上 海 纽 约 大 学
NYU SHANGHAI

CSCI-410 Project: TEXbook Documentation

Bilal Munawar, Yuechuan Zhang, Jorge Barreno
GitHub Repo: /Robertation256/TEXbook

December 17, 2020

Contents

1 Introduction

College education does not go without books. Too often, we find our textbooks and reference books come at a price too high. In addition, students are asked to purchase new hard copies of textbooks. This is an environmentally unfriendly practice, which significantly increases the overall release of carbon in our environment through production of new books. This platform encourages college students within their community to sell and rent used hard copies to fellow students. By doing so, we do not only help students reduce the costs of the course material but also promote an environmentally friendly and sustainable practice.

The project's minimum viable product is a web application where buyers and sellers posts textbook requests and listings respectively. The ultimate goal of the platform is to simply connect students interested in purchasing or selling a textbook. Students are allowed to register through their university's affiliated email address. The list of textbooks allowed to sell or request on the website is regulated through integrating the core curriculum of a university's courses.

2 Adopted Process

2.1 The Agile Process

Our team adopted the Agile method for fast, iterative development of the project. Unlike the Waterfall model, the development and testing activities are concurrent in the Agile model. In particular, we used the agile framework called Scrum, which focuses on short term iterations called sprints. Since all team members were located in different time zones, daily stand up meetings were replace by real time chatting on Whatsapp, where we discussed and shared progress on sprint tasks. Apart from that, we collaborated on joint working documents for user stories and APIs.

2.2 Iterative Development

One crucial feature of the Agile Process is the fast development iterations, or in the case of Scrum Method, sprints. Our team would meet on each Thursday to showcase sprint deliveries and do sprint planning. On Tuesdays, the team meets again to discuss software designs and major implementation challenges. However due to personal reasons and schedule problems, there were time when all team members could not meet to define sprint goals. As a result, our team had to merge several sprints into a big iteration.

2.3 Refactoring

During our implementation, refactoring is introduced from two major aspects: renaming and decoupling. Renaming is introduced for code cohesiveness and interpretability. For example, our unit test test cases are named by adding the "test" prefix to the tested function name. We also follow the Single Responsibility Principle in our design, which means that sometimes, we would refactor large units by separating them into smaller independent units. All refactoring modifications are introduced during code review phase after a teammate makes a commit.

2.4 Testing

During fast iterative development, we follow the principle that everyone should test their own code to ensure minimum functionalities. We use python unittest module to generate automated test cases for main functionalities of our application. In addition to test functions, these test scripts have `SetUpClass()` and `TearDownClass()` methods which create and remove dummy data after testing.

3 Requirements and Specifications

3.1 Project Technology Stack

Here's our detailed technology stack:

1. Languages
   - Python
   - JavaScript
2. Frameworks
   - Flask
3. Database
   - MySQL
   - Redis

3.2 Stakeholders

Stakeholders represents the individuals who will use, monitor and maintain the TEXbook platform. In particular, stakeholders include the following individuals:

1. **Users**: These are individuals within a college community who will interact with each other on the TEXbook website in order to sell, rent or purchase textbooks. These individuals include the following:
   - Students: Students within a college community will purchase, sell and rent textbooks from students tailored to their curriculum.
   - Faculty members: Faculty members can purchase, rent or recommend textbooks of their courses to fellow students or other faculty members
   - Alumnus
2. **Development Team**: The development team includes individuals who are responsible to ensure efficient operations of the website. The team include the following individuals:

(a) Engineers: The engineers are responsible to ensure the efficient executions of the system's functionality. This includes the Textbook search engine, user login interface and database integration. The Engineers also monitor the student activity and ensure cyber-security protocols are not violated.

(b) Textbook team: This team updates the textbooks for each registered college community annually. This ensures that the most up to date titles are sold, purchase or rented on the website.

3.3 Use Cases

1. **User register:** a user is defined as a person who possess affiliation to a university, which is verified through a verification email sent to their respective email addresses. After verification, the user sets a password for their respective accounts and finish the registration process.

2. **User login:** for login, email and password are required. Five failed attempts in login should result in another email verification process with the previously registered email.

