LIBMAN REPORT

Service Oriented Architecture

COURSE PROJECT

Instructor: Dr. Võ Đình Hiếu

Team members

- Nguyễn Thạc Thống Student ID: 12020624
- Nguyễn Thị Thủy Student ID: 12020377
- Ngô Thị Thúy Lan Student ID: 12020212

Table of Contents

1. Requirements

We need to build a simple library application which provides services for two types of users: students and librarians.

- Librarians use the program that run on the desktop computer to perform these tasks:
    - View the list of all existing books
    - Add a new book to the list
    - Delete an existing book in the list
    - Edit information of a book
- Students can see list and details of books through the Web interface and Android application.

2. Analysis

- LibMan is a simple library management system which is based on Web Services Architecture.
- LibMan consists three parts including web application, desktop application and mobile application. The web and mobile application are for students, and the desktop application, with more functions, is for librarians.

Flow chart:

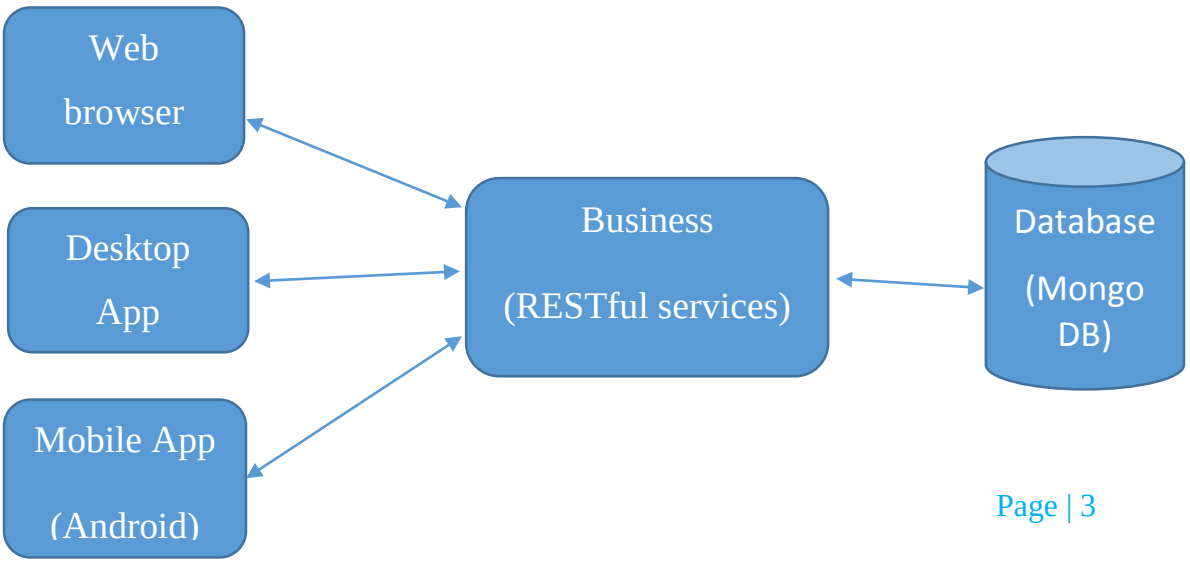
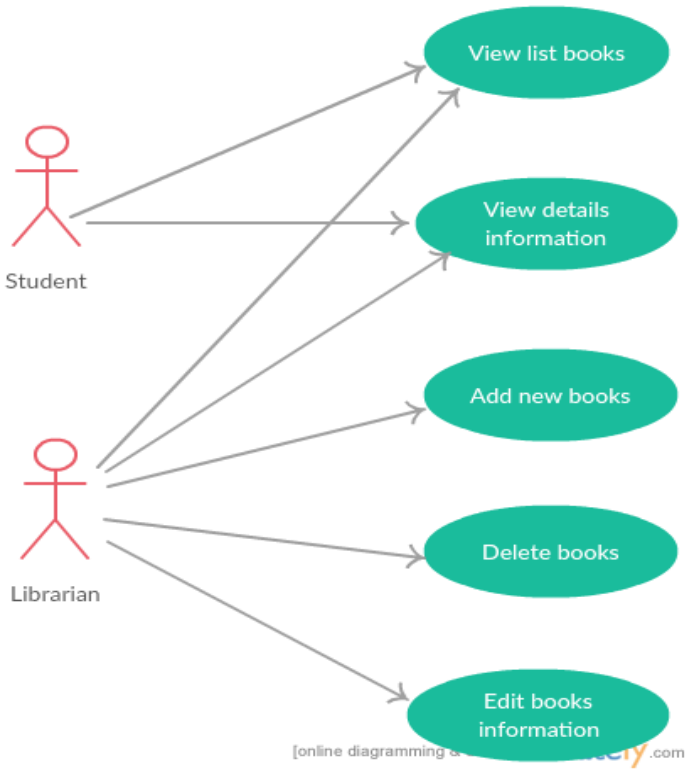


Table of tasks that depend on actors:

Actor	Task
Students	- View list of books - View book information - Doing in Web browser/ Mobile App
Employees	- View list of books - View book's information - Add new books - Delete books from list - Edit book information - Doing in Web browser/ Desktop App

Use case diagram:



3. *Technical*

a) Web Service

We use **RESTful web service architecture** (a method using web services to create a simple way to exchange data and independent platform with high performance) in combination with **Spring MVC framework**.

