

**I.T.U.**

**Faculty of Computer and Informatics**

**Computer Engineering**



**DATABASE MANAGAMENT SYSTEMS**

**E-LIBRARY PROJECT REPORT**

# CONTENTS

<b>CONTENTS</b>	<b>1</b>
<b>PART I</b>	<b>5</b>
<b>1 GENAREL INFORMATION</b>	<b>5</b>
1.1 Project Description	6
1.2 Development and Execution Environment	6
1.3 Tasks of Group Members	8
1.3.1 Mustafa Uçar	8
1.3.2 Tugrul Yatağan	8
1.3.3 Emre Gökrem	9
1.3.4 Hüseyin Erdoğan	9
<b>PART II</b>	<b>10</b>
<b>2 SETUP</b>	<b>10</b>
2.1 Connection to MySQL Dataabase Server	11
2.2 Database Server Installation and Setup	11
<b>PART III</b>	<b>13</b>
<b>3 USER MANUAL</b>	<b>13</b>
3.1 Guest Interface	14
3.1.1 Guest Home Page	14
3.1.2 Guest Search Procedure	14
3.1.3 Most Popular Sources	17
3.1.4 Request & Suggestions Page	19
3.1.5 Contact Us Page	19
3.1.6 About Us Page	20
3.2 User Interface	21
3.2.1 Register	21
3.2.2 Login	21
3.2.3 Logout	22
3.2.4 User Home Page	22

3.2.5	User Searching Procedure	23
3.2.6	Most Popular Page	25
3.2.7	User Profile	25
3.2.8	User Edit Profile	26
3.2.9	User Archive	28
<b>3.3</b>	<b>Admin Interface</b>	<b>28</b>
3.3.1	Admin Home Page	28
3.3.2	Admin Profile	28
3.3.3	Admin Edit Profile	29
3.3.4	Admin Search Procedure	30
3.3.4.1	Source Placed Page	31
3.3.4.2	Source's Records	31
3.3.4.3	Source's Records Edit	32
3.3.4.4	Source Edit	32
3.3.4.5	Source Delete	34
3.3.5	Adding Source	34
3.3.5.1	Adding Book	35
3.3.5.2	Adding DVD	36
3.3.5.3	Adding Magazine	36
3.3.6	Admin Archive	37
<b>PART IV</b>		<b>38</b>
<b>4</b>	<b>TECHNICAL MANUAL</b>	<b>38</b>
<b>4.1</b>	<b>Database Design</b>	<b>39</b>
4.1.1	Tables	39
4.1.1.1	"users" Table	39
4.1.1.2	"logs" Table	39
4.1.1.3	"authors" Table	39
4.1.1.4	"books" Table	40
4.1.1.5	"magazines" Table	40
4.1.1.6	"DVDs" Table	40
4.1.1.7	"book_categories" Table	41
4.1.1.8	"bookcases" Table	41
4.1.1.9	"bookshelves" Table	41
4.1.1.10	"DVD_categories" Table	42
4.1.1.11	"electronic_resources" Table	42
4.1.1.12	"physical_resources" Table	42
4.1.1.13	"waiting_resources" Table	42
4.1.1.14	"floors" Table	43
4.1.1.15	"libraries" Table	43
4.1.1.16	"magazine_categories" Table	43
4.1.1.17	"records" Table	43
4.1.1.18	"suggestions" Table	44
4.1.1.19	"types" Table	44
4.1.2	Views	44
4.1.2.1	"book_information" View	44

4.1.2.2	“DVD_information” View	45
4.1.2.3	“electronic_books” View	45
4.1.2.4	“electronic_magazines” View	45
4.1.2.5	“magazine_informations” View	45
4.1.2.6	“physical_magazines” View	46
4.1.2.7	“electronic_magazines” View	46
4.1.2.8	“physical_books” View	47
4.1.2.9	“physical_magazines” View	47
4.1.3	Foreign Keys	47
4.1.3.1	In “bookscases” table	47
4.1.3.2	In “bookshelves” table	48
4.1.3.3	In “electronic_resources” table	48
4.1.3.4	In “floors” table	48
4.1.3.5	In “physical_resources” table	48
4.1.4	Entity Relationship Diagram	48
<b>4.2</b>	<b>Software Design</b>	<b>49</b>
4.2.1	Tugrul Yatagan’s classes	49
4.2.1.1	AddSourcePage.java	49
4.2.1.2	Common.java	49
4.2.1.3	DatabaseConnection.java	50
4.2.1.4	EditAdminProfilePage.java	55
4.2.1.5	EditProfilePage.java	55
4.2.1.6	UserEditProfileForm.java	56
4.2.1.7	SourcePlacedPage.java	57
4.2.1.8	SourcePlacedForm.java	58
4.2.1.9	SourceEditPage.java	62
4.2.1.10	SourceEditForm.java	62
4.2.1.11	Start.java	63
4.2.1.12	WicketApplication.java	63
4.2.2	Emre Gökrem’s classes	64
4.2.2.1	AddBookPage.java	64
4.2.2.2	AddBookForm.java	64
4.2.2.3	AdminNavigationPanel.java	65
4.2.2.4	ArchivePage.java	65
4.2.2.5	BasePage.java	66
4.2.2.6	BookEditPage.java	66
4.2.2.7	BookEditForm.java	67
4.2.2.8	Book.java	68
4.2.2.9	BookPage.java	68
4.2.2.10	GuestNavigationPanel.java	70
4.2.2.11	HomePage.java	70
4.2.2.12	NavigationPanel.java	70
4.2.2.13	PopularSourcesPage.java	70
4.2.2.14	RequestsSuggestionsPage.java	71
4.2.2.15	ShowSuggestionsPage.java	71
4.2.2.16	UserNavigationPanel.java	72
4.2.3	Mustafa Uçar’s classes	72
4.2.3.1	AddMagazineForm.java	72
4.2.3.2	AddMagazinePage.java	73

4.2.3.3	BasicAuthenticationSession.java	73
4.2.3.4	MagazineEditForm.java	74
4.2.3.5	MagazineEditPage.java	74
4.2.3.6	Magazine.java	74
4.2.3.7	MagazinePage.java	75
4.2.3.8	register.java	76
4.2.3.9	registerPage.java	77
4.2.3.10	searchForm.java	77
4.2.3.11	searchPage.java	77
4.2.3.12	SignInPage.java	79
4.2.3.13	UserControl.java	79
4.2.3.14	User.java	80
4.2.3.15	UserLogin.java	80
4.2.4	Hüseyin Erdoğan's classes	80
4.2.4.1	AboutUsPage.java	80
4.2.4.2	ContactPage.java	80
4.2.4.3	AdminProfilePage.java	80
4.2.4.4	UserProfilePage.java	81
4.2.4.5	UserPage.java	83
4.2.4.6	Dvd.java	83
4.2.4.7	DvdPage.java	84
4.2.4.8	AddDVDPage.java	85
4.2.4.9	AddDVDForm.java	86
4.2.4.10	DVDEditPage.java	86
4.2.4.11	DVDEditForm.java	86

# **PART I**

## **1 Genarel Information**

## 1.1 Project Description

The aim of this project is providing a whole library system which is accessible via web. Users can view sources (book, e-book, DVD, magazine etc.) and if users are member of the system, they can simply rent sources. Users can find the location of the sources from website and they can take it actual material from library. If a user wants to use electronic sources from website, he/she can simply access information about electronic source and a link of the electronic source. When a user takes a source from library, user will have 30 days to bring it back.

The system have public, regular and administrator users. Administrator users should maintain and control the system (adding source, deleting source, updating source). The system will have different search types (type of source, author of source, category of source etc.). In addition, 10 most popular source listed on the website according to popular search topics and popular rented sources. If a user does not enter the system for a year, account of the user will be deleted by the system.

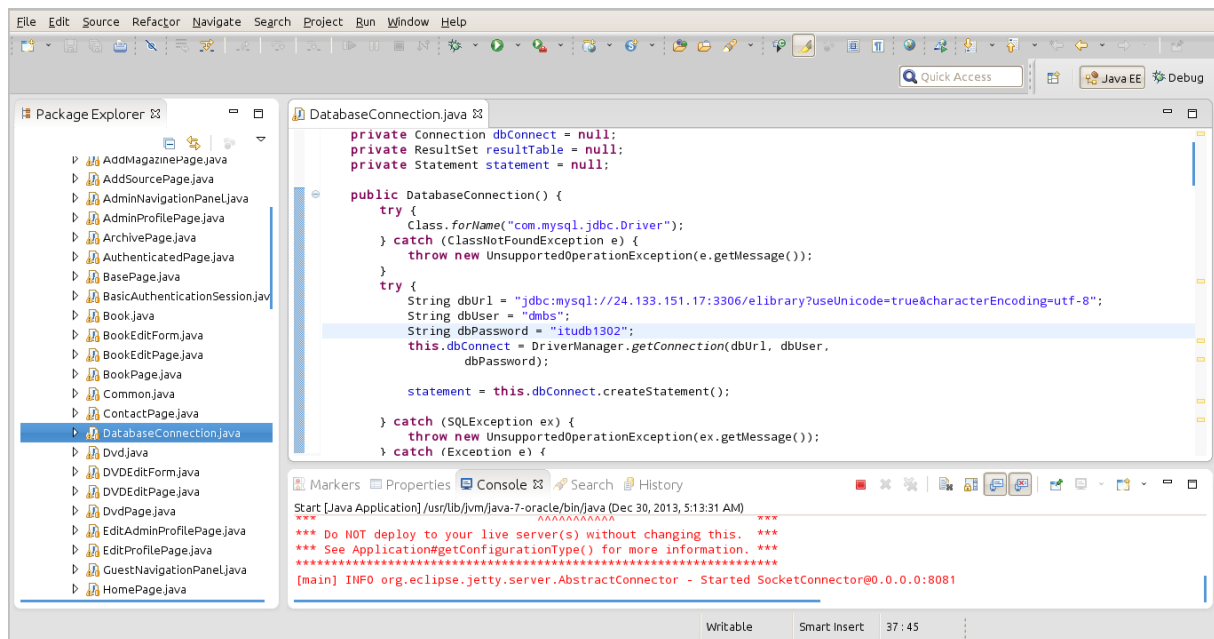
Admin users are the main responsible for continuity of the system, only admins can add source and records, edit source and records. Admin users can see renting process and requests&suggestions. Regular user and public user can give feedback via requests&suggestions page. Public users can only view and search sources but they cannot rent sources.

Everybody can register and be a regularly member. Regular member can view, search and rent sources. If source is not available regular member can add source to their waiting lists. Admin users are responsible for placing materials to library. For these reason admin should enter source information like name, year, category, etc. and then admin should place this source to library. There can be multiple materials of one source in library. If admins wants to place material of existing source they can place source directly to library without entering source information like name, year, category, etc. They only enter place information like bookcase, bookshelf, floor, library, etc. If place is taken a warning message will display.

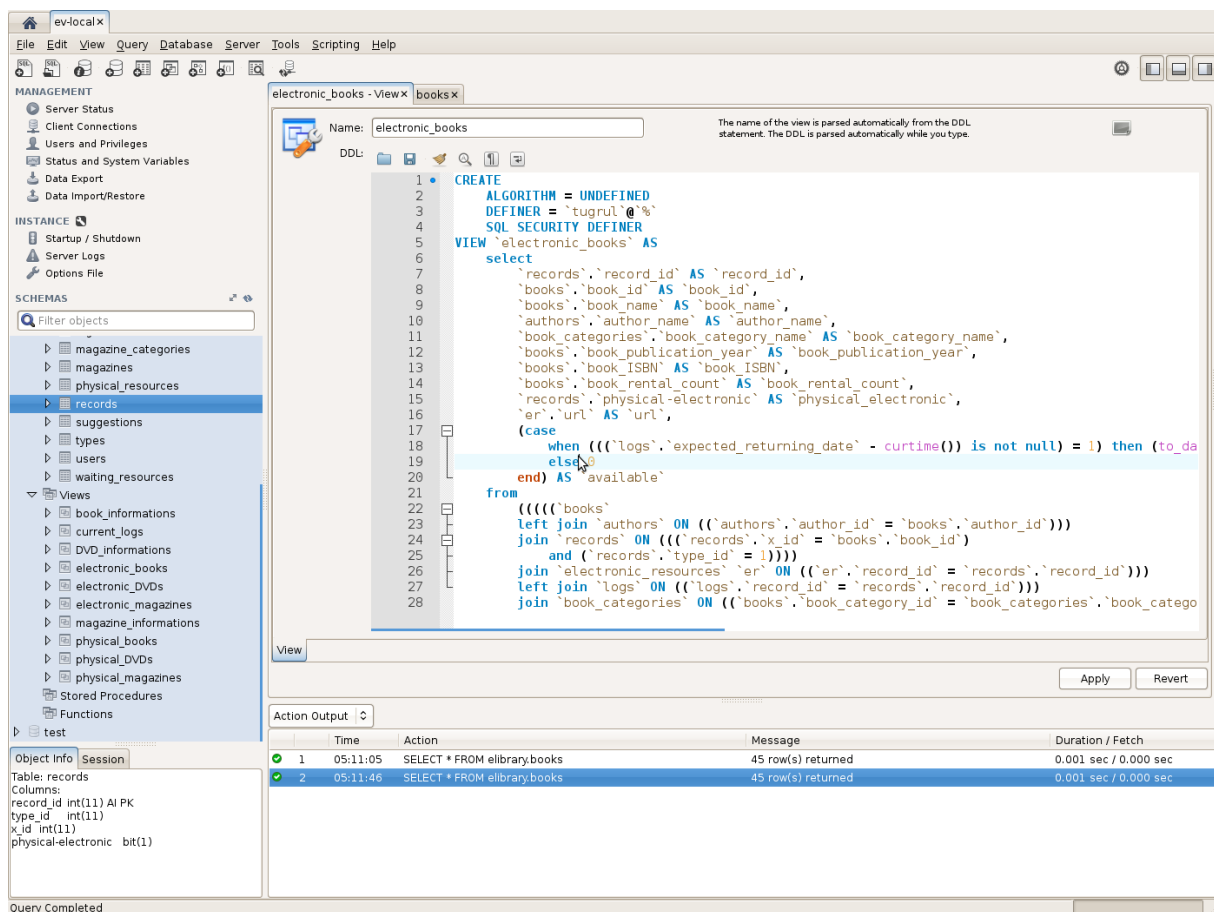
## 1.2 Development and Execution Environment

Project is developed using Java programming language with Apache Wicket web framework and MySQL database server. Project is executed on Apache Tomcat web server. Apache Wicket framework provides connection between Java and HTML. In addition JDBC libraries' MySQL driver is used for connection MySQL server with Java application. Over JDBC SQL queries can be executed directly.

Project is developed on Eclipse under Linux Ubuntu 12.04 operating system. Eclipse is the integrated development environment and it is also used for developing, debugging and starting Tomcat web server.



Also graphical Interface tool MySQL Workbench has facilitated the database operations during development.





## 1.3 Tasks of Group Members

### 1.3.1 Mustafa Uçar

Student's Number : 040100113

Creation of some classes, management of some tables and views in the database are done. These java classes are;

- AddMagazineForm.java
- AddMagazinePage.java
- BasicAuthenticationSession.java
- MagazineEditForm.java
- MagazineEditPage.java
- Magazine.java
- MagazinePage.java
- register.java
- registerPage.java
- SearchForm.java
- SearchPage.java
- SignInPage.java
- UserControl.java
- User.java
- UserLogin.java

### 1.3.2 Tugrul Yatağan

Student's Number: 040100117

- Database server setup and database initialization.
- Creating project repository on Pikacode and setup project's Mercurial version control
- Classes which are implemented by me:
  - AddSourcePage.java
  - Common.java
  - DatabaseConnection.java
  - EditAdminProfilePage.java
  - ditProfilePage.java
  - UserEditProfileForm.java
  - SourceEditForm.java
  - SourceEditPage.java
  - SourcePlacedForm.java
  - SourcePlacedPage.java
  - Start.java
  - WicketApplication.java

### 1.3.3 Emre Gökrem

Student's Number: 040100124

Creation of some classes, management of some tables, views and all pages design styles, icons in the database are done. These java classes are;

- AddBookForm.java
- AddBookPage.java
- AdminNavigationPanel.java
- ArchivePage.java
- BasePage.java
- BookEditForm.java
- BookEditPage.java
- Book.java
- BookPage.java
- GuestNavigationPanel.java
- HomePage.java
- NavigationPanel.java
- PopularSourcesPage.java
- RequestsSuggestionsPage.java
- ShowSuggestionsPage.java
- UserNavigationPanel.java

### 1.3.4 Hüseyin Erdoğan

Student's Number: 040100054

Preparing Group Report

Preparing Presentation

Classes which are created by me

- AboutUsPage.java
- AddDVDForm.java
- AddDVDPage.java
- AdminProfilePage.java
- ContactPage.java
- DVDEditForm.java
- DVDEditPage.java
- Dvd.java
- DvdPage.java
- UserProfilePage.java
- UserPage.java

# **PART II**

## **2 Setup**

## 2.1 Connection to MySQL Database Server

Project uses one main dedicated MySQL database server. All nodes connects this common server via MySQL JDBC driver. This structure gives us data union while developing the project. Main database connection is established in *DatabaseConnection* class. In *DatabaseConnection* database *host*, *port*, *name*, *user name*, *user password* and *encoding type* is specified.

```
try {
    Class.forName("com.mysql.jdbc.Driver"); // JDBC MySQL driver class selection
} catch (ClassNotFoundException e) {
    throw new UnsupportedOperationException(e.getMessage());
}
try {
    String dbUrl = "jdbc:mysql://" + // JDBC driver
        "24.133.151.17" + ":" + // database server host address
        "3306" + "/" + // database server port
        "elibrary" + // database name
        "?useUnicode=true&characterEncoding=utf-8"; // connection encoding
    String dbUser = "dmbs"; // database user name
    String dbPassword = "itudb1302"; // database user password
    this.dbConnect = DriverManager.getConnection(dbUrl, dbUser,
        dbPassword);

    statement = this.dbConnect.createStatement();
} catch (SQLException ex) {
    throw new UnsupportedOperationException(ex.getMessage());
}
```

## 2.2 Database Server Installation and Setup

A Linux Debian Wheezy server is used for dedicated database server. MySQL database server installed on Debian Wheezy from standard package manager with following commands:

```
$ sudo apt-get install mysql-server-5.5
```

During installation user name, user password and root password are entered. When installation ends MySQL configuration file is opened with following command:

```
$ sudo emacs /etc/mysql/my.cnf
```

And the following line is edited for changing server's listening address (listen all address):

```
bind-address = *
```

Server is reloaded for changing to take effect

```
$ sudo service mysql reload
```

And than we can access to database server on command line by giving user name, host name, host port with following command:

```
$ mysql -u root -p -h 24.133.151.17 -P 3306
```

During development of this project host name was: 24.133.151.17

Adding new users to database server is done by following code:

```
mysql> CREATE USER 'tugrul'@'%' IDENTIFIED BY 'password';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'tugrul'@'%' WITH GRANT OPTION;

mysql> CREATE USER 'mustafa'@'%' IDENTIFIED BY 'password';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'mustafa'@'%' WITH GRANT OPTION;

mysql> CREATE USER 'emre'@'%' IDENTIFIED BY 'password';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'emre'@'%' WITH GRANT OPTION;

mysql> CREATE USER 'huseyin'@'%' IDENTIFIED BY 'password';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'huseyin'@'%' WITH GRANT OPTION;
```

After that new database is created on database server with UTF-8 encoding via following command:

```
mysql> CREATE DATABASE elibrary CHARACTER SET utf8 COLLATE utf8_general_ci;
```

## **PART III**

### **3 User Manual**

## 3.1 Guest Interface

### 3.1.1 Guest Home Page

Guest will see this page, when he/she enter to website.



