Online Book Store

Bookshelf: Enhancing the Online Book Buying Experience through Personalized Recommendations, Easy Wish list Management, Secure Payment Options, and Streamlined Seller Processes

Project Description:

Bookshelf is an online bookstore where you can search for a book based on a genre you like, add it to a wish list, add it to a cart, proceed to choose a payment method, and checkout at the end. This assumes that you are a customer. If you are a seller, you can add a book or remove a book you added before, updating it as long as the admin approves it.

Functional Requirements:

- **Inventory Management:**

The system should be able to keep track of all books in stock.

It should allow adding new books to the inventory.

It should facilitate updating book information such as title, author, genre, price, and quantity available.

It should support removing books from the inventory when they are sold out or discontinued.

- **Order Management:**

Users should be able to browse available books.

Users should be able to search for books based on title, author, genre, or other criteria.

Users should be able to add books to their shopping cart.

Users should be able to place orders for books in their shopping cart.

The system should send confirmation emails upon successful order placement.

- **User Management:**

Users should be able to create accounts.

Users should be able to log in and log out.

Registered users should have access to additional features such as order history and wish lists.

- **Payment Processing:**

The system should securely handle online payments.

It should support various payment methods like credit/debit cards, PayPal, etc.

It should encrypt sensitive payment information to ensure security.

- **Admin Panel:**

Admins should have access to a dashboard to manage inventory, users, and orders.

Admins should be able to approve or reject added books to the inventory.

Admins should be able to view and manage user accounts.

Admins should be able to view and manage orders, including order status updates and cancellations.

- **Seller panel:**

Sellers should be able to add, edit, or remove books from the inventory.

Sellers can see and update the books they uploaded.

Non-Functional Requirements:

- **Performance:**

The system should be responsive, with quick page load times.

It should handle a large number of concurrent users without significant performance degradation.

- **Security:**

The system should implement proper authentication and authorization mechanisms to prevent unauthorized access.

It should encrypt sensitive user data and payment information.

It should protect against common web vulnerabilities like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

- **Scalability:**

The system should be designed to scale easily to accommodate increased traffic and growing inventory.

It should be able to handle spikes in demand during peak periods without downtime or performance issues.

- **Reliability:**

The system should be highly reliable, with minimal downtime.

It should have backups and disaster recovery mechanisms in place to ensure data integrity and availability.

- **Usability:**

The user interface should be intuitive and easy to navigate.

It should provide clear instructions and feedback to users throughout the shopping and checkout process.

