ELMS Project

Presented by

Group 4 of Tutorial 3

Background

- A university in Western Sydney is looking to develop an electronic library management system; they have hired our group to carry out this task
- The project "aims to enable remote borrowing of e-books and journals by users (students, lecturers and professors) without having to physically come to the library."

Team

Chantel Mills

- Junior full-stack developer

Mattias Przyrembel

- Project lead
- Senior full-stack developer

Pulkit Jain

- Full-stack developer

Mohammad Karim

- Full-stack developer

Umair Khalidi

- Junior back-end developer

Yize Ma

- Junior back-end developer

Scope and Requirements

- User login/registration: Users (students, staff) have to register themselves first with their university email ids; after registering successfully, they can then login into the system.
- Admin login: administers the system maintaining books in the catalogue; an administrator does not need to register themselves
- Add, update and remove e-books: can only be done by administrators who would maintain details such as category (book, journal, research paper or article) , title, author, reference number, image and other information specific to each item
- Search option: Admin, students and staff can search for books by different search parameters, e.g. category, name and/or author and can select one or more to 'borrow' provided all copies have not already been issued
- Calculation of fines: after issue and if an e-book is not returned by a set time, a fine becomes due and the student or staff member is notified by SMS and/or email
- Users can view books issued to them, their due dates and fines outstanding
- Users (and administrators) can renew or return books with or without fines depending upon how long they have held the book
- Staff can request introduction of new or essential books for the library by completing a book request form

Objectives

- Reduce the need for in-person borrowing of books
- Provide a searchable database for finding books
- Allow remote checks for book stock levels
- Improve the ease of tracking due dates
- Automate fines for exceeding due dates
- Allow staff to prescribe books for their students

Out of Scope

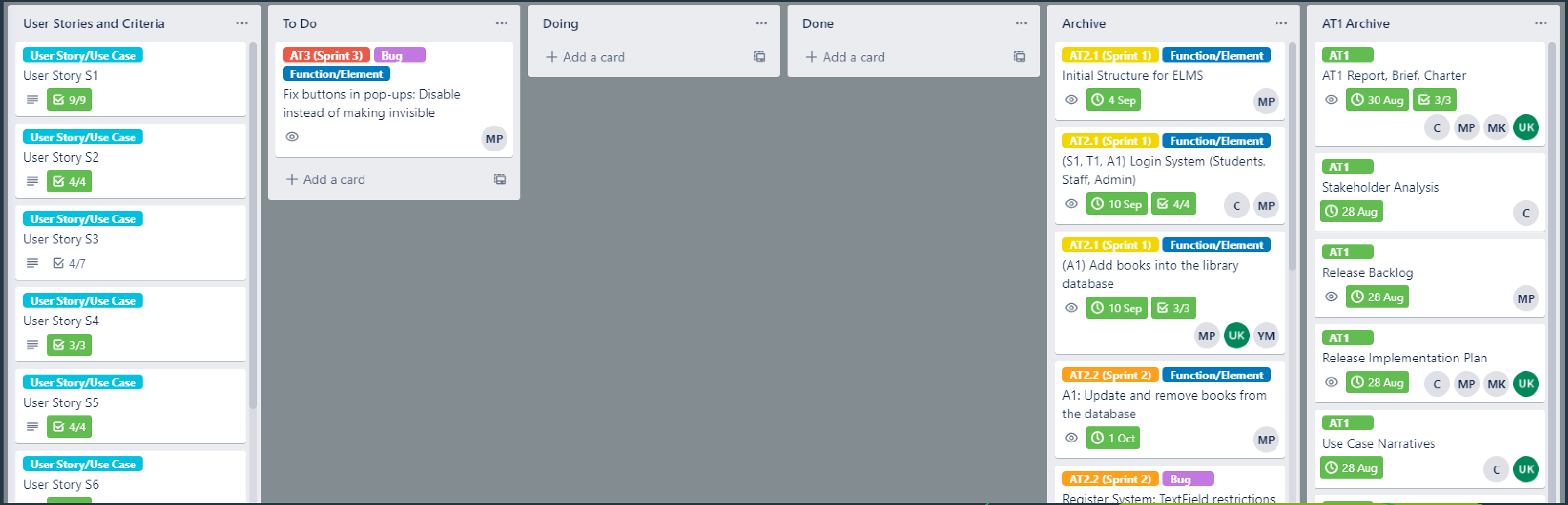
- Registration for logging into the system as the accounts will be connected to the client university's system - a small placeholder login database will be used in demonstrations
- Likewise, a secure credential handling system for the logins will also be handled by the client university
- The push-notification functionality for due dates and fines
- Establishing and connecting the official library database – a small placeholder database will be used in demonstrations
- Issuing of bills/fines electronically for overdue books – fines will be visible on the account, but users will be required to visit the library in-person to pay their fines

