Virtual library manager

Analysis and Design Document

Student: Marian-Florin Salvan

**Group: 30238**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <07/Apr/17> | <1.0> | Design | Marian-Florin Salvan |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

Implement a client-server application that will allow users to manage their virtual libraries in a very useful way. The application will have two types of users:

* **The regular user** who can do the following operations**:**

1. Register/Log in
2. Browse the available books
3. Add new books
4. Keep track of the books read so far
5. Add/Remove book to read list/whish list
6. Rate/review read books
7. Recommend books to other users
8. Send messages to other users
9. Report other users

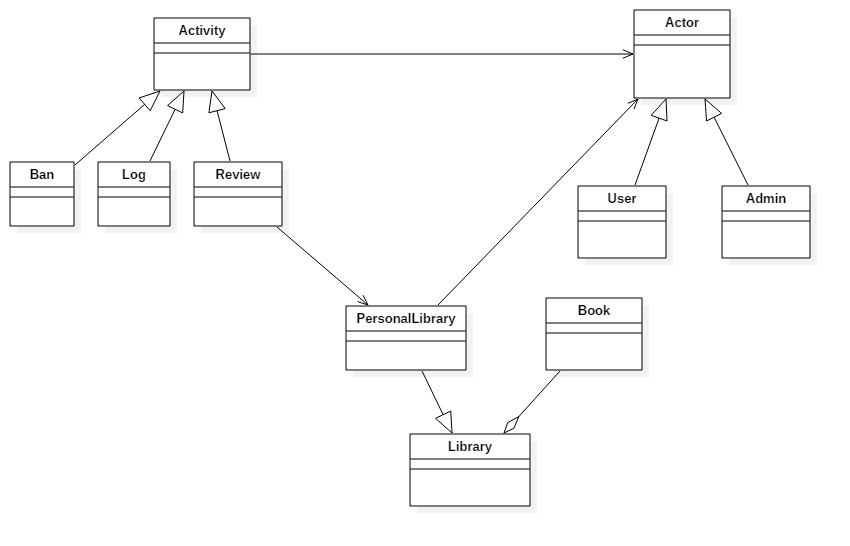
* **The system administrator** who can do the following operations:

1. Add books
2. See/Ban users
3. See reports
4. Delete reviews

# Elaboration – Iteration 1.1

# Domain Model

The domain model consists in a several abstractions of some real life objects like users, books, libraries and user activities.



# Architectural Design

## Conceptual Architecture

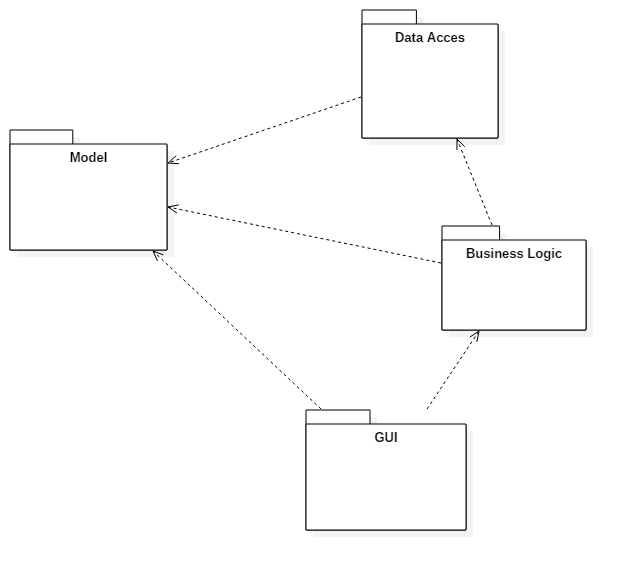
## For the implementation of this application I have chosen the Microservices architectural pattern using the API REST - based topology -t expose small, self-contained individual services through some sort of API.

