

CS157A Team 2

Final Report

A Book Reading and Review Web Application — "OpenBooks"

Dispoto, Brett

Kamel, Adham

Cai, Feiyu

December 7, 2019

# Contents

<b>1</b>	<b>Project Requirements</b>	<b>3</b>
1.1	Project Description . . . . .	3
1.1.1	Application Overview . . . . .	3
1.1.2	Project Motivation . . . . .	3
1.1.3	Project Goal . . . . .	3
1.1.4	Benefits to Users . . . . .	3
1.1.5	Stakeholders and Importance . . . . .	4
1.2	System Environment . . . . .	4
1.2.1	Presentation Layer . . . . .	4
1.2.2	Application Layer . . . . .	5
1.2.3	Data Layer . . . . .	5
1.2.4	Hardware/ Software Used . . . . .	5
1.3	Functional Requirements . . . . .	6
1.3.1	Comprehensive List of Features . . . . .	6
1.4	Non-Functional Issues . . . . .	8
1.4.1	Graphical User Interface . . . . .	8
1.4.2	Security . . . . .	8
1.4.3	Concurrency Control . . . . .	9
1.4.4	Access Control . . . . .	9
<b>2</b>	<b>Database Project Design</b>	<b>10</b>
2.1	Updated Entity-Relationship Diagram . . . . .	10
2.2	Database Schema . . . . .	10
2.3	Entities . . . . .	11
2.3.1	Users . . . . .	11
2.3.2	Content . . . . .	11
2.3.3	Books . . . . .	11
2.3.4	Magazines . . . . .	12
2.3.5	Newspapers . . . . .	12
2.3.6	Articles . . . . .	12
2.3.7	Poems . . . . .	12
2.4	Relationships . . . . .	12
2.4.1	User Reviews Content . . . . .	12
2.4.2	User Favorites Content . . . . .	13
2.4.3	User Downloads Content . . . . .	13
2.5	Boyce Codd Normal Form (BCNF) Proofs . . . . .	13
2.5.1	Users — BCNF Proof . . . . .	13
2.5.2	Reviews — BCNF Proof . . . . .	13
2.5.3	Favorites — BCNF Proof . . . . .	14
2.5.4	Downloads — BCNF Proof . . . . .	14
2.5.5	Poem — BCNF Proof . . . . .	14
2.5.6	Content — BCNF Proof . . . . .	14
2.5.7	Magazine — BCNF Proof . . . . .	15

2.5.8	Article — BCNF Proof . . . . .	15
2.5.9	Newspaper — BCNF Proof . . . . .	15
2.5.10	Book — BCNF Proof . . . . .	16
2.6	Database Table Instances . . . . .	16
2.6.1	Users . . . . .	16
2.6.2	Reviews . . . . .	17
2.6.3	Favorites . . . . .	17
2.6.4	Downloads . . . . .	17
2.6.5	Poem . . . . .	18
2.6.6	Content . . . . .	19
2.6.7	Magazine . . . . .	20
2.6.8	Article . . . . .	20
2.6.9	Newspaper . . . . .	20
2.6.10	Book . . . . .	21
<b>3</b>	<b>Implementation</b>	<b>21</b>
3.1	Implementation Overview . . . . .	21
3.2	Detailed Use-case Implementation . . . . .	22
3.2.1	Use Case: Create Account . . . . .	22
3.2.2	Use Case: Log In . . . . .	23
3.2.3	Use Case: Search for Content . . . . .	24
3.2.4	Use Case: View Profile . . . . .	25
3.2.5	Use Case: Sorting Results . . . . .	26
3.2.6	Use Case: View Content Profile . . . . .	27
3.2.7	Use Case: Write Review . . . . .	28
3.2.8	Use Case: Read Review . . . . .	29
3.2.9	Use Case: Favorite/ Download Content . . . . .	30
3.2.10	Use Case: Log Out . . . . .	31
3.3	Project Setup/ Configuration . . . . .	32
<b>4</b>	<b>Project Conclusion</b>	<b>32</b>
4.1	Lessons from Project — Teammate Statements . . . . .	32
4.1.1	Brett Dispoto . . . . .	32
4.1.2	Feiyu Cai . . . . .	33
4.1.3	Adham Kamel . . . . .	33
4.2	Future Improvements — Discussion . . . . .	34

# 1 Project Requirements

## 1.1 Project Description

### 1.1.1 Application Overview

Our team will be developing a database application where users can find free books and are able to download them via multiple formats and leave reviews on those books for other customers to see. The books that the user can see will be ranging from textbooks to novels. A user will be given the option to create an account, or if they already registered, login to their account. After this, the user will be able to search for a specific book, whether that is by the title of the book or its ISBN number. If the user does not know the specific book they want to find, they can search for a specific author, or genre. Searches will include various filtering options such as date published, publisher, book length, user favorites, etc. Users can sort search results by the title alphabetically, the author alphabetically, or by book rating.

In our application, users will be able to leave reviews on books that they have read, which will be seen by other users and possibly influence their decision on their book. These reviews will contain both a comment subsection and a star rating system. Users will be able to comment how they either liked or disliked the book and give a star rating from one to five. The average star rating and total number of reviews will be displayed next to the book. Users will also be able to share the book they like with others with a shareable link that they can distribute how they please.

### 1.1.2 Project Motivation

The motivation behind this project is that as college students, we find ourselves paying several hundreds of dollars on college textbooks alone every semester, only to have them sitting on a shelf afterwards not being opened. Those hundreds of dollars spent on semester long textbooks could have been better utilized for students on other needs such as groceries or rent. We want to make a website to give students that option, giving them a website where they can download their desired textbooks for free.

### 1.1.3 Project Goal

The Goal of this project is to provide a free and open source infrastructure where users can read books from the public domain. We wish to provide an easy-to-use ecosystem, which allows the user to read or review books with ease.

### 1.1.4 Benefits to Users

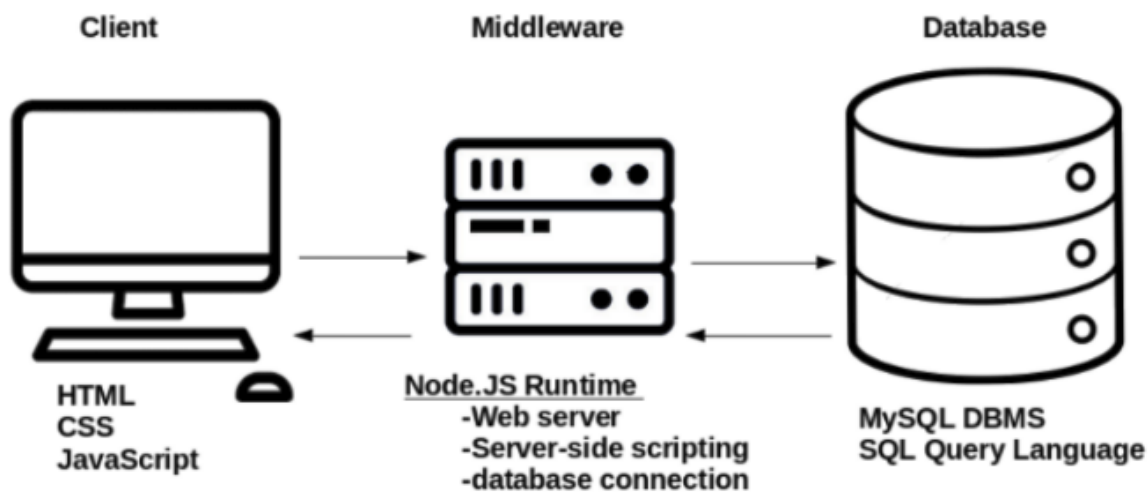
Users will have a community where they can **read and review books for free**. Right now, websites such as this exists, but they only serve one of the two desired functionalities. Websites like [gutenberg.org](http://gutenberg.org) give the users access to free book repositories, but do not allow for users to read and write reviews on books. Conversely, websites like [bookpage.com](http://bookpage.com) provide reviews for books, but they do not provide free access to such books. Our web application

will take the best aspects of both these websites to provide the user an efficient solution for picking and reading online books for free.

### 1.1.5 Stakeholders and Importance

The stakeholders of our applications will be students who want to find free textbook alternative to the paid bookstore alternative, as well as casual and dedicated book readers who can find free books online and download them for instant reading. This application is important because it provides a streamlined way for book-readers to gain easy access to the books they want and get them in a timely manner.