Assumptions

- The client university has an existing database of all staff and students that can be connected to the product upon completion
- Likewise, that the university has an existing database with all their books
- Library staff will be given administrator access in order to remove fines from users once they have been paid in-person
- All development and administrative control will be passed on to the library staff upon the project's completion

Project Development and Tracking

- We followed an agile framework with three sprints
- Trello board as of the 27th:



User Stories and Criteria ···

User Story/Use Case
User Story S1
9/9

User Story/Use Case
User Story S2
4/4

User Story/Use Case
User Story S3
4/7

User Story/Use Case
User Story S4
3/3

User Story/Use Case
User Story S5
4/4

User Story/Use Case
User Story S6

To Do ···

AT3 (Sprint 3) Bug
Function/Element
Fix buttons in pop-ups: Disable instead of making invisible
MP

+ Add a card

Doing ···

+ Add a card

Done ···

+ Add a card

Archive ···

AT2.1 (Sprint 1) Function/Element
Initial Structure for ELMS
4 Sep
MP

AT2.1 (Sprint 1) Function/Element
(S1, T1, A1) Login System (Students, Staff, Admin)
10 Sep 4/4
C MP

AT2.1 (Sprint 1) Function/Element
(A1) Add books into the library database
10 Sep 3/3
MP UK YM

AT2.2 (Sprint 2) Function/Element
A1: Update and remove books from the database
1 Oct
MP

