Software Architecture Document

Bibli.io

Prepared by

Joshua Butler	40045825	butler.joshua355@gmail.com
Mohammad Naimur Rashid	40027867	rashidn96@gmail.com
Suruthi Raju	40084709	suruthi25@gmail.com
Razine Ahmed Bensari	40029076	bensaria97@gmail.com
Nicolas Samaha	40027961	nicktherealog@gmail.com
Vincent Cerri	40034135	vincent.cerri@gmail.com
Justin Carrier	40026363	justind.carrier@gmail.com

Instructor: Dr. Constantine Constantinos

Course: Software Process (SOEN 341)

Date: 23 November 2018

Table of contents

List of figures

1. Introduction

This Software Architecture Document provides the comprehensive overview of the Bibli.io system. It provides the Purpose, Scope, Definitions, acronyms, abbreviations and Reference used for this document.

Purpose

The Bibli.io Software Architecture Document is used to help stakeholders identify requirements that affect high-level structure of the system. It is composed of many different sections (see Table of Contents), each having a role in describing and illustrating a particular piece of the system by using design artifacts. It also describes any relations between 1 or more software elements. This document is intended for all stakeholders, but mostly developers, who wish to understand more about the design decisions and patterns used to build the system.

Scope

The scope of this document is the entire Bibli.io architecture. It does not cover imported python packages like flask or flask-mysql. It does not describe how to deploy Bibli.io onto a production server. All other processes are described in this document. It describes The class structure and how they communicate. It describes the actions taken by each class for every critical use case.

Definitions, acronyms, and abbreviations

Provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the Software Architecture Document. This information may be provided by reference to the project's Glossary. For example:

- **UML**: Unified Modeling Language
- **SAD**: Software Architecture Document
- **SRS**: Software Requirements Specifications
- **SQL:** Structured Query Language. Relational database language used to store data coming from the server.
- **HTTP:** HyperText Transfer Protocol. Defines how messages are formatted and transmitted through web servers.
- **HTML:** HyperText Markup Language. Standard language for creating web pages and web applications.
- **CSS:** Cascading Style Sheet. Standard language used for styling web pages and web applications
- **POST:** Type of HTTP request which is used to transfer data from the client to the server. Has HTTP request parameters stored in message body.
- **GET:** Similar to POST, but carries request parameters appended in the URL.
- **URL:** Uniform Resource Locator. Reference to a web resource that is stored on a computer network.

2. Architectural representation

The top-level architectural style of the system and the view model that we adopted is that based of many enterprise software systems using the 4+1 view illustrated in Figure 1. We have the logical view which is the functionality that is provided to our end-users depicted in our UML class diagram models. We have the process view which shows how our system is intended to run which is shown through our activity diagram and communication diagrams. Our development view is mostly covered in our SRS. For the physical view it is being shown as substantiation to our physical view through our 3-layer architecture model along with the descriptions of the technologies that are being used in section 7: python, flask, html, css, MySQL. And, all these views are tied up with the intent of application usability which is depicted as a view in our use case diagram referred in the SRS as possible scenarios. Above the 4+1 view, we also included a deployment view of our software and a data view of the database.

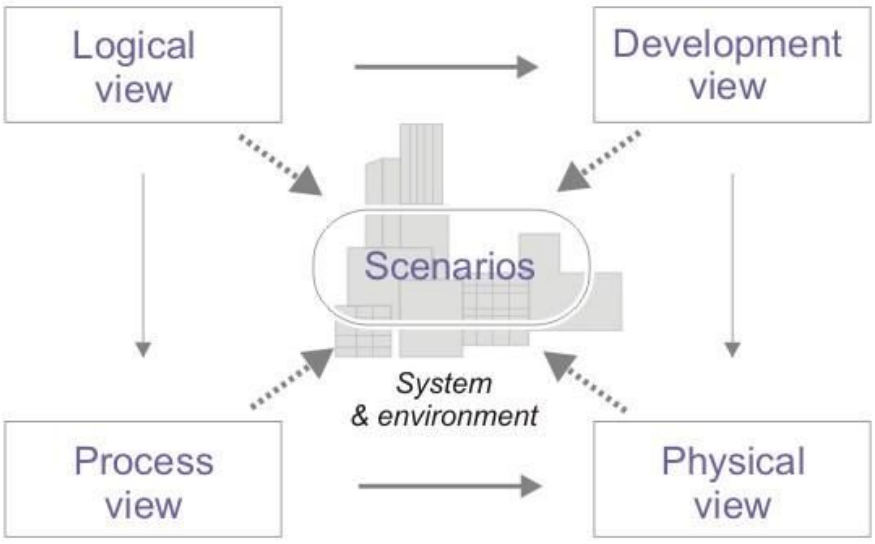


Figure 1: The 4+1 view model.