## 1.2 System Environment



### 1.2.1 Presentation Layer

- **HTML**
  - Purpose: HTML is the markup language supported by all major web-browsers. It allows content to be presented to a readable manner to the end-user. We will take advantage of HTML formatting tools such as hyperlinks, tables, and lists. Further, we will be using HTML forms for user input of email, username, password, search boxes, and comments.
  - Version: HTML 5
- **Cascading Style Sheets (CSS)**
  - Purpose: CSS will be used to improve the user experience of our web application. CSS gives us the ability to make animations, colored content, as well as more control of the appearance of our content.
  - Version: CSS 3
  - Framework: Bootstrap

- JavaScript (Client Side):
  - Purpose: Provide interactivity such as collection of user input, improve visual responsiveness, and sending alerts to users.
  - Version: ECMA2016

### 1.2.2 Application Layer

- Web Server: Node.js HTTP Module
  - Version: 10.6.3 LTS
- Server Side Application Language: JavaScript
  - Purpose: Provide commination between the presentation layer and the database layer
  - Version: EMCAScript 2016
- NPM Package Manger:
  - Purpose: Provide easy managment of external Node.js libraries such as express js, sql module, connect module, and http module.
  - Version: 6.9.0

### 1.2.3 Data Layer

- This web application will require the use of a relational database management system (RDBMS), the specific RDBMS we will use is MySQL.
- We will take advantage of the SQL programming language for tasks such as data definition, manipulation, query, control,and transaction control.
- MySQL RDBMS will take care of concurrency control, and will maintain the ACID principle for our database.
- MySQL Version: 5.7.27
- Query Language: SQL

### 1.2.4 Hardware/ Software Used

- Client Software Requirements
  - A web browser supporting the following is required to run the web application:
    - \* ECMAScript2016
    - \* HTML5
    - \* CSS3
  - Since the 3-tier architecture will only be virtual, (no remote web server or DMBS), the client will be required to install the proper versions of Node.js as well as MySQL as specified.

- MySQL and Node.js are available on many operating systems such as Linux, MacOS, Microsoft Windows, FreeBSD, and OpenBSD
- Client Hardware Requirements:
  - Any hardware with support by the above software will be sufficient to run the web application.

## 1.3 Functional Requirements

Because this application is to be accessed from a web browser, the **input** is a keyboard or mouse, while the **output** is stylized HTML to be displayed on a web browser.

### 1.3.1 Comprehensive List of Features

- Search Book:
  - Users can search for books that on the system database
  - Input: ISBN, author, or title
  - Output: list of books that match with the input information
- Order Search Results:
  - User can order the book search results. Default search behavior is based on the number of favorites a book has received
  - Input: release date or number of favorites
  - Output: ordered list of book that has characteristic from input
- Select Book:
  - Once user finds the desired book, they can select the book and view its profile.
  - Input: selecting single book
  - Output: book profile that includes book information and comments
- View Book Profile:

A book's profile will consist of the following:

  - The title of the book,
  - The author,
  - the release date,
  - the publisher,
  - the ISBN,
  - the reviews/comments left for the book,
  - the number of "favorites" the book has received

- Read/Download Book **requires login**:
  - User can download the book and view it in their web browser.
  - Input: selecting single book
  - Output: reading page or download to local
- Add/delete book to/from favorites **requires login**:
  - User can add the list to their "favorites", indicating that they enjoyed the book
  - Input: selecting single book
  - Output: add/delete book to/from the favorited book list that can be viewed on user's profile
- View Favorites **requires login**:
  - User can browse their list of favorited books, with the same sorting/filtering mechanisms as noted above
  - Input: Click view favorite book list in user profile
  - Output: list of books that users marked as favorite
- Leave Book Review **-requires login**:
  - Users can leave comments on the profiles of certain books
  - Input: comments that relating to the selected book
  - Output: comments added to the book's repository
- Register as User:
  - User can register for an account if they want to have the ability to comment and favorite books
  - Input: user information includes username, password, email
  - Output: user's account created
- Login as User:
  - Once registered, users will have the ability to login using the credentials they have provided during registration
  - Input: email and password
  - Output: login successfully give the access to login user's profile
- Go to home page:
  - The homepage will have information about the website and recent news regarding the website
  - Input: clicked website logo on navigator bar



- Output: redirect user to homepage
- View profile **-requires login**:
  - All profiles are anonymous because this is **not a social network**. Users are not allowed to change their username.
  - Input: successfully login
  - Output: Profile page includes username and email information, favorites book list
- Log out:
  - login user exits login condition, become visitor to the website
  - Input: login user clicks "Log Out" button
  - Output: becomes visitor and no longer has access to profile page, return to homepage if logout successfully

## 1.4 Non-Functional Issues

### 1.4.1 Graphical User Interface

The Graphical interface of the system will have the following qualities: attractiveness, usability, and responsiveness

**Attractiveness** The GUI will have a color palate which makes the system attractive to users. We will use high quality fonts and images where applicable. The GUI as a whole will have a coherent and consistent theme.

**Usability** The GUI will be intuitive. Users will not have to read documentation on how to use the system in order to use it. Options available to the user will be kept to a minimum as to encourage simplicity.

**Responsiveness** Users will be aware that their requests have been recorded. For example, a user clicks a button, there will be an indication that the button has been successfully clicked, such as a change in color. We will keep bloat to a minimum, such as animations and videos, as to encourage responsiveness for older hardware.

### 1.4.2 Security

**User Security** To protect user data and privacy, users entities will have the following attributes:

- A unique username
- A password, required to follow the minimum standards set by the system: which is a minimum of 8 characters, 1 uppercase character, 1 lowercase character, 1 numerical character, and 1 special character

**SQL Injection Protection** To protect from SQL injections, we will take precautions to inspect all user input and escape any characters which could breach security.

**Encryption** Any sensitive information collected from users will be encrypted fields in the relational database. All passwords will be hashed and we will take advantage of MySQL's built in password functionality.

**Anonymity** When a user makes a comment, **only their username will show up**, not their real name. Anonymity protects users from being identified against their will, and prevents personal attacks or blackmail. Users on this web application should be allowed to enjoy and review any books they like, and not have to be worried about ramifications.

### 1.4.3 Concurrency Control

The Relational Database Management System (RDBMS), MySQL, will take care of all needs relating to concurrency control. From the user's perspective, all transactions can be made concurrently.

### 1.4.4 Access Control

#### Types of Users

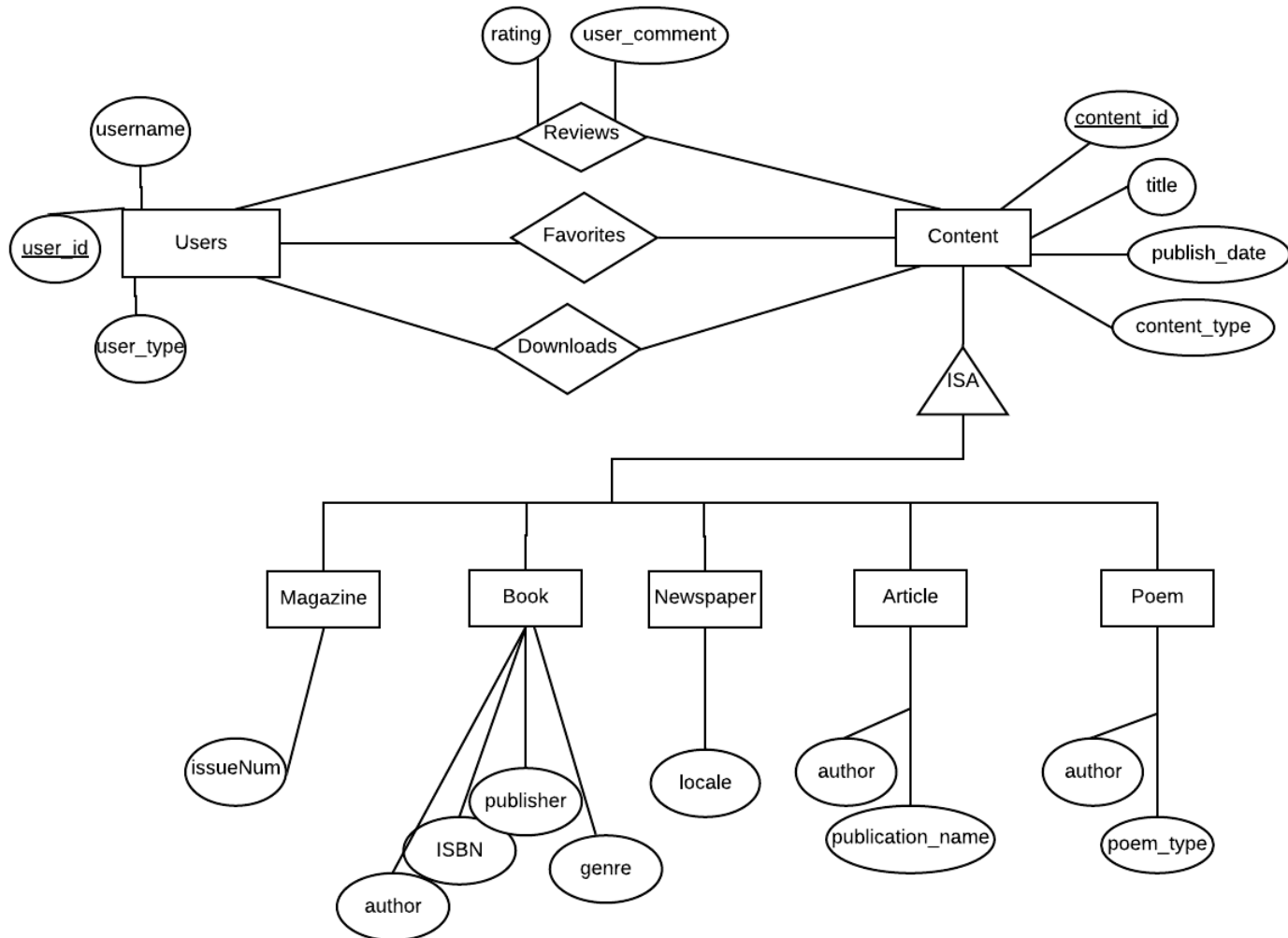
- **User** Users will only have read access to the following:
  - Their own account profile
  - Book repositories with open access

