Analysis and Design Document

Student: Grigor Sonia Eufrosina Maria

**Group: 30233**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 24/April/19 | 1.0 | Write first two capitols of this document | Grigor Sonia |
| 01/May/19 | 2.0 | Elaboration – Iteration 1.2 - | Grigor Sonia |
| 28/Mai/19 | 4.0 | Elaboration – Iteration 2 | Grigor Sonia |
|  |  |  |  |

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

Software Design Project is a web application named "Tell your friend".

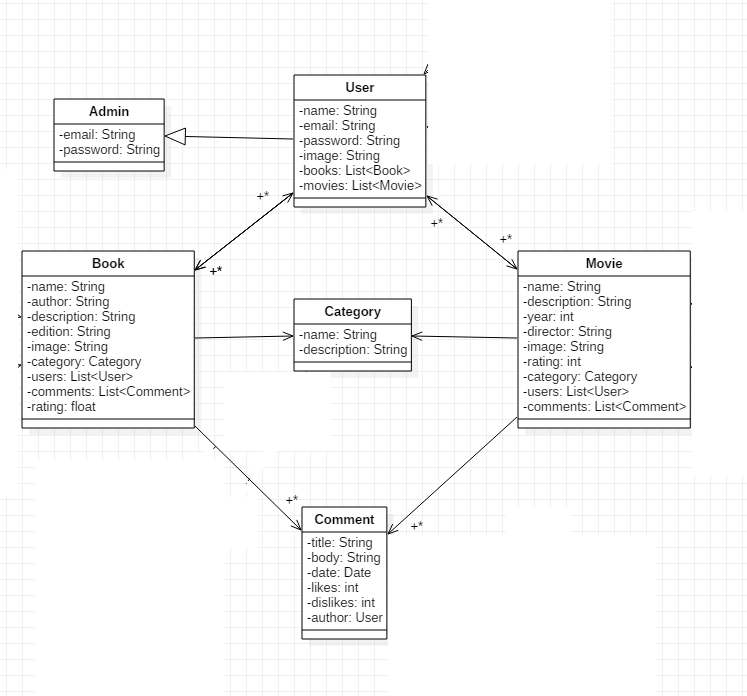
This will be a web application which can be used by everyone who wants to read or add a good review about a book or a movie. A user can register in the system, log in the system, add a new review, view all the reviews and add other comments on exiting reviews.

This application is implemented in **.Net** using **Entity Framework 6** for the back-end, and the front-end application is implemented using **Angular 7** and using the **Angular Material** Theme.

# Elaboration – Iteration 1.1

# Domain Model

A domain model is a conceptual model of the domain that incorporates both behavior and data.



# Architectural Design

## Conceptual Architecture

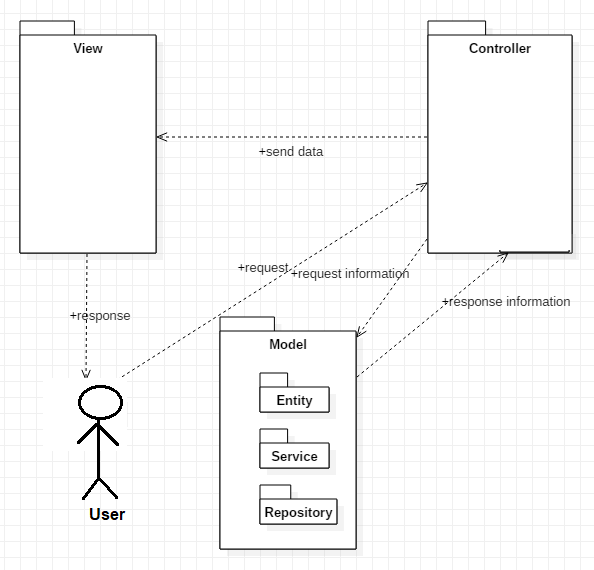
Layered pattern can be used to structure programs that can be decomposed into group of sub-tasks, each of which is a particular level of abstraction. Each layer provides services to the next high layer. The most commonly found four layers of a general application systems are as follows:

1. Presentation Layer
2. Business Logic Layer
3. Data Access Layer

Model-View-Controller architectural pattern follows an elementary idea – we must separate the responsibilities in any application on the following basis:

* **Model:** Handles data and business logic. Model represents an object carrying data. It can also have logic to update controller if its data changes.
* **View:** Presents the data to the user whenever asked for. View represents the visualization of the data that model contains.
* **Controller:** Entertains user requests and fetch necessary resources. Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Client/server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection. This system shares computing resources. Client/server architecture is also known as a networking computing model or client/server network because all the requests and services are delivered over a network.