3. **User Profile**: users are asked to fill in information about their name, grade, major, location and contact details. Users can select their avatar from a given set of options.

4. **A Textbook Collection**: The listings of textbooks for each university are exclusively updated each year. This platform should support a search on the latest textbook curriculum from a school's website. Users should be able to view all the related buyer requests and seller listings after clicking on a specific textbook.

5. **Publishing Listings and Requests:** Users should be able to post requests on intended textbooks from the platform's collection, offer a price, and indicate whether to borrow or buy. Similarly, users can publish listings on the textbooks they intend to sell, upload pictures of their items, indicate book condition and mark a price. The items requested or sold should come from system collection. Once a user selects the book intended to sell of buy from a search bracket drop-down, the corresponding book information is displayed.

6. **Unlock Contact Information from Listing and Requests:** Every user is given five free unlock chances per day. When a use is interested in knowing the contact information of the owner of a listing/request, he or she may choose to use up one of the five free chances to unlock that information.

7. **Managing Listing and Requests:** At any time, users may choose to permanently delete a listing or request. He or she may also temporarily remove a listing from the shelf, which would result in the listing being removed from search results. A user can also put an off-shelf listing back to shelf. User would be able to see the unlocked listings/requests and their corresponding contact information.

8. **Notifications:** a real time user notification system is required where users are sent notifications in the following two instances: (a) one of user's requested listing has now been made available through a fellow student and (b) a student unlocked the user's contact information on a buyer or seller post. Users can choose to delete or mark notifications as unread and subscribe to email notifications.

4 Architecture and Design

4.1 General Architecture

As a web application, our project structure is very straight forward. There are three major classes in our project: classes that inherit from `BaseResource` mainly consists of APIs; classes that inherit from `BaseService` will be handling logic, computation and database modifications by utilizing our model class. And finally, we have classes that inherits from `BaseModel` that serves as our database models. As shown in Fig2, `ImageResource` handles API requests by calling its corresponding `ImageService`, which then executes database updates and information retrieval from Image, which is the database model. Apart from that, we have

1. **Crontab:** Classes inherited from `BaseCronTab` are tasks that will be executed regularly. They are registered by `BackgroundScheduler`.

2. **Event:** Two main events include publication of a listing and unlock seller information. The events are managed by an `EventManager` class.

3. **EventHandler:** Event handler handles a specific range of events with respect to their responsibility. We have an event handler designated for each event.

4.2 MVC Structure

We follow the MVC structure, an framework developed by Microsoft that implements model-view-controller pattern. Our MVC Structure is organised as follows:

1. **Models**: The Database mapping objects are placed under */models. The models are defined for `Textbook, User, Profile, Listing, Notification and image.` The models for each of these core component defines its data schema, including some foreign key constraints.