AT2.2 (Sprint 2) Bug
Register System: TextField restrictions

AT1 Archive ···

AT1
AT1 Report, Brief, Charter
30 Aug 3/3
C MP MK UK

AT1
Stakeholder Analysis
28 Aug
C

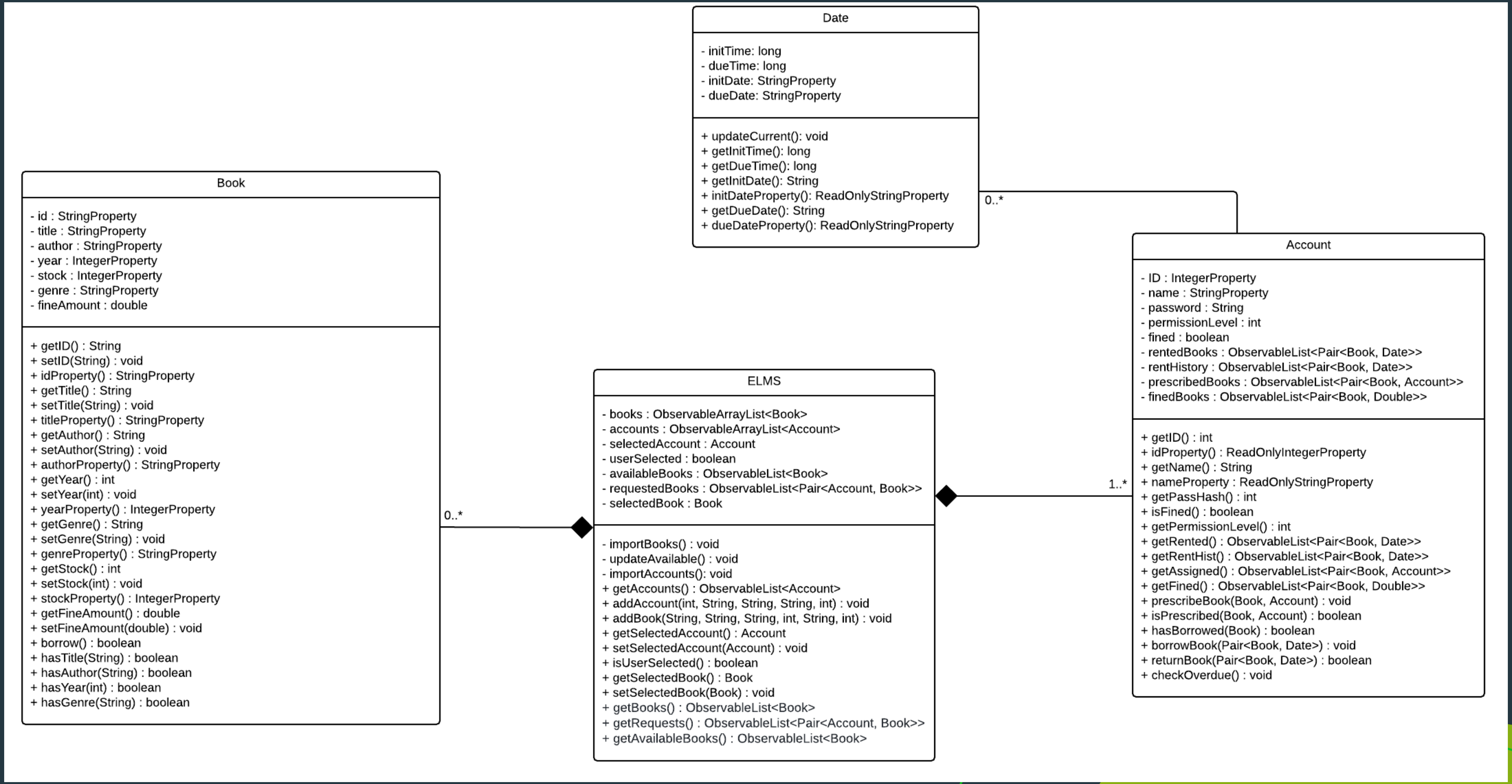
AT1
Release Backlog
28 Aug
MP

AT1
Release Implementation Plan
28 Aug
C MP MK UK

AT1
Use Case Narratives
28 Aug
C UK

Project Structure

- The application uses an MVC pattern, specifically, JavaFX


Date
- initTime: long
- dueTime: long
- initDate: StringProperty
- dueDate: StringProperty
+ updateCurrent(): void
+ getInitTime(): long
+ getDueTime(): long
+ getInitDate(): String
+ initDateProperty(): ReadOnlyStringProperty
+ getDueDate(): String
+ dueDateProperty(): ReadOnlyStringProperty
0..*
Book
- id : StringProperty
- title : StringProperty
- author : StringProperty
- year : IntegerProperty
- stock : IntegerProperty
- genre : StringProperty
- fineAmount : double
+ getID() : String
+ setID(String) : void
+ idProperty() : StringProperty
+ getTitle() : String
+ setTitle(String) : void
+ titleProperty() : StringProperty
+ getAuthor() : String
+ setAuthor(String) : void
+ authorProperty() : StringProperty
+ getYear() : int
+ setYear(int) : void
+ yearProperty() : IntegerProperty
+ getGenre() : String
+ setGenre(String) : void
+ genreProperty() : StringProperty
+ getStock() : int
+ setStock(int) : void
+ stockProperty() : IntegerProperty
+ getFineAmount() : double
+ setFineAmount(double) : void
+ borrow() : boolean
+ hasTitle(String) : boolean
+ hasAuthor(String) : boolean
+ hasYear(int) : boolean
+ hasGenre(String) : boolean
0..*
ELMS
- books : ObservableArrayList<Book>
- accounts : ObservableArrayList<Account>
- selectedAccount : Account
- userSelected : boolean
- availableBooks : ObservableList<Book>
- requestedBooks : ObservableList<Pair<Account, Book>>
- selectedBook : Book
- importBooks() : void
- updateAvailable() : void
- importAccounts(): void
+ getAccounts() : ObservableList<Account>
+ addAccount(int, String, String, String, int) : void
+ addBook(String, String, String, int, String, int) : void
+ getSelectedAccount() : Account
+ setSelectedAccount(Account) : void
+ isUserSelected() : boolean
+ getSelectedBook() : Book
+ setSelectedBook(Book) : void
+ getBooks() : ObservableList<Book>
+ getRequests() : ObservableList<Pair<Account, Book>>
+ getAvailableBooks() : ObservableList<Book>
1..*
Account
- ID : IntegerProperty
- name : StringProperty
- password : String
- permissionLevel : int
- fined : boolean
- rentedBooks : ObservableList<Pair<Book, Date>>
- rentHistory : ObservableList<Pair<Book, Date>>
- prescribedBooks : ObservableList<Pair<Book, Account>>
- finedBooks : ObservableList<Pair<Book, Double>>
+ getID() : int
+ idProperty() : ReadOnlyIntegerProperty
+ getName() : String
+ nameProperty : ReadOnlyStringProperty
+ getPassHash() : int
+ isFined() : boolean
+ getPermissionLevel() : int
+ getRented() : ObservableList<Pair<Book, Date>>
+ getRentHist() : ObservableList<Pair<Book, Date>>
+ getAssigned() : ObservableList<Pair<Book, Account>>
+ getFined() : ObservableList<Pair<Book, Double>>
+ prescribeBook(Book, Account) : void