The motivation using those two architectural patterns start from the bottom because considering is a web application, delimitation between view, logic and database is truly needed. The MVC design has an organizational structure that better supports scalability. Also organization in layers is a good practice regarding to software engineering. First of all, it is known the layered pattern offers greater structure to the project and also speeds up the developing process once done, because services like the database connection and operations (CRUD) are going to have their own package(**Persistence**) and classes(**UserRepository, BookRepository, Connection, etc**). Another layer, with it’s own package(**Business**) will be the business layer. The package will contain classes like: **UserService, BookService etc**, classes which will implement operations on the stored data, accessed through the Persistence layer. Finally, we’ll have the Presentation layer, which will encapsulate everything related to the user interface and all that’s related to the visual part of the project.

In my project builder is used in model package in order to build all the objects like Users, Books, Movies, etc. The implementation of this pattern is easy. You need to create a new object using a constructor default, and implements all the setters for every single attribute in the class (setName, setPassword, setComments, and the last step is to implement a method build() witch return your student.

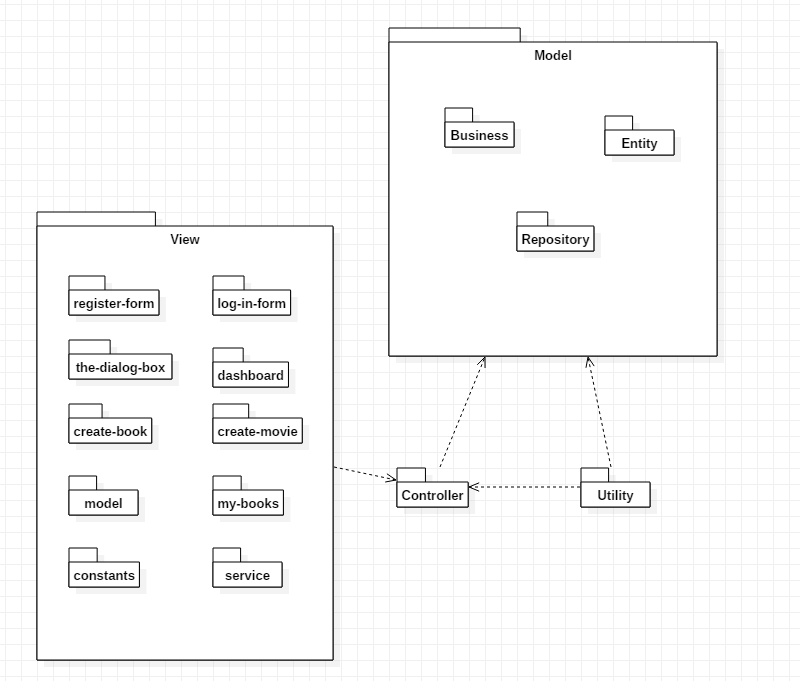
In this project Singleton is used in order to be sure that there is a single instance of the database instance.

## Package Design

“Tell your friends” application is a web application which is develop in two separated IDEs so it has two different solutions. As it is specified above, application is design using layered architecture pattern, so it is divided in three different layers.

The back-end solution is separated in two packages: Model and Controller. Inside the Controller package there are other three packages: Entity, Repository and Business. Each of them has a specific role and it has a specific layer associated.

