I.T.U.

FACULTY OF COMPUTER AND INFORMATICS

COMPUTER ENGINEERING

2013-2014 FALL

BLG 361E DATABASE MANAGAMENT SYSTEMS

CRN: 10795

GROUP 2 TERM PROJECT INDIVIDUAL REPORT

E-LIBRARY

Tuğrul YATAĞAN

040100117

TABLE OF CONTENT

1 General Informations

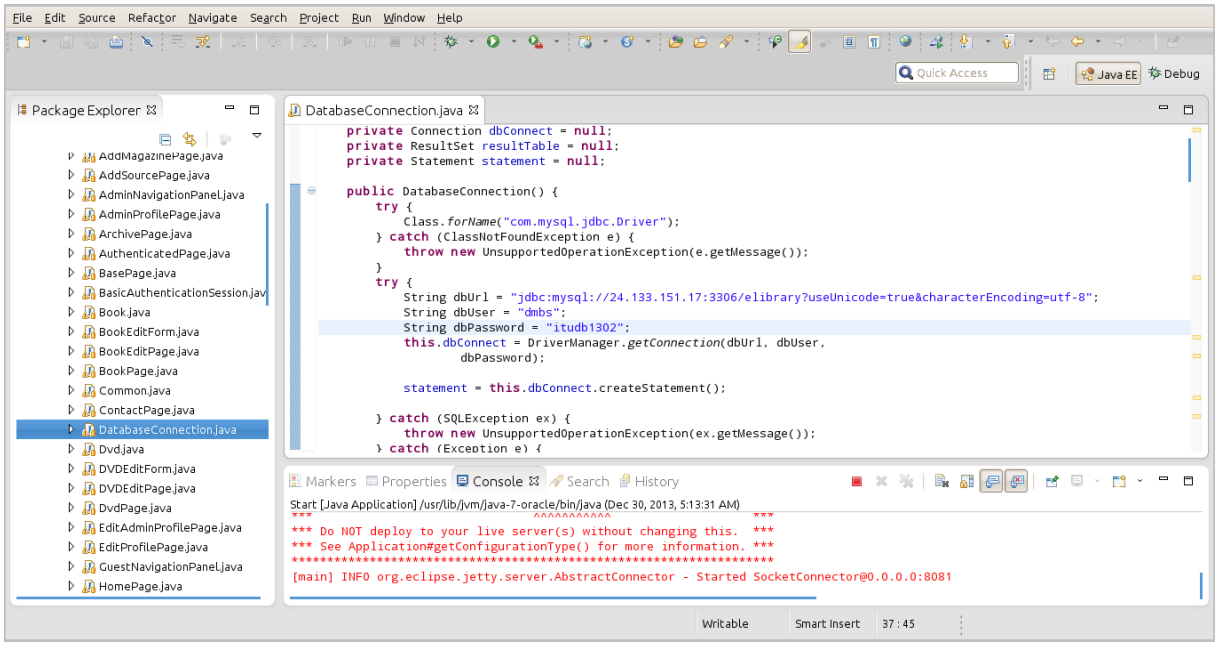
1.1 Project Description

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 hey 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.

