A1

CS230 Group 47

Tawe-Lib Design Document

Contents

Section 1 – Introduction

1.1 Purpose

The purpose of this document is to describe the implementation of the Tawe-Lib Specification given to us in assignment one of CS230. The Tawe-Lib software is a library management system.

1.2 Scope

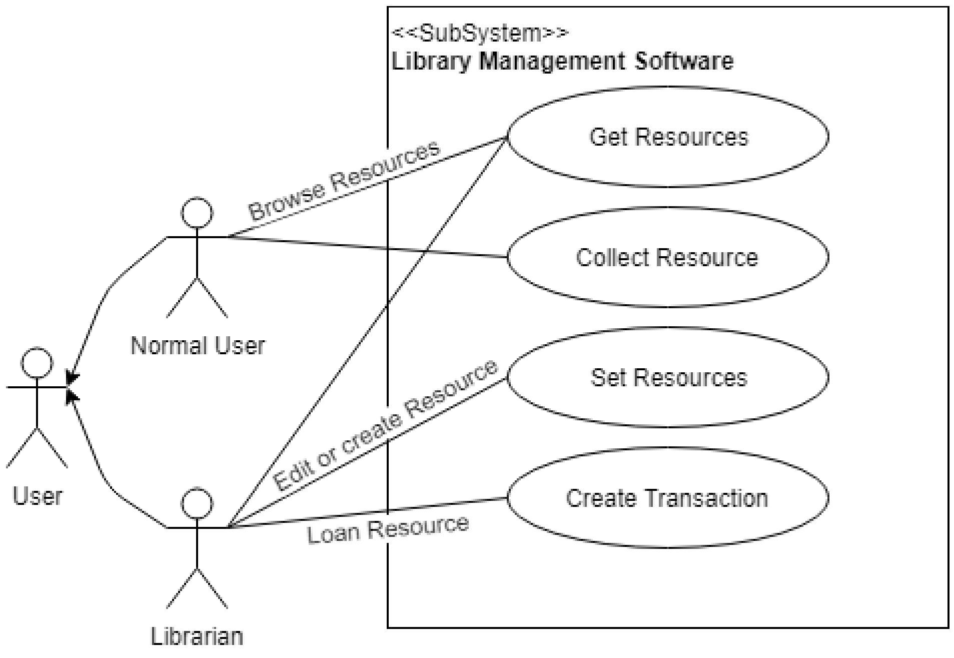
The Tawe-Lib software is a rudimental library system and is designed to keep track of all library resources, the check-in and check-out of these resources, fines for overdue loans, and multiple copies. All managed through a simple GUI with two distinct types of user.