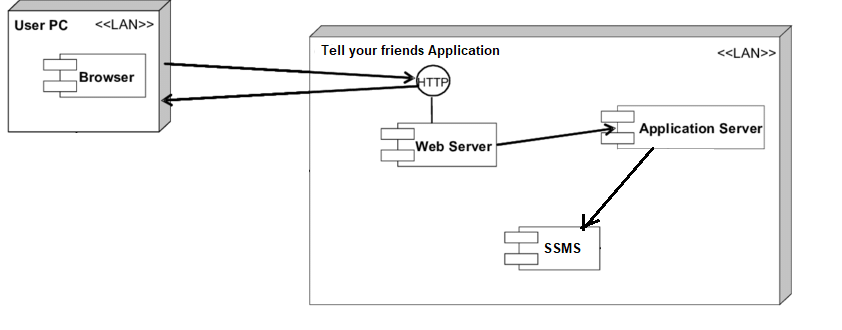
The front-end solution is associated with the View part of a MVC architecture project but also with presentation layer of a layered architecture project. Front-end solution is organized in more than 10 packages because each of them is the representation of a component. Each package has three types of files in order to customize, analysis, process data and organize each web-page.

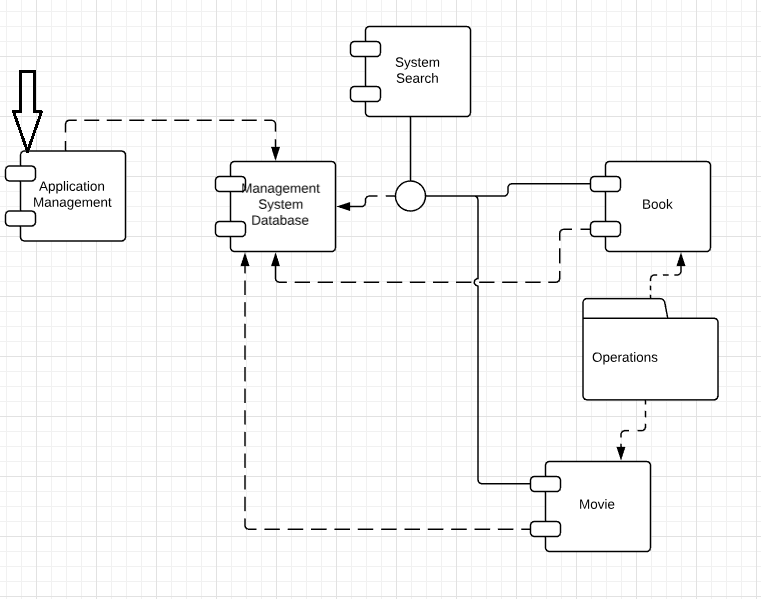


## Component and Deployment Diagrams

A component is a code module. Component diagram are physically analogs of class diagram.

UML Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.

Deployment diagrams show their physical configurations of software and hardware. Deployment Diagram show the structure of the run-time system, capture the hardware that will be used to implement the system and the links between different items of hardware.



# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

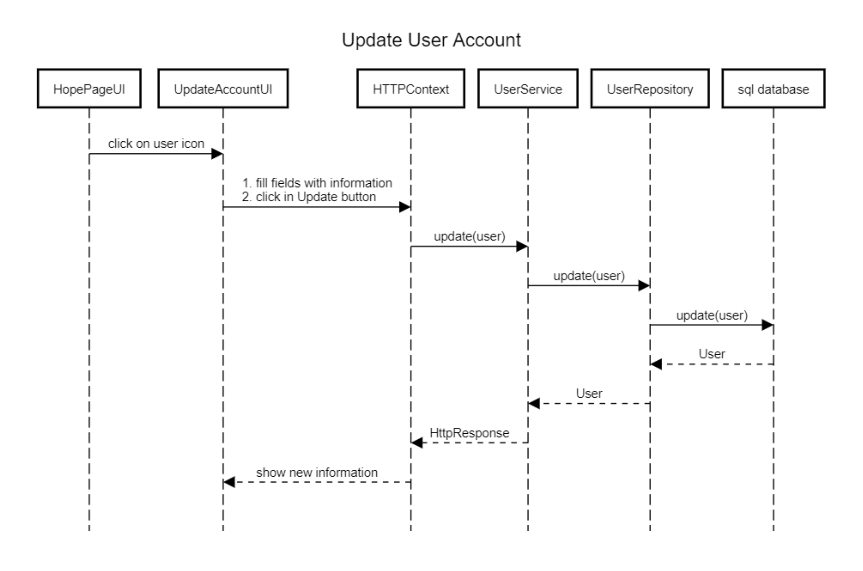
**Communication diagram** (called collaboration diagram in UML 1.x) is a kind of UML interaction diagram which shows interactions between objects and/or parts (represented as lifelines) using sequenced messages in a free-form arrangement.

