DOKUZ EYLÜL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 2210

Object Oriented Analysis and Design

LIBRARY MANAGEMENT SYSTEM

by

Nuri Çilengir 2016510111

Hasan Berk Kocabaş 2016510046

Mustafa Deniz 2016510018

CHAPTER ONE
INTRODUCTION

1.1 Subject

The subject of the project is library management. Library management system will designed to digitize and simplfy the library management.

1.2 Purpose

The purpose of that document is to present a description of the Library Management System. This document explain the all details this software project that database design, operations, interfaces and use of the application.

1.3 Scope

Library Management System is a desktop base application for students, teachers and managers. The system will allow universities to manage their libraries easily with user friendly interfaces and system tools.

More specifically, the LibraryManagementSystem will also allow students to search for books and request them. Teachers also can search books, request them and add publication to system. Managers can edit all informations about users, books and respond to book requests.

1.4 References

[1] IEEE Software Engineering Standards Committee, "IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications", October 20, 1998

CHAPTER TWO
REQUIREMENTS

2.1 External Interfaces

Registiration Screen: That page is for the user registiration.

Login Screen: That page is allow to user and manager login.

Profile Page: That page is allow the show user informations and edit them. In this page also show user activity history.

Collection Page: In this page books are listed. User can search and request for available book.

Manager Page: Manager can manage all users, books and request in this page.

Report page: In this page all reports are listed. For example: book requests and logs.

Aplly Page: For the users allow the book request. Users can see available date interval to the take book. (In the modal)

2.2 Functions

Crud Operations: Users and books can be deleted, updated, created and readed.

Search Operations: Collections, authors, publishers, users can be searched.

Filter Operations: Category, author, publisher, publication date filter can be applied.

Request Operation: User can request for a book.

Calculation Operations: Total user, total book, total request, success – unsuccess requests can be calculated.

Book isAvailable function: This function show to users that book is available or not.

2.3 Performance Requirements

We will use hibernate ecosystem for ORM operations. The software should be capable of storing all the data so arraylist, stack and queue structures will be used.

2.4 Logical Database Requirements

All data that user accounts, collections, etc. will be saved in the database. Program requires a fast and simple design to reliability and maintainability. Hibernate ecosystem and mysql will be used.

2.5 Design Constraints

User Interfaces will be written in Java Swing. N-Tier architecture will be used for reusable and systematical application.

2.6 Software System Quality Attributes

The program must be reliable, support new versions, maintainable for necessary updates and security is the other important thing for Library Management System. Spring security ecosystem will be used for authentication. Test driven development will be used with Junit.

2.7 Analysis Class Model

Class Author: name

Class Book: title, authors, category_ıd, publısher_ıd, type, isbn, edition, publication_year, description

Class Barrow: book_ıd, user_ıd, status, request_date, return_date, return_status

Class BookDetails: book_ıd, return_date, availability

Class BookImages: book_ıd, image

Class Category: name

Class Publisher: name

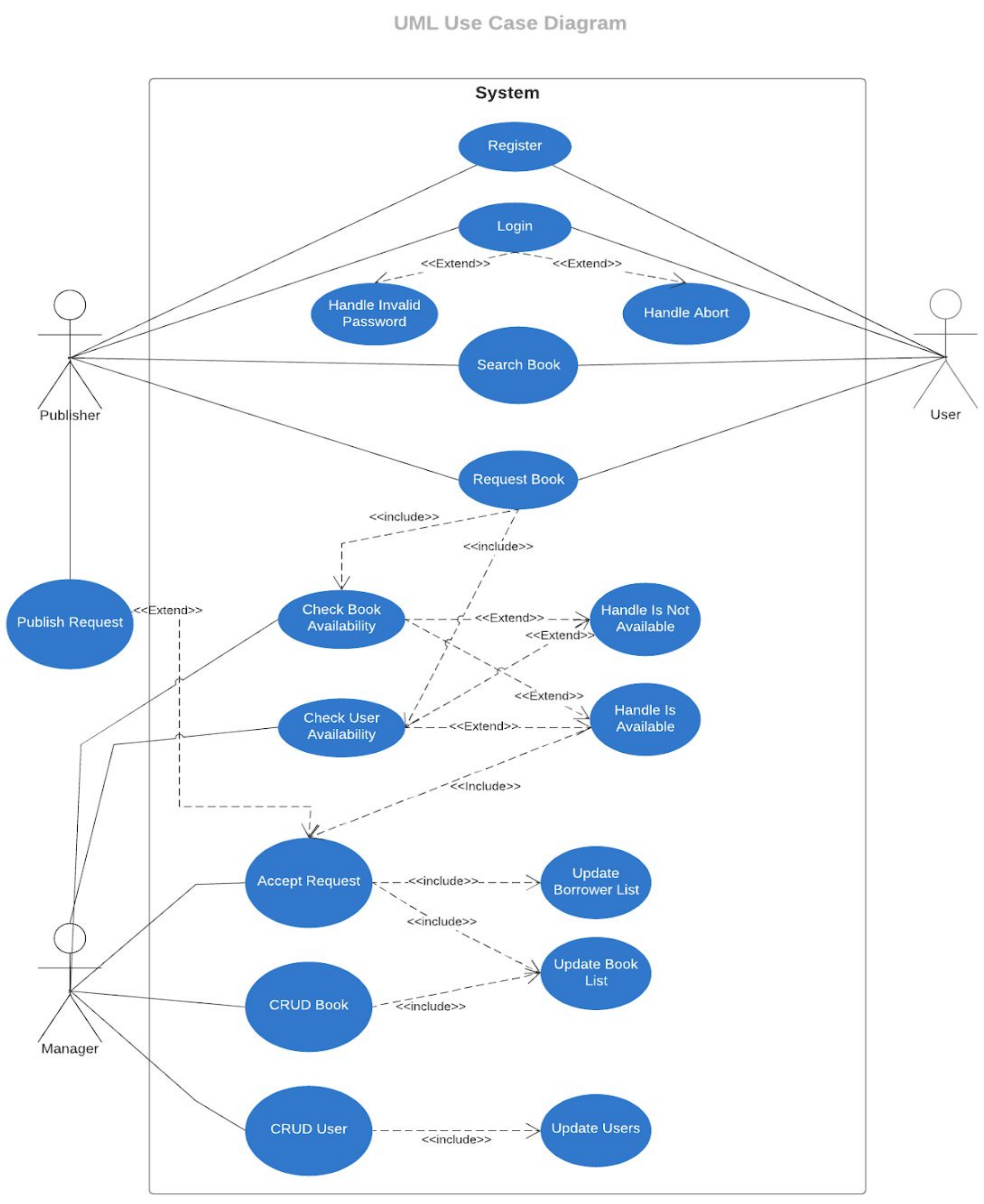
Class User: name, email, password, city_id, phone, address, zipcode, sex, bdate, isadmin