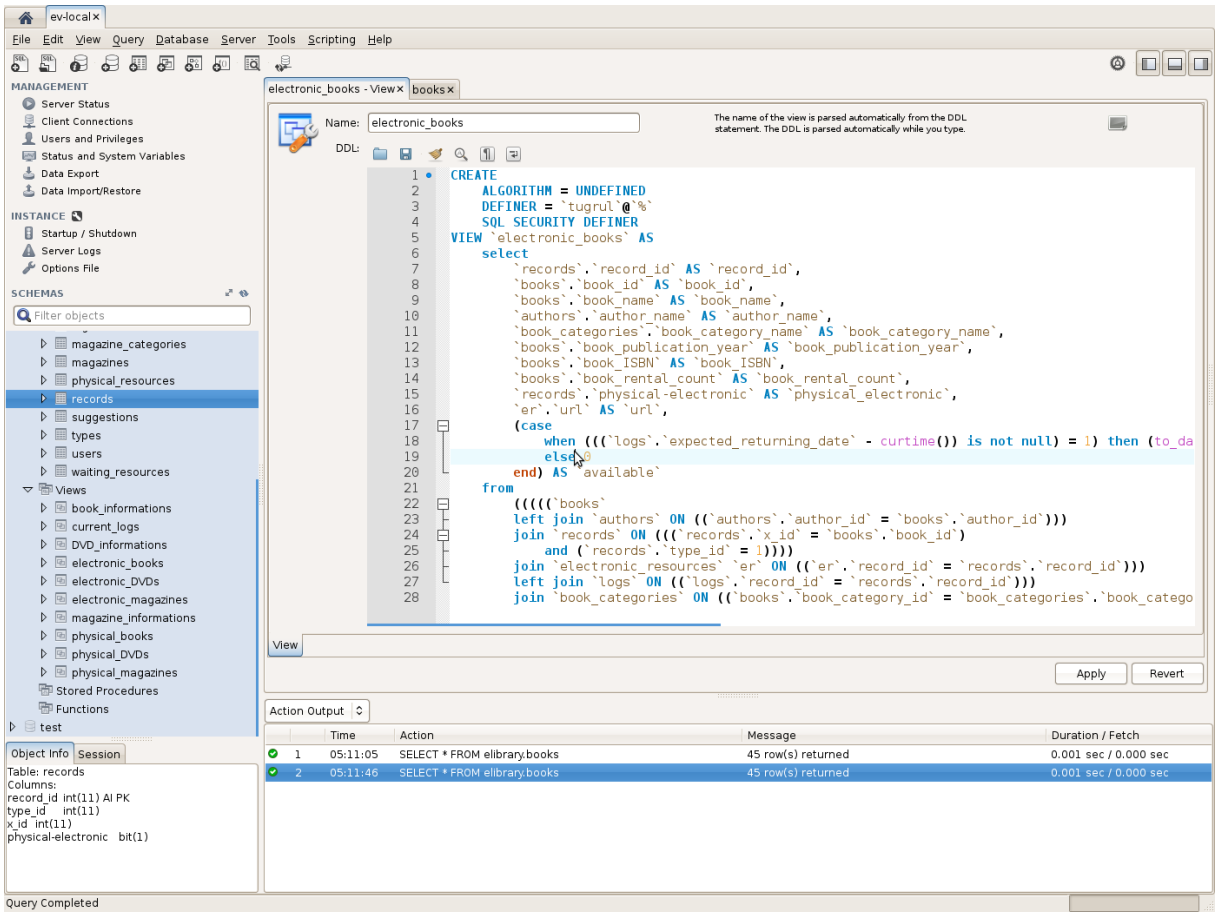
1.2 Development and Execution Environment

