Bookstore Inventory System
Software Design Document

Version 1.0

Revision History

Date	Version	Description	Author
17 November, 2010	0.1	Initial Draft	Gerson Recinos Ho Nam Ho Jimar Miller Adam Wurtzel David Altum Francisco Diaz Finan Bariagabr
22 November, 2010	0.2	Initial Draft – Revision	Gerson Recinos Ho Nam Ho Jimar Miller Adam Wurtzel David Altum Francisco Diaz Finan Bariagabr
24 November, 2010	0.3	Assignment – Class Diagram	Gerson Recinos Ho Nam Ho Jimar Miller Adam Wurtzel David Altum Francisco Diaz Finan Bariagabr
04 December, 2010	0.4	Revision of Class Diagram	Gerson Recinos Ho Nam Ho Jimar Miller Adam Wurtzel David Altum Francisco Diaz Finan Bariagabr
06 December, 2010	0.5	Final Revisions	Gerson Recinos Ho Nam Ho Jimar Miller Adam Wurtzel

			David Altum Francisco Diaz Finan Bariagabr
07 December, 2010	0.7	Final Draft	Gerson Recinos Ho Nam Ho Jimar Miller Adam Wurtzel David Altum Francisco Diaz Finan Bariagabr
08 December, 2010	1.0	Final Draft – Finalized	Gerson Recinos Ho Nam Ho Jimar Miller Adam Wurtzel David Altum Francisco Diaz Finan Bariagabr

Table of Contents

Software Design Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system.

1.2 Scope

This program will be used as an all-around back-end bookstore inventory system. Some of the key features of the system are the following. The software will enable the client (business) to have real time statistics of sales and book inventory, automatically produce End of Day sales reports and End of Day low-inventory notices (purchase order suggestions).

It will also enable the users (customers) to view the real time inventory and extract book information, enable users to place orders online and pick up books in-store and access to a personalized account profile, order history, etc.

Lastly, the program will enable vendors to have access to part of the system – it will allow any vendor to update book information, add/remove new books to/from the inventory and updated prices.

1.3 Definitions, Acronyms, and Abbreviations

This is a comprehensive list of all terms used in this vision document.

POS (Point of Sale) – An electronic terminal that handles all credit/cash transactions.
Vendor -A company/person who is in the business of selling products and goods to businesses.
Inventory – A detailed list of goods and materials that are in stock.
User – a person who can interact with the software – can be an employee or end user (customer).
Client – The UNLV bookstore.
Book Inventory – The detailed list of books in stock.
Database (DB) – An organized (structured) body of related information.
End of Day Report –A report that is done after business hours are over. Typically, it includes sales and inventory.
Sales – The overall money transaction during a specified time interval.
Transaction –The exchange of goods or services for legal tender.

1.4 References

None.

1.5 Overview

In the following sections we outline the software product in higher detail. We will start with defining the key features that will be implemented. Next, we will discuss the constraints that will be imposed upon the software and the quality ranges, in other words, the robustness, fault tolerance and usability of the software product amongst other things. In the precedence and priority section we will comment on the most important functionalities that the software product must have and the integrity of the sales system.

In the following sections will discuss all other product requirements, such as, performance requirements, platform requirements and environmental requirements. Lastly, we will comment on the documentation requirements, such as, user manuals, online help & support, installation and packaging.

