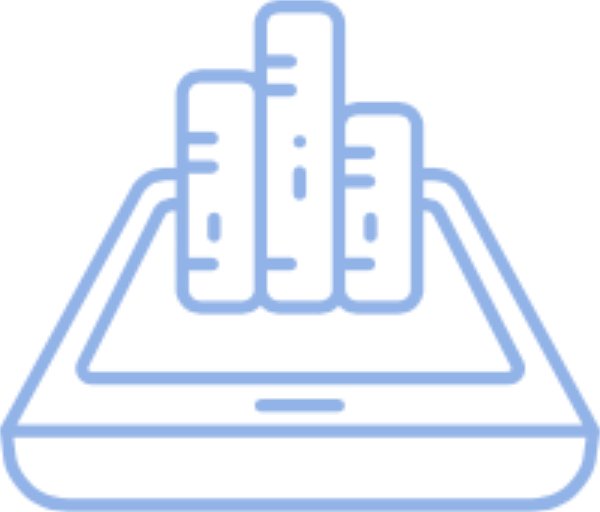
# CS3342 Software Design

Semester B 2018/19

Pocket Library™

one broke student at a time

Pocket Library™ Report



|  |  |  |
| --- | --- | --- |
| Group 9 Members | | |
| TANDON Rishabh | 54106672 | Project Manager |
| WOODHOUSE Olivia | 40115805 | Analyst |
| FENG Fanjuan | 54779590 | Designer |
| SABHARWAL Rohan | 54592018 | Programmer |
| DING Linli | 54382328 | Tester |
| ZHANG Xilin | 54780379 | Maintainer |

## 

## Table of Contents

[**CS3342 Software Design**](#_4ewsouesetz1) **0**

[Background & Objectives:](#_p6er8pohcgdh) 2

[Scope:](#_ofur44f6t9oj) 3

[Functional Requirements:](#_gdnw91damk5) 4

[Process Model:](#_91znm1tp3ujx) 4

[Use Cases:](#_fsjvof87ogc1) 5

[Registration Module:](#_sjqol5no6yp4) 5

[Login Module:](#_sfrpyox5o1wh) 9

[Exploration Module:](#_lt41s9g4t86y) 11

[Lending/Purchasing Module:](#_hjgroa1c0kut) 15

[Admin Module:](#_u4hqj9w6jqse) 22

[Non-Functional Requirements:](#_6gs9jf8hj32g) 28

[Class Diagrams:](#_mqrsfkwblrrf) 29

[Design Principles and Patterns:](#_t7rxnrezj5hj) 30

Liskov Substitution Principle: 30

Open Closed Principle: 30

Singleton: 31

State Pattern: 32

[Command Pattern:](#_ptcg7pgikgat) 33

[Sequence Diagrams:](#_2vl3qcj1n11n) 34

[Question Module:](#_ftjnl5747mbd) 34

[Buy/Sell/Borrow/Lend Module:](#_m93zw1lg7gtw) 35

[Subscription Module:](#_6mggw0o1olzu) 36

[Prototype:](#_or82xgvhogtr) 38

[Conclusion:](#_hpp6023627rg) 42

[Teamwork Timetable:](#_h1g8ym1o18gz) 43

## 

## Background & Objectives:

With about 1.2 trillion dollars in student loan debt affecting nearly 44 million borrowers, each of whom averaged an outstanding loan balance of $37,172 in the United States alone, it is safe to say that student debt is a global epidemic [[1]](#footnote-0). To make matters worse, university textbooks don’t help to ameliorate this problem in the slightest. Textbook prices have risen to an outrageous inflation in recent years. A study done approximated that the price of university textbooks rose by 82 percent between 2002 and 2013 [[2]](#footnote-1). This makes getting an education highly inaccessible for students who simply cannot afford these steep figures.

For this reason we have created Pocket Library™, a subscription-based platform for students to gain access to e-books, community, guidance, as well as a venue for buying/selling copies of used books. It is basically a one stop shop for students to manage all of their textbook needs. Not only does it help to alleviate some of the heavy costs that textbooks may require, but it also helps to mitigate some of the stress of being a student by allowing students to not only interact and help one another on our many discussion platforms but also to have access to guidance and answers from our verified tutors.

Our goal as a team of students is to make life a little easier for our peers and well as ourselves who are also struggling through the day-to-day obstacles of being university students. Based on a survey of more than 2,000 students from over 150 different campuses, 65 percent declared that they had decided against buying a textbook due to its price tag [[3]](#footnote-2). We are here to make sure no student goes without an adequate education due to the lack of resources that are accessible to them. It is time to make education accessible with Pocket Library™!

## Scope:

## Functional Requirements:

1. Registration Module
2. Login Module
3. Exploration Module
4. Lending/Purchasing Module
5. Admin Module

## Process Model:

**Prototype Model:**

Our product is heavily reliant on our customers because without widespread customer support and feedback, our product becomes invaluable. This is why we chose the Prototype Evolutionary Process Model as this allowed us to communicate, strategize, and survey our target market throughout the process of development.We developed a mock prototype to show the user and built our entire project around prototype refinements based on the responses we receive from both our internal development team and a few external users used for testing purposes.

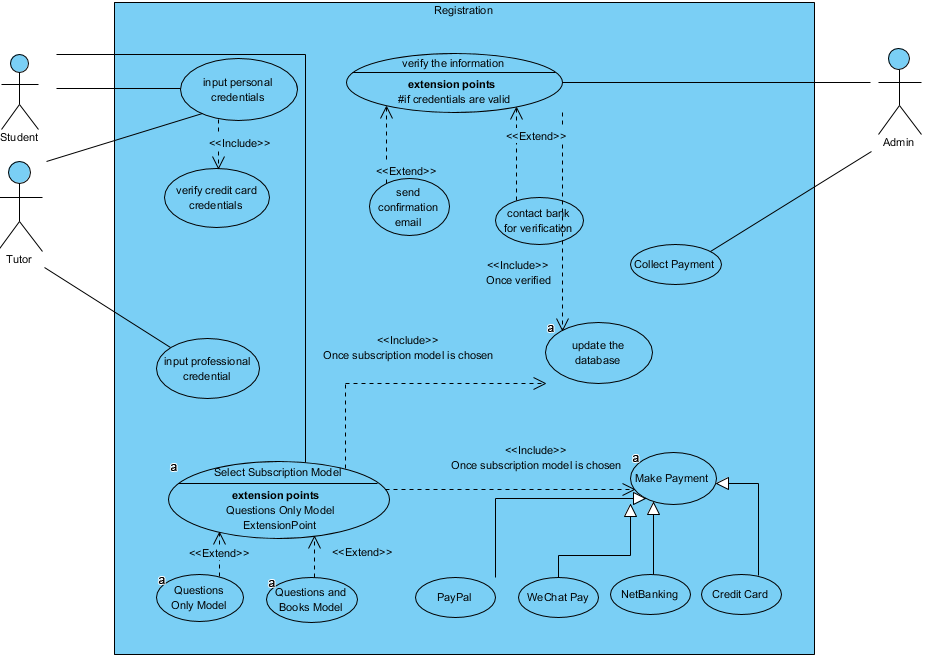
## 

## 

## 

## Use Cases:

### Registration Module:



The Registration Module is a use case diagram for a new user to register an account to access to this software application. The user needs to input their personal information as well as their credit card for this step.

The use case mainly involves three actors: **Students** who should input their personal credentials, credit card, select a subscription model and then pay for that model, **Tutors** who need to input professional credentialsand **Admin**, who is responsible for verifying the information.

There are various use cases for these three actors. **Input Personal Credentials** is the first step for **Students** to register a new account, which also includes the credit card information. Then students should **Select Subscription Model**, which has two extensions, one is **Choose Questions Only Model** and the other named **Choose Books and Questions Model**. After selecting step, students should pay for the model they have chosen. **Make Payment** generalises four payment approach, **PayPal**, **Wechat Pay**, **NetBanking** and **Credit Card**. After the payment is done, the **Admin** will **Collect Payment.**

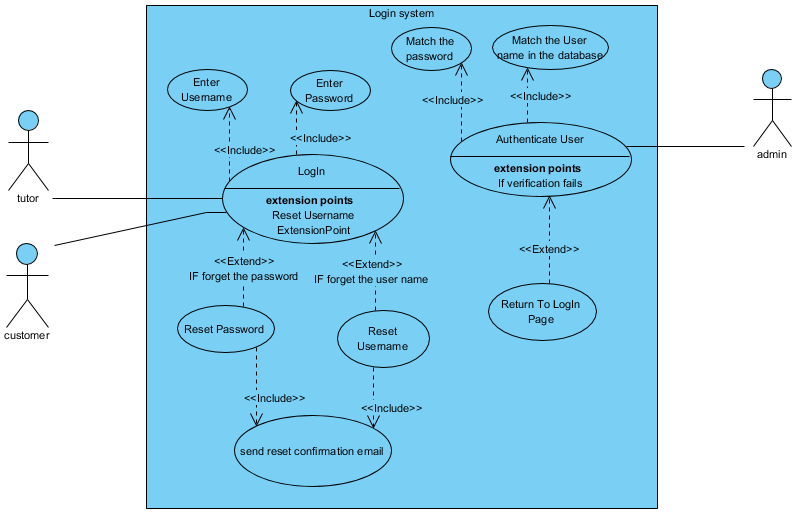
As for **Tutors**, they should first input their professional credentials as well as credit card information. After the user has submitted their credentials, **Admin** will begin to **Verify the Information**. If the personal/professional credentials are valid, the **Admin** will contact bank to verify the corresponding credit card information, which is named **Contact Bank for verification**. If all the credentials are valid, the **Admin** will **Send Confirmation Email** to the new users and **Update Database**.