Project is developed using Java programing 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 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());
}
```

1.4 My Tasks

- 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

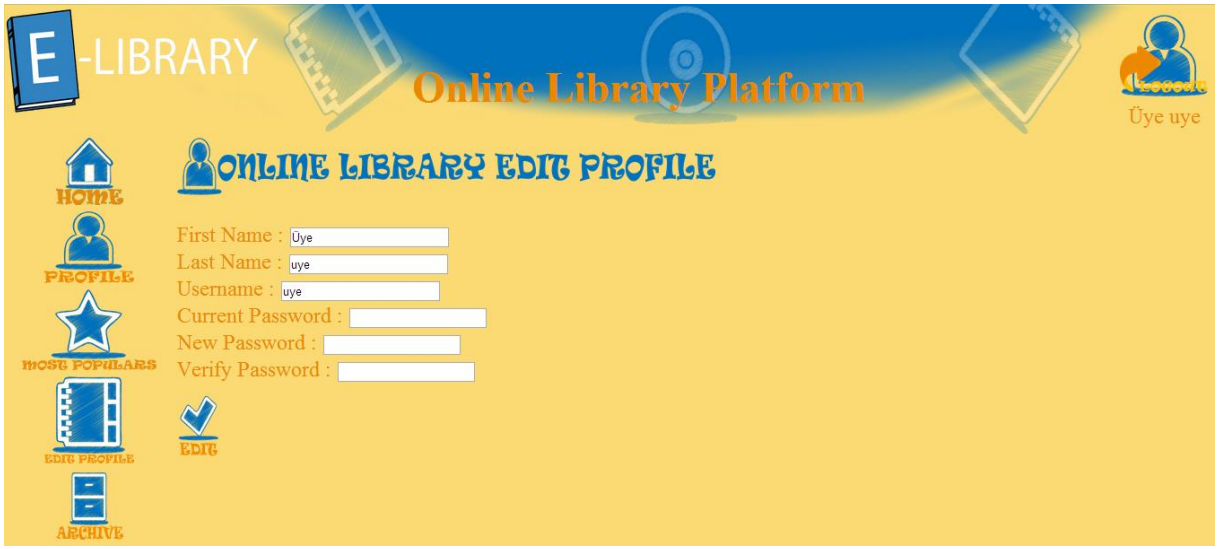
2 User Manual

2.1 User Interface

This Interface is accessible for regular registered users. Everyone can register and can be a member of the system.

2.1.1 User Edit Profile Page

In this page regular user can update his/her informations like name, surname, username and password. User have to reenter his/her current password to change his/her information. When user clicks the edit button user's informations will be updated on database.

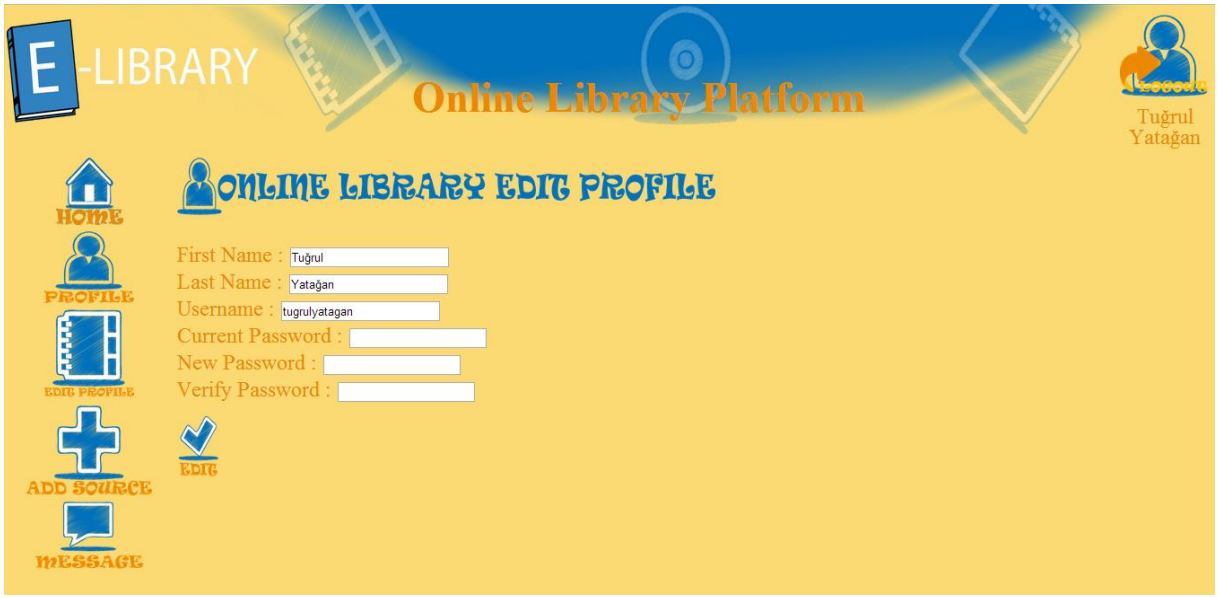


2.2 Administrator Interface

This Interface is accessible for only administrator users. Administrative operations are located in this module. Admins are responsible for using this module correct way. Sources and resources informations are taken from admin in this Interface.

2.2.1 Admin Edit Profile Page

In this page regular admins can update his/her informations like name, surname, username and password like regular users. Admin have to reenter his/her current password to change his/her informations. When admin clicks the edit button admin's informations will be updated on database.

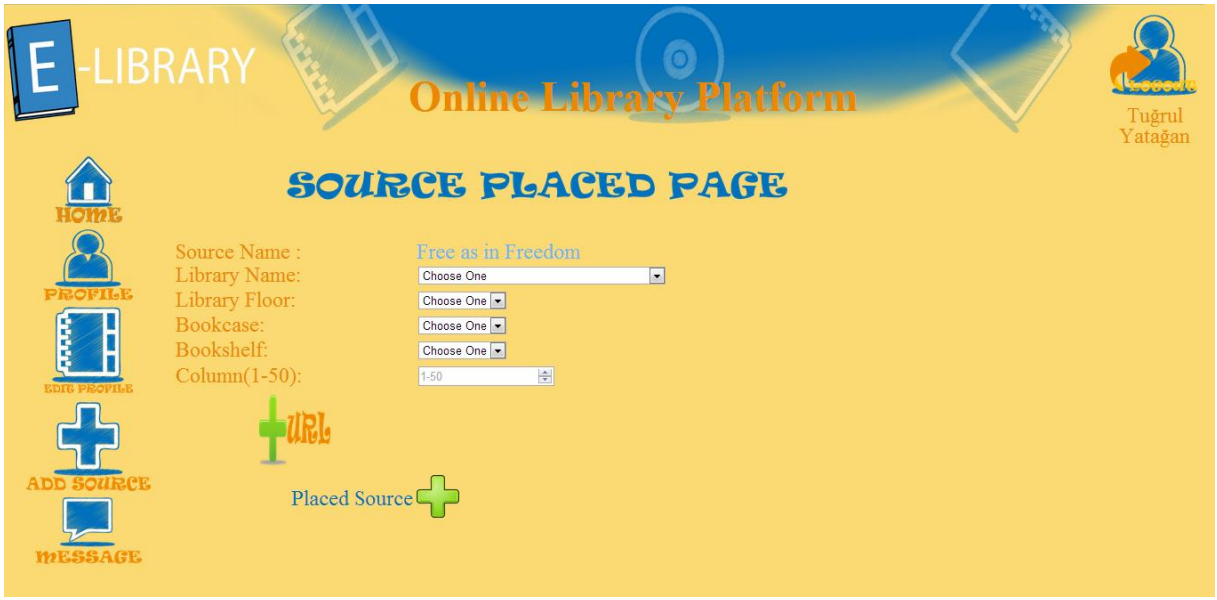


2.2.2 Add Source Page

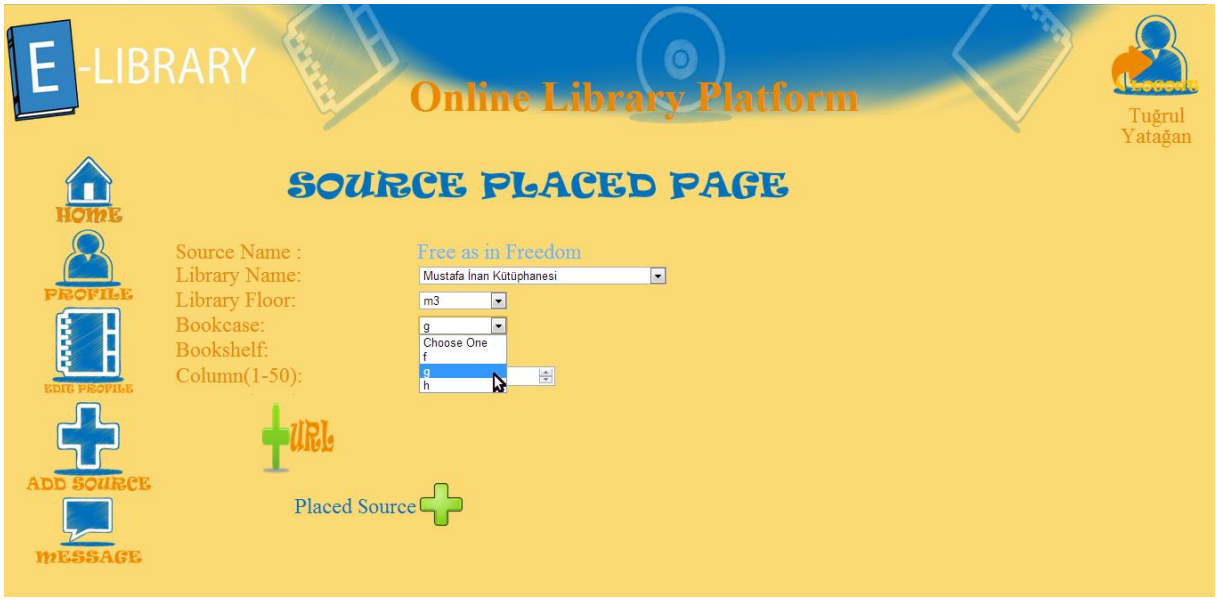
In this page admin can add source like book, DVD and magazine. Only the admin user can view this page. When admin clicks one of the add source icons, page will redirect to relative source add page. For example if admin clicks add book icon, page will redirect to *AddBookPage.*

2.2.3 Source Placed Page

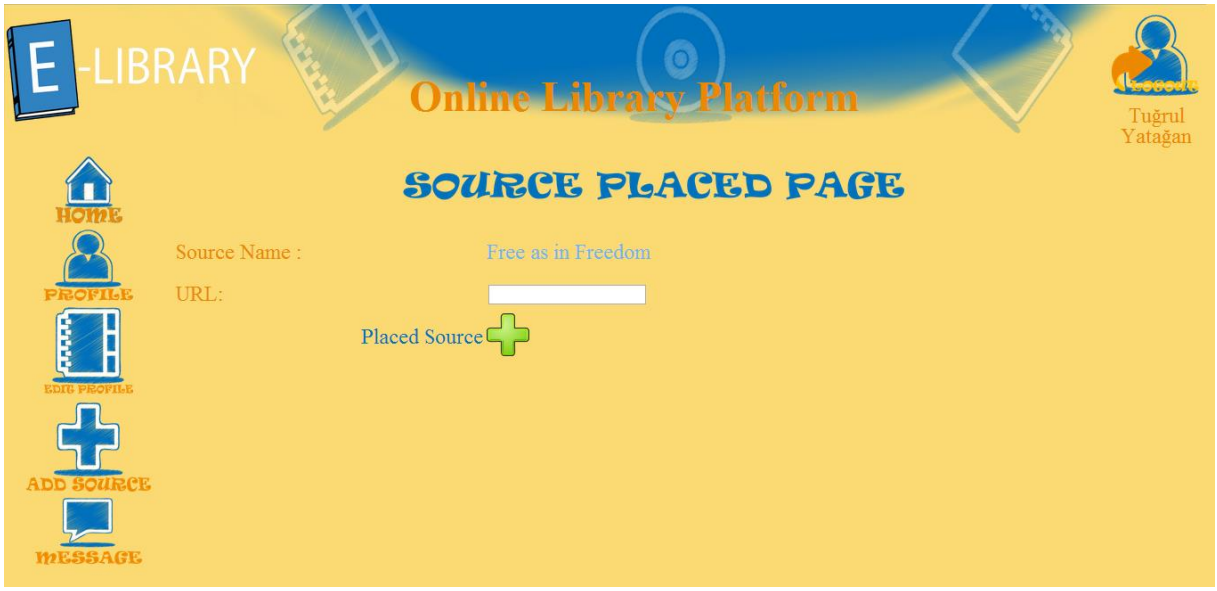
In this page admin can place source to library. Admin gives place informations like library, floor, bookcase, bookshelf and column.