In addition, users will have write access to the following:

- Their own account profile information
  - Their favorites list
  - Their downloads list
  - Reviews for books
- **Administrator** Administrators have access **all user functionality** plus extra permissions to data which is not available to the general public. Admins have the following additional privilege:
  - Delete comments/reviews by users

## 2 Database Project Design

### 2.1 Updated Entity-Relationship Diagram



### 2.2 Database Schema

- Users(user\_id, username, use\_type)
- Reviews(user\_id, content\_id, rating, user\_comment)
- Favorites(user\_id, content\_id)
- Downloads(user\_id, content\_id)
- Content(content\_id, title, publish\_date, content\_type)
- Article(content\_id, author, publication\_name)

- Magazine(content\_id, issueNum)
- Book(content\_id, ISBN, author, genre, publisher)
- Newspaper(content\_id, locale)
- Poem(content\_id, poem\_type, author)

## 2.3 Entities

### 2.3.1 Users

Users are defined as those who have a valid account login for the website. The attributes for Users are as follows:

- user\_type: A user can be of two types, admin or regular. Admins have special privileges
- username: The unique username by which a user logs into the web application
- user\_id: the unique identifier by which a user's information is tracked

### 2.3.2 Content

Content comes in many forms. There are four subclasses of Content, but it possible that there exist an item that is only classifiable as Content and none of its subclasses. The attributes for Content are as follows:

- content\_id: The unique content identifier
- title: The name/title of the content
- publish\_date: The date the content was published, if available
- content\_type: the type of content, if applicable.

### 2.3.3 Books

Books are a special case of Content. They have special attributes which sets them apart; however, books inherit the primary key from content. In addition, books have the following attributes:

- ISBN: the unique book ISBN number
- publisher: the publisher of the book
- author: The author of the book
- genre: the genre of the book

### 2.3.4 Magazines

Magazines are a special case of Content. They have special attributes which sets them apart; however, magazines inherit the primary key from content. In addition, magazines have the following attributes:

- `issueNum`: the issue number of the magazine.

### 2.3.5 Newspapers

Newspapers are a special case of Content. They have special attributes which sets them apart; however, Newspapers inherit the primary key from content. In addition, magazines have the following attributes:

- `locale`: the locality of a Newspaper, if available.

### 2.3.6 Articles

Articles are standalone pieces of writing which may, in some cases, be significant enough to publish on the web application as it's own piece of media. Articles inherit the primary key from Content but have additional relationships, as described in the relationships subsection. Articles also have the following additional attributes:

- `author`: the author of the article
- `publication_name`: the name of the publication where the article is from.

### 2.3.7 Poems

Poems are a specialized form of content, usually artistic in nature. Poems inherit the primary key from Content but have additional attributes, as described in the relationships subsection. Articles also have the following additional attributes:

- `author`: the author of the poem (if available)
- `poem_type`: the type of poem (ballad, epic, prose, haiku, etc.)

## 2.4 Relationships

### 2.4.1 User Reviews Content

A user (admin or regular) is the author of zero or many reviews on zero or many pieces of content.

The attributes for Reviews are as follows:

- `rating`: a star rating between 1 and 5
- `user_comment`: a comment about the piece of content (optional)

### 2.4.2 User Favorites Content

A user can favorite content, which allows them to keep track of their favorite items. Content can be favorited by zero or many users. Users can favorite zero or many pieces of content.

### 2.4.3 User Downloads Content

We would like to keep track of user's download history. A user can download zero or many pieces of content, and a piece of content may be downloaded by zero or many users.

## 2.5 Boyce Codd Normal Form (BCNF) Proofs

### 2.5.1 Users — BCNF Proof

*Claim: the Users relation is in BCNF*

*Proof.* The relational schema for the **Users** table is:

$Users(user\_id, user\_type, username)$

The minimal cover of the functional dependencies for Users is:

$user\_id \rightarrow user\_type, username$

Now, we have to check if  $user\_id$  is a superkey, because it is the left side of our only functional dependency.

Computing the closure of  $user\_id$ , we find that...

$user\_id^+ = \{user\_id, username, user\_type\}$

Which means that  $user\_id$  is a superkey, so the only non-trivial functional dependency in the minimal cover of FDs for this relation does **not** violate BCNF

$\therefore Users \in BCNF$  □

### 2.5.2 Reviews — BCNF Proof

*Claim: the Reviews relation is in BCNF*

*Proof.* The relational schema for the **Reviews** table is:

$Reviews(user\_id, content\_id, user\_comment, rating)$

The minimal cover of the functional dependencies for Reviews is:

$user\_id, content\_id \rightarrow user\_comment, rating$

Now, we have to check if the set of attributes  $\{user\_id, content\_id\}$  is a superkey, because it is the left side of our only functional dependency.

Computing the closure of  $\{user\_id, content\_id\}$ , we find that...

$\{user\_id, content\_id\}^+ = \{user\_id, content\_id, user\_comment, rating\}$

Which means that  $\{user\_id, content\_id\}$  is a superkey, so the only non-trivial functional dependency in the minimal cover of FDs for this relation does **not** violate BCNF

$\therefore Reviews \in BCNF$  □

### 2.5.3 Favorites — BCNF Proof

*Claim: the Favorites relation is in BCNF*

*Proof.* The relational schema for for the **Favorites** table is:

*Favorites*(*user\_id*, *content\_id*)

There are no non-trivial Functional Dependencies in this relation, so it is true that for all non-trivial FD's  $\in$  Favorites (none), the left side of the FD is a superkey.

$\therefore \text{Favorites} \in \text{BCNF}$

□

### 2.5.4 Downloads — BCNF Proof

*Claim: the Downloads relation is in BCNF*

*Proof.* The relational schema for for the **Downloads** table is:

*Downloads*(*user\_id*, *content\_id*)

There are no non-trivial Functional Dependencies in this relation, so it is true that for all non-trivial FD's  $\in$  Downloads (none), the left side of the FD is a superkey.

$\therefore \text{Downloads} \in \text{BCNF}$

□

### 2.5.5 Poem — BCNF Proof

*Proof.* The relational schema for for the **Poem** table is:

*Poem*(*content\_id*, *poem\_type*, *author*)

The minimal cover of the functional dependencies is:

$\text{content\_id} \rightarrow \text{poem\_type}, \text{rating}$

Now, we have to check if the attribute *content\_id* is a superkey, because it is the left side of our only functional dependency.

Computing the closure of  $\{\text{content\_id}\}$ , we find that...

$\{\text{content\_id}\}^+ = \{\text{content\_id}, \text{author}, \text{poem\_type}\}$

Which means that  $\{\text{content\_id}\}$  is a superkey, so the only non-trivial functional dependency in the minimal cover of FDs for this relation does **not** violate BCNF

$\therefore \text{Poem} \in \text{BCNF}$

□

### 2.5.6 Content — BCNF Proof

*Proof.* The relational schema for for the **Content** table is:

*Content*(*content\_id*, *content\_type*, *title*, *date\_published*)

The minimal cover of the functional dependencies is:

$\text{content\_id} \rightarrow \text{content\_type}, \text{title}, \text{date\_published}$

Now, we have to check if the attribute *content\_id* is a superkey, because it is the left side of our only functional dependency.