1.3 Design Overview

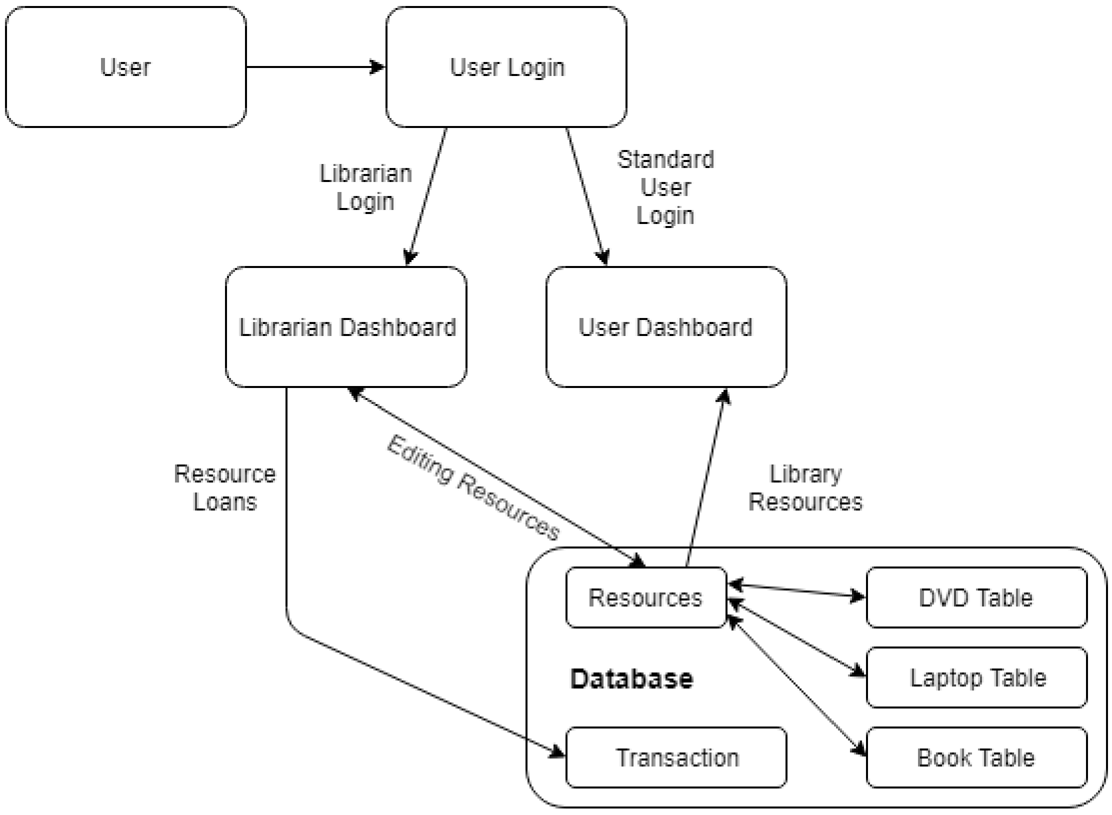
1.3.1 Description of problem

Users of the library must be able to browse all available library resources, get information on a resource and be able to check-out the resource if a copy of it is available. Or be Queued until one is available. Librarians must be able to add or remove resources or copies from the library and facilitate the payment of fines alongside the check-in and check-out of resources by users.

1.3.2 Use Case Diagram



1.3.3 Architecture



1.3.4 Statistics

Users are capable of seeing a set of statistics about how many items they borrow per week, per month and per year. The data will be available through the getHistory() method which pulls the dates of user's borrowed books from the Transactions table. The data will be displayed both textually and graphically on user's choice. This section will be implemented and illustrated in A2.

Section 2 - Candidate Classes and Responsibilities

2.1

Resources	
Super Class: None **Sub Classes:** DVD, Book, Laptop	
Responsibilities • Store unique resource ID • Store year • Store title • Store thumbnail image path • Get unique resource ID • Get year • Get title • Get thumbnail Image path • Set thumbnail Image path • Edit Resource	**Collaborations** • DVD • Book • Laptop • Copies
Rough Description: Abstract class which holds data which the subclasses have in common, along with abstract operations which can be used by all the subclasses.	

2.2

Book	
Super Class: Resources **Sub Classes:** None	
Responsibilities • Create new book instance • Stores author • Stores publisher • Stores isbn • Stores genre • Stores language • Get author • Get publisher • Get ISBN number • Get genre • Get language	**Collaborations** • Resource • Copies
Rough Description: A subclass of the superclass Resources, which holds the data and methods relevant to only book and no other Resources subclasses. This class models the physical object that is a real book owned by Tawe-Lib.	

2.3

Laptop	
Super Class: Resources **Sub Classes:** None	
Responsibilities • Create a new Laptop instance • Store operating system • Store manufacturer • Store model • Get operating system • Get manufacturer • Get model	**Collaborations** • Resources • Copies
Rough Description: A subclass of the superclass Resources, which holds the data and methods relevant to only laptop and no other Resources subclasses. This class models the physical object of a real laptop owned by Tawe-Lib.	

2.4

DVD	
Super Class: Resources **Sub Classes:** None	
Responsibilities • Create a new DVD instance • Store director • Store runtime • Store language • Store subtitle language • Get Director • Get Runtime • Get Language • Get Subtitle language	**Collaborations** • Resources • Copies
Rough Description: A subclass of the superclass Resources, which holds the data and methods relevant to only DVD and no other Resources subclasses. This class models the physical object of a real DVD owned by Tawe-Lib.	