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Input Personal Confidential | |
| **Actor(s):** | Student & Tutor | |
| **Description:** | This use case describes the process of a customer registering the account | |
| **Reference ID:** | RM 1.1 | |
| **Typical course of events:** | **Actor Action** | **System Response** |
|  | **Step 1:** the student /the tutor registers new accounts by inputting personal credentials.  **Step 2:**The student/ the tutor inputs his/her credit card credentials. | **Step 3:**The system stores the information and forward to the admin. |
| **Alternative course of events:** | N/A | |
| **Precondition:** | The student and the tutor must have access to a credit card. | |
| **Postcondition:** | Registration Module 1.2 is activated | |

|  |  |  |
| --- | --- | --- |
| **Actor(s):** | Admin | |
| **Description:** | This use case describes the process of a admin to verify the information of customers and tutors | |
| **Reference ID:** | RM 1.2 | |
| **Typical course of events:** | **Actor Action** | **System Response** |
|  | **Step 2:**The admin verifies the user credentials. | **Step 1:**The system is activated when the system provides the admin with the user credentials.  **Step 3:**Verify the user basic credentials of the students.  **Step 4:**Verify the professional credentials of the tutors. |
| **Alternative course of events:** | **Step 3a:** if the basic credentials are verified successfully, then the system will contact the bank for verification of the credit card.  **Step 3b:** if the credit card are verified successfully, then the admin will send a confirmation email to the student.  **Step 4a:** if the professional credentials are verified successfully, then the system will contact the bank for verification of the credit card.  **Step 4b:** if the credit card are verified successfully, then the admin will send a confirmation email to the tutors. | |
| **Precondition:** | The tutor and student have already input their personal information. | |
| **Postcondition:** | The student/tutor receives a confirmation email. | |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Select the subscribe model | |
| **Actor(s):** | Student | |
| **Description:** | This use case describes the process of a customer selecting the subscribe model | |
| **Reference ID:** | RM 1.3 | |
| **Typical course of events:** | **Actor Action** | **System Response** |
|  | **Step 2:**The student selects Questions and Books subscription | **Step 1:**The system shows two models for the subscribe model.  **Step 3:**The system stores the selected information of the student to the database.  **Step 4:**The students’ credit card is charged by the system as per the chosen subscription model. |
| **Alternative course of events:** | **Step 2a:**The student selects Questions only subscription model. | |
| **Precondition:** | The use case is activated once the student’s basic credentials are verified. | |
| **Postcondition:** | The customer successfully chooses one of two models. | |

### Login Module:



The LogIn Module is a use case for the user who have already possessed an account and hope to have access to the software.

The LogIn Module contains three actors **Tutor** & **Student** who have the same functionalities to log into his/her account and **Admin**, who is responsible to authenticate the user to allow them be exposed to the content in the software.

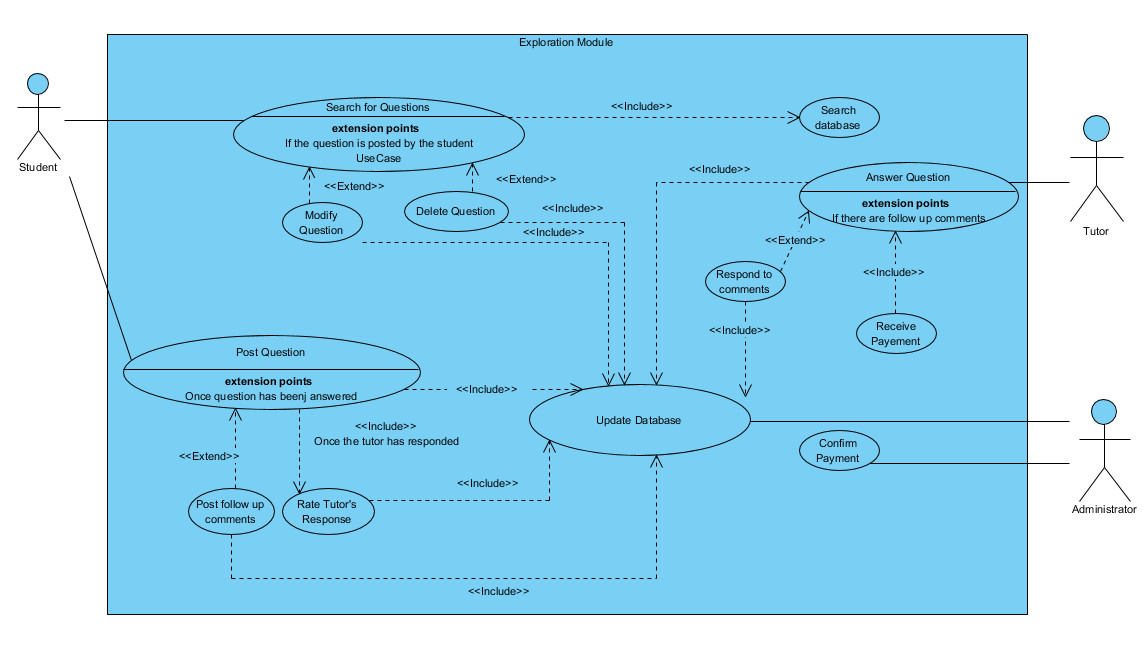
For **Tutor** & **Student**, **LogIn** is the use case to log into their accounts, which includes two functionalities named **Enter Username** and **Enter Password**.If the user forgets his username, he/she can choose to **Reset Username**, similarly, he/she can **Reset Password** if the user forgets the password. After resetting the username/password, a reset confirmation email will be sent, which is named **Send Confirmation email**.

**Admin** has the functionality named **Authenticate User**. First, admin will match the username in the database by invoking the use case named **Match the Username in Database**. If there exists this username, then compare the password with the record in the database, which is named by **Match the password**. If either username or password is incorrect, the use case named **Return to LogIn Page** will be invoked. And then it is up to **Tutor** & **Student**, retry, **Reset Username** or **Reset** **Password**.

|  |  |  |
| --- | --- | --- |
| **Actor(s):** | Student & Tutor & Admin | |
| **Description:** | This use case describes the process of a member hoping to log in his/her account | |
| **Reference ID:** | LI 1.1 | |
| **Typical course of events:** | **Actor Action** | **System Response** |
|  | **Step 1:** The use case is initiated when a user, what is either a student or tutor, attempts to log into his/her account.  **Step 4:**The user enters the Username and the password.  **Step 7:** The user has been verified and successfully logged in. | **Step 2:**The system presents to ask the user to input Username.  **Step 3:**The system presents to ask the user to input respective password.  **Step 5:**The system verifies the user by searching the database for the username.  **Step 6:** The system then matches the username to the password in the database in order to complete the verification process. |
| **Alternative course of events:** | **Step 3a:** If the user forgets his username, they will be asked to reset their username and then the reset confirmation email will be sent, then back to **Step 1**.  **Step 3b:** If the user forgets his password, they will be asked to reset their password and then the reset confirmation email will be sent, then back to **Step 1**.  **Step 4a:**If either the username or the password user inputs are incorrect, the system will back to **Step 1**. The user can choose whether to input one more time or reset his password or Username. | |
| **Precondition:** | The user possesses a valid account. | |
| **Postcondition:** | The user logs into his/her account successfully. | |

### 

### Exploration Module:

The Exploration Module represents the use case that provides a student with the functionality of searching for and posting a question and/or an assignment on to Pocket Library. Since this functionality in its entirety, is only accessible to students with a subscription module, therefore registered users would have to login to their accounts before they have are allowed to utilize this feature.

The Exploration Module involves three actors: **Students** who post/modify/delete questions and follow up comments, **Tutors,** who respond to questions and comments that have been posted and the **Administrator**,who validates payments to tutors once they provide answers to questions.

There are various use cases under the Exploration Module that various actors invoke. **Search for Questions** is a use case that the **Student** calls when they are looking for a question on the system, following which the **Search database** use case is invoked which searches the question database and displays the search results to the user. Additionally, if the **Student** posted the question, he/she then has the option of editing or deleting the question by invoking the **Modify Question** and **Delete Question** use cases respectively. All the changes that the **Student** chooses to make is updated on the question database accordingly and is done through the invocation of the **Update Database** use case, which is managed by the **Administrator.**

Under this module, a **Student** can also call the **Post Question** use case if he/she wants to post queries or assignments that they would like answered. This is where the **Tutor** would answer the query posted on to the system through the **Answer Questions** case. Once the question is answered by the tutor, the **Student** then has the option of asking follow up questions through the **Post follow up comments** use case, to which, the **Tutor** would be required to respond in order to receive their payment from the System **Administrator**. This is done by the **Tutor** through the **Respond to comments** use case. Since Pocket Library takes pride in the reliability of its system, once a question has been answered by, a **Student** would have to **Rate the Tutor’s Response**, so as to ensure the focus of the system is still maintained. All questions, follow up comments, responses and ratings that are posted on to the system are updated on database through the invocation of the **Update Database,** which as previously mentioned, the system **Administrator** regulates.

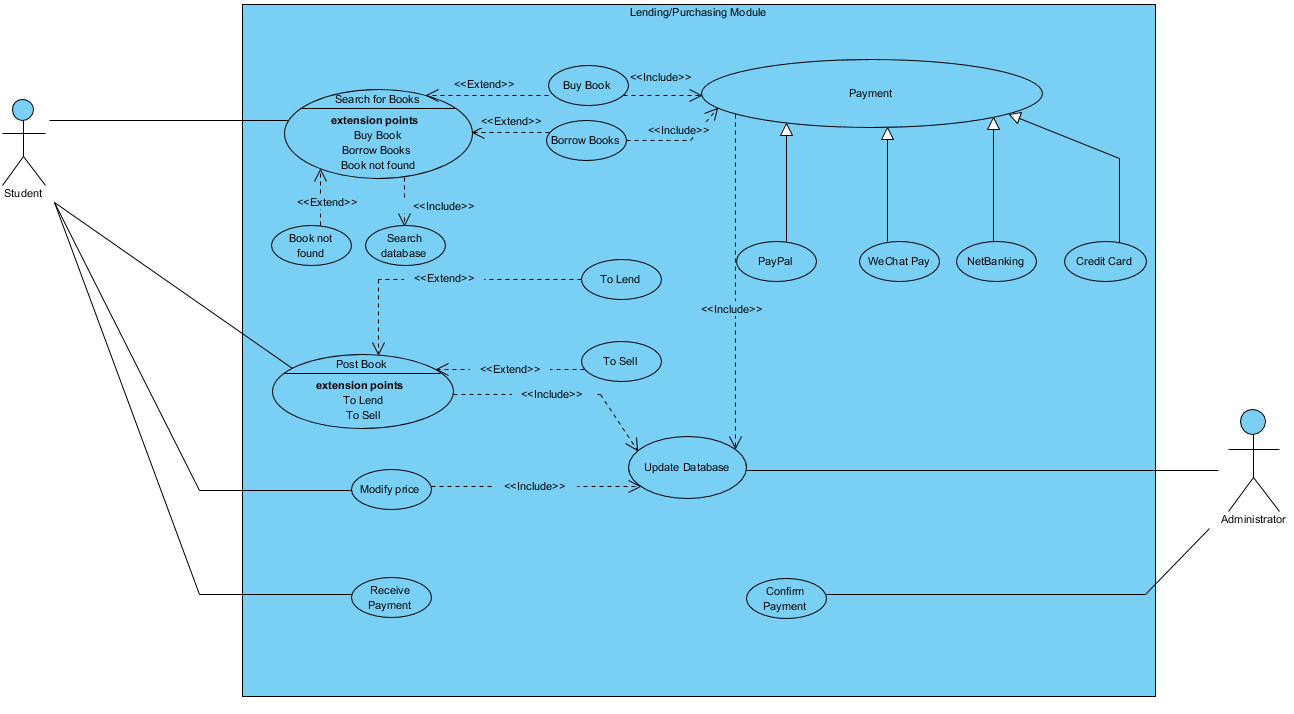
All the **Tutors** on the system are paid every time they respond to/solve a question. The **Administrator** decides and settles this amount by means of the **Confirm Payment** use case. The **Tutor** collects this payment by through the invocation of the **Receive Payment** use case.

