String pwd = userBean.getPassword();  
 Connection conn = null;  
 PreparedStatement ps = null;  
 String registerQuery = "INSERT INTO users (FirstName, MiddleName, LastName, Username, Password) VALUES (?,?,?,?,?)";  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(registerQuery);  
 ps.setString(1, fName);  
 ps.setString(2, mName);  
 ps.setString(3, lName);  
 ps.setString(4, uName);  
 ps.setString(5, pwd);  
 int i = ps.executeUpdate();  
 if (i != 0){  
 return "SUCCESS.";  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return "Something went wrong.";  
 }

Фигура 26 – Метод „registerUser“

//Check if User exists  
public boolean doExist(UserBean userBean){  
 String username = userBean.getUsername();  
 String checkQuery = "SELECT \* FROM users WHERE Username='"+username+"'";  
 Connection conn = null;  
 PreparedStatement ps = null;  
 ResultSet rs = null;  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(checkQuery);  
 rs = ps.executeQuery();  
  
 if (rs.next()){  
 return false;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return true;  
}

Фигура 27 – Метод „doExist“

Метода „registerUser“ ще върща String стойност „SUCCESS.“, ако потребителя успешно е регистриран, или „Something went wrong.“ ако регистрацията не е била успешна. В самия метод ще се създава връзка с базата данни, след което ще се изпълнява SQL заявка за попълване на таблица „users“. Резултат от изпълнението на тази заявка се представлява като число. Ако числото е по-голямо от 0 то означава че заявката е изпълнена успешно. Другия метод („doExist“) връща „boolean“ стойност т.е. „TRUE“ или „FALSE“. Основна задача на този метод ще бъде да провери за съвпадане на данни и ако няма такива да позволи изпълнението на „registerUser“ метода.

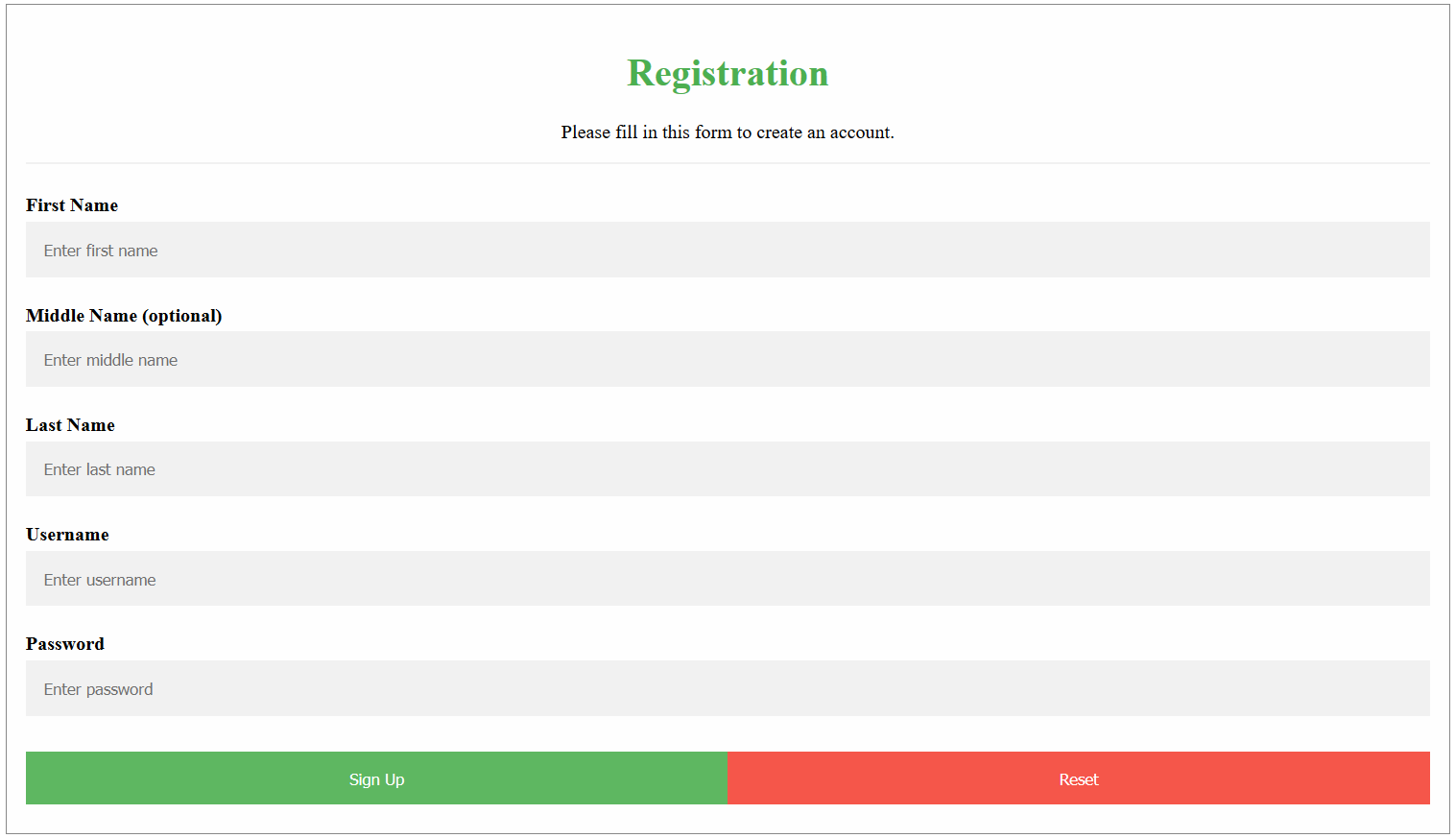
Следваща стъпка е създаване на сървлет, който ще обедини дефинираните методи. Java сървлет представлява програма на Java, която приема като вход някакви данни от потребителя, обработват ги и връщат като резултат динамично генериран HTML или друг документ. Ще бъде създаден сървлет „RegisterServlet“ в пакета „controllers“. Задача на този сървлет ще бъде събиране на информация от текстови полета на страницата за регистрация на потребители. За тази цел ще трябва да се създаде и самата страница, като и връзка до нея. В папката „web“ ще бъде създаден нов JSP документ наречен „Register“. В него ще бъде създадена форма за регистрация, която съдържа текстови полета и два бутона, един за регистриране, а другия за изтриване на информация от полетата. На началната страница „index.jsp“ ще бъде създадена връзка към „Register“ страницата. При отваряне на връзката, потребителя ще бъде препратен към „Register.jsp“. След като текстовите полета бъдат запълнени, чрез бутона „Sing Up“ ще бъде изпълнен „RegisterServlet“. За да всичко това работи в „action“ HTML тага на самата форма за регистрация ще бъде зададено името на сървлета. Формата е представена на фигури 28, 29 и 30. На фигура 28 е представен JavaScript код, който проверява за празни полета, на фигура 29 са представени всичките елементи на страничката, а на фигура 30 е представена самата страничка която потребителя вижда. Към страничката е прикачен вече създаден CSS файл за дизайна й.

<script>  
 function validate()  
 {  
 var firstName = ***document***.form.first.value;  
 var lastName = ***document***.form.last.value;  
 var username = ***document***.form.uname.value;  
 var password = ***document***.form.pwd.value;  
  
 if (firstName==null || firstName==""){  
 alert("First Name field required");  
 return false;  
 }else if (lastName==null || lastName==""){  
 alert("Last Name field required");  
 return false;  
 }else if (username==null || username==""){  
 alert("Username field required");  
 return false;  
 }else if(password==null && password.length<8){  
 alert("Password must be at least 8 characters long.");  
 return false;  
 }  
 }  
 </script>

Фигура 28 – JavaScript за проверка на полетата

<body>  
<form action="RegisterServlet" method="post" onsubmit="return validate()" accept-charset="UTF-8" class="modal-content">  
 <div class="container">  
 <h1 id="signIn">Registration</h1>  
 <p id="signInP">Please fill in this form to create an account.</p>  
 <hr>  
 <label for="FirstName"><b>First Name</b></label>  
 <input type="text" placeholder="Enter first name" name="first" required>  
 <label for="MiddleName"><b>Middle Name (optional)</b></label>  
 <input type="text" placeholder="Enter middle name" name="middle">  
 <label for="LastName"><b>Last Name</b></label>  
 <input type="text" placeholder="Enter last name" name="last" required>  
 <label for="Username"><b>Username</b></label>  
 <input type="text" placeholder="Enter username" name="uname" required>  
 <label for="Password"><b>Password</b></label>  
 <input type="password" placeholder="Enter password" name="pwd" required>  
 <div class="clearfix">  
 <button type="submit" class="signupbtn">Sign Up</button>  
 <button type="reset" class="resetbtn">Reset</button>  
 </div>  
 </div>  
</form>  
</body>

Фигура 29 – Елементите на страничката и техните имена



Фигура 30 – Финалния изглед на страничката за регистрация

Следваща стъпка за създаване на пълно функционална форма за регистриране е самото кодиране на „RegisterServlet“. В „doPost()“ метода на сървлета, ще бъде взета информация, въведена от потребителя и след това съхранена в променливи. След това ще бъде създаден обект тип „UserBean“ и на неговите свойства зададени съхранените променливи. След това ще се създаде обект тип „RegisterDao“, който ще проверява за вече заети потребителски имена. В случй да няма съвпадения, ще бъде извикан метода за регистрация и резултата от изпълнението му ще бъде записан в променлива. След това ще се провери дали текста, който се получава от изпълнение на метода за регистрация, съвпада със „SUCCESS.“ или „Something went wrong“ и спрямо това ще бъде определено дали потребителя е направил успешна регистрация и на коя страница да бъде изпратен (Фигури 31 и 32).

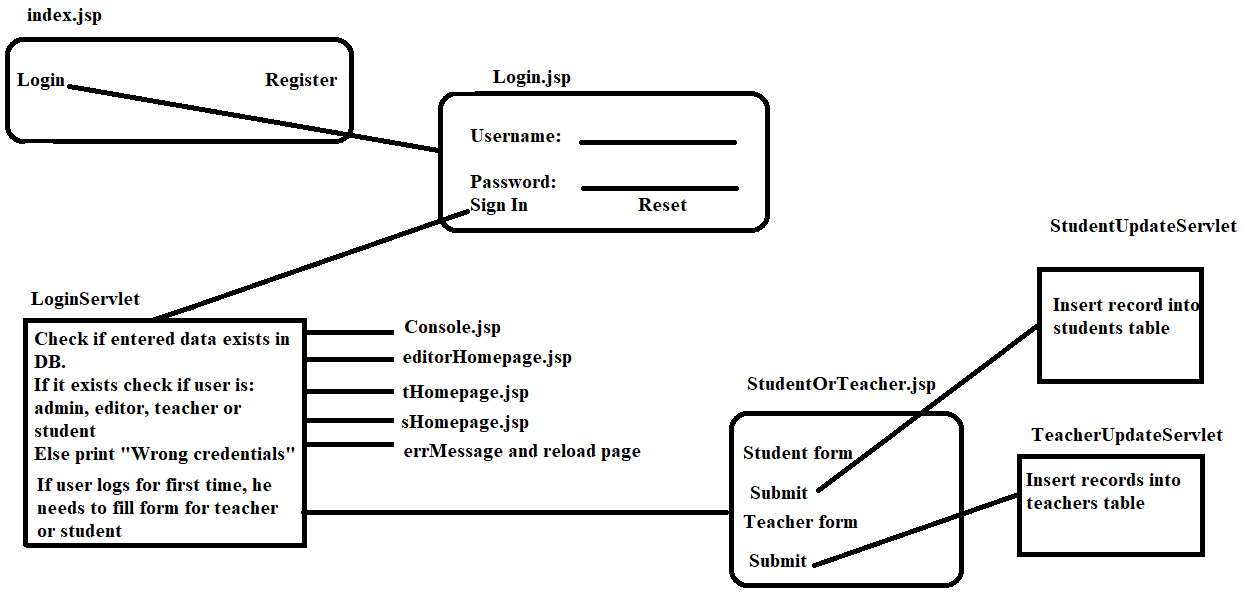
@WebServlet(name = "RegisterServlet")  
public class RegisterServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String firstName = request.getParameter("first");  
 String middleName = request.getParameter("middle");  
 String lastName = request.getParameter("last");  
 String userName = request.getParameter("uname");  
 String password = request.getParameter("pwd");  
 //Assign values to UserBean object  
 UserBean userBean = new UserBean();  
 userBean.setFirstName(firstName);  
 userBean.setMiddleName(middleName);  
 userBean.setLastName(lastName);  
 userBean.setUsername(userName);  
 userBean.setPassword(password);

Фигура 31 – RegisterServlet

//Creating RegisterDao object to perform check and register  
 RegisterDao registerDao = new RegisterDao();  
 //Check if username already exists  
 if (registerDao.doExist(userBean)) {  
 //If NOT then register and redirect to Login (index) page  
 String userRegistered = registerDao.registerUser(userBean);  
 if (userRegistered.equals("SUCCESS.")) {  
 request.getRequestDispatcher("/index.jsp").forward(request, response);  
 } else {  
 //If username exists display Error Message and reload page  
 request.setAttribute("errMessage", userRegistered);  
 request.getRequestDispatcher("/Register.jsp").forward(request, response);  
 }  
 } else {  
 request.getRequestDispatcher("/Register.jsp").forward(request, response);  
 }  
}

Фигура 32 – RegisterServlet

След успешна регистрация, потребителите на системата ще могат да използва потребителските си имена и паролите си за влизане в профилите си. Този процес ще бъде по-детайлно разгледан с помощ на схема от фигура 33.



Фигура 33 – Стъпки за влизане на потребителя в профила си

От началната страничка, потребиля ще може да влезе в профила си с натискане на бутон „Login“. Този бутон ще отваря нова страничка, където ще се въвеждат потребителско име и парола. След натискане на бутона „Sing In“ ще бъде извикан „LoginServlet“. Този сървлет ще използва помощни класове за валидация на потребителите. В случай че е въведена грешна информация, на екрана на потребителя ще се извежда съобщение за грешно потребителско име или парола. От друга страна ако потребителя успешно влезе в профила си, в зависимост от потребителското му име, той ще бъде препратен към съответна страница. Администратора на системата ще бъде препратен към „Console.jsp“, едитора ще бъде препратен към „editorHomepage.jsp“, а останалите потребители към „tHomepage.jsp“ или „sHomepage.jsp“. Когато потребителя влиза в системата за пръв път, той ще бъде принуден да попълни формуляр, където декларира дали е студент или преподават. Този формуляр се намира на страничка „StudentOrTeacher.jsp“. След потвърждение на внесената информация, ще бъдат извикани „StudentUpdateServlet“ или „TeacherUpdateServlet“, които ще записват въведената от потребителя информация в съответни таблици „students“ или „teachers“.