2.5

Copies	
Super Class: None **Sub Classes:** None	
Responsibilities • Stores a unique copy ID • Stores the associated unique resource ID • Stores a loan duration • Create a new instance of Copies • Get unique copy ID • Get unique resource ID • Get issued to • Get issued date • Get issued by • Set issued to • Return copy • Request item • Collect item • Housekeeping	**Collaborations** • User • Book • Laptop • DVD
Rough Description: This class is standalone and doesn't have any super/sub classes. Instead, it references other objects using a foreign key. A librarian can issue a copy to a User which requires them to specify which resource (i.e. type of book/DVD/laptop) through a foreign key for that object.	

2.6

User	
Super Class: None **Sub Classes:** Librarian, User	
Responsibilities • Stores current account balance • Create a new instance of user • Get balance • Set Balance • Pay fines	**Collaborations** • AccountBaseUser
Rough Description: A subclass of AccountBaseUser holding the attributes and operations unique to the User class, this class models the intended normal user of the system.	

2.7

Librarian	
Super Class: Account **Sub Classes:** None	
Responsibilities • Stores employment date • Stores a unique staff ID number • Can create an instance of librarian • Get employment date • Get unique staff ID • Get history of transactions • Get list of overdue resources	**Collaborations** • AccountBaseUser
Rough Description: This class models the characteristics of a valid librarian's account within the Tawe-Lib. This is a subclass of AccountBaseUser	

2.8

AccountBaseUser	
Super Class: User, Librarian **Sub Classes:** None	
Responsibilities • Stores unique username • Stores first name • Stores last name • Stores telephone number • Stores address • Stores a path to a profile image • Get unique username • Get first name • Get last name • Get telephone number • Get address • Get profile image • Set address • Set telephone number • Set first name • Set last name • Choose profile image • Draw profile image	**Collaborations** • User • Librarian
Rough Description: This abstract class is used to store attributes and abstract methods used within both of its subclasses: User and Librarian.	

Section 3 – Descriptions

3.1 Hierarchy Descriptions

Within the class diagram, shown on 3.2, there are two inheritance hierarchies involved. The first hierarchy is the resources hierarchy, which generalizes the shared attributes of the various resources. Laptops, Books, and DVDs are all a kind of resource. As such, the subclasses specialize the attributes and behaviours of the different types of resources, including: Laptops, Books, and DVDs. As indicated in the class diagram, Resources is an abstract class containing multiple abstract methods. With the use of abstract methods in the superclass, each of the subclasses can have their own methods of implementation. The second hierarchy involved with the class diagram is the accounts hierarchy. Here the attributes of an account are generalized in the abstract class called AccountBaseUser. User and Librarian are both a kind of account. The specialized attributes and behaviours of different types of accounts, users and librarians, are then introduced within subclasses User and Librarian.

3.2 Collaborations Descriptions

In our method of implementing the design, we decided to make associations between classes. As a result, there are no aggregation or composition relationships demonstrated within our class diagrams.

Users are capable of viewing and requesting copies via their interface, while they browse through the library's resources. This search will query the database for all the available copies and present the user with the details of his request.

Librarians are capable of creating and editing resources and copies through their interface. They can also make changes to both Resources and all Copies databases. Those changes will be done by querying the Copies or Resources tables and submitting the required information.