Class city: name

CHAPTER THREE

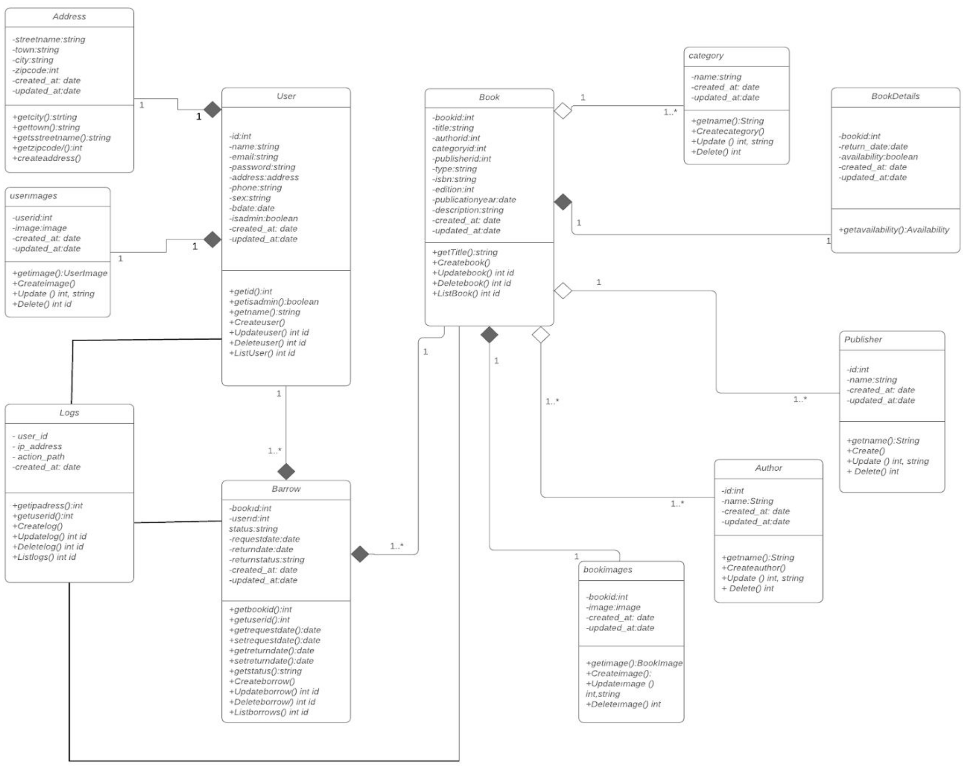
UML DIAGRAMS

3.1 Use Case Diagram



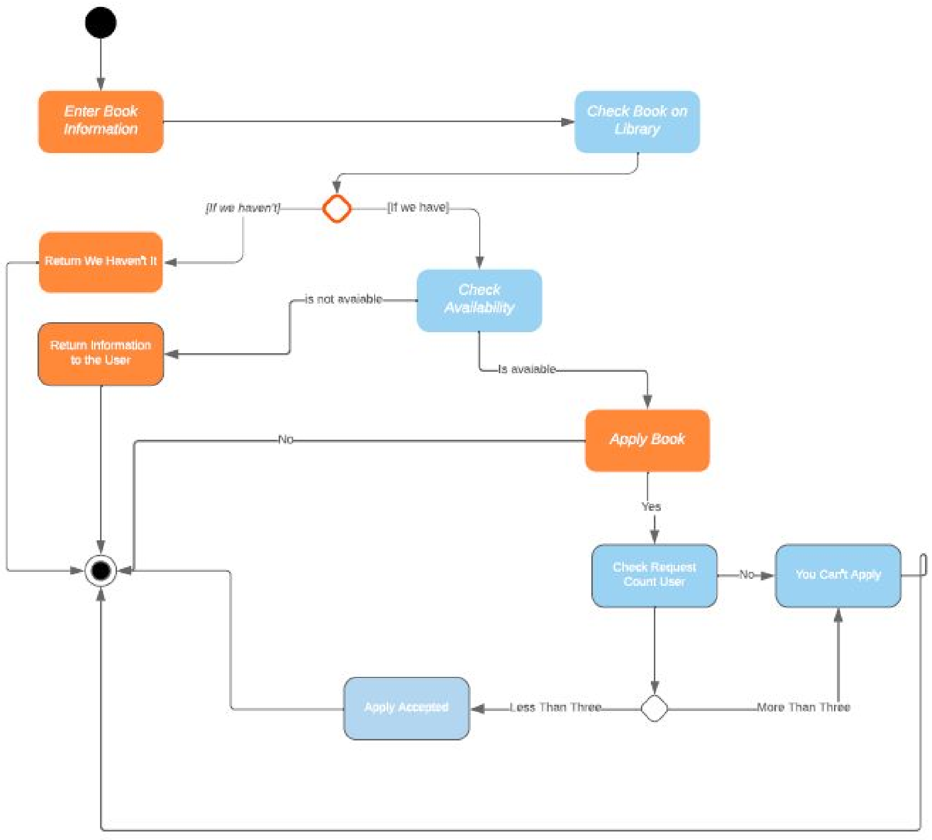
This diagram shows the abilities of the actors in the program in general.The program has 3 actors which are Manager, Standart User(Generally Students) and Publisher. Standart User has ability for registering, searching and requesting books. Publishers, in addition to Standart User they can publish publishment. Managers can access all managerials functions such as deleting, creating, updating and reading all data. Other manager task is the control requrests and accept or decline them. Managers perform this task based on book and user status.