**Sequence Diagrams** are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

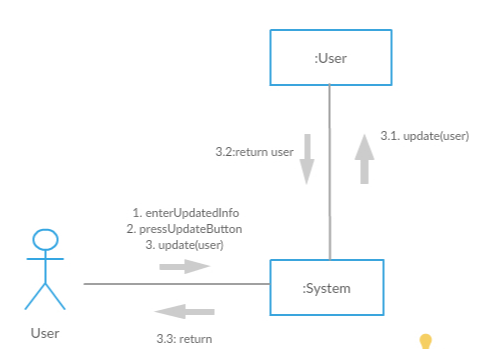
For creating the interaction diagrams it was chosen two relevant scenarios. First scenario is update user information and the other one is leave a comment/review for a book.

1.1.1 Scenario 1 – Update user information

1.1.1.1 Sequence diagram

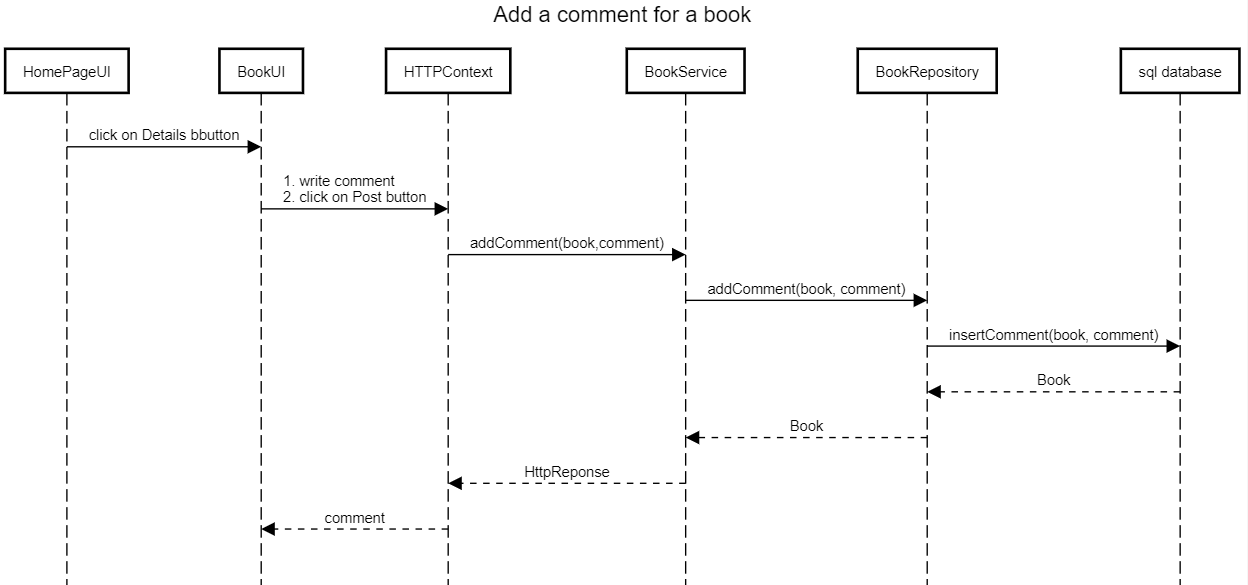


1.1.1.2 Communication diagram