- **RESTful web services**

  RESTful web services are built to work best on the Web. Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modify ability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers

exchange representations of resources by using a standardized interface and protocol.

- **Spring MVC**

  The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

  - The **Model** encapsulates the application data and in general they will consist of POJO.
  - The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
  - The **Controller** is responsible for processing user requests and building appropriate model and passes it to the view for rendering.

Access this site for more information: *http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/mvc.html*

b) Database

We use **MongoDB** to manage our database.

- **MongoDB**

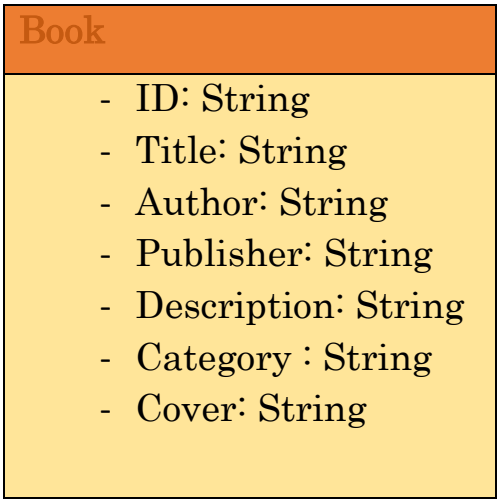
MongoDB is a cross – platform document – oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table – based relational database structure in favor of JSON – like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster.

MongoDB is free and open – source software.

- Apply in LibMan

We use only one table: Book

In this table, we define some attributes: ID, title, author, publisher, description, cover.



3. Rest API Implementations

Method	URL	Params	Description
GET	api/books		Get list of all existing books **Response:** : [{ id : id, title: title, author: author, description: description }, ….]

GET	api/book	**id** : string	Get information of a specific book
			Response:
			{
			id : id,
			title: title,
			author: author,
			description: description
			}

POST	api/add	**id**: string **title**: string **author**: string **publisher**: string **description**: string **category**: string **cover** : string	Add a new book or update book with specified id.
POST	api/book/delete	**id**: string	Delete an existing book with specified id

- **View list books**

```
@RequestMapping(value = "/", method = RequestMethod.GET)
public ModelAndView home() {
    List<Book> books = bookRepo.findAll();
    ModelAndView mv = new ModelAndView("home");
    mv.addObject("books", books);
    return mv;
}
```

- **Add methods**

  Add books method uses POST request method. It contains six request parameters which are distributed of a book: id, title, author, publisher, description and cover. Note that, two of six parameters are compulsory (id and title).

```
public void addBook(@RequestParam(value= "id", required=true) String id,
                    @RequestParam(value= "title", required=true) String title,
                    @RequestParam(value= "author", required=false)String author,
                    @RequestParam(value= "publisher", required=false)String publisher,
                    @RequestParam(value= "description", required=false)String description,
                    @RequestParam(value= "cover", required=false)String cover){
    logger.info("Inside addBook() method");
    Book book = new Book(id, title);
    book.setAuthor(author);
    book.setPublisher(publisher);
    book.setDescription(description);
    book.setCover(cover);
    bookRepo.save(book);
}
```

- **Delete method**

  Delete books method also uses POST request method. It contains only request parameter: id.

```
@RequestMapping(value="/book/delete", method= RequestMethod.POST)
@ResponseBody
public void delBook(@RequestParam(value= "id", required=true) String id){
    logger.info("Inside method delBook()");
    Book book = bookRepo.findOne(id);
    bookRepo.delete(book);
}
```

- **Edit method**

  Edit books method uses POST request method.