2. Architectural Representation

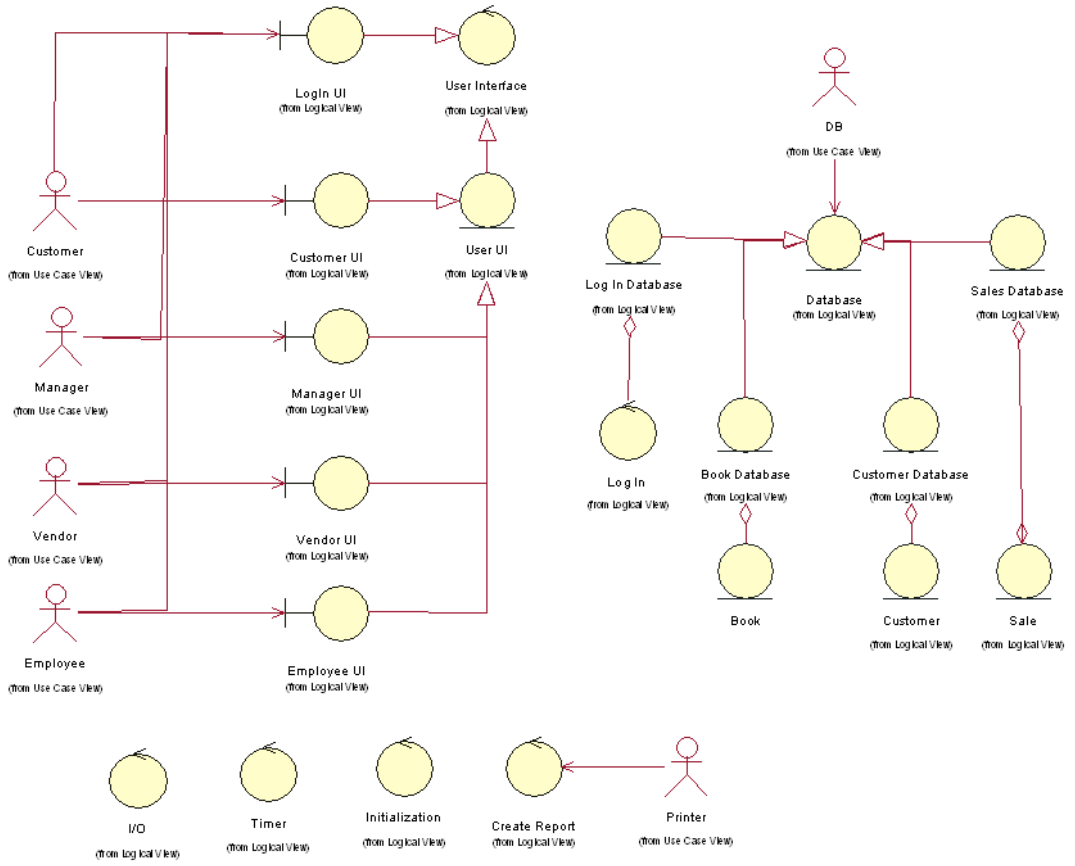

Figure 1 - Architectural Representation

3. Architectural Goals and Constraints

- The Bookstore Inventory System will run on a dedicated platform with access to a SQL database
- The Bookstore Inventory System receives input from the external POS and from UPC Code Scanner.
- The Database save all information that is provided.
- All transactions are performed within a timely manner.