3. Architectural requirements: goals and constraints

3.1 User Registration/Login

Users need to be able to register and log in to the system through the HTML page. User and admin accounts are stored in the Bibli.io SQL database server upon receiving a successful POST request generated by the HTML page. Administrators can only be created by other administrators using a special interface only accessible to them. There are distinct homepage views after authentication that effectively enforces access levels between admins and clients. Also some malicious attempts to access an unauthorized area would redirect you to a unauthorized page.

3.2 View Catalog

Users and Admins are able to perform a simple view on the entire item catalog stored in the SQL database. These items are filtered by category which include: book, magazine, movie or music. The catalog is presented in a table view where each row is associated to a unique item.

3.3 Search Catalog

Users should be able to search and filter the catalog view by entering any specific data about any of the library items into an HTML form which is passed to the server using a POST request. The page containing the search results is then rendered to the client.

3.4 Loan Item

Users are able to loan items by clicking the “Make Loan” button. A new interface will appear where they can enter any item’s (except magazines) unique ID into the input box and click “Checkout”. An association between the user and the loaned item is then created and added to the database’s loans table. Limitations are that a loan can only include one item; however, a client my initiate as many loans as possible. The DBMS will not accept an item to be loaned more than once, an error would be raised.

3.5 View Loans

Users can view their loans by clicking the “View Active Loans” button seen on the home page. This will show a table view of all loan items that are associated to the current user.

3.6 Add Item

Administrators can add new items to the library catalog by clicking the “Add Library Record” button on the home page. A new HTML form will load were the admin can insert details about the new item. Confirming the changes will send a POST request to the server where the item is then added to the database. Users should not be able to perform this action.

3.7 Modify Item

Administrators can modify existing items in the library catalog by clicking the “Modify Library Record” button on the home page. A textbox will load where the admin can input the item ID of the item he wishes to modify. The selected item’s attributes are then retrieved from the database and displayed on the screen through an HTML form. The administrator can then modify each of the item’s attributes and click “Confirm changes” to persist the changes into the database. Users should not be able to perform this action.

3.8 Remove Item

Administrators can remove items from the library catalog by clicking the “Remove Library Record” button on their navigation menu. Inserting the item ID will perform a deletion of the selected item from the database. Users should not be able to perform this action.

3.9 Register Administrator

Administrators can register other administrators in the system by clicking the “Register Administrator” button on their navigation menu. A new HTML form will appear allowing the admin to insert all details regarding the new account. Once submitted, it is sent to the server for processing where the account is created and added to the database. Admins are given a special privilege level to differentiate them from users (privilegelevel == ADMIN).

3.10 View Active Users

It is possible for an administrator to view the number of logged in users. The number of logged in users is monitored by a counter variable in the backend that has known issues for

which in the case where a server would crash this number may be compromised. A better implementation is advised for the future of the software system.

3.11 Return Loan

In our implementation of returning a loan, it was determined that this should be ultimately an administrator feature. On reception of the returned item, an administrator logs into his account here there is an option to process the return which effectively deletes the loan association between the client and the item from persisted memory. Drawbacks of this design is that there are no logs of returned items.

4. Use case view (also known as Scenarios)

Audience: all the stakeholders of the system, including the end-users. The description of an architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. Related Artifacts : **Use-Case Model included in SRS.**

5. Logical view :

Audience: Designers. The logical view is concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include **Class diagram**, and **interaction diagrams** (**communication diagrams**, or **sequence diagrams**).

5.1 Overview:

The E-library application “Bibli.io”, is divided into the following physical layers namely: Presentation, Domain and Data Source.