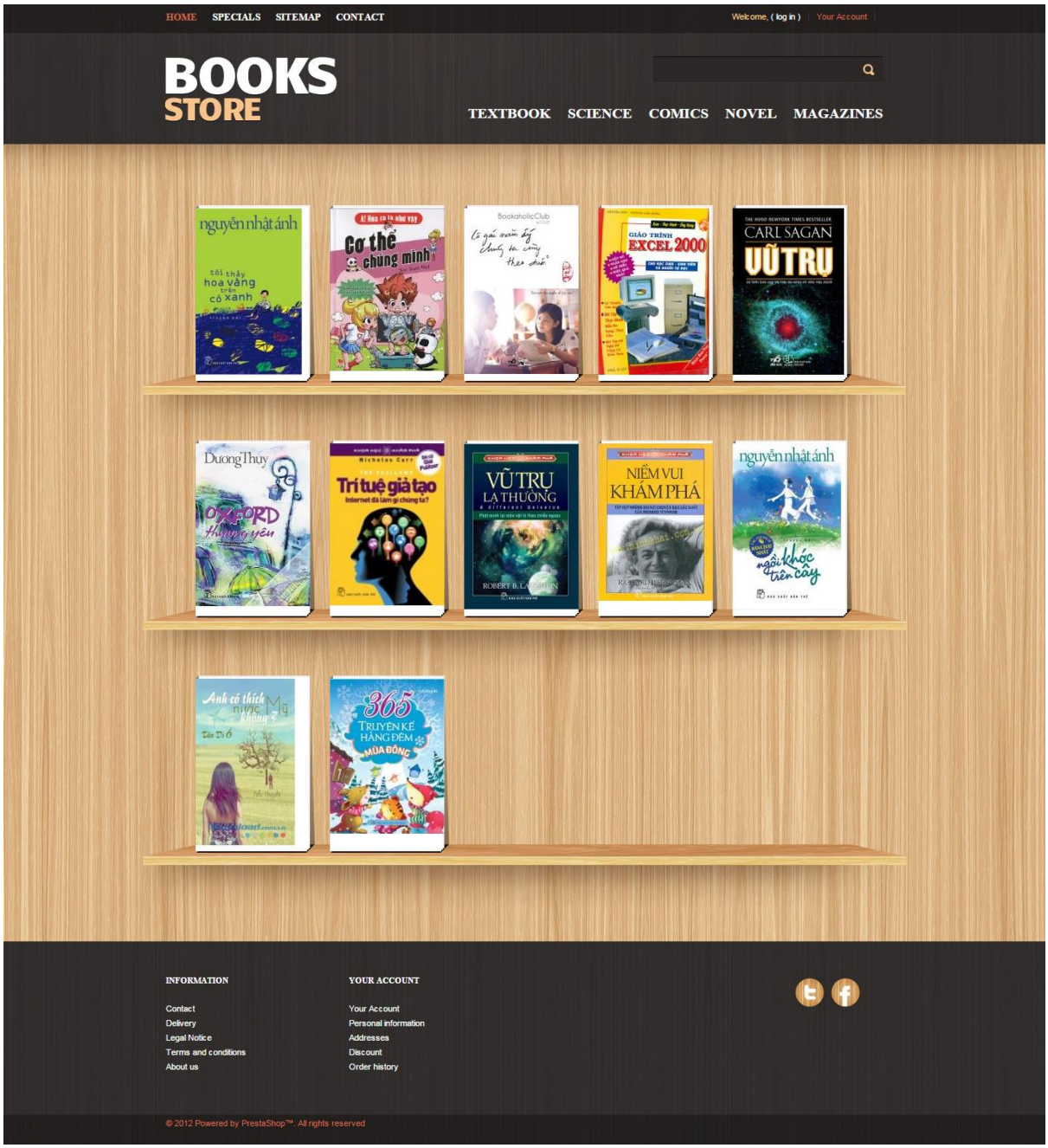
```
@RequestMapping(value="/book/update", method = RequestMethod.POST)
@ResponseBody
public void testBook(){
    logger.info("Inside testBook() method");
}
```

4. Screen shot

a) Website

Homepage : https://soalibman.herokuapp.com/

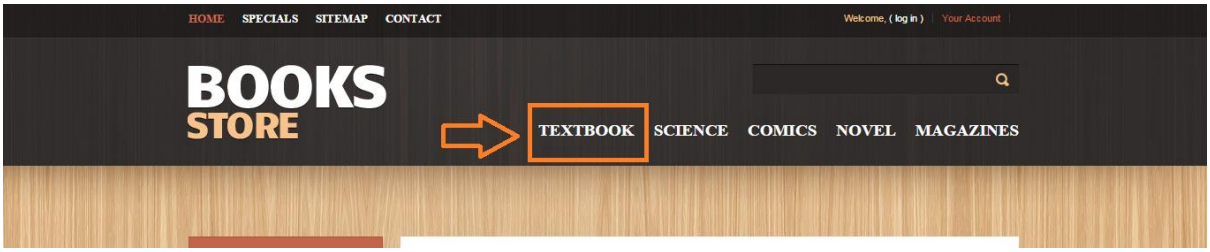
- View list of all existed books :



- View details of a book:

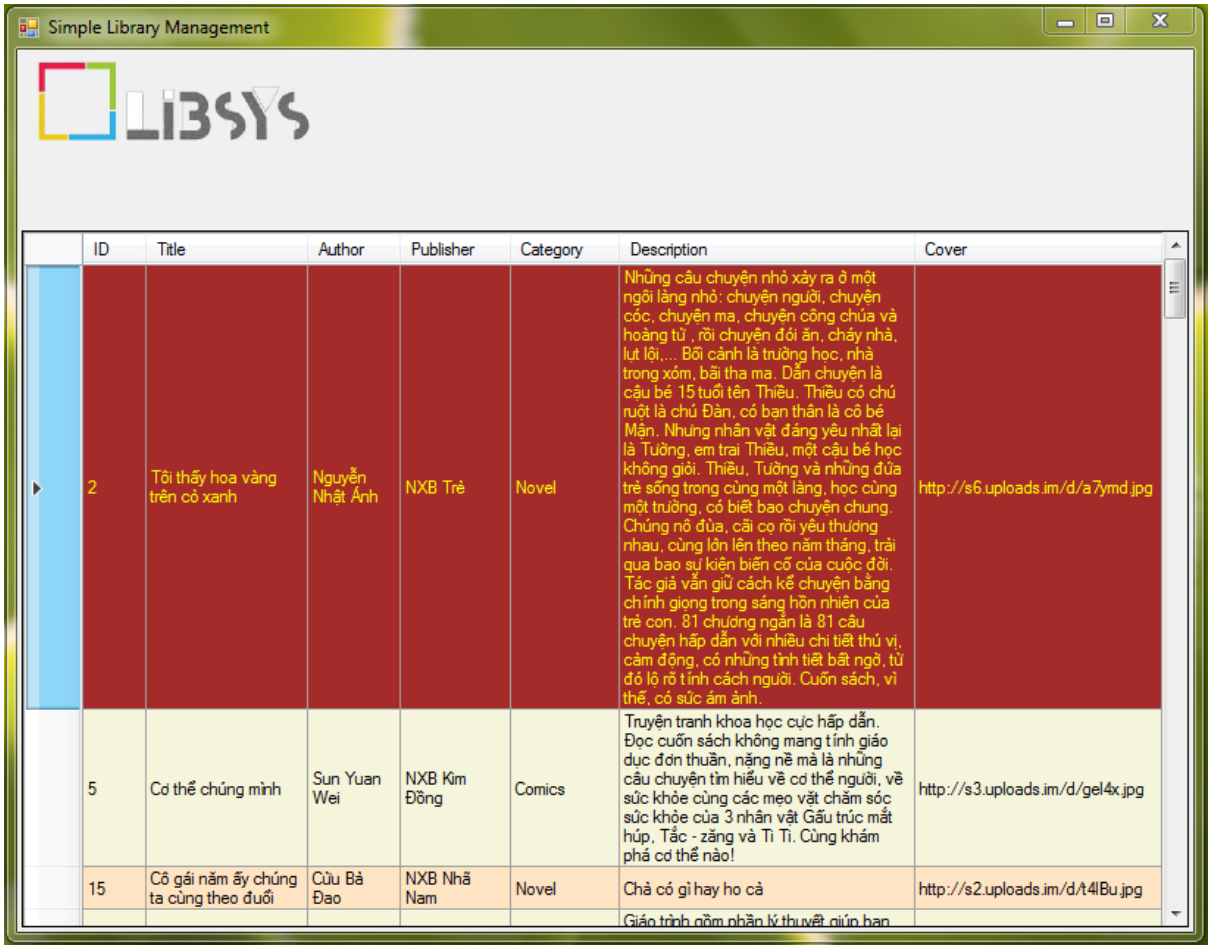
  https://soalibman.herokuapp.com/view/{book_id}

- View books by categories:

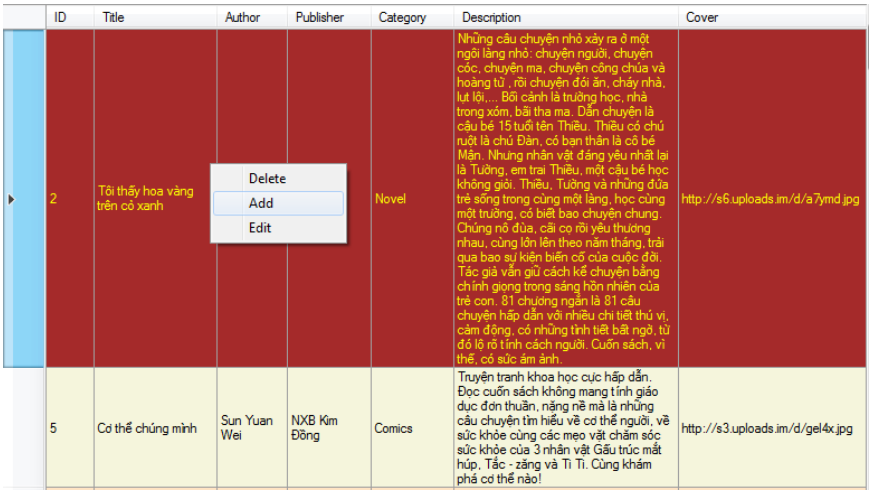


b) Desktop App

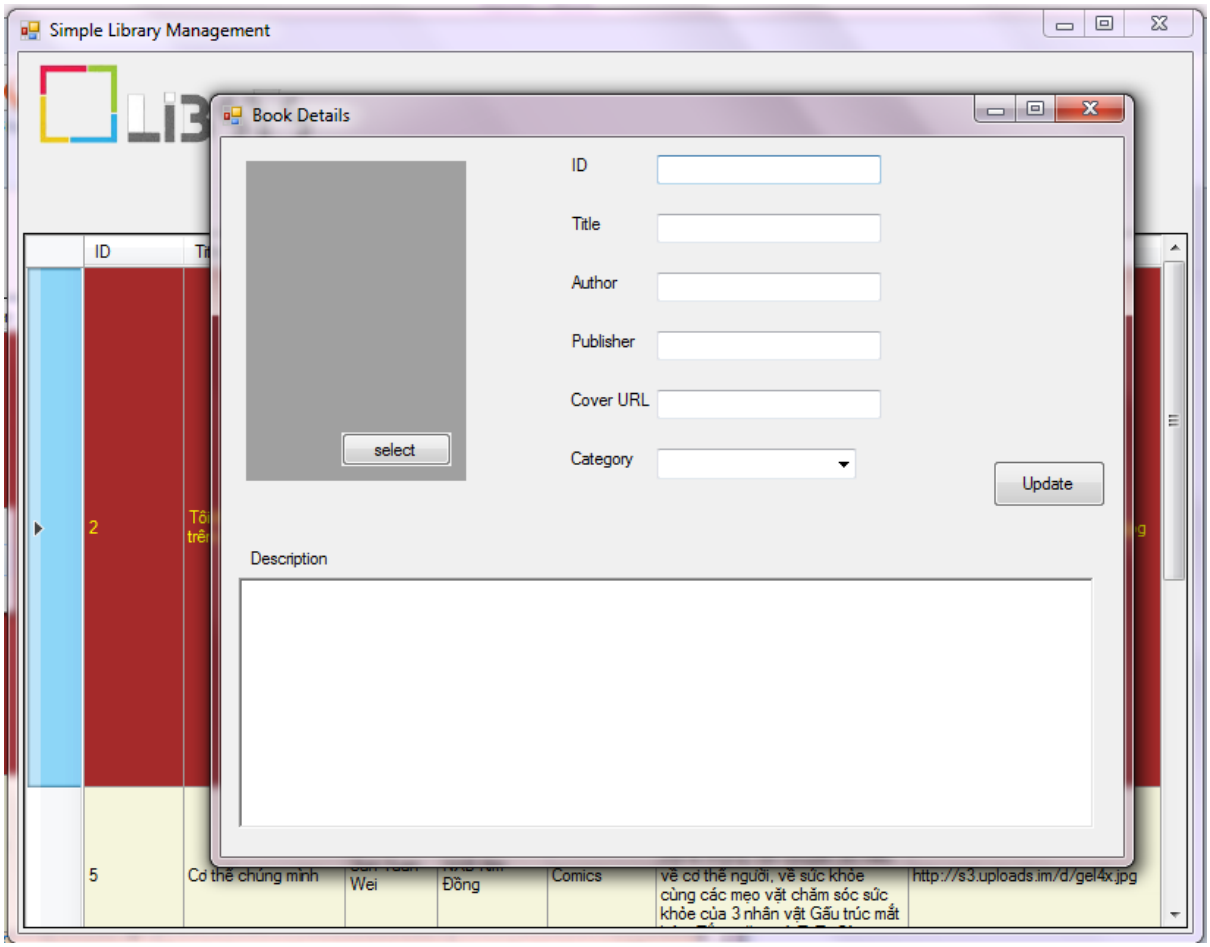
- **View list of existing books**



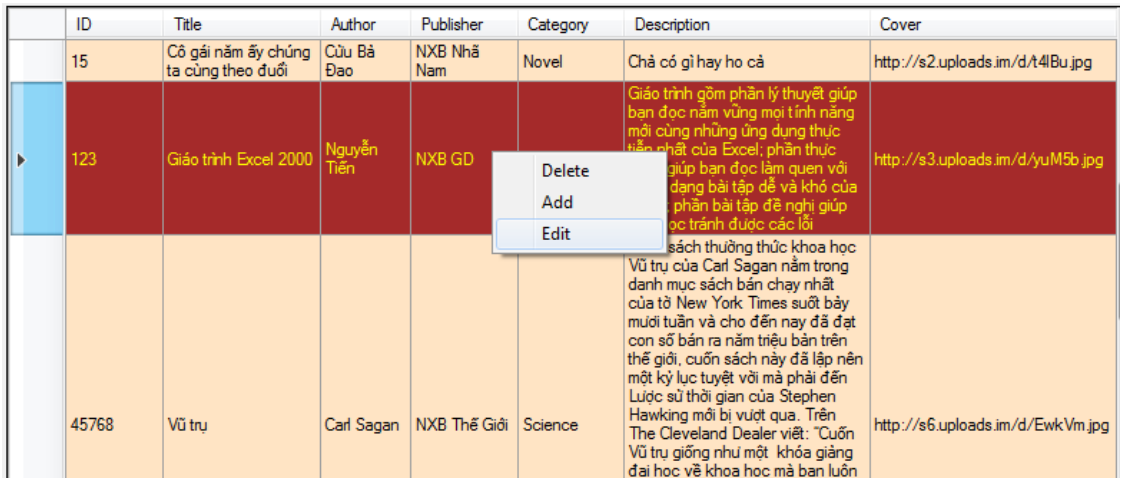
- **Add new books** : On the right-clicked menu of the table -> Click "Add"