3.3 Operation Descriptions

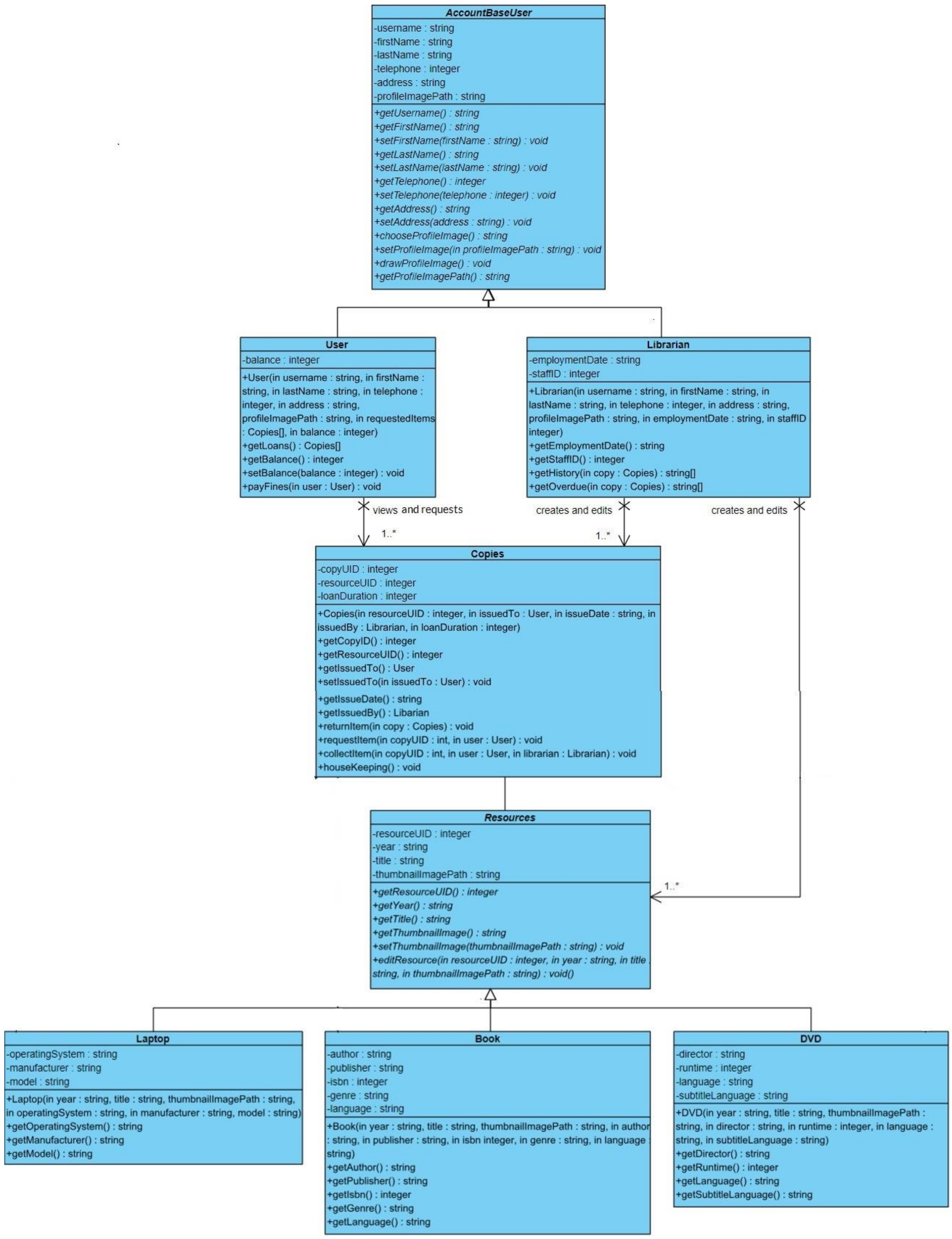
Operation descriptions, as they apply mainly to the database can be found in section 5.2.

Section 4 – UML Class Diagram

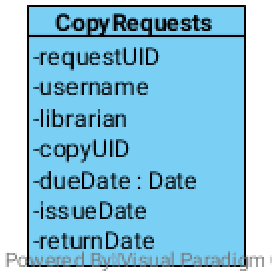
4.1 Summary

Below is the proposed structural class diagram for the Tawe-lib software solution. This diagram shows the proposed classes with their respective relations, attributes and methods. This is only the proposed structure and is likely to be altered minimally within the implementation. An example of this would be the sates of the methods and attributes, in respect of their visibility and whether they are static or not. These two properties are currently shown as all are public and non-static, this is likely to change during the implementation stage.