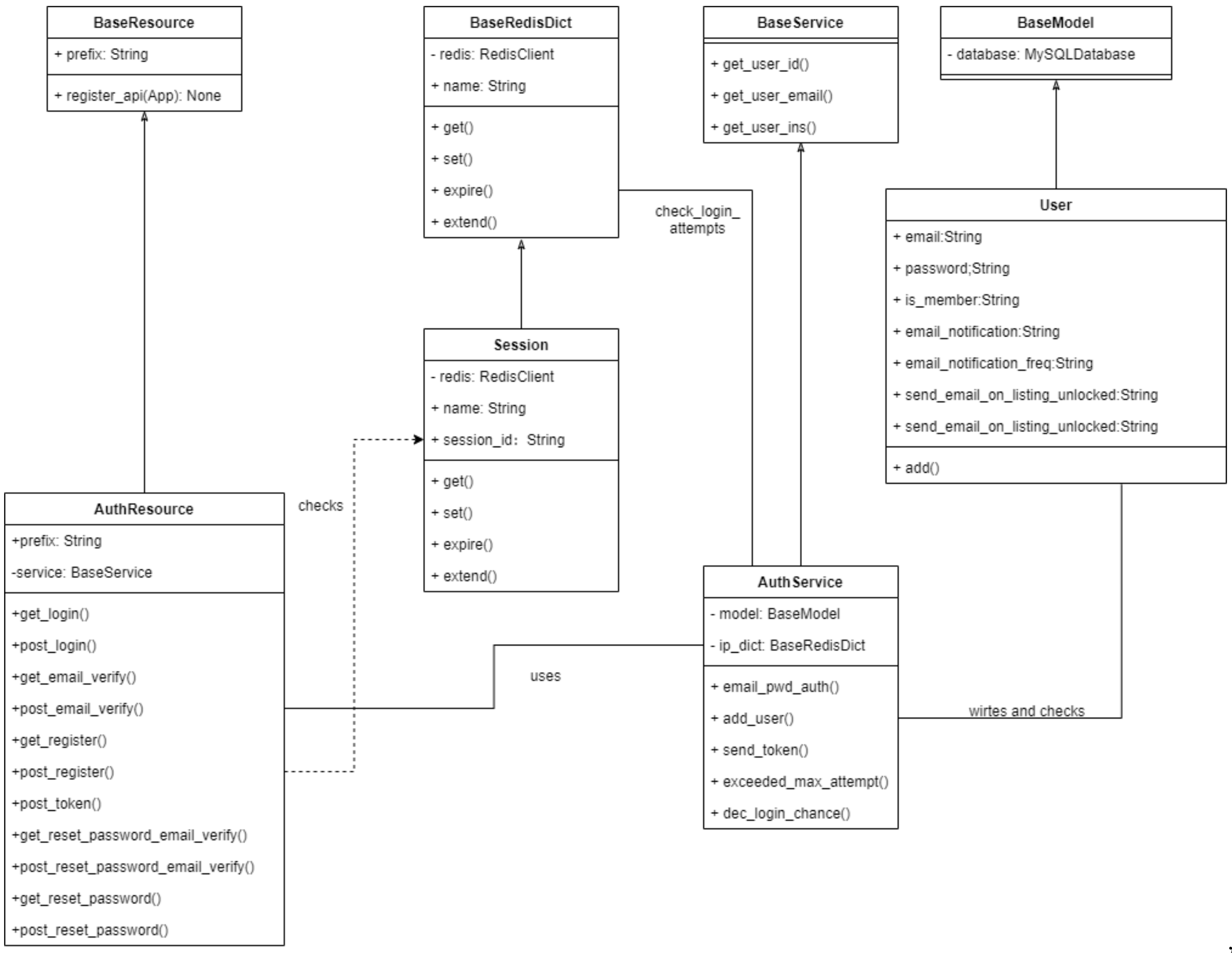
2. **Views**: HTML templates powered by Jinja2 engines are placed under /templates. We use JQuery to dynamically render page for textbook search results.

3. **Controllers**: Our API Handlers post requests by calling Services, which handle the logic and computation behind each helper function. The API directory of each class defines the routes prefix. The first split of method name defines ["GET", "POST"] methods, whereas the rest of its name defines the endpoint.

We have three main components of our working directory:

1. **Addons**: This includes major business objects and future addons. These addons are seen as reusable components, such as user, profile, textbook, listings, notification and image.
2. **Base**: This includes all the base classes which a set of defined classes inherit from.
3. **Common**: This includes shared models(mapping models for intermediate tables), services (email subscription) and handlers, including event handlers