4. Use-Case View

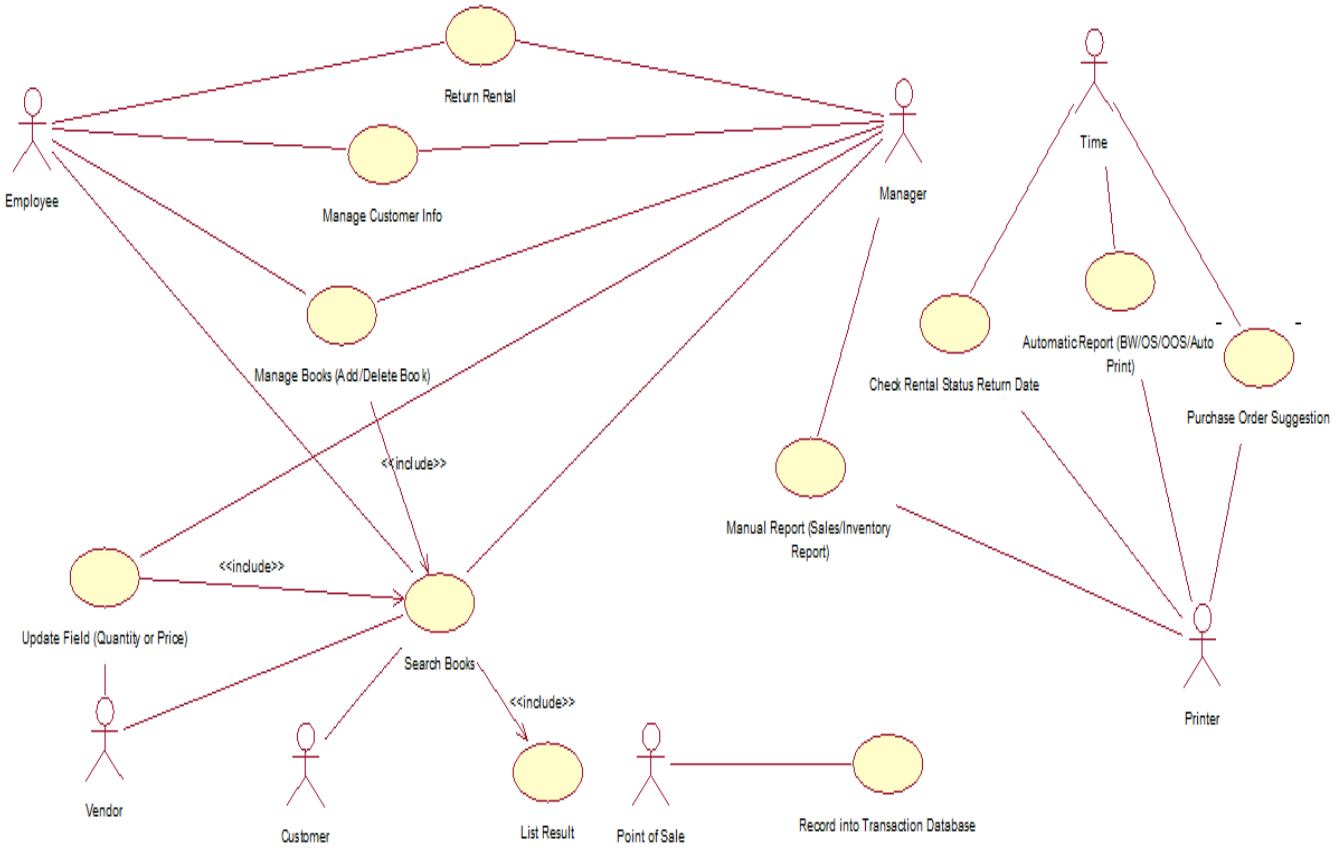

Figure 2 - Use Case Diagram

4.1 User Interfaces

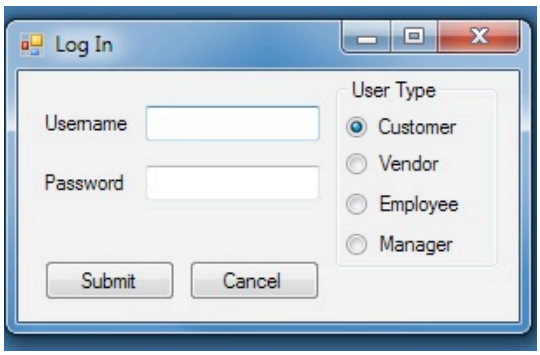

Figure 3 - Log In User Interface

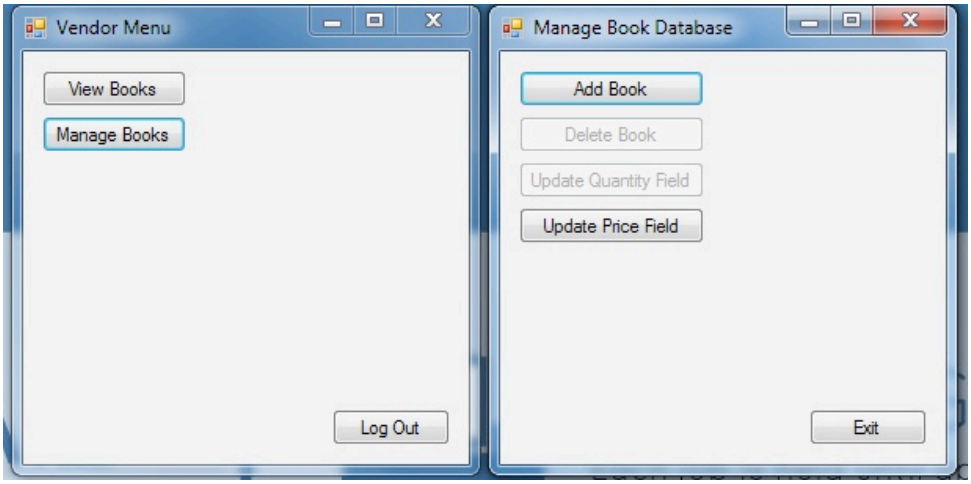

Figure 4 - Vendor UI and Manage Books UI

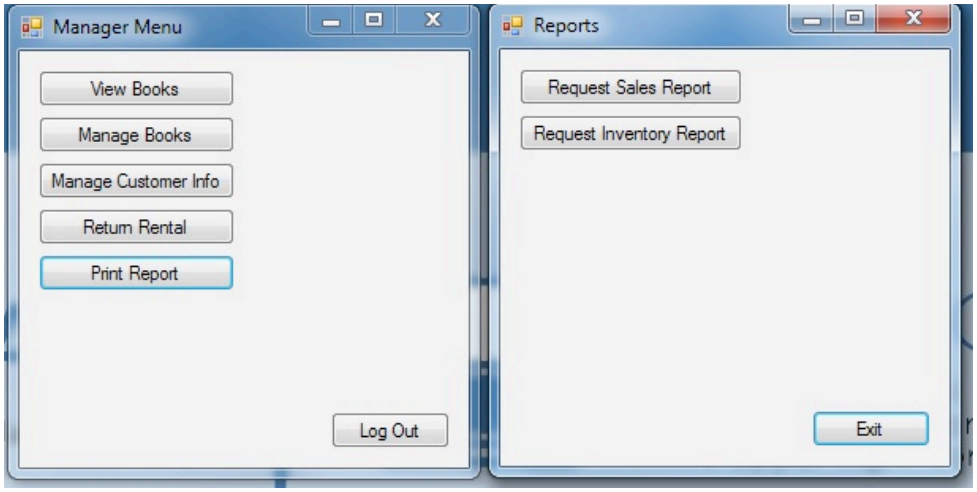

Figure 5 - Manage UI & Reports UI

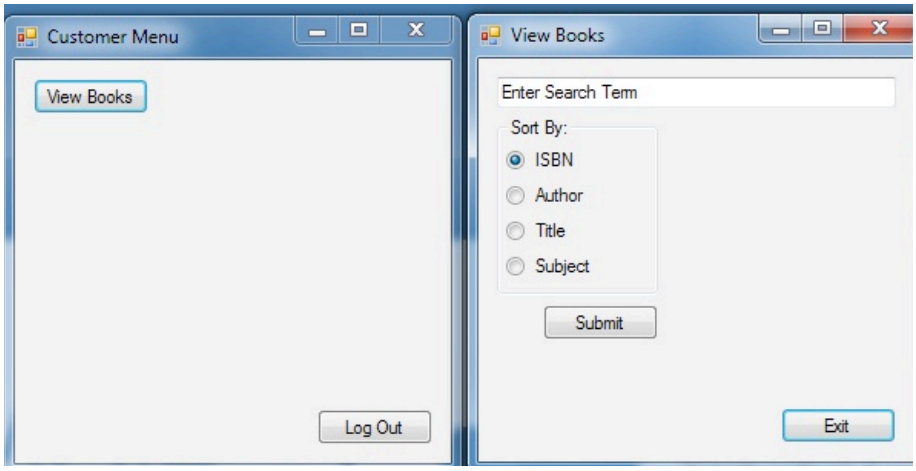

Figure 6 - Customer UI & View Books UI

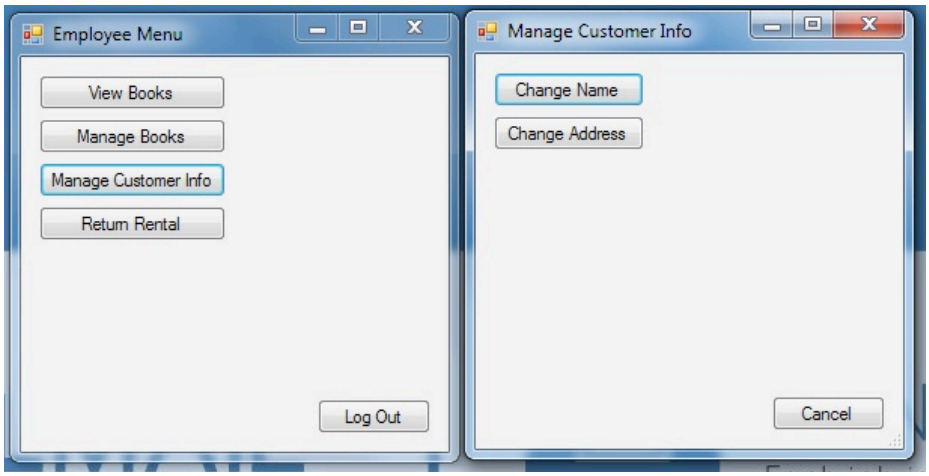

Figure 7 - Employee UI & Customer Info UI

4.2 System Inputs and Outputs

Inputs

- The system accepts input from the POS system.
- The system accepts input from a keyboard & mouse from a terminal that employees/staff/vendors can access.