When user selects library dropdown choice, floor dropdown choice is restricted according to user's selection and it continuous to the bookshelf dropdown choices. Every selection restricts next dropdown choice's selection. When admin clicks placed source icon, source is added to library.

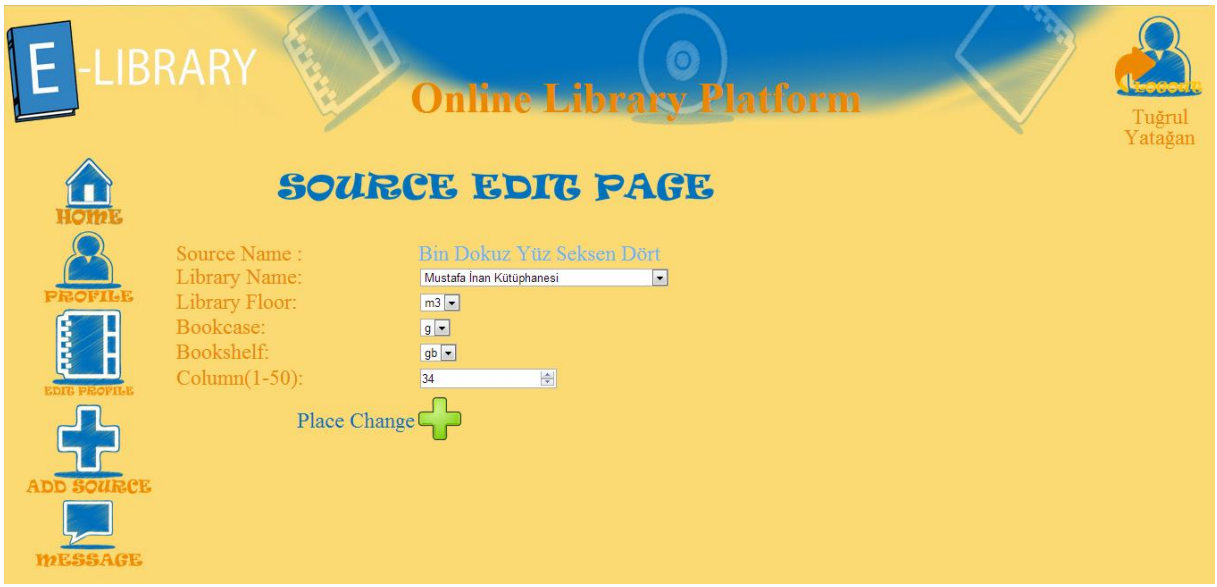


One source can have only one URL field. When admin clicks URL icon, a page with URL field is displayed then admin can enter URL information. When a URL information is entered to a source, add URL icon and field is disappear for the source.



2.2.4 Edit Source Placed Page

In this page admin can edit source's place information of record. Admin updates place informations like library, floor, bookcase, bookshelf and column.



If record is electronic resource, a page with URL field is displayed then admin can update URL information. When admin clicks place change icon, URL information of electronic record is

updated.

3 Technical Manual

3.1 Database Design

3.1.1 Tables

3.1.1.1 "types" Table

This table consist of type informations like book, DVD, magazine. *books*, *DVDs* and *magazines* tables connects to *records* table via *type_id* coloumn.

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

3.1.1.2 "books" Table

This table consist of book informations like name, author, year, ISBN and category. Author_id and book_category_id are foreign key of *authors* and *book_categories* table.

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

3.1.1.3 "magazines" Table

This table consist of magazine informations like name, issue number, year and category. *magazine*_category_id is foreign key of *magazine_categories* table.

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

3.1.1.4 “DVDs” Table

This table consist of DVD informations like name, duration, year, and category. *DVD*_category_id is foreign key of *DVD_categories* table.