**REST**:

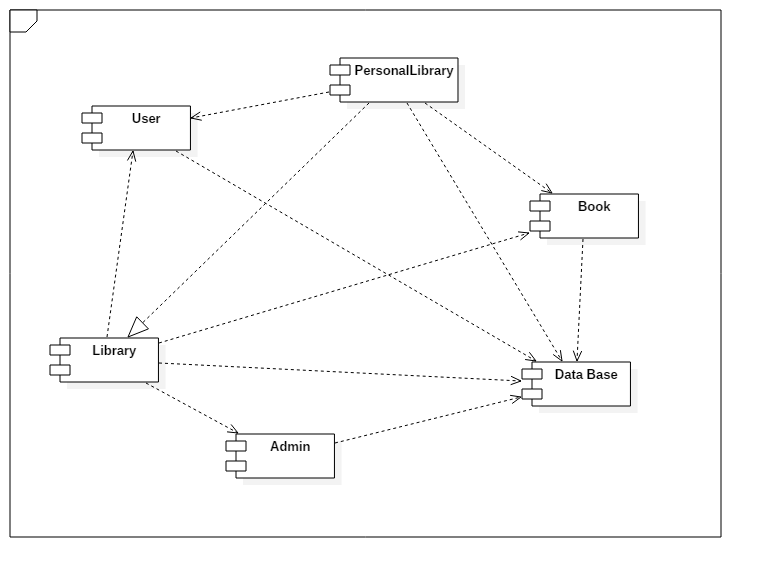
* REST is easy to understand: it uses HTTP and basic CRUD operations, so it is simple to write and document.
* REST also makes efficient use of bandwidth., REST is designed to be stateless and REST reads can be cached for better performance and scalability.
* REST supports many data formats, but the predominant use of JSON means better support for browser clients. JSON sets a standardized method for consuming API payloads, so you can take advantage of its connection with JavaScript and the browser.
* No expensive tools require to interact with the Web service
* Smaller learning curve
* Efficient (REST can use smaller message formats)
* Fast (no extensive processing required)
* Closer to other Web technologies in design philosophy