### 3.1.2 Guest Search Procedure

Guest can search using source's name, type of source(electronic or physical) and genre of source.



Genre of search part has two dropdown. First drop box have Book, Dvd, Magazine options.



Second drop box have subspecies of selected type of source. For book : education, engineering, history, novel, science, story, technology. For DVD : education, game, movie, song, documantery. For Magazine :comic, computer, engineering, life, science.



Also guest can search book name, author name, dvd name or magazine name. Queries for searching is used in SearchPage.java class. They are like as follows :



After clicking to search button records are shown. Under the Rent row, record have link to record page.

Guest can see information about source but they can not rent sources.



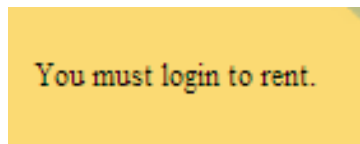
# BOOK INFORMATION PAGE

**Book Name :** Bin Dokuz Yüz Seksen Dört  
**Author :** George Orwell  
**Publication Year :** 1949  
**Category :** Novel  
**ISBN :** 43  
**Rental Count :** 0

## SOURCE'S RECORDS

#	Location	Availability	Rent
1	Mustafa İnan Kütüphanesi - m3 floor - 7.bookcase - 24.bookshelf - 34.order	Available	<b>RENT</b>

If they want to rent a source, they will see this error message.



### 3.1.3 Most Popular Sources

Guest can see most rented sources in this part.

## ★ MOST POPULAR 5 BOOKS

Book Name	Author	Category	Publication Year	Popularity	Link
Dava	Franz Kafka	Novel	1925	1	<a href="#">Book's Page</a>
İlahi Komediya-Cehennem	Dante Alieghieri	Novel	2013	0	<a href="#">Book's Page</a>
Tarihe Yön Veren Büyük Komutanlar	Barry Strauss	History	2013	0	<a href="#">Book's Page</a>
Sayısal Elektronik	M. Kaya Yazgan	Engineering	2013	0	<a href="#">Book's Page</a>
Mühendislik Mekaniği Dinamik	Mehmet Bakioğlu	Education	2012	0	<a href="#">Book's Page</a>

## ★ MOST POPULAR 5 MAGAZINES

Magazine Name	Issue Number	Category	Publication Year	Popularity	Link
Penguen	2	Comic	2005	1	<a href="#">Magazine's Page</a>
Oyungezer	6	Computer	2005	1	<a href="#">Magazine's Page</a>
Penguen	1	Comic	2004	0	<a href="#">Magazine's Page</a>
Chip	10	Computer	1992	0	<a href="#">Magazine's Page</a>
IEEE	5	Science	1960	0	<a href="#">Magazine's Page</a>

## ★ MOST POPULAR 5 DVDs

DVD Name	Duration	Category	Publication Year	Popularity	Link
Best of Mozart	126	Song	2000	1	<a href="#">DVD's Page</a>
Sefiller	150	Movie	2012	0	<a href="#">DVD's Page</a>
Beatles - Please Please Me	60	Song	1963	0	<a href="#">DVD's Page</a>
Cosmos	60	Documentary	1980	0	<a href="#">DVD's Page</a>
Battlefield	99	Game	2013	0	<a href="#">DVD's Page</a>

Queries are used for this purpose as follows :

For book :

```
// BOOK
DatabaseConnection dbc = new DatabaseConnection();
String query = "SELECT book_id, book_name, author_name, book_category_name, book_publication_year, book_rental_count "
+ "FROM books JOIN authors ON (books.author_id = authors.author_id) "
+ "JOIN book_categories ON (books.book_category_id = book_categories.book_category_id) "
+ "ORDER BY book_rental_count DESC LIMIT 5;";
ResultSet rs = dbc.getResult(query);
List<Book> booklist = new ArrayList();
```

For dvd :

```
// DVD
dbc = new DatabaseConnection();
query = "SELECT DVD_id, DVD_name, DVD_category_name, DVD_duration, DVD_publication_date, DVD_rental_count "
+ "FROM DVDs "
+ "JOIN DVD_categories ON (DVDs.DVD_category_id = DVD_categories.DVD_category_id) "
+ "ORDER BY DVD_rental_count DESC LIMIT 5;";

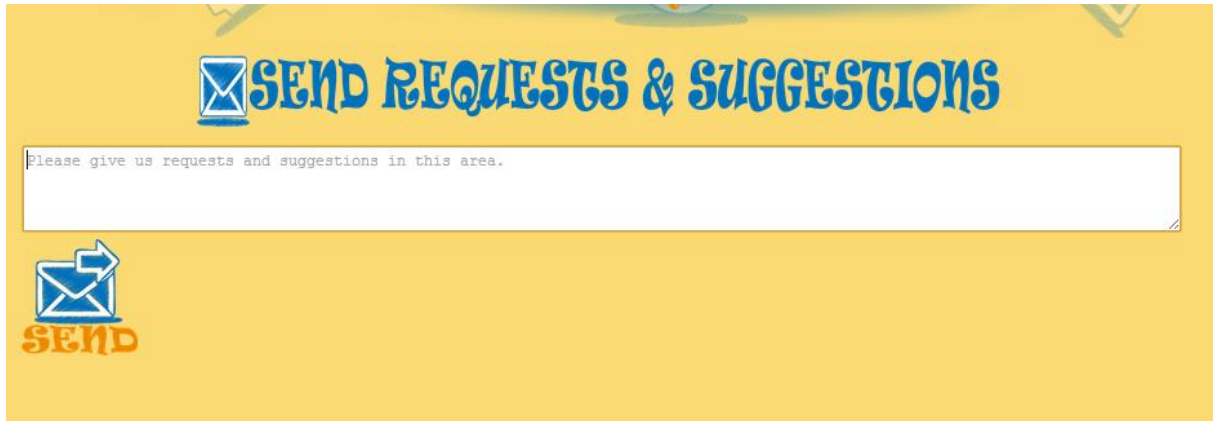
rs = dbc.getResult(query);
List<Dvd> dvdlist = new ArrayList();
```

For magazine

```
// MAGAZINE
dbc = new DatabaseConnection();
query = "SELECT magazine_id, magazine_name, magazine_category_name, magazine_issue_number, magazine_publication_date, magazine_rental_count "
+ "FROM magazines "
+ "JOIN magazine_categories ON (magazines.magazine_category_id = magazine_categories.magazine_category_id) "
+ "ORDER BY magazine_rental_count DESC LIMIT 5;";
rs = dbc.getResult(query);
List<Magazine> magazinelist = new ArrayList();
```

### 3.1.4 Request & Suggestions Page

In this part guests can send their requests and suggestions to admin for sake of website. If guests does not register to website, their requests and suggestions will send to admin with “publicuser” nickname.



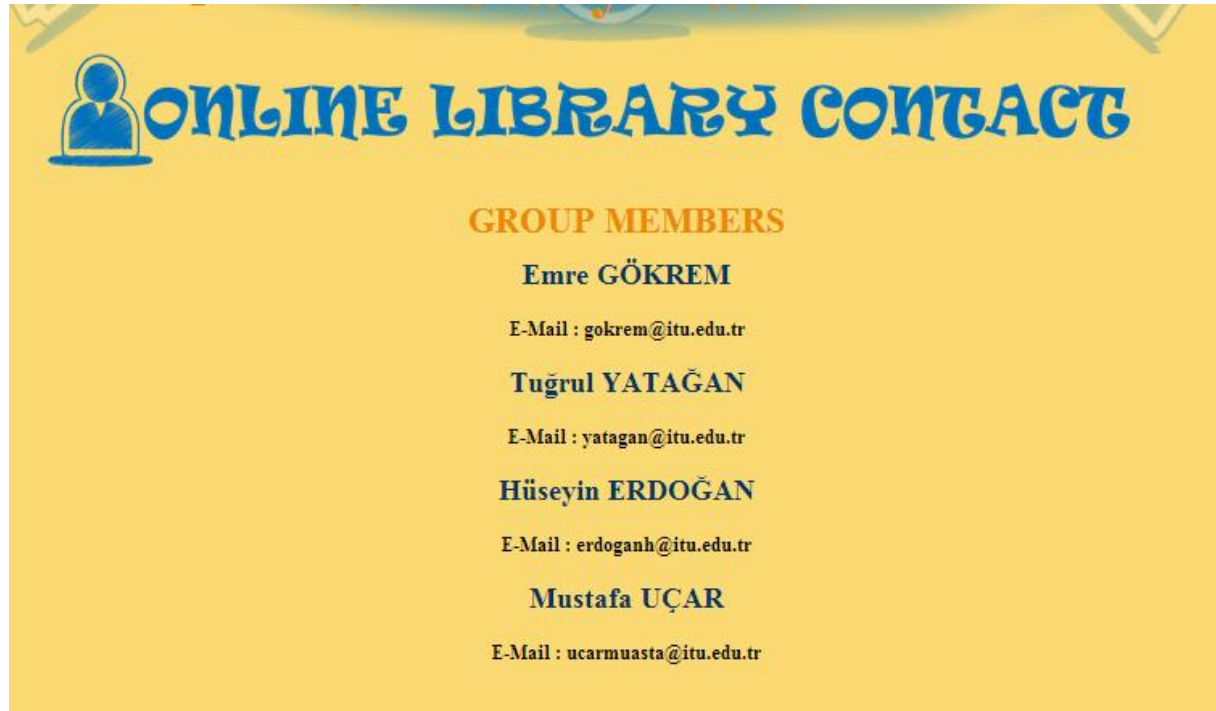
These queries and “suggestions” table are used for requests and suggestions part. Queres are as follows :

```
if (((BasicAuthenticationSession) BasicAuthenticationSession
    .get()).isSignedIn()) {
    final Integer user_id = ((BasicAuthenticationSession) BasicAuthenticationSession
        .get()).getUser().get_id();

    str = String
        .format("INSERT INTO suggestions (`suggestion`,`user_id`) VALUES ('%s',%d)",
            input, user_id);
} else {
    str = String
        .format("INSERT INTO suggestions (`suggestion`) VALUES ('%s')",|
            input);
}
```

### 3.1.5 Contact Us Page

In this page group members’s e-mail addresses are showed so guest can interact with us.



**ONLINE LIBRARY CONTACT**

**GROUP MEMBERS**

**Emre GÖKREM**  
E-Mail : gokrem@itu.edu.tr

**Tuğrul YATAĞAN**  
E-Mail : yatagan@itu.edu.tr

**Hüseyin ERDOĞAN**  
E-Mail : erdoganh@itu.edu.tr

**Mustafa UÇAR**  
E-Mail : ucarmuasta@itu.edu.tr

### 3.1.6 About Us Page

In this page information about project is given. Also informations about libraries that are used in database and links of them are given.



**ONLINE LIBRARY ABOUT US**

**INFORMATION ABOUT E-LIBRARY PROJECT**

The aim of this project is providing whole library system which is accessible via web. Users will be able to see sources (book, e-book, DVD, magazine etc.) and if users are member of the system, they will be rent sources. Users will be able to find the location of the sources from website and they can take actual material from library. If a user wants to use electronic sources from website, he/she will be able to find information about electronic source and a link of the electronic source. The system will have administrator users. Administrator users will be able to control the system (adding source, deleting source, updating source,). The system will have different searching types (type of source and genre of search). In addition, popular sources will be listed on the website according to popular search topics and popular rented sources. If a user does not enter the system for a year, account of the user will be deleted by the system.

**CONTACT INFORMATION ABOUT LIBRARIES**

Mustafa İnan Kütüphanesi Tel : (0212) 285 35 96  
Mustafa İnan Kütüphanesi hakkında daha fazla bilgi için [tıklayın](#)  
Makine Fakültesi Ratip Berker Kütüphanesi Tel : (0212) 293 13 00/2163  
Makine Fakültesi Ratip Berker Kütüphanesi hakkında daha fazla bilgi için [tıklayın](#)

## 3.2 User Interface

### 3.2.1 Register

Guest can register anytime by clicking register link under login symbol. Guest encounter this page :



If guest enter another user information, he/she will see this error message.

**Bu Kullanıcı Adı Zaten Kayıtlı, Yeniden Deneyin.**

### 3.2.2 Login

After registering, user can login to website using Username and Password.



After login we take user informations from database as follows :

```

String query = String.format(
    "SELECT * FROM users where user_nickname = '%s'", username);
rs = dbc.GetResult(query);
while (rs.next()) {
    tempUser = new User(rs.getString("user_name"),
        rs.getString("user_surname"),
        rs.getString("user_nickname"),
        rs.getString("user_password"),
        rs.getInt("user_authority_state"),
        rs.getInt("user_point"));
    tempUser.set_id(rs.getInt("user_id"));
    tempUser.set_last_access_time(rs.getTimestamp("user_last_access_time"));
}

```

According to user\_authority\_state, user is redirected to user or admin homepage. Also user can do everything that guest can do.

And user\_last\_access\_time is updated in these lines :

```

query = String
    .format("UPDATE users SET `user_last_access_time` = NOW() WHERE user_id = '%d'",
        tempUser.get_id());
dbc.Insert(query);

```

### 3.2.3 Logout

User can logout anytime from website by clicking logout icon. Also there is name and surname under the logout symbol. After Logout user is redirected to guest Home Page .



After Logout user is redirected to guest Home Page .

### 3.2.4 User Home Page

User will see this page, when he/she logs in to website.





### 3.2.5 User Searching Procedure

Searching a source is same as guest search procedure. Difference between this two procedure is at source information page. In this page user can rent source which are available. Also user can see location of source and when source is available.



If user press rent link of source that available, information message will be shown. Also user can put sources ,which are not available, to waiting list by clicking **WAIT** link in information page.

For every source type different queries are used for renting part.

- Queries which are used for renting Book are as follows :



```

private void rent(Book book) throws Exception {
    DatabaseConnection dbc = new DatabaseConnection();

    String str = String.format(
        "select `user_id` from users where `user_nickname` = '%s';",
        ((BasicAuthenticationSession) BasicAuthenticationSession.get())
            .getUser().getUsername());
    ResultSet rs = dbc.GetResult(str);
    rs.next();
    int x = rs.getInt("user_id");
    dbc.Insert(String
        .format("INSERT INTO logs (`user_id`,`record_id`,`renting_date`,`expected_returning_date`)"
            + " values (%d,%d,current_date(),DATE_ADD(current_date(),INTERVAL 30 DAY));",
            x, book.get_record_id()));
    dbc.Insert(String
        .format("UPDATE physical_resources MODIFY SET `current_log_id` = (SELECT max(log_id) FROM logs)"
            + " where record_id = %d;", book.get_record_id()));
    dbc.Insert(String
        .format("UPDATE books MODIFY SET `book_rental_count` = `book_rental_count` + 1"
            + " where `book_id` = (SELECT `x_id` from records where `record_id`= %d)",
            book.get_record_id()));
    dbc.close();
}

```

- Queries which are used for renting DVD are as follows :

```

private void rent(Dvd dvd) throws Exception {
    DatabaseConnection dbc = new DatabaseConnection();

    String str = String.format(
        "select `user_id` from users where `user_nickname` = '%s';",
        ((BasicAuthenticationSession) BasicAuthenticationSession.get())
            .getUser().getUsername());
    ResultSet rs = dbc.GetResult(str);
    rs.next();
    int x = rs.getInt("user_id");
    dbc.Insert(String
        .format("INSERT INTO logs (`user_id`,`record_id`,`renting_date`,`expected_returning_date`)"
            + " values (%d,%d,current_date(),DATE_ADD(current_date(),INTERVAL 30 DAY));",
            x, dvd.get_record_id()));
    dbc.Insert(String
        .format("UPDATE physical_resources MODIFY SET `current_log_id` = (SELECT max(log_id) FROM logs)"
            + " where record_id = %d;", dvd.get_record_id()));
    dbc.Insert(String
        .format("UPDATE DVDs MODIFY SET `DVD_rental_count` = `DVD_rental_count` + 1"
            + " where `DVD_id` = (SELECT `x_id` from records where `record_id`= %d)",
            dvd.get_record_id()));
    dbc.close();
}

```

- Queries which are used for renting Magazine are as follows :

```

private void rent(Magazine magazine) throws Exception {
    DatabaseConnection dbc = new DatabaseConnection();

    String str = String.format(
        "select `user_id` from users where `user_nickname` = '%s';",
        ((BasicAuthenticationSession) BasicAuthenticationSession.get())
            .getUser().getUsername());
    ResultSet rs = dbc.GetResult(str);
    rs.next();
    int x = rs.getInt("user_id");
    dbc.Insert(String
        .format("INSERT INTO logs (`user_id`,`record_id`,`renting_date`,`expected_returning_date`)"
            + " values (%d,%d,current_date(),DATE_ADD(current_date(),INTERVAL 30 DAY));",
            x, magazine.get_record_id()));
    dbc.Insert(String
        .format("UPDATE physical_resources MODIFY SET `current_log_id` = (SELECT max(log_id) FROM logs)"
            + " where record_id = %d;", magazine.get_record_id()));
    dbc.Insert(String
        .format("UPDATE magazines MODIFY SET `magazine_rental_count` = `magazine_rental_count` + 1"
            + " where `magazine_id` = (SELECT `x_id` from records where `record_id`= %d)",
            magazine.get_record_id()));
    dbc.close();
}

```

Sources, which are puts to waiting list by user, are stored in “waiting\_resources” table. Queries are as follows.

```

item.add(new Link("waitlink"). {
    @Override
    public void onClick() {
        try {
            String str = String
                .format("select `user_id` from users where `user_nickname` = '%s';",
                    ((BasicAuthenticationSession) BasicAuthenticationSession
                        .get()).getUser()
                        .getUsername());

            ResultSet rs;

            rs = dbc.GetResult(str);
            rs.next();
            int x = rs.getInt("user_id");
            dbc.Insert(String
                .format("INSERT INTO waiting_resources (`user_id`,`record_id`) values (%d,%d);",
                    x, magazine.get_record_id()));

            this.setResponsePage(new MagazinePage(magazine, ""));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});

```

This part is same for all types.