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Searching for Questions. | |
| **Actor(s):** | Student, Tutor & Administrator | |
| **Description:** | This use case describes the process of a student searching for and posting questions and assignments on the Pocket Library platform. | |
| **Reference ID:** | EM1.0 | |
| **Typical course of events:** | Actor Action | System Response |
|  | **Step 1:** This use case is initiated when a student searches for a particular question on the Pocket Library platform  **Step 3**: The student can now view the searched question and the relevant discussion under said question. | **Step 2:** The system processes the student’s input and searches on the database for the relevant question. Once a match is found, it is displayed by the system to the user. |
| **Alternative course of events:** | **Step 2a:** If no match is found on the database, the student will be notified by the system that his search results returned no match and he would be prompted to search for another question.  **Step 3a:** The student can also choose to modify the question if he/she posted on the platform.  **Step 3b:** The student can additionally choose to delete the question if he/she posted on the platform.  **Step 3c:** The system processes the student’s request of modification or deletion and stores the relevant changes on the database. | |
| **Pre-condition:** | The student needs to login or register before they are allowed to use the search, modify or delete a question. | |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Posting Questions | |
| **Actor(s):** | Student, Tutor & Administrator | |
| **Description:** | This use case describes the process of a student searching for and posting questions and assignments on the Pocket Library platform. | |
| **Reference ID:** | EM2.0 | |
| **Typical course of events:** | Actor Action | System Response |
|  | **Step 1:** This use case is initiated when a student posts a question on the platform.  **Step 3**: On receiving the notice from the system, the tutor is promoted to answer the question.      **Step 5**: The answer to the question is now made available to the student. The student will rate the tutor.    **Step 7**: On being prompted by the system, the administrator confirms the amount to be paid to the tutor.    **Step 9**: The tutor can receive the payment amount after being notified by the system. | **Step 2:** The system processes the student’s input and records the question onto the database. Following this, the system sends a notice to the tutor to answer the question.    **Step 4**: The system records the tutor’s answer on to the database and notifies the student that his/her question has been answered. The system also notifies the administrator that the tutor is to received payment for his answer.    **Step 6**: The rating provided by the student is recorded by the system and stored in the database.      **Step 8**: The system takes note of this amount and notifies the tutor that they can receive the payment amount that has been confirmed by the administrator. |
| **Alternative course of events:** | **Step 5a:** The student can also post follow up comments to the answer made by the tutor.  **Step 6a:** The system makes note of the follow up questions asked by the student and records it in the database. The tutor is then notified by the system to respond to the comments.  **Step 7a:** The tutor answers the comments received by the student. | |
| **Pre-condition:** | The student needs to login or register before they are allowed to post or answer questions. | |

### 

### Lending/Purchasing Module:

The Lending/Purchasing Module represents the use case that provides a student with the functionality of buying/borrowing a book or selling/lending a book via the Pocket Library platform. Since this functionality, is only accessible to students with a valid subscription, only registered users who have logged in are allowed to utilize this feature.

The Lending/Purchasing Module involves two actors: Students who can choose to buy/borrow a book on the Pocket Library platform or sell/lend out an existing book they own on the Pocket Library platform. The Administrator is responsible for updating the database once depending on if a book should be added or removed and is also responsible for confirming the payment once a transaction has been made.

Several use case can be invoked by actors in the Lending/Purchasing module. **Search for Books** is a use case that the Student calls when they are looking for a book on the system, following which the **Search database** use case is invoked which searches for the book on the database and displays the search results to the user. If the book that the Student enters is not found, then the **Book not found** use case is invoked that returns a message to the Student stating the book has not been found. After the list of books is displayed to the Student, he/she then can choose whether they want to buy/borrow a particular book. If the Student wishes to buy a particular book then the **Buy Book** use case will be invoked following which the **Payment** use case is invoked wherein the user can pay for his/her purchase via a selection of payment methods all generalized to four use cases namely **PayPal, WeChat Pay, NetBanking and Credit Card**. If the Student wishes to borrow a particular book then the **Borrow Book** use case will be invoked following which the **Payment** use case is invoked wherein the user can pay for his/her purchase via a selection of payment methods all generalized to four use cases namely **PayPal, WeChat Pay, NetBanking and Credit Card.** The Student can also choose to sell/lend out a book via the **Post Book** use case – if the Student wishes to lend out the book then the **Lend Book** use case will be invoked following the **Post Book** use case or if the Student wishes to sell the book then the **Sell Book** use case will be invoked following the **Post Book** use case. Once a student posts book either to sell/lend out, the database is updated via the **Update Database** use case which is managed by the Administrator actor. The Student can also choose to modify the price of the book posted either to sell/lend out via the **Modify Price** use case. Finally, the Students on the system are paid once a book is sold/lent out. The Administrator settles this amount by means of the **Confirm Payment** use case. The Student collect their payment through the invocation of the **Receive Payment** use case.

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Borrow Books | |
| **Actor(s):** | Student & Administrator | |
| **Description:** | This use case describes the process of a student borrowing books on the Pocket Library platform. | |
| **Reference ID:** | Student Module 1.2 | |
| **Typical course of events:** | Actor Action | System Response |
|  | **Step 1:** A student searches for a particular book on the Pocket Library platform      **Step 3**: The student processes the choices displayed by the system and decides which book he/she wants to borrow from the system. Multiple selections are allowed.    **Step 5**: The student can choose one of four different payment methods to complete the transaction.    **Step 6:** The administrator verifies if the payment has been completed and updates the database accordingly by marking the book as ‘Borrowed until dd/mm/yyyy’    **Step 7:** If the payment is successful then the student receives the selected book(s). | **Step 2:** The system processes the student’s input and searches on the database for the relevant book. The matches are displayed to the user            **Step 4**: The system processes the student’s input and proceeds to the payment screen |
| **Alternative course of events:** | **Step 2a:** If no matches are found then it returns a ‘No match found’ message  **Step 6a:** If the payment fails then the user returns back to the ‘Payment’ screen  **Step 7a:** No books are received if the ‘Payment’ is unsuccessful and the database remains the same | |
| **Pre-condition:** | Student needs to add one or more books to the basket to proceed to the payment screen. | |
| **Post-condition:** | The student receives the purchased book if it is borrowed. | |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Buy Books | |
| **Actor(s):** | Student & Administrator | |
| **Description:** | This use case describes the process of a student buying books on the Pocket Library platform. | |
| **Reference ID:** | Student Module 1.1 | |
| **Typical course of events:** | Actor Action | System Response |
|  | **Step 1:** A student searches for a particular book on the Pocket Library platform      **Step 3**: The student processes the choices displayed by the system and decides which book he/she wants to buy from the system. Multiple selections are allowed.    **Step 5**: The student can choose one of four different payment methods to complete the transaction.    **Step 6:** The administrator verifies if the payment has been completed and updates the database accordingly by removing the number of copies bought from the database.    **Step 7:** If the payment is successful then the student receives the selected book(s). | **Step 2:** The system processes the student’s input and searches on the database for the relevant book. The matches are displayed to the user            **Step 4**: The system processes the student’s input and proceeds to the payment screen |
| **Alternative course of events:** | **Step 2a:** If no matches are found then it returns a ‘No match found’ message  **Step 6a:** If the payment fails then the user returns back to the ‘Payment’ screen  **Step 7a:** No books are received if the ‘Payment’ is unsuccessful and the database remains the same | |
| **Pre-condition:** | Student needs to add one or more books to the basket to proceed to the payment screen. | |
| **Post-condition:** | The student receives the purchased book if it is bought. | |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Sell Books | |
| **Actor(s):** | Student & Administrator | |
| **Description:** | This use case describes the process of a student selling books on the Pocket Library platform. | |
| **Reference ID:** | Student Module 1.3 | |
| **Typical course of events:** | Actor Action | System Response |
|  | **Step 1:** The student can also post a book to sell on the Pocket Library system  **Step 2:** The administrator adds the book that the student wants to sell onto the database  **Step 4:** The student can choose to modify the price of the book    **Step 5:** If the book is sold to another student, then the current student receives the payment.    **Step 6:** The administrator confirms if the payment is successful. | **Step 3:** The book(s) to be sold is added to the database            **Step 7:** The sold book(s) is removed from the database |
| **Alternative course of events:** | **Step 3a:** If the price of the book has been modified, the administrator updates the database to reflect the new price.  **Step 6a:** If the payment is unsuccessful, the book(s) and the money does not change hands. | |
| **Pre-condition:** | Student needs to own one or more books to sell to other students. | |
| **Post-condition:** | The student receives the appropriate dollar amount if his/her book is sold to another student. | |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Lend Books | |
| **Actor(s):** | Student & Administrator | |
| **Description:** | This use case describes the process of a student lending books on the Pocket Library platform. | |
| **Reference ID:** | Student Module 1.4 | |
| **Typical course of events:** | Actor Action | System Response |
|  | **Step 1:** The student can also post a book to lend on the Pocket Library system    **Step 2:** The administrator adds the book that the student wants to lend onto the database    **Step 4:** The student can choose to modify the price of the book    **Step 5:** If the price of the book has been modified, the administrator updates the database to reflect the new price.    **Step 6:** The administrator verifies if the payment is successful and updates the database accordingly by marking the book as ‘Lent until dd/mm/yyyy’ | **Step 3:** The book to be lent is added to the database              .      **Step 7:** The book is marked as ‘Lent until dd/mm/yyyy’ on the database. |
| **Alternative course of events:** | **Step 3a:** If the price of the book has been modified, the administrator updates the database to reflect the new price.  **Step 5a:** If the payment is unsuccessful, the book(s) and the money does not change hands. | |
| **Pre-condition:** | Student needs to own one or more books to lend to other students. | |
| **Post-condition:** | The student receives the appropriate dollar amount if his/her book is lent out to another student. | |

### **Admin Module:**

The Admin Module is a use case diagram for the representation of all of the responsibilities the admin must accomplish in order to keep the system well managed and organized. Due to the fact that this use case is a representation of the relationship between the admin and the system, the **admin** is the sole actor in this use case diagram.

There are six key use cases for the admin use case diagram: **Manage Customer Subscriptions**, **Manage Tutors**, **Manage Book Exchanges**, **Record Tutor Responses**, **Supervise Customer Discussion**, **Record Student Ratings of Tutor Responses**, and **Update Database**. All six use cases are described below with the use case specifications.