4.3 Class Diagrams



,

Figure 1: Authentication Module Class Diagram

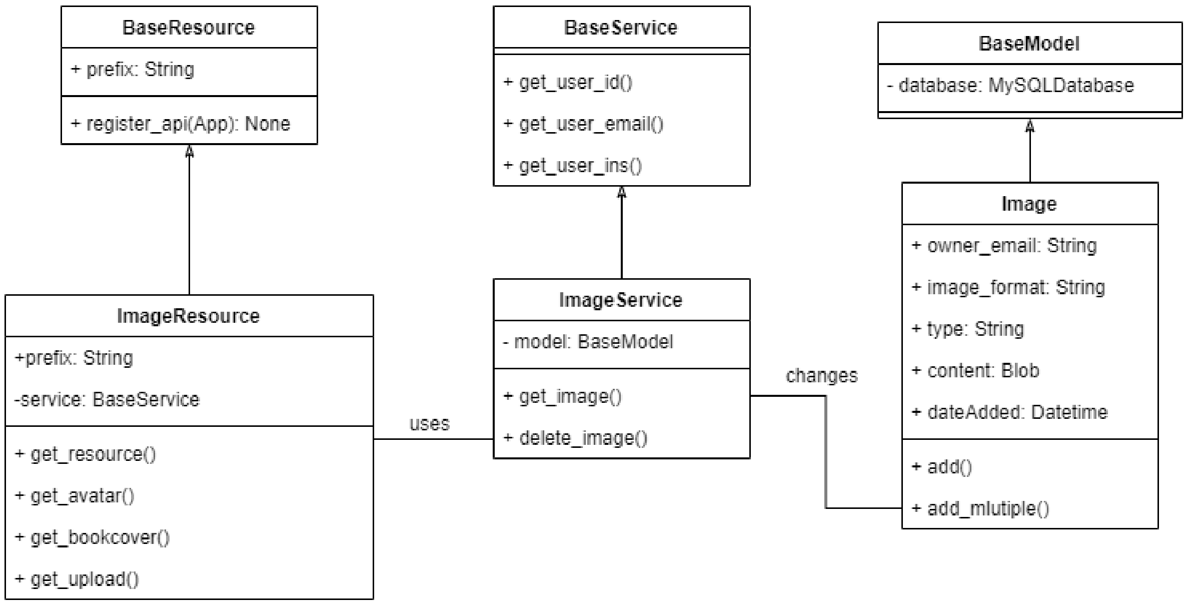
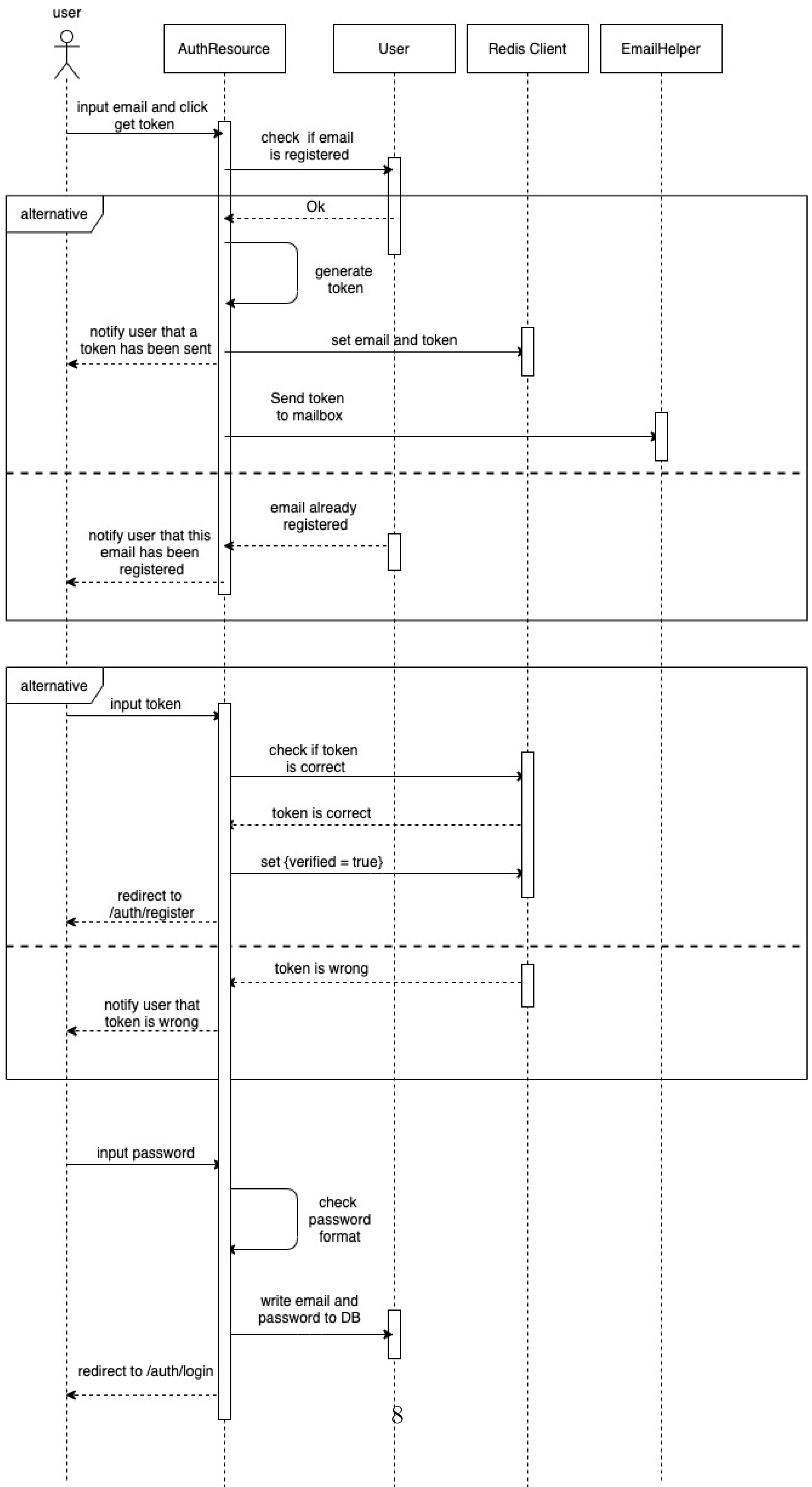