„LoginServlet“ се нуждае от помощни класове за да може да работи, затова ще бъде създаден „LoginDao“ клас, който ще съдържа различни методи. Главна задача на този клас ще бъде да провери дали въведената от потребителя информация е вярна. Тази задача ще изпълнява метод „login“ (Фигура 34), който като параметър взима обект тип „UserBean“. Метода ще трябва да получи информация от обекта за потребителско име и парола записани в него и след това да ги запомни в променливи. След тов метода ще трябва да създаде връзка с базата данни за да провери дали съществуват в нея. Като резулт от изпълнението си, метода върща String, който ще бъде използван от сървлета по-късно за генериране на HTML отговор.

public class LoginDao {  
 public String login(UserBean userBean){  
 String username = userBean.getUsername();  
 String password = userBean.getPassword();  
 Connection conn = null;  
 PreparedStatement ps = null;  
 ResultSet rs = null;  
 String dbUsername = "";  
 String dbPassword = "";  
 conn = DBConnection.*createConnection*();  
 String query = "SELECT Username, Password FROM users";  
 try {  
 ps = conn.prepareStatement(query);  
 rs = ps.executeQuery();  
 while (rs.next()){  
 dbUsername = rs.getString("Username");  
 dbPassword = rs.getString("Password");  
 if ((username.equals(dbUsername)) && (password.equals(dbPassword))){  
 return "SUCCESS.";  
 }  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return "Invalid Username or Password.";  
 }

Фигура 34 – Метод „login“

За проверка дали потребителя влиза в системата за пръв път ще бъдат създадени отделни методи методи „isStudent“ (Фигура 35) и „isTeacher“ (Фигура 36). Тези методи ще имат за задача да проверят дали логнатият потребител е попълнил формуляра от страницата „StudentOrTeacher.jsp“ т.е. дали „ID“ на потребителя е записан в таблиците „students“ или „teachers“.

//Check if ID of logged user exists in students table  
public boolean isStudent(int id){  
 String checkQuery = "SELECT \* FROM students WHERE ID='"+id+"'";  
 Connection conn = null;  
 PreparedStatement ps = null;  
 ResultSet rs = null;  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(checkQuery);  
 rs = ps.executeQuery();  
 while (rs.next())  
 return true;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return false;  
}

Фигура 35 – Метод „isStudent“

//Check if ID of logged user exists in teachers table  
public boolean isTeacher(int id){  
 String checkQuery = "SELECT \* FROM teachers WHERE ID='"+id+"'";  
 Connection conn = null;  
 PreparedStatement ps = null;  
 ResultSet rs = null;  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(checkQuery);  
 rs = ps.executeQuery();  
 while (rs.next())  
 return true;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return false;  
}

Фигура 36 – Метод „isTeacher“

За получаване на „ID“ на логнатият потребител ще бъде използван отделен метод „obtainUserID“ (Фигура 37). Този метод като параметър ще взима обект тип „UserBean“ и от него извлича информация за запазеното в него потребителско име. Това потребителско име ще бъде обработено с помощ на SQL заявка обработва за да се намери кореспондиращо „ID“.

//Obtain logged user's ID  
public int obtainUserID(UserBean userBean){  
 String username = userBean.getUsername();  
 String query = "SELECT ID FROM users WHERE Username='"+username+"'";  
 int loggedUserID = 0;  
 Connection conn = null;  
 PreparedStatement ps = null;  
 ResultSet rs = null;  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(query);  
 rs = ps.executeQuery();  
 while (rs.next()){  
 loggedUserID = rs.getInt("ID");  
 return loggedUserID;  
 }  
 }  
 catch (SQLException e) {  
 e.printStackTrace();  
 } return 0;  
}

Фигура 37 – Метод „obtainUserID“

След създаване на помощни методи, може да се премине към кодиране на „LoginServlet“. Сървлета ще взима въведената от потребителя информация във формата на „Login.jsp“ страница (Фигура 38). Страницата ще съдържа скрипт (Фигура 39), който ще проверява дали всички полета са попълнени.

<body>  
<form action="LoginServlet" method="post" onsubmit="return validate()" class="modal-content">  
 <div class="container">  
 <h1 id="logIn">Login</h1>  
 <label for="uname"><b>Username</b></label>  
 <input type="text" placeholder="Enter Username" name="uname" required>  
 <label for="pwd"><b>Password</b></label>  
 <input type="password" placeholder="Enter Password" name="pwd" required>  
 <span style="color:red"><%=(request.getAttribute("errMessage") == null) ? "": request.getAttribute("errMessage")%></span></td>  
 <div class="clearfix">  
 <button type="submit" name="submitbtn" class="submitbtn">Login</button>  
 <button type="reset" name="resetbtn" class="resetbtn">Reset</button>  
 </div>  
 </div>  
</form>  
</body>

Фигура 38 – Елементи на страница „Login.jsp“

<script>  
 function validate(){  
 var username = ***document***.form.username.value;  
 var password = ***document***.form.password.value;  
 if (username==null || username=="") {  
 alert("Username required");  
 return false;  
 } else if(password==null || password=="") {  
 alert("Password required");  
 return false;  
 }  
 }  
</script>

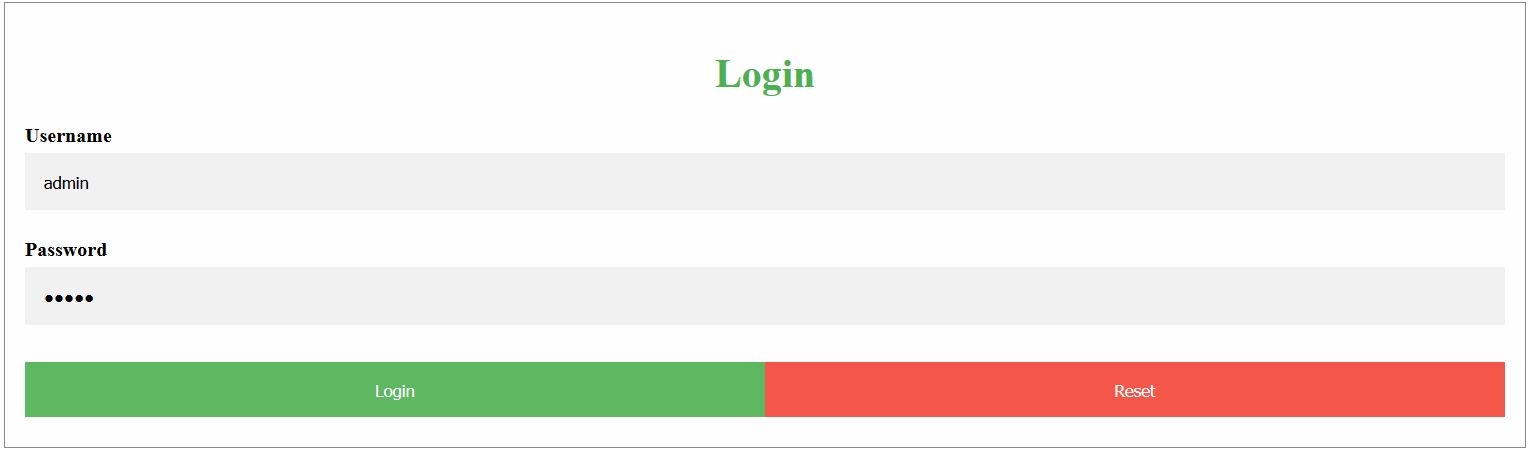
Фигура 39 – Скрипт validate()

Информацията получена от „Login“ страницата ще бъде запомнена в полета на обект тип „UserBean“. Този обект ще бъде използван за параметри на методи от „LoginDao“ класа. Следваща задача на сървлета ще бъде изпълнение на метода „login“ от „LoginDao“ класа. Ако метода „login()“ като резултат от изпълнението си върне съобщение „SUCCESS.“, ще се премине към изпълнение на останалите методи от класа „LoginDao“. Първо ще се провери дали въведеното потребителско име е „admin“. При съвпадение, потребителя ще бъде препратен към страница „Console.jsp“. Когато потребителското име се различава от „admin“, а потребителя е извършил успшно логване ще бъде извикан метода „obtainUserID“ за да се получи „ID“ на потребителя. След това ще бъде създадена HTTP сесия, в която ще се запомни ID-то на логнатия потребител. Този „ID“ ще бъде използван за проверка дали потребителя е студент или преподавател. Ако потребителя е студент той ще бъде препратен към начална страница за студенти „sHomepage.jsp“, а ако потребителя е преподавател, той ще бъде препратен към начална страница за преподаватели „tHomepage.jsp“. В случай да ID-то не бъде намерено нито в таблицата за студенти нито в таблицата за преподаватели, потребителя ще бъде препратен към „StudentOrTeacher.jsp“ страница. На тази страница потребителите ще трябва да попълни необходима информация за да бъде записан в една от таблиците „students“ или „teachers“. Последната роля на сървлъта ще бъде да изведе съобщение „Invalid Username or Password.“ в случай че въведените потребителско име и парола са грешни. На фигура 40 се намира код записан в сървлета „LoginServlet“. На фигура 41 представена „Login“ страницата.

@WebServlet(name = "LoginServlet")  
public class LoginServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String username = request.getParameter("uname");  
 String password = request.getParameter("pwd");  
 UserBean userBean = new UserBean();  
 userBean.setUsername(username);  
 userBean.setPassword(password);  
 LoginDao loginDao = new LoginDao();  
 String userValidation = loginDao.login(userBean);  
 if (userValidation.equals("SUCCESS.")){  
 if (username.equals("admin")){  
 request.getRequestDispatcher("/Console.jsp").forward(request, response);  
 } else {  
 int userID = loginDao.obtainUserID(userBean);  
 HttpSession session = request.getSession();  
 session.setAttribute("id", userID);  
 if (loginDao.isStudent(userID)) {  
 request.getRequestDispatcher("/sHomepage.jsp").forward(request, response);  
 } else if (loginDao.isTeacher(userID)) {  
 request.getRequestDispatcher("/tHomepage.jsp").forward(request, response);  
 } else if (username.equals("editor")){

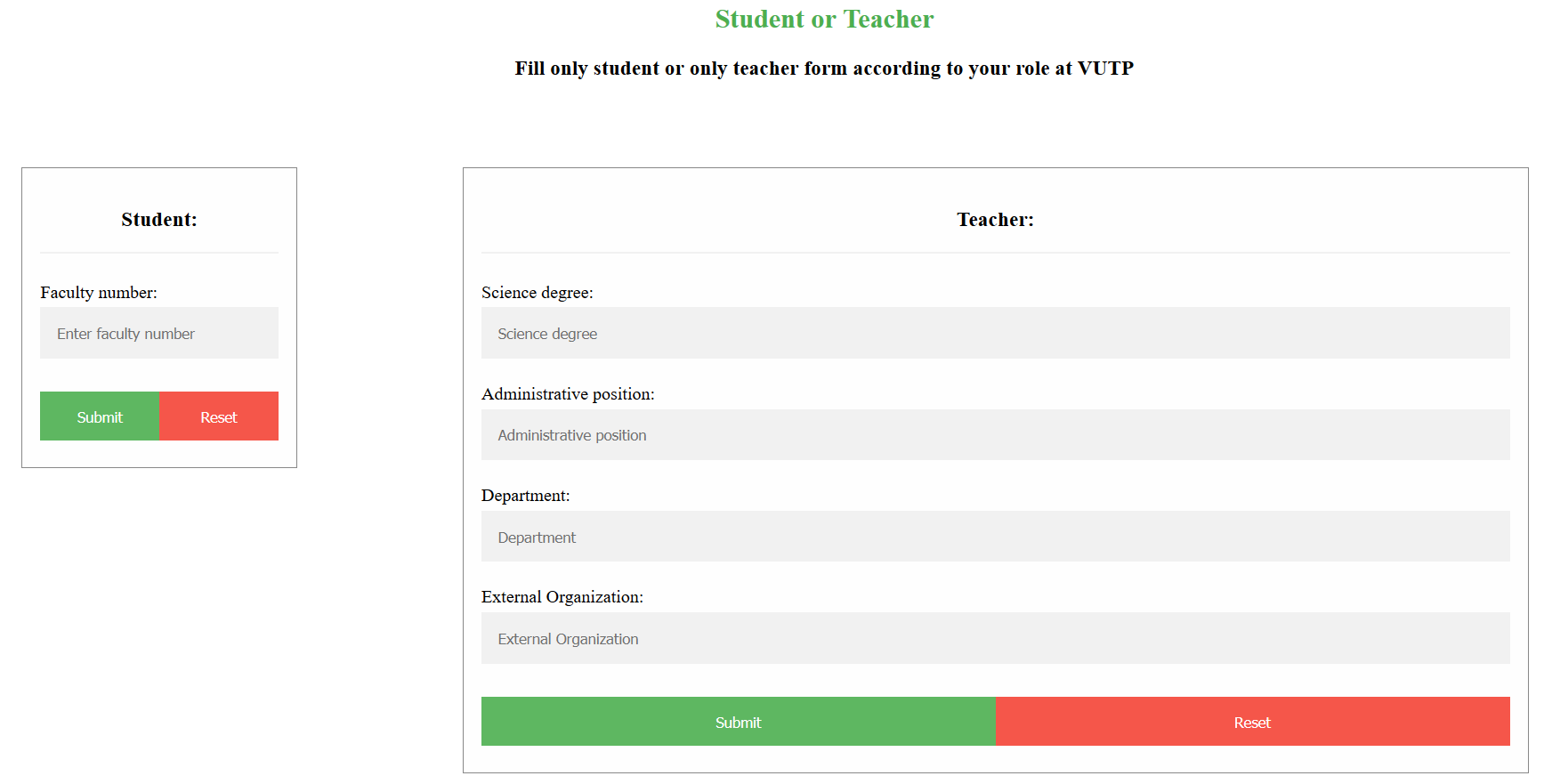
request.getRequestDispatcher("/editorHomepage.jsp").forward (request,response);  
 } else request.getRequestDispatcher("/StudentOrTeacher.jsp").forward(request, response);  
 }  
 } else {  
 request.setAttribute("errMessage", userValidation);  
 request.getRequestDispatcher("/Login.jsp").forward(request, response);  
 }  
 }

Фигура 40 – LoginServlet



Фигура 41 – Login страница

Много важна роля в базата данни ще имат таблиците „students“ и „teachers“. Тези таблици ще разделят потребителите на студенти и преподаватели. Таблицата „students“ съдържа две колони описващи студенти: „ID“ и „FacultyNumber“. От друга страна таблицата „teachers“ съдържа пет колони, които описват всеки регистиран преподавател: „ID“, „NaucnaStepen“, „AdministrativnaDlaznost“, „Katedra“ и „VansnaOrganizacija“. Понеже всеки потребител има уникално „ID“, то ще бъде използвано за разпределяне на потребителите. „ID“ на логнатите потребителите, както беше казано, ще бъде запаметено в предишно създадена сесия. Когато потребителя бъде препратен към „StudentOrTeacher.jsp“ страница, той ще трябва да попълни само исканата от него информация, която системата, след натискане на бутона „Submit“, ще обработи и попълни таблиците. На фигура 42 представена е страницата „StudentOrTeacher“. Тя ще се състои от две форми. В „Student“ формата от потребителя ще се изисква факултетния му номер. Тази форма ще се попълва само от студенти. Другата, „Teacher“ форма, от потребителя ще изисква информация за научната му степен, административната му длъжност, катедра на която преподава и външна организация ако е член на такава. Тази форма ще попълват само преподаватели. Двете форми ще използват два различни сървлета „StudentUpdateServlet“ и „TeacherUpdateServlet“.