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Manage Customer Subscriptions | |
| **Actor(s):** | Admin | |
| **Description:** | This use case portrays the responsibilities of the admin when a new customer subscription is being registered. | |
| **Reference ID:** | Admin Module 1 | |
| **Typical Course of Events:** | Actor Action | System Response |
|  | **Step 2:** The admin verifies the customer’s implemented credentials.  **Step 4:** The admin verifies the customer’s implemented payment information. | **Step 1:** When a customer attempts to make a new subscription the admin is notified by the system.  **Step 3:** The payment method is provided by the system. |
| **Alternative Course of Events:** | **Step 2a:** If the customer’s credentials are either invalid or already being used for another subscription, then the subscription is cancelled.  **Step 4a:** If the customer’s payment information is not successfully verified, then the subscription is cancelled.  **Step 5:** If the customer’s credentials and payment method are successfully verified, then the admin adds the customer to the system’s database. | |
| **Pre-condition:** | The customer must request a new subscription. | |
| **Post-condition:** | The customer is added to the database (if all is successful) and thus the new account is created. | |

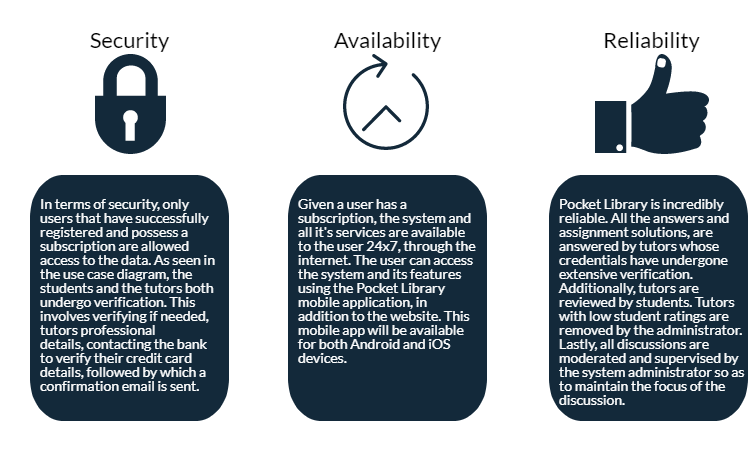
|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Manage Tutors | |
| **Actor(s):** | Admin | |
| **Description:** | This use case portrays the responsibilities of the admin when a new tutor is attempting to be registered with the system. | |
| **Reference ID:** | Admin Module 2 | |
| **Typical Course of Events:** | Actor Action | System Response |
|  | **Step 2:** The admin verifies the  tutor’s implemented credentials. | **Step 1**: When customers are attempting to implement a book exchange, the system notifies the admin.  **Step 3:** If the customer’s credit card is successfully verified, the system completes the transaction. |
| **Alternative Course of Events:** | **Step 2a:** If the customer’s credentials are not successfully verified, then the tutorial-status subscription is cancelled.  **Step 3**: If the tutors’s credentials are successfully verified, then the admin adds the customer to the system’s database. | |
| **Pre-condition:** | The tutor must request to be verified for a tutorial-status subscription. | |
| **Post-condition:** | The tutor is added to the database (if all is successful) and thus the new tutor account is created. | |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Record Tutor Responses | |
| **Actor(s):** | Admin | |
| **Description:** | This use case portrays the responsibilities of the admin when a tutor responds to a question within the system. | |
| **Reference ID:** | Admin Module 4 | |
| **Typical Course of Events:** | Actor Action | System Response |
|  | **Step 2:** The admin verifies the  tutor’s bank information. | **Step 1**: When a tutor responds to a inquiry, the system informs the admin.  **Step 3:** If the tutors’s bank information is successfully verified, the system compensates the tutor appropriately for their response. |
| **Alternative Course of Events:** | **Step 2a:** If the tutors’s bank information is not successfully verified, the admin notifies the tutor that the payment was unable to be made. | |
| **Pre-condition:** | A tutor must respond to an inquiry within the system. | |
| **Post-condition:** | The tutor is successfully compensated for their response (if all is successful). | |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Supervise Customer Discussion | |
| **Actor(s):** | Admin | |
| **Description:** | This use case portrays the responsibilities of the admin in the supervision of the customer discussion board. | |
| **Reference ID:** | Admin Module 5 | |
| **Typical Course of Events:** | Actor Action | System Response |
|  | **Step 2:** It is the admin’s  responsibility to glance over all the  customer discussion posts/be  vigilant for reported posts in order  to secure the safety and integrity of  the platform. | **Step 1:** Customers make posts on the respective platform. |
| **Alternative Course of Events:** | **Step 2a:** If the admin finds a post it deems inappropriate, the admin removes said post. | |
| **Pre-condition:** | Customers must post on the respective platform. | |
| **Post-condition:** | The admin creates an environment of safety and integrity for all of its customers. | |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Update Database | |
| **Actor(s):** | Admin | |
| **Description:** | This use case portrays the responsibilities of the admin in the process of updating the database. | |
| **Reference ID:** | Admin Module 7 | |
| **Typical Course of Events:** | Actor Action | System Response |
|  | **Step 1:** The admin must update the  system’s database on a regular  basis in order to keep the system  current.  **Step 3:** After each respective  process of verification, the admin  adds customers, tutors and books  to the database.  **Step 4:** The admin thus removes  customers, tutors, and books that  must be removed from the  database.  **Step 6:** The admin removes said  inappropriate posts/responses. | **Step 1:** Customers make ratings for tutor responses.  **Step 2:** Customers, tutors, and books request to be added to the system.  **Step 3:** For various reasons, customers, tutors, and books must be removed from the system.  **Step 5:** Inappropriate posts/responses are made. |
| **Alternative Course of Events:** | N/A | |
| **Pre-condition:** | The database must be updated on a regular basis. | |
| **Post-condition:** | The admin must uphold the responsibility of regularly updating the database in order to keep the system current. | |

## Non-Functional Requirements:



## 

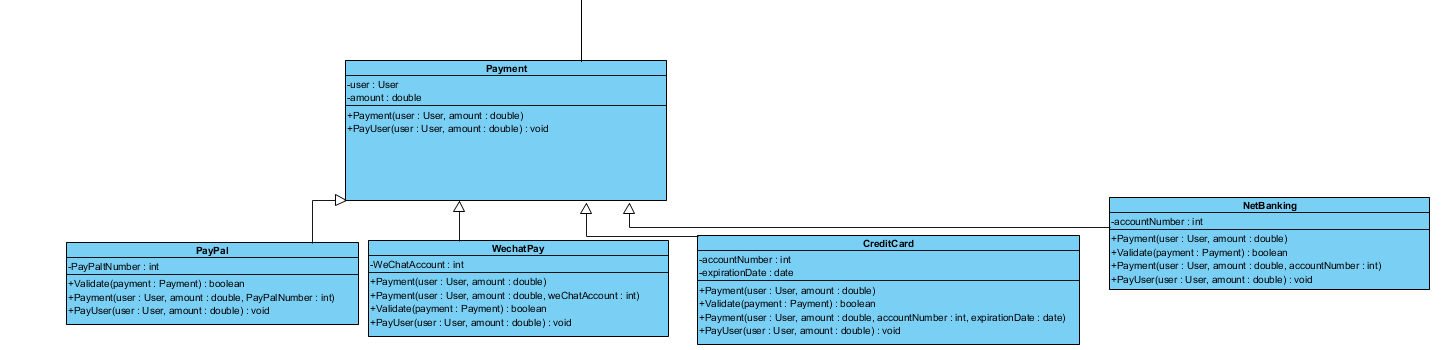
## Class Diagrams:

\*For detailed view please see ***Class Diagram\_Group9.jpg*** as attached in canvas submission.

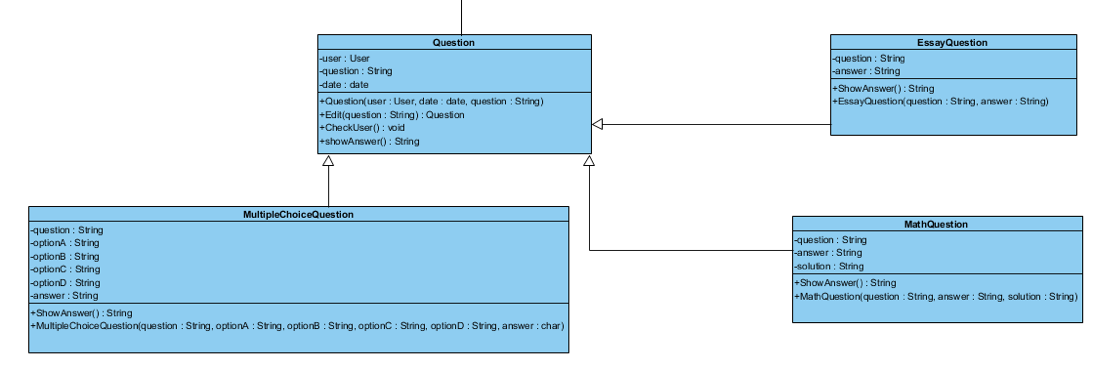
## Design Principles and Patterns:

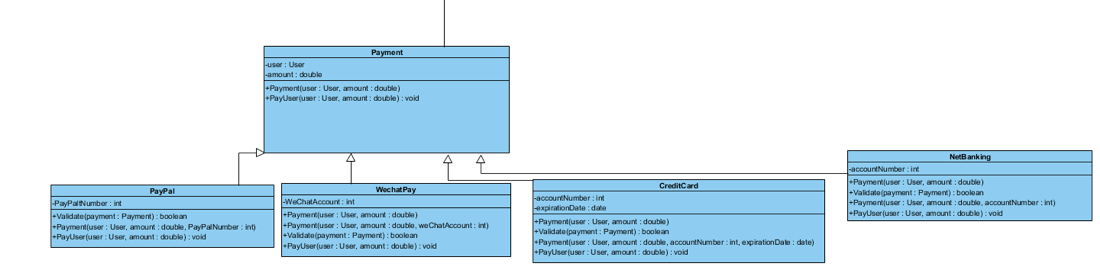
The design principles and design patterns are integrated with a purpose of adhering an efficient-working Pocket Library™ system with industry standard practice.

**Liskov Substitution Principle (LSP):**

LSP shows that the design of a subclass can be a substitute of its parent class. The subclasses should be able to implement all the functions and attributes of the superclass. In this project the Payment class has 4 subclass which can implement its constructor, Payment() and function, PayUser(). 

**Open Closed Principle (OCP):**

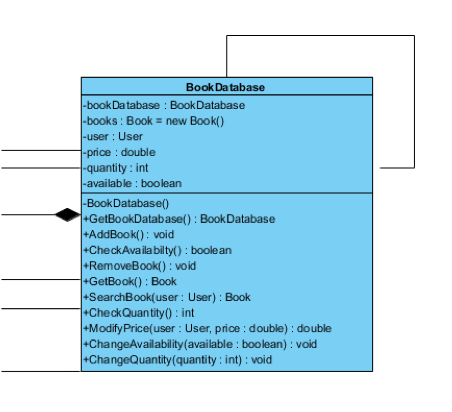
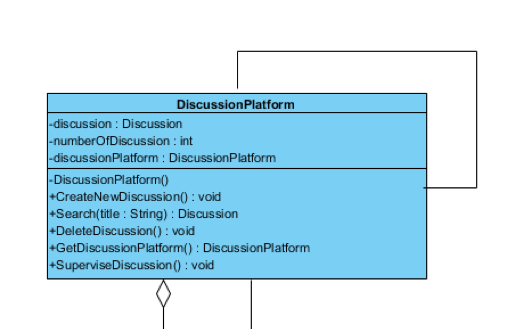




According to the Open Closed Principle (OCP), a software entity such as a class, module or even a function, should be implemented in such a way that it is open for extension however closed for modification. This means that the function behavior should allow for further extension without requiring a change in the core functionality.

