Mibrary

Technical Report

Group:
Edwin Xeon Gutierrez
Debarshi Kundu
Lucy Liu
Austin Mager
Diemi Pham
Aaron Saldana

CS 373
Summer 2018
Glenn Downing

Contents

1 Motivation 3
2 User Stories 3
3 RESTful API 4
3.1. Retrieve User 5
3.2. Retrieve Book 6
3.3. Retrieve Meeting 7
3.4. Retrieve Course 8
3.5. Retrieve Request 8
3.6. Retrieve Offer 9
3.7. Retrieve Report 10
4 Models 11
4.1 Book 11
4.2 Review 12
4.3 Meeting 12
4.4 User 12
4.5 Class 12
5 Tools 13
5.1 Postman 13
5.2 Bootstrap 13
5.3 Slack 13
5.4 Git 13
5.5 MySQL 14
5.6 Flask 14
5.7 React 14
5.7.1 React-Router 14
5.8 Docker 14
5.9 Mocha 14
5.10 unittest 14
5.11 Selenium 14
5.12 PlantUML 14
6 Hosting 15
6.1 Namecheap 15

6.2 AWS 15
8 Pagination 15
9 DB 15
11 Charts 17

1 Motivation

The motivation of the Mibrary project is to provide an avenue for students to donate or exchange their textbooks. This will help students who have a financial barrier to their education, and in turn be of community service.

2 User Stories

User Story 1: There are many textbooks provided by our traders. When a user needs to know what textbooks we offer, the application displays the list of the textbooks exist in the system. Assuming that the user does not give any filter criteria for listing the textbook, so all of the textbooks will be shown.

Estimate time: 1 hour

Actual time: 1 hour

User Story 2: When a user clicks on a textbook from the textbook list displayed on the screen, the system will show the user information of that textbook includes its ISBN number, title, author, edition, and cover photo.

Estimate time: 1 hour

Actual time: 1 hour

User Story 3: Users are able to know what classes they can take at UT and information about the classes such as the classes' numbers, titles, and required textbooks. The system will display the list of classes and users can get any class's information by clicking on the class from the displayed list. Assuming users do not give any filter criteria for listing available classes.

Estimate time: 1 hour

Actual time: 1 hour

User Story 4: Users are able to know who is trading (donating or requesting) textbooks across the system and which textbooks are corresponding to each trader. Assuming traders are human.

Estimate time: 1 hour

Actual time: 1 hour

User Story 5: Users are able to know the methods to contact traders via contact information that traders post in the system. Users can get any trader's contact information when they click on the trader from the displayed user list.

Estimate time: 1 hour

Actual time: 1 hour

3 RESTful API

The parameters of the API design requirement were vague and ambiguous at first. It was unclear if we were required to map the API calls that we were consuming to build our database, or the ones that will ultimately be provided to the front end and other users of Mibrary, or both. Once the requirements were cleared up, this part became relatively simple.

The Mibrary API is a RESTful API which can be used by developers to retrieve information about the books which are requested, offered, or traded by the users of Mibrary as well as information about the trades between the users.

Details of Mibrary RESTfull API can be found in this documentation.

Endpoints

The API is hosted under https://mibrary.me/api

Endpoint	Description
/users	Get all users in the system
/user/<username>	Get information of a user given the username
/books	Get all books in the system
/book/<isbn>	Get a book detail given an ISBN
/book/<query>	Get books that match the given query
/reviews/<isbn>	Get all reviews of a book given its ISBN
/meeting/<meeting_id>	Get a meeting detail given the meeting's id
/course/<course_number>	Get information of a course given its course number
/courses/<institution>	Get all course offered by an institution given institution code
/requests	Get all book requests by the users in the system
/request/<isbn>	Get a request detail given ISBN of the requested book

/offers	Get all book offers by the users in the system
/offer/<isbn>	Get an offer detail given ISBN of the offered book
/reports	Get all book reports by the users in the system
/report/<isbn>	Get a report detail given ISBN of the reported book

Status codes and Error messages

In the JSON response, the field status indicates the status of the reponse as well as error messages which can be “OK” if everything is fine or “INVALID_ID” if the given id was not found in the database.

3.1. Retrieve User

Get list of users

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/users
```

Response format

Returns a JSON file with a list of all users; each user will have the information shown in the table below.

Get details of a user

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/users
```

Response format

Returns a JSON file with information of a user.