### 3.2.6 Most Popular Page

This page is same as guest Most Populars page, except renting procedure. After clicking source’s links user can see location of sources and user can rent them.

### 3.2.7 User Profile

User can see firstname, surname, last login time and sources which is user waiting for.

Name	Available	Link	
Best of Mozart	29	<a href="#">Dvd's Page</a>	X
Oyungezer	28	<a href="#">Magazine's Page</a>	X

[Give Us Suggestions & Requests](#)

Sources are taken from “waiting\_resources” table in database by using queries as follows :


```


dbc = new DatabaseConnection();
String str = String.format(
    "select `user_id` from users where `user_nickname` = '%s';",
    ((BasicAuthenticationSession) BasicAuthenticationSession.get())
        .getUser().getUsername());
ResultSet rs = dbc.getResult(str);
rs.next();
int x = rs.getInt("user_id");
add(getBookTable(String
    .format("SELECT * FROM book_informations where record_id in "
        + "(SELECT record_id FROM waiting_resources where user_id = %d)",
        x)));
add(getDvdTable(String
    .format("SELECT * FROM DVD_informations where record_id in "
        + "(SELECT record_id FROM waiting_resources where user_id = %d)",
        x)));
add(getMagazineTable(String
    .format("SELECT * FROM magazine_informations where record_id in "
        + "(SELECT record_id FROM waiting_resources where user_id = %d)",
        x)));
dbc.close();
}

```

Also there is a link to suggestion and request page and if user sends request, user name will be seen by admins.

User can delete sources which are in the waiting source list. User\_id and record\_id are used for finding sources.

Name	Available	Link	
Best of Mozart	29	<a href="#">Dvd's Page</a>	

If user clicks  button, source which is in waiting list will be deleted. Queries ,which are used for this purpose, are same for all types.

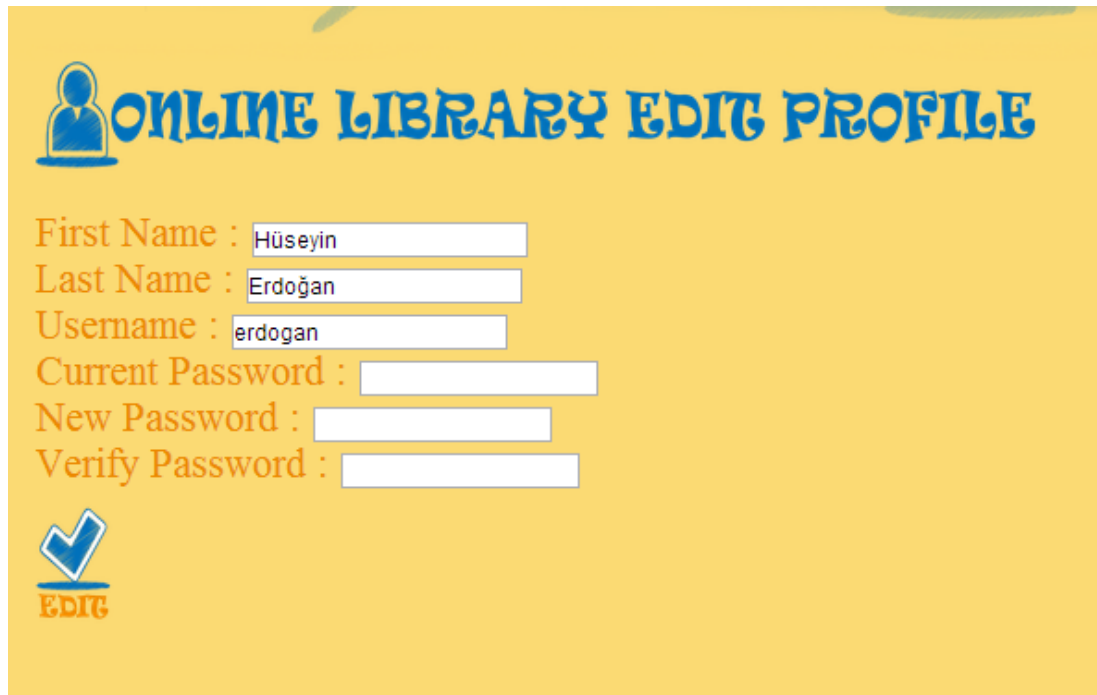
```

String query = String
    .format("DELETE FROM waiting_resources WHERE user_id = %d;",
        ((BasicAuthenticationSession) BasicAuthenticationSession
            .get()).getUser().get_id(), "AND record_id = %d;", dvd.get_record_id());

```

### 3.2.8 User Edit Profile

User can change first name, last name, user name and password. Current Password is necessary for changing all of this information.



**ONLINE LIBRARY EDIT PROFILE**

First Name :


Last Name :

Username :

Current Password :

New Password :

Verify Password :

 **EDIT**

Functions ,that are used in this page, are in UserEditProfileForm.java class. Queries are as follows:

```

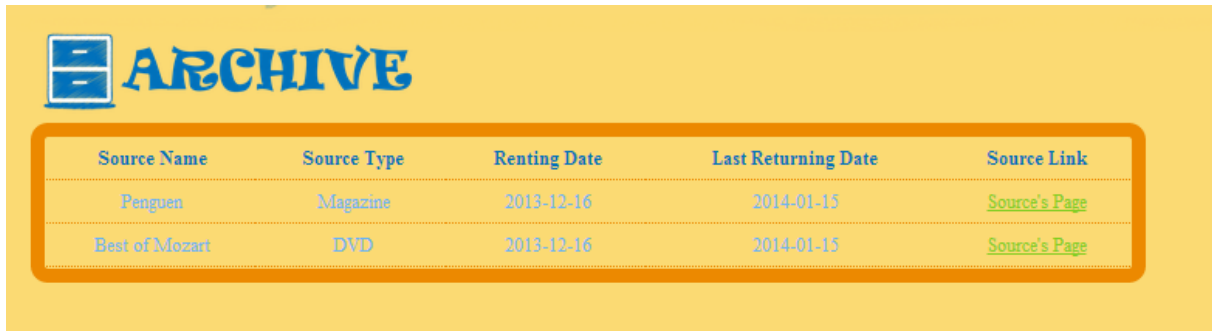
ResultSet rs;
try {
    rs = dbc.GetResult(String
        .format("select `user_id`,`user_password` from users where `user_nickname` = '%s';",
            ((BasicAuthenticationSession) BasicAuthenticationSession
                .get()).getUser().getUsername()));

    rs.next();
    int x = rs.getInt("user_id");
    String u_pass = rs.getString("user_password");
    if (u_name.getModelObject() != null) {
        dbc.Insert(String
            .format("UPDATE users MODIFY SET `user_name` = '%s' where `user_id` = %d;",
                u_name.getModelObject(), x));
    }
    if (u_lname.getModelObject() != null) {
        dbc.Insert(String
            .format("UPDATE users MODIFY SET `user_surname` = '%s' where `user_id` = %d;",
                u_lname.getModelObject(), x));
    }
    if (u_uname.getModelObject() != null) {
        dbc.Insert(String
            .format("UPDATE users MODIFY SET `user_nickname` = '%s' where `user_id` = %d;",
                u_uname.getModelObject(), x));
    }
    String cur_pass = (String) currentpassword.getModelObject();
    if (cur_pass.equals(u_pass)
        && newpassword.getModelObject() != null
        && anewpassword.getModelObject() != null
        && newpassword.getModelObject().toString()
            .equals(anewpassword.getModelObject().toString())) {
        dbc.Insert(String
            .format("UPDATE users MODIFY SET `user_password` = '%s' where `user_id` = %d;",
                newpassword.getModelObject(), x));
    }
}

```

### 3.2.9 User Archive

In this page sources are listed which are rented by user.



Source Name	Source Type	Renting Date	Last Returning Date	Source Link
Penguen	Magazine	2013-12-16	2014-01-15	<a href="#">Source's Page</a>
Best of Mozart	DVD	2013-12-16	2014-01-15	<a href="#">Source's Page</a>

In this page “logs” table is used for taking sources which are rented by user. Functions ,that are used in this page, are in ArchivePage.java class. Queries are as follows:

```
String query = String
    .format("SELECT x_id, type_id, renting_date, expected_returning_date "
        + "FROM logs JOIN records ON (records.record_id = logs.record_id) WHERE user_id = %d "
        + "ORDER BY log_id DESC LIMIT 15", person_id);
ResultSet rs = dbc.GetResult(query);
```

## 3.3 Admin Interface

Admin login to website same as user but different than user admin's user\_auotority\_state is “1”.


### 3.3.1 Admin Home Page

Admin will see this page, when he/she logins to website.



### 3.3.2 Admin Profile

Admin will see profil information and last 10 renting process.



## ONLINE LIBRARY ADMIN PROFILE

Name :   
 Surname :   
 Last Login Time : 2013-12-16 19:30:19

### Last 10 Library Renting Process

User	User Last Access	Source	Source Type	Renting Date	Expected Returning Date	Source Link
Hüseyin Erdoğan	2013-12-16 19:31:52	Penguen	Magazine	2013-12-16	2014-01-15	<a href="#">Source's Page</a>
Hüseyin Erdoğan	2013-12-16 19:31:52	Best of Mozart	DVD	2013-12-16	2014-01-15	<a href="#">Source's Page</a>
Tuğrul Yatağan	2013-12-15 14:43:11	Oyungezer	Magazine	2013-12-15	2014-01-14	<a href="#">Source's Page</a>
Üye Kullanıcı	2013-12-16 19:16:57	Dava	Book	2013-12-15	2014-01-14	<a href="#">Source's Page</a>

Last 10 renting process is used for controlling user activities and functions about this page in AdminProfilePage.java class. Queries, which are used for displaying last 10 renting process, is as follows :

```
try {
    String query = "SELECT user_name, user_surname, user_last_access_time, "
        + "x_id, type_id, renting_date, expected_returning_date "
        + "FROM logs JOIN users ON (logs.user_id = users.user_id) "
        + "JOIN records ON (records.record_id = logs.record_id) "
        + "ORDER BY log_id DESC LIMIT 10; ";
    ResultSet rs = dbc.GetResult(query);
```

### 3.3.3 Admin Edit Profile

This part is same as user edit profile.



## ONLINE LIBRARY EDIT PROFILE

First Name :   
 Last Name :   
 Username :   
 Current Password :   
 New Password :   
 Verify Password :

  
 EDIT

### 3.3.4 Admin Search Procedure

Search part is same as user search but admins have extra opportunity. Admins can search any item and also they can delete them, update them and put them to libraries. After searching admin will see page like this, which may change according to source type or genre of source.



Name	Category	Issue Number	Publication Date	Rent
Chip	Computer	10	1992	<a href="#">Magazine's Page</a>
IEEE	Science	5	1960	<a href="#">Magazine's Page</a>
Oyungezer	Computer	6	2005	<a href="#">Magazine's Page</a>
Penguen	Comic	1	2004	<a href="#">Magazine's Page</a>
Penguen	Comic	2	2005	<a href="#">Magazine's Page</a>
Time	Life	6	1923	<a href="#">Magazine's Page</a>

When admins click to [Magazine's Page](#), he/she will see this page unlike guest and user.



**Magazine Name :** Penguen Magazine Information  
**Publication Year :** 2004  
**Category :** Comic  
**Issue :** 1  
**Rental Count :** 0  
**ADD EDIT X**

### SOURCE'S RECORDS

#	Location	Availability	Edit	Delete
1	<a href="http://www.penguen.com">http://www.penguen.com</a>	Available	<b>EDIT</b>	<b>X</b>
2	Mustafa İnan Kütüphanesi - m2.floor - d.bookcase - da.bookshelf - 20.order	Available	<b>EDIT</b>	<b>X</b>



### 3.3.4.1 Source Placed Page

When admin clicks **ADD**, he/she will see Online Library Source Placed Page. In this page, admin enters library name and according to library, admin select library floor, bookcase, bookshelf and column number. If source is also electronic, admin can enter source's URL.

Source Placed Page is common for all tree type.(book, DVD, magazine).

If source is physical, source will be stored in “physical\_resources” table. Queries as follows :

```
query = String
        .format("INSERT INTO physical_resources (record_id, bookshelf_id, book_column) VALUES ('%d', '%d', '%d');",
                c.get_record_id(), c.get_bookshelf_id(),
                c.get_column());
dbc.Insert(query);
```

If source is electronic, source will be stored in “electronic\_resources” table. Queries as follows :

```
query = String
        .format("INSERT INTO electronic_resources (`record_id`, `url`) VALUES ('%d', '%s');",
                c.get_record_id(), c.get_url());
dbc.Insert(query);
```

### 3.3.4.2 Source's Records

In this part user sees location of sources. If user is admin, he/she can edit and delete location of source.

#	Location	Availability	Edit	Delete
3	Mustafa İnan Kütüphanesi - m3.floor - g.bookcase - gb.bookshelf - 34.order	Available	EDIT	X

Admin can delete location of source by clicking **X**.



### 3.3.4.3 Source's Records Edit

If admin clicks **EDIT** in source's records table, he/she will see this page. If there is a need update for source's location, admin can do it in this page.

**SOURCE EDIT PAGE**

Source Name : Bin Dokuz Yüz Seksen Dört

Library Name: Mustafa İnan Kütüphanesi

Library Floor: m3

Bookcase: g

Bookshelf: gb

Column(1-50): 34

Place Change

### 3.3.4.4 Source Edit

When admin clicks **EDIT** in in book information page, he/she will see this page for updating book informations.

**BOOK EDIT PAGE**

Book Name : Avrupa Hunlari

Author : Ali Ahmetbeyoğlu

Publication Year : 2010

ISBN : 9786055200282

Category : History

**EDIT**

Query, which is used in this part as follows :

```

dbc = new DatabaseConnection();
query = String
    .format("UPDATE books SET book_name = '%s', author_id = '%d', book_publication_year = '%d', " +
        " book_ISBN = '%s', book_category_id = %d WHERE book_id = %d;",
        book.get_name(), author_id,
        book.get_publish_year(), book.get_ISBN(),
        book.get_category_id(), book.get_book_id());

dbc.Insert(query);

```

When admin clicks **EDIT** in DVD information page, he/she will see this page for updating DVD informations.

Query, which is used in this part as follows :

```

dbc = new DatabaseConnection();
query = String
    .format("UPDATE DVDs SET DVD_name = '%s', DVD_duration = %d, "
        + " DVD_publication_date = '%d-01-01', DVD_category_id = %d WHERE DVD_id = %d;",
        dvd.get_name(), dvd.get_duration(),
        dvd.get_publish_year(), dvd.get_category_id(), dvd.get_DVD_id());

dbc.Insert(query);

```

When admin clicks **EDIT** in magazine information page, he/she will see this page for updating magazine informations.



## MAGAZINE EDIT PAGE

Magazine Name :

Issue Number :

Publication Year :

Category :


 **EDIT**

Query, which is used in this part as follows :

```
dbc = new DatabaseConnection();
query = String
    .format("UPDATE magazines SET magazine_name = '%s', magazine_issue_number = %d, "
        + " magazine_publication_date = '%d-01-01', magazine_category_id = %d WHERE magazine_id = %d;",
        magazine.get_name(), magazine.get_issue_number(),
        magazine.get_publish_year(), magazine.get_category_id(), magazine.get_magazine_id());

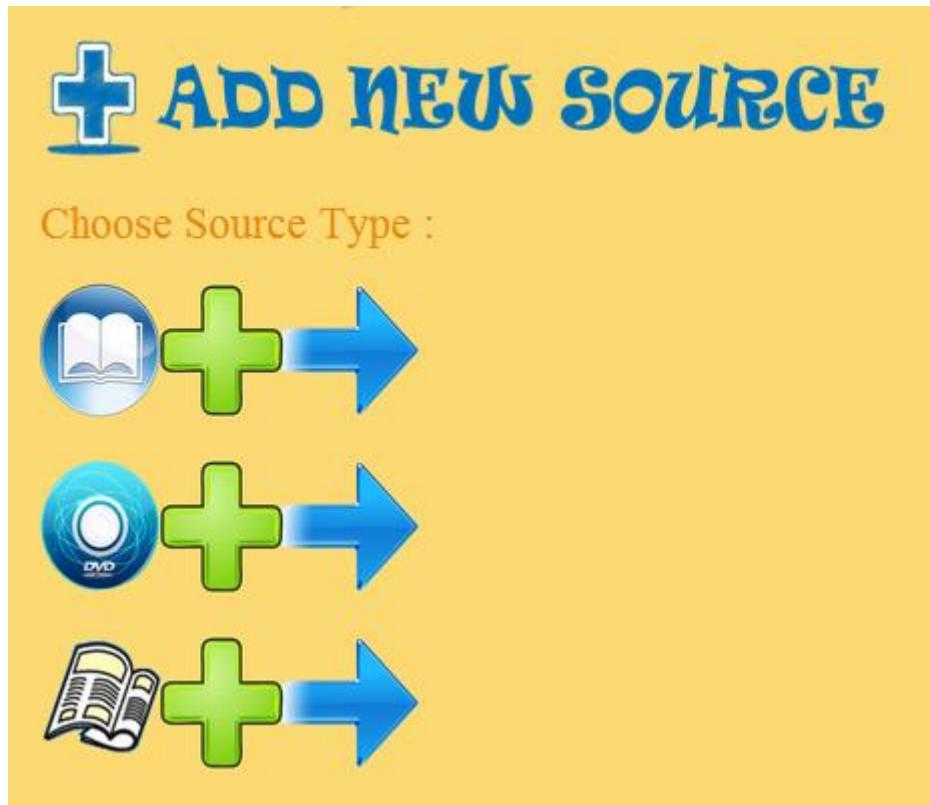
dbc.Insert(query);
```

### 3.3.4.5 Source Delete

When admin clicks  for any type, he/she will be redirected to admin home page and source will be deleted with all of its records.