Фигура 42 – StudentOrTeacher страница

За да работят, и двата сървлета ще имат нужда от помощни Java класове, в които ще бъдат дефинирани методи за изпълнение на SQL заявки за попълване на таблиците „students“ и „teachers“. Първо ще бъдат създадени помощните класове „StudentDao“ и „TeacherDao“. В класа „StudentDao“ ще бъде дефиниран метод за попълване на таблицата „students“, наречен „updateStudentInfo“ (Фигура 43). Този метод като параметри ще взима обект тип „UserBean“ и една целочислена променлива „int facNum“. На обекта ще бъде зададен ID взет директно от сесията, където е запомнен. В променливата „facNum“ ще бъде съхранено цело число взето от потребителя. При изпълнение на този метод, първо ще бъде създадена връзка с базата данни, а след това ще бъде изпълнена SQL заявка за попълване на таблица „students“. Ако заявката бъде успешно изпълнена като резултат метода ще връща String „SUCCESS.“, в противен случай върнатият String ще бъде „Something went wrong.“

package dao;  
import bean.DBConnection;  
import bean.UserBean;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.SQLException;  
public class StudentDao {  
 public String updateStudentInfo(UserBean userBean, int facNum){  
 int ID = userBean.getID();  
 String query = "INSERT INTO students (ID, FacultyNumber) VALUES (?, ?)";  
 Connection conn = null;  
 PreparedStatement ps = null;  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(query);  
 ps.setInt(1, ID);  
 ps.setInt(2, facNum);  
 int i = ps.executeUpdate();  
 if (i != 0){  
 return "SUCCESS.";  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return "Something went wrong.";  
 }  
}

Фигура 43 – Метод „updateStudentInfo“

Следваща стъпка ще бъде имплементиране на дефинирания метод в сървлет „StudentUpdateServlet“ (Фигура 44). Сървлета ще вземе запомнения „ID“ на потребителя от сесията и ще го запомни в променлива. След това ще вземе внесения от потребителя факултетен номер и него ще съхрани в променлива. След това ще бъде създаден обект тип „UserBean“, на който за стойност на поле „ID“ ще бъде зададена запомнената от сесията стойност. За да се осигури достъп до метода за попълване на таблица „students“ трябва да бъде създаден обект тип „StudentDao“. В параметрите на метода ще бъдат зададени обект тип „UserBean“ и променливата, която съхранява факултетния номер. След изпълнение на сървлета, потребителя ще бъде изпратен на начнала страница за студенти, при успешно изпълнение, или ще получи съобщение при възникване на грешка.

@WebServlet(name = "StudentUpdateServlet")  
public class StudentUpdateServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 HttpSession session = request.getSession(false);  
 int userID = (int) session.getAttribute("id");  
 int facultyNumber = Integer.*parseInt*(request.getParameter("studentFacNum"));  
 UserBean userBean = new UserBean();  
 userBean.setID(userID);  
 StudentDao studentDao = new StudentDao();  
 String updatedStudentInfo = studentDao.updateStudentInfo(userBean, facultyNumber);  
 if (updatedStudentInfo.equals("SUCCESS.")){  
 request.getRequestDispatcher("/sHomepage.jsp").forward(request, response);  
 } else {  
 request.setAttribute("errMessage", updatedStudentInfo);  
 request.getRequestDispatcher("/StudentOrTeacher.jsp").forward(request, response);  
 }  
 }

Фигура 44 – StudentUpdateServlet

Същите стъпки ще бъдат повторени за попълване на „teachers“ таблица. Ще бъдат създадени помощен Java клас „TeacherDao“ и сървлет „TeacherUpdateServlet“. В помощния клас ще бъде дефиниран метод „updateTeacherInfo“ (Фигура 45), който за задача ще има да попълни таблицата „teachers“ с помощ на SQL заявка. Метода за параметри ще използва обект тип „UserBean“ и четири String-а, в които ще бъде съхранена информация за научна степен, административна длъжност, катедра и външна организация. Този метод ще бъде имплементиран в сървлета „TeacherUpdateServlet“ (Фигура 46), където от създадената сесия ще бъде взет „ID“ на потребителя, а от полетата на формата останалата необходима информация.

public class TeacherDao {  
 public String updateTeacherInfo(UserBean userBean, String ns, String ad, String ka, String vo){  
 int ID = userBean.getID();  
 String query = "INSERT INTO teachers (ID, NaucnaStepen, AdministrativnaDlaznost, Katedra, VansnaOrganizacija) VALUES (?, ?, ?, ?, ?)";  
 Connection conn = null;  
 PreparedStatement ps = null;  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(query);  
 ps.setInt(1, ID);  
 ps.setString(2, ns);  
 ps.setString(3, ad);  
 ps.setString(4, ka);  
 ps.setString(5, vo);  
 int i = ps.executeUpdate();  
 if (i != 0){  
 return "SUCCESS.";  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return "Something went wrong.";  
 }

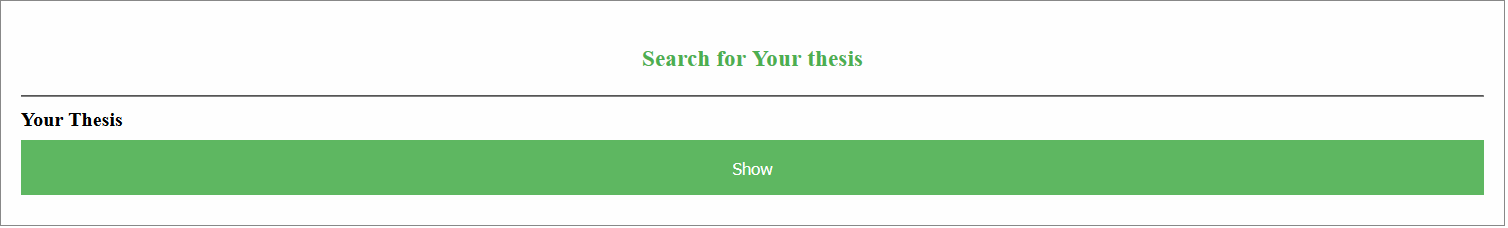
Фигура 45 – Метод „updateTeacherInfo“

@WebServlet(name = "TeacherUpdateServlet")  
public class TeacherUpdateServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 HttpSession session = request.getSession(false);  
 int userID = (int) session.getAttribute("id");  
 String ns = request.getParameter("naucnaStepen");  
 String ad = request.getParameter("administrativnaDlaznost");  
 String ka = request.getParameter("katedra");  
 String vo = request.getParameter("vansnaOrganizacija");  
 UserBean userBean = new UserBean();  
 userBean.setID(userID);  
 TeacherDao updateTeacherDao = new TeacherDao();  
 String updatedTeacherInfo = updateTeacherDao.updateTeacherInfo(userBean, ns, ad, ka, vo);  
 if (updatedTeacherInfo.equals("SUCCESS.")){  
 request.getRequestDispatcher("/sHomepage.jsp").forward(request,response);  
 } else {  
 request.setAttribute("errMessage", updatedTeacherInfo);  
 request.getRequestDispatcher("/StudentOrTeacher.jsp").forward(request,response);  
 }  
 }

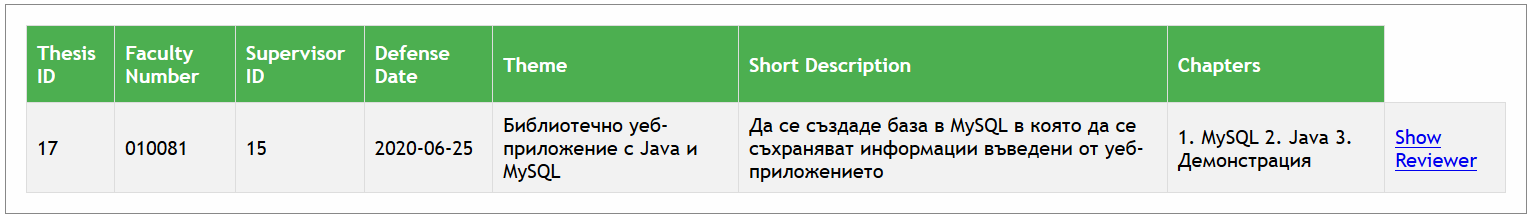
Фигура 46 – TeacherUpdateServlet

След успешно влизане в профилите си, потребителите ще имат достъп до различни възможности. Например. студентите ще имат възможност да получат необходима информация за дипломна работа, която трябва да защитават. Също така ще имат възможност за търсене на дипломни работи по различни критерии и за изтеглянето им. Освен това, студентите ще могат да качват своите готови дипломни работи в системата. За преподавателите най-важната функция ще бъде задаване на дипломни работи. Преподавателите ще могат да видят всички дипломни работи, където те са ръководители. Освен тази, преподавателите ще имат достъп и до всички възможности зададени и на студентите. Едиторите ще бъдат специални потребители, които могат да изменят имена на потребителите, и да модифицират информацията свързана с дипломните работи. Администратора на системата ще има възможност да сменя пароли и потребителски имена на потребителите. Всички потребители трябва да имат възможност да прекъснат вързка със системата, това ще им бъде предоставено с бутона „Logout“. Ако има нужда от допълнителни функции те по предложение могат да бъдат лесно добавени към определена група потребители.

Както беше казано, студентите ще имат възможност да получат информация за дипломна работа, която те трябва да защитават. На фигура 47 представен е бутон чрез който се отваря таблица, съдържаща информация за дипломна работа (Фигура 48).



Фигура 47 – Бутон за приказване на дипломна работа



Фигура 48 – Таблица съдържаща информация за дипломната работа

При натискане на бутона „Show“ студента ще изпраща HTTP заявка за получаване на информация от таблица „thesis“ съдъражща информации за всички дипломни работи. За да получи информация за интересуващата го дипломна работа, сървлета ще взима факултетния му номер и ще търси за съвпадения в таблицата „thesis“. За да работи, сървлета ще се нуждае от помощен клал „ThesisDao“, в който ще бъдат дефинирани методи за получване на факултетен номер, за изваждане на информация от таблицата и за принтиране на получените резултати във вид на HTML таблица.

Метод, който ще бъде използван за получване на факултетен номер на студента, като параметър ще взима „ID“ на логнатия студент. След това ще изпълни SQL заявка, която в „ResultSet“ обекта ще върща факултетен номер на студента (Фигура 49).

//Obtain students Faculty Number  
public int getFacultyNumber (int ID){  
 int facNum = 0;  
 String query = "SELECT FacultyNumber FROM students WHERE ID="+ID;  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(query);  
 rs = ps.executeQuery();  
 while (rs.next()){  
 facNum = rs.getInt("FacultyNumber");  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return facNum;  
}

Фигура 49 – Метод „getFacNum“

Следваща стъпка ще бъде обработка на получения факултетен номер. Той ще бъде подаден на метод „showYourThesis“ като параметър и след изпълнение на SQL заявка в „ResultSet“ обект ще бъдат запомнени резултатите от изпълнението й (Фигура 50).

//Get thesis row for given Faculty Number  
public void showYourThesis(int facNum){  
 String query = "SELECT \* FROM thesis WHERE FacultyNumber='"+facNum+"'";  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(query);  
 rs = ps.executeQuery();  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
}

Фигура 50 – Метод „showYourThesis“

Последния метод, който ще бъде използван за изпълнение на тази задача е „printThesisTable“. Това е метод, който ще създаде HTML таблица и в нея запише информация запазена в „ResultSet“ обекта. В последната колона на таблицата ще се намира връзка, чрез която се показват рецензенти заделени за тази дипломна работа. Този метод ще има нужад от нов Java клас „ThesisBean“. „ThesisBean“ ще се състои от променливи, които представляват колоните на таблицата „thesis“. За всеко поле на този клас ще бъдат дефинирани и „Getters & Setters“.

След като бъдат дефинирани всички необходими методи ще бъде създаден сървлет за изпълнение на HTTP заявката, подадена от студента. Сървлета ще използва създадената сесия за получаване на „ID“ на студента. Този „ID“ ще бъде обработен от метода „getFacNum“. Получения факултетен номер, след това ще бъде зададен като параметър на метода „showYourThesis“. Резултата от изпълнение на този метод ще бъде запазен в ResultSet обект. Информацията запомнена в ResultSet обекта ще бъде принтира във вид на таблица с помощ на метода „printYourThesis“. Таблица с резултатите ще бъде запомнена в променлива с помощ на „setAttribute“ метода на Java Servlet. Този метод запазва таблицата под име „yourThesisTable“. След изпълнение на всички методи потребителя ще бъде препратен на страница „resultYourThesisTable.jsp“. Страницата на която потребителя ще бъде препратен ще използва „getAttribute“ метод за да полъчи запомнената таблица. Като праметър на „getAttribute“ метода ще бъде зададено името на променливата, в която е запазен обекта, в случая „yourThesisTable“. Структура на страницата е приказана на фигура 52, а сървлета „SearchYourThesisServlet“ е представен на фигура 51.