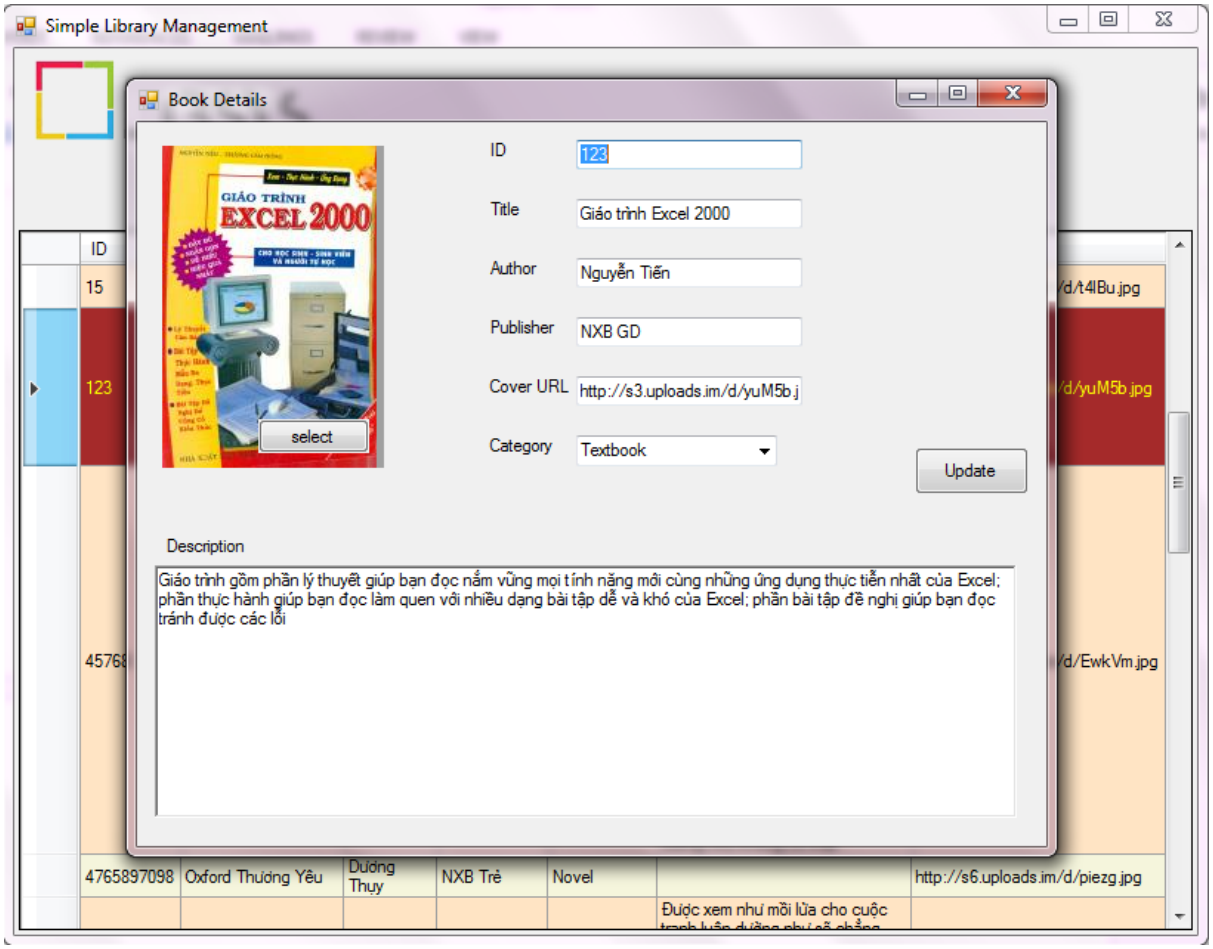
⇨ The book editor frame will be showed to add book information



