```java
package prj.library;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.chart.*;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory
;
import javafx.scene.layout.Pane;
import prj.library.models.Book;
import prj.library.models.Customer;
import prj.library.models.Genre;
import prj.library.models.Lends;
import javafx.scene.layout.GridPane;
import prj.library.networking.ClientController;
import prj.library.utils.CLIUtils;
import java.io.IOException;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

/**
 * The ViewController class is responsible for
handling the GUI of the application.
 */
public class ViewController {

    private ClientController clientController;

    @FXML
    private Pane homePane;
    @FXML
    private Pane addBookPane;
    @FXML
    private Pane searchBooksPane;
    @FXML
    private Pane lendPane;
    @FXML
    private Pane statsPane;
```

```
40      @FXML
41      private Pane customersPane;
42      @FXML
43      private TableView<Book> homeTableView;
44      @FXML
45      private TableView<Book> searchBookTableView;
46      @FXML
47      private TableView<Lends> searchLendTableView;
48      @FXML
49      private TableView<Customer> customersTableView;
50      @FXML
51      private TableColumn<Book, Integer> idHColumn;
52      @FXML
53      private TableColumn<Book, String> titleHColumn;
54      @FXML
55      private TableColumn<Book, String> authorHColumn;
56      @FXML
57      private TableColumn<Book, Genre> genreHColumn;
58      @FXML
59      private TableColumn<Book, Integer> yearHColumn;
60      @FXML
61      private TableColumn<Book, Integer> copiesHColumn;
62      @FXML
63      private TableColumn<Book, Integer> idSColumn;
64      @FXML
65      private TableColumn<Book, String> titleSColumn;
66      @FXML
67      private TableColumn<Book, String> authorSColumn;
68      @FXML
69      private TableColumn<Book, Genre> genreSColumn;
70      @FXML
71      private TableColumn<Book, Integer> yearSColumn;
72      @FXML
73      private TableColumn<Book, Integer> copiesSColumn;
74      @FXML
75      private TableColumn<Customer, Integer> idCColumn;
76      @FXML
77      private TableColumn<Customer, String> nameCColumn
    ;
78      @FXML
79      private TableColumn<Customer, String>
```

```
 79 emailCColumn;
 80     @FXML
 81     private TableColumn<Customer, String>
    phoneCColumn;
 82     @FXML
 83     private TableColumn<Customer, String>
    addressCColumn;
 84     @FXML
 85     private TableColumn<Lends, Integer> idLColumn;
 86     @FXML
 87     private TableColumn<Lends, Integer> bidLColumn;
 88     @FXML
 89     private TableColumn<Lends, Integer> cidLColumn;
 90     @FXML
 91     private TableColumn<Lends, String> returnLColumn
    ;
 92     @FXML
 93     private TableColumn<Lends, Boolean>
    returnedLColumn;
 94     @FXML
 95     private TextField titleABTextField;
 96     @FXML
 97     private TextField authorABTextField;
 98     @FXML
 99     private ChoiceBox<Genre> genreABChoiceBox;
100     @FXML
101     private TextField yearABTextField;
102     @FXML
103     private TextField copiesABTextField;
104     @FXML
105     private TextField titleSBTextField;
106     @FXML
107     private TextField authorSBTextField;
108     @FXML
109     private ChoiceBox<Genre> genreSBChoiceBox;
110     @FXML
111     private TextField yearSBTextField;
112     @FXML
113     private TextField titleLendTextField;
114     @FXML
115     private DatePicker returnLendDatePicker;
```

```
116     @FXML
117     private TextField phoneLendTextField;
118     @FXML
119     private ChoiceBox<Boolean> returnedLendChoiceBox
    ;
120     @FXML
121     private Button addateCustomerButton;
122     @FXML
123     private TextField nameCustomerTextField;
124     @FXML
125     private TextField phoneCustomerTextField;
126     @FXML
127     private TextField emailCustomerTextField;
128     @FXML
129     private TextField addressCustomerTextField;
130     @FXML
131     private PieChart genrePieChart;
132     @FXML
133     private BarChart<String, Number> numusBarChart;
134
135
136
137     public ViewController() {
138         try {
139             clientController = new ClientController
    ();
140         } catch (IOException e) {
141             if (clientController != null)
    clientController.close();
142             CLIUtils.clientCriticalError(e.
    getMessage());
143             showErrorDialog("Error", "Critical error
    ", "Could not connect to the server. Please try
    again later.");
144             LibraryApplication.close();
145         }
146     }
147
148     @FXML
149     public void initialize() {
150
```

```
151         hideAllPanes();
152 
153         //Set up the home view
154         showPane(homePane);
155 
156         initHomeTableView();
157         initSearchTableView();
158         initCustomersTableView();
159         initLendsTableView();
160         initChoiceBoxes();
161         initGenrePieChart();
162         initNumusBarChart();
163 
164         loadBooks();
165 
166         //Add listener to the ListView
167         customersTableView.getSelectionModel().