4.2 Diagram


AccountBaseUser
-username : string
-firstName : string
-lastName : string
-telephone : integer
-address : string
-profileImagePath : string
+getUsername() : string
+getFirstName() : string
+setFirstName(firstName : string) : void
+getLastName() : string
+setLastName(lastName : string) : void
+getTelephone() : integer
+setTelephone(telephone : integer) : void
+getAddress() : string
+setAddress(address : string) : void
+chooseProfileImage() : string
+setProfileImage(in profileImagePath : string) : void
+drawProfileImage() : void
+getProfileImagePath() : string
User
-balance : integer
+User(in username : string, in firstName : string, in lastName : string, in telephone : integer, in address : string, profileImagePath : string, in requestedItems : Copies[], in balance : integer)
+getLoans() : Copies[]
+getBalance() : integer
+setBalance(balance : integer) : void
+payFines(in user : User) : void
Librarian
-employmentDate : string
-staffID : integer
+Librarian(in username : string, in firstName : string, in lastName : string, in telephone : integer, in address : string, profileImagePath : string, in employmentDate : string, in staffID integer)
+getEmploymentDate() : string
+getStaffID() : integer
+getHistory(in copy : Copies) : string[]
+getOverdue(in copy : Copies) : string[]
views and requests
1..*
creates and edits
1..*
creates and edits
Copies
-copyUID : integer
-resourceUID : integer
-loanDuration : integer
+Copies(in resourceUID : integer, in issuedTo : User, in issueDate : string, in issuedBy : Librarian, in loanDuration : integer)
+getCopyID() : integer
+getResourceUID() : integer
+getIssuedTo() : User
+setIssuedTo(in issuedTo : User) : void
+getIssueDate() : string
+getIssuedBy() : Libarian
+returnItem(in copy : Copies) : void
+requestItem(in copyUID : int, in user : User) : void
+collectItem(in copyUID : int, in user : User, in librarian : Librarian) : void
+houseKeeping() : void
1..*
Resources
-resourceUID : integer
-year : string
-title : string
-thumbnailImagePath : string
+getResourceUID() : integer
+getYear() : string
+getTitle() : string
+getThumbnailImage() : string
+setThumbnailImage(thumbnailImagePath : string) : void
+editResource(in resourceUID : integer, in year : string, in title
string, in thumbnailImagePath : string) : void()
Laptop
-operatingSystem : string
-manufacturer : string
-model : string
+Laptop(in year : string, title : string, thumbnailImagePath : string, in operatingSystem : string, in manufacturer : string, model : string)
+getOperatingSystem() : string
+getManufacturer() : string
+getModel() : string
Book
-author : string
-publisher : string
-isbn : integer
-genre : string
-language : string
+Book(in year : string, title : string, thumbnailImagePath : string, in author : string, in publisher : string, in isbn integer, in genre : string, in language : string)
+getAuthor() : string
+getPublisher() : string
+getIsbn() : integer
+getGenre() : string
+getLanguage() : string
DVD
-director : string
-runtime : integer
-language : string
-subtitleLanguage : string
+DVD(in year : string, title : string, thumbnailImagePath : string, in director : string, in runtime : integer, in language : string, in subtitleLanguage : string)
+getDirector() : string
+getRuntime() : integer
+getLanguage() : string
+getSubtitleLanguage() : string

Section 5 – Database Design



5.1 Interpretation

Each class has it's own table in the database. Each resource type's table (i.e. Book, Laptop, DVD) is referentially linked to the master Resource table through the resourceUID primary key. The same principle applies for the BaseUser class with the standard User and Librarian. However, things get different when we start to work with Copies. Each copy is stored in the Copy table but all the requests for these are stored in a CopyRequests table not represented as a class. Every request for a copy is stored here and is filtered using the copyUID to list all requests for a copy, or the username to find all historical transactions for a user. This also allows the librarian to search for all overdue items with ease as only one table is being searched.

5.2 Engineering (Complex Operations)

- Method returnItem(copy:Copies):void from Class Copies:

The purpose of this method is to update the status of a copy record. It will run within the Copies class and access the ResourceRequests table.

The method searches the ResourceRequests table for the records associated with the unique copyID. CopyID will be available from the parameter copy (of type Copies) by the public/protected method getCopyID().It will select the last record with an issueDate set and update it's returnDate field to today's date. A fine will be issued if necessary. If the view set has a "next record" (meaning there is a pending request), set the next record's dueDate to today.

- Method requestItem(copyID:integer, user:User):void from Class Copies:

The purpose of this method is to update the status of a copy record. It will run within the Copies class and access the ResourceRequests table.