Outputs

- The system outputs reports to a printer.
- The system outputs search results, book inventory, etc. to a screen.

4.3 Use-Case Realizations

4.3.1 Return Rental

- **Brief Description**
  - In this use case the system updates the customer information in the database after a customer return a book from the POS
- **Actors**

  POS, Customer Info Database
- **Basic Flow of Events:**

  1. This use case begins with the system receiving input from the POS.
  2. The system receives the customer name and book returned, and then retrieves the customer info from the customer database.
  3. The system verifies if book returned is the same book in record.
  4. The system confirms the customer has returned the book.
  5. End of use case.

- **Alternate Flow of Events #1: The system does not find the rental book in the customer info database.**

  1. This use case begins with the system receiving input from the POS.

2. The system receives the customer name and book returned, and then retrieves the customer info from the customer database.
3. The system could not verify the book returned is the same book in record.
4. The system displays error at the interface to prompt the user.
5. This use case ends.

4.3.2 Rented Book Returned

- **Brief Description**
    - In this use case, the system searches the customer database for all rented books that are due at a specified date. Checks whether the book(s) has been returned.
- **Actors**

  Employee, Customer Info Database, Printer
- **Basic Flow of Events:**

1. This use case begins when a staff member requests the report for book rentals due.
2. The system prompts the user for a specified date.
3. The database iterates through the customer info database and looks for a flag that shows a customer has a rented book.
4. The system finds the field and checks whether a rented books is due on the provided date.
5. The system confirms that a book is due on the given date.
6. The system checks whether the customer has returned the book.
7. The system verified that the book(s) have been returned.
8. End of use case.

- **Alternate Flow of Events #1: The system find that a rented book has not been returned.**

1. This use case begins when a staff member requests the report for book rental due.
2. The system prompts the user for a date.
3. The system iterates through the customer database and looks for a rented books flag.
4. The system finds the rented books flag and checks whether the given date matches the due date.
5. The system confirms that a books is due on the given date.
6. The system checks whether the book(s) has been returned.
7. The system is unable to confirm that the book has been returned.
8. The system prints out the customer information.
9. This use case ends.

- **Alternate Flow of Events #2: The system finds that a customer does not have book rental due**

1. This use case begins when a staff member request the report for book rentals due.
2. The staff member enters a specific date to check.
3. The system iterates through the customer info database and looks for a “Rented Books” flag.
4. The system does not find the rented books flag for the customer.
5. The system continues to the next customer in the database.

6. This use case end

- **Alternate Flow of Events #3: The system finds that a customer has a rented book (or rented books) in their possession, but they are not due at the specified date.**
  1. This use case begins when a staff member requests the report for book rentals due.
  2. The system prompts the user for a specific date.
  3. The system iterates through the customer info database and looks for a flag that show a customer has a rented book.
  4. The system finds the flag and compares the specified date with the date in the database.
  5. The system determines that they are not the same.
  6. The system moves on to the next customer.
  7. This use case ends.

4.3.3 Record Transaction into Transaction Database

- **Brief Description**
  - In this use case the system records a transaction into the transaction database.
- **Actors**

  POS, Transactions Database
- **Basic Flow of Events:**

  1. This use case begins with the system receiving input from the POS of a new sale.
  2. The system creates a new entry in the database
  3. The system records the date, time, transaction ID, name of customer, number of items and purchase amount into the database.
  4. End of use case.

- **Alternate Flow of Events: The system receives notification of a transaction – refund**

  1. This use case begins when the system receives input from the POS of a refund.
  2. The system creates a new entry in the database.
  3. The system records the date, time, transaction id, name of customer, number of items and purchase amount to the database.
  4. The system sets a flag to record that the transaction was a refund.
  5. End of use case.

4.3.4 Update Customer Info

- **Brief Description**
  - The purpose of this use-case is for the system to update the customer information based on the input from the POS
- **Actors**

  POS, Customer Info Database
- **Basic Flow of Events:**

1. This use case begins with the system receiving input from the POS.
2. The system receives the customer name and retrieves the customer info from the customer database. If it is a new customer, then the system adds a new entry to the database.
3. The system records the customer's name, address and telephone number into the database.
4. If the customer has rented a book it will also be recorded along with the due date of the book(s) rented.
5. End of use case

- **Alternate Flow of Events #1: The system does not find a customer in the customer database - adds a new entry. This use case begins with the system receiving input from the POS.**

1. This use case begins with the system receiving input from the POS.
2. The system receives the customer name, telephone number, address and searches the customer database for the given customer.
3. The system does not find the customer.
4. The system creates a new entry and records the customer details into the database.
5. This use case ends.

- **Alternate Flow of Events #2: The system finds a match for a customer by name, but other details are different. Customer responds verifies information. The system updates the customer info.**

1. This use case begins with the system receiving input from the POS.
2. The system receives the customer name, telephone number and address. The system searches the customer database by name and finds a match. Other details for that customer are different.
3. The system sends a prompt to the POS system to determine if the customer's details have changed.
4. If the POS returns yes, then the details are updated.
5. The use-case ends

- **Alternate Flow of Events #3: The system finds a match for a customer by name, but other details are different. Customer is unable to verify the information. New entry is added to the database.**

1. This use case begins with the system receiving input from the POS.
2. The system receives the customer name, telephone number and address. The system searches the customer database by name and finds a match. Other details for that customer are different.
3. The system sends a prompt to the POS system to determine if the customer's details have changed.
4. The POS returns no, a new entry is added into the customer info database.
5. This use case ends.