Field	Description	Datatype
id	User ID	integer
username	Username of the user	string
email	Email of the user	string

3.2. Retrieve Book

Get list of books

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/books
```

Response format

Return a JSON file with a list of book; each book will have information as shown in the table below.

Get details of a book

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/book/9781934759486
```

Response format

Return a JSON file with information of a book as following.

Field	Description	Datatype

isbn	ISBN number of the book	string
publisher	Name of book publisher	string
identifier	Type of ISBN	string
classification	System of organizing book	string
title	Name of the book	string
subtitle	Accompanying Title Name	string
url	Online location of the book	string
pages	Number of pages	string
subjects	Genre or topic of the book EG:Nonfiction, Art, etc…	string
date published	Date of publication	date
excerpts	Samples of writing from the book	string
author	The first writer of the book	string

3.3. Retrieve Meeting

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/meeting/3
```

Response format

Return a JSON file information of a meeting specified by the meeting ID.

Field	**Description**	**datatype**
meeting_id	ID of the meeting	integer
time	Date and time of the meeting	string
location	Location of the meeting	string

3.4. Retrieve Course

Get list of course

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/courses/utaustin
```

Response format

Return a JSON file with a list of courses specified by the institution; each request has information as shown in the table below.

Get details of a course

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/course/cs373
```

Response format

Return a JSON file with information of a course given its course name as shown below.

Field	**Description**	**Datatype**
Course ID	ID of Course	integer
Course #	Number of Course	string
Course Name	Name of Course	string
Institution ID	ID of the Institution	integer

3.5. Retrieve Request

Get list of requests

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/requests
```

Response format

Return a JSON file a list of request; each request has the following information as shown in the table below.

Get details of a request

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/request/9781934759486
```

Response format

Return a JSON file information of a request as shown below.

Field	**Description**	**datatype**
date	Date of request	date
book_id	ISBN of the requested book	string
user_id	ID of user who requests the book	integer
meeting_id	ID of the meeting	integer

3.6. Retrieve Offer

Get list of offers

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/offers
```

Response format

Return a JSON file a list of offer; each offer has the following information as shown in the table below.

Get details of an offer

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/offer/9781934759486
```

Response format

Return a JSON file with an offer with following information as shown in the table below.

Field	**Description**	**datatype**
date	Date of offer	date
book_id	ISBN of the offered book	string
user_id	ID of user who offers the book	integer
meeting_id	ID of the meeting	integer

3.7. Retrieve Report

Get list of reports

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/reports
```

Response format

Return a JSON file a list of report; each report has the following information as shown in the table below.

Get details of a report

Example request:

```
curl --request GET \
  --url https://mibrary.me/api/report/9781934759486
```

Response format

Return a JSON file with a report with following information as shown in the table below.

Field	**Description**	**datatype**
date	Date of report	date
book_id	ISBN of the reported book	string
user_id	ID of user who reports the book	integer
meeting_id	ID of the meeting	integer

4 Models

4.1 Book

Attribute	***Description***
isbn	ISBN number of the book
publisher	Publisher of the book.
identifiers	Identifiers for the book.

classification	Classification of the book.
links	Links to the book.
title	Title of the book.
subtitle	Subtitle of the book.
url	url of the book.
pages	Number of pages.
subject	Subject of the book,
publish_date	Date the book was published.
excerpt	Excerpt from the book.
image	Image of the cover of the book.
author	Author of the book.
reviews	Reviews of the book.

4.2 Review

Attribute	***Description***
content	Content of the review.
user	User writing the review.

4.3 Meeting

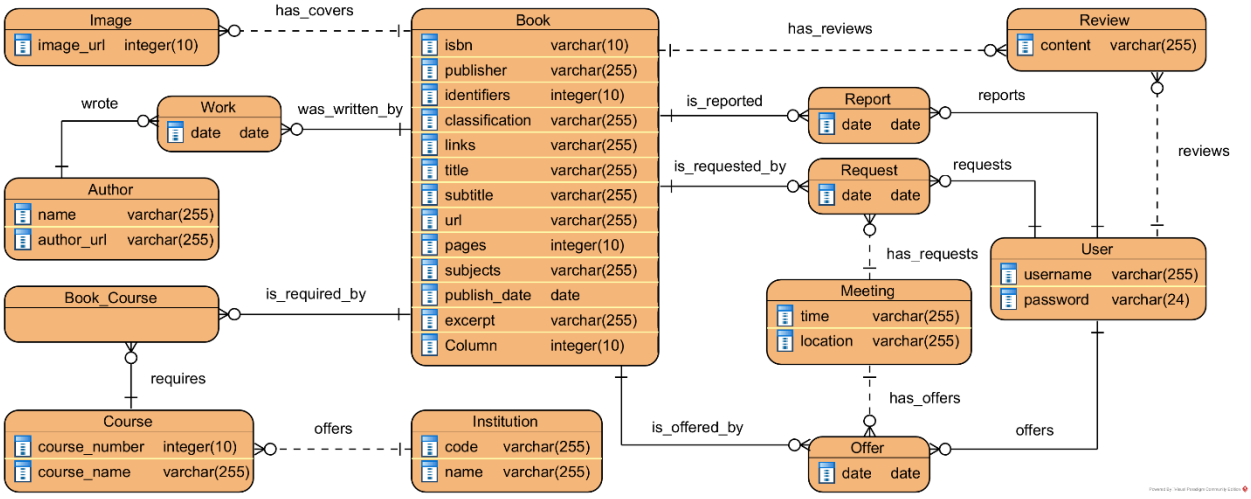
Attribute	***Description***
time	Time of the meeting
location	Location of the meeting
users	Users meeting
books	Book(s) being exchanged