+ isPrescribed(Book, Account) : boolean
+ hasBorrowed(Book) : boolean
+ borrowBook(Pair<Book, Date>) : void
+ returnBook(Pair<Book, Date>) : boolean
+ checkOverdue() : void

Demonstration

There’ll now be a brief pause as we switch to the application demonstration

What’s Next?

- Add maintenance lockout and book preview
- Improve usability of some sections
- Unify design choices across the views
- Resolve any minor issues or bugs that are discovered during final testing

Contribution Tables (Slides)

Chantel Mills, 13543638

Sprint 1: 'Model code for the search function' Sprint 2: 'Model code for borrowing functionality' Sprint 2: 'Added some additional methods to for search function'	Sprint 1 and 2: Developed java-based code for the model component of the search functionality. Methods were developed to search for books based on different parameters, such as the genre, author, and title of books available. For the one section of sprint 2, I developed a few more methods for the search functionality. A method to search for all the author names returned as a list, and a similar method to return a list of all the genres available. The other section of sprint 2 focused on java code for the model of the borrowing functionality. The main issue in sprint 1 and 2 was that I was unfamiliar with the javafx fxml language, so I was not able to contribute to much of the UI side of coding for these functions.
Sprint 3: 'Renewal Functionality'	Sprint 3: Sprint 3 involved the code needed for the renewal functionality, done inside the current user's account profile. I added a couple of elements to the previously developed user profile view fxml, and created the handle in the associated controller class, basing the ability to renew on if the book wanting to be renewed was in the fined books list. It was done in the second week of the designated two-week sprint, as that second week was when I learnt the associated JavaFX FXML coding language in Application programming. Unit testing was conducted on the renewal function using a temporary ObservableList of the required type to see if the date would change if renewal was available. Further testing was done to see if the renewal would not go through if the book was in the fined books list. All testing was successful, and the renewal ran as intended.

Mattias Przyrembel, 13892409

This section is going to be very long… In short, if you're looking at a line of code in the application, there's an extremely high probability I wrote it. The tasks I am assigned to on Trello are those that I was originally assigned to complete. However, in actuality, there was only one task - the renewals, which Chantel completed - that I did not complete. This can all be seen clearly in the GitHub repo commit history (shallow proof: 77/109 commits are mine). Despite the many hours I spent explaining what to do and aiding some members of the group, I ultimately ended up having to write the vast majority of the code for the ELMS. Nonetheless, the specifics are as follows. Given its length, I've had to split it across multiple slides.