4.3.5 Update Quantity

- **Brief Description**
  - The purpose of this use-case is for the system to increase or decrease the number of books in the book database
- **Actors**

  POS, Book Inventory Database
- **Basic Flow of Events:**

1. This use case begins with the system receiving input from the POS.
2. The system receives the ISBN number of the book sold and a code for refund or sale.
3. The system searches the book database (by ISBN) for the sold book.
4. If the code indicates that the book was sold it decrements the book quantity. If the code indicates a refund, it will increment the book count by one.
5. End of use case.

- **Alternate Flow of Events #1: Attempts to decrease the number of books, when quantity is zero.**

1. This use case begins with the system receiving input from the POS.
2. The system receives the ISBN number of the book sold and a code for refund or sale.
3. The system searches the book database for the specified book.
4. The quantity field is decremented and the book count becomes -1.
5. The system then prompts staff that the quantity is below 0.
6. This use case ends

4.3.6 Delete Book/Category

- **Brief Description**

  This use-case describes the process by which the system deletes a book record in the database. This use-case also describes the process if a manager wants to delete a category of books.

- **Actors**

  Managers, Employees

- **Basic Flow of Events: Delete book**

1. The use-case begins when a manager or employee utilizes the search function.
2. The employee or manager employs the search with any of the search choices listed by the system.
3. The employee or manager activates the search as long as they selected a search choice.
4. The system locates and highlights the desired book.
5. The employee or manager deletes the book by clicking delete on the system menu.
6. The system displays a message to the employee or manager to confirm the deletion transaction
7. The use-case ends.

- **Alternate Flow of Events #1: Delete book or category when there is nothing to delete**

1. The use-case begins when a manager or employee utilizes the search function.
2. The employee or manager employs the search with any of the search choices listed by the system.
3. The employee or manager activates the search as long as they selected a search choice.
4. The system displays an error message to the employee or manager that the book or category does not exist.
5. The employee or manager acknowledges this message.
6. The use-case ends.

- **Alternate Flow of Events #2: Delete Category**

This flow of events describes the steps if a manager wants to delete a category

1. The use-case begins when a manager wants to delete a book category.
2. The manager navigates to the category location that is to be deleted.
3. The manager highlights/selects a book category.
4. The manager selects delete on the system menu.
5. The system displays a message to the manager to confirm the deletion transaction.
6. The manager either confirms or denies the transaction.
7. The use-case ends.

- **Alternate Flow of Events #3: Delete Category with books in category**

  This flow of events describes the steps if a manager wants to delete a category that contains books

1. The use-case begins when a manager wants to delete a book category.
2. The manager highlights/selects a book category.
3. The manager selects delete on the system menu.
4. The system counts how many books reside in the category that the manager wants to delete.
5. The system displays a message of how many books are in the category.
6. The manager must acknowledge and confirm the amount of books in the category.
7. The system displays a message to the manager to confirm the deletion transaction.
8. The manager either confirms or denies the transaction.
9. The use-case ends.

- **Alternate Flow of Events #4: Delete Category with books and other categories in a category**

  This flow of events describes the steps if a manager wants to delete a category that contains books and other categories

1. The use-case begins when a manager wants to delete a book category.
2. The manager highlights/selects a book category.
3. The manager selects delete on the system menu.
4. The system detects if categories reside in the category that the manager wants to delete.
5. The system displays a message that informs the manager that other book categories exist in the category the manager wants to delete.
6. The manager must acknowledge this message from the system.
7. The system displays "Please try again" message when the other categories move to a different location or have been deleted.
8. The manager acknowledges the message.
9. The use-case ends.

4.3.7 Add Book/Category

- **Brief Description**
    - The purpose of the add books/categories . This ensures that employees sell books that customers need for their classes. The system grants extra privileges to managers where they can create categories for any book in the database. This use-case describes the process of adding books or categories.

- **Actors**

  Managers, Employees

- **Basic Flow of Events: Add book**

1. The use-case begins when either a manager or employee choose to add a book.
2. The employee or manager scan the ISBN.
3. The system fetches all pertinent information to populate our database.
4. The system inserts the information from the Library of Congress into the database.
5. The system has added the scanned book into the database.
6. The use-case ends.

- **Alternate Flow of Events #1: Add category**

  This flow of events describes the steps taken if a manager wants to add a category:

1. The use-case begins when a manager chooses to add a book category.
2. The manager navigates to the location where the category is to be added.
3. The system displays a dialog box for the new category.
4. The manager types the book category name.
5. The system verifies that a duplicate category does not exist in the destined location.
6. The system informs the manager that the addition of a book category succeeded.
7. The use-case ends.

- **Alternate Flow of Events #2: Add book, but ISBN is not found**

  This flow of events describes the steps taken if an employee or manager add a book, when the ISBN is not found from the Library of Congress:

1. The use-case begins when an employee or manager chooses to add a book.
2. The employee or manager scan the ISBN.
3. The system fails to locate the ISBN from the Library of Congress.