4.4 User

Attribute	***Description***
name	Name of the user
meetings	Meetings the user is attending
reviews	Reviews the user has made
books	Books the user has requested

4.5 Class

Attribute	***Description***
name	Course name
number	Course number
books	List of required textbooks
uni	University name



5 Tools

5.1 Postman

Postman is the tool used to create documentation for the Mibrary API. Although Postman was a complicated tool in the beginning, its usefulness became apparent after working with it for longer. Postman allows us to easily document a restful API through creating and categorizing requests. These requests include the type, URL, description including sample usage, and parameters. A sample request can also be created and shown in a JSON or XML format.

5.2 Bootstrap

Bootstrap is the JavaScript and CSS framework that we used to enhance the user experience of our website. In order to implement Bootstrap, we integrated small pieces of scripting code into our html files. This code provides the link to a “CDN,” which is essentially a server with Bootstrap installed. Applying Bootstrap to all of our web pages ensured that the fonts were aesthetically pleasing and easy-to-read. Furthermore, we were better able to organize our information.

5.3 Slack

Slack has proven essential for communication. Our entire team is available on a single slack channel, connected to our phones and computers, allowing us to be in constant communication. Through this mechanism, we are able to communicate, meet up, ask clarifying questions, and coordinate the shared assignment and division of tasks.

5.4 Git

Git has been an invaluable tool to allow multiple contributions to this project. We have set up a shared Mibrary repo with CI. We commit to the repo often so we can always be working with the latest code. This allows all of us to update the same website codebase frequently, whenever individually we see a change that should be made.

5.5 MySQL

We used a MySQL database to store the data on the various models that are scraped from our API. MySQL commands allow our API to quickly and easily extract data from our database. A more detailed example of our models is found in the Models section.

5.6 Flask

Flask was the base for our backend, using Python. With Flask we spun up the backend of the site that connected to the database and exposed the Mibrary API for the React frontend to consume.

5.7 React

React is the JavaScript framework that we used for the frontend of the website. There was an initial time to learn how to use React, but after a while the site began to come together. After a few hours, the initial site was ported to React. React contains multiple components, allowing different pages and pieces of the website to be reused and displayed conveniently.

5.7.1 React-Router

React-Router is the library for React that allows linking to different pages. The site must be under a <BrowserRotuer> tag, and should be loaded through a Component that has a series of Switch tags to load different pages depending on the path being linked to by Link tags in other pages. Once installed, and after looking through a tutorial, this library was relatively easy to understand and use.

5.8 Docker

We used Docker to create images for the front and back end of the website. This allowed us to all have a common version and set of tools for the website, and allowing it to run consistently both on the Amazon hosting, and part of the GitLab CI.

5.9 Mocha

Mocha was the testing library used to test our frontend JavaScript code. Although it was rather confusing to set up, eventually tests were created and integrated with the GitLab CI.

5.10 unittest

Unittest is the Python unit testing framework to test the backend code. These tests will also be integrated with the GitLab CI.

5.11 Selenium

Selenium was used to test the GUI and interface. These tests were acceptance tests to verify that the site does what the users want. These weren’t programmatic tests of individual program functions.

5.12 PlantUML

PlantUML was used to generate and plan the features and characteristics of the data that is stored in the backend database.

6 Hosting

6.1 Namecheap

Our domain name was acquired from Namecheap. To use this domain with Amazon hosting, we used Namecheap's functionality to change the CNAME record to map our domain to the domain provided to us by AWS.