In our project, the Question and Payment parent classes implement the Open Closed Principle. Both these parent classes contain sub classes, i.e., different question types and different payment options, that call the same function, showAnswer() and PayUser() respectively and therefore, can be called through polymorphism without any modification to the parent classes. Additionally, new Question and Payment subclasses can be added easily, implying the capability of extension of the parent class.

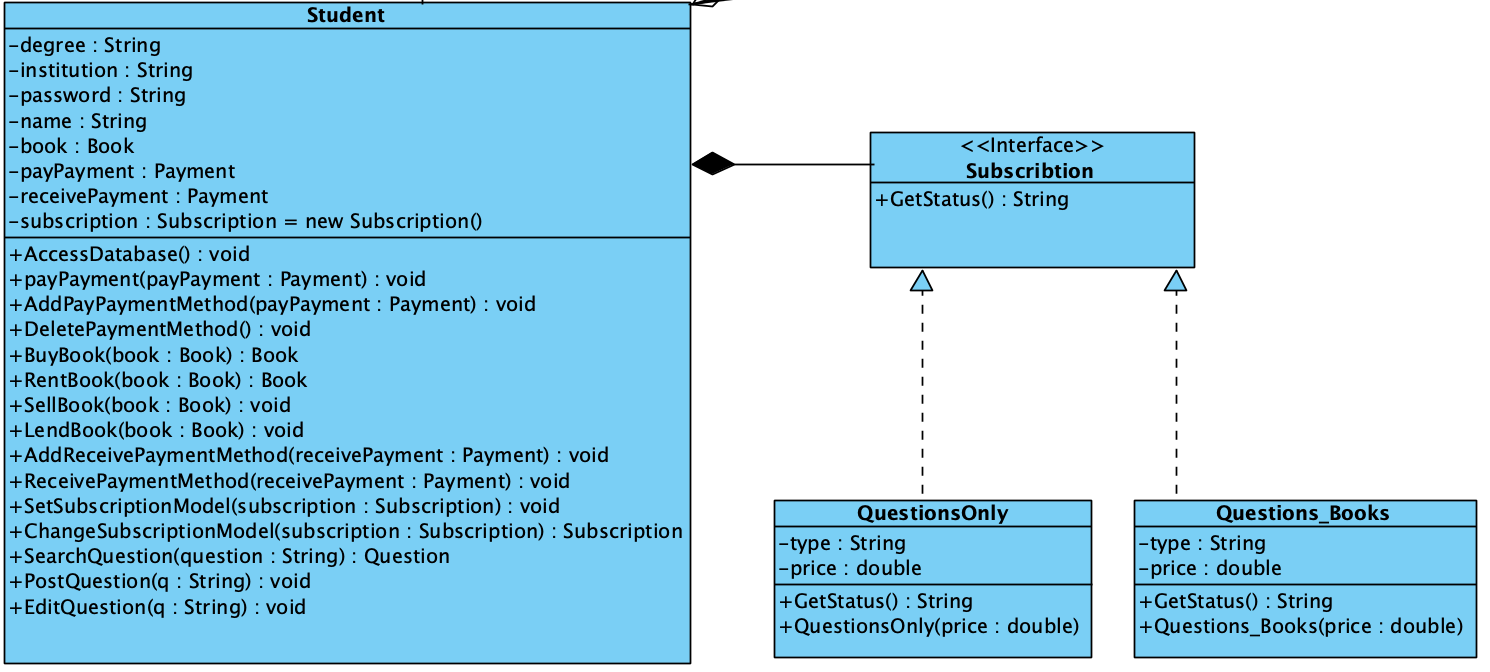
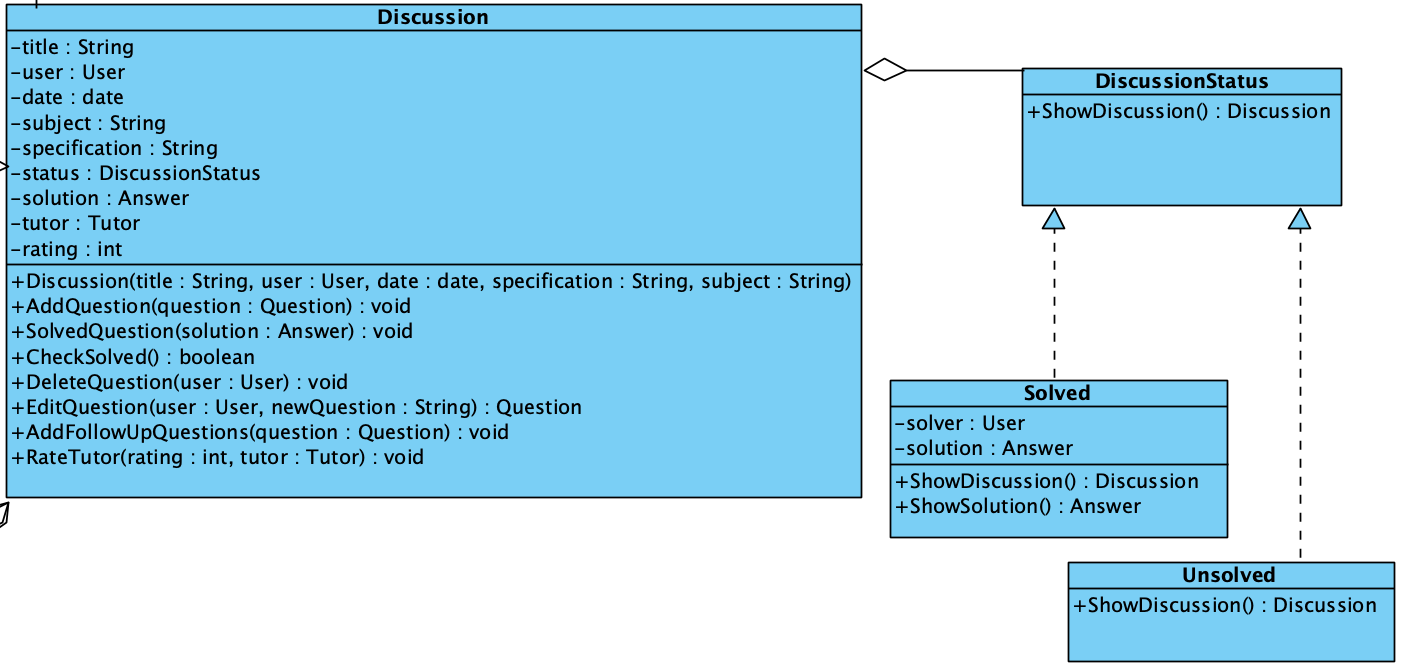
**Singleton:**



Our project utilizes the Singleton pattern for the above classes that require just a single instance of the class to exist at all time. Since the DiscussionPlatform class stores every discussion that is has occurred on the system, therefore it is required that we have a single instance that maintains this functionality. All the functions with regard to ccreation of a new discussion and deleting a discussion has to be regulated through this class, so as to safely store all discussion threads.

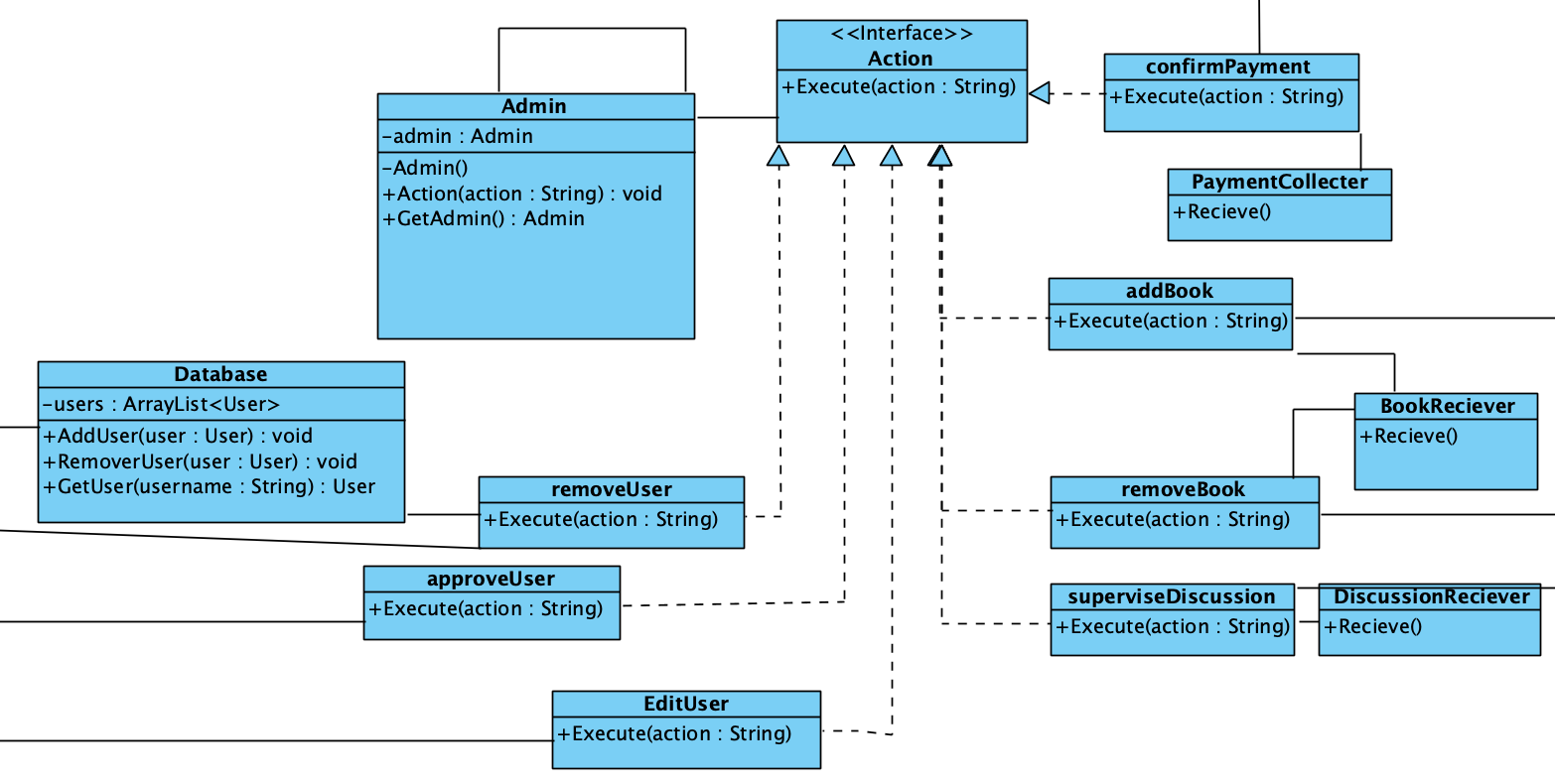
Similarly, since the BookDatabase class requires that a single instance of the class contains every book that is registered to the Pocket Library platform, therefore we utilize the Singleton design pattern. This ensures the reliability of information as all functions are implemented through this singular instance.