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

3.1.1.5 “bookshelves” Table

This table consist of list of all bookshelves. One bookshelf keeps name, limit and *bookcase_id* informations. *bookcase_id* is foreign key of *bookcases* table. One bookcases may contain multiple bookshelves. Every bookshelf has their limits.

Name	Type	Not Null	Primary Key
bookshelf_id	Int(11)	Auto increment	1
bookcase_id	Int(11)	Not Null	0
bookshelf_name	Varchar(10)	Not Null	0
book_limit	Int(11)	Not Null	0

3.1.1.6 “bookcases” Table

This table consist of list of all bookcases. One bookcase keeps name and *floor_id* informations. *floor_id* is foreign key of *floors* table. One floor may contain multiple bookcases.

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

3.1.1.7 “floors” Table

This table consist of list of all floors. One floor keeps name and *library_id* informations. *library_id* is foreign key of *libraries* table. One library may contain multiple floors.

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

3.1.1.8 “libraries” Table

This table consist of list of all libaries. One library keeps name and address informations. *library_id* is root of all record place tables.

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

3.1.1.9 “records” Table

This table consist of records of sources. Every source at least has one record. On record keeps *type_id*, *x_id* and *physical-electronic* information. Type id defines type of source Exp: book, DVD, magazine. *x_id* connect sources table(Exp: *books, DVDs, magazines*) to *records* table. x_id indicates that primary key of source’s id. Exp: for books*, x_id* is *book_id. physical-electronic* coloumn indicates that the source is physical or electronic. Every record mut be inserted *physical_resources* or *physical_resources* table.

Name	Type	Not Null	Primary Key
record_id	Int(11)	Auto increment	1
type_id	Int(11)	Not Null	0
x_id	Int(11)	Not Null	0