    selectedItemProperty().addListener((observable,
    oldValue, newValue) -> {
168             if (newValue != null) {
169                 //Update TextFields with selected
    customer data
170                 nameCustomerTextField.setText(
    newValue.getName());
171                 phoneCustomerTextField.setText(
    newValue.getPhone());
172                 emailCustomerTextField.setText(
    newValue.getEmail());
173                 addressCustomerTextField.setText(
    newValue.getAddress());
174 
175                 //Change button text to "Update"
176                 addateCustomerButton.setText("\uD83D
    \uDCA1 update");
177             }
178         });
179     }
180 
181     @FXML
182     private void onAddBookButtonClick(){
183         String title = titleABTextField.getText();
```

```
184         String author = authorABTextField.getText();
185         Genre genre = genreABChoiceBox.getValue();
186         String year = yearABTextField.getText();
187         String copies = copiesABTextField.getText();
188 
189 
190         if(title.isEmpty() || author.isEmpty() ||
    genre == null || year.isEmpty() || copies.isEmpty
    ()) {
191             showErrorDialog("Error", "Invalid input"
    , "Please fill all fields.");
192             return;
193         }
194 
195         try {
196             int yearInt = Integer.parseInt(year);
197             int copiesInt = Integer.parseInt(copies
    );
198 
199             Book book = new Book(title, author,
    yearInt, genre, copiesInt);
200             clientController.createBook(book);
201         } catch (NumberFormatException e) {
202             showErrorDialog("Error", "Invalid input"
    , "Please enter a valid year and number of copies."
    );
203         } catch (IllegalArgumentException e) {
204             showErrorDialog("Error", "Invalid input"
    , "Please enter a valid genre.");
205         }
206 
207         clearBookFields();
208     }
209 
210     @FXML
211     private void onCancelButtonClick(){
212         titleABTextField.clear();
213         authorABTextField.clear();
214         genreABChoiceBox.setValue(null);
215         yearABTextField.clear();
216         copiesABTextField.clear();
```

```
217     }
218
219     @FXML
220     protected void onSearchBookButtonClick() {
221         String title = titleSBTextField.getText();
222         String author = authorSBTextField.getText();
223         int year = yearSBTextField.getText().isEmpty
    () ? 0 : Integer.parseInt(yearSBTextField.getText
    ());
224         Genre genre = genreSBChoiceBox.getValue();
225         List<Book> books;
226         int choice = 0;
227
228         if(!title.isEmpty() && !author.isEmpty() &&
    year != 0 && genre != null) choice = 0;
229         if(!title.isEmpty() && author.isEmpty() &&
    year == 0 && genre == null) choice = 1;
230         if(title.isEmpty() && !author.isEmpty() &&
    year == 0 && genre == null) choice = 2;
231         if(title.isEmpty() && author.isEmpty() &&
    year != 0 && genre == null) choice = 3;
232         if(title.isEmpty() && author.isEmpty() &&
    year == 0 && genre != null) choice = 4;
233         if(!title.isEmpty() && !author.isEmpty() &&
    year == 0 && genre == null) choice = 5;
234         if(!title.isEmpty() && author.isEmpty() &&
    year != 0 && genre == null) choice = 6;
235         if(!title.isEmpty() && author.isEmpty() &&
    year == 0 && genre != null) choice = 7;
236         if(title.isEmpty() && !author.isEmpty() &&
    year != 0 && genre == null) choice = 8;
237         if(title.isEmpty() && !author.isEmpty() &&
    year == 0 && genre != null) choice = 9;
238         if(title.isEmpty() && author.isEmpty() &&
    year != 0 && genre != null) choice = 10;
239         if(!title.isEmpty() && !author.isEmpty() &&
    year != 0 && genre == null) choice = 11;
240         if(!title.isEmpty() && !author.isEmpty() &&
    year == 0 && genre != null) choice = 12;
241         if(!title.isEmpty() && author.isEmpty() &&
    year != 0 && genre != null) choice = 13;
```

```
242         if(title.isEmpty() && !author.isEmpty() &&
    year != 0 && genre != null) choice = 14;
243         if (title.isEmpty() && author.isEmpty() &&
    year == 0 && genre == null) choice = 15;
244
245         Book tmp = new Book(title, author, year,
    genre);
246         books = clientController.searchBooksBy(
    choice, tmp);
247         showBooksOnTableView(books,
    searchBookTableView);
248
249         clearSearchBooksFields();
250     }
251
252     @FXML
253     protected void onUpdateBookButtonClick() {
254         Book selectedBook = searchBookTableView.
    getSelectionModel().getSelectedItem();
255
256         if (selectedBook == null) {
257             System.out.println("No book selected for
     editing.");
258             return;
259         }
260
261         Dialog<Book> dialog = new Dialog<>();
262         dialog.setTitle("Edit Book");
263
264         // Set the button types.
265         ButtonType saveButtonType = new ButtonType("
    Save", ButtonBar.ButtonData.OK_DONE);
266         dialog.getDialogPane().getButtonTypes().
    addAll(saveButtonType, ButtonType.CANCEL);
267
268         // Create the fields and populate with
    current book data.