Sprint 1

Initial Structure	In sprint 1, I started by creating the initial structure of the application so that everyone had a spring board that they could add their code to. I felt it would be easier and more uniform if everyone was adding to an existing structure than creating their own. This consisted of creating the application java class, the placeholder menus, and the models. With the exception of the application class, all of these would later be updated in the following sprints to follow the correct structure. I also created a template for the views so that they would remain consistent throughout the application. At a later point, after completing the login/registration (see below), I also added in the basic database for the demo in the form of a 'book' and 'user' csv file.

Mattias Przyrembel, 13892409

Sprint 1, cont.

Login/ Registration	After completing the initial structure, I began working on my allocated task for the sprint. My task was to create the login system. I added a simple grid pane structure to the fxml to display the labels and text fields for the login and a button below to login the user. This button was set as default so that the user could also fire it by hitting enter to ensure using the application was as simple and familiar. I set up the controller as necessary and added a change listener to force the contents of the 'ID' field to be numeric, again, for usability and simplicity for the user. In order to improve the security, I added in basic hash security for handling the passwords - at no point would the actual contents of the password field be transferred to another part of the application. After a discussion with Teji, it was then concluded that, despite our scope stating otherwise with valid justification, it would be worth including a registration system. Therefore, this was also added soon after following the same general principles as the login but with a larger grid pane.
Add Books	As Yize and Umair had not completed Application Programming, I expected that I would need to convert their code into a JavaFX format and create the fxml view for them at the end of the sprint. The code they completed had some usable elements, but it largely needed to be re-written for the section to be completed properly. While carrying out this task, I realised that I had failed to add in the demo 'database', as aforementioned, and added that in accordingly. The view and controller for adding the books was largely the same as the registration; I again used change listeners to force the contents of some text fields, I used a grid pane to define the basic layout, and I added button at the bottom which would change according to what was happening on the stage.

Mattias Przyrembel, 13892409

Sprint 2

The First Merge	As I was the only member of the group familiar with GitHub, I conducted the branch merges and any conflict resolution. This occurred at the beginning of each sprint, hence it being listed as part of sprint 2. At was at this point when I first began to become concerned with the group's progress; half of the tasks had not been completed and were not even close to completion. Nonetheless, I merged the two branches that had completed code (the two I had completed as discussed in the sprint 1 section) into the main branch.
View Account	After not hitting the deadlines, Pulkit began sending more messages asking for assistance with his tasks. In the end, I ended up creating the basic structure for the 'view account' scene - which was Pulkit's sprint 1 task. The fxml followed a simple layout with a VBox of buttons, to toggle the contents of the table, on the left and a table with the selected contents on the right. This was later updated to also have buttons below the table to allow for actions to be carried out on the selected items.
Modifying Library	After fixing Pulkit's section, I moved on to my allocated task which was to allow the admins to modify the library. I created a button in the admin menu to change the scene to 'modify library'. In the view, I created a simple table to display the books in the library - only those currently not borrowed as per the requirements - with two buttons underneath. One button removes the book from the library while the other create a new stage 'pop-up' that is identical to the 'add book' view except that the fields are prefilled with the details of the selected book. The admin can the change these fields' contents and save the changes to update the books details.
Staff Book Request	Once my task was complete I moved to finishing Yize and Umair's section. The code they had written however, was a complete mess and entirely unusable. I had to write their entire section from scratch. As with many of the other views, the structure consisted of a grid pane with labels and text fields and a HBox of buttons at the bottom. As with the others, change listeners were used to force the contents and binds were used to ensure the buttons couldn't be fired prematurely. The controller for this section was the first to have any unique features compared to those completed previously; in order to track which staff member requested the book, a pair structure was used for the contents of the observable list. I also added a scene in the admin menu to view the requests following fundamentally the same structure as the 'modify' view. The main difference was that the contents required an extra method call to get the value (or key, depending on the column) of the pair.