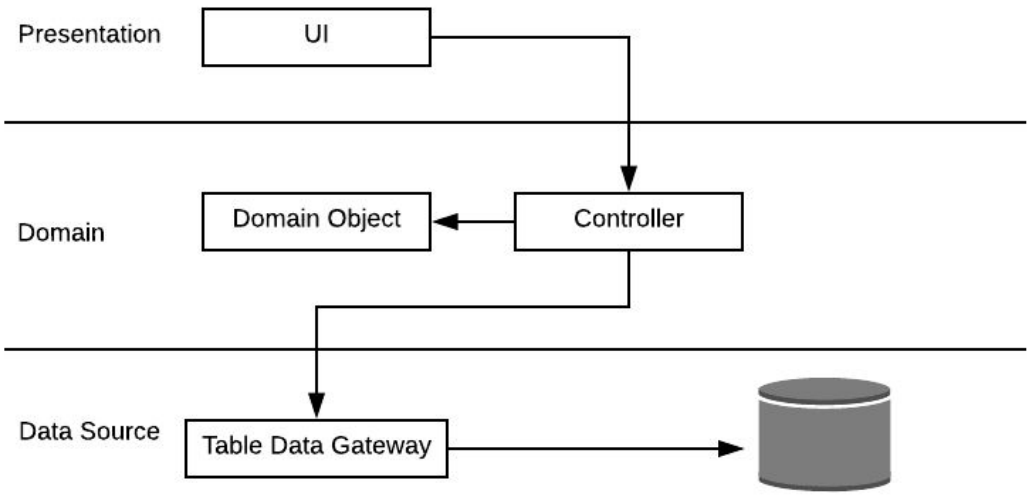
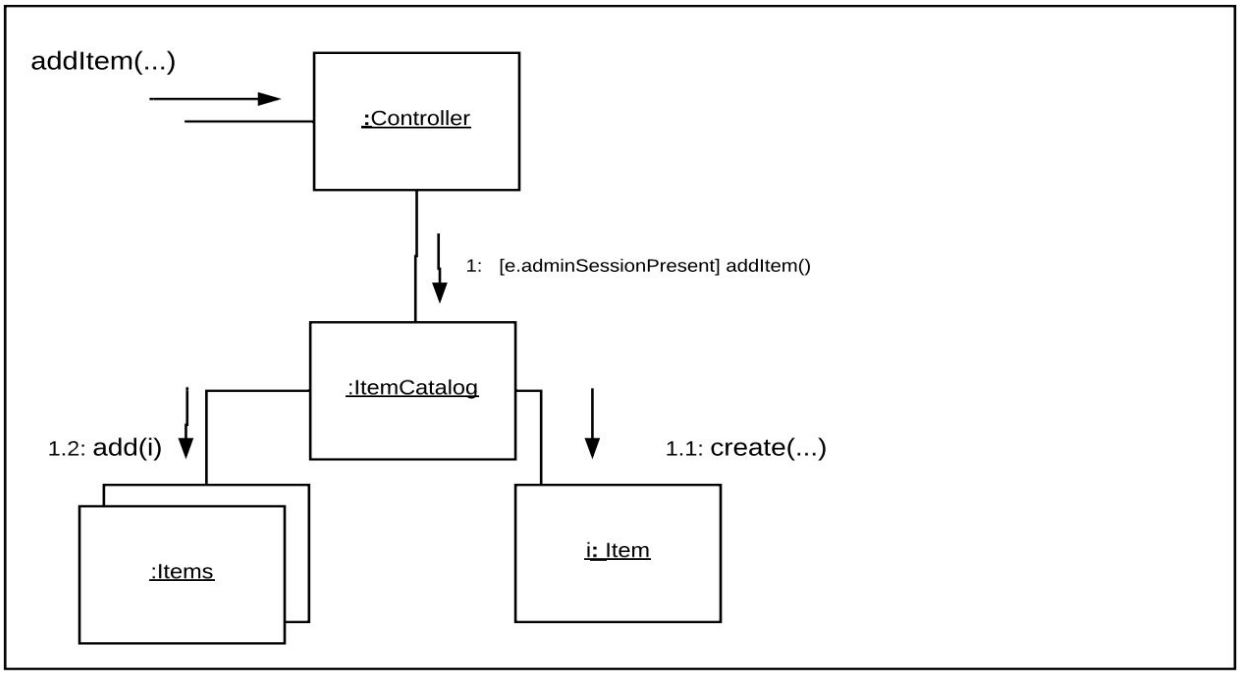