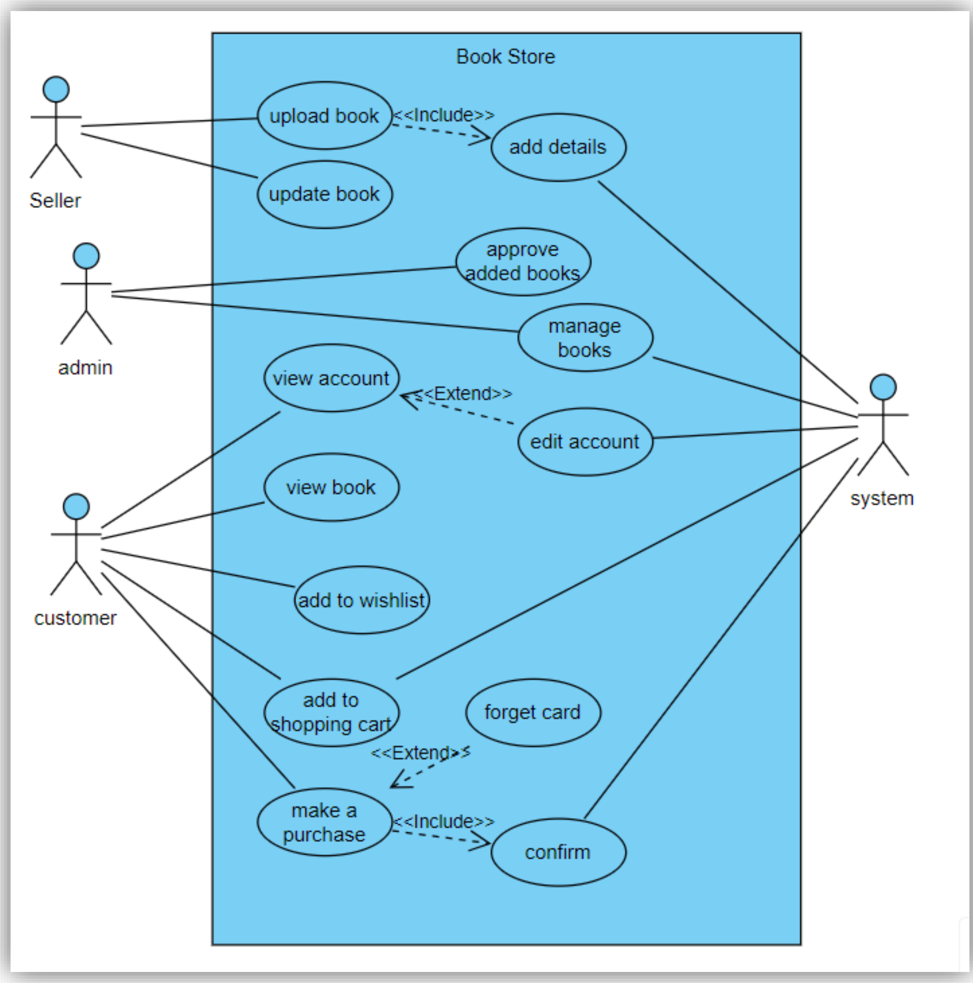
- **Accessibility:**

The system should be accessible to users with disabilities, complying with accessibility standards like WCAG (Web Content Accessibility Guidelines).

- **Compatibility:**

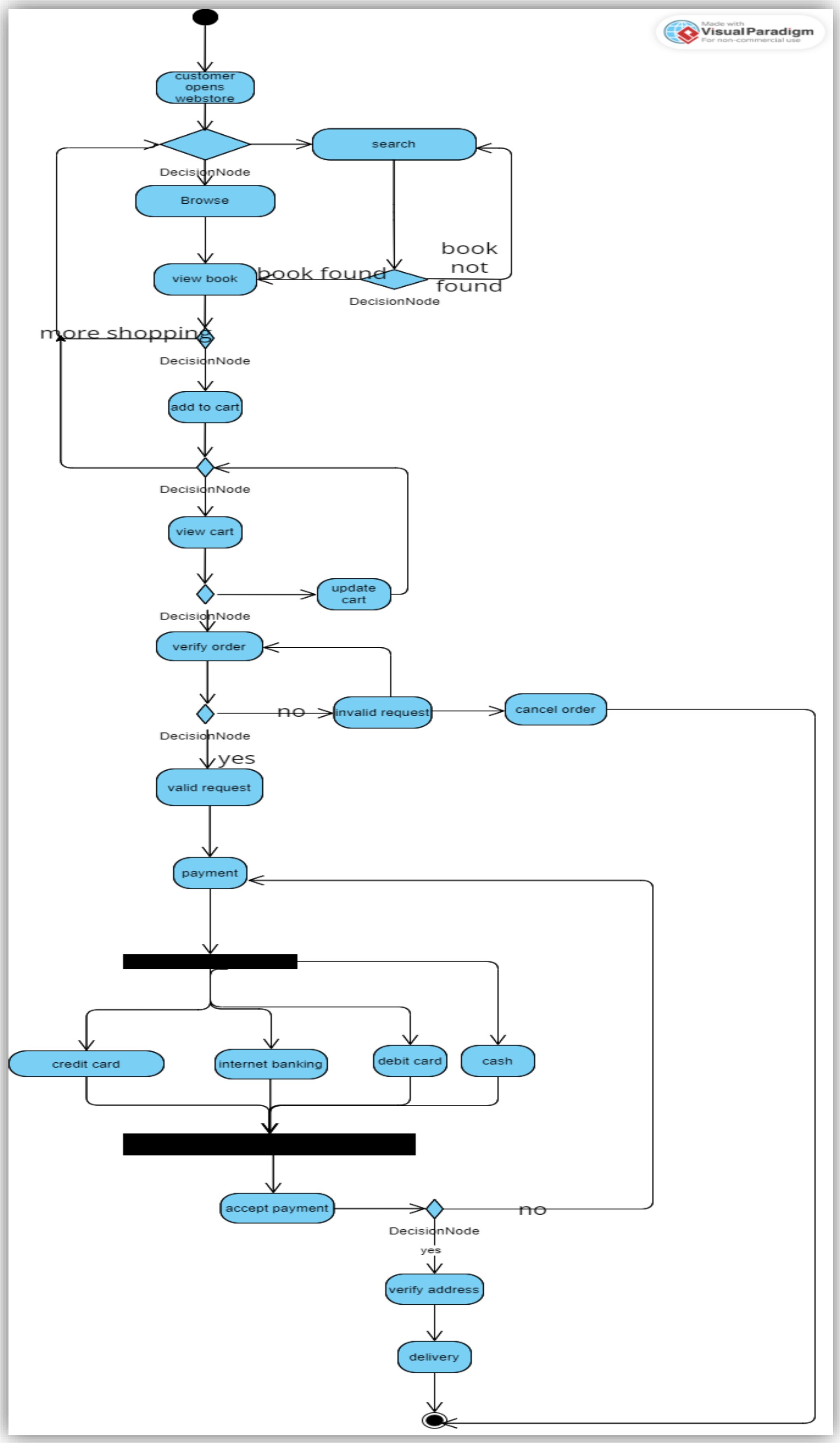
The system should be compatible with a wide range of devices and browsers to ensure a consistent user experience across different platforms.