### State Pattern:

The DiscussionStatus, Solved and Unsolved apply the state pattern. The Subscription, QuestionOnly, and Question\_Books also apply the state pattern.

The reason of using the design pattern is that when the status of the Discussion object alters, the behavior of the object will also change. The pattern will allow run time alteration of the status instead of recompiling the whole project.

### Command Pattern:

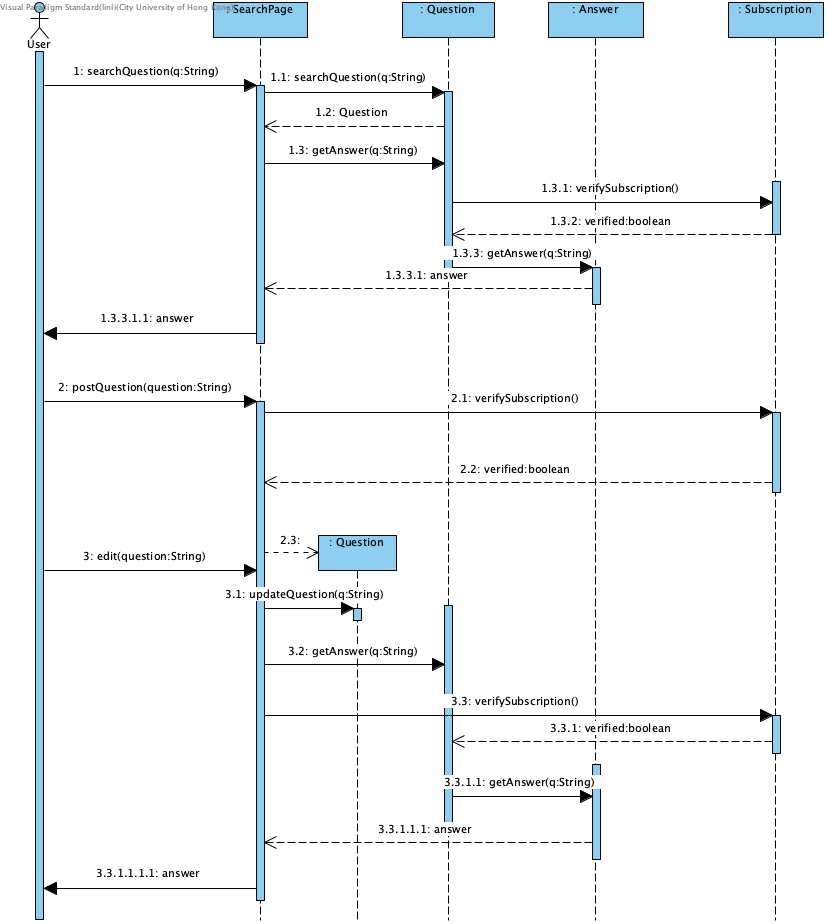
The command design pattern is a behavioral design pattern in which an object is used to represent and encapsulate all the information needed to call a method at a later time. In other words, there exists a Command object that encapsulates a request by exposing just one method execute() that causes some actions to be invoked on the receiver.

In our class diagram, we use the command design pattern to invoke the Admin class and all of its responsibilities. The Admin class invokes the command pattern using the Action object (which acts as our Command object in this case) to process commands based on the request type. This was an appropriate design pattern to use for this specific class as the admin has a plethora of actions to implement. This design allows for a loose coupling and an indirect control of all of the actions/commands the admin must complete.

## 

## Sequence Diagrams:

### Question Module:



The question module mainly deals with the student asking question process.

1.The student first search the question on the search page. The search page will input the keywords as a String.

1.1The SearchPage object will then call the function in the Question class.

1.2 The Question class will then return a question object to the search page.

1.3 The SearchPage then calls getAnswer of Question.

1.3.1 & 1.3.2 & 1.3.3 & 1.3.3.1 The answer object will be returned only after user’s subscription is verified.

1.3.3.1.1 The SearchPage then renders the answer object for viewer.

2. Student may also choose to post a question if there is no satisfactory result.

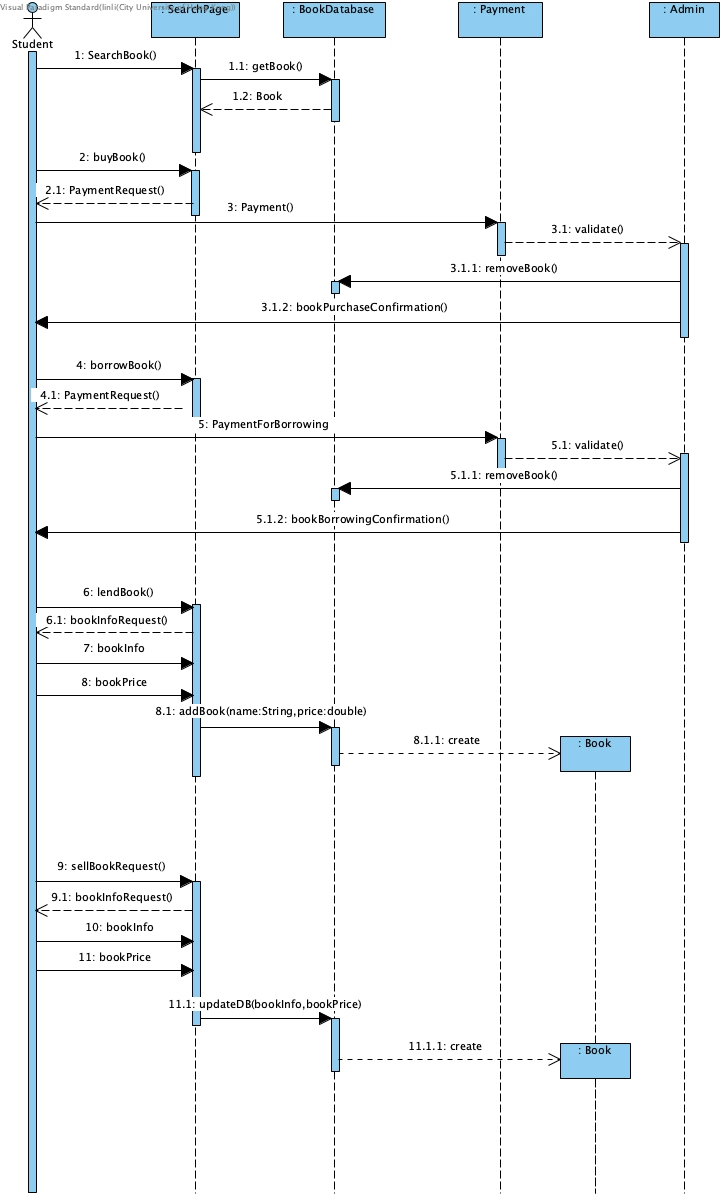
2.1 & 2.2 Post question function needs subscription verification.

2.3 Once verified, a new Question object will be created.

3. Further modifications can be made to the question owned by the user.

3.2 Once the question has an answer, the search page can access the answer by getAnswer function. \*The same procedures will be in place as 1.3 to 1.3.3.1.1.

### Buy/Sell/Borrow/Lend Module:



The question module mainly deals with the book transactions behavior, including selling, buying, borrowing, and lending books.

1. The user request for a book.

1.1 & 1.2 The SearchPage retrive book object from book database.

2. The user request for buying a book.

2.1 The SearchPage asks for payment

3 & 3.1 The payment will be handed in and validated by the system admin.

3.1.1 Admin then remove book from book database.

3.1.2 Admin send back book purchase confirmation information to the user.

\*Borrowing a book follows the same procedures as buying a book.

6. User request for lending a book.

6.1 SearchPage asks user to provide book information.

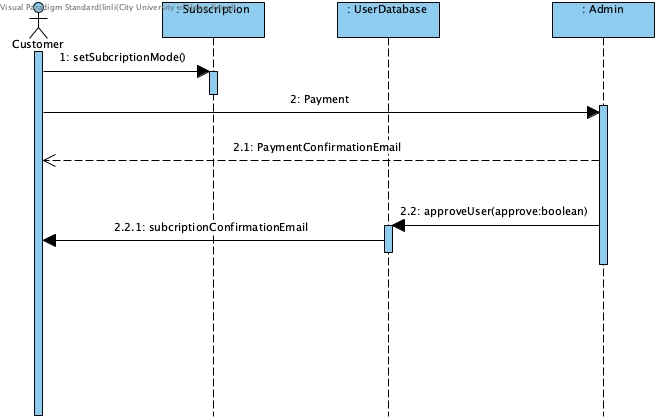
7&8 User gives book information and price.

8.1 SearchPage calls the function in BookDatabase to update the book information.

8.1.1 The function then creates a new Book object.

\*Selling a book follows the same procedures as lending a book.

### Subscription Module:



This module mainly deals with the subscription process of a new customer.

1. After a customer logs into his or her account and wants to subscribe to our service, he or she can enter his account managing page and select the monthly subscription mode. We currently provide 2 subscription modes. One is question-only, the other is question-books. For the first mode, the user can post questions and view answers for the questions they need answers for. For the second mode, They can access the online book resources in the form of buy/sell/lend/borrow, within their monthly quota.

2. Once the mode of subscription is selected, the customer will provide payment.

2.1 The system admin will validate the payment and reply with a payment confirmation email.

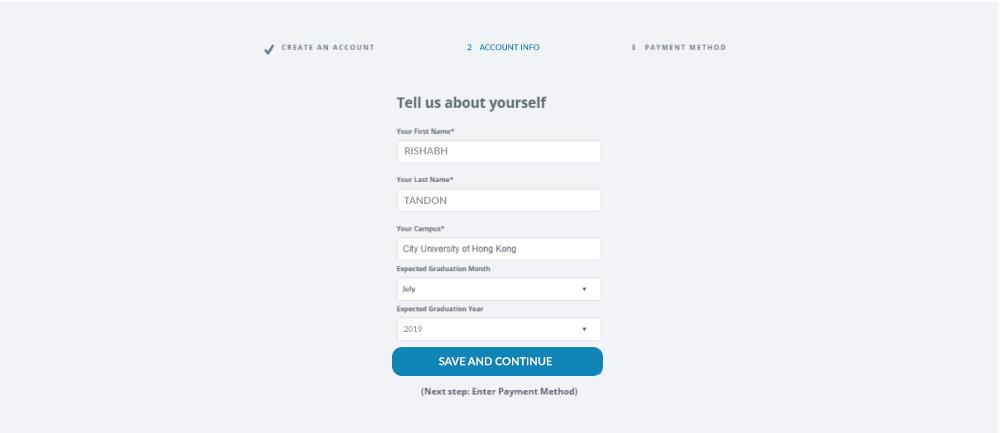
2.2 Then the admin will approve the user’s request of subscription and update the user database to add a record of the user.