4. The system informs the employee or manager that the ISBN was not found and did not add this book to the database.
5. The employee or manager acknowledges this message.
6. The employee or manager gathers all required information for the database from the book.
7. The employee or manager inputs all required data into the database.
8. The system informs the employee or manager that the book was added to the database.
9. The employee or manager acknowledges the message.
10. The use-case ends

4.3.8 Search Book

- **Brief Description**
  - This use-case describes the process by which the system look thru the book database and find a list of books that fulfilled the parameters given. (Authors, ISBN, subjects, etc…)
- **Actors**

  Book database, customer, managers, employee, vendor, website
- **Dependencies**

  List Result.
- **Basic Flow of Events: Search for book/s, found the book/s**

1. The use-case begins when an actor requests for a search of book(s).
2. The use-case prompts the user for a search parameter
   - **i.** Search by ISBN
   - **ii.** Search by Authors
   - **iii.** Search by publishers
   - **iv.** Search by Subjects
   - **v.** Search by Etc…
3. The use-case prompts the user for appropriate information based on the choice of method (author name, department name and/or course number, etc...)
4. The user presses “search book” on screen.
   - **i.** The use-case processes the provided information for a search.
   - **ii.** The use-case displays a summary of requested parameters for finding the book/s
5. The use-case generates and returns the list from the found books using “List Result” use-case.
6. The use-case ends.

- **Alternate Flow of Events #1: Search for book/s, did not find specific book/s**

This flow of events describes the process of making changes to previous selection. It follows the basic flow of events up to step three (3)

1. The use-case begins when an actor requests for a search of book(s).
2. The use-case prompts the user for a search parameter;
   - **i.** Search by ISBN
   - **ii.** Search by Authors
   - **iii.** Search by publishers
   - **iv.** Search by Subjects
   - **v.** Search by Etc…

3. The use-case prompts the user for appropriate information based on the choice of method (author name, department name and/or course number, etc...)
4. The use-case attempts to process the provided information
5. The use-case displays an appropriate message indicating the provided information is incorrect and what information is required to continue with the search.
6. The use-case restarts the process from step three;
    - The use-case can restart from step one if the user wishes to change the search parameter
7. The use-case ends.

4.3.9 Login

- **Brief Description**
    - This use-case describes the process of verifying identity to access a certain feature. In this case, it is being used to limit access of managing bookstore book prices to only managers and vendor book prices to only vendors.

- **Actors**

  Managers, Vendors, Employees

- **Dependencies**

  Add book, Delete book, Update quantity, update price, sales/refund, request sales report, request inventory report (Pretty much everything that request you to login before hand)

- **Basic Flow of Events: Login, No error**

1. This use-case begins when an actor attempts to manage book prices.
2. The use-case prompts the user for a login and password information.
3. The use-case processes this information after the user presses "enter".
4. The use-case verifies the login information provided is correct.
5. The use-case gives access either to the bookstore prices or the vendor prices depending on the login information.
6. The use-case ends.

- **Alternate Flow of Events #1: Login, incorrect password/username**

  This flow of events describes the process of making changes to previous selection. It follows the basic flow of events up to step three (3)

1. This use-case begins when an actor attempts to manage book prices.
2. The use-case prompts the user for a login and password information.
3. The use-case processes this information after the user presses "enter".
4. The use-case attempts to verify the provided login information is correct.
5. The use-case displays an appropriate message indicating the provided information is incorrect and how many attempts left.
6. The use-case restarts the process from step one if the number of attempts left is not zero.
7. The use-case locks access to this feature if number of tries left is zero.
8. The use-case ends.

4.3.10 Low/Over/Out of Stock Alert

- **Brief Description**
  - This use-case describes the how alerts are handled when book-store items are in a low stock, out of stock and over stock alert state.
- **Actors**

  Book-store System
- **Dependencies**

  Daily Report
- **Basic Flow of Events: Low/Over/Out of Stock Alert occurs**

1. The use-case begins automatically at the end of the day when the Book-store system checks the book database searching for three alert types which are low stock, out of stock, or overstock books.
2. For each low/out of/overstock book found the system writes into the daily report the books information such as book name, ISBN, alert type, and quantity.
3. The system creates an alert message that will be seen on next Manager Login that the daily report has new alerts.
4. The use-case ends.

- **Alternate Flow of Events #1: Low/Over/Out of Stock Alert does not occur.**

1. The use-case begins automatically at the end of the day when the Book-store system checks the book database searching for three alert types which are low stock, out of stock, or overstock books.
2. No alerts are found within the book database.
3. The system writes into the daily report that no alerts were found.
4. The use-case ends.