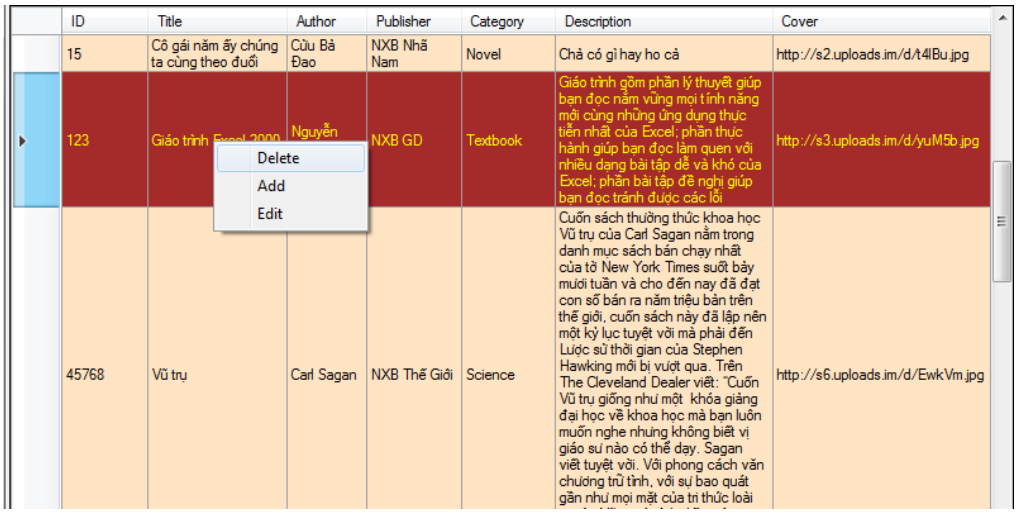
- **Edit book's information:** On the right-clicked menu of the table -> Click "Edit"



⇨ The book's information editor frame will be showed to edit.

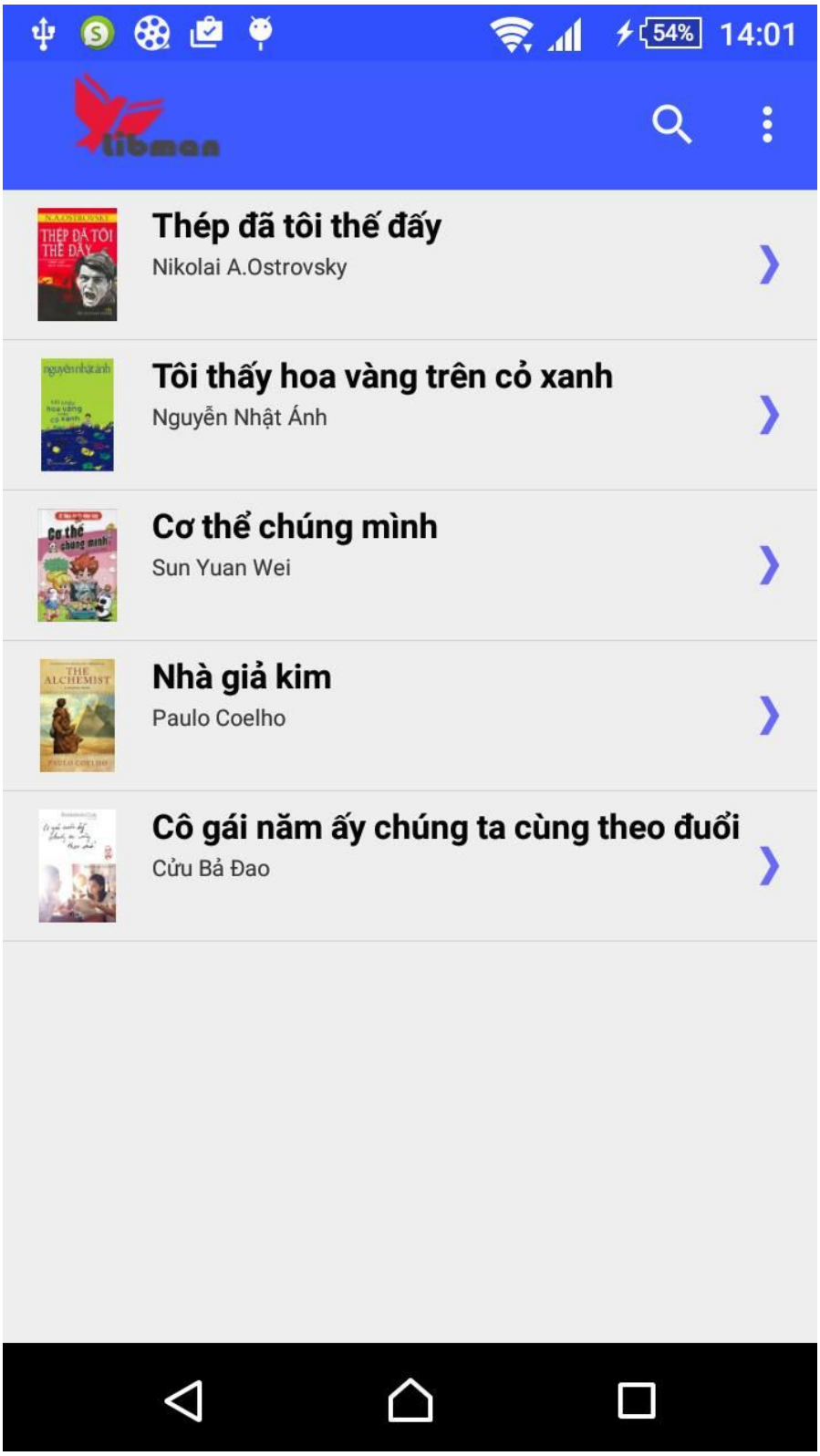


- **Delete a book** : Click to "Delete" items on right-clicked menu of the table to delete a books