@WebServlet(name = "SearchYourThesisServlet")  
public class SearchYourThesisServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 //Get logged user ID from session  
 HttpSession session = request.getSession(false);  
 int userID = (int) session.getAttribute("id");  
 ThesisDao thesisDao = new ThesisDao();  
 //Get logged user Faculty Number  
 int facNum = thesisDao.getFacultyNumber(userID);  
 //Get thesis records for obtained Faculty Number  
 thesisDao.showYourThesis(facNum);  
 //Printing records as a table  
 String yourThesisTable = thesisDao.printThesisTable();  
 //Saving table into request to be used on another page  
 request.setAttribute("yourThesisTable", yourThesisTable);  
 //Redirecting to another page with results  
 RequestDispatcher dispatcher = request.getRequestDispatcher("resultYourThesisTable.jsp");  
 dispatcher.forward(request,response);  
 }

Фигура 51 – Сървлет SearchYourThesisServlet

<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<%  
 String resultTable = (String) request.getAttribute("yourThesisTable");  
%>  
<html>  
<head>  
 <title>VUTP DB Project: Results For Your Thesis</title>

<link href="css/thesisTablePage.css" rel="stylesheet" type="text/css">  
</head>

<body>  
<h2>Thesis information</h2>  
<div class="modal-content">  
 <div class="container">  
 <%=resultTable%>  
 </div>  
</div>  
<form action="showUsersInfo" method="get" class="modal-content">  
 <div class="container">  
 <h2>Show Teachers</h2>  
 <div class="clearfix">  
 <button type="submit" class="submitbtn">Show</button>  
 </div>  
 </div>  
</form>  
</body>

Фигура 52 – Елементите на страница resultYourThesisTable

Страницата „resultYourThesisTable.jsp“ съдържа и форма която покзва таблица с преподаватели, за да може студента да разбере кой му е ръководител. Тази форма при натискане на бутона „Show“ ще извика сървлет „ShowUserInfoServlet“, който ще прикаже таблица с преподаватели и техните ID-та. За създаване на този сървлет ще бъде създаден помощент клас „ShowUserInfoDao“ и в него дефинирани методи за извличане на частична информация от таблица „users“. На студента ще бъдат приказани имена и ID-та на преподавателите.

Първия метод, който ще използва сървлета е „showTeachers“ (Фигура 53). Този метод ще изпълнява SQL заявка и резултата от изпълнението й ще запомня в ResultSet обект.

public class ShowUsersInfoDao {  
 Connection conn = DBConnection.*createConnection*();  
 PreparedStatement ps;  
 ResultSet rs;  
 public void showTeachers (){  
 String query = "SELECT ID, FirstName, MiddleName, LastName FROM users WHERE NOT Username='admin' AND NOT Username='editor'" +  
 "AND ID IN (SELECT ID FROM teachers)";  
 try {  
 ps = conn.prepareStatement(query);  
 rs = ps.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();}}

Фигура 53 – Метод „showTeachers“

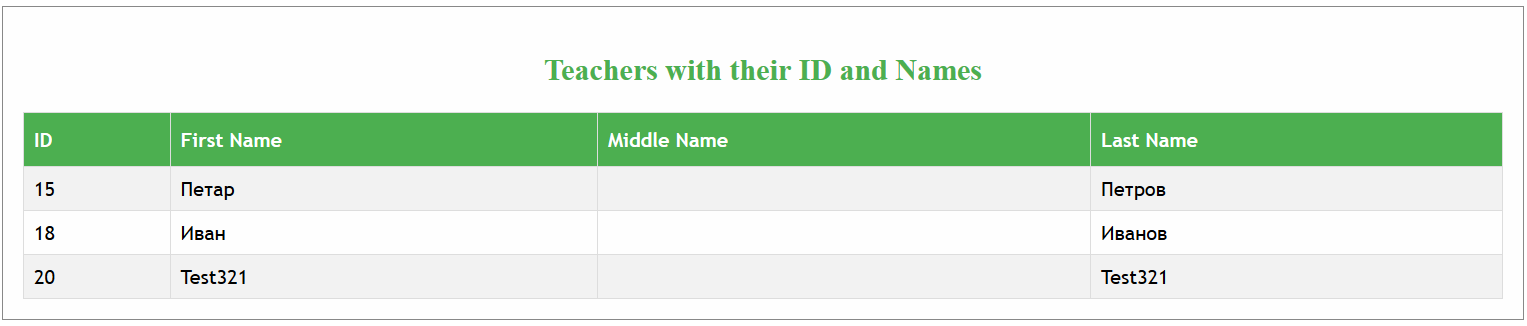
Другия метод, който ще бъде дефиниран е „printUsersInfoTable“. Той ще създаде HTML таблица и запише в нея резултатите от изпълнената SQL заявка. Сървлета, който ще използва тези методи е представен на фигура 54. Той, както и „SearchYourThesisServlet“ ще използва „getAttribute“ и setAttribute методи на Java Servlet, за да прикаже таблицата с резултати на друга уеб страница.

@WebServlet(name = "ShowUsersInfoServlet")  
public class ShowUsersInfoServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 ShowUsersInfoDao suid = new ShowUsersInfoDao();  
 suid.showTeachers();  
 String usersInfoTable = suid.printUsersInfoTable();  
 request.setAttribute("usersInfoTable", usersInfoTable);  
 RequestDispatcher dispatcher = request.getRequestDispatcher("/usersInfo.jsp");  
 dispatcher.forward(request, response);  
 }

}

Фигура 54 – ShowUsersInfoServlet

Резултатите от изпълнение на този сървлет са приказани на фигура 55.



Фигура 55 – Резултати от изпълнение на ShowUsersInfoServlet

Както беше казано, студента ще има възможност да види рецензентите, задължени за неговата дипломна работа. Това ще се случва след натискане на връзка „Show Reviewer“ от таблицата с дипломната работа. При натискане на тази вързка, ще бъде задействан „ShowReviewerServlet“ сървлет, който с помоща на методи дефинирани в помошния клас „ShowUserInfoDao“, ще обработват базата данни и извеждадт резултат на екрана във форма на таблица. Помощен метод, който ще бъде използване представен е на фигура 56.

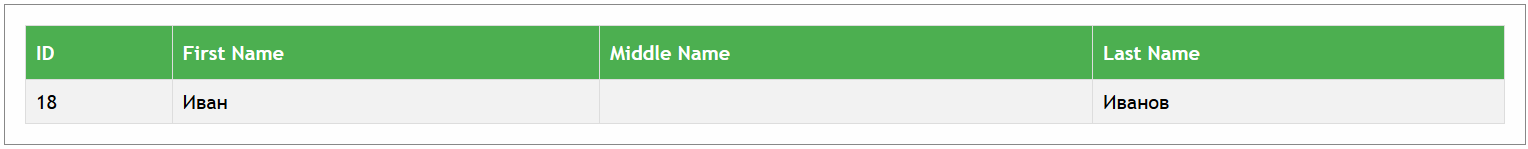
public void showReviewer (int thesisID){  
 String query = "SELECT ID, FirstName, MiddleName, LastName FROM users WHERE NOT Username='admin' AND NOT Username='editor'" +  
 "AND ID IN (SELECT ID FROM recenzent WHERE thesisID=?)";  
 try {  
 ps = conn.prepareStatement(query);  
 ps.setInt(1, thesisID);  
 rs = ps.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();}}

Фигура 56 – Метод „showReviewer“

Този метод ще използва „ID“ на дипломната работа за да провери в таблица с рецензенти има ли съвпадения. Ако има такива метода ще ги извади и запомни в „ResultSet“ обект. След това ще бъде използван предишно дефинирания метод за извеждане на резултата във вид на HTML таблица. Сървлета „ShowReviewerServlet“ (Фигура 57) ще запомня таблицата с метода „setAttribute“ и ще я извежда на уеб страница „reviewerInfoPage“ с помощ на „getAttribute“. Таблицата ще съдържа колони с „ID“, име, бащино име и фамилия (Фигура 58).

@WebServlet(name = "ShowReviewerServlet")  
public class ShowReviewerServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 int thesisID = Integer.*parseInt*(request.getParameter("thesisID"));  
 ShowUsersInfoDao suid = new ShowUsersInfoDao();  
 suid.showReviewer(thesisID);  
 String resultReviewerTable = suid.printUsersInfoTable();  
 request.setAttribute("resultReviewerTable", resultReviewerTable);  
 String url = "/reviewerInfoPage.jsp";  
 RequestDispatcher dispatcher = request.getRequestDispatcher(url);  
 dispatcher.forward(request, response);  
 }

Фигура 57 – ShowReviewerServlet

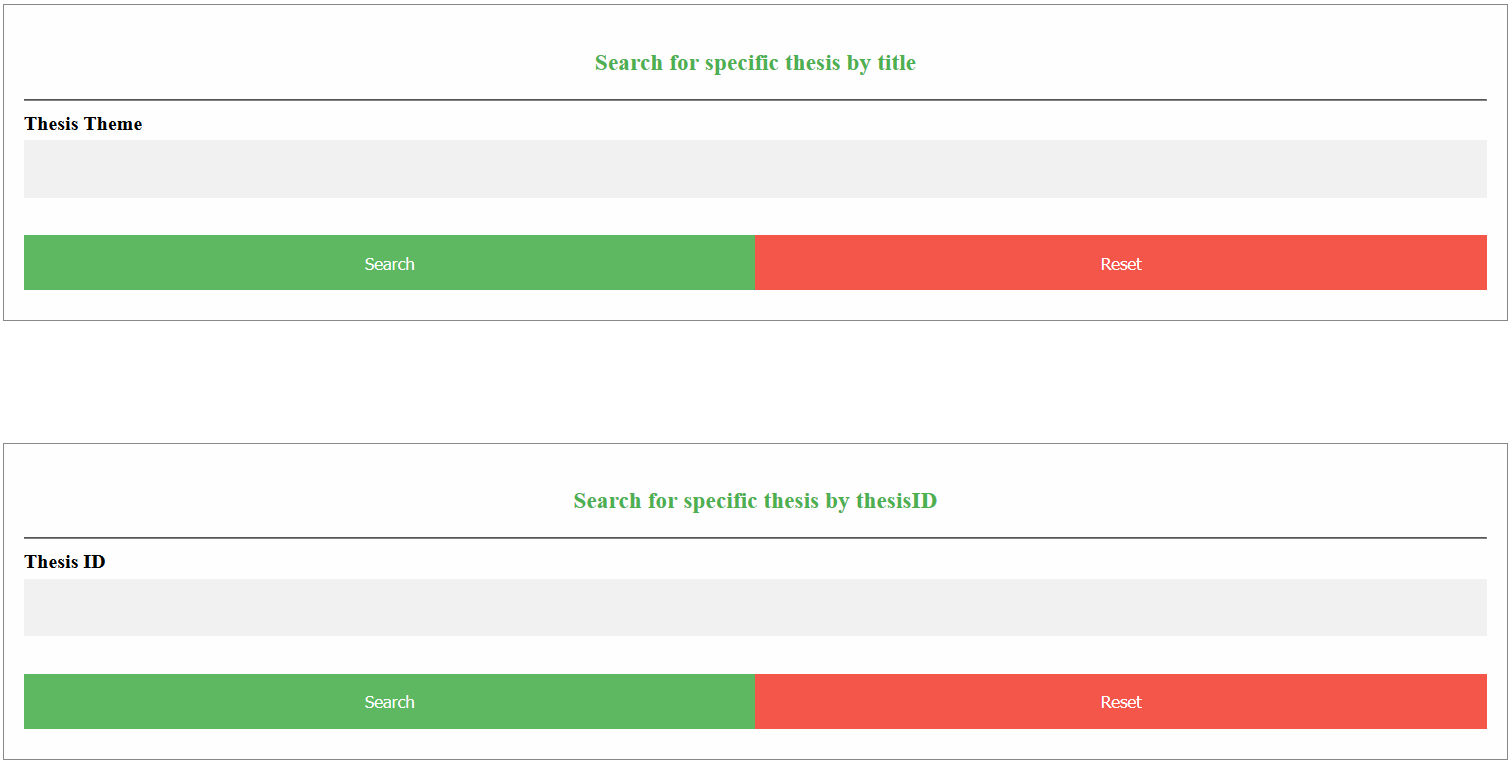


Фигура 58 – Резултат от изпълнение на ShowReviewerServlet

Освен да виждат информация за своята дипломна работа, студентите ще имат възможност и да търсят други дипломни работи. Ще бъдат предоставени две критерии за търсене:

* Търсене чрез заглавие или част от заглавие и
* Търсене чрез ID на дипломната работа.

Изгледа на тези две форми за търсене, представен е на фигура 59.



Фигура 59 – Форми за търсене на дипломи работи

За да работят тези две форми ще използват два различни сървлета „SearchByTitleServlet“ и „SearchByThesisIDServlet“. Те ще бъдат използвани, както от студентите, така и от преподавателите. И двата сървлета ще имат нужда от помощни методи. Помощните методи ще бъдат дефинирани във вече създадения помощен клас „ThesisDao“, тъй като ще се отнасят до обработка на информация свързана с дипломни работи. За търсене по заглавие ще бъде използван помощен метод „searchForThesisByTitle“ (Фигура 60). Този метод за параметър ще използва String, който потребителя въвежда и след това ще изпълнява SQL заявка с която ще се търсят заглавия на дипломни работи от таблица „thesis“ съдържащи внесената дума или съвкупност от думи. Резултата от изпълнение на този метод ще бъде запомнене в „ResultSet“ обект, който след това ще бъде представен като HTML таблица с помощ на вече дефинирания метод „printResultTable“.

public void searchForThesisByTitle(String title) {  
 String query = "SELECT \* FROM thesis WHERE Theme LIKE '%" + title + "%'";  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(query);  
 rs = ps.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
}

Фигура 60 – Метод „searchForThesisByTitle“

На фигура 61 е представен „SearchByTitleServlet“, който ще използва описаните методи за да генерира HTTP отговор за клиента. Задача на този сървлет ще бъде да вземе внесената информация от текстовото поле на формата и да я подаде като параметър на метода „searchForThesisByTitle“. След изпълнение на метода, резултата ще бъде записан като HTML таблица и запомнен в „resultTitleTable“ променлива. След това ще бъде приказан на „resultThesisByTitle“ уеб страница. На фигура 62 представен е резултат от изпълнение на този сървлет.