1-Use case



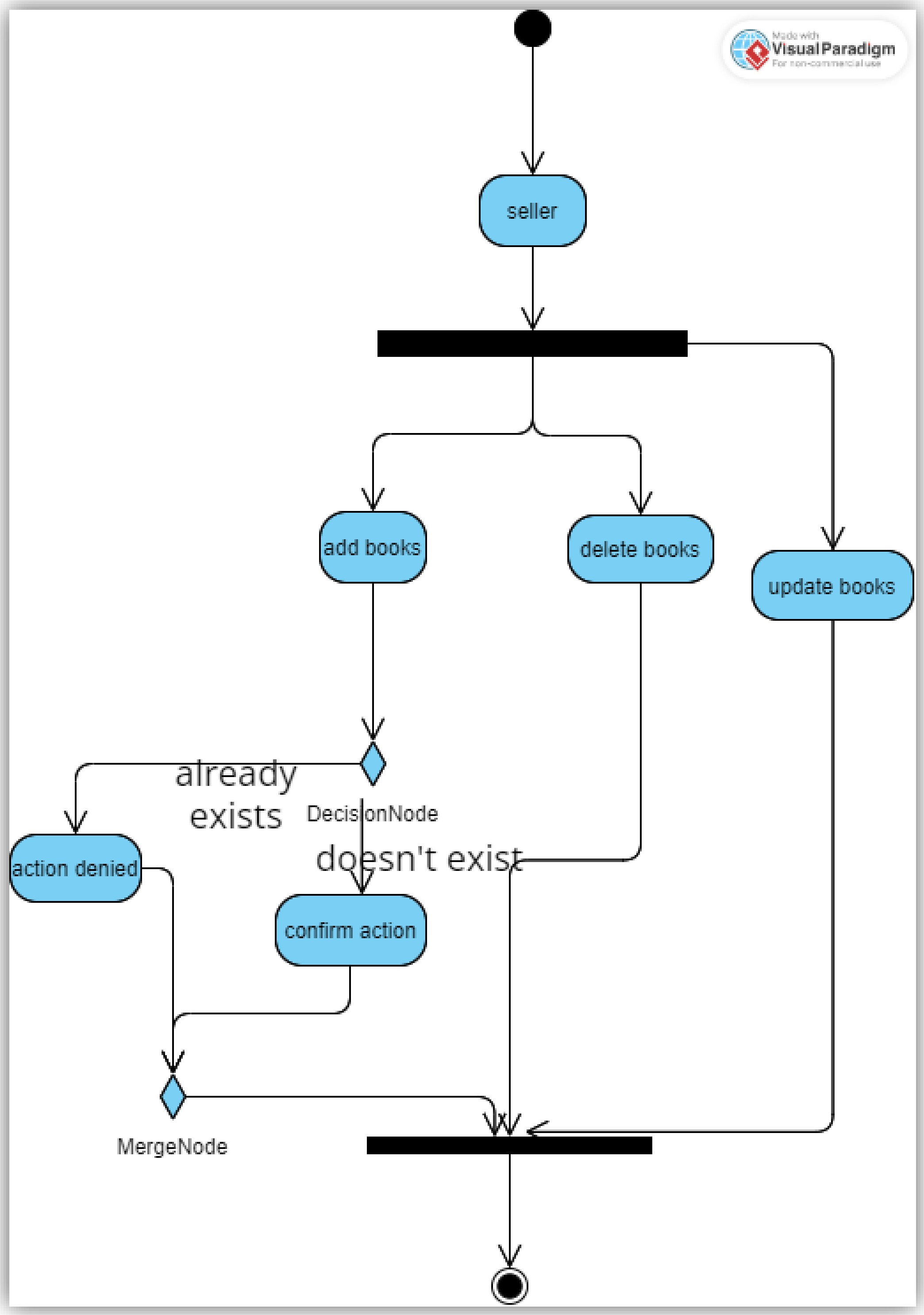
2-Activity Diagram

2.1 Customer



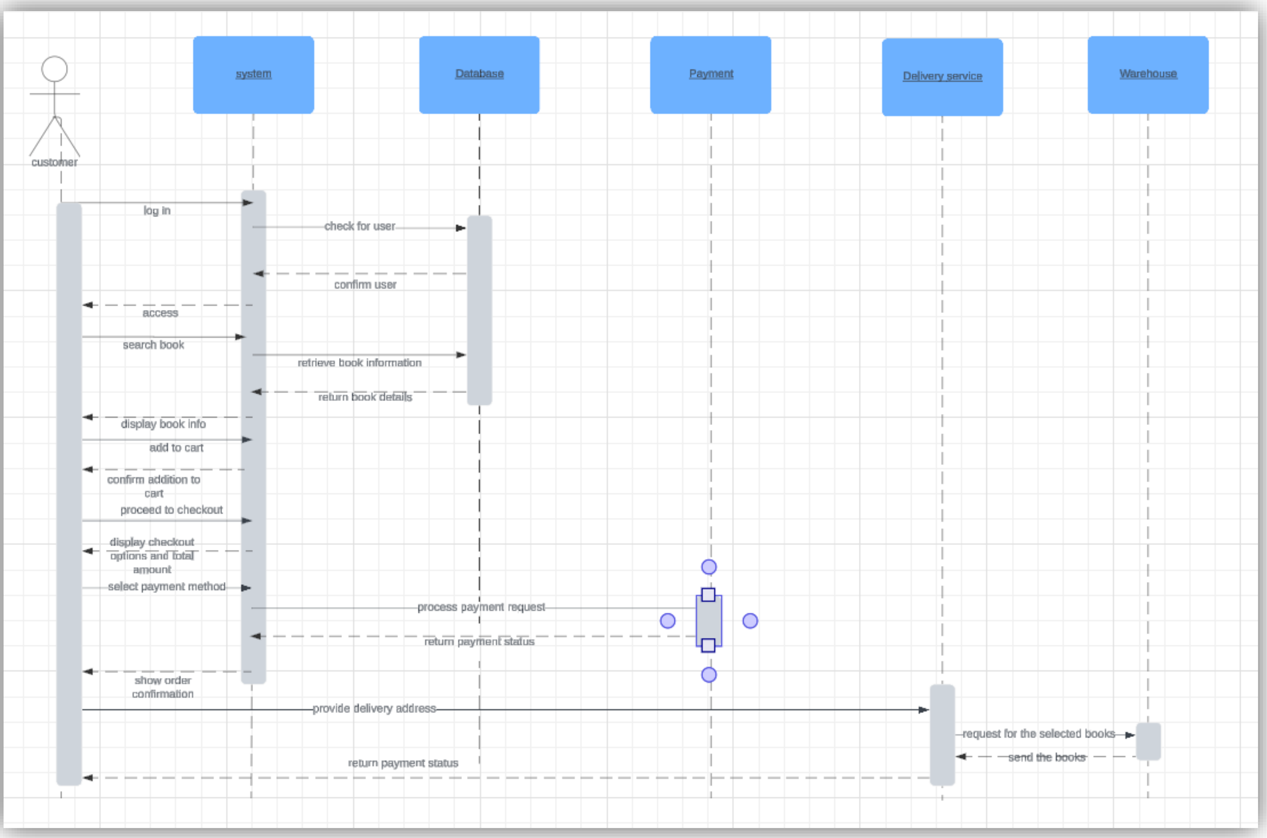
Made with Visual Paradigm For non-commercial use
customer opens webstore
DecisionNode
search
Browse
book not found
view book
book found
DecisionNode
more shopping
DecisionNode
add to cart
DecisionNode
view cart
update cart
DecisionNode
verify order
no
invalid request
cancel order
DecisionNode
yes
valid request
payment
credit card
internet banking
debit card
cash
accept payment
no
DecisionNode
yes
verify address
delivery