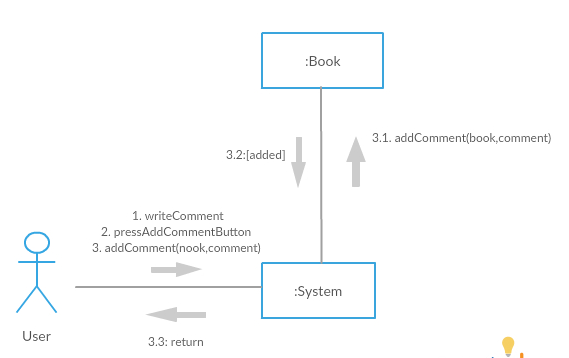


1.1.2 Scenario 2 – Comment a book

1.1.2.1 Sequence diagram

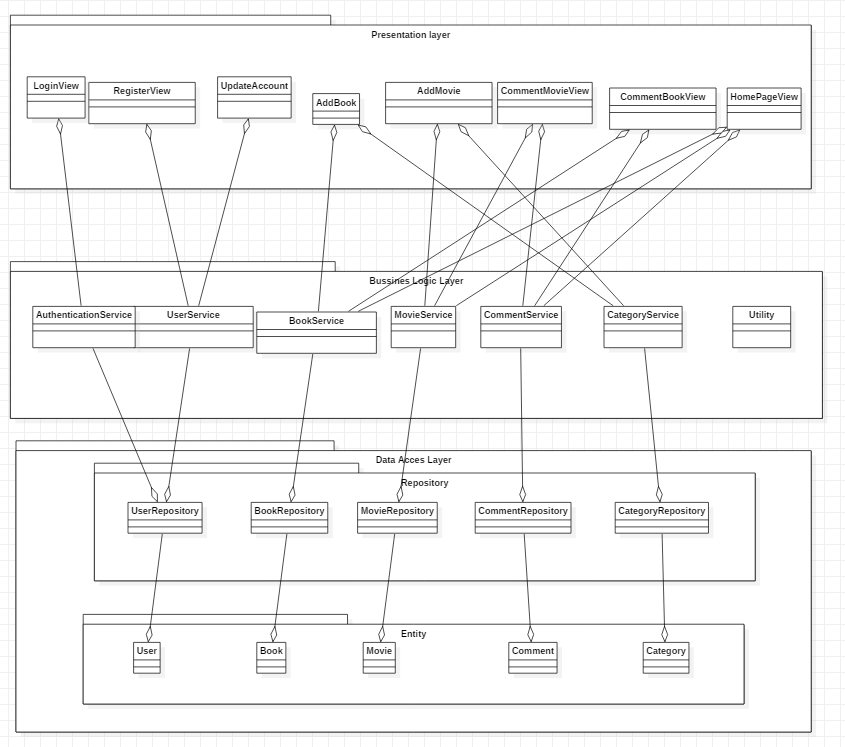


1.1.2.2 Communication diagram



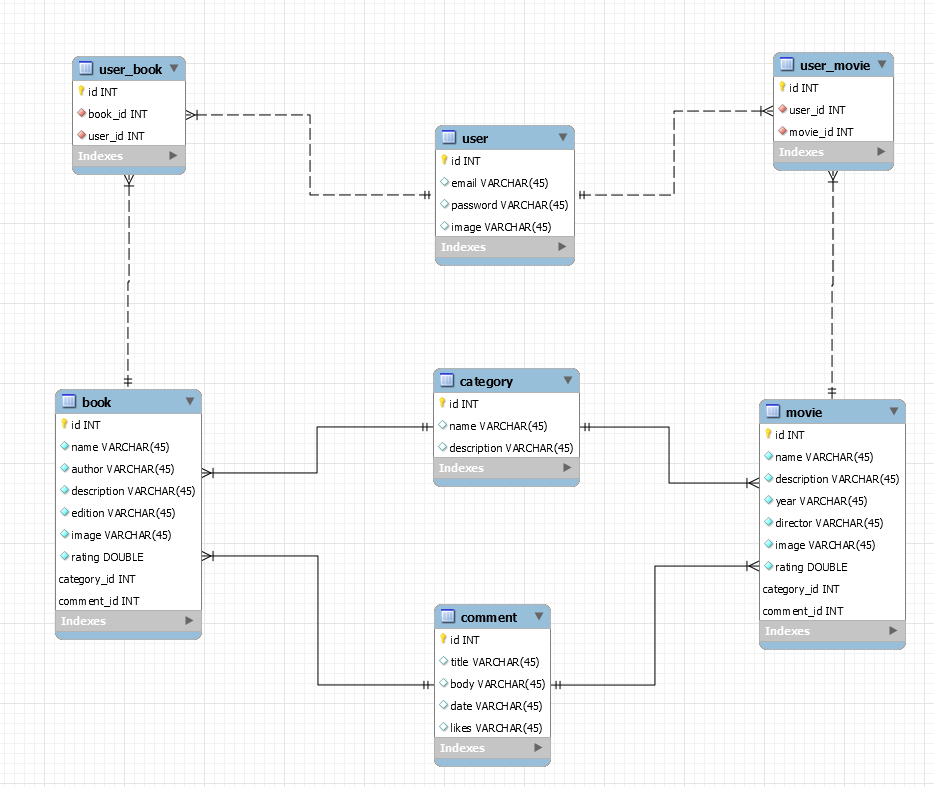
## Class Design

Design of this project is structured using Layers and Model-View-Controller architectural patterns. Considering this, it is needed a separation on layers of classes and also their relations.