@WebServlet(name = "SearchByTitleServlet")  
public class SearchByTitleServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String thesisTitle = request.getParameter("thesisTitle");  
 ThesisDao thesisDao = new ThesisDao();  
 thesisDao.searchForThesisByTitle(thesisTitle);  
 String resultTitleTable = thesisDao.printResultTable();  
 request.setAttribute("resultTitleTable", resultTitleTable);  
 RequestDispatcher dispatcher;  
 dispatcher = request.getRequestDispatcher("resultThesis.jsp");  
 dispatcher.forward(request, response);  
 }

Фигура 61 – SearchByTitleServlet



Фигура 62 – Резултат от изпълнение на „SearchByTitleServlet“

За търсене по „ID“ на дипломна работа ще бъде използван друг сървлет и друг помощен метод. Помощният метод „searchForThesisByID“ (Фигура 63) ще използва въведения от потребител „thesisID“ за да претърси таблицата „thesis“. Резултата от изпълнението си ще запомня в „ResultSet“ обект, който след това ще бъде представен във вид на таблица с помощ на „printResultTable“ метод.

public void searchForThesisByID(int thesisID) {  
 String query = "SELECT \* FROM thesis WHERE thesisID="+thesisID;  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(query);  
 rs = ps.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
}

Фигура 63 – Метод „searchForThesisByID“

Сървлета, който ще използва този метод, е представен на фигура 64. Той, както и „SearchByTitleServlet“ ще взима внесения от потребителя „Thesis ID“ и след обработка ще запомня резултата в таблица, която ще бъде представена на „resultThesis“ страница.

@WebServlet(name = "SearchByThesisIDServlet")  
public class SearchByThesisIDServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 int thesisID = Integer.*parseInt*(request.getParameter("thesisID"));  
 ThesisDao thesisDao = new ThesisDao();  
 thesisDao.searchForThesisByID(thesisID);  
 String resultIDTable = thesisDao.printResultTable();  
 request.setAttribute("resultTable", resultIDTable);  
 RequestDispatcher dispatcher;  
 dispatcher = request.getRequestDispatcher("resultThesis.jsp");  
 dispatcher.forward(request, response);  
 }

Фигура 64 – SearchByThesisIDServlet

Крайния резултат от изпълнение на този сървлет е представен на фигура 65. Потребителя при въвеждане на число 21 в текстовото поле и като резултат ще получи дипломна работа със съответния „Thesis ID“.



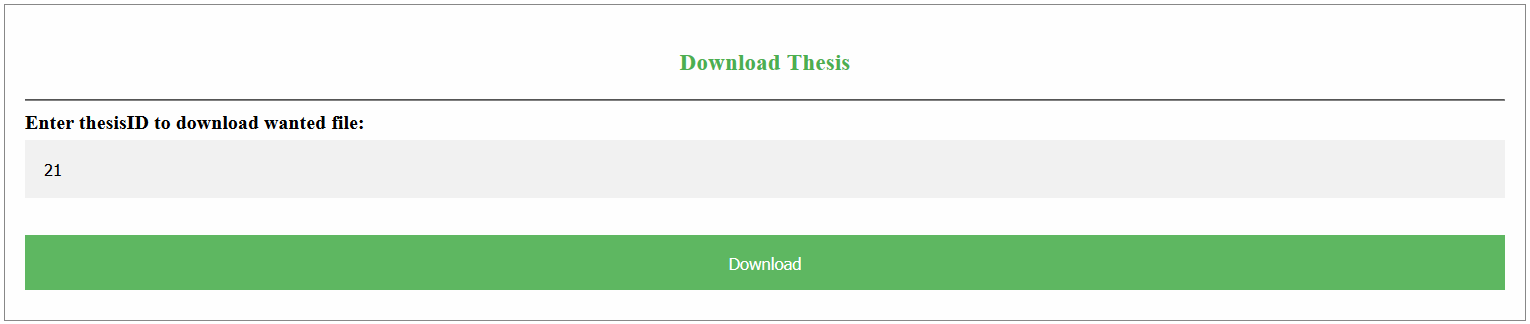
Фигура 65 – Резултат от изпълнение на SearchByThesisIDServlet

Освен да виждат информация за намерените дипломни работи, потребителите ще имат възможност и да ги изтеглят. За да се предостави тази възможност, уеб приложението ще използва „DownloadServlet“ (Фигура 66). Задача на този сървлет ще бъде да вземе въведения от потребител „Thesis ID“ и да го намери в таблица „files“. В таблицата „files“ се намират всички качени файлове, както и съответните ID-та на дипломните работи. Файлове в MySQL се съхраняват като „blob“ (Binary Large Object) тип данни. При получване на HTTP заявка за изтегляне на файл, сървлета първо ще провери дали съществува записан файл за въведения „ThesisID“. В случай че съществува, ще бъде създаден обект тип „Blob“ и в него, с помощ на „getBlob“ метод, ще бъде запазен файла. HTTP протоколс изисква данните да бъдат предавани в контекста на подобните на имейл съобщения, въпреки че най-често информацията не е точно електронно писмо. За преобразуване на файла във формат поддържан от HTTP протокола ще бъде използван MIME стандарт. Съдържанието на файла ще бъде форматирано и ще му бъдат зададени хедър атрибути. Следваща стъпка ще бъде форматирания файл да бъде изпратен на клиента. След като клиента получи файла, ще бъде попитан с къква програма да го отвори. Препоръчително е да това бъде текстов редактор, понеже самия файл представлява текстов документ. На фигура 67 е представена формата за итегляне на дипломни работи.

@WebServlet(name = "DownloadFileServlet")  
public class DownloadFileServlet extends HttpServlet {  
 private static final int *BUFFER\_SIZE* = 4096;  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 int thesisID = Integer.*parseInt*(request.getParameter("thesisID"));  
 Connection conn = DBConnection.*createConnection*();  
 String query = "SELECT \* FROM files WHERE thesisID = ?";  
 PreparedStatement ps = null;  
 try {  
 ps = conn.prepareStatement(query);  
 ps.setInt(1, thesisID);  
 ResultSet rs = ps.executeQuery();  
 if (rs.next()) {  
 int fileName = Integer.*parseInt*(rs.getString("thesisID"));  
 Blob blob = rs.getBlob("file");  
 InputStream inputStream = blob.getBinaryStream();  
 int fileLength = inputStream.available();  
 System.*out*.println("fileLength = " + fileLength);  
 ServletContext context = getServletContext();  
 String mimeType = context.getMimeType(String.*valueOf*(fileName));  
 if (mimeType == null) {  
 mimeType = "application/octet-stream";}  
 response.setContentType(mimeType);  
 response.setContentLength(fileLength);  
 String headerKey = "Content-Disposition";  
 String headerValue = String.*format*("attachment; filename=\"%s\"", fileName);  
 response.setHeader(headerKey, headerValue);  
 OutputStream outStream = response.getOutputStream();  
 byte[] buffer = new byte[*BUFFER\_SIZE*];  
 int bytesRead = -1;  
 while ((bytesRead = inputStream.read(buffer)) != -1) {  
 outStream.write(buffer, 0, bytesRead);}  
 inputStream.close();  
 outStream.close();  
 } else {  
 response.getWriter().print("<h2 style=\"text-align:center; color:#4CAF50\"" +  
 "File NOT found for thesis ID: "+thesisID+"</h2>");}  
 } catch (SQLException e) {  
 e.printStackTrace();}

}

Фигура 66 – DownloadServlet



Фигура 67 – Форма за DownloadFileServlet

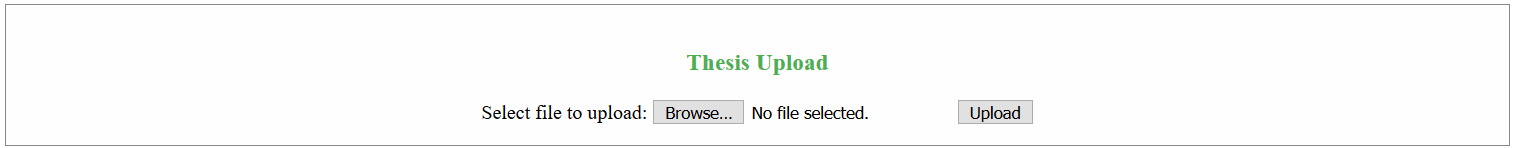
Когато студентите създадът дипломните си работи, те ще имат възможност да ги качват в базата данни, за това ще се погрижи „FileUploadServlet“. Понеже отново ще се работи с дипломни работи, ще бъде използван помощен метод дефиниран в „ThesisDao“ класа. Сървлета ща използва вече дефинирания метод за получване на факултетен номер „getFacultyNumber“. Сървлета ще има за задача от създадената сесия, при самото логване на потребителя, да вземе ID-то му и подаде го като параметър на метода „getFacultyNumber“. Метода след изпълнение ще върне факултетния номер на студента и ще го запомни за по-нататъчн обработка. Следваща стъпка на сървлета ще бъде взимане на зададения текстов документ от потребителя. Ще бъде извършена проверка върху документа за валидността му и след това ще бъде изпълнена SQL заявка. Заявката ще запише документа и необходимата информация в таблицата „files“. При успешно записване на екрана ще бъде изведено съобщение „File successfully uploaded.“. Сървлета „FileUploadServlet“ е представен на фигура 68 с всичките му компоненти. На фигура 69 представена е формата, която потребителя вижда.

WebServlet(name = "FileUploadServlet")

@MultipartConfig (maxFileSize = 16177215) //Upload file up to 16MB  
public class FileUploadServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 HttpSession session = request.getSession(false);  
 int userID = (int) session.getAttribute("id");  
 ThesisDao thesisDao = new ThesisDao();  
 int facNum = thesisDao.getFacultyNumber(userID);  
 int thesisID = thesisDao.showYourThesisID(facNum);  
 request.setAttribute("thesisID", thesisID);  
 Part filePart = request.getPart("thesisFile");  
 InputStream inputStream = null;  
 if (filePart != null) {  
 inputStream = filePart.getInputStream();  
 }  
 String message = null;  
 String insertQuery = "INSERT INTO files (thesisID, file) VALUES (?,?)";  
 Connection conn = null;  
 PreparedStatement ps = null;  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(insertQuery);  
 ps.setInt(1,thesisID);  
 if (inputStream != null){  
 ps.setBlob(2, inputStream);  
 }  
 int row = ps.executeUpdate();  
 if (row > 0){  
 message = "File successfully uploaded.";  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 request.setAttribute("Message", message);  
 getServletContext().getRequestDispatcher("/sHomepage.jsp").forward(request, response);

}

Фигура 68 - FileUploadServlet



Фигура 69 – Thesis Upload форма

Последната възможност, която студентите ще имат е да прекъснат връзка с уеб сървъра, т.е. да излезат от профилите си. Това ще бъде реализирано с „LogoutServlet“ (Фигура 70). Сървлета ще прекъсне сесията и ще препрати потребителя към началната страница на уеб приложението. На потребителя ще бъде изписано съобщение за успешно прекъсване на връзката. Този сървлет ще бъде използван от всички потребители на системата.

@WebServlet(name = "LogoutServlet")  
public class LogoutServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 HttpSession session = request.getSession(false);  
  
 if (session != null){  
 session.invalidate();  
 request.setAttribute("errMessage", "You have successfully logged out.");  
 request.getRequestDispatcher("/index.jsp").forward(request, response);  
 }  
 }