2.2 Seller

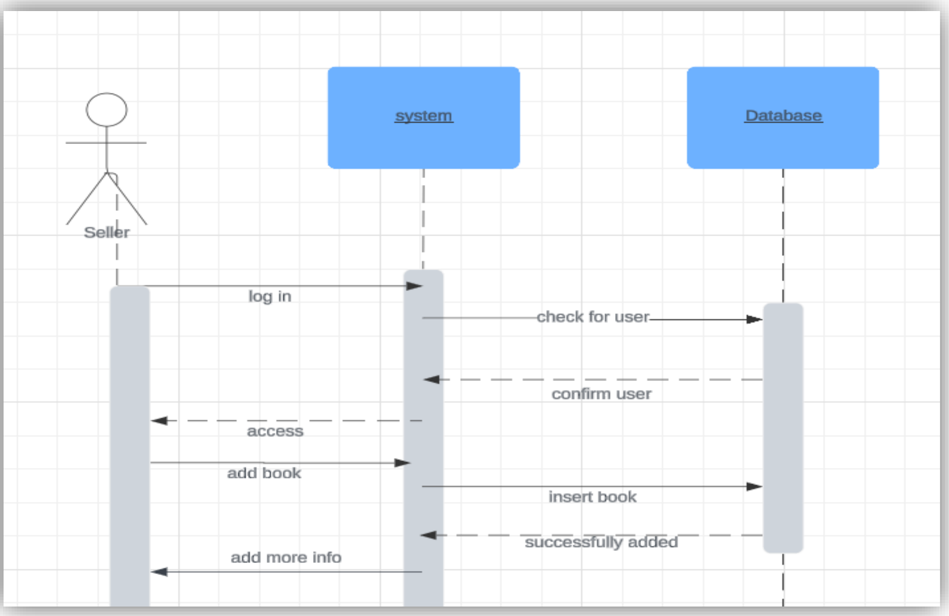

seller
add books
delete books
update books
already exists
DecisionNode
doesn't exist
action denied
confirm action
MergeNode