5. Logical View

5.1 Overview

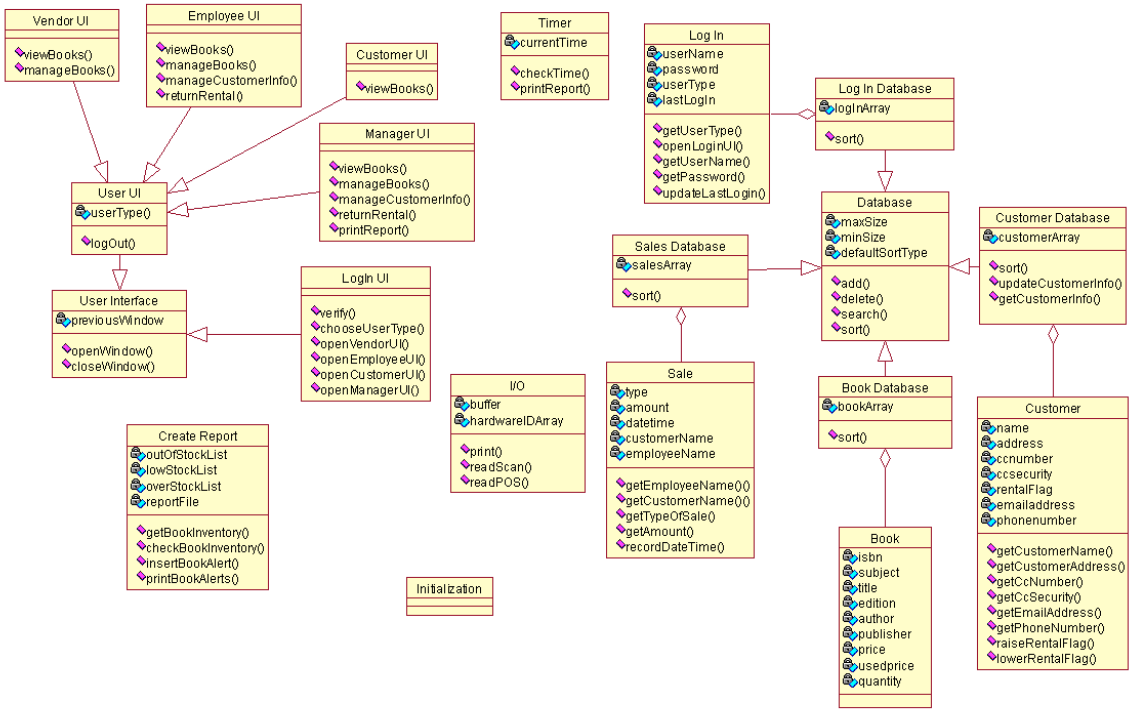

Figure 8 - System Class Diagram

5.2 Architecturally Significant Design Packages

5.2.1 ManagerUI

- **Brief Description**

  Boundary – Manager UI provides an interface for a manager to interact with the system.

- **Methods**

Access	Return	Name	Description
Public	Void	viewBooks()	This method is the user interface that comes up when a manager needs to view books
Public	Void	manageBooks()	This method is the user interface that comes up when a manager needs to manage books
Public	Void	manageCustomerInfo()	This method is the user interface that comes up when a manager needs to manage a customer information
Public	Integer	returnRental()	This method is the user interface that comes up when a manager needs to update a customer's book rental status
Public	Void	printReport()	This method is the user interface that comes up when a manager needs to request a report to be printed

5.2.2 LoginUI

- **Brief Description**

  Boundary – Login UI provides an interface for every user to log in to the system.

- **Methods**

Access	Return	Name	Description
Private	Void	submit()	This method verifies the login information
Private	Void	chooseUserType()	This method is used to determine what type of user needs to login
Public	Void	openVendor()	This method is the user interface that comes up when a vendor needs to login
Public	Void	openEmployeeUI()	This method is the user interface that comes up when an employee needs to login
Public	Void	openCustomerUI()	This method is the user interface that comes up when a customer needs to login
Public	Void	openManagerUI()	This method is the user interface that comes up when a manager needs to login

5.2.3 User Interface

- **Brief Description**

  Control – This class provides functionality to its subclasses to open, close and remember the previous window.

- **Methods**

Access	Return	Name	Description
Public	Void	previousWindow()	This attribute holds the previous window that was opened before the current one
Public	Void	openWindow()	This method is used open a new user interface window
Public	Void	closeWindow()	This method is used close a currently opened user interface window

5.2.4 Timer

- **Brief Description**

  Control – This class generates reports automatically.

- **Attributes**

Type	Access	Name	Description
Integer	Private	Timer	An internal timer that holds the current time in seconds.

- **Methods**

Access	Return	Name	Description
Private	Boolean	CheckTime()	A timer event that activates periodically that checks if the current time matches the time to create a daily report.
Private	Void	StartCreateReport()	Once the timer event is true this method wakes up the processes in the class CreateReport()

5.2.5 Create Report

- **Brief Description**

  Control – This class generates automatic reports on a daily or specified basis. Its output is passed to I/O for printing.

- **Attributes**

Type	Access	Name	Description
BookList	Private	OutOfStockList	List of Books which are out of stock
BookList	Private	LowStockList	List of Books which are over stocked
BookList	Private	OverStockList	List of Books which are low stocked
File	Private	ReportFile	A file which contains all the alerts that is used to send over to IO

- **Methods**

Access	Return	Name	Description
Private	List array	getBookInventory()	A timer event that activates periodically that checks if the current time matches the time to create a daily report.
Private	void	checkBookInventory()	Checks each book for low stock, out of stock, or over stock alerts. If a book is in a alert state checkBookInventory() calls insertBookAlert().
Private	Void	insertBookAlert()	Inserts a book into one of three lists : Low-stock books, Out-of Stock books and Over-Stocked books.
Private	Boolean	PrintBookAlerts()	Creates a file with the three alerts lists and sends it over to the IO device for printing.

5.2.6 IO

- **Brief Description**

  Control – This class provides the system with the functionality of printing reports and showing data to a screen.

- **Attributes**

Type	Access	Name	Description

Void	Private	Print Buffer	Holds the files sent for printing

- **Methods**

Access	Return	Name	Description
Private	Boolean	PrintToScreen()	Prints the file on to screen specified in the parameter
Private	Void	PrintFile()	Prints the file specified in the parameter

5.2.7 Sales Database

- **Brief Description**

  Entity - Class that contains all necessary attributes and methods associated with bookstores sales.

- **Attributes**

Type	Access	Name	Description
sales	Public	salesArray	the array that contains all records of type sales

- **Methods**

Access	Return	Name	Description
Public	Void	sort()	inherits from parent class 'database' but focuses the sort based off any of the fields associated with of type 'sale'

5.2.8 Sale

- **Brief Description**

  Entity - Class that stores all information associated with a bookstore transaction

- **Attributes**

Type	Access	Name	Description
Boolean array	Private	Type	a boolean array where the value corresponds to either a sale or refund
float	Private	Amount	contains the total amount of the purchase
String	Private	Datatime	contains the date and time of when the transaction took place
String	Private	Customer name	contains the first and last name of a customer who made the purchase
String	Private	Employee name	contains the first and last name of the employee who conducted the purchase Methods: NA

- **Methods**

Access	Return	Name	Description
Public	Void	getEmployeeName()	This method gets the employee's name handling the sale transaction from POS
Public	Void	getCustomerName()	This method gets the purchasing customer's name in the sale transaction from POS
Public	Void	getType()	This method gets the type of transaction taking place from POS
Public	Void	getAmount()	This method gets the total amount charged in that transaction from POS
Private	void	recordDateTime()	This method records the date and time of the transaction from the system timer

5.2.9 Database

- **Brief Description**

  Entity - The generic class for all databases. Contains all attributes and methods that every database shares or inherits.