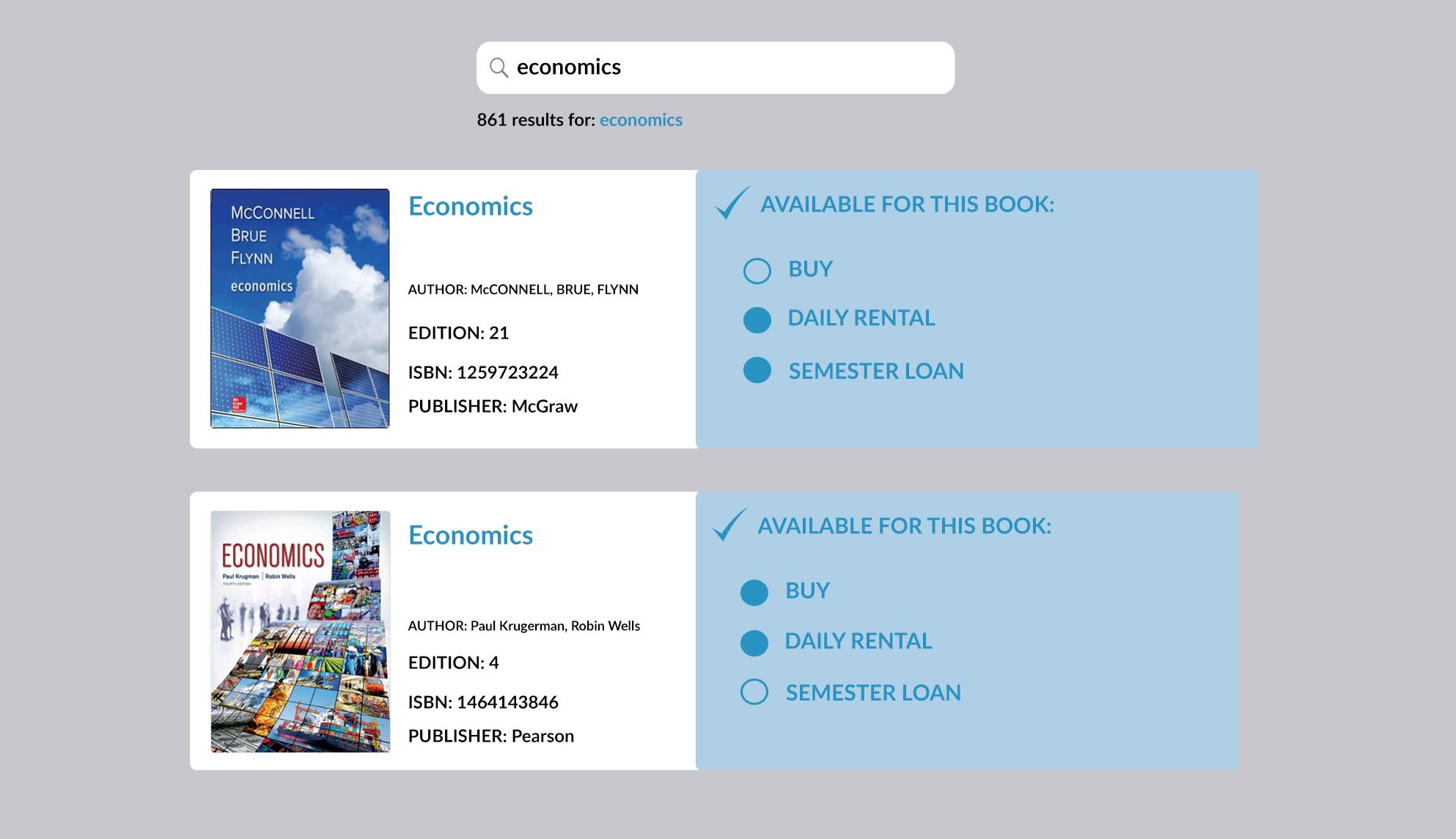
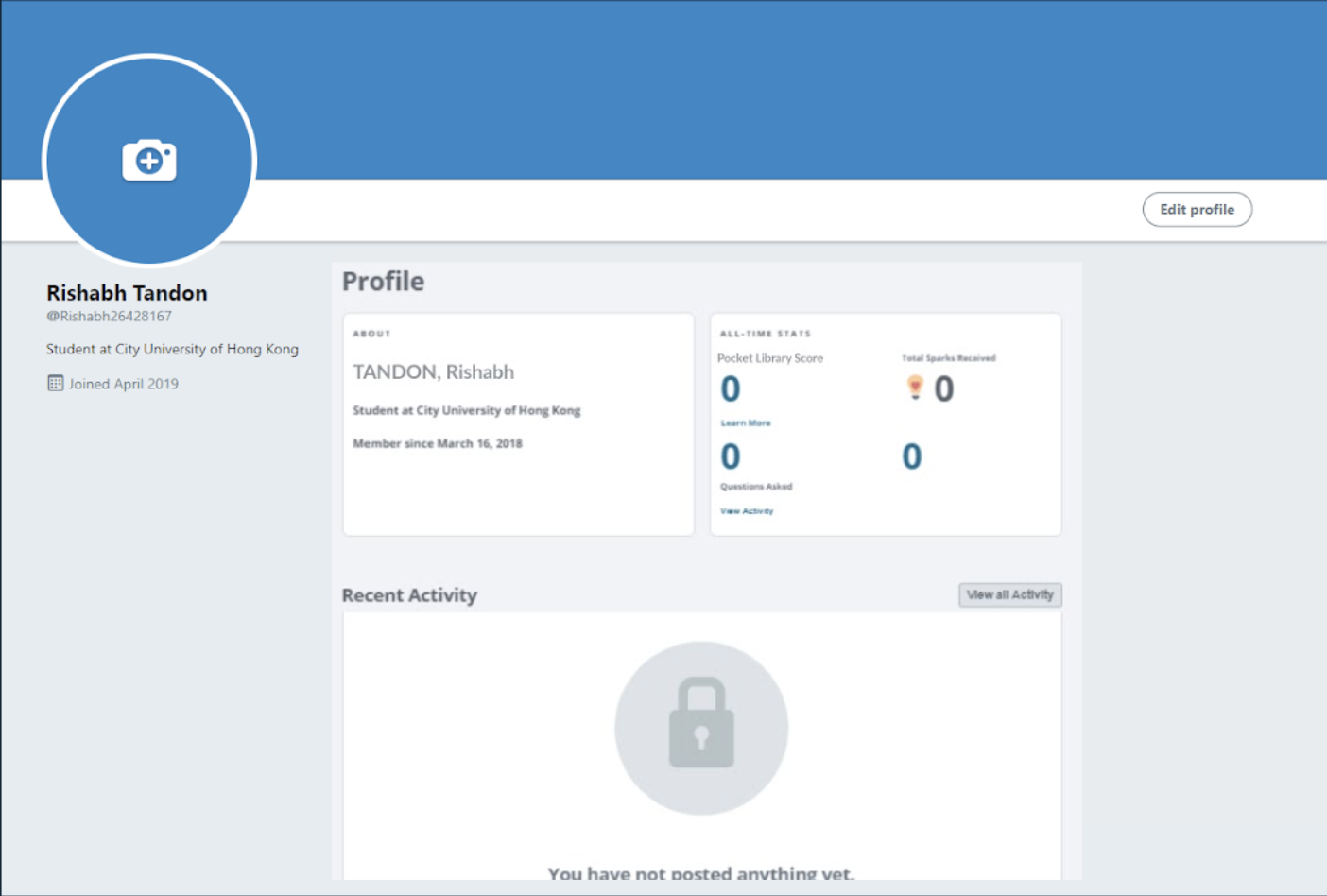
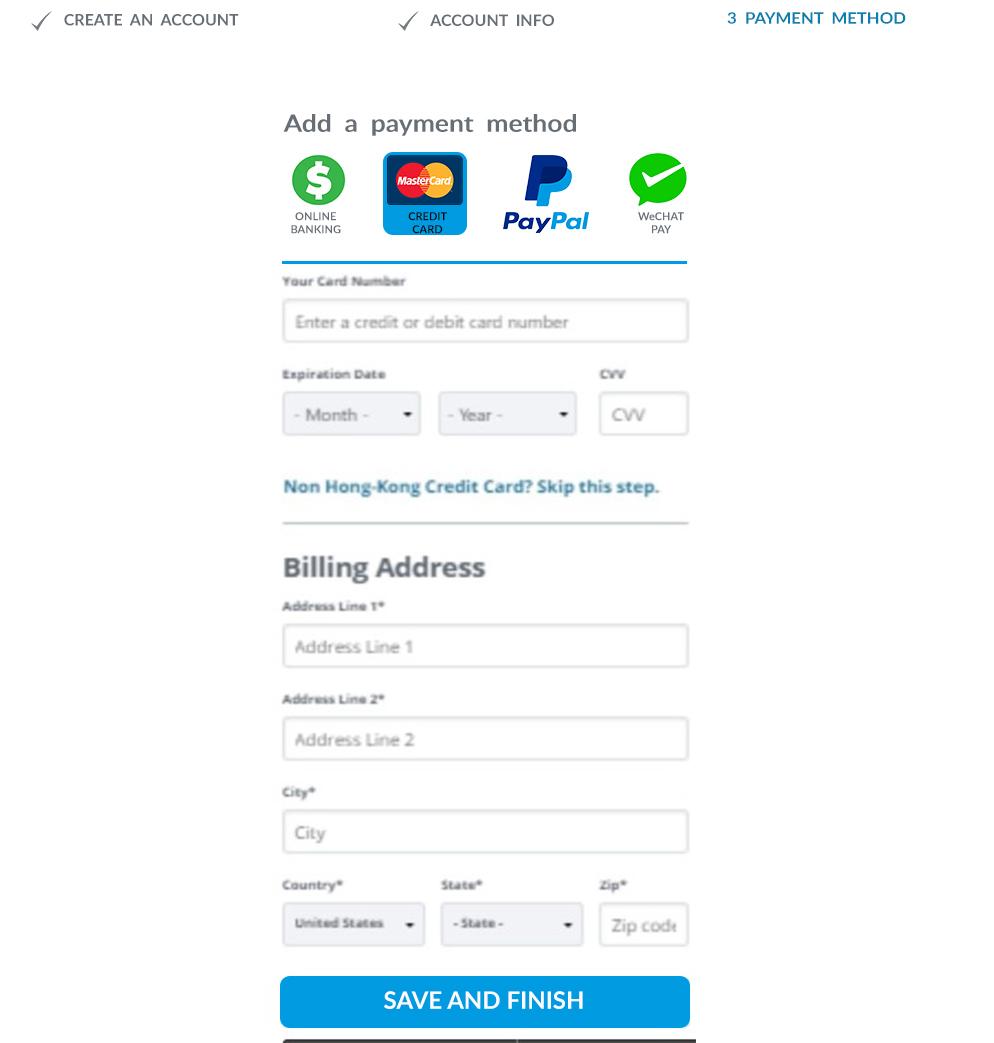
2.2.1 Once the user database has an add-user update, it will generate a subscription confirmation email and send it to the user.