### 3.3.5 Adding Source

Admins can add source. First admin selects type of source(book, magazine, dvd).



### 3.3.5.1 Adding Book

After selecting type of source, admin enters information of source. For books : book name, author name, publication year, ISBN and book category.

In this part AddBookForm.java and AddBookPage.java classes are used. Query is as follows :

```
dbc = new DatabaseConnection();
query = String
    .format("INSERT INTO books (book_name, author_id, book_publication_year, book_ISBN, book_category_id) VALUES ('%s', '%d', '%d', '%s', '%d');",
        book.get_name(), author_id,
        book.get_publish_year(), book.get_ISBN(),
        book.get_category_id());
dbc.Insert(query);
```

### 3.3.5.2 Adding DVD

Admin enters information of source. For DVDs : DVD name, publication year, duration and DVD category.

In this part AddDVDForm.java and AddDVDPage.java classes are used. Querie is as follows :

```
dbc = new DatabaseConnection();
query = String
    .format("INSERT INTO DVDs (DVD_name, DVD_duration, DVD_publication_date, DVD_category_id) VALUES ('%s', '%d', '%d-01-01', '%d');",
        dvd.get_name(), dvd.get_duration(),
        dvd.get_publish_year(), dvd.get_category_id());

dbc.Insert(query);
```

### 3.3.5.3 Adding Magazine

Admin enters information of source. For magazines : magazine name, publication year, issue number, and magazine category.

In this part AddMagazineForm.java and AddMagazinePage.java classes are used. Querie is as follows :

```
dbc = new DatabaseConnection();
query = String
    .format("INSERT INTO magazines (magazine_name, magazine_issue_number, magazine_publication_date, magazine_category_id) VALUES ('%s', '%d', '%d-01-01', '%d');",
        magazine.get_name(), magazine.get_issue_number(),
        magazine.get_publish_year(),
        magazine.get_category_id());

dbc.Insert(query);
```

### 3.3.6 Admin Archive

Admin can see requests and suggestions in this page.

Users Requests & Suggestions		
Message	User	Delete
Site çok güzel olmuş elinize sağlık	Public User	<a href="#">Delete This Message</a>
Bu siteyi sürekli kullanıyorum harikasınız	Public User	<a href="#">Delete This Message</a>
Mustafa hariç geri kalanınızı çok seviyorum, başarılar	Public User	<a href="#">Delete This Message</a>
Çok güzel site olmuş elinize sağlık.	Public User	<a href="#">Delete This Message</a>

Also admin can delete this requests and suggestions.

Users Requests & Suggestions		
Message	User	Delete
Site çok güzel olmuş elinize sağlık	Public User	<a href="#">Delete This Message</a>
Bu siteyi sürekli kullanıyorum harikasınız	Public User	<a href="#">Delete This Message</a>
Mustafa hariç geri kalanınızı çok seviyorum, başarılar	Public User	<a href="#">Delete This Message</a>

In this part ShowSuggestionsPage.java class is used. Querie is as follows :

```
String query = "SELECT suggestion_id, suggestion, user_name, user_surname FROM "
+ "suggestions LEFT JOIN users ON suggestions.user_id = users.user_id;";
ResultSet rs = dbc.GetResult(query);
```

If suggestion is send by guest, Public User is written instead of user name.

If suggestion is send by user, user\_name and user\_surname ,which are in "users" table, are used.

## **PART IV**

### **4 Technical Manual**

## 4.1 Database Design

### 4.1.1 Tables

In this chapter tables and some variables are going to be explained.

#### 4.1.1.1 “users” Table

This table stores user data. Informations in this table as follows:

Name	Type	Not Null	Primary Key
user_id	int(11)	Auto increment	1
user_nickname	Varchar(45)	1	0
user_name	Varchar(45)	1	0
user_surname	Varchar(45)	1	0
user_password	Varchar(45)	1	0
user_authority_state	bit(1)	1	0
user_last_access_time	Datetime	1	0
user_point	Varchar(45)	1	0

- user\_authority\_state : information about user that is normal user or admin.
- user\_last\_access\_time : last access time of user. This information used for deleting user that does not enter website for over a year.
- user\_point : penalty point. This points are deleted only by admin.

#### 4.1.1.2 “logs” Table

This table stores data about login procedure and sources that are rented. Informations in this table as follows:

Name	Type	Not Null	Primary Key
log_id	int(11)	1	1
user_id	int(11)	1	0
record_id	int(11)	1	0
renting_date	date	1	0
expected_returning_date	date	1	0
returning_date	date	1	0

- renting\_date : if user rent a source , this shows renting date
- expected\_returning\_date : expected return date of rented source
- user\_authority\_state : return date of rented source

#### 4.1.1.3 “authors” Table



This table stores data about writers of books. Informations in this table as follows:

Name	Type	Not Null	Primary Key
author_id	int(11)	Auto increment	1
author_name	Varchar(45)	1	0
author_information	mediumtext	1	0

- author\_information : Brief introduction about author

#### 4.1.1.4 “books” Table

This table stores book data. Informations in this table as follows:

Name	Type	Not Null	Primary Key
book_id	int(11)	Auto increment	1
book_name	Varchar(45)	1	0
author_id	int(11)	1	0
book_publication_year	Year(4)	1	0
book_ISBN	Varchar(45)	1	0
book_rental_count	int(11)	1	0
book_category_id	int(11)	1	0

- book\_ISBN : this is for electronic books
- book\_rental\_count : renting number of books and used for displaying most popular books

#### 4.1.1.5 “magazines” Table

This table magazine data. Informations in this table as follows:

Name	Type	Not Null	Primary Key
magazine_id	int(11)	Auto increment	1
magazine_name	Varchar(45)	1	0
magazine_issue_number	int(11)	1	0
magazine_publication_date	Date	1	0
magazine_rental_count	int(11)	1	0
magazine_category_id	int(11)	1	0

- magazine\_issue\_number: issue of magazine
- magazine\_rental\_count : renting number of magazines and used for displaying most popular magazines

#### 4.1.1.6 “DVDs” Table

This table stores DVD data. Informations in this table as follows:

Name	Type	Not Null	Primary Key
DVD_id	int(11)	Auto increment	1
DVD_name	Varchar(45)	1	0
DVD_duration	int(11)	1	0
DVD_publication_date	Date	1	0
DVD_rental_count	int(11)	Default "0"	0
DVD_category_id	int(11)	1	0

- DVD\_rental\_count : renting number of DVDs and used for displaying most popular DVDs

#### 4.1.1.7 "book\_categories" Table

This table stores data about book categories. Informations in this table as follows:

Name	Type	Not Null	Primary Key
book_categorie_id	int(11)	Auto increment	1
book_categorie_name	Varchar(45)	1	0

- book\_categorie\_name : used for categorizing books. Ex : bilim, eğitim, roman.

#### 4.1.1.8 "bookcases" Table

This table stores data about bookcases and this informations are used for locating books. Informations in this table as follows:

Name	Type	Not Null	Primary Key
bookcase_id	int(11)	Auto increment	1
floor_id	int(11)	1	0
bookcase_name	Varchar(10)	1	0

- floor\_name : floor name that is bookcase in there

#### 4.1.1.9 "bookshelves" Table

This table stores data about bookshelves and this informations are used for locating bookcase. Also bookshelves have limit. Informations in this table as follows:

Name	Type	Not Null	Primary Key
41lookshelf_id	int(11)	Auto increment	1
bookcase_id	int(11)	1	0
41lookshelf_name	Varchar(10)	1	0
book_limit	int(11)	1	0

- bookcase\_id : bookcase id that is 41lookshelf in there

- book\_limit : maximum number of book that 's Bookshelf contains

#### 4.1.1.10 “DVD\_categories” Table

This table stores data about categories of DVDs. Informations in this table as follows:

Name	Type	Not Null	Primary Key
DVD_category_id	int(11)	Auto increment	1
DVD_category_name	Varchar(45)	1	0

- DVD\_category\_name : used for categorizing DVDs. Ex : dizi, film, eğitim.

#### 4.1.1.11 “electronic\_resources” Table

This table stores data about sources that are electronic. Informations in this table as follows:

Name	Type	Not Null	Primary Key
record_id	int(11)	Auto increment	1
url	Varchar(45)	1	0

- url : users can reach sources on internet using url

#### 4.1.1.12 “physical\_resources” Table

This table stores data about sources that are physical. Informations in this table as follows:

Name	Type	Not Null	Primary Key
record_id	int(11)	Auto increment	1
42bookshelf_id	int(11)	1	0
book_column	int(11)	1	0
current_log_id	int(11)	1	0

#### 4.1.1.13 “waiting\_resources” Table

This table stores data about users that are waiting for sources. Informations in this table as follows:

Name	Type	Not Null	Primary Key
user_id	int(11)	1	1
record_id	int(11)	1	1

- record\_id : source that is users waiting for

#### 4.1.1.14 “floors” Table

This table stores data about floors that libraries have. Informations in this table as follows:

Name	Type	Not Null	Primary Key
floor_id	int(11)	Auto increment	1
library_id	int(11)	1	1
floor_name	Varchar(45)	1	1

- library\_id : library that's contains that floor

#### 4.1.1.15 “libraries” Table

This table stores library data. Informations in this table as follows:

Name	Type	Not Null	Primary Key
library_id	int(11)	Auto increment	1
library_name	Varchar(45)	1	0
library_address	Varchar(45)	1	0

#### 4.1.1.16 “magazine\_categories” Table

This table stores data about magazine categories. Informations in this table as follows:

Name	Type	Not Null	Primary Key
magazine_categorie_id	int(11)	Auto increment	1
magazine_categorie_name	Varchar(45)	1	0

- magazine\_categorie\_name : used for categorizing magazines. Ex : bilim, eğitim ,mühendislik

#### 4.1.1.17 “records” Table

This table stores data about sources and type of this sources. Informations in this table as follows:

Name	Type	Not Null	Primary Key
record_id	int(11)	Auto increment	1
type_id	int(11)	1	0
x_id	int(11)	1	0
Physical_electronic	bit(1)	1	0

- type\_id : type of record. Ex : book,DVD, magazine
- physical-electronic : record is physical or electronic

#### 4.1.1.18 “suggestions” Table

This table stores suggestions that are going to send to admin. Informations in this table as follows:

Name	Type	Not Null	Primary Key
suggestion_id	int(11)	1	1
suggestion	text	1	0
User_id	int(11)	1	0

#### 4.1.1.19 “types” Table

This table stores data about source type. Informations in this table as follows:

Name	Type	Not Null	Primary Key
type_id	int(11)	1	1
type_name	Varchar(45)	1	0

- type\_name : contains types. Ex: book, DVD, magazine

### 4.1.2 Views

In this chapter views are going to be displayed. Also usage way of views are going to be explained.

In source page this views are used.

#### 4.1.2.1 “book\_information” View

This view is used when just book is selected in search page.

Name	Type
record_id	int(11)
book_name	Varchar(45)
author_name	Varchar(45)
book_publication_year	Year(4)
book_ISBN	Varchar(45)
book_rental_count	int(11)
library_name	Varchar(45)
floor_name	Varchar(10)
bookcase_id	int(11)
bookshelf_id	int(11)
book_column	int(11)
physical_electronic	bit(1)
available	int(11)
url	Varchar(45)

#### 4.1.2.2 “DVD\_information” View

This view is used when just DVD is selected in searce page.

Name	Type
DVD_id	int(11)
DVD_name	Varchar(45)
physical_electronic	bit(1)
DVD_duration	int(11)
DVD_publication_date	date
DVD_rental_count	int(11)
available	int(7)

#### 4.1.2.3 “electronic\_books” View

This view is used when book and electronic is selected in searce page.

Name	Type
record_id	int(11)
book_name	Varchar(45)
author_name	Varchar(45)
book_publication_year	Year(4)
book_ISBN	Varchar(45)
book_rental_count	int(11)
physical_electronic	bit(1)
url	Varchar(45)
available	int(7)

#### 4.1.2.4 “electronic\_magazines” View

This view is used when magazine and electronic is selected in searce page.

Name	Type
record_id	int(11)
magazine_name	Varchar(45)
magazine_publication_date	date
magazine_issue_number	int(11)
magazine_rental_count	int(11)
physical_electronic	bit(1)
url	Varchar(45)
available	int(7)

#### 4.1.2.5 “magazine\_informations” View

This view is used when just magazine is selected in searce page.

Name	Type
record_id	int(11)
magazine_name	Varchar(45)
magazine_publication_date	date
magazine_issue_number	int(11)
magazine_rental_count	int(11)
library_name	Varchar(45)
floor_name	Varchar(10)
bookcase_id	int(11)
bookshelf_id	int(11)
book_column	int(11)
physical_electronic	bit(1)
Available	int(11)
url	Varchar(45)

#### 4.1.2.6 “physical\_magazines” View

This view is created for simplicity of magazine\_informations code.

Name	Type
record_id	Int(11)
magazine_name	Varchar(45)
magazine_publication_date	date
magazine_issue_number	Int(11)
magazine_rental_count	Int(11)
library_name	Varchar(45)
floor_name	Varchar(10)
bookcase_id	Int(11)
bookshelf_id	Int(11)
book_column	Int(11)
physical_electronic	bit(1)
Available	Int(11)

#### 4.1.2.7 “electronic\_magazines” View

This view is created for simplicity of magazine\_informations code.

Name	Type
record_id	Int(11)
magazine_name	Varchar(45)
magazine_publication_date	date
magazine_issue_number	Int(11)
magazine_rental_count	Int(11)
physical_electronic	bit(1)
url	Varchar(45)
available	Int(7)

#### 4.1.2.8 “physical\_books” View

This view is used when book and physical is selected in search page.

Name	Type
record_id	int(11)
book_name	Varchar(45)
author_name	Varchar(45)
book_publication_year	Year(4)
book_ISBN	Varchar(45)
book_rental_count	int(11)
library_name	Varchar(45)
floor_name	Varchar(10)
bookcase_id	int(11)
bookshelf_id	int(11)
book_column	int(11)
physical_electronic	bit(1)
Available	int(11)

#### 4.1.2.9 “physical\_magazines” View

This view is used when magazine and physical is selected in search page.

Name	Type
record_id	int(11)
magazine_name	Varchar(45)
magazine_publication_date	date
magazine_issue_number	int(11)
magazine_rental_count	int(11)
library_name	Varchar(45)
floor_name	Varchar(10)
bookcase_id	int(11)
bookshelf_id	int(11)
book_column	int(11)
physical_electronic	bit(1)
Available	int(11)

### 4.1.3 Foreign Keys

#### 4.1.3.1 In “bookscases” table

floor\_id references floor\_id in “floors” table

- On delete cascade
- On update cascade



#### 4.1.3.2 In “bookshelves” table

bookcase\_id references bookcase\_id in “bookcase” table.

- On delete cascade
- On update cascade

#### 4.1.3.3 In “electronic\_resources” table

record\_id references record\_id in “records” table.

- On delete cascade
- On update cascade

#### 4.1.3.4 In “floors” table

library\_id references library\_id in “libraries” table.

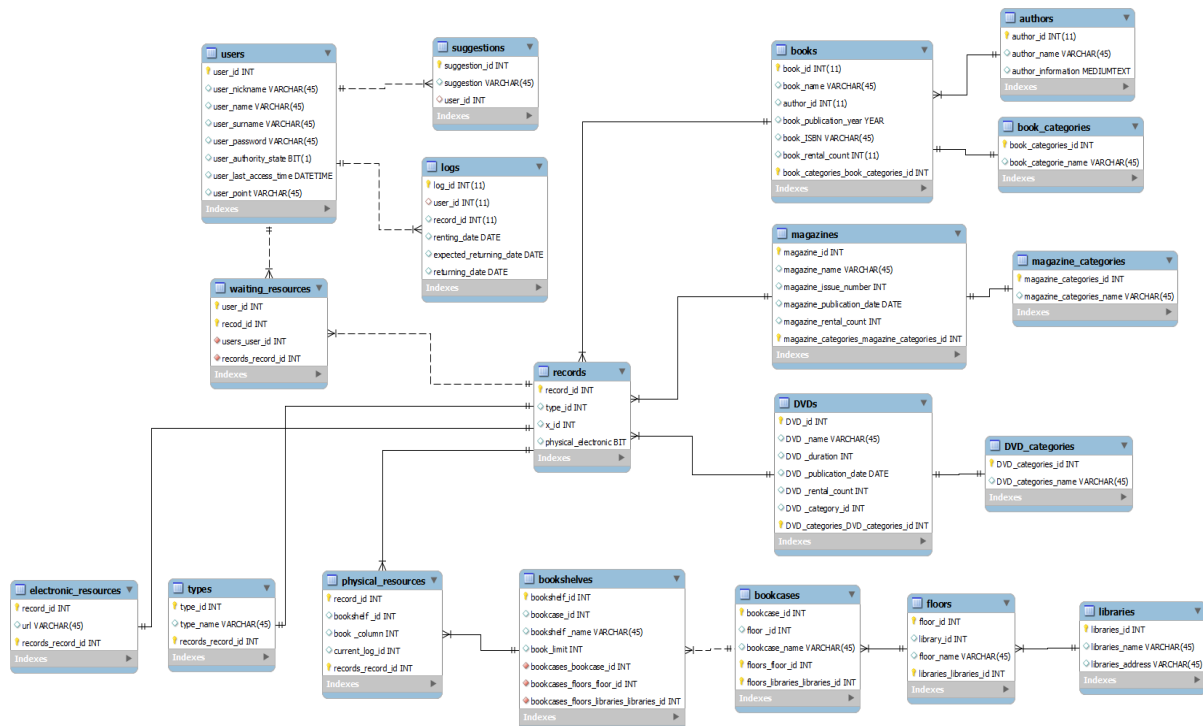
- On delete cascade
- On update cascade

#### 4.1.3.5 In “physical\_resources” table

record\_id references record\_id in “records” table.