Computing the closure of  $\{\text{content\_id}\}$ , we find that...

$\{\text{content\_id}\}^+ = \{\text{content\_id}, \text{content\_type}, \text{title}, \text{date\_published}\}$

Which means that  $\{content\_id\}$  is a superkey, so the only non-trivial functional dependency in the minimal cover of FDs for this relation does **not** violate BCNF  
 $\therefore Content \in BCNF$  □

### 2.5.7 Magazine — BCNF Proof

*Proof.* The relational schema for for the **Magazine** table is:

$Magazine(content\_id, issueNum)$

The minimal cover of the functional dependencies is:

$content\_id \rightarrow issueNum$

Now, we have to check if the attribute  $content\_id$  is a superkey, because it is the left side of our only functional dependency.

Computing the closure of  $\{content\_id\}$ , we find that...

$\{content\_id\}^+ = \{content\_id, issueNum\}$

Which means that  $\{content\_id\}$  is a superkey, so the only non-trivial functional dependency in the minimal cover of FDs for this relation does **not** violate BCNF

$\therefore Magazine \in BCNF$  □

### 2.5.8 Article — BCNF Proof

*Proof.* The relational schema for for the **Article** table is:

$Article(content\_id, author, publication\_name)$

The minimal cover of the functional dependencies is:

$content\_id \rightarrow author, publication\_name$

Now, we have to check if the attribute  $content\_id$  is a superkey, because it is the left side of our only functional dependency.

Computing the closure of  $\{content\_id\}$ , we find that...

$\{content\_id\}^+ = \{content\_id, author, publication\_name\}$

Which means that  $\{content\_id\}$  is a superkey, so the only non-trivial functional dependency in the minimal cover of FDs for this relation does **not** violate BCNF

$\therefore Article \in BCNF$  □

### 2.5.9 Newspaper — BCNF Proof

*Proof.* The relational schema for for the **Article** table is:

$Newspaper(content\_id, locale)$

The minimal cover of the functional dependencies is:

$content\_id \rightarrow locale$

Now, we have to check if the attribute  $content\_id$  is a superkey, because it is the left side of our only functional dependency.

Computing the closure of  $\{content\_id\}$ , we find that...

$\{content\_id\}^+ = \{content\_id, locale\}$

Which means that  $\{content\_id\}$  is a superkey, so the only non-trivial functional dependency in the minimal cover of FDs for this relation does **not** violate BCNF



$\therefore \text{Newspaper} \in \text{BCNF}$

□

### 2.5.10 Book — BCNF Proof

*Proof.* The relational schema for the **Book** table is:

$\text{Book}(\text{content\_id}, \text{ISBN})$

The minimal cover of the functional dependencies is:

$\text{content\_id} \rightarrow \text{ISBN}, \text{author}, \text{publisher}, \text{genre}$

Now, we have to check if the attribute  $\text{content\_id}$  is a superkey, because it is the left side of our only functional dependency.

Computing the closure of  $\{\text{content\_id}\}$ , we find that...

$\{\text{content\_id}\}^+ = \{\text{content\_id}, \text{ISBN}, \text{author}, \text{publisher}, \text{genre}\}$

Which means that  $\{\text{content\_id}\}$  is a superkey, so the only non-trivial functional dependency in the minimal cover of FDs for this relation does **not** violate BCNF

$\therefore \text{Book} \in \text{BCNF}$

□

## 2.6 Database Table Instances

### 2.6.1 Users

user_id	username	user_type
1	bobby12	regular
10	AI	regular
11	boot	regular
12	goolger	regular
13	password	regular
14	enterpirseUser	regular
15	dispo341	regular
2	tim12	admin
3	joe01	admin
4	tim	admin
5	joseph	regular
6	mikeWU	regular
7	timard	regular
8	WUmiek	regular
9	jame	regular

### 2.6.2 Reviews

user_id	content_id	rating	user_comment
1	23	3	fun
1	28	1	good book
1	46	3	nicee
1	49	5	decent
1	59	1	NULL
1	60	2	NULL
1	61	1	NULL
1	66	1	NULL
10	18	1	NULL
2	52	4	bad book
5	22	2	boring
5	24	1	bad
5	73	4	NULL
7	61	2	NULL
8	55	7	I love this book so much it is so good best book ever!
8	62	2	NULL

### 2.6.3 Favorites

user_id	content_id
1	1
1	4
1	7
1	12
1	17
10	4
11	4
12	4
12	6
13	14
14	4
19	4
5	2
6	4
8	4

### 2.6.4 Downloads

user_id	content_id
1	1
1	5
1	7
1	12
1	17
10	4
11	4
12	4
12	6
13	14
14	4
19	4
5	2
6	4
8	4

### 2.6.5 Poem

content_id	author	poem_type
61	Georgre Bush	Haiku
62	James Jones	Haiku
63	James Reds	Ballad
64	John Renyolds	Ballad
65	Patrick Renyolds	Ballad
66	Harry Renyolds	Ballad
67	Daisy Renyolds	Haiku
68	Patrick James	Haiku
69	Patrick Bush	Ode
70	Patrick Renyolds	Ballad
71	James Bush	Haiku
72	Tomard Renyolds	Love
73	Gustavo Renyolds	Love
74	Frank Renyolds	Ballad
75	J. Renyolds	Ballad

## 2.6.6 Content

content_id	title	publish_date	content_type
1	Snowbaording Magazine	2018-08-01	magazine
2	Snowbaording Magazine	2018-09-01	magazine
3	Snowbaording Magazine	2018-10-01	magazine
4	Snowbaording Magazine	2018-11-01	magazine
5	Snowbaording Magazine	2018-12-01	magazine
6	Snowbaording Magazine	2019-01-01	magazine
7	Snowbaording Magazine	2019-02-01	magazine
8	Snowbaording Magazine	2018-03-01	magazine
9	Snowbaording Magazine	2019-04-01	magazine
10	Snowbaording Magazine	2019-05-01	magazine
11	Snowbaording Magazine	2019-06-01	magazine
12	Snowbaording Magazine	2019-07-01	magazine
13	Snowbaording Magazine	2019-08-01	magazine
14	Snowbaording Magazine	2019-09-01	magazine
15	Snowbaording Magazine	2019-10-01	magazine
16	In the Electric Mist	2019-02-01	book
17	Babylon 5	2018-04-01	book
18	Mortuary	2017-11-11	book
19	Counterfeit Coin	2018-06-05	book
20	Ah, Wilderness!	2018-10-13	book
21	Independencia	2018-10-24	book
22	Lost Son, The	2018-11-20	book
23	Wild Child	2019-07-14	book
24	Little Big Soldier	2018-11-18	book
25	Medea	2017-12-16	book
26	One Fine Spring Day	2019-07-05	book
27	Galaxina	2018-12-18	book
28	Godzilla 2000	2017-11-09	book
29	Free Soul, A	2019-08-21	book
30	Attack on the Iron Coast	2019-04-30	book
31	New York Time	2018-06-28	newspaper
32	New York Time	2018-04-10	newspaper
33	New York Time	2018-10-04	newspaper
34	New York Time	2018-08-30	newspaper
35	New York Time	2018-05-29	newspaper
36	New York Time	2019-09-05	newspaper
37	New York Time	2019-08-16	newspaper
38	New York Time	2018-12-01	newspaper
39	New York Time	2018-07-24	newspaper
40	New York Time	2018-09-07	newspaper
41	New York Time	2018-09-17	newspaper
42	New York Time	2019-01-18	newspaper
43	New York Time	2019-05-13	newspaper
44	New York Time	2019-10-02	newspaper
45	New York Time	2018-04-30	newspaper
46	Why WWII was Bad	1994-12-11	article
47	WWI -- The Good War	1994-12-10	article
48	Vietnam War: 20 Year Later	1994-12-09	article
49	Cold War	1994-12-01	article
50	The Good War	1994-12-01	article
51	The Bad War	1994-12-01	article
52	10 Reasons Why Poems are Good	1994-12-05	article
53	5 Reasons to use NoSQL	1994-12-05	article
54	7 Reaons to use MySQL	1994-12-05	article
55	12 Reasons to Normalize Database	1994-12-05	article
56	Neural Nets for Preschoolers	1994-12-11	article
57	How to Perform Surgery	1994-12-11	article
58	How to write ER Diagrams	1994-12-01	article
59	SJSU Announces New Gym	1994-12-01	article
60	SJSU Power Outage	1994-12-12	article
61	A Walk	1666-01-11	poem
62	A Run	1266-01-11	poem
63	A Jog	1631-12-10	poem
64	A Dog	1435-10-10	poem
65	A CAT	1123-10-10	poem
66	A Harry Potter	1823-01-09	poem
67	Table	1923-01-09	poem
68	I love Databases	1941-11-09	poem
69	Best Poem Ever	1921-01-09	poem
70	Consumerism	1941-01-09	poem
71	Capitalism	1952-01-09	poem
72	Canda	2035-11-09	poem
73	Food	1635-11-09	poem
74	A Long Run in the Park	1835-04-12	poem
75	A Ballad to James	1935-11-12	poem