3.2 Class Diagram



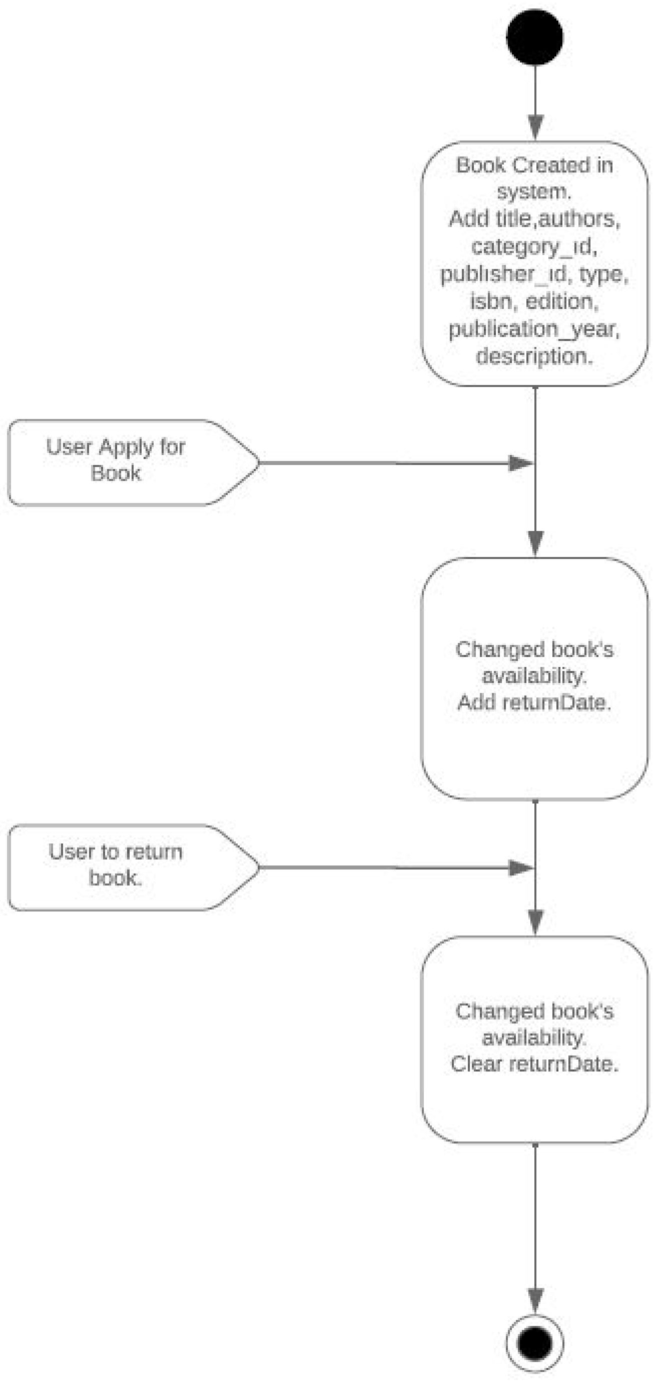
Class diagram is listed above. We have user and book class like a main class. Barrow class connected user and book classes. All user types having address, ımage class and some attributes. Books also have category, details, publisher, author and images classes. Barrow class connected user and book classes. Logs class keeping records belong to user and book operations.

3.3 Activity Diagram



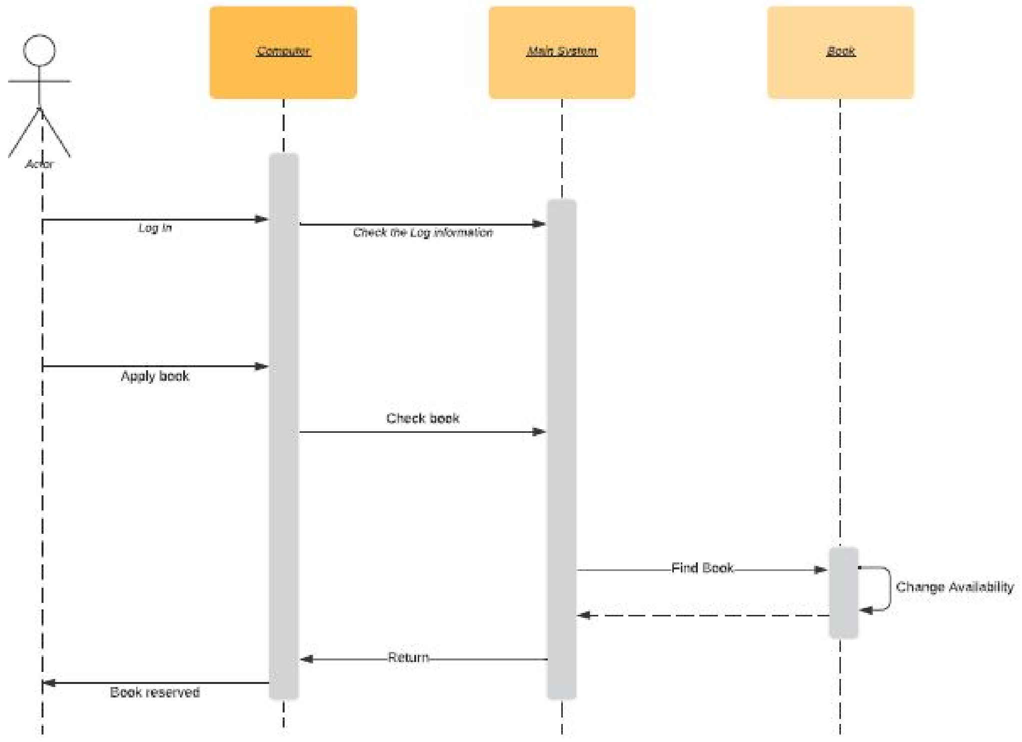
This diagram show us the path of books when hire. First time user enter a book then Book checked on the system if it can be borrow user can apply it but if it can't available system return the book isn't available. When user apply for book system checks request count of user. If it more than three user can't apply for borrow a book if less than three he can apply and borrow the book.

3.4 State Diagram


Book Created in system.
Add title,authors, category_id, publisher_id, type, isbn, edition, publication_year, description.
User Apply for Book
Changed book's availability.
Add returnDate.
User to return book.
Changed book's availability.
Clear returnDate.

In this state diagram we tell book status. First time book created in system we add some information about book. Then when user apply for book book’s availability status is change and system add a returnDate. When user to return the book system made book status available and clear the returnDate.