physical-electronic	bit(1)	Not Null	0

3.1.1.10 “physical_resources” Table

This table consist of physical informations of records. *records* and placing informations like *bookshelves* connects via this table. *booksehlf_id* is foreign key of *bookshelves* table. This table also keeps coloumn of the metarial and *current_log_id*. *current_log_id* is forign key of *logs* table.

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

3.1.1.11 “electronic_resources” Table

This table consist of URL informations of records. *records* and URL informations connects via this table. Every source can have only one URL information so *url* has unique constraint.

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

3.1.2 Views

3.1.2.1 “book_information” View

This view combines one book’s all informations like name, author, category, year, ISBN, rental count, library name, floor name, bookcase name, bookshelf name, physical status, availability and URL. This view joins *books, book_categories, authors, libraries, floors, bookcases,*

bookshelves, records, physical_records, electronic_resources and *logs tables*. This view is union of *electronic_books* and *physical_books* views.

Name	**Type**
record_id	Int(11)
book_id	Int(11)
book_name	Varchar(45)
author_name	Varchar(45)
book_category_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)
bookcase_name	Varchar(45)
bookshelf_name	Varchar(45)
book_column	Int(11)
physical_electronic	bit(1)
available	bit(1)
url	Varchar(45)

3.1.2.2 "DVD_information" View

This view combines one book's all informations like name, category, year, duration, rental count, library name, floor name, bookcase name, bookshelf name, physical status, availability and URL. This view joins *DVDs, DVD_categories, libraries, floors, bookcases, bookshelves, records, physical_records, electronic_resources* and *logs tables*. This view is union of *electronic_DVDs* and *physical_DVDs* views.