### 2.6.7 Magazine

content_id	issueNum
1	291
2	292
3	293
4	294
5	295
6	296
7	297
8	298
9	299
10	300
11	301
12	302
13	303
14	304
15	305

### 2.6.8 Article

content_id	author	publication_name
46	Patrick	New York Times
47	Jones	New York Times
48	Tim Cook	New York Times
49	Steve Job	New York Times
50	Bill Gate	New York Times
51	Tom Gate	New York Times
52	Michael Gate	New York Times
53	John Lennon	New York Times
54	John Lennon	New York Times
55	Timard Jones	New York Times
56	Mike Wu	New York Times
57	Mike Wu	New York Times
58	Harry Potter	New York Times
59	Harry Potter	New York Times
60	James Potter	New York Times

### 2.6.9 Newspaper

content_id	locale
31	New York City, NY
32	New York City, NY
33	New York City, NY
34	New York City, NY
35	New York City, NY
36	New York City, NY
37	New York City, NY
38	New York City, NY
39	New York City, NY
40	New York City, NY
41	New York City, NY
42	New York City, NY
43	New York City, NY
44	New York City, NY
45	New York City, NY

### 2.6.10 Book

content_id	ISBN	author	publisher	genre
16	304555343-5	Murdoch Cave	Leela Buterton	Comedy
17	659196532-7	Goldy Aimeric	Ali Gaitskell	Comedy
18	491743522-6	Jelene Fancet	Linzy Shelf	Comedy
19	346181073-8	Paulie Ouldcott	Adrianna Gresch	Adventure
20	868282781-6	Taber Deguara	Shanta Daouze	Horror
21	109670642-3	Clareta O'Carran	Winifield	Sci-Fi
22	239189700-6	Osbourne Kingsly	Antonella Fancett	Crime
23	671528801-1	Stevena Gotling	Mabelle MacFaul	Drama
24	583092606-7	Debi Vala	Madelon Croux	Drama
25	674056408-6	Merilee Blackall	Oates Raspison	Crime
26	273454549-7	Wanda Morbey	Shawn Kosel	Crime
27	544491903-6	Clyde Marginson	Jocko Connors	Comedy
28	469558624-X	Magdaia Killelay	Prudence Rudsdel	Crime
29	541188048-3	Philippa Frays	Noel Doerrling	Action
30	133956813-6	Sharron Earley	Roze Sterzaker	Comedy

## 3 Implementation

### 3.1 Implementation Overview

On the server, our project uses the Node.js runtime environment and express.js framework for routing and http requests/ responses. Each HTML page is implemented using EJS (Embedded JavaScript) as a templating engine.

Here is a general overview of how the application works:

- The server is initialized, this includes:
  - Claiming the 8080 port of local host using express.js — `app.listen()`
  - Establishing a database connection, using the Node.js MySQL module — `mysql.createConnection(JSON obj)`
  - Defining where all static content is located (files, photos, etc), using express.js — `express.static(/path/to/static/content)`
  - Initializing all routes/ modules, such as content, auth, and search
- HTTP requests from the client trigger various database operations such as INSERT and SELECT, which are executing using the MySQL module function `database.query(string sql_code, function callback)`. `Database.query()` is an asynchronous function, meaning that there is no guarantee that it will be executed immediately upon calling it. For this reason, we often must wrap our server HTTP responses in the callback function, assuming we want to use data in our webpages which we have just queried.
- Dynamic content queried from the database is then served using EJS templates. EJS allows us to embed server-like JavaScript code into our HTML code. The templating engine then truncates the js/HTML hybrid into proper HTML code for the browser to parse.

## 3.2 Detailed Use-case Implementation

### 3.2.1 Use Case: Create Account

For account creation, we use Google Firebase for storing and validating user credentials. Once we extract the user information from the HTML form, we use the `firebase.auth` function, `createUserWithEmailAndPassword()`. If a user is successfully created, we can then extract the unique user ID which firebase provides us to store the user in our own MySQL bookstore.User table. Firebase ensures proper password encryption.

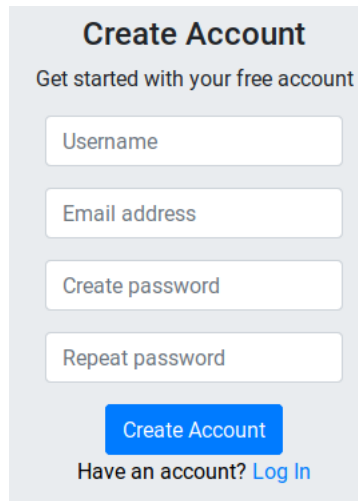


Figure 1: client's perspective: user creating account

```
else if (password === confirm_password) {  
  firebase.auth().createUserWithEmailAndPassword(email, password)  
    .then(function(user){  
      sql = `INSERT INTO Users (user_id, username) VALUES ('${firebase.auth().currentUser.uid}', '${username}')`  
      server.database.query(sql, function(err, results)  
      {  
        if (err) {  
          console.log(err);  
          return;  
        }  
        console.log('User created successfully');  
      });  
    });  
}
```

Figure 2: Server: user creating account — code

### 3.2.2 Use Case: Log In

For account log in, we use Google Firebase for storing and validating user credentials. Once we extract the email and password from the HTML form, we use the `firebase.auth` function, `signInWithEmailAndPassword()`. If a user successfully logs in we can then extract the unique user ID which firebase provides us to retrieve information about the user in our MySQL database. Upon login, the user is redirected to the homepage using the `express.js` function `response.redirect(string path)`.

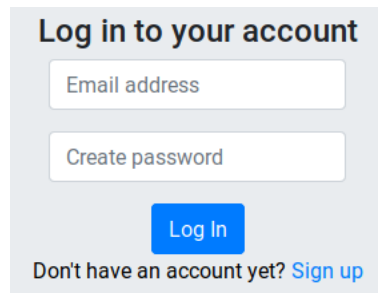


Figure 3: client's perspective: user logging in

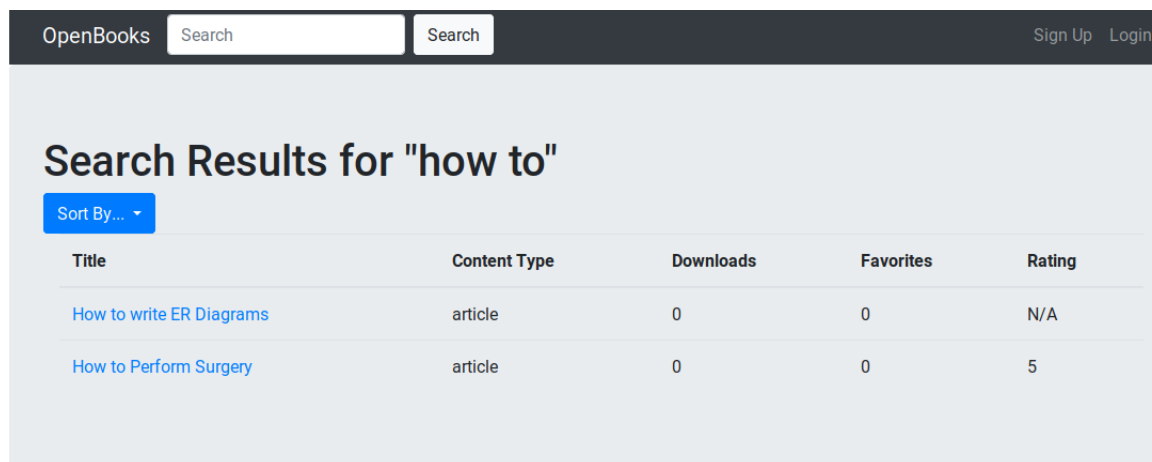
```
router.post('/login/submit', function (req, res) {  
  const email = req.body.email;  
  const password = req.body.password;  
  firebase.auth().signInWithEmailAndPassword(email, password)  
    .then(function(response) {  
      res.redirect('/')  
    })  
    .catch(function(error) {  
      var errorMessage = error.message;  
      res.render("pages/login", {error: error});  
    });  
});
```

Figure 4: Server: user logging in — code



### 3.2.3 Use Case: Search for Content

To search for a book, user use the search bar found at the header of each page. Search queries are done via a HTTP POST request, which sends data from client to server. Once the server receives the POST request, it queries the database for the search term. Users are able to search by author, title, publisher, and more, therefore our SQL query must look check many attributes. For this reason, we use a LIKE clause to ensure that non-exact matches are still found, by using wildcard (%) operators. Once the search results are found, we parse the JSON provided by the MySQL module, format it into an HTML table, then send the HTML to the EJS templating engine. Once the search is executed, users can decide if they want the results to be ordered.