# Data Model

The data model is a representation of the project and entities ad all relations between themselves. It was already presented in the section above, but here it is more clear.



# Unit Testing

Mocking Frameworks are used to create fake objects. Creating mock objects manually is very difficult and time-consuming. So, to increase your productivity, you can go for the automatic generation of mock objects by using a Mocking Framework. A developer can build his/her unit test by using any of the NUnit, MbUnit, MSTest, xUnit etc. unit test frameworks.

The associated test case scenario are express in the following:

3.1. On User side:

* get all users
* add user
* update user

3.2. On book side:

* get all books
* add book

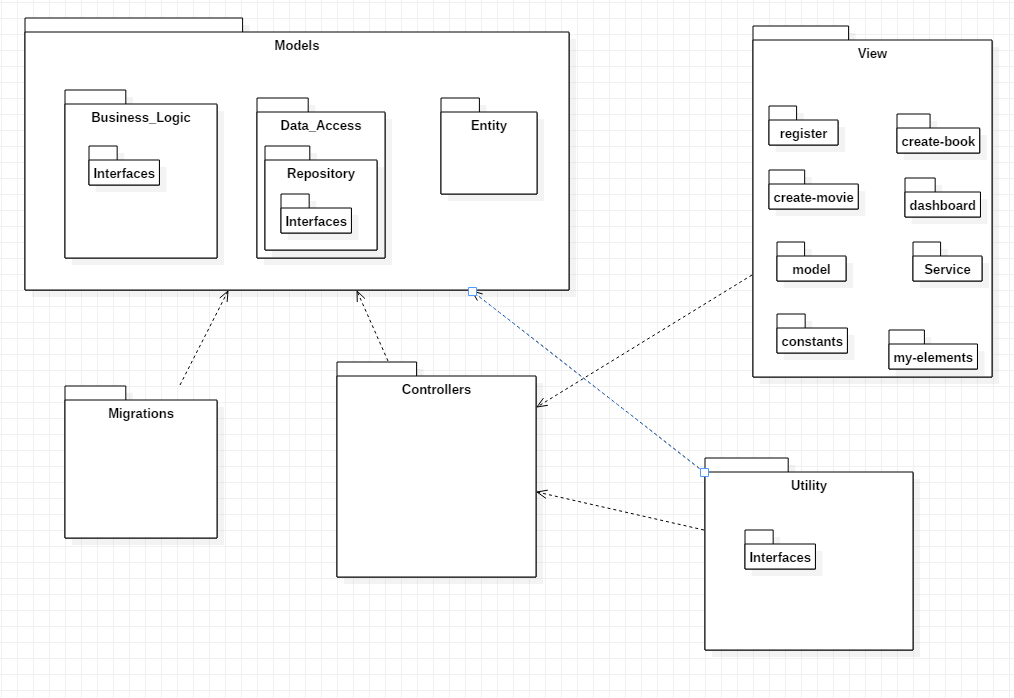
3.3. On movie side:

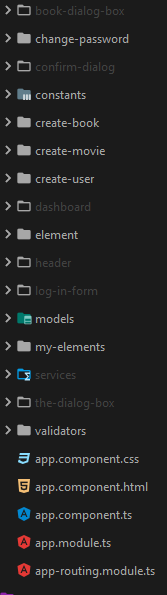
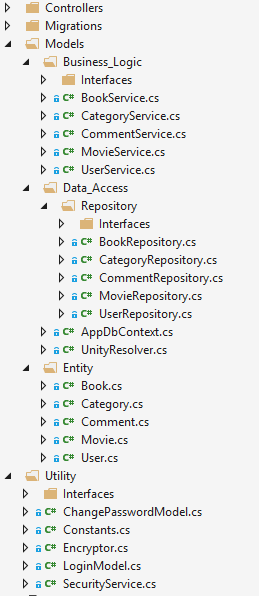
* get all movies
* add movie

# Elaboration – Iteration 2

# Architectural Design Refinement

Below I have the refined package design diagram, it suffers some modifications regarding to structure and adding some others packages like Interfaces for Repositories and Business Logic.