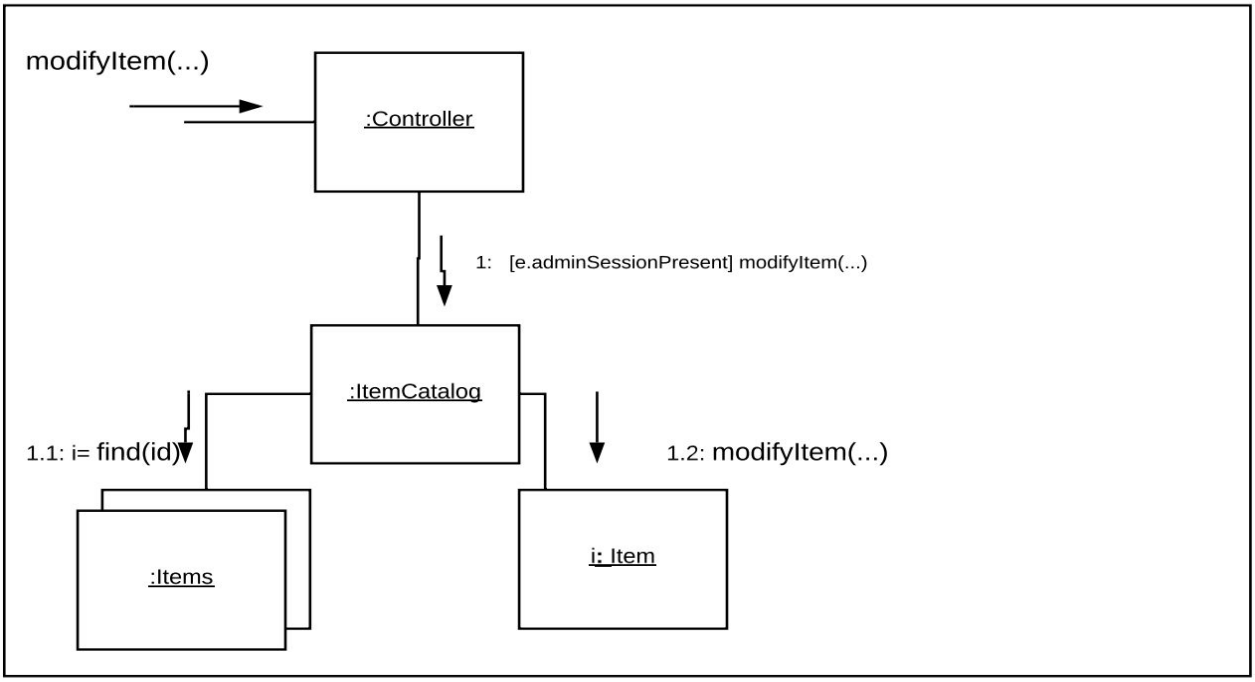
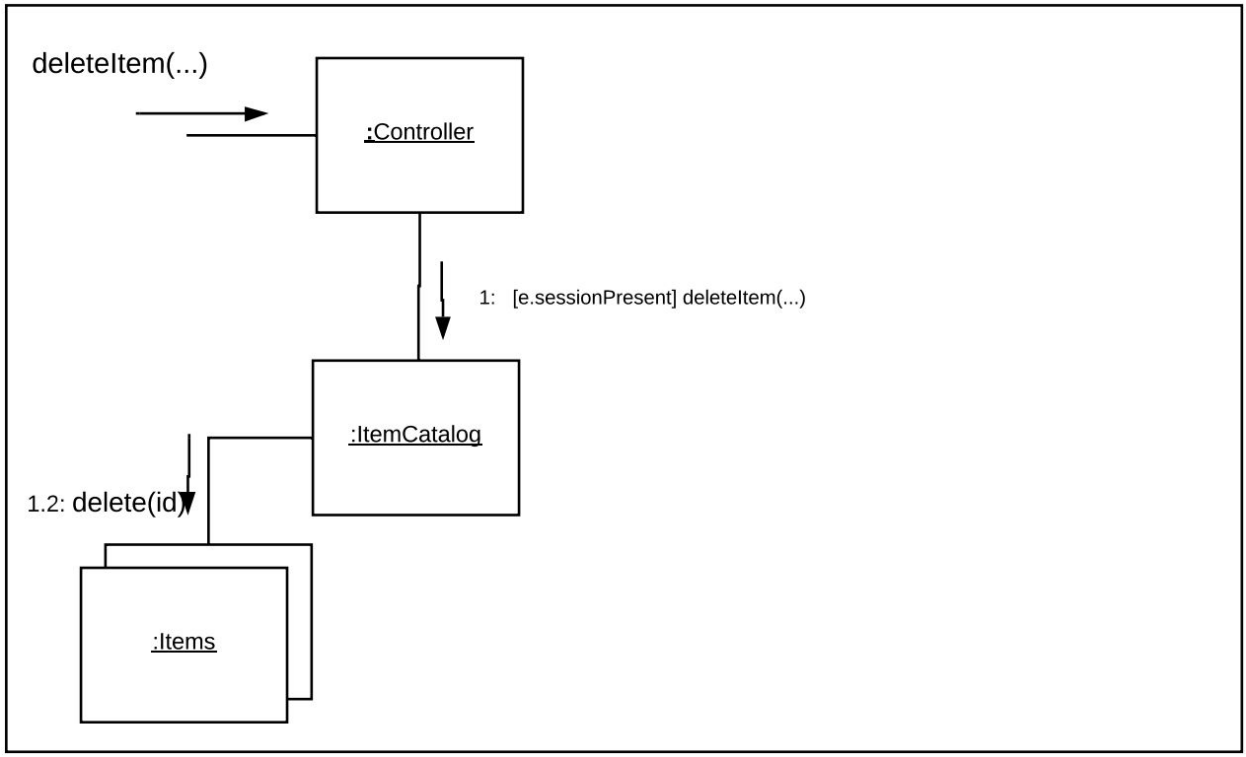