Title	Content Type	Downloads	Favorites	Rating
<a href="#">How to write ER Diagrams</a>	article	0	0	N/A
<a href="#">How to Perform Surgery</a>	article	0	0	5

Figure 5: client's perspective: search results

```
router.post('/find*', function (req, res) {
  order_by = 'results.content_id'
  if(req.url !== "/find") order_by = req.url.replace('/find/', '')
  if(req.body.q !== undefined)
    q = req.body.q
  sql = `SELECT results.*, reviews.num_reviews, downloads.num_downloads, reviews.avg_rating, favorites.num_favorites FROM
    (SELECT Content.content_id, Content.title, Content.content_type FROM Content LEFT JOIN Book on Book.content_id=Content.content_id
    LEFT JOIN Poem ON Poem.content_id=Content.content_id
    LEFT JOIN Article on Article.content_id=Content.content_id
    LEFT JOIN Newspaper ON Newspaper.content_id=Content.content_id
    LEFT JOIN Magazine on Magazine.content_id=Content.content_id
    WHERE CONCAT(content_type, IFNULL(title, ''), IFNULL( Book.author, ''), IFNULL( Book.ISBN, ''),
    IFNULL( Poem.poem_type, ''), IFNULL( Book.genre, ''), IFNULL(Article.author, ''),
    IFNULL(Poem.author, ''),
    IFNULL(Article.publication_name, ''))
    LIKE '%${q}%' ) results
  LEFT JOIN
  (SELECT content_id, COUNT(*) as num_reviews, AVG(rating) AS avg_rating FROM Reviews GROUP BY content_id) reviews
  ON results.content_id = reviews.content_id
  LEFT JOIN
  (SELECT content_id, COUNT(*) as num_downloads FROM Downloads GROUP BY content_id) downloads
  ON results.content_id = downloads.content_id
  LEFT JOIN
  (SELECT content_id, COUNT(*) as num_favorites FROM Favorites GROUP BY content_id) favorites
  ON results.content_id = favorites.content_id
  ORDER BY ${order_by} DESC`
  rows = ''
  server.database.query(sql, function(err, results){
```

Figure 6: Server: searching through content/poem/book/magazine/newspaper/article tables

### 3.2.4 Use Case: View Profile

Once logged in, users can view their profile. Users view their profile by clicking the "Profile" link on the top right of the header. A users profile shows the content they've downloaded and favorited. To find the books which a user has favorited/downloaded JOIN the Users/Downloads tables and the Users/Favorites tables; in both cases we JOIN using the user\_id as the attribute to join on.

The screenshot shows a web application interface for a user profile. At the top, there's a dark header with 'OpenBooks' on the left, a search bar in the center, and 'My Profile' and 'Log out' links on the right. Below the header, a light gray banner says 'Welcome back, MikeWu2012!'. Underneath, the 'Favorites' section contains a table with 8 rows of favorited items. Below that, the 'Downloads' section contains a table with 2 rows of downloaded items.

Title	Content Type
Snowbaording Magazine	magazine
Counterfeit Coin	book
New York Time	newspaper
12 Reasons to Normalize Database	article
Table	poem
Capitalism	poem
Food	poem
A Long Run in the Park	poem

Title	Content Type
A CAT	poem
A Long Run in the Park	poem

Figure 7: client's perspective: user profile

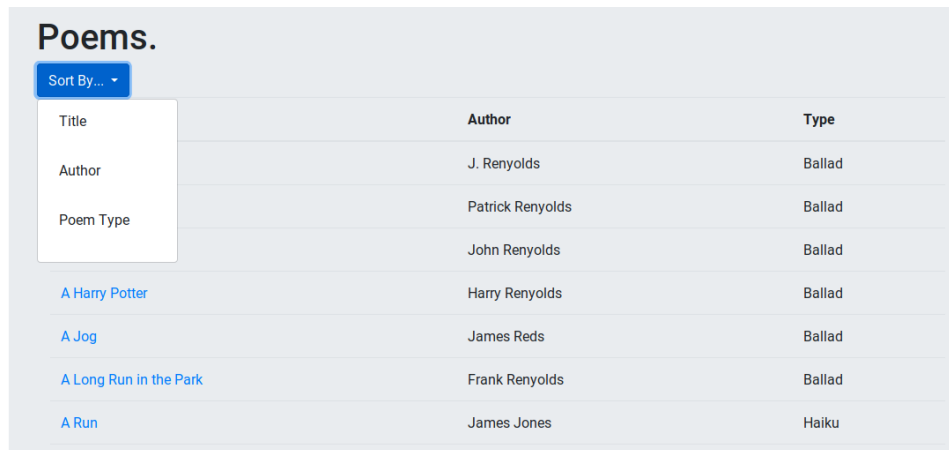
```
app.get("/profile", function (req, res) {
  if(currentUser.username != null)
  {
    rows = ''
    sql= "SELECT title, content_type, content_id FROM Content WHERE content_id IN (SELECT content_id From Favorites WHERE user_id='"+
      + currentUser.user_id + "'); "
    sql += "SELECT title, content_type, content_id FROM Content WHERE content_id IN (SELECT content_id From Downloads WHERE user_id='"+
      + currentUser.user_id + "');"
    database.query(sql, [1,2], function (err, results) {
```

Figure 8: Server: user profile — code

### 3.2.5 Use Case: Sorting Results

Where applicable, the user can sort results of various content. Content is sorted in several contexts. For example, when a user clicks "search" or "browse all content", the available sort options are title, content type, rating, number of downloads, number of favorites, or average rating. When specific content is being displayed, more options are available. For example, when a user is viewing the Poems page, they can sort by poem type or author. Books can be sorted by author, genre, publisher. Newspapers can be sorted by date or location. Articles can be sorted by author or publication. Magazines can be sorted by issue number. All content can be sorted by title.

Sorting is implemented using HTTP GET requests. For example, when a user is on the Books page, and they click "Sort by Title", they are redirected to the URL /books/title. We then re-render the page, this time displaying the results, this time using an SQL ORDER BY clause.



The screenshot shows a web interface for 'Poems'. At the top left, there is a blue button labeled 'Sort By...' with a downward arrow. Below this button is a white dropdown menu with three options: 'Title', 'Author', and 'Poem Type'. The main part of the page is a table with three columns: 'Title', 'Author', and 'Type'. The table contains six rows of data. The first row has 'J. Renyolds' as the author and 'Ballad' as the type. The second row has 'Patrick Renyolds' as the author and 'Ballad' as the type. The third row has 'John Renyolds' as the author and 'Ballad' as the type. The fourth row has 'Harry Renyolds' as the author and 'Ballad' as the type. The fifth row has 'James Reds' as the author and 'Ballad' as the type. The sixth row has 'Frank Renyolds' as the author and 'Ballad' as the type. The last row has 'James Jones' as the author and 'Haiku' as the type. The titles of the poems are: 'A Harry Potter', 'A Jog', 'A Long Run in the Park', and 'A Run'.