- On delete cascade
- On update cascade

### 4.1.4 Entity Relationship Diagram



## 4.2 Software Design

### 4.2.1 Tugrul Yatagan's classes

#### 4.2.1.1 AddSourcePage.java

This page is used for adding new sources to databases. These sources can be book, DVD or magazine. Only the admin can access this page so admin navigation panel is embedded this page. In this page admin must choose the source type via their links for adding new source. Like all page classes this page also inherits *BasePage*.

This page calls *AddBookPage*, *AddDVDPage* and *AddMagazinePage* via

```
Link addBookPageLink = new Link("addbook")
Link addDVDPageLink = new Link("adddvd")
Link addMagazinePageLink = new Link("addmagazine")
```

These links respectively when user clicks one of these iconic buttons. After that these sources pages calls relative forms for them.

#### 4.2.1.2 Common.java

This class is mainly used for operations which are commonly used for all types for example adding source library. Taking physical library informations are same for all types like *bookcase*, *bookshelf*, *floor* and *library* informations. Also all records has type, log, *date*, *record name* and *record\_id* informations. All these common attributes combines in this class. *x\_id* keeps sources real id's. For example for book *x\_id* is equal to *book\_id*.

Declarations and data types of this class's variables are:

```

public class Common implements Serializable {
    private String _name = null;
    private Integer _x_id = null;
    private Integer _type_id = null;
    private Integer _physical_electronic = null;
    private String _library_name = null;
    private Integer _library_id = null;
    private String _floor_name = null;
    private Integer _floor_id = null;
    private String _bookcase_name = null;
    private Integer _bookcase_id = null;
    private String _bookshelf_name = null;
    private Integer _bookshelf_id = null;
    private Integer _column = null;
    private Integer _record_id = null;
    private Integer _current_log_id = null;
    private Date _renting_date = null;
    private Date _expected_returning_date;
    private String _url = null;
    private String _type_name = null;
}

```

All these variables has their own public getter and setter methods. Common class inherits Serializable class for serial operations so we can use these class as list of objects.

#### 4.2.1.3 DatabaseConnection.java

This class is bridge class for all database JDBC operations in this project. All other classes which uses database connections, must take connection object from this class and use them for database operations. This class handles constant database connection procedures like giving hostname, host port name, database name, database user name, database user password and database connection type parameters in its constructor so initial database connection can be provided. Insert, update and select operations have their own methods. These methods takes string type queries and returns database *ResultSet* objects. Class itself has three variables:

```

private Connection dbConnect = null;
private ResultSet resultTable = null;
private Statement statement = null;

```

*dbConnect* object supplies database connections and is closed in *close()* method. *resultTable* object is returned from *GetResult* and *Insert* methods to caller. *Statement* object executes string queries in *executeQuery* method of JDBC.

JDBC MySQL driver selection is done with:

```

try {
    Class.forName("com.mysql.jdbc.Driver");
} catch (ClassNotFoundException e) {
    throw new UnsupportedOperationException(e.getMessage());
}

```

JDBC MySQL database connection is provided. Database *host*, *port*, *name*, *user name*, *user password* and *encoding type* is specified in this block:

```

try {
    String dbHost = "24.133.151.17";
    Integer dbPort = 3306;
    String dbName = "elibrary";
    String dbUser = "dmbs";
    String dbPassword = "itudb1302";
    String dbUrl = String.format("jdbc:mysql://%s:%d/%s?useUnicode=true&characterEncoding=utf8",
                                dbHost, dbPort, dbName );
    this.dbConnect = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
    statement = this.dbConnect.createStatement();
} catch (SQLException ex) {
    throw new UnsupportedOperationException(ex.getMessage());
}

```

Constructor creates *statement* object for other methods query executions.

*GetResult* method takes select queries, executes them and returns *ResultSet* objects. This method is commonly used in whole projects, all select queries is done by this method. String type queries executed in *Statement* objects which is supplied by class's constructor. One *Statement* objects can be used multiple times until object's *close* method's execution.

```

public ResultSet GetResult(String query) throws Exception {
    try {
        resultTable = statement.executeQuery(query);
        return resultTable;
    } catch (SQLException e) {
        throw new UnsupportedOperationException(e.getMessage());
    } catch (Exception e) {
        throw e;
    }
}

```

*Insert* method takes insert queries and executes them. This method is commonly used in whole projects, all insert queries is done by this method. String type queries executed in *Statement* objects which is supplied by class's constructor. One *Statement* objects can be used multiple times until object's *close* method's execution.

```

public void Insert(String query) throws SQLException {
    try {
        statement.executeUpdate(query);
    } catch (SQLException e) {
        throw e;
    } catch (Exception e) {
        throw e;
    }
}

```

*close* method closed database connection object, database statement object, and *resultset*

object. After execution this method, object of this class and return value of class *resultset* object cannot be used.

```
public void close() throws Exception {
    try {
        if (resultTable != null) {
            resultTable.close();
        }
        if (statement != null) {
            statement.close();
        }
        if (dbConnect != null) {
            dbConnect.close();
        }
    } catch (Exception e) {
        throw e;
    }
}
```

*get\_books* method returns list of book object which is retrieved from *book\_informations* view in database. Getting book information is an operation which is commonly used in project so a function is written for getting all books information at one time. All book object's variables retrieved from database and setted to objects by setter methods as possible as. Common variables like *bookcase*, *bookshelf*, *floor*, *library*, *rental count*, *year*, *type* and specific variables to books like *book name*, *author*, *category*, *ISBN* are retrieved from database:

```

public List<Book> get_books(String book_query) throws Exception {
    ResultSet rs = this.GetResult(book_query);
    List<Book> booklist = new ArrayList();

    while (rs.next()) {
        Book tempBook = new Book();
        tempBook.set_record_id(rs.getInt("record_id"));
        tempBook.set_book_id(rs.getInt("book_id"));
        tempBook.set_name(rs.getString("book_name"));
        tempBook.set_author_name(rs.getString("author_name"));
        tempBook.set_category_name(rs.getString("book_category_name"));
        tempBook.set_publish_year(rs.getInt("book_publication_year"));
        tempBook.set_ISBN(rs.getString("book_ISBN"));
        tempBook.set_rental_count(rs.getInt("book_rental_count"));
        tempBook.set_library_name(rs.getString("library_name"));
        tempBook.set_floor_value(rs.getString("floor_name"));
        tempBook.set_bookcase_id(rs.getInt("bookcase_id"));
        tempBook.set_bookshelf_id(rs.getInt("bookshelf_id"));
        tempBook.set_bookcase_name(rs.getString("bookcase_name"));
        tempBook.set_bookshelf_name(rs.getString("bookshelf_name"));
        tempBook.set_column_id(rs.getInt("book_column"));
        tempBook.set_physical_electronic(rs.getInt("physical_electronic"));
        tempBook.set_available(rs.getInt("available"));
        tempBook.set_url(rs.getString("url"));
        booklist.add(tempBook);
    }
    return booklist;
}

```

Example select query for *get\_books* method in book page is:

```

DatabaseConnection dbc = new DatabaseConnection();
query = "SELECT * FROM book_informations WHERE book_id = " + book.get_book_id();
List<Book> list = dbc.get_books(query);
dbc.close();

```

*get\_dvds* method returns list of DVD object which is retrieved from *dvd\_informations* view in database. Getting dvd information is an operation which is commonly used in project so a function is written for getting all DVDs information at one time. All DVD object's variables retrieved from database and setted to objects by setter methods as possible as. Common variables like *bookcase*, *bookshelf*, *floor*, *library*, *rental count*, *year*, *type* and specific variables to DVDs like *DVD name*, *duration*, *category* are retrieved from database:

```

public List<Dvd> get_dvds(String dvd_query) throws Exception {
    ResultSet rs = this.GetResult(dvd_query);
    List<Dvd> dvdlist = new ArrayList();

    while (rs.next()) {
        Dvd dvd = new Dvd();
        dvd.set_record_id(rs.getInt("record_id"));
        dvd.set_DVD_id(rs.getInt("DVD_id"));
        dvd.set_name(rs.getString("DVD_name"));
        dvd.set_category_name(rs.getString("DVD_category_name"));
        dvd.set_publish_year(1900 +
            (rs.getDate("DVD_publication_date").getYear()));
        dvd.set_duration(rs.getInt("DVD_duration"));
        dvd.set_rental_count(rs.getInt("DVD_rental_count"));
        dvd.set_library_name(rs.getString("library_name"));
        dvd.set_floor_value(rs.getString("floor_name"));
        dvd.set_bookcase_id(rs.getInt("bookcase_id"));
        dvd.set_bookshelf_id(rs.getInt("bookshelf_id"));
        dvd.set_bookcase_name(rs.getString("bookcase_name"));
        dvd.set_bookshelf_name(rs.getString("bookshelf_name"));
        dvd.set_column_id((rs.getInt("DVD_column")));
        dvd.set_physical_electronic(rs.getInt("physical_electronic"));
        dvd.set_availability(rs.getInt("available"));
        dvd.set_url(rs.getString("url"));
        dvdlist.add(dvd);
    }
    return dvdlist;
}

```

Example select query for *get\_dvds* method in DVD page is:

```

DatabaseConnection dbc = new DatabaseConnection();
query = "SELECT * FROM DVD_informations WHERE DVD_id = " + Dvd.get_DVD_id();
List<Dvd> list = dbc.get_books(query);
dbc.close();

```

*get\_magazines* method returns list of magazine object which is retrieved from *magazines\_informations* view in database. Getting magazine information is an operation which is commonly used in project so a function is written for getting all magazines information at one time. All magazine object's variables retrieved from database and setted to objects by setter methods as possible as. Common variables like *bookcase*, *bookshelf*, *floor*, *library*, *rental count*, *year*, *type* and specific variables to books like *magazines name*, *issue number*, *category* are retrieved from database:

```

public List<Magazine> get_magazines(String magazine_query) throws Exception {
    ResultSet rs = this.GetResult(magazine_query);
    List<Magazine> magazinelist = new ArrayList();

    while (rs.next()) {
        Magazine magazine = new Magazine();
        magazine.set_record_id(rs.getInt("record_id"));
        magazine.set_magazine_id(rs.getInt("magazine_id"));
        magazine.set_name(rs.getString("magazine_name"));
        magazine.set_category_name(rs.getString("magazine_category_name"));
        magazine.set_publish_year(1900 +
            (rs.getDate("magazine_publication_date").getYear()));
        magazine.set_issue_number(rs.getInt("magazine_issue_number"));
        magazine.set_rental_count(rs.getInt("magazine_rental_count"));
        magazine.set_library_name(rs.getString("library_name"));
        magazine.set_floor_value(rs.getString("floor_name"));
        magazine.set_bookcase_id(rs.getInt("bookcase_id"));
        magazine.set_bookshelf_id(rs.getInt("bookshelf_id"));
        magazine.set_bookcase_name(rs.getString("bookcase_name"));
        magazine.set_bookshelf_name(rs.getString("bookshelf_name"));
        magazine.set_column_id(rs.getInt("book_column"));
        magazine.set_physical_electronic(rs.getInt("physical_electronic"));
        magazine.set_availability(rs.getInt("available"));
        magazine.set_url(rs.getString("url"));
        magazinelist.add(magazine);
    }
    return magazinelist;
}

```

Example select query for *get\_magazines* method in magazine page is:

```

DatabaseConnection dbc = new DatabaseConnection();
query = "SELECT * FROM magazine_informations WHERE magazine_id = " +
        magazine.get_magazine_id();
List<Magazine> list = dbc.get_magazines(query);
dbc.close();

```

#### 4.2.1.4 EditAdminProfilePage.java

This page is used for updating admin's profile informations like nickname, name, surname, and password. For updating information, admin has to enter his/her current password. Obviously only the admin can access this page so admin navigation panel is embedded to this page. After opening this page, it is redirected to common *EditProfilePage*. Like all page classes this page also inherits *BasePage*.

```

public void onSubmit() {
    this.setResponsePage(new EditProfilePage());
}

```

#### 4.2.1.5 EditProfilePage.java

This page is used for updating users profile informations like nickname, name, surname, and password. For updating information, user has to enter his/her current password. Users and admins can access this page so admin navigation panel or user navigation panel is embedded to



the page according to person's *authorityState*. After user/admin distinguish, a form object *UserEditPrfileForm* is called. Like all page classes this page also inherits *BasePage*.

```

if(((BasicAuthenticationSession) BasicAuthenticationSession.get()).
    getUser().get_authorityState() == 0)
{
    get("adminNavigation").setVisible(false);
}
else
{
    get("userNavigation").setVisible(false);
}
add(new UserEditProfileForm("user_edit_profile"));

```

#### 4.2.1.6 UserEditProfileForm.java

This form is used for updating users profile informations like nickname, name, surname, and password in database. For updating information, user has to enter his/her current password. These variables keeps in class as string and taken from user as *TextField*, or *PasswordField* component.

```

private TextField u_name;
private TextField u_lname;
private TextField u_uname;
private PasswordTextField currentpassword;
private PasswordTextField newpassword;
private PasswordTextField anewpassword;

```

Logged user's information is taken from session class:

```

String name = ((BasicAuthenticationSession) BasicAuthenticationSession.get())
    .getUser().getName();
String surname = ((BasicAuthenticationSession) BasicAuthenticationSession.get())
    .getUser().getSurname();
String username = ((BasicAuthenticationSession) BasicAuthenticationSession.get())
    .getUser().getUsername();

```

Firstly current user's *user\_id* is founded via a select query. Than according to user's input name or surname or nickname which was not null so only the valid inputs are updated on the database. To find a user's row, *user\_id* is enough for WHERE clause. Finally if user's actual current password is same as the input's current password, password information is updated on the database.

```

DatabaseConnection dbc = new DatabaseConnection();
ResultSet rs;
try {
    rs = dbc.GetResult(String.format(
        "select `user_id`,`user_password` from users where `user_nickname` = '%s';",
        ((BasicAuthenticationSession) BasicAuthenticationSession
            .get()).getUser().getUsername()));
    rs.next();
    int x = rs.getInt("user_id");
    String u_pass = rs.getString("user_password");
    if (u_name.getModelObject() != null) {dbc.Insert(String.format(
        "UPDATE users MODIFY SET `user_name` = '%s' where `user_id` = %d;",
        u_name.getModelObject(), x));
    }
    if (u_lname.getModelObject() != null) {dbc.Insert(String.format(
        "UPDATE users MODIFY SET `user_surname` = '%s' where `user_id` = %d;",
        u_lname.getModelObject(), x));
    }
    if (u_uname.getModelObject() != null) {dbc.Insert(String.format(
        "UPDATE users MODIFY SET `user_nickname` = '%s' where `user_id` = %d;",
        u_uname.getModelObject(), x));
    }
    String cur_pass = (String) currentpassword.getModelObject();
    if (cur_pass.equals(u_pass)
        && newpassword.getModelObject() != null
        && anewpassword.getModelObject() != null
        && newpassword.getModelObject().toString()
            .equals(anewpassword.getModelObject().toString())) {
        dbc.Insert(String.format(
            "UPDATE users MODIFY SET `user_password` = '%s' where `user_id` = %d;",
            newpassword.getModelObject(), x));
    }
} catch (Exception e) {
    e.printStackTrace();
}

```

And then new name and surname setted to session class for displaying them on main navigation bar in home page:

```

((BasicAuthenticationSession) BasicAuthenticationSession.get())
    .getUser().setName(u_name.getDefaultModelObjectAsString());
((BasicAuthenticationSession) BasicAuthenticationSession.get())
    .getUser().setSurname(u_lname.getDefaultModelObjectAsString());
this.setResponsePage(new HomePage());

```

#### 4.2.1.7 SourcePlacedPage.java

This page is used for adding source information to library like bookcase, bookshelf, floor and library for physical sources or URL for electronic sources. For adding library information user must be admin. Obviously only the admin can access this page so admin navigation panel is embedded to this page. After opening the page, it is redirected to *SourcePlacedForm*. Like all page classes this page also inherits *BasePage*. Class's constructor takes two arguments; first one is *Common* object for source which is placed in library and second is *checkUrl* bit for taking URL information for electronic sources or taking place information for physical sources.

```

public SourcePlacedPage(Common common, Boolean checkUrl) {
    this.add(new AdminNavigationPanel("adminNavigation"));
    this.add(new SourcePlacedForm("sourceplacedform", common, checkUrl));
}

```

#### 4.2.1.8 SourcePlacedForm.java

This form is used for taking source information and adding to library like bookcase, bookshelf, floor and library for physical sources or URL for electronic sources. Class's constructor takes two arguments; first one is *Common* object for source which is used to place in library and second is *checkUrl* bit for taking URL information for electronic sources or taking place information for physical sources.