269         TextField titleField = new TextField(
    selectedBook.getTitle());
270         TextField authorField = new TextField(
    selectedBook.getAuthor());
```

```
271         TextField yearField = new TextField(String.
    valueOf(selectedBook.getYear()));
272         ChoiceBox<Genre> genreChoiceBox = new
    ChoiceBox<>(FXCollections.observableArrayList(Genre.
    values()));
273         genreChoiceBox.setValue(selectedBook.
    getGenre());
274         TextField copiesField = new TextField(String
    .valueOf(selectedBook.getCopies()));
275 
276         // Create a grid pane and add the fields
277         GridPane grid = new GridPane();
278         grid.add(new Label("Title:"), 0, 0);
279         grid.add(titleField, 1, 0);
280         grid.add(new Label("Author:"), 0, 1);
281         grid.add(authorField, 1, 1);
282         grid.add(new Label("Year:"), 0, 2);
283         grid.add(yearField, 1, 2);
284         grid.add(new Label("Genre:"), 0, 3);
285         grid.add(genreChoiceBox, 1, 3);
286         grid.add(new Label("Copies:"), 0, 4);
287         grid.add(copiesField, 1, 4);
288 
289         dialog.getDialogPane().setContent(grid);
290 
291         // Convert the result to a Book object when
    the save button is clicked.
292         dialog.setResultConverter(dialogButton -> {
293             if (dialogButton == saveButtonType) {
294                 selectedBook.setTitle(titleField.
    getText());
295                 selectedBook.setAuthor(authorField.
    getText());
296                 selectedBook.setYear(Integer.
    parseInt(yearField.getText()));
297                 selectedBook.setGenre(genreChoiceBox
    .getValue());
298                 selectedBook.setCopies(Integer.
    parseInt(copiesField.getText()));
299                 return selectedBook;
300             }
```

```
301             return null;
302         });
303 
304         dialog.showAndWait().ifPresent(editedBook
     -> {
305             clientController.updateBook(editedBook);
306             List<Book> books = new ArrayList<>();
307             books.add(clientController.readBook(
    editedBook.getId()));
308             showBooksOnTableView(books,
    searchBookTableView);
309         });
310     }
311 
312     @FXML
313     protected void onDeleteBookButtonClick() {
314         Book selectedBook = searchBookTableView.
    getSelectionModel().getSelectedItem();
315 
316         if (selectedBook == null) {
317             CLIUtils.clientInfo("No book selected
    for editing.");
318             showErrorDialog("Error", "No book
    selected", "Please select a book to delete.");
319             return;
320         }
321         clientController.deleteBook(selectedBook.
    getId());
322         searchBookTableView.getItems().remove(
    selectedBook);
323     }
324 
325     @FXML
326     protected void onLendBookButtonClick() {
327         Book selectedBook = searchBookTableView.
    getSelectionModel().getSelectedItem();
328         final Lends[] lend = new Lends[1];
329 
330         if (selectedBook == null) {
331             CLIUtils.clientInfo("No book selected
    for editing.");
```

```
332             showErrorDialog("Error", "No book
    selected", "Please select a book to lend.");
333             return;
334         }
335 
336         Book refreshedBook = clientController.
    readBook(selectedBook.getId());
337         if (refreshedBook.getCopies() < 1) {
338             showErrorDialog("Error", "No copies
    available", "There are no copies of this book
    available.");
339             return;
340         }
341 
342         Dialog<Book> dialog = new Dialog<>();
343         dialog.setTitle("Lend Book");
344 
345         // Set the button types.
346         ButtonType saveButtonType = new ButtonType("
    Save", ButtonBar.ButtonData.OK_DONE);
347         dialog.getDialogPane().getButtonTypes().
    addAll(saveButtonType, ButtonType.CANCEL);
348 
349         // Create the fields and populate with
    current book data.
350         ChoiceBox<Customer> customerChoiceBox = new
    ChoiceBox<>();
351         customerChoiceBox.getItems().addAll(
    clientController.getCustomers());
352         DatePicker returnDatePicker = new DatePicker
    ();
353         Label booktitleLabel = new Label(
    selectedBook.getTitle());
354 
355         // Create a grid pane and add the fields
356         GridPane grid = new GridPane();
357         grid.add(new Label("Chose customer:"), 0, 0
    );
358         grid.add(customerChoiceBox, 1, 0);
359         grid.add(new Label("Return date:"), 0, 1);
360         grid.add(returnDatePicker, 1, 1);
```

```
361         grid.add(new Label("Book's title:"), 0, 2);
362         grid.add(booktitleLabel, 1, 2);
363 
364         dialog.getDialogPane().setContent(grid);
365         lend[0] = null;
366 
367         // Convert the result to a Book object when
    the save button is clicked.