The method searches the ResourceRequests table for the records associated with the unique copyID. If the last record in the view set has the returnDate set, create a new record and set it's issueDate as today and dueDate as null. If the last record in the view set doesn't have a dueDate set, update it to the minimum length specified by the copies duration attribute.

- Method collectItem(copyID:integer, user:User):void from Class Copies:

The purpose of this method is to update the status of a copy record. It will run within the Copies class and access the ResourceRequests table.

The method searches the ResourceRequests table for the records associated with the unique copyID. If the last record in the view set has not passed it's dueDate and the requesting user is the correct one, set the issueDate as today. Moreover, if the view set has a "next record" (meaning there is a pending request), set the dueDate to the minimum loan duration.

- Method housekeeping():void from Class Copies:

The purpose of this method is to update the status of an uncollected copy. It will constantly run within the Copies class and access the ResourceRequests table.

The method keeps track of all the due dates of all the copies, by comparing their dueDate attribute with today's date. If the dueDate has expired(is smaller than today's date) and there is no issueDate (as it is therefore an expired collection). Before it selects the next request record (if there is one) and set it's dueDate to today.

- Abstract Method editResource (resourceUID:integer, year: string, title string, thumbnailImagePath: string): void from Class Resources:

The purpose of this method is to allow a librarian to edit a currently existing resource in the database. It will get implemented in the classes Laptop, Book and DVD. The method accesses the respective resource database table and changes the required attribute values to the ones entered by the librarian. Only the librarian has access to this method through the edit button available in his interface.

5.3 Strategy and Implementation

As can be seen above, there are a couple issues with this database schema. One such issue is that all the data is stored in one long table when it could be split based as one table per copy. For ease of development, we'll stick to this design. We have to use a lot of SQL logic to get the data we want, however in the real world this won't cause a significant slowdown of the application. The biggest issue however is that we require a housekeeping function. When an item is returned or requested it automatically sets the collection dates and due dates, however if no one collects the item the system can't automatically expire it as no function is being run. In an ideal world, we would have this running as an asynchronous operation constantly looking for expired collections and then remove that record. In this case, we run a task at midnight to clear the uncollected requests and set the next request ready to collect. This could be done through a Cron Job on the host computer.

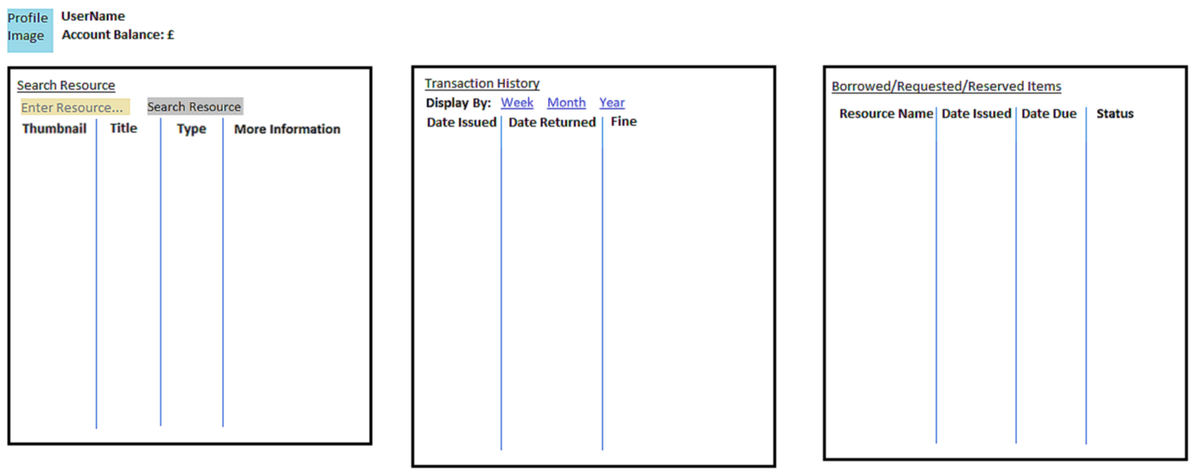
The software we will use to host the database will be SQLite3. We chose this over other options such as MariaDB, MongoDB, MySQL and others simply because of how lightweight it is as well as the integrations it offers for many IDE's and languages. We can install the relevant library and spin it up within the program itself without requiring a separate service to host the database. We recognise that SQLite does have a disadvantage in that it's slower compared to the alternatives and can't handle as much data. Although in this case that doesn't matter as we won't be dealing with hundreds-of-thousands of records. It does have the issue whereby if the program crashes, so does the database – although in all the tests I've seen the data has been safe.