3-Sequence Diagram

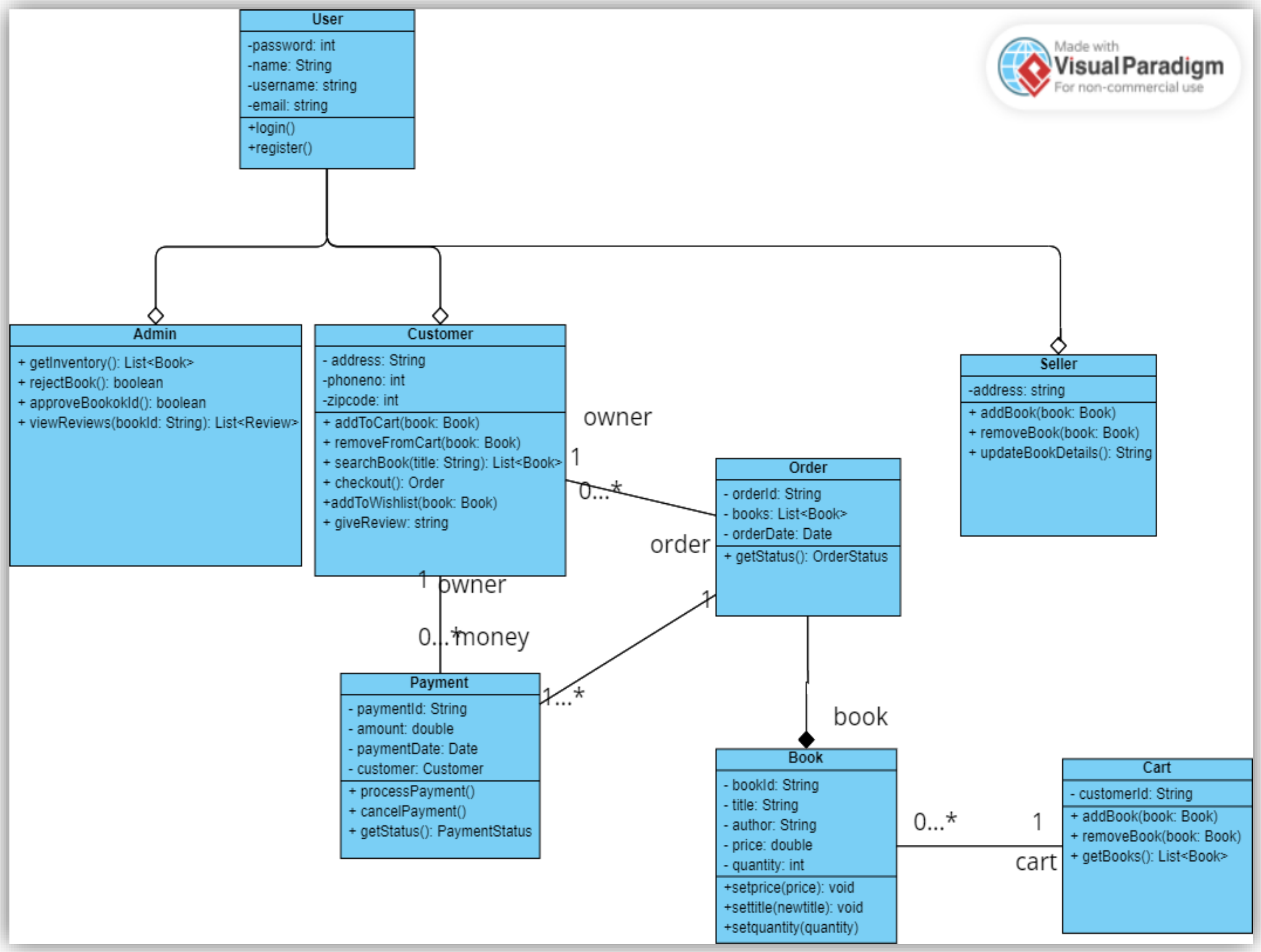
3.1 Customer

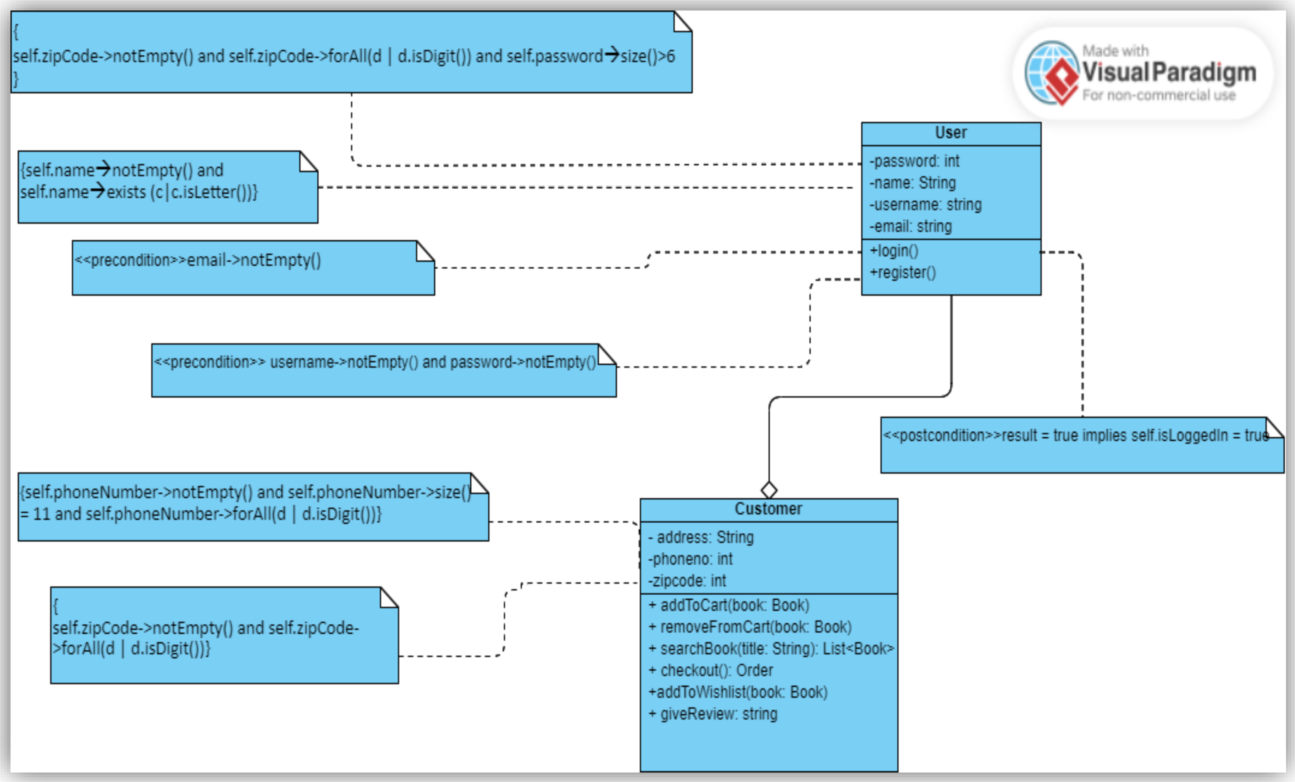


3.2 Seller



4- Class Diagram




{
self.zipCode->notEmpty() and self.zipCode->forAll(d | d.isDigit()) and self.password→size()>6
}
{self.name→notEmpty() and
self.name→exists (c|c.isLetter())}
<<precondition>>email->notEmpty()
<<precondition>> username->notEmpty() and password->notEmpty()
{self.phoneNumber->notEmpty() and self.phoneNumber->size()
= 11 and self.phoneNumber->forAll(d | d.isDigit())}
{
self.zipCode->notEmpty() and self.zipCode-
>forAll(d | d.isDigit())}
Made with Visual Paradigm For non-commercial use
User
-password: int
-name: String
-username: string
-email: string
+login()
+register()
<<postcondition>>result = true implies self.isLoggedIn = true
Customer
- address: String
-phoneno: int
-zipcode: int
+ addToCart(book: Book)
+ removeFromCart(book: Book)
+ searchBook(title: String): List<Book>
+ checkout(): Order
+addToWishlist(book: Book)
+ giveReview: string