3.5 Sequence Diagram

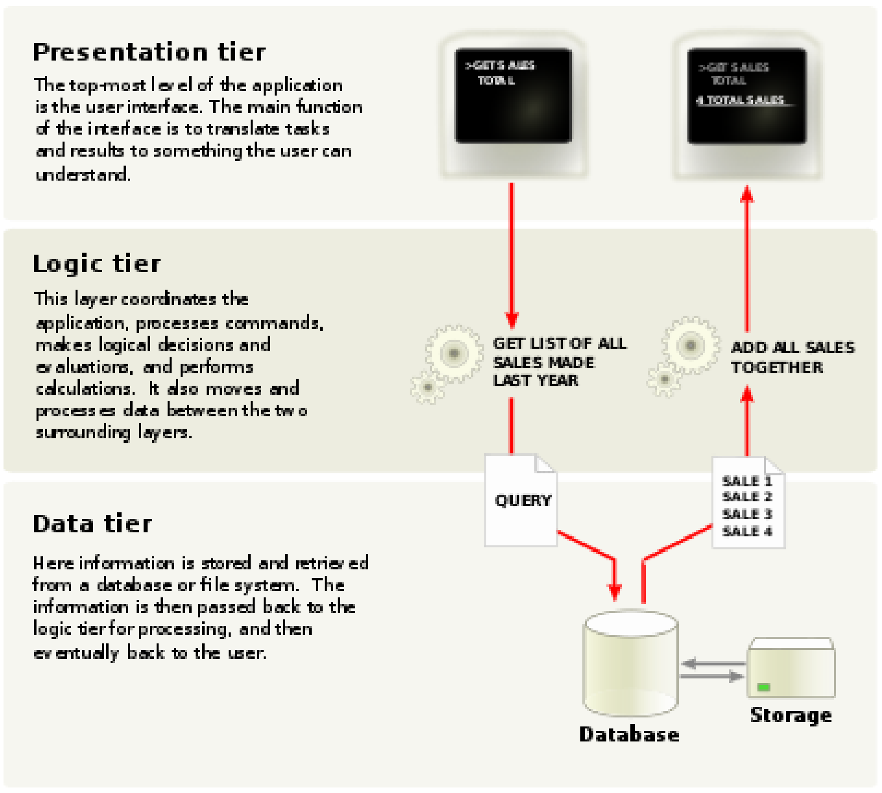


In this diagram our objects ae computer main system and book. First action is logged in system by user then system check the information about the user ıf the information true user can apply for books. Second action is applying book. In this diagram we calculate best case scenario so user applied book and system confirm book in the system and return book can borrow. So computer return this information to user. The object book just change its availability.

CHAPTER FOUR

IMPLEMENTATIONS

1) Project Structure



The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

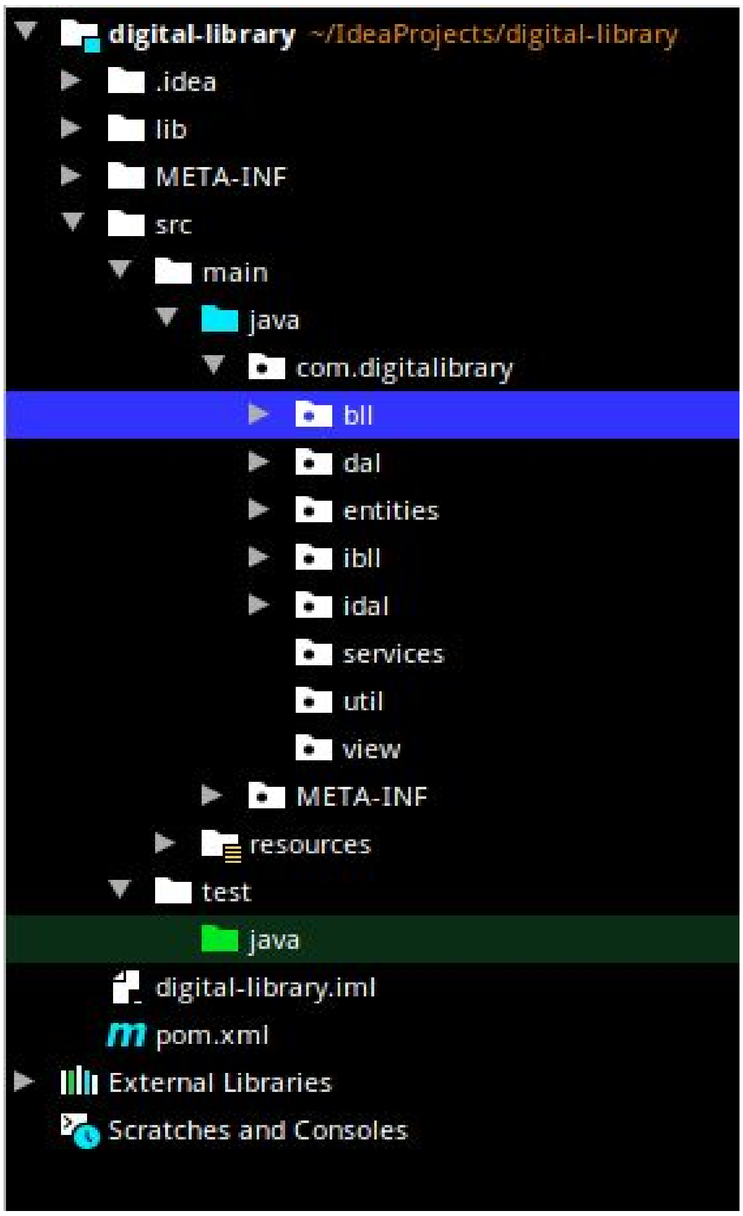
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

Database connections are indispensable in our projects, it will not be very accurate to load the entire load UI (user interface) when we want to perform operations such as pulling, adding, deleting, updating data from the database.

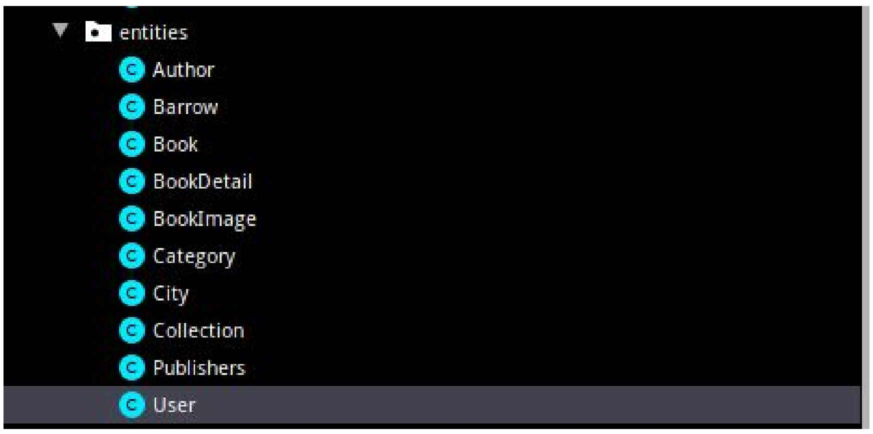
At this stage, the layered architecture will be activated, and the data access layer will be BLL (Busines Logic Layer) layer and the UI (User Interface) layer will be reproduced and load sharing can be made. The layers will communicate with each other by referencing, so the data flow will be under control and more manageable.