368         dialog.setResultConverter(dialogButton -> {
369             if (dialogButton == saveButtonType) {
370                 if(clientController.readBook(
    selectedBook.getId()).getCopies() < 1) {
371                     showErrorDialog("Error", "No
    copies available", "There are no copies of this book
     available.");
372                     return null;
373                 } else {
374                     selectedBook.setCopies(
    selectedBook.getCopies()-1);
375                     return selectedBook;
376                 }
377             } else {
378                 return null;
379             }
380         });
381 
382         dialog.showAndWait().ifPresent(editedBook
     -> {
383             lend[0] = new Lends(selectedBook.getId
    (), customerChoiceBox.getValue().getId(),
    returnDatePicker.getValue(), false);
384             clientController.createLend(lend[0]);
385             clientController.updateBook(editedBook);
386             List<Book> books = new ArrayList<>();
387             books.add(clientController.readBook(
    editedBook.getId())); //TODO check to remove
388             showBooksOnTableView(books,
    searchBookTableView);
389         });
390     }
391 
```

```
392     @FXML
393     protected void onLateLendButtonClick() {
394         List<Lends> lends = clientController.
    searchLateLends();
395         clearSearchLends();
396         showLendsOnTableView(lends,
    searchLendTableView);
397     }
398 
399     @FXML
400     protected void onSearchLendButtonClick(){
401         searchLendTableView.getItems().clear();
402 
403         String title = titleLendTextField.getText();
404         LocalDate date = returnLendDatePicker.
    getValue();
405         String cell = phoneLendTextField.getText();
406         Boolean unclepear = returnedLendChoiceBox.
    getValue();
407 
408         int b_id=0; //book id
409         int c_id=0; //customer id
410         boolean returned;
411         boolean sentinel = false;
412 
413         if(unclepear != null && unclepear) returned
     = true;
414         else if (unclepear != null && !unclepear) {
415             returned = false;
416             sentinel = true;
417         } else returned = false;
418 
419         //checks if book and customer exists
420         if (!title.isEmpty()) {
421             List<Book> books = clientController.
    searchBooksBy(1, new Book(0, title, "", 0, Genre.
    Genre, 0));
422             if (books == null || books.isEmpty()) {
423                 showErrorDialog("Error", "Book not
    found", "Book you are looking for is not found. Try
    something else.");
```

```
424                 clearSearchLends();
425                 return;
426             }
427             b_id = books.get(0).getId();
428         }
429         if(!cell.isEmpty()) {
430             List<Customer> customers =
    clientController.searchCustomersBy(2, new Customer(
    "", "", cell, ""));
431             if (customers == null || customers.
    isEmpty()) {
432                 showErrorDialog("Error", "Customer
    not found", "Customer you are looking for is not
    found. Try something else.");
433                 clearSearchLends();
434                 return;
435             }
436             c_id = customers.get(0).getId();
437         }
438
439         int choice=7;
440         if (b_id != 0 && c_id != 0 && date != null)
    choice = 0;
441         else if (b_id != 0 && c_id == 0 && date ==
    null) choice = 1;
442         else if (b_id == 0 && c_id != 0 && date ==
    null) choice = 2;
443         else if (b_id == 0 && c_id == 0 && date !=
    null) choice = 3;
444         else if (b_id != 0 && c_id != 0 && date ==
    null) choice = 4;
445         else if (b_id != 0 && c_id == 0 && date !=
    null) choice = 5;
446         else if (b_id == 0 && c_id != 0 && date !=
    null) choice = 6;
447
448         List<Lends> lends = clientController.
    searchLendsBy(choice, b_id, date, c_id, returned,
    sentinel);
449         showLendsOnTableView(lends,
    searchLendTableView);
```

```
450         clearSearchLends();
451     }
452 
453     @FXML
454     protected void onReturnLendButtonClick() {
455         Lends selectedLend = searchLendTableView.
    getSelectionModel().getSelectedItem();
456         if (selectedLend == null) {
457             showErrorDialog("Error", "No lend
    selected", "Please select a lend to return.");
458             return;
459         }
460 
461         Book book = clientController.readBook(
    selectedLend.getBookId());
462 
463         book.setCopies(book.getCopies()+1);
464         clientController.updateBook(book);
465         selectedLend.setReturned(true);
466         clientController.updateLend(selectedLend);
467 
468         List<Lends> lends = new ArrayList<>();
469         lends.add(clientController.readLend(
    selectedLend));
470         showLendsOnTableView(lends,
    searchLendTableView);
471         clearSearchLends();
472     }
473 
474     @FXML
475     protected void onDeleteLendButtonClick() {
476         Lends selectedLend = searchLendTableView.
    getSelectionModel().getSelectedItem();
477         if (selectedLend == null) {
478             showErrorDialog("Error", "No lend
    selected", "Please select a lend to delete.");
479             return;
480         }
481 
482         clientController.deleteLend(selectedLend);
483         searchLendTableView.getItems().remove(
```

```
483 selectedLend);
484         clearSearchLends();
485     }
486 
487     @FXML
488     protected void onUpdateLendButtonClick() {
489         Lends selectedLend = searchLendTableView.
    getSelectionModel().getSelectedItem();
490         if (selectedLend == null) {
491             showErrorDialog("Error", "No lend
    selected", "Please select a lend to update.");
492             return;
493         }
494 
495         Dialog<Lends> dialog = new Dialog<>();
496         dialog.setTitle("Update Lend");
497 
498         ButtonType saveButtonType = new ButtonType("
    Save", ButtonBar.ButtonData.OK_DONE);
499         dialog.getDialogPane().getButtonTypes().
    addAll(saveButtonType, ButtonType.CANCEL);
500 
501         Label customerLabel = new Label(
    clientController.readCustomer(selectedLend.