Mattias Przyrembel, 13892409

Sprint 3

The Second Merge	Once again, a large portion of the tasks were incomplete going into the third sprint. In order to get things merged in a non-broken state so that the team could move on with the sprint 3 work, I ended up having to complete Pulkit's sprint 2 work. I also had to explain to Mohammad that he was not to change the provided (by UTS in AppProg) abstract controller class, which caused a delay in the merge as he had to fix the myriad of errors that were caused by this change.
Prescribe Books	My task for this sprint was to allow staff to prescribe books to students. As the search was not fully functioning, I initially wrote this as a separate scene in the staff menu; however, it has since been added in as a button to the search menu for the staff's convenience - and as with all other role-restricted features, it has been set to be invisible for non-staff level accounts. Clicking the 'prescribe' button allows the staff member to see a new stage 'pop-up' with a table of students who have not had the books prescribed to them by the currently logged in account. The staff member can then select the students and click 'prescribe' to prescribe the books to them. The controller creates a pair of the book and staff members account when this button is clicked allowing the student to see who prescribed the book and determine which class it is need for. As with all other sections, I also added in the respective functionality in the 'view account' scene to view the prescribed books.
Adding in a New Model	Initially I suggested to Mohammad that it would be best to use the java.util.Date class for the dates for borrowing. However, I concluded that this would add unnecessary complexity when it came to displaying it in the tables. Therefore, I created a new date model, with properties, to be used with the renting and fine calculation functions.
View Account	Pulkit was still struggling with the 'view account' scene, so while I added in the functionality to handle viewing the prescribed books, I also completed the sections for handling viewing rented books and the renting history. This was a very small addition overall as this tasks was more-or-less completed at the end of sprint 2. Nonetheless, it was a contribution I made and it is therefore listed here.

Mattias Przyrembel, 13892409

Sprint 3, cont.

Overhauling the Search Function	Despite having worked on it for 7 weeks, 5 of which he had Chantel's assistance, Mohammad had still not completed the search functionality at the end of the third sprint. Fearing this may happen given the previous performances, I created a branch to test a 'new' layout - Mohammad's design used multiple pages and strayed for my original design recommendations in the wireframes. Consequently, commits 48e6c3f and f2bdda6 introduced some very large changes. The former removed the messy structure that had been created and the latter replaced it with a unified, one view interface for the library search. The controller for this page is arguably the most complicated as its view is the most dense; it uses a combination of all of the aforementioned structures and elements. To remove the need for buttons, unlike the other scenes, this scene features a change listener on the contents of the toggle group and list view so that the data can be updated seamlessly and instantly for the user. I had planned for this to be a backup, and so only spent 3.5 hours on it; however, in the Monday meeting after the sprint ended, the team unanimously voted for my version, and so it was merged into the main instead of 'SearchSystemS3' (which is still an active branch if you would like to view Mohammad's version).
Fines	Once again, the code Yize and Umair had written was completely unusable and so I had to write their section from scratch. I added methods to the account which are called when logging in and when viewing their account that check if any books are beyond their due date (which we set at 2 weeks and 1 minute post-borrowing). The fine is calculated based on the fine amount set for each book (currently a universal value though), with an additional charge being earned for each extra week it is overdue. The recurring fines were listed as being outside of the scope; however, as it is a simple addition, I decided to include it figuring it would be easier for the customer to remove it should they not like this system than it would be to add it in. I also added in the ability to view the fined books and fine amount in the 'view account' scene. I added a button to 'pay' the fine mostly for the sake of the demo as 'secure payment handling of fines' was listed as outside of our scope.