As seen in the picture, we preferred to use n-tier architecture pattern in our project. The reason for this was to achieve a more readable and refoctable structure. N-tier architecture Entities consists of Data Layer Logic and Businiues Layer Logic. We also added the service layer, but we did not implement it. This is because we have added User interface yet. Service layer is our User interface controller.



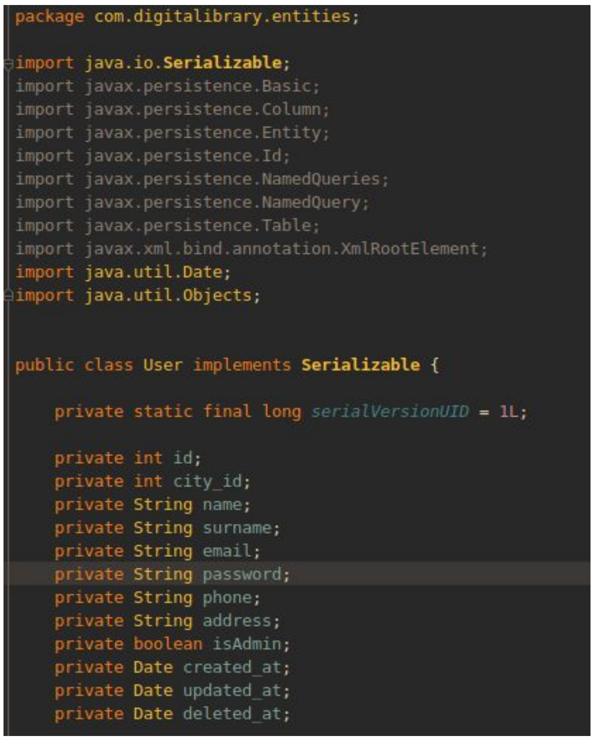
```
▼ digital-library ~/IdeaProjects/digital-library
    ▶ .idea
    ▶ lib
    ▶ META-INF
    ▼ src
        ▼ main
            ▼ java
                ▼ com.digitalibrary
                    ▶ bll
                    ▶ dal
                    ▶ entities
                    ▶ ibll
                    ▶ idal
                      services
                      util
                      view
                ▶ META-INF
            ▶ resources
        ▼ test
              java
      digital-library.iml
      pom.xml
▶ External Libraries
  Scratches and Consoles
```

2) Entities



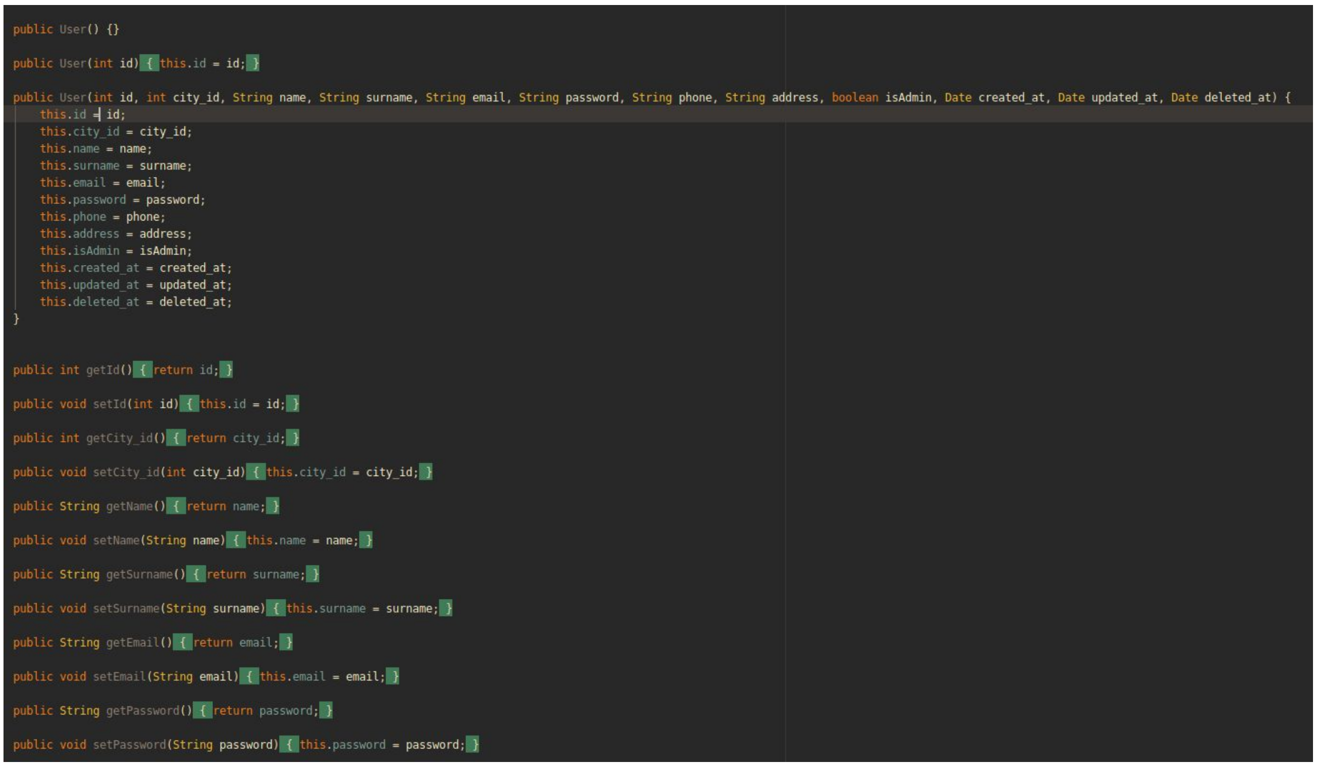
We have implements our previously mentioned classes in the entities package. We have not yet implemented the JAVA implementation because we have the idea to turn the project into an API-based one. To give an example, let's show our preme on the main scheme of the USER class.

First of all, we have implemented our attributes.