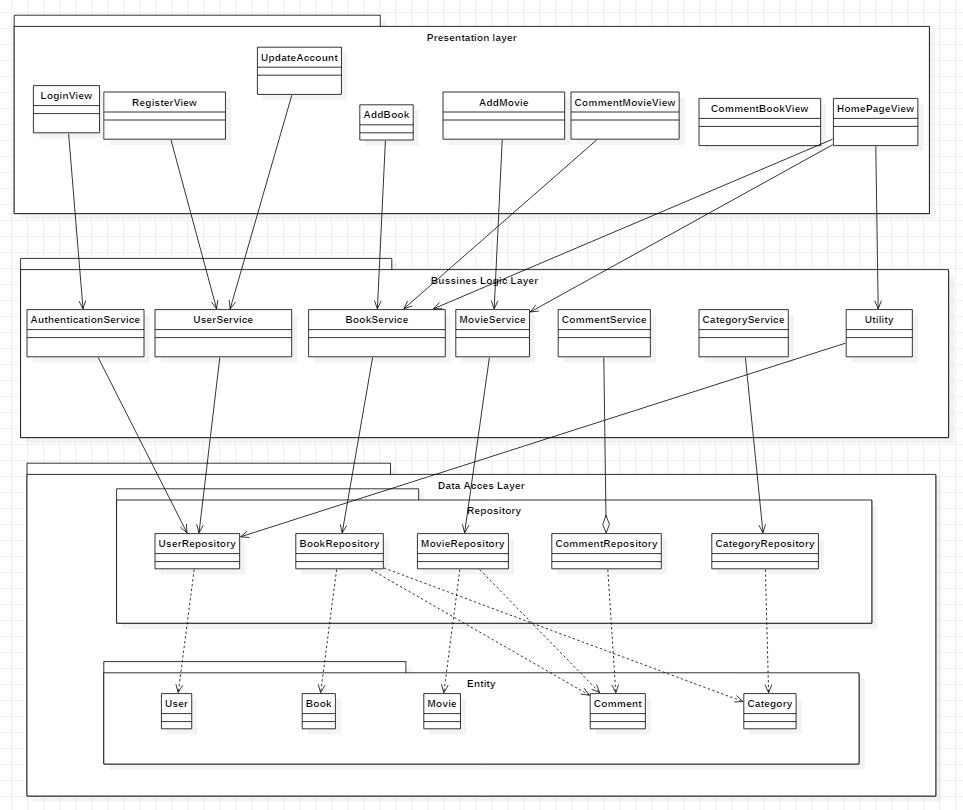


The component and deployment diagrams have not suffered any modifications because we didn’t add new components, which means that the analysis was properly done before actually starting to write code for the product.

# Design Model Refinement

An important modification was made by changing the relations between classes on the view side, but also on the logic/business side of the application. The new diagram is presented in the following figure.

I decided to change the configuration in order to respect the SOLID and OOP principles.

**

# Construction and Transition

# System Testing

Integration Testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

During development, I integrated every unit or module when it was finished and tested it it can be integrated with the others. After I finished the project, I integrated all component and test them manually by doing a full flow of the application. Basically, Integration testing was done after unit testing, and it revealed that every case scenario works properly and fits the description provided in the Use Case Models document.

# Future improvements

In terms of future improvements, the app could use an upgrade on the visual side, something to make it look more user friendly and modern. Another improvement could be the migration to a non-relational database, because that’s where industry today is heading. We could also add authentication, for security purpose, if needed.

# Bibliography

1. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/>

2. <https://www.codeproject.com/Articles/552846/Why-s-How-s-of-Asp-Net-MVC-Part-1>

3. <https://www.c-sharpcorner.com/article/moq-mocking-framework-with-xunit-net-testing-fr/>

4. <https://www.uml-diagrams.org/communication-diagrams.html>

5. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>