Title	Author	Type
	J. Renyolds	Ballad
	Patrick Renyolds	Ballad
	John Renyolds	Ballad
A Harry Potter	Harry Renyolds	Ballad
A Jog	James Reds	Ballad
A Long Run in the Park	Frank Renyolds	Ballad
A Run	James Jones	Haiku

Figure 9: client's perspective: sorting results

```
router.get("/", function (req, res) {
  let sql =
    "SELECT c.content_id, c.title, b.author, b.genre, b.publisher, b.ISBN FROM Book b INNER JOIN Content c on c.content_id = b.content_id";
  processBooks(req, res, sql);
});

router.get("/author", function (req, res) {
  let sql =
    "SELECT c.content_id, c.title, b.author, b.genre, b.publisher, b.ISBN FROM Book b INNER JOIN Content c on c.content_id = b.content_id ORDER BY b.author";
  processBooks(req, res, sql);
});

router.get("/publisher", function (req, res) {
  let sql =
    "SELECT c.content_id, c.title, b.author, b.genre, b.publisher, b.ISBN FROM Book b INNER JOIN Content c on c.content_id = b.content_id ORDER BY b.publisher";
  processBooks(req, res, sql);
});

router.get("/title", function (req, res) {
  let sql =
    "SELECT c.content_id, c.title, b.author, b.genre, b.publisher, b.ISBN FROM Book b INNER JOIN Content c on c.content_id = b.content_id ORDER BY c.title";
  processBooks(req, res, sql);
});

router.get("/genre", function (req, res) {
  let sql =
    "SELECT c.content_id, c.title, b.author, b.genre, b.publisher, b.ISBN FROM Book b INNER JOIN Content c on c.content_id = b.content_id ORDER BY b.genre";
  processBooks(req, res, sql);
});
```

Figure 10: Server: user profile — code

### 3.2.6 Use Case: View Content Profile

If a user is interesting in learning more about a piece of content, they can view the content's profile. The content profile contains extra information about the content such as when it was published, the publisher, the average rating, the number of favorites/downloads, and more. On the server, we query the database to find more information on the content the user is requesting. The user requests the content using an HTTP GET request. When a user clicks on a particular piece of content, first they are redirected to a URL which such as \$CONTENT\_TYPE/\$CONTENT\_ID. From there, we find extra information about the content depending on its subclass, then send all necessary information and redirect to /content-profile.

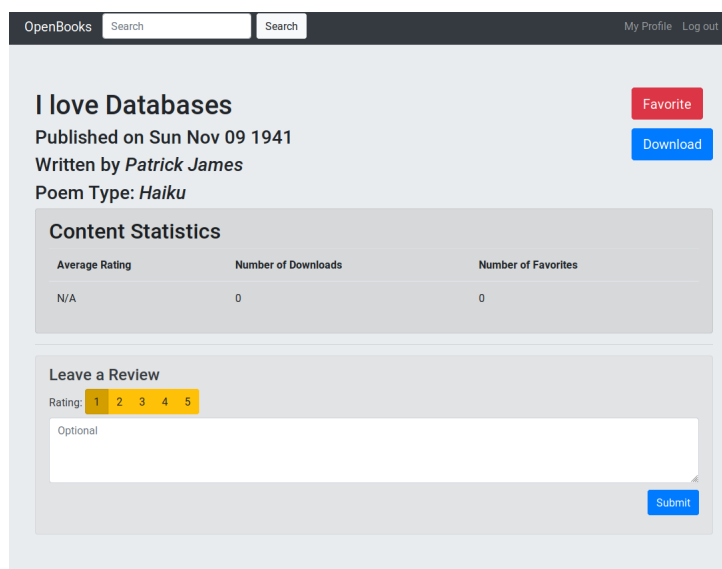


Figure 11: client's perspective: viewing content profile

```
router.get("/poem-profile/*", function (req, res) {
  poemNum = req.url.replace(/[^0-9]/g, "");
  let sql =
    `SELECT c.publish_date, c.content_type, c.content_id, c.title, p.author, p.poem_type
    FROM Poem p INNER JOIN Content c on c.content_id = p.content_id where c.content_id = ${poemNum}`;
  database.query(sql, function (err, results) {
    server.currentUser.currentContent.content_id = results[0].content_id;
    server.currentUser.currentContent.title = results[0].title;
    server.currentUser.currentContent.content_type = results[0].content_type;
    server.currentUser.currentContent.author = results[0].author;
    server.currentUser.currentContent.publish_date = results[0].publish_date;
    server.currentUser.currentContent.poem_type = results[0].poem_type;
    isFavorite(function(){
      res.redirect("/content-profile");
    })
  });
});
```

Figure 12: Server: querying database for content information

### 3.2.7 Use Case: Write Review

Once logged in, users can add reviews for content. A review does not require a comment, but does require a rating. Reviews are implemented using HTML forms and HTTP POST requests. The server inserts the data directly from the form into the MySQL bookstore.Reviews table.

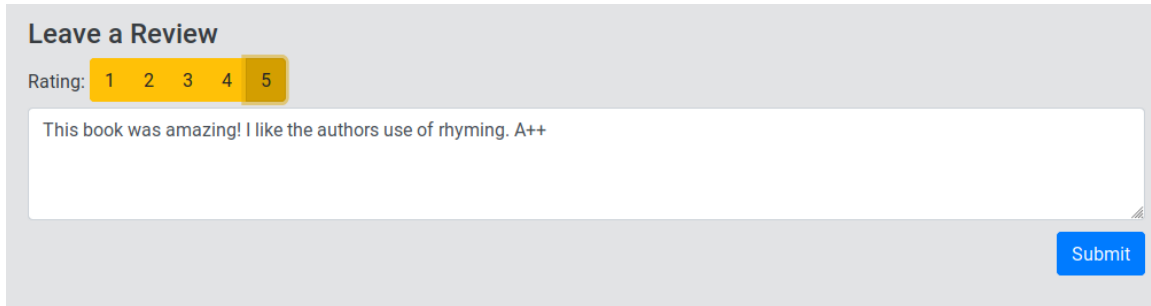


Figure 13: client's perspective: writing review

```
router.post('/submit', function(req,res){
  // insert the comment, reload the page
  let comment = req.body.review;
  let rating = req.body.rating;
  let uid = firebase.auth().currentUser.uid

  if(comment != '')
    values = "(" + uid + ", " + server.currentUser.currentContent.content_id + ", " + comment + ", " + rating + "));";
  else
    values = "(" + uid + ", " + server.currentUser.currentContent.content_id + ", null, " + rating + "));";

  let sql = "INSERT INTO Reviews (user_id, content_id, user_comment, rating) VALUES " + values;

  server.database.query(sql, function (err, results) {
    //once the query is done, we can reload the page.
    res.redirect("/content-profile")
  })
})
```

Figure 14: Server: insert reviews

### 3.2.8 Use Case: Read Review

On the content profile, we display all reviews left for the content. We get this data using a sequence of simple SELECT and JOIN statements. We format the JSON output from the MySQL Node.js module into HTML divisions to make the review more visually appealing.

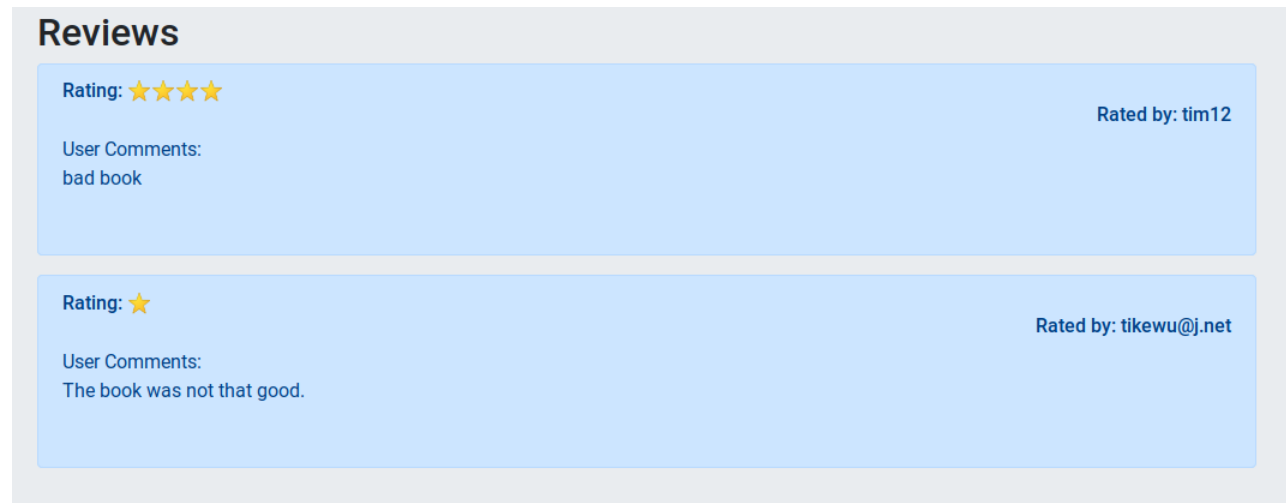


Figure 15: client's perspective: reading a review

```

reviewSql = "SELECT * FROM Reviews r, Users u WHERE r.content_id = " +
            currentUser.currentContent.content_id + " AND r.user_id = u.user_id"
server.database.query(reviewSql, function(err, results) //make query
{
    if (err) throw err;
    star = "★"
    comments = ""
    comments += "<div class=\"list-group\">"
    results.forEach(Element => { //process the reviews

```

Figure 16: Server: selecting review from database

### 3.2.9 Use Case: Favorite/ Download Content

Once logged in, users can favorite or download any content. Users can "undo" a favorite but cannot "undo" a download. On the top of each content page, there are two buttons: Favorite and Download. These are actually HTML links which redirect to URLs indicating we need to add/remove a favorite or add a download to/from its appropriate table. Once a user "favorites" a content, the "favorite" button on top of the page becomes a "remove favorite" button. This is accomplished using the EJS templating system, we make a simple statement that says if the user has favorite this content, show a different button (unfavorite).