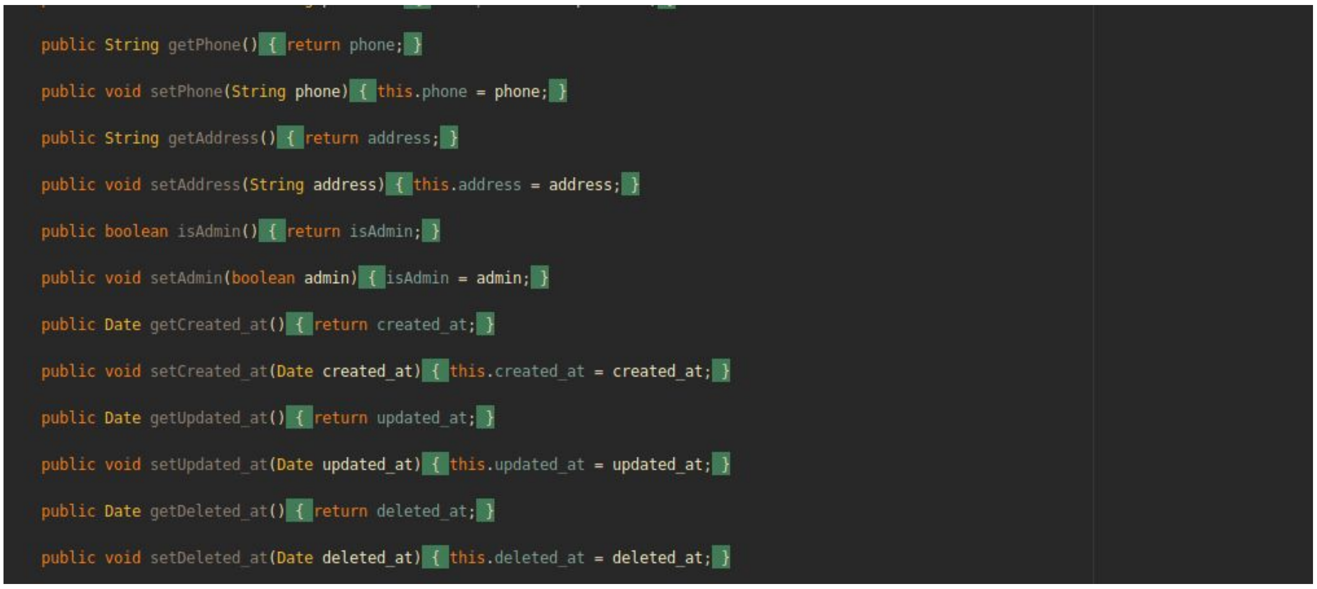


```java
package com.digitalibrary.entities;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;
import java.util.Date;
import java.util.Objects;


public class User implements Serializable {

    private static final long serialVersionUID = 1L;

    private int id;
    private int city_id;
    private String name;
    private String surname;
    private String email;
    private String password;
    private String phone;
    private String address;
    private boolean isAdmin;
    private Date created_at;
    private Date updated_at;
    private Date deleted_at;
```

Then we added our constructor and getter / setter methods.

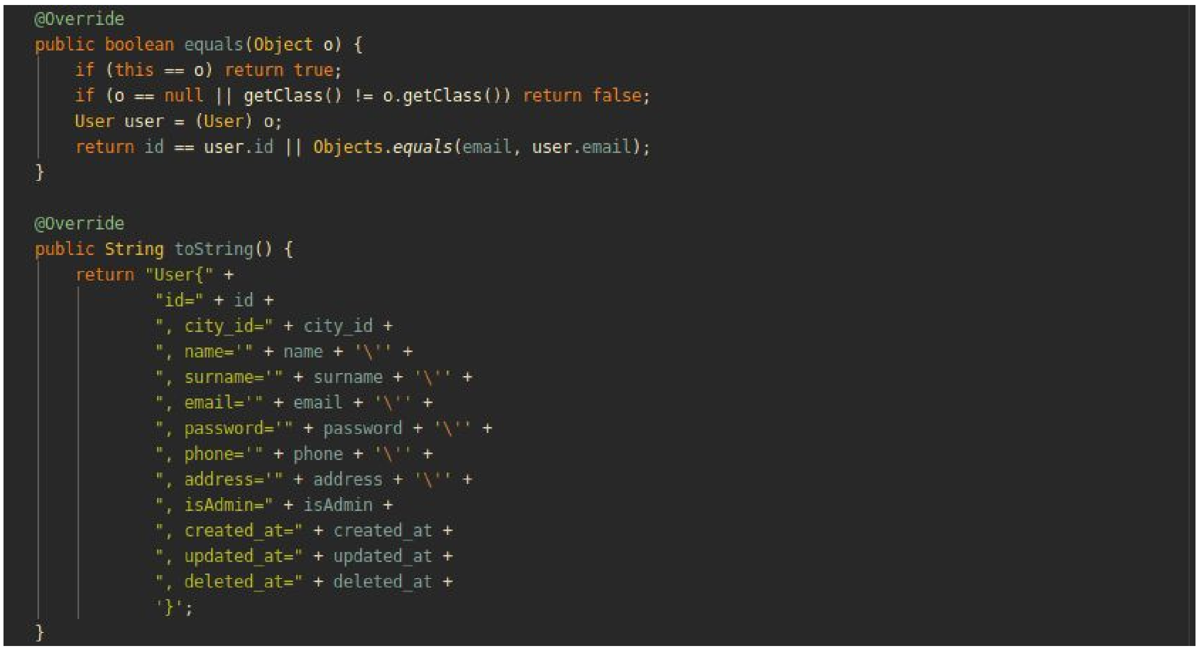


```java
public User() {}

public User(int id) { this.id = id; }

public User(int id, int city_id, String name, String surname, String email, String password, String phone, String address, boolean isAdmin, Date created_at, Date updated_at, Date deleted_at) {
    this.id = id;
    this.city_id = city_id;
    this.name = name;
    this.surname = surname;
    this.email = email;
    this.password = password;
    this.phone = phone;
    this.address = address;
    this.isAdmin = isAdmin;
    this.created_at = created_at;
    this.updated_at = updated_at;
    this.deleted_at = deleted_at;
}


public int getId() { return id; }

public void setId(int id) { this.id = id; }

public int getCity_id() { return city_id; }

public void setCity_id(int city_id) { this.city_id = city_id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getSurname() { return surname; }

public void setSurname(String surname) { this.surname = surname; }

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

public String getPassword() { return password; }

public void setPassword(String password) { this.password = password; }
```