Figure 2: Image Module Class Diagram

4.4 Sequence Diagrams

User Sign Up



4.5 Constraints

Flask is light-weighted web framework that provides powerful features such as RESTful request dispatching and Jinja2 Template rendering. However due to Python's Global Interpreter Lock, only one thread is executed at a time, which severely hinders the concurrency performance within the system. For tasks that requires concurrency, we will be incorporating other powerful Python packages such as Gevent.

4.6 Adopted Design Pattern

We follow a Singleton Design pattern. This ensures that our python module, including Redis connection (a python module-level singleton) only instantiate once during a process. In addition, we use the Observer mode to ensure that events, such as the publication of a listing or request are dispatched by the `EventManager` to their corresponding handler.

4.7 Asynchronous Task Management

One core aspect of our application is asynchronous task management email service and notifications. The application does not hang for the User if an email is being generated or a notification is being issued. The following figure demonstrates an implementation of asynchronous email service.

```python
from base import base_cron_tab
import datetime

class EmailTask(base_cron_tab.BaseCronTab):
    def __init__(self, data):
        super().__init__()
        self.__data = data

    def job(self, data):
        from common.service.email_service import EmailHelper
        email_helper = EmailHelper("")
        for d in data:
            email_helper.receiver = d["address"]
            email_helper.send_email(subject=d["subject"], content=d["content"])

    def schedule(self):
        from utils.scheduler import scheduler
        start_time_bench_mark = datetime.datetime.now()+datetime.timedelta(minutes=1)
        for i in range(0, len(self.__data),10):
            start_time = start_time_bench_mark+datetime.timedelta(seconds=i*30)
            batch_data = self.__data[i:i+10]
            scheduler.add_job(self.job, "date", run_date=start_time,args=[batch_data])
```

Sample Code 1: Asynchronous email service

5 Reflections

Throughout this project, we got a better understanding of the communication and technology protocols in a professional environment. In particular, we learned how different components of a full-stack program are essential to produce a minimum viable product. We learned the importance of risk management since one of our former team members had to abandon the project. While each of the three team members specialized in back end development, we faced some challenges on UI/UX design. We have realized the importance of a front end developer to enhance the user experience. A crucial learning outcome is defining the user stories and following an architecture protocol in order to ensure smooth integration of new code. For example, we divided each core component into model, service and api modules. This practice helped us avoid writing dead code. We have also realized the importance of using a task manager, such as Azure in order to cohesively define user stories and monitor progress on assigned tasks. In addition, we have learned the importance of abiding by Git protocols, documenting the code and writing meaningful variables and class method names to allow for seamless integration of work with fellow developers.

Fortunately, we aim to deploy this project to be used by students in our community to sell and purchase textbooks. Since the product will be used on a full-scale university platform, we continue to actively monitor our future needs, such as privacy of our users and robust maintenance of the database and its servers.