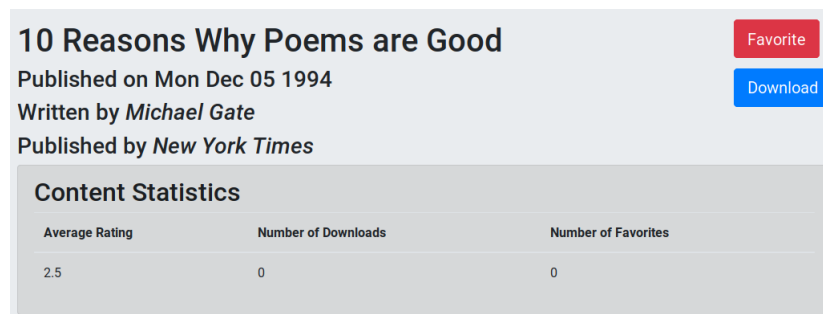


Figure 17: client's perspective: downloading/ favoriting content

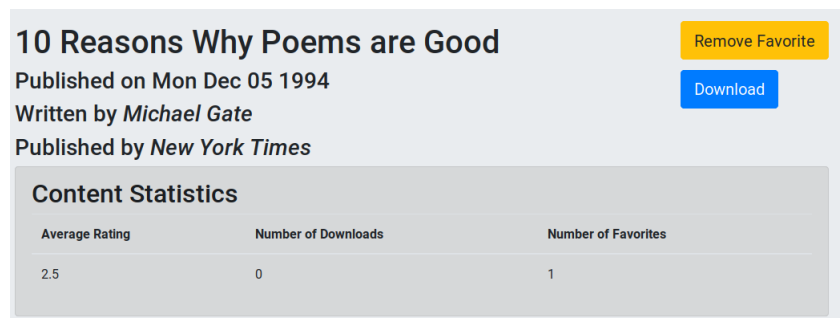


Figure 18: client's perspective: un-favoriting content

```
app.get("/download", (req, res) => {
  if(currentUser.currentContent == null)
    return
  const file = `${__dirname}/public/files/${currentUser.currentContent.content_id}.txt`
  res.download(file)
  if(currentUser.user_id != null)
  {
    sql = `INSERT INTO Downloads (user_id, content_id) VALUES
          ('${currentUser.user_id}',
           ${currentUser.currentContent.content_id})`
    database.query(sql, function(err, results){
      console.log(err)
    })
  }
})
```

Figure 19: Server: inserting favorites/ downloads into their tables

### 3.2.10 Use Case: Log Out

Once user logout, we logout his/her account on firebase and clear current user info on session.

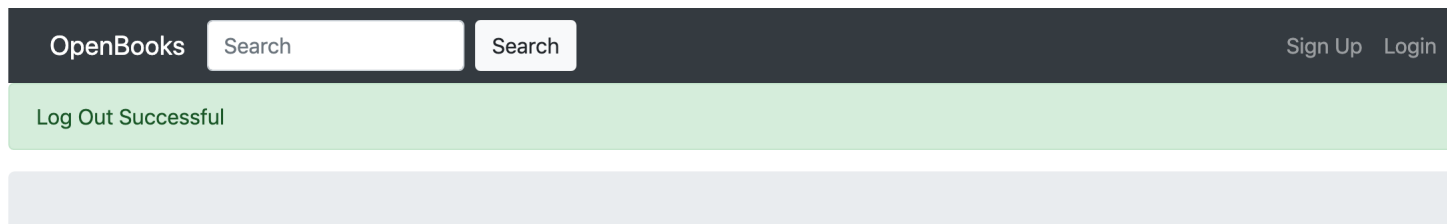


Figure 20: client's perspective: user logged out

```
currentUser.username = null
firebase.auth().signOut().then(function() {
  currentUser.username = null
  currentUser.email = null
  currentUser.user_type = null
  res.render('pages/index', {message: 'Log Out Successful'})
}, function(error) {
  // An error happened.
  console.log('faileu')
});
```

Figure 21: Server: user logging out — code



### 3.3 Project Setup/ Configuration

To set up the project, follow these steps:

1. Download + install MySQL
2. Download + install Node.js
3. Clone the repository <https://github.com/CS157A-Team2/CS157A-Team2>
4. Set up MySQL to allow remote access.
5. Set the MySQL root password to 'root'
6. cd to PATH/TO/PROJECT/src/model/ then login to MySQL command line using 'mysql -u root -p' then run the following commands:
  - (a) 'source init.sql'
  - (b) 'source loaddata.sql'
  - (c) confirm success by typing 'source queryAll.sql'
7. cd to PATH/TO/PROJECT/src/web-server/ then run 'node server.js'
8. go to your web browser (while the node process is running) and go to <http://localhost:8080>
9. email [brett.dispoto@sjsu.edu](mailto:brett.dispoto@sjsu.edu) if any issues arise

## 4 Project Conclusion

### 4.1 Lessons from Project — Teammate Statements

#### 4.1.1 Brett Dispoto

Prior to working on OpenBooks, I had no knowledge of practical database design/ application development. I now understand the following:

- The importance of an E/R design and analysis **prior** to implementing the relational schemas in the DBMS. Because our group put a large amount of work into our relational schema design, the normalization process into BCNF was extremely minimal.
- How to perform database integration on a virtualised 3-tier architecture. This includes
  - How to install and configure a local MySQL server, allowing remote access.
  - How to use Node.js to perform MySQL database queries, then parsing the JSON output of those queries.
  - How to install and configure the MySQL Node.js module.
- How to use embedded javascript (EJS) to display dynamically generated content using data from a MySQL database.

- How to write SQL (DDL/DML) scripts for automated database setup.
- How to effectively work as a team, where the responsibilities include writing functional specifications, designing database schemas, and implementing a database application.

#### 4.1.2 Feiyu Cai

Working on OpenBooks, gives me a opportunity that applying what I have learned about database design and management on real life application. In fact, there are some specific area I had practiced in this project:

- I have learned converting a E/R diagram of database to database table in practice. We spent 1/3 of the class time discussed about E/R diagram and converting it into table for these project. For the amount of table we used in this project, I had a great practice how the conversion works and have better understand of relational database
- I have learned normalization of database. I had used a lot of NoSQL database in the past because normalization was painful for me. Errors sometime were caused by deleting something in the database. In this project, I have a lot of practice about normalizing database into smaller table to avoid errors and bugs happen.
- This is my first time working with relational database with Node.js. Writing sql query with javascript syntax is first time for me.
- Writing sql file and importing it with sql CL tools is useful and convenience while inputing large amount of data.
- Learning to use MySQLWorkBench for virtualising database is useful when having multiple tables.
- Communicate with teammates and collaborate on Github makes working together easier.

#### 4.1.3 Adham Kamel

Before working on OpenBooks, I have had no database experience as well as minimal website creation experience. Working on this project has allowed me to gain the following skills:

- How to use embedded javascript (EJS) as the primary language for frontend design
- How to use 3-tier architecture with database systems, which includes:
  - How to link the MySQL database to a remote server
  - Performing MySQL operations in Node.js
- How to be an effective team leader by organizing team meetings and planning out what the plan of action was in meetings.

## 4.2 Future Improvements — Discussion

A *wishlist* of future improvements for this project would be as follows:

1. Deploying the database/ middleware on a remote server, eliminating the virtualised 3-tier architecture we are currently running on in favor of a proper client-server-database architecture. This would allow OpenBooks to be accessed on the web, rather than having to manually install, configure, and deploy on the client's local machine.
2. Adding advertisements — the data stored in our tables would be valuable to advertising firms running targeted advertisements. Information such as "*John Doe likes to read the new york times on Tuesday afternoons*" is very useful to such firms, and the revenue gained could be used to help OpenBooks achieve goal (1) above.
3. Add more tables to track user data such as average session time.
4. Add more features / customizability user profiles — such as profile photos, following users, view other user's profile.
5. Email notification system — When a new book is released by an author of which a user has favorited their book, the user should receive an email notification.
6. Look into the possibility of having tables be 4NF/5NF.
7. A terms of service, copyright licence, and privacy policy.