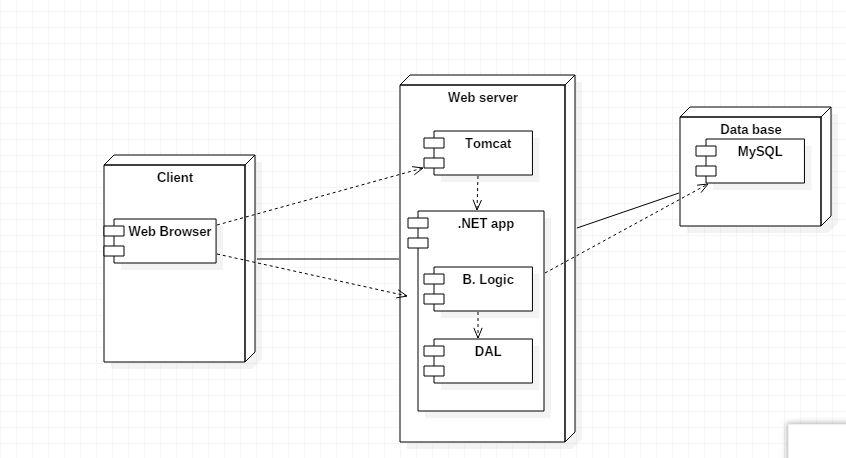
## 

## Package Design



## Component and Deployment Diagrams



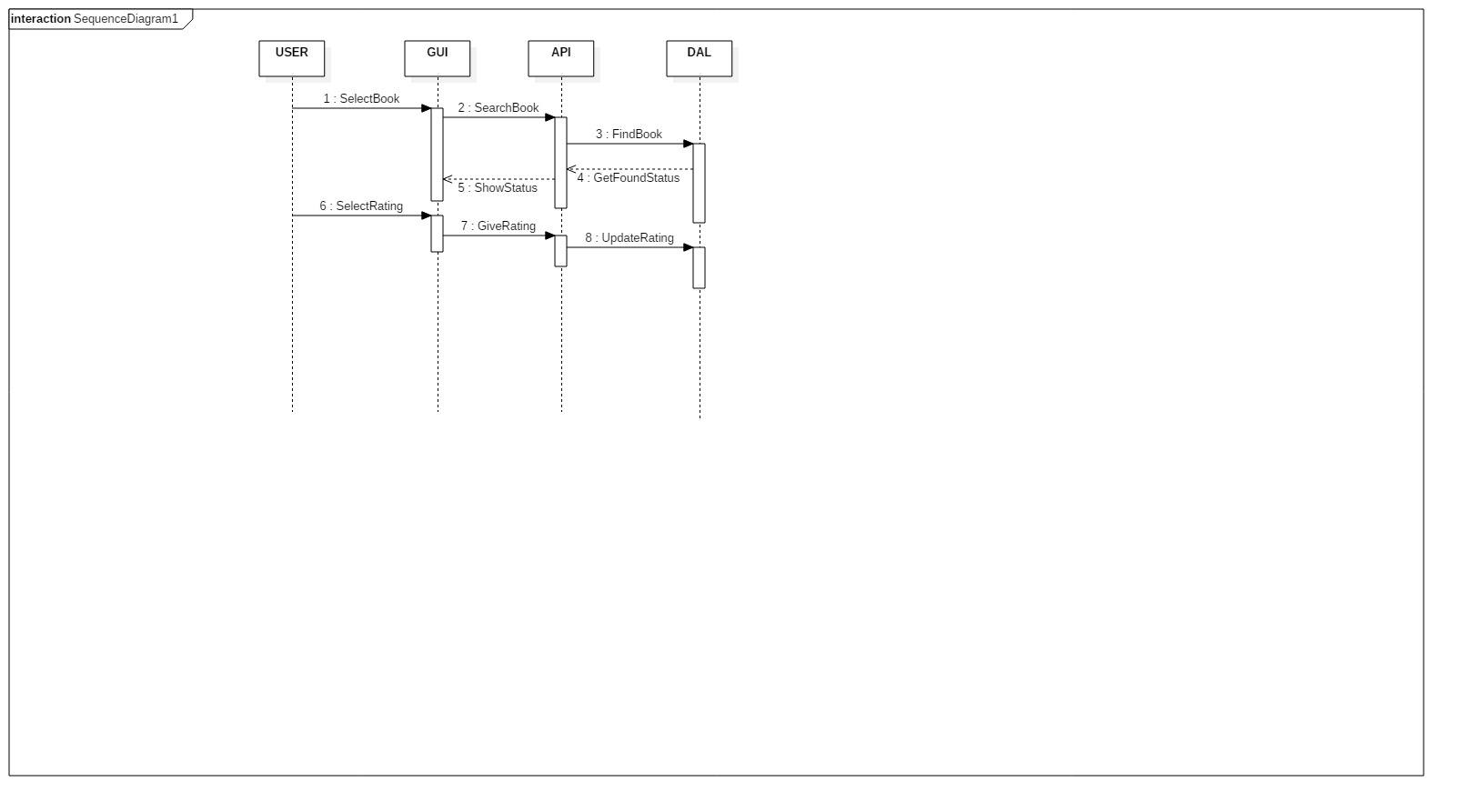


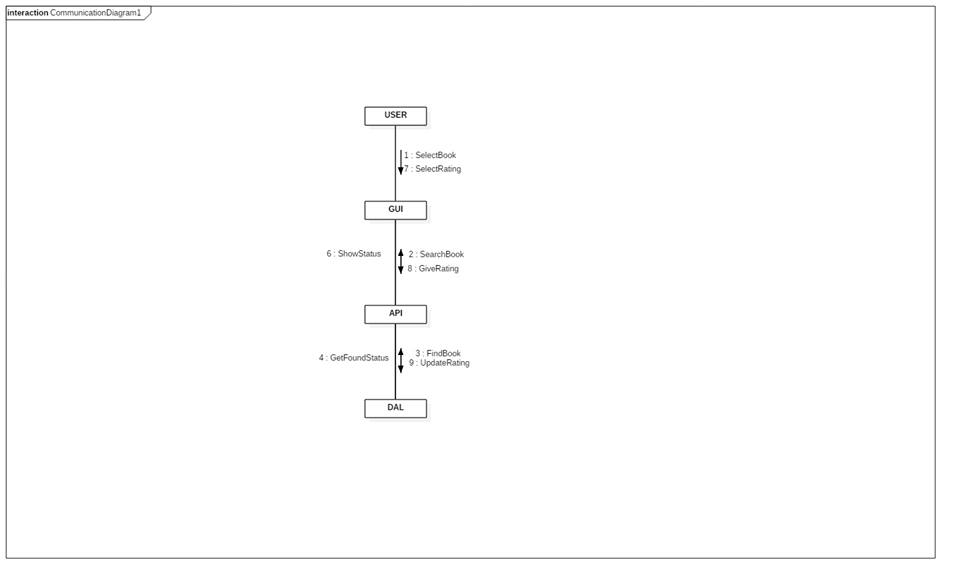