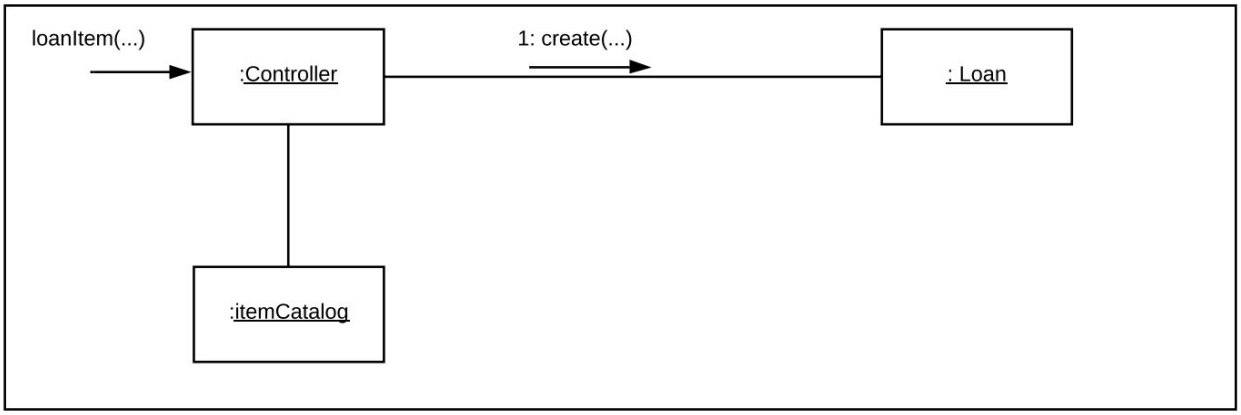
Figure 2: Data Mapper with TDG Diagram

5.2 Communication Diagrams






deleteItem(...)
:Controller
1: [e.sessionPresent] deleteItem(...)