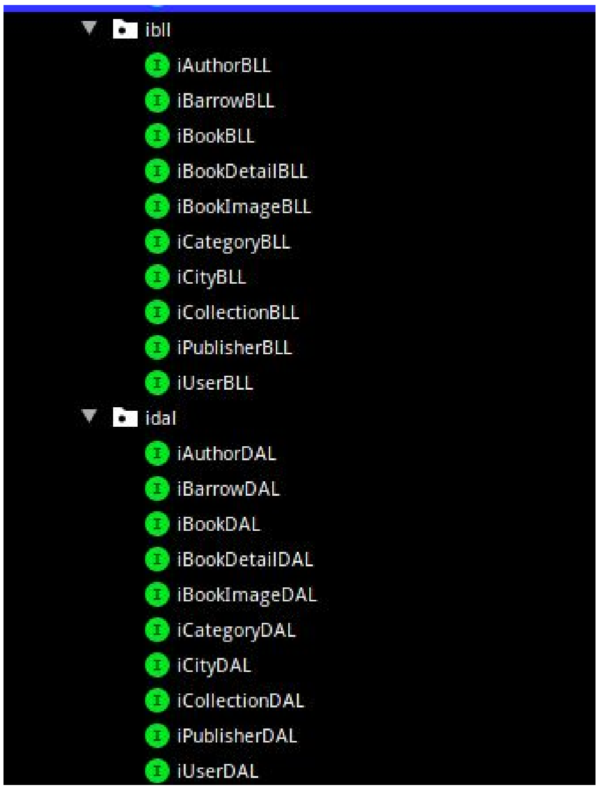
```java
public String getPhone() { return phone; }

public void setPhone(String phone) { this.phone = phone; }

public String getAddress() { return address; }

public void setAddress(String address) { this.address = address; }

public boolean isAdmin() { return isAdmin; }

public void setAdmin(boolean admin) { isAdmin = admin; }

public Date getCreated_at() { return created_at; }

public void setCreated_at(Date created_at) { this.created_at = created_at; }

public Date getUpdated_at() { return updated_at; }

public void setUpdated_at(Date updated_at) { this.updated_at = updated_at; }

public Date getDeleted_at() { return deleted_at; }

public void setDeleted_at(Date deleted_at) { this.deleted_at = deleted_at; }
```

Finally, we override our EQUALS and TOSTRING methods.



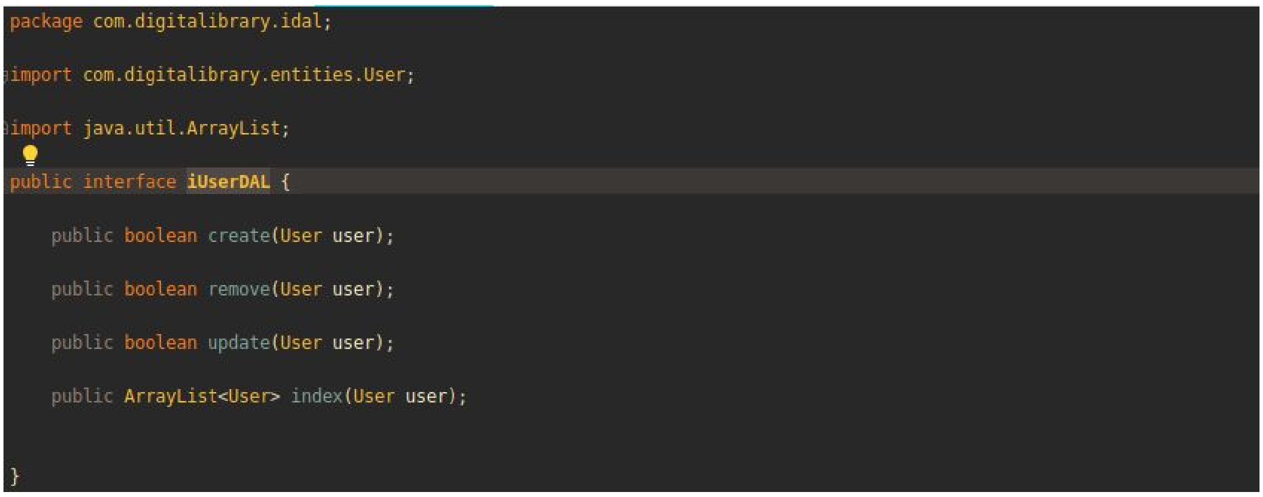
```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    User user = (User) o;
    return id == user.id || Objects.equals(email, user.email);
}

@Override
public String toString() {
    return "User{" +
            "id=" + id +
            ", city_id=" + city_id +
            ", name='" + name + '\'' +
            ", surname='" + surname + '\'' +
            ", email='" + email + '\'' +
            ", password='" + password + '\'' +
            ", phone='" + phone + '\'' +
            ", address='" + address + '\'' +
            ", isAdmin=" + isAdmin +
            ", created_at=" + created_at +
            ", updated_at=" + updated_at +
            ", deleted_at=" + deleted_at +
            '}';
}
```

Our user entity / model is ready. Then we would set the interface classes of our Data Access Layer and Business Logic Layer.



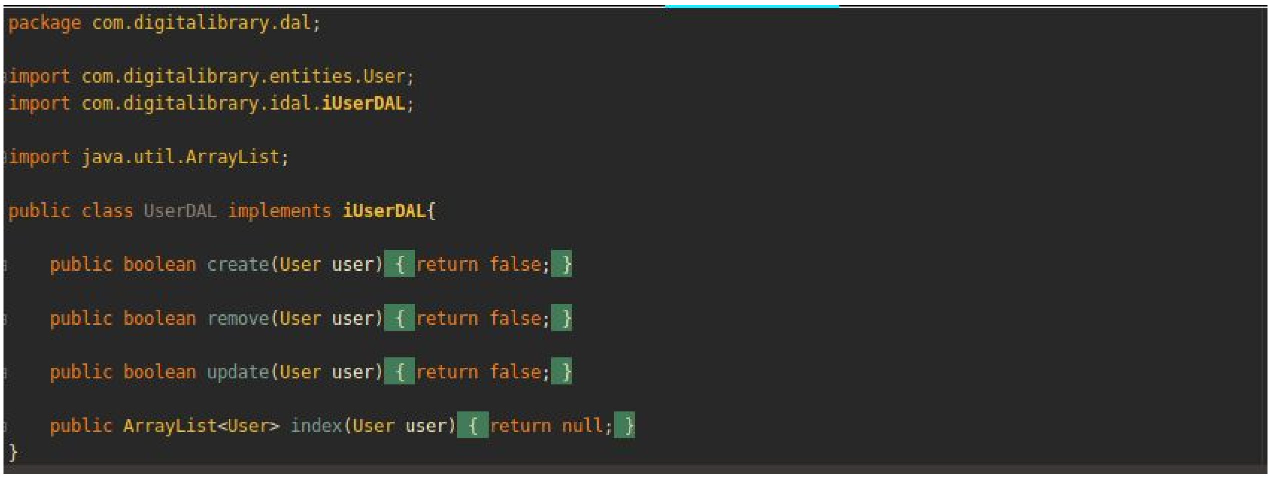
```
▼ ibll
    iAuthorBLL
    iBarrowBLL
    iBookBLL
    iBookDetailBLL
    iBookImageBLL
    iCategoryBLL
    iCityBLL
    iCollectionBLL
    iPublisherBLL
    iUserBLL
▼ idal
    iAuthorDAL
    iBarrowDAL
    iBookDAL
    iBookDetailDAL
    iBookImageDAL
    iCategoryDAL
    iCityDAL
    iCollectionDAL
    iPublisherDAL
    iUserDAL
```

We implemented only CRUD operations in BLL and DAL interfaces.



