Student: Iulia Ungur

**Group: 30235**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The application was designed to help the employees of a book store sell and search books of customers’ preferences. They are able to search for a book by its title, author or genre from a list and sell it to the customer while on the stock.

# Functional Requirements

Features included on the employees’ side are:

* Searching for books by genre, title, author from a list
* Selling the books to customers

Features included on the administrators’ side are:

* creating, viewing, updating and deleting an employees’ information
* creating, viewing, updating and deleting an books information
* generating reports of books out of sold that are exported as xml files

# Non-functional Requirements

Availability: the system is available as long as the xml files that contain the information about the books are up-to-date

Performance: the system is intuitive for the users to work with, so any fair amount of work can be performed in a short period of time

Security: the system is secured on the login access level, but considering the information that is handled, the security wasn’t a great issue

Testability: due to the fact that the system is organized on layers, every component can be tested individually

Usability: as mentioned, the system is intuitive and each type of user has its display out of the login page, which makes the flow of work much easier

2. Use-Case Model

Use case: Admin Activity

Level: summary level

Primary actor: Administrator

Main success scenario: Admin logs in and handles books information or regular user information on the corresponding tab

Extensions: Admins can do multiple actions on the same tab or session

3. System Architectural Design

**3.1 Architectural Pattern Description**

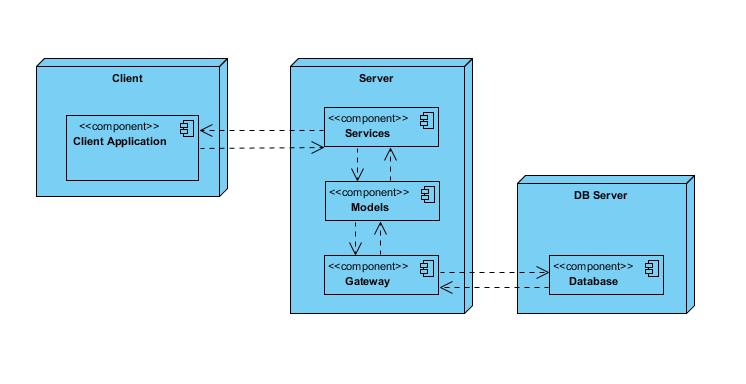
The Model-View-Controller (MVC) pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes:

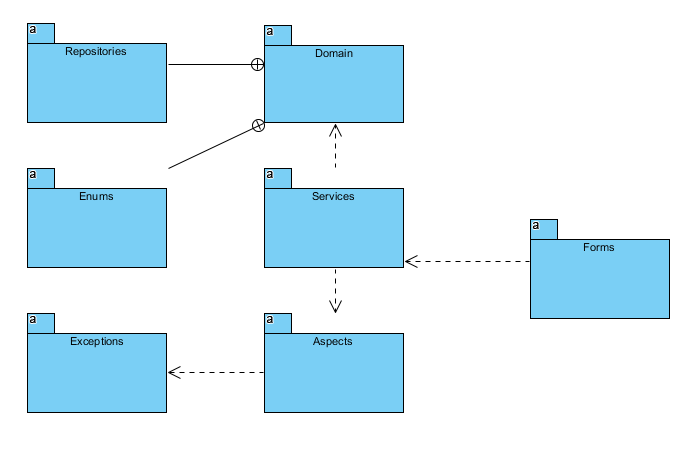
Model. The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

View. The view manages the display of information.

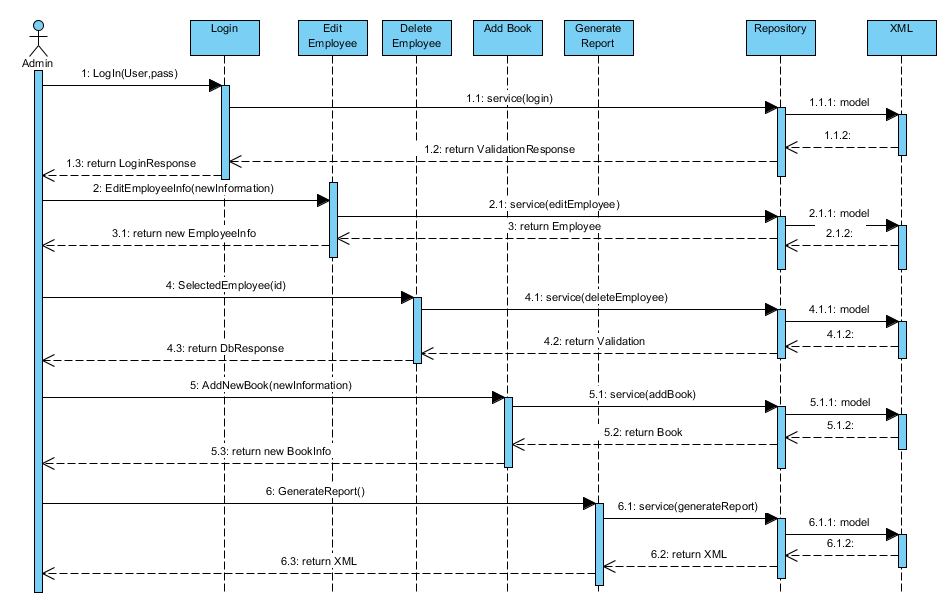
Controller. The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.