    getCustomerId()).getName());
502         DatePicker returnDatePicker = new DatePicker
    ();
503         Label booktitleLabel = new Label(
    clientController.readBook(selectedLend.getBookId()).
    getTitle());
504 
505         GridPane grid = new GridPane();
506         grid.add(new Label("Customer:"), 0, 0);
507         grid.add(customerLabel, 1, 0);
508         grid.add(new Label("Return date:"), 0, 1);
509         grid.add(returnDatePicker, 1, 1);
510         grid.add(new Label("Book's title:"), 0, 2);
511         grid.add(booktitleLabel, 1, 2);
512 
513         dialog.getDialogPane().setContent(grid);
514 
```

```
515         dialog.setResultConverter(dialogButton -> {
516             if (dialogButton == saveButtonType) {
517                 selectedLend.setReturnDate(
    returnDatePicker.getValue());
518                 return selectedLend;
519             }
520             return null;
521         });
522 
523         dialog.showAndWait().ifPresent(editedLend
     -> {
524             clientController.updateLend(editedLend);
525             List<Lends> lends = new ArrayList<>();
526             lends.add(clientController.readLend(
    editedLend));
527             showLendsOnTableView(lends,
    searchLendTableView);
528         });
529         clearSearchLends();
530     }
531 
532     @FXML
533     protected void onAddateCustomerButtonClick() {
534         Customer selectedCustomer =
    customersTableView.getSelectionModel().
    getSelectedItem();
535         if (selectedCustomer != null) {
536             updateCustomer(selectedCustomer);
537         } else {
538             addCustomer();
539         }
540     }
541 
542     @FXML
543     protected void onDeleteCustomerButtonClick() {
544         Customer selectedCustomer =
    customersTableView.getSelectionModel().
    getSelectedItem();
545         if(selectedCustomer == null) {
546             showErrorDialog("Error", "No customer
    selected", "Please select a customer to delete.");
```

```
547             return;
548         }
549         clientController.deleteCustomer(
    selectedCustomer);
550         customersTableView.getSelectionModel().
    clearSelection();
551         onCancelCustomerButtonClick();
552         customersTableView.getItems().remove(
    selectedCustomer);
553     }
554 
555     @FXML
556     protected void onSearchCustomerButtonClick(){
557         String name = nameCustomerTextField.getText
    ();
558         String phone = phoneCustomerTextField.
    getText();
559         String email = emailCustomerTextField.
    getText();
560         String address = addressCustomerTextField.
    getText();
561         int choice;
562         if(!name.isEmpty() && !phone.isEmpty() && !
    email.isEmpty() && !address.isEmpty()) choice = 0;
563         else if(!name.isEmpty() && phone.isEmpty
    () && email.isEmpty() && address.isEmpty()) choice
     = 1;
564         else if(name.isEmpty() && !phone.isEmpty
    () && email.isEmpty() && address.isEmpty()) choice
     = 2;
565         else if(name.isEmpty() && phone.isEmpty
    () && !email.isEmpty() && address.isEmpty()) choice
     = 3;
566         else if(name.isEmpty() && phone.isEmpty
    () && email.isEmpty() && !address.isEmpty()) choice
     = 4;
567         else if(!name.isEmpty() && !phone.isEmpty
    () && email.isEmpty() && address.isEmpty()) choice
     = 5;
568         else if(!name.isEmpty() && !phone.isEmpty
    () && !email.isEmpty() && address.isEmpty()) choice
```

```
568  = 6;
569          else if(!name.isEmpty() && phone.isEmpty
    () && !email.isEmpty() && address.isEmpty()) choice
     = 7;
570          else if(!name.isEmpty() && phone.isEmpty
    () && email.isEmpty() && !address.isEmpty()) choice
     = 8;
571          else if(name.isEmpty() && !phone.isEmpty
    () && !email.isEmpty() && address.isEmpty()) choice
     = 9;
572          else if(name.isEmpty() && !phone.isEmpty
    () && email.isEmpty() && !address.isEmpty()) choice
     = 10;
573          else if(name.isEmpty() && phone.isEmpty
    () && !email.isEmpty() && !address.isEmpty()) choice
     = 11;
574          else if(!name.isEmpty() && phone.isEmpty
    () && !email.isEmpty() && !address.isEmpty()) choice
     = 12;
575          else if(name.isEmpty() && !phone.isEmpty
    () && !email.isEmpty() && !address.isEmpty()) choice
     = 13;
576          else if(!name.isEmpty() && !phone.isEmpty
    () && email.isEmpty() && !address.isEmpty()) choice
     = 14;
577          else choice = 15;