:ItemCatalog
1.2: delete(id)
:Items


loanItem(...)
:Controller
1: create(...)
: Loan
:itemCatalog

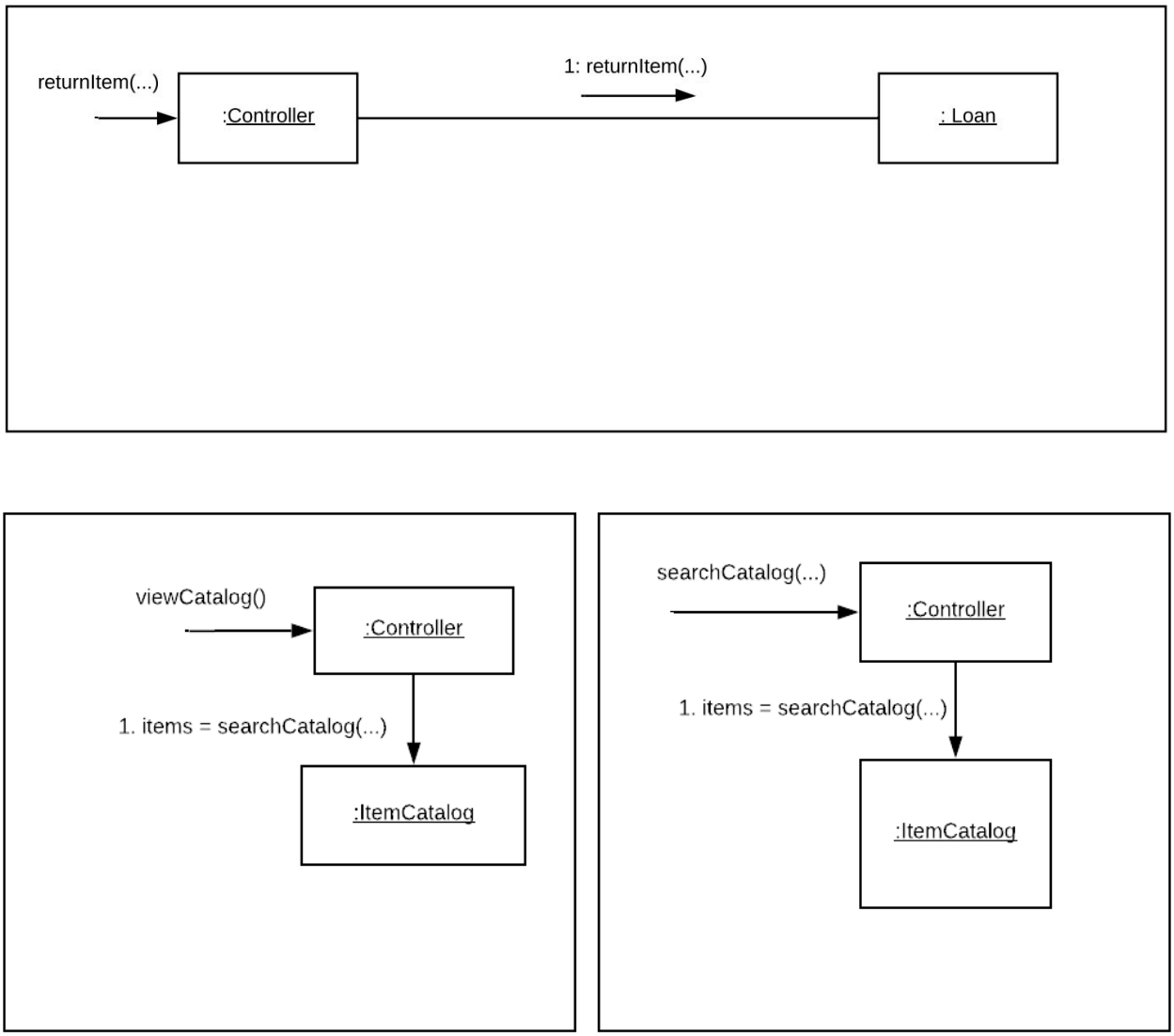
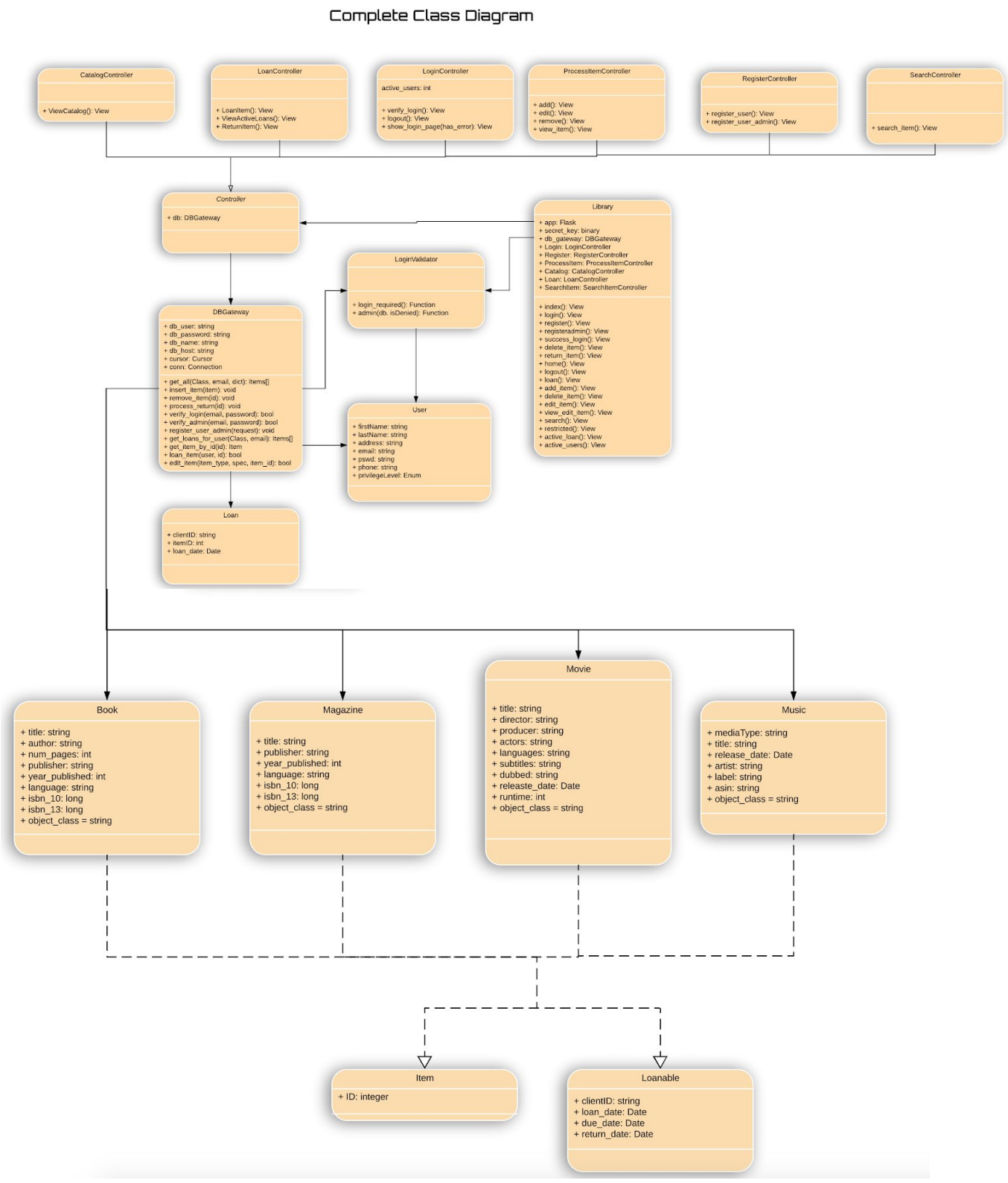