There is 4 dropdown choices in this page. AJAX is used for dropdown choices. *SelectedMake* string variables keeps choice of the user and *modelsMap* map variables connects one choice to another. When user selects top dropdown choice, second dropdown choice is restricted according to user's selection and it continuous to the last dropdown choices. Every selection restricts next dropdown choice's selection.

```

private String selectedMake;
private String selectedMake2;
private String selectedMake3;
private String selectedMake4;
private final Map<String, List<String>> modelsMap = new HashMap<String, List<String>>();
private final Map<String, List<String>> modelsMap2 = new HashMap<String, List<String>>();
private final Map<String, List<String>> modelsMap3 = new HashMap<String, List<String>>();

```

Labels and fields of this pages are URL, column, library, floor, bookcase and bookshelf. URL and column information is taken from user by manual in text or number fields.

```

TextField url = new TextField("_url");
NumberTextField cloumn = New NumberTextField("_column").setRequired(false);

Label librarylabel = new Label("librarylabel", "Library Name: ");
Label floorlabel = new Label("floorlabel", "Library Floor: ");
Label bookcaselabel = new Label("bookcaselabel", "Bookcase: ");
Label bookshelflabel = new Label("bookshelflabel", "Bookshelf: ");
Label columnlabel = new Label("columnlabel", "Column(1-50): ");
Label urllabel = new Label("urllabel", "URL: ");

```

During dropdown choice restriction all information about place is retrieved from database. All selections are mapped and listed next selections so user can be select dropdown choices by one by. Retrieving place information from database is done by following code block:

```

try {
    DatabaseConnection dbclib = new DatabaseConnection();
    String query = "SELECT * FROM libraries;";
    ResultSet lib = dbclib.GetResult(query);
    while (lib.next()) {
        DatabaseConnection dbcfloor = new DatabaseConnection();
        query = String.format(
            "SELECT * FROM floors WHERE library_id = %d;",
            lib.getInt("library_id"));
        ResultSet floor = dbcfloor.GetResult(query);
        List<String> floorList = new ArrayList<String>();
        while (floor.next()) {
            DatabaseConnection dbccase = new DatabaseConnection();
            query = String.format(
                "SELECT * FROM bookcases WHERE floor_id = %d;",
                floor.getInt("floor_id"));
            ResultSet cases = dbccase.GetResult(query);
            List<String> casesList = new ArrayList<String>();
            while (cases.next()) {
                DatabaseConnection dbcshelf = new DatabaseConnection();
                query = String.format(
                    "SELECT * FROM bookshelves WHERE bookcase_id = %d;",
                    cases.getInt("bookcase_id"));
                ResultSet shelf = dbcshelf.GetResult(query);
                List<String> shelfList = new ArrayList<String>();
                while (shelf.next()) {
                    shelfList.add(shelf.getString("bookshelf_name"));
                }
                dbcshelf.close();

                casesList.add(cases.getString("bookcase_name"));
                modelsMap3.put(cases.getString("bookcase_name"), shelfList);
            }
            dbccase.close();

            floorList.add(floor.getString("floor_name"));
            modelsMap2.put(floor.getString("floor_name"), casesList);
        }
        dbcfloor.close();

        modelsMap.put(lib.getString("library_name"), floorList);
    }
    dbclib.close();
} catch (SQLException e) {
    e.printStackTrace();
}

```

When user clicks Placed Source icon, according to type of the record (electronic or physical) source will be added to library. Following code block inserts physical record Into *records* table and retrieves record, library, floor, bookcase and bookshelf id's from database then inserts this id informations Into *physical\_resources* tables. All these id's connects on *physical\_resources* table like this:

```

if (c.get_library_name() != null && c.get_floor_name() != null
    && c.get_bookcase_name() != null
    && c.get_bookshelf_name() != null) { // fiziksel
    dbc = new DatabaseConnection();
    query = String.format(
        "INSERT INTO records (`type_id`, `x_id`, `physical-electronic`)
        VALUES ('%d', '%d', %s);", c.get_type_id(), c.get_x_id(), "True");
    dbc.Insert(query);

    dbc = new DatabaseConnection();
    query = "SELECT MAX(record_id) FROM records;";
    ResultSet result = dbc.GetResult(query);
    result.next();
    c.set_record_id(result.getInt(1));

    dbc = new DatabaseConnection();
    query = String.format(
        "SELECT library_id FROM libraries WHERE library_name = '%s';",
        c.get_library_name());
    result = dbc.GetResult(query);
    result.next();
    c.set_library_id(result.getInt(1));

    dbc = new DatabaseConnection();
    query = String.format(
        "SELECT floor_id FROM floors WHERE (floor_name = '%s')
        AND (library_id = '%d');", c.get_floor_name(), c.get_library_id());
    result = dbc.GetResult(query);
    result.next();
    c.set_floor_id(result.getInt(1));

    dbc = new DatabaseConnection();
    query = String.format(
        "SELECT bookcase_id FROM bookcases WHERE (bookcase_name = '%s')
        AND (floor_id = '%d');", c.get_bookcase_name(), c.get_floor_id());
    result = dbc.GetResult(query);
    result.next();
    c.set_bookcase_id(result.getInt(1));

    dbc = new DatabaseConnection();
    query = String.format(
        "SELECT bookshelf_id FROM bookshelves WHERE (bookshelf_name = '%s')
        AND (bookcase_id = '%d');", c.get_bookshelf_name(), c.get_bookcase_id());
    result = dbc.GetResult(query);
    result.next();
    c.set_bookshelf_id(result.getInt(1));

    dbc = new DatabaseConnection();
    query = String.format(
        "INSERT INTO physical_resources (record_id, bookshelf_id, book_column)
        VALUES ('%d', '%d', '%d');",
        c.get_record_id(), c.get_bookshelf_id(), c.get_column());
    dbc.Insert(query);
}

```

When user clicks Placed Source icon, according to type of the record (electronic or physical) source will be added to library. Following code block inserts electronic record Into *records* table and retrieves record id's from database then inserts this id information Into *electronic\_resources* tables. All these id's connects on *electronic\_resources* table like this:

```

if (c.get_url() != null) { // elektronik
    dbc = new DatabaseConnection();
    query = String.format(
        "INSERT INTO records (`type_id`, `x_id`, `physical-electronic`)
        VALUES ('%d', '%d', %s);", c.get_type_id(), c.get_x_id(), "False");
    dbc.Insert(query);

    dbc = new DatabaseConnection();
    query = "SELECT MAX(record_id) FROM records;";
    ResultSet result = dbc.GetResult(query);
    result.next();
    c.set_record_id(result.getInt(1));

    dbc = new DatabaseConnection();
    query = String.format(
        "INSERT INTO electronic_resources (`record_id`, `url`)
        VALUES ('%d', '%s');", c.get_record_id(), c.get_url());
    dbc.Insert(query);
}

```

If a conflict occurs while adding record to *records* table, a *DuplicateKeyException* exception is thrown. *physical\_resources* table has unique key constraint on its *record\_id*, *bookshelf\_id*, *book\_column* columns and *electronic\_resources* table has unique key constraint on its *record\_id*, *url* columns. If a conflict occurs *record\_id* column in *records* table must be deleted. Following catch statement handles this duplicate key exception:

```

catch (Exception DuplicateKeyException) {
    duplicate_error = "Location is taken.
        Record cannot be placed. Please enter another location.";
    dbc = new DatabaseConnection();
    query = String.format("SELECT COUNT(*) FROM physical_resources
        WHERE record_id = '%d';", c.get_record_id());
    ResultSet result = dbc.GetResult(query);
    result.next();
    if (result.getInt(1) > 0) {
        dbc = new DatabaseConnection();
        query = String.format("DELETE FROM records
            WHERE record_id = '%d';", c.get_record_id());
        dbc.Insert(query);
    }
}

```

After all adding source to place operations user is redirected to sources page according to source type. Redirection need source's type and id. Following code makes this redirection:

```

if (common.get_type_id() == 1) { // Book
    Book b = new Book();
    b.set_book_id(c.get_x_id());
    this.setResponsePage(new BookPage(b, duplicate_error));
} else if (common.get_type_id() == 2) { // Magazine
    Magazine m = new Magazine();
    m.set_magazine_id(c.get_x_id());
    this.setResponsePage(new MagazinePage(m, duplicate_error));
} else if (common.get_type_id() == 3) { // DVD
    Dvd d = new Dvd();
    d.set_DVD_id(c.get_x_id());
    this.setResponsePage(new DvdPage(d, duplicate_error));
} else {
    this.setResponsePage(new HomePage());
}

```

#### 4.2.1.9 SourceEditPage.java

This page is used for updating source information in library like bookcase, bookshelf, floor and library for physical sources or URL for electronic sources. For updating library information user must be admin. Obviously only the admin can access this page so admin navigation panel is embedded to this page. After opening the page, it is redirected to *SourceEditForm*. Like all page classes this page also inherits *BasePage*. Class's constructor takes an argument, *Common* object for source which is updated in library information like URL for electronic sources or taking place information for physical sources.

```

public SourceEditPage (Common common) {
    this.add(new AdminNavigationPanel("adminNavigation"));
    this.add(new SourcePlacedForm("sourceplacedform", common));
}

```

#### 4.2.1.10 SourceEditForm.java

This form is used for editing source information and updating in library like bookcase, bookshelf, floor and library for physical sources or URL for electronic sources. Class's constructor takes an argument *Common* object for source which is used to place in library and it takes URL information for electronic sources or takes place information for physical sources.

There is 4 dropdown choices in this page like *SourcePlacedForm*. AJAX is used for dropdown choices. *SelectedMake* string variables keeps choice of the user and *modelsMap* map variables connects one choice to another. When user selects top dropdown choice, second dropdown choice is restricted according to user's selection and it continuous to the last dropdown choices. Every selection restricts next dropdown choice's selection.

This page is almost same as *SourcePlacedForm*, only differences between *SourcePlacedForm* and *SourceEditForm* is default variables. For editing source current source information has to be inserted textboxes and dropdown choices. Embedding initial default values into form is made by following code blocks all other codes are same as the *SourcePlacedForm* so same codes are not explained again, only the different code blocks are explained:

```

column = new NumberTextField("_column", Model.of(this.common.get_column()));
url = new TextField("_url", Model.of(common.get_url()));

selectedMake = this.common.get_library_name();
selectedMake2 = this.common.get_floor_name();
selectedMake3 = this.common.get_bookcase_name();
selectedMake4 = this.common.get_bookshelf_name();

```

Giving default values to string *SelectedMake* variables is enough for embedding default values.  
Giving *Model* objects to field are also enough for embedding default values.

#### 4.2.1.11 Start.java

This class is main class of the project. Project starts from this class. In this class mainly server settings are done like port number, timeout and debug mode selection.

```

int timeout = (int) Duration.ONE_HOUR.getMilliseconds();
Server server = new Server();
SocketConnector connector = new SocketConnector();

connector.setMaxIdleTime(timeout);
connector.setSoLingerTime(-1);
connector.setPort(8081);

```

When application is start a starting message is printed to console and then server is started.  
When developer enters something on console, a stopping message is printed to console and then server is stopped. When server is stopped application should be stopped either.

```

System.out.println(">>> STARTING EMBEDDED JETTY SERVER, PRESS ANY KEY TO STOP");
server.start();
System.in.read();
System.out.println(">>> STOPPING EMBEDDED JETTY SERVER");
server.stop();

```

#### 4.2.1.12 WicketApplication.java

This class provides integration between wicket application and project. This class mostly remained untouched, changes on this class for session class adjustment. The class inherits *AuthenticatedWebApplication* class for session operation by this Session class can control whole project. Sign in page is overridden for session log in operations. Finally *WicketApplication* class's *init* method is called.



```

public class WicketApplication extends AuthenticatedWebApplication {
    @Override
    public Class<HomePage> getHomePage() {
        return HomePage.class;
    }
    @Override
    protected Class<? extends AbstractAuthenticatedWebSession> getSessionClass() {
        return BasicAuthenticationSession.class;
    }
    @Override
    protected Class<? extends WebPage> getSignInPageClass() {
        return SignInPage.class;
    }
    @Override
    public void init() {
        super.init();
    }
}

```

## 4.2.2 Emre Gökrem's classes

### 4.2.2.1 AddBookPage.java

This page is seen by only admins. Main process of this page is that new book sources are added by admin to library. But this process doesn't provide a new material. Because this process provides only new sources information. After this page, new material of current source is added in "SourcePlacedPage.java". Admin navigation and base navigation are used in this page and needing processes are done by "AddBookForm.java".

### 4.2.2.2 AddBookForm.java

This form depends to "AddBookPage.java". It is initialized by "AddBookPage.java". Admin adds new source thanks to this form. In this form, needing textfields and numberfields are generating. In addition, category informations about dropdown is pulled from database.

Thanks to this query:

```

DatabaseConnection dbc = new DatabaseConnection();
String query = "SELECT book_category_name FROM book_categories ORDER BY book_category_name;";
ResultSet cat = dbc.GetResult(query);

```

Pulling book category from database

When admin clicks add button, informations about new book source is transferred to book object from textfields and numberfields variables. Then, creating book object is inserted to database. Books and authors tables are changed with this process. Thanks to these queries:

```

query = String.format(
    "INSERT INTO authors (author_name) VALUES ('%s');",
    book.get_author_name());
dbc.Insert(query);

```

### Inserting new author

```

dbc = new DatabaseConnection();
query = String
    .format("INSERT INTO books (book_name, author_id, book_publication_year, book_ISBN, book_category_id) VALUES ('%s', '%d', '%d', '%s', '%d');",
        book.get_name(), author_id,
        book.get_publish_year(), book.get_ISBN(),
        book.get_category_id());

dbc.Insert(query);

```

### Inserting new book source

In addition, current book or author that already exist in database, not inserted. This situations are controlled thanks to these queries:

```

dbc = new DatabaseConnection();
query = String.format(
    "SELECT author_id FROM authors WHERE author_name = '%s';",
    book.get_author_name());
ResultSet result = dbc.GetResult(query);
result.next();
Integer author_id = result.getInt(1);

```

### Controlling author

```

dbc = new DatabaseConnection();
query = String.format("SELECT book_id FROM books WHERE book_ISBN = '%s';", book.get_ISBN());
ResultSet result = dbc.GetResult(query);
result.next();
book.set_book_id(result.getInt(1)); // her türlü book_id bul
dbc.close();

```

### Controlling book source

#### 4.2.2.3 AdminNavigationPanel.java

In this class, panel objects for admin pages are described and their connections with pages are created. Sections of admin navigation panel are home page link, admin profile page link, admin edit profile page link, add source page link and messages page link.

#### 4.2.2.4 ArchivePage.java

This page is accessed by current user. User sees informations (renting time, expected returning time and source common information with common object) of last 15 renting process in this page. We select logs record from logs table in database with user id. For each object (book,

DVD, magazine), different query is run on database. Thanks to this queries, renting informations are pulled from database:

```
String query = String
    .format("SELECT x_id, type_id, renting_date, expected_returning_date "
        + "FROM logs JOIN records ON (records.record_id = logs.record_id) WHERE user_id = %d "
        + "ORDER BY log_id DESC LIMIT 15", person_id);
ResultSet rs = dbc.GetResult(query);
```

Pulling records from database

```
query = String
    .format("SELECT book_name FROM books WHERE book_id = '%d';",
        log.get_x_id());
ResultSet rsource = dbcsource.GetResult(query);
rsource.next();
log.set_name(rsource.getString("book_name"));
log.set_type_name("Book");
```

Pulling books from database

```
query = String.format(
    "SELECT DVD_name FROM DVDs WHERE DVD_id = '%d';",
    log.get_x_id());
ResultSet rsource = dbcsource.GetResult(query);
rsource.next();
log.set_name(rsource.getString("DVD_name"));
log.set_type_name("DVD");
```

Pulling DVDs from database

```
query = String
    .format("SELECT magazine_name FROM magazines WHERE magazine_id = '%d';",
        log.get_x_id());
ResultSet rsource = dbcsource.GetResult(query);
rsource.next();
log.set_name(rsource.getString("magazine_name"));
log.set_type_name("Magazine");
```

Pulling magazines from database

#### 4.2.2.5 BasePage.java

This class is a navigation that base navigation for all website pages. We call main navigation thanks to this class.

```
this.add(new NavigationPanel("mainNavigation"));
```

Adding main navigation

#### 4.2.2.6 BookEditPage.java

This page is accessed by only admin from book page. This class includes main and admin navigation. In addition, book edit form provides needing process for this page.

```
this.add(adminNavigation);
add(new BookEditForm("book_edit_profile", book));
}
```

## Adding book edit form

### 4.2.2.7 BookEditForm.java

This form mainly provides edits informations of a book object. Firstly, numberfield and textfields are creating for providing html connections. Then, book informations are pulled from book object and written to fields on html. For category editing dropdown, category types are pulled from database thanks to this code:

```
DatabaseConnection dbc = new DatabaseConnection();
String query = "SELECT book_category_name FROM book_categories ORDER BY book_category_name;";
ResultSet cat = dbc.GetResult(query);
```

### Pulling book categories from database

When admin clicks edit button, book object informations are updated on database. In addition author name and book name is controlled for no adding again same author or book. Database process is done with these codes:

```
query = String.format(
    "INSERT INTO authors (author_name) VALUES ('%s');",
    book.get_author_name());
dbc.Insert(query);

dbc = new DatabaseConnection();
query = String.format(
    "SELECT author_id FROM authors WHERE author_name = '%s';",
    book.get_author_name());
ResultSet result = dbc.GetResult(query);
result.next();
Integer author_id = result.getInt(1);
```

### Inserting and controlling author

```
dbc = new DatabaseConnection();
query = String
    .format("SELECT book_category_id FROM book_categories WHERE book_category_name = '%s';",
        book.get_category_name());
result = dbc.GetResult(query);
result.next();
Integer category_id = result.getInt(1);
book.set_category_id(category_id);
```

### Controlling category name

```

dbc = new DatabaseConnection();
query = String
    .format("UPDATE books SET book_name = '%s', author_id = '%d', book_publication_year = '%d', " +
        " book_ISBN = '%s', book_category_id = %d WHERE book_id = %d;",
        book.get_name(), author_id,
        book.get_publish_year(), book.get_ISBN(),
        book.get_category_id(), book.get_book_id());

dbc.Insert(query);