Фигура 70 – LogoutServlet

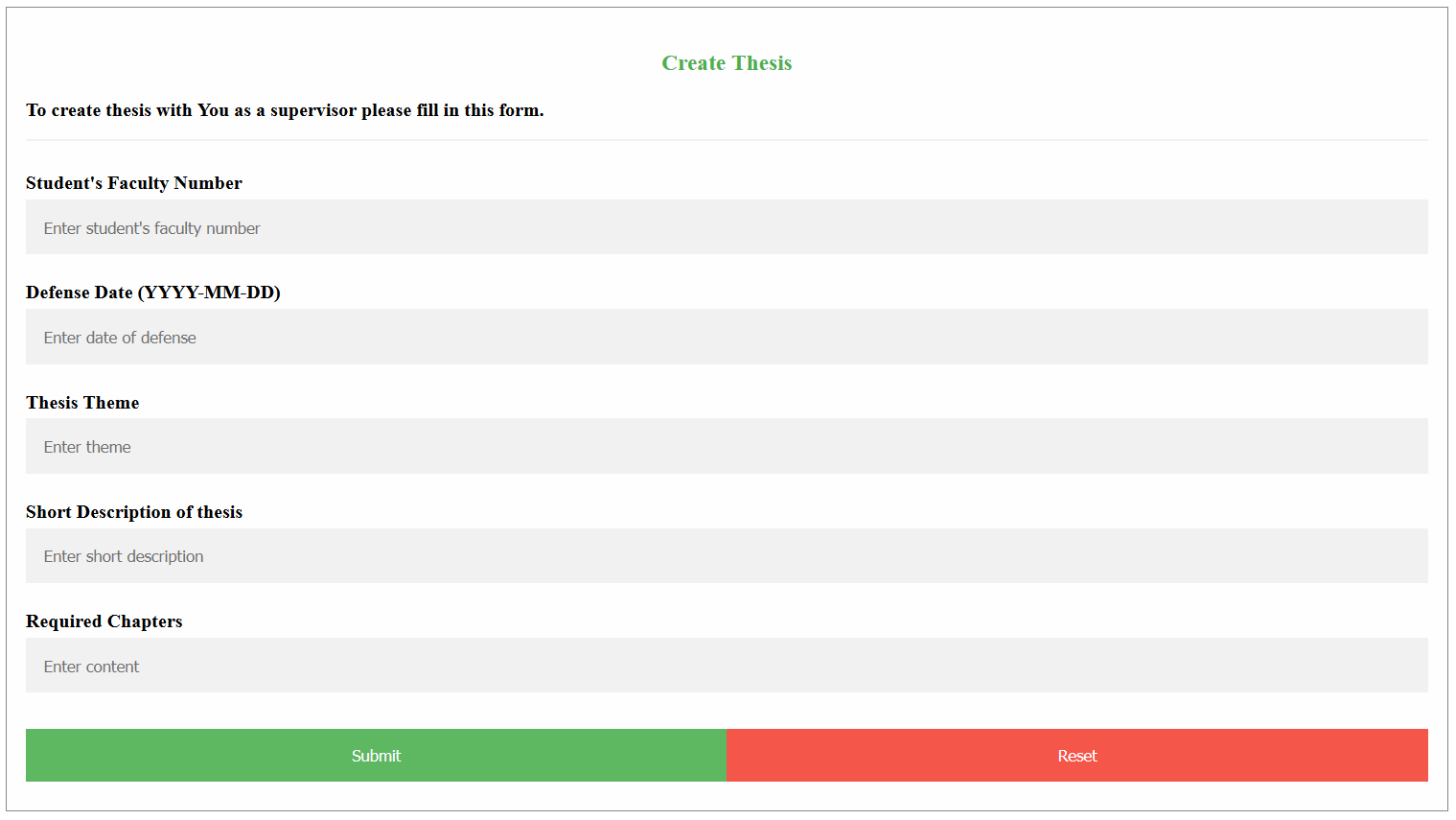
Преподавателите, освен всичките възможности на студентите, ще имат и допълнителна възможност за създаване на задания за дипломни работи. Това ще бъде реализирано посредством „CreateThesisServlet“. Този сървлет за да изпълнява задачата си ще има нужда от допълнителен метод „createThesis“ (Фигура 71). Понеже метода се отнася до дипломни работи, той ще бъде създаден в класа „ThesisDao“. Като параметър метода ще използва обект тип „ThesisBean“, който предоставя всички необходими полета за създаване на едно задание за дипломна работа. Метода от „ThesisBean“ обекта ще взима необходима информация за попълване на таблица „thesis“ чрез SQL заявка. Метода след изпълнението си ще върща съобщение за успешно създаване или проблем. Сървлета „CreateThesisServlet“ (Фигура 72) ще взима попълнената от потребителя информация, във формата за създаване на задание, и ще я обработва. След това ще създаде обект тип „ThesisBean“ и запълни свойствата му с получената информация. Този обект ще бъде използван като параметър на „createThesis“ метода. След като бъде изпълнен, метода ще върне съобщение, което сървлета ще използва за да генерира HTTP отговор. В случай че създаването на заданието е извършено успешно, на потребителя ще бъде приказана таблица с информация за създаденото задание. В противен случай на потребителя ще бъде изведено съобщение за грешка. Формата за попълване, която се предоставя на потребителя е на фигура 73, а резултата от изпълнение на сървлета е представен на фигура 74.

public String createThesis (ThesisBean thesisBean) throws ParseException {  
 int facNum = thesisBean.getFacNum();  
 int id = thesisBean.getTeacherID();  
 String defDate = thesisBean.getDefenseDate();  
 java.sql.Date defenseDate = thesisBean.processDate(defDate);  
 String theme = thesisBean.getTheme();  
 String shortDescription = thesisBean.getDescription();  
 String content = thesisBean.getContent();  
 String query = "INSERT INTO thesis (FacultyNumber, ID, DefenseDate, Theme, ShortDescription, Content) VALUES (?,?,?,?,?,?)";  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(query);  
 ps.setInt(1, facNum);  
 ps.setInt(2, id);  
 ps.setDate(3, defenseDate);  
 ps.setString(4, theme);  
 ps.setString(5, shortDescription);  
 ps.setString(6, content);  
 int i = ps.executeUpdate();  
  
 if (i != 0){  
 return "SUCCESS.";  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return "Something went wrong.";  
}

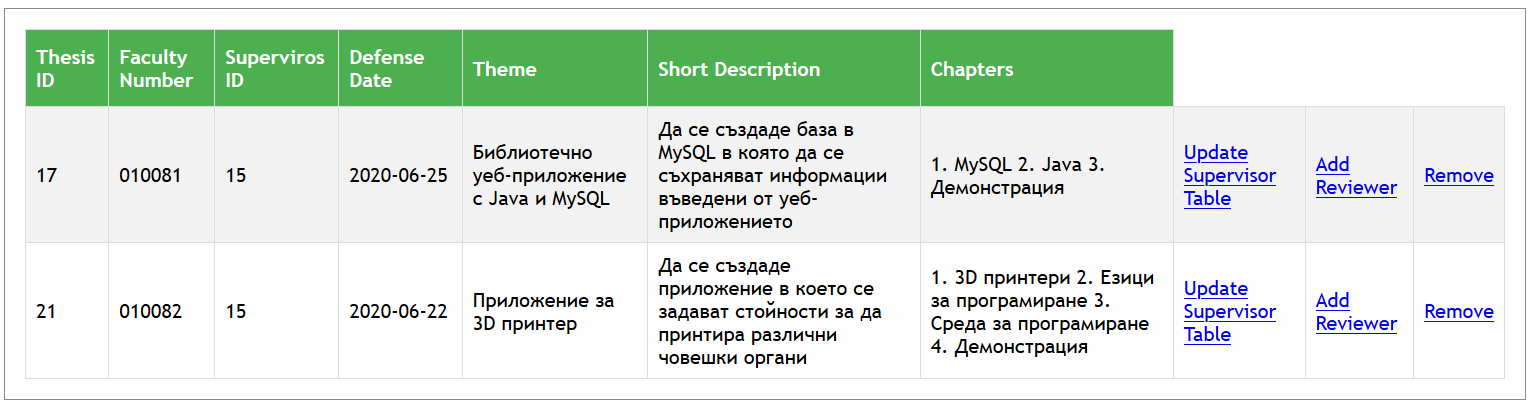
Фигура 71 – Метод „createThesis“

@WebServlet(name = "CreateThesisServlet")  
public class CreateThesisServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 ThesisBean thesisBean = new ThesisBean();  
 //Getting values from fields and saving them to variables.  
 //After that setting values for thesisBean obj.  
 int facultyNumber = Integer.*parseInt*(request.getParameter("facNum"));  
 thesisBean.setFacNum(facultyNumber);  
 HttpSession session = request.getSession(false);  
 int teacherID = (int) session.getAttribute("id");  
 thesisBean.setTeacherID(teacherID);  
 String date = request.getParameter("defDate");  
 thesisBean.setDefenseDate(date);  
 String theme = request.getParameter("theme");  
 thesisBean.setTheme(theme);  
 String shortDescription = request.getParameter("shortDesc");  
 thesisBean.setDescription(shortDescription);  
 String content = request.getParameter("content");  
 thesisBean.setContent(content);  
 ThesisDao thesisDao = new ThesisDao();  
 String thesisCreated = null;  
 try {  
 thesisCreated = thesisDao.createThesis(thesisBean);  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
  
 if (thesisCreated.equals("SUCCESS.")){  
 RequestDispatcher dispatcher = request.getRequestDispatcher("/SupervisorSearchThesesServlet");  
 dispatcher.forward(request, response);  
 } else {  
 request.setAttribute("errMessage", thesisCreated);  
 request.getRequestDispatcher("/tHomepage.jsp").forward(request, response);  
 }  
 }

Фигура 72 – CreateThesisServlet



Фигура 73 – Форма за създаване на задание за дипломна работа



Фигура 74 – Резултат получен след изпълнение на CreateThesisServlet

По подразбиране, когато преподавател създава задание за дипломна работа той самия се взима за ръководител. Една от по-важните възможности на преподавателите е да виждат всички дипломни работи където те са ръководители. Това ще им бъде предоставено с помощ на „SupervisorSearchThesesServlet“. Сървлета ще използва помощни методи дефинирани в „ThesisDao“ класа. Основния метод, който ще бъде използван за получаване на записи от базата данни ще бъде „showSupervisorTheses“ (Фигура 75). Този метод ще използва „ID“ на логнатия преподавател за да претърси „thesis“ таблица и извади всички редове, където „ID“ на преподаватела съвпада. Тези редове ще бъдат записани в „ResultSet“ обект и след това принтирани във вид на таблица, с помощ на метод „printTeachersThesisTable“.

public void showSupervisorTheses(int userID){  
 String query = "SELECT \* FROM thesis WHERE ID='"+userID+"'";  
 conn = DBConnection.*createConnection*();  
 try {  
 ps = conn.prepareStatement(query);  
 rs = ps.executeQuery();  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }}

Фигура 75 – Метод „showSupervisorTheses“

На фигура 76 представен е сървлета „SupervisorSearchThesesServlet“. Задача на този сървлет ще бъде да получените резултатио от изпълнение на метода „showSupervisorThese“ изведе на уеб страница „resultThesis.jsp“ с помощ на „setAttribute“ и „getAttribute“ методи.

@WebServlet(name = "SupervisorSearchThesesServlet")  
public class SupervisorSearchThesesServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 HttpSession session = request.getSession(false);  
 int userID = (int) session.getAttribute("id");  
 ThesisDao thesisDao = new ThesisDao();  
 thesisDao.showSupervisorTheses(userID);  
 String supervisorThesesTable = thesisDao.printTeacherThesisTable();  
 request.setAttribute("resultTable", supervisorThesesTable);  
 RequestDispatcher dispatcher = request.getRequestDispatcher("resultThesis.jsp");  
 dispatcher.forward(request,response);}

Фигура 76 – SupervisorSearchThesesServlet

Много важна задача за ръководителите ще бъде да записват своя „ID“ и „thesisID“ на заданията за дипломни работи, където те са ръководители, в таблица „rukovoditel“. Това ще им позволи по всеко време да могат да проверяват към кои дипломни работи имат задължения. Тази задача ще бъде решена с помощ на „UpdateSupervisorServlet“. Сървлета ща използва помощни методи за изпълнение на задачата си. Затова ще бъде създаден клас „UpdateSupervisorDao“ и в него дефиниран помощен метод „doUpdate“ (Фигура 77). Този метод ще създаде връзка с базата данни и след това ще запълни таблицата „rukovoditel“ с необходимата информация.

public class UpdateSupervisorDao {  
 Connection conn = DBConnection.*createConnection*();  
 public void doUpdate (int thesisID, int rukovoditelID){  
 String query = "INSERT INTO rukovoditel (thesisID, ID) VALUES (?,?)";  
 try {  
 PreparedStatement ps = conn.prepareStatement(query);  
 ps.setInt(1, thesisID);  
 ps.setInt(2, rukovoditelID);  
 ps.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();}}}

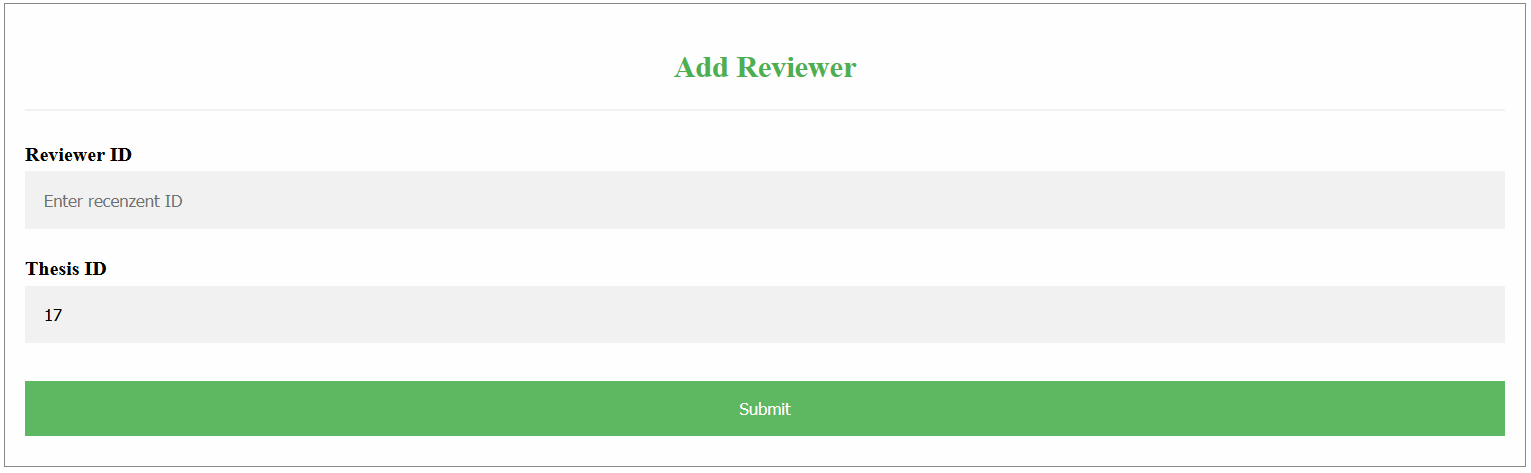
Фигура 77 – Метод „doUpdate“

Сървлета „UpdateSupervisorServlet“ (Фигура 78) ще използва този метод за изпълнение на HTTP заявка. Самият сървлет ще бъде извикан след натискане на бутона „Update Supervisor Table“. Чрез URL адреса ще бъдат взети „Thesis ID“ и „Supervisor ID“, които ще се използват като параметри за метода „doUpdate“. След изпълнение на сървлте текущата страница ще бъде презаредена.