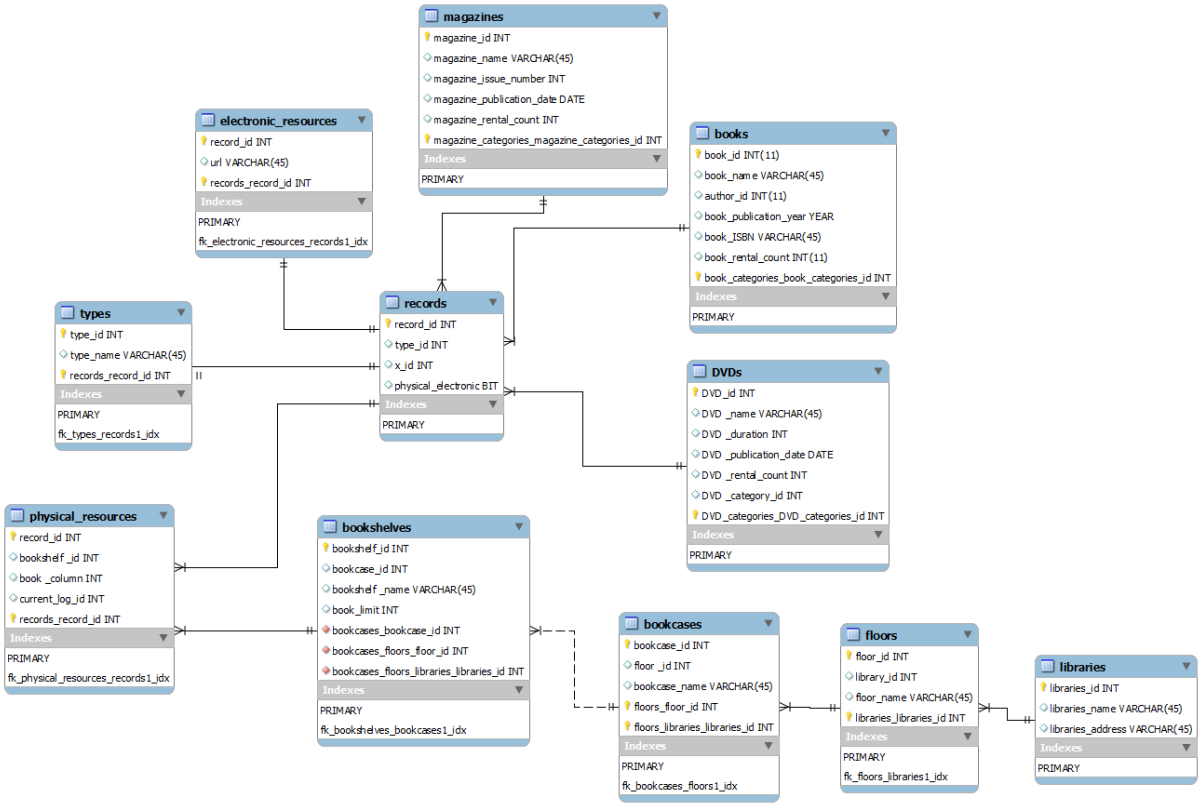
Name	**Type**
record_id	Int(11)
DVD_id	Int(11)
DVD_name	Varchar(45)
DVD_category_name	Varchar(45)
DVD_publication_year	Year(4)
DVD_duration	Varchar(45)
DVD_rental_count	Int(11)
library_name	Varchar(45)
floor_name	Varchar(10)
bookcase_id	Int(11)
bookshelf_id	Int(11)
bookcase_name	Varchar(45)
bookshelf_name	Varchar(45)
DVD_column	Int(11)
physical_electronic	bit(1)
available	bit(1)
url	Varchar(45)

3.1.2.3 “magazine_information” View

This view combines one magazine’s all informations like name, category, year, issue number, rental count, library name, floor name, bookcase name, bookshelf name, physical status, availability and URL. This view joins *magazines, magazine_categories, libraries, floors, bookcases, bookshelves, records, physical_records, electronic_resources* and *logs tables*. This view is union of *electronic_magazines* and *physical_magazines* views.

Name	Type
record_id	Int(11)
magazine_id	Int(11)
magazine_name	Varchar(45)
magazine_category_name	Varchar(45)
magazine_publication_year	Year(4)
magazine_issue_number	Varchar(45)
magazine_rental_count	Int(11)
library_name	Varchar(45)
floor_name	Varchar(10)
bookcase_id	Int(11)
bookshelf_id	Int(11)
bookcase_name	Varchar(45)
bookshelf_name	Varchar(45)
book_column	Int(11)
physical_electronic	bit(1)
available	bit(1)
url	Varchar(45)

3.1.3 Entity Relationship Diagram



3.2 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;
```

3.3 Software Design

3.3.1 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.

3.3.2 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.

3.3.3 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 *ResulSet* 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.

```java
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.

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

```java
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:

```java
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:

```java
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:

```java
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:

```java
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:

```java
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:

```java
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();
```

3.3.4 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*.

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

3.3.5 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"));
```

3.3.6 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());
```

3.3.7 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));
}
```

3.3.8 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:

```java
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 dbcsehlf = new DatabaseConnection();
				query = String.format(
					"SELECT * FROM bookshelves WHERE bookcase_id = %d;",
					cases.getIsnt("bookcase_id"));
				ResultSet shelf = dbcsehlf.GetResult(query);
				List<String> shelfList = new ArrayList<String>();
				while (shelf.next()) {
					shelfList.add(shelf.getString("bookshelf_name"));
				}
				dbcsehlf.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);
}
```

İf a conflict occurs while adding record to *records* table, a *DuplicateKeyException* exception is thrown *physical_resources* tables has unique key constraInt on its *record_id, bookshelf_id, book_column* coloumns and *electronic_resources* table tables 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());
}
```

3.3.9 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));
}
```

3.3.10 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.

3.3.11 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();
```

3.3.12 WicketApplication.java

This class provides entegration 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.

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