Mohammad Karim, 13140429

Sprint 1: 'View and controller code for search function'	In sprint 1, I worked on building an overall GUI for search functionality which could be improved in the future sprints. For all the sprints my responsibility was to handle the GUI code for search and borrow function. To get a functioning GUI, the code for view and controller needed to be completed. By the end of sprint 1, There were 4 different views created for search functionality menu, search by book name, search by book author, search by book genre and 4 controllers for each of the view parts. which were not linked with each other, and the buttons were also not working.
Sprint 2: 'completion of search function and Borrow function'	At the starting of sprint 2, I worked on making the buttons work, so I setup the event handlers. Code was added to all the controllers so that it could handle any button clicks. There was a problem with the GUI crashing after the event handlers were setup which had to be fixed as soon as possible. In the middle phase of sprint 2, I worked on getting the list views setup for the graphical user interface. After setting up the list views, I worked on the methods in controller to get search results shown in the list. After this I worked on showing all available books, authors, genre in their respective view parts. Then I worked on selecting Item from that list. In the last phase of sprint 2, I worked on the borrow function, I had to add buttons in view to initiate the borrow function and link the buttons to controller so that it could call borrow methods in model. I set up a text node in view so that it could show the results of borrow function.
Sprint 3: 'Changes in code structure of Borrow function and worked on improving the GUI'	In sprint 3, I worked on improving the code and graphical user interface for search and borrow function. I also had to make edits in model code of borrow function. In the first phase of sprint 3, I work on setting up a new method for borrow function which uses a paired observable list to work. In the middle phase, I worked with replacing the list view with table views which had quite a bit of work because some of the code needed to be changed and some changes needed to be made according to how table views work with table columns. I also had to add availabilities of books in the tables and a column for year. In the end phase, I had to add the search by year function. For that I had to make view, controller for that function and complete all the tasks to make it fully functional. By the end of sprint 3, all the objectives were completed.

Pulkit Jain, 13653934

Sprint 1: 'Programmed the fxml and controllers'	Programmed the Books borrowed, return date and outstanding filed fxml and controllers during Sprint 1 using GitHub and NetBeans using the coding language Java 8 and Javafx. There were a lot of bugs that had come forward while programming the code that were fixed with the help of the debugger on NetBeans. Had a lot of help and guidance from Mattias which was greatly appreciated during all the three Sprints.
Sprint 2: 'Programmed the books borrowed button and table'	In Sprint 2, I Programmed the books borrowed button and table. There were a lot of bugs that had come forward while programming the code that were fixed with the help of the debugger on NetBeans.
Sprint 3: 'Programmed the tables, buttons, and final touch up'	In Sprint 3, Programmed the outstanding fines, fine-tuned the books borrowed by adding the return date, programmed the rent history and fines tables and buttons and final touch up. There were a lot of bugs that had come forward while programming the code that were fixed with the help of the debugger on NetBeans.

Umair Khalidi, 14326336

Sprint 1: 'Add books into library database'	In sprint 1, I worked on developing a Java-based API to add registered books into the library. A bit of modification by adding addBook method and controller (MVC). Table added to view requests in admin menu. In the next cycle, controllers were added to allow the admin to have accessibility of the requests. Now E-books may only be added, updated, and removed by administrators, who keep track of features like category (book, journal, research paper, or article), title, author, reference number, image, and other information particular to each item.
Sprint 2: 'Staff book Request'	The API for this Java-based code was modified a bit to resolve empty data. Model added so staff can store requester and book. The ViewRequest for admin menu is incomplete for the controller class AdminController. ViewAccountController coded in JavaFX to make it a GUI application. The modifications have now fixed many occurring errors that previously existed. Staff can request introduction of new or essential books for the library by completing a book request form
Sprint 3: 'Calculation of fines'	The model package was modified by adding Constructor variant for book requests. Borrow class was added to record each borrowed book to calculate the fine imposed. View account controller was created to (FXML object) generate the message which displays the fine imposed. Fines are calculated as follows: if an e-book is not returned within a certain time frame, a fine is assessed and the student or staff member is alerted by SMS and/or email.

Yize Ma, 13149594

Sprint 1: 'Add books into library database'	In Sprint 1 I worked on making Books model and Adding book function is a based function for our project to add new books in our database, with ID, title, author, genre and years.
Sprint 2: 'Staff book Request'	In Sprint 2 I worked on Issue book function, which is adding a function allow user and staff to order the books they would like to read, so send request to the library, and before the function send request to library, it will do a Id check for the books and to check id the boos are already in the database, if not it will send the request.
Sprint 3: 'Calculation of fines'	In sprint 3, I worked under AccountView section to make a fine calculation using Java FX, it any user had borrow a book for 2 weeks without renew it they will need to pay a fee for not return the books before they can return and borrow next books.