```

Updating book source

#### 4.2.2.8 Book.java

In this class, book object is implemented. Attributes of book objects are determined and added setter getter methods for these attributes. This class was used for all book process which is writing information from database and pushing information to database. Attributes of book object is:

```

private String _name = null;
private String _author_name = null;
private Integer _publish_year = null;
private String _ISBN = null;
private Integer _rental_count = 0;
final private Integer _type_id = 1;
private Integer _category_id = null;
private String _category_name = null;
private Integer _physical_electronic = null;
private String _library_name = null;
private String _floor_value = null;
private Integer _bookcase_id = null;
private Integer _bookshelf_id = null;
private String _bookcase_name = null;
private String _bookshelf_name = null;
private Integer _column_id = null;
private Integer _available = null;
private String _url = null;
private Integer _record_id = null;
private Integer _book_id = null;

```

#### 4.2.2.9 BookPage.java

This page is an information page for current book source. Users, admins or guests see book common information (book name, category, author name, publication year, rental count and ISBN number). In addition, they can see material' table depends of current book source. In this table, material URL (for electronic material) and place informations (which library, floor, bookcase, bookshelf and column). In addition, this page has links for accessing to book edit page (for admin), add material page (for admin) and a few buttons for processes which deleting source (for admin), deleting material (for

admin), editing material (for admin), renting material (for user) and waiting material (for user). These processes are done with these codes:

```
String query = String
    .format("SELECT book_name, author_name, book_publication_year, book_ISBN, "
        + "book_rental_count, book_category_name FROM books "
        + "JOIN authors ON (books.author_id = authors.author_id) "
        + "JOIN book_categories ON (books.book_category_id = book_categories.book_category_id) "
        + "WHERE book_id = %d;", book.get_book_id());

ResultSet rs = dbc.GetResult(query);
```

### Pulling book informations

```
dbc.Insert(String.format(
    "DELETE FROM books where `book_id` = %d",
    book.get_book_id()));

dbc.Insert(String
    .format("DELETE FROM records where type_id = 1 and `x_id` = %d",
    book.get_book_id()));
```

### Deleting book source

```
query = "SELECT * FROM elibrary.book_informations where book_id = "
    + book.get_book_id();
List<Book> list = dbc.get_books(query);
```

### Pulling materials of current book source

```
dbc.Insert(String
    .format("DELETE FROM records where `record_id` = %d",
    book.get_record_id()));
```

### Deleting material of current book source

```
dbc.Insert(String
    .format("INSERT INTO waiting_resources (`user_id`,`record_id`) values (%d,%d);",
    x, book.get_record_id()));
```

### Inserting new materials to current book source

```
dbc.Insert(String
    .format("INSERT INTO logs (`user_id`,`record_id`,`renting_date`,`expected_returning_date`) "
        + " values (%d,%d,current_date(),DATE_ADD(current_date(),INTERVAL 30 DAY));",
    x, book.get_record_id()));

dbc.Insert(String
    .format("UPDATE physical_resources MODIFY SET `current_log_id` = (SELECT max(log_id) FROM logs) "
        + " where record_id = %d;", book.get_record_id()));

dbc.Insert(String
    .format("UPDATE books MODIFY SET `book_rental_count` = `book_rental_count` + 1 "
        + " where `book_id` = (SELECT `x_id` from records where `record_id`= %d)",
    book.get_record_id()));

dbc.close();
```

### Editing and adding book changes

#### 4.2.2.10 GuestNavigationPanel.java

In this class, panel objects for all guest pages are described and their connections with pages are created. Sections of guest navigation panel are home page link, most popular page link, contact page link, about us page link and requests suggestions page link.

#### 4.2.2.11 HomePage.java

This page has changeable navigation panels and search process. Main navigation is stable but guest, admin and user navigation is determined according to authority of current user. In addition, search process is done in this page. Needing operation is performed by search form.

#### 4.2.2.12 NavigationPanel.java

This navigation depends to main navigation and includes main panel objects (login page link, logout page link, register page link and home page link). Visibility of login link, register page link and logout link changes according to cases. Home page link is stable.

#### 4.2.2.13 PopularSourcesPage.java

This class and page are accessed by users and guests, admins don't accesses this class. This class mainly provide to list most popular 5 sources from each type (book, DVD, magazine). This list is determined according to rental count. Then sources are pulled from database. In addition, each source has own source page link in table. These processes are done with these queries:

```
// BOOK
DatabaseConnection dbc = new DatabaseConnection();
String query = "SELECT book_id, book_name, author_name, book_category_name, book_publication_year, book_rental_count "
    + "FROM books JOIN authors ON (books.author_id = authors.author_id) "
    + "JOIN book_categories ON (books.book_category_id = book_categories.book_category_id) "
    + "ORDER BY book_rental_count DESC LIMIT 5;";
ResultSet rs = dbc.GetResult(query);
List<Book> booklist = new ArrayList();
```

Pulling books from database

```
// DVD
dbc = new DatabaseConnection();
query = "SELECT DVD_id, DVD_name, DVD_category_name, DVD_duration, DVD_publication_date, DVD_rental_count "
    + "FROM DVDs "
    + "JOIN DVD_categories ON (DVDs.DVD_category_id = DVD_categories.DVD_category_id) "
    + "ORDER BY DVD_rental_count DESC LIMIT 5;";

rs = dbc.GetResult(query);
List<Dvd> dvdlist = new ArrayList();
```

Pulling DVDs

```
// MAGAZINE
dbc = new DatabaseConnection();
query = "SELECT magazine_id, magazine_name, magazine_category_name, magazine_issue_number, magazine_publication_date, magazine_rental_count "
      + "FROM magazines "
      + " JOIN magazine_categories ON (magazines.magazine_category_id = magazine_categories.magazine_category_id) "
      + " ORDER BY magazine_rental_count DESC LIMIT 5;";
rs = dbc.GetResult(query);
List<Magazine> magazinelist = new ArrayList();
```

Pulling magazines

#### 4.2.2.14 RequestsSuggestionsPage.java

This class and page are accessed by users and guests, admins don't access this class. This class mainly provide to send any suggestions or requests to admins. These messages are inserted all admins this process is done thanks to these codes:

```
if (((BasicAuthenticationSession) BasicAuthenticationSession
    .get()).isSignedIn()) {
    final Integer user_id = ((BasicAuthenticationSession) BasicAuthenticationSession
        .get()).getUser().get_id();

    str = String
        .format("INSERT INTO suggestions (`suggestion`,`user_id`) VALUES ('%s',%d)",
            input, user_id);
} else {
    str = String
        .format("INSERT INTO suggestions (`suggestion`) VALUES ('%s')",
            input);
}
try {
```

Inserting new suggestions

#### 4.2.2.15 ShowSuggestionsPage.java

This class and page are accessed by only admins, guests are users don't access this class. This class mainly provide to show suggestions and requests for admins. Admins control these messages and if they want, can delete messages. Messages are pulled from database and listed. If deleting process is applied, database are updated so messages are deleted from database. These processes are done with these codes:

```
String query = "SELECT suggestion_id, suggestion, user_name, user_surname FROM "
      + "suggestions LEFT JOIN users ON suggestions.user_id = users.user_id;";
ResultSet rs = dbc.GetResult(query);
```



```

DatabaseConnection dbcreq = new DatabaseConnection();
String query = String
    .format("DELETE FROM suggestions WHERE suggestion_id = %d;",
        req.get_suggestion_id());
dbcreq.Insert(query);

```

Pulling suggestions

#### 4.2.2.16 UserNavigationPanel.java

In this class, panel objects for all user pages are described and their connections with pages are created. Sections of user navigation panel are home page link, most popular page link, user edit page link, user profile page link and archive page link.

### 4.2.3 Mustafa Uçar's classes

#### 4.2.3.1 AddMagazineForm.java

This class is inherited from *Form* class. Class has three fields, *magazine*, *common* and *selected*. *magazine* will be used to store the data that is taken from form components. These components are, one TextField for *\_name* field of *magazine* object, two NumberTextField for *\_publish\_year* and *\_issue\_number* fields and one DropDownChoice for *\_category\_id*.

```

final List categoryList = new ArrayList();
    try {
        DatabaseConnection dbc = new DatabaseConnection();
        String query = "SELECT magazine_category_name FROM
magazine_categories ORDER BY magazine_category_name;";
        ResultSet cat = dbc.GetResult(query);
        while (cat.next()) {
            categoryList.add(cat.getString("magazine_category_name"));
        }
    }

```

This code block creates a list for DropDownChoice. It takes category names from *magazine\_category* table in database.

In onSubmit function;

First, component values are inserted to magazine fields with *getModelObject()* function.

Code block that executes necessary queries to add a magazine to the database;

```

query = String
    .format("SELECT magazine_category_id FROM magazine_categories WHERE
magazine_category_name = '%s';",
        selected);
ResultSet result = dbc.GetResult(query);
result.next();
Integer cetagory_id = result.getInt(1);
magazine.set_category_id(cetagory_id);

dbc = new DatabaseConnection();
query = String
    .format("INSERT INTO magazines (magazine_name, magazine_issue_number,
magazine_publication_date, magazine_category_id) VALUES ('%s', '%d', '%d-01-01',
'%d');",
        magazine.get_name(), magazine.get_issue_number(),
        magazine.get_publish_year(),
        magazine.get_category_id());

dbc.Insert(query);

```

Firstly, *category\_id* is taken from database by use of *selected* field, which is the value of drop down choice of the user. With *category\_id* value, all necessary magazine fields are fully filled. Then all fields of magazine object are inserted to *magazines* table of the database. *DuplicateKeyException* is caught to prevent duplicated insert operation to *magazines* table.

```

dbc = new DatabaseConnection();
query = String
    .format("SELECT magazine_id FROM magazines WHERE ((magazine_name = '%s')
AND (magazine_publication_date = '%d-01-01') AND (magazine_issue_number = '%d'));",
        magazine.get_name(),
        magazine.get_publish_year(),
        magazine.get_issue_number());
ResultSet result = dbc.GetResult(query);
result.next();
magazine.set_magazine_id(result.getInt(1));
dbc.close();

```

And then *magazine\_id* is taken from *magazines* table, while *magazine\_name* is equal to the form *TextField* value. By *magazine\_id*, *type\_id* and *name* fields of magazine object, fields of *common* object are filled. Then, this *common* object is send to the *SourcePlacedPage* to add record for this magazine.

#### 4.2.3.2 AddMagazinePage.java

This class is inherited from *BasePage* class. It simply has an *AddMagazineForm* and an *AdminNavigationPanel*.

#### 4.2.3.3 BasicAuthenticationSession.java

This class is inherited from *AuthenticationWebSession*. It simply does sign in and sign out operations. A *User* object is added to this class, for controlling page navigation and other operations on the website. *Authenticate* function takes *username* and *password* as a string value from login page, and compares these strings with database records.

```
String query = String.format(
    "SELECT * FROM users where user_nickname = '%s'", username);
rs = dbc.GetResult(query);
while (rs.next()) {
    tempUser = new User(rs.getString("user_name"),
        rs.getString("user_surname"),
        rs.getString("user_nickname"),
        rs.getString("user_password"),
        rs.getInt("user_authority_state"),
        rs.getInt("user_point"));
    tempUser.set_id(rs.getInt("user_id"));
    tempUser.set_last_access_time(rs.getTimestamp("user_last_access_time"));
}
```

Code block above, creates a temporary *User* for comparison.

```
if (tempUser.getUsername().equals(username)
    && tempUser.getPassword().equals(hash_password)) {
    check = true;
    query = String.format("UPDATE users SET `user_last_access_time` = NOW() WHERE
user_id = '%d'", tempUser.get_id());
    dbc.Insert(query);
} else {
    check = false;
    user.setUsername("");
    user.setName("");
    user.setSurname("");
    user.set_authorityState(0);
}
```

And this part of the code, if both comparisons are true, function returns *check* object as a true value and updates *user\_last\_access\_time* in *users* table. Otherwise, *check* will be false and fields of *user* will be empty.

#### 4.2.3.4 MagazineEditForm.java

This class is inherited from *Form* class. It's used in *MagazineEditPage* to edit a magazine's detail. Only admins can access this page. In addition to string *id*, this form takes a *magazine* object with its constructor. Fields of this magazine object is used to fill components as default vales. Admins will see magazine's current details in these *TextField*, *NumberTextField*, *DropDownChoice* etc. And then, similar algorithm with *AddMagazineForm* is used to update. Instead of insert query, update query is used this time.

#### 4.2.3.5 MagazineEditPage.java

This class is inherited from *BasePage*. It simply has a *MagazineEditForm* and an *AdminNavigationPanel*.

#### 4.2.3.6 Magazine.java

*Magazine* class is created to implement magazine objects in real life and to use datas in *magazine* table. Fields of this class are;

```

private String _name = null;
private Integer _rental_count = 0;
private Integer _publish_year = null;
final private Integer _type_id = 2;
private Integer _issue_number = null;
private Integer _availability = null;
private String _library_name = null;
private String _floor_value = null;
private Integer _bookcase_id = null;
private Integer _bookshelf_id = null;
private String _bookcase_name = null;
private String _bookshelf_name = null;
private Integer _column_id = null;
private String _url = null;
private Integer _physical_electronic = null;
private Integer _magazine_id = null;
private Integer _record_id = null;
private Integer _category_id = null;
private String _category_name = null;

```

These field names represents column names of tables which are associated with magazine, like *magazines* table, *records* table, *magazine\_categories* table.

#### 4.2.3.7 MagazinePage.java

This class is inherited from *BasePage* class. Its constructor takes a *Magazine* object. This object is send by *SearchPage*'s magazine link. *SearchPage* sends that object with only *magazine\_id* value. In constructor;

```

String query = String
.format("SELECT magazine_name, magazine_publication_date, magazine_issue_number, "
      + "magazine_rental_count, magazine_category_name FROM magazines "
      + "JOIN magazine_categories ON (magazines.magazine_category_id = "
magazine_categories.magazine_category_id) "
      + "WHERE magazine_id = %d;", magazine.get_magazine_id());
ResultSet rs = dbc.GetResult(query);
rs.next();
magazine.set_name(rs.getString("magazine_name"));
magazine.set_category_name(rs.getString("magazine_category_name"));
magazine.set_publish_year(1900 + (rs
.getDate("magazine_publication_date")).getYear());
magazine.set_issue_number(rs.getInt("magazine_issue_number"));
magazine.set_rental_count(rs.getInt("magazine_rental_count"));

```

This code block takes all the information that is needed for the page by using *magazine\_id* value of magazine object. Common details like, magazine name or category name are stored to Label's.

There are three links blow of these labels. These are; *addLink*, which directs user to the *SourcePlacedPage*, *editMagazineLink*, which directs users to the *MagazineEditPage* and *deleteMagazineLink*, which deletes magazine from *magazines* table by using *magazine\_id*. All of these links are visible to only admins.

After that, all records of this magazine are listed in a table. For this, *DataView* class is used. A *List*, which is type of magazine, takes magazine objects from *magazines* table with *get\_magazine* function of *DataBaseConnection* class. Then, this list is sent to a

ListDataProvider object's constructor as its source. And then, data provider is used in creation of the DataView. In addition to record details (such as location, availability), there are four links in every row of this table. These are;

*Rent* – It is visible for users and guests. If a guest clicks it or a user clicks a rent link of an unavailable record, a warning message will be shown depends on the situation. Otherwise rent operation will be executed.

*editLink* – Link created a *Common* object and fills it by using *magazine* object of the page. Then it sends this *Common* object to SourceEditPage. The link is visible for only admins.

*deleteLink* – Link deletes only the record which corresponds to the current row. Only admins can see this link.

*waitLink* – Link is visible for users when the record in this row is unavailable for renting. It simply adds a waiting row to the *waiting\_resources* table.

BasicAuthenticationSession class's *getUser* is often used in this class to control, if the user is logged in and if he/she is logged in, is he/she is a admin or not. These controls set visibility of the components in the page.

There is also a rent function in the class, which is executed in the *rent* link if necessary conditions are met. Function takes magazine object of the page, and uses its fields in queries. Three tables must be updated in the renting operation;

```
dbc.Insert(String.format("INSERT INTO logs
(`user_id`,`record_id`,`renting_date`,`expected_returning_date`)
+ " values (%d,%d,current_date(),DATE_ADD(current_date(),INTERVAL 30 DAY));", x,
magazine.get_record_id()));
```

This query inserts a new log to *logs* table in database.

```
dbc.Insert(String.format("UPDATE physical_resources MODIFY SET `current_log_id` =
(SELECT max(log_id) FROM logs)" + " where record_id = %d;",
magazine.get_record_id()));
```

And this query, updates *current\_log\_id* of the record looking to max *log\_id* of *logs* table. Because a new insertion is just done, max (*log\_id*) will get the new renting operation's *log\_id* to the query.

```
dbc.Insert(String.format("UPDATE magazines MODIFY SET `magazine_rental_count` =
`magazine_rental_count` + 1"
+ " where `magazine_id` = (SELECT `x_id` from records where `record_id` = %d)",
magazine.get_record_id()));
```

Finally, this code increments *magazine\_rental\_count* of the corresponding record.

#### 4.2.3.8 register.java

This class is inherited from *Form* class. It simply takes *name*, *surname*, *username* and *password* to create a new *User* object. After creation of the object, it will be sent to an object of *UserControl* class. This class will add the user to the database, if insertion operation throws an exception, visitor will be directed to the *registerPage*. But this time a warning message, which is saying that this username is already in user, will be shown.

#### 4.2.3.9 registerPage.java

This class is inherited from *BasePage*. It takes a string in its constructor, which will be used to show a warning message. It contains a *register* form class and *guestNavigation*.

#### 4.2.3.10 searchForm.java

This class is inherited from *Form*. This form is shown in the *HomePage*. The form consists of three part; source type selection, category selection and search box.

- Source type selection has two CheckBox, *electronic* and *physical*.
- Category selection has two DropDownChoice. First one is for selection between book, DVD, magazine or ALL. After user select one of this choices, second DropDownChoice list, which depends on first choice, is created dynamically.
- Search box simply adds a constriction to the search operation. String, which is entered to this TextField, will be searched not just only in book names but also in author names.

After adding of components, in onSubmit function, necessary objects are sent to SearchPage. These objects are;

*book*, *dvd* and *magazine*; They are Boolean type of objects and they contains data of the first DropDownChoice component.

*category\_id*; It is an Integer type of object and it sends the id of the category which is selected in second DropDownChoice. This *category\_id*, is taken from database with select queries in the onSubmit method. Depending on the first DropDownChoice, different queries are used to get *category\_id*.

