System Design Document for Chalmers Book Market

An application encouraging reuse of student literature

Pegah Amanzadeh, Simon Holst, Kevin Pham, Carl-Magnus Wall

Group X

TDA367 Object-oriented programming project

Software Engineering
Chalmers University of Technology
October 2021

1 Introduction

In this document the system architecture of the application Chalmers Book Market will be asserted and evaluated. Chalmers Book Market is intended to function as a meeting place for people who wants to deal in used course literature. The idea behind the project is that students of Chalmers might benefit from a book market application that is directly connected to their Chalmers ID. The students would be easily able to interact with each other in order to buy or sell their used course literature.

The application is targeted to mobile users.

1.1 Definitions, acronyms, and abbreviations

- CBM - Chalmers Book Market
- CID - Chalmers ID

2 System architecture

The current iteration of the application does not make use of any external components. The functionality is contained within the code itself. When launching the application, the user is greeted by a login screen. The idea is that users will enter their CID and their respective password.
Once logged in, the user will see the main browsing page, where popular books are presented in lists, and where one can search for books one might be interested in. At the bottom there is a global navigation bar, where the user can access their account, add a book for sale, and access the main shop page. Users are encouraged to browse the application freely.
Like previously stated, in the current iteration, the applications functionality is contained within the code. When the application is closed, the session will not be saved as of the current version.

3 System design

Due to technical difficulties with SceneBuilder, more specifically the FXMLLoader-class, the separation of view and controller, to establish the MVC-pattern, has been inconceivable. The main reason is that FXMLLoader requires a controller when the view is initiated, at which state the controller-objects has not yet been instantiated, due to the fact that they want to have a reference to the view objects. The end result is a necessity to merge the view and controller into the same classes, as shown in Figure 1.

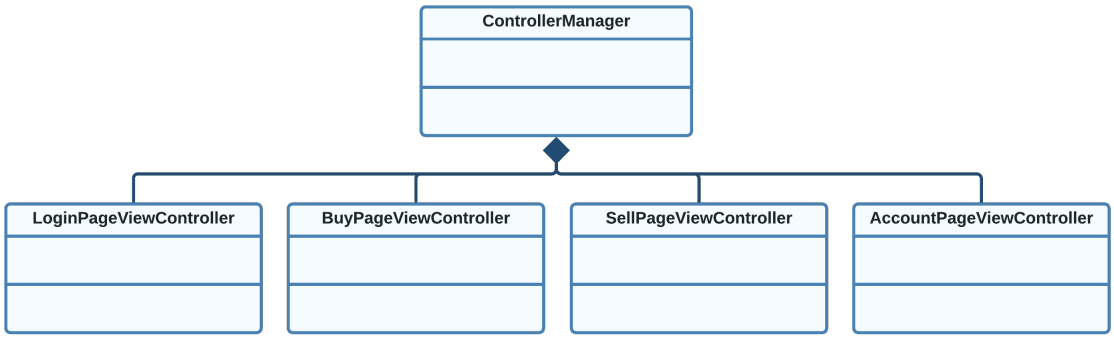

Figure 1: The application's solution to SceneBuilder conflicting standard MVC implementation.

The consequence of this becomes a reduction of packages, since the controller and view must share the same package of classes, as shown in Figure 2.

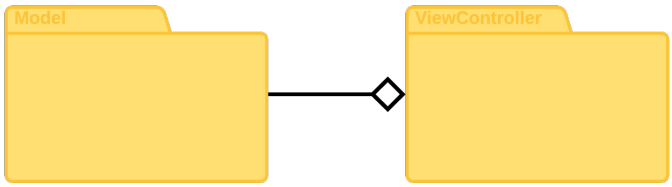

Figure 2: The application's package division.

The controller has a reference of the model, enabling it to call the required methods in the model corresponding to the graphical interface elements

interacted with by the user. The model changes accordingly and notifies the view that an update has been made and that it should change.

The current iteration has not yet separated the elements of MVC into the two different packages, but the separation is handled through proper uses of composition, aggregation and the Observer pattern. The current structure of the application is shown in Figure 3.