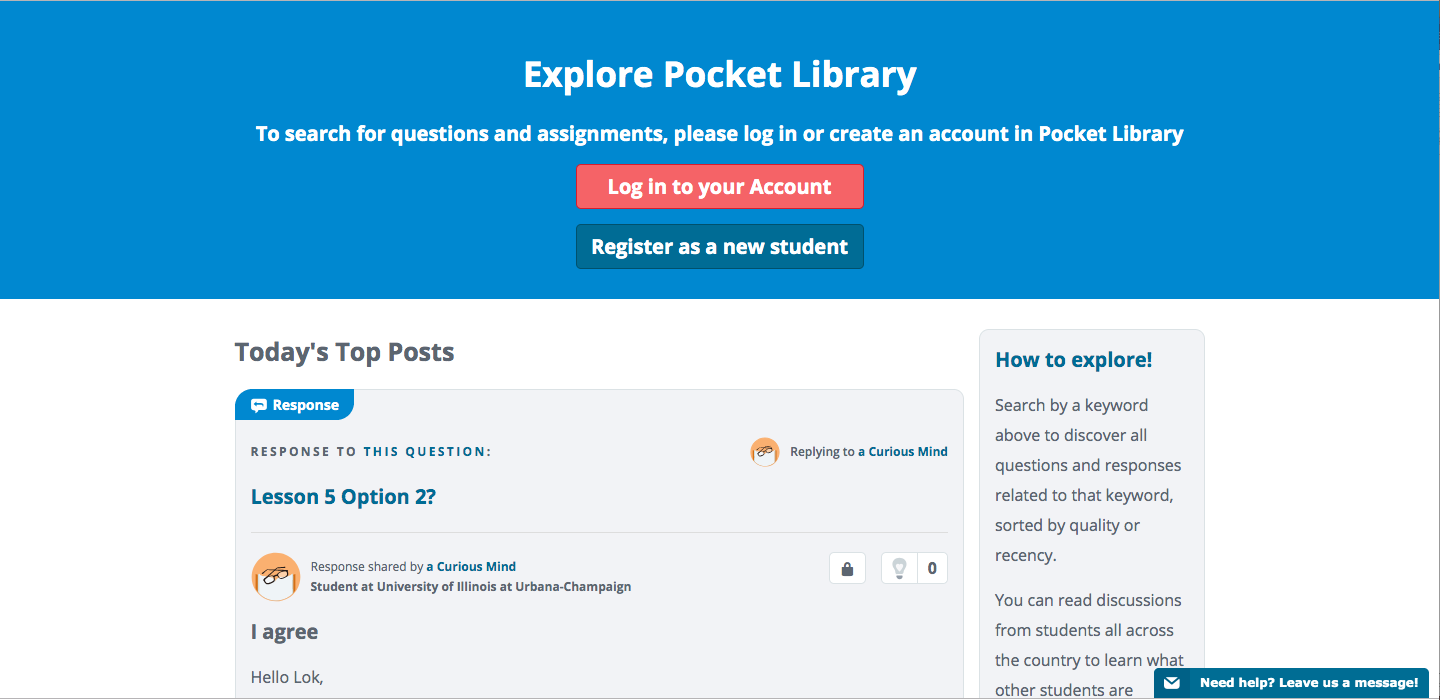
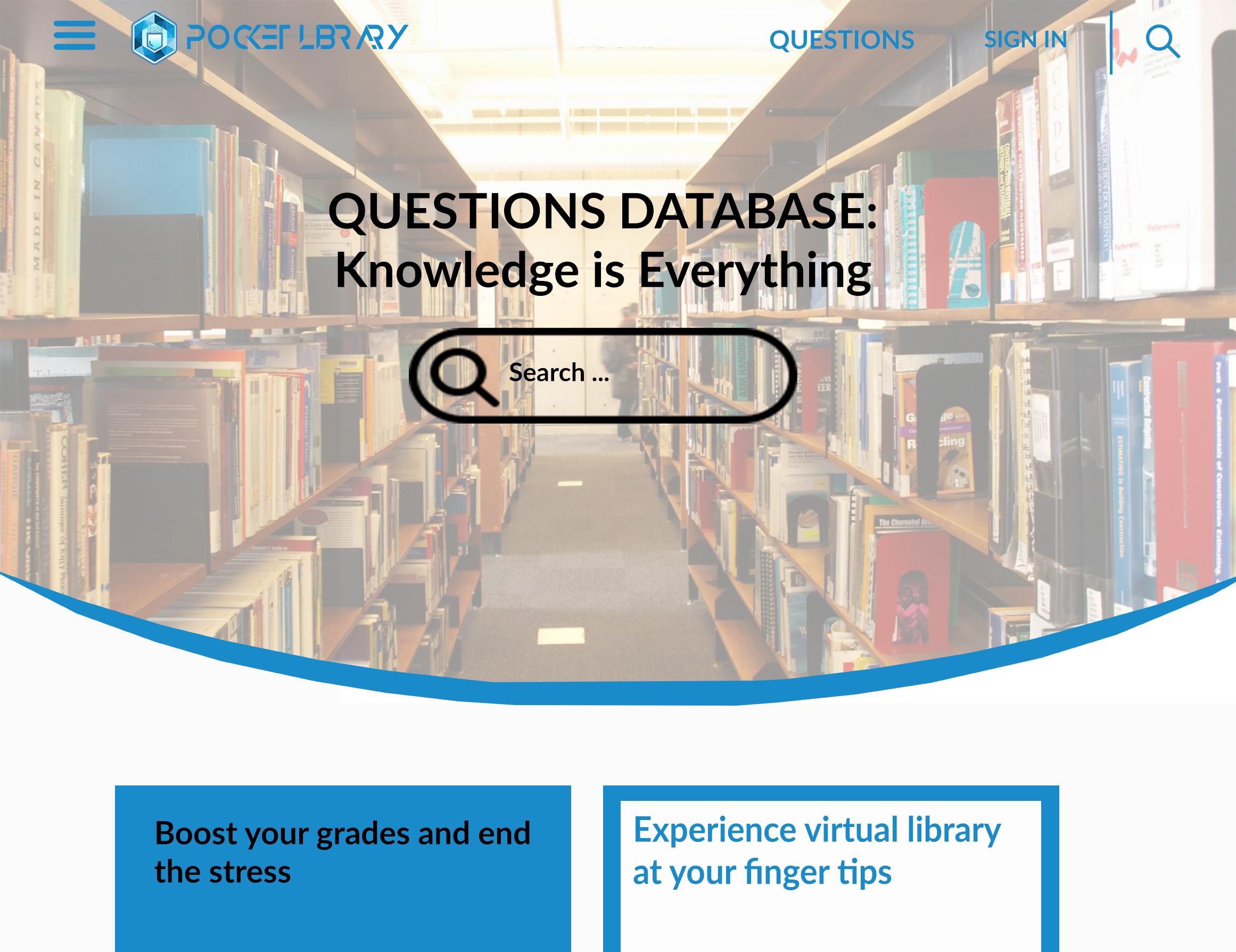
## 

## Prototype:

## 







## Conclusion:

Pocket Library™ is a software designed for all students to gain access to education anywhere, at any time. In the beginning stages of the development of our project we constructed a rough prototype in order to facilitate feedback and surveys from our targeted audience along every step of our development process. We chose to implement the Prototype Process Model as our project is heavily contingent upon widespread customer support.

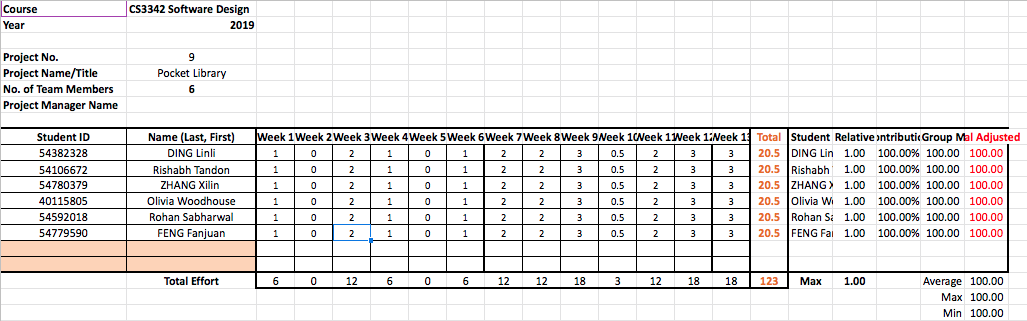
With the use case development of our project we determined that our system interacts with three actors: Student/Customer, Tutor, and Admin. Students are able to subscribe to our system in order to gain access to the many services that Pocket Library™ has to offer. The admin and tutors are responsible for enhancing a student’s experience with our system to the best of their ability.

Additionally, we applied various design principles and patterns of which were best suited for our project. These include the Liskov Substitution Principle, Open Closed Principle, State Pattern, Command Pattern, and the Singleton Principle. These design principles helped to produce a more efficient Pocket Library™ using industry standard practice.

Maintenance is arguably the largest portion of a software design project. We still have much work to do in the future. PR and advertising are possibly the most crucial future areas of work for our project. Due to the fact that many similar softwares already exist in this market, we will need to continue our use of the Prototype Process Model to continue to make our system better and more attractive to customers than its competitors. We will continue to focus our attention on feedback and surveys from our customers to make our product the most suitable for students within its competing market. It is also important that we closely watch the size of our database, because as our data on user accounts and other information increases, we must also increase the size of our database to hold larger amounts of data. We will also consider developing apps for different devices (Android and iOS applications) as our system gains more customers and support. Finally, we will consider adding special features to our system such as private tutoring/classes or additional payment methods such as Apple Pay.

In conclusion, while we still have much to do in terms of development and maintenance of this project, we believe that Pocket Library™ is a great way for students to manage all of their textbook needs. It is important that in the age of technology, education gets adapted to be more accessible and sustainable for everyone. It is time to make education accessible with Pocket Library™!

## Teamwork Timetable:



1. Weisbaum, Herb. “College Textbook Costs More Outrageous than Ever.” *CNBC*, CNBC, 28 Jan. 2014, www.cnbc.com/2014/01/28/college-textbook-costs-more-outrageous-than-ever.html. [↑](#footnote-ref-0)
2. Dickler, Jessica. “Student Loan Interest Rates Edge Higher and Higher.” *CNBC*, CNBC, 18 July 2017, www.cnbc.com/2017/07/18/student-loan-interest-rates-edge-higher-and-higher.html. [↑](#footnote-ref-1)
3. Weisbaum, Herb. “College Textbook Costs More Outrageous than Ever.” *CNBC*, CNBC, 28 Jan. 2014, www.cnbc.com/2014/01/28/college-textbook-costs-more-outrageous-than-ever.html. [↑](#footnote-ref-2)