6.2 AWS

To host our static website, we used Amazon's AWS S3 service, a cloud-based static content storage service. There was some initial confusion regarding how to navigate Amazon's AWS service, but following the step by step process found online here was very helpful. After registering an account with AWS, the S3 console can be found at https://console.aws.amazon.com/s3/. We created buckets for both the mibrary.me and www.mibrary.me domains, and uploaded the same files to both. When uploading files, all the permissions must be set to public readable by everyone. Finally, to enable website hosting, static website hosting must be enabled in the properties of each bucket.

8 Pagination

Pagination was relatively simple to implement through the use of the React framework. Each page has a state that can be updated in different functions, and the variables included in the state can be rendered as part of the HTML rendered in the render call. Our state for the model pages included a list: models, an int: current page, and an int: pageModelCount. When the page number was clicked, we updated the currentPage count. pageModelCount was the amount of model links that would be displayed on each page. When the page itself was loaded, we would get the list of models from the Mibrary API and give those to the page state as the models list.

9 DB

The UML diagram shown below describes how the tables are laid out in our database.

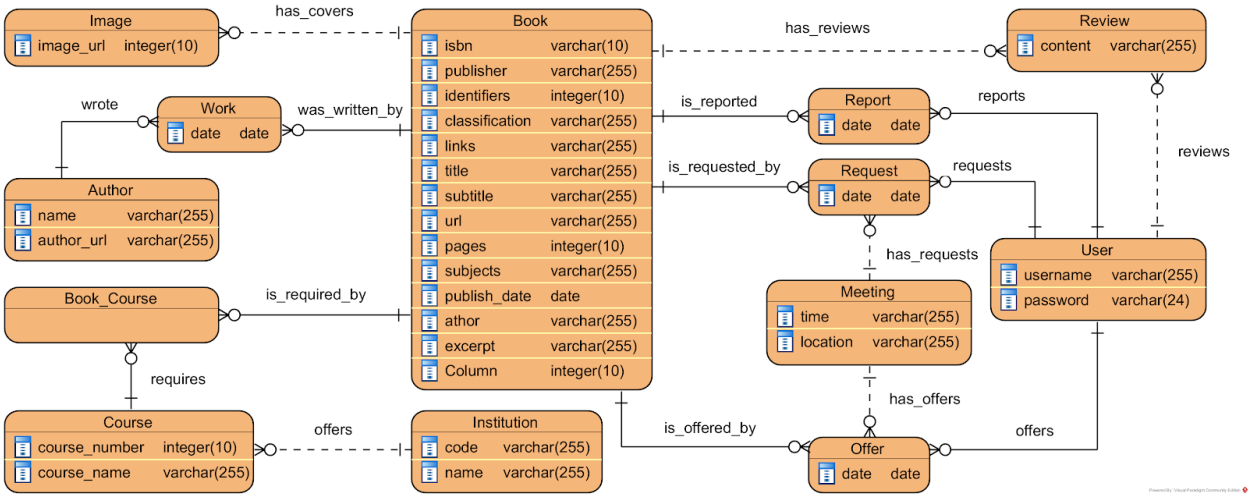


Figure: Mibrary UML model

Our database is formed by eight main models such as Author, Book, Course, Image, Institution, Meeting, Review, and User. Those models contain information that will be displayed on the model pages of the website and hold relationships to other models. Besides those eight models, our database also includes five other models that are Course_Book, Offer, Report, and Work. These five models are association classes which describe the many-to-many relationships between the eight main models.

Association classes explanations:

- Course_Book: Each book can be required by different courses and vise versa, each course is able to require more than one book.
- Offer: An offer is a connection between a meeting, a user, and a book. Indeed, a meeting consists of two users and books; each offer gives information about a book and the user who offers the book as well as the meeting that is involved
- Report:
- Request: A request, like an offer, is a connection between a meeting, a user, and a book. Indeed, a meeting consists of two users and books; each request gives information about a book and the user who requests the book as well as the meeting that is involved.

Work: In fact, a book can be a collaborated work of many authors and an author writes many different books. A work implies the relationship between an author and a book; that means this specific book was written by this author and this author wrote this book.10 Testing

We used four different testing mechanisms to test our website. To unit test our API we used Postman, as well as testing the results from the frontend. To unit test our backend we used the unittest framework for Python. This involved testing our API scraping calls and our own API, as well as accessing the database. To unit test the frontend we used mocha with JavaScript. We

tested our calls to our API and our loading of data from that API on the frontend. Lastly we made acceptance tests for our GUI using Selenium. These test the general GUI that the user will interact with.

11 Charts