# Elaboration – Iteration 1.2

# Design Model

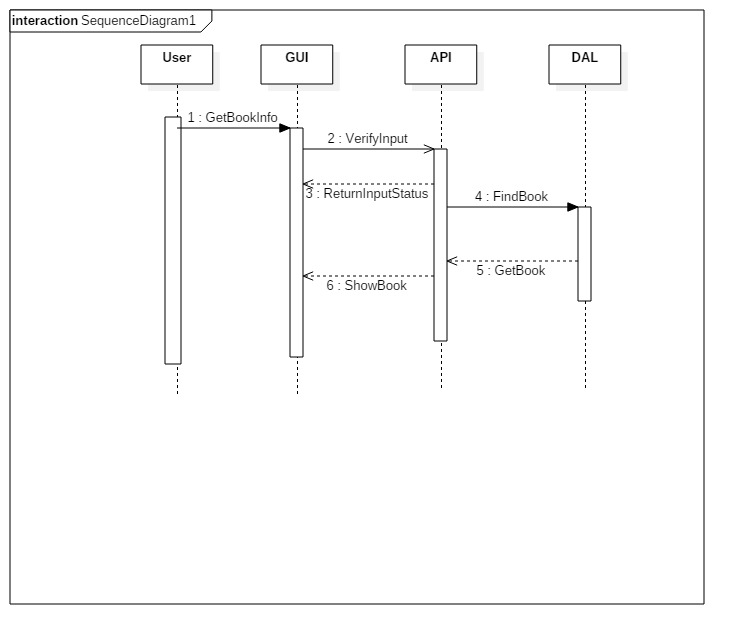
## Dynamic Behavior

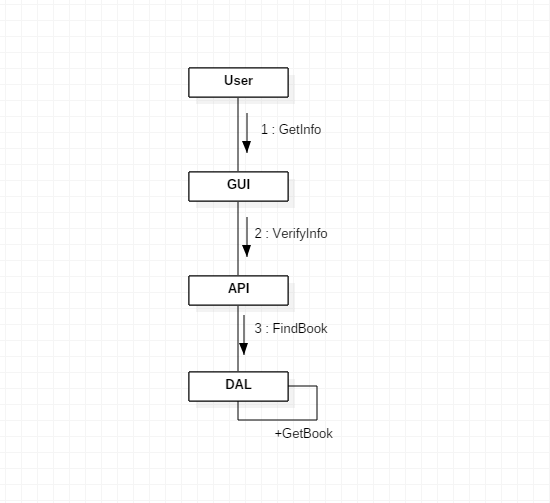
Book rating:



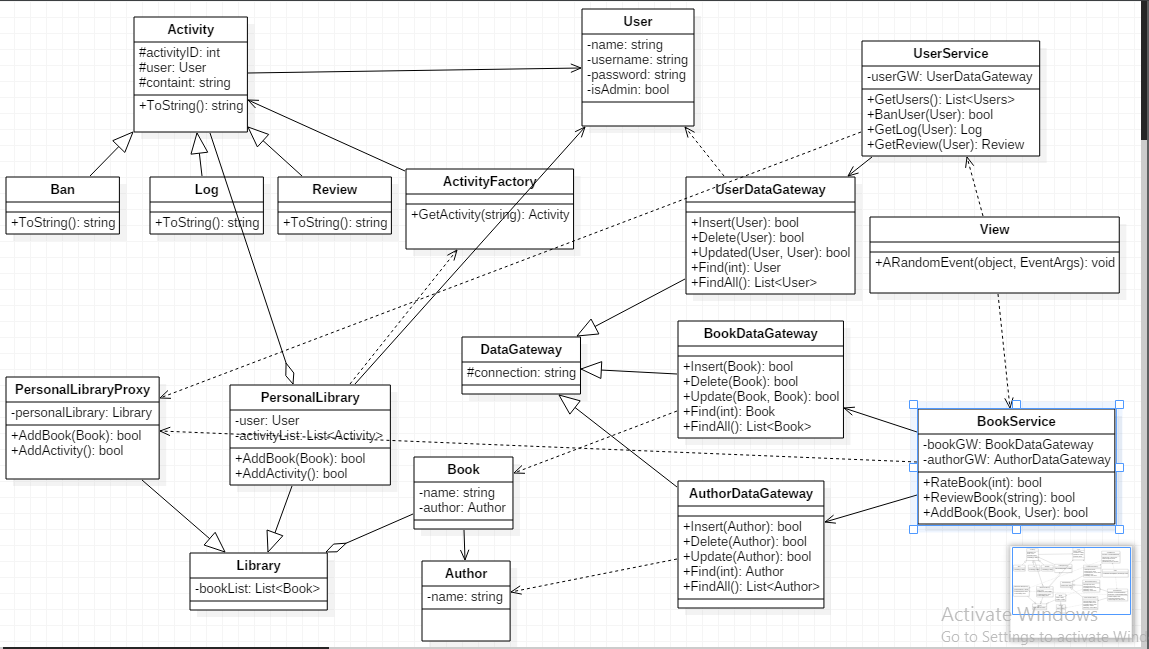


Find book:





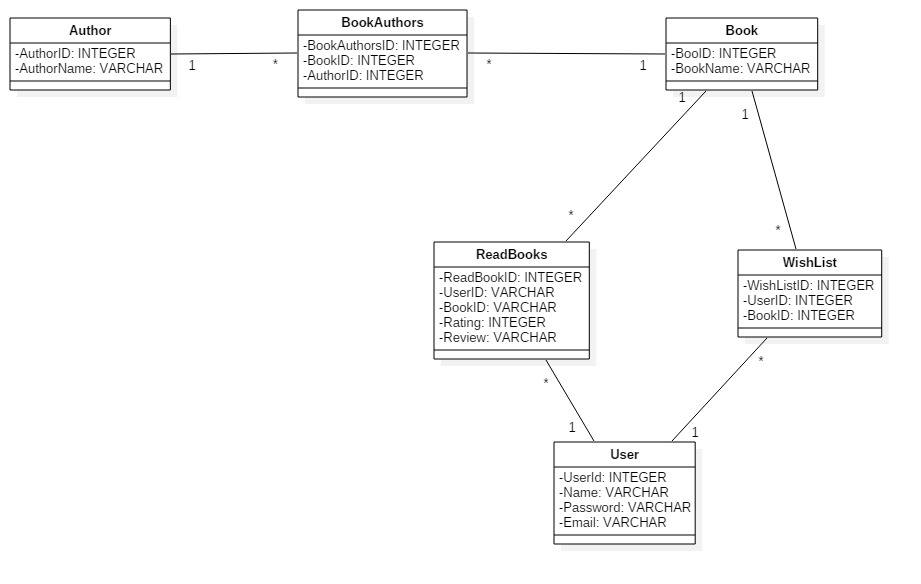
## Class Design



The next 2 design patterns were used:

* Factory design pattern (In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface, in our case what kind of activity the user does)
* Proxy design pattern (In proxy pattern, we create object having original object to interface its functionality to outer world, in our case PersonalLibraryProxy is used while PersonalLibrary’s logic stays hidden)

# Data Model



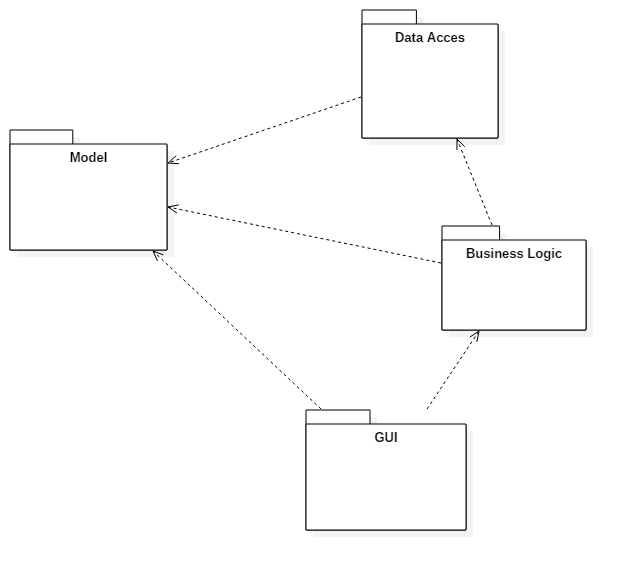
# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

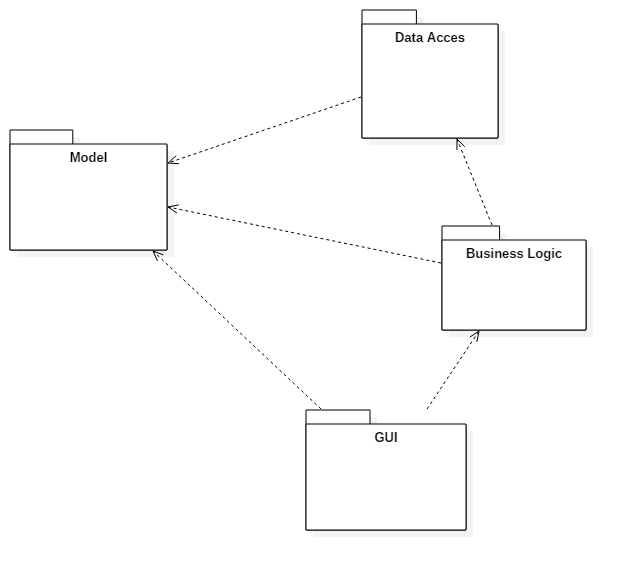
# Elaboration – Iteration 2

# Architectural Design Refinement

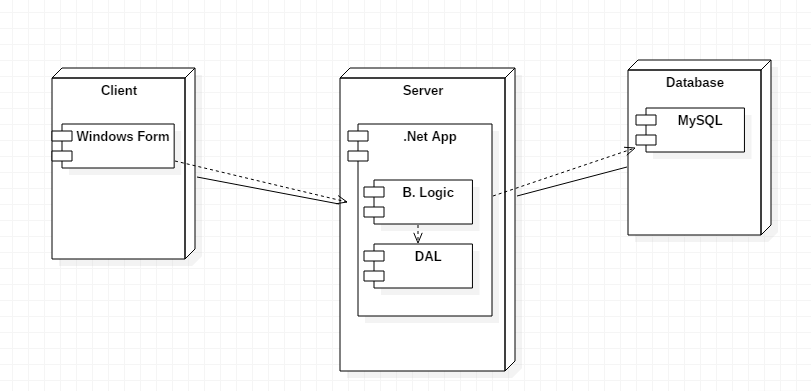
Package design:



Component diagram:

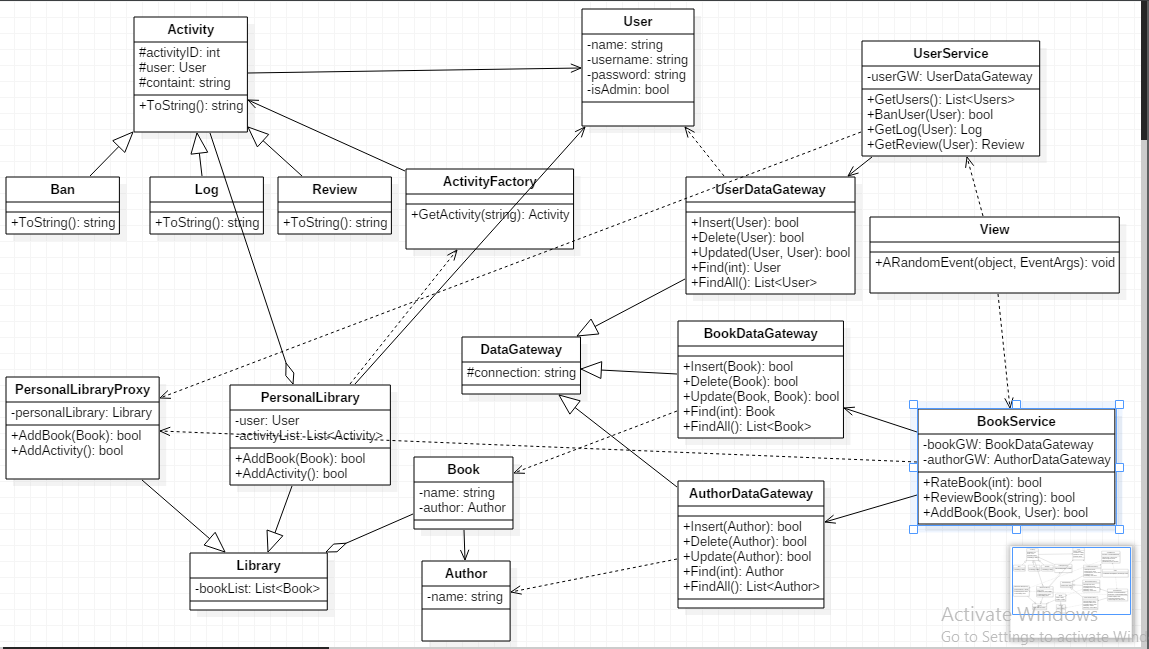


Deployment diagram:



The only modification that appeared is the fact that this app is not a web one. I changed it to desktop application.