@WebServlet(name = "UpdateSupervisorServlet")  
public class UpdateSupervisorServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 int thesisID = Integer.*parseInt*(request.getParameter("thesisID"));  
 int rukovoditelID = Integer.*parseInt*(request.getParameter("rukovoditelID"));  
 UpdateSupervisorDao usd = new UpdateSupervisorDao();  
 usd.doUpdate(thesisID, rukovoditelID);  
 String url = "/SupervisorSearchThesesServlet";  
 RequestDispatcher dispatcher = request.getRequestDispatcher(url);  
 dispatcher.forward(request, response);

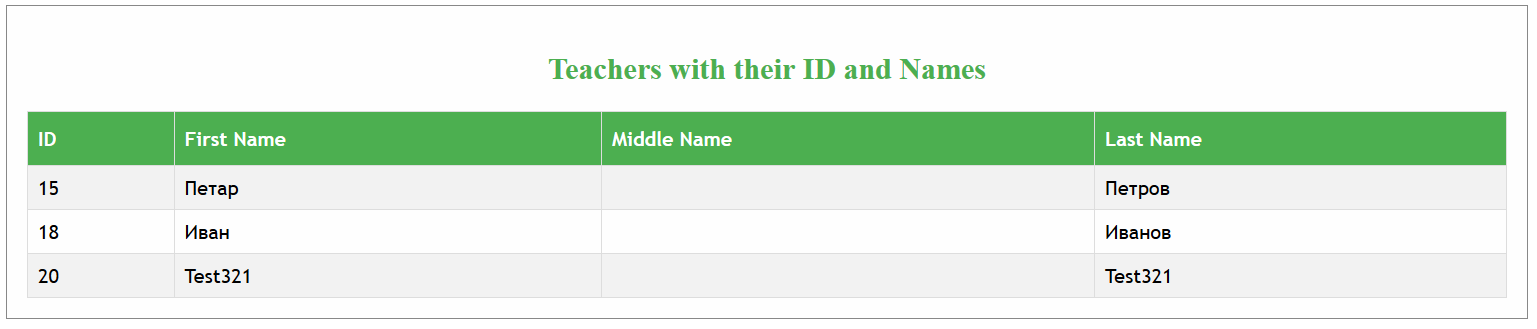
Фигура 78 – UpdateSupervisorServlet

Втора важна задача за ръководителите на дипломните работи ще бъде добавяне на рецензенти, за съответни дипломни работи. Всяка дипломна работа трябва да има поне два рецензента, които студентите ще могат да виждат. За да бъде предоставена тази възможност, ръководителите ще могат да използват „Add Reviewer“ бутон. Този бутон ще задейства „AddReviewerFormServlet“, който ще отваря форма за попълване (Фигура 79).



Фигура 79 – Add Reviewer page

На същата страничка, ръководителите ще имат възможност да видят таблица със всички регистрирани потребители и техните ID-та (Фигура 80).



Фигура 80 – Таблица с информация за преподаватели

Когато формата за добавяне на рецензенти бъде попълнена, с натискане на бутона „Submit“, ще бъде изпълнен „AddReviewerServlet“. Този сървлет ще използва помощен клас „ReviewerBean“, който характеризира рецензенти. За да бъде обработена информацията попълнена от ръководитела, сървлета ща използва и помощен метод „addReviewer“. Този метод ще изпълнява SQL заявка за попълване на „recenzent“ таблица. На фигура 81 представен е сървлета „AddReviewerServlet“. Той след изпълнението си ще върща ръководитела към таблицата с дипломните работи, а резултат от изпълнението му ще могат да видят само студентите чрез отваряне на страница със рецензенти.

@WebServlet(name = "AddReviewerServlet")  
public class AddReviewerServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

int reviewerID = Integer.*parseInt*(request.getParameter("reviewerID"));  
 int thesisID = Integer.*parseInt*(request.getParameter("thesisID"));  
 ReviewerBean reviewerBean = new ReviewerBean();  
 reviewerBean.setThesisID(thesisID);  
 reviewerBean.setReviewerID(reviewerID);  
 ReviewerDao rd = new ReviewerDao();  
 rd.addReviewer(reviewerBean);  
 String url = "/SupervisorSearchThesesServlet";  
 RequestDispatcher dispatcher = request.getRequestDispatcher(url);  
 dispatcher.forward(request, response);  
 }

Фигура 81 – AddReviewerServlet

Последна възможност за преподавателите ще бъде изтриване на задания за дипломни работи. Тази възможност ще бъде предоставена посредством „DeleteThesisServlet“. Понеже ще се работи със дипломни работи, в класа „ThesisDao“ ще бъде дефиниран метод „deleteThesis“ (Фигура 82). Този метод ще изпълнява SQL заявка за изтриване, а като параметръ ще използва „thesisID“.

public void deleteThesis (int thesisID){  
 conn = DBConnection.*createConnection*();  
 String query = "DELETE FROM thesis WHERE thesisID=?";  
 try {  
 ps = conn.prepareStatement(query);  
 ps.setInt(1, thesisID);  
 ps.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
}

Фигура 82 – Метод „deleteThesis“

Сървлета „DeleteThesisServlet“ (Фигура 83) ще бъде извикан с натискане на бутон „Remove“. „ID“ на заданието ще бъде взет от URL адреса. Това „ID“ ще бъде използвано като параметър за метода „deleteThesis“.

@WebServlet(name = "DeleteThesisServlet")  
public class DeleteThesisServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 int thesisID = Integer.*parseInt*(request.getParameter("thesisID"));  
 ThesisDao thesisDao = new ThesisDao();  
 thesisDao.deleteThesis(thesisID);  
 String url = "/SupervisorSearchThesesServlet";  
 RequestDispatcher dispatcher = request.getRequestDispatcher(url);  
 dispatcher.forward(request, response);  
 }

Фигура 83 – DeleteThesisServlet

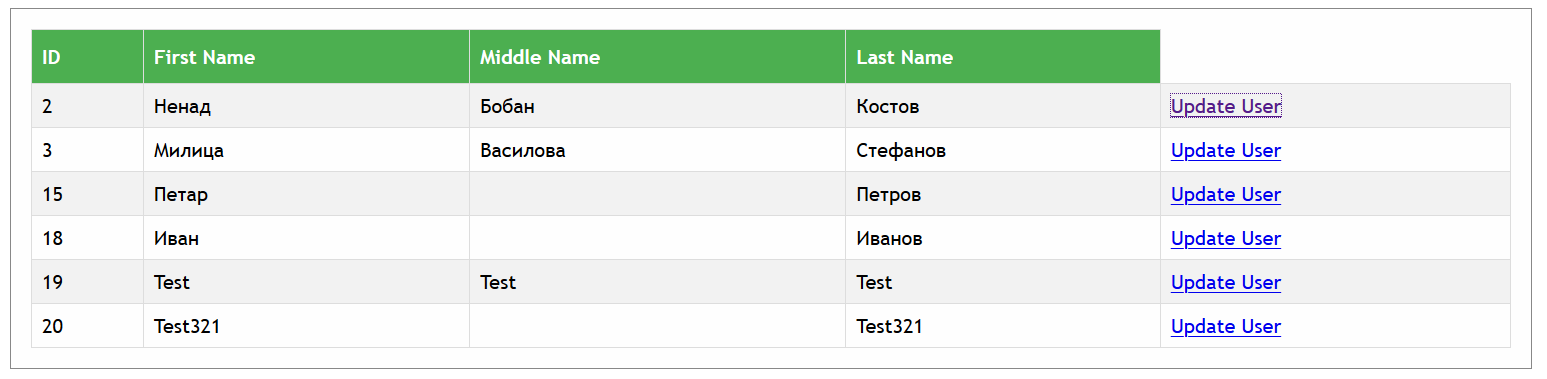
Едитор в системта е отделен потребител, който ще има права да изменя информация свързана с потребителите и дипломните работи. До едитора ще се обърщат потребители, които имат нужда за смяна на имената си или смяна на факултетни номера в заданията за дипломни работи.

Началната страница на едитора ще съдържа две форми, една за модифициране на потребителите и друга за модифициране на заданията за дипломни работи (Фигура 84).



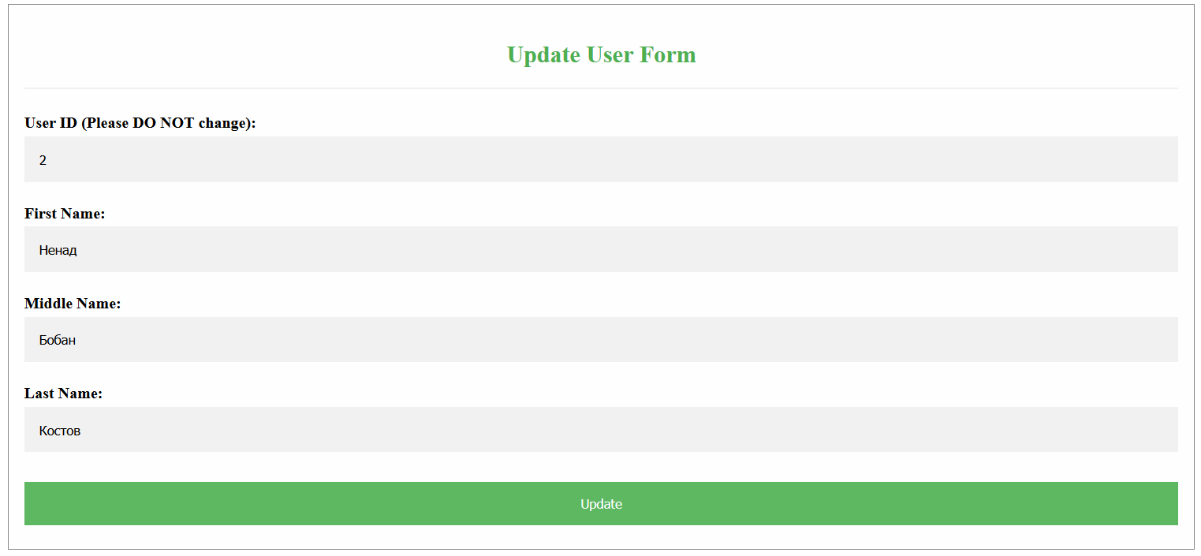
Фигура 84 – Начална страница на едитор

От първата форма едитора ще има възможност да отвори информация за всичките потребители на системата и след това чрез бутона „Update User“ ще може да променя имената на потребителите (Фигура 85).



Фигура 85 – Таблица с информация за потребителите на системата

За изпълнение на тази задача ще бъде използван сървлет „EditorShowUserInfoServlet“. С помощен метод „showUsersInfo“ ще бъде изпълнена SQL заявка за извличане на данни от таблица „users“. Процеса за извеждане на информация в табличен вид е един и същия, като и при предишните методи. При натискане на бутона „Update User“, едитора ще задейства сървлет, който ще взима вече запазената информация на потребителя, ще отвори форма за попълване и в нея ще сложи запазената информация. Формата за попълване представена е на фигура 86.



Фигура 86 – Форма за модифициране на информация на потребителите

Сървлета, който ще изпълнява всички тези функции представен е на фигура 87.

@WebServlet(name = "EditorUpdateUserFormServlet")  
public class EditorUpdateUserFormServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 //Get userID  
 int userID = Integer.*parseInt*(request.getParameter("ID"));  
 //Use the userID and EditorDao obj to get the user data  
 EditorDao editorDao = new EditorDao();  
 UserBean userBean = editorDao.editorDoRead(userID);  
 request.setAttribute("user", userBean);  
 //Redirect to editorUpdateUserForm  
 String url = "/editorUpdateUserForm.jsp";  
 RequestDispatcher dispatcher = request.getRequestDispatcher(url);  
 dispatcher.forward(request, response);  
 }

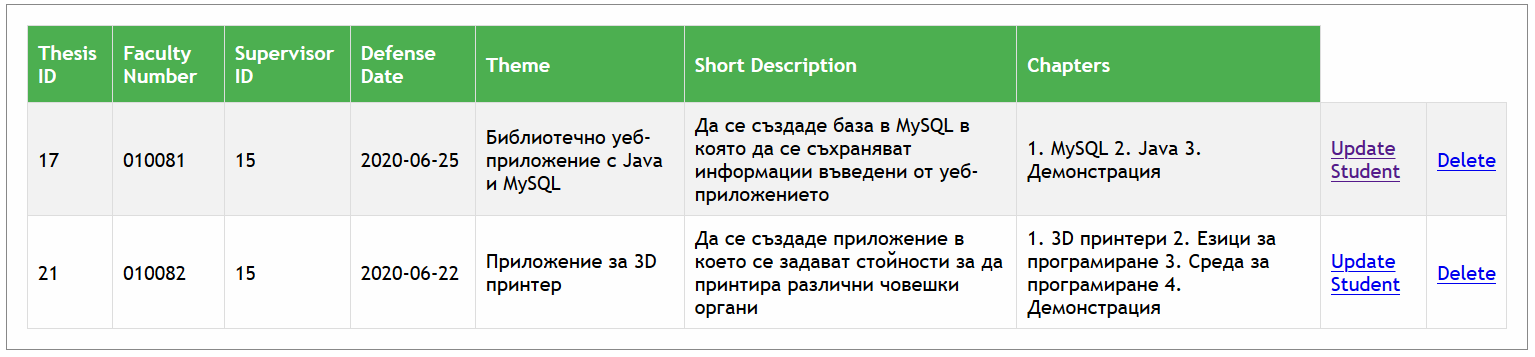
Фигура 87 – EditorUpdateUserFormServlet

След модифициране на информацията, едитора чрез натискане на бутона „Update“ извиква „EditorUpdateUserServlet“ (Фигура 88). Този сървлет ще взима въведената информация от едитора и ще коригира записите в таблицата „users“.

@WebServlet(name = "EditorUpdateUserServlet")  
public class EditorUpdateUserServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 int userID = Integer.*parseInt*(request.getParameter("uid"));  
 String firstName = request.getParameter("fName");  
 String middleName = request.getParameter("mName");  
 String lastName = request.getParameter("lName");  
 UserBean userBean = new UserBean();  
 userBean.setID(userID);  
 userBean.setFirstName(firstName);  
 userBean.setMiddleName(middleName);  
 userBean.setLastName(lastName);  
 EditorDao editorDao = new EditorDao();  
 editorDao.editorDoUpdate(userBean);  
 String url = "/editorShowUsersInfo";  
 RequestDispatcher dispatcher = request.getRequestDispatcher(url);  
 dispatcher.forward(request, response);

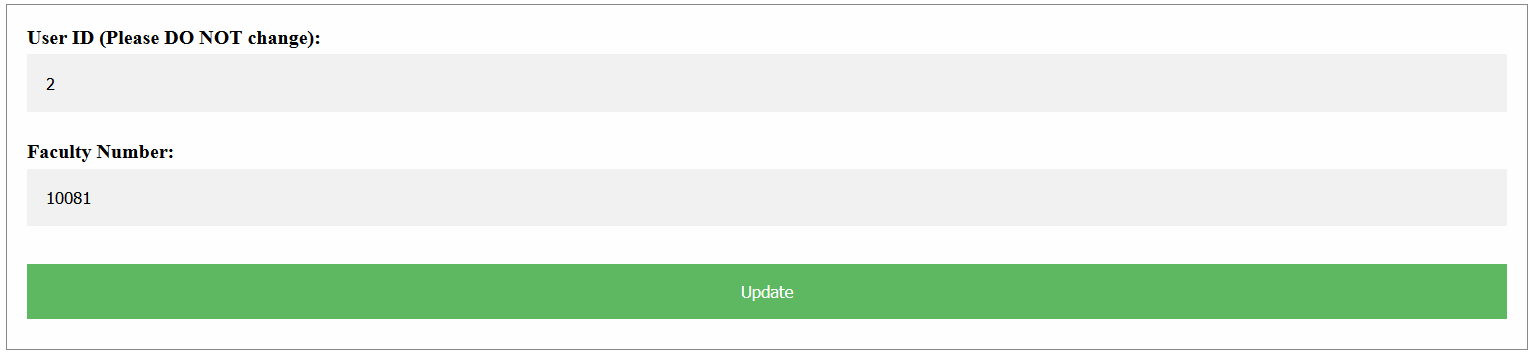
Фигура 88 – EditorUpdateUserServlet

Едитора ще има възможност да модифицира и факултетен номер на студент за определено задание за дипломна работа. Първа стъпка при изпълнение на тази задача е да се получи списък със всички задания за дипломни работи и след това с бутон „Update Student“ ще бъде извършена модификацията. За да се отвори таблицата със всички задания за дипломни работи ще трябва да се натисне бутон „Show“ във формата „Show all theses and their information“. Това ще извика „EditorShowAllThesesServlet“, който ше се изпълни и покаже таблицата със заданията (Фигура 89).