Figure 3: Communication Diagrams

5.3 Class Diagram (figure 4):



5.4 Activity Diagram:

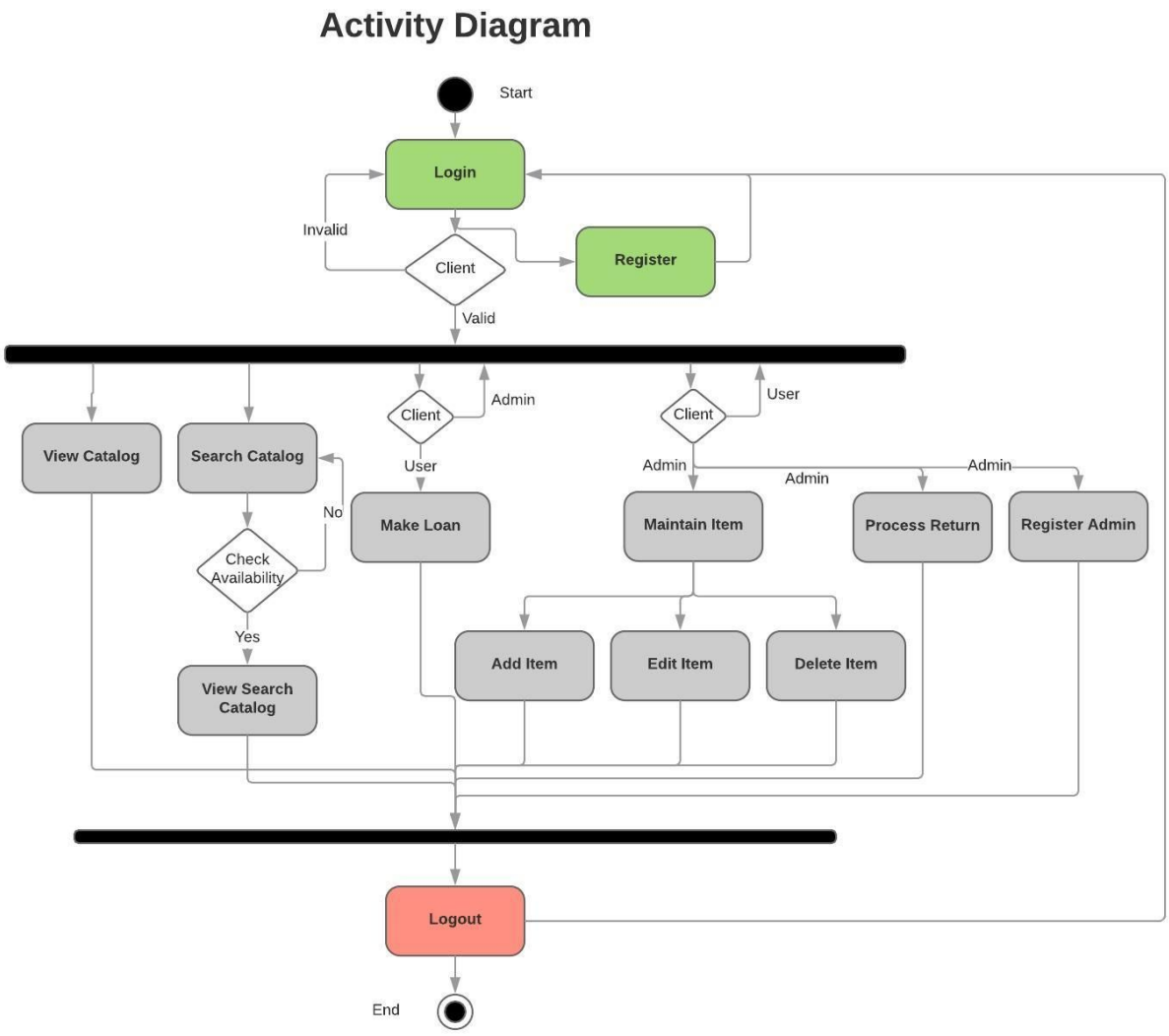


Figure 5: Activity Diagram

6. Deployment View:

The Deployment view of our project is as shown in the diagram

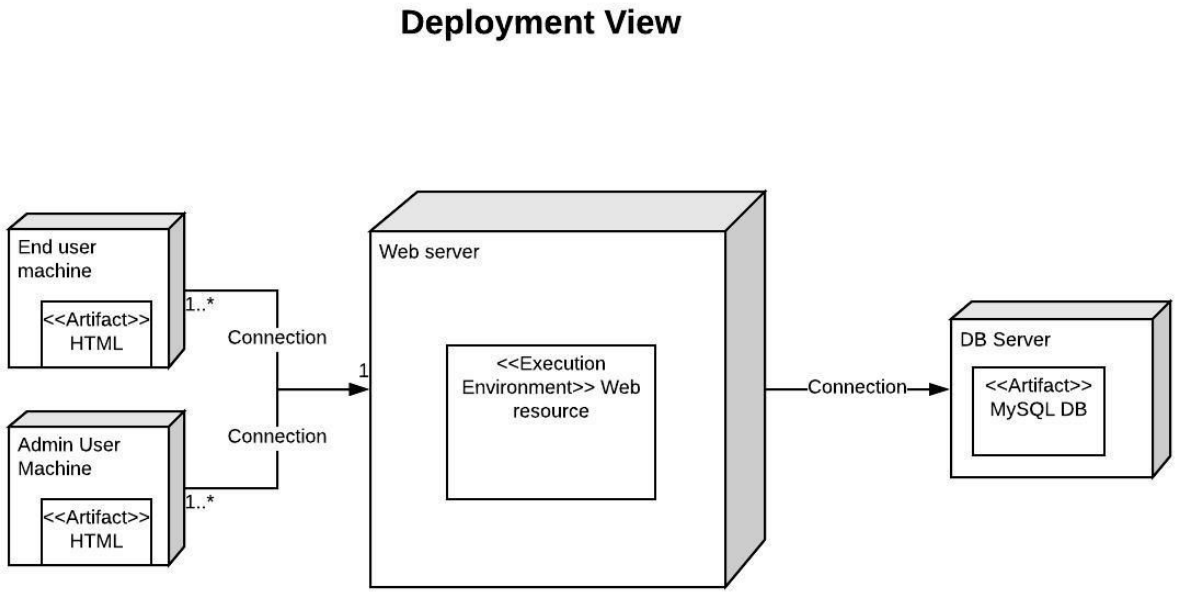


Figure 6: Deployment Diagram

7. Data view :

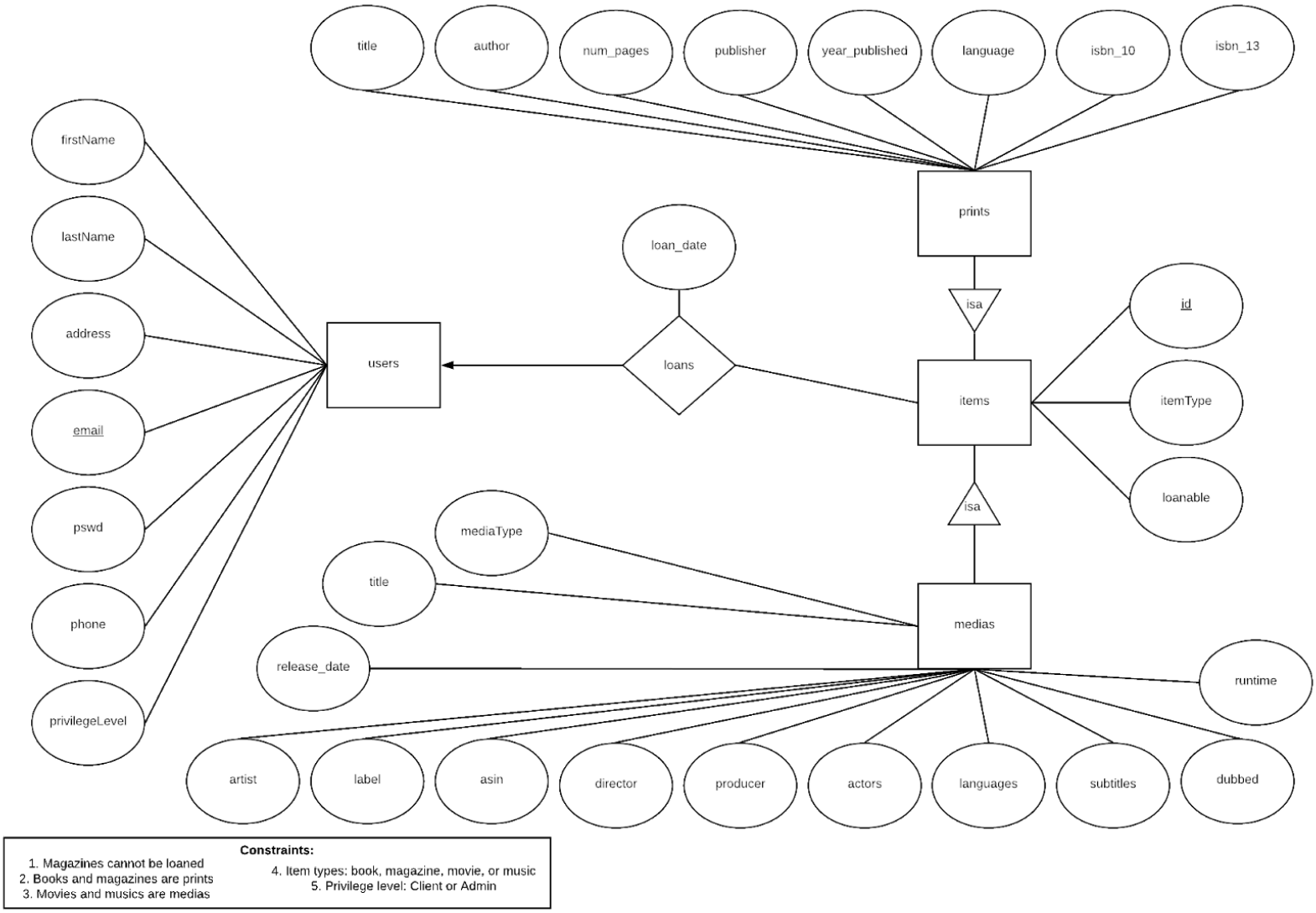


Figure 7: ER Model

8. Quality:

Reliability, Portability, Availability

- The system can be used in any machine provided it has access over internet. A browser is needed to access the web application.
- The system backend runs on Python which is an interpreted language which works on all major server operating systems.
- The system can be run with a load balancer, so that if one server goes down, the other will be more than able to handle incoming requests.

Security:

- Authentication and authorization mechanisms is provided to every user and administrator. The password is encrypted to ensure safety and hacking. The encoding scheme used is *MD5*.
- There is no DDOS protection, however there is little value in attacking this system that way.

Scalability:

- Flask scales per request. On one core, flask can handle at least one thousand requests per second. Since this is a local library, there should not be that many requests.