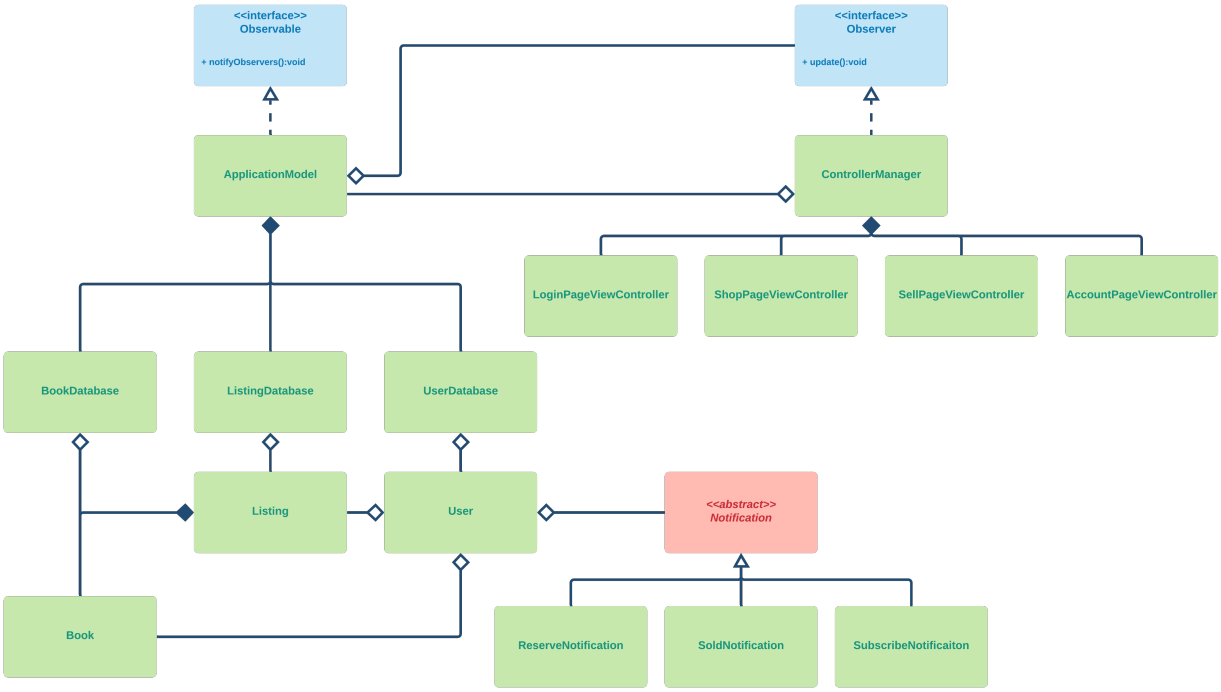

Figure 3: The application's design UML diagram

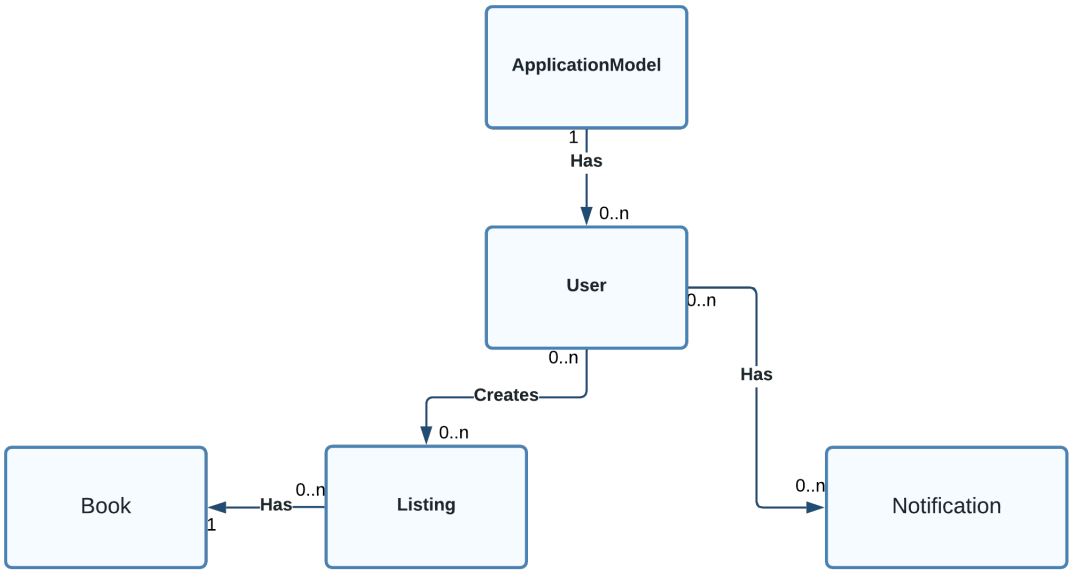

Figure 4: The application's domain model.

The domain model describes the basic composition and functionality of the application. The main idea is that the model consists of users, listings, notifications and books, and these are the building blocks of the application. The design model shows how the model data is contained on a more local, detailed level, and how it can be controlled and represented by the GUI. It describes in detail how the classes of the application are connected and how they are meant to communicate.

4 Persistent data management

The application insists on having having three different database classes, where books, listings and users are stored in their respective database class. Due to time constraints, the decision to go for a proof-of-concept approach was made quite early on in the project. The current version does not make use of any persistent data, other than stock images. A simple solution would be to dump the data, requiring saving between instances, in text files. A more sophisticated approach would be to make use of proper databases. Currently the stock images for books and listings are stored in a folder in the repository.

5 Quality

The current iteration has planned unit tests, but these are not incorporated. The application is built with Java 16, which seems to be incompatible with the standalone STAN application. Using STAN in Eclipse results in complaints about a certain class folder being null, when it most definitely is not. Further research is required.

5.1 Access control and security

The application has user logins, but these consist of mocked up data in the current iteration.

6 References

The following platforms, build tools, libraries and design tools were used to develop the application.

6.1 IDE

JetBrains IntelliJ IDEA

6.2 Build Tools

Maven 3.8.1

6.2.1 Imported libraries

- JUnit
- JavaFX 16

6.2.2 Structure analysis

N/A as of current iteration.

6.2.3 Quality analysis

N/A as of current iteration.

6.3 Design Tools

- LucidChart for UML and domain model design
- Figma for GUI design
- SceneBuilder to construct the GUI and connect it to the code