578
579          List<Customer> customers = clientController.
    searchCustomersBy(choice, new Customer(name, email,
    phone, address));
580          if(customers==null || customers.isEmpty())
    showErrorDialog("Error", "No customers found", "
    Customers you are looking for are not found. Try
    something else.");
581          else {
582              ObservableList<Customer> customerNames
     = FXCollections.observableArrayList();
583              customerNames.addAll(customers);
584              customersTableView.setItems(
    customerNames);
585          }
```

```
586     }
587 
588     @FXML
589     protected void onCancelCustomerButtonClick() {
590         clearSearchCustomers();
591         addateCustomerButton.setText("\uD83D\uDE4B
    add");
592     }
593 
594     @FXML
595     public void onHomeClick(){
596         hideAllPanes();
597         showPane(homePane);
598         loadBooks();
599     }
600 
601     @FXML
602     public void onAddClick(){
603         hideAllPanes();
604         showPane(addBookPane);
605     }
606 
607     @FXML
608     public void onSearchClick(){
609         hideAllPanes();
610         showPane(searchBooksPane);
611     }
612 
613     @FXML
614     public void onLendClick(){
615         hideAllPanes();
616         showPane(lendPane);
617     }
618 
619     @FXML
620     public void onCustomerClick(){
621         hideAllPanes();
622         showPane(customersPane);
623     }
624 
625     @FXML
```

```
626     public void onStatsClick(){
627         hideAllPanes();
628         showPane(statsPane);
629         updateGenreChart();
630         updateCustomerLendsBarChart();
631     }
632 
633     /**
634      * Shows an error dialog with the given title, header and content.
635      * @param title the title of the dialog
636      * @param header the header of the dialog
637      * @param content the content of the dialog
638      */
639     private void showErrorDialog(String title, String header, String content) {
640         Alert alert = new Alert(Alert.AlertType.ERROR);
641         alert.setTitle(title);
642         alert.setHeaderText(header);
643         alert.setContentText(content);
644         alert.showAndWait();
645     }
646 
647     /**
648      * Shows the books on the given table view.
649      * @param books the books to show
650      * @param tableView the table view to show the books on
651      */
652     private void showBooksOnTableView(List<Book> books, TableView tableView) {
653 
654         if(books == null) {
655             showErrorDialog("Error", "Critical error", "Could not connect to the server. Please try again later.");
656             return;
657         }
658 
659         ObservableList<Book> observableList =
```

```
659 FXCollections.observableArrayList(books);
660         tableView.setItems(observableList);
661     }
662 
663     /**
664      * Loads the books from the server and shows
    them on the home table view.
665      * This method is called when the home view is
    shown.
666      */
667     private void loadBooks() {
668         List<Book> books = clientController.getBooks
    ();
669         showBooksOnTableView(books, homeTableView);
670     }
671 
672     /**
673      * Initializes the home table view.
674      */
675     private void initHomeTableView() {
676         idHColumn.setCellValueFactory(new
    PropertyValueFactory<>("id"));
677         titleHColumn.setCellValueFactory(new
    PropertyValueFactory<>("title"));
678         authorHColumn.setCellValueFactory(new
    PropertyValueFactory<>("author"));
679         genreHColumn.setCellValueFactory(new
    PropertyValueFactory<>( "genre"));
680         yearHColumn.setCellValueFactory(new
    PropertyValueFactory<>("year"));
681         copiesHColumn.setCellValueFactory(new
    PropertyValueFactory<>("copies"));
682     }
683 
684     /**
685      * Initializes the search table view.
686      */
687     private void initSearchTableView() {
688         idSColumn.setCellValueFactory(new
    PropertyValueFactory<>("id"));
689         titleSColumn.setCellValueFactory(new
```

```
689 PropertyValueFactory<>("title"));
690         authorSColumn.setCellValueFactory(new
    PropertyValueFactory<>("author"));
691         genreSColumn.setCellValueFactory(new
    PropertyValueFactory<>( "genre"));
692         yearSColumn.setCellValueFactory(new
    PropertyValueFactory<>("year"));
693         copiesSColumn.setCellValueFactory(new
    PropertyValueFactory<>("copies"));
694     }
695 
696     /**
697      * Initializes the lends table view.
698      */
699     private void initLendsTableView() {
700         idLColumn.setCellValueFactory(new
    PropertyValueFactory<>("id"));
701         bidLColumn.setCellValueFactory(new
    PropertyValueFactory<>("bookId"));
702         cidLColumn.setCellValueFactory(new
    PropertyValueFactory<>("customerId"));
703         returnLColumn.setCellValueFactory(new
    PropertyValueFactory<>("returnDate"));
704         returnedLColumn.setCellValueFactory(new
    PropertyValueFactory<>("returned"));
705     }
706 
707     /**
708      * Initializes the customers table view.
709      */
710     private void initCustomersTableView() {
711         idCColumn.setCellValueFactory(new
    PropertyValueFactory<>("id"));
712         nameCColumn.setCellValueFactory(new
    PropertyValueFactory<>("name"));
713         emailCColumn.setCellValueFactory(new
    PropertyValueFactory<>("email"));
714         phoneCColumn.setCellValueFactory(new
    PropertyValueFactory<>("phone"));
715         addressCColumn.setCellValueFactory(new
    PropertyValueFactory<>("address"));
```

```
716     }
717 
718     /**
719      * Updates the customer with the data from the
    text fields.