-context Customer

inv: self.phoneNumber->not Empty() and self.phoneNumber->size() = 11 and self.phoneNumber->forAll(d | d.isDigit())

-context
Customer

inv: self.zipCode->not Empty() and self.zipCode->forAll(d | d.isDigit())

-context user

Inv: self.password.isletter () and self.password.isDigit () and self. Password→size ()>6

-context user

Inv: self.name→not Empty () and self.name→exists (c|c.isLetter ())

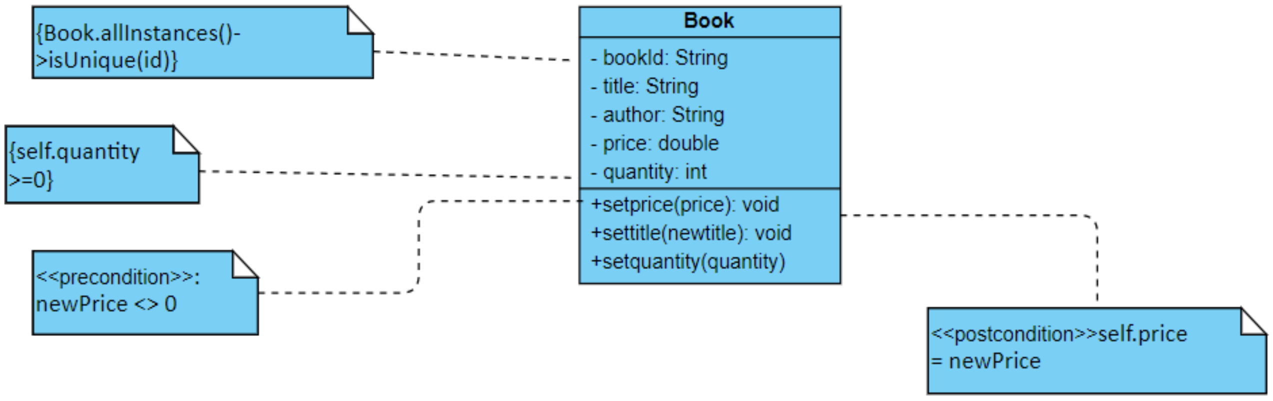
-context User::login(email: String, password: String): Boolean

pre: email->not Empty()

post: result = true implies self.isLoggedIn = true

-context Customer::register (username: String, password: int)

pre: username->not Empty() and password->not Empty()



-context Book

Inv: self.quantity >=0

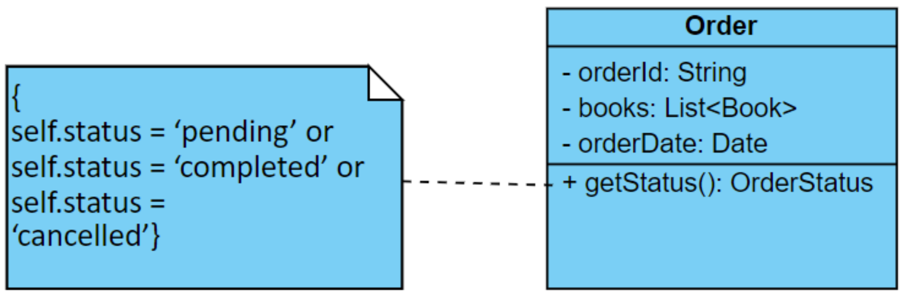
-context Book

Inv: Book.allInstances () >isUnique (id)

-context Book::setPrice (newPrice: Real)