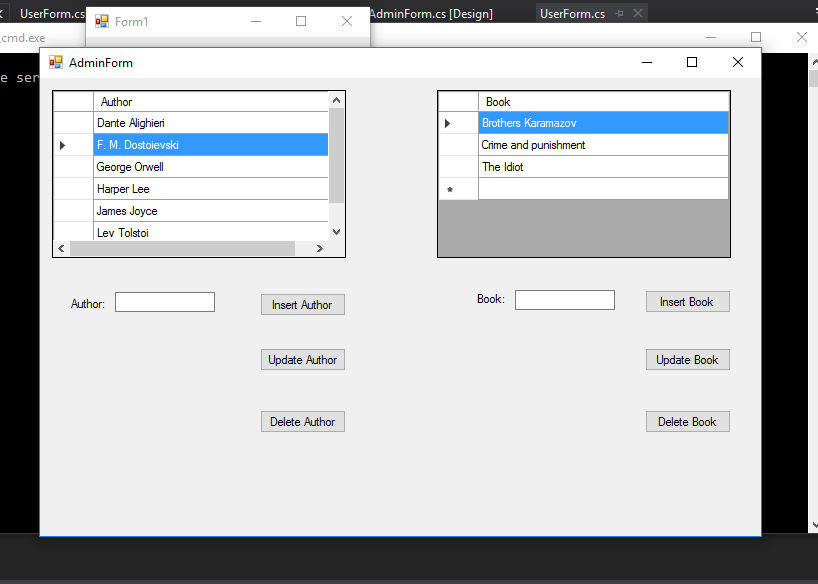
# Design Model Refinement

**

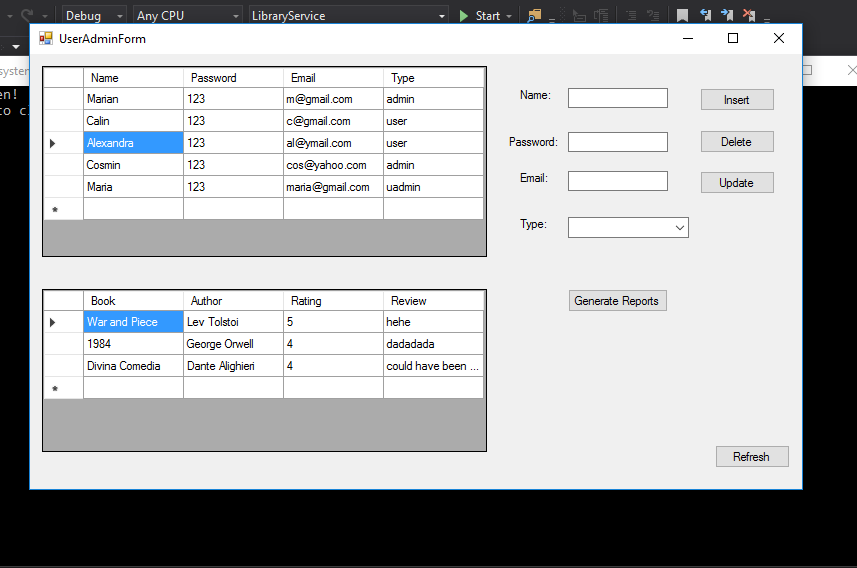
# Construction and Transition

# System Testing

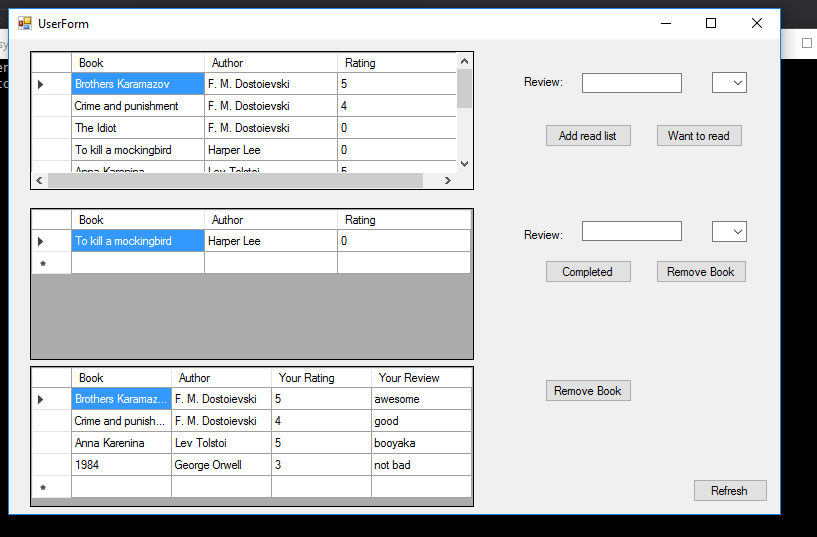
Admin:



User admin:



User:



# Future improvements

We could add the following operations:

- Recommend books to other users

- Send messages to other users

- Report other users

# Bibliography