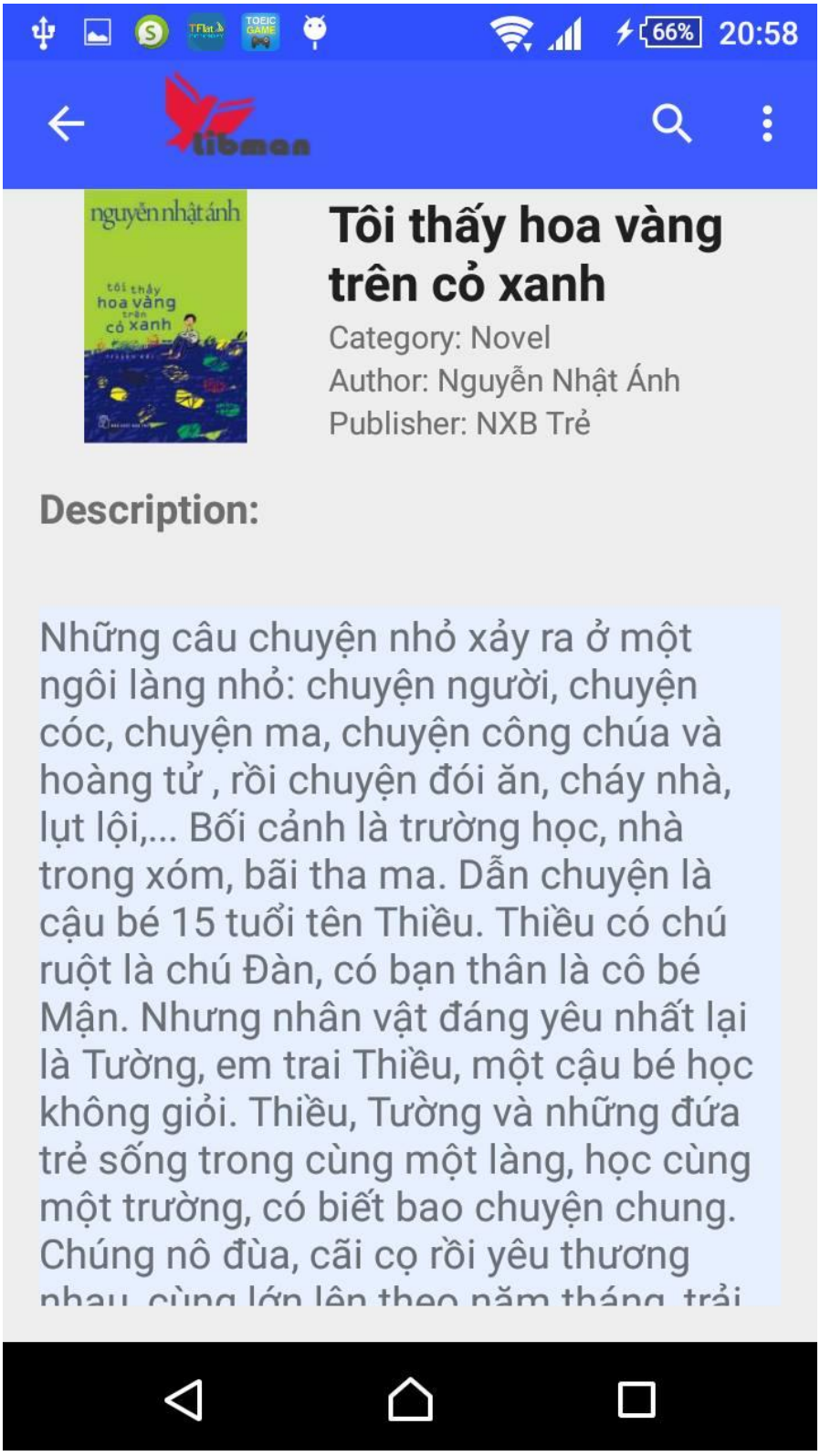


c) Android App

- **Viewing list of books**: When the mobile app is opened, a list of all books will be showed (as described in following screenshot)



- **Viewing details of a book**: Tap on each book to see its information about author, publisher, description



5. Summary

- Link Github : https://github.com/zatcsc/SOA-Project1
- All applications are included in “Final-Products” folder:
  - Desktop: Double click to “LibManDA.exe” to run desktop application
  - Android: Install “app-release.apk” on Android and run mobile app
  - Website : https://soalibman.herokuapp.com/
- Source code are stored in other folders