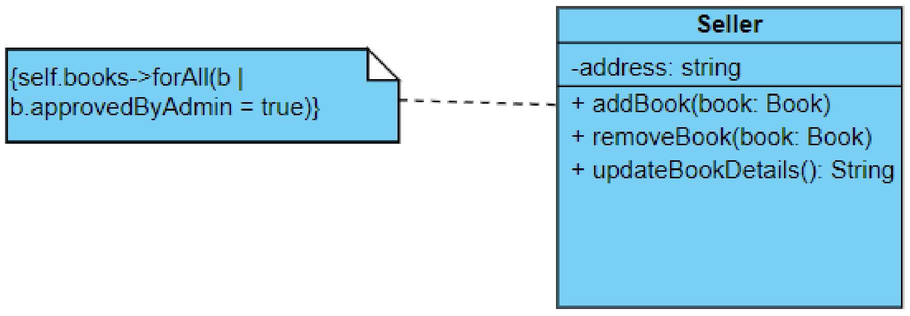
Pre: newPrice < > 0

Post: self. Price = newPrice



-context Order

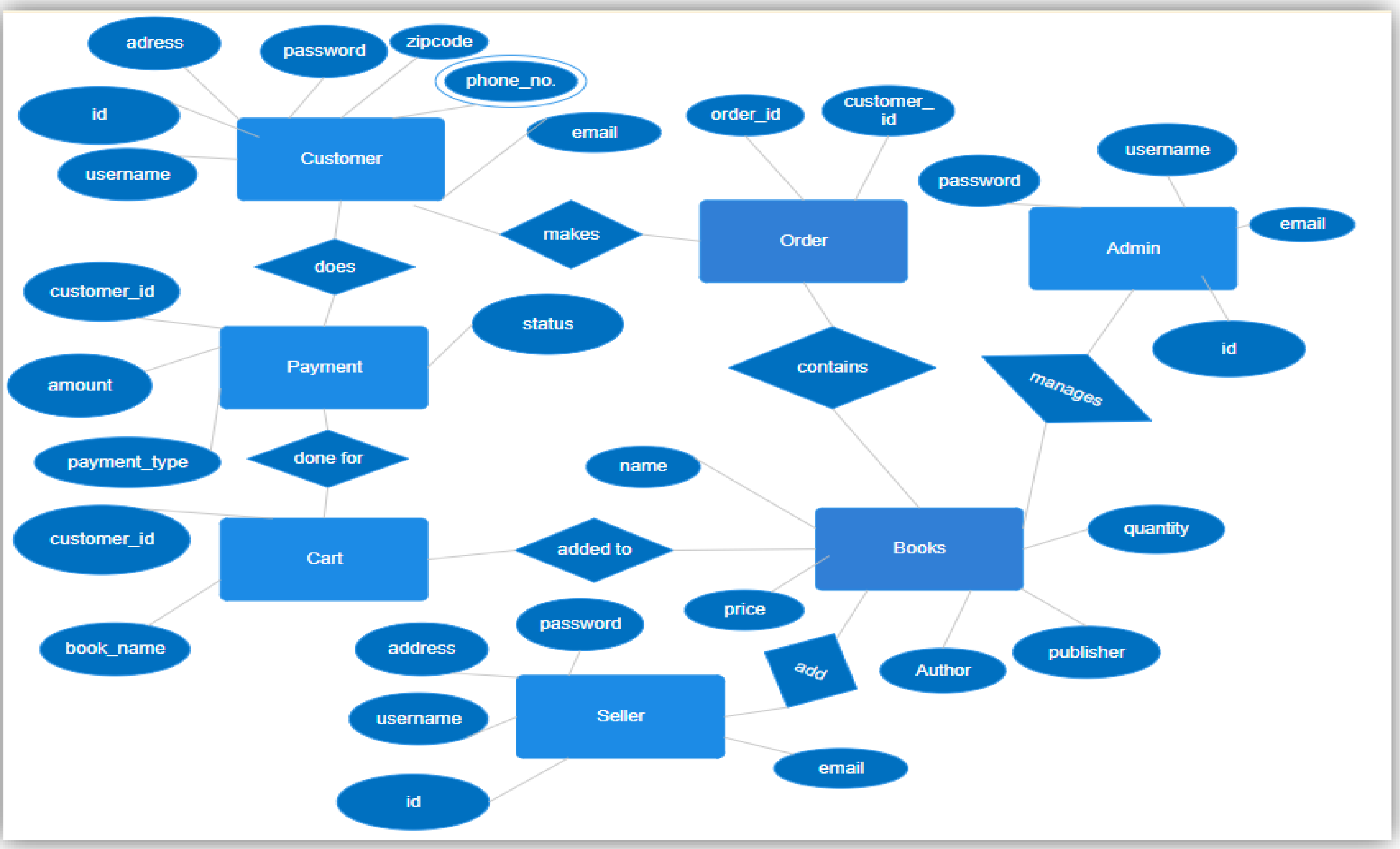
Inv: self.status = ‘pending’ or self.status = ‘completed’ or self.status = ‘cancelled’



-context Seller

inv: self.books->forAll(b | b.approvedByAdmin = true)

5- ERD


adress
password
zipcode
phone_no.
id
username
Customer
email
order_id
customer_ id
username
password
Admin
email
Order
makes
does
customer_id
Payment
status
amount
contains
manages
id
payment_type
done for
name
customer_id
Cart
added to
Books
quantity
price
book_name
address
password
add
Author
publisher
username
Seller
email
id