*electronic*, *physical*; They are Boolean type of objects and they contains date of the CheckBox choices. If both of them are not selected, program will consider this situation as both of them are selected and they will be sent with *true* value.

*search*; It is a String type of object and it contains the entered value of Textfield.

#### 4.2.3.11 searchPage.java

This class is inherited from *BasePage* class. It takes seven arguments which are described in the *searchFrom.java* section. Three queries, *book\_query*, *dvd\_query*, *magazine\_query* is created with these object as shown below;

```
String book_query = "SELECT book_name, author_name, book_category_name,
book_publication_year,book_id,book_ISBN,book_rental_count FROM books"
+ " inner join authors on (authors.author_id = books.author_id) "
+ " join book_categories on (books.book_category_id =
book_categories.book_category_id) WHERE"
+ " (book_name LIKE \"%\"
+ search
+ \"%\" OR author_name LIKE \"%\" + search + "%\");"
String dvd_query = "SELECT
DVD_name,DVD_publication_date,DVD_category_name,DVD_duration,DVD_id FROM
DVDs "
+ "JOIN DVD_categories ON (DVDs.DVD_category_id =
DVD_categories.DVD_category_id) WHERE (DVD_name LIKE \"%\"
+ search + "%\");"
String magazine_query = "SELECT * FROM magazines JOIN magazine_categories ON
(magazines.magazine_category_id = magazine_categories.magazine_category_id) WHERE
(magazine_name LIKE \"%\"
+ search + "%\");"
```

While in these states, queries will only look up for search constriction. Depends on the value of the Boolean arguments, new clauses will be added to the queries;

```
if (category_id != 0) {
    book_query += " and books.book_category_id = " + category_id;
    dvd_query += " and DVDs.DVD_category_id = " + category_id;
    magazine_query += " and magazines.magazine_category_id = "
        + category_id;
}
if (physical != electronic) {
    book_query += " and book_id in (select x_id from records where type_id = 1
and `physical-electronic` = "
        + physical + " )";
    dvd_query += " and DVD_id in (select x_id from records where type_id = 3 and
`physical-electronic` = "
        + physical + " )";
    magazine_query += " and magazine_id in (select x_id from records where
type_id = 2 and `physical-electronic` = "
        + physical + " )";
}
```

After that, queries will be completed. Three functions of the class, *getBookTable*, *getDvdTable*, *getMagazineTable* will take these queries and return a *DataView* by using result tables of the queries. All, three of *DataView*, will be created and added to the page, but only the selected ones will be visible. Selection will be done by using, *book*, *dvd*, *magazine* arguments of the constructor.

In *getBookTable* function:

Query, which is taken as a string argument, is executed and result data is stored to a List, which is type of *Book*. This list will be the source of *listDataProvider* and *listDataProvider* will populate the *dataView*.

```

DataView<Book> dataView = new DataView<Book>("book_rows", listDataProvider) {
    @Override
    protected void populateItem(Item<Book> item) {
        final Book book = item.getModelObject();
        item.add(new Label("name", book.get_name()));
        item.add(new Label("author", book.get_author_name()));
        item.add(new Label("category",
book.get_category_name()));
        item.add(new Label("year", book.get_publish_year()));
        item.add(new Link("link") {
            @Override
            public void onClick() {
                try {
                    this.setResponsePage(new
BookPage(book, ""));
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
};

```

Populating operation of the *dataView* is shown above. There is a *link* component in the rows of *dataView* which will direct the user to the corresponding *BookPage* when it is clicked. After creation of *dataView*, it will be returned to constructor.

*getDvdTable* and *getMagazineTable* functions are quite similar to *getBookTable*. Only some column names are different in these two functions.

#### 4.2.3.12 SignInPage.java

This class is inherited from *BasePage* class. It simply contains *guestNavigation* and a *UserLogin* form.

#### 4.2.3.13 UserControl.java

*UserControl* class is created for adding a new user. It takes a *User* object in its constructor. In *AddUser* function, it uses fields of this *User* object and creates a query to insert a user to *users* table in database. Inserting the query is shown below;

```

String query = String
.format("INSERT INTO "
+ "users (`user_nickname`, `user_name`, `user_surname`,
`user_password`, `user_authority_state`) "
+ "VALUES ('%s', '%s', '%s', '%s', %d);",
unkown.getUsername(), unkown.getName(),
unkown.getSurname(), unkown.getPassword(),
unkown.get_authorityState());
try {
    dbcon.Insert(query);
} catch (SQLException e) {
    e.printStackTrace();
}

```

Here, catching a *SQLException* causes *register* form class to throw an exception, which will eventually create an warning message in the *registerPage*.



#### 4.2.3.14 User.java

User class is created to implement a user in real life. It contains these fields;

```
private Integer _id = null;
private String _name = null;
private String _surname = null;
private String _username = null;
private String _password = null;
private Integer _point = null;
private Integer _authorityState = null;
private Timestamp _last_access_time = null;
```

User objects are used almost every part of the project. It provides a control mechanism to decide which components, when will be visible.

#### 4.2.3.15 UserLogin.java

*UserLogin* class is inherited from *Form* class. It is a simple login form with two *TextField*; *usernameField*, *passwordField* and a link to the *registerPage*. In its *onSubmit* function, entered values of two *TextField*'s are taken as strings and sent to *signIn* function of *BasicAuthenticationSession* class. This function returns a Boolean value. If it is true, it means user is successfully logged in and will be directed to *HomePage*. If it is false, guest will have to try again or use link to *registerPage* to create a new user.

### 4.2.4 Hüseyin Erdoğan's classes

#### 4.2.4.1 AboutUsPage.java

- This class is in guest navigation panel
- This class inherited from *BasePage* class
- Used for giving information about Project and libraries
- There is no SQL code in this class
- This part will be seen only guest user not normal user and admin.

#### 4.2.4.2 ContactPage.java

- This class is in guest navigation panel
- This class inherited from *BasePage* class
- Used for giving information about group members so users can contact with us.
- There is no SQL code in this class
- This part will be seen only guest user not normal user and admin.

#### 4.2.4.3 AdminProfilePage.java

- This class is in admin navigation panel
- Admin can see his/her profile information.
- Admin can see last 10 renting activities thanks to this class.
- This class inherited from *BasePage* class
- This query is used for selecting information about last 10 renting activities

```
String query = "SELECT user_name, user_surname, user_last_access_time, "
    + "x_id, type_id, renting_date, expected_returning_date "
    + "FROM logs JOIN users ON (logs.user_id = users.user_id) "
    + "JOIN records ON (records.record_id = logs.record_id) "
    + "ORDER BY log_id DESC LIMIT 10; ";
ResultSet rs = dbc.GetResult(query);
```

According to type\_id, source type are determined. Queries as follows :

```
DatabaseConnection dbcsource = new DatabaseConnection();
if (log.get_type_id() == 1) {
    query = String
        .format("SELECT book_name FROM books WHERE book_id = '%d';",
            log.get_x_id());
    ResultSet rsource = dbcsource.GetResult(query);
    if (rsource.next()) {
        log.set_name(rsource.getString("book_name"));
        log.set_type_name("Book");
    }
} else if (log.get_type_id() == 2) {
    query = String
        .format("SELECT magazine_name FROM magazines WHERE magazine_id = '%d';",
            log.get_x_id());
    ResultSet rsource = dbcsource.GetResult(query);
    if (rsource.next()) {
        log.set_name(rsource.getString("magazine_name"));
        log.set_type_name("Magazine");
    }
} else if (log.get_type_id() == 3) {
    query = String.format(
        "SELECT DVD_name FROM DVDs WHERE DVD_id = '%d';",
        log.get_x_id());
    ResultSet rsource = dbcsource.GetResult(query);
    if (rsource.next()) {
        log.set_name(rsource.getString("DVD_name"));
        log.set_type_name("DVD");
    }
}
```

#### 4.2.4.4 UserProfilePage.java

- This class is in user navigation panel
- User can see his/her profile information.
- User can see last waiting resources list and delete them thanks to this class.
- This class inherited from BasePage class
- This query is used for taking waiting resources from "waiting\_resources" table by using "user\_id".

```

dbc = new DatabaseConnection();
String str = String.format(
    "select `user_id` from users where `user_nickname` = '%s';",
    ((BasicAuthenticationSession) BasicAuthenticationSession.get())
        .getUser().getUsername());
ResultSet rs = dbc.GetResult(str);
rs.next();
int x = rs.getInt("user_id");
add(getBookTable(String
    .format("SELECT * FROM book_informations where record_id in "
        + "(SELECT record_id FROM waiting_resources where user_id = %d)",
        x)));
add(getDvdTable(String
    .format("SELECT * FROM DVD_informations where record_id in "
        + "(SELECT record_id FROM waiting_resources where user_id = %d)",
        x)));
add(getMagazineTable(String
    .format("SELECT * FROM magazine_informations where record_id in "
        + "(SELECT record_id FROM waiting_resources where user_id = %d)",
        x)));
dbc.close();

```

- Deleting waiting resources is different for tree source type.
- For books this query is used.

```

item.add(new Link("delete") {
    @Override
    public void onClick() {
        try {
            DatabaseConnection dbcreq = new DatabaseConnection();
            String query = String
                .format("DELETE FROM waiting_resources WHERE user_id = %d;",
                    ((BasicAuthenticationSession) BasicAuthenticationSession
                        .get()).getUser().get_id(), "AND record_id = %d;", book.get_record_id());
            dbcreq.Insert(query);

            this.setResponsePage(new UserProfilePage());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});

```

- For DVDs this query is used.

```

item.add(new Link("delete") {
    @Override
    public void onClick() {
        try {
            DatabaseConnection dbcreq = new DatabaseConnection();
            String query = String
                .format("DELETE FROM waiting_resources WHERE user_id = %d;",
                    ((BasicAuthenticationSession) BasicAuthenticationSession
                        .get()).getUser().get_id(), "AND record_id = %d;", dvd.get_record_id());
            dbcreq.Insert(query);

            this.setResponsePage(new UserProfilePage());
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
});

```

- For magazines this query is used.

```

item.add(new Link("delete")) {
    @Override
    public void onClick() {
        try {
            DatabaseConnection dbcreq = new DatabaseConnection();
            String query = String
                .format("DELETE FROM waiting_resources WHERE user_id = %d;",
                    ((BasicAuthenticationSession) BasicAuthenticationSession
                        .get()).getUser().get_id(),"AND record_id = %d;",magazine.get_record_id
                );
            dbcreq.Insert(query);

            this.setResponsePage(new UserProfilePage());
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
};

```

#### 4.2.4.5 UserPage.java

- This class is used for login procedure.
- This class inherited from BasePage class

#### 4.2.4.6 Dvd.java

- This class is used for storing DVD informations.
- Variables, that are used in this, class as follows :

```

private String _name = null;
private Integer _rental_count = 0;
private Integer _publish_year = null;
private Integer _duration = null;
private Integer _availability = null;
private Integer _physical_electronic = null;
final private Integer _type_id = 3;
private String _library_name = null;
private String _floor_value = null;
private Integer _bookcase_id = null;
private Integer _bookshelf_id = null;
private String _bookcase_name = null;
private String _bookshelf_name = null;
private Integer _column_id = null;
private String _url = null;
private Integer _record_id = null;
private Integer _DVD_id = null;
private Integer _category_id = null;
private String _category_name = null;

```

- All of these variables have a getter and setter methods.
- This class implements Serializable class which is in java library
- This class has tree constructor
- First one is constructor with no parameter
- Second one is used :

```

public Dvd(String _name, Integer _publish_year, Integer _duration,Integer _DVD_id) {
    this._name = _name;
    this._publish_year = _publish_year;
    this._duration = _duration;
    this._DVD_id = _DVD_id;
}

```

This constructor is used for storing informations that are taken in add DVD page.

- Third one is like this :

```
public Dvd(String _name, Integer _rental_count, Integer _publish_year,
    Integer _duration, Integer _availability,
    Integer _physical_electronic, String _library_name,
    String _floor_value, Integer _bookcase_id, Integer _bookshelf_id,
    Integer _column_id, String _url, Integer _record_id, Integer _DVD_id, Integer _category_id) {
    super();
    this._name = _name;
    this._rental_count = _rental_count;
    this._publish_year = _publish_year;
    this._duration = _duration;
    this._availability = _availability;
    this._physical_electronic = _physical_electronic;
    this._library_name = _library_name;
    this._floor_value = _floor_value;
    this._bookcase_id = _bookcase_id;
    this._bookshelf_id = _bookshelf_id;
    this._column_id = _column_id;
    this._url = _url;
    this._record_id = _record_id;
    this._DVD_id = _DVD_id;
    this._category_id = _category_id;
}
```

This constructor is used for storing all variables of DVD.

#### 4.2.4.7 DvdPage.java

- This class is in all navigation panel
- This class inherited from BasePage class
- After searching procedure, this page is showed.
- This class has one constructor and one function that is used for renting procedure.
- This query is used for taking DVD information from database

```
final DatabaseConnection dbc = new DatabaseConnection();
String query = String
    .format("SELECT DVD_name, DVD_publication_date, DVD_duration, "
        + "DVD_rental_count, DVD_category_name FROM DVDs "
        + "JOIN DVD_categories ON (DVDs.DVD_category_id = DVD_categories.DVD_category_id) "
        + "WHERE DVD_id = %d;", dvd.get_DVD_id());

ResultSet rs = dbc.GetResult(query);
```

- Tree label used for renting, editing and deleting

```
add(new Label("rentColumn", "Rent"));
add(new Label("editColumn", "Edit"));
add(new Label("deleteColumn", "Delete"));
```

- For deleting DVDs these queries are used

```
try {
    dbc.Insert(String.format(
        "DELETE FROM DVDs where `DVD_id` = %d",
        dvd.get_DVD_id()));

    dbc.Insert(String
        .format("DELETE FROM records where type_id = 3 and `x_id` = %d",
            dvd.get_DVD_id()));
}
```

- For deleting DVD's place these query is used

```
dbc.Insert(String
    .format("DELETE FROM record_id where `record_id` = %d",
        dvd.get_record_id()));
```

- Adding DVDs to waiting resources list

First user\_id is selected, after user\_id and record\_id are used for inserting DVD to "waiting\_resources" table.

```
item.add(new Link("waitLink")) {
    @Override
    public void onClick() {
        try {
            String str = String
                .format("select `user_id` from users where `user_nickname` = '%s';",
                    ((BasicAuthenticationSession) BasicAuthenticationSession
                        .get()).getUser()
                        .getUsername());

            ResultSet rs;

            rs = dbc.GetResult(str);
            rs.next();
            int x = rs.getInt("user_id");
            dbc.Insert(String
                .format("INSERT INTO waiting_resources (`user_id`,`record_id`) values (%d,%d);",
                    x, dvd.get_record_id()));

            this.setResponsePage(new DvdPage(dvd, ""));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
};
```

- Renting function as follows :

```
private void rent(Dvd dvd) throws Exception {
    DatabaseConnection dbc = new DatabaseConnection();

    String str = String.format(
        "select `user_id` from users where `user_nickname` = '%s';",
        ((BasicAuthenticationSession) BasicAuthenticationSession.get())
            .getUser().getUsername());
    ResultSet rs = dbc.GetResult(str);
    rs.next();
    int x = rs.getInt("user_id");
    dbc.Insert(String
        .format("INSERT INTO logs (`user_id`,`record_id`,`renting_date`,`expected_returning_date`)"
            + " values (%d,%d,current_date(),DATE_ADD(current_date(),INTERVAL 30 DAY));",
            x, dvd.get_record_id()));
    dbc.Insert(String
        .format("UPDATE physical_resources MODIFY SET `current_log_id` = (SELECT max(log_id) FROM logs)"
            + " where record_id = %d;", dvd.get_record_id()));
    dbc.Insert(String
        .format("UPDATE DVDs MODIFY SET `DVD_rental_count` = `DVD_rental_count` + 1"
            + " where `DVD_id` = (SELECT `x_id` from records where `record_id` = %d)",
            dvd.get_record_id()));
    dbc.close();
}
```

#### 4.2.4.8 AddDVDPage.java

- This class is used in admin navigation panel
- This class inherited from BasePage class
- In this class DVD informations are taken from admin.
- Has one constructor with no parameter.
- And consists "adddvdform"

#### 4.2.4.9 AddDVDForm.java

- This class is used in admin navigation panel
- This class extends Form class
- This class has two function; AddDVDForm and OnSubmit method.
- In AddDVDForm function this query is used.

```

// {
DatabaseConnection dbc = new DatabaseConnection();
String query = "SELECT DVD_category_name FROM DVD_categories ORDER BY DVD_category_name;";
ResultSet cat = dbc.GetResult(query);

```

- In OnSubmit function this query is used for storing DVD informations in database.

```

dbc = new DatabaseConnection();
query = String
    .format("INSERT INTO DVDs (DVD_name, DVD_duration, DVD_publication_date, DVD_category_id) VALUES ('%s', '%d', '%d-01-01', '%d');",
        dvd.get_name(), dvd.get_duration(),
        dvd.get_publish_year(), dvd.get_category_id());

dbc.Insert(query);

```

#### 4.2.4.10 DVDEditPage.java

- This class is used in admin navigation panel
- This class inherited from BasePage class
- In this class DVD informations are updated by user.
- Has one constructor with parameter. This constructor takes variable that's type is "DVD".
- And consists "dvd\_edit\_profile"

#### 4.2.4.11 DVDEditForm.java

- This class is used in admin navigation panel
- This class extends Form class
- This class has two function; AddDVDForm and OnSubmit method.
- In OnSubmit function DVD's name, duration, publication date, and category id is updated. DVD\_id is used for locating DVD which is going to be updated in database.

Query is as follows :

```

- - - - -
dbc = new DatabaseConnection();
query = String
    .format("UPDATE DVDs SET DVD_name = '%s', DVD_duration = %d, "
        + " DVD_publication_date = '%d-01-01', DVD_category_id = %d WHERE DVD_id = %d;",
        dvd.get_name(), dvd.get_duration(),
        dvd.get_publish_year(), dvd.get_category_id(), dvd.get_DVD_id());

dbc.Insert(query);
- - - - -

```