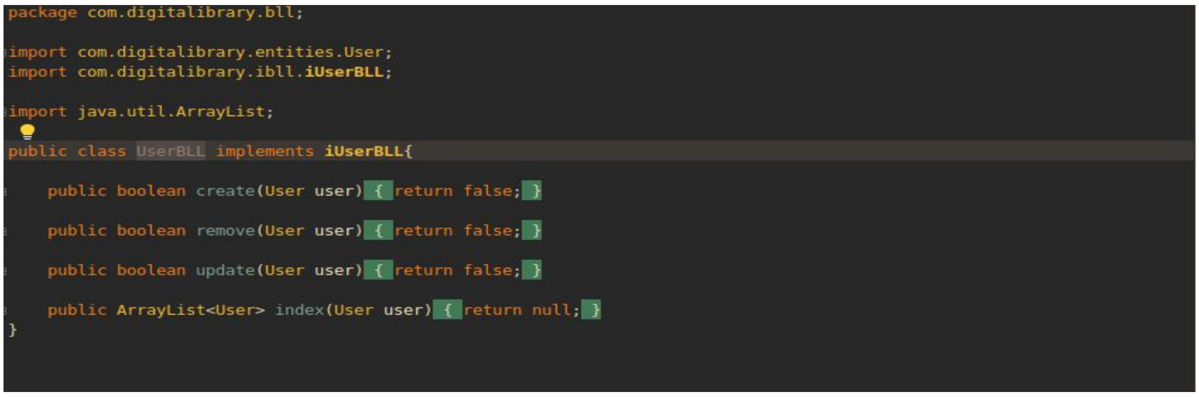
```java
package com.digitalibrary.idal;

import com.digitalibrary.entities.User;

import java.util.ArrayList;

public interface iUserDAL {

    public boolean create(User user);

    public boolean remove(User user);

    public boolean update(User user);

    public ArrayList<User> index(User user);


}
```

After then in the UserDAL class in the DAL packages, we implements iUserDAL interfaces. Our purpose in the UserDAL class, connection with DB and data operations. So, we'ill use JPA in DAL classes. If operations are success return true to BLL layer. So we can distributed control our data and our workflow operations. If user will added successfully, then BLL chose after response and operations.



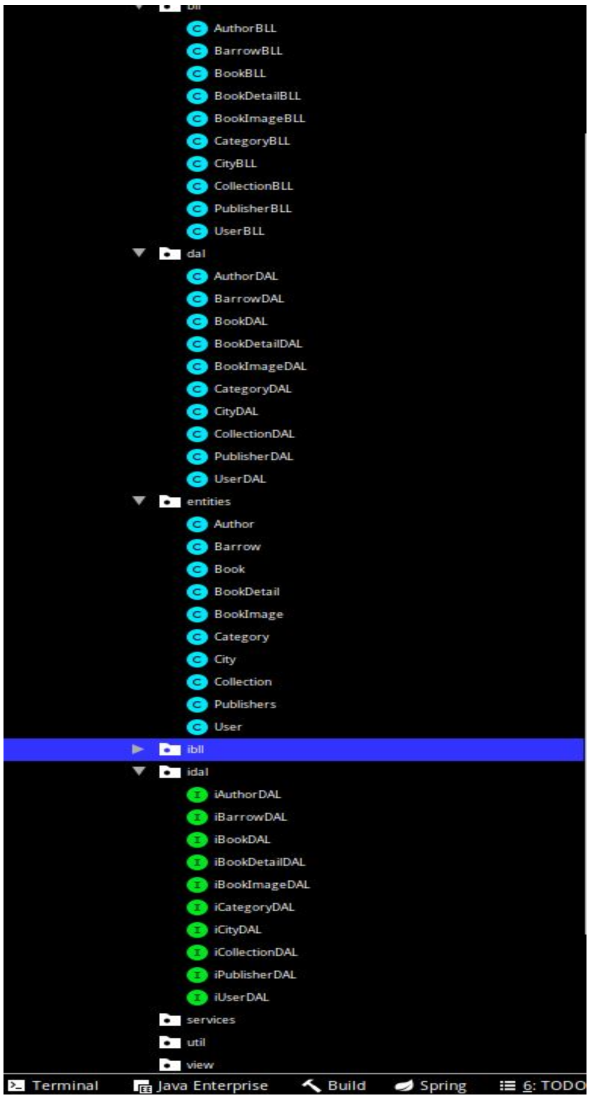
```java
package com.digitalibrary.dal;

import com.digitalibrary.entities.User;
import com.digitalibrary.idal.iUserDAL;

import java.util.ArrayList;

public class UserDAL implements iUserDAL{

    public boolean create(User user) { return false; }

    public boolean remove(User user) { return false; }

    public boolean update(User user) { return false; }

    public ArrayList<User> index(User user) { return null; }
}
```

UserBLL implementation is the same. But we explain our data workflow above.



```
package com.digitalibrary.bll;

import com.digitalibrary.entities.User;
import com.digitalibrary.ibll.iUserBLL;

import java.util.ArrayList;

public class UserBLL implements iUserBLL{

    public boolean create(User user) { return false; }

    public boolean remove(User user) { return false; }

    public boolean update(User user) { return false; }

    public ArrayList<User> index(User user) { return null; }
}
```

Some of classes and interfaces;



```
        AuthorBLL
        BarrowBLL
        BookBLL
        BookDetailBLL
        BookImageBLL
        CategoryBLL
        CityBLL
        CollectionBLL
        PublisherBLL
        UserBLL
    dal
        AuthorDAL
        BarrowDAL
        BookDAL
        BookDetailDAL
        BookImageDAL
        CategoryDAL
        CityDAL
        CollectionDAL
        PublisherDAL
        UserDAL
    entities
        Author
        Barrow
        Book
        BookDetail
        BookImage
        Category
        City
        Collection
        Publishers
        User
    ibll
    idal
        iAuthorDAL
        iBarrowDAL
        iBookDAL
        iBookDetailDAL
        iBookImageDAL
        iCategoryDAL
        iCityDAL
        iCollectionDAL
        iPublisherDAL
        iUserDAL
    services
    util
    view
```