720      * @param customer the customer to update
721      */
722     private void updateCustomer(Customer customer) {
723         customer.setName(nameCustomerTextField.
    getText());
724         customer.setPhone(phoneCustomerTextField.
    getText());
725         customer.setEmail(emailCustomerTextField.
    getText());
726         customer.setAddress(addressCustomerTextField
    .getText());
727 
728         clientController.updateCustomer(customer);
729 
730         List<Customer> customers = new ArrayList
    <>();
731         customers.add(clientController.readCustomer(
    customer.getId()));
732         showCustomersOnTableView(customers,
    customersTableView);
733         onCancelCustomerButtonClick();
734 
735     }
736 
737     /**
738      * Adds a new customer with the data from the
    text fields.
739      */
740     private void addCustomer() {
741         String name = nameCustomerTextField.getText
    ();
742         String phone = phoneCustomerTextField.
    getText();
743         String email = emailCustomerTextField.
    getText();
744         String address = addressCustomerTextField.
```

```
744 getText();
745 
746         if (name.isEmpty() || phone.isEmpty() ||
    email.isEmpty() || address.isEmpty()) {
747             showErrorDialog("Error", "Invalid input"
    , "Please fill all fields.");
748             return;
749         }
750 
751         Customer newCustomer = new Customer(name,
    email, phone, address);
752         clientController.createCustomer(newCustomer
    );
753 
754         List<Customer> customers = new ArrayList
    <>();
755 
756         customers = clientController.
    searchCustomersBy(0, new Customer(name, email, phone
    , address));
757         showCustomersOnTableView(customers,
    customersTableView);
758         clearSearchCustomers();
759     }
760 
761     /**
762      * Shows the customers on the given table view.
763      * @param customers the customers to show
764      * @param tableView the table view to show the
    customers on
765      */
766     private void showCustomersOnTableView(List<
    Customer> customers, TableView tableView) {
767         if(customers == null) {
768             showErrorDialog("Error", "Critical error
    ", "Could not connect to the server. Please try
    again later.");
769             return;
770         }
771         ObservableList<Customer> observableList =
    FXCollections.observableArrayList(customers);
```

```
772         tableView.setItems(observableList);
773     }
774 
775     /**
776      * Shows the lends on the given table view.
777      * @param lends the lends to show
778      * @param tableView the table view to show the
    lends on
779      */
780     private void showLendsOnTableView(List<Lends>
    lends, TableView tableView) {
781         if (lends == null) {
782             showErrorDialog("Error", "Critical error
    ", "Could not connect to the server. Please try
    again later.");
783             return;
784         }
785 
786         ObservableList<Lends> observableList =
    FXCollections.observableArrayList(lends);
787         tableView.setItems(observableList);
788     }
789 
790     /**
791      * Initializes choice boxes.
792      */
793     private void initChoiceBoxes() {
794         genreABChoiceBox.getItems().addAll(Genre.
    values());
795         genreSBChoiceBox.getItems().addAll(Genre.
    values());
796         returnedLendChoiceBox.getItems().addAll(true
    , false);
797     }
798 
799     /**
800      * Clears the book fields.
801      */
802     private void clearBookFields() {
803         titleABTextField.clear();
804         authorABTextField.clear();
```

```
805         genreABChoiceBox.setValue(null);
806         yearABTextField.clear();
807         copiesABTextField.clear();
808     }
809 
810     /**
811      * Clears the search books fields.
812      */
813     private void clearSearchBooksFields() {
814         titleSBTextField.clear();
815         authorSBTextField.clear();
816         yearSBTextField.clear();
817         genreSBChoiceBox.setValue(null);
818     }
819 
820     /**
821      * Clears the search lends fields.
822      */
823     private void clearSearchLends() {
824         titleLendTextField.clear();
825         returnLendDatePicker.setValue(null);
826         phoneLendTextField.clear();
827         returnedLendChoiceBox.setValue(null);
828     }
829 
830     /**
831      * Clears the search customers fields.
832      */
833     private void clearSearchCustomers() {
834         nameCustomerTextField.clear();
835         phoneCustomerTextField.clear();
836         emailCustomerTextField.clear();
837         addressCustomerTextField.clear();
838     }
839 
840     /**
841      * Shows the add book view.
842      */
843     private void showPane(Pane pane) {
844         pane.toFront();
845         pane.setVisible(true);
```

```
846     }
847 
848     /**
849      * Shows the home view.
850      */
851     private void hideAllPanes() {
852         homePane.toBack();
853         homePane.setVisible(false);
854         addBookPane.toBack();
855         addBookPane.setVisible(false);
856         searchBooksPane.toBack();
857         searchBooksPane.setVisible(false);
858         lendPane.toBack();
859         lendPane.setVisible(false);
860         customersPane.toBack();
861         customersPane.setVisible(false);
862         statsPane.toBack();
863         statsPane.setVisible(false);
864     }
865 
866 
867     /**
868      * Updates the genre pie chart.