Section 6 – UI Mock-ups

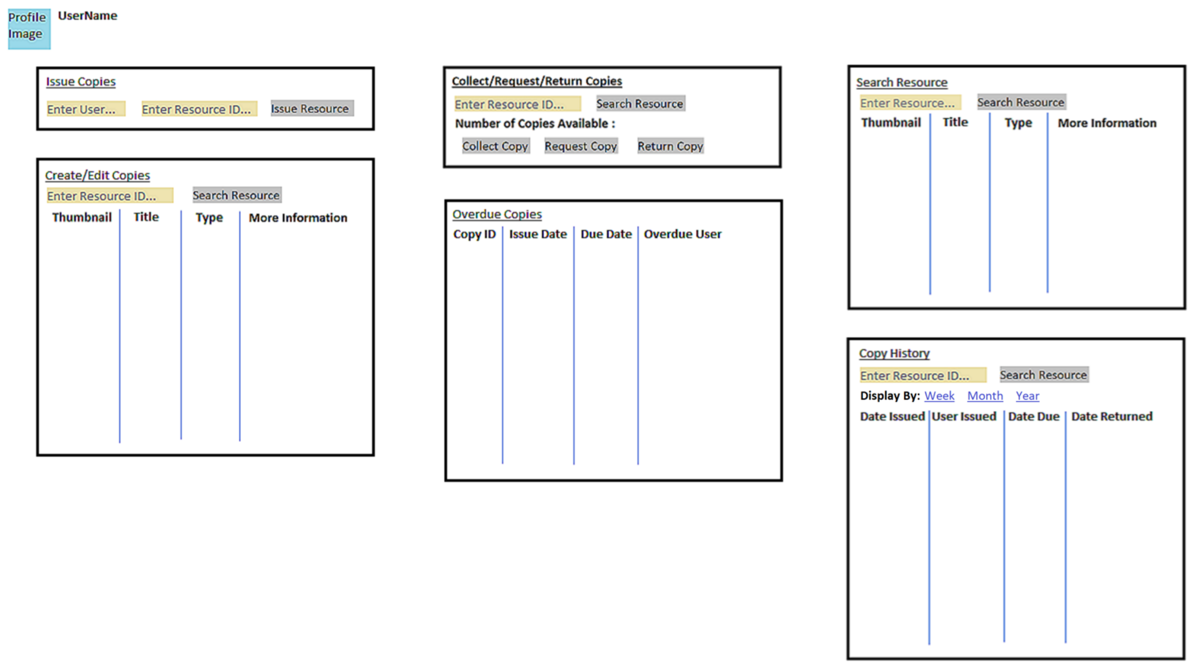
6.1 Summary

Some rudimentary GUI mock-ups to show a basic layout of how the dashboards should work for each user. These two interfaces minus a few secondary interfaces are the only way a user should interface with the system. Users can browse all Resources, their transaction history and statistics. Librarians can browse, edit or create resources, and administer resource collections, loans or overdue copies.

6.2 User Interface



6.3 Librarian Interface


Profile Image
UserName
Issue Copies
Enter User...
Enter Resource ID...
Issue Resource
Create/Edit Copies
Enter Resource ID...
Search Resource
Thumbnail
Title
Type
More Information
Collect/Request/Return Copies
Enter Resource ID...
Search Resource
Number of Copies Available :
Collect Copy
Request Copy
Return Copy
Overdue Copies
Copy ID
Issue Date
Due Date
Overdue User
Search Resource
Enter Resource...
Search Resource
Thumbnail
Title
Type
More Information
Copy History
Enter Resource ID...
Search Resource
Display By: Week Month Year
Date Issued
User Issued
Date Due
Date Returned