- **Attributes**

Type	Access	Name	Description
Integer	Public	Max size	An integer value that stores the current number of entries within the database.
Integer	Public	Min size	An integer value that stores the minimum number of entries possible within the database(normally set to 0, but any number of 0 denotes default database entries that are not counted).
string	Public	Default sort type	A string value that stores the current sort type that the database is sorted by.

- **Methods**

Access	Return	Name	Description
Public	Void	Add()	A void method that takes in an entry and inserts it into the database in the appropriate index based on the current default sort type.
Public	Void	Delete()	A void method that takes in a value and deletes the appropriate entry with that value based on the current default sort type.
Public	Void	Search()	An integer method that returns the index of the entry that contains the string value given as a parameter based on the current default sort type.
Public	void	Sort()	A void method that performs a Quick Sort based on the parameter given, which is the sort type performed. The Sort() function then sets the default sort type to the type of sort that was just performed.

5.2.10 Book Database

- **Brief Description**

  Entity - A class for the creation of a database that stores values of type "Book."

- **Attributes**

Type	Access	Name	Description
book	Public	bookArray	The actual array that contains values of type "Book."

- **Methods**

Access	Return	Name	Description
Public	void	sort()	Inherits from the superclass "Database." Only allows for the array to be sorted based on the appropriate field types that "Books" can have.

5.2.11 Book

- **Brief Description**

  Entity - A class for storing all the pertinent information contained within each unique book.

- **Attributes**

Type	Access	Name	Description
Int	Private	ISBN	The ISBN of the book (constant).
String	Private	Subject	Contains information about what class this book is used in (for example, CS 472, would be stored for any textbook used in the class CS 472) (variable).
String	Private	Title	The full title of the book (constant).
Float	Private	Edition	The edition of the book (constant).
String	Private	Author	The author of the book (constant).
String	Private	Publisher	The publisher of the book (constant).
Float	Private	Price	The current price of the book (variable).
Float	Private	UsedPrice	The current price for a used version of this book (variable).
Int	Private	Quantity	The current number of this book in stock (variable).

5.2.12 Login

- **Brief Description**

  Control – This class provides a user interface for users to log in to the system.

- **Attributes**

Type	Access	Name	Description
String	Private	userName	This attribute is the login name of the user
String	Private	Password	This attribute is the login password of the user
String	Private	userType	This attribute is the type of the user
int	Private	lastLogin	This attribute stores the last recorded login time and date

- **Methods**

Access	Return	Name	Description
Public	void	getUserType()	This method gets the type of user from user through a user interface window
Public	void	openLoginUI()	This method opens the appropriate loginUI
Public	void	getUserName()	This method gets the user login name through the login UI
Public	void	getUserPassword()	This method gets the user password through the login UI
Public	void	updateLastLogin()	This method updates the user's last login record with the current login.

5.2.13 Login Database

- **Brief Description**

  Entity – This class stores all username and passwords for users that have access to the system.

- **Attributes**

Type	Access	Name	Description
String	Public	logInArray	An array of information required for login

- **Methods**

Access	Return	Name	Description
Public	void	Sort()	This method can sort all the usernames and passwords in specific parameters such as Type, Last name, privilege,

5.2.14 Customer

- **Brief Description**

Control – This class provides functionality to the system to store customer information in the database.

- **Attributes**

Type	Access	Name	Description
String	Private	Name	This attribute is the customer 's name
String	Private	Address	This attribute is the customer's address
String	Private	Ccnumber	This attribute is the customer 's credit card number
String	Private	Ccsecruity	This attribute is the customer 's credit card security code
Boolean	Private	RentalFlag	This attribute is the a flag that indicates if the customer has a rental book or not
String	Private	Emailaddress	This attribute is the customer 's email address
string	Private	phonenumber	This attribute is the customer 's phone number

- **Methods**

Access	Return	Name	Description
Public	Void	getName()	This method gets the name of the customer
Public	Void	getAddress()	This method gets the customer’s address
Public	Void	getCCNumber()	This method gets the customer’s credit card number
Public	Void	getCCSecurity()	This method gets the customer’s credit card security code
Public	Void	getEmailAddress()	This method gets the customer’s email address
Public	Void	getPhoneNumber()	This method gets the customer’s phone number
Public	Void	raiseRentalFlag()	This method raises the rental flag when the customer rents a book
Public	Void	lowerRentalFlag()	This method lowers the rental flag when the customer returns a rented book

5.2.15 Customer Database

- **Brief Description**

Entity – This class stores all the information from each customer.

- **Attributes**

Type	Access	Name	Description
string	Public	customerArray	This array holds all the customer information.

- **Methods**

Access	Return	Name	Description
Public	Void	Sort()	This method can sort all the username and password in specific parameters such as Type, Last name, privilege,
Public	Void	getCustomer()	This method gets a new customer's information and stores it in the customer database
Public	Void	updateCustomer()	This method updates an existing customer's information

6. Risk Rank of Classes

Login Database	High
Sales Database	
Customer Database	
Book Database	
User UI	
Log In	
Create Report	Low

7. Size and Performance

- The Bookstore Inventory System must perform all functions with minimal time delays.
- The system must also accurately save all information transactions.
-

8. Quality

- In order to maintain the highest degree of system integrity our system will ensure that all information transactions are saved.
- Backup of all databases will occur on a daily basis during minimum activity hours.
- The system will allow users to print out receipts and transaction history for a specified time period.