Фигура 89 – Резултат от изпълнение на EditorShowAllThesesServlet

От тази таблица, чрез натискане на бутон „Update Student“, едитора ще бъде препратен към форма за модификация на факултетен номер за съответно задание(Фигура 90). Тази функция може да се използва за отстраняване на случайни грешки по време на създаване на задания за дипломни работи.



Фигура 90 – Форма за модификация на факултен номер

При натискане на бутона „Update“, ще бъде изпълнене сървлет „EditorUpdateStudentServlet“. Този сървлет ще използва помощни методи за обработка на въведената информация от едитора. След изпълнението си, сървлета, ще изпрати едитора към таблицата със задания за дипломни работи.

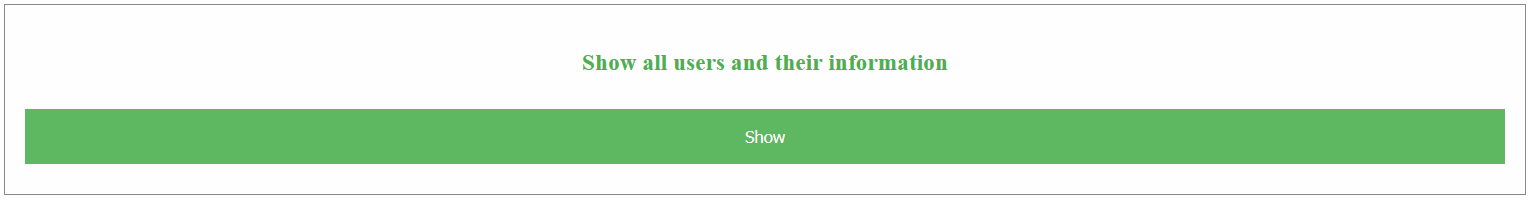
Едитора ще има възможност и да изтрива задания за дипломни работи, но тази възможност ще използва код, който вече беше създаден за ръководителите. За да изтрие някое задание, едитора, ще трябва да натисне бутона „Delete“. Тази функция ще изтрие дори и файла от „files“ таблицата, със съответния „ThesisID“.

Администратора на системата ще има възможност да модифицира всички информации на потребителите. Към него ще могат да се обърнат потребителите които иската да:

* Модифицират имената си,
* Модифицират потребителското си име,
* Модифицират паролата си или
* Бъдат изтрити от системата.

Администратора ще има възможност и да сменя рольите на потребителите. Всичките потребители ще имат роля „Guest“ след регистриране на профила си в системата. Администратора ще може да смени ролите им на „Full User“ и с това да им бъдат одбрени за използване некои нови функции, които могат да бъдат добавени по предложение.

На фигура 91 представена е началната страница на администратора. С бутона „Show“ ще бъде отворена таблица съдържаща информация за всичките потребители на системата.

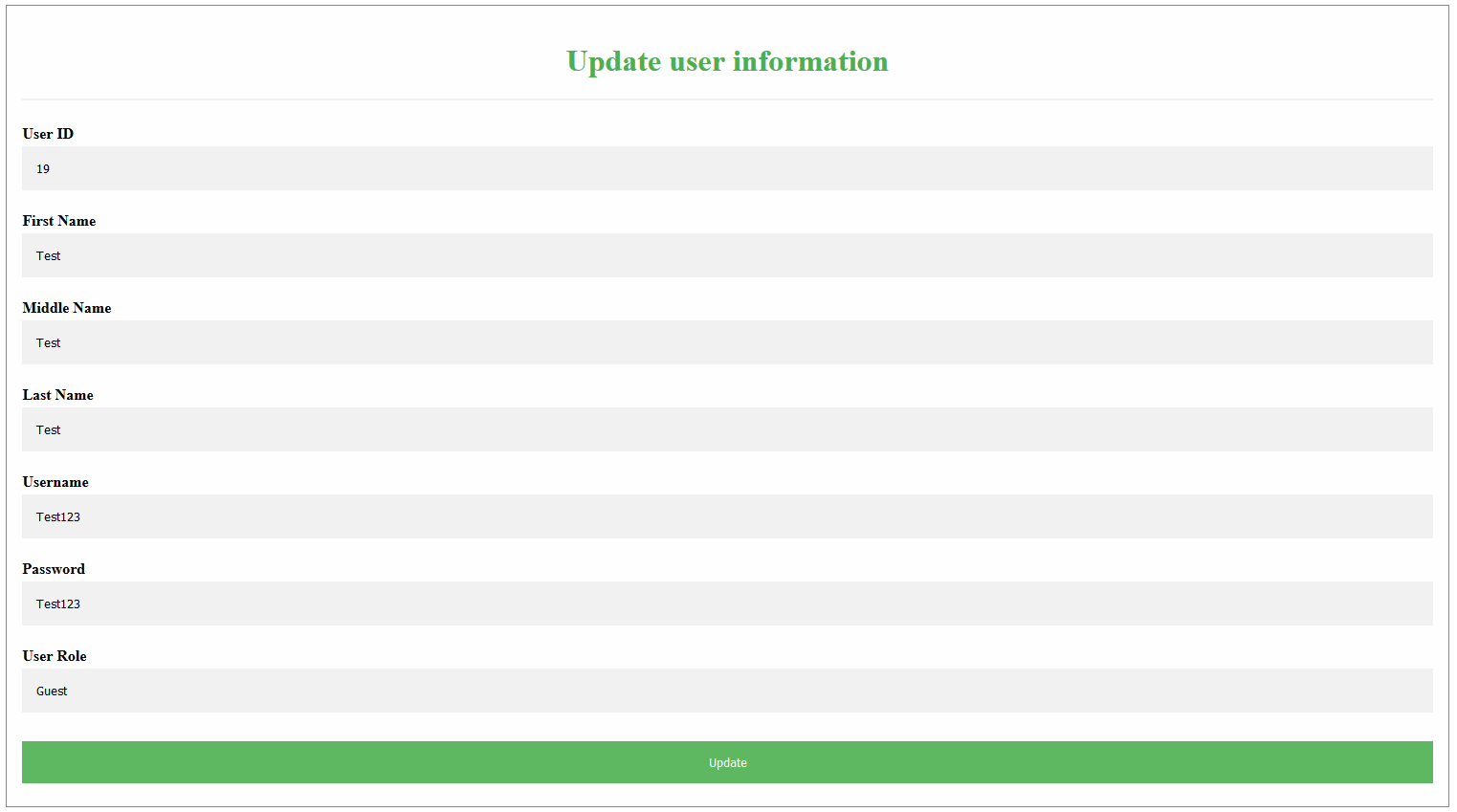


Фигура 91 – Бутон за отваряне на таблица съдържаща информация за всички потребители

От таблицата с информация за потребителите може да се избере „Update“, функция, която ще дава възможност за модифициране на информация на избрания потребител. При натискане на бутона „Update“, администратора ще бъде изпратен към страница със форма за попълване. От там ще може да се модифицира:

* Име
* Бащино име
* Фамилия
* Потребителско име
* Парола и
* Роля

На фигура 92 представена е формата, която ще вижда администратора.



Фигура 92 – Update User форма за администратор

След натискане на бутона „Update“, ще се изпълни „AdminUpdateUserServlet“, който ще вземе внесената от администратора информация и ще направи модификация в „users“ таблицата.

Другата възможност, която има администратора е да изтрие потребител. Това ще бъде изпълнено след натискане на бутона „Delete“. За изпълнение на тази функция ще бъде задължен „AdminDeleteServlet“. Той ще взима „ID“ на потребителя и ще го използва като параметър за метода „doDelete“. След изпълнението си, сървлета ще презареди страницата.

## **Заключение**

Целта на дипломната работа беше да се покажат възможностите на уеб приложенията. Уеб приложенията са навсякъде. Те дават много възможности, които статични уеб страници не би могли. Създаването на едно уеб приложение е сложен процес, който изисква много планиране и тестване. Добре оптимизирано уеб приложение трябва да бъде обмислено преди да бъде кодирано. В глава 1. беше обърнато вмнимани към самите уеб приложения, за да се запознае читатела със структурата им, предностите и недостатъците им. В глава 2. бяха описани всички необходими технологии за създаване на едно уеб приложение. Като модел за реализиране на базата данни в тази дипломна работа беше подходящо да се използва релационнен модел. За реализиране на този модел беше използвана безплатната версия на MySQL. Тя може да се свали от официалната страница на MySQL, https://www.mysql.com, където се предлага за различни операционни системи. MySQL дава възможност за работа в графичен (MySQL Workbench) и команден (MySQL Shell) интерфейс, както и възможност за графчно представяване на базите данни. MySQL предлага набор от функции, инструменти за управление и техническа поддържка на бази данни. Освен това MySQL има вградени конектори за различни програмни езици, между които е и Java. Като език за програмиране беше използвана Java, поради архитектурната неутралност, която дава възможност приложенията написани в Java да се компилират и на Windows и на Linux операционни системи. За да можем да компилираме и създаваме програми на Java, имаме нужда от инсталирана Java платформа. За създаване на нашето уеб проложение беше използвана Java Enterprise Еdition. При инсталиране на Java EE на персоналния компютър, трябва да се избере подходящата дистрибуция. Съществуват две основни дистрибуции:

* Java Development Kit (JDK) и
* Java Runtime Environment (JRE).

JDK включва в себе си както виртуална машина, така и Java компилатор, а също и множество помощни инструменти, докато JRE включва в себе си единствено виртуална машина. За удобство при писане на кода беше използвана програмна среда. Обикновено програмите са съставени от много файлове, понякога дори десетки хиляди. IntelliJ IDEA беше използван за като програмна среда при създаване на това уеб приложение. За комуникация между разработваното уеб приложение и потребителя беше използван Apache Tomcat. В глава 3. представен е самия процес на създаване на уеб приложението, от проектиране на базата данни, преко кодиране до краен работещ продукт.

Приложението може лесно да бъде модифицирано с цел добавяне на нови функции и обогатяване на потребителския интерфейс.

# Използвани източници

[1] Web Aplikacije, http://download.tutoriali.org/Tutorials/Web\_Aplikacije/web\_aplikacije .pdf, последно видян на 19.04.2020г.

[2] Наков, С. и др. (2005). Програмиране за .net Framework – том 1: Достъп до данни с ADO.NET. София. Фабер.

[3] Two-tier Vs Three-tier Architecture, https://medium.com/@gacheruevans0/2-tier-vs-3-tier-architecture-26db56fe7e9c, последно видян на 20.04.2020г.

[4] Simdim софтуерни разработки, https://www.simdim.com/services/web-app, последно видян на 20.04.2020г.

[5] Internet Tehnologije, http://predmet.singidunum.ac.rs/pluginfile.php/5569/mod\_folder /content/0/Literatura%20%28e-test%29/Praktikum%20-%20Internet%20i%20Web%20tehno logije% 20%28Uvod%29.pdf?forcedownload=1, последно видян на 14.01.2020г.

[6] Kratak uvod u World Wide Web i Web aplikacije, https://programirajmozajedno.wordpress .com/2015/01/09/kratak-uvod-u-world-wide-web-i-web-aplikacije/, последно видян на 14.01.2020г.

[7] Google Drive, https://www.google.com/drive/, последно видян на 14.01.2020г.

[8] Office 365, https://www.office.com/?omkt=en-US, последно видян на 14.01.2020г.

[9] Помагало, http://www.pomagalo.com/, последно видян на 14.01.2020г.

[10] Бази данни, http://www-it.fmi.uni-sofia.bg/courses/WDB/db.htm, последно видян на 14.01.2020г.

[11] Информатика II – Бази от данни и СУБД, https://ff.tu-sofia.bg/~bogi/StopFak/Bazi Danni/BaziDanni1.pdf, последно видян на 14.01.2020г.

[12] Microsoft SQL Server, https://www.microsoft.com/en-us/sql-server/sql-server-downloads, последно видян на 14.01.2020г.

[13] MySQL, https://www.mysql.com/, последно видян на 14.01.2020г.

[14] WEB DESIGN от KWB1, https://www.kwb1.com/bgкакво-е-java.html, последно видян на 22.04.2020г.

[15] Наков, С. и др. (2008). Въведение в програмирането с Java: Въведение в програмирането. София. Фабер.

[16] Кацаров, Г. Кои са отличителните черти на програмния език Java , https://softuni.bg/blog/java-basic-characteristics#, последно видян на 21.04.2020г.

[17] IntelliJ IDEA, https://www.jetbrains.com/idea/, последно видян на 21.04.2020г.

[18] Какво е уеб-сървър, https://www.icn.bg/bg/help/drugi/kakvo-e-ueb-srvr/, последно видян на 21.04.2020г.

[19] JavaTpoint: Servlets, https://www.javatpoint.com/servlet-tutorial, последно видян на 21.04.2020г.

[20] Разлики между Apache и Tomcat Server, https://bg.mort-sure.com/blog/ difference-between-apache-and-tomcat-server/, последно видян на 21.04.2020г.