**3.2 Diagrams**





4. UML Sequence Diagrams

**

5. Class Design

**5.1 Design Patterns Description**

In class-based programming, the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

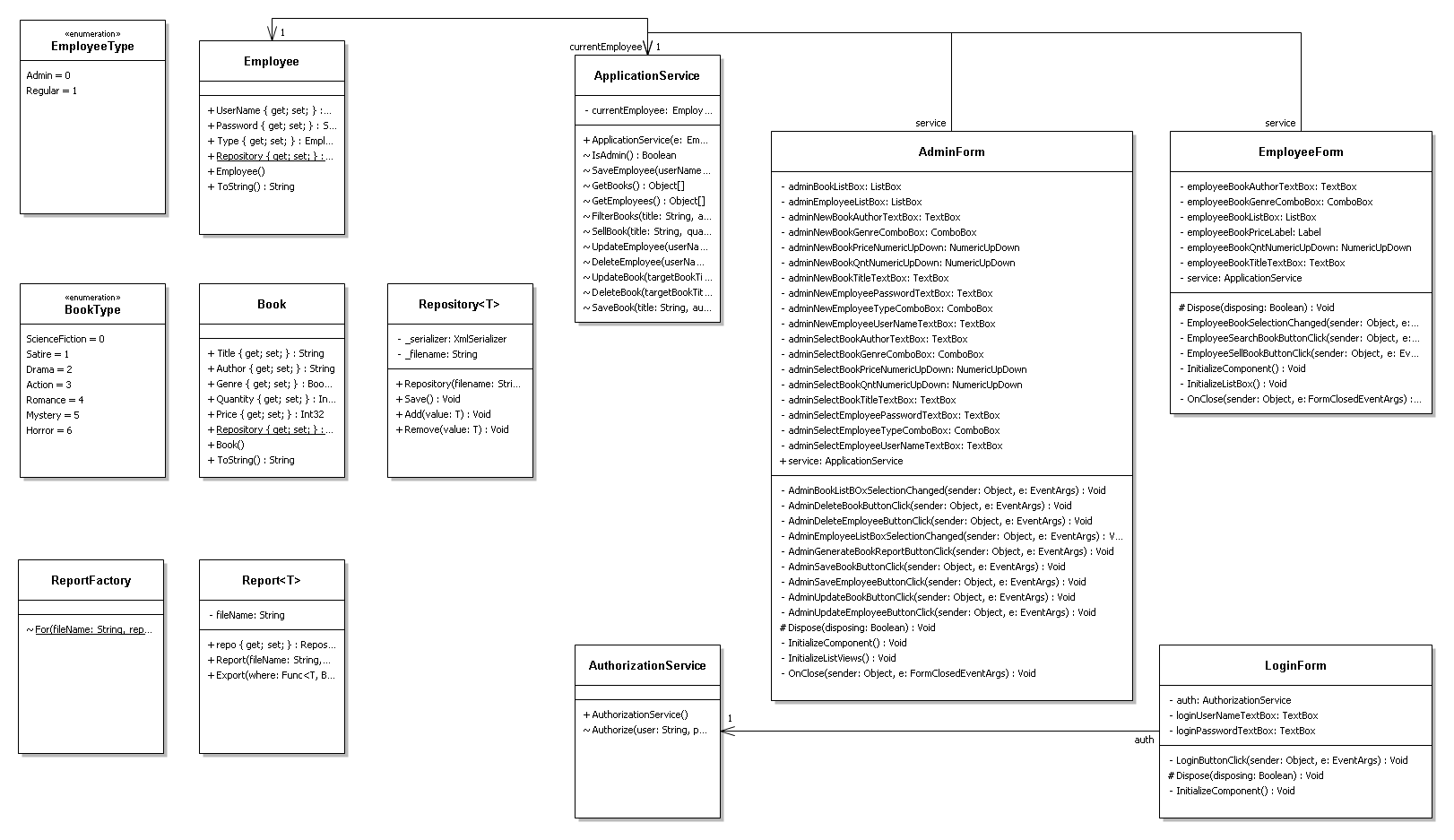
Factory Method is to creating objects as Template Method is to implementing an algorithm. A superclass specifies all standard and generic behavior (using pure virtual "placeholders" for creation steps), and then delegates the creation details to subclasses that are supplied by the client.

Factory Method makes a design more customizable and only a little more complicated. Other design patterns require new classes, whereas Factory Method only requires a new operation.

**5.2 UML Class Diagram**

The form controller calls the service to access the data. The service in order to retrieve the data from the xml file calls the afferent repository of the data model from which it can tell the xml to update according to the new data provided. Also the repositories are wrote in as minimal as it could be throughout an abstract class.

The implementation is based on low coupled classes that have minimal associations with each other except the basic instance.



6. Data Model

Employee - contains information about itself (username, password, type) – they can be admins or regular users

Books – contains details about the books that will be sold (tile, author, genre, quantity, price)

7. Bibliography

<https://msdn.microsoft.com/en-us/library/ff649643.aspx>

<https://en.wikipedia.org/wiki/Factory_method_pattern>

<https://sourcemaking.com/design_patterns/factory_method>