869      */
870     public void updateGenreChart() {
871         Map<Genre, Long> genreStats =
    clientController.calculateGenreLendingStats();
872 
873         // Calcola il totale dei prestiti per il
    calcolo delle percentuali
874         long totalLends = genreStats.values().stream
    ().mapToLong(Long::longValue).sum();
875 
876         // Crea i dati per il PieChart
877         ObservableList<PieChart.Data> pieChartData
     = FXCollections.observableArrayList();
878 
879         genreStats.forEach((genre, count) -> {
880             // Calcola la percentuale
881             double percentage = (count.doubleValue
    () / totalLends) * 100;
```

```
882             // Aggiungi i dati al PieChart con la
    percentuale formattata
883             pieChartData.add(new PieChart.Data(
884                     String.format("%s (%.1f%%)",
    genre.toString(), percentage),
885                     count
886             ));
887         });
888 
889         // Aggiorna il PieChart
890         genrePieChart.setData(pieChartData);
891         genrePieChart.setTitle("Lends Distribution
    by Genre");
892     }
893 
894     /**
895      * Initializes the genre pie chart.
896      */
897     private void initGenrePieChart() {
898         genrePieChart.setLabelLineLength(10);
899         genrePieChart.setLabelsVisible(true);
900         genrePieChart.setStartAngle(90);
901     }
902 
903     /**
904      * Updates the customer lending chart.
905      */
906     public void updateCustomerLendsBarChart() {
907         numusBarChart.getData().clear();
908 
909         // Crea una nuova serie per i dati
910         XYChart.Series<String, Number> series = new
    XYChart.Series<>();
911         series.setName("Distribution of Lends by
    Customers");
912 
913         // Ottieni le statistiche
914         Map<Customer, Integer> customerStats =
    clientController.calculateCustomerLendingStats();
915 
916         // Ordina i clienti per numero di prestiti (
```

```
916 decrescente)
917         customerStats.entrySet().stream()
918                 .sorted(Map.Entry.<Customer, Integer
    >comparingByValue().reversed())
919                 .forEach(entry -> {
920                     Customer customer = entry.getKey
    ();
921                     Integer lendCount = entry.
    getValue();
922 
923                     if (lendCount > 0) {  // Mostra
    solo clienti con almeno un prestito
924                         // Crea l'etichetta per il
    cliente
925                         String customerLabel =
    String.format("ID: %d", customer.getId());
926 
927                         // Aggiungi i dati alla
    serie con il valore numerico corretto
928                         XYChart.Data<String, Number
    > data = new XYChart.Data<>(customerLabel, lendCount
    );
929                         series.getData().add(data);
930                     }
931                 });
932 
933         // Aggiungi la serie al grafico
934         numusBarChart.getData().add(series);
935 
936         // Imposta il range dell'asse Y
937         NumberAxis yAxis = (NumberAxis)
    numusBarChart.getYAxis();
938         yAxis.setAutoRanging(false);
939         int maxLends = customerStats.values().stream
    ()
940                 .mapToInt(Integer::intValue)
941                 .max()
942                 .orElse(10);
943         yAxis.setUpperBound(maxLends + 1);
944         yAxis.setLowerBound(0);
945         yAxis.setTickUnit(1);
```

```
946
947         // Personalizza il grafico
948         numusBarChart.setTitle("Lends per customer"
    );
949         CategoryAxis xAxis = (CategoryAxis)
    numusBarChart.getXAxis();
950         xAxis.setLabel("Customers");
951         yAxis.setLabel("# of Lends");
952
953         // Aggiungi tooltip e stile hover
954         series.getData().forEach(data -> {
955             Node bar = data.getNode();
956             Tooltip tooltip = new Tooltip(
957                     String.format("Customer: %s%n
    Lends: %d",
958                             data.getXValue(),
959                             data.getYValue().
    intValue())
960             );
961             Tooltip.install(bar, tooltip);
962
963             // Effetto hover
964             bar.setOnMouseEntered(e -> bar.setStyle(
    "-fx-bar-fill: #ff8c00;"));
965             bar.setOnMouseExited(e -> bar.setStyle(
    ""));
966         });
967     }
968
969     /**
970      * Initializes the BarChart.
971      */
972     private void initNumusBarChart() {
973         numusBarChart.setTitle("Lends Distribution
    by Customers");
974
975         //configures the x and y axis
976         CategoryAxis xAxis = (CategoryAxis)
    numusBarChart.getXAxis();
977         NumberAxis yAxis = (NumberAxis)
    numusBarChart.getYAxis();
```

```
978
979         xAxis.setLabel("Customers");
980         yAxis.setLabel("# of Lends");
981
982         //forces the Y axis to use only integer
    numbers
983         yAxis.setTickUnit(1);
984         yAxis.setMinorTickVisible(false);
985
986         //rotate the labels of the x-axis for
    better readability
987         xAxis.setTickLabelRotation(45);
988
989         numusBarChart.setAnimated(true);
990         numusBarChart.setBarGap(8);
991         numusBarChart.setCategoryGap(20);
992
993         numusBarChart.getStyleClass().add("custom-
    bar-chart");
994     }
995
996 }
997
```

```java
package prj.library;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class LibraryApplication extends Application {

    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(LibraryApplication.class.getResource("main-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 1400, 900);
        stage.setTitle("Library Management System");
        stage.setScene(scene);
        stage.setFullScreen(false);
        stage.setResizable(false);
        stage.show();
    }

    /**
     * Close the application
     */
    public static void close() {
        System.exit(0);
    }

    public static void main(